SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Date: June 10, 2025

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Logan Calder
Grant Johnson
John Alvarado
Jack Landers

ENTITLED

EMT Vision

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
BACHELOR OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

Thesis Advisor [Dr. Krishna Ramamoorthy]

Department Chair [Dr. Silvia Figueira]

Department Chair [Dr. Shoba Krishnan]

EMT Vision

by

Logan Calder
Grant Johnson
John Alvarado
Jack Landers

Submitted in partial fulfillment of the requirements
for the degrees of
Bachelor of Science in Computer Science and Engineering
Bachelor of Science in Electrical and Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 10, 2025

EMT Vision

Logan Calder
Grant Johnson
John Alvarado
Jack Landers

Department of Computer Science and Engineering
Department of Electrical and Computer Engineering
Santa Clara University
June 10, 2025

ABSTRACT

Augmented Reality (AR) has demonstrated considerable promise for future mobile technologies, offering the ability to overlay crucial information within a user's vision while they can still maintain awareness of the surrounding environment. Similarly, Artificial Intelligence (AI) is an increasingly influential technology with significant potential to revolutionize the medical field. Its ability to rapidly learn and adapt to specific tasks makes it particularly promising for supporting paramedics during emergency calls. AI can efficiently analyze real-time data and present it in a concise, actionable format, enhancing decision making in critical situations.

Given the potential of these technologies, we have developed a smart AR headset designed to leverage AI for the benefit of first responders. This innovative device transcribes and analyzes real-time dialogue between patients and paramedics, extracting and displaying key information. This allows paramedics to review essential details during patient care and when completing necessary documentation, ultimately improving the quality and efficiency of emergency medical services. Furthermore, our headset is capable of providing 3D holographic visuals that display procedures and checklists for patient treatment, as well as using cloud computing algorithms that interpret conversations to automate the collection of patient data for scenario reports.

Table of Contents

List of Figures

Chapter 1

Introduction

Paramedics, EMTs, and other first responders work in a high-stress environment surrounded by panic, danger, and an overwhelming influx of information. To prepare for these scenarios, paramedics are expected to memorize hundreds of procedures and recall the correct instructions as clockwork. Even a simple error or hesitation risks the life of a patient, and as a result, the job is incredibly physically and mentally demanding. A review of five studies found 'overall burnout' among paramedics to be between 16 percent to 56 percent [22]. Also, paramedics were found to struggle to manage their responsibilities due to the demands of working in high-pressure, unpredictable conditions and the sudden transitions from calm situations to emergencies [1]. For example, on a call with a patient with chest pain, a paramedic must retain crucial information about the patient, assess injuries, follow strict procedures perfectly, and administer intervention. These scenarios are further complicated with the need for accurate retention of all relevant procedures taken and communication received by the patient to ensure seamless turnover between paramedics and hospitals. The vast concurring responsibilities present an opportunity to improve the quality of life of such essential services.

AR (Augmented Reality) is a headset worn by a user to merge a screen with reality, resulting in a view of a 3D hologram. The inclusion of AR in fast-paced environments is apparent, utilizing the benefits of easily-accessible information from a tablet without restricting the users' hands or distracting the user's gaze from their environment. As well as this, we discovered AI, artificial intelligence, could implement streamlines to communication. AI is a technology that enables computers to simulate tasks and create interpretations of data that are typically done by humans. With paramedics, AI provides opportunities to summarize, specify, and organize conversations to automatically recall critical information during emergency situations. By including AR technology in conjunction with AI, we present EMT Vision, an application that offers solutions to support paramedics with checklist management, speech transcription and analysis, optical character recognition, and more.

1.1 Use Cases

A few motivations for our project, EMT Vision, are as follows:

1. To record audio in noisy and loud environments, while still maintaining the capability of analyzing the data.
2. To transform audio data into JSON (JavaScript Object Notation); concise information that is easily digestible and comprehensible.
3. To store patient information securely, permitting the retrieval of on-call data for improved patient care later on.
4. To display a summary of patient data as an overlay on the AR headset.
5. To effectively analyze patterns in bulk sections of patient data, using cloud computing to determine potential risks and conditions a patient may have.
6. To offer an easy, hands-free method of visually contacting off-site doctors, showing them recorded patient data obtained on scene.
7. To provide an interactive checklist for tracking patient care progress.

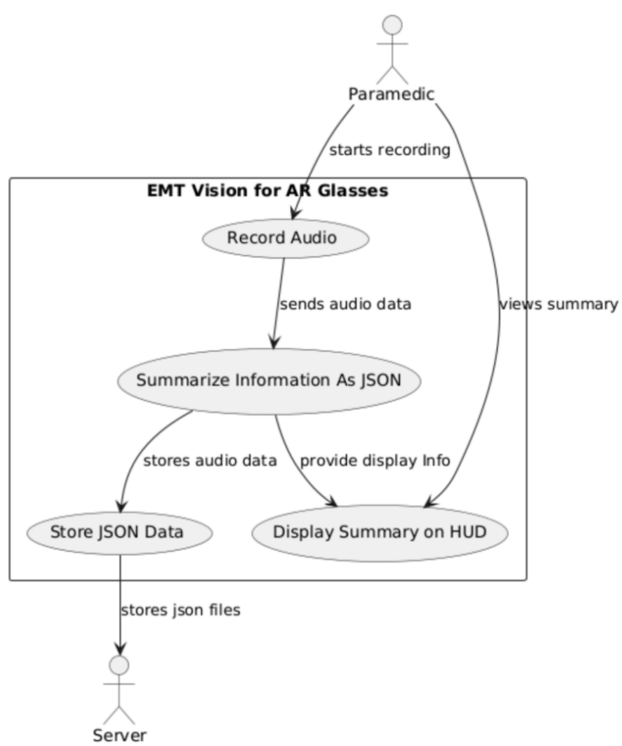


Figure 1.1: Diagram showing usage of the headset as well as the architecture of information recording and display.

1.2 Problem Statement

Paramedics work in a high-stress, demanding job that necessitates efficiency and reliability to save lives. During or after a call, paramedics are expected to complete a prehospital care record (PCR) form, which contains relevant patient data such as vital signs, personal background, and initial patient assessment. This information is relayed to the hospital to maintain continuity of treatment, so the form's accuracy is imperative [23]. However, these professionals are ultimately human and, given the circumstances, can be imperfect. In fact, a study comparing EMS documentation to body-worn cameras found an average of 11.7 errors, demonstrating the need for equipment to assist in recollection and documentation [14]. Additionally, the cognitive attention needed to recall this information and actively complete it reduces the time spent caring for the patient. Existing systems function on tablets that require two hands and undivided attention to operate, which tends to require an additional responder to perform separately to patient care. Overall, paramedics have lacked recent and universally-accessible technological advancements to assist with everyday tasks. Given this, we present EMT Vision an AR and AI-based technology that offers solutions to support paramedics in their field of work. This device presents solutions in a newly developed form.

1.3 Background

Before starting our project, we researched various other papers in related areas. From these sources, we gained inspiration and developed our understanding of other projects in the field and also considered what technologies and solutions could be applied.

1.3.1 Inspiration

In our research, we found examples of many cases where virtual reality simulations have been used for improved emergency response. Virtual simulations can allow firefighters to develop their skills further at a lower cost and at greater convenience to other alternative training scenarios. It is considered, because in educational settings, VR has been proven to improve skills that can be translated into the real world for children.

In his paper, Philip Braun proved that the use of virtual reality is incredibly effective for training firefighters to prepare for communication in a high stress environment. They prototyped an application to practice in virtual disaster management tasks and used motion capture to confirm that they were able to improve trainee firefighter performance. [3]

As well as this, Andrzej Grabowski showed a qualitative analysis that with his virtual training system, firefighters training improved. He concluded that more than 90% of users found it to be helpful, and intended to use the application again. This study used haptic feedback on top of this, so that trainees also had further simulation complexity of a virtual world than simply visual stimulation, which Gabrowski explains is the study's greatest success. [12]

In ‘The Future of Smart Firefighting’, Raveendran evaluates extended reality as it could be used in the field by firefighters. He highlights the efficacy of using artificial intelligence for real time assistance, on top of having a 360 degree perspective of the scene. However, he also outlines the barriers to development. This includes the high standards for training data that is sensitive to acquire, and the expense of making the product durable for firefighting scenarios. It is for this reason that we pivoted to looking for more achievable ways to help our local firefighters using augmented reality, while still introducing something new to virtual reality simulations. [21]

Ms. Cooper wrote an article in the Texas State University magazine that describes the plummeting costs of virtual and augmented reality. In this, she proposes that VR and AR training could also be used for medical purposes. Primarily, she suggests that the expense of training materials for medics working in ambulances significantly outweighs the affordability of virtually simulating the equipment. [6]

With this, we began to research how augmented reality is used in the healthcare industry and came across an article that describes the surmounting potential of doing so. The author explains the improvements that have been made by using artificial intelligence (AI) technology for medical imaging, surgery assistance, and symptom detection. Furthermore, they state that AR has also been used effectively for physical therapy, dental treatment, and supporting healthy routines like exercise. This is where we identified that there was a gap in the field where first responders in ambulances could also benefit from new innovations. [17]

1.3.2 Related Work

In preparation, we identified prior research papers that have aimed to develop a similar solution. It was important for us to understand where other projects succeeded, and what limitations they faced, which held them back from making a scalable distributed system.

How AR Glasses Can Help First Responders Save Lives, by Sasha Brodsky, outlines a similar product on the market. [4] It includes thermal vision, access to patient records, voice/hand control, building layouts, and more. It is also built on a custom headset. The device allows very similar info display features as our project with more access to patient data than is within our scope, and even has furthrer improvements, but it lacks our use of AI and sensors that could help our headset stand out. An article on AR in healthcare, suggests machine learning (ML) and AR can be used to display real-time patient information onto headsets for medical workers to view, as well as suggested treatments with assessed injury. [10] Another article, by Kelly Peng, also emphasizes the benefits of using augmented reality (AR) technology in the training of Emergency Medical Services (EMS) technicians and paramedics. [20] AR technology provides a safe and realistic training environment by overlaying images, videos, and 3D models onto the real world in real time. By providing a safe and realistic training environment, AR technology not only improves skill acquisition but also enhances confidence and performance outcomes. Additionally, the article emphasizes the flexibility, cost-effectiveness, and time-saving benefits of AR-based training. We found an example where Japanese EMTs use AR glasses while transporting patients to actively aid in treating them and communicate in real-time with doctors at the hospital. [7] The EMTs primarily used the goggles for this communication with the hospital, enabling them to ask questions, and ensure a smooth transition from treatment in the ambulance to treatment at the destination healthcare facility. While this aligns with some of our goals for communicating events more effectively, it uses some integrations that are outside of our reach, but again does not implement AI as effectively as we aim to.

1.3.3 Technical Research

To best understand other techniques and applications that could be combined with artificial intelligence assistance and augmented reality, we reviewed prior technical research of the field. While these were too complex to integrate for our project, they were worth researching to fully understand the state of the field and its potential.

The Respond-A project approach found that when combining augmented reality and AI, that first responders could benefit from resource allocation data being visualized, as well as information from drones interpreting the scene. While this system was not thoroughly tested, it was proposed that multiple teams and units could collaborate with the combinations of these emerging technologies. [19]

To prepare to use object detection, we researched Voxel 51 as an open source machine learning application for creating computer vision models. Built in evaluation functions could help us determine which models would best suit our program. This depends on the type of data that we aim to extract from our imaging, and the levels of detail that it needs to interpret. There were potential limitations to implementing this application on a headset as it would be overly demanding on our hardware, the Hololens 2. Voxel also relies on only imaging from video and would not utilize other sensors which would be ineffective in low visibility conditions. However because it is open source and well documented it had potential for helping us develop any object detection features. [25]

We also considered ways in which we could source pre-built models for our use case. Researchers in Maryland designed a computer vision model that could identify details from patients driver's licenses and vital monitor screens within an ambulance and record this data for the paramedic's convenience. These researchers failed to implement their algorithm on hardware. They had aimed to use smart glasses integrated with an app but settled for connecting the video feed to a zoom call that would interpret the data from an offsite device. This would cause issues where network connectivity is limited. On the other hand, if we could overcome this bottleneck, we could have a high confidence identification system. [8] In the end, we relied on a cloud based AI approach, but used Azure cloud services alternatively for their

Furthermore, a journal from the 2022 International Conference on Decision Aid Sciences and Applications (DASA) outlined the training of an object detection algorithm for detecting emergency vehicles using both audio and video. This could be used for other object detection features in our ambulance based application. In addition, this detection system can be easily replicated and they express ideas for how it could be applied in a variety of ways. It is particularly accurate and outlines an effective methodology for developing a model. [15]

Researchers at Stanford Medicine found that AR is capable of digitally displaying multiple screens that would otherwise be put elsewhere, making patient vital data such as electrocardiograms (ECG) much more accessible. We found that we could potentially apply these same concepts to our goggles, where we could ideally have an EKG reading of the patient projected to the goggles, as well as their patient data. [2]

1.4 Sequence and State Diagrams

The diagrams below display the broad process of audio recording, AI-based summarization, and the visual display of information obtained. The summarized architecture for the running HoloLens software is provided in the following sequence. Firstly, whilst idle, the headset will not record nor display data. During this time, the user can adjust the information panels or search through department procedures. The application will wait for the user to activate ”start recording”, which will initiate the repeated acquisition of data as audio files. These files are converted to text, removed of identifying characteristics, and sent to the AI to analyze. The AI outputs the summarized dialogue as a JSON file and updates the current display text with the requested relevant information. If prompted, the headset will visualize patient data in subcategories previously laid out by Santa Clara County EMS, enabling the user to easily fill out the required patient documentation prior to arriving at the hospital.

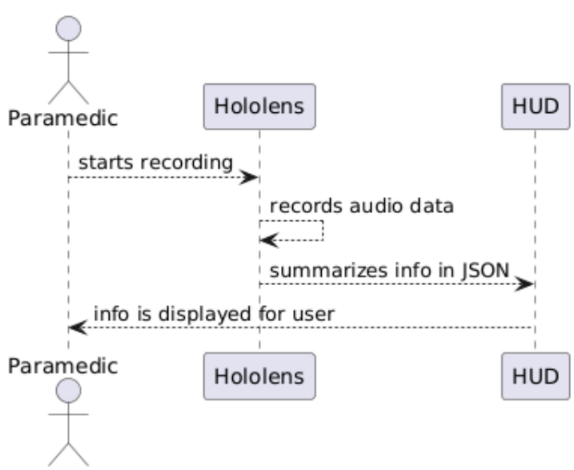


Figure 1.2: Sequence diagram displaying the process of audio recording to heads up display.

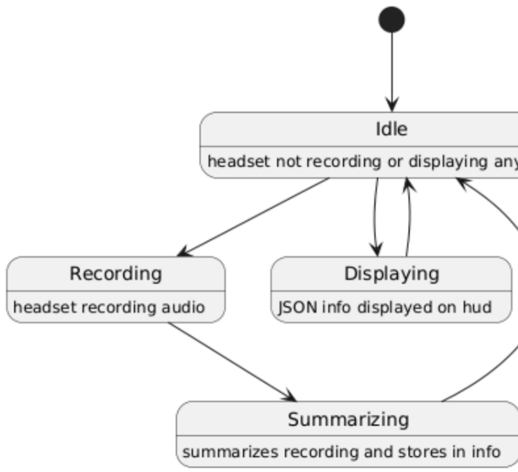


Figure 1.3: State diagram displaying the process of audio recording to heads up display.

1.5 Objectives

To efficiently design, develop, and improve our design, we constructed a series of objectives that we desired to implement in order. Continual development was also supplemented with multiple meetings with the Santa Clara County Emergency Department, obtaining valuable insight and suggestions for what features should be prioritized, designed, and refined.

1. Integrate an accurate voice recording system, capable of identifying words in a loud and chaotic environment.
2. Use voice-to-text software that may translate any dialect or language into easy-to-read, concise patient data visible on the Heads Up Display (HUD).
3. Efficiently implement a secure and confidential patient information database, capable of recalling data from calls earlier in the day.
4. Migrate the pre-existing Santa Clara County EMS App and its respective checklists, procedures, and protocols into the headset, permitting paramedics to keep track of their progress treating a patient. This was a notable feature specifically requested by the paramedics in Santa Clara County.
5. Engineer a versatile and lightweight battery pack capable of retaining battery life for an extended call.
6. Develop a live audio feed for calling medical professionals on scene while sharing all recorded data.

1.6 Our approach

Given the extensive range of features we initially aimed to implement on the Microsoft HoloLens, a carefully planned approach was necessary to ensure efficient development, thorough debugging, and effective testing. Key features of our project included interactive checkboxes and protocol lists, live audio recording and transcription, a lightweight and unobtrusive battery pack, and a calling functionality to facilitate communication with medical professionals.

Given the complexity and scale of our project, which includes numerous large-scale features operating simultaneously, a structured and methodical approach was essential to ensure efficient development, debugging, and testing over the months-long timeline. Our process began with outlining the project's initial layout and creating a detailed roadmap. This roadmap included clearly defined milestones, deadlines, testing phases, and the foundational design of the application's architecture. To manage the scope of development effectively, we prioritized the implementation of the audio recording and transcription software, followed by the checklist and protocol display. Once these core functionalities were completed, we initiated simulation testing in collaboration with the Santa Clara County paramedics to refine the design based on real-world feedback.

Recognizing that EMT Vision was intended to be utilized by Santa Clara County paramedics during actual emergency calls, we engaged in recurring consultations with paramedics from the department. These sessions provided a valuable platform to gather diverse perspectives and professional insights from EMTs. Through these discussions, it became evident that opinions within the department were divided regarding the role and potential of augmented reality (AR) in their field. Some paramedics viewed AR as a natural and inevitable evolution in emergency response, advocating for EMT Vision to streamline and automate as many on-call responsibilities as possible. Conversely, others expressed concerns that such technology might be viewed as untrustworthy by both paramedics and patients, potentially causing distractions and raising fears of job displacement or undermining professional judgment. This spectrum of viewpoints significantly shaped the direction of our project. Our goal evolved to focus on creating a solution that complements the work of paramedics rather than replacing or obstructing it. EMT Vision was designed to assist with background and supportive tasks, minimizing cognitive overload and allowing paramedics to concentrate fully on their critical responsibilities. Features such as the live chat with medical professionals and the interactive checklist display emerged directly from these discussions as key priorities, aligning with the feedback we received from Santa Clara County EMS teams. Ultimately, the insights and feedback gathered during these consultations played a pivotal role in shaping EMT Vision's development. By actively incorporating the input of the very professionals who would rely on the technology, we ensured that our device serves as an effective, desirable, and unobtrusive tool that enhances, rather than disrupts, the essential work of paramedics.

Development of the headset was conducted using Unity and Visual Studio, permitting for the compilation and deployment of the scripts onto the HoloLens. Given that we are a group of four working together to design and test software, we opted to use Git to ensure an effective workflow and sharing of code. We also aimed to not be redundant in our code, using assets already developed for fair use on Unity's asset store, and libraries that can be imported through the package manager. As our project heavily revolves around using AI, we utilized OpenAI's GPT 4o mini API technologies to effectively analyze audio and populate previously designed JSON templates, as well as Azure speech and vision services for audio transcription and optical character recognition.

We also were mindful to adhere to HIPAA protocols, laws, and procedures whilst developing our software, and as such developed technologies to censor sensitive patient-identifying information. This includes names, addresses, and other confidential data that we redacted before it was sent to AI services that use prompts for unsupervised learning. While we did conduct field-testing in simulations, we have collectively made the decision to not let our software be used for real-life calls as it would require extensive IRB approval that would be too time consuming for our development timeline.

Chapter 2

Requirements

2.1 Functional requirements

Functional requirements describe what EMT Vision must do. These are the specific features and behaviors that our project should have to perform in order to meet the needs of the user. These requirements essentially define the core functionality of the project and as such are critical to it's ability to achieve its intended goals. Different requirements have different priority levels for our project, that we used to effectively order our development process.

The first functional requirements for EMT vision are the highest priority and most important to the project. These objectives are essential to EMT Vision's mission and purpose to ultimately meet the needs of our users. These parts of the project are key and as such we consider them nonnegotiable end goals. The following are what we identified as our core functional requirements:

Audio Recording and Transcription The system must be able to record and transcribe audio. It must record do so through onboard microphones on the HoloLens 2 and then save this locally before being passed to an internal agent that can process the audio file, turning it into a transcription of the recorded audio in the form of a text string. It is also imperative that this audio buffer should then be immediately deleted for the sake of privacy.

Information Analysis The system must analyze and summarize key information from the transcription of the audio recording in a new JSON. It must do this by going through the audio transcription and identifying the important/relevant information and adding it to a JSON to be displayed.

Information Display The system must be able to display the summarized information on the HUD of the HoloLens 2 by reading the JSON with the summarized information and displaying it on screen in a simple, user friendly manner.

These three features are at the core of what our project aims to do. Therefore they are the highest priority functional requirements for EMT vision. These methods work together to aid the user by showing them important information from their conversations with patients in a clear and concise manner. It also stores the information securely in the database so that the user can easily go back and reference past data when they need to.

In addition to these requirements, there are other functional requirements for EMT Vision. These features and functions are not as high priority as the previously identified functional requirements, but this does not mean they are not important. These requirements are still essential for EMT Vision to function as a powerful tool to assist paramedics. Some of these requirements are also features that were requested by those we met with Santa Clara EMTs. The following requirements are essential to overall functionality and user experience:

Translate Languages The system must be able to translate different languages. It must translate the audio to English before the audio recording is transcribed and summarized. This allows EMT Vision to be used in scenarios where English is not being spoken.

County EMS Protocols The system must display official Santa Clara county protocols for the user. It must access official documentation that lists treatments and procedures before displaying them in an interface that is clear and easy to use. The menu for navigating this should be similar to the existing Santa Clara EMT app that is currently on tablets used by Santa Clara EMTs, and should have the ability to track progress.

Detect Medication Labels The system must be able to detect and analyze a medication label. It should be able to identify a medicine bottle and find the label for what medicine it is, along with other important information on the bottle. It must then display this information on the HUD of the HoloLens 2 for the user to see.

Although these functional requirements are not as high priority as the previous, their implementation is still required to further improve the usefulness of EMT Vision. The ability to translate different languages is particularly helpful in diverse communities that speak multiple different languages. This feature prevents a scenario in which a paramedic cannot properly care for people due to the patient speaking a foreign language.

The other two requirements, county protocols and medication label detection, were specifically requested during one of our meetings with Santa Clara EMTs. They requested the county protocols feature specifically so that they could actively keep track of their progress when caring for a patient. The medication label detection was another highly requested feature, as they sometimes need to sift through lots of medicine bottles at once and having a way to quickly identify what each one is for would improve their efficiency in caring for patients.

Although these functional requirements are not as high priority as the previous, their implementation is in no way nonessential; however, they are not at the core of what we set out to accomplish with EMT Vision and therefore have a less significant priority than the other functional requirements.

2.2 Nonfunctional requirements

Nonfunctional requirements describe the operational qualities of the system, focusing on performance, security, user experience, and other qualitative attributes. These requirements ensure that the project is reliable and efficient while also ensuring that it adheres to important standards and guidelines. In short, nonfunctional requirements describe how our system should be, rather than what it must do.

Similarly to our functional requirements, we have separated our nonfunctional requirements by priority. The more important nonfunctional requirements have a higher priority as they are important qualitative features for EMT Vision to have. The following nonfunctional requirements are what we identified as the highest priority:

Compliance The system must adhere to healthcare regulations regarding privacy and security, such as HIPAA. This affects how EMT Vision stores and uses patient information, as violations of these standards could result in serious legal issues.

Accessibility The system should be easy to use and simple to learn. It should be easy for a user who has never interacted with AR technology before to figure out how to utilize EMT Vision to its fullest when they are in the field.

User Interface (UI) The system should have a clean and easy-to-use UI. Menus should be well formatted, text should be clear and readable, and the general flow of the various interfaces should be simple and easy to follow.

User Experience (UX) The system should have multiple features that enhance usability and UX. It should have various quality of life features like scrolling through menus, increasing or decreasing the text size, or hiding certain parts of the menu as to avoid cluttering the HoloLens 2 HUD.

These features are our core nonfunctional requirements because they focus on two key issues. Firstly, complying with standards such as HIPAA is very high priority and is critical to EMT Vision avoiding legal issues regarding patient privacy. Secondly, making sure EMT Vision is accessible and user friendly is vital to its success. Some paramedics may not have a lot of experience, if any, with this kind of technology. As such, it is important that it is easy to use and understand. It would be counterintuitive if the project we specifically designed to aid and assist paramedics instead made performing their jobs more difficult.

The rest of our nonfunctional requirements focus on the consistency and efficiency of the system. These requirements lay out qualitative features of EMT Vision that are important for overall usability. Fulfilling these requirements will help EMT Vision achieve its goal of being a useful tool for paramedics as it is intended to be.

Reliability The system should be reliable and consistent when performing tasks. It should be capable of completing the various tasks from the use cases in Section 1.1 reliably, otherwise it could end up becoming a hindrance to the user instead of helping make their job easier like it should.

Efficiency The system should be able to update the information in the HUD in a reasonable time frame. It is important that the system can keep up with the paramedic using it, as they often work in fast moving and stressful environments.

Readability The system's menus and interfaces should be easy to read. The display of the summarized information from the JSON, as well as other features that display text on screen, should be clear when displaying on the Hololens 2 HUD. Users should have no problem reading button or menu labels, in addition to the actual patient information.

These nonfunctional requirements are not as high priority but still important for the overall usability of EMT Vision. If all of these nonfunctional requirements are satisfied, then we will have made the ideal version of our project and it will be a powerful tool that will aid and assist Santa Clara EMTs as they care for patients when in the field.

Chapter 3

User Research

3.1 Methods

To gain a complete understanding of the technical challenges faced by paramedics in the field, we conducted interviews and surveys to assess the technologies they currently use. Paramedics are tasked with collecting critical information about patients and the nature of their emergencies, both at the scene and en route to the hospital. This data is typically communicated in real-time to the receiving medical facility to ensure that hospital staff are adequately prepared upon the patient's arrival. Currently, this information is manually entered into an iPad application, a process that has proven to be inconvenient and inefficient in the dynamic environment of a moving ambulance. The application is called ImageTrend and is accessed by paramedics simply through a web browser that relies on the ambulance's wifi connections susceptible to inconsistency in remote areas or tunnels. Manual data entry not only increases the likelihood of typing errors but also diverts the paramedics' attention away from direct patient care, which can be detrimental in high-stress, time-sensitive situations.

When we reached out to the Santa Clara Fire Department, who serves our campus and the surrounding areas, to validate our ideas. The response from the fire chief was highly encouraging, leading to an in-depth discussion about the operational challenges firefighters and paramedics face daily. We presented our preliminary ideas, including computer vision powered diagnosis and real-time transcription using natural language processing for summarization. Through these discussions, we aimed to understand how integrating augmented reality (AR) and artificial intelligence (AI) into ambulance workflows could assist first responders during high-pressure emergencies.

We engaged directly with these professionals and gained insight into the hardest parts of their work day. We found that they needed immediate access to patient history, step-by-step procedural guidance, and real-time hazard awareness. With this input, we refined our project's objectives and ensured that the final product would be both practical and beneficial to those working in emergency medical services.

3.2 Stakeholder Needs

Emergency scenarios are inherently unpredictable, requiring highly trained first responders who can rapidly assess situations and make critical decisions under pressure. Recognizing this, our objective is to develop an assisting AR headset that serves as a decision-support tool rather than a replacement for paramedics or firefighters. These first responders remain our primary stakeholders, as they are the individuals who will directly interact with the technology in real-world emergency settings.

The Santa Clara Fire Department plays a crucial role in safeguarding the lives of over 225,000 residents in our community, relying on a team of just 66 firefighters and officers to provide 24/7 emergency response services. Given the high-stress nature of their work, these first responders frequently deal with fatigue, split-second decision-making, and the mental strain associated with witnessing traumatic events. The cumulative impact of this exposure can lead to burnout and, in many cases, post-traumatic stress disorder (PTSD). By optimizing their workflow using our AI-assisted augmented reality, we aim to reduce their cognitive load, minimize any human errors, and enhance situational awareness, which will ultimately improve both their efficiency and well-being.

In addition to the firefighters and paramedics, our secondary stakeholders include the broader population that depends on these emergency services. By equipping first responders with tools that enhance their efficiency and decision-making accuracy, we indirectly improve outcomes for patients and accident victims. The ability to provide faster, more precise care in critical situations benefits everyone who may one day rely on emergency services.

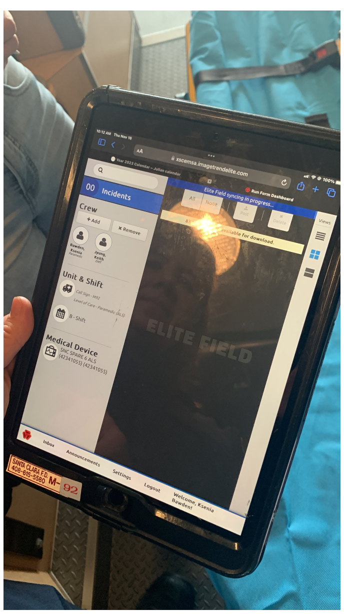


Figure 3.1: Paramedics use an application called ImageTrend to collect and communicate information to the hospital, which is accessed through a web browser.

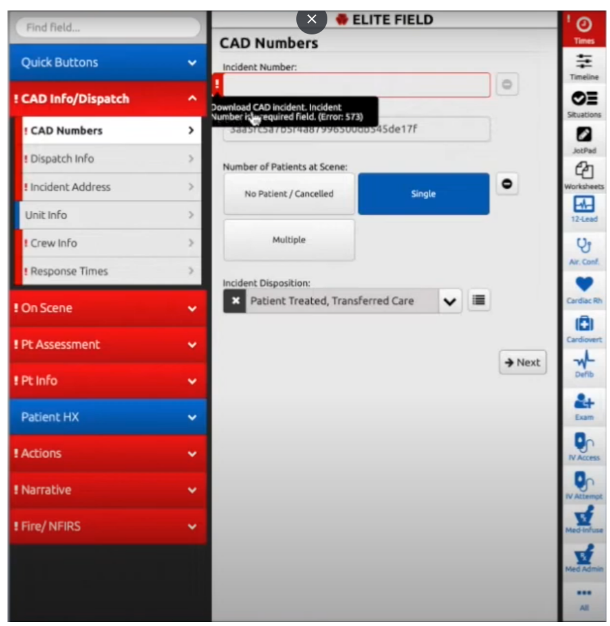


Figure 3.2: The ePCR form layout in the ImageTrend application.

3.3 Paramedics Experiences

To fully understand the experience of being a paramedic, we spoke to Jack, a young EMT who had joined the Santa Clara Fire Department last December and was still on probation. This is a stressful period for him, as he is expected to prove himself in his first two years on the team or could be subject to being removed at any moment. This position comes after having already gone through a rigorous six months of training in the academy, which he could only do after passing the NREMT. He then had to be successful enough in the academy to be one of the few competitive enough to join the crew, who hold high standards for their recruits.

Becoming a paramedic is a rigorous process that typically takes between two to four years. It involves completing an accredited training program, gaining hands-on experience in both clinical and field settings, and passing the National Registry of Emergency Medical Technicians (NREMT) exam, an assessment that is both academically challenging and physically demanding. Despite these significant requirements, paramedics in Santa Clara earn a median base salary of an estimated $87,000 [11], which falls far below the area's median household income of $174,000 [24]. Given Santa Clara's exceptionally high cost of living, particularly in housing, this wage gap can place financial strain on people working in the profession.

On top of the financial stress, the emotional and psychological toll of working as a paramedic is overwhelming. When they respond to high-stakes emergencies paramedics witness traumatic injuries and handle extreme emotions. They have to be in life-and-death situations on a regular basis and this can take a lasting toll. Research has shown that paramedics experience significantly higher rates of PTSD compared to the general population. A systematic review and analysis by the University of Heidelberg found that altogether the 12-month prevalence of PTSD among paramedics was 20.0% (95% CI = 16.1–24.3%), which is significantly higher than the 3.1% observed in the non-exposed general population [13]. The effects of this psychological burden often result in chronic anxiety and depression, as well as sleep disturbances and emotional detachment. This not only affects their personal well-being and relationships but also increases the risks of burnout, substance abuse, and even suicidal thoughts, which can potentially compromise the quality of emergency care that we rely on.

Figure 3.3: The ambulance interior seats one paramedic and a patient. Most of the space is occupied by equipment.

Figure 3.4: The fire engine interior seats the station captain, one paramedic, an engine technician, and two additional firefighters.

3.4 Current Tools

Paramedics are tasked with collecting critical information about patients and the nature of their emergencies, both at the scene and en route to the hospital. This data, known as electronic patient care reporting (ePCR), is typically uploaded synchronously to the medical facility receiving the patient, enabling hospital staff to prepare for the patient's arrival. Currently, much of this information is manually entered into an iPad in a process that, while functional, proves inconvenient and inefficient in the dynamic environment of a medical emergency scene or the unstable conditions of a moving ambulance. The iPad application used by Santa Clara paramedics to handle patient data transfers is called ImageTrend and is accessed through a web browser using the ambulance's Wi-Fi connection. The ambulance's cellular connection can be slow in remote areas, in tunnels, or other network dead zones, posing the risk that patient data could be lost or communicated inaccurately. Manual data entry not only increases the likelihood of user input errors but also forces the paramedic to prioritize staring down at a screen, diverting the paramedics' attention away from upholding optimal patient care. This no-win dilemma is ultimately detrimental in high-stress, time-sensitive situations. From our conversations, ePCR applications generally are viewed to be intuitive and lengthy, although ImageTrend is regarded more positively due to its customization and streamlined UI that appeals to less tech-savvy users. Nonetheless, this existing means of communication is not in line with other technological advancements in other industries. Meanwhile, the only other notable communication equipment available to paramedics consists of standard radios or cell phones, which lack specialized features and are inefficient for the variety of situations paramedics experience on a consistent basis.

Language barriers further complicate the efficiency of emergency medical response, particularly here in Santa Clara County, where approximately 21% of residents are not proficient in English [5]. These communication challenges can delay the accurate relay of critical information, potentially impacting patient treatment. However, advancements in natural language processing (NLP) technology present promising solutions. Real-time translation tools using NLP can bridge language gaps so that paramedics and patients can always communicate. Further, NLP can automate the extraction of important information from conversations, reducing the work for paramedics and enhancing the accuracy of data transmitted to hospital staff. With this, we can also ensure that sensitive patient data is redacted and privacy is upheld.

In addition to NLP, image processing technologies offer transformative potential in ambulance care. High-resolution imaging combined with artificial intelligence (AI) algorithms can help the diagnosis of conditions like fractures, internal bleeding, and dermatological issues. For example, computer vision analysis has demonstrated diagnostic accuracy rates of up to 94.6% for identifying skin cancer lesions, which is similar to that of experienced dermatologists [9]. Integrating such technologies into emergency medical services could expedite the identification of critical conditions by helping to guide pre-hospital treatment decisions, and improve patient outcomes with fast and accurate diagnoses.

3.5 User Stories

To learn about the needs and concerns of emergency responders, we presented our project proposal to two local Santa Clara fire departments and had open discussions to get their feedback. Considering our lack of on-call experience, we hoped that these conversations could provide invaluable insights into the realities of emergency response work and help us better understand the more subtle, day-to-day struggles. Although not all suggested features will be included in our project, we hope to incorporate the options that match the general feature set of our project.

One recurring concern raised during our discussions was the issue of privacy. Many emergency victims are in distress and may not want their condition or situation recorded or exposed in any way. Taking this feedback into account, we have made privacy a top priority in our system design. Our approach ensures that any data collected by the headset has identifying patient information redacted, securely stored, and only accessed when necessary. Rather than relying on video recording, our system primarily uses real-time overlays and AI-generated summaries to assist first responders without compromising the dignity and confidentiality of the individuals receiving care. Another request was to include department protocols in the project. Often, EMS professionals are in situations that require strict procedures to be met exactly, which while already memorized, can benefit from a HUD overlay flowchart. This is particularly helpful when in less common situations like extreme emergencies involving coordination between different departments or patients with obscure medical symptoms. During these calls, adrenaline can overcome preparation, causing incoordination and inefficiency when such mistakes are potentially the difference between life and death. A further comment provided by the EMS was to incorporate the ability to translate a variety of languages. Considering our location in California, this feature was a consensus-beneficial addition to better assist the arrangement of unique cultures and languages in the greater Santa Clara community. We hope that through the use of AI summarization, our model will automatically and accurately translate the more frequent languages in the local area, as well as English, to improve communication with previously under-represented immigrant communities. Another surprising piece of feedback we received was the capacity to call hospitals. During medical calls, serious conditions frequently extend beyond the permitted abilities of a paramedic, but depending on the situation, they still may be able to continue aid through feedback from doctors. Currently, this may require the professional to remove their gloves, find their phone, and call the corresponding hospital assistance line, which can waste time. Through the headset, this could be streamlined functionality to swiftly acquire hands-free consultation. Although this isn't a priority, this is a feature we believe we could accomplish after completing our more pressing capabilities. Another lower-priority goal we would like to tackle, if time allows, is the ability to scan medicine bottle labels for automatic documentation into patient records. While engaged in a medical call, a frequent experience among paramedics we met with was that patients often possess a substantial quantity of medicine pill bottles, and each must be manually cataloged. Despite this inconvenience generally not posing life-threatening risks, it is infamous for being tedious and slow. Consequentially, these challenges open the risk for mistakes in documentation, increase call durations, and distract the medical professional from attending to the patient.

3.6 Ride-along Immersion

We deepened our understanding of the real-world challenges faced by paramedics by each participating in ride-along experiences with different crews in the Santa Clara Fire Department. These immersive experiences provided first-hand exposure to the high-pressure environments in which our technology will be deployed. By combining direct field experience with a thorough review of existing case studies, we aimed to design a system that aligns seamlessly with the needs of those who will rely on it in life-and-death situations.

Figure 3.5: The firefighters let Jack Landers try on protective gear and carry a hose, so that he could experience firsthand the physical strain of their work.

Chapter 4

Design and Rationale

4.1 Overall Design

EMT Vision is designed to obtain, analyze, and display information for the paramedic who uses it. Featuring an interactive protocol display, patient information list, live feedback, and cloud computing, the headset is an incredibly versatile tool capable of being utilized in a variety of situations. We designed this headset to function as an assistant rather than replacing paramedics, thus improving the efficiency and quality of patient care, while also reducing stress and workload from the responding crew.

Our headset functions by visualizing a non-obtrusive interactive display for the user while still maintaining the capability for viewing real life. The entirety of our features may be enabled or disabled, either through physical hand gestures or vocal commands.

When designing this project, user experience was a forefront priority, as we were developing software catered towards those who were, for the most part, unfamiliar with this technology. Given this, we designed our headset to be as accessible as possible, providing multiple options for human interaction and allowing the user to interact with the HoloLens 2 in a way that feels most native to them. Furthermore, we wanted to ensure the confidentiality of patients, and thus securely store patient information in an encrypted database, hidden behind authentication and authorization layers.

4.2 Headset Screenshots

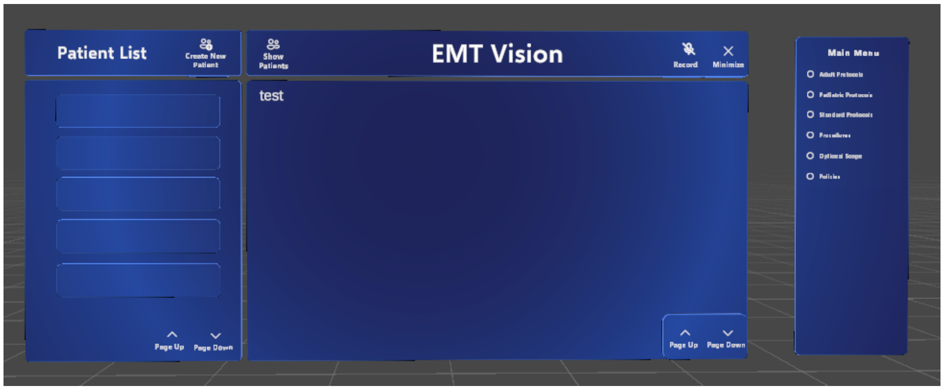


Figure 4.1: The full User Interface (UI) visible on the headset.

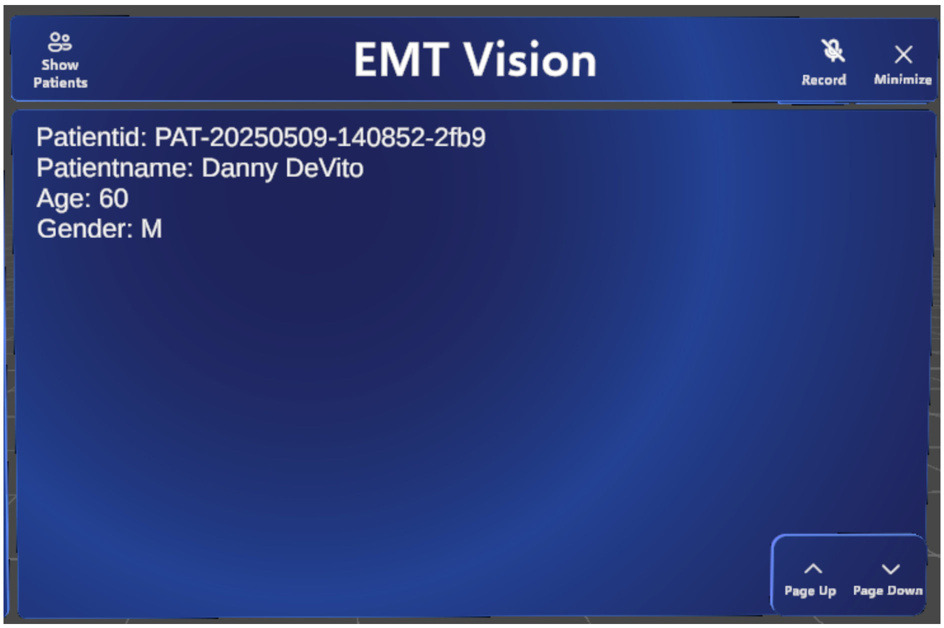


Figure 4.2: The patient data display, visible per each patient.

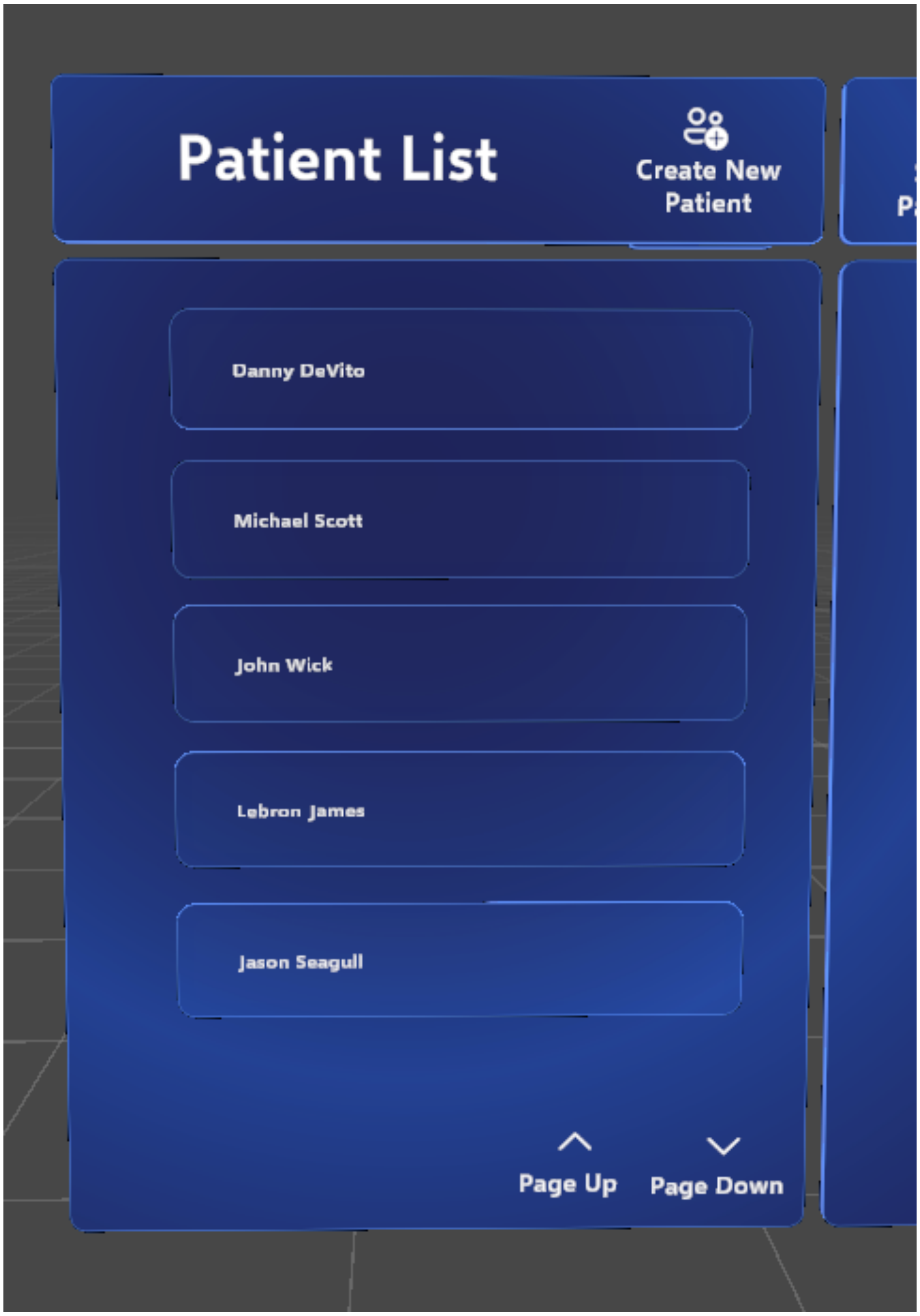


Figure 4.3: The patient list visible on the UI.

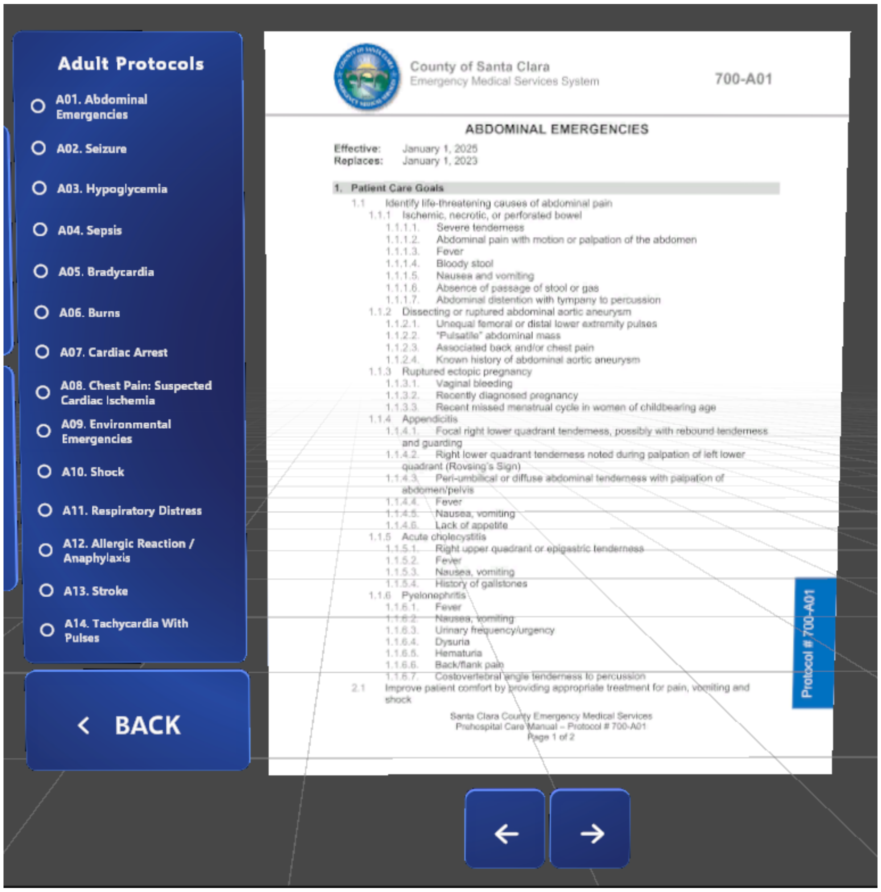


Figure 4.4: Our protocol and procedure display for Santa Clara County standards.

4.3 Website Screenshots

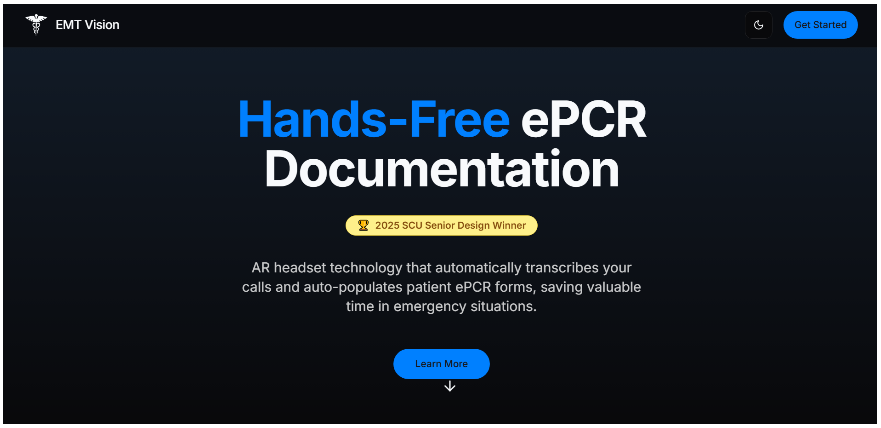


Figure 4.5: Website commercial landing page.

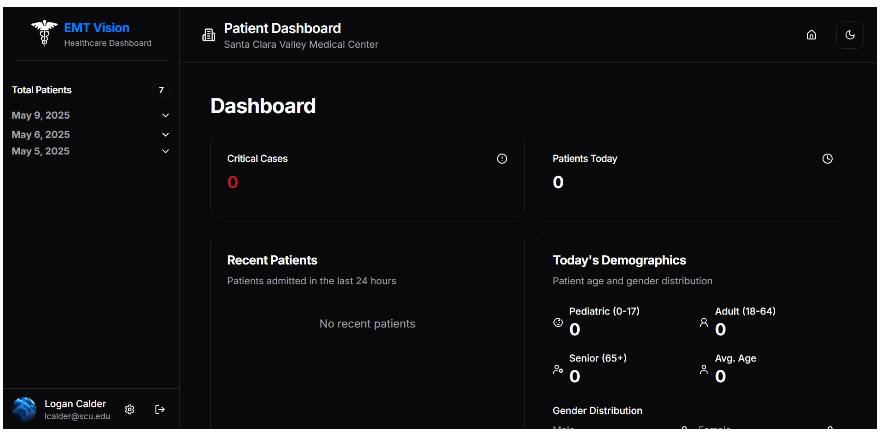


Figure 4.6: Website interactive Dashboard, showing overall statistics and important information.

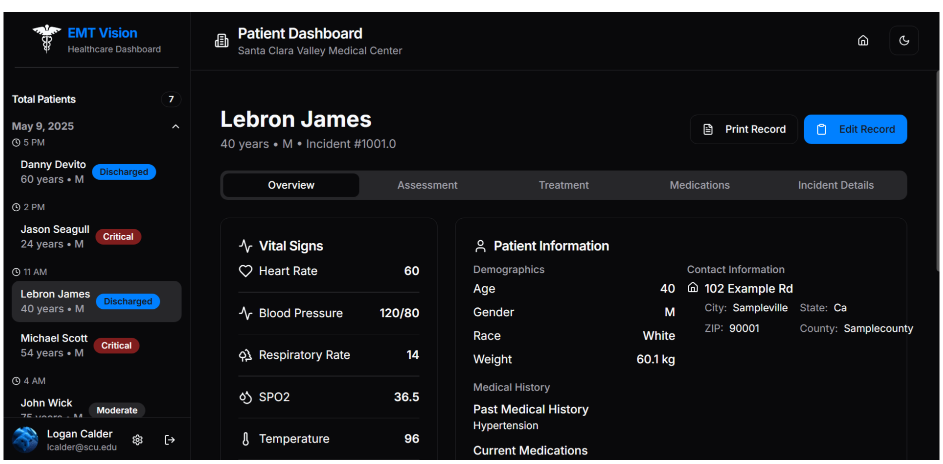


Figure 4.7: Per-Patient information display, showcasing ePCR data for that individual.

4.4 Protocol and Procedure Display

During one of our meetings with Santa Clara EMTs, we were explicitly asked to create a feature to view and track the progress of protocols and procedures on calls. To do so, we accessed the official 2025 documents listing these treatment plans and imported them into Unity. We decided to design an interface that is easy to navigate and friendly to those unfamiliar with AR headsets, with easy-to-read buttons that may be either clicked by hand or verbally selected. We also designed our GUI to be as accessible as possible, mimicking the already-familiar design of the official Santa Clara EMT app which is used on rig's tablets.

Utilizing MRTK3 pre-made assets, we designed a navigatable menu from which you may select various categories, each copied directly from the Santa Clara county policies and procedures app. Users may then view protocols and procedures in 3D space, with the capability of tracking which step they have visited using checkmarks rendered over the buttons. While basic, this was surprisingly one of the most demanded features from the general EMT population for the city.

Another diagram of procedures used by paramedics on an Ambulance is the hospital details chart. While on a ride-along with the Santa Clara fire department ambulance, we were shown a chart inside the apparatus showing all local hospitals, their specialties, and their phone numbers. The paramedic is expected to have this flow memorized, but informed us that easier access to it would still be helpful. This includes hospital names and their corresponding phone numbers, locations, ID, and approved treatment services. The patient's preferred hospital is generally chosen, although for code three incidents (emergencies), the medic will choose the closest facility that specializes in that condition (such as burn center, comprehensive stroke center, etc).

4.5 Audio Recording and Analysis

Information acquisition is initiated through activating a "start recording" button on the HUD, which activates conversation transcription via the HoloLens 2's onboard microphones, although this collectively hears both ambient and user sound. This data is dissected into intervaled 20-second subsections and saved into a local file. An internal "watcher" script, which records the local path of the directory where recordings are stored, detects new-added audio files. This information is then passed to our Azure AI Agent, which utilizes Microsoft's Speech Services to process the .wav audio file into a string. Documenting dialogue as text instead of audio files is essential to transfer this data from the headset to the backend server and to OpenAI for processing, especially since ambulance WiFi is often unreliable outside urban areas.

After a recording is transcribed, the text string is sent to our OpenAI summarizer which filters text from the conversation by comparing it to desired categories of user information such as allergies, injuries, medications, and dozens of other key descriptions. Relevant information is populated into the corresponding JSON text field, with unmentioned sections remaining blank. Utilizing a highly customized prompt and the ability to read from local files, we can write and store user data in extreme detail while discarding redundant information. This data is then sent to two services, one being our Unity Text Render service, which displays the summarized information on the HoloLens 2's HUD, whereas another instance of the data is sent to our external database hosted by Supabase, a PostgreSQL service.

To implement the capability of continuously recording and updating a patient file, every time a new patient is created on the headset by the user, a unique patient ID and time stamp is generated. This is then set as a local variable within the script, which is used to either create or add columns within its respective row in Supabase. This logic also applies to the patient select menu, which allows paramedics to resume the recording for a previously treated patient by reopening their file. While the "start recording" button remains toggled, the 20-second audio recording and processing will loop until the button is toggled off, simulating a contiguous conversation. However, passing the AI such defined durations of conversation had presented flaws in our initial design with inaccurate data analysis. Firstly, due to the strict recording cutoff time, audio recordings are frequently begun or concluded within a sentence from another audio file. This causes the AI to lack the necessary context needed to understand the spoken words, such as the sentence saying "I do not have an allergy to peanuts" potentially being read as "I do not-" and "-have an allergy to peanuts" between two different files, enabling incorrect documentation. Lengthy conversations also resulted in the AI overriding or incorrectly joining previously established information, such as a patient previously saying their name is "Tom" at the start of a call and a later audio recording detecting someone else stating their name is "Bob", leading to inconsistency. We have worked to alleviate these concerns by providing extended, overlapping recordings to include needed context, as well as defining stricter guidelines for the AI to overcome provided data inconsistencies.

Another complication we experienced with implementing the audio recording and transcription scripts was that the libraries for OpenAI we planned to use were not compiling on the headset. The Microsoft Hololens 2 is built in a UWP framework, also known as Universal Windows Platform, allowing for easy porting of applications between UWP devices. However, after creating a Python script for the headset, we learned that UWP does not support the Python language, forcing us to research alternatives. Despite trying to still access the Python imported libraries that contained the AI transcription software we planned to use, we ultimately pivoted to instead utilizing AI libraries from the supported CSharp language.

4.6 Database

Provided that we utilize an external database to store user data, patient confidentiality is a core concern. Supabase offers HIPAA compliance for ensured security, allowing us to store identifying information securely. It should be noted, however, that we elected to not pursue this option as it would require a monthly fee of roughly 600USD, though this would be a viable option if this project were to scale to a startup.

We may access our data through POST and GET calls to Supabase, which allow us to recall previous call information. This is beneficial in the instance the paramedic is treating multiple patients at a time and needs to resume treatment on the former. Further, this data may then be streamed to medical professionals awaiting at regional hospitals or treatment facilities.

Considering that we are storing potentially confidential patient information, we must implement all security measures possible. To ensure the encryption and hiding of patient data from unauthorized users, all calls to the database are constructed on the backend. For our interactive dashboard, Supabase calls are run on the server side, returning their value to the client. In Unity, we may simply put these calls within our scripts. To prevent the access of databases from external users viewing our source code, we also have hidden access keys and other tokens within local environment variables and app configuration files, from which they are read by scripts. These files are intentionally not pushed to our GitHub repositories, ensuring they remain secret.

4.7 User Interface

Implementing a simple, quickly-navigable display for the headset user was essential for efficient usage while assisting a patient. Our solution was to separate different user actions using clickable buttons with labels and icons to improve identifying available actions to the user. Key quality of life options for the patient info include text size increase or decrease buttons, next or previous page buttons (for patients with data that extends beyond a single page), and the ability to drag the HUD in any direction (while it still rotates to face the user). For general menu options, there is a reset layout option to move the HUD back to its default position, a minimize that collapses the HUD to improve environment view ability, and a start record toggle to begin recording information about the patient. Additionally, we have buttons to show or hide further menus, one for county procedures and one for a list of available patients. As a result of all of these options, we hope that users can easily customize their layout to fit their needs and traverse menus in a way that feels intuitive. Considering the variety of technical experience within the department, we opted to retain all of the project's functionality on a central panel, referred to as a slate, to avoid users getting lost in menus within the application. In the body of the slate is a text box, containing the summarized JSON information that is adjustable by the previously-mentioned buttons.

In order to display the server info, the headset has a script calling the API every ten seconds (the length of ten seconds matches the length of each recording), ensuring any changes to the user's info are reflected on the HUD in a timely manner. This loop also calls the ten most recent patients from the database and displays their name, allowing the user to select one as their ”active patient” to display and record conversations for. After engaging in a ride-along with Santa Clara Fire Department, we learned that certain information is more commonly exchanged with hospitals, including patient name, age, demographic, gender, chief complaint (the patient's main injury), pertinent info about complaint (key symptoms), most recent vitals, paramedic's treatments, and HAM (patient history, allergies, medication). This preference led us to emphasizing these details (if available) on the first page of patient information and having supplemental data on the additional other available pages.

4.8 Patient Dashboard

To provide the capability for viewing patient data in real-time to those who are not wearing a headset, we designed an interactive patient dashboard, protected by secure sign on and authorization for medical professionals granted access. Our goal in mind for this feature was for doctors to be able to converse with paramedics on calls, providing them live feedback and suggestions if the EMT were to struggle for next steps. Whatever information that had been recorded and displayed for the headset user to see is instantly sent to Supabase, from which is displayed on the patient dashboard.

We implemented this dashboard using Next.JS, a Vercel React Framework. This tool utilizes TypeScript, HTML and CSS to deliver powerful web tools for many different applications. Designing our landing and dashboard pages in HTML, these client frontends then tie to our server backends, which fetch encrypted and protected data from the database. To prohibit unauthorized users from accessing confidential user information, we implemented Single Sign On (SSO) with Google, tied in with explicit user whitelisting for authentication. Doing so prevents users outside of medical institutions from signing in, and only allows certain medical professionals from within those institutions to actually read patient information.

To present a working demonstration of our technology, we launched and deployed our web service live, allowing anybody to test and use it. We did so using Vercel, which allows for the live deployment from GitHub repositories, while also ensuring that our service passes a build (quality) test.

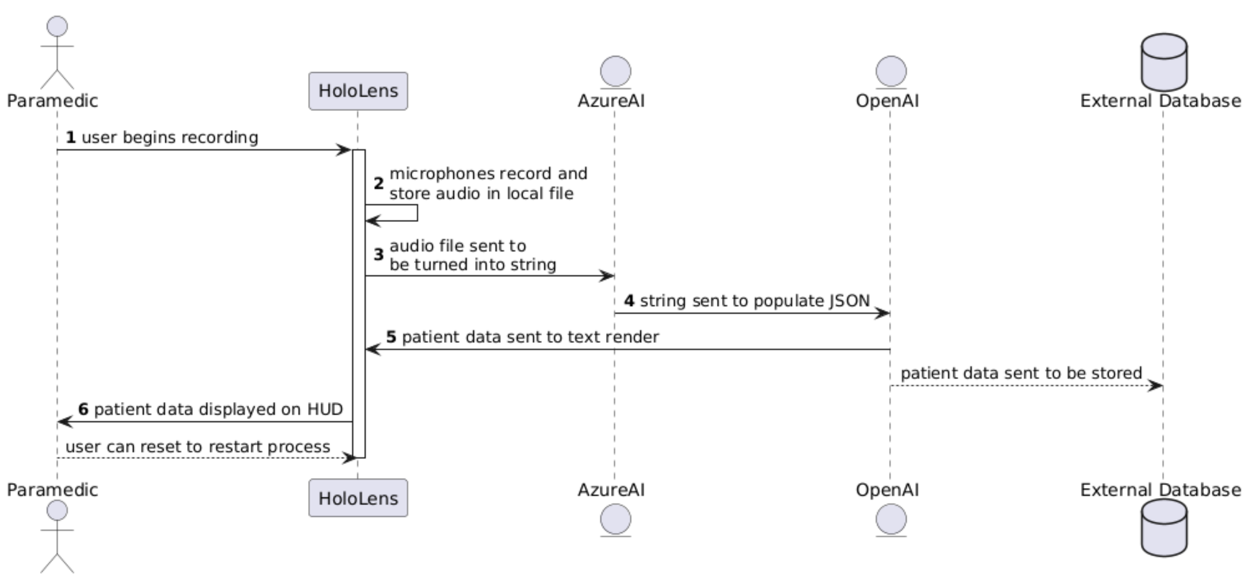


Figure 4.8: Overview of each step taken in the information display process

Chapter 5

Technologies

5.1 System Components

To successfully implement a smart assistant AR headset, we rely on various hardware and software to integrate a seamless experience that is not just efficient but also easy to use for those unfamiliar with more advanced technologies. Given this, we aim to utilize the most renowned hardware for the job, as well as highly rated and reliable engines for running our software. The design goals and selection criteria for the hardware include being lightweight, portable, long-lasting, and reliable. For developmental purposes, we also have the requirement that the software should be easily deployed on the hardware. The software requirements include efficiency, accuracy, security, and testability.

5.2 Hardware

5.2.1 Microsoft Hololens 2

The Microsoft HoloLens 2 is a versatile piece of equipment, offering much to our project that is unique to its design. Featuring an Augmented Reality (AR) visor, robust development tools, and a lightweight design, the HoloLens 2 is an excellent choice for our vision. While there are many other competitors and devices to choose from, we ultimately select this headset due to a few key factors.

The Microsoft HoloLens 2 is unique in its ability to maintain human interaction while wearing the headset, with eyes visible either directly or through the glass visor, depending on how the operator wears the device. This is an extremely crucial feature to us, as we want to ensure that patients still feel as though the person caring for them is human, not a robot. As a study by the British Journal of General Practice reports, the second-most common non-verbal signal among healthcare providers is eye contact, with patients reporting positive feedback, such as ”You can see it from the doctor’s eyes that he cares and is involved,” and ”You can feel [the personal attention by] how someone looks into your eyes, not making any notes or writing on a computer at that time; I can see the interest” [16]

We also chose the HoloLens 2 for its lightweight and comfortable design, with an overall mass of 579 grams. Since the headset may be used for calls lasting upwards of two hours, it is extremely important that it remains comfortable for the user to wear for prolonged periods. In addition, the headset's AR capabilities and hands-free gestures provide convenience, making interacting with the headset easy for those unacquainted with the technology.

The headset also features extensive software development support, with many resources available for easy development of AR applications through Unity, using Microsoft's Mixed Reality Toolkit (MRTK) and Visual Studio. Such pre-existing infrastructures permit easy development, debugging, and deployment of programs written for the HoloLens.

5.3 Software & Digital Technologies

Provided that this is where the majority of all work lay for this project, we utilized many various software to construct EMT Vision. Making use of some of the most influential and prominent technologies today, we aimed to design a project that would be not only practical now, but also serve as a strong piece of equipment for the future.

5.3.1 Unity

As a popular video game engine, Unity is the top choice for developing software for the headset. Featuring an easy-to-use HUD, many public assets ready for use, and pre-existing support for developing specifically on Microsoft's HoloLens 2, Unity provides much-needed versatility to the project.

Development for the headset is enabled through Microsoft's MRTK3 packages, which provide the framework for client-based interactions to function in an AR environment (more on this technology in the following subsection). The general architecture for such a project consists of a "Scene," or every asset visible in the running application, and various other nested components. "Canvases" are frames containing UI components such as text, buttons, "toggles" (or checkboxes), and panels. Within these, we can set up the aforementioned UI assets or configure more advanced and directly catered interfaces, including menus, scroll lists, and visual database displays.

While the number of features Unity offers may seem overwhelming at first, it is beginner-friendly and easy to learn. It offers basic development tools for those with little experience and highly advanced tools for professionals and seasoned developers. Every asset in a scene is fully customizable, with pre-developed settings available to alter these components to function uniquely. In addition, if none of the designs meet your design criteria, Unity offers the flexible option to code your own in C#, providing full creative freedom.

Canvases

Diving into specifics regarding Unity project architecture and design, we may explore the varying tools that we utilized to construct our programs. Primarily, the most commonly used components are UI Canvases, which are essentially 2D "screens" digitally projected into the AR scene. These screens may be resized to fit any criteria desired, and are versatile in allowing the developer to select certain behavioral properties, such as always being in the field of view, or acting as an independent 3D object in space, which stays where placed. Canvases act as a baseline for developing other, more advanced components, such as our checklist and JSON displays. This is due to their ”screen” property, which allows us to place other assets within the Canvas, effectively creating a user interface.

Notably, we *raycast* our Canvases in Unity, which means projecting ”rays” from the camera’s location. These may be perceived as light particles that bounce from an object into the eye, which we may know what the user is currently looking at (this technique is most commonly used to render shadows in a 3D space). This method effectively allows the HoloLens 2 to be aware of our rendered objects, and thus permits us to interact with them. If we were not to use raycasting, the headset would lack the capability to be aware of depth, size, and the intractability of the Canvas.

Panels

Panels are another important aspect of Unity components, serving as the parent of a ”visual tree.” Visual trees, in all simplicity, are a collection of UI objects and are reliant on having a parent in order to render. Provided this, panels are a necessary aspect of our design, prohibiting our technologies from being rendered to the user. Further, panels provide versatility in UI intractability, possessing configuration settings for object grabbing and collide boxes. These two are notable aspects of designing a successful AR UI, so we shall explore these in more detail.

Object grabbing refers to the ability of an object to be selected and moved using your hand in 3D space. We may configure certain aspects of object grabbing, such as what mobility is permitted (rotating, scaling, dragging) and gravity-align (Z axis goes unchanged). We utilize these settings to specifically design interactions for objects within the headset, ensuring the most convenient and natural movement of our UI.

Collide boxes are equally as important as object-grabbing settings, and work simultaneously to ensure their function. In short, a collide box is a ”hit” box, or a 3D cube that counts as the volume where an object ”is.” Objects, as are, do not have any 3D space which they are considered to occupy unless specified otherwise. Provided this, we add a collide box to give a volume where we may interact with the object. Namely, this is used for hand interactions such as grabbing and moving UI panels around.

Prefabs

Prefabs are previously made assets, or reusable components within Unity. These may have been designed by other developers, been imported through external packages, or have simply been a method of duplicating an object. Prefabs store game components, property values, and child objects as data, and permit the user to reuse such configurations with ease. Within our project, we utilize prefabs imported through MRTK3’s UI Components, as well as several that we manufactured ourselves.

MRTK3 provides several useful prefabs for us to use, including already-configured and designed buttons, slates (comparable to a text panel), and toggles (checkboxes). While these are relatively simple to engineer, the reality of these prefabs being already created and ready-to-implement made for swift development and user design. Such assets also allowed us to focus more on the back end rather than the front end, and when it came time to develop a user-friendly design, we were able to focus on layout much more than color theory, font styling, and margin sizes.

We also took the liberty to design our own prefabs, including our dynamic protocol and procedure display, which consists of an upscaled JPG image configured with varying interactable and transformable components. These are automatically reused and configured by scripts written in C#, which create an instance of the JPG as an object.

5.3.2 MRTK3

The Mixed Reality Toolkit 3 (MRTK3) is an essential framework for developing the headset's user interface and interactions. This toolkit accelerates AR development on the HoloLens 2 through Unity. Its reusable prefabs, accessibility, and UI/UX additions make configuring the headset easy and flexible. Primary features of MRTK3 include voice commands—which we use for hands-free control of the headset—spatial awareness, hand tracking, and configurable UI assets. This package also contains technologies for eye-gaze tracking, allowing for hover-over effects and selection to occur simply by looking at an object.

We select MRTK3 over other versions of the framework (MRTK2, MRTK4) due to its extensive support system, learning resources, and the quantity and specific selection of prefabs and features available inside Unity. Despite MRTK2 containing unique features that MRTK3 lacks, such as a scrollable menu, and MRTK4 offering prototype features, we opt for MRTK3 due to its reliability, quantity of assets, and efficient development capabilities.

MRTK UI Components (Non-Canvas)

As previously mentioned in the Unity section, we utilize Prefabs to rapidly develop a functional UI with a working back end. Reusing already configured assets saves us time and leads to consistent design choices and formatting. Within MRTK3, many Prefabs are provided which we use, including slates, buttons, and toggles. These are highly configurable Game Objects, despite being initialized with set attributions.

5.3.3 UWP

Universal Windows Platform (UWP) is a common app platform that permits devices to run Windows. Namely, UWP allows applications to run seamlessly on the HoloLens 2, our choice of hardware, without having to install the operating system or implement other workarounds. UWP also possesses the capability of providing direct support to Window's MRTK3 software and APIs, permitting AR applications constructed in Unity to run on the headset with ease.

5.3.4 Visual Studio

Visual Studio serves as the primary Integrated Development Environment (IDE) for coding and debugging the application. Its seamless integration with Unity, built-in support for deploying UWP applications, and powerful debugging tools make it indispensable for developing and testing software directly on the HoloLens 2. Builds (Unity project compilations) are directly deployable to the headset inside Visual Studio by specifying the device's IP and ARM64 architecture, with the software being loaded over a USB connection.

5.3.5 Microsoft Azure

To effectively translate audio files (recorded in .wav for quality assurance as opposed to .mp3, which is compressed), we utilized Azure's Speech Services. Through their online dashboard, we were able to configure a service running on US-West region, permitting fast responses and providing the capability of audio-to-text translation. The speech service, as is, is capable of translating any words into a string, which may then be used in OpenAI prompts. However, provided the unique diversity of Santa Clara County, we opted for language translation as well, which is further supported. The downside of such a capability with this specific service, however, is that the language being translated *from* must prior be specified, which will not work when out in the field. As such, we rely on an AI-powered text service, also provided by Azure, to detect the language prior to transcribing it. From this, we may inform the speech service which language it should detect, which it then translates to English.

Furthermore, we implemented Microsoft Azure's computer vision model to interpret information from the scene. This applies two different APIs each for a unique vision function. The first is a document scanner, which utilized Azure Document Intelligence to read any pdf format which might include tables and handwritten checklists. This helps EMTs quickly extract any medical information that might be found on scene which was a common case that we saw during ride-alongs, when paramedics visit elderly homes. Secondly, there is the pill bottle reader, which detects text in the image that can then be processed by OpenAI API in order to make a list of any prescription medication bottles the first responders might find on scene.

5.3.6 OpenAI API

OpenAI API enables natural language processing for tasks like conversation summarization and text generation. With multiple models to choose from, each featuring strengths and weaknesses, this API provided much flexibility regarding how we analyze our data. Opting for a cheaper alternative that was still computationally smart (and does not hallucinate for our task of choice), we selected GPT-4.

GPT 4

Selected as our OpenAI model of choice, GPT 4 is capable of effectively analyzing and filtering out key data from its passed context. Utilized for eliminating redundant information and filling out patient data forms, GPT 4 serves as an example of the revolutionary technology of AI, showcasing its expanse capabilities opening new doors within the medical world.

We utilize this technology through UnityWebRequests, sending a JSON payload to OpenAI containing a prompt, template to populate, and an authorization token. This, on average, takes roughly 5 to 10 seconds to populate, which when done is then returned back to Unity. We utilize this data to then populate the headset displays and our external database.

5.3.7 Supabase

Supabase is a PostgreSQL database alternative to Firebase, permitting users to employ real-time subscriptions, authentication, authorization, and edge functions with their data. We employed this service to store all of our confidential patient information, being securely managed through several layers of security. With built-in HIPAA compliancy, Supabase was not only a good choice, but a clear one.

We employ Supabase's GET and POST features to receive and send information, respectively, to the database. POST requests are only employed on the HoloLens 2, where data is live-recorded, analyzed, and sent in JSON format to Supabase. GET requests, on the other hand, are both utilized on the headset and our interactive patient dashboard, the visual frontend for analyzing live and historical data from calls. We ensure security for GET requests by hiding them from the client side, implementing this feature by having the client request data from the server, which then calls the GET.

Provided we handle confidential and sensitive identifying patient information, we must ensure that all data is securely stored and managed on the database. To accomplish this, no user is permitted to view any data from Supabase unless they are logged in with a Google account (authentication), and are then a specifically whitelisted user within the organization (authorization). We designed our data interaction stream to work this way so that only registered medical professionals who are designated to work with the data are capable of interacting with it. Further, the HIPAA compliance feature that Supabase offers would further secure data in a way such that it being stored on this service would not violate any patient confidentiality laws (we did not opt to implement this due to pricing).

Information is fetched by passing a project URL and service key into an HTTP request to Supabase. While the database provides either an anon (public) or service role (private) key to access data, provided that we must maintain high-level security, we utilize the service role key for the most secure data transmission and access. These keys are then stored securely in private environment variables, never pushed to any repository or displayed publically for others to access without permission.

5.3.8 NextJS

The entirety of our frontend is implemented in NextJS, an incredibly versatile React framework implemented in TypeScript. This tool has built-in optimizations, dynamic HTML streaming, and permits for advanced routing and real-time database information fetching. Further, NextJS makes it simple to implement a fullstack web application, linking our dynamic UI to our powerful backend services.

NextJS also offers Server-Side Rendering (SSR) and Static Site Generation (SSG), which work together to provide efficiency and up-to-date data. SSR is the primary feature used on our webpage, working to ensure that patient data is always up-to-date on any page displayed. This process works by rendering the page at request time, then sending the rendered HTML to the client. Doing so thus displays the most recent information from the database, and combined with a refresh of every five seconds, we ensure that live data is always updated without the need to reload the page. SSG, on the other hand, is used for static web pages such as our login page, which does not require fetching any patient data and never changes based off of database information.

5.3.9 Vercel

We utilize Vercel to deploy our website to the internet, allowing other users to remotely connect to our services. Utilizing this tool, we may publish our most recent changes to be viewed publicly with ease, as we have configured Vercel to watch and read from a public git repository. Once changes are detected, it begins a ”build,” a term for compiling, packaging, and optimizing code written. This also serves as a quality control, ensuring that no faulty code or fatal errors may make it to production.

If a build passes, the new scripts are instantly pushed to the production deployment, visible for other users to interact with. We may also view user traffic, debugging information, and other data regarding our web service from the Vercel portal. Provided that this web service is free, it made for a great way to deploy the dashboard for our project, while still ensuring quality and the capability of accessing patient data from any location.

Chapter 6

Software Design

6.1 Architecture

The entirety of the software written for the HoloLens 2 is written in C#, whereas all of the frontend patient dashboard is written in the Next.js, a TypeScript-based framework. To ensure an efficient design and developmental process, we desgined flowchart diagrams to serve as blueprints for the project. The below architecture diagrams showcases the low-level and high-level overview of our headset and website design.

6.2 High-Level Architecture

The High-Level architecture for our product (both the patient dashboard and headset) is relatively simple—users will equip the headset and commence recording through an interactive dashboard. This HUD will be rendered on the headset's visor, permitting for both real-life and digital viewing. Upon a recording being started, we will store both the entirety of the conversation in memory, along with a seperate file for key words.

Once the recording has concluded (upon the user's request), we send the two conversations, along with a prompt, timestamp, and patient ID, to our AI agents. These return back formatted data which is then forwarded to our database, and rendered on both the patient dashboard and the headset's HUD.

Users utilizing the headset may also view county procedures via our protocol menu, which is attached on the right side of the patient recording menu. These are customizable, and may be configured per each county's standards.

Medical professionals and EMTs alike may access our patient dashboard website, which is protected by a secure authorization and authentication process, temporarily using Google's OAuth for login purposes. If a user is permitted to access the website, they may view incoming patient information and revise it as they see fit. This is all conducted on the web, as we have no mobile app in production.

6.3 Low-Level Architecture

Low-Level Architecture showcases in great detail the interaction between several technological components within software. We expand on the architecture for both our patient dashboard, as well as our headset's functionality.

6.3.1 Patient Dashboard

Below you may view Figure 6.1, showcasing the Low-Level Architecture for our patient dashboard website. Notably, it may be exhibited that the user first connects to the login page, which then handles the process utilizing Supabase's Auth, routed through Google Cloud Platform (GCP)'s OAuth. Once the user has been authenticated, they are then authorized if they are a set whitelisted user, at which point their connection is routed to the dashboard page.

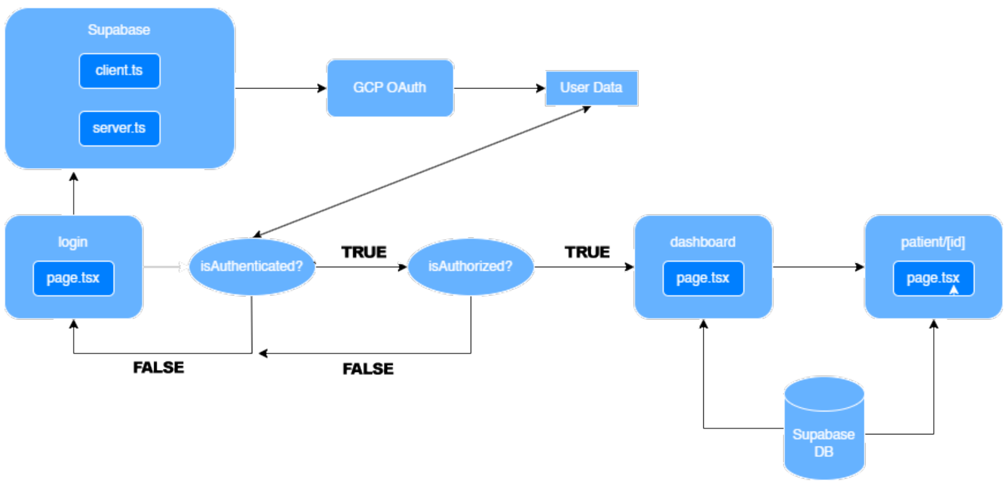


Figure 6.1: Low-Level Patient Dashboard Architecture Diagram

6.3.2 HoloLens 2

Below is a diagram showcasing our headset's architecture in a low-level view, displaying the varying technologies we employed both internally and externally to ensure a fully automated system of patient data population.

External tools include OpenAI's GPT-4o and Supabase DB, whereas Azure Speech Services were actually loaded internally onto the headset. Such dependencies require us to maintain an internet connection at all times while using the headset—a limitation to our design, though a necessary one whilst using AI.

On the left of our diagram is the headset's frontend—the aspects visible to the user—and on the right is the backend, or the parts hidden from sight. Our product functions by recording all audio once initiated by the wearer, then storing this input in a buffer. This is then sent to the Audio Manager, which processes the information to a string and, ultimately, a JSON format readable by our database. It is sent there upon completion, where it can be accessed by our patient displays. We do provide the capability to revise pre-existing patient records, which may be done through selecting a patient on the patient list and pressing "record" (not detailed in the diagram below).

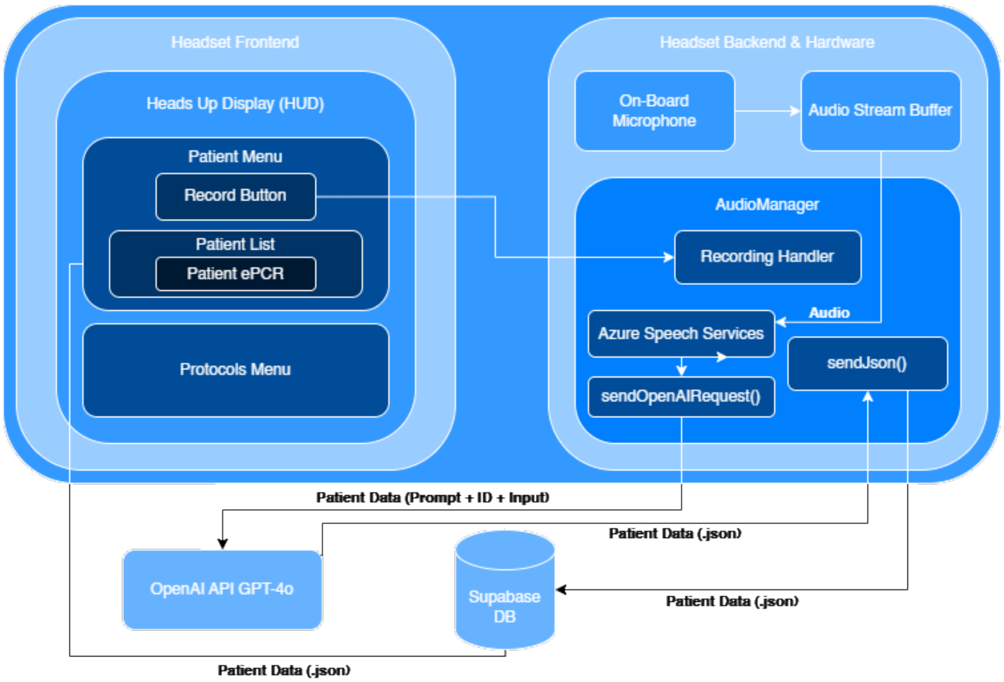


Figure 6.2: High-Level Headset Flowchart Diagram

Chapter 7

Risk Analysis

Provided the complexity and quantity of layers that will comprise this project, an expanse risk analysis must be conducted to evaluate rather certain features should or should not be implemented, and if so, the amount of time and resources which should be appropriately allocated toward solving each one. In doing so, we acknowledge the inherit nature of software development; its risks, rewards, and trade-offs amidst developing and designing an advanced technology. Through this methodology, we ensure the consistent progression of our project and that we do not waste resources or time on an unnecessary or excessively risky aspect of the design.

The risk analysis that we provide in the table below (Table 7.1) showcases our analysis of the tasks we deemed to be notable risks within our project, according to the probability, severity, impact, consequences, and mitigation for each individual risk. Risks are predicted and analyzed based on their theoretical occurrence of the product post-development, as it enters the production lifecycle. Severity is rated on a scale of 1–10, where 1 is the lowest risk and 10 being the highest, and impact is calculated as the product of severity and probability.

Risk	Probability	Severity	Impact	Consequences	Mitigation
Cannot provide HIPAA compliance	0.5	1	0.5	Will prevent the product from reaching a production level serving others in the field.	Obtain funding for HIPAA-secure database and auditor.
Unable to read headset in bright environment	0.9	5	4.5	Screen becomes unreadable in daylight, severely limiting use in outdoor emergency settings.	Implement voice-driven UI and high-contrast visual themes.
Network connectivity	0.5	7	3.5	Cannot connect to internet which inhibits the usage of AI processing.	Have a backup hotspot available for times that ambulance WiFi is unavailable.
Battery capacity	0.9	2	1.8	HoloLens 2 cannot function for extended durations of time without battery recharge.	Integrate a non-obtrusive cable or battery system to the headset.
Cannot guarantee patient confidentiality	0.8	2	1.6	Cannot conduct real field tests with paramedics.	Conduct simulated tests in substitution of authentic ones to ensure the functionality of device.
No hospitals or paramedic departments are willing to collaborate	0.7	6	4.2	Inability to collect real-world feedback and validation data.	Seek partnerships through university networks and offer co-authorship in research publications.
Thermal overheating of headset during prolonged use	0.6	4	2.4	Device shuts down during emergencies, interrupting operation.	Monitor temperature and design for duty cycles with cooldown intervals.
Insufficient onboard storage for logs and audio	0.5	3	1.5	Limits amount of patient data that can be saved locally on RAM.	Implement a periodic sync and deletion mechanism to clear old data.
Inaccurate voice transcription from strong accents or dialects	0.6	4	2.4	Misinterpretation of patient data or history.	Use multilingual transcription models that have been trained on accents.

Table 7.1: Risk Analysis Table

Chapter 8

Testing

8.1 Beta Testing

After roughly a month of development, we felt it adequate to attempt and test our software on the physical headset—prior, we had been testing solely within Unity's developer application. Provided that our program had been working and properly analyzing microphone input, we assumed that it would not be a challenge to ensure it worked on the headset.

Unfortunately, this was not the case, as a brief test revealed that not only did our software malfunction, but the language we had been coding our software in (Python) was not supported by the headset's operating system [18]. This was an enormous setback, and to tell the truth, sort of a wake-up call for the team, as this was an easily avoidable mistake caused by a lack of preliminary research. To add insult to the injury, our audio analysis API also was unsupported by the headset, given that it was developed in the same language as our now paralyzed software.

Provided that we had no working program at this point, we hastily delegated work among ourselves to release a patch, entirely rewriting our scripts from Python to C#. We also resolved issues with an outdated framework for the headset, migrating our Unity project from MRTK2 to MRTK3 (Microsoft's Mixed Reality Toolkit). These changes, along with some simpler bug fixes (such as resolving an issue where the microphone worked only on the computer, but not the headset), were encouraging results and led us to develop our first functioning prototype.

8.2 Prototype Testing

For initial prototype testing, we designed test cases that would ensure the proper functioning of basic components, such as audio translation, data population in the dashboard, and the functionality of the protocol menu, along with its ability to retain memory of checkboxes that had been marked. Prototype testing did not include edge cases or stressing the system in harsh environments, such as loud environments, multiple patients being recorded at once, or any other confusing stream of audio.

8.2.1 Prototype Results

As a result of our initial testing, we found many bugs that were essential to be patched prior to a production deployment. Some of these included the incapability to retain memory of object interactions in Unity (such as what back to navigate ”back” to and marked checkboxes). Further, our initial audio processing was solid, but we discovered that as the total conversation duration increased, our recording length decreased, ultimately resulting in lost data and incorrect analysis. These issues, paired with a poor GPT-4o prompt, resulted in poor population of the patient dashboard, rendering our analysis of audio inaccurate and useless.

Provided the severity of the errors discovered during initial testing, we hastily issued patches for the bugs. Resolving memory with Unity objects proved quite simple (storing in local memory within a GameObject script), with the audio processing issues requiring more time to fix. To resolve the problem of incorrect audio recording, we pivoted to recording the entire conversation, then processing at the end. While this not only fixed missing data (likely due to overlapping recordings), it also improved the total cost per patient, provided that each GPT-4o call costs a small amount of money to run. Combined with a lengthy, in-depth prompt sent to OpenAI, our results became highly accurate, going so far as to only replace incorrect data and retaining smaller, often missed pieces of information.

8.3 Simulation Testing

To obtain near-real life results, we constructed several simulated calls to extensively test the headset's capabilities in loud environments with unclear audio inputs. While initial testing covered the basic functionality for a quiet, calm environment, we had yet to test a scenario in which multiple people are talking over each other or the patient were to be unresponsive.

Each test case was given a score out of 100, which depended on the headset's accuracy in obtaining correct information from a pre-determined (and randomized) list of patient data. Five scenarios were run for each test case, and the percent accuracy for each field of data collected was then averaged over these trials to obtain the overall score for the experiments.

8.3.1 Test Case A: Multiple People Speaking

To simulate this test case, we had a simulated ”paramedic” and another as the ”patient.” Two other people then maintained a verbal conversation in the adjacent background. The purpose of testing this case was to see if the headset could accurately maintain awareness of the correct, intended information being acquired by the microphone amid other voices.

The results we obtained for this trial were phenomenal, and much higher than our expectations. The accuracy rating for multiple people speaking at the same time was an 89%, mainly due to the headset's onboard microphone's capability of filtering background noise.

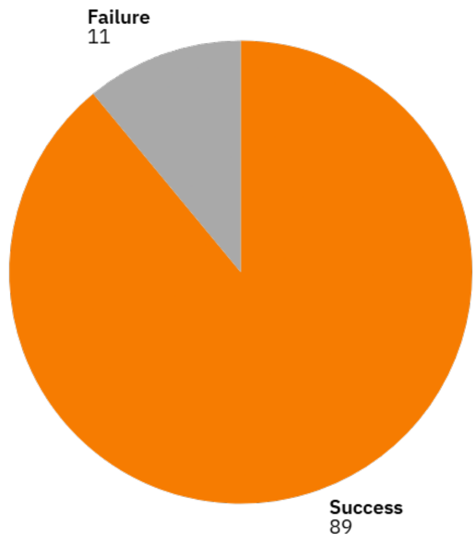


Figure 8.1: Accuracy score for Test Case A: Multiple People Speaking

8.3.2 Test Case B: Noisy/Loud Environment

Simulating a noisy environment included testing the headset outdoors, with a car idling nearby, and audio of both traffic and crowd chatter being played loud mere feet from the headset. The purpose of this test case was to see if the headset was capable of functioning in realistic outdoor environments, with a variety of sounds that could influence the ability to record decipherable audio.

These results came back high, scoring a 94% accuracy rating in an environment of 80dB or higher. This was quite surprising, as we did not expect the headset to be able to analyze entire words when speech was audibly impacted by significant background noise. These results were also very encouraging to us, primarily given that emergency scenes are often loud, outdoors, and full of noises that could impair audio recognition.

8.3.3 Test Case C: Incorrect Data Recovery

For our third test case, we would occasionally state incorrect data to the headset in reference to the patient. Following this, we verbally indicated that we had provided the wrong information and informed the AI to correct it with new data. This trial was, unlike the other two, going to be entirely dependent on our software and AI prompt implementation, as we are not impairing audio recording in any way.

Similarly to the other tests, the data was incredibly promising, outputting a 98% accuracy rate. This result was crucial because patients may be disoriented, scared, or incomprehensible at times, so the ability to later adjust the data

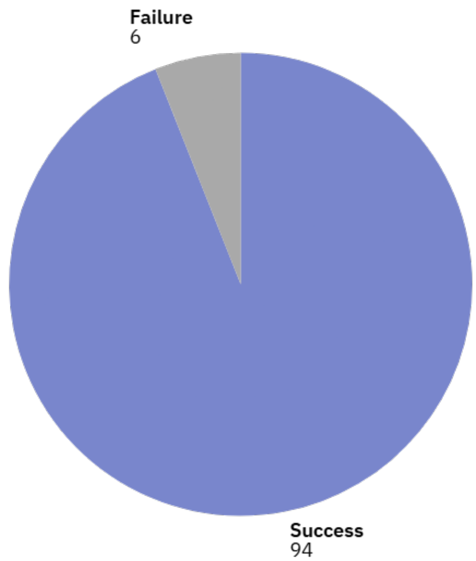


Figure 8.2: Accuracy score for Test Case B: Noisy/Loud Environment

fields enables immediate corrections as new information is provided. The results for Test Case C were our highest of the three, scoring a 98% accuracy rating for being capable of differing between incorrect and correct data. When instructed that input data was wrong and needed revised, the AI was able to successfully replace and select the right information needed for each field.

8.4 Theoretical Field Testing

Provided the many restrictions regarding patient data recording and storage, we were unable to receive permission to conduct real-life field tests during the academic year. However, had we been able to pass the regulations regarding these criteria, we would have liked to test the headset onboard ambulances and on scenes. Due to the device requiring an internet connection to function, we would have liked to see how the headset performed in an outdoor environment, and if it could have been capable of functioning on the rig's WiFi or a hotspot.

We also would have loved to conduct data analysis of patient data, potentially exploring pattern recognition of recurring symptoms and treatments. Had we been able to study realistic input data, we potentially could have caught illnesses, injuries, and suggested remedies prior to human detection.

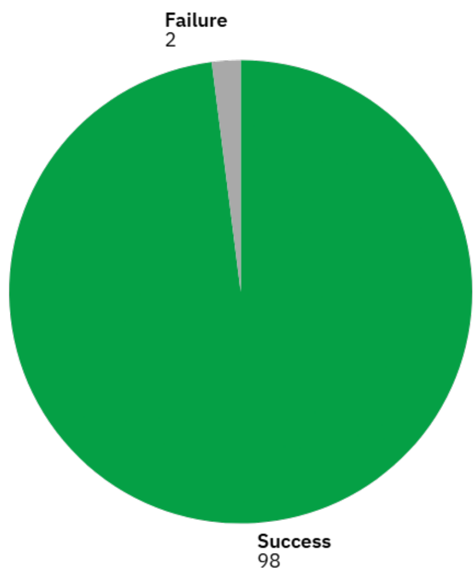


Figure 8.3: Accuracy score for Test Case C: Incorrect Data Recovery

Chapter 9

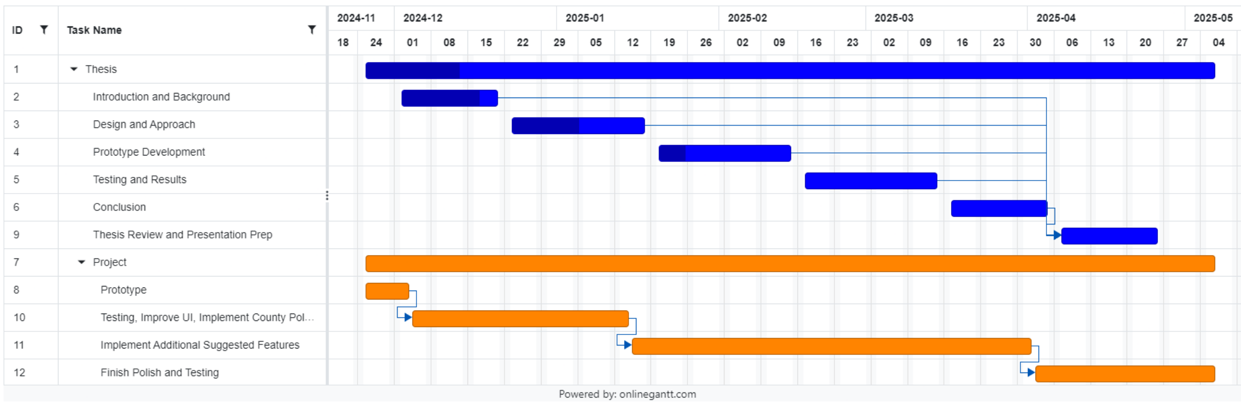
Schedule

9.1 Overview

Developing our multi-phase application is very complex, requiring planned stages of research, development, field-testing, and debugging to guarantee that goals and expectations are met. Consequently, our project required a well-planned road map to allow us to remain on track while also allowing flexibility as setbacks occurred. To aid in visualizing this process, we provided a detailed outline of our initial development process timeline, including, but not limited to, the steps for writing our senior thesis, the software integration on the HoloLens 2, and project testing with Santa Clara first responders.

9.2 Gantt Chart Overview

The chart is organized into two sections: thesis and project. Both of these deliverables are organized by milestones within a desired time frame. The thesis sections are all independent of one another, requiring less coordination to organize together. We planned each phase of this to demand a similar time commitment. However, we ultimately did not develop the thesis in the same order of categories, primarily due to the improved efficiency of delegating these chapters as we completed the corresponding component of the project. Although this approach reduced organization from the ”one component at a time” ideology, it ultimately allowed for better depth and detail while the topics were current. Meanwhile, the project followed a phased linear methodology with iterative refinement, resulting in a similar pipeline to the waterfall model. We split the project into key functionalities, including prototype, testing and feedback-based improvements, additional features, and final polish. The prototype consisted of core functionality required for a minimum viable product (MVP). This meant audio recording, analysis, and display on the headset with a basic interface, although this timeline was expanded through February due to unexpected software constraints that limited expected development. After this was functional, we began creating a more interactive UI, implemented a database, and started testing to discover potential improvements and shortcomings with our early design. This milestone transitioned into our additional suggested features phase, where, after visiting Santa Clara Fire Department once again, we embedded county policies into the headset and developed a web app portal for external access to the data. Our final stage was a last round of polishing, bug fixing, and testing, ensuring our product was seamless to use, reliable, and simple. This provided valuable, subtle optimization to fulfill our initial goals of integration into the often not technologically savvy firefighters' routine equipment.



9.3 Setbacks

As previously alluded to, our project development experienced multiple key setbacks that postponed expected deadlines. Although we ultimately managed to adapt and achieve our primary goals, such challenges led to additional features being reduced or removed to still satisfy our vision within the given time frame. Our first was the limitation of Python being incompatible with UWP, the development framework of the HoloLens 2, preventing our planned dependencies and scripts from being used. As a result, we shifted towards supported programming languages like C++ and C# and external API available through these languages, specifically OpenAI and Azure Speech Services, while rewriting existing scripts to incorporate these changes. Another complication was that Hololens-specific Unity imports were needed to access device functionality. While simple in theory, these often lacked clear documentation for different versions, dependencies, and compatibility. As a result, we spent several weeks researching compilation errors and deployment requirements specific to the headset, particularly since these issues did not occur in the Unity desktop environment. This presented a recurring obstacle throughout the development cycle and significantly extended our debugging timeline. As a consequence of these major setbacks, certain features we had hoped to include, namely automatic language translation and cloud-based data analysis, were unfortunately cut from the project.

Chapter 10

Constraints and Standards

10.1 Constraints

Developing a medical device poses several constraints, restricting many potential features and leading to other workarounds for problems that have no realistic solution. In the development of EMT Vision, we faced several challenges through technical and legal constraints.

10.1.1 HIPAA

The strict legalities regarding patient confidentiality and data analysis present a challenge to this project, which prevents us from testing the headset on real calls. This is due to compliance with HIPAA, which states that if information that can be used to identify a patient is discussed, a breach has occurred in the Privacy Act. To combat this, we would have to develop our own AI model, so that any recordings containing potentially sensitive identifying information would not be sent to an AI model to be trained on.

10.1.2 Hololens 2 Technological Limitations

Technological limitations greatly impact this project. While the Hololens 2 is a very capable tool, it still has its limitations. The headset has limited processing power and battery life, as well as being a costly piece of equipment. and is costly

The HoloLens 2 is severely limited by a few defining hardware designs, most so by the battery life. While the battery is capable of running for 2-3 hours, due to the resource-intensive software running on the headset, it tends to die in under one. This has imposed a challenge of the headset being capable of running for the entirety of a call, and thus led us to develop external hardware to extend battery life.

Processing power is also a leading factor in what we may or may not implement on the HoloLens 2. With 4GB DRAM, we are incapable of running many powerful AI models locally on the headset, and thus must connect via Internet to external libraries. This, as a result, lowers the efficiency of our software as transmission and propagation delay must now be factored in to the total run time of AI computations. Smaller hardware constraints include the HoloLens' weight, which while light, may tire the user's neck or head after extended usage, and thermals, as the headset can often get warm with prolonged use.

Software constraints for the headset include the platform on which it runs: UWP, which severely limits the flexibility and liberty of selecting which tools we use to develop the program. For example, Python is not natively supported, and thus could not be used for this project (this also limited which libraries we could choose from). We are also limited by the headset's 64 GB of storage, which leads us to develop highly efficient and memory-compact algorithms and procedures.

We also had to adhere to real-time processing, and as such, the headset posed some constraints on this feat. Given the need for fast responses, a strong preference was developed to run ML models locally on the HoloLens 2. However, its RAM and GPU severely restricted our capabilities of doing so.

10.1.3 AI Bias & Fairness

Given that our software actively utilizes OpenAI's GPT 4o mini to analyze conversational data, we had to conduct a multitude of testing to ensure there was no AI Bias toward any demographic or minority. This included the accurate transmission of data amid varying accents, dialects, and culturally related information (such as ethnic names).

10.1.4 Accessibility (WCAG 2.1, Section 508)

A notable constraint that we adhered to was accessibility, and ensuring that our device would be usable by paramedics of all kinds. Provided that many are not extremely knowledgeable with technology, let along more advanced concepts such as AR, we intentionally implemented user-friendly and easy-to-learn designs into our software. Verbal commands and hand gestures work together to ensure such accessible usage for those who may not be familiar with our technologies.

10.1.5 Data Security & Encryption

Provided that our device works primarily with medical patients, it was a top concern and priority that sensitive information would be kept secure once obtained. Given that we cannot leak any identifying information, we opted to keep our project in the testing stage, as we did not have enough time to develop our own AI model, ensuring data security. Further, an encrypted method of sending data to and from the headset and local server was rather challenging, and posed as a large constraint for our project.

10.2 Standards

When designing a medical device, it is crucially important to adhere to the many policies, rules, and laws regarding patient confidentiality and safety. To ensure the privacy of patient data and adhere to a multitude of technological policies, we adhered to the following standards.

10.2.1 IEEE Standards

The IEEE Code of Ethics is a framework of principles provided by the IEEE that promotes responsible and socially safe technical solutions. Due to the headset overlapping with life-or-death situations, the AR software must ensure the ability of first responders to provide aid to patients without inhibition or unauthorized sharing of patient data (e.g., HIPAA or GDPR).

- **IEEE 1012 V&V**: Individual software modules must be tested before integration to reasonably expect the application to run as expected, which includes components such as speech recognition, data retrieval, and AR visualization. In validation testing, the project should be conducted under simulated field trials with acceptable effectiveness before human trials are begun. As testing progresses, risk testing must account for the dangers of system failures, such that offline access or manual alternatives to AR capabilities are always available to fall back on if needed.

10.2.2 ISO Standards

Due to the nature of our project being a medical technology, we must comply with and follow several medical standards.

- **ISO 14971 (Risk Management):** This standard requires us to identify sources of risk and implement control measures. Due to this, we identified several patient confidentiality risks and selectively chose our technology stack in a way that would be most secure. While we did not offer HIPAA compliance, we did select frameworks that would allow us to offer it in the future, albeit with a bit of extra funding.

- **ISO 62304 (Medical Device Software):** Outlines software that is a medical device, software that is used in the production of a medical device, or software that is embedded in a medical device. In our case, the second and third apply to EMT Vision, given that we serve patients on calls. This gave us a framework to design our architecture with.

- **ISO 13485 (Quality Control):** Our headset is intended to be used in the field. While it will not be deployed this year, this standard should still be followed so that the project may be ready to deploy as soon as possible.

- **ISO 13482:** Outlines safety requirements for personal care robots (provided we are engineering an AR headset, this does indeed apply). Had we removed AI from the project, we could potentially get around this ISO standard, but due to our reliance on GPT-4o for populating data forms, we had to abide by several safety standards from ISO 13482.

- **ISO/IEC 27001:** Details information security management (for security and privacy). This revolves around ensuring patient data remains safe, secure, and confidential, which was a primary factor in preventing us from conducting field testing.

10.2.3 HIPAA

HIPAA establishes standards that protect confidential patient information, ensuring each individual has privacy, maintains trust with healthcare professionals, and removes the risk of disclosing patient details without consent. Due to our project serving as a technology in the medical field, we must adhere to HIPAA's strict patient confidentiality procedures and standards, ensuring that all sensitive and identifying information is only disclosed to those operating on the patient.

To ensure HIPAA compliance, an auditor along with secure certified data storage must be present, both of which cost a plentiful sum. Due to our lack of funding to pay for a full-time salary, we did not opt to implement HIPAA compliance, but we did select frameworks that would allow for us to implement it in the future with minimal effort. Additional considerations for HIPAA compliance would include censoring patient data upon AI input and ensuring that all data is encrypted and decrypted securely upon being transmitted.

Chapter 11

Societal Issues

11.1 Ethical

Within our project, the usage of AR headsets, particularly with audio recording capabilities, offers a variety of positive improvements to EMS communication. However, such features also pose several significant ethical concerns regarding patient and user privacy that must be carefully addressed.

11.1.1 Patient Privacy and Confidentiality

The primary ethical concern is the protection of patient privacy and the confidentiality of their data. HIPAA, along with other public protection safeguards, restricts the access and acquisition of patient data. Recording paramedic-patient interactions could lead to unintended privacy breaches, particularly if audio data is stored insecurely or accessed without proper authorization. Ensuring data is kept locally and that uploaded data is adjusted to maintain patient anonymity are critical measures to mitigate these risks. In particular, we retain all audio recordings locally on the device, and dissect JSON conversation transcripts of names and other identifiers, ensuring patient privacy.

11.1.2 Informed Consent

Naturally, the inclusion of recorded data poses ethical challenges regarding receiving the appropriate consent from the sources involved. Often in emergency scenarios, patients may be unconscious or disoriented, restricting their ability to reasonably provide consent. Moreover, in high-pressure situations, paramedics may not have the time or ability to explain the recording process. Even in non-emergency circumstances, the headset does not automatically notify people around it or the paramedic when it is recording, so patients may not be aware their data is being acquired. To address this issue, it is imperative to provide indicators to identify, for clarity, when the device is actively recording the paramedic and others involved. We have considered an external LED “active recording” marker, as well as an icon on the user’s HUD.

11.1.3 Ethical Responsibility of Data Usage

The usage and storage of recorded data present further considerations. The team must determine if the recordings should be used solely for documentation and communication purposes, or if could they be leveraged in research. Would there be an opportunity to ethically train machine learning with real patient interactions or is using even a written transcript of the conversation ethically inappropriate? Ethical guidelines must be upheld to prevent misuse of data and maintain the inherent trust between patients and EMS. If any data is analyzed, we will confirm that all unique patient identifiers will be separated in compliance with HIPAA. Any data within the headset should be carefully monitored to confirm temporary data is removed and that stored data does not pose reasonable security vulnerabilities or potential data leaks.

11.2 Social

An incautious implementation of AR headset audio recording technology poses social risks to both paramedics and the broader community.

11.2.1 Trust Between Patients and Paramedics

Patients place immense trust in paramedics to provide immediate, accurate medical care. The introduction of audio recording may be perceived as intrusive, especially if patients are not aware of what data is being acquired and the limited capacity it is used. While some may view it as an enhancement to patient care and accountability, others may see it as an infringement on personal privacy. Patients are often in vulnerable, traumatic states and may not want this view of themselves to be retained without prior consent. California is incredibly diverse, containing millions from different cultures, backgrounds, and communities, all of which may have altering impressions towards AR usage in an emergency. Additionally, if the headset is only available in wealthier districts and communities, it could widen the disparity in healthcare between higher and lower economic areas. If not acknowledged, this could erode the existing positive rapport between healthcare providers and patients. Transparency in communication about the purpose and benefits of recording is crucial to maintaining public trust. Stored data should be filtered of personal details and kept highly secure, as any breach of data could jeopardize the entire public trust in EMS. In addition, the inclusion of an AR device worn on a paramedic's face could reduce the sense of personalized care or human interaction. Thankfully, the HoloLens 2 is low-profile relative to comparable hardware on the market and enables users to engage intimately with eye contact.

11.2.2 Workforce Acceptance and Adaptation

The use of AR technology in EMS may be met with resistance from paramedics due to concerns over increased surveillance, performance evaluation, or legal liability. Paramedics we met with expressed this, although the project's focus on capturing audio as opposed to video helped reduce these fears of department oversight. Still, the unease about recording misuse as a means for disciplinary action will have to be overcome. Proper training, organizational support, and clearly defined policies on how recordings will be used are essential to encourage adoption among EMS personnel. Additionally, through exposure to the application, EMS will discover and hopefully appreciate the improvements to communication and documentation.

11.3 Political

Government regulations and policies play a critical role in the restrictions and acceptable capabilities of AR headset technology, especially in EMS.

11.3.1 Legislative Barriers

The legality of recording medical interactions varies by jurisdiction. Some states and counties require dual-party consent for recording, while others allow single-party consent. Specifically, California requires all-party consent for private conversations (i.e. within private domiciles or businesses), although this does not extend to interactions in public. Further complications arise from the duration of data retention and whether there should be a simple means for patients to request the deletion of their data. Ensuring compliance with local and national laws before deploying this technology is essential.

11.3.2 Potential for Legal Challenges

The existence of recording devices in medical settings may lead to legal disputes, particularly concerning malpractice claims and liability. Specifically, the question can be posed of who owns the recordings between the paramedics, the EMS department, or the patient. Utilizing the ethical frameworks for the use and storage of recordings discussed previously under Chapter 8.1: Ethical Considerations can help mitigate these risks.

11.4 Economic

The financial impacts on fire departments of developing, implementing, and maintaining our AR headset application must be assessed.

11.4.1 Cost of Implementation

Deploying AR headsets across EMS units could involve significant costs, including potential hardware acquisition, software development, and training programs. Within the organization, a cost-benefit analysis could be required to justify the investment.

11.4.2 Financial Sustainability

Beyond the initial investment, ongoing costs such as device maintenance, IT infrastructure, and data storage must be accounted for. Exploring funding options, including government grants and private-sector partnerships, may be options to offset these expenses. Alternatively, departments may already have funding channels to devote to investing in upcoming equipment, and ongoing costs could be relatively low, but departments would need to be convinced the benefits outweigh initial as well as continued costs.

11.4.3 Potential for Cost Savings

While costly upfront, the technology could lead to financial savings in the long run by improving efficiency, reducing paperwork, and minimizing medical errors. In addition, as AR headset hardware continues to become cheaper, more efficient, and more compact, the cost of integrating AR capabilities will decrease.

11.5 Health and Safety

Ensuring the application's use expands the safety of both patients and paramedics is fundamental.

11.5.1 Impact on Paramedic Performance

AR headsets must be designed to reduce a paramedic's cognitive load. Poorly designed interfaces or excessive alerts may distract and hinder rather than assist paramedics. Information panels must be unobstructed, and buttons and menus should be quick and reliable to navigate. Paramedics should at all times be able to observe their surroundings for potential dangers and preserve focus on the patient. Paramedics may also experience constant pressure from being recorded and consequentially could undergo stress or altered behavior. These outcomes need to be mediated considering a natural, calm demeanor is necessary for the paramedic to soothe the patient and instill confidence in their ability.

11.5.2 Enhanced Documentation for Patient Safety

Accurate audio documentation could help reduce errors in medication administration, treatment protocols, and patient handovers, ultimately improving patient safety by providing hospitals with optimized patient reports. However, if the data is inconsistent or unreliable without an EMS proof-checking, it could increase errors, especially if paramedics rely solely on or become too dependent on the technology.

11.6 Usability

To be effective, the technology must be easy for users to utilize, especially to avoid distractions or missteps as paramedics provide aid to patients.

11.6.1 Intuitive Interface

The AR system should be simple to navigate in high-pressure situations. The layout is simply designed to focus on crucial information, navigation buttons, and intuitive actions. The HoloLens 2 utilizes hand-tracking instead of controllers, allowing users to grab, select, and drag panels akin to a smartphone, improving beginner usability. Navigating to a specific section should be accessible through minimal clicks and the option to reduce or remove the HUD will be seamless.

11.6.2 Minimal Training Requirement

The system should be designed for near-immediate adoption and only require simple training. Users should be able to understand and navigate the application intuitively, needing at most a few initial hints or instructions. The goal is to ensure that paramedics can operate the system effectively on their first use without extensive training, so instincts on calls can enable the user to operate effectively without a notable cognitive impact.

11.6.3 Customization

Optimization is key to ensuring efficiency in a fast-paced medical environment. Paramedics should have the ability to adjust the interface to suit their workflow, including options to move, scale, and minimize panels as needed. This level of flexibility allows users to customize their experience based on personal preferences and situational requirements, enhancing comfort, operational effectiveness, and overall satisfaction.

11.6.4 Hardware

The HoloLens 2 needs to offer reliable performance and security for the user. The hardware should be tested to ensure it continues to function optimally under extended use or during prolonged calls. If the device's internal battery is not able to reliably last under simulated conditions, an additional solution such as an external battery pack may be needed to allow usage throughout the day.

11.7 Compassion

An important aspect of engineering is developing solutions with awareness and sympathy towards the misfortune of others. It is, as a result, the role of engineers to work with compassion, which was a reason our team was drawn to the project. Paramedics face an array of obstacles when attempting to provide compassionate, empathetic care under stressful conditions. By enhancing documentation and communication, we hope to alleviate some of the suffering in our community among patients and those who serve them. Through this project, we hope to inspire further research into improving the technology of EMS and reducing the stress from a naturally intense environment.

Chapter 12

Final Product

12.1 Differences Between Prototype and Finalized Project

In our initial prototype, EMT Vision solely consisted of an audio interpreter and interactive protocol display. While complex in nature, we found ourselves to have more time—and resources—on our hands than originally perceived, permitting us to develop additional features and upgrade user experience beyond the initial design for the project. Further, we developed several new technology stacks and interactions between varying APIs and libraries, leading to some stark differences between our initial prototypes and finalized product.

12.1.1 Technological and Design Differences

Within our initial design, we conceived an architecture for audio interpretation that consisted of Python-based scripts sending and receiving HTTP requests to OpenAI's GPT4o-mini and Whisper. However, upon the commencement of development, we found that Unity posed integration troubles, and more severely, the HoloLens 2 prevented any usage of Python scripts. This was a significant setback, sending us back to the drawing board, and leading us to revise our architecture. From this, we proceeded to our currently used C#-based technology stack, comprised of GPT-4o and Azure Speech Services (as Whisper does not run natively on C#).

Our prototypes for the interactive protocol display involved recreating the entire mobile app within the headset and redesigning the procedure menu to be a set of buttons within a scrollable viewport. However, when it came to implementing this feature, we found it not only redundant but hard to use on the headset, leading us to develop our current system, which displays the official city PDFs, navigable through selective buttons. While this design is somewhat similar to the prototype, we did not opt for a scrollable view; rather, we chose an iterative page menu. Further, instead of writing each individual bullet point and treating it as a game object, we rendered checkboxes over PDFs, improving format and accessibility for those already familiar with the protocol sheets.

12.1.2 Feature Differences

Within the early stages of prototyping our project, we outlined a few features that did not make it to production. One of these, most notably, was the capability to present potential patient diagnoses through analyzing a live video feed. While this technology certainly would have been impressive, we concluded that it would fall significantly short of being revolutionary or impactful, mainly due to ethical and error concerns. Due to these, along with discussions with local paramedics, we decided to eliminate this feature.

12.2 User Guide

Our technology was engineered with accessibility as a forefront priority, making it easy to navigate, read, and analyze patient data. Both the perspectives of the EMT and licensed medical professionals offer simple user interfaces, permitting for efficient usage.

12.2.1 Headset Manual

To begin with the Holo Lens 2 headset, log in with the provided credentials and launch the EMT Vision app. From here, you will be prompted with several menus, of which we shall elaborate how to navigate.

BLS/ALS Protocols Menu

The first menu, the BLS/ALS Protocols Menu, showcases various procedures and protocols provided by Santa Clara County. These would be the same protocols visible within the SCC EMT app, and as such, navigating this menu is identical to the structure you are already familiar with. To begin, please select a broad category of the patient being treated (adult, pediatric), then select a scenario. If your selection does not appear on the menu, please utilize our up and down arrow buttons to navigate through multiple pages.

Once selected, the protocol for that scenario will appear on the right, being movable through a ”pinch-and-drag” motion, utilizing the hand. Simply grab and place the object where you would like it as if you were physically interacting with it. If the protocol has multiple pages, similarly to the menu, please utilize our left and right arrow keys to move through the pages. You will also view checkboxes next to each bullet point. These are for you to track your progress throughout a call and are intractable if you click on them through a tapping motion with a finger.

Figure 12.1: View of the Protocol Menu in Unity. The user can navigate through the protocols and their respective PDFs using the labeled buttons on the left.

Audio Recording and Analysis

To initiate audio recording, simply "tap" on the record button, visible on the left side of the user menu. This will commence an active recording utilizing the on-board headset microphone, so be cautious so as to not accidentally trigger this action. Once you have completed your call or wish to take a break, simply press the stop recording button, at which point in time the full-length dialogue will be processed and sent to the patient dashboard. You will be able to tell this is done from the patient data visibly rendering on the patient list, visible next to the record button.

If you wish to view patient data or resume recording for a patient, select the desired name from the list of patients, at which point all recorded information will be visualized for you to view. To resume recording, press the record button on the patient page to record specifically for this patient.

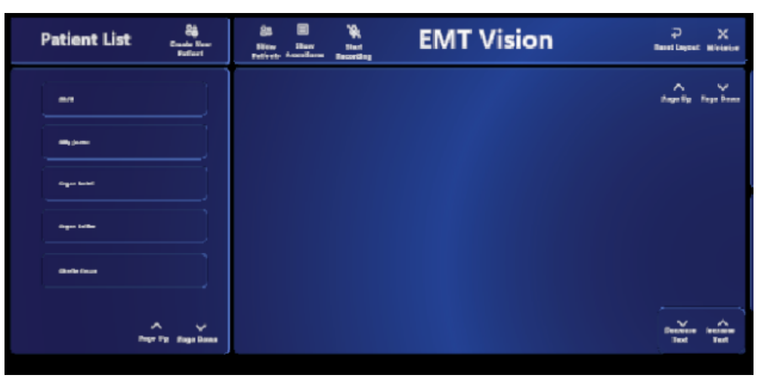


Figure 12.2: View of the primary menu in Unity. The left panel contains buttons for switching between patients while the center panel is where the information from the recorded conversation with a patient will appear.

12.2.2 Patient Portal Manual

The Patient Portal is designed for medical professionals to operate at regional hospitals. To ensure patient confidentiality and the security of information, please first sign in to your portal using Google's OAuth process. You will be prompted to sign in, at which point you may select your Google account. If your account has been approved, you may view the patient dashboard.

On the dashboard, you will be able to view the number of patients and critical cases, along with a full list of all recorded cases on the left-hand side. To view more information on the patient, click on their name. From here, you may view vitals, patient identifying information, symptoms, medical history, and other general ePCR form data that you may already be accustomed to.

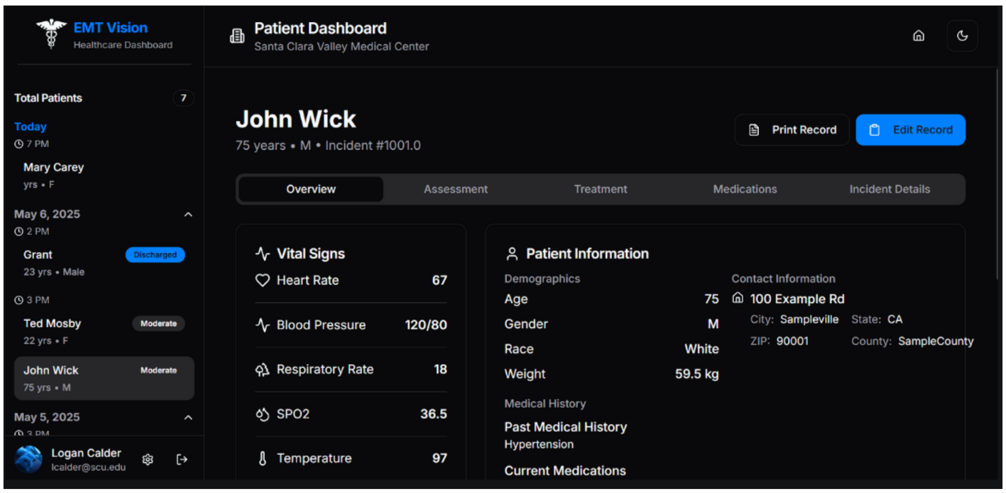


Figure 12.3: The main dashboard of the Patient Portal

12.3 Next Steps

Now that we have developed a functional prototype of the headset, there are still steps to be taken before we can get this technology into use. For one, the greatest barrier we will face will be in approval from the health department. This often slows progress, such as their very late adoption of digital mapping, where regulatory approval meant that they were stuck using paper maps when the technology was available. We would also have to consider how to get the necessary hardware, AR glasses, purchased by the fire departments, so that they can run our application. In addition, we have considered additional features that could be implemented in the future.

12.3.1 Health Department Approval

Because the application would collect protected health information, HIPAA approval is required to prove that it is handled with care. This is an extensive regulatory barrier to overcome. However, a manageable solution could be to use AI models that have been pre-approved. In this instance, we would be able to justify that so long as our application is not sharing this data, and only acting as a front end to these applications, then getting re-approved is unnecessary. This would save a great deal of effort and we would only have to change our application to use variations of the models we have already implemented, such as using BastionGPT, which is a HIPAA approved version of the natural language processing model that we used, ChatGPT. We could also use Amazon Textract, a HIPAA approved vision model that would work in place of Azure Document Intelligence. We did not use these pre-approved models for the sake of affordability and convenience in developing our prototype, but this transition would take negligible effort compared to producing a new model and trying to get it accepted. The hardware, the Hololens 2 has already been approved by the FDA and the FCC as being safe for consumers to use without interrupting protected communication bandwidths, making it trustworthy in the eyes of the department.

12.3.2 Distributable Hardware

The Hololens 2 is no longer in production nor is it distributed by Microsoft. This means that in order to purchase them, fire departments would have to source in bulk from licensed distributors or source used headsets online. This makes it more affordable than we initially projected, but it is not a realistic hardware option for the long term. As a result, we would have to consider other options for an AR headset that runs the application. An alternative could be the XREAL Air glasses, an Android headset that can be purchased for $400. This would be easy to port to our project because Android XR apps are also built in Unity and would simply need different compilation settings. For maximizing affordability, we could also consider developing our own hardware, as Unity can be compiled for OpenXR which is an open source XR operating system that can be used for ARM SoC embedded systems. There would be limitations to this route. For one, developing a display system comparable to those done by competitors in industry is incredibly difficult. The Hololens 2 uses highly complex MEMS systems with state of the art waveguides which enable it to achieve an ultra high resolution at high lumens. This is what makes the display so visible, even with daylight shining through the glasses. When developing our own, we might aim to use a microOLED waveguide display system, but this would be limited to fewer lumens and a lower resolution. On top of this, we would then have to go through the process of getting FCC and FDA approval, making it a less realistic option.

12.4 Future Features

The functional prototype that we have created has all of the key features that we set out to implement. Besides these key features, there are others that could be added in the future to continue improving the project. This section outlines these potential features and improvements. These features were not included in the prototype due to various constraints, the most prominent among them being time, but could greatly benefit usability and overall impact. In future iterations of this project, we would like to include some of these features if possible.

12.4.1 Elevator Rescue

During our ride along visits with Santa Clara EMTs, they informed us that elevator rescues present a unique challenge for them. During these calls, reaching individuals who are trapped inside the elevator is a lengthy and difficult process. Since they do not have training for operating elevators, they often have to wait for a specialist to come and fix the elevator before they can reach the people inside. This can greatly increase how long the call takes, and can be an even bigger problem if someone inside the elevator is having a medical emergency.

A potential feature that was suggested was a scanner of sorts that could identify the type of elevator and display the relevant information about it. This would then allow those on the call to reach anyone trapped inside the elevator faster, reducing the time of the call and allowing them to treat those inside if they are having a medical emergency.

12.4.2 Apartment Mapping

To quickly identify apartments within large complexes, stations have maps of the buildings in their local area, and even their own keys to provide expedited access. An ideal feature would be to have the necessary map displayed when they arrive on scene, so that they can save time searching for the right map, reducing preparation work and overall stress. We chose not to develop this feature as it would likely require an additional layer of regulatory approval given that the project would be sourcing location data. However, by collaborating with the department, this would easily be feasible and could even be integrated with their current mapping systems.

12.4.3 Equipment Checklist

At the start of each day, it is the responsibility of the firefighters to check and test all of their equipment. A checklist feature to record their actions could increase their confidence in the equipment they use to ensure that the firefighters have taken the necessary steps to prepare for a call. In the mornings, they are always tired after being woken up for calls throughout the night, so any technology that can help them remember everything would reduce the risk that they might make a tired mistake that would be detrimental on a call. As well, paramedics on an Ambulance are required to complete a form called ”check offs” daily to review all equipment and refill all used disposables from previous calls. Paramedics informed us that if this list were viewable on the HUD, it could notably streamline this process, especially for newer paramedics or EMTs who are still becoming familiar with county procedures, storage expectations, and supplies. A further improvement within this could be expanded information about all the equipment, how to use medication and expected dosages, and visual guides for where exactly to locate or store instruments.

12.4.4 Diagnosis

Computer vision models are already used to help radiologists and other hospital specialists identify diagnoses that could be difficult to spot by eye. When combined with transcription interpretation of the scene, a camera diagnosis model might help EMTs recognize health issues with visual reasoning. On top of this, computer vision diagnosis could also be paired with the AR format to provide first responders with visual guides of patient anatomy and emergency medical operations.

12.4.5 Hazard Detection

First responders regularly work in hazardous environments. An example of this is that traffic accidents are the second leading cause of firefighter deaths. The combination of AR with computer vision could help warn first responders of potentially life threatening hazards that they might be overlooking in a high stress situation.

12.4.6 Missing Field Errors

The current databasing technology that the Santa Clara Fire Department use is ImageTrend. The primary reason this application was preferred by the department was that it does not allow captains to leave a report unresolved. As a result, all information about a scenario is recorded, which ensures that should the department have to go to court, then information from the case never missing.

12.4.7 Government ID Scanner

Currently, some ePCRs, such as the one used by Santa Clara County, feature the capability to scan the bar code on government IDs and automatically parse that information, such as name, age, height, and weight. Our team inquired with a company to implement similar functionality into our application, but it ultimately did not fit into our time frame. Nonetheless, allowing paramedics to merely look at an ID and automatically retain key identifying characteristics would be a beneficial quality-of-life feature.

12.4.8 Hospital Chart

Another element we plan to include in perspective iterations of our project would be a chart of local hospitals. Currently, the ambulance contains a paper chart of all local hospitals, their specialties, and their phone numbers. The patient's preferred hospital is generally chosen, although for code three (time-sensitive emergencies) they will choose the closest that specializes in that condition. The paramedic is expected to have these classifications memorized, but, as with multiple other aspects of the application, it would be beneficial to have access to quickly cross reference, perhaps even by automatically suggesting the optimal hospitals based on distance, patient symptoms, and severity of situation.

12.4.9 Medication Guidance Interface

This feature would aim to provide first responders with an easily accessible resource for county-approved medication protocols. While this information is already available on county policy pages within their iPads, such extensive documents are not as conducive to optimized, reliable retrieval for specific scenarios. This would include detailed information for each medication, including indications, contraindications, and administration routes to enable confirmation of correct procedures, especially considering mandated policies update at least weekly. As well, the interface would clearly display standard dosage recommendations, including initial and secondary dose quantity and time frame based on patient factors such as age, weight, or clinical condition. Finally, the paramedic could be presented with medication usage standards based on specific medical scenarios, helping reduce cognitive load for uncommon scenarios.

Chapter 13

Conclusion

Altogether, our project highlights the potential of combining extended reality and artificial intelligence by applying it to a real-world scenario. The adoption of technology among emergency response teams is notably slow, resulting in unnecessary stress for first responders and patient care not reaching its full potential due to underutilized resources. By researching, designing, developing, and testing this prototype, our project proves that a hands-free augmented reality device could serve a currently under-addressed role in available first responder equipment, and we hope this prototype inspires further development in this technology to improve our community.

During the last year dedicated to this project, our team has dramatically grown in our understanding of emergency medical technology, AI and AR development, the role and operation of first responders, and firsthand experience tailoring a product to stakeholders through testing, feedback, and timely adjustments. Additionally, we were forced to navigate development constraints such as interdisciplinary coordination, achieving project expectations, balancing technological feasibility, and adjusting to optimize our team's personnel. Although our development cycle was ultimately limited, we were satisfied with how we effectively delivered and met the expectations of Santa Clara Fire Department.

Two critical hurdles, the extensive costs to achieve HIPAA approval and the visual limitations of current AR glasses hardware, are obstacles for this project's viability in the near future. In time, however, we wholeheartedly believe that the hardware will become more practical, affordable, and subtle in design, enabling EMT Vision to be a deployable, scalable solution. If our team continues this project, we believe integrating this technology into real paramedic scenarios would streamline workflows, optimize information acquisition and exchange, and reduce cognitive load. Moreover, some features initially envisioned for the project, such as automatic language translation and cloud-based data analytics, could be reintroduced in future development, further solidifying EMT Vision as a comprehensive tool of everyday first responders.

Ultimately, EMT Vision underscores a persistent struggle within emergency response, which for decades has left first responders with limited access to modern, supportive technology designed to aid them, rather than just the patient, through the pressures and responsibilities of saving lives. We hope this small yet significant step in exploring how engineers and emerging technologies will inspire future innovation in this essential field, and one day, better emergency care for those in need.

Chapter 14

Acknowledgments

We are so grateful to have had the opportunity to work directly with the Santa Clara Fire Department and see firsthand the vital work they do for our community. We were welcomed to the stations to meet with them personally, and we could never have completed this project without their support. It is easy to take for granted the heroes who show up when we need them most — we appreciate that we had the chance to learn from them and are inspired to continue building solutions that can help.

Thank you to the Santa Clara University School of Engineering, who allocated the time and resources for us to be able to collaborate on this project. Most notably, this includes access to work on the Hololens 2, an expensive and new piece of technology that we could not have accessed without them. Furthermore, thank you to our professors who have motivated and inspired us throughout these four years, especially Dr. Ramamoorthy, who advised and mentored us across the duration of this project.

Chapter 15

References

Bibliography

[1] Agata Gawron Aneta Grochowska and Iwona Bodys-Cupak. Stress-inducing factors vs. the risk of occupational burnout in the work of nurses and paramedics, 2022. Accessed: 2024-08-19.

[2] Hanae Armitage. Stanford medicine uses augmented reality for real-time data visualization during surgery, 2024. Accessed: 2025-02-21.

[3] Philip Braun. Virtual reality for immersive multi-user firefighter training scenarios, 2024. Accessed: 2025-02-21.

[4] Sasha Brodsky. First responders ar mr solutions for frontline emergency professionals, 2024. Accessed: 2025-02-21.

[5] US Census Bureau. 2020 census results, 2024. Accessed: 2025-04-23.

[6] Julie Cooper. Virtual reality training helps first responders, 2024. Accessed: 2025-02-21.

[7] Ben Coxworth. Japanese emts to start using ar glasses while transporting patients, 2024. Accessed: 2025-02-21.

[8] Dr. Mathias Unberath Dr. Nick Dalesio, Dr. Laeben Lester. Vital monitor and id detection through machine vision for improving ems communication efficiency, 2024. Accessed: 2025-02-21.

[9] Esteva et al. Dermatologist-level classification of skin cancer with deep neural networks, 2017. Accessed: 2025-04-23.

[10] Finget. Augmented reality in the healthcare industry: practical use cases, 2024. Accessed: 2025-02-21.

[11] Glassdoor. How much does a firefighter make in santa clara, ca?, 2024. Accessed: 2025-01-26.

[12] Andrzej Grabowski. Practical skills training in enclosure fires: An experimental study with cadets and firefighters using cave and hmd-based virtual training simulators, 2024. Accessed: 2025-02-21.

[13] Andreas Hoell, Eirini Kourmpeli, and Harald Dressing. Work-related posttraumatic stress disorder in paramedics in comparison to data from the general population of working age: A systematic review and meta-analysis. *Frontiers in Public Health*, 11, 2023.

[14] Evan M. McKay Jeremy J. Taliercio Scott D. White Blair J. Woodbury Mark A. Sandefur Jeffrey D. Ho, Donald M. Dawes and James R. Miner. Effect of body-worn cameras on ems documentation accuracy: A pilot study. *Prehospital Emergency Care*, 21(2):263–271, 2017. PMID: 27636021.

[15] Kim Carol Maligalig. Machine vision system of emergency vehicle detection system using deep transfer learning, 2024. Accessed: 2025-02-21.

[16] Ludmila Marcinowicz, Jerzy Konstantynowicz, and Cezary Godlewski. Patients' perceptions of gp non-verbal communication: a qualitative study. *British Journal of General Practice*, 60(571):83–87, 2010.

[17] Mical Adamski Marta Dunajko. Augmented reality in the healthcare industry: practical use cases, 2024. Accessed: 2025-02-21.

[18] Microsoft Docs. Unity development overview - mixed reality, 2024. Accessed: 2025-04-22.

[19] Harbil Arregui Others. An augmented reality framework for first responders: the respond-a project approach, 2024. Accessed: 2025-02-21.

[20] Kelly Peng. The future is here: Ems training with augmented reality, 2024. Accessed: 2025-02-21.

[21] N Raveendran. Future of smart firefighting, 2024. Accessed: 2025-02-21.

[22] Matthew Reardon, Raquel Abrahams, Liz Thyer, and Paul Simpson. Review article: Prevalence of burnout in paramedics: A systematic review of prevalence studies. *Emergency Medicine Australasia*, 32(2):182–189, 2020.

[23] M Short and S Goldstein. *EMS Documentation*. StatPearls [Internet]. StatPearls Publishing, Treasure Island, FL, updated 2022 sep 26 edition, September 2022. Available from: https://www.ncbi.nlm.nih.gov/books/NBK448107/.

[24] Data USA. Santa clara, ca: Census, 2023. Accessed: 2025-04-23.

[25] Voxel51. Evaluating ml models for computer vision, 2024. Accessed: 2025-02-21.

Appendix A

Code Files

A.1 auth_callback.ts

```
import { createRouteHandlerClient } from "@supabase/auth-helpers-nextjs"
import { cookies } from "next/headers"
import { NextResponse } from "next/server"
import type { NextRequest } from "next/server"

export async function GET(request: NextRequest) {
  const requestUrl = new URL(request.url)
  const code = requestUrl.searchParams.get("code")

  // Check if environment variables are set
  const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL
  const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY

  if (!supabaseUrl || !supabaseAnonKey) {
    // Redirect to an error page or back to login with an error parameter
    return NextResponse.redirect(new URL("/login?error=missing_env_vars", request.url))
  }

  if (code) {
    try {
      const cookieStore = cookies()
      const supabase = createRouteHandlerClient({ cookies: () => cookieStore })

      await supabase.auth.exchangeCodeForSession(code)
    } catch (error) {
      console.error("Error exchanging code for session:", error)
      return NextResponse.redirect(new URL("/login?error=auth_error", request.url))
    }
  }

  // URL to redirect to after sign in process completes
  return NextResponse.redirect(new URL("/dashboard", request.url))
}
```

A.2 callback.ts

```
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs'
import { cookies } from 'next/headers'
import { NextResponse } from 'next/server'

export async function GET(request: Request) {
  const requestUrl = new URL(request.url)
  const code = requestUrl.searchParams.get('code')

  if (code) {
```

```
    const supabase = createRouteHandlerClient({ cookies })
    await supabase.auth.exchangeCodeForSession(code)
  }

  // URL to redirect to after sign in process completes
  return NextResponse.redirect(`${requestUrl.origin}/dashboard`)
}
```

A.3 dashboard.tsx

```
"use client"

import { useEffect, useState } from "react"
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card"
import { AlertCircle, Clock, User,  Baby, User2, UserCog } from "lucide-react"
import { ScrollArea } from "@/components/ui/scroll-area"
import { Badge } from "@/components/ui/badge"
import { formatMedicalCondition } from "@/utils/format"

interface DashboardStats {
  totalPatients: number
  criticalCases: number
  recentPatients: number
  recentPatientsList: any[]
  demographicStats?: {
    ageRanges: {
      pediatric: number // 0-17
      youngAdult: number // 18-39
      middleAge: number // 40-64
      senior: number // 65+
    }
    gender: {
      male: number
      female: number
      other: number
    }
    averageAge: number
  }
}

// Helper function to get acuity badge variant - matching sidebar
function getAcuityBadgeVariant(acuity: string | undefined): "default" | "secondary" | "
    destructive" | "outline" {
  if (!acuity) return "default"
  const acuityLower = acuity.toLowerCase()
  if (acuityLower.includes("critical") || acuityLower.includes("severe")) {
    return "destructive"
  } else if (acuityLower.includes("moderate")) {
    return "secondary"
  } else if (acuityLower.includes("minor") || acuityLower.includes("low")) {
    return "outline"
  }
  return "default"
}

// Helper function to categorize age
function categorizeAge(age: string | undefined): string {
  if (!age) return "other"
  const ageNum = parseInt(age)
  if (isNaN(ageNum)) return "other"

  if (ageNum < 18) return "pediatric"
  if (ageNum < 40) return "youngAdult"
  if (ageNum < 65) return "middleAge"
  return "senior"
}

```

```
export default function Dashboard() {
  const [stats, setStats] = useState<DashboardStats | null>(null)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    async function fetchStats() {
      try {
        const response = await fetch('/api/dashboard/stats')
        const data = await response.json()
        console.log('Dashboard received data:', data)

        // Calculate demographic stats
        const demographicStats = {
          ageRanges: {
            pediatric: 0,
            youngAdult: 0,
            middleAge: 0,
            senior: 0
          },
          gender: {
            male: 0,
            female: 0,
            other: 0
          },
          averageAge: 0
        }

        let totalAge = 0
        let validAgeCount = 0

        data.recentPatientsList?.forEach((patient: any) => {
          // Categorize age
          const ageCategory = categorizeAge(patient.Age)
          demographicStats.ageRanges[ageCategory as keyof typeof demographicStats.ageRanges]++

          // Calculate average age
          const ageNum = parseInt(patient.Age)
          if (!isNaN(ageNum)) {
            totalAge += ageNum
            validAgeCount++
          }

          // Count gender
          const gender = patient.Gender?.toLowerCase() || "other"
          if (gender === "male" || gender.includes('m')) {
            demographicStats.gender.male++
          } else if (gender === "female" || gender.includes('f')) {
            demographicStats.gender.female++
          } else {
            demographicStats.gender.other++
          }
        })

        demographicStats.averageAge = validAgeCount > 0 ? Math.round(totalAge / validAgeCount) : 0

        setStats({ ...data, demographicStats })
      } catch (error) {
        console.error('Error fetching dashboard stats:', error)
      } finally {
        setLoading(false)
      }
    }

    fetchStats()
    // Poll for updates every 5 seconds to match sidebar
    const interval = setInterval(fetchStats, 5000)
    return () => clearInterval(interval)
```

```
  }, [])

  if (loading) {
    return <div>Loading...</div>
  }

  // Debug render
  console.log('Dashboard rendering with stats:', stats)

  const totalGenderCount = Object.values(stats?.demographicStats?.gender || {}).reduce((a, b) => a + b, 0)

  return (
    <div className="p-5 h-full">
      <h1 className="text-3xl font-bold tracking-tight mb-6">Dashboard</h1>

      <div className="grid gap-6 h-[calc(100vh-8rem)]">
        <div className="grid gap-4 md:grid-cols-2">
          <Card>
            <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
              <CardTitle className="text-sm font-medium tracking-tight">Critical Cases</CardTitle>
              <AlertCircle className="h-4 w-4 text-destructive" />
            </CardHeader>
            <CardContent>
              <div className="text-2xl font-bold tracking-tight text-red-700">{stats?.criticalCases || 0}</div>
            </CardContent>
          </Card>

          <Card>
            <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
              <CardTitle className="text-sm font-medium tracking-tight">Patients Today</CardTitle>
              <Clock className="h-4 w-4 text-muted-foreground" />
            </CardHeader>
            <CardContent>
              <div className="text-2xl font-bold tracking-tight">{stats?.recentPatients || 0}</div>
            </CardContent>
          </Card>
        </div>

        <div className="grid gap-4 md:grid-cols-2 h-[calc(100%-8rem)]">
          <Card className="col-span-1 flex flex-col">
            <CardHeader>
              <CardTitle className="text-lg font-bold tracking-tight">Recent Patients</CardTitle>
              <CardDescription>Patients admitted in the last 24 hours</CardDescription>
            </CardHeader>
            <CardContent className="flex-1 p-0">
              <ScrollArea className="h-full px-6">
                {stats?.recentPatientsList && stats.recentPatientsList.length > 0 ? (
                  <div className="space-y-2">
                    {stats.recentPatientsList.map((patient) => (
                      <div key={patient.PatientID} className="flex items-center justify-between p-2 hover:bg-muted/50 rounded-lg">
                        <div className="flex-1 min-w-0">
                          <p className="font-medium truncate">{patient.PatientName}</p>
                          <p className="text-sm text-muted-foreground truncate">
                            {patient.Age} years     {patient.Gender}
                          </p>
                        </div>
                        <Badge variant={getAcuityBadgeVariant(patient.Severity)}>
                          {formatMedicalCondition(patient.Severity)}
                        </Badge>
                      </div>
                    ))}
                  </div>
```

```
              ) : (
                <div className="p-4 text-center text-muted-foreground">No recent patients</div>
              )}
            </ScrollArea>
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle className="text-lg font-bold tracking-tight">Today's Demographics</CardTitle>
            <CardDescription>Patient age and gender distribution</CardDescription>
          </CardHeader>
          <CardContent>
            <div className="space-y-6">
              <div className="grid grid-cols-2 gap-4">
                <div className="flex items-center space-x-2">
                  <Baby className="h-4 w-4 text-pink-500" />
                  <div>
                    <p className="text-sm font-medium">Pediatric (0-17)</p>
                    <p className="text-2xl font-bold">{stats?.demographicStats?.ageRanges.pediatric || 0}</p>
                  </div>
                </div>
                <div className="flex items-center space-x-2">
                  <User2 className="h-4 w-4 text-blue-500" />
                  <div>
                    <p className="text-sm font-medium">Adult (18-64)</p>
                    <p className="text-2xl font-bold">
                      {(stats?.demographicStats?.ageRanges.youngAdult || 0) +
                       (stats?.demographicStats?.ageRanges.middleAge || 0)}
                    </p>
                  </div>
                </div>
                <div className="flex items-center space-x-2">
                  <UserCog className="h-4 w-4 text-purple-500" />
                  <div>
                    <p className="text-sm font-medium">Senior (65+)</p>
                    <p className="text-2xl font-bold">{stats?.demographicStats?.ageRanges.senior || 0}</p>
                  </div>
                </div>
                <div className="flex items-center space-x-2">
                  <User className="h-4 w-4 text-green-500" />
                  <div>
                    <p className="text-sm font-medium">Avg. Age</p>
                    <p className="text-2xl font-bold">{stats?.demographicStats?.averageAge || 0}</p>
                  </div>
                </div>
              </div>

              <div className="space-y-2">
                <div className="flex justify-between text-sm">
                  <span className="font-medium">Gender Distribution</span>
                </div>
                <div className="grid grid-cols-2 gap-4">
                  <div className="space-y-1">
                    <div className="flex justify-between text-sm">
                      <span className="text-muted-foreground">Male</span>
                      <span>{stats?.demographicStats?.gender.male || 0}</span>
                    </div>
                    <div className="h-2 bg-muted rounded-full overflow-hidden">
                      <div
                        className="h-full bg-blue-500 transition-all duration-500 ease-out"
                        style={{
                          width: `${((stats?.demographicStats?.gender.male || 0) / totalGenderCount) * 100}%`
```

```
                          }}
                        />
                      </div>
                    </div>
                    <div className="space-y-1">
                      <div className="flex justify-between text-sm">
                        <span className="text-muted-foreground">Female</span>
                        <span>{stats?.demographicStats?.gender.female || 0}</span>
                      </div>
                      <div className="h-2 bg-muted rounded-full overflow-hidden">
                        <div
                          className="h-full bg-pink-500 transition-all duration-500 ease-out"
                          style={{
                            width: `${((stats?.demographicStats?.gender.female || 0) / totalGenderCount) * 100}%`
                          }}
                        />
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </CardContent>
          </Card>
        </div>
      </div>
    </div>
  )
}
```

A.4 layout.tsx

```
import type React from "react"
import "@/app/landing.css"
import { ThemeProvider } from "@/components/theme-provider"
import { Inter } from "next/font/google"

// Load Inter font - Apple-like typography
const inter = Inter({
  subsets: ["latin"],
  variable: "--font-inter",
  display: "swap",
})

export const metadata = {
  title: "EMT Vision - Advanced Emergency Medical Technology",
  description: "Empowering first responders with cutting-edge technology for emergency medical services.",
    generator: 'v0.dev'
}

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode
}>) {
  return (
    <html lang="en" suppressHydrationWarning className={`${inter.variable}`}>
      <body className={inter.className}>
        <ThemeProvider attribute="class" defaultTheme="system" enableSystem disableTransitionOnChange>
          {children}
        </ThemeProvider>
      </body>
    </html>
  )
}
```

```


import './globals.css'
```

A.5 login.tsx

```
"use client"
import Link from "next/link"
import { Button } from "@/components/ui/button"
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card"
import { createClientComponentClient } from '@supabase/auth-helpers-nextjs'
import { useRouter } from 'next/navigation'
import { AuthButton } from "@/components/auth-button";
import { FaGoogle } from "react-icons/fa";
import { Footer } from "@/components/footer"

export default function LoginPage() {
  const router = useRouter()
  const supabase = createClientComponentClient()

  const handleGoogleLogin = async () => {
    try {
      const { data, error } = await supabase.auth.signInWithOAuth({
        provider: 'google',
        options: {
          redirectTo: `${window.location.origin}/auth/callback`
        }
      })

      if (error) throw error
    } catch (error) {
      console.error('Error logging in with Google:', error)
    }
  }

  return (
    <div className="flex min-h-screen flex-col">
      <div className="flex-1 flex items-center justify-center bg-background">
        <div className="absolute top-4 left-4">
          <Link href="/" className="text-sm text-muted-foreground hover:text-primary transition-
    colors">
                Return to Home
          </Link>
        </div>
        <Card className="w-[350px]">
          <CardHeader className="space-y-1">
            <CardTitle className="text-2xl font-bold text-center">Welcome</CardTitle>
            <CardDescription className="text-center">
              Sign in to access the EMT Vision Dashboard
            </CardDescription>
          </CardHeader>
          <CardContent>
            <div className="grid gap-4">
              <AuthButton
                provider="google"
                label="Continue with Google"
                icon={<FaGoogle className="mr-2 h-4 w-4" />}
                onClick={handleGoogleLogin}
              />
            </div>
          </CardContent>
        </Card>
      </div>
      <Footer />
    </div>
  )
}
```

A.6 patient.ts

```
import { supabase } from "@/utils/supabase/server"
import { NextResponse } from "next/server"
export const dynamic = 'force-dynamic';

export async function GET(request: Request) {
  const { searchParams } = new URL(request.url)
  const patientId = searchParams.get('id')

  if (!patientId) {
    return NextResponse.json({ error: 'Patient ID is required' }, { status: 400 })
  }

  const { data, error } = await supabase
    .from("PatientData")
    .select("*")
    .eq('PatientID', patientId)
    .order('Time', { ascending: false })

  if (error) {
    return NextResponse.json({ error: error.message }, { status: 500 })
  }

  return NextResponse.json(data)
}
```

A.7 patient.tsx

```
"use client"

import { useState, useEffect } from "react"
import { useParams } from "next/navigation"
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card"
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs"
import { Badge } from "@/components/ui/badge"
import { Button } from "@/components/ui/button"
import { Separator } from "@/components/ui/separator"
import {
  Activity,
  AlertCircle,
  Clipboard,
  FileText,
  Heart,
  Home,
  Info,
  Pill,
  User,
  Stethoscope,
  Thermometer,
  Droplets,
  TreesIcon as Lungs,
  Brain,
  Ambulance,
  Pencil,
  AlertTriangle,
  CheckCircle2,
} from "lucide-react"
import { generatePatientPDF } from "@/utils/pdf-generator"
import { PatientEditModal } from "@/components/patient-edit-modal"
import { MedicationHistory } from "@/components/medication-history"
import { Patient } from "@/types/patient"
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/
    select"
import { formatValue, formatList, formatName, formatAddress, formatMedicalCondition,
    formatMedicalConditions } from "@/utils/format"
```

```
// Helper function to get acuity badge color
function getAcuityBadgeVariant(acuity: string): "default" | "secondary" | "destructive" | "outline" {
  const acuityLower = acuity.toLowerCase()
  if (acuityLower.includes("critical") || acuityLower.includes("severe")) {
    return "destructive"
  } else if (acuityLower.includes("moderate")) {
    return "secondary"
  } else if (acuityLower.includes("minor") || acuityLower.includes("low")) {
    return "outline"
  }
  return "default"
}

// Add this helper function at the top level
function MissingField({ value, children }: { value: any, children: React.ReactNode }) {
  return (
    <span className={!value ? 'text-destructive' : ''}>
      {children}
    </span>
  )
}

export default function PatientPage() {
  const params = useParams()
  const [patient, setPatient] = useState<Patient | null>(null)
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState<string | null>(null)
  const [editModalOpen, setEditModalOpen] = useState(false)

  // Function to fetch patient data
  const fetchPatient = async () => {
    try {
      const response = await fetch(`/api/patient?id=${params.id}`)
      const data = await response.json()

      if (!response.ok) {
        throw new Error(data.error || "Failed to fetch patients")
      }

      setPatient(data.length > 0 ? data[0] : null)
    } catch (err) {
      console.error("Caught error:", err)
      setError(err instanceof Error ? err.message : "An unexpected error occurred")
    } finally {
      setLoading(false)
    }
  }

  useEffect(() => {
    // Initial fetch
    fetchPatient()

    // Set up polling interval (every 30 seconds)
    const intervalId = setInterval(fetchPatient, 5000)

    // Cleanup interval on component unmount
    return () => clearInterval(intervalId)
  }, [params.id])

  const handlePatientUpdated = (updatedPatient: Patient) => {
    setPatient(updatedPatient)
  }

  const getSeverityColor = (severity: string) => {
    switch (severity.toLowerCase()) {
      case "critical":
```

```
      return "bg-red-500/10 text-red-500 hover:bg-red-500/20"
    case "severe":
      return "bg-orange-500/10 text-orange-500 hover:bg-orange-500/20"
    case "moderate":
      return "bg-yellow-500/10 text-yellow-500 hover:bg-yellow-500/20"
    case "mild":
      return "bg-green-500/10 text-green-500 hover:bg-green-500/20"
    case "discharged":
      return "bg-blue-500/10 text-blue-500 hover:bg-blue-500/20"
    default:
      return "bg-gray-500/10 text-gray-500 hover:bg-gray-500/20"
  }
}

const getSeverityIcon = (severity: string) => {
  switch (severity.toLowerCase()) {
    case "critical":
      return <AlertTriangle className="h-4 w-4" />
    case "severe":
      return <Activity className="h-4 w-4" />
    case "moderate":
      return <Heart className="h-4 w-4" />
    case "mild":
      return <CheckCircle2 className="h-4 w-4" />
    case "discharged":
      return <CheckCircle2 className="h-4 w-4" />
    default:
      return <Activity className="h-4 w-4" />
  }
}

if (loading) {
  return (
    <div className="flex items-center justify-center h-[70vh]">
      <div className="flex flex-col items-center">
        <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-primary mb-4"></
div>
        <p className="text-muted-foreground">Loading patient information...</p>
      </div>
    </div>
  )
}

if (error) {
  return (
    <Card className="border-destructive bg-destructive/10 mx-auto max-w-3xl mt-8">
      <CardHeader>
        <CardTitle className="text-destructive flex items-center">
          <AlertCircle className="mr-2 h-5 w-5" />
          Error Loading Patient Data
        </CardTitle>
      </CardHeader>
      <CardContent>
        <p className="text-destructive">{error}</p>
        <Button variant="outline" className="mt-4" onClick={() => window.location.href = '/
dashboard'}>
          Go Back
        </Button>
      </CardContent>
    </Card>
  )
}

if (!patient) {
  return (
    <Card className="border-muted bg-muted/10 mx-auto max-w-3xl mt-8">
      <CardHeader>
        <CardTitle className="flex items-center">
```

```
            <Info className="mr-2 h-5 w-5" />
            Patient Not Found
          </CardTitle>
        </CardHeader>
        <CardContent>
          <p>The requested patient record could not be found.</p>
          <Button variant="outline" className="mt-4" onClick={() => window.history.back()}>
            Go Back
          </Button>
        </CardContent>
      </Card>
    )
  }

  return (
    <div className="container mx-auto py-6 px-4 max-w-7xl">
      <div className="flex flex-col md:flex-row justify-between items-start md:items-center mb-6 gap-4">
        <div>
          <div className="flex items-center gap-2">
            <h1 className="text-3xl font-bold">{formatName(patient.PatientName)}</h1>
          </div>
          <p className="text-muted-foreground mt-1">
            {formatValue(patient.Age)} years     {formatValue(patient.Gender)}     Incident #{formatValue(patient.IncidentNumber)}
          </p>
        </div>
        <div className="flex gap-2">
          <Button variant="outline" onClick={() => patient && generatePatientPDF(patient)}>
            <FileText className="mr-2 h-4 w-4" />
            Print Record
          </Button>
          <Button onClick={() => setEditModalOpen(true)}>
            <Clipboard className="mr-2 h-4 w-4" />
            Edit Record
          </Button>
        </div>
      </div>

      {/* Edit Modal */}
      {patient && (
        <PatientEditModal
          patient={patient}
          open={editModalOpen}
          onOpenChange={setEditModalOpen}
          onPatientUpdated={handlePatientUpdated}
        />
      )}

      <Tabs defaultValue="overview" className="w-full">
        <TabsList className="grid grid-cols-5 mb-6">
          <TabsTrigger value="overview">Overview</TabsTrigger>
          <TabsTrigger value="assessment">Assessment</TabsTrigger>
          <TabsTrigger value="treatment">Treatment</TabsTrigger>
          <TabsTrigger value="medications">Medications</TabsTrigger>
          <TabsTrigger value="incident">Incident Details</TabsTrigger>
        </TabsList>

        <TabsContent value="overview" className="space-y-6">
          <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
            {/* Vital Signs Card */}
            <Card className="md:col-span-1">
              <CardHeader className="pb-2">
                <CardTitle className="text-lg flex items-center">
                  <Activity className="mr-2 h-5 w-5 text-primary" />
                  Vital Signs
                </CardTitle>
              </CardHeader>
```

```
              <CardContent>
                <div className="space-y-4">
                  <div className="flex justify-between items-center">
                    <div className="flex items-center">
                      <Heart className="h-5 w-5 mr-2 text-red-500" />
                      <span>Heart Rate</span>
                    </div>
                    <span className="font-semibold">{formatValue(patient.HeartRate)}</span>
                  </div>
                  <Separator />

                  <div className="flex justify-between items-center">
                    <div className="flex items-center">
                      <Activity className="h-5 w-5 mr-2 text-blue-500" />
                      <span>Blood Pressure</span>
                    </div>
                    <span className="font-semibold">{formatValue(patient.BloodPressure)}</span>
                  </div>
                  <Separator />

                  <div className="flex justify-between items-center">
                    <div className="flex items-center">
                      <Lungs className="h-5 w-5 mr-2 text-green-500" />
                      <span>Respiratory Rate</span>
                    </div>
                    <span className="font-semibold">{formatValue(patient.RespiratoryRate)}</span>
                  </div>
                  <Separator />

                  <div className="flex justify-between items-center">
                    <div className="flex items-center">
                      <Droplets className="h-5 w-5 mr-2 text-purple-500" />
                      <span>SPO2</span>
                    </div>
                    <span className="font-semibold">{formatValue(patient.SPO2)}</span>
                  </div>
                  <Separator />

                  <div className="flex justify-between items-center">
                    <div className="flex items-center">
                      <Thermometer className="h-5 w-5 mr-2 text-orange-500" />
                      <span>Temperature</span>
                    </div>
                    <span className="font-semibold">{formatValue(patient.Temperature)}</span>
                  </div>
                  <Separator />

                  <div className="flex justify-between items-center">
                    <div className="flex items-center">
                      <Droplets className="h-5 w-5 mr-2 text-yellow-500" />
                      <span>Glucose</span>
                    </div>
                    <span className="font-semibold">{formatValue(patient.Glucose)}</span>
                  </div>
                </div>
              </CardContent>
            </Card>

            {/* Patient Information Card */}
            <Card className="md:col-span-2">
              <CardHeader className="pb-2">
                <CardTitle className="text-lg flex items-center">
                  <User className="mr-2 h-5 w-5 text-primary" />
                  Patient Information
                </CardTitle>
              </CardHeader>
              <CardContent>
                <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
```

```
                <div>
                  <h3 className="text-sm font-medium text-muted-foreground mb-1">Demographics</h3>
                  <div className="space-y-2">
                    <div className="flex justify-between">
                      <span>Age</span>
                      <span className={`font-medium ${!patient.Age ? 'text-destructive' : ''}`}>{formatValue(patient.Age)}</span>
                    </div>
                    <div className="flex justify-between">
                      <span>Gender</span>
                      <span className={`font-medium ${!patient.Gender ? 'text-destructive' : ''}`}>{formatValue(patient.Gender)}</span>
                    </div>
                    <div className="flex justify-between">
                      <span>Race</span>
                      <span className={`font-medium ${!patient.Race ? 'text-destructive' : ''}`}>{formatValue(patient.Race)}</span>
                    </div>
                    <div className="flex justify-between">
                      <span>Weight</span>
                      <span className={`font-medium ${!patient.WeightKg ? 'text-destructive' : ''}`}>{patient.WeightKg ? `${formatValue(patient.WeightKg)} kg` : "N/A"}</span>
                    </div>
                  </div>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground mb-1">Contact Information</h3>
                  <div className="space-y-2">
                    <div className="flex items-start gap-2">
                      <Home className="h-4 w-4 mt-0.5 flex-shrink-0 text-muted-foreground" />
                      <div className="space-y-1">
                        <p className={`font-medium ${!patient.HomeAddress ? 'text-destructive' : ''}`}>{formatAddress(patient.HomeAddress)}</p>
                        <div className="grid grid-cols-2 gap-2 text-sm">
                          <div>
                            <span className="text-muted-foreground">City:</span>
                            <span className={`ml-2 ${!patient.City ? 'text-destructive' : ''}`}>{formatValue(patient.City)}</span>
                          </div>
                          <div>
                            <span className="text-muted-foreground">State:</span>
                            <span className={`ml-2 ${!patient.State ? 'text-destructive' : ''}`}>{formatValue(patient.State)}</span>
                          </div>
                          <div>
                            <span className="text-muted-foreground">ZIP:</span>
                            <span className={`ml-2 ${!patient.ZIPCode ? 'text-destructive' : ''}`}>{formatValue(patient.ZIPCode)}</span>
                          </div>
                          <div>
                            <span className="text-muted-foreground">County:</span>
                            <span className={`ml-2 ${!patient.County ? 'text-destructive' : ''}`}>{formatValue(patient.County)}</span>
                          </div>
                        </div>
                        {patient.ContactInfo && (
                          <div className="mt-2">
                            <span className="text-muted-foreground">Contact:</span>
                            <span className={`ml-2 ${!patient.ContactInfo ? 'text-destructive' : ''}`}>{formatValue(patient.ContactInfo)}</span>
                          </div>
                        )}
                      </div>
                    </div>
                  </div>
```

```
              </div>

              <div className="md:col-span-2">
                <h3 className="text-sm font-medium text-muted-foreground mb-1">Medical History</h3>
                <div className="space-y-3 mt-2">
                  <div>
                    <p className="font-medium">Past Medical History</p>
                    <p className="text-sm">{formatMedicalConditions(patient.PastMedicalHistory)}</p>
                  </div>
                  <div>
                    <p className="font-medium">Current Medications</p>
                    <p className="text-sm">{formatMedicalConditions(patient.CurrentMedications)}</p>
                  </div>
                  <div>
                    <p className="font-medium">Allergies</p>
                    <p className="text-sm">{formatMedicalConditions(patient.MedicationAllergies)}</p>
                  </div>
                  <div>
                    <p className="font-medium">Advance Directives</p>
                    <p className="text-sm">{formatMedicalCondition(patient.AdvanceDirectives)}</p>
                  </div>
                </div>
              </div>
            </div>
          </CardContent>
        </Card>
      </div>

      {/* Primary Complaint & Impression */}
      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-lg font-bold tracking-tight flex items-center">
            <Stethoscope className="mr-2 h-5 w-5 text-primary" />
            Primary Complaint & Impression
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
            <div>
              <h3 className="text-sm font-medium text-muted-foreground mb-2">Primary Complaint</h3>
              <p className="text-lg font-medium">{formatMedicalCondition(patient.PrimaryComplaint)}</p>
              {patient.Duration && patient.TimeUnits && (
                <p className="text-sm text-muted-foreground mt-1">
                  Duration: {formatValue(patient.Duration)} {formatValue(patient.TimeUnits)}
                </p>
              )}
              {patient.PrimarySymptom && (
                <div className="mt-3">
                  <h4 className="text-sm font-medium">Primary Symptom</h4>
                  <p>{formatMedicalCondition(patient.PrimarySymptom)}</p>
                </div>
              )}
              {patient.OtherSymptoms && (
                <div className="mt-3">
                  <h4 className="text-sm font-medium">Other Symptoms</h4>
                  <p>{formatMedicalConditions(patient.OtherSymptoms)}</p>
                </div>
              )}
            </div>

            <div>
```

```
              <h3 className="text-sm font-medium text-muted-foreground mb-2">Primary Impression</h3>
              <p className="text-lg font-medium">{formatMedicalCondition(patient.PrimaryImpression)}</p>

              <div className="mt-4 grid grid-cols-2 gap-2">
                <div>
                  <h4 className="text-sm font-medium">Current Acuity</h4>
                  <Badge variant={getAcuityBadgeVariant(patient.Severity || "")}>
                    {formatMedicalCondition(patient.Severity)}
                  </Badge>
                </div>
                {patient.CardiacArrest && (
                  <div>
                    <h4 className="text-sm font-medium">Cardiac Arrest</h4>
                    <p>{formatMedicalCondition(patient.CardiacArrest)}</p>
                  </div>
                )}
                {patient.PossibleInjury && (
                  <div>
                    <h4 className="text-sm font-medium">Possible Injury</h4>
                    <p>{formatMedicalCondition(patient.PossibleInjury)}</p>
                  </div>
                )}
              </div>
            </div>
          </div>
        </CardContent>
      </Card>
    </TabsContent>

    <TabsContent value="assessment" className="space-y-6">
      {/* GCS Assessment */}
      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-lg flex items-center">
            <Brain className="mr-2 h-5 w-5 text-primary" />
            Glasgow Coma Scale
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
            <div className="bg-muted/30 p-4 rounded-lg text-center">
              <h3 className="text-sm font-medium text-muted-foreground">Eye</h3>
              <p className={`text-2xl font-bold mt-2 ${!patient.GCS_Eye ? 'text-destructive' : ''}`}>
                {patient.GCS_Eye || "N/A"}
              </p>
            </div>
            <div className="bg-muted/30 p-4 rounded-lg text-center">
              <h3 className="text-sm font-medium text-muted-foreground">Verbal</h3>
              <p className={`text-2xl font-bold mt-2 ${!patient.GCS_Verbal ? 'text-destructive' : ''}`}>
                {patient.GCS_Verbal || "N/A"}
              </p>
            </div>
            <div className="bg-muted/30 p-4 rounded-lg text-center">
              <h3 className="text-sm font-medium text-muted-foreground">Motor</h3>
              <p className={`text-2xl font-bold mt-2 ${!patient.GCS_Motor ? 'text-destructive' : ''}`}>
                {patient.GCS_Motor || "N/A"}
              </p>
            </div>
            <div className="bg-primary/10 p-4 rounded-lg text-center">
              <h3 className="text-sm font-medium text-primary">Total Score</h3>
              <p className={`text-2xl font-bold mt-2 ${!patient.GCS_Score ? 'text-destructive' : ''}`}>
                {patient.GCS_Score || "N/A"}
```

```
                  </p>
                  {patient.GCS_Qualifier && <p className="text-xs mt-2">{patient.GCS_Qualifier}</p>}
                </div>
              </div>
            </CardContent>
          </Card>

          {/* Physical Examination */}
          <Card>
            <CardHeader className="pb-2">
              <CardTitle className="text-lg font-bold tracking-tight flex items-center">
                <Stethoscope className="mr-2 h-5 w-5 text-primary" />
                Physical Examination
              </CardTitle>
            </CardHeader>
            <CardContent>
              <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
                <div className="space-y-4">
                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Mental Status</h3>
                    <p className={‘font-medium mt-1 ${!patient.MentalStatus ? ’text-destructive’ : ’’}‘}>
                      {formatMedicalCondition(patient.MentalStatus)}
                    </p>
                  </div>
                  <Separator />

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Chest Exam</h3>
                    <p className={‘font-medium mt-1 ${!patient.ChestExam ? ’text-destructive’ : ’’}‘}>
                      {formatMedicalCondition(patient.ChestExam)}
                    </p>
                  </div>
                  <Separator />

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Abdomen Exam</h3>
                    <p className={‘font-medium mt-1 ${!patient.AbdomenExam ? ’text-destructive’ : ’’}‘}>
                      {formatMedicalCondition(patient.AbdomenExam)}
                    </p>
                  </div>
                  <Separator />

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Lung Exam</h3>
                    <p className={‘font-medium mt-1 ${!patient.LungExam ? ’text-destructive’ : ’’}‘}>
                      {formatMedicalCondition(patient.LungExam)}
                    </p>
                  </div>
                </div>

                <div className="space-y-4">
                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Skin Assessment</h3>
                    <p className={‘font-medium mt-1 ${!patient.SkinAssessment ? ’text-destructive’ : ’’}‘}>
                      {formatMedicalCondition(patient.SkinAssessment)}
                    </p>
                  </div>
                  <Separator />

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Back/Spine Exam</h3
```

```
>
                <p className={`font-medium mt-1 ${!patient.BackSpineExam ? 'text-destructive' : ''}`}>
                  {formatMedicalCondition(patient.BackSpineExam)}
                </p>
              </div>
              <Separator />

              <div>
                <h3 className="text-sm font-medium text-muted-foreground">Extremities Exam</h3>
                <p className={`font-medium mt-1 ${!patient.ExtremitiesExam ? 'text-destructive' : ''}`}>
                  {formatMedicalCondition(patient.ExtremitiesExam)}
                </p>
              </div>
              <Separator />

              <div>
                <h3 className="text-sm font-medium text-muted-foreground">Eye Exam</h3>
                <p className={`font-medium mt-1 ${!patient.EyeExam_Bilateral && !patient.EyeExam_Left && !patient.EyeExam_Right ? 'text-destructive' : ''}`}>
                  {formatMedicalCondition(patient.EyeExam_Bilateral) ||
                    (patient.EyeExam_Left && patient.EyeExam_Right
                      ? `Left: ${formatMedicalCondition(patient.EyeExam_Left)}, Right: ${formatMedicalCondition(patient.EyeExam_Right)}`
                      : "Not assessed")}
                </p>
              </div>
            </div>
          </div>
        </CardContent>
      </Card>
    </TabsContent>

    <TabsContent value="treatment" className="space-y-6">
      {/* Procedures */}
      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-lg flex items-center">
            <Stethoscope className="mr-2 h-5 w-5 text-primary" />
            Procedures
          </CardTitle>
        </CardHeader>
        <CardContent>
          {patient.Procedure ? (
            <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
              <div className="space-y-4">
                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Procedure</h3>
                  <p className={`font-medium mt-1 ${!patient.Procedure ? 'text-destructive' : ''}`}>
                    {formatMedicalCondition(patient.Procedure)}
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Location</h3>
                  <p className={`font-medium mt-1 ${!patient.ProcLocation ? 'text-destructive' : ''}`}>
                    {formatMedicalCondition(patient.ProcLocation)}
                  </p>
                  {patient.IVLocation && (
                    <p className={`text-sm ${!patient.IVLocation ? 'text-destructive' : ''}`}>
                      IV Location: {formatMedicalCondition(patient.IVLocation)}
                    </p>
                  )}
```

```
                </div>

                {patient.Size && (
                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Size</h3>
                    <p className={`font-medium mt-1 ${!patient.Size ? 'text-destructive' : ''}`}>
                      {formatValue(patient.Size)}
                    </p>
                  </div>
                )}
              </div>

              <div className="space-y-4">
                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Time</h3>
                  <p className={`font-medium mt-1 ${!patient.ProcTime ? 'text-destructive' : ''}`}>
                    {patient.ProcTime || "Not recorded"}
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Attempts/Success</h3>
                  <p className={`font-medium mt-1 ${!patient.Attempts && !patient.Successful ? 'text-destructive' : ''}`}>
                    {patient.Attempts ? `${patient.Attempts} attempts` : "Not recorded"}
                    {patient.Successful && `     ${patient.Successful}`}
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Response</h3>
                  <p className={`font-medium mt-1 ${!patient.ProcResponse ? 'text-destructive' : ''}`}>
                    {patient.ProcResponse || "Not recorded"}
                  </p>
                </div>
              </div>
            </div>
          ) : (
            <div className="text-center py-6 text-destructive">
              <Stethoscope className="h-10 w-10 mx-auto mb-2 opacity-30" />
              <p>No procedures performed</p>
            </div>
          )}
        </CardContent>
      </Card>

      {/* Disposition */}
      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-lg flex items-center">
            <Ambulance className="mr-2 h-5 w-5 text-primary" />
            Disposition & Transport
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
            <div className="space-y-4">
              <div>
                <h3 className="text-sm font-medium text-muted-foreground">Transport Disposition</h3>
                <p className={`font-medium mt-1 ${!patient.TransportDisposition ? 'text-destructive' : ''}`}>
                  {formatMedicalCondition(patient.TransportDisposition)}
                </p>
```

```
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Level of Care</h3>
                  <p className={`font-medium mt-1 ${!patient.LevelOfCareProvided ? 'text-destructive' : ''}`}>
                    {formatMedicalCondition(patient.LevelOfCareProvided)}
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Transport Agency/Unit</h3>
                  <p className={`font-medium mt-1 ${!patient.TransportAgency && !patient.TransportUnit ? 'text-destructive' : ''}`}>
                    {formatMedicalCondition(patient.TransportAgency)}
                    {patient.TransportUnit && `     Unit: ${formatValue(patient.TransportUnit)}`}
                  </p>
                </div>
              </div>

              <div className="space-y-4">
                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Final Patient Acuity</h3>
                  <p className="font-medium mt-1">
                    <Badge variant={getAcuityBadgeVariant(patient.Severity || "")}>
                      {formatMedicalCondition(patient.Severity)}
                    </Badge>
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Primary Care Provider</h3>
                  <p className={`font-medium mt-1 ${!patient.EMSPrimaryCareProvider ? 'text-destructive' : ''}`}>
                    {formatName(patient.EMSPrimaryCareProvider)}
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Transport Reason</h3>
                  <p className={`font-medium mt-1 ${!patient.TransportReason ? 'text-destructive' : ''}`}>
                    {formatMedicalCondition(patient.TransportReason)}
                  </p>
                </div>
              </div>
            </div>
          </CardContent>
        </Card>
      </TabsContent>

      <TabsContent value="medications" className="space-y-6">
        <MedicationHistory patientId={patient.PatientID} />
      </TabsContent>

      <TabsContent value="incident" className="space-y-6">
        {/* Incident Information */}
        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-lg flex items-center">
              <Info className="mr-2 h-5 w-5 text-primary" />
              Incident Information
            </CardTitle>
```

```
            </CardHeader>
            <CardContent>
              <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
                <div className="space-y-3">
                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Incident Number</h3>
                    <p className={`font-medium mt-1 ${!patient.IncidentNumber ? 'text-destructive' : ''}`}>
                      {patient.IncidentNumber || "Not recorded"}
                    </p>
                  </div>

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Service Requested</h3>
                    <p className={`font-medium mt-1 ${!patient.ServiceRequested ? 'text-destructive' : ''}`}>
                      {patient.ServiceRequested || "Not recorded"}
                    </p>
                  </div>

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Primary Role</h3>
                    <p className={`font-medium mt-1 ${!patient.PrimaryRole ? 'text-destructive' : ''}`}>
                      {patient.PrimaryRole || "Not recorded"}
                    </p>
                  </div>
                </div>

                <div className="space-y-3">
                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Response Mode</h3>
                    <p className={`font-medium mt-1 ${!patient.ResponseMode ? 'text-destructive' : ''}`}>
                      {patient.ResponseMode || "Not recorded"}
                    </p>
                  </div>

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">EMS Shift</h3>
                    <p className={`font-medium mt-1 ${!patient.EMSShift ? 'text-destructive' : ''}`}>
                      {patient.EMSShift || "Not recorded"}
                    </p>
                  </div>

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Scene Type</h3>
                    <p className={`font-medium mt-1 ${!patient.SceneType ? 'text-destructive' : ''}`}>
                      {patient.SceneType || "Not recorded"}
                    </p>
                  </div>
                </div>

                <div className="space-y-3">
                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Category</h3>
                    <p className={`font-medium mt-1 ${!patient.Category ? 'text-destructive' : ''}`}>
                      {patient.Category || "Not recorded"}
                    </p>
                  </div>

                  <div>
                    <h3 className="text-sm font-medium text-muted-foreground">Back In Service</h3
```

```
                >
                  <p className={`font-medium mt-1 ${!patient.BackInService ? 'text-destructive'
                   : ''}`}>
                    {patient.BackInService || "Not recorded"}
                  </p>
                </div>

                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Crew Members</h3>
                  <p className={`font-medium mt-1 ${!patient.CrewMembers ? 'text-destructive' :
                   ''}`}>
                    {patient.CrewMembers || "Not recorded"}
                  </p>
                  {patient.NumberOfCrew && (
                    <p className={`text-sm ${!patient.NumberOfCrew ? 'text-destructive' : ''
                  }`}>
                      Number: {patient.NumberOfCrew}
                    </p>
                  )}
                </div>
              </div>
            </div>
          </CardContent>
        </Card>

        {/* Location Information */}
        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-lg flex items-center">
              <Home className="mr-2 h-5 w-5 text-primary" />
              Location Information
            </CardTitle>
          </CardHeader>
          <CardContent>
            <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
              <div>
                <h3 className="text-sm font-medium text-muted-foreground mb-2">Dispatch
                  Location</h3>
                <div className="space-y-2">
                  {patient.DispatchCity && (
                    <p>
                      <span className="font-medium">City:</span>{" "}
                      <span className={!patient.DispatchCity ? 'text-destructive' : ''}>
                        {formatValue(patient.DispatchCity)}
                      </span>
                    </p>
                  )}
                  {patient.DispatchState && (
                    <p>
                      <span className="font-medium">State:</span>{" "}
                      <span className={!patient.DispatchState ? 'text-destructive' : ''}>
                        {formatValue(patient.DispatchState)}
                      </span>
                    </p>
                  )}
                  {patient.DispatchZIP && (
                    <p>
                      <span className="font-medium">ZIP:</span>{" "}
                      <span className={!patient.DispatchZIP ? 'text-destructive' : ''}>
                        {formatValue(patient.DispatchZIP)}
                      </span>
                    </p>
                  )}
                  {patient.DispatchCounty && (
                    <p>
                      <span className="font-medium">County:</span>{" "}
                      <span className={!patient.DispatchCounty ? 'text-destructive' : ''}>
                        {formatValue(patient.DispatchCounty)}
```

```
                    </span>
                  </p>
                )}
              </div>
            </div>

            <div>
              <h3 className="text-sm font-medium text-muted-foreground mb-2">Timing Information</h3>
              <div className="space-y-2">
                {patient.ArrivedOnScene && (
                  <p>
                    <span className="font-medium">Arrived On Scene:</span>{" "}
                    <span className={!patient.ArrivedOnScene ? 'text-destructive' : ''}>
                      {patient.ArrivedOnScene}
                    </span>
                  </p>
                )}
                {patient.FirstOnScene && (
                  <p>
                    <span className="font-medium">First On Scene:</span>{" "}
                    <span className={!patient.FirstOnScene ? 'text-destructive' : ''}>
                      {patient.FirstOnScene}
                    </span>
                  </p>
                )}
                {patient.PatientContactMade && (
                  <p>
                    <span className="font-medium">Patient Contact Made:</span>{" "}
                    <span className={!patient.PatientContactMade ? 'text-destructive' : ''}>
                      {patient.PatientContactMade}
                    </span>
                  </p>
                )}
                {patient.StagePriorToContact && (
                  <p>
                    <span className="font-medium">Stage Prior To Contact:</span>{" "}
                    <span className={!patient.StagePriorToContact ? 'text-destructive' : ''}>
                      {patient.StagePriorToContact}
                    </span>
                  </p>
                )}
              </div>
            </div>
          </div>
        </CardContent>
      </Card>

      {/* Additional Information */}
      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-lg font-bold tracking-tight flex items-center">
            <FileText className="mr-2 h-5 w-5 text-primary" />
            Additional Information
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
            <div className="space-y-3">
              {patient.OtherAgencies && (
                <div>
                  <h3 className="text-sm font-medium text-muted-foreground">Other Agencies</h3>
                  <p className={`font-medium mt-1 ${!patient.OtherAgencies ? 'text-destructive' : ''}`}>
                    {formatMedicalCondition(patient.OtherAgencies)}
                  </p>
                </div>
```

```
                  )}

                  {patient.OtherAgencyOnScene && (
                    <div>
                      <h3 className="text-sm font-medium text-muted-foreground">Other Agency On Scene</h3>
                      <p className={`font-medium mt-1 ${!patient.OtherAgencyOnScene ? 'text-destructive' : ''}`}>
                        {formatMedicalCondition(patient.OtherAgencyOnScene)}
                      </p>
                    </div>
                  )}

                  {patient.NumberOfPatients && (
                    <div>
                      <h3 className="text-sm font-medium text-muted-foreground">Number Of Patients</h3>
                      <p className={`font-medium mt-1 ${!patient.NumberOfPatients ? 'text-destructive' : ''}`}>
                        {formatValue(patient.NumberOfPatients)}
                      </p>
                    </div>
                  )}
                </div>

                <div className="space-y-3">
                  {patient.AlcoholDrugUse && (
                    <div>
                      <h3 className="text-sm font-medium text-muted-foreground">Alcohol/Drug Use</h3>
                      <p className={`font-medium mt-1 ${!patient.AlcoholDrugUse ? 'text-destructive' : ''}`}>
                        {formatMedicalCondition(patient.AlcoholDrugUse)}
                      </p>
                    </div>
                  )}

                  {patient.SignsOfAbuse && (
                    <div>
                      <h3 className="text-sm font-medium text-muted-foreground">Signs Of Abuse</h3>
                      <p className={`font-medium mt-1 ${!patient.SignsOfAbuse ? 'text-destructive' : ''}`}>
                        {formatMedicalCondition(patient.SignsOfAbuse)}
                      </p>
                    </div>
                  )}

                  {patient["5150Hold"] && (
                    <div>
                      <h3 className="text-sm font-medium text-muted-foreground">5150 Hold</h3>
                      <p className={`font-medium mt-1 ${!patient["5150Hold"] ? 'text-destructive' : ''}`}>
                        {formatMedicalCondition(patient["5150Hold"])}
                      </p>
                    </div>
                  )}
                </div>
              </div>
            </CardContent>
          </Card>
        </TabsContent>
      </Tabs>
    </div>
  )
}
```

A.8 stats.ts

```
import { supabase } from '@/utils/supabase/server'
import { NextResponse } from 'next/server'

export const dynamic = 'force-dynamic'

export async function GET() {
  try {
    // Get total patients count
    const { count: totalPatients, error: countError } = await supabase
      .from('PatientData')
      .select('PatientID', { count: 'exact', head: true })

    if (countError) throw countError

    // Get recent patients - last 24 hours
    const twentyFourHoursAgo = new Date(Date.now() - 24 * 60 * 60 * 1000)

    const { data: recentPatients, error: recentError } = await supabase
      .from('PatientData')
      .select('PatientID, PatientName, Age, Gender, Severity, Time')
      .gte('Time', twentyFourHoursAgo.toISOString())
      .order('Time', { ascending: false })

    if (recentError) throw recentError

    // Calculate critical cases from recent patients
    const criticalCases = recentPatients?.filter(patient =>
      patient.Severity?.toLowerCase() === 'critical'
    ) || []

    // Debug logs
    console.log('Dashboard Stats Query:', {
      twentyFourHoursAgo: twentyFourHoursAgo.toISOString(),
      recentPatientsQuery: {
        timeRange: `${twentyFourHoursAgo.toISOString()} to now`
      }
    })

    console.log('Dashboard Stats Results:', {
      totalPatients,
      criticalCasesCount: criticalCases.length,
      recentPatientsCount: recentPatients?.length || 0,
      sampleCriticalCase: criticalCases[0],
      sampleRecentPatient: recentPatients?.[0]
    })

    return NextResponse.json({
      totalPatients: totalPatients || 0,
      criticalCases: criticalCases.length,
      recentPatients: recentPatients?.length || 0,
      recentPatientsList: recentPatients || []
    })

  } catch (error) {
    console.error('Dashboard stats error:', error)
    return NextResponse.json(
      { error: 'Internal Server Error' },
      { status: 500 }
    )
  }
}
```

A.9 globals.css

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
  :root {
    --background: 0 0% 100%;
    --foreground: 240 10% 3.9%;
    --card: 0 0% 100%;
    --card-foreground: 240 10% 3.9%;
    --popover: 0 0% 100%;
    --popover-foreground: 240 10% 3.9%;
    --primary: 210 100% 50%;
    --primary-foreground: 0 0% 98%;
    --secondary: 240 4.8% 95.9%;
    --secondary-foreground: 240 5.9% 10%;
    --muted: 240 4.8% 95.9%;
    --muted-foreground: 240 3.8% 46.1%;
    --accent: 240 4.8% 95.9%;
    --accent-foreground: 240 5.9% 10%;
    --destructive: 0 84.2% 60.2%;
    --destructive-foreground: 0 0% 98%;
    --border: 240 5.9% 90%;
    --input: 240 5.9% 90%;
    --ring: 240 5.9% 10%;
    --radius: 0.75rem;
  }

  .dark {
    --background: 240 10% 3.9%;
    --foreground: 0 0% 98%;
    --card: 240 10% 3.9%;
    --card-foreground: 0 0% 98%;
    --popover: 240 10% 3.9%;
    --popover-foreground: 0 0% 98%;
    --primary: 210 100% 50%;
    --primary-foreground: 240 5.9% 10%;
    --secondary: 240 3.7% 15.9%;
    --secondary-foreground: 0 0% 98%;
    --muted: 240 3.7% 15.9%;
    --muted-foreground: 240 5% 64.9%;
    --accent: 240 3.7% 15.9%;
    --accent-foreground: 0 0% 98%;
    --destructive: 0 62.8% 30.6%;
    --destructive-foreground: 0 0% 98%;
    --border: 240 3.7% 15.9%;
    --input: 240 3.7% 15.9%;
    --ring: 240 4.9% 83.9%;
  }
}

@layer base {
  * {
    @apply border-border;
  }
  body {
    @apply bg-background text-foreground;
  }
}

.dark body {
  background-color: hsl(220 20% 10%);
  color: hsl(210 40% 98%);
}

.dark svg {
  color: hsl(210 40% 98%);
}
```

A.10 styles.css

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  font-family: Arial, Helvetica, sans-serif;
}

@layer utilities {
  .text-balance {
    text-wrap: balance;
  }
}

@layer base {
  :root {
    --background: 0 0% 100%;
    --foreground: 0 0% 3.9%;
    --card: 0 0% 100%;
    --card-foreground: 0 0% 3.9%;
    --popover: 0 0% 100%;
    --popover-foreground: 0 0% 3.9%;
    --primary: 0 0% 9%;
    --primary-foreground: 0 0% 98%;
    --secondary: 0 0% 96.1%;
    --secondary-foreground: 0 0% 9%;
    --muted: 0 0% 96.1%;
    --muted-foreground: 0 0% 45.1%;
    --accent: 0 0% 96.1%;
    --accent-foreground: 0 0% 9%;
    --destructive: 0 84.2% 60.2%;
    --destructive-foreground: 0 0% 98%;
    --border: 0 0% 89.8%;
    --input: 0 0% 89.8%;
    --ring: 0 0% 3.9%;
    --chart-1: 12 76% 61%;
    --chart-2: 173 58% 39%;
    --chart-3: 197 37% 24%;
    --chart-4: 43 74% 66%;
    --chart-5: 27 87% 67%;
    --radius: 0.5rem;
    --sidebar-background: 0 0% 98%;
    --sidebar-foreground: 240 5.3% 26.1%;
    --sidebar-primary: 240 5.9% 10%;
    --sidebar-primary-foreground: 0 0% 98%;
    --sidebar-accent: 240 4.8% 95.9%;
    --sidebar-accent-foreground: 240 5.9% 10%;
    --sidebar-border: 220 13% 91%;
    --sidebar-ring: 217.2 91.2% 59.8%;
  }
  .dark {
    --background: 0 0% 3.9%;
    --foreground: 0 0% 98%;
    --card: 0 0% 3.9%;
    --card-foreground: 0 0% 98%;
    --popover: 0 0% 3.9%;
    --popover-foreground: 0 0% 98%;
    --primary: 0 0% 98%;
    --primary-foreground: 0 0% 9%;
    --secondary: 0 0% 14.9%;
    --secondary-foreground: 0 0% 98%;
    --muted: 0 0% 14.9%;
    --muted-foreground: 0 0% 63.9%;
    --accent: 0 0% 14.9%;
    --accent-foreground: 0 0% 98%;
```

```
    --destructive: 0 62.8% 30.6%;
    --destructive-foreground: 0 0% 98%;
    --border: 0 0% 14.9%;
    --input: 0 0% 14.9%;
    --ring: 0 0% 83.1%;
    --chart-1: 220 70% 50%;
    --chart-2: 160 60% 45%;
    --chart-3: 30 80% 55%;
    --chart-4: 280 65% 60%;
    --chart-5: 340 75% 55%;
    --sidebar-background: 240 5.9% 10%;
    --sidebar-foreground: 240 4.8% 95.9%;
    --sidebar-primary: 224.3 76.3% 48%;
    --sidebar-primary-foreground: 0 0% 100%;
    --sidebar-accent: 240 3.7% 15.9%;
    --sidebar-accent-foreground: 240 4.8% 95.9%;
    --sidebar-border: 240 3.7% 15.9%;
    --sidebar-ring: 217.2 91.2% 59.8%;
  }
}

@layer base {
  * {
    @apply border-border;
  }
  body {
    @apply bg-background text-foreground;
  }
}
```

A.11 ActivePatient.cs

```
using Newtonsoft.Json.Linq;

public static class ActivePatient
{
    public static string PatientID { get; set; }
    public static JObject PatientJSON { get; set; }
}
```

A.12 AudioFileLogger.cs

```
/*
 * EMT Vision Dashboard
 * Copyright    2025 Logan Calder. All Rights Reserved.
 *
 * This software and its contents are protected by copyright law. The EMT Vision Dashboard,
 * including but not limited to its source code, design, architecture, and implementation,
 * is the exclusive property of Logan Calder.
 *
 * Usage Restrictions:
 * Without explicit written consent from Logan Calder, you are strictly prohibited from:
 * 1. Copying, reproducing, or distributing any part of this codebase
 * 2. Modifying, adapting, or creating derivative works based on this software
 * 3. Using this code or its design as a reference or foundation for other projects
 * 4. Reverse engineering, decompiling, or disassembling the software
 * 5. Removing or altering any copyright notices or proprietary labels
 *
 * Legal Notice:
 * Any unauthorized use, reproduction, or distribution of this software or its contents
 * is strictly prohibited and may result in severe legal consequences. This includes but
 * is not limited to civil and criminal penalties under applicable copyright laws.
 *
 * For inquiries regarding usage rights or permissions, please contact:
 * Logan Calder | lcalder@scu.edu
 *
```

```
 * Disclaimer:
 * This software is provided "as is" without warranty of any kind, either express or implied.
 * The author assumes no responsibility for any damages arising from the use of this software.
 */

using System;
using System.IO;
using UnityEngine;
using System.Collections;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using UnityEngine.Networking;
using System.Text;
using System.Collections.Generic;
using System.Text.Json.Serialization;
using System.Net.Http;
using System.Net.Http.Headers;

// AudioFileLogger.cs
// This script is used to monitor a directory for new audio files, then process them through
    Microsoft Azure Speech Services and OpenAI GPT-4o.
//
// Usage: Attach to a GameObject in the scene. Call StartRecording() to begin recording. Call
    StopRecording() to stop recording.

public class AudioFileLogger : MonoBehaviour
{
    // IMPORTANT: API Keys & Configs (accessed from appsettings.json, not included in repo)
    // AZURE DATA

    private string azureKey = "
    E3RJqLwLmIfbN9tTb8JyHn4myM7Y7OJ1M1mgLKdX3fzcyfr625SVJQQJ99BCAC4f1cMXJ3w3AAAYACOGYVtf";
    private string region = "westus";
    // OPEN AI DATA
    private string openAIKey = "sk-proj-xSmEbAuZzh3V2ChhulNd1oSOsLyxO5SqlhIyMQM-6
    yiwRIhec3M3vZCOQwM4POD6J6lfZE9V6bT3BlbkFJawka0V444zByXe9kPRX0c0OYXPpQjXIlqilS_4f9gc7PofF38t66xpeLzFRJXhYVomkb
    ";
    private string openAIURL = "https://api.openai.com/v1/chat/completions";
    // SUPABASE DATA
    private string supabaseUrl = "https://yuwrsuaqhbbfxqlrybgg.supabase.co/rest/v1/PatientData";
    private string supabaseUrlMedication = "https://yuwrsuaqhbbfxqlrybgg.supabase.co/rest/v1/
    Medications";
    private string supabaseKey = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Inl1d3JzdWFxaGJiZnhxbHJ5YmdnIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTc0MDA3N
    .oDOmFPwxbq9FosgsJb4YPs3xwVTPdNL4ihNlw3oZwTk";
    // File monitoring variables
    private string conversation = "";
    private string medicineConversation = "";
    private SpeechRecognizer recognizer;
    private AudioConfig audioConfig;
    private SpeechConfig speechConfig;

    private string directoryPath;
    private FileSystemWatcher fileWatcher;
    private string patientId = "";

    private string json_template;
    private string current_data = "";

    private string timestamp;
    private string QueuedText = null;
    private string QueuedMedicineText = null;
    string medication_json_template = $@"{{""MedicationID"":""MedicationName"":""
    QuantityAdministered"":""Timestamp"":""PatientID"":}}";
    string medicinePrompt = @"
Youre GPT-4o mini, an advanced AI agent responsible for transcribing and recording confidential
    patient information into a structured JSON format. Your primary task is to extract, filter,
```

```
and organize relevant information from the provided input while preserving existing data that remains relevant. Your output must be a fully formatted JSON string, ensuring accuracy, consistency, and adherence to predefined constraints. Follow the instructions below with utmost precision:

---

**CSV Format** (column headers):
`Medication`,`Dosage`,`DosageUnits`,`Route`,`TimeGiven`,`PersonAdministering`,`Effect`,`MedicationID`,`PatientID`,`Notes`

---

**General Instructions:**
1. **Maintain Data Integrity:** Do not delete existing data unless new information explicitly overrides it or renders it invalid.
2. **Do Not Fabricate Data:** If a field has no information, leave it blank but do not create speculative or incorrect data.
3. **Retain Identifying Information:** Fields such as `PatientID`, `PatientName`, `Age`, `WeightKg`, and `ZIPCode` should remain singular values and must not be converted into lists.
4. **Reformat for Clarity:** Simplify and standardize input when possible. For example, `Patient has a history of severe hypertension` should be recorded as `hypertension.`
5. **Follow Structured Field Assignments:** Ensure every piece of data is allocated to its correct field and is not misplaced.
6. **MOST IMPORTANT:** ONLY RETURN A JSON OBJECT. DO NOT RETURN ANYTHING ELSE. DO NOT ENCASE IN ```JSON``` TAGS. RETURN IN A VALID JSON FORMAT. DO NOT ADD COMMENTS. IF JSON IS INVALID, IT WILL FAIL.

---

**Field Definitions:**
- **`MedicationID`**: (uuid) The unique identifier for the medication.
- **`MedicationName`**: (Text) Name of the drug administered.
- **`QuantityAdministered`**: (int8) Numeric amount administered in ml.
- **`Timestamp`**: (timestamptz) Timestamp of when the medication was administered.
- **`PatientID`**: (text)The patient ID of the patient being treated. This will be provided to you.

---

**Input Types:**
1. `Medication ID`: The unique identifier for the medication. This will be provided to you.
2. `Transcript`: A brief transcription of what was said by a clinician or responder.
3. `Timestamp`: The timestamp of the transcription.
4. `Template`: A template for the JSON object.
5. `Patient ID`: The patient ID of the patient being treated. This will be provided to you.

---

**MOST IMPORTANT:** RETURN A SINGLE JSON. NO QUOTES UNLESS INSIDE A FIELD. LEAVE UNMENTIONED FIELDS EMPTY.";
    string gptPrompt = @"
        You are GPT-4o mini, an advanced AI agent responsible for transcribing and recording confidential patient information into a structured JSON format. Your primary task is to extract, filter, and organize relevant information from the provided input while preserving existing data that remains relevant. Your output must be a fully formatted JSON string, ensuring accuracy, consistency, and adherence to predefined constraints. Follow the instructions below with utmost precision:

**General Instructions:**
1. **Maintain Data Integrity:** Do not delete existing data unless new information explicitly overrides it or renders it invalid.
2. **Do Not Fabricate Data:** If a field has no information, leave it blank but do not create speculative or incorrect data.
3. **Retain Identifying Information:** Fields such as `PatientID`, `PatientName`, `Age`, `WeightKg`, and `ZIPCode` should remain singular values and must not be converted into lists.
4. **Append Multiple Entries Where Appropriate:** If multiple values apply to a field (e.g., medications, symptoms, or past medical history), store them as a list while ensuring they are
```

```
   relevant and distinct.
5. **Reformat for Clarity:** Simplify and standardize input when possible. For example, ‘Patient has a history of severe hypertension‘ should be recorded as ‘hypertension.‘
6. **Follow Structured Field Assignments:** Ensure every piece of data is allocated to its correct field and is not misplaced.
7. **MOST IMPORTANT:** ONLY RETURN A JSON OBJECT. DO NOT RETURN ANYTHING ELSE. DO NOT ENCASE IN ‘‘‘JSON‘‘‘ TAGS. RETURN IN A VALID JSON FORMAT. DO NOT ADD COMMENTS. IF JSON IS INVALID, IT WILL FAIL.

---

**Input Types:**
1. **Database Record (Existing Data):**
   - You will receive a JSON object ‘current_data‘ containing previously recorded patient information. Use this as the baseline data.
2. **Recent Audio Transcription (40s Context):**
   - This contains the latest spoken details from an emergency medical responder, nurse, or doctor.
3. **Additional Context:**
   - Information regarding the circumstances under which the data is recorded (e.g., emergency setting, follow-up assessment).

Additionally, you are provided with ‘json_template‘, which contains the empty JSON structure that must be populated.

---

**Field Processing Rules:**
**1. Patient Identification & Demographics**
- **‘PatientID‘** (Mandatory, Unique)     **Do not modify or delete.**
- **‘PatientName‘** (String)     Retain unless an explicit correction is provided.
- **‘Age‘** (Integer/String)     Ensure only one value is present.
- **‘Gender‘** (M/F)     Extract from input if mentioned; use ‘M‘ for male and ‘F‘ for female.
- **‘HomeAddress‘**     Only update if a full address is provided.
- **‘City, County, State, ZIPCode‘**     Populate if new, but do not override unless certain.
- **‘Race‘**     Retain or update only if explicitly stated.

**2. Incident Details**
- **‘IncidentNumber‘**     Must remain singular.
- **‘ServiceRequested‘**     Include service type (e.g., BLS, ALS, transport).
- **‘PrimaryRole‘**     Define based on responder‘s role (e.g., paramedic, firefighter, nurse).
- **‘ResponseMode‘**     Extract from responder dialogue (e.g., ‘Code 3‘ or ‘Non-emergent‘).

**3. Scene & Patient Interaction**
- **‘SceneType‘**     Capture relevant setting (e.g., ‘residential‘, ‘public street‘).
- **‘Category‘**     Medical, trauma, behavioral, etc.
- **‘CrewMembers‘**     List all names or IDs if provided.
- **‘NumberOfCrew‘**     Integer, representing responding crew.
- **‘PatientContactMade‘**     Boolean (true if contact established).

**4. Clinical Observations & Symptoms**
- **‘PrimaryComplaint‘**     Capture the main reason for the call.
- **‘OtherSymptoms‘**     Extract all additional symptoms.
- **‘AlcoholDrugUse‘**     Mention only if stated.
- **‘InitialAcuity‘**     Determine severity (e.g., minor, severe).
- **‘CardiacArrest‘**     Boolean (true if present).
- **‘PossibleInjury‘**     Boolean (true if reported).
- **‘SignsOfAbuse‘**     Boolean (true if noted).

**5. Medical History & Medications**
- **‘PastMedicalHistory‘**     Convert spoken history into a concise list.
- **‘CurrentMedications‘**     Extract and list.
- **‘MedicationAllergies‘**     Ensure clarity (e.g., ‘penicillin‘ instead of ‘I can‘t take that one antibiotic‘).

**6. Vital Signs**
- **Heart Rate, Blood Pressure, Respiratory Rate, SPO2, Temperature, Glucose**     Numeric values only.
```

```
- **GCS Score & Breakdown**     Ensure accurate parsing of Eye, Verbal, and Motor scores.

**7. Assessment & Impressions**
- **`PrimaryImpression`**     The clinician`s primary diagnosis.
- **`PrimarySymptom`**     The main reported symptom.
- **`OtherSymptoms`**     List any additional complaints.

**8. Treatment & Procedures**
- **`Medication`, `Dosage`, `Route`**     Extract administered drugs and details.
- **`Procedure`**     Include any procedures performed.
- **`IVLocation, Size, Attempts, Successful`**     Record all IV details.

**9. Transport & Disposition**
- **`CrewDisposition`**     Capture decision made by the crew (e.g., treated and released,
    transported).
- **`TransportDisposition`**     Specify transport details.
- **`LevelOfCareProvided`**     Define level (BLS, ALS, etc.).
- **`TransportReason`**     Capture the reason for transport.
- **`TransportAgency, TransportUnit`**     Include agency and vehicle ID.

**10. Severity Determination**
- **`Severity`**     Assign one of the following based on patient condition:
  - `Undetermined`
  - `Good`
  - `Fair`
  - `Serious`
  - `Critical`

---

**Final Output Requirements:**
- **Complete JSON Object:** Ensure every field is present, even if empty.
- **Flat Structure:** No nested structures unless explicitly needed.
- **No Additional Formatting:** Output must be a valid JSON string without extra spaces or
    newlines.
- **Preserve All Data:** Retain prior records unless explicitly replaced by new input.
- **Timestamp:** You should add the timestamp given into the Time field.

Your task is to process patient data while maintaining compliance with these rules. Follow these
    instructions meticulously to ensure high data fidelity and accuracy.

**MOST IMPORTANT:** RETURN A SINGLE VALID CSV ROW IN THE EXACT ORDER OF COLUMNS GIVEN. NO HEADERS
    . NO EXTRA TEXT. NO JSON. NO QUOTES UNLESS INSIDE A FIELD. LEAVE UNMENTIONED FIELDS EMPTY.";

    // AppSettings class
    // This class is used to store the API keys and region.
    // It is accessed from appsettings.json, not included in repo.
    [Serializable]
    public class AppSettings
    {
        public string OpenAIApiKey;
        public string AzureSubscriptionKey;
        public string AzureRegion;
    }


    private async void OnDestroy()
    {
        if (recognizer != null)
        {
            await recognizer.StopContinuousRecognitionAsync();
            recognizer.Dispose();
        }
    }

    public async void StartRecording()
    {
        GeneratePatientIdAndTimestamp();
```

```
        if (ActivePatient.PatientID != null)
        {
            Debug.Log("was null");
            patientId = ActivePatient.PatientID;
        }
        Debug.Log($"Starting recording for patient ID: {ActivePatient.PatientID}");

        json_template = $@"{{""PatientID"":""{patientId}"":""PatientName"":""Age"":""Gender"":""HomeAddress"":""City"":""County"":""State"":""ZIPCode"":""WeightKg"":""Race"":""IncidentNumber"":""ServiceRequested"":""OtherAgencies"":""PrimaryRole"":""ResponseMode"":""EMSShift"":""DispatchCity"":""DispatchState"":""DispatchZIP"":""DispatchCounty"":""SceneType"":""Category"":""BackInService"":""CrewMembers"":""NumberOfCrew"":""OtherAgencyOnScene"":""NumberOfPatients"":""PatientContactMade"":""ArrivedOnScene"":""FirstOnScene"":""StagePriorToContact"":""PrimaryComplaint"":""Duration"":""TimeUnits"":""AlcoholDrugUse"":""InitialAcuity"":""CardiacArrest"":""PossibleInjury"":""BaseContactMade"":""SignsOfAbuse"":""5150Hold"":""PastMedicalHistory"":""CurrentMedications"":""MedicationAllergies"":""AdvanceDirectives"":""HeartRate"":""BloodPressure"":""RespiratoryRate"":""SPO2"":""Temperature"":""Glucose"":""GCS_Eye"":""GCS_Verbal"":""GCS_Motor"":""GCS_Score"":""GCS_Qualifier"":""MentalStatus"":""AbdomenExam"":""ChestExam"":""BackSpineExam"":""SkinAssessment"":""EyeExam_Bilateral"":""EyeExam_Left"":""EyeExam_Right"":""LungExam"":""ExtremitiesExam"":""PrimaryImpression"":""PrimarySymptom"":""OtherSymptoms"":""SymptomOnset"":""TypeOfPatient"":""MedTime"":""MedCrewID"":""Medication"":""Dosage"":""MedUnits"":""Route"":""MedResponse"":""MedComplications"":""ProcTime"":""ProcCrewID"":""Procedure"":""ProcLocation"":""IVLocation"":""Size"":""Attempts"":""Successful"":""ProcResponse"":""PatientEvaluationCare"":""CrewDisposition"":""TransportDisposition"":""LevelOfCareProvided"":""TransferredCareAt"":""FinalPatientAcuity"":""TurnaroundDelay"":""TransportAgency"":""TransportUnit"":""LevelOfTransport"":""EMSPrimaryCareProvider"":""TransportReason"":""CrewSignature"":""CrewMember_PPE"":""PPEUsed"":""SuspectedExposure"":""MonitorTime"":""MonitorEventType"":""Time"":""Severity"":}}";
        conversation = ""; // Reset conversation
        medicineConversation = "";

        // Initialize speech recognition if not already initialized
        if (recognizer == null)
        {
            await InitializeSpeechRecognition();
        }

        await recognizer.StartContinuousRecognitionAsync();
        Debug.Log("Started recording...");
    }
    private async Task InitializeSpeechRecognition()
    {

        speechConfig = SpeechConfig.FromSubscription(azureKey, region);
        speechConfig.SpeechRecognitionLanguage = "en-US"; // Change as needed

        // Use the default microphone
        audioConfig = AudioConfig.FromDefaultMicrophoneInput();
        recognizer = new SpeechRecognizer(speechConfig, audioConfig);

        // Subscribe to recognition events
        recognizer.Recognizing += (s, e) =>
        {
            Debug.Log($"Recognizing: {e.Result.Text}");

        };

        recognizer.Recognized += (s, e) =>
        {
            if (e.Result.Reason == ResultReason.RecognizedSpeech)
            {
                Debug.Log($"Final Result: {e.Result.Text}");
                if (e.Result.Text.ToLower().Contains("medicine") || e.Result.Text.ToLower().Contains("medication"))
                {
                    string timestamp = GenerateTimestamp();
                    medicineConversation += $"(\n[{timestamp}] PatientID: {patientId} : {e.Result
```

```
.Text}) ";
                }

                conversation += " " + e.Result.Text;
                Debug.Log($"Conversation: {conversation}");
                Debug.Log($"Medicine Conversation: {medicineConversation}");

            }
            else if (e.Result.Reason == ResultReason.NoMatch)
            {
                Debug.Log("No speech recognized.");
            }

        };

        recognizer.Canceled += (s, e) =>
        {
            Debug.LogError($"Canceled: {e.Reason}, Error: {e.ErrorDetails}");
        };

        recognizer.SessionStopped += (s, e) =>
        {
            Debug.Log("Speech session stopped.");
        };

        Debug.Log("Speech recognition initialized...");
    }

    public async void StopRecording()
    {
        await recognizer.StopContinuousRecognitionAsync();
        Debug.Log("Stopped recording. Final conversation:");
        Debug.Log(conversation);
        QueuedText = conversation;
        QueuedMedicineText = medicineConversation;
    }

    // LoadConfiguration()
    // This function loads the configuration from appsettings.json.
    // You must import this yourself as git will ignore it.
    // private void LoadConfiguration()
    // {
    //     string configPath = Path.Combine(Directory.GetParent(Application.dataPath).FullName, "
appsettings.json");
    //     if (File.Exists(configPath))
    //     {
    //         try
    //         {
    //             string jsonContent = File.ReadAllText(configPath);
    //             var config = JsonUtility.FromJson<AppSettings>(jsonContent);

    //             openAIKey = config?.OpenAIApiKey;
    //             region = config?.AzureRegion;
    //             azureKey = config?.AzureSubscriptionKey;
    //         }
    //         catch (Exception ex)
    //         {
    //             Debug.LogError($"Error loading configuration: {ex.Message}");
    //         }
    //     }
    //     else
    //     {
    //         Debug.LogError($"appsettings.json not found at: {configPath}");
    //         Debug.LogError($"Please ensure appsettings.json exists in the project root
directory: {Path.GetDirectoryName(configPath)}");
    //     }
    // }

```

```
    // GeneratePatientId()
    // This function generates a Patient ID in the format PAT-YYYYMMDD-HHMMSS-XXXX
    private string GeneratePatientId()
    {
        string datePart = DateTime.Now.ToString("yyyyMMdd");
        string timePart = DateTime.Now.ToString("HHmmss");
        string randomPart = Guid.NewGuid().ToString().Substring(0, 4); // Get the first 4 characters of a new GUID
        return $"PAT-{datePart}-{timePart}-{randomPart}";
    }

    private string GenerateTimestamp()
    {
        // Get current UTC time
        DateTime utcNow = DateTime.UtcNow;
        // Convert to Pacific Time (UTC-8 for PST, UTC-7 for PDT)
        TimeSpan pacificOffset = TimeZoneInfo.Local.GetUtcOffset(DateTime.Now);
        DateTime pacificTime = utcNow.Add(pacificOffset);
        return pacificTime.ToString("yyyy-MM-dd HH:mm:ss");
    }

    public void GeneratePatientIdAndTimestamp()
    {
        patientId = GeneratePatientId();
        timestamp = GenerateTimestamp();
        Debug.Log($"Generated Patient ID: {patientId}");
        Debug.Log($"Generated Timestamp: {timestamp}");
    }

    // When we detect a new recording, we need to process the text.
    void Update()
    {
        // Process any queued text
        if (QueuedText != null)
        {
            string textToProcess = QueuedText;
            QueuedText = null;
            StartCoroutine(FetchCurrentDataAndProcessText(textToProcess));
        }

        if (QueuedMedicineText != null)
        {
            string textToProcess = QueuedMedicineText;
            Debug.Log("MEDICINEtextToProcess: " + textToProcess);
            QueuedMedicineText = null;
            StartCoroutine(SendOpenAIRequest(textToProcess, true));
        }
    }

    // New method to fetch current data before processing text
    private IEnumerator FetchCurrentDataAndProcessText(string textToProcess, bool isMedicine = false)
    {
        // Fetch current data for this patient ID
        string fetchUrl = $"{supabaseUrl}?PatientID=eq.{patientId}";
        UnityWebRequest fetchRequest = UnityWebRequest.Get(fetchUrl);
        fetchRequest.SetRequestHeader("apikey", supabaseKey);
        fetchRequest.SetRequestHeader("Authorization", "Bearer " + supabaseKey);

        yield return fetchRequest.SendWebRequest();

        if (fetchRequest.result == UnityWebRequest.Result.Success)
        {
            string response = fetchRequest.downloadHandler.text;
            // Check if we got any data back (empty array means no existing record)
            if (response != null && response.Length > 2 && !response.Equals("[]"))
            {
                // Remove the array brackets since we expect only one record
```

```
                current_data = response.Trim().TrimStart('[').TrimEnd(']');
                Debug.Log("Fetched current data: " + current_data);
            }
            else
            {
                current_data = "{}"; // Set to empty JSON object if no data found
                Debug.Log("No existing data found for patient ID: " + patientId);
            }
        }
        else
        {
            Debug.LogWarning("Failed to fetch existing data: " + fetchRequest.error);
            current_data = "{}"; // Set to empty JSON object if fetch fails
        }

        // Now process the text with the updated current_data
        StartCoroutine(SendOpenAIRequest(textToProcess));
    }

    // SendOpenAIRequest(string rawText)
    // Parameters: rawText - the text to be sent to OpenAI.
    // Returns: None
    // This function sends the transcribed text to OpenAI and returns the JSON data.
    private IEnumerator SendOpenAIRequest(string rawText, bool isMedicine = false)
    {
        Debug.Log($"     Sending to OpenAI: {rawText}");
        Debug.Log($"Current data being used: {current_data}");

        string escapedSystemPrompt = "";
        string escapedUserContent = "";
        string medicationID = "";

        // Properly escape strings for JSON
        if (isMedicine)
        {
            medicationID = Guid.NewGuid().ToString();
            Debug.Log("Medication  true");
            escapedSystemPrompt = EscapeJsonString(medicinePrompt);
            escapedUserContent = EscapeJsonString($"Medication ID: {medicationID}\nTranscript: {rawText}\nEmpty template: {medication_json_template}\nTimestamp: {timestamp}\nPatient ID: {patientId}");
        }
        else
        {
            escapedSystemPrompt = EscapeJsonString(gptPrompt);
            escapedUserContent = EscapeJsonString($"Audio input: {rawText}\nEmpty template: {json_template}\nCurrent db info: {current_data}\nTimestamp: {timestamp}\nPatient ID: {patientId}");
        }

        // Construct JSON payload with properly escaped strings
        string jsonPayload = @"{
            ""model"": ""gpt-4o"",
            ""messages"": [
                {
                    ""role"": ""system"",
                    ""content"": """ + escapedSystemPrompt + @"""
                },
                {
                    ""role"": ""user"",
                    ""content"": """ + escapedUserContent + @"""
                }
            ]
        }";

        Debug.Log($"JSON payload size: {jsonPayload.Length} bytes");
        Debug.Log($"JSON Payload preview: {(jsonPayload.Length > 200 ? jsonPayload.Substring(0, 200) + "..." : jsonPayload)}");
```

```
        byte[] bodyRaw = Encoding.UTF8.GetBytes(jsonPayload);
        Debug.Log($"Request body size: {bodyRaw.Length} bytes");

        UnityWebRequest request = new UnityWebRequest(openAIURL, "POST");
        request.uploadHandler = new UploadHandlerRaw(bodyRaw);
        request.downloadHandler = new DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type", "application/json");
        request.SetRequestHeader("Authorization", "Bearer " + openAIKey);

        // Log request details before sending
        Debug.Log($"Sending request to URL: {openAIURL}");
        Debug.Log($"Using API key: {openAIKey.Substring(0, 10)}..."); // Only show first 10 chars for security

        yield return request.SendWebRequest();

        // Log detailed response information
        Debug.Log($"Response code: {request.responseCode}");

        if (request.result == UnityWebRequest.Result.Success)
        {
            Debug.Log("OpenAI request successful!");
            string responseText = request.downloadHandler.text;

            string responseTextContent = ExtractMessage(request.downloadHandler.text);
            responseTextContent = responseTextContent.Replace("\"PatientID\":\"\"", $"\"PatientID\":\"{patientId}\"");
            responseTextContent = responseTextContent.Replace("\"Time\":\"\"", $"\"Time\":\"{timestamp}\"");

            StartCoroutine(SendJsonToSupabase(responseTextContent, isMedicine));

        }
        else
        {
            Debug.LogError($"OpenAI request failed with error: {request.error}");
            Debug.LogError($"Error details: {request.downloadHandler?.text ?? "No response body"}");

            // Check for common error causes
            if (request.responseCode == 401)
            {
                Debug.LogError("Authentication error: Check if your OpenAI API key is valid");
            }
            else if (request.responseCode == 400)
            {
                Debug.LogError("Bad request: The request format might be incorrect or the prompt might be too long");
            }
            else if (request.responseCode == 429)
            {
                Debug.LogError("Rate limit exceeded: You might be sending too many requests or have exceeded your quota");
            }
            else if (request.responseCode == 500)
            {
                Debug.LogError("Server error: OpenAI's servers might be experiencing issues");
            }

            // Try to log the first part of the payload for debugging
            if (jsonPayload.Length > 500)
            {
                Debug.LogError($"First 500 chars of payload: {jsonPayload.Substring(0, 500)}...");
            }
        }
    }
```

```
    // Helper method to properly escape strings for JSON
    private string EscapeJsonString(string str)
    {
        if (string.IsNullOrEmpty(str))
            return string.Empty;

        // Replace special characters with escape sequences
        str = str.Replace("\\", "\\\\");
        str = str.Replace("\"", "\\\"");
        str = str.Replace("\n", "\\n");
        str = str.Replace("\r", "\\r");
        str = str.Replace("\t", "\\t");
        str = str.Replace("\b", "\\b");
        str = str.Replace("\f", "\\f");

        return str;
    }

    // ExtractMessage(string jsonResponse)
    // Parameters: jsonResponse - the response from OpenAI.
    // Returns: A string of the JSON data.
    // This function extracts the JSON data from the response from OpenAI.
    private string ExtractMessage(string jsonResponse)
    {
        OpenAIResponse response = JsonUtility.FromJson<OpenAIResponse>(jsonResponse);
        return response.choices[0].message.content;
    }

    [System.Serializable]
    public class OpenAIResponse
    {
        public Choice[] choices;
    }

    [System.Serializable]
    public class Choice
    {
        public Message message;
    }

    [System.Serializable]
    public class Message
    {
        public string content;
    }

    // SendJsonToSupabase(string jsonData)
    // Parameters: jsonData - the JSON string to be sent to Supabase.
    // Returns: None
    // This function sends the provided JSON data to Supabase.

    private IEnumerator SendJsonToSupabase(string jsonData, bool isMedicine = false)
    {
        Debug.Log("Current JSON: " + jsonData);

        // First, fetch existing data for this patient
        string fetchUrl;
        if (isMedicine)
        {
            fetchUrl = $"{supabaseUrlMedication}";
        }
        else
        {
            fetchUrl = $"{supabaseUrl}?PatientID=eq.{patientId}";
        }
        UnityWebRequest fetchRequest = UnityWebRequest.Get(fetchUrl);
        fetchRequest.SetRequestHeader("apikey", supabaseKey);
```

```
        fetchRequest.SetRequestHeader("Authorization", "Bearer " + supabaseKey);

        yield return fetchRequest.SendWebRequest();

        bool rowExists = false;

        Debug.Log("Sending JSON to Supabase...");
        Debug.Log("Final JSON to send: " + jsonData);

        byte[] bodyRaw = Encoding.UTF8.GetBytes(jsonData);

        // If row exists, use PATCH to update it, otherwise use POST to create a new row
        string requestMethod = rowExists ? "PATCH" : "POST";
        string requestUrl;
        if (isMedicine)
        {
            requestUrl = supabaseUrlMedication;
        }
        else
        {
            requestUrl = rowExists ? $"{supabaseUrl}?PatientID=eq.{patientId}" : supabaseUrl;
        }

        Debug.Log($"Using {requestMethod} request to {requestUrl}");

        UnityWebRequest updateRequest = new UnityWebRequest(requestUrl, requestMethod);
        updateRequest.uploadHandler = new UploadHandlerRaw(bodyRaw);
        updateRequest.downloadHandler = new DownloadHandlerBuffer();
        updateRequest.SetRequestHeader("Content-Type", "application/json");
        updateRequest.SetRequestHeader("apikey", supabaseKey);
        updateRequest.SetRequestHeader("Authorization", "Bearer " + supabaseKey);

        // For both POST and PATCH, we want to return the representation
        updateRequest.SetRequestHeader("Prefer", "return=representation");

        // For POST specifically, we want to handle duplicates by merging
        if (requestMethod == "POST")
        {
            updateRequest.SetRequestHeader("Prefer", "resolution=merge-duplicates,return=representation");
        }

        yield return updateRequest.SendWebRequest();

        if (updateRequest.result == UnityWebRequest.Result.Success)
        {
            Debug.Log($"Successfully {(rowExists ? "updated" : "created")} record in Supabase: " + updateRequest.downloadHandler.text);
        }
        else
        {
            Debug.LogError($"Error {(rowExists ? "updating" : "creating")} record in Supabase: " + updateRequest.error);
            Debug.LogError("Response: " + updateRequest.downloadHandler.text);

            // If PATCH fails, try POST as a fallback
            if (requestMethod == "PATCH")
            {
                Debug.Log("PATCH failed, trying POST as fallback...");
                yield return StartCoroutine(FallbackPostToSupabase(jsonData));
            }
        }
    }

    // Fallback method to use POST if PATCH fails
    private IEnumerator FallbackPostToSupabase(string jsonData)
    {
        byte[] bodyRaw = Encoding.UTF8.GetBytes(jsonData);
```

```
        UnityWebRequest request = new UnityWebRequest(supabaseUrl, "POST");
        request.uploadHandler = new UploadHandlerRaw(bodyRaw);
        request.downloadHandler = new DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type", "application/json");
        request.SetRequestHeader("apikey", supabaseKey);
        request.SetRequestHeader("Authorization", "Bearer " + supabaseKey);
        request.SetRequestHeader("Prefer", "resolution=merge-duplicates,return=representation");

        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
            Debug.Log("Fallback POST successful: " + request.downloadHandler.text);
        }
        else
        {
            Debug.LogError("Fallback POST also failed: " + request.error);
            Debug.LogError("Response: " + request.downloadHandler.text);
        }
    }
}
```

A.13 ClearText.cs

```
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using Newtonsoft.Json;

public class ClearJsonValues : MonoBehaviour
{
    public string filePath = "Assets/StreamingAssets/patient_data.json"; // Update the path if needed

    public void ClearValues()
    {
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            Dictionary<string, string> data = JsonConvert.DeserializeObject<Dictionary<string, string>>(json);

            if (data != null)
            {
                foreach (var key in new List<string>(data.Keys))
                {
                    data[key] = "";
                }

                string updatedJson = JsonConvert.SerializeObject(data, Formatting.Indented);
                File.WriteAllText(filePath, updatedJson);
                Debug.Log("JSON values cleared.");
            }
        }
        else
        {
            Debug.LogError("JSON file not found!");
        }
    }
}
```

A.14 DynamicChecklist.cs

```
using UnityEngine;
using UnityEngine.UI;
```

```
using System.Collections.Generic;

public class DynamicChecklist : MonoBehaviour
{
    public GameObject scrollContent; // Assign Scroll View's Content here
    public Button buttonPrefab; // Assign a Button prefab in the Inspector
    public Toggle checkboxPrefab; // Assign a Checkbox prefab in the Inspector
    public Button backButtonPrefab; // Assign a Back Button prefab in the Inspector

    // Sample Data
    private string[] mainMenuOptions = { "Adult Protocols", "Pediatric Protocols", "Standard Protocols", "Procedures", "Optional Scope", "Policies" };
    private Dictionary<string, string[]> subMenuOptions = new Dictionary<string, string[]>
    {
        { "Adult Protocols", new[] {  "A01. Abdominal Emergencies", "A02. Seizure", "A03. Hypoglycemia", "A04. Sepsis", "A05. Bradycardia", "A06. Burns", "A07. Cardiac Arrest", "A08. Chest Pain-Suspected Cardiac Ischemia", "A09. Environmental Emergencies", "A10. Shock", "A11. Respiratory Distress", "A12. Allergic Reaction / Anaphylaxis", "A13. Stroke", "A14. Tachycardia with Pulses", "A15. Poisoning and Overdose", "A16. Trauma Care", "A18. Gynecological and Obstetrical Emergencies", "A19. Crush Injury Syndrome", "A20. Behavioral Emergency - Combative"  } },
        { "Pediatric Protocols", new[] { "Pediatric Procedure 1", "Pediatric Procedure 2" } },
        { "Standard Protocols", new[] { "Standard Procedure 1", "Standard Procedure 2", "Standard Procedure 3" } },
        { "Procedures", new[] { "Procedure 1", "Procedure 2", "Procedure 3", "Procedure 4" } },
        { "Optional Scope", new[] { "Scope Option 1", "Scope Option 2" } },
        { "Policies", new[] { "Policy 1", "Policy 2", "Policy 3" } }
    };

    private List<GameObject> currentUIElements = new List<GameObject>();
    private Stack<System.Action> navigationStack = new Stack<System.Action>();

    // Dictionary to save checkbox states
    private Dictionary<string, bool> checkboxStates = new Dictionary<string, bool>();

    private void Start()
    {
        // Generate main menu buttons on start
        GenerateButtons(mainMenuOptions, OnMainMenuButtonClick);
    }

    private void GenerateButtons(string[] options, System.Action<string> onClickCallback)
    {
        ClearCurrentUI(); // Clears any previously generated buttons

        foreach (string option in options)
        {
            // Instantiate a new button from the prefab
            Button newButton = Instantiate(buttonPrefab, scrollContent.transform);
            TMPro.TextMeshProUGUI buttonText = newButton.GetComponentInChildren<TMPro.TextMeshProUGUI>();

            // Set the text of the button to the current option
            if (buttonText != null)
            {
                buttonText.text = option;  // This should correctly set the button text
            }
            else
            {
                Debug.LogError("Button prefab is missing a Text component! Please ensure the prefab has a Text component.");
            }

            // Add an onClick listener to the button
            newButton.onClick.AddListener(() => onClickCallback(option));

            // Keep track of the newly created UI element
            currentUIElements.Add(newButton.gameObject);
```

```
        }

        // Add a back button if there is a previous menu
        if (navigationStack.Count > 0)
        {
            AddBackButton();
        }
    }

    private void GenerateCheckboxes(string[] options)
    {
        ClearCurrentUI();
        foreach (var option in options)
        {
            Toggle newCheckbox = Instantiate(checkboxPrefab, scrollContent.transform);
            Text checkboxText = newCheckbox.GetComponentInChildren<Text>();
            if (checkboxText != null)
            {
                checkboxText.text = option;  // This should correctly set the checkbox text
            }
            else
            {
                Debug.LogError("Checkbox prefab is missing a Text component! Please ensure the prefab has a Text component.");
            }

            // Restore checkbox state if it was previously saved
            if (checkboxStates.ContainsKey(option))
            {
                newCheckbox.isOn = checkboxStates[option];
            }
            else
            {
                newCheckbox.isOn = false; // Default to unchecked
            }

            // Save checkbox state when its value changes
            newCheckbox.onValueChanged.AddListener((value) =>
            {
                checkboxStates[option] = value;
            });

            currentUIElements.Add(newCheckbox.gameObject);
        }

        // Add a back button if there is a previous menu
        if (navigationStack.Count > 0)
        {
            AddBackButton();
        }
    }

    private void AddBackButton()
    {
        if (backButtonPrefab != null)
        {
            Button backButton = Instantiate(backButtonPrefab, scrollContent.transform);
            Text backButtonText = backButton.GetComponentInChildren<Text>();
            if (backButtonText != null)
            {
                backButtonText.text = "Back";  // Ensure the Back button text is correctly set
            }
            backButton.onClick.AddListener(OnBackButtonClick);
            currentUIElements.Add(backButton.gameObject);
        }
        else
        {
            Debug.LogError("Back Button Prefab is missing. Please assign a back button prefab.");
```

```
        }
    }

    private void OnMainMenuButtonClick(string selectedOption)
    {
        if (subMenuOptions.ContainsKey(selectedOption))
        {
            // Save the current menu generator function to the navigation stack (main menu button click)
            navigationStack.Push(() => GenerateButtons(mainMenuOptions, OnMainMenuButtonClick));

            // Generate the submenu
            GenerateButtons(subMenuOptions[selectedOption], OnSubMenuButtonClick);
        }
        else
        {
            Debug.LogError($"Submenu for '{selectedOption}' not found!");
        }
    }

    private void OnSubMenuButtonClick(string selectedSubOption)
    {
        // Save the current submenu generator function to the stack
        string[] checkboxOptions = new[] { $"{selectedSubOption} Task 1", $"{selectedSubOption} Task 2", $"{selectedSubOption} Task 3" };
        navigationStack.Push(() => GenerateButtons(subMenuOptions[selectedSubOption], OnSubMenuButtonClick));

        // Generate checkboxes for the selected procedure or protocol
        GenerateCheckboxes(checkboxOptions);
    }

    private void OnBackButtonClick()
    {
        if (navigationStack.Count > 0)
        {
            // Get the last menu generator function and invoke it (this should take us to the previous menu)
            var previousMenu = navigationStack.Pop();
            previousMenu.Invoke();
        }
        else
        {
            Debug.LogError("No previous menu in the navigation stack!");
        }
    }


    private void ClearCurrentUI()
    {
        foreach (var element in currentUIElements)
        {
            Destroy(element);
        }
        currentUIElements.Clear();
    }
}
```

A.15 FetchAPILoop.cs

```
// While record button toggle is active, fetch data from the server every 10 seconds.

using UnityEngine;
using System.Collections;
using MixedReality.Toolkit.UX;

public class FetchAPILoop : MonoBehaviour
```

```
{
    public PressableButton recButton;
    private Coroutine fetchCoroutine;
    private bool isFetching = false;
    public SupabaseAPI supabaseAPI;

    void Start()
    {
        fetchCoroutine = StartCoroutine(PollServerLoop());
    }

    private IEnumerator PollServerLoop()
    {
        while (true)
        {
            yield return new WaitForSeconds(10f);
            FetchServer();
            Debug.Log("Fetching server data...");
        }
    }

    private void FetchServer()
    {
        supabaseAPI.GetUserInfo("");
        if (string.IsNullOrEmpty(ActivePatient.PatientID))
        {
            Debug.LogWarning("No patient selected.");
            return;
        }
        supabaseAPI.GetUserInfo(ActivePatient.PatientID);
    }
}
```

A.16 FlashDot.cs

```
using System.Collections;
using UnityEngine;

public class FlashingObject : MonoBehaviour
{
    public GameObject targetObject;
    public float flashInterval = 0.5f; // Adjust the speed of flashing
    private bool isFlashing = false;
    private Coroutine flashCoroutine;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.N) && !isFlashing)
        {
            isFlashing = true;
            flashCoroutine = StartCoroutine(FlashObject());
        }
        else if (Input.GetKeyDown(KeyCode.M) && isFlashing)
        {
            isFlashing = false;
            StopCoroutine(flashCoroutine);
            targetObject.SetActive(true);
        }
    }

    private IEnumerator FlashObject()
    {
        while (isFlashing)
        {
            targetObject.SetActive(!targetObject.activeSelf);
            yield return new WaitForSeconds(flashInterval);
        }
```

```
    }
}
```

A.17 IMixedRealityPointerHandler.cs

```
  internal   interface IMixedRealityPointerHandler
{
}
```

A.18 JsonRender.cs

```
// Display a single user's JSON data, while ignoring empty categories

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Newtonsoft.Json.Linq;
using System.IO;
using TMPro;

public class JsonRender : MonoBehaviour
{
    public TMP_Text[] displaySections;

    void Start()
    {
        // Start with empty text displays
        //displayText1.text = "";
        //displayText2.text = "";
    }

    public void DisplayPatientData(JObject jsonObject)
    {
        List<string> formattedLines = new List<string>();

        foreach (var property in jsonObject.Properties())
        {
            if (!string.IsNullOrEmpty(property.Value?.ToString()))
            {
                string fieldName = System.Text.RegularExpressions.Regex.Replace(
                    property.Name, "_", " ").ToLower();
                fieldName = System.Globalization.CultureInfo.CurrentCulture.TextInfo
                    .ToTitleCase(fieldName);

                formattedLines.Add($"{fieldName}: {property.Value}");
            }
        }

        int linesPerSection = formattedLines.Count > 0
            ? Mathf.CeilToInt((float)formattedLines.Count / 8f)
            : 1;

        for (int i = 0; i < displaySections.Length; i++)
        {
            if (i < 8 && i < displaySections.Length)
            {
                if (formattedLines.Count > 0)
                {
                    int start = i * linesPerSection;
                    int count = Mathf.Min(linesPerSection, formattedLines.Count - start);

                    if (start < formattedLines.Count)
                    {
                        displaySections[i].text = string.Join("\n", formattedLines.GetRange(start
    , count));
                    }
```

```
                    else
                    {
                        displaySections[i].text = ""; // Clear unused sections
                    }
                }
                else
                {
                    displaySections[i].text = "";
                }
            }
        }
    }

    /*
    void CheckForJsonUpdates()
    {
        if (string.IsNullOrEmpty(jsonFileName)) return;

        string jsonPath = Path.Combine(folderPath, jsonFileName);

        if (File.Exists(jsonPath))
        {
            string newJsonText = File.ReadAllText(jsonPath);
            if (newJsonText != lastJsonText)
            {
                lastJsonText = newJsonText;
                DisplayPatientData(newJsonText);
            }
        }
    }

    public void SetJsonFile(string newFileName)
    {
        jsonFileName = newFileName;
        LoadAndDisplayJsonData();
    }

    void Update()
    {
        timeSinceLastUpdate += Time.deltaTime;
        if (timeSinceLastUpdate >= updateInterval)
        {
            CheckForJsonUpdates();
            timeSinceLastUpdate = 0f;
        }
    }
    */
}
```

A.19 MenuManager.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;

public class MenuManager : MonoBehaviour
{
    public List<RawImage> menuImages = new List<RawImage>(); // Ensure the list is initialized
    public List<GameObject> selectorMenu = new List<GameObject>(); // Ensure the list is
    initialized

    void Start()
    {
        // Hide all images by default
        foreach (RawImage image in menuImages)
        {
            image.gameObject.SetActive(false);
```

```
        }

        // Hide all selectors by default
        foreach (GameObject selector in selectorMenu)
        {
            selector.SetActive(false);
        }
    }

    public void ShowImage(RawImage selectedImage)
    {
        foreach (RawImage image in menuImages)
        {
            image.gameObject.SetActive(image == selectedImage);
        }
    }

    public void ShowSelector(GameObject selectedSelector)
    {


        foreach (GameObject selector in selectorMenu)
        {
            selector.SetActive(selector == selectedSelector);
        }
    }
}
```

A.20 NewPatient.cs

```
// Enables user to start a new patient for recording

using UnityEngine;
using Newtonsoft.Json.Linq;

public class NewPatient : MonoBehaviour
{
    public JsonRender jsonRender;
    private void Start()
    {
        OnNewPatientButtonClicked();
    }

    // Called by CreateNewPatientButton
    public void OnNewPatientButtonClicked()
    {
        ActivePatient.PatientID = null;
        ActivePatient.PatientJSON = new JObject();
        // Clear the JSON data display on slate to signify a new patient
        jsonRender.DisplayPatientData(ActivePatient.PatientJSON);

        Debug.Log("Changed active patient.");
    }
}
```

A.21 PatientScroll.cs

```
using TMPro;
using UnityEngine;

public class PatientScroll : MonoBehaviour
{
    public GameObject page1;
    public GameObject page2;


```

```
    public void ShowPage1()
    {
        if (page1 != null)
        {
            page1.gameObject.SetActive(true);

        }
        if (page2 != null)
        {
            page2.gameObject.SetActive(false);
        }
    }

    // Function to activate text object 2, deactivate text object 1, and set font size
    public void ShowPage2()
    {
        if (page1 != null)
        {
            page1.gameObject.SetActive(false);
        }
        if (page2 != null)
        {
            page2.gameObject.SetActive(true);
        }
    }
}
```

A.22 PatientsRender.cs

```
// Render list of patients for PatientList section of slate. The list is split into two pages of
    5 buttons.
// Each button can be clicked to display the patient's data in the PatientData section of the
    slate using JsonRender.cs

using System.Collections.Generic;
using TMPro;
using Newtonsoft.Json.Linq;
using UnityEngine;
// using Microsoft.MixedReality.Toolkit;
using System.Linq;
using MixedReality.Toolkit.UX;

public class PatientsRender : MonoBehaviour
{
    public GameObject[] buttons;
    private JArray jsonArray;

    public JsonRender jsonRender;

    public void DisplayPatients(string jsonString)
    {
        if (string.IsNullOrEmpty(jsonString))
        {
            Debug.LogError("DisplayPatients: jsonString is null or empty.");
            return;
        }

        jsonArray = JArray.Parse(jsonString);

        if (buttons == null || buttons.Length == 0)
        {
            Debug.LogError("DisplayPatients: buttons array is null or empty.");
            return;
        }

        int loopCount = Mathf.Min(buttons.Length, jsonArray.Count); // Ensure we don't exceed the
```

```
     smaller size

        for (int i = 0; i < loopCount; i++)
        {
            if (buttons[i] == null)
            {
                Debug.LogError($"DisplayPatients: Button GameObject at index {i} is null.");
                continue;
            }

            var pressableButton = buttons[i].GetComponent<PressableButton>();
            if (pressableButton == null)
            {
                Debug.LogError($"DisplayPatients: MRTK PressableButton component not found on GameObject '{buttons[i].name}' at index {i}.");
                continue;
            }

            int capturedIndex = i;
            JObject patient = (JObject)jsonArray[capturedIndex];

            // Update button text with patient name
            JToken nameToken;
            if (patient.TryGetValue("PatientName", out nameToken))
            {
                string patientName = nameToken.ToString();

                TMP_Text tmpText = buttons[i].GetComponentInChildren<TMP_Text>();
                if (tmpText != null)
                {
                    tmpText.text = patientName;
                }
                else
                {
                    Debug.LogWarning($"DisplayPatients: No TMP_Text found in button '{buttons[i].name}' at index {i}.");
                }
            }
            else
            {
                Debug.LogWarning($"DisplayPatients: 'PatientName' not found in JSON at index {i}.");
            }

            // Set up click event
            pressableButton.OnClicked.RemoveAllListeners();
            pressableButton.OnClicked.AddListener(() => HandlePatientClick(patient));
        }

    }

    void HandlePatientClick(JObject patient)
    {
        string patientId = patient["PatientID"]?.ToString();
        if (!string.IsNullOrEmpty(patientId))
        {
            ActivePatient.PatientID = patientId;
            ActivePatient.PatientJSON = patient;
            Debug.Log($"Selected PatientID: {patientId}");
        }
        else
        {
            Debug.LogWarning("HandlePatientClick: PatientID not found in JSON object.");
        }
        jsonRender.DisplayPatientData(ActivePatient.PatientJSON);
    }
}
```

A.23 SlateFollower.cs

```
using UnityEngine;

public class SlateFollower : MonoBehaviour
{
    public float followSpeed = 3.0f;       // Speed of following user position
    public float distanceFromUser = 1.5f;  // Distance from the user
    public float rotationSpeed = 5.0f;     // Speed at which the slate rotates to face user

    private Transform userHead;
    private Vector3 offsetFromUser;        // Fixed offset from user position
    private bool isBeingMoved = false;
    private bool hasBeenMovedByUser = false;

    void Start()
    {
        userHead = Camera.main.transform;
        SetInitialPosition();
    }

    void Update()
    {
        if (!isBeingMoved && !hasBeenMovedByUser)
        {
            FollowUserPosition();
        }
        // Always face the user regardless of movement state
        FaceUser();
    }

    void SetInitialPosition()
    {
        if (userHead != null)
        {
            // Set initial position in front of user
            transform.position = userHead.position + Vector3.forward * distanceFromUser;
            // Store the offset from user position
            offsetFromUser = transform.position - userHead.position;
            // Initial rotation to face user
            FaceUser();
        }
    }

    void FollowUserPosition()
    {
        if (userHead != null)
        {
            // Update position based on user position only, maintaining the same offset
            Vector3 targetPosition = userHead.position + offsetFromUser;
            transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime
    * followSpeed);
        }
    }

    void FaceUser()
    {
        if (userHead != null)
        {
            // Calculate direction to user
            Vector3 directionToUser = userHead.position - transform.position;
            // Create rotation to face user
            Quaternion targetRotation = Quaternion.LookRotation(-directionToUser);
            // Smoothly rotate towards user
            transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.
    deltaTime * rotationSpeed);
        }
```

```
    }

    public void StartMoving()
    {
        isBeingMoved = true;
    }

    public void StopMoving()
    {
        isBeingMoved = false;
        hasBeenMovedByUser = true;
        // Update the offset when user manually places the slate
        offsetFromUser = transform.position - userHead.position;
    }

    public void ResetToGaze()
    {
        hasBeenMovedByUser = false;
        SetInitialPosition();
    }
}
```

A.24 SlateResetButton.cs

```
using UnityEngine;

public class SlateResetButton : MonoBehaviour
{
    private Vector3 defaultPosition;
    private Quaternion defaultRotation;
    private Vector3 defaultScale;
    private SlateFollower slateFollower;
    private Transform slateTransform;

    void Start()
    {
        // Find the top-level Slate object (not just the button's direct parent)
        slateTransform = transform.parent;

        if (slateTransform == null)
        {
            Debug.LogError("SlateResetButton: No Slate parent found! Make sure this button is a
    child of the Slate.");
            return;
        }

        // Store the exact initial position, rotation, and scale of the Slate
        defaultPosition = slateTransform.position;
        defaultRotation = slateTransform.rotation;
        defaultScale = slateTransform.localScale;

        // Get reference to the SlateFollower script
        slateFollower = slateTransform.GetComponent<SlateFollower>();
    }

    public void ResetSlate()
    {
        if (slateTransform != null)
        {
            // **Reset position, rotation, and scale to the original values**
            slateTransform.position = defaultPosition;
            slateTransform.rotation = defaultRotation;
            slateTransform.localScale = defaultScale;

            Debug.Log("Slate Reset: Position, Rotation, and Scale restored.");

            // **Reset SlateFollower if it exists**
```

```
            if (slateFollower != null)
            {
                slateFollower.ResetToGaze();
                Debug.Log("SlateFollower Reset: Gaze-following restored.");
            }
        }
        else
        {
            Debug.LogError("SlateResetButton: No Slate reference found!");
        }
    }
}
```

A.25 SlateVisibilityToggle.cs

```
using UnityEngine;

public class SlateVisibilityToggle : MonoBehaviour
{
    private bool isSlateHidden = false;

    public void ToggleSlateVisibility()
    {
        // Iterate through all child objects of the Slate except the button itself
        foreach (Transform child in transform.parent)
        {
            if (child != transform)
            {
                child.gameObject.SetActive(isSlateHidden);
            }
        }
        isSlateHidden = !isSlateHidden;
    }
}
```

A.26 SupabaseAPI.cs

```
// Used with PatientsRender.cs and JsonRender.cs to retrieve and display patient data

using UnityEngine;
using UnityEngine.Networking;
using System.Collections;
using Newtonsoft.Json.Linq;
using System.Text.Json.Nodes;
using Microsoft.CognitiveServices.Speech.Transcription;

public class SupabaseAPI : MonoBehaviour
{
    private const string SUPABASE_URL = "https://yuwrsuaqhbbfxqlrybgg.supabase.co/rest/v1/
    PatientData";
    private const string SUPABASE_KEY = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Inl1d3JzdWFxaGJiZnhxbHJ5YmdnIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTc0MDA3N
    .oDOmFPwxbq9FosgsJb4YPs3xwVTPdNL4ihNlw3oZwTk";
    public JsonRender jsonRender;
    public PatientsRender PatientsRender;
    JArray jsonArray;

    // Fills patientData with the data from the API on start
    private void Start()
    {
        GetUserInfo("");
    }

    public void GetUserInfo(string userId = "")
    {
        StartCoroutine(FetchUserInfo(userId));
```

```
    }

    IEnumerator FetchUserInfo(string userId)
    {
        string endpoint;
        bool isUserIdEmpty = string.IsNullOrEmpty(userId);

        // Set the endpoint based on whether looking for a single patient (for JSON update) or list of patients
        if (isUserIdEmpty)
        {
            endpoint = $"{SUPABASE_URL}?order=PatientID.desc&limit=10";
        }
        else
        {
            endpoint = $"{SUPABASE_URL}?PatientID=eq.{userId}";
        }

        UnityWebRequest request = UnityWebRequest.Get(endpoint);
        request.SetRequestHeader("apikey", SUPABASE_KEY);
        request.SetRequestHeader("Authorization", $"Bearer {SUPABASE_KEY}");
        request.SetRequestHeader("Content-Type", "application/json");

        yield return request.SendWebRequest();

        // Depending on type of request, call corresponding script to display that info accordingly
        if (request.result == UnityWebRequest.Result.Success)
        {
            Debug.Log($"Successful Fetching User Data: {request.downloadHandler.text}");
            if (isUserIdEmpty)
            {
                Debug.Log("Fetching all patients");
                PatientsRender.DisplayPatients(request.downloadHandler.text);
            }
            else
            {
                Debug.Log($"Fetching user with ID: {userId}");
                jsonArray = JArray.Parse(request.downloadHandler.text);
                JObject jsonObject = jsonArray[0] as JObject;
                if (jsonObject != null)
                {
                    jsonRender.DisplayPatientData(jsonObject);
                }
                else
                {
                    Debug.LogError("Failed to parse JSON object.");
                }
            }
        }
        else
        {
            Debug.LogError($"Error Fetching User Data: {request.error}");
        }
    }
}
```

A.27 TextResize.cs

```
using TMPro;
using UnityEngine;

public class TextResize : MonoBehaviour
{
    public TMP_Text textObject; // Reference to the TextMeshPro object
    public float sizeStep = 2f; // Step size for font change
    public float minSize = 10f;
```

```
    public float maxSize = 100f;

    // Function to increase text size
    public void IncreaseTextSize()
    {
        if (textObject != null && textObject.fontSize < maxSize)
        {
            textObject.fontSize += sizeStep;
        }
    }

    // Function to decrease text size
    public void DecreaseTextSize()
    {
        if (textObject != null && textObject.fontSize > minSize)
        {
            textObject.fontSize -= sizeStep;
        }
    }
}
```

A.28 TextToggle.cs

```
using UnityEngine;
using TMPro;

public class TextToggle : MonoBehaviour
{
    public TMP_Text[] displayTexts; // Assign all 8 text objects in the Inspector
    private int currentIndex = 0;
    private float defaultFontSize = 20f;

    void Start()
    {
        // Make sure only the first is visible at start
        SetActiveText(currentIndex);
    }

    // Activate text at currentIndex, deactivate all others
    private void SetActiveText(int index)
    {
        for (int i = 0; i < displayTexts.Length; i++)
        {
            if (displayTexts[i] != null)
            {
                displayTexts[i].gameObject.SetActive(i == index);
                if (i == index)
                {
                    displayTexts[i].fontSize = defaultFontSize;
                }
            }
        }
    }

    // Go to the next text display (looping)
    public void ShowNextText()
    {
        currentIndex = (currentIndex + 1) % displayTexts.Length;
        SetActiveText(currentIndex);
    }

    // Go to the previous text display (looping)
    public void ShowPreviousText()
    {
        currentIndex = (currentIndex - 1 + displayTexts.Length) % displayTexts.Length;
        SetActiveText(currentIndex);
    }
```

```
}
```

A.29 ToggleGameObjects.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ToggleGameObjects : MonoBehaviour
{
    public GameObject objectToHide;
    public GameObject objectToShow;

    public void ToggleObjects()  // Ensure it's public and has no parameters
    {
        if (objectToHide != null) objectToHide.SetActive(false);
        if (objectToShow != null) objectToShow.SetActive(true);
    }
}
```

Appendix B

Unity Structure

B.1 Checklist Menu

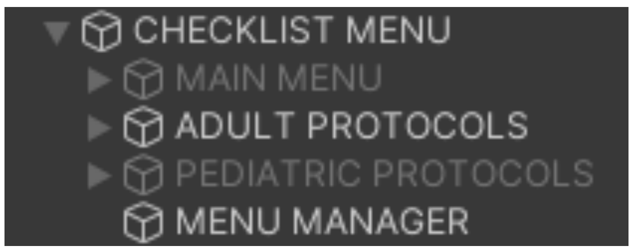


Figure B.1: Structure of the CHECKLIST MENU Game Object, with children menu objects and manager.

B.1.1 Main Menu

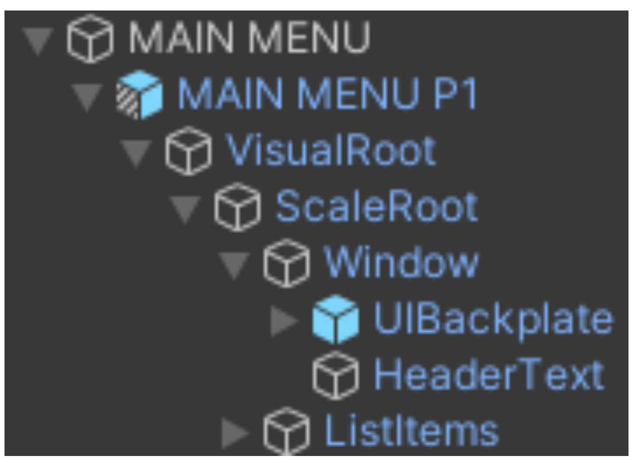


Figure B.2: Structure of the MAIN MENU Game Object, containing a list of buttons to navigate to other submenus. It also contains child objects defined by MRTK3's UI Components.

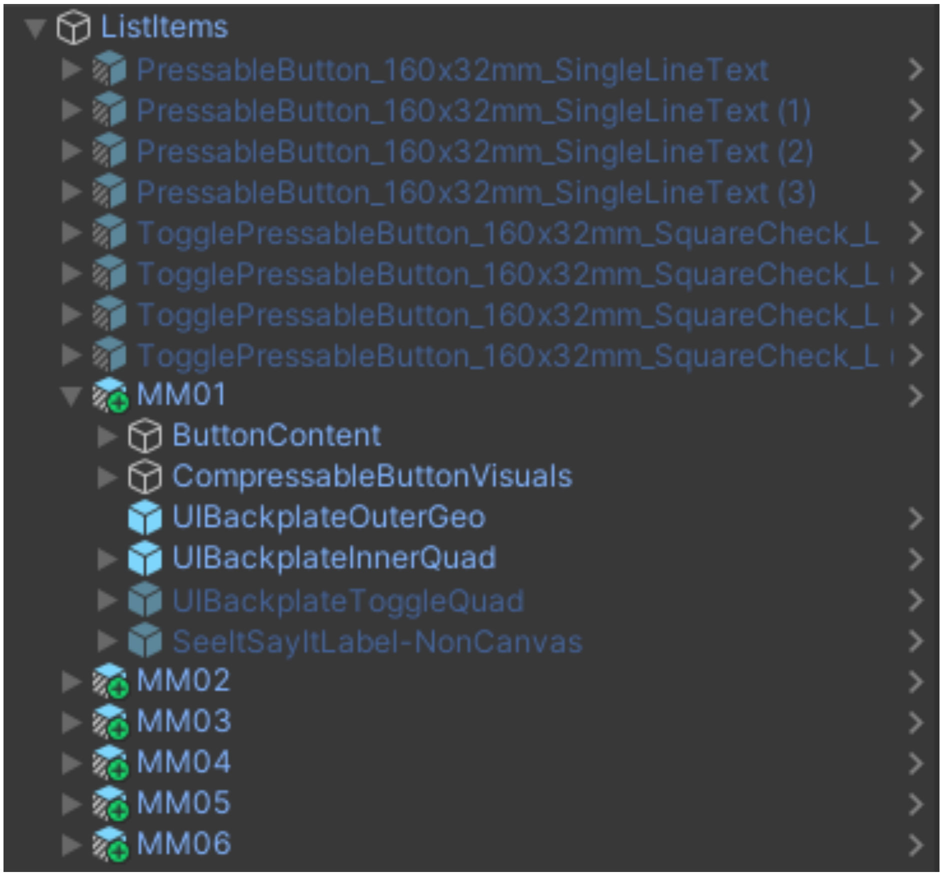


Figure B.3: An in-depth breakdown of the main menu's list of buttons, further showcasing the content which comprises a button Game Object.

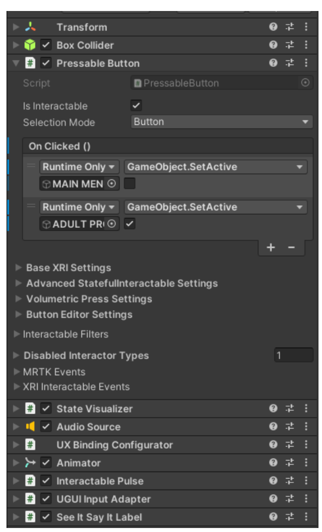


Figure B.4: Inspector view for a button Game Object, showcasing the several MRTK3 scripts utilized to provide intractability, as well as the functions which are called upon being activated.

B.1.2 Protocols

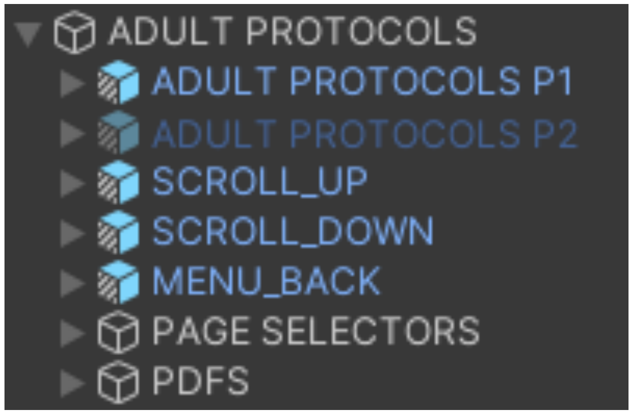


Figure B.5: Structure of a PROTOCOL Game Object, containing PDFs provided by the County, buttons, and pages.