Final “As-Built” Report

Team Shining Sky

Rosze Voronin, Skyler Hanson, Ashleea Holloway, Logan O’Donnell

May 5th, 2022 - Version 1.0

Sponsored by: Dr. Morgan Vigil-Hayes

Mentored by: Felicity Escarzaga

Table of Contents

Table of Contents 2

1. Introduction 3

2. Process Overview 5

3. Requirements 6

4. Architecture and Implementation 10

5. Testing 13

6. Timeline 15

7. Future Work 17

8. Conclusion 18

Appendix A: Development Environment & Toolchain 19

1. Introduction

Rural Native Americans struggle daily with mental health, with young people especially struggling with rising rates of depression, anxiety, and behavioral issues. This is due to Native American communities often not having funding for mental health resources, and a lack of network connectivity preventing members from accessing online help. Native American communities therefore may seek help from academic research labs and other sources of low-cost software development to attempt to address these problems.

The client, Dr. Morgan Vigil-Hayes, is the director of CANIS Lab (Community Aware Networks and Information Systems) and the founder of the ARORA project. Dr. Vigil-Hayes' research, funded by issued research grants, focuses largely on asynchronous networking and network infrastructure that does not rely on a constant internet connection. Her research aims to address issues caused by low network connectivity within Indigenous and other low-access communities. CANIS lab's ultimate goal is to take the products developed for tribal communities such as the Navajo Nation and the Hopi tribe and genericize them to be offered to other North American Indigenous communities.

The Hopi Tribe is currently creating a Community Mentorship program that will connect community mentors with Hopi youth mentees that they would like to tie into the ARORA project, but the program still needs a software tool to aid in communication and organization that the existing ARORA app does not provide. The developers of team Shining Sky worked with Dr. Vigil-Hayes to create the start of the ARORA Community Mentor Portal App that provides the software support that is missing for the Hopi mentorship program.

The goal of this initial stage of the project is to produce a strong starting point for future development, which Dr. Vigil-Hayes has already displayed as part of her application for a 2.5 million dollar research grant that she expects to see funding from around July 2022 when negotiations with the grant-giver wrap up. The Community Mentor Portal App currently consists of a version of the project that is entirely self-contained, providing features like viewing of mentee mood reports and information, appointment scheduling, a chat interface, and a separate view for supervisors. CANIS Lab's future development will focus on integrating it with the rest of the ARORA project, and writing network code that takes advantage of CANIS Lab's specialty in limited online connectivity networks, reflecting the rural area the app is targeted use in.

1.1 Solution Vision

The Community Mentor Portal App is a separate sister application to the ARORA mental wellness app that addresses the missing features needed for Hopi's Community Mentorship program. The Hopi Community Mentorship program does not have a technological or physical resource that connects mentors with young Hopi people (now mentees within the program) to address their mental health struggles. CANIS Lab's ARORA mental wellness app was only made for mentees to use to track their own mental health, so the mood reports and stress levels reported within the app are not currently viewable to potential mentors within the mentorship program. Therefore, the ARORA project is currently missing ways for mentors to:

- View mood report data of mentees from the ARORA app.
- Track mentees showing signs of struggling with mental health.
- Communicate with mentees that show signs of struggling with mental health.
- Answer questions mentees may have related to mental health.

The solution to these missing features is the Community Mentor Portal App. The Community Mentor Portal App provides a clean UI that organizes mentee data and allows mentors to directly communicate with mentees that may be struggling with their mental health. The mentorship app includes the beginning of implementation for key features of the final product:

- A login system to ensure that only authorized mentors and supervisors of the mentorship program can access the application.
- Mood report and stress level mentee data from the ARORA app organized in list format filterable by the mentor.
- A question list system that displays anonymous questions sent in from mentees or other Hopi youth for the mentor to respond to.
- A chat system for mentors to directly communicate with mentees.
- A calendar system for mentors to track their appointments.

2. Process Overview

In the course of developing the initial version of the Community Mentor Portal App, Shining Sky followed a fairly consistent development cycle. Each week, the team met to go over each member's tasks from the last week and briefly discuss what was accomplished solutions to any problems that arose, and tasks that each member would complete in the next week. This was followed the next day by a meeting with our mentor, Felicity, where we'd discuss upcoming assignments, ask questions that we had, check our understanding to make sure we were progressing on our deliverables correctly, and receive any other guidance we might need. Later in the week, we'd have a meeting with Dr. Vigil-Hayes, where we'd show off new features and provide other updates on development, go over what was coming next, plan out anything that required her involvement or assistance, and discuss her feedback as well as any changes to plans or scope.

As development progressed, we all began to find our specialized roles within the project based on our skills. Rosze kept her initial role of team lead and client liaison, continuing to manage client meetings and expectations, as well as taking a major creating role on most non-code deliverables. Ashleea settled into the role of head developer, managing the code and producing the newest features and bug fixes. Skyler worked on style and aesthetics code, as well as helping out on non-code deliverables. Logan focused on testing our software, as well as helping out on non-code deliverables.

The two tools that had the most impact on our development were Discord, a chat client that allowed us to keep up constant communication throughout the week and not have to call a meeting for every little discussion or clarification, and GitHub, which allowed us to ensure that all code ended up in one place and could be safely merged together into a master copy of the project. From there, most team members used whichever equivalent tools they found best suited to their task-completion process, selecting their own IDEs and other relevant items. The only other shared standout is the Google office suite; Drive, Slides, Docs, and Sheets were all used during development to provide easy access to documents that all or multiple team members would need to edit.

3. Requirements

There were many requirements that needed to be implemented into the final product. These requirements are specific functions the app must provide the user, a baseline efficiency of these functions, and other details that allow the project to exist with other apps. The client and our team created these requirements through several meetings and discussions about what was desired and what was feasible, with refinements during our weekly meetings as needed. We eventually settled on the scope of the project and specific functions that needed to be implemented. The requirements include several categories focused on functionality:

- Registration
- Login
- Profile
- Anonymous Questions
- Home
- Chat
- Calendar

Requirements that aren't explicit functions but relate to the usability of those functions are considered performance requirements. An efficient and enjoyable user experience is necessary. Therefore the latency of different actions was recorded and reduced where possible. Furthermore, the usability of the app became quantifiable through testing. Aspects of usability include startup times, page loading times, and the intuitivity and efficiency of interactions.

Environmental requirements are designed to guarantee that the product works in conjunction with a larger system. The client needs the project to work with a separate preexisting application that is used by the mentees each mentor is assigned. This will be accomplished by using compatible software and database systems. Making the apps appear as one cohesive product by utilizing similar themes and assets will allow for visual consistency. Lastly, the product needs to be available to users with Android devices.

3.1 Functional Requirements

Registration - Setting up mentor and supervisor accounts.

- **Identification Code** - ID Code provided to mentor from supervisor.
- **Name** - Name to be associated with the account
- **Username** - Used alongside password to log in to account.
- **Recovery Email** - Email to tie to account for recovery.
- **Password** - Unique password for secure entry. Includes section form for password confirmation.

Login - Accessing accounts.

- **Username** - Created during registration and referenced for proper credentials.
- **Password** - Used to securely log in to the account.

Profile - Provides users information and minor functionality.

- **Access Code Generation** - Codes are generated to be used when registering new accounts.
- **Logout** - Returns the user to the login page

Anonymous Questions - Asked by mentees for mentors and supervisors to answer.

- **View Unanswered Questions** - Ability to view questions that have yet to be answered.
- **Answer Questions** - Ability to send answers to questions.

Home Page - Allows navigation to mentees, or mentors if a supervisor.

- **Mentees** - Opens to the selected mentee with their mood reports and buttons to jump to messages or add a flag
- **Color Codes** - Color-coded butterfly for quick reference to summarized mood report values
- **Flags** - Can be used to highlight a mentee for any reason

Chat System - Communicating with mentees.

- **Identifying Information** - Includes name of the recipient for quick selection.

- **Sorted by Most Recent** - Conversations are shown from most recent to oldest in terms of the last message sent.
- **Search Conversations** - Search for a conversation by name of the recipient.
- **Create Conversations** - Ability to make a new conversation with a mentee.
- **Send Messages** - Ability to send messages in a conversation.

Calendar System - For users to keep track of upcoming events.

- **Adding Events** - Adding events into the calendar.
- **Editing Events** - Events are deleted and the forms are repopulated with the previous information.
- **Deleting Events** - Events can be entirely deleted and removed from the calendar.

3.2 Performance Requirements

Beyond meeting the functional requirements, there are certain non-functional requirements that will increase the performance and ease of use of the Community Mentor Portal App. Dr. Vigil-Hayes has not made any formal requests for these requirements, so they have largely been determined by researching common app development practices.

Latency

- **< 1 Second with minimal data** - When actions are being performed that include small amounts of data, the latency is less than one second.
- **< 1 Minute with data** - When actions are being performed that include more than small amounts of data, the latency is less than one minute.

Usability

- **< 10 Second Startup Times** - The app will open to the login page in under ten seconds.
- **< 1 Second Page Load Times** - Each page within the app will be loaded in under one second.
- **Intuitivity** - The layout of features will be continuously modified until all testers correctly identify them at a rate greater than 50%.

3.3 Environmental Requirements

The app is intended to work in conjunction with a preexisting application, the ARORA Mentee App, to strengthen an upcoming mentorship program. To guarantee that these programs appear as one cohesive program, they will be visually similar and work together. The app will be usable on Android devices.

Visual Consistency

- **Color Scheme** - The color scheme will be similar to ARORA Mentee App.
- **Theme** - The theme will be consistent with ARORA Mentee App.

Operating Systems

- **Android** - Ability to use the app on an Android run device.

4. Architecture and Implementation

The ARORA app is made up of a front end and a back end, which work in conjunction to provide users with a variety of useful features. The front-end system, pictured in Figure 4.1, is made up of the various screens and modules with which the user will interact while in the app. The first screen which will appear to a user is the login screen, where they will be able to create an account or log into a preexisting one. After a successful login, they will be taken to the home screen, which will show a list of mentees or mentors which the user is responsible for, depending on their position within the mentorship program. The user can now navigate to any of the other screens using the main navigation module, as shown in Figure 4.1. These include a feed of anonymous questions, a chat screen for communicating with other members, the pages of individual mentees or mentors (the latter of which displays the list of mentees associated with that user), a profile containing information about the currently signed-in user, and a calendar screen which can be used to record upcoming appointments and events.

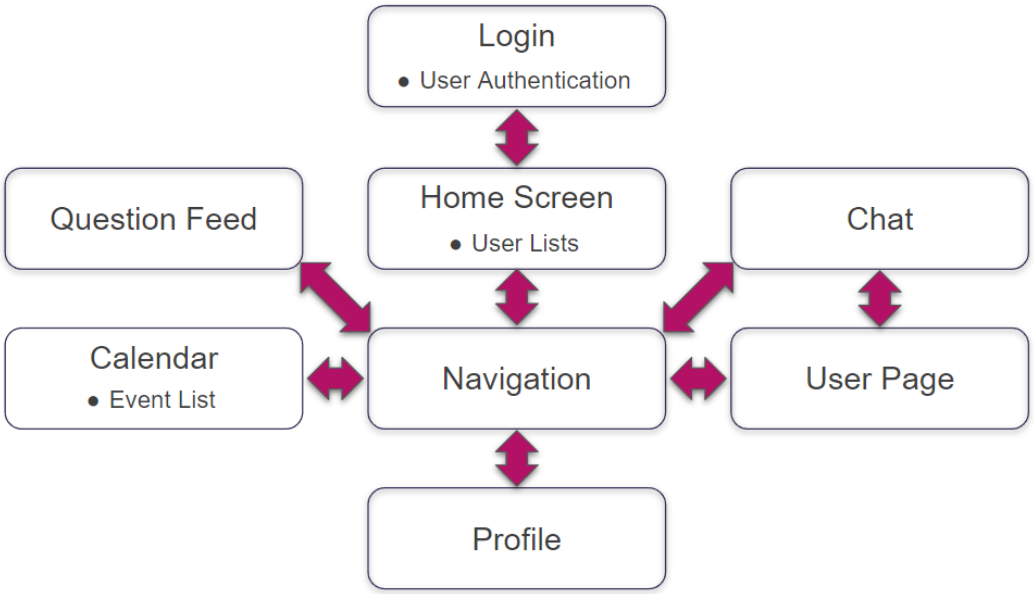

Figure 4.1: Front-End Architectural Diagram

The back-end, pictured in Figure 4.2, is responsible for managing the data which will be loaded into and collected from each of these screens. Each module has specific data with which it is associated. For example, the calendar screen pulls from a list of events, each of which is

made up of fields for date, time, and description. Mentee pages must correctly display that mentee's name, representations of their recent mood reports, their current risk level, and whether or not they have been flagged as requiring additional attention.

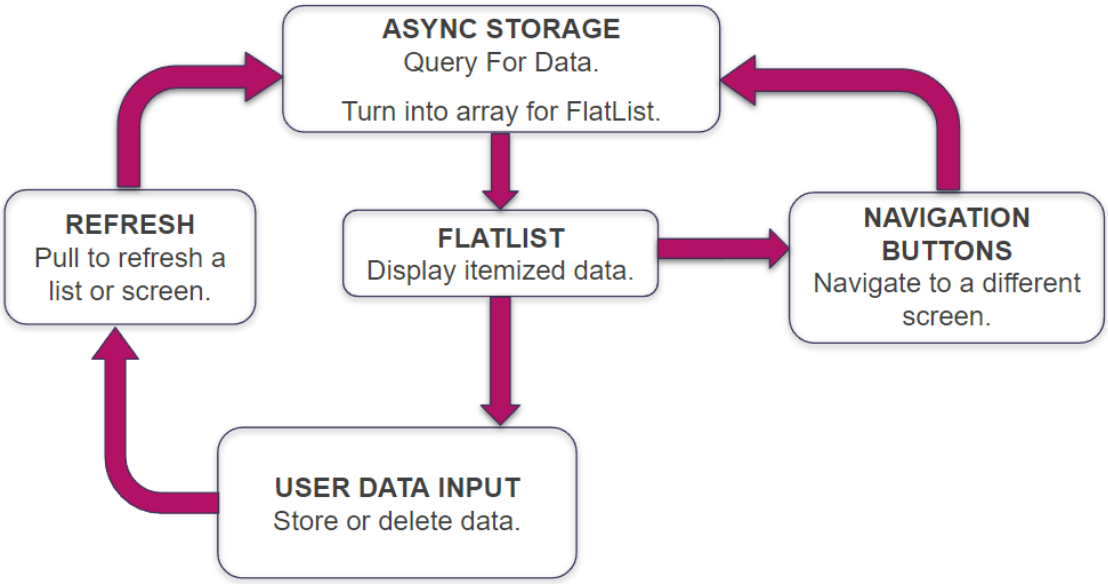

Figure 4.2: Back-End Architectural Diagram

This data is managed using the package AsyncStorage. When a particular module is loaded, its data is loaded into a flatlist which is then displayed to the user. If the user makes changes to data, either by adding an entry, editing an existing entry, or deleting an entry, the screen must be refreshed in order to show the updated list. This causes the data to again be pulled from storage and displayed. The user is also able to navigate to a different screen, at which point a query is made for the data required.

Our project had some changes in scope over the course of development. Initially, we expected to write a lot more network code to integrate the app with the rest of the ARORA project. However, in conversing with Dr. Vigil-Hayes, she decided that she wanted to focus on those things later in development, and asked us to make a self-contained 1.0 version of the project that would support future development. This cut out a lot of our plans for back-end functionality, and we ended up focusing on front-end features and usability concerns. Our database was also a challenge during development; we went through two failed iterations before eventually settling on the AsyncStorage package as the best method for data storage

within the project's constraints. A lot of our architectural changes consisted of cutting things and tightening our scope, so when looking back a development model where we started with a tight scope and expanded as we better learned what we could implement may have been a better model than including everything we thought would make for a cool feature or useful functionality.

5. Testing

To further improve the product and guarantee its functionality, we conducted as much testing as possible. The testing we did was broken down into three stages: unit testing, integration testing, and user testing. Unit testing is a formal way of ensuring that individual components of the project work correctly and as intended. Once we could prove that each component worked, we moved on to integration testing. This testing checks that each module works together properly. For the most part, we completed our unit and integration tests by hand, manually ensuring that each component worked and worked in concert. We added console outputs to verify that things were functioning properly on the backend, as well. Since we had a relatively limited scope and small codebase, this process worked alright and allowed us to root out all but one bug. This is definitely something we would have liked to spend more time on if possible when looking back on development, but we focused primarily on user testing because Dr. Vigil-Hayes assured us that having a good starting point was more important and that future development would likely introduce new bugs into the codebase anyway.

Having completed both of these testing processes, we moved on to the final stage, user testing. Having the product in the hands of different users allowed us to gain additional perspective on our project and discover any usability and style issues that we could fix. We broke our user testing down into three components, each with specific steps. We began with sample users, or users who represent possible end-users. We then moved to the expert review stage where we had Dr. Vigil-Hayes interact with the app to get her feedback.

The product is meant to be used alongside the previously mentioned mentorship program which means that prospective users have a large age range. We kept this in consideration when making changes to our design, trying to keep in mind that, while our average age was younger users, there would definitely be a few on the older end who would benefit from increased app clarity. This is important because technological literacy tends to vary greatly and accessibility is always a concern. Our user testing process consisted of handing our sample users the app and asking them to perform open-ended tasks, such as “send John Smith a chat message”. We then observed them performing the task, and asked them to comment on their thought process when figuring out how to do so, as well as provide feedback on what they did and did not like about the UI and user experience.

Throughout the user testing process, we made several changes to the app. Our main focus was on usability; a lot of users felt that our pressable items were too close together and that text could be spaced out more appealingly. To address these concerns, we made several CSS edits, including adding margins and increasing the text size. We also spread our pressable features out and tried to make it more clear what portions of the visible app represented a single pressable object. We also added some splashes of color to make the app more visually interesting.

The final plan for the project was to hand it off to CANIS Labs, therefore we wanted to make sure that at least Dr. Vigil-Hayes was able to test the app herself and make any suggestions she might have. To do this, we helped her set up our toolchain on her machine in advance. We did discover a troublesome issue through this process, that being that the app was a bit finicky when data was not filled in in advance. To address this, we added a function that creates the admin tool if nothing else is present, and then a button within the admin tool to fill in our sample dataset so that starting from zero and creating all the data by hand was an option and not a requirement.

Once we had the previous groups test the app and made modifications, we wanted to take the app to the Hopi community to get more feedback. We began setting up a remote meeting with Dr. Vigil-Hayes and her connections in the community, but due to time constraints and logistical issues on all sides, we, unfortunately, had to cut this portion of our plan for time.

6. Timeline

Throughout this project, we kept track of where we were and where we should be by using a Gantt chart constructed on Google Sheets. The chart pictured in Figure 6.1 is what we continuously updated as we progressed. Being in the first week of May, there is no ‘Now’ line on the chart. Every block is green because we have completed all of the necessary components and finished our testing and refinement. Color coding helped us quickly understand what was still in progress and what needed to be worked on next.

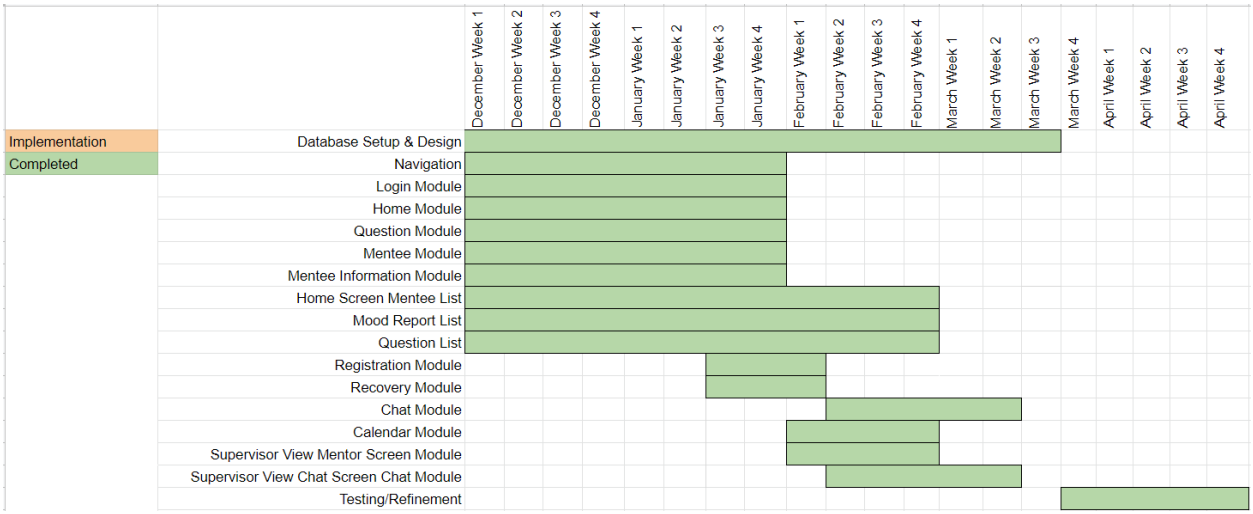

Figure 6.1: Completed Gantt Schedule

We began the semester by working on all of the initial necessities in order for the project to function. This included the database, navigation, and several modules. These modules were the core components of the project. The login module, for example, handles everything needed for user authentication and moves the app to the home screen if the credentials entered are correct. For these modules to function correctly we also needed lists to hold the information, which we worked on at the same time.

Once the skeleton of the project was completed, we dedicated the next several weeks to adding the differences between users. These included the mentor view, supervisor view, and admin view. The mentors only need to view their own mentees and not the mentees of other mentors. Supervisors though should be able to view every mentor and every mentee assigned

to them. An administrator is solely meant to create supervisor accounts. Having the basics of the project done made this process more straightforward.

After we completed every part, we moved on to our testing and refinement processes. Testing consisted of working with several sample users to gather as much info as we could on the usability of the project. We planned to include testing with CANIS Lab and with some real-world users, but sadly those two things were scrapped due to time and logistics constraints. Our refinement process consisted of adding little features to make the app more functional, including fleshing out our admin tool to fill out test data. We also worked on changes based on feedback from testing and mainly improved our CSS and usability using those results.

For the first portion of the semester, we typically split up the roles and responsibilities between everyone. Two people would work on the database while the other two worked on the other modules. Documents would also be split among the team depending on each person's strengths. Very quickly, this strategy became no longer feasible due to the volume of work required. From then on, members focused on portions of the project most suited to their skills. Although this meant that some members received less of a learning experience in certain areas, we were able to get everything done on time and complete the handoff on schedule.

7. Future Work

Now that we have concluded our handoff process, the Community Mentor Portal App is in the hands of CANIS Lab. Dr. Vigil-Hayes was able to use the prototype of the Community Mentor Portal app to apply for a $2.5 million grant. She is currently in talks with the grant-giving entity and expects that she will be receiving funding around July 2022 once the process wraps up. After obtaining funding, she plans to continue the development of the Community Mentor Portal app through CANIS Lab. This future development will focus on low-connectivity network code to allow the app to function smoothly in areas with poor internet access like the Hopi reservation. This type of network code is one of CANIS' specialties as a research lab, so being able to focus on that will allow them to use grant money more efficiently. In addition, CANIS will integrate the Community Mentor Portal app with the rest of the ARORA project, allowing users of the existing youth mental wellness app and users from the mentorship program to interact more effectively.

Once the ARORA project has been expanded to suit the needs of the Hopi reservation, Dr. Vigil-Hayes plans to offer a genericized version of it to other Indigenous communities in North America. Through this, she hopes to address the mental healthcare gap faced by North American indigenous communities and move towards a future in which everyone has the access to mental healthcare that they need.

Dr. Vigil-Hayes has also discussed further work on the app with Rosze and Ashleea in the summer of 2022, as they are interested in contributing a little bit more in order to ensure that the project is able to get a strong start and continue successfully. Currently, Dr. Vigil-Hayes has brought Ashleea on as a developer and is expecting to use Rosze in a minor consulting role. While this is still in its initial stages, the three have met briefly to discuss timings and estimations of what work will be done, and are keeping in touch for when things calm down after the conclusion of the academic year.

8. Conclusion

The ARORA Community Mentor Portal app serves as an important milestone in the continued success of the ARORA project. It provides the features and functionalities needed to support the Hopi peer mentorship program and integrate it with the ARORA project, furthering the goal of providing better access to mental healthcare for young people on the Hopi reservation. Once CANIS Lab completes the second stage of development, the app will be integrated with the rest of the ARORA project, allowing both the peer mentorship program and the ARORA youth mental wellness app to function more effectively in tandem.

Team Shining Sky is glad to have been part of a project that will have a direct positive impact on people's lives, and we are looking forward to checking back in on the project in a few years to see the great place it will surely have reached. We are happy that our project allowed Dr. Vigil-Hayes to see success in obtaining grant funding, and sure that what we implemented will allow that grant money to be used more efficiently by laying a solid starting ground for future work.

While development was not always a smooth process, we do feel that our Capstone class experience has been as "real-world" as it gets for not being a spot at a software development firm. We dealt with clashes in expectations, crunch, changing scope, and all kinds of bugs and setbacks. To go with the lows, we also had plenty of highs – hearing the good news from Dr. Vigil-Hayes about her grant application, a really smooth final Capstone presentation, and of course, the great feeling of coming out alive on the other side of the project. Looking back, we see some improvements to be made in terms of our process and our project, but recognizing those points is an important part of the learning process. In the end, we worked hard, learned a lot about our skills both personal and team-oriented and produced something we are proud of.

Appendix A: Development Environment & Toolchain

This appendix covers the important setup steps in order to achieve an equivalent development environment to the one Shining Sky was using. The below sections describe an appropriate hardware setup, an explanation of the necessary parts of the toolchain, instructions to set up that toolchain, and finally a look into the production cycle.

A.1 Hardware

Shining Sky developed the app on Windows 10, using Powershell as our terminal of choice. Our development machines ranged from simple, office-grade computers to high-end gaming PCs. We did not find that we had any issues with our environment on even our worst pieces of hardware, so we would not say there are any minimum specs for our toolchain besides a reasonable office-quality PC that can be expected to be able to be used as a development machine. From testing with Dr. Vigil-Hayes, we were able to get everything set up on macOS using the default terminal, and due to similarities between the two platforms and explicit support for all of our tools, we expect that Linux setup would go the same way.

We performed testing of the actual app itself mainly on Android phones, all running Android 9. As far as we understand, the app in its current state should be light enough to run on most semi-modern to modern Android phones with no issue. The app is currently optimized for Android, so it has some issues when running on iOS and as a web app due to the three platforms having different system calls and CSS styling systems.

A.2 Toolchain

This subsection goes over the toolchain setup process, starting with prerequisite tools and moving into installation steps. Following along with this subsection will result in a development environment that is ready to run the Community Mentor Portal App when paired with a developer’s chosen IDE and access to a copy or repository of the project code. These instructions should only need to be performed once, discounting infrequent repetition when updating the items within the toolchain.

In order to set up the environment for the ARORA app, a developer must have the following prerequisite tools installed:

- **Node JS** - https://nodejs.org/en/
    - The version of Node JS used herein is the latest at the time: 16.15.0 x64. We always install the LTS version of Node JS, and we expect that using the latest version will work in the future as well.
    - Proceed through the install wizard or install via the binaries for your chosen operating system. We used the Windows Installer (.msi) with all of the default settings.
- **Expo CLI** - https://docs.expo.dev/workflow/expo-cli/
    - With Node JS installed, Expo CLI can be installed using the terminal command **npm install -g expo-cli**
    - At the time of writing, Expo CLI 5.4.3 is the current version.
- **Expo Go** - https://expo.dev/client
    - Running the Community Mentor Portal App on Android requires the Expo Go app which can be downloaded from the Google Play Store.
    - The only other thing to note regarding Expo Go is that the Android device it is installed on will need to be on the same network as the PC from which Expo CLI is being run.

Developers should have access to the GitHub repository from which the Community Mentor Portal App can be downloaded. Download the files and place them wherever you would like to run the app from, or run a git pull if using an IDE that supports that functionality. From there, run the command **npm install** from within the ARORA_App_v2 folder in order to automatically install the packages that the code depends on. Running **npm audit fix** should repair all but one vulnerability, an issue with node-fetch that cannot be fixed until the package developers fix it and that cannot be removed as expo itself relies on it, but thankfully has no potential impact on this project. A complete list of packages in use can be found within the app's source code, in the file **package.json** located in the root of the project file (ARORAMentorshipDemo / ARORA_App_v2 / package.json when navigating to the file from the GitHub repository as it is currently organized).

A.3 Setup

This subsection describes the process for running the project on a test phone. Code in Expo is updated dynamically, so having a test version running and refreshing it during development is very useful. This setup requires the toolchain described in the previous section and continues with the assumption that those steps have been completed. The instructions in this section represent the initial, pre-coding startup required to begin editing code and therefore are steps that may need to be performed at the beginning of a day of development before changing any code. This subsection assumes a Powershell terminal on a Windows 10 machine and a test phone running Android 9 but should be usable on equivalent systems.

1. **cd** into the directory where the app's code is. This should be, assuming a download of the code in the state it was handed off in, the **ARORA_App_v2** folder within the root directory.
2. If using Powershell, running the command **powershell.exe -executionpolicy bypass** is required. This starts Powershell with the execution policy preventing the running of arbitrary scripts bypassed. This should not be a dangerous step unless you purposefully run a malicious script (not anything from our project) from that Powershell terminal.

3. Run the **expo start** command to build and set up the code for execution. Your terminal window should show the following:

4. From there, simply scan the QR code from the Expo Go app on an Android phone connected to the same network as the PC running the terminal window, and the app will start up on that phone. Other development tools can also be used from the terminal window, per the options presented on the screen and visible in the above screenshot.

A.4 Production Cycle

After following the above two subsections, a developer should be ready to write code. While development processes can vary by team and organization, this subsection describes what Shining Sky used, and serves as a suggested process for a situation where there is not one yet in place. It can be used as-is, refined to suit the new team, or replaced entirely at the discretion of future developers.

We recommend editing the code locally using the steps outlined in the above two subsections. Following those instructions, a developer should be able to make any changes

desired to their local version and then refresh the app on the test phone to see them reflected and ensure that they work properly.

Following this, we recommend rerunning the steps in A.3, or, already having the correct command line setup, to just **CTRL+C** to shut down the current run and **expo start** to rebuild and rerun it. Shutting down Expo Go completely and restarting it is also recommended for this step. This allows developers to see a freshly compiled and built version of what they have been working on, and ensure everything is looking right before pushing code.

Finally, push the code to the shared repository on GitHub. We used branches to organize our individual pushes, and then worked together to merge them during our weekly meetings, but this process can be altered depending on the frequency of pushes in the new development environment.