

TITLE

By:

Mounir Aït Hamou -

Laurier Gascon-Miller -

Mohamed Yatera - 300280976

Alvin-Thomas Tran -

Xavier Colin -

Work submitted as part of the SEG 2105A course

University of Ottawa

December 4th, 2023

Table of content (will be added at the end)

This report presents the final implementation of the Healthcare Appointment Management System (HAMS), a mobile application developed with the following technology stack: Java as the programming language, Firebase for backend services like real-time databases and authentication, Android Studio as the integrated development environment and GitHub to facilitates the seamless management of software development workflows between the developers. The application was made to streamline the process of healthcare appointment scheduling and management for a telehealth clinic.

The following sections will give a detailed description of the application functionalities from the point of view of the different users (Patient, Doctor and Administrator). This includes the sign-up process, the list of appointments, shift scheduling and a lot more. The objective of this report is to give a comprehensive understanding of the functionalities and capabilities of the HAMS application.

[illegible]

Contribution

Deliverable	Mounir	Laurier	Mohamed	Alvin	Xavier
1	20%	20%	20%	20%	20%
2	20%	20%	20%	20%	20%
3	20%	20%	20%	20%	20%
4	20%	20%	20%	20%	20%

Screenshots of the app

- Login page

3:24

Email Adresse
e.g someone@example.com

Password
password

☐ Remember me [Forgot Password](#)

Login

Sign up

- Signup page

2:35

3G

Back

Patient or Doctor

☒ Patient

☐ Doctor

Name

First Name

Last Name

Contact Information

Phone Number

Email

Address

Additional Information

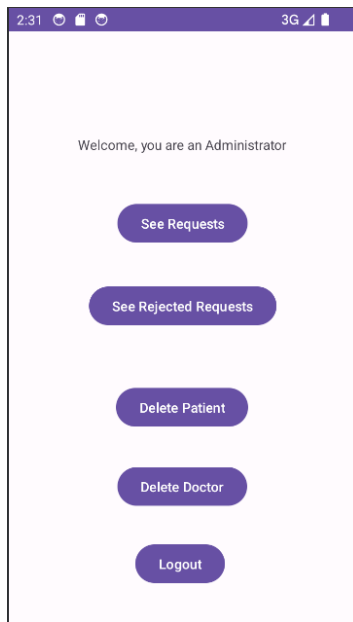
Health Card Number

Password

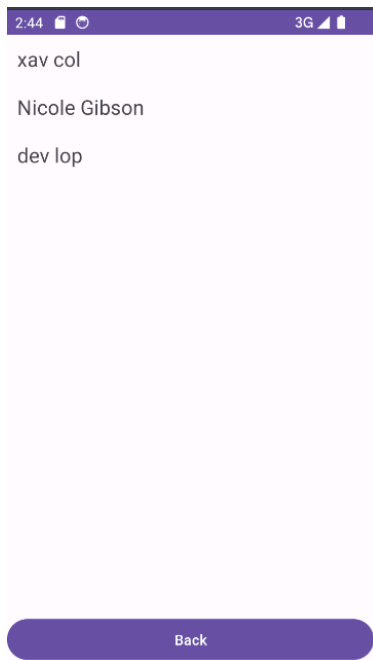
Password

Submit registration request

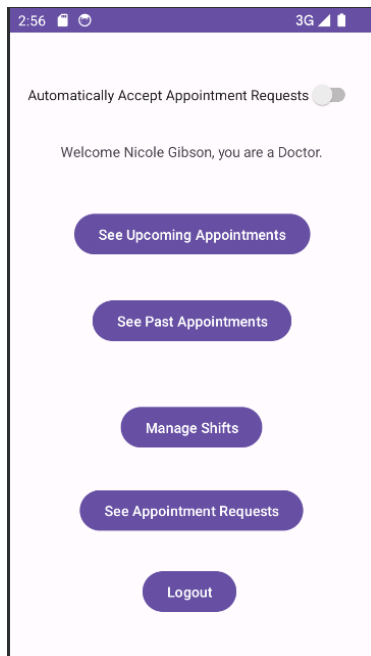
- Admin page



- Admin request page



- Doctors page



2:56 3G

Automatically Accept Appointment Requests ☐

Welcome Nicole Gibson, you are a Doctor.

See Upcoming Appointments

See Past Appointments

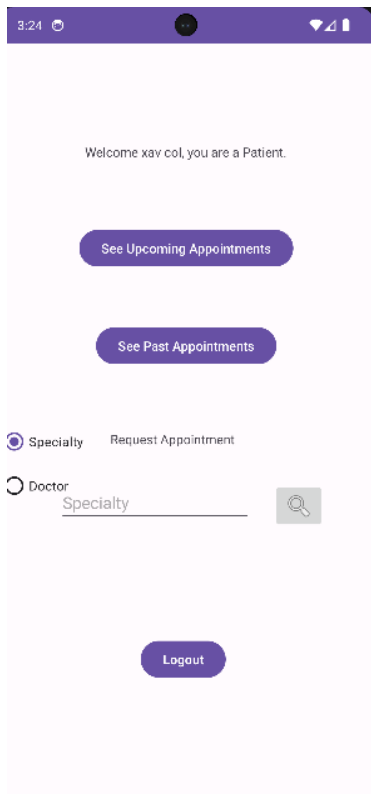
Manage Shifts

See Appointment Requests

Logout

This is a mobile app mockup for a doctor's dashboard. It features a purple header bar with the time 2:56 and 3G signal. Below the header, there is a toggle switch for 'Automatically Accept Appointment Requests'. A welcome message reads 'Welcome Nicole Gibson, you are a Doctor.' Below this, there are five purple buttons with white text: 'See Upcoming Appointments', 'See Past Appointments', 'Manage Shifts', 'See Appointment Requests', and 'Logout'.

- User page



3:24

Welcome xav col, you are a Patient.

See Upcoming Appointments

See Past Appointments

☒ Specialty Request Appointment

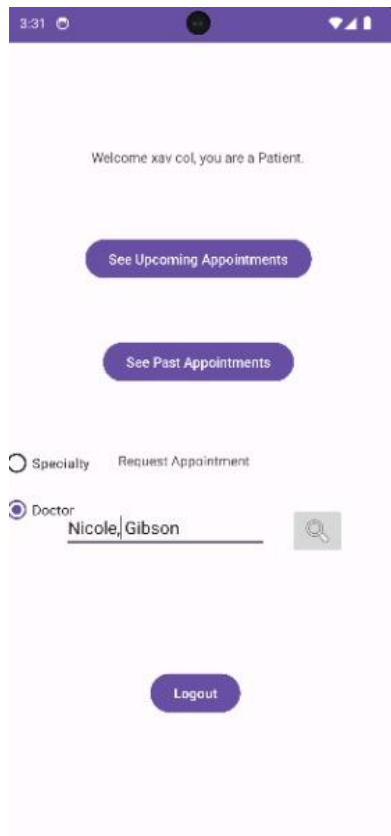
☐ Doctor

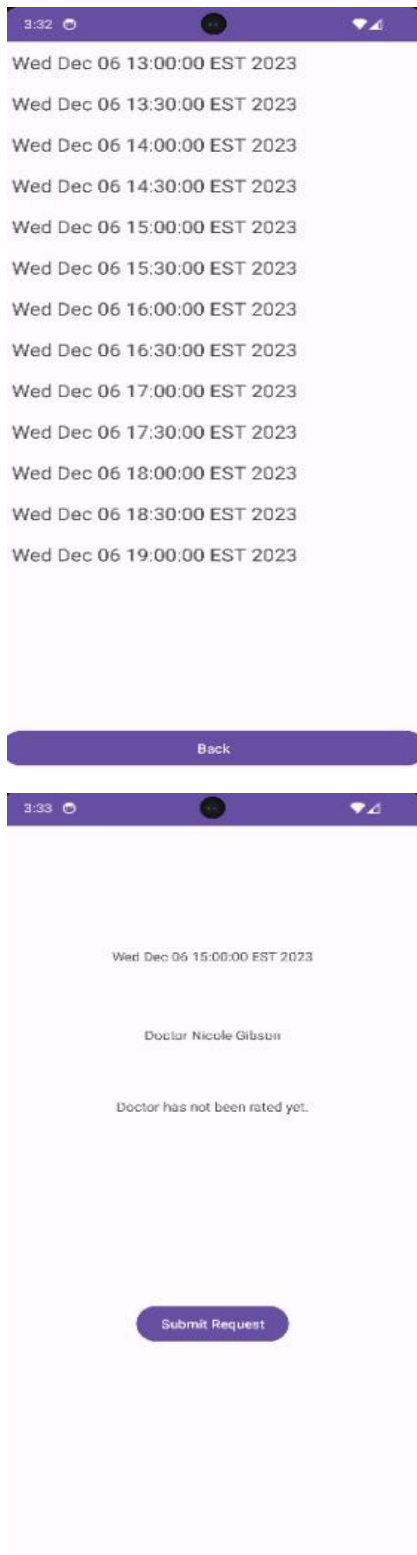
Specialty

Logout

This is a mobile app mockup for a patient's dashboard. It features a purple header bar with the time 3:24. Below the header, there is a welcome message: 'Welcome xav col, you are a Patient.' Below this, there are two purple buttons with white text: 'See Upcoming Appointments' and 'See Past Appointments'. Below these buttons, there are two radio buttons. The first is labeled 'Specialty' and is checked, with the text 'Request Appointment' to its right. The second is labeled 'Doctor'. Below the radio buttons, there is a text input field with the placeholder text 'Specialty' and a magnifying glass icon to its right. At the bottom, there is a purple button with white text: 'Logout'.

- User side of appointments





- **Doctor side of appointments**

Automatically Accept Appointment Requests ☐

Welcome Nicole Gibson, you are a Doctor.

See Upcoming Appointments

See Past Appointments

Manage Shifts

See Appointment Requests

Logout

Wed Dec 06 15:00:00 EST 2023

Back

ust@gmail.com

rav

col

34526

712street

135135

Ned Dec 06 15:00:00 EST 2023



-Shifts



7:37

Wed Dec 06 18:30:00 EST 2023 to Wed Dec 06 22:30:00 EST 2023

Mon Dec 04 21:30:00 EST 2023 to Mon Dec 04 22:30:00 EST 2023

Shifts

Start Date

Select Start Date

Start Time

Select Start Time

End Date

Select End Date

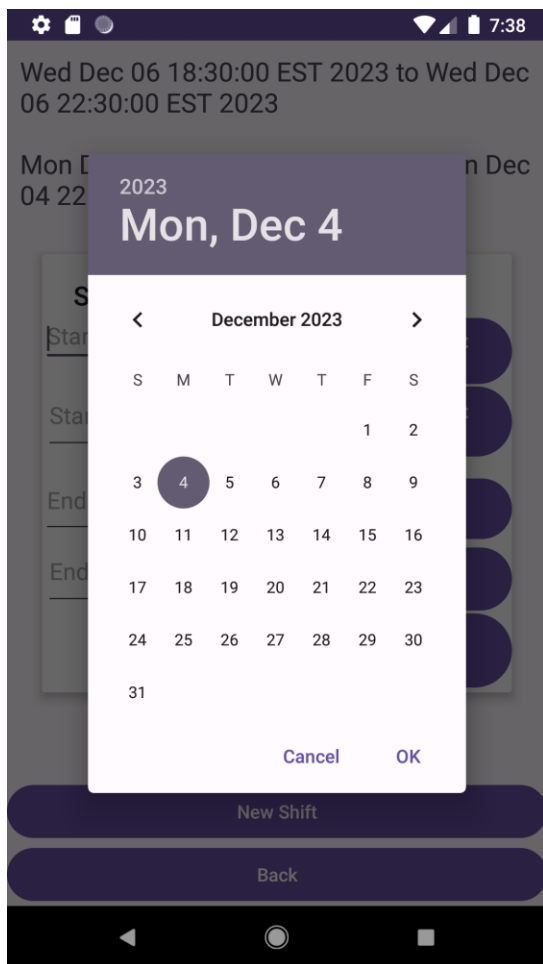
End Time

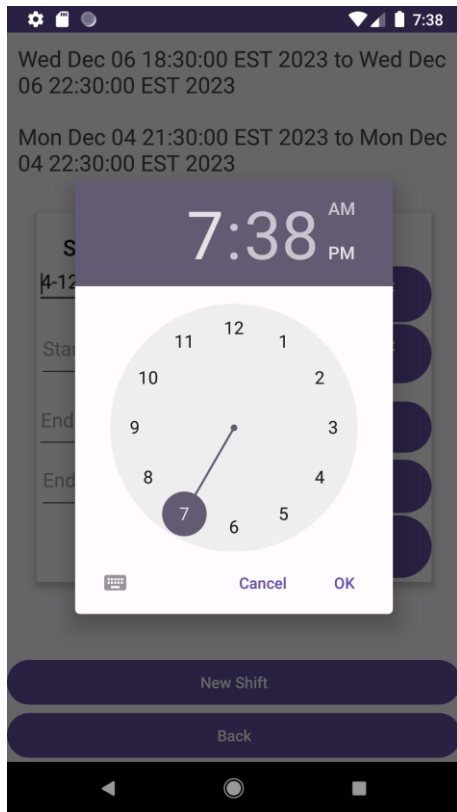
Select End Time

Confirm

New Shift

Back





Lessons learned

In this section we will go through the development journey of the Health Appointment Management System application (HAMS). We will discuss the challenges we faced, the solutions to those challenges and the knowledge we've gained throughout the whole process. This serves not only as a record of our growth, but also as a guide to avoid these mistakes in future projects and allow us to replicate the solutions to the problems we've faced.

During the initial stages of the project, we were met with some difficulties with the way we wanted to implement the Firebase database as this was a new technology for everyone on the team. After learning the different methods (services) that Firebase allows us to use, we, on one hand needed an efficient way to add data to the real time database. On the other hand, we need an equally efficient way to extract that data from the database to be able to use it throughout the application. This is where we used the single responsibility principle without knowing the formal term for it. The principal basically states that every class or module in a program should only provide one specific functionality. This principle was used for most of the classes we created to help maintain flexibility for future changes and is something we will continue to use as its not only an efficient way to design a program, but it is also used withing many large companies.

Getting back on track, we created a “DatabaseHelper.java” class that would contain all Firebase-related methods. In other words, every other class that would need an instance of the firebase would call the methods from this class, avoiding code duplication across multiple classes. The initial development stages of the firebase also showcased the importance of organization within a programming project. We mutually decided on the names of the nodes we would use for the Firebase JSON tree during the writing process. This led to us coming up with conventions for naming instance variables, naming class names, etc ... This taught us firsthand how coding conventions are crucial when working on a programming project because they allow for consistency; and make code easier to read and understand as though it appears to be written by a single developer, maintainability; easier to spot errors and understand code structure and efficiency; reducing the time spent on reading and understanding the code.

Speaking of time, the development of this project highlighted the importance of future-proofing. This refers to the process of designing and implementing the code in a way that anticipates potential future changes. By the same token, this resulted in thinking of scalability while creating the app. An example of this would be during the second deliverable, when we needed to find a way to list all the users to the admin and had to make a choice between a list view or a recycler view. The list view would have been easier to implement compared to the recycler view, but we decided to go with the recycler view as it had features that would be far more beneficial to our project. During the creation of this project, our goal was to make it as realistic as possible, and in a real-world setting, the recycler view just made more sense. Let us say that if we were to have 10,000 users, the list view would have loaded all the user's information, even if they were outside the limitation of the screen. For all phones, this would have been a very demanding task. In comparison, the recycler view only loads the number of users that the screen can show so that any time a user comes off the screen (because of scrolling), its object and information will be deleted, and it will be loaded back when it needs to come on the screen. As stated, the implementation was harder because we needed to create a class called MyRecyclerViewAdapter that would bind the data to the TextView in each row, store and recycle views as they are scrolled off the screen and the list goes on. To finish this point, as we took the much harder route, we invertedly improved our understanding of the intricacies of the world of software engineering.

Another key point, previously mentioned, is optimization. Having to work on the project with different devices and keeping in mind that in the real world, this would also be the case, led us to optimize our code as much as possible. Consequently, this led to the app running on average much faster, because fewer computational resources (CPU time or memory) were used. The optimization process was closely connected with algorithmic efficiency. An efficient algorithm can handle larger data sets and perform complex tasks more quickly; however, it takes longer to develop. Under those circumstances, we were able to delve into the world of debugging. Debugging is the process of finding and resolving bugs within the software. The goal is to identify and correct an error's root cause. We learned various ways of debugging such as using the logcat or toast as a form of error handling. These two tools were especially helpful post-completion of an algorithm

to know what types of error occurred (how it affected the rest of the code) and the location of the specific group or single line of code responsible for the error.

Additionally, another lesson learned and used in the world of software engineering is the use of the Scrum Agile environment. In a Scrum environment work is delivered in short cycles, which are called sprints. This is essentially the project management framework that we decided to use because it promoted learning through experiences and was highly compatible with the type of work we were doing. We analyzed all the requirements and concepts needed for the completion of the project before starting anything. The backlog in this case was given by the requirements per deliverable given to use beforehand, the duration of the sprints was also predetermined by the deliverable due dates, the sprint review and daily (weekly in our case) sprint date were chosen through discussion and the sprint retrospective was done at every deliverable release to learn from those experiences. This highlighted the importance of choosing the right model before working on a project. The chosen model should be the one that aligns the most with the project in question so that it can allow for maximum efficiency when collaborating (speed and organization). The use of collaboration in the scrum environment taught us the importance of team dynamics. Team dynamics refers to the way team members interact, communicate, and cooperate. The quality of these interactions can greatly influence the team's productivity, performance, and overall success. The quality of said interactions was better when the team members got to know each other outside of the scope of the project, whether that be joking with each other or simply talking about things other than the task at hand. This enabled interactions with each other during the sprint review and daily sprints to flow better. Parallely, we learned to leverage one's strengths and weaknesses to promote collaboration and workflow. Learning the team dynamics allowed us to learn about each other's strengths and weaknesses. During the sprint reviews and daily sprints, we were able to delegate tasks based on these factors. For example, in our team dynamics, some are better at doing database related tasks while others are better at creating the xml's; logically it would make sense for these team members to be delegated the task they are better at. This is important to establish before delegating tasks so that every team member will be able to excel at their tasks. If everyone has similar strengths and weaknesses, the team may struggle in certain areas. Diverse skills complement each other and lead to a more effective team.

Going back to the organizational aspect of the lessons we've learned; we were finally able to make use of our UML diagram knowledge learnt in class. Of course, knowing something is good for comprehension, but actively using that knowledge further solidifies it. Unified modeling language (UML) provides a visual representation of a system's architecture design and relationship between its components. In this case it was a UML class diagram of our domain model. The UML diagram was generally done before starting a new deliverable to be able to know what must be done, and visually see the connected aspects of the code such as variables, interfaces, subclasses, multiplicity, security practices, etc ... Yes, it would be possible to go straight to coding, but the UML diagrams proved very useful to not only have the final product (per deliverable) seen visually, but also to see how our app has change every deliverable.

A point often overlooked, is the ability to be able to scower the internet or and the resources at your disposal to be able to solve a problem. This creation of the app inevitably created cases where we would have to go look for solutions or learn concepts that were not previously known to us. This led to us having specific websites for specific types of problems. For example, problems related to the coding aspect were generally solved through debugging or finding inspiration from ideas found on stack overflow or any other coding forums. On the other hand, when it came to using android studios for building the xml files, YouTube videos proved more effective because it provided a step-by-step rundown. These instances served as examples of problem-resource relationships. We identified multiple relationships of a similar nature during the development of the HAM's app. Furthermore, we were also able to learn new things indirectly (someone used solution x for problem y, but we were expecting solution z) such as making a class serializable.

In conclusion, the development journey of the Health Appointment Management System application (HAMS) proved to be a valuable experience because it provided us valuable insights and lessons that extend beyond mere project completion. We've had to overcome challenges in many different technologies and concepts such as implementing Firebase and adhering to principles like the single responsibility principle, opting for future proofing coding to allow for better performance and scalability in the real world, learning efficient debugging methods, adherence to common and our own coding conventions, and the adoption of Scrum Agile for project optimization and collaboration. Furthermore, we learnt the true value of using UML diagrams to provide a visual roadmap for projects of any domain.