

**SEG2105**

**Term Project Report**

***MedLink:*** **Walk-in Clinic Services App**

Amani Chikh (8855250)

Daoud Hamadneh (8252115)

Katerina Rinchon (8615643)

Danny Vinh (8598966)

**Professor: Miguel Garzón**

**TA: Wassim El Ahmar**

December 3rd, 2019

Table of Contents

[1. Introduction 3](#_Toc26305110)

[2. Final UML 3](#_Toc26305111)

[3. Work Distribution 3](#_Toc26305112)

[4. App Screenshots 7](#_Toc26305113)

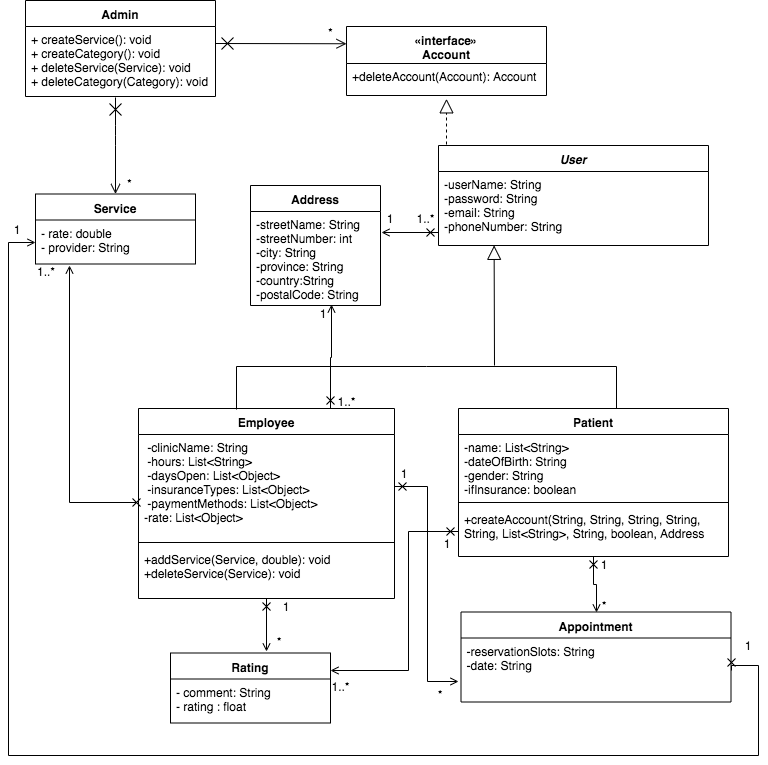
[5. Lessons Learned 17](#_Toc26305114)

# Introduction

MedLink is a walk-in clinic services application for Android devices. It strives to centralize the interactions between patients who are looking to book appointments, and the walk-in clinics that provide the services. The target demographic of MedLink is individuals without a family doctor who frequent many different walk-in clinics at the same time (since clinics often do not provide the same services). MedLink displays a comprehensive list of upcoming appointments at all the different clinics that they have booked to the user. This application is extremely necessary due to the lack of booking system of walk-in clinics. It allows more accessibility and organization for patients and clinics–respectively.

Future steps including uploading the application through Play Store. However, one of the biggest limitations of this application is Firebase. Since user credentials are stored on Firebase, it is important to prevent the public from accessing it. MedLink team decided to start with a free version of Firebase since MedLink was under development. However, the team is planning to pay for privacy. It is also important to note that even if unethical hackers could have access to MedLink database, the passwords credentials are hashed, and therefore, would not greatly affect our user privacy.

# Final UML



# Work Distribution

KEY:

|  |  |  |  |
| --- | --- | --- | --- |
| AC = Amani Chikh | DH = Daoud Hamadneh | KR = Katerina Rinchon | DV = Danny Vinh |

**Deliverable 1**

|  |  |
| --- | --- |
| **Task** | **Completed By** |
| Repository creation | DH |
| Commit at least once | AC, DH, KR, DV |
| UML class diagram creation | DV, DH, KR, AC |
| APK building | KR |
| Firebase integration | DV |
| Design on Figma | AC, KR |
| Figma to Android Studio Layout:   1. activity\_loggedin 2. activity\_main 3. activity\_iam 4. activity\_registration\_\_employee 5. activity\_welcome\_page | KR  KR  KR  KR  KR |
| Java class files:   1. MainActivity 2. IAmActivity 3. Registration\_Employee 4. Registration\_Patient 5. WelcomePage | KR, DV  KR, DV  KR, DV  KR, DV  DV, KR |
| Java models:   1. Account 2. Address 3. Admin 4. Appointment 5. Employee 6. Patient 7. User | DH  DH  DH  DH  DH  DH  DH |
| Field validation:   1. MainActivity 2. Registration\_Employee 3. Registration\_Patient | KR  KR, DH  KR, DH |

**Deliverable 2**

|  |  |
| --- | --- |
| **Task** | **Completed By** |
| Update UML class diagram | DV, DH, KR, AC |
| 5 unit test cases:   1. Gradle file modification + 1 test case 2. 2 test cases 3. 2 test cases | KR  AC  DH |
| APK building | KR |
| Integration with CircleCI | DV |
| Design on Figma | AC |
| Figma to Android Studio Layout:   1. activity\_delete\_dialogue 2. activity\_employee\_account 3. activity\_manage\_accounts 4. activity\_manage\_services 5. activity\_welcome\_page\_admin | DV, DH  KR  AC, KR, DH  KR  KR |
| Java class files   1. ClinicDeleteDialogue 2. DeleteDialogue 3. EmployeeAccountActivity 4. ManageAccountsActivity 5. ManageServicesActivity 6. WelcomePageAdmin | DV, KR, DH  DV, DH  DV, KR  AC, KR, DV, DH  DV, KR  KR, DV |
| Java models:   1. Admin 2. Category 3. CategoryList 4. Employee 5. Service | DH  DV, DH  DV, DH  DH  DV, DH |
| Field validation:   1. ManageAccountsActivity 2. ManageServicesActivity | DH  DH |

**Deliverable 3**

|  |  |
| --- | --- |
| **Task** | **Completed By** |
| Update UML class diagram | AC |
| APK building | KR |
| 2 unit test cases | AC |
| Design on Figma | AC |
| Figma to Android Studio Layout:   1. activity\_category\_layout 2. activity\_clinic\_delete\_dialogue 3. activity\_clinic\_view\_patient 4. activity\_delete\_dialogue 5. activity\_edit\_hours\_dialogue 6. activity\_employee\_account 7. activity\_manage\_services\_clinic 8. activity\_registration\_\_employee2 9. activity\_registration\_\_employee3 10. activity\_set\_rate\_dialogue 11. activity\_welcome\_page | DV, DH  KR, DV, DH  KR  DV, DH  KR  KR  KR  KR  KR  KR, DV, DH  KR |
| Java class files:   1. EditHoursDialogue 2. ClinicDeleteDialogue 3. EmployeeAccountActivity    1. View Working Hours    2. View Insurance + Payment Methods 4. RegistrationEmployee2 5. RegistrationEmployee3 6. SetRateDialogue 7. ManageServicesClinic | KR, DV, DH  DV, DH  /  DV, KR  DH, KR  KR, DV, DH  KR, DV, DH  DV, KR  DV, KR |
| Java models:   1. Account 2. Address 3. Admin 4. Category 5. CategoryList 6. Employee 7. Patient 8. Service 9. User | DH  DH  DH  DH, DV  DH, DV  DH, DV, KR  DH  DH, DV  DH |
| Field validation:   1. RegistrationEmployee2 2. RegistrationEmployee3 | KR, DH  KR, DH |

**Deliverable 4**

|  |  |
| --- | --- |
| **Task** | **Completed By** |
| Update UML class diagram |  |
| APK building | KR |
| Design on Figma | AC |
| 10 unit test cases | AC |
| CircleCI finalizing | KR, DV |
| Figma to Android Studio Layout:   1. activity\_book\_appointment 2. activity\_choose\_service\_for\_appointment 3. activity\_clinic\_view\_patient 4. activity\_patient\_account 5. activity\_search\_for\_clinics | DV  DV  KR  KR  KR |
| Java classes:   1. BookAppointmentActivity 2. ChooseServiceForAppointmentActivity 3. ClinicViewPatient    1. Check-In    2. Book Appointment    3. Working Hours    4. Comment    5. Rating 4. PatientAccountActivity 5. SearchForClinicsActivity    1. Search by Name    2. Search by Address    3. Search by Days    4. Search by Hours    5. Search by Services | DV  DV  /  DV  DV  DV  KR  KR, DV  KR, DV, DH  DH  DH  DH, AC  KR, DV  DV |
| Java models:   1. Appointment 2. Employee 3. ClinicList 4. Rating | DH  DH, DV, KR  DV  KR |
| Final report:   1. Title page 2. Short introduction 3. UML screenshot 4. Table of contributions 5. App screenshots    1. Screenshots    2. Function 6. Lessons learned | DH  KR, DH  AC  KR  /  DV  KR  DV, DH, AC |

# App Screenshots

|  |  |
| --- | --- |
| **Function** | **Screenshot** |
| Sign in page for all users (admin, patient, clinic/employee). Error appears if the user credentials are incorrect upon clicking Sign In.  Sign Up will allow user to create an account. |  |
| Users can choose between signing up as an employee/clinic or a patient. Depending on their choice, they will be led to a different “Create Account” page. |  |
| Shown: Employee/Clinic Sign Up  Scrollable layout allows the user to input many different types of information into the appropriate fields.  Validation has been done to force the user to input correct information.  Upon clicking of the next button, a new Employee object is created. |  |
| User that has decided to create an employee/clinic account will be prompted to set their clinic information.  Validation has been done to force the user to input start and end hours, choose at least 1 day, 1 insurance type, and 1 payment method. |  |
| User that has decided to create an employee/clinic account will be prompted to choose their services and assign a corresponding rate to this service that is exclusive to their clinic.  Finally, user is able to sign up. |  |
| Shown: Patient Sign Up  Scrollable layout allows the user to input many different types of information into the appropriate fields.  Validation has been done to force the user to input correct information.  Upon clicking of the Sign Up button, a new Patient object is created. |  |
| Shown: Admin Welcome Page  Upon successful sign-in admin is able to either manage accounts, manage services or sign out. |  |
| Admin can manage accounts through searching for an inputted username and then deleting it.  Upon successful deletion, a Toast is shown to confirm. |  |
| Admin can manage services by creating new services and assigning a role (doctor, staff, nurse) to the service. This is retained across all clinics. Admin cannot assign a rate to the service that it creates.  Admin may also delete and edit services. |  |
| Shown: Clinic Welcome Page  Upon successful sign-in clinic is able to view their account preferences and sign out. |  |
| Shown: Once Account Button Clicked  Clinics are able to view and edit their working hours, view and edit their services, and view the different insurance and payment methods that they accept.  While editing the working hours, the user is asked to input new hours and choose at least one new day. |  |
| Shown: Patient Welcome Page  Upon successful sign-in patient is able to view their account preferences and sign out. |  |
| Shown: Once Account Button Clicked  Patients are able to view their upcoming appointments and also find a clinic. |  |
| Shown: Once Find A Clinic Button Clicked  Patients are able to search by service, address, name, hours or day. The relevant clinics will show up and are able to be booked. |  |
| Shown: Once Clinic Long Clicked  Patients are able to view the wait time of the clinic, the clinic’s working hours, the clinic’s rating, and they’re also able to submit a rating and a comment.  Additionally, the patient is able to check in and book an appointment. |  |
| Shown: On Book an Appointment Button  Patients are able to choose the service that they seek and book an appointment for it once they have chosen a service. |  |

# Lessons Learned

Throughout the development of this project, there have been many tasks and difficulties that gave rise to very important lessons being learned – not only in terms of software, but also in group dynamic.

A critical principle learnt was the importance and usefulness of documentation. Many times, during the implementation of functionality in our application, we stumbled with run-time errors. A significant amount of these did not have an elaborate description of the error, and we were stuck heading to Google and Stack Overflow for a quick and simple answer. While sometimes effective, many of the problems we encountered, specifically with Firebase, was not a simple fix we could find on an Internet forum. There have been countless times where we would blindly apply a solution found on the internet to our code, in hopes it would rid the bugs, but only to make it worse. However, once we had resorted to the documentation, the problem would be fixed within 5 minutes. For example, there was a time where we kept on encountering a run-time error that did not have an elaborate identifier but was clearly due to some aspect of Firebase Database. After countless hours of ineffective Googling and debugging, the documentation was finally read, and the error was due to a simple mistake – Firebase Database is not completely compatible with custom Classes. An error like this could have been easily avoided if the documentation was briefly read beforehand, and as a result in the future we will not be intimidated by the “documentation”, since understanding the workings behind a package is more effective in debugging than it is to blindly surf Internet forums.

Additionally, another essential lesson was the importance of committing on GitHub. It is important for the commit to substantially discuss all the changes so that other developers would not have difficulties understanding the changes. It is also important to add comments and TODOs for while adding methods and activities since it helps other developers to understand the flow of the code better. Not only does it help members understand what each chunk of code does, it helps the developer who wrote it understand/expand/fix their code once they’re out of that flow state they were in when initially writing the code. Furthermore, importing libraries and using already built-in frameworks is another feature that enhance reusability of code.

SEG2105 being a course on software architecture, it would not be a well completed project if it did not teach us the importance of structural planning. It was crucial to work on the software architecture (UML) before starting with prototyping and coding. A UML class diagram is not a final unchangeable entity and updates are required to adapt to unforeseen issues or limitations. Constantly updating it to meet the requirements for each deliverable enabled us to be more efficient with our time. For instance, we first planned on using categories and services as one would with folders and files. The implementation of that feature turned out to be time consuming, we ended up discarding it. Having a clear overview of our project with the UML diagram permitted us to quickly adapt and change activities that were first implemented with the concept of categories.