# Project: Seraphine - The Smart E-health Booking System

**Java - Objected Oriented Programming**

Loc Bui Nhien (1370292)

Vinh Truong Canh Thanh (1370069)

Linh Ngo Phuc (1370616)

Tri Nguyen Minh (1370360)

An Truong Nguyen Thien (1370030)

Course: Java - Objected Oriented Programming
Lecturer: Luigi La Blunda
Due date: 17th February, 2022

Loc Bui Nhien
Team 19
Class: Java OOP
Frankfurt: University of Applied Sciences

Prof. Luigi La Blunda
Frankfurt University of Applied Sciences
Nibelungenplatz 1, 60318 Frankfurt am Main

Dear Prof. La Blunda,

Please find below our report on eHealth, which is part of the subject Java Object-Oriented Programming.

This report gives you an insight into the inner workings of the project from a software design perspective while providing an easy-to-understand diagram and further improvements for the future.

Thank you for taking the time to read this report and I hope it meets with your approval. If you have any further questions, please feel free to contact me by phone at 0049 1522 3832811 or by email: loc.buinhien@stud.fra-uas.de.

Sincerely yours,

[illegible]

Loc Bui Nhien
Team Leader

## 1 Disclaimer

We declare that this report is a product of our own work, unless otherwise referenced. We also declare that all opinions, results, conclusions and recommendations are our own and may not represent the policies or opinions of Frankfurt University of Applied Sciences.

Loc Bui Nhien
1370292
Team leader

[illegible]

Vinh Truong Canh Thanh
1370069

[illegible]

Linh Ngo Phuc
1370616

[illegible]

Tri Nguyen Minh
1370360

[illegible]

An Truong Nguyen Thien
1370030

[illegible]

## 2 Abstract

The eHealth system is a web application built using Java for the backend with Spring Boot to provide a graphical user interface with account registration and login for users to browse a list of available physicians in a user-selected area and symptom category, then make and manage appointments.

The system also includes an integrated database provided by spring boot for data management, an administration interface that allows an authorized administrator to modify or delete user accounts and appointments, and a reminder feature that notifies users of upcoming physician appointments via email.

This project is part of the Computer Science curriculum, evaluated and guided by Mr.Luigi La Blunda, and was carried out so that we could gain initial experience in Java programming and the implementation of a database and email dispatcher functionality.

**Keywords:** ***Java, Web Application, Integrated Database, Spring Boot Framework, React JS***

## Contents

## 3 Acknowledgements

We would like to thank the faculty of the Frankfurt University of Applied Sciences for their explanations and assistance in writing technical reports, especially Prof. Luigi La Blunda, the supervisor of this project.

During this project, the tasks were divided as shown in the following table:

| | **Involvement** | | | | |
|---|---|---|---|---|---|
| **Task** | LB | VT | LN | TN | AT |
| Write documentation | x | x | x | x | x |
| Prepare slides for presentation | x | | | | x |
| User Interface (Front-end) | | x | x | | |
| Design overall structure | x | x | x | x | x |
| Draw diagrams | x | | | x | |
| Login and Register function | x | | | | |
| Making a new appointment function | | x | x | | |
| Distance of search function | | x | | | |
| Sending email function | | | | | x |
| Export health information as pdf | | | | x | x |
| Reminder function | | x | | x | |
| Cancel, shift appointment function | | | x | | |
| Encryption method | x | | x | | |
| Administrator | | x | | x | |
| Data fetch | | x | x | | |
| API tester | | | | x | |

*LB = Loc Bui Nhien, VT = Vinh Truong Canh Thanh, LN = Linh Ngo Phuc, TN = Tri Nguyen Minh, A = An Truong Nguyen Thien*

## 4 Introduction

### 4.1 Purpose

This final report for the Java Object Oriented Programming Project, eHealth, is submitted to partially satisfy the requirements of the Java Object Oriented Programming course at Frankfurt University of Applied Sciences, and includes a technical description and architecture of the program, the data flow diagram through the program. Our goal is to realize the e-health consultation system as a web-based platform where patients can find the most suitable doctors for them in terms of symptoms and distance. This will bring people the quality of life improvement they need, especially in another wave of Covid-19 approaching.

### 4.2 Audience

This report is for the instructor's review to determine our grade for the Java Object-Oriented Programming course.

## 4.3 Project Scope

This report has been written to familiarize the audience with the features of the program while presenting the schedule and progress of the project.
A detailed explanation of the framework used and the applications and functions are not included in this report. To mark this project as a success, we have to achieve the following requirements [1]:

- The system also provides an administrator who can access, edit, and delete user profiles.
- The user should be able to make a new appointment with a specialized doctor based on their health problem. The program also has a search function to find the most appropriate doctor near the patient. After confirming the appointment, an email is sent to the user to confirm the appointment. Then, a reminder function is available to email the user to remind them of the appointment.
- The user should be able to cancel or reschedule his appointment and then receive an email about the changes. He can also export his health data as a PDF file.
- The source code should come with a Javadoc file.

## 4.4 Constraints

During our time working as a team, we encountered the following setbacks:

- Lack of experience in working with industry and professional projects.
- Correct English must be used in the program.

The project is expected to be completed in 7 weeks. Progress is documented below. The process has been very smooth, with only minor setbacks along the way. Nevertheless, we believe that there is room for improvement in this project. With an extended deadline, the project can become a full-fledged website application.

# 5 Project Overview

## 5.1 Overall Architecture

Seraphine project is created under one of the state-of-the-art approaches for a web application, the Model-View-Controller. Database access is established via many layers of service. For this approach, the user can easily communicate with the server or database through a controller.

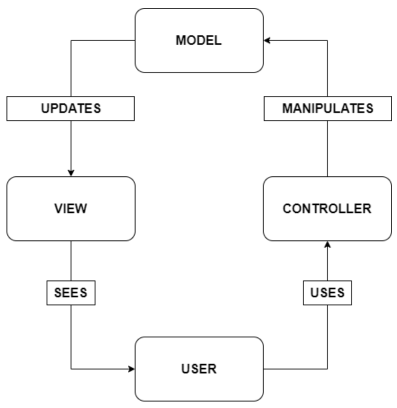


Figure 5.1: Overall Function

## 5.2 Brief Description of Spring Boot Architecture

The project is created as a web-based application, with an integrated graphical user interface for users and admins to access their respective functions. Information processing is handled by Spring Boot application and connected to a local H2 database - provided by Spring Boot application. The project is created based on the module of the Spring Framework. It is used to create stand-alone, production-grade Spring Based Applications with hand-to-hand approaching. Instead of reloading a remarkable number of built-in libraries for the controller, Spring boot allows us to use without reloading or using "servlet" dependency. There are four layers: Presentation Layer (i.e front end and is coded using React), Business Layer to performs authorization and validation, Persistence Layer contains all the storage logic and translates business objects from and to database rows (both coded using Java), and Database layer[2]. Our data is retrieved from the database to the business layer (service) and the persistence layer, and vice versa. In detail, first and foremost, the model, which is said to be mapping into the JPA entity. Secondly, a service layer will deal with the business logic of the web application, which contains an interface service and a corresponding template to implement it. Thirdly, a repository layer is to create an entity and utilize all the functions and queries inherited from JPA repository dependency. Finally, a controller to receive a request from clients.

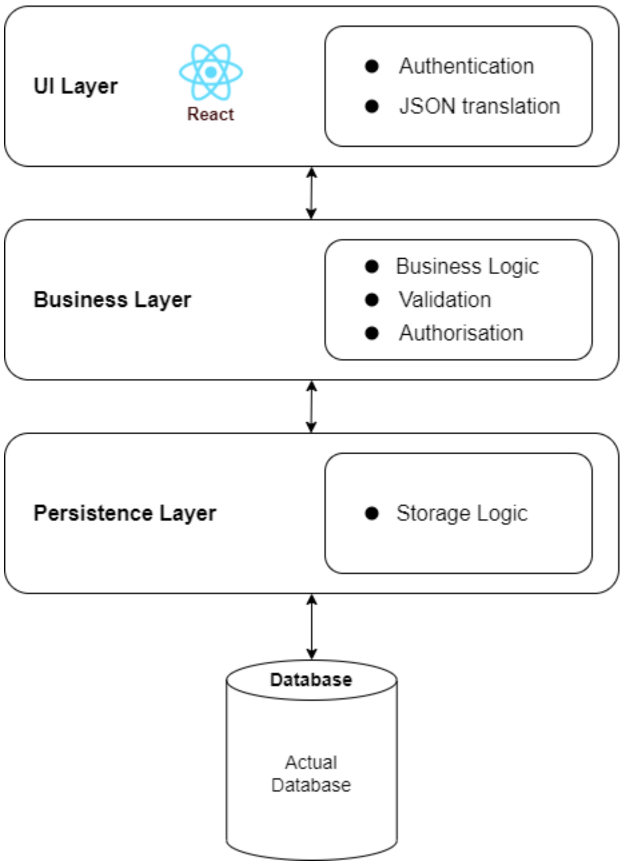


Figure 5.2: System Architecture

## 5.3 Class Diagram

Realizing the basic rule of a web application, which is user-oriented. Standing from the user's story, we use basic role-based operation for defining user and admin. It is clear to see that doctors and users share a common appointment class, which can simplify the whole system. Although our approach is not unique and still has some limitations, the lightweight system is a necessary method to implement and fulfill the requirement of the project.

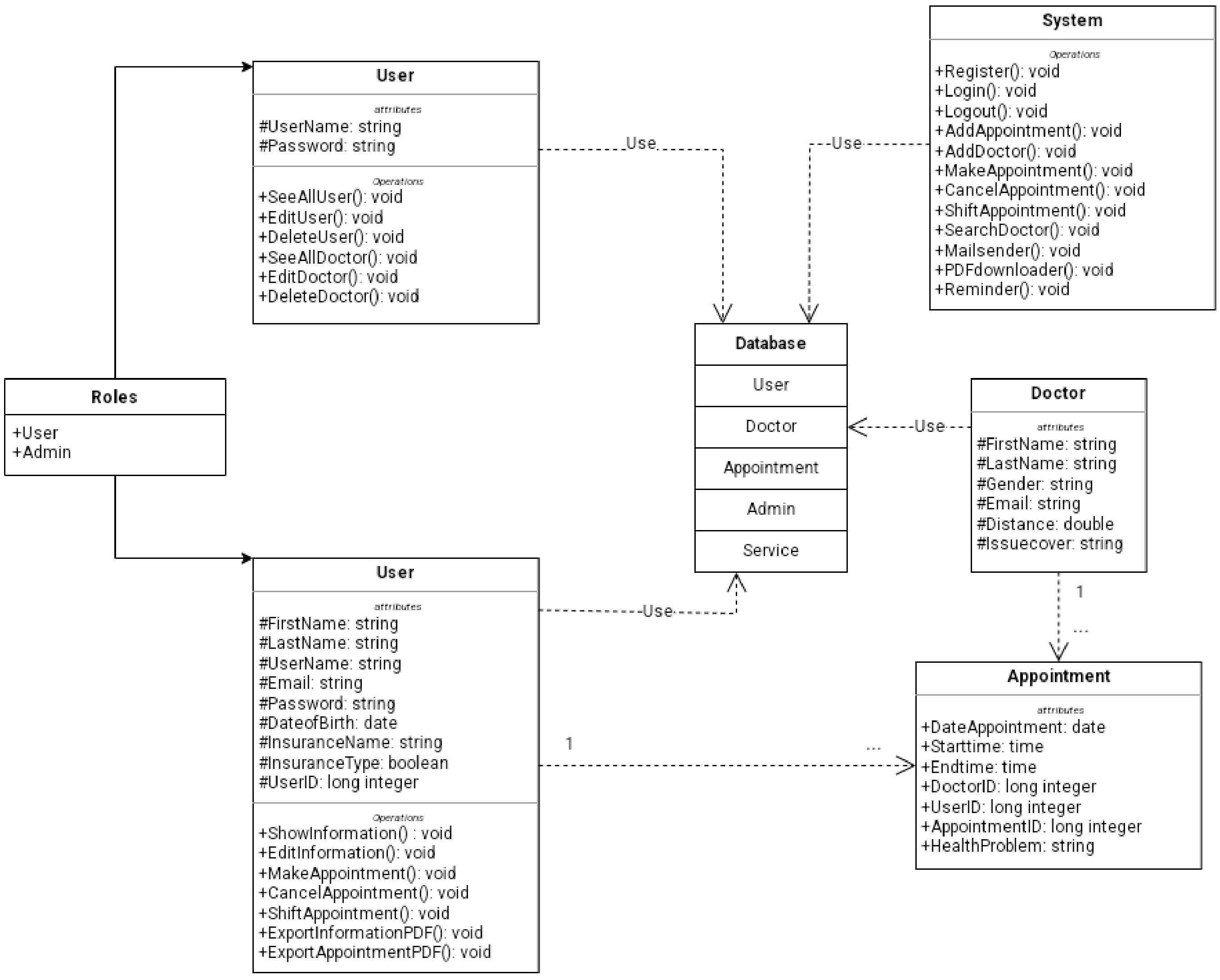


Figure 5.3: Class Diagram representation

## 5.4 Use Case Diagrams

At first, it is obvious that users have 2 fundamental services, which are login and appointment. In the login system, register and forget-password services are provided based on token service, which is a big highlight in our security implementation. Meanwhile, in the appointment service, we choose a simple approach that is booking an appointment for one of the appropriate doctors in the list displayed by the query search function.

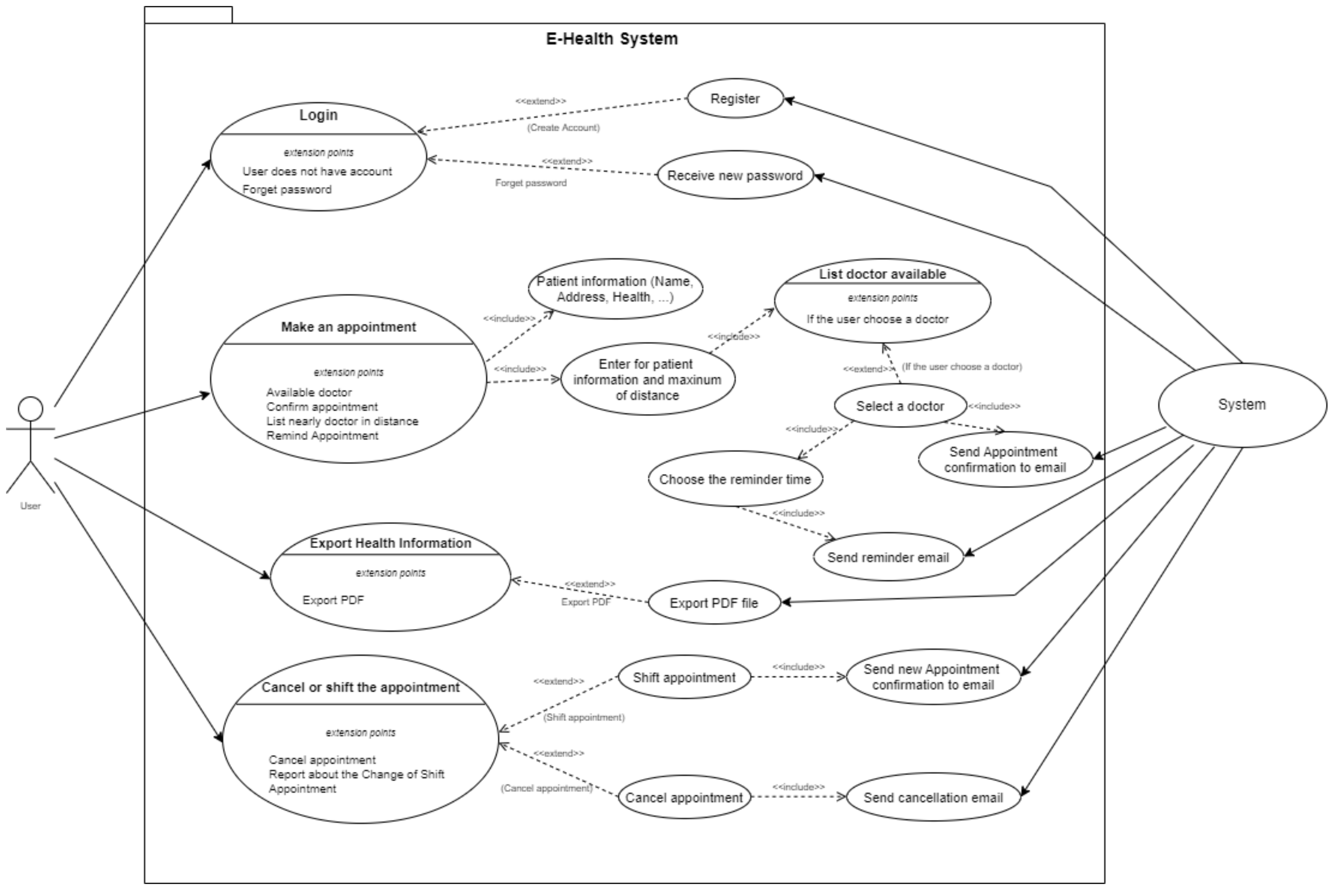


Figure 5.4: Use case Diagram for overall system

## 5.5 Activity Diagrams

The activity figure below illustrates the basic flow for client when accessing to our service. Our main concentration is on the booking appointment function, therefore, we try to split the operations into detail.

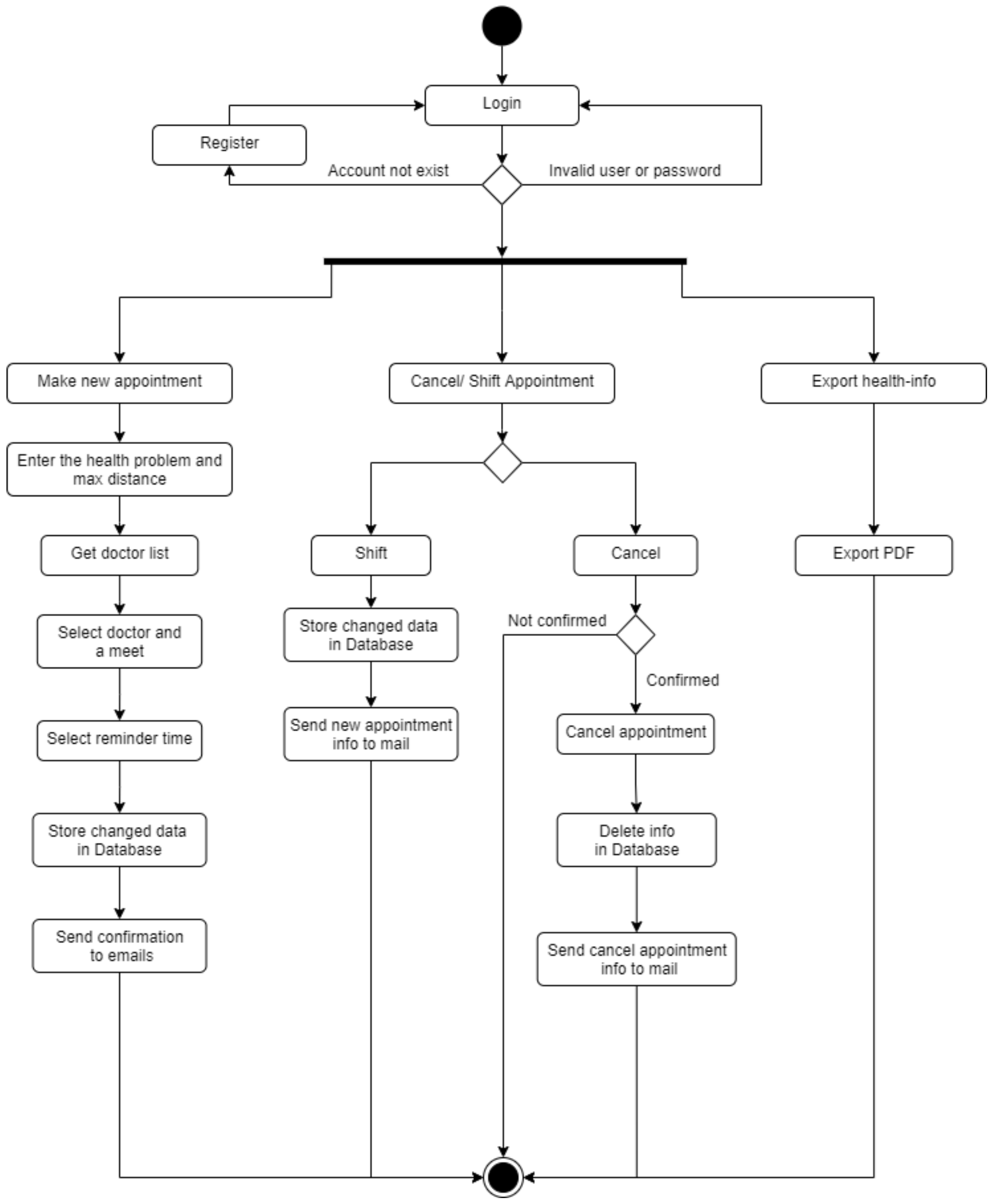


Figure 5.5: Activity Diagram for overall system

## 5.6 Sequence Diagrams

The given sequence figure represents how a user can interact with our website from back-end to front-end service, which happens to be in JSON format. In this case, a request from the user can be performed in CRUD operation, meanwhile, the server's response is varied based on the action of the user, basically, the server can return a message to the user, throw an exception or simply return data in JSON.

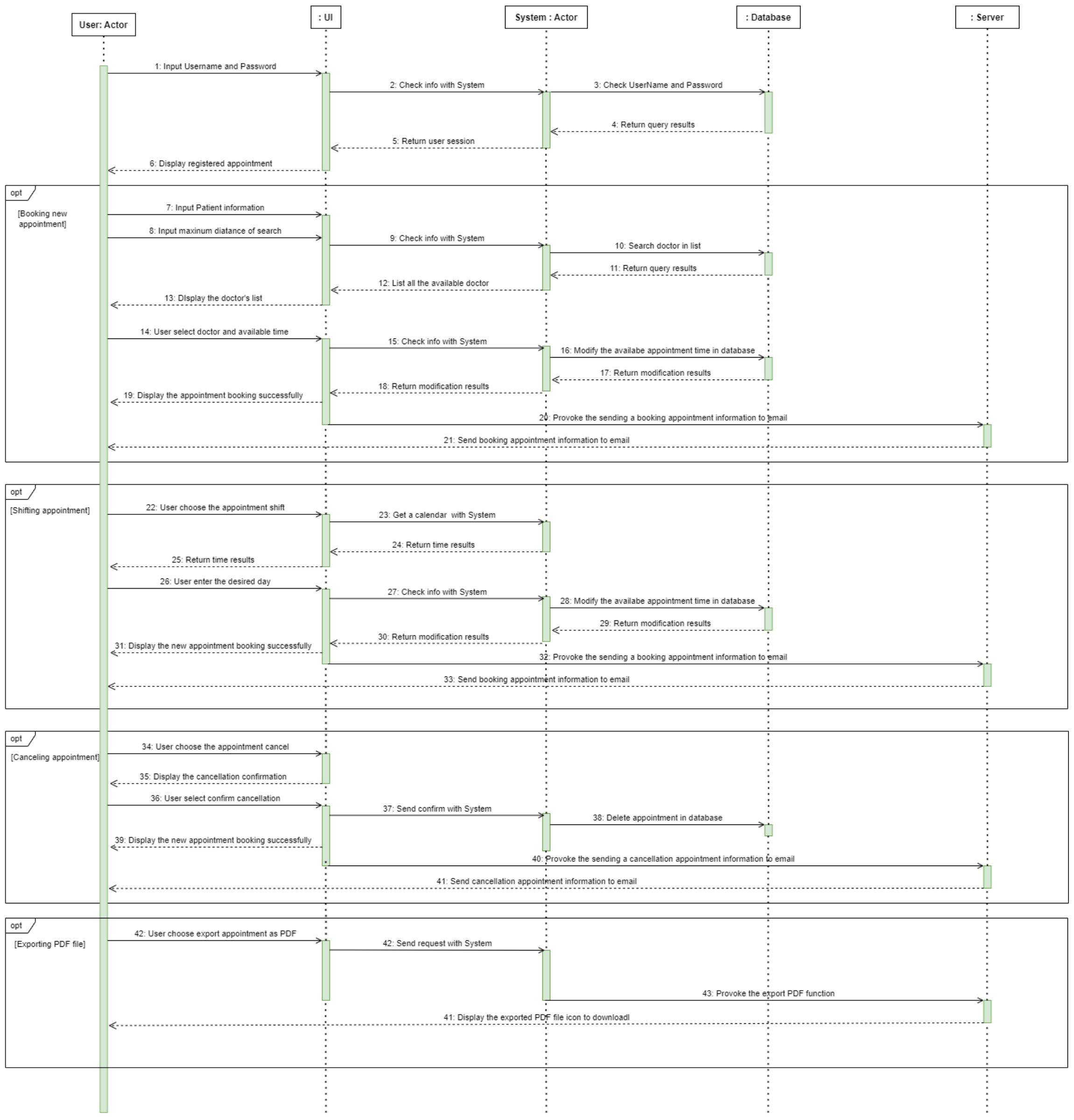

User: Actor
: UI
System : Actor
: Database
: Server
1: Input Username and Password
2: Check info with System
3: Check UserName and Password
4: Return query results
5: Return user session
6: Display registered appointment
opt
[Booking new appointment]
7: Input Patient information
8: Input maxinum diatance of search
9: Check info with System
10: Search doctor in list
11: Return query results
12: List all the available doctor
13: DIsplay the doctor's list
14: User select doctor and available time
15: Check info with System
16: Modify the availabe appointment time in database
17: Return modification results
18: Return modification results
19: Display the appointment booking successfully
20: Provoke the sending a booking appointment information to email
21: Send booking appointment information to email
opt
[Shifting appointment]
22: User choose the appointment shift
23: Get a calendar with System
24: Return time results
25: Return time results
26: User enter the desired day
27: Check info with System
28: Modify the availabe appointment time in database
29: Return modification results
30: Return modification results
31: Display the new appointment booking successfully
32: Provoke the sending a booking appointment information to email
33: Send booking appointment information to email
opt
[Canceling appointment]
34: User choose the appointment cancel
35: Display the cancellation confirmation
36: User select confirm cancellation
37: Send confirm with System
38: Delete appointment in database
39: Display the new appointment booking successfully
40: Provoke the sending a cancellation appointment information to email
41: Send cancellation appointment information to email
opt
[Exporting PDF file]
42: User choose export appointment as PDF
42: Send request with System
43: Provoke the export PDF function
41: Display the exported PDF file icon to downloadl

Figure 5.6: Sequence Diagram representation for user appointment manipulation and file export

# 6 Software

## 6.1 Server Runtime

The web application runs on Tomcat Server version 9.0.56

## 6.2 User Interface

The User Interface is created using React.

## 6.3 Database

We decided to use H2 database for its convenience and lightweight. In-memory database relies on system memory as opposed to disk space for the storage of data. Because memory access is faster than disk access. We use the in-memory database when we do not need to persist the data. The in-memory database is an embedded database. The in-memory databases are volatile, by default, and all stored data is lost when we restart the application. H2 is an embedded, open-source, and in-memory database. It is a relational database management system written in Java. It is a client/server application and stores data in memory, not persist the data on disk. With no further configuration, easy to use, lightweight and fast, simple configuration to switch between a real database and in-memory database, provides a web console to maintain in the database, and supports standard SQL and JDBC API, we found that it is the best approach to the project.[3]

## 6.4 Login and Register

The login page is the first page users see in the application. It contains two text fields - one for entering a login name and one for entering a password - and a command button that initiates password verification. If either of the text fields is left blank, this is an error that should be communicated to the user. If both fields are filled in, but there is no entry for the user name or the password is incorrect, this must also be communicated to the user.
Users who have not yet registered will not be able to log in. They must first register by clicking the Register button. The application also provides the option to reset the password for existing users in the database. All the information for the user will be hashed and saved in the database using the "bcrypt" hashing function. It allows us to build a password security platform that scales with computation power and always hashes every password with a salt. Its purpose is to slow down hackers using dictionary and brute force attacks.[4]

## 6.5 Forgot Password

This is not part of the original requirements of the project, but a necessary extension in case of the user can not remember their passwords. The forgot password function is the action of invalidating the current password for an account, and then creating a new one. To do so, we utilize the email service to send a link when the user request to reset the password. However, before that, the user has to fill in the email they use to create the account, the new password, and re-enter the new password. When the user fails to complete this requirement, error windows can pop up in different situations: "The email entered is invalid", "Can't find the account with the entered email", or "The re-enter password doesn't match". When the requirement is met, the system will generate a unique token and save it. This will work in conjunction with the email service to send the token to the user's email as a link. By clicking this link, the system will check if it is the correct token and the time of clicking is within the time limit. If the user follows the steps correctly, a message will pop up saying their password has been changed successfully and return to the login page.

## 6.6 Appointment Shared Service

Booking an appointment is an essential function in our project. With that being said, the conduction on analyzing in both doctor and user point of view is quite necessary.

### 6.6.1 From doctor's vision

With careful consideration, the approach for our solution architecture focuses on the relationship between doctor and appointment entity, doctor and user. One doctor can have many appointments, it can also be called the "time slots" for users to book an appointment with that doctor. Each appointment is booked, it will reset the status become "booked", which will be eliminated in the doctor appointment list. With a one-to-many relationship, it is convenient for users to choose the appropriate time and for the admin to manage the appointments between doctor and user.

### 6.6.2 From user's vision

Similarly, there exists a list of appointments in the user entity, which means that a user can book a variety of appointments. In the process of developing our application, we are facing many issues in verifying the appointment between user and doctor. In detail, the user can choose an appointment from a list of appointments stored in the doctor entity, which can be queried based on the doctor's primary key. In this case, the appointment entity will contain two foreign keys, doctor ID and user ID. The doctor ID maps to the list of free appointment that is available for a specific doctor, meanwhile, the user ID references the booked appointment that the user has got from a corresponding doctor. To check whether an appointment exists or not and also check whether it is booked or not, we simply implement an algorithm to check the doctor ID (primary key) stored in the appointment attribute is null or not. If the doctor ID is null, the appointment is in the database but belongs to no doctor. If the appointment status is not "booked", the appointment is still available for the user to make and vice versa. The application tends to be user-friendly, which is shown in the way users can create an appointment with a doctor, shift the appointment and also cancel it with only a button. With this relationship, the web service can scale up with multiple users in the future.

## 6.7 Book, Cancel and shift appointment

Standing on the point of view of the user and doctor, we can easily implement the full service for appointment operation. Users can book, shift and cancel an appointment with a single button. Moreover, the admin can create an appointment for a doctor, change doctor's appointments based on their calendar and also delete doctor appointments. To perform these functions, we choose JPA to make access to the database and to create CRUD operations (create, read, update, delete) for our API. Then, React framework will catch the server and retrieve the JSON from API to display to the user.

## 6.8 Sending email

All email is handled by Spring Boot Java Mail Sender. It is configured to use an already-existing G-mail account and send established emails to users. Email sending may be a minor function, but it is also important because it provides validation for the user and sends an announcement about the appointment to the user. In our web application, the email function accounts for 3 main functions, firstly, verifying newly registered users by sending them confirmation links, secondly, if the user forgets their password, email can send a restore link with corresponding restore token to solve the problem of forgetting password. Finally, the email sender is used to send an announcement about a new, shift or cancel appointment. With the collaboration of Google email, the security and reaction of our service are enhanced significantly.

## 6.9 Administration

The administration controller and the administrator are secured by customizing Spring Security framework. Spring Security is an easy-to-use framework. It offers some methods for role-based access control implementation for our application.

### 6.9.1 Administrator

The administrator of the server is defined statically before the server starts. The admin is a user inside of the database. The admin can only use the administration controller.

### 6.9.2 Administration controller

The admin controller provides the admin of the web server with performing basic operations related to the users, such as accessing all of the user profiles, editing user's basic information, and deleting any user profiles.

### 6.9.3 User's role

For the security layer to distinguish between user and admin, we declare a variable that stores the user's role. Every user when having an account registered and confirmed by the server will have their default user role as a user, but the admin will have the user role as admin.

### 6.9.4 Security layer

The security layer is configured so that only the user or only the admin can access the endpoint of the URL. Every time an anonymous user wants to access the endpoint specified only for the admin or for the user, they must prove their identity by logging in. The security layer will then decide whether or not to give access to that endpoint or to deny it based on the user's role in that account. For example, a user has logged in to the application by a user account but wants to access the /admin endpoint. That request will be rejected by the server. To access that endpoint, the user must log out and then use the admin account logged in to access the /admin endpoint.

## 6.10 Search for doctor

### 6.10.1 Three elements for searching problem

To book an appointment for a specific doctor, we provide users with tools that can make their search more efficient than ever. By identifying based on users' views, we find out three aspects that can affect user choice. Firstly, the health problem that they are facing. Secondly, their current location for booking. Finally, the range they want to search. For example, a user can search for "fever, Berliner 18, 64521, and 10 kilometers". As a result, the suitable doctor will be displayed immediately, which can be proof of the efficiency of our algorithm for searching doctors based on distance.

### 6.10.2 The query search

After identifying the problem carefully and precisely, the question of how a website can query for a doctor when the user provides three elements: health problem, current address, and range to search are raised. The answer lies in the query builder, which is a very useful tool for filtering search problems. With the provided parameters from the user, it is not difficult to use a query to make access to the database and search for matching phrases. Luckily, in spring boot, the query builder can be created by using the appropriate dependencies. In this project, we choose to use three classes from Java Persistence API (JPA), which are Predicate, Criteria Builder, Criteria Query.

### 6.10.3 The search distance algorithm

Finally, when dealing with the query search, the problem about search based on user distance is raised. Our group's approach is not a very special or premium method but it is elegant, intuitive, and intelligible. We use open street map API to get user and doctor location data based on what they provide[5]. From that point of view, it is clear to witness that the location of the user is always changing because the user can book appointments not only from their house but also anywhere, however, the doctor's address is stable even though the user's place can change constantly. With the query search, it is easy to find doctors who live in the same city or area where the user is standing. The list of doctors in the same areas helps our algorithm to be more efficient. Firstly, the back-end service will find every single doctor in the database who is currently in the same area as the user and append them to the built-in list. Then, we choose to run a simple "for" loop in that list and constantly apply Haversin's formula to calculate the distance between doctor and user. The Haversin's formula:

$$\Delta\lambda = 2 \cdot \arcsin(\frac{sin(d/2R)}{cos\Phi})$$

In which, R is the radius of Earth. This formula is to find the shortest distance between 2 points in the globular. Finally, the result distance will be assigned for the "distance" attribute in the doctor object. The "distance" attribute of the doctor class is unstable, it can be changed depending on user location. Thus, the complexity of this algorithm is about O(N), in which N is the number of doctors in range, because the doctor in a range, for instance, 10 kilometers or 20 kilometers, is limited, the algorithm can operate constantly and solid.

## 6.11 Export portable document format file

Exporting portable document format (PDF) files are handled by "com.lowagie.text.pdf" - a dependency for PDF reading and downloading. The dependency receives a string and processes to download to the user's local device in PDF format when clicking on the "Export PDF" button. The PDF file stores all the information about the appointment that the user has booked.

## 6.12 Reminder function

When making an appointment, users have the option to choose a reminder with different time intervals: 1 week, 3 days, 1 hour, 10 minutes before the upcoming appointment. Based on the appointment and the reminder interval, we can calculate the notification time by a simple algorithm, and in conjunction with the email sender, inform the user about the upcoming appointment. Built based on the "TimerTask" API, the reminder function runs in another thread, which results in huge progress in the smoothness and simplification of the whole project. With that being said, in this task, we utilized the true power of Java in multi-threading.

# 7 Conclusion

In conclusion, Seraphine is chosen to be an open-source project about the smart medical booking web application, which is built based on Spring Boot and React JavaScript Framework. Our report tends to describe briefly how the application works and the technology we used under the hood. The structure of Seraphine focuses on the lightweight and minimal principle in architecture design, which does not only fulfills the requirements of the Java module but also saves our clients' time and effort.

## 7.1 Limitation

Like a coin has two sides, Seraphine is not the best methodology to implement in the health industry. There are some drawbacks in the simplification of the architecture and also security, which is always a difficult task for even a big company. However, we manage to cover the basics to the user, and with that being said, users can access state-of-the-art technology which helps their life more comfortable when booking appointments with a doctor.

## 7.2 Lesson Learned

By the time of writing this report, growing is just a process for not only as individuals but also as a team. Having different input on how to face adversity and to work together as a team, we have learned to be more flexible, to think outside the box, and to deal with argumentative situations. Particularly, we always keep updating the project to discover the limitations, fix the bugs, and enhance specific functions of the web application. Hence, it is very important to be open to sharing experiences and receiving comments from the community.

## 7.3 Future Development

We truly believe in the key to development lies in the constant update in technology and system design. Micro-service architecture can be applied to scale up the system with more functions and attributes. Moreover, when the number of users increases significantly, a complex load balancing is required to solve the load-bearing problem.

## References

[1] L. L. Blunda, "eHealth", 2021.

[2] M. Sharma, "Spring Boot Architecture and Workflow", Link: *https://dzone.com/articles/spring-boot-architecture-and-workflow*, Last accessed: 31st December, 2020.

[3] JavaPoint, "Spring Boot H2 Database", Link: *https://www.javatpoint.com/spring-boot-h2-database*

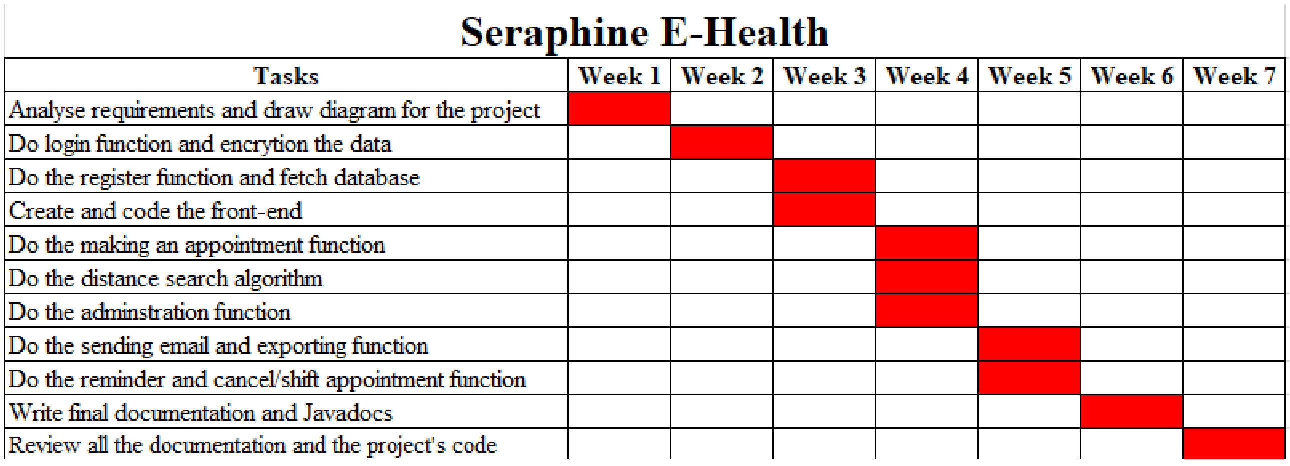
[4] D. Arias, "Hashing in Action: Understanding bcrypt", Link: *https://auth0.com/blog/hashing-in-action-understanding-bcrypt/*, Last accessed: 25th February, 2021.

[5] OpenStreetMap Foundation, "OpenStreetMap (OSM)", Link: *https://www.openstreetmap.org/map=6/51.330/10.453*, Updated: every minute

## 8 Appendix A: Glossary of Terms

| | |
|---|---|
| JPA | Java Persistence API |
| API | Application Programming Interface |
| ID | Identification |
| SQL | Structured Query Language |
| JDBC | Java Database Connectivity |
| CRUD | Create, Read, Update and Delete |
| JSON | JavaScript Object Notation |

## 9 Appendix B: Gantt Chart



Seraphine E-Health

| Tasks | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 |
|---|---|---|---|---|---|---|---|
| Analyse requirements and draw diagram for the project | 🟥 | | | | | | |
| Do login function and encrytion the data | | 🟥 | | | | | |
| Do the register function and fetch database | | | 🟥 | | | | |
| Create and code the front-end | | | 🟥 | | | | |
| Do the making an appointment function | | | | 🟥 | | | |
| Do the distance search algorithm | | | | 🟥 | | | |
| Do the adminstration function | | | | 🟥 | | | |
| Do the sending email and exporting function | | | | | 🟥 | | |
| Do the reminder and cancel/shift appointment function | | | | | 🟥 | | |
| Write final documentation and Javadocs | | | | | | 🟥 | |
| Review all the documentation and the project's code | | | | | | | 🟥 |

Figure 9.1: Project Gantt Chart