Architecture Document

SWEN90007

SWEN90007 SM2 2020 Project

Team: SDA8

In charge of: Jiashuai Yu, Sufan Xia, Thomas Capicchiano, and Zhuolun Wu

Revision History

Date	Version	Description	Author
25/08/2021	01.00-D01	Initial draft	Zhuolun Wu
15/08/2021	01.00-D02	Added Section 6 scenarios	Zhuolun Wu
20/08/2021	01.00-D03	Added Section 4	Zhuolun Wu
20/08/2021	01.00	First version of the document	Zhuolun Wu
23/08/2021	02.00-D01	Added Section 5.1	Zhuolun Wu
23/08/2021	02.00-D02	Added Section 5.2 and 5.4	Zhuolun Wu
25/08/2021	02.00-D03	Added components part in Section 4	Zhuolun Wu
26/08/2021	02.00	Added Section 5.3	Jiashuai Yu
26/08/2021	03.00-D1	Added Section 7	Zhuolun Wu
26/08/2021	03.00	Review on the document	Zhuolun Wu and Jiashuai Yu

Table of Contents

1. Introduction

This document specifies the system's architecture Covid Vaccine Reservation, describing its main standards, module, components, frameworks, and integrations.

This document will be updated along with our development in the following sprints.

1.1 Proposal

The purpose of this document is to give, in high level overview, a technical solution to be followed, emphasizing the components and frameworks that will be reused and researched, as well as the interfaces and integration of them.

1.2 Target Users

This document is aimed at the project team, with a consolidated reference to the research and evolution of the system with the focus on technical solutions to be followed.

1.3 Conventions, terms, and abbreviations

This section explains the concept of some important terms that will be used throughout this document. These terms are detailed alphabetically in the following table.

Term	Description
Component	Reusable and independent software element with well-defined public interface, which encapsulates numerous functionalities, and which can be easily integrated with other components.
Module	Logical grouping of functionalities to facilitate the division and understanding of software.

2. Architectural representation

The specification of the system's architecture Covid Vaccine Reservation follows the *framework* "4+1" **Error! Reference source not found.**, which defines a set of views, as shown in Figure 1. Each of these views approaches aspects of architectural relevance under different perspectives:

- The **logical view** shows the significant elements of the project for the adopted architecture and the relationship between them. Between the main elements are modules, components, packages, and the application main classes.
- The **process view** is omitted, a few figures are included in patterns.
- The **development view** focuses on aspects relating to the organization of the system's source code, architectural patterns used and orientations and the norms for the system's development.
- The **physical view** is omitted.
- The **scenarios** show a subset of the architecturally significant use cases of the system.

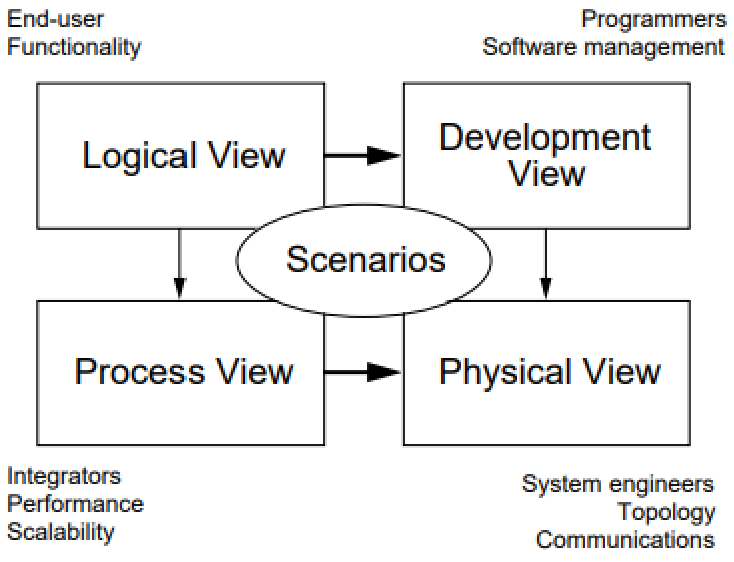


Figure 1. Views of *framework* "4+1"

source: Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE software*, *12*(6), 42-50.

3. Architectural Objectives and Restrictions

The defined architecture's main objective is to make the system portable, scalable, tolerance to imperfections, and has great performance. It should use all necessary patterns to make the structure efficient, easy to understand, and achieves high cohesion and low coupling.

The architecture would be restricted to certain requirements and deployment environments, which would be discussed in the following section.

3.1 Requirements of Architectural Relevance

This section lists the requirements that have impact on the system's architecture Covid Vaccine Reservation and the treatment given to each of them.

Requirement	Impact	Treatment
Questionnaires should be used	Users' answers are only for eligibility test, pointless to store those in database	Database would only store questions and desired answers for each vaccine types, users' answers would only be stored in memory for eligibility test.
Object oriented architecture (domain model)	Memory consumption must be minimized	Lazy load pattern would be considered, which not only reduce query time in database, also save memories for initialized in-memory objects.
Deploy on Heroku	High network latency	Heroku's servers are in the US and Ireland, so there is nothing we can do unless we deploy the system somewhere else.

4. Logical View

This section shows the system's architecture from a domain model view, the main elements are logical classes and their relations, while implementation details are omitted.

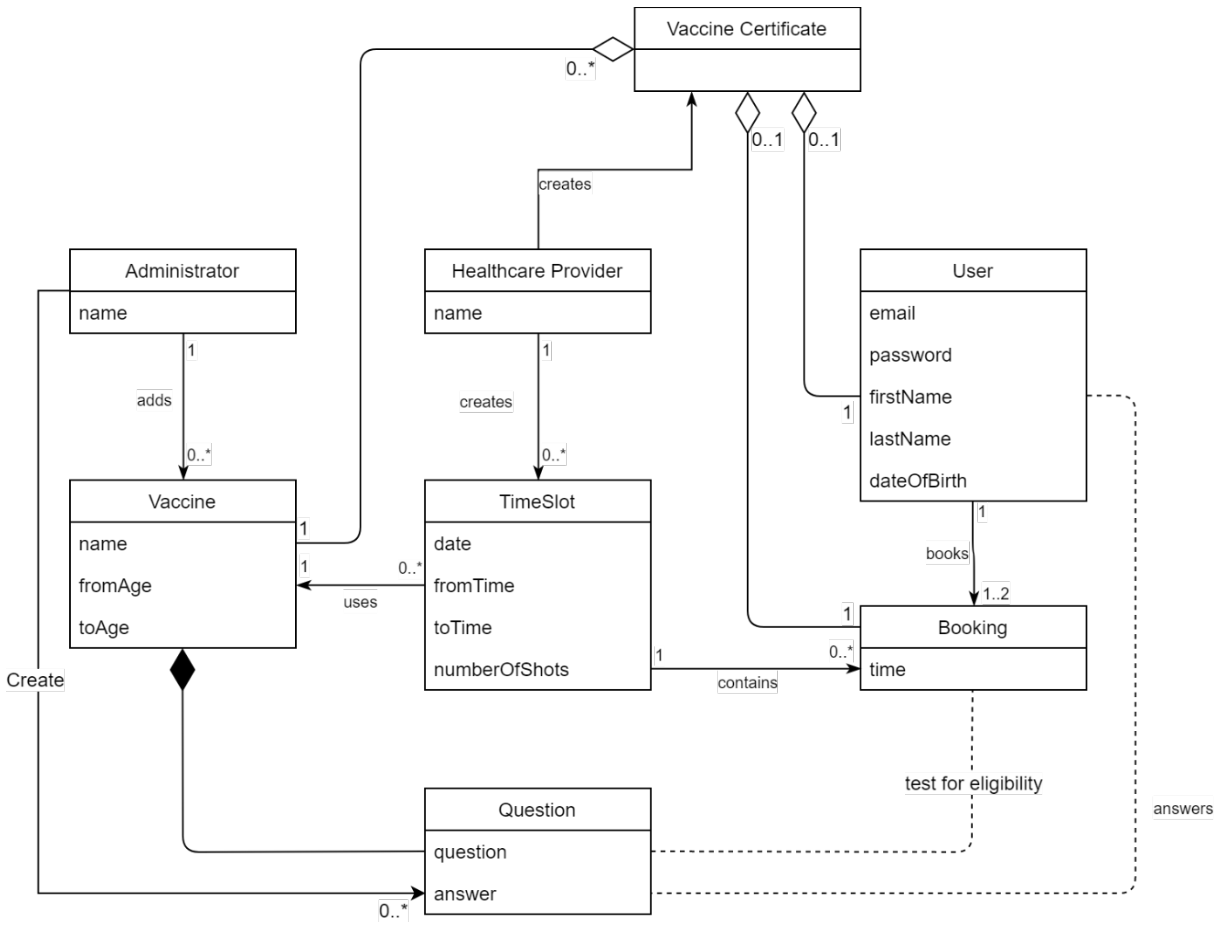


Figure 2. Domain model of the architecture

Key domain logics:

1. A questionnaire is considered as a collection of questions related to a specific vaccine type.
2. Answering questionnaire for eligibility is a prerequisite for booking a vaccination.
3. Vaccine types are added by the admin, timeslots are created by the healthcare provider.

5. Development View

This section provides orientations to the project and system implementation in accordance with the established architecture. Figure 2 illustrates the implementation view of system architecture.

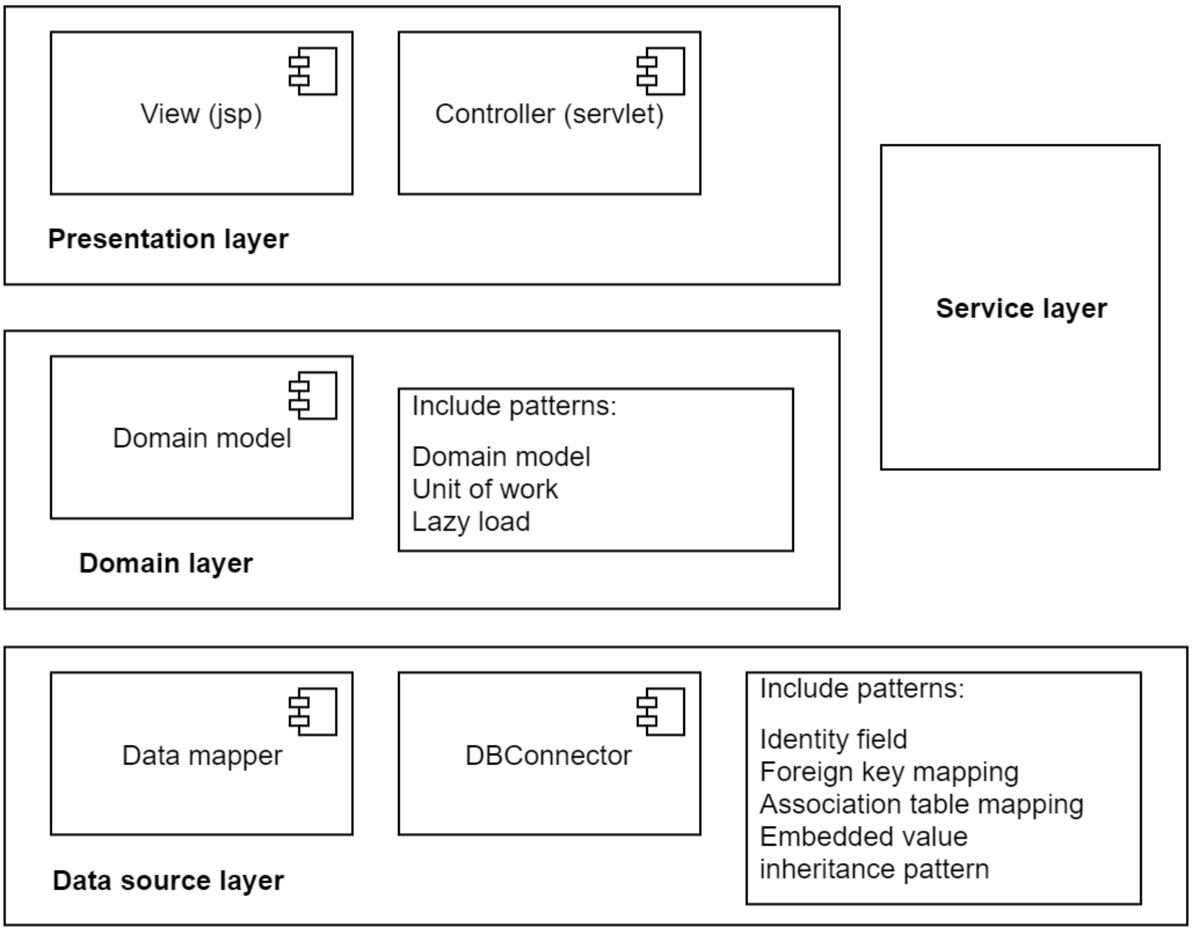


Figure 2. Development view of system architecture

5.1 Architectural Patterns

	Pattern	Reason
Domain Logic	Domain model	Meets architecture requirements. It provides high extensibility, scalability, and re-usability, which are cons of other patterns (transaction script and table module).
Data Source	Data mapper	Isolate data source (database) from domain objects, results in low coupling.
O2R Behavioral	Unit of work	Track changes of certain domain objects and reduce loads on database.
	Lazy load	Delay the loading of an object, reduce system memory usage.
O2R Structural	Identity field	Maintain object identity between an in-memory object and a database row.
	Foreign key mapping	Maps an association between objects to a foreign key reference between tables.
	Association table mapping	Saves an association as a table with foreign keys to the tables that are linked by the association.
	Embedded value	Maps an object into several fields of another object's table.
	Concrete table inheritance	Represents an inheritance hierarchy of classes with one table per concrete class in the hierarchy.
Authentication & Authorization	Authentication	(Not implemented yet)
	Authorization	

5.1.1 Domain model & Service layer

To support object-oriented architecture, domain model is an ideal pattern. It provides each entity certain logic and methods, guarantees extensibility, scalability, and re-usability.

Whereas basic methods and logics are wrapped as services, therefore they are managed more easily.

5.1.2 Data Source

5.1.2.1 Data mapper

Data mapper is used as an entry for every pair of model and database table (e.g., *userModel* class and users table). It moves data between objects and a database while keeping them independent of each other and the mapper itself.

In addition, some specific models/classes with simple structures use equivalent row data gateway pattern and active record pattern, while the data mapper dominates the overall design.

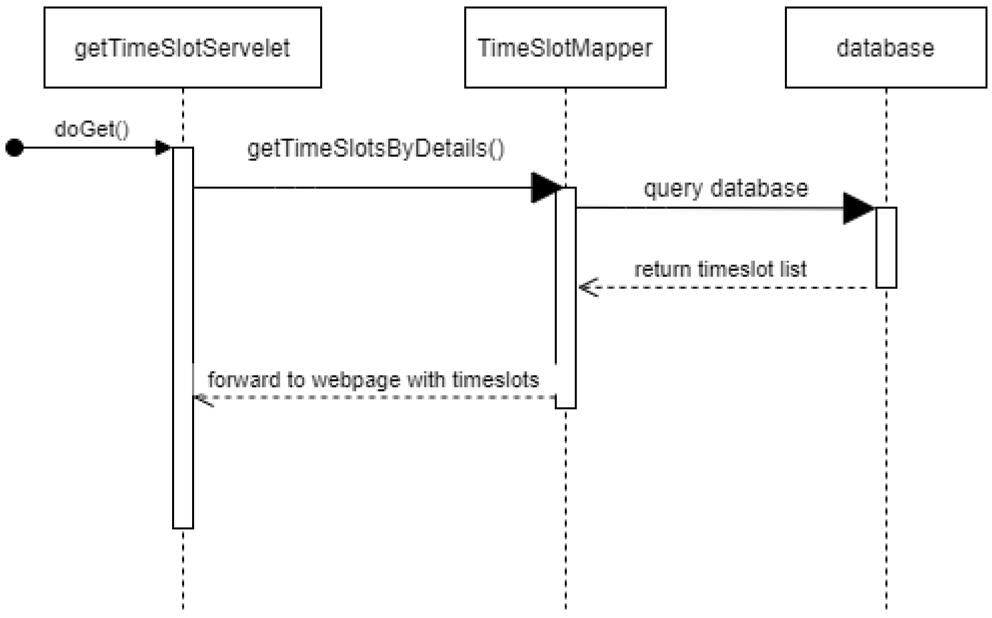


Figure 4. An example sequence diagram for data mapper (timeslot)

5.1.3 O2R Behavioral

5.1.3.1 Unit of work

The unit of work is used to substitute a series of commits caused by a mutual initial request. Take deleting a certain timeslot would use multiple queries as an example (not implemented in part 2), the unit of work pattern would wrap and execute all queries together. Sequences of queries are carefully considered as well, delete all bookings first and timeslots second.

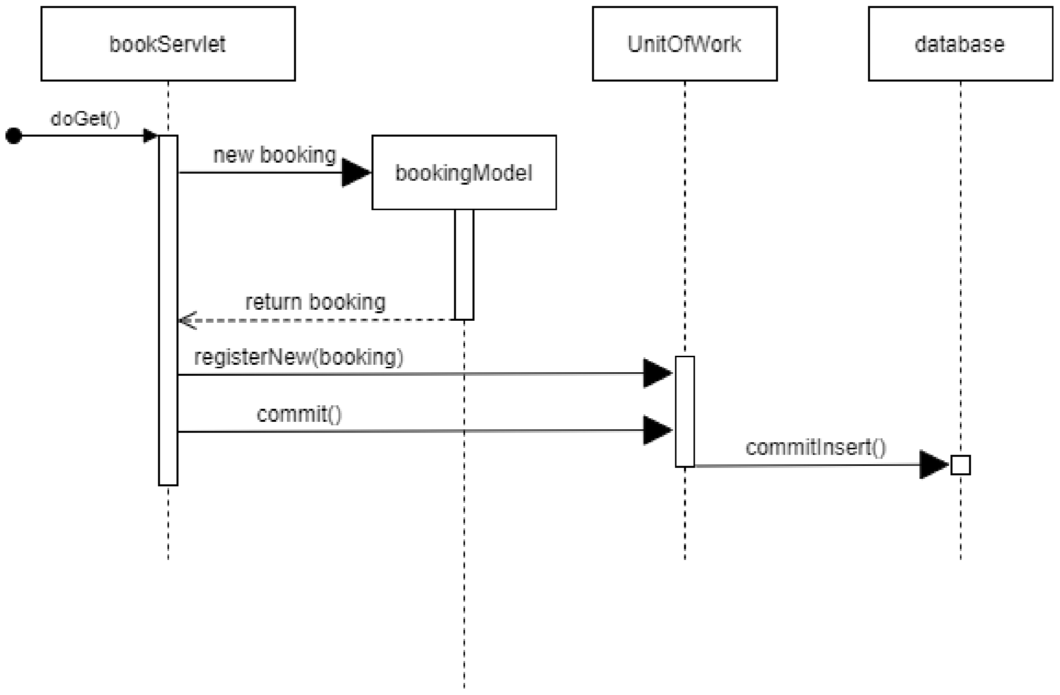


Figure 5. An example sequence diagram for unit of work

5.1.3.2 Lazy load

As an inverse of the unit of work pattern, data read from the database are reduced as much as possible. For example, when a user signed in, an *userModel* object would be initialized with only necessary data, such as email. Other details would be read and loaded when they are needed.

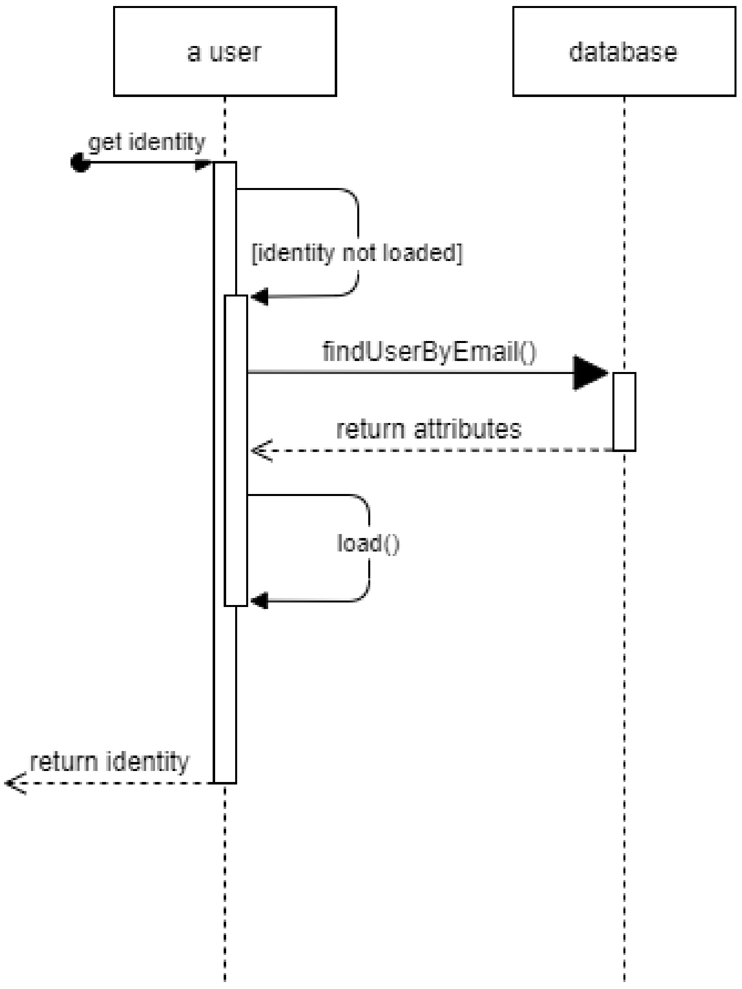
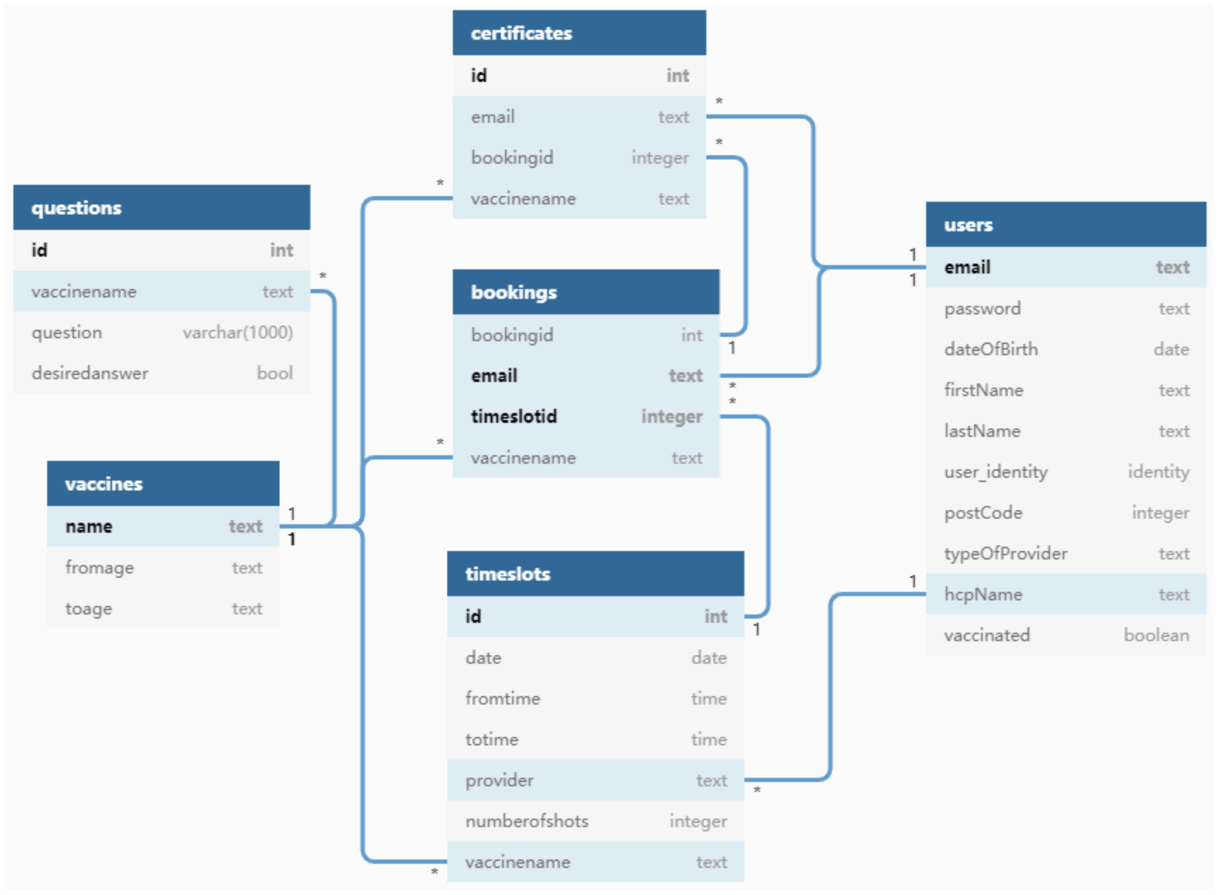


Figure 5. An example sequence diagram for lazy load

5.1.4 O2R Structural patterns

Figure 4. Development view of system architecture



5.1.4.1 Identity field

Each table (object) contains an ID or ID equivalent field. For example, table timeslot has an ‘id’ field, and table users uses ‘email’ as an ID field. These ID field should contain only unique values.

5.1.4.2 Foreign key mapping

Foreign keys are used in some tables to create firm associations. For example, timeslots would contain a foreign key from vaccines to identify the type of vaccine that is applicable to this timeslot.

5.1.4.3 Association table mapping

Some tables are created by association table mapping, like table booking and certificates. These tables only contain foreign keys from other tables, and themselves represent an association relationship.

5.1.4.4 Embedded value

The table timeslots contain a time range (*fromtime* and *totime*), which is awkward to have these time values in an independent table (object).

5.1.4.5 Concrete table inheritance

A class is designed for each concrete table, like *userModel* class and vaccineModel class. Table like certificates would not have a class for it since it is formed from association table mapping.

5.2 Source Code Directories Structure

```
- docs  // Report documentations
- src   // Source code
    - src/main
        - org/unimelb/cis/swen90007sda8
            - DBConnector   // DBConnector
            - Mappers       // Data mapper
            - Models        // Domain model
            - UnitOfWork
            - Servlets      // Controller (servlet)
        - webapp    // View (jsp)
```

5.3 Libraries and *Frameworks*

This section describes libraries and *frameworks* used by the system Covid Vaccine Reservation.

<Omitted for part 2>

5.4 Development Environment

The development environment will be formed by:

Development requirement	**Tool**	**Description**
Architecture designing graphs	Draw.io	Draw.io is used for drawing diagrams for architecture designing
IDE	IntelliJ	(V 2021.2)
Servlet	Apache Tomcat	Tomcat provides an HTTP web server environment in which Java code can run (V 9.0.52)
Database	PostgreSQL	Deployed on Heroku as a PostgreSQL extension (V 13.4)
Deployment	Heroku	Deployed on Heroku (US server)

6. Scenarios

1. Admin heard a new type of vaccine is order by the Australian government, so admin wants to add a vaccine type to system. At meantime, admin collects details and restrictions of the vaccine from medical advises and adds age restrictions and health condition related questionnaires, therefore healthcare providers can add timeslots to these vaccines and normal users can book for these vaccines.
2. Healthcare providers received new vaccines, and they want to add timeslots so that these vaccines can be booked by normal users.
3. A normal user who is seeking for vaccination registers himself to the system and finds available timeslots. Then the user answers a questionnaire which includes questions of age and health conditions. The system shows the user that he is eligible for this type of vaccine after the user has finished the questionnaire, therefore the user now has successfully booked the vaccination.
4. A normal user has successfully vaccinated, not the healthcare provider wants to set the user to the vaccinated state.
5. Based on scenarios above, the user wants to have a vaccination certificate. Based on the user's vaccinated state, it shows the user his certificate and make it downloadable.

Note: Functionalities of questionnaire and certificate are not implemented in part 2.

7. References

Detailed use cases (another deliverable for part 2)