
Architecture Document

SWEN90007

SWEN90007 SM2 2021 Project

Team: SDA8

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



SCHOOL OF
**COMPUTING &
INFORMATION
SYSTEMS**

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-D01	Added Section 7	Zhuolun Wu
26/08/2021	03.00	Review on the document	Zhuolun Wu and Jiashuai Yu
14/10/2021	04.00-D01	Fix patterns for part 2	Zhuolun Wu
15/10/2021	04.00	Fix rationales for part 2	Jiashuai Yu
15/10/2021	05.00-D01	Concurrency issues and pattern	Jiashuai Yu
--/10/2021	05.00-D01	Concurrency testing	Thomas Cappiciano
17/10/2021	05.00	Final review	All team members

Table of Contents

1. Introduction	4
1.1 Proposal.....	4
1.2 Target Users.....	4
1.3 Conventions, terms, and abbreviations	4
2. Architectural representation	5
3. Architectural Objectives and Restrictions	6
3.1 Requirements of Architectural Relevance	6
4. Class diagram	7
5. Logical View	11
6. Development View	12
6.1 Architectural Patterns	13
6.1.1 Patterns overview	13
6.1.2 Domain model & Service layer	14
6.1.3 Data Source	14
6.1.4 O2R Behavioral	15
6.1.5 O2R Structural patterns	17
6.1.6 Authentication and Authorization	18
6.1.7 Concurrency	19
6.2 Source Code Directories Structure	29
6.3 Libraries and Frameworks.....	29
6.4 Development Environment.....	29
7. Scenarios.....	30
8. References.....	31

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.

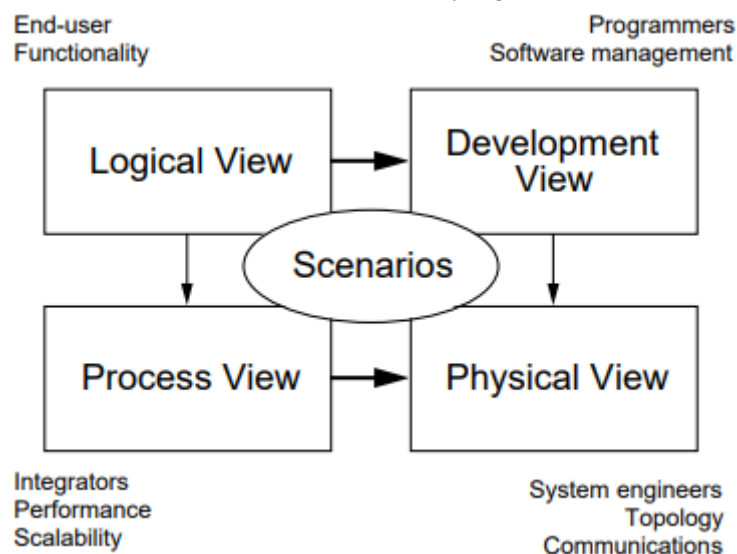


Figure 1. Views of *framework* "4+1"

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. Class diagram

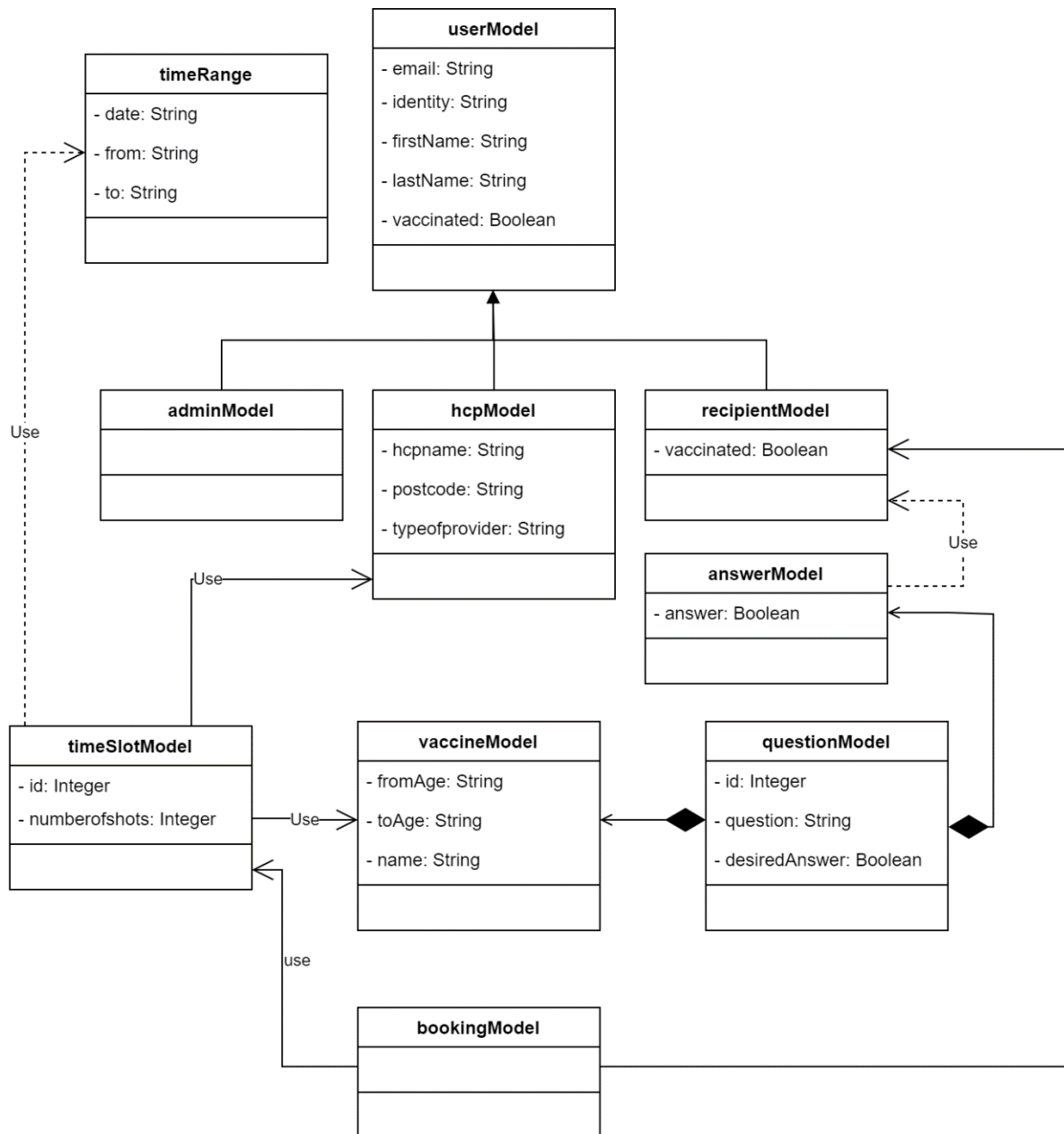


Figure 2.1 Design class diagram (part 1 - exclude methods)

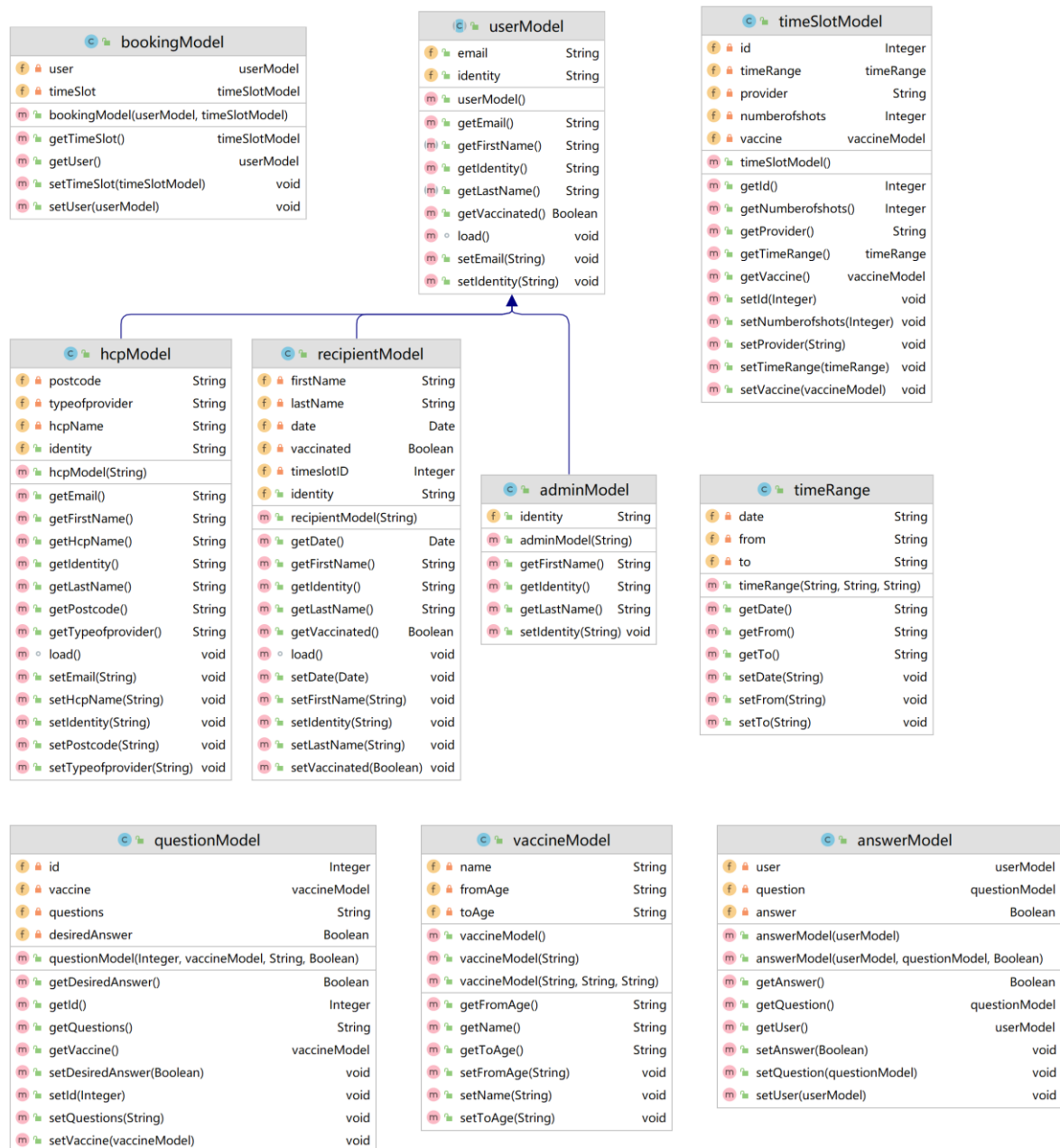


Figure 2.2 Design class diagram (part 2 - exclude associations)

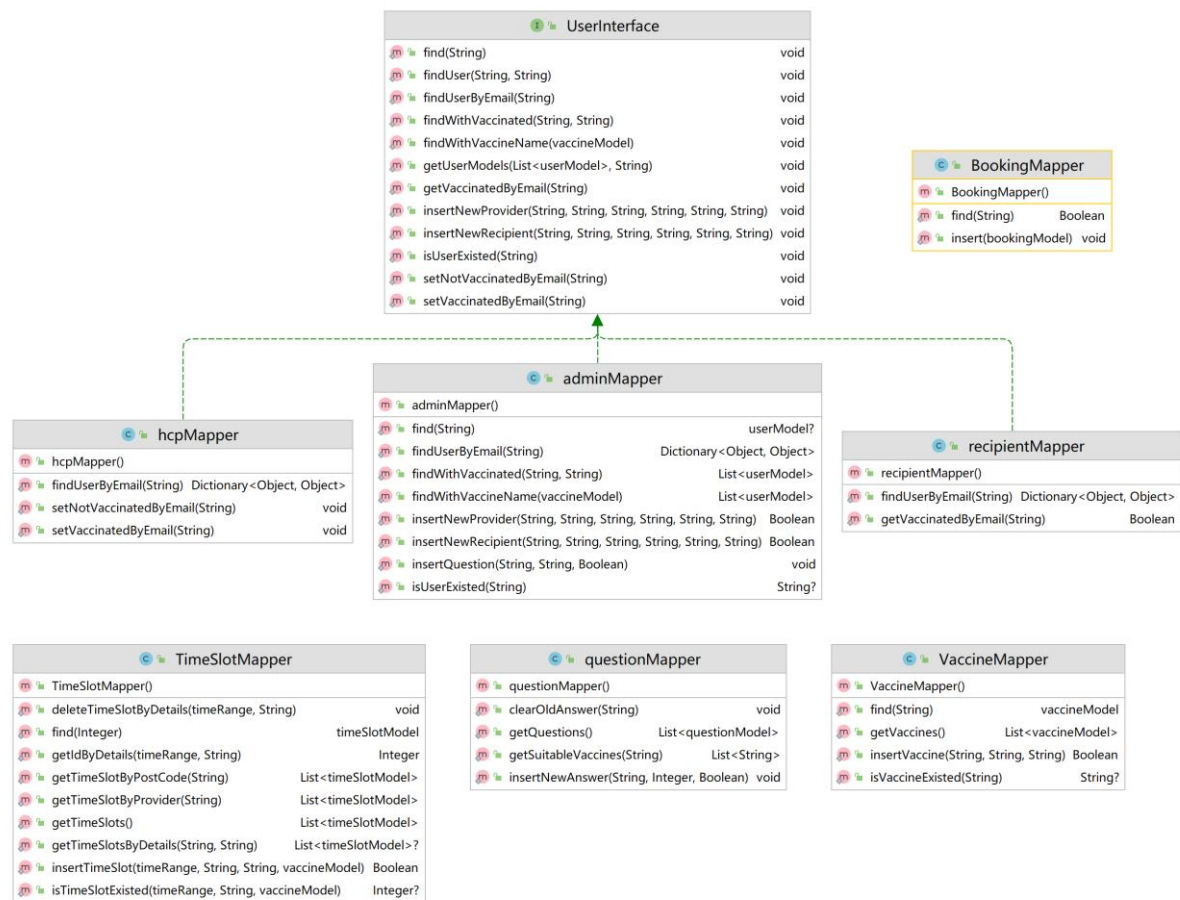


Figure 2.3 Mapper classes

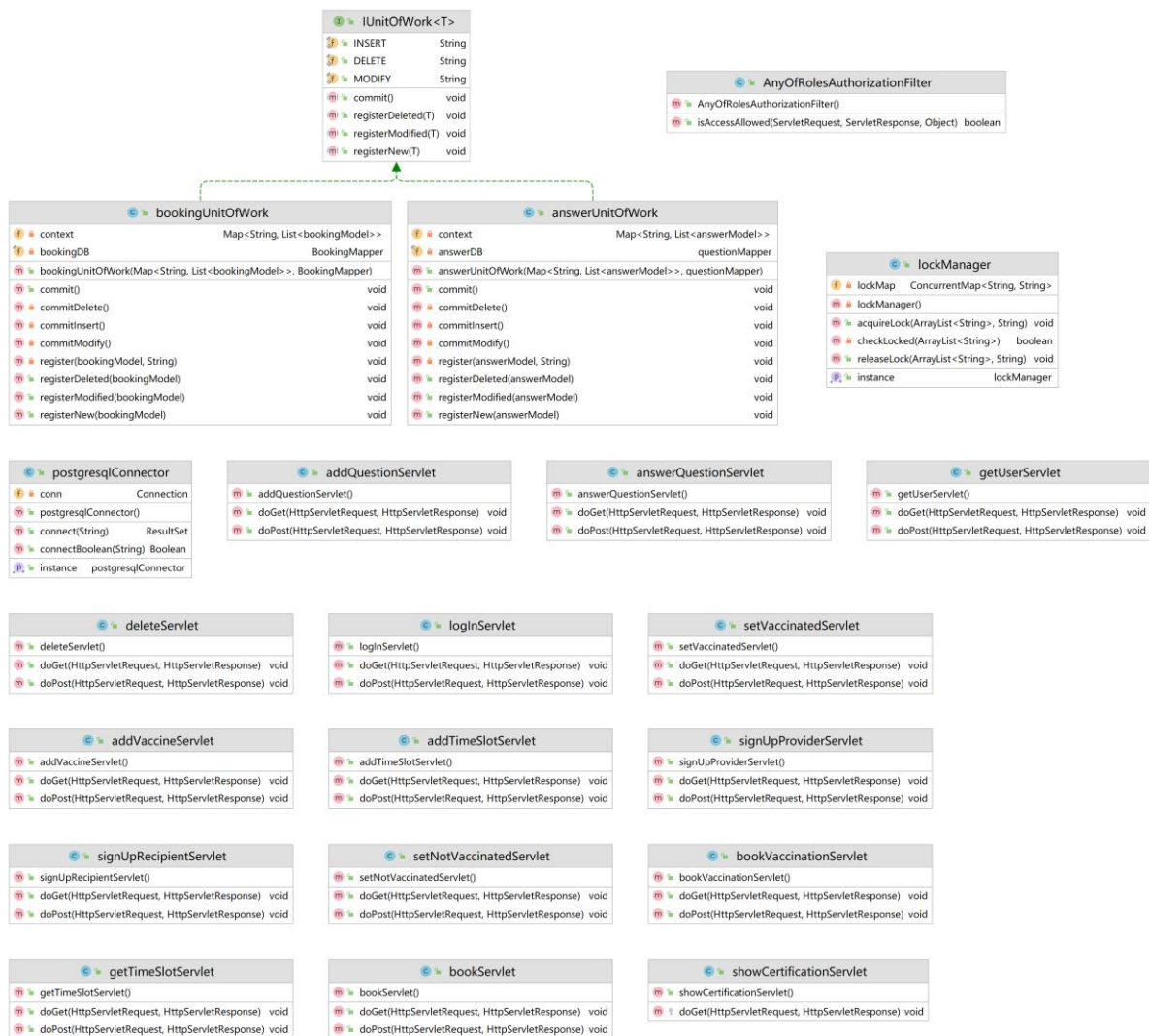


Figure 2.4 Servlets and other classes (lockManager, unitOfWork, etc.)

5. 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.

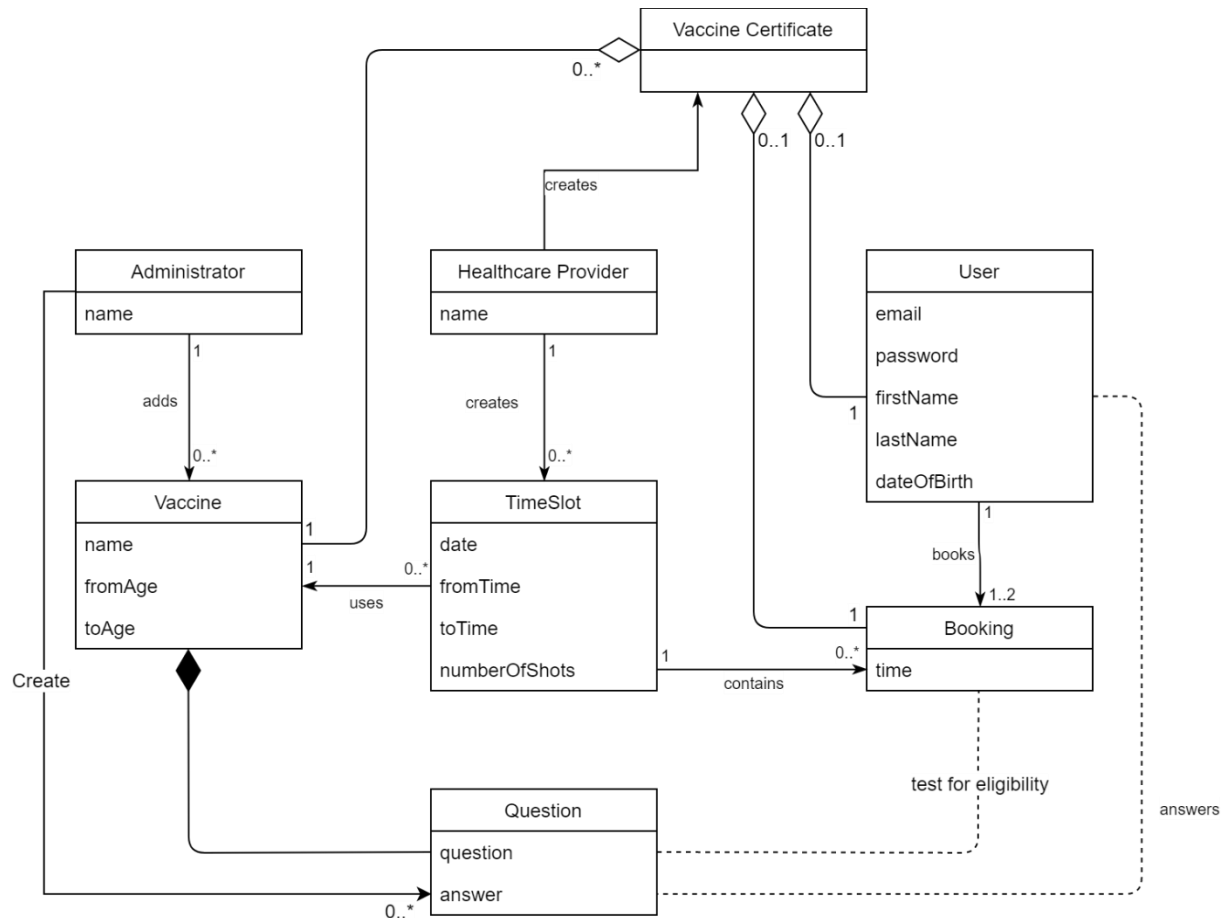


Figure 3. Domain model of the architecture

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.

6. Development View

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

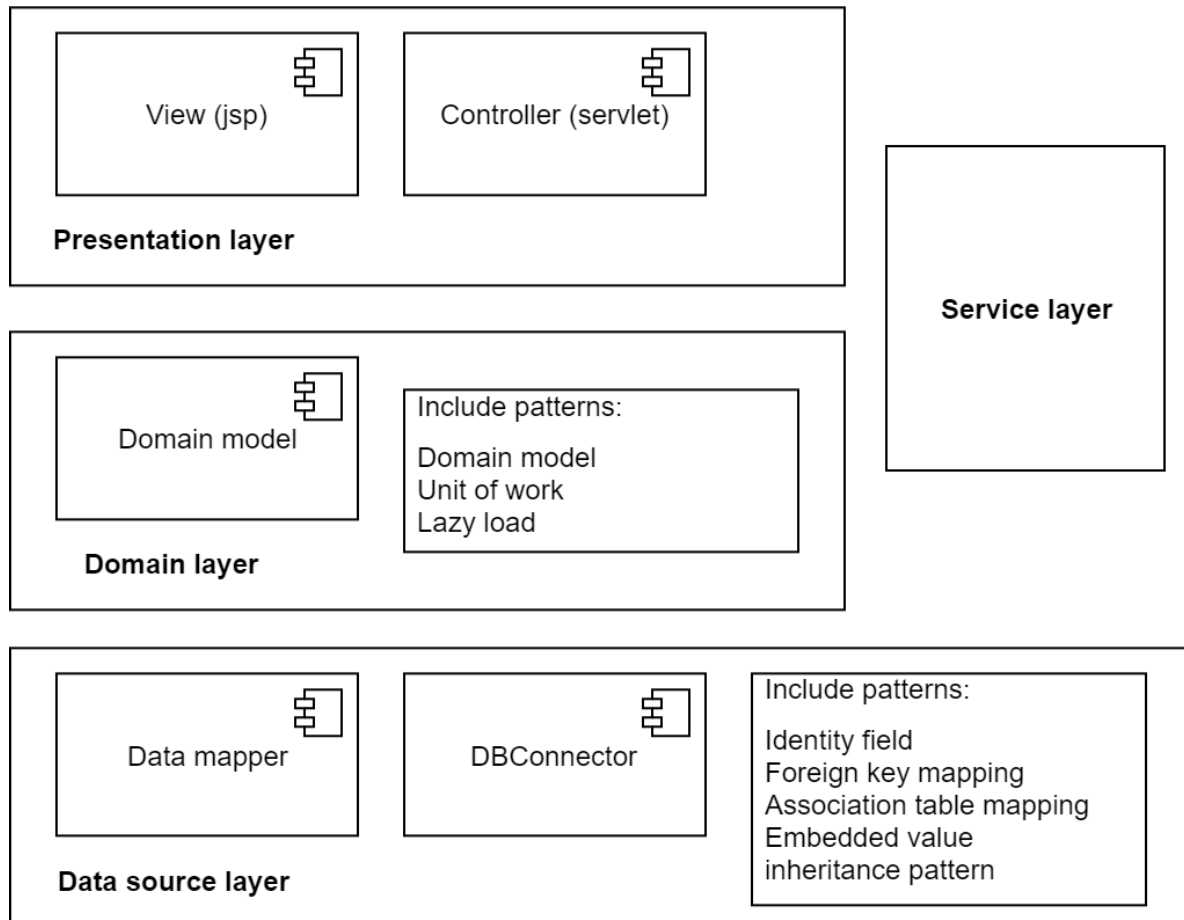


Figure 4. Development view of system architecture

6.1 Architectural Patterns

6.1.1 Patterns overview

	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	Apache Shiro is used for both authentication and authorization.
	Authorization	
Concurrency	Pessimistic concurrency control	Concurrency issues are handled in a pessimistic way, a singleton lock manager class (object) is implemented.

6.1.2 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.

6.1.3 Data Source

6.1.3.1 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.

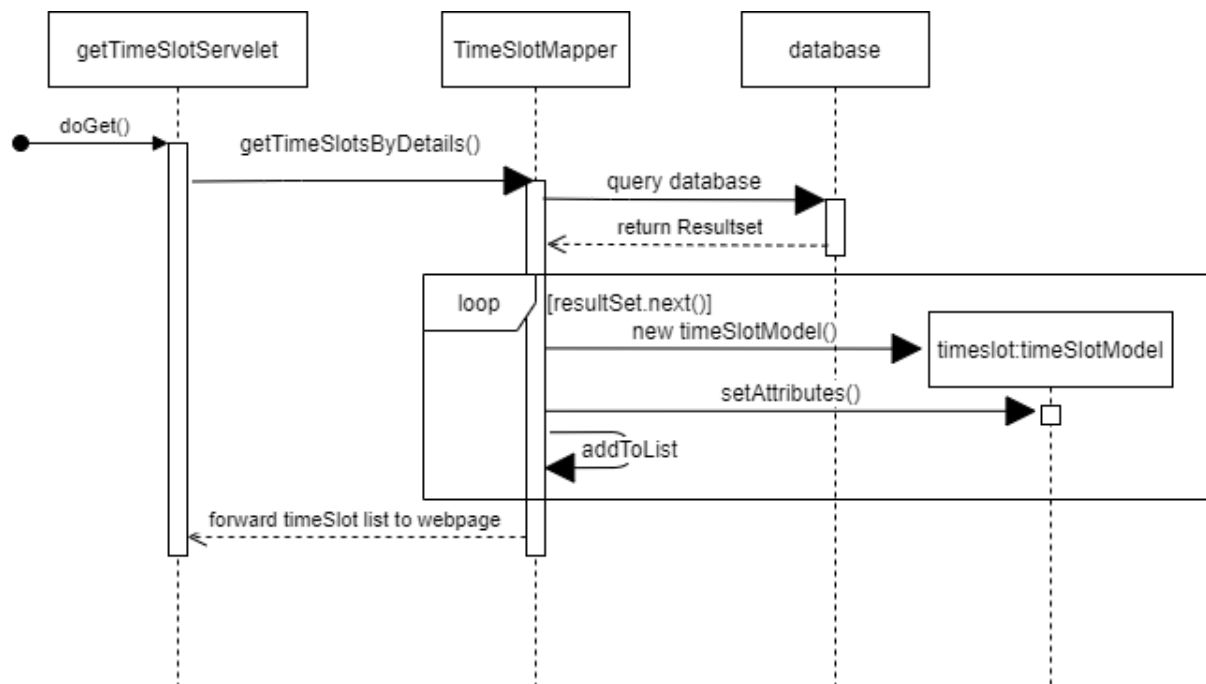


Figure 5. An example sequence diagram for data mapper (timeslot)

6.1.4 O2R Behavioral

6.1.4.1 Unit of work

The unit of work pattern encapsulates multiple queries made by a user into a unit and then commits them together. It retains a list of objects that is modified or updated, and flush all when an in-memory transaction is finished.

In the designing of our system, it is used for transactions that would change multiple rows or even multiple tables. For example, one use case is recipient answering questionnaire for vaccine eligibility. These questions' answer would be registered as new object in UoW object, and then commit in once to database.

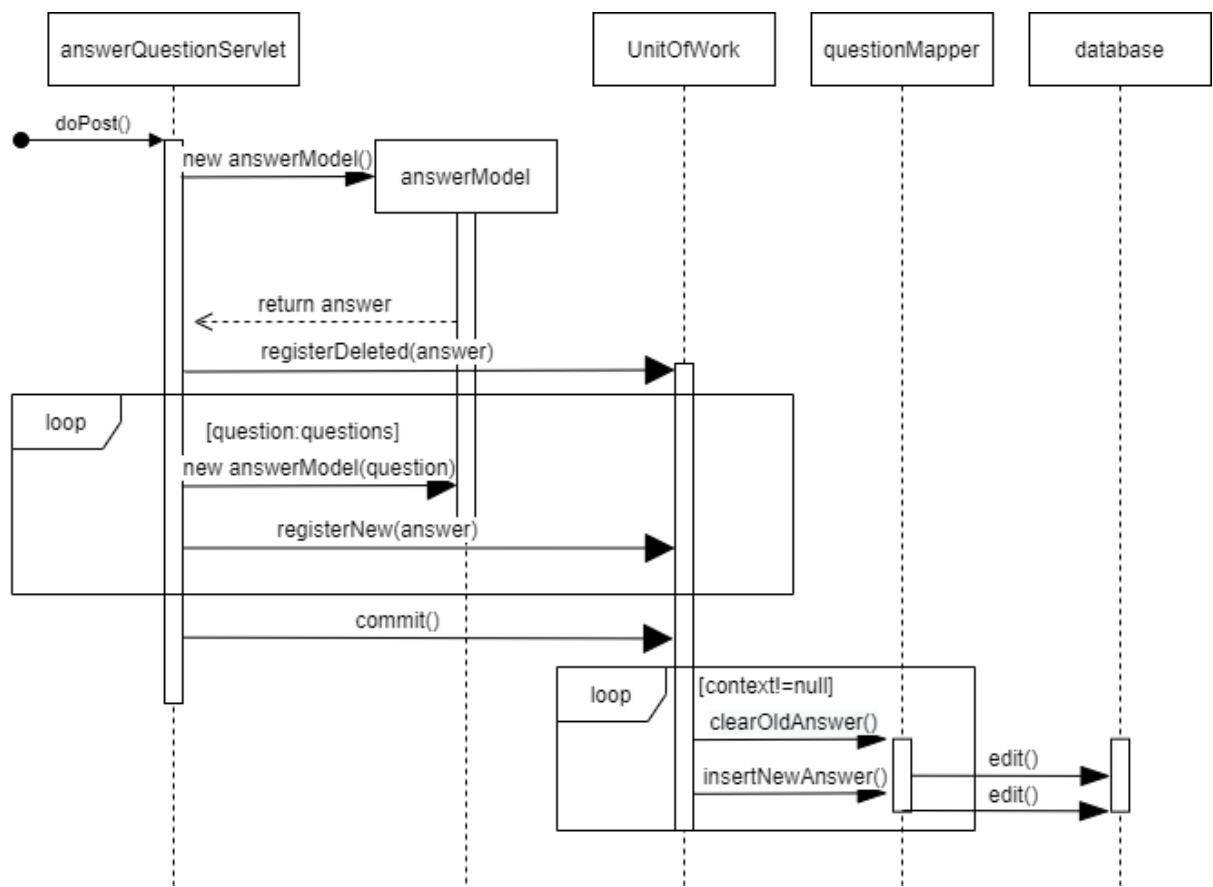


Figure 6. Sequence diagram of answering question (UoW)

6.1.4.2 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 Admin or HCP fetch try to fetch users from database, they just get a minimized userModel which only contains the email, but when they try to get the user’s data, the corresponding userModel loads the data by the email. This can drastically reduce the time it takes for the initial userModel creation time.

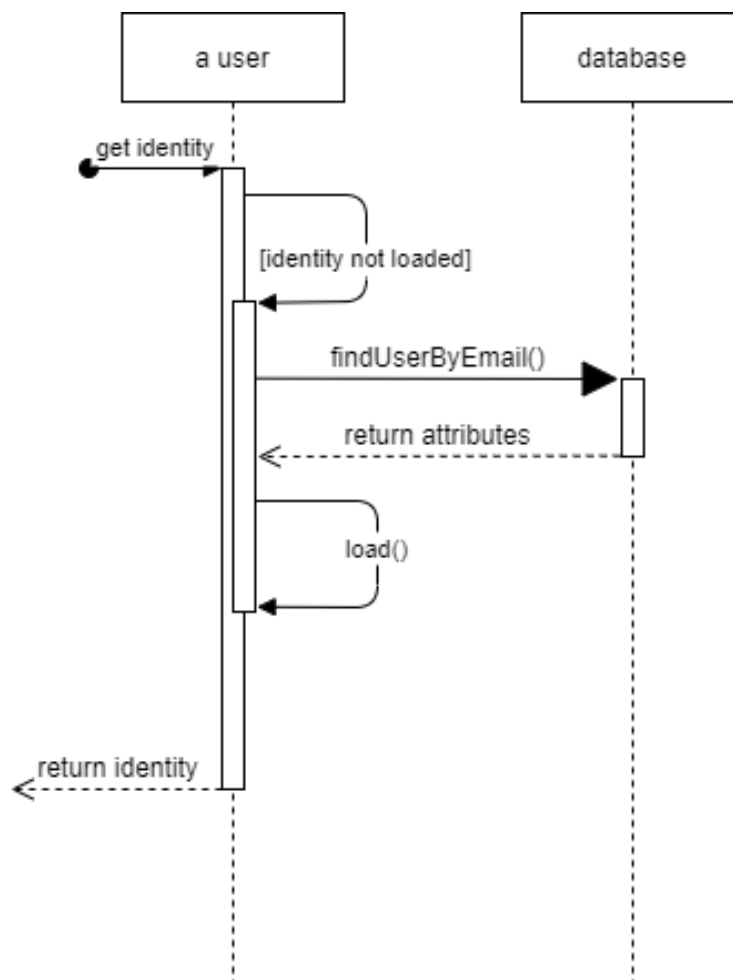


Figure 7. An example sequence diagram for lazy load

6.1.5 O2R Structural patterns

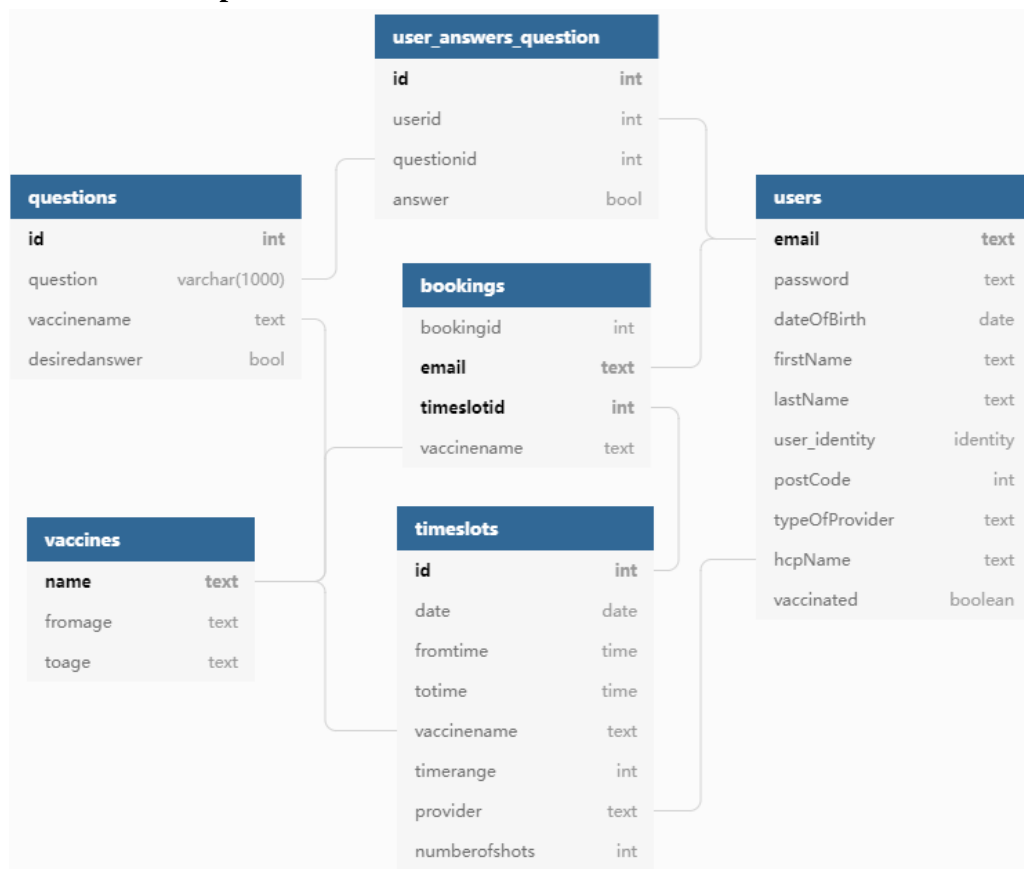


Figure 8. Database architecture

6.1.5.1 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 and being an identifier variable in each class object.

6.1.5.2 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, and a timeSlotModel object would contains a reference to vaccineModel object.

6.1.5.3 Association table mapping

There is a many-to-many relationship in our designing, which is that a user may answer multiple questions, and a question can be answered by multiple users as well. The table user_answers_question is designed to record the association between users and questions, users' answers are stored at the same time.

6.1.5.4 Embedded value

In our architecture design, it is noticed that table timeslots contain a time range attribute. It does not make sense to create a separate time range table for only date, so they are embedded into the timeslot's attributes. Correspondingly, the time range is implemented as a *timeRange* class object owned by the *timeSlotModel* class.

6.1.5.5 Single table inheritance

There is an abstract *userModel* class for the users table in database, and three different *recipientModel*, *adminModel*, and *hcpModel* classes inheritance from the *userModel*. Each of them represents one of the three types of users: recipient, admin, and health care provider.

Correspondingly, three different data mappers are used for each type of user, while all of them implement a *UserInterface*.

One major advantage is that no table joins, or multiple data queries are required to retrieve an instance. But at mean time, it also causes an increase in space complexity because there are some fields left NULL due to different user types.

6.1.6 Authentication and Authorization

Apache Shiro library is used for both authentication and authorization, the main reason is that it is simple but efficient.

6.1.6.1 Authentication

Shiro uses JDBC Realm to access user account information in our PostgreSQL database, and authenticate users accordingly. If an unauthenticated user tries to access any webpages, the user would be redirected to the `index.jsp` page for register and login.

6.1.6.2 Authorization

Shiro is not only granted to access user email and passwords, but also capable of reading user identity from the database. For public resources, the user only needs to be authenticated. And for limited pages, only authenticated user with appropriate identity can access.

In addition, passwords are hashed using hash methods provided by Shiro, and no explicit password is stored in database.

6.1.7 Concurrency

In this section, all possible concurrency issue would be raised, relative design rationales, pattern utilizations, and testing solutions would also be provided.

6.1.7.1 General Notes on Testing Strategy

To test for vulnerabilities in our concurrency implementation we decide to use Apache JMeter. JMeter was chosen because running parallel test scripts is a core inbuilt feature and is perfect for mimicking scenarios that involve multiple users. With this we can confidently replicate many of the scenarios listed below. JMeter is also not coupled to the code, so we avoid adding additional packages and dependencies to our build file. For each of the issues we've identified below, an equivalent script is created in JMeter. We will demonstrate the end results in our database with and without concurrency patterns to show that they are working as intended, and that there is no unexpected behaviour.

6.1.7.2 Issue 1

Description	Multiple users request booking on the same timeslot.
Normal transaction	Query (update) to database, find the timeslot through its ID, examine if the remaining booking slot is larger than zero, and deduct one for a success booking. Then a new row in booking table would be inserted.
Issue	When there is only one left for the selected timeslot, the second request may read the remaining as one before the one being deducted by the first request.
Pattern	Pessimistic pattern (lock manager) – Exclusive read lock
Reason	In the optimistic way, it would be huge frustrations when several users booking at the same time but only one or few users get successful, due to the version number checking commit 1 query only after read.
Implementation	<p>In bookServlet.java</p> <p>A lock on target timeslot is acquired when the first request is under processing, and the second request would be on hold (wait) until first transaction is finished and lock gets released.</p> <p>It is an exclusive read lock accordingly because either read and write are not expected to happen during examine remaining slot and deduct by one if true.</p>

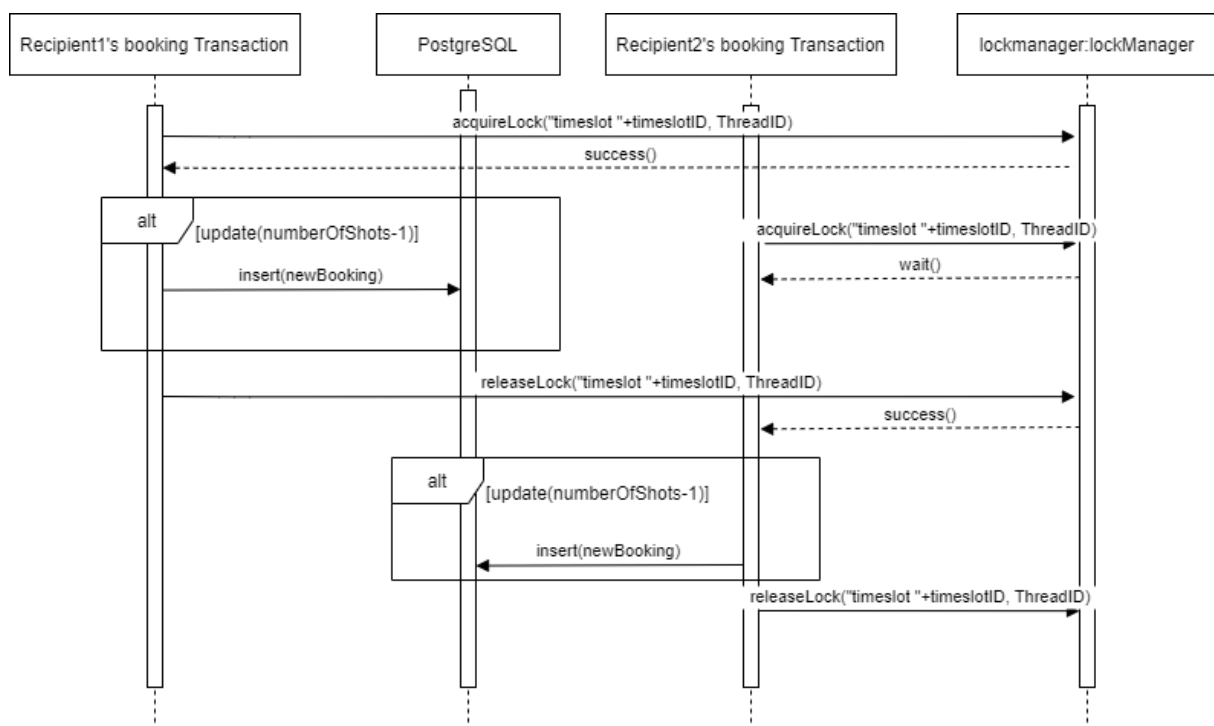


Figure 9. Sequence diagram for issue 1

Testing Strategy

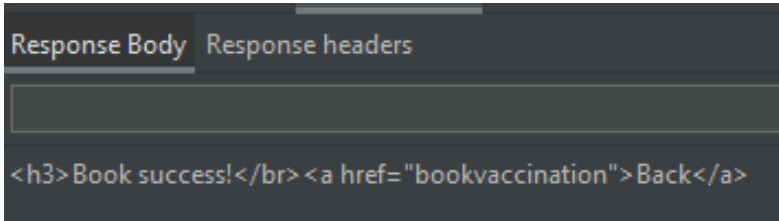
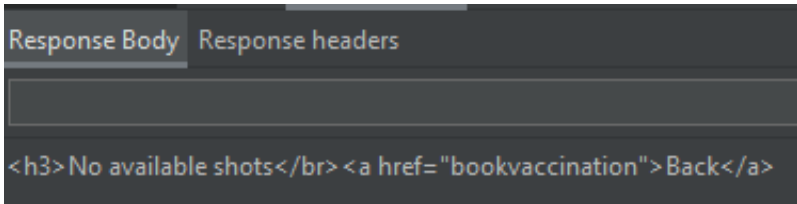

With Apache JMeter we started by configuring the testing path for a **single** user. That testing path is:

- User logs into the application by sending their username and password
- User is redirected to the main page
- User sends a booking request for a specific timeslot.

It is also worth noting that JMeter can store the session cookie that is needed for Authentication.

Using JMeter's built in thread function, we can run this testing path as many times as we want in parallel. We decided to run 2 threads simultaneously. Each of these threads can be configured to login with a different set of credentials, but otherwise perform the same path.

Testing Outcome and Discussion

Name	Test 1
File	The source file for this test is located in the repository under <i>testfiles/recipients-book-same-timeslot-local.jmx</i>
Description	Two (we can have more users in test) users attempt to book a timeslot at the same time
Success Criteria	<ol style="list-style-type: none"> 1. One user successfully books the timeslot and is informed of this. 2. One user is unsuccessful in booking the timeslot and is informed of this. 3. Database has one new booking 4. The timeslot in the database is updated to have no more shots available.
Outcomes	<ol style="list-style-type: none"> 1. One user successfully books the timeslot and is informed of this.  <p>Figure: user 1 receives success confirmation</p> 2. One user is unsuccessful in booking the timeslot and is informed of this.  <p>Figure: user 2 receives failure confirmation</p> 3. Database has one new booking  <p>Figure: single new booking is recorded in DB</p>

4. The timeslot in the database is updated to have no more shots available.

id [PK] integer	date date	fromtime time without time zone	totime time without time zone	provider text	numberofshots integer	vaccinename text
1	2021-10-30	11:11:00	13:13:00	Allenhall	3	AstraZeneca
2	2021-10-29	11:11:00	13:13:00	Allenhall	1	AstraZeneca

id [PK] integer	date date	fromtime time without time zone	totime time without time zone	provider text	numberofshots integer	vaccinename text
1	2021-10-30	11:11:00	13:13:00	Allenhall	3	AstraZeneca
2	2021-10-29	11:11:00	13:13:00	Allenhall	0	AstraZeneca

Figure: Number of shots (highlighted) attribute in the timeslot is decremented by one

Discussion

This test shows that the functionality of the locks is sound. We only ran two concurrent threads, but the system should exhibit the same behaviour for any number. Because we're running strict locks, we can be confident that the implementation is sound. Also, we feel like increasing the number of threads moves further from real life scenarios and is not that valuable. If it is required, we can have more users in week 12's demo, but there would make result unstable.

6.1.7.3 Issue 2

Description	Multiple workers from a same HCP adding the same timeslot through the same account.
Normal transaction	Query (select) to see if there is existing timeslot sharing the same requesting time range, and query (insert) the new timeslot if not.
Issue	When the first request passes existence checking, the second request may also get passed before the timeslot of first requested being inserted.
Pattern	Pessimistic pattern (lock manager) – Exclusive write lock
Reason	Using optimistic pattern is reasonable with good liveness, but the question is other workers' work would loss even if they are not requesting to create the timeslot with the same time range.
Implementation	<p>In addTimeSlotServlet.java</p> <p>A lock on requesting HCP is acquired when the first request is under processing, and the second request would be on hold (wait) until first transaction is finished and lock gets released. Therefore, an HCP account can only process one timeslot at the same time, and all work done would not be lost as well.</p> <p>Notice that reading HCP's account information is still allowed (e.g., Admin can still view all users including HCP's details).</p>

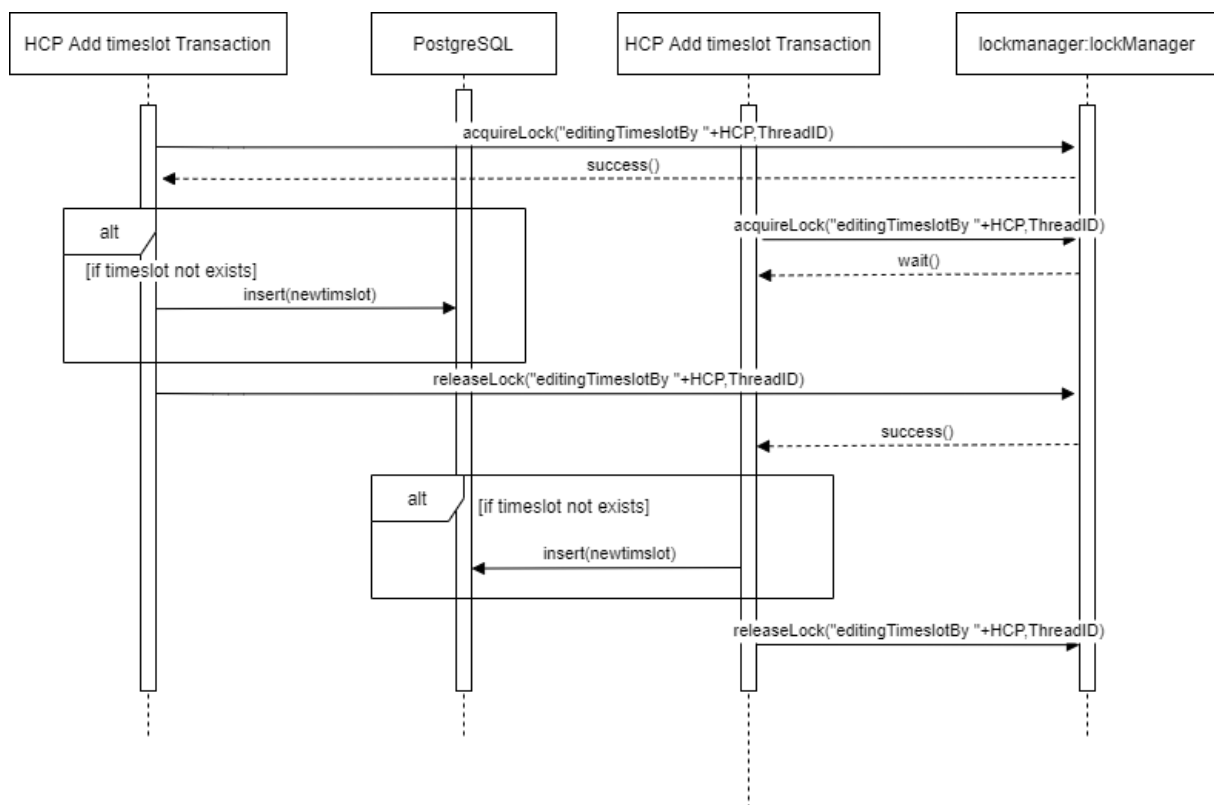


Figure 10. Sequence diagram for issue 2

Testing Strategy

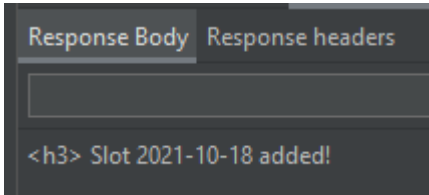
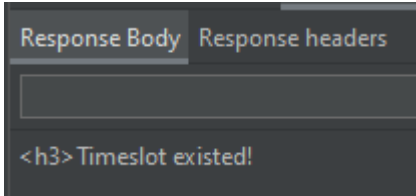
With Apache JMeter we started by configuring the testing path for a **single** Healthcare Provider. That testing path is:

- Provider logs into the application by sending their username and password
- Provider is redirected to the main page
- Provider attempts to create a timeslot at a specific time

Using JMeter's built in thread function, we can run this testing path as many times as we want in parallel. We decided to run 2 threads simultaneously. Each of these threads log in using the exact same credentials. This is to mimic multiple users logging into the one Healthcare Provider account.

It is also worth noting that JMeter can store the session cookie that is needed for Authentication.

Testing Outcome and Discussion

Name	Test 2
File	<i>Testfiles/multiple-hcp-create-same-timeslot.jmx</i>
Description	Multiple workers are logged into the same Healthcare Provider account and attempt to create the same timeslot.
Success Criteria	<ol style="list-style-type: none"> 1. One worker successfully creates the timeslot and is informed of this. 2. All other workers are unsuccessful in creating the timeslot and are informed of this. 3. Database has only one new timeslot
Outcomes	<ol style="list-style-type: none"> 1. One worker successfully creates the timeslot and is informed of this.  <p>Figure: user 1 receives success confirmation</p> 2. All other workers are unsuccessful in creating the timeslot and is informed of this.  <p>Figure: User 2 receives failure confirmation</p> 3. Database has only one new timeslot

	<table><tr><th>id</th><th>date</th><th>fromtime</th><th>totime</th><th>provider</th><th>numberofshots</th><th>vaccinename</th></tr><tr><td>[PK] integer</td><td>date</td><td>time without time zone</td><td>time without time zone</td><td>text</td><td>integer</td><td>text</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>4</td><td>2021-10-18</td><td>14:00:00</td><td>15:00:00</td><td>Allenhall</td><td>2</td><td>AstraZeneca</td></tr></table> <p>Figure: Timeslot table before and after test is run. Note that only one new timeslot entry is created.</p>	id	date	fromtime	totime	provider	numberofshots	vaccinename	[PK] integer	date	time without time zone	time without time zone	text	integer	text															1	4	2021-10-18	14:00:00	15:00:00	Allenhall	2	AstraZeneca
id	date	fromtime	totime	provider	numberofshots	vaccinename																															
[PK] integer	date	time without time zone	time without time zone	text	integer	text																															
1	4	2021-10-18	14:00:00	15:00:00	Allenhall	2	AstraZeneca																														
Discussion	<p>This test mimics a possible scenario that could occur within a healthcare provider’s office. We’re comfortable that the behaviour shown by the system is desired and that there won’t be any unintended consequences in the rare case that this happens.</p>																																				

6.1.7.4 Issue 3 (Not a use case)

Description	HCP deletes a timeslot while a user trying to book it.
Normal transaction	Query (delete) all bookings which uses the target timeslot's ID as foreign key, and then query (delete) the timeslot.
Issue	Between the finish of first query and the starting of second query, a new booking would be created and use the target timeslot's ID as foreign key. Therefore, the deletion of timeslot would return error.
Pattern	Pessimistic pattern (lock manager) – exclusive read lock
Reason	The only object to get locked is clearly the target timeslot, and it make no sense to add a version number for deleting rows. Pessimistic pattern must be used.
Implementation	<p>In deleteServlet.java and bookServlet.java</p> <p>When the deleting request is processed, timeslot for deletion would be locked by lock manager, any incoming booking request with target timeslot ID would be on hold, and later returns recipients an error that timeslot does not exist.</p> <p>It is not expected to see any read or write request during the lock to achieve the implementation described above.</p>

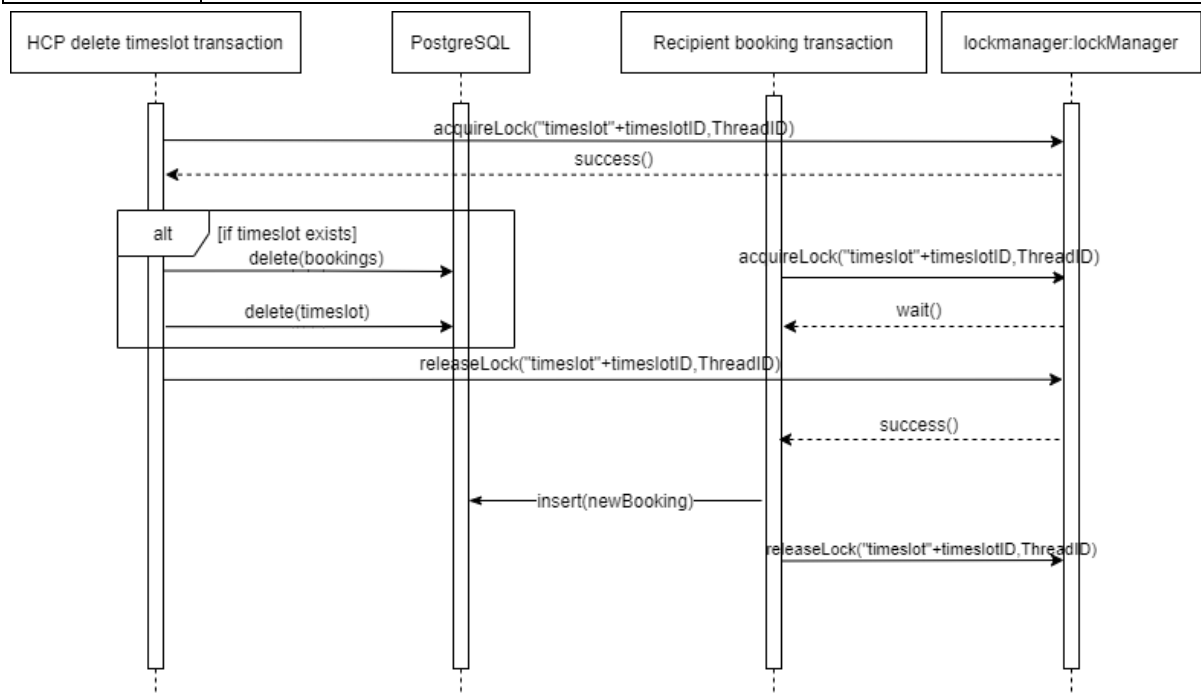


Figure 11. Sequence diagram for issue 3

Testing Strategy

With Apache JMeter we started by configuring two testing paths – one for the healthcare provider and one for the user

Healthcare provider:

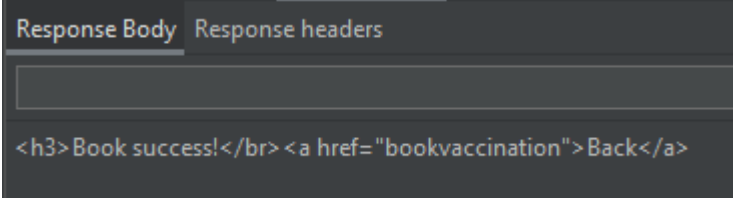
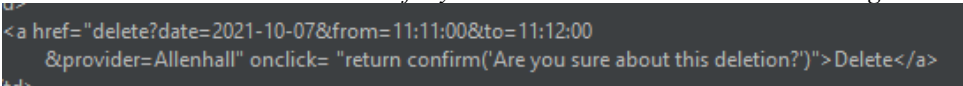
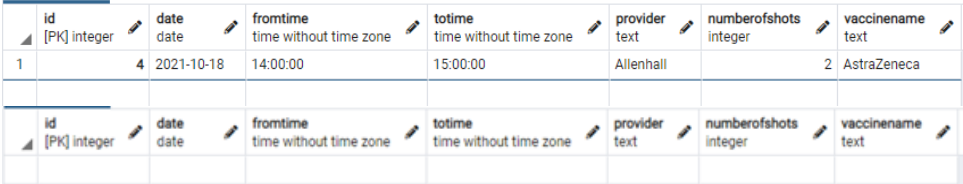
- Provider logs into the application by sending their username and password
- Provider attempts to delete a timeslot

User:

- User logs into the application by sending their username and password
- User attempts to book into the timeslot

We decided to run these two paths at the same time to ensure there was no unexpected behaviour. At the end of execution, the timeslot must be fully deleted from the database.

Testing Outcome and Discussion

Name	Test 3
File	<i>testfiles/delete-timeslot-book-timeslot-same-time.jmx</i>
Description	One (or multiple) user attempt to book a timeslot, while a healthcare provider attempts to delete that timeslot at the same time.
Success Criteria	<ol style="list-style-type: none"> 1. <i>User either successfully adds timeslot or is unsuccessful (this depends on timing)</i> 2. <i>Healthcare Provider successfully deletes timeslot and all related bookings</i> 3. <i>Timeslot and all related bookings are deleted from the database</i>
Outcomes	<ol style="list-style-type: none"> 1. <i>User either successfully adds timeslot or is unsuccessful (this depends on timing)</i> <div data-bbox="416 1160 1152 1357">  </div> <p>Figure: User is shown that their booking is successful</p> 2. <i>Healthcare Provider successfully deletes timeslot and all related bookings</i> <div data-bbox="416 1447 1383 1532">  </div> <p>Figure: Healthcare provider successfully deletes timeslot. This includes the booking that was concurrently made above.</p> 3. <i>Timeslot and all related bookings are deleted from the database</i> <div data-bbox="416 1657 1383 1839">  </div> <p>Figure: timeslot table before and after test. Note that the timeslot has been deleted successfully</p>

	<div> <div>bookingid</div> <div>integer</div> </div> <div> <div>email</div> <div>[PK] text</div> </div> <div> <div>timeslotid</div> <div>[PK] integer</div> </div> <div> <div>vaccinename</div> <div>text</div> </div>
	<p>Figure: booking table after test. Note that the booking made in criteria 1 has been deleted successfully</p>
Discussion	<p>The results of this test case depend on the miniscule differences in thread timing. One scenario is the User gets the lock first and successfully books, then the healthcare provider gets the lock and deletes the timeslot. The alternative is the Healthcare Provider gets the lock first and deletes the timeslot, then the user gets the lock, attempts to book, and is told that the timeslot no longer exists. We're confident that the addition of strong locking to this scenario ensures that these two scenarios are the only ones that can occur.</p>

6.2 Source Code Directories Structure

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

6.3 Libraries and Frameworks

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

Library/Framework Name	GroupID	Artifact
Servlet	javax.servlet	javax.servlet-api V 4.0.1
Database connection	org.postgresql	postgresql V 42.2.14
Authentication & Authorization	org.apache.shiro	shiro-core V 1.6.0 shiro-web V 1.6.0
	org.slf4j	jcl-over-slf4j V 2.0.0-alpha1

6.4 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)
Testing	JMeter	Concurrency testing for part 3 (V 5.4.1)

7. Scenarios

Admin

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 questions, therefore healthcare providers can add timeslots to these vaccines and normal users can book for vaccination.

Health care provider

Healthcare providers receives new vaccines, and they want to add timeslots so that these vaccines can be booked by normal users.

Vaccine recipient

A normal user who is seeking for vaccination registers himself to the system, answers several questions about his health conditions, and the system shows the user that he is eligible for a type of vaccine (Pfizer for instance). Now user looks for all vaccination timeslots and choose one to book. After the vaccination, the user is marked as vaccinated, and he wants to show his vaccination certificate to others.

8. References

Detailed use cases (another deliverable for part 3 in the same folder with this document)