Product Requirements Document
Software Architecture Design Report

[SDA7]
SWEN90007 SM2 2020 Project

Yihan Dong yihdong@student.unimelb.edu.au
John McCleary jmccleary@student.unimelb.edu.au
Benjamin Yi benjaminchen@student.unimelb.edu.au

SCHOOL OF
COMPUTING &
INFORMATION
SYSTEMS

Revision History

*<**students** must use this table to track individuals' contributions to this document.*

Every time you change this document, add the date you changed it, a description of your performed task and your name. For the version, please adopt the following format:

***01.00-D<number>** for draft versions related to Part 1 (any version before the final submission is considered draft). When your document is reviewed and finally ready to be submitted, change it to 01.00. For Part2, start with 02.00-D<number> and so on. This document should always be kept on GitHub>*

Date	Version	Description	Author
23/09	02.00-D01	Make the skeleton of the document and add a class diagram to it.	Yihan Dong.
26/09	02.00-D02	Fill in most of the description of patterns and add several diagrams to it	Yihan Dong.
28/09	02.00-D03	Fill in the content related to the lazy load pattern and improve the description written before.	Yihan Dong.
28/09	02.00-D04	Rewrite the lazy load description and add the rationale of lazy load.	Yihan Dong.
30/09	02.00-D05	Add unit of work pattern description.	Benjamin Yi.

Contents

1. Introduction

1.1 Proposal

This document specifies the SWEN90007 project use cases, describing the flow of events, inputs and outputs of each use case to be implemented.

1.2 Target Users

This document is mainly intended for SWEN90007 students and the teaching team.

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
LMS	Learning Management System
GitHub	Version control website for the project and the documents
Slack	Online communication channel for team members

2. Actors

Actor	Description
Vaccine Recipient	Vaccine recipients are the people who want to take a vaccine and apply for it.
Health Care Providers	Health care providers are doctors or organizations which have the capability to provide vaccines to recipients.
Administrator	The administrator takes the responsibility to manage the data of this system.

3. Class Diagram & Database Model Diagram

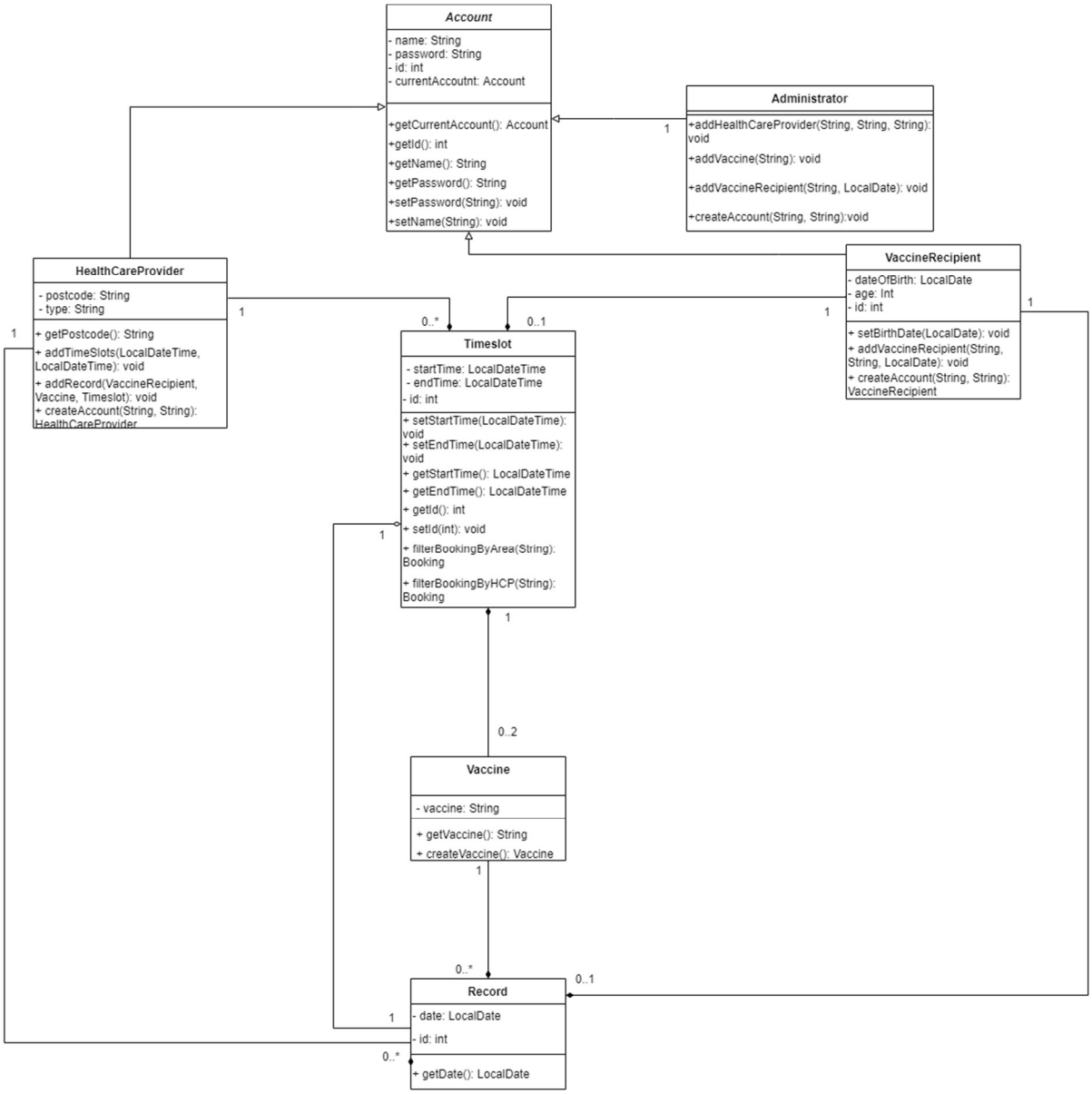


Fig. 1: The class diagram of the vaccination system, excluding the database pattern classes.

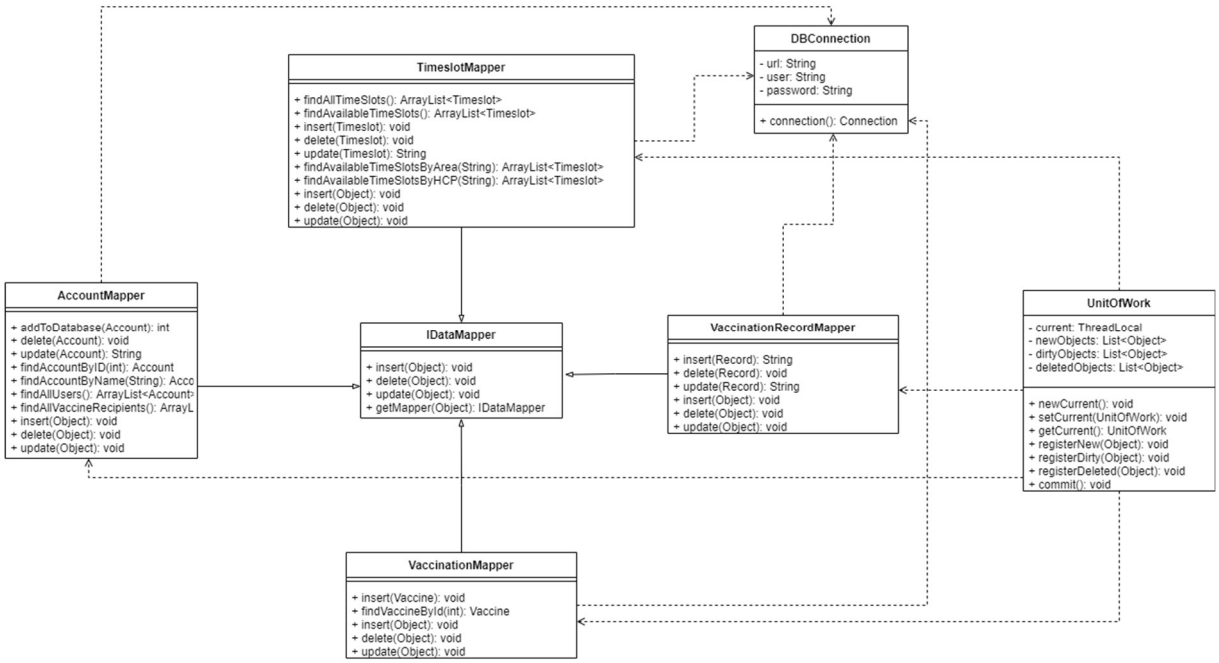


Fig. 2: The class diagram of the database pattern classes.

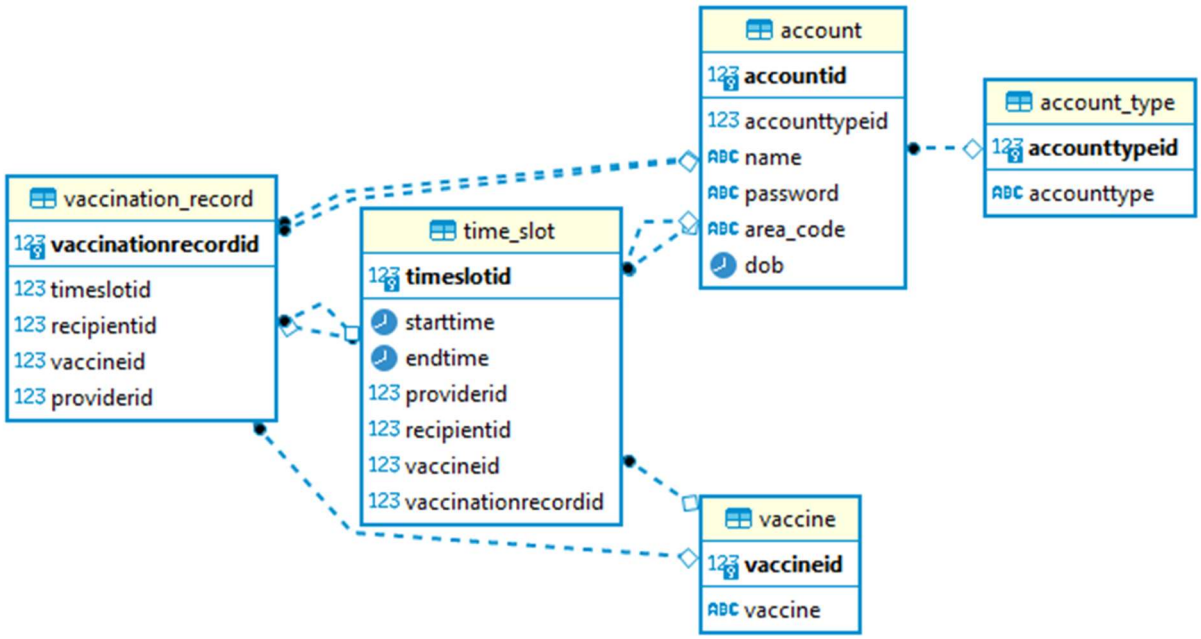


Fig.3: Database model diagram for the system.

4. Description of pattern

4.1 Domain Model

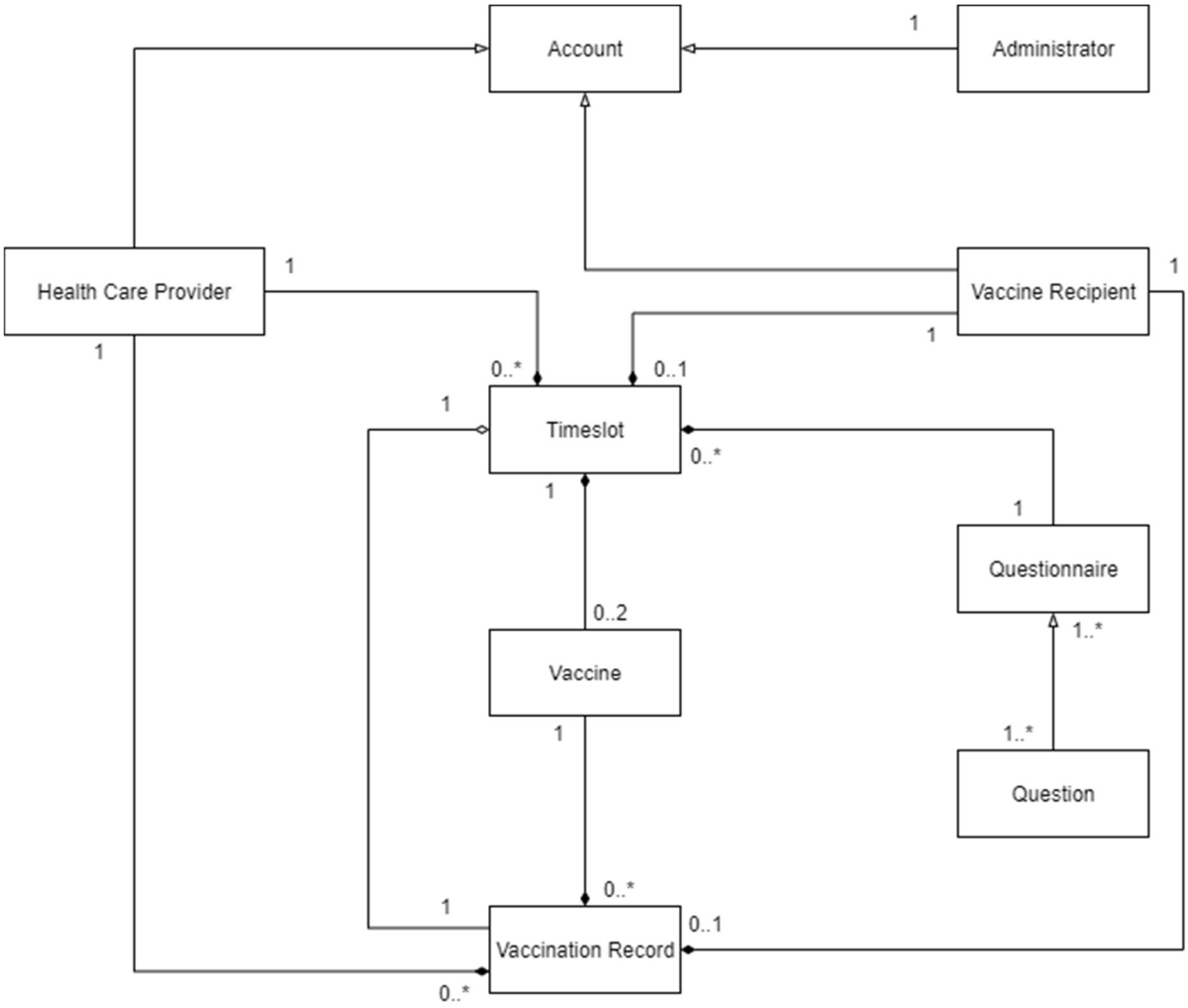


Fig. 4: The domain model diagram of the system

The design and implementation of the vaccination system follow the domain model's principles, as illustrated in the domain model diagram. For each object in the vaccination system, we designed a corresponding class for them.

There are three different roles in the system: the administrator, vaccine recipients and health care providers. They all implement from the class Account, with separate class functions to do their job. For example, the administrator could manage the accounts and view the data in the system; Health care providers could create timeslots and vaccination records for vaccine recipients; Vaccine recipients could book vaccines by answering questionnaires.

Timeslots contain a health care provider, a vaccine recipient, a kind of vaccine and a questionnaire that is used to check if the vaccine recipient is eligible to book vaccination. Similarly, a vaccination record has a health care provider, a vaccine recipient, a kind of vaccine and the timeslot when the vaccine recipient took the vaccine.

4.2 Data Mapper

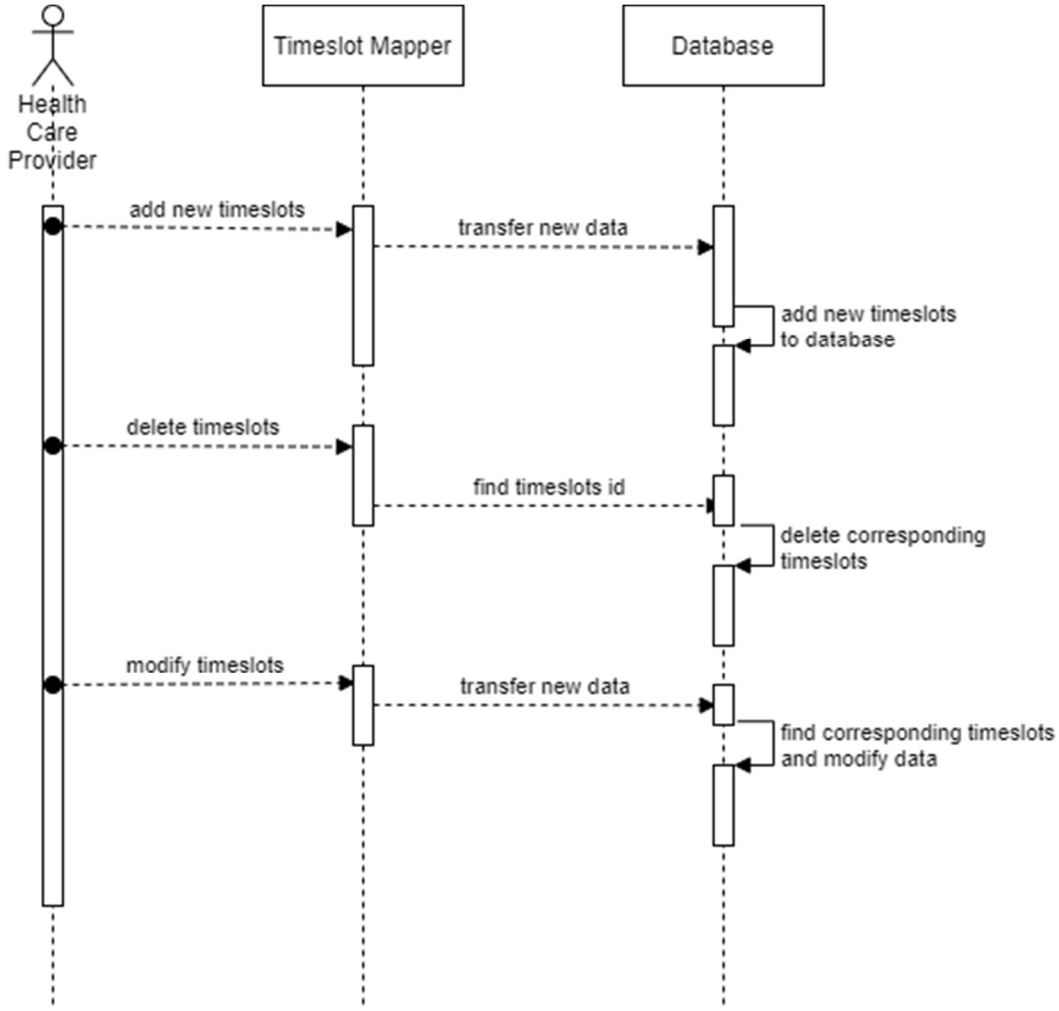


Fig. 5: The sequence diagram illustrates the data mapper pattern.

We use the data mapper pattern to design and implement all the classes related to the modification of the database, including adding data, changing data and deleting data.

Taking the process of a health care provider dealing with timeslots as an example. As illustrated in the figure, the timeslot mapper is responsible for transferring data between two schemas: the object and the database.

When the health care provider wants to add new timeslots, delete some timeslots or modify some timeslots, the health care provider object only calls the function in the timeslot mapper to send the queries to the database and the parameters are all objects. By using the data mapper pattern in this scenario, the health care provider object doesn't need to know the SQL interface code and the database schema.

Similarly, we use this pattern in all of the database classes related to the modification of the data in the database. Since the sequence diagrams are similar to each other, so we only illustrate one process as an example.

4.3 Unit of Work

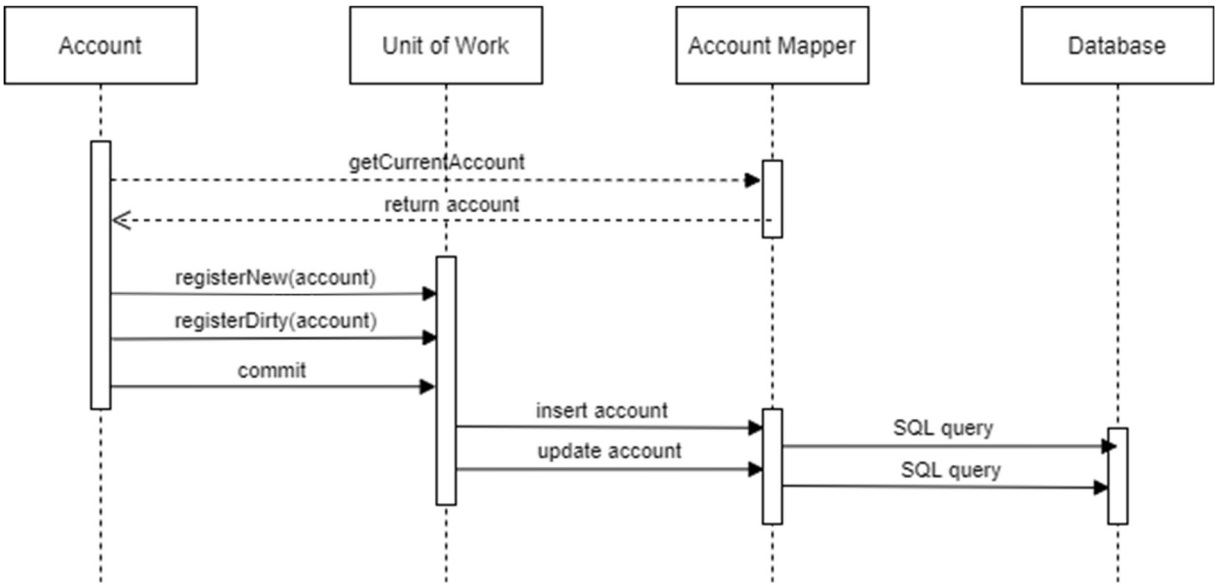


Fig. 6: The unit of work pattern example in creating a new account

When an object wishes to write, update, or delete records from the database, it calls the corresponding data mapper through the Unit of Work class. This is done by registering database objects as new, dirty, or deleted in the Unit of Work class.

The Unit of Work class keeps a record of all the objects involved in a single transaction, and executes all the data mapper calls at the same time. All new objects registered in the Unit of Work class are inserted into the database, all dirty objects are updated, and all deleted objects are deleted.

4.4 Lazy Load

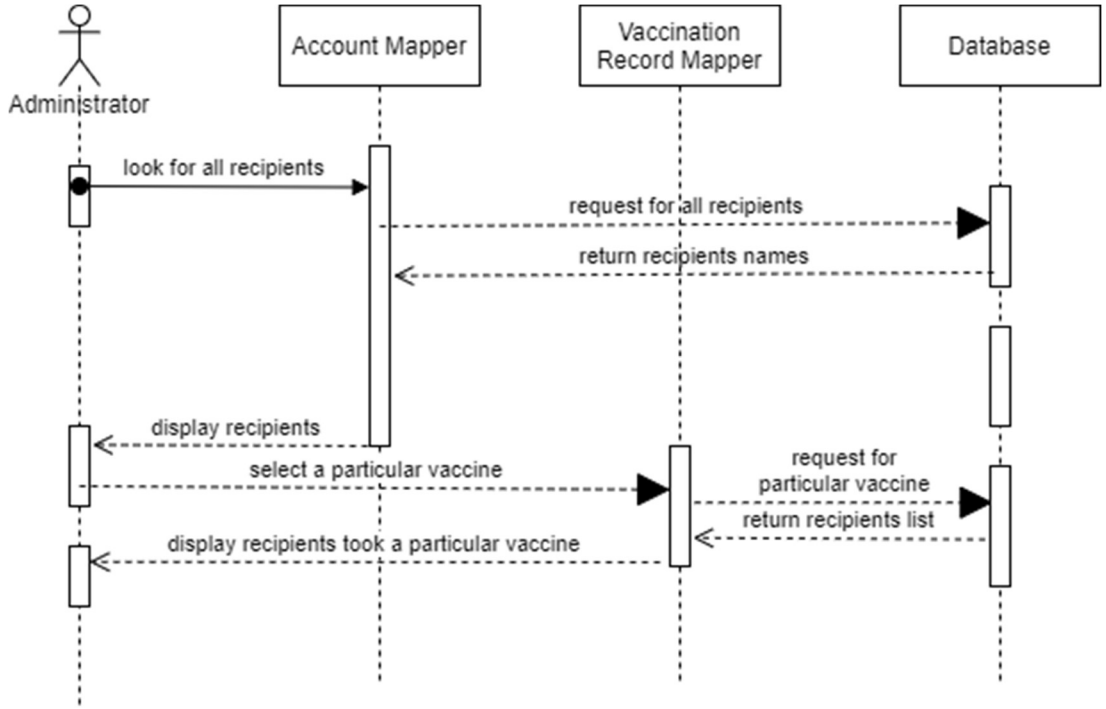


Fig. 7: The sequence diagram of the lazy load pattern in the system.

When the system displays all vaccine recipients to the administrator, each vaccine recipient has a record that displays which vaccine he took. Presenting the vaccine with the recipients is unnecessary when the administrator only wants to view the list of the vaccine recipients. Therefore, the system only stores the recipients' user id associated with each record when the administrator browses all recipients and then calls the vaccination record mapper class to display the details only when the administrator selects a recipient.

4.5 Identify Field

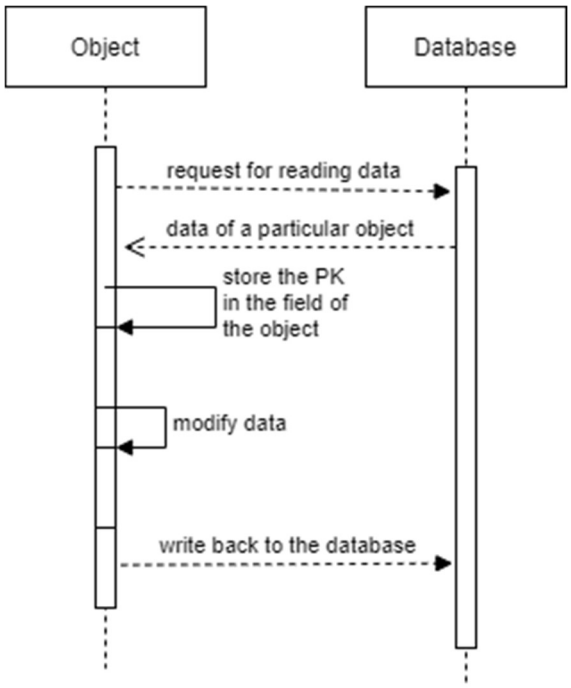


Fig. 8: The sequence diagram of the identify field pattern in the system.

For all the database tables and the corresponding objects in the system, we use the identify field pattern to tie the database table to a particular object when we write the data back. As illustrated in the figure, all the primary keys of the tables would be written to the field of an object when the system read data from the database.

4.6 Foreign Key Mapping

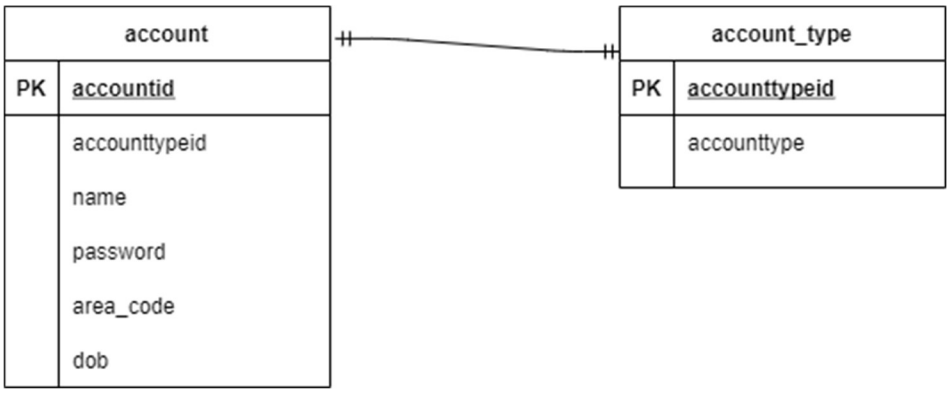


Fig. 9: The database model diagram case 1 for the foreign key mapping pattern.

As illustrated in the figure, we use the foreign key mapping pattern to store all the accounts in the database. The primary key in the table “account_type” is stored in the table “account” as a field as well, and we can look for the category of each account by searching it.

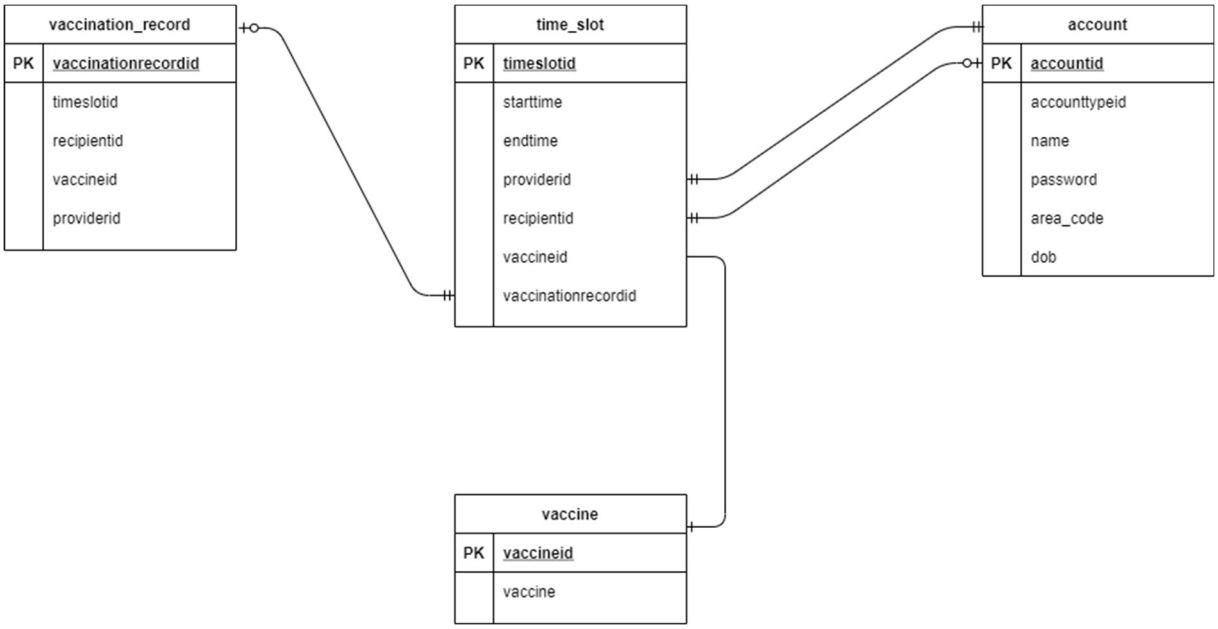


Fig. 9: The database model diagram case 2 for the foreign key mapping pattern.

We also use this pattern in the storage of the timeslots in the database. The “providerid”, the “recipientid” are stored as fields in the table “time_slot”, and they are primary keys in the table “account”. Besides, the fields “vaccineid” and “vaccinationrecordid” in the table “time_slots” are also the primary keys in the table “vaccine” and “vaccination_record” relatively. By using this pattern, we could search for the corresponding health care provider, vaccine recipient, vaccine and vaccination record without building multiple tables.

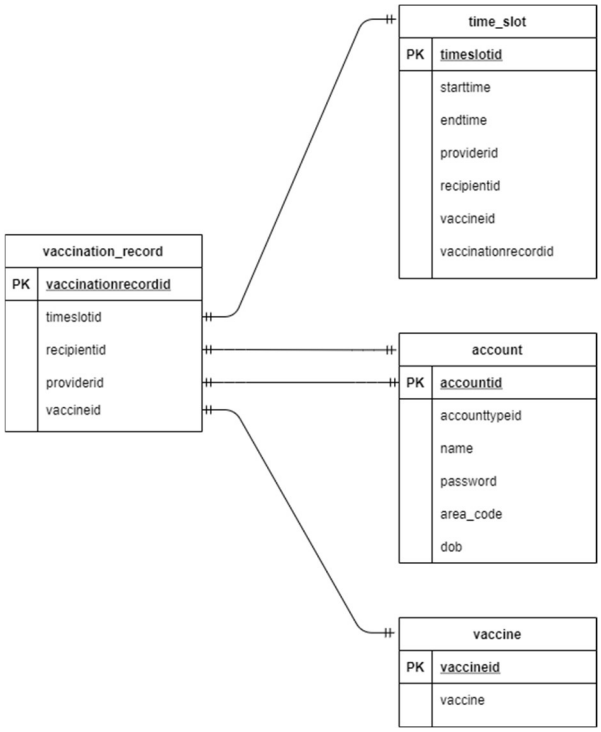


Fig. 9: The database model diagram case 1 for the foreign key mapping pattern.

Besides, we use this pattern in the storage of the vaccination record in the database. The primary keys in the table “time_slot”, “account”, and “vaccine” are all stored as fields in the table “vaccination_record”.

By using this foreign key mapping pattern in storing these data, we can simply simulate the relationship among objects in the database.

4.7 Association table mapping

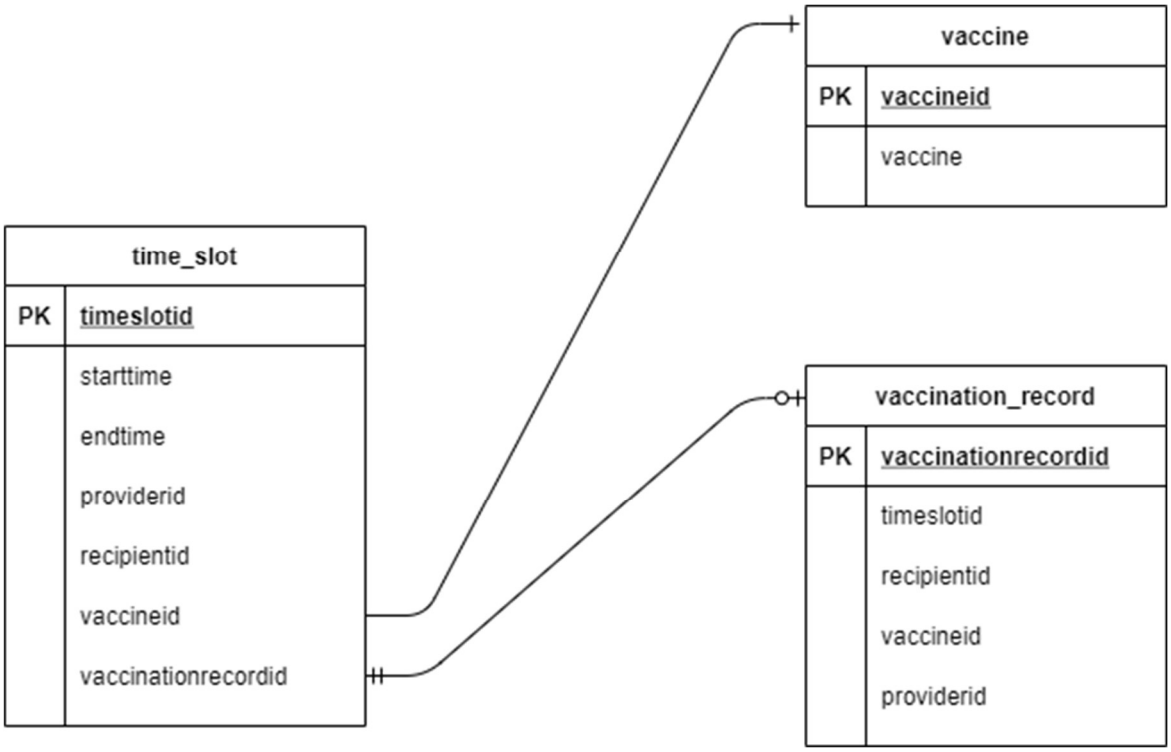
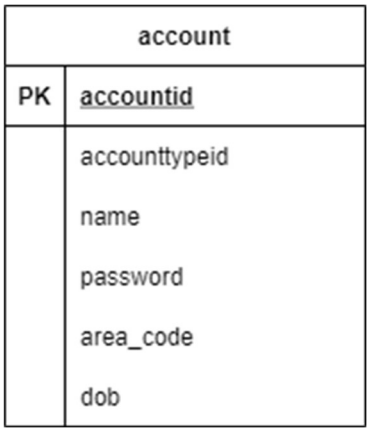


Fig. 9: The database model diagram for the association table mapping pattern.

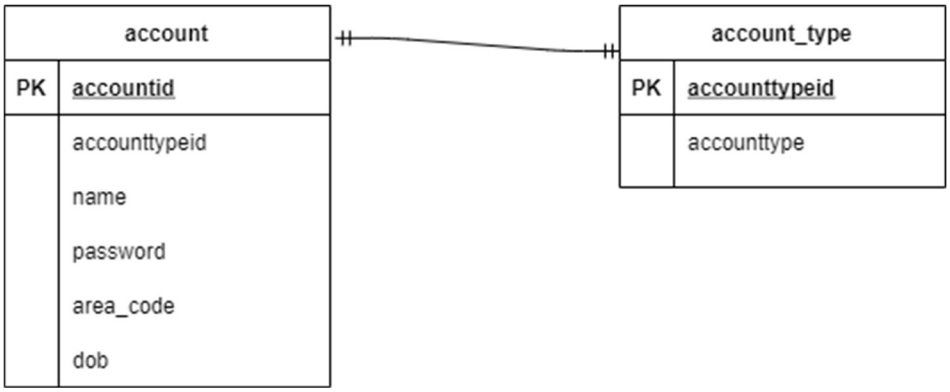
We use the association table mapping pattern in presenting the relationship between the questionnaires and the questions. Since the relationship between these two attributes is many-to-many, as illustrated in the figure, we create another table called “questionnairequestion” that includes two fields that are the primary key of the table “questionnaire” and the table “question” separately to present this relationship. When the system needs to display a questionnaire, this extra table could help to match the questionnaire and its questions.

4.8 Embedded value



We use the embedded value pattern in demonstrating the attributes of the users. For different users, there are different attributes stored in the object, such as the postcode of the health care providers and the date of birth of vaccine recipients. However, for these attributes, creating a table in the database to store them is meaningless to some extent. Therefore, we design the account table to store these attributes of the users.

4.9 Single table inheritance



We use the single table inheritance in presenting the inheritance relationship between three particular roles and the interface account. As illustrated in the figure, all the accounts are stored in a single table called “account”, with a primary key called “accountid”. Then, a field called “accounttypeid” in this table is the primary key of another table called “account_type”. By using this pattern, we don’t need to create multiple tables in the database to present different kinds of users.

4.10 Authentication and Authorization

The Shiro framework is used to design and implement the authentication and authorization pattern in the system. Shiro builds upon the Java servlet MVC model, replacing the login servlet provided a suitable login jsp. Login is validated against the postgresql database, referencing the accounts table. The accounts table contains a username field, a password field hashed using Shiro’s hashers, and an account type field which Shiro uses to grant permissions. Each account type is only allowed access to their own pages, and non-logged in users only have access to the login page.

5. Design Rationale

5.1 Lazy load pattern

We use the lazy load pattern in the progress of presenting the vaccine recipients list to the administrator and the progress of the administrator searches for recipients who took a particular vaccine. In the system, we design and implement mapper pattern classes for every process of data transfer. Besides, each recipient has a record of what kind of vaccine they took before. The administrator has the authority to view all the recipients in the system. When the administrator views the recipient list, loading all the details is unnecessary and it's also a waste of time. Therefore, only when the administrator searches for the recipient who took a specific vaccine, the system would load the vaccination record from the Vaccination Record Mapper class to display on the frontend.

5.2 Unit of work pattern

We use the unit of work pattern when recording transactions to the database. This allows for transactions to be written to the database as a whole, making rollbacks simpler. It also helps with concurrency issues with multiple users writing to the database at the same time. The unit of work pattern is implemented for all data mappers at the domain object level, with objects being registered new, dirty or deleted within their own methods, and transactions being initialized and committed for each HTTP get/post request.