Product Requirements Document

Part 3 Specification

SCHOOL OF
COMPUTING &
INFORMATION
SYSTEMS

Team Member's name	Student ID	Unimelb Usernames	Github Username	Emails
Zhiming Deng	981607	zhimingd	ZhimingDeng5	zhimingd@student.unimelb.edu.au
Gaoli Yi	1048049	gaoliy	GAOLIY	gaoliy@student.unimelb.edu.au
Lingge Guo	1211499	linggeg1	linggeg1	linggeg1@student.unimelb.edu.au
Chutong Wang	1185305	CHUTONGW	CHUTONGW	CHUTONGW@student.unimelb.edu.au

Release tag: SWEN90007_2021_Part3_<Flying Tiger>

Revision History

Date	Version	Description	Author
01/10/2021	03.00-D01	Initial draft	zhiming
04/10/2021	03.00-D02	Review and the second version of the document	chutong
06/10/2021	03.00-D03	Draft document contents	gaoli
08/10/2021	03.00-D04	Add the class diagram	zhiming
10/10/2021	03.00-D05	Update the class diagram	lingge
11/10/2021	03.00-D06	Add descriptions of the patterns concurrency issues	lingge
12/10/2021	03.00-D07	Update descriptions of the patterns concurrency issues	chutong
13/10/2021	03.00-D08	Add design rationale for patterns concurrency issues	gaoli
14/10/2021	03.00-D09	Update design rationale for patterns concurrency issues	zhiming
15/10/2021	03.00-D10	Add pattern implementation	lingge
16/10/2021	03.00-D11	Add testing strategy and outcomes	chutong
17/10/2021	03.00	The final version of the document	gaoli

Heroku App link: https://flying-tiger.herokuapp.com/

Data Sample

We have provided some data for you test, you can also find them in the github-docs/data samples/datasample.txt

administrator ID: 1000 Password:abcd ID: 7 Password:abcd

vaccine recipient ID: 2000 Password:abcd ID: 16 Password:abcd

health care provider ID: 200000000 Password:abcd ID: 200000001 Password:abcd

Contents

Introduction

1.1 Proposal

This document specifies use cases, actors to be implemented, and the system's domain model of the COVID-19 Vaccine Booking and Management System.

1.2 Target Users

This document is mainly intended for vaccine recipients, health care providers and administrators of the system.

1.3 Conventions, terms and abbreviations

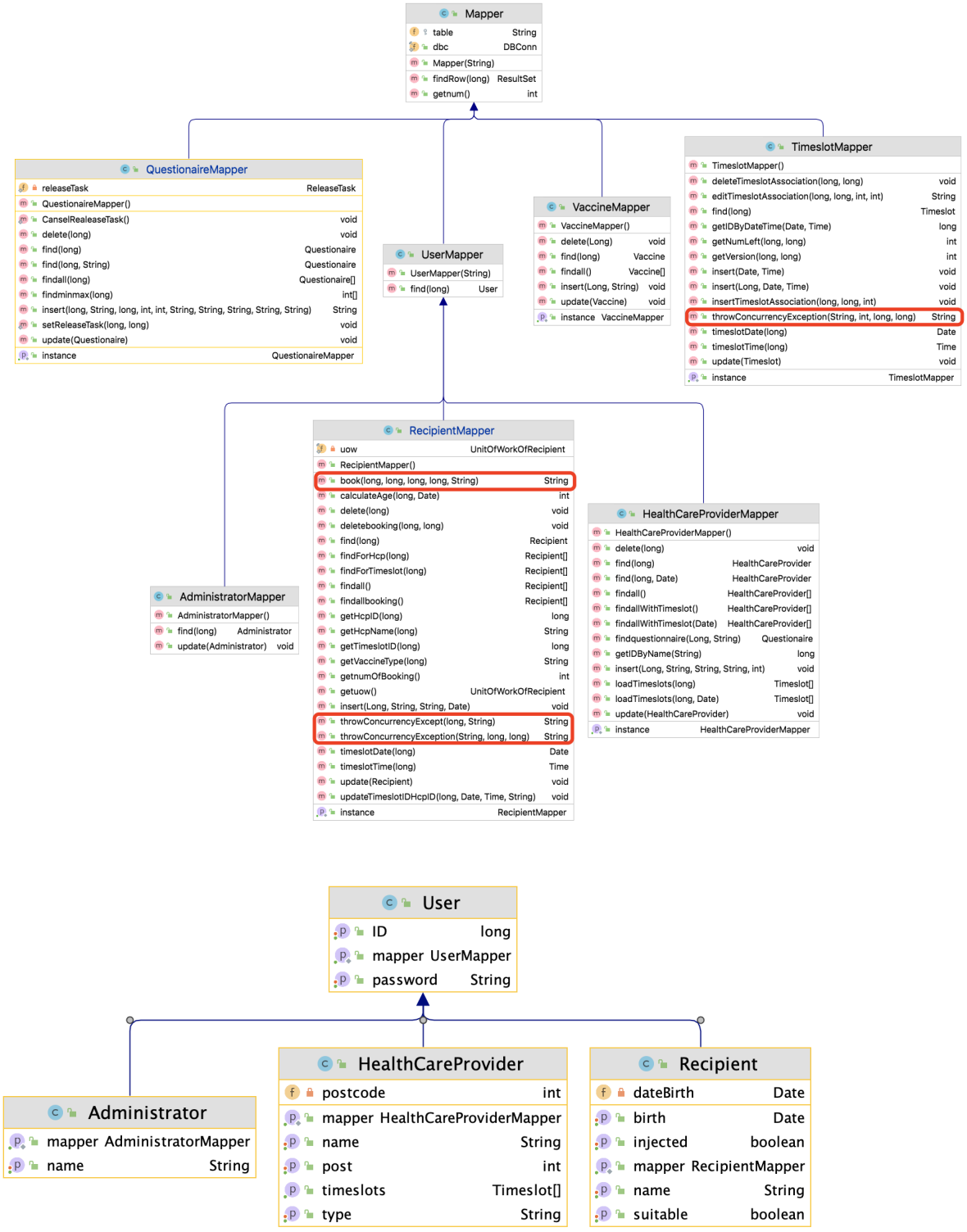
This section explains the concept of some important terms that will be used throughout this document. These terms are detailed alphabetically in the following table.

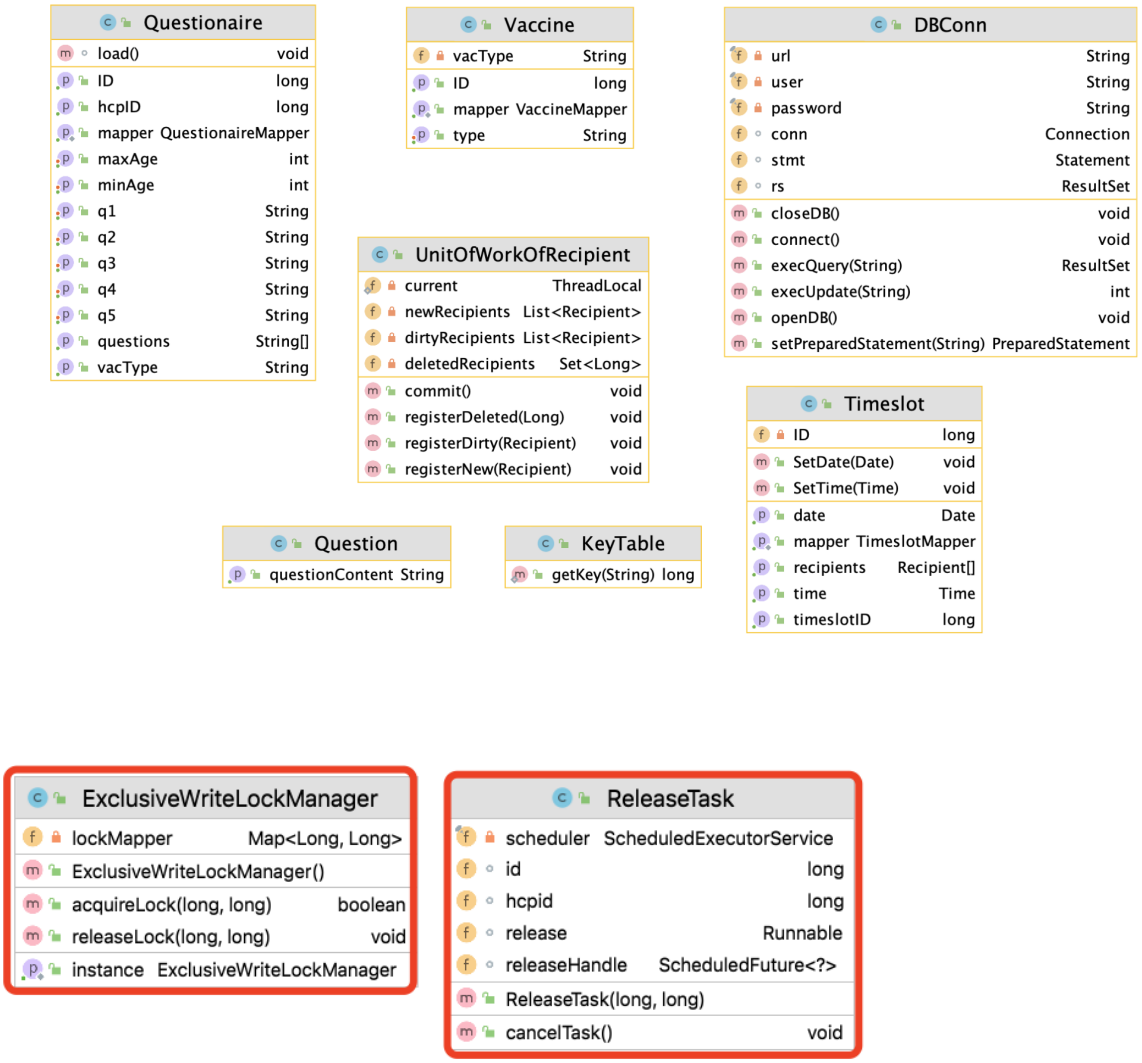
Term	Description
COVID-19	Coronavirus disease 2019, which is an infectious disease caused by Severe Acute Respiratory Syndrome coronavirus type 2 (abbreviation SARS-CoV-2).

Actors

Actor	Description
Administrator	The person who manages the COVID-19 Vaccine Booking and Management System.
Vaccine recipients	People who can make a book on vaccination.
Health care providers	People who provide vaccines, add injection timeslots and confirm the vaccination of a Vaccine recipient.

Class Diagram




Questionaire
load() void
ID long
hcpID long
mapper QuestionaireMapper
maxAge int
minAge int
q1 String
q2 String
q3 String
q4 String
q5 String
questions String[]
vacType String
Vaccine
vacType String
ID long
mapper VaccineMapper
type String
DBConn
url String
user String
password String
conn Connection
stmt Statement
rs ResultSet
closeDB() void
connect() void
execQuery(String) ResultSet
execUpdate(String) int
openDB() void
setPreparedStatement(String) PreparedStatement
UnitOfWorkOfRecipient
current ThreadLocal
newRecipients List<Recipient>
dirtyRecipients List<Recipient>
deletedRecipients Set<Long>
commit() void
registerDeleted(Long) void
registerDirty(Recipient) void
registerNew(Recipient) void
Timeslot
ID long
SetDate(Date) void
SetTime(Time) void
date Date
mapper TimeslotMapper
recipients Recipient[]
time Time
timeslotID long
Question
questionContent String
KeyTable
getKey(String) long
ExclusiveWriteLockManager
lockMapper Map<Long, Long>
ExclusiveWriteLockManager()
acquireLock(long, long) boolean
releaseLock(long, long) void
instance ExclusiveWriteLockManager
ReleaseTask
scheduler ScheduledExecutorService
id long
hcpid long
release Runnable
releaseHandle ScheduledFuture<?>
ReleaseTask(long, long)
cancelTask() void

Patterns Concurrency issues and pattern description

4.1 Concurrency issues about timeslot

4.1.1 Multiple users with the same health care provider account edit the same timeslot's capacity

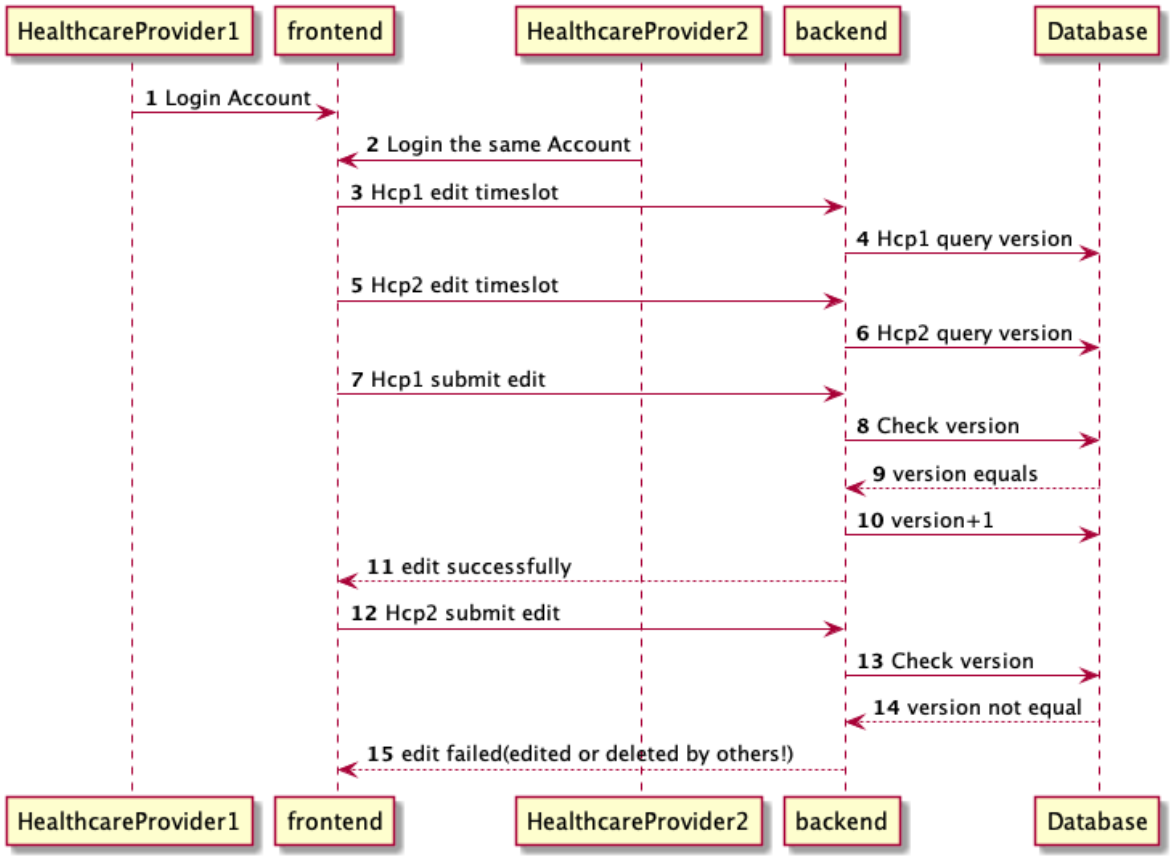
Description:
An user of a health care provider account edits a timeslot's capacity and then submit meanwhile another user with the same account does the same operations. (Note that in our system, a health care provider could only edit the capacity of its own timeslots , users logging in with different health care provider accounts will not trigger concurrency issues)

The expected output:
Update submitted first will be successfully processed, while others failed to process.

Choice of pattern:
Optimistic offline lock

Implementation details:
In the timeslot and health care provider association table,there are 'numLeft' column and 'version' column. The former describes how many seats are left and the latter is used for optimistic lock. Once a user successfully updates the 'numLeft' of a row , the version will be added by 1. And when an update submitted, it will compare the update version with the version saved in the corresponding row in database. Once they match, the update will be processed. If they do not match, the system will further find out whether the version saved in database is different or it is not existed, then return alert message of 'have edited!' or 'have deleted!' to the front-end.

Sequence diagram:



4.1.2 Recipient book a timeslot while its capacity set to 0 or it has been deleted

Description:
This could be divided into 3 possibilities:

1.More than one vaccine recipient book a timeslot with capacity which smaller than the number of recipients who book concurrently;

2.A recipient books a timeslot while its capacity is set to 0 by its health care provider.

3.A recipient books a timeslot when it is being deleted

The expected output:
For All situations: process books one by one; a book will be successful if the timeslot capacity is larger than 0 when the book is submitted.

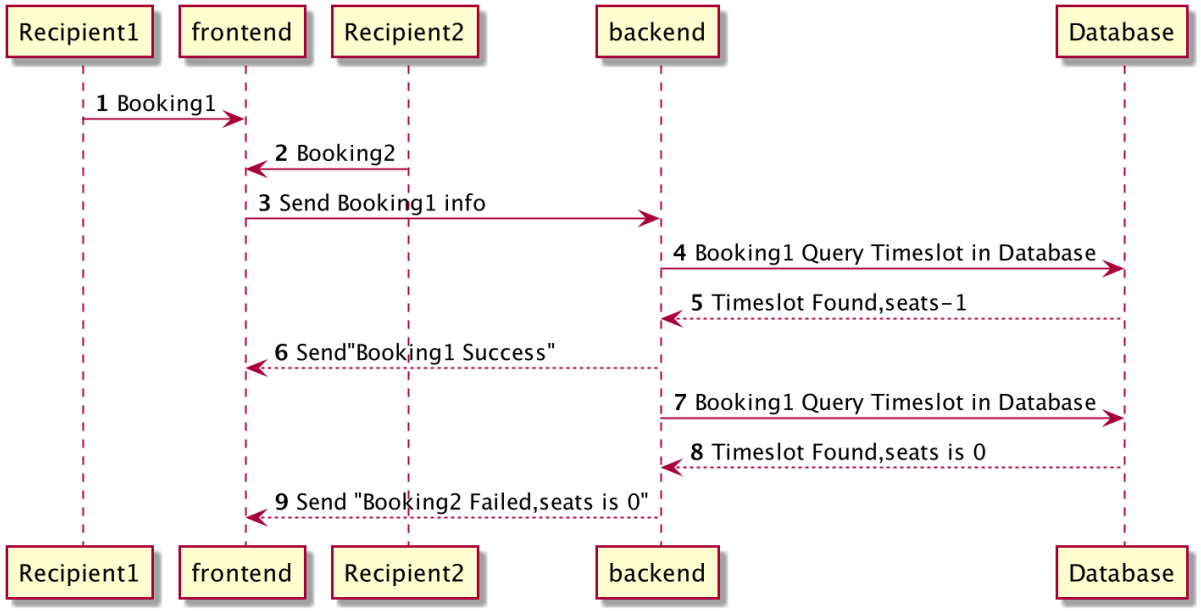
Choice of pattern:
Optimistic offline lock

Implementation details:
After a recipient answers the questionnaire and submits, back-end code will check the capacity of the selected timeslot via database query. If that is 0, an alert massage notifying “no enough seat” will be

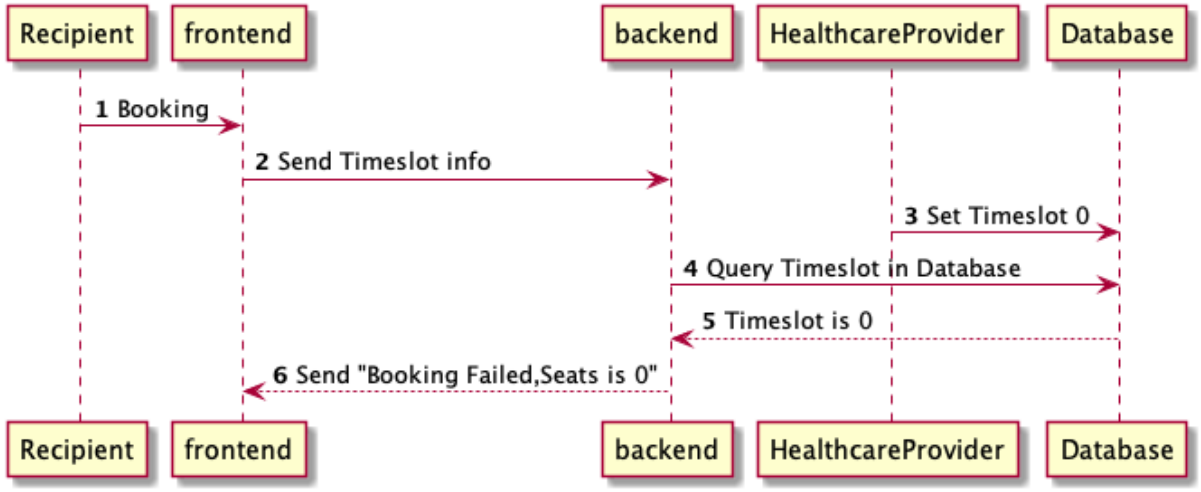
sent to the front end. If that timeslot does not exist, an alert message notifying “has been deleted” will be sent to the front end.

Sequence diagram:

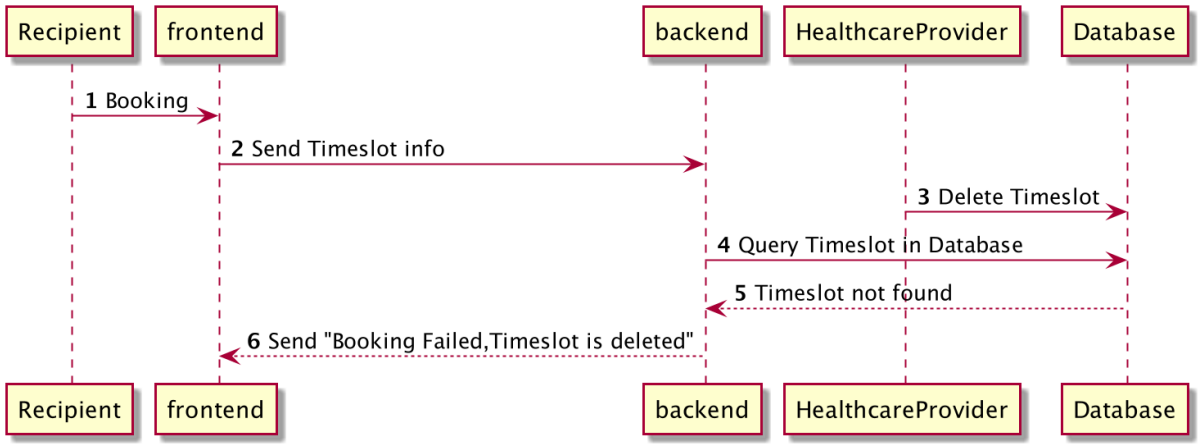
This sequence diagram corresponds to Case 2 in description. Case 1 and Case 3 has the similar sequence as Case 2, you can view the differences among them in the description above.



Case 1: Two recipients booking 1 seats timeslot



Case 2: Set timeslot 0 when Booking



Case 3: Delete timeslot when Booking

4.2 Concurrency issues about vaccine type

4.2.1 Administrator edit or delete a vaccine type while the Recipient is booking it.

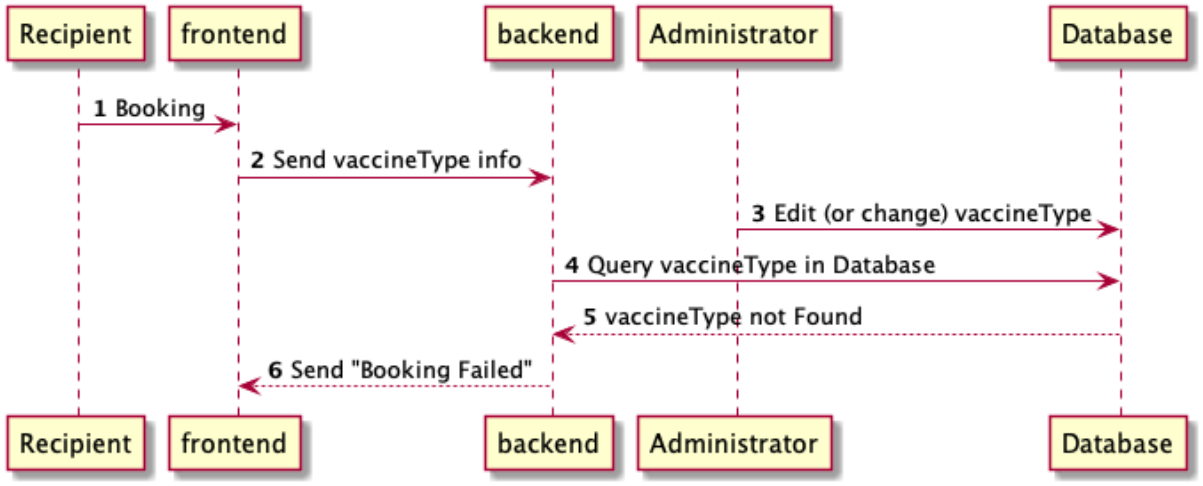
Description:
When the recipient is booking a vaccine, the administrator makes changes or deletes the vaccine type. (After the recipient visits the booking vaccine interface and selects a certain vaccine type, then the administrator makes changes to the system before the booking is successful).

The expected output:
the administrator edit or delete the vaccine type successfully, but the recipient fails booking. the booking page shows “the vaccine type is changed”(when the type is edited) and “the vaccine type is deleted”(when the type is deleted by the administrator).

Choice of pattern:
Optimistic offline lock

Implementation details:
In this system, the administrator has the highest operating authority, so we allow the administrator to make changes at any time. In addition, we manage the booking operations of recipients. When they submit the booking information, the backend receives the type of vaccine sent by the front end and performs query operations in the database. If the same type of vaccine is found, it proves that the administrator has not made any changes. The system can proceed to the next step of booking. If no vaccine of the same type is found, the system throws an exception and judges whether the vaccine type has been changed or deleted (according to the id corresponding to the vaccine type). Finally, send an alert message to the front-end page：“the vaccine type is changed”(when the type is edited) and “the vaccine type is deleted”(when the type is deleted by the administrator).

Sequence diagram:



4.2.2 Administrator edit or delete a vaccine type while Health care provider is Adding questionnaire for it.

Description:

When the healthcare provider is adding a questionnaire for a vaccine, the administrator makes changes or deletes the vaccine type. (After the healthcare provider visits the Add questionnaire interface and selects a certain vaccine type, then the administrator makes changes to the vaccine type before the operation by the healthcare provider is successful).

The expected output:

the administrator edit or delete the vaccine type successfully, but the healthcare provider fails to add the questionnaire. And the Adding page shows “the vaccine type is changed or deleted”.

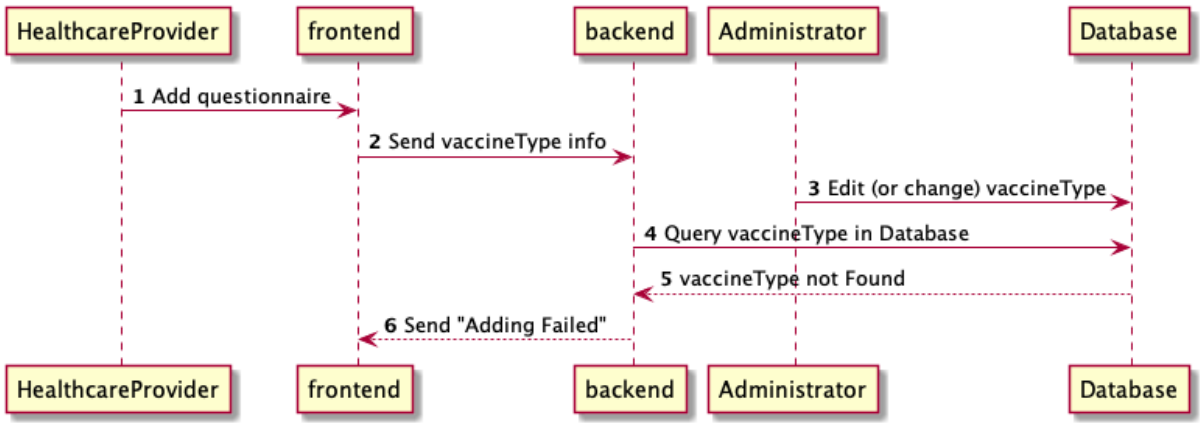
Choice of pattern:

Optimistic offline lock

Implementation details:

In this system, the administrator has the highest operating authority, so we allow the administrator to make changes at any time. In addition, we manage the operations of the healthcare provider. When they submit the questionnaire adding information, the back-end receives the type of vaccine sent by the front-end and performs query operations in the database. If the same type of vaccine is found, it proves that the administrator has not made any changes, and the system returns a message indicating that the addition was successful. If no vaccine of the same type is found, the system throws an exception and sends a prompt message to the front-end page: "Adding failed, vaccine type has been changed or deleted".

Sequence diagram:



4.3 Concurrency issues about questionnaire

Description:

When a healthcare provider is editing the questionnaire about some specific vaccine type, another user that logs in to the same healthcare provider account edit the same questionnaire at the same time.

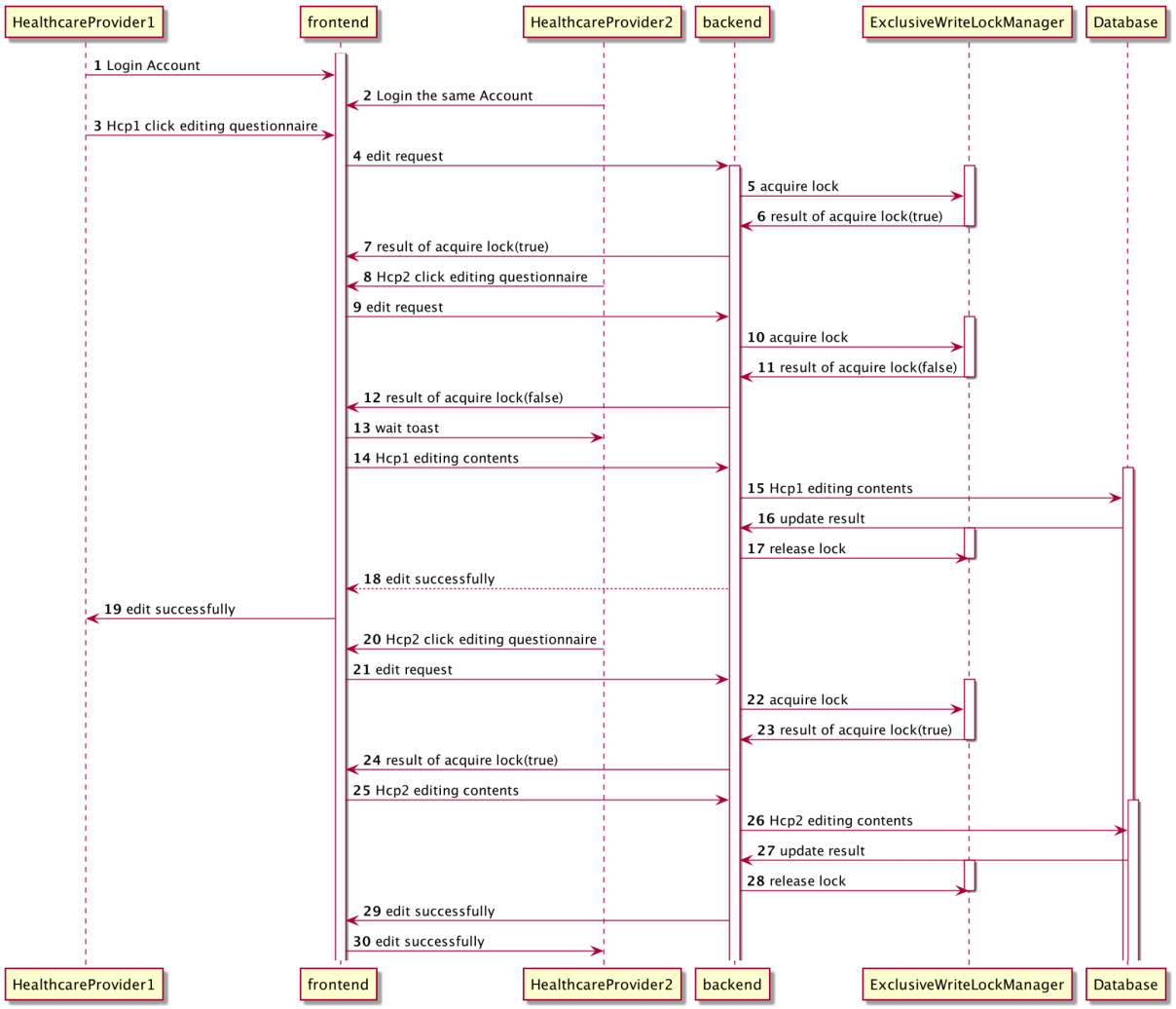
The expected output:
The one who acquires the lock after clicking editing the questionnaire can continue processing, while others with the same account failed to enter the editing page until the one finishes editing by clicking submit or going back.

Choice of pattern:
Pessimistic offline lock

Implementation details:
To fix this concurrency issue, we use an exclusive write lock, which allows other processes to read, like one recipient getting a questionnaire to fill in, while the process has the lock. It requires processes to acquire the lock to write data. Here, we use a ConcurrentHashMap to implement it which maps the current questionnaire ID which is being edited to the health care provider ID which's doing it. When someone clicks editing the questionnaire, it will acquire the lock by putting one map into ConcurrentHashMap when there's no lock about this specific questionnaire. After that, another one couldn't acquire the lock as there's already one in the ConcurrentHashMap until the one before releases the lock by removing the map from the ConcurrentHashMap after finishing editing(submit or going back). This will cause a problem which is when the one who has the lock close the web page and doesn't click the submit or "back" button, the ConcurrentHashMap will keep record of the lock and no one can edit the questionnaire anymore. Therefore, we added one schedule thread which will release the lock an hour after the first one acquired the lock successfully. And clicking the submit or

"back" button will cancel the scheduled thread in order to prevent releasing others' lock, as it already released before by normal means.

Sequence diagram:



Design rationale (concurrency pattern(s))

Reason why choose the pattern:

5.1 For concurrency issues about timeslot

In 4.1.1 part, we choose an optimistic offline lock pattern.

This issue will not happen frequently. For example, when a manager mistakenly tell two employees to edit the capacity of the same timeslot, the issue will happen. but it is not very usual. In addition, a failed update will only cause loss of a single number which describe the capacity of a timeslot.

Therefore, the cost of a failed update is very tiny and acceptable. Hence, it is appropriate to use optimistic offline lock here to handle the issues with lower cost to the system performance.

In 4.1.2 part, we choose an optimistic offline lock pattern.

The issue will happen when timeslot capacity is very small or it is modified or deleted suddenly. Hence it is not frequent. Moreover, the performance of booking a timeslot is very important, because it is probable that many recipients book a timeslot on the sametime. And if we use pessimistic lock here, other recipients have to wait when a recipient is making a book, which is very low efficient. Therefore, it is a good tradeoff to use optimistic lock to handle these issue.

5.2 For concurrency issues about vaccine type

In 4.2.1 part, we choose an optimistic offline lock pattern.

First of all, when the administrator makes changes to the vaccine type, there is a recipient ordering the vaccine at the same time. The probability of such an event is low. Optimistic lock has high efficiency, allowing recipients and administrators to operate at any time. Pessimistic lock will affect the recipient's normal booking vaccine, and the efficiency is low. Therefore, in this issue we use optimistic lock.

In 4.2.2 part, we choose an optimistic offline lock pattern.

When the administrator changes the vaccine type, a healthcare provider adds a questionnaire at the same time. The probability of such an event is very low. Optimistic locking is highly efficient, allowing healthcare providers and administrators to perform operations at any time. The pessimistic lock will affect the healthcare provider to add questionnaires (when the administrator changes the vaccine type, they cannot add questionnaires), which is inefficient. Therefore, in this issue we use optimistic lock.

5.3 For concurrency issues about questionnaire

In 4.3 part, we choose a pessimistic offline lock pattern.

This issue will not happen frequently. For example, when a manager mistakenly tells two employees to edit the questionnaire contents, the issue will happen. but it is not very usual. In addition, wait is only needed when acquiring lock failed, instead of any work or data is thrown away. Therefore, the cost of a failed update is only waiting for others to finish and it’s easy to inform others to retry in the same hospital. Hence, it is appropriate to use a pessimistic offline lock here to handle the issues.

Testing strategy and outcomes

6.1 Testing strategy

We use two methods for testing, one is Apache's open-source testing tool Jmeter, which uses a local dynamic assembly of request data and sends the request through HTTP protocol post, and the other is manual testing. Both are to judge the test result by observing the data in the database.

6.2 Testing scenario and Outcomes

There are a total of 5 scenarios in this test, and the specific scenarios are described as follows:

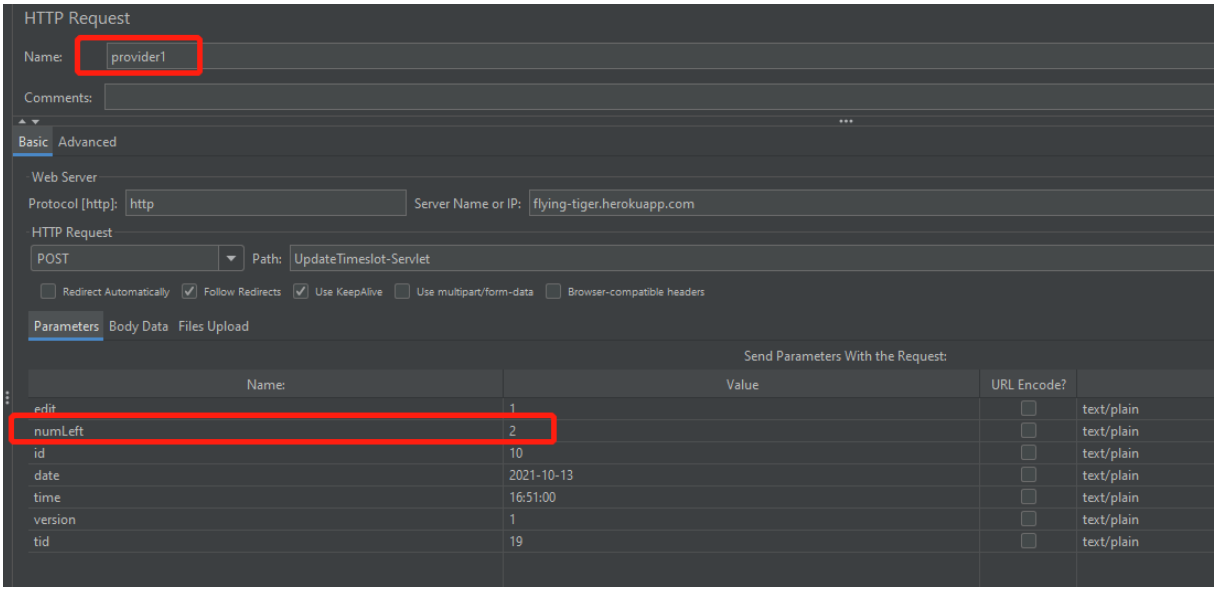
6.2.1Multiple users with the same health care provider account edit the same timeslot's capacity

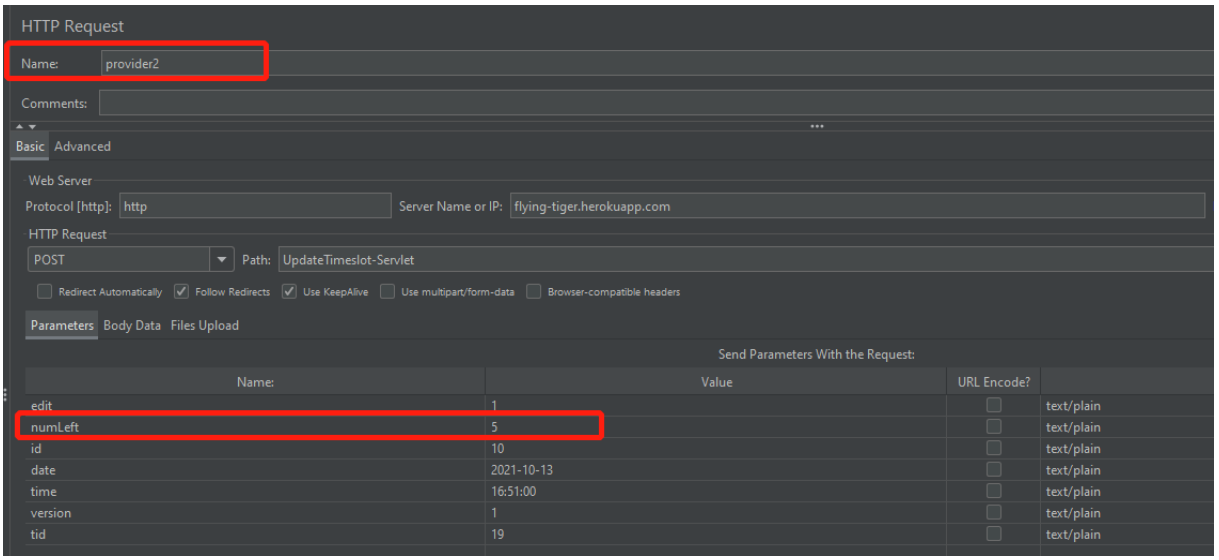
Test tool:

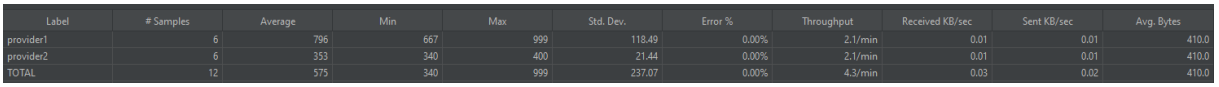
Jmeter

Test target:

When multiple users use the same health care provider account to edit the timeslot, the update submitted first will be processed successfully, and others will not be able to process it.



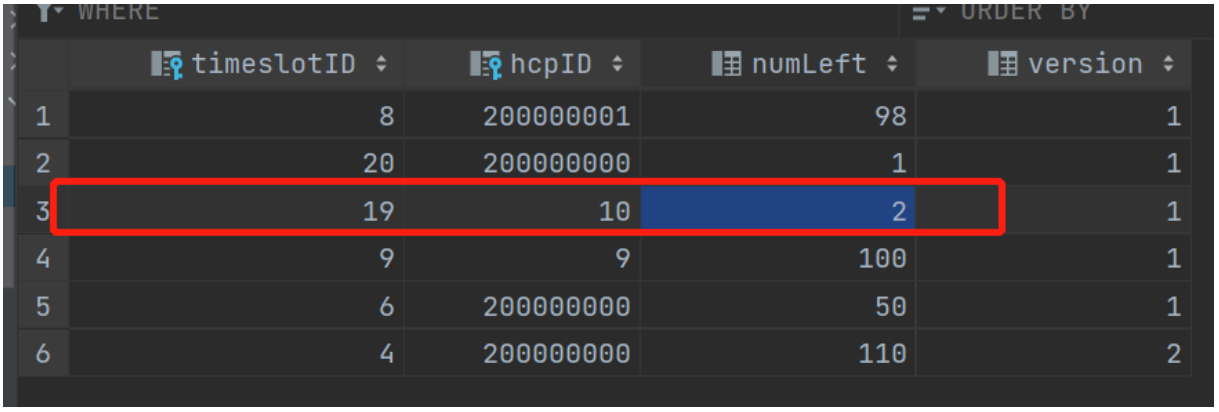




Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
provider1	6	796	667	999	118.49	0.00%	2.1/min	0.01	0.01	410.0
provider2	6	353	340	400	21.44	0.00%	2.1/min	0.01	0.01	410.0
TOTAL	12	575	340	999	237.07	0.00%	4.3/min	0.03	0.02	410.0

Test outcomes:

The database displays the information submitted first.



	timeslotID	hcpID	numLeft	version
1	8	200000001	98	1
2	20	200000000	1	1
3	19	10	2	1
4	9	9	100	1
5	6	200000000	50	1
6	4	200000000	110	2

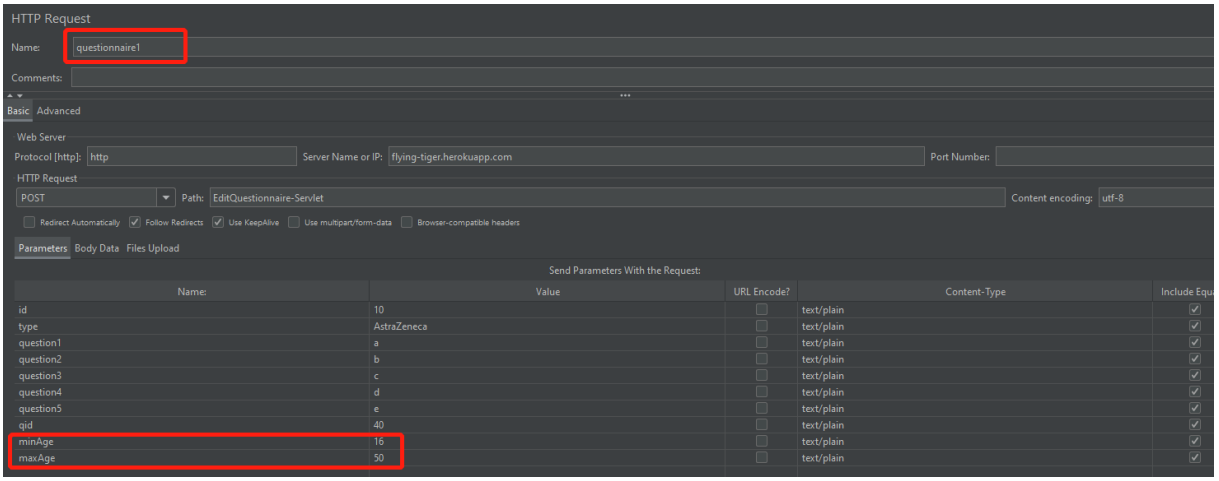
6.2.2 Multiple users with the same health care provider account edit the same questionnaire's capacity

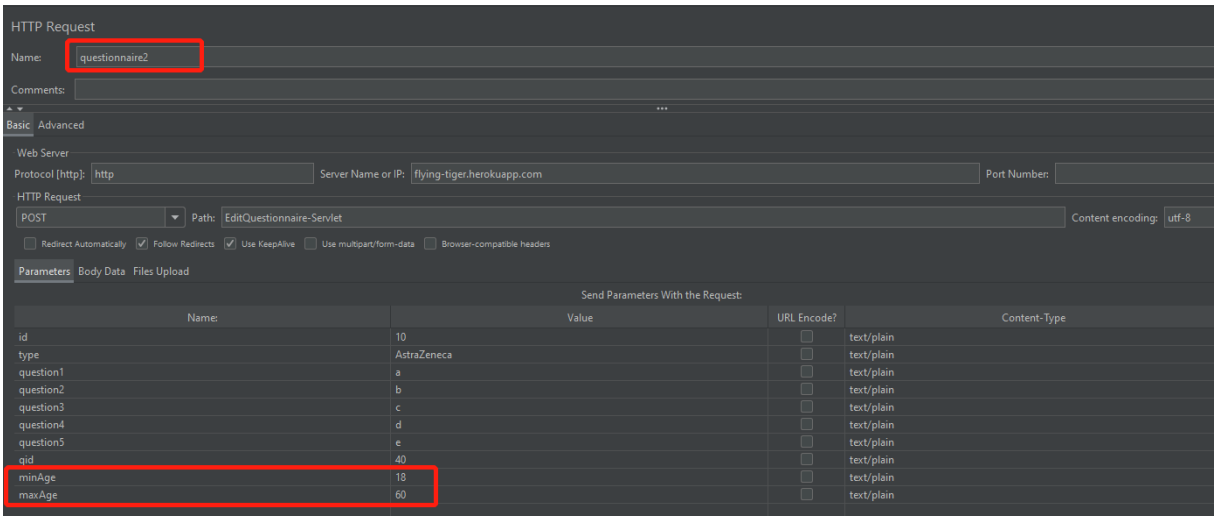
Test tool:

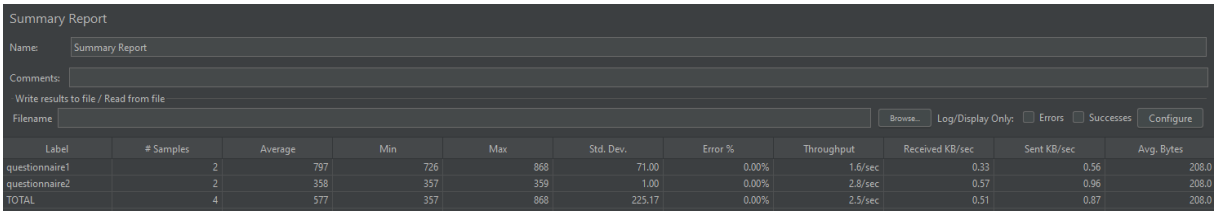
Jmeter

Test target:

When multiple users use the same health care provider account to edit the questionnaire, the update submitted first will be processed successfully, and others will not be able to process it.

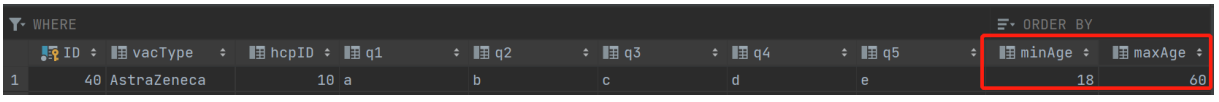






Test outcomes:

The database displays the information submitted first.



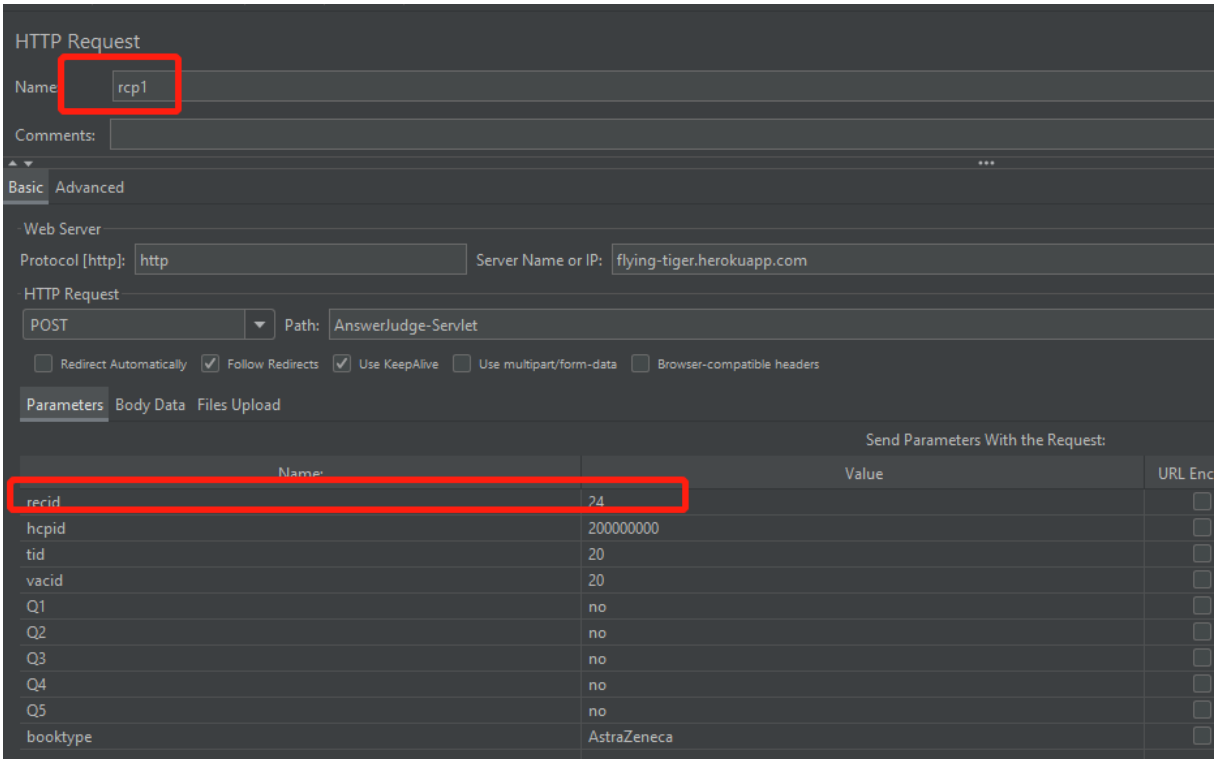
6.2.3 Recipient book a timeslot while its capacity set to 0 or it has been deleted

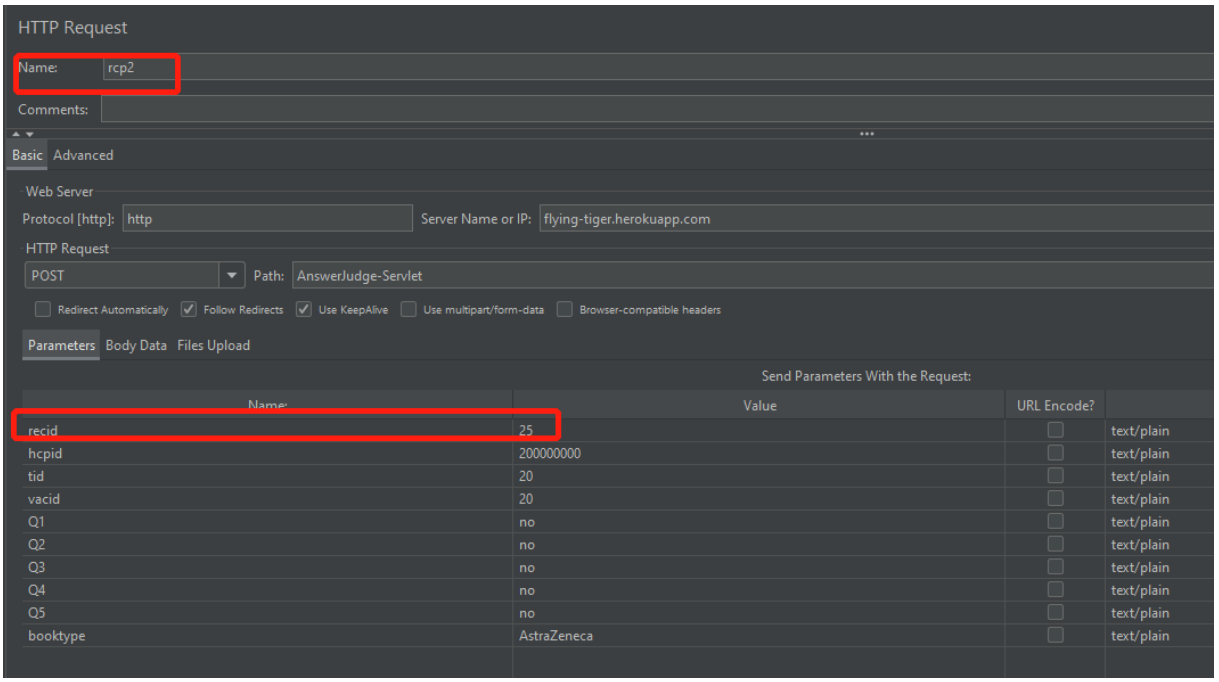
Test tool:

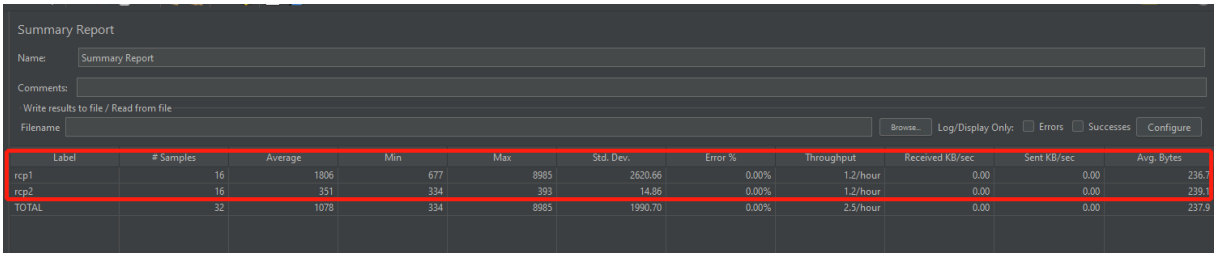
Jmeter

Test target:

When there is only one position in the timeslot, the recipient who is booked first succeeds. When timeslot has no place or is deleted, all recipients reservation is unsuccessful.

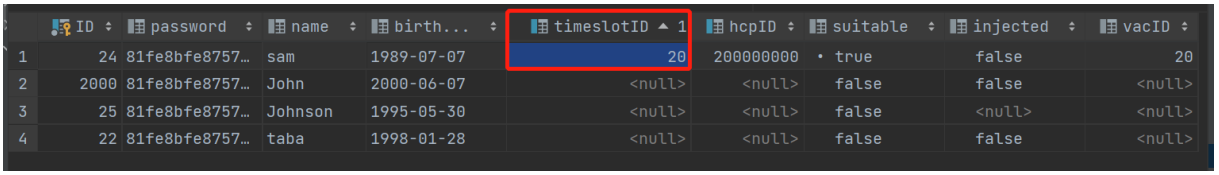






Test outcomes:

The recipient who submitted the first booking successfully. The recipient who submitted later failed to book.



	ID	password	name	birth...	timeslotID ▲ 1	hcpID	suitable	injected	vacID
1	24	81fe8bfe8757…	sam	1989-07-07	20	200000000	true	false	20
2	2000	81fe8bfe8757…	John	2000-06-07	<null>	<null>	false	false	<null>
3	25	81fe8bfe8757…	Johnson	1995-05-30	<null>	<null>	false	<null>	<null>
4	22	81fe8bfe8757…	taba	1998-01-28	<null>	<null>	false	false	<null>

6.2.4 Administrator edit or delete a vaccine type while the Recipient is booking it.

Test tool:

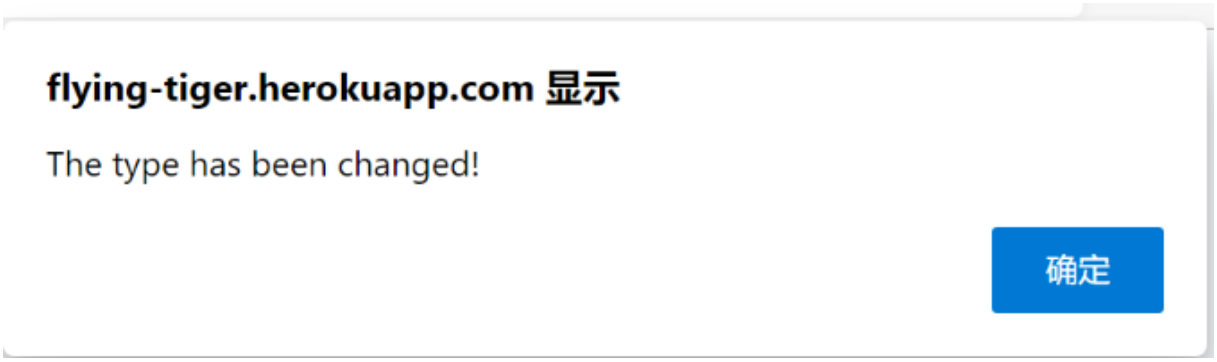
Manual test

Test target:

The administrator edit or delete the vaccine type successfully, but the recipient fails booking. the booking page shows “the vaccine type is changed”(when the type is edited) and “the vaccine type is deleted”(when the type is deleted by the administrator).

Test outcomes:

After the administrator successfully edits the vaccine type, the recipient reservation fails, and the reservation page displays "The type has been changed".



6.2.5 Administrator edit or delete a vaccine type while Health care provider is Adding a questionnaire for it.

Test tool:

Manual test

Test target:

The administrator edit or delete the vaccine type successfully, but the healthcare provider fails add questionnaire,. the Adding page shows “the vaccine type is changed or deleted”.

Test outcomes:

After the administrator successfully edits the vaccine type, the healthcare provider reservation fails, and the reservation page displays "The type has been changed".