Final Project Report

50.003: Elements of Software Construction

Members: Wang Yueheng (1004866), Lim Rui-Chong Anthony (1005264), Mohammed Fauzaan Mahaboob (1005404), Theresa Lam (1005477), Alphonsus Tan Yong Shing (1005534)

Done by
Cohort 4, Group 7

Table of Contents

Deliverables

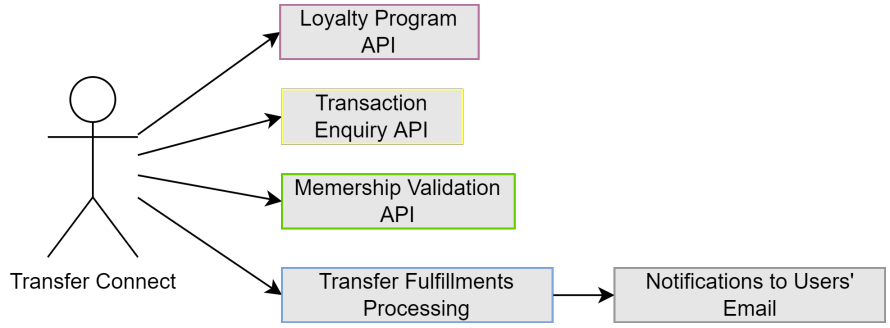
Deliverable	**Link**
GitHub repository	https://github.com/fauzxan/Ascendas-Loyalty
Features video	https://youtu.be/DROlS7mrSno
Testing video	https://youtu.be/o0jbY-yUr_0

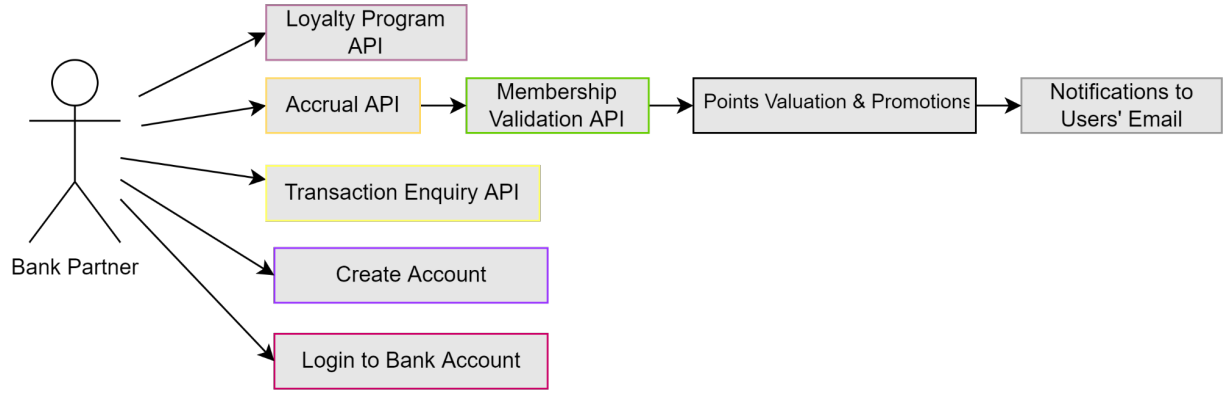
Documentation of Use Case Diagrams and Documents

Legend:

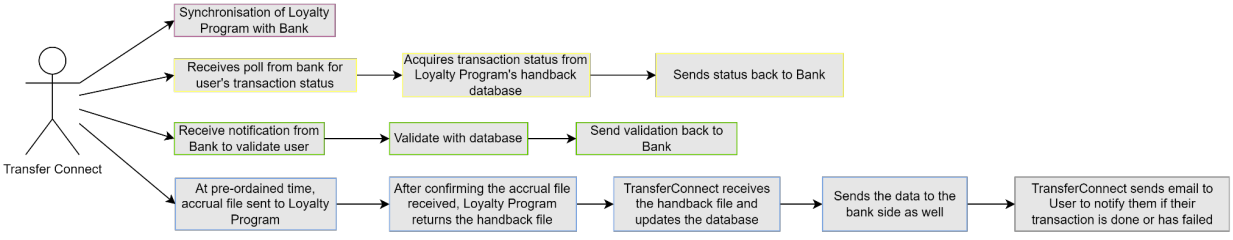
+ Use Case 1: Loyalty Program API
+ Use Case 2: Membership Validation API
+ Use Case 3: Accrual API
+ Use Case 4: Transaction Enquiry API
+ Use Case 5: Transfer Fulfillments Processing
+ Use Case 6: Notifications to Users' Email
+ Use Case 7: Points Valuation & Promotions
+ Use Case 8: Create Account
+ Use Case 9: Login to Bank Account
+ Use Case 10: Admin Account
+ Misuse Cases
+ Security Solutions

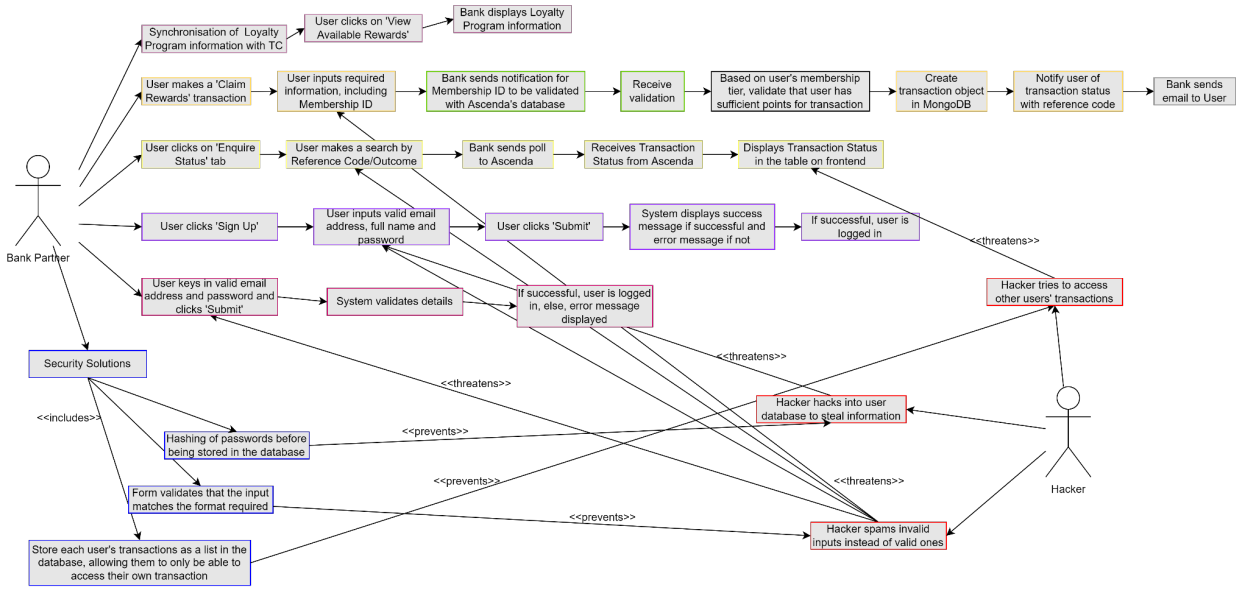
Overview:



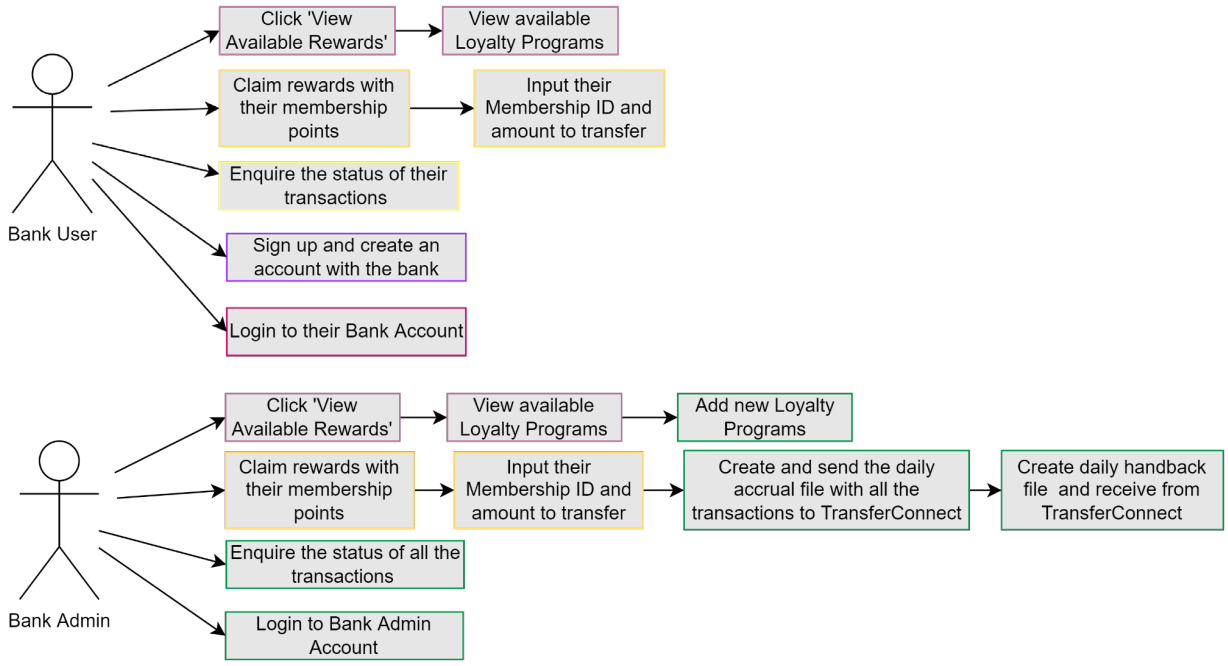


Detailed:





Bank User vs Bank Admin:



ID	Ascenda Loyalty Use Case 1
Name	Loyalty Program API
Objective	To display the latest updated information for the user on loyalty programs available for rewards points exchange
Pre-Conditions	1. The data in the loyalty program database must be accurate and must not have any NA/invalid values 2. The data in the loyalty program database must be correctly updated in the database when the admin makes a change by adding a loyalty program
Post-Conditions	*Success:* 1. Loyalty points information is successfully displayed on the website *Failure:* 1. The loyalty points information is unable to render into the UI frame correctly 2. There are missing critical values
Actors	*Primary:* 1. Loyalty program database *Secondary:* 1. User 2. Web application
Trigger	User desires to interact with the loyalty points marketplace for any of the following reasons: 1. Wants to find information about loyalty programs available 2. To redeem/transfer loyalty points
Normal flow	1. Loyalty program information is synchronised between the bank and loyalty program database 2. The user wants to view available loyalty points 3. The user enters correct credentials and logs into their account and views the required information 4. The user interacts with the website to view loyalty programs and its related information.

Alternative flow	1. Loyalty program information synchronisation between the bank and loyalty program database fails 2. The user wants to view available loyalty points 3. The user enters correct credentials and logs into their account and views the required information. Information field in the UI frame is left blank and makes it known to the user it does not exist
Interacts with	Get loyalty program information from database, check if values are correct, reflect changes input by the admin, login
Open Issues	-

ID	Ascenda Loyalty Use Case 2
Name	Membership validation API
Objective	To validate the provided membership number of a given loyalty program, to ensure that the user provides an input corresponding to any of the expected expressions and format for the loyalty programs given to TransferConnect.
Pre-conditions	Regular expressions and formats of all the loyalty programs must be stored in the system or database to be validated against the form input
Post-conditions	Success: Form input matches a regular expression stored, input validated Fail: No match found, error message displayed indicating invalid input
Actors	Primary: Customer submitting input of the form, the input referring to the loyalty program they wish to transfer their points to. Secondary: Form validator, database system
Trigger	Customers who wish to transfer points
Normal Flow	1. Wants to transfer points to a certain loyalty program X 2. User enters the expression of loyalty program X 3. System performs check to determine if input matches any of the expected expressions and format given to TransferConnect by the loyalty programs 4. If successful, the input is validated and no error is thrown
Alternative Flow	1. Wants to transfer loyalty points 2. Enters an input, but the input does not match any expected expressions in the database 3. System indicates input does not match any expected expressions 4. Error message displayed

Interacts With	Get form input, check form input, notify success or failure use case, selection of loyalty program
Open Issues	Is the trade-off for a more thorough check at the expense of a longer time taken to complete a round-trip request worth?

ID	**Ascenda Loyalty Use Case 3**
Name	Accrual API
Objective	To process credit requests for loyalty programs after receiving accrual request on behalf of banks

Pre-conditions	- Receive the following credit information from banks:
	  - Loyalty program ID
	  - Member ID
	  - Member first name
	  - Member last name
	  - Transfer date
	  - Amount
	  - Any number of additional information as required
	- Validated Member ID using Membership validation API
	- Customer has sufficient points for redemption

Post-conditions	- Success
	  - Create transaction object in MongoDB
	  - Provide customer with reference code
	  - Send email notification
	- Failure
	  - Display error to customer (Invalid Membership ID, etc)

Actors	- Primary
	  - Bank partner
	- Secondary
	  - TransferConnect database

Trigger	Receive an incoming credit request
Normal flow	1. Receive an incoming credit request 2. Validate member ID follows regex format for loyalty programme 3. Validate that user has sufficient points 4. Multiply points redeemed based on user's tier 5. Create transaction object in MongoDB (used for rendering transaction enquiry page) 6. Notify customer of transaction status (success/ failure) with reference code
Alternative flow	2.a Membership validation fails, display error popup that states that the membership ID has failed validation 3.a. Insufficient points, display error popup that states that the user has insufficient points
Interacts with	Membership validation, Notification to bank client
Open issues	1. How do we verify the integrity of the credit transfer received?

ID	**Ascenda Loyalty Use Case 4**
Name	Transaction Enquiry API
Objective	To allow users to enquire about the status of their transactions through their banks
Pre-Conditions	1. The user must be logged into their bank account to be able to have access to be able to create a poll from the bank to Ascenda to retrieve their transaction status from the Loyalty Program 2. The user must have an existing ongoing transaction with the Loyalty Program
Post-Conditions	*Success:* 1. Transaction status is shown *Failure:* 1. Error message showing reason for failure is shown 2. Poll to Ascenda does not go through 3. Retrieval of transaction status from Loyalty Program fails
Actors	*Primary:* 1. Bank user through the bank *Secondary:* 1. Administrator (Transactions Manager from Ascenda) 2. Users database 3. Bank Administrator
Trigger	Concern for their transaction status prompts them to check
Normal flow	1. Desire to check the status of the transaction 2. In their bank account, the user can then click on the enquire status tab to check the status 3. Bank sends a poll for the transaction status to Ascenda 4. Ascenda acquires transaction status from the Loyalty Program's handback database and sends it back to Bank's user database 5. All transactions the user has made is successfully rendered onto a table format, with each row indicating the transaction reference code, whether it is a success, pending or fail, and the reason for failure if applicable 6. User can search for transaction made using the reference code generated during submission
Alternative flow	Case 1: Polling to Ascenda's database fails 1. Desire to check the status of the transaction 2. In their bank account, the user can then click on the enquire status tab to check the status 3. Polling of Ascenda's TC system fails

	4. Ascenda acquires transaction status from the Loyalty Program and sends it back to Bank 5. No rendering of any transactions on webpage Case 2: User tries to search for transaction he did not make 1. Desire to check the status of the transaction 2. In their bank account, the user can then click on the enquire status tab to check the status 3. Bank sends a poll for the transaction status to Ascenda 4. Ascenda acquires transaction status from the Loyalty Program and sends it back to Bank 5. Transactions are rendered, user is unable to view a transaction he did not make
Interacts with	Login, Record Transaction, Membership Validation API use cases
Open Issues	1. How will bank users be identified from bank administrators? 2. How will the system identify Ascenda administrators from other users? 3. How long will it take for the Ascenda system to acquire the transaction status from the Loyalty Program and send it back to the Bank?

ID	Ascenda Loyalty Use Case 5
Name	Transfer fulfilments processing
Objective	To provide fulfilment of transfers to loyalty programs and to carry out file processing
Pre-conditions	The specific loyalty program has a partnership with Ascenda Loyalty
Post-conditions	Success: Client loyalty program performs accrual and handback of programs once a day, the handback file is automatically generated and sent back to Ascenda. Accrual file format is correct, and the file processing is done correctly to generate the correct handback file. Fail: Accrual file format is wrong, and handback file generated is of the wrong format.
Actors	Primary: Ascenda Loyalty system Secondary: Client loyalty programs, bank apps, Accrual API
Trigger	If there is any accrual file sent from Ascenda, the system is called to do the loyalty program fulfilment process.
Normal Flow	1. Ascenda sends the accrual file a pre-ordained time. Client loyalty programs perform the accrual and handback of programs once a day at a pre-ordained time 2. After confirming the accrual file Ascenda sent, the loyalty program creates and sends a handback file to Ascenda's system along with the relevant outcome codes 3. Ascenda receives the handback file and sends the data to the client side
Alternative Flow	Case 1: Accrual file not sent 1. Ascenda does not send the accrual file at a pre-ordained time to the loyalty programs. 2. Handback file is not generated and sent to Ascenda Case 2: More than one file is sent 1. Ascenda accidentally sends out more than one file a day 2. The extra files will be ignored by the partners and only the first file received is used

	3. Based on the chosen file, the loyalty program confirms the accrual file, then creates and sends a handback file back to Ascenda's system 4. Ascenda receive the handback file
Interacts With	Accrual API, send accrual file to SFTP server, receive handback file from SFTP server, file processing
Open Issues	

ID	**Ascenda Loyalty Use Case 6**
Name	Notifications to users' email
Objective	To provide user(customers) with an update on their transaction status to their emails
Pre-conditions	1. User must have signed up with a valid email 2. User must be logged in 3. User must have made a transaction
Post-conditions	Success: 1. User is sent an email consisting of acknowledgement of transaction 2. Emails are sent on another occasion, when the handback file is received and emails are sent to notify customers of the result of their transaction and any reasons why it has failed
Actors	Users, users' email account
Trigger	User made a transaction, or upon receive of handback file
Normal flow	Sending notification email to notify successful transaction: 1. User claims loyalty rewards using his points 2. Email is automatically generated and sent to the email provided during the creation of account phase Sending notification email to notify status of transaction: 1. Handback file is received back from the SFTP server 2. Email notification is sent from Ascenda to notify a registered customer that the transaction is done, or has failed
Alternative flow	Sending of email fails on making a transaction 1. User claims loyalty rewards using his points 2. Email scheduler malfunctions, pending email is not sent to customer Sending of email fails on receiving handback file 1. Handback file was not received successfully from SFTP server 2. Email notification not sent
Interacts with	Login, create account, make transaction, receive handback file
Open issues	

ID	**Ascenda Loyalty Use Case 7**
Name	Points valuation & promotions
Objective	To create a monetary valuation for the user's bank points with the Loyalty Program's based on the user's membership tier with the bank
Pre-conditions	- User must have registered for a bank account - User must be sorted into a membership tier

Post-conditions	- Success
	  - User is able to enjoy an enhanced earn rate when redeeming their points
	- Failure
	  - The user's earn rate does not differ from other users despite being in a higher membership tier
	  - Error message for failed request

Actors	- Primary
	  - User
	- Secondary
	  - Bank Admin

Trigger	Desire to claim rewards.
Normal flow	1. User wants to claim their points 2. User inputs their Membership ID and amount of points they wish to claim 3. User clicks 'Submit' (Use case 6 is carried out) 4. With the same amount of points, a user in a higher membership tier is able to claim a reward that requires more points than a user in a lower membership tier 5. Successful claim rewards pop-up displayed with the reference code for the transaction
Alternative flow	4. The user in the higher membership tier is unable to claim the reward (point ratio conversion malfunctions) 5. Error message displayed
Interacts with	Process Credit Requests
Open issues	

ID	**Ascenda Loyalty Use Case 8**
Name	Create account
Objective	To enable users to create a bank account

Pre-conditions	- Potential users must be able to provide the following:
	  - Email address
	  - Full name
	  - A sufficiently memorable password

Post-conditions	- Success
	  - User information is saved in TransferConnect's database
	  - User credentials can subsequently be used for logging in to their bank account
	  - User is allocated their loyalty points
	- Failure
	  - Error regarding failure to create account is displayed

Actors	- Primary
	  - User
	- Secondary
	  - TransferConnect database

Trigger	Desire to create a bank account
Normal flow	1. Key in valid email address 2. Key in full name 3. Key in sufficiently memorable password 4. Click submit 5. Success message is displayed 6. User is logged in
Alternative flow	5.a Error message is displayed. Potential errors: Invalid email address, account already exists
Interacts with	Login
Open issues	1. How should we hash the passwords?

ID	**Ascenda Loyalty Use Case 9**
Name	Log in to bank account
Objective	To enable users to log into their bank account
Pre-conditions	- User must have registered for a bank account - User must remember their sufficiently memorable password and email address - User must key in the above-mentioned credentials into input boxes

Post-conditions	- Success
	  - User gains access to bank account
	- Failure
	  - Error message is displayed. Potential errors: No account exists, invalid password, etc.

Actors	- Primary
	  - User
	- Secondary
	  - TransferConnect database

Trigger	Desire to log into bank account
Normal flow	1. Key in valid email address 2. Key in sufficiently memorable password 3. Click submit 4. Success message is displayed 5. User is logged in
Alternative flow	4.a Error message is displayed. Potential errors: Invalid log in credentials, account already exists
Interacts with	Create account
Open issues	1. How do we improve the speed of account lookup?

ID	**Ascenda Loyalty Use Case 10 (Only for Bank Admin)**
Name	Admin account
Objective	Help gauge functionality of key features of the app without having to wait for a set time period that will trigger certain actions. Key features include, but not limited to: 1. Creation of accrual file 2. Creation of handback file 3. Creation of new loyalty programs 4. Enquiry of transaction statuses of ALL users
Pre-conditions	1. The user must know the login credentials of the admin account: Email: admin_ascendas@gmail.com Password: admin 2. For the creation of loyalty program, a. the user must provide URL of an image that is hosted in a remote server b. the user must have knowledge of relevant regex string for membership ID validation
Post-conditions	1. Transactions hashmap of each user changes in the client database 2. New accrual and handback files are created in the SFTP server
Actors	Admin
Trigger	Desire to have access to any of the aforementioned features. (in the objective section)
Normal flow	1. Trigger event occurs 2. User logs into admin_ascendas@gmail.com 3. User performs one of the actions mentioned in the objective section 4. User logs out of the account
Alternative flow	1. One of the features does not work as intended as the connection with the TC server is not established correctly
Interacts with	Loyalty Program API, Membership validation API, Accrual API, Transaction Enquiry API, Transfer fulfillments processing, Notifications to users' email, Points valuations & promotions, Create Account, Log in to bank account
Open issues	

Design of different subsystems

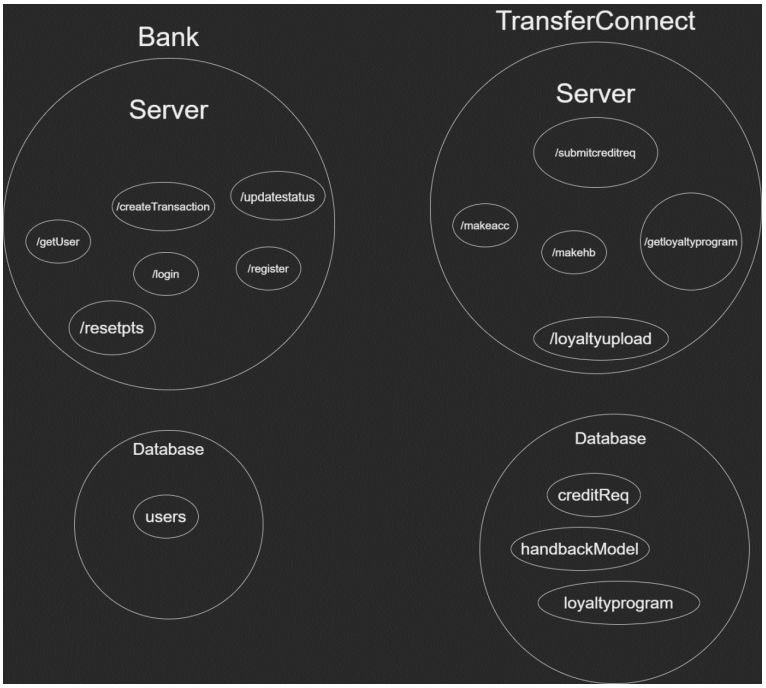
JavaScript, the language we chose to program in, is not an OOP language, therefore it would not be apt to represent it based on classes and relationships between them. Instead, we chose to represent the relationship between the Client side, server side, and the various files that are transmitted between them in the database.

Backend/ Database

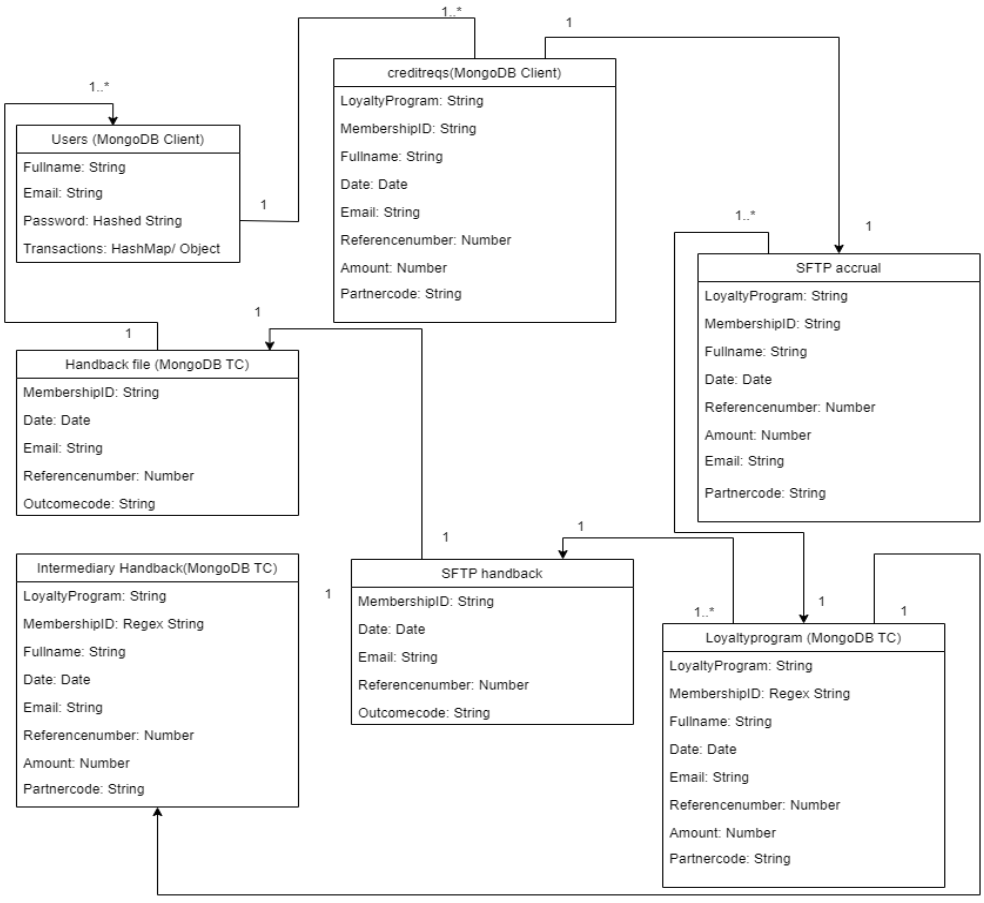
Class explanation:

1. **Users:** Represents each individual user on the database. Also contains transactions hashmap for that is later rendered in the Enquire transactions page
2. **Creditreq (mongodb bank):** local buffer of all the transactions. At most, once a day, the admin clicks "create handback file", during which the content of the creditreq database in mongodb is pushed to accrual csv file and the local buffer's content is deleted.
3. **SFTP Accrual File:** All the requests from all the users are consolidated and sorted by the loyalty program and sent to the respective loyalty programs. The accrual file is manually refreshed every day to reflect the changes.
4. **SFTP Handback File:** Results from each Loyalty Program is sent as a handback csv file to the bank.
5. **Handback (mongodb TC):** The handback file is used to update the transactions list of the users, that will later be reflected on the enquire transactions page
6. **Intermediary handback (mongodb TC):** The intermediary handback file is a local buffer stored by transferconnect, and is used as an API call to help the bank send emails to their customers, when the transaction statuses have been updated
7. **Loyalty Program:** Represents the data for each loyalty program that we store in the database. This also contains the membership validation regex[1].

[1] **Membership validation: Each loyalty program has a unique verification number**

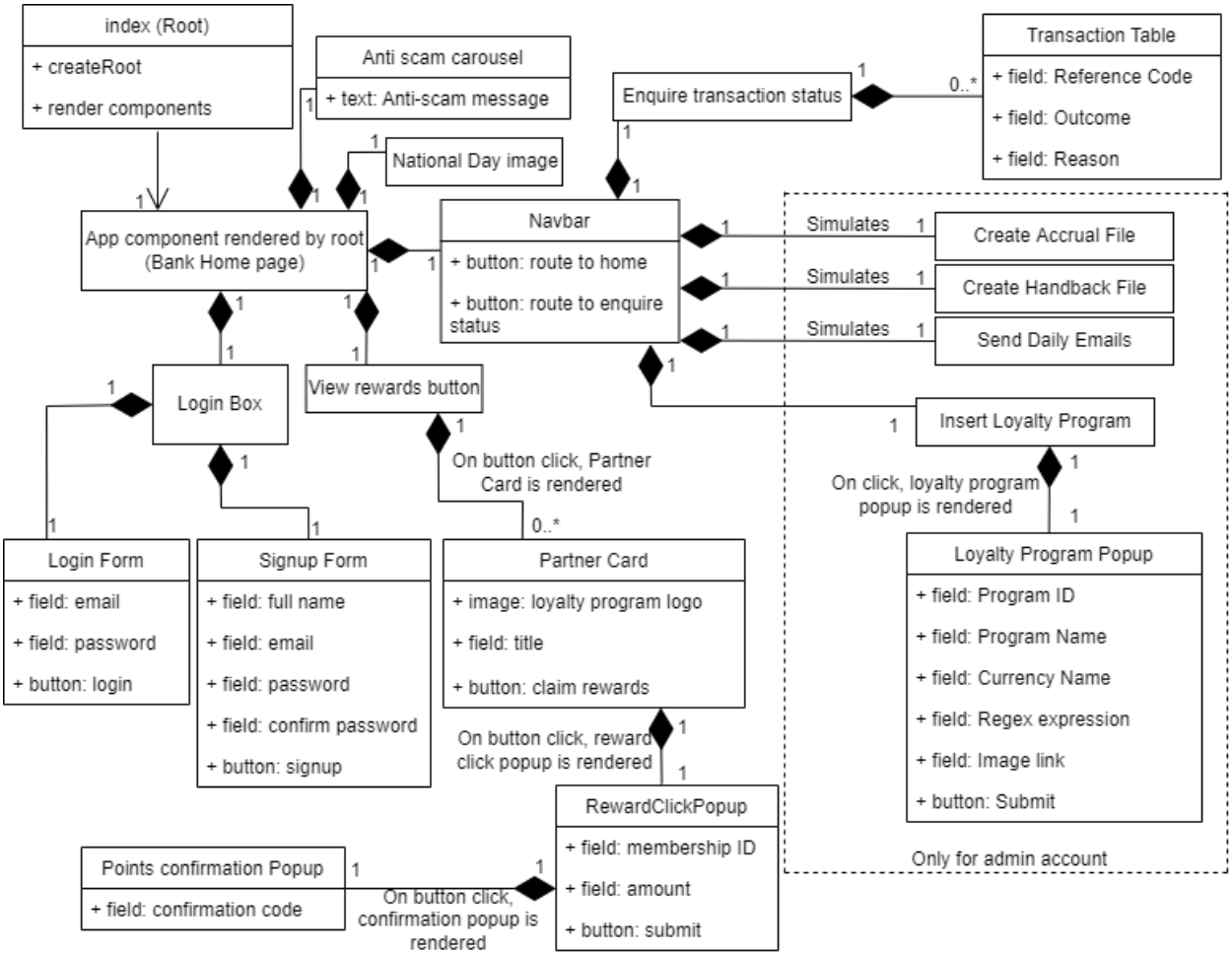


Throughout the entire process, we ensured that the bank and TransferConnect operate independently on the backend, communicating only through API calls as shown above.



Frontend

For the frontend, we used React framework, and the main components of it can be modelled as such below.



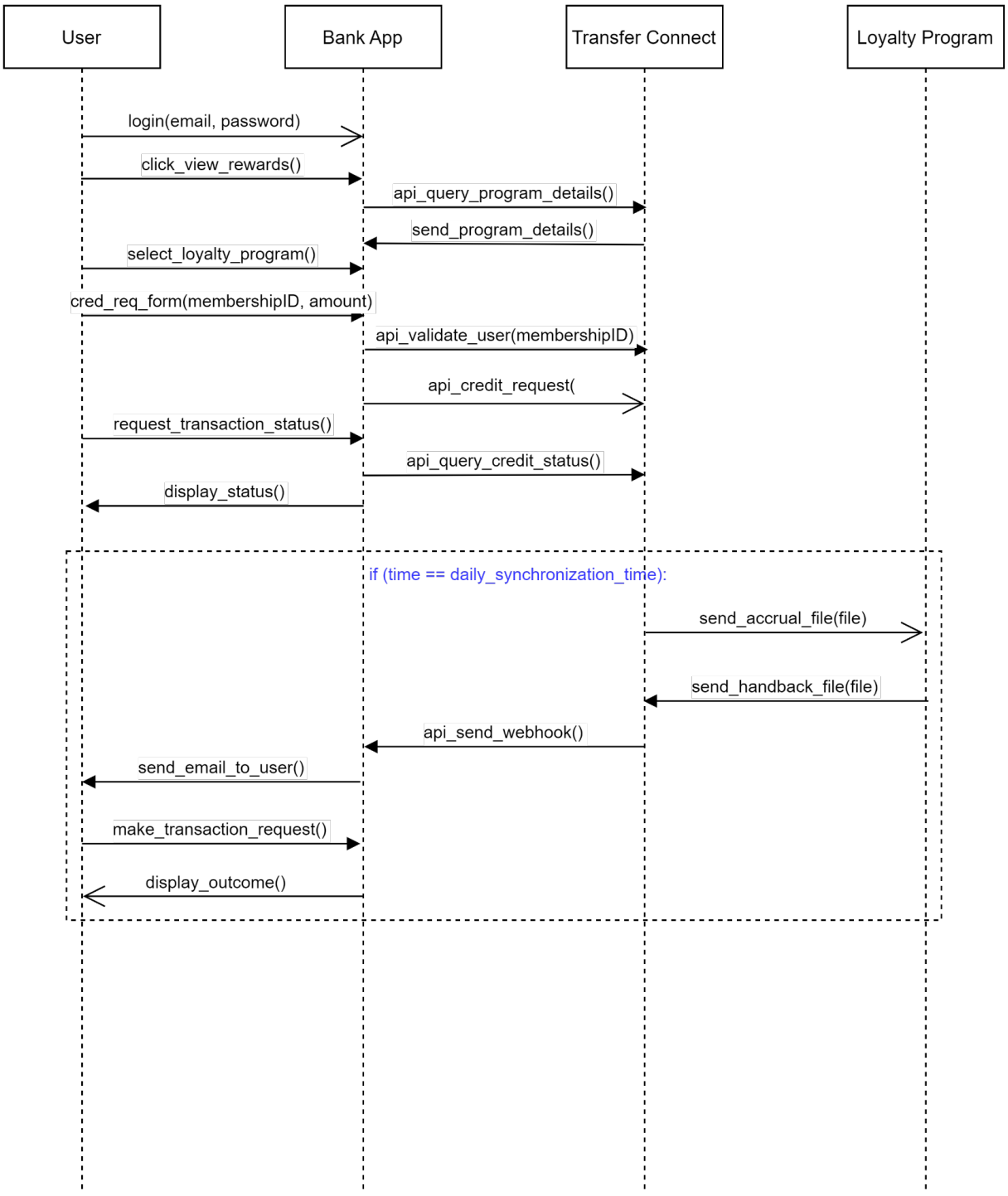
The UML diagram shows the components used in React, and the attributes that make up each component are listed as shown.

The frontend components are mainly related by composition because there is a strong dependency between components, and when the parent component is deleted, the child component cannot exist. Most of the composition relationships are also 1 parent to 1 child. Exceptions are the rendering of Partner cards since the number rendered is dependent on the number of loyalty programs available, and transactions since the number rendered is dependent on the number of transactions made by the user.

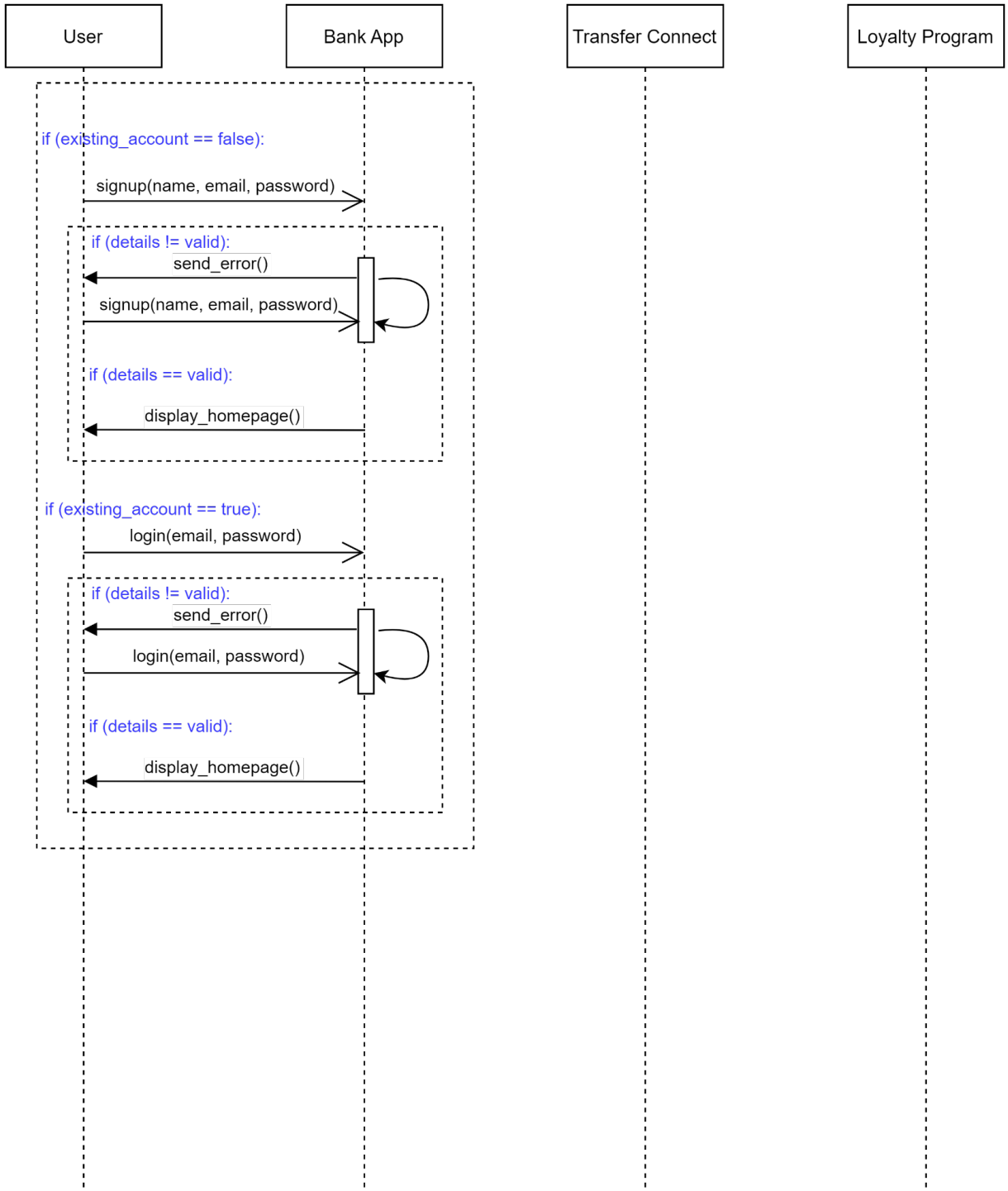
We implemented conditional rendering; whereby only certain components are shown if the account logged in is the admin account. The admin is able to simulate sending of accrual file, creation of handback file, sending daily emails and insert new loyalty programs.

Sequence Diagrams

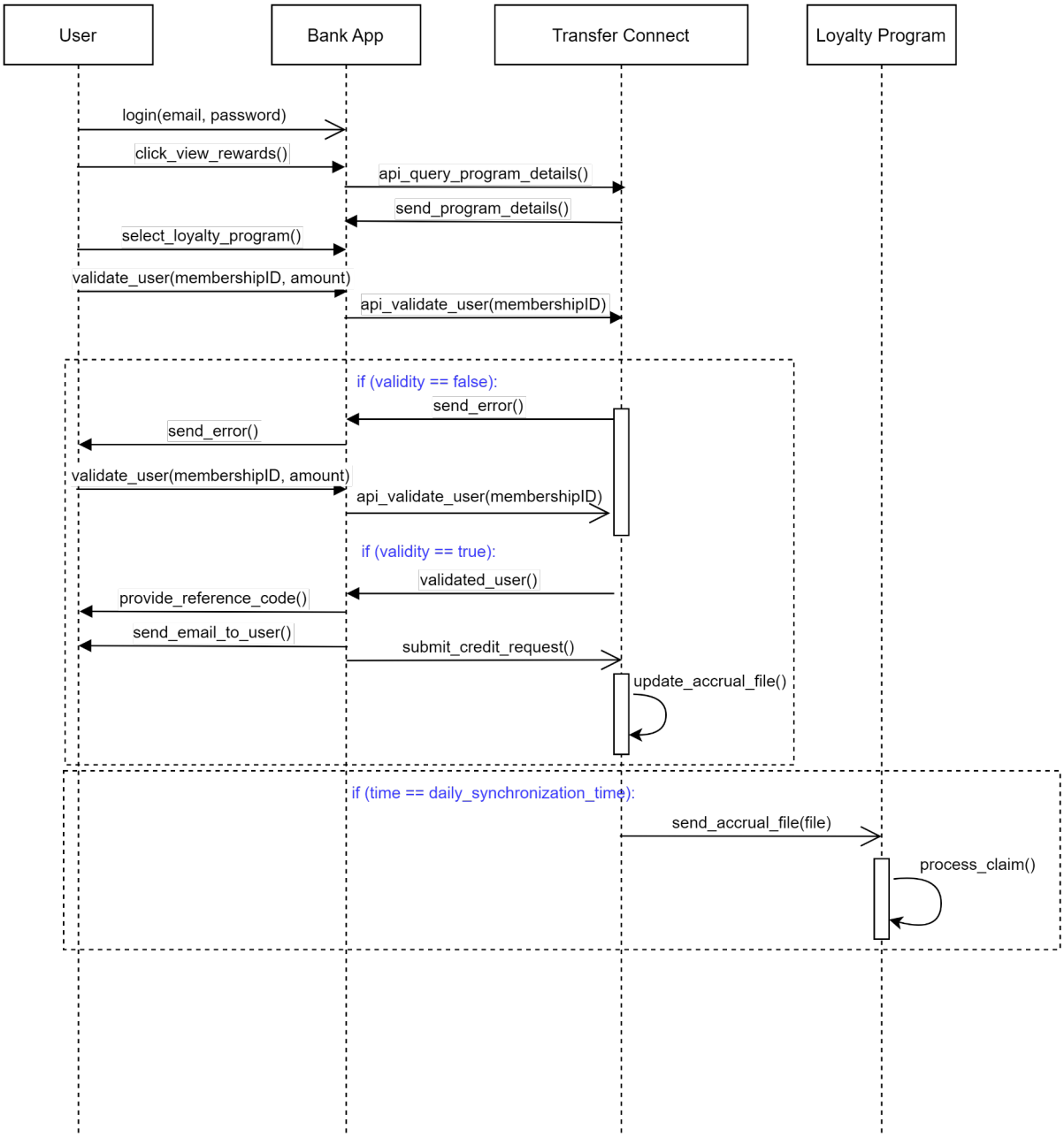
Sequence for the overall features of our project:



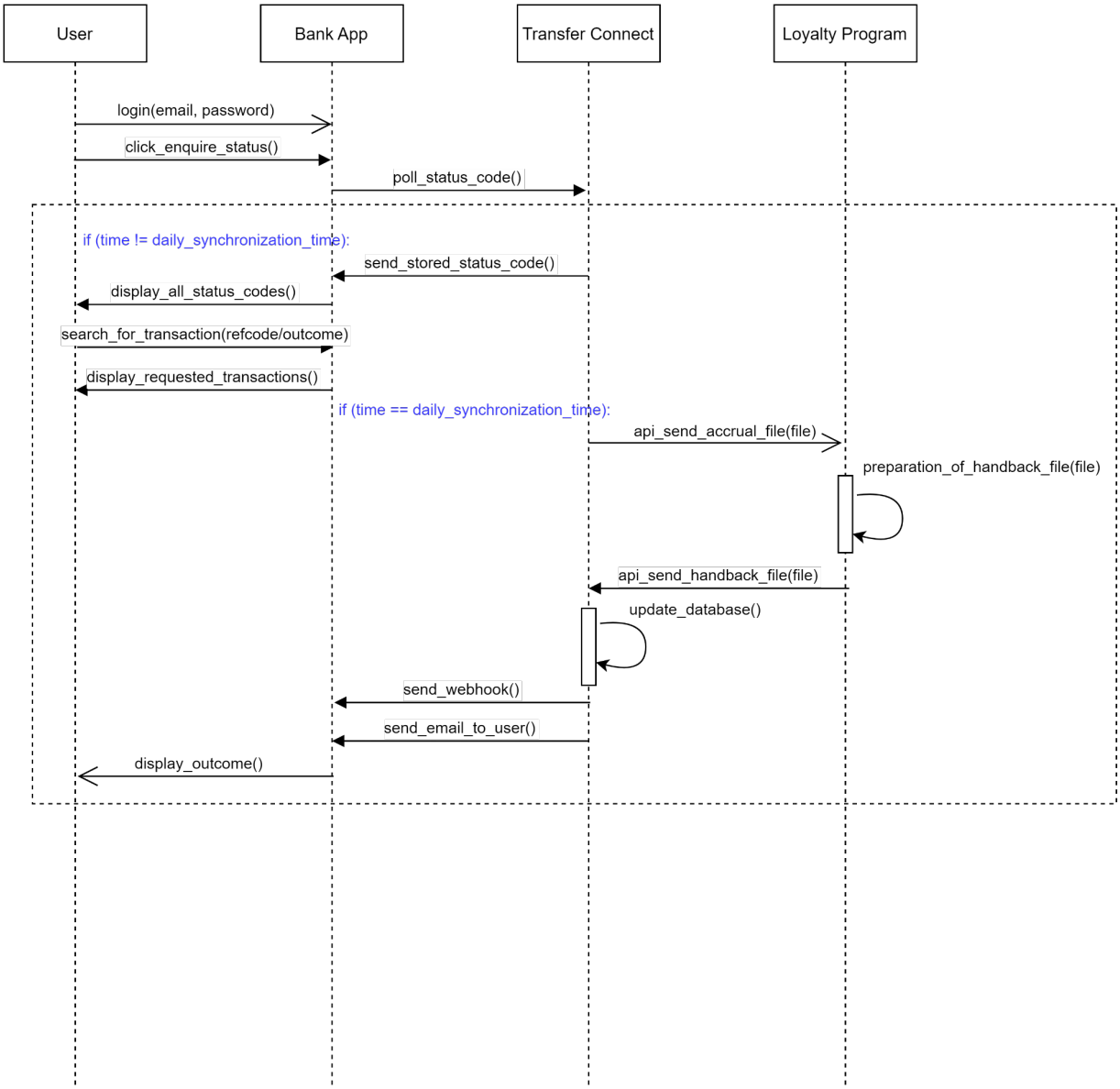
In-depth sequence for user login and signup (Use Case 8 and 9):



In-depth sequence for when a user clicks on ‘View Available Rewards’ and makes a new credit request (Use Case 1, 2, 3, 6, 7):



In-depth sequence for when a user makes a transaction status enquiry (Use Case 4, 5, 6):



Implementation Challenges

Algorithmic challenges

We faced difficulties when trying to organise data in the most optimal way possible.

Problem:

A given user should only access their own transactions and should not be able to access the outcome codes of other users, if given access to their reference number in some way (maybe the users met in person and shared the reference code). The raw data is received from the handback file (including reference code and outcome code) and we were required to display it on a given user's client side, i.e., user A's bank account.

Solution:

Once we obtained the reference code and outcome code associated with that reference code, we had to figure out how to store them in the bank-side database in a way that User A can only access their list of transactions. To do this, we decided to maintain a "transactions" **hash map** for each user, called transactions. When the user clicks on the enquiry transactions page, the data is actually being pulled from the transactions map of that user.

Engineering challenges

1. Problem:

   There was no way to create the accrual and the handback file without waiting for the pre-ordained time.

   Solution:

   Creation of an admin account to ensure ease of demonstration of features, and also to add additional features like being able to add loyalty programs to the database.

2. Problem:

   The bank's database has a locally maintained database that is used as an information buffer before it pushes information onto the SFTP server, called the creditreqs database (contains information to be pushed into the remote accrual file). Currently, this buffer is set to grow in size as users make more transactions. The handback file received also requires a remote buffer.

   Solution:

   The locally maintained buffer is overwritten each day, so the problem of high local storage space requirement is minimised. However, we must still be able to provide the client with access to the list of all the transactions so it can be rendered in the transaction's enquiry page. So, the following has been implemented to resolve this issue:

1. When admin clicks on “create accrual file”, the accrual buffer information is pushed into the sftp server, which maintains a permanent copy, and the local buffer in the database is deleted.
2. When the admin clicks on “create handback file”, the content of the local handback buffer is deleted, and the new information overwrites the current data. The handback content is also filtered such that a transactions map is created and sent to each user in the client side user database.

Testing challenges

1. Problem:

   When creating the tests for our web application, we faced challenges when designing test cases that are accurate and effective to our web application. Because the project was divided into different parts, backend and frontend implementation and testing, and delegated to different group members to execute, when the time came to create test cases, there was a lack of information on how the implementation was done, what features to test, and how to go about testing it. For example, lack of information on user journey and navigation through the web application, and the possible outcomes, made creating effective frontend tests challenging.

   Solution:

   Increased communication between members through regular group meetings to ensure that every member was on the same page with how features were implemented as well as the flow of the entire project.

2. Problem:

   We are unable to use automated selenium tests for some tests, such as to confirm whether the email has been sent and received successfully since the Selenium Webdriver is unable to access Gmail accounts due to privacy and security concerns.

   Solution:

   For these kinds of tests pertaining to data security, we have no choice but to do manual testing and verification to ensure the email has been sent out correctly, and that it correctly reflects the transaction number and points claimed for each transaction. It would also be beneficial to do so because if this were to be a real application, the number of points to be claimed must be accurate. Having human verification would be more reliable in this case.

Testing Strategy

Backend Testing

Testing files

Purpose of testing the accrual and handback files:

Handback files are meant to serve as a response to the accrual files sent from the client side, and the **accrual file** represents the data that the loyalty program requires to validate if the transaction requested by the user is successful.

Input generation:

Input for the accrual file will be taken from both the manual and the automated tests (using Selenium, discussed later) that we provide. The accrual file will be stored in the SFTP server provided to us by the client.

Testing

We intend on creating sample handback files (such as the one below) on the SFTP server, whose data will be pulled from the accrual files residing on the server. Afterwards, we will send the handback file, along with randomised outcome code[2] to our front end using a webhook to test if the simulated outcome codes can be displayed on the website.

The Accrual File:

```
"memberid","fullname","email","date","amount","partnercode","loyaltyprogramme","referenceNumber"
"1234567890","stun","seed0@gmail.com","8/8/2022",150,"DBS","Sands Group",2022886
"1234567890","demo","demo@gmail.com","9/8/2022",466,"DBS","Sands Group",2022890
"388561182G","test1","test1@test.com","9/8/2022",,"DBS","Sands Group",2022890
"881048187","test1","test1@test.com","9/8/2022",,"DBS","Sands Group",2022891
"121333221","test1","test1@test.com","9/8/2022",,"DBS","Sands Group",2022892
"1234567890","test1","test1@test.com","9/8/2022",,"DBS","Sands Group",2022893
"1234567890","test","test_enquiry@test.com","9/8/2022",7176,"DBS","Sands Group",2022894
```

[2] **List of outcome codes:**
0000 - success, **0001** - member not found, **0002** - member name mismatch, **0003** - member account closed, **0004** - member account suspended, **0005** - member ineligible for accrual, **0099** - unable to process, please contact support for more information

The expected Handback File:

```
"date","amount","referenceNumber","outcomecode","email"
8/8/2022,150,2022886,0001,seed0@gmail.com
9/8/2022,466,2022890,0002,demo@gmail.com
9/8/2022,,2022890,0005,test1@test.com
9/8/2022,,2022891,0000,test1@test.com
9/8/2022,,2022892,0004,test1@test.com
9/8/2022,,2022893,0004,test1@test.com
9/8/2022,7176,2022894,0001,test_enquiry@test.com
```

Testing API endpoints

Postman

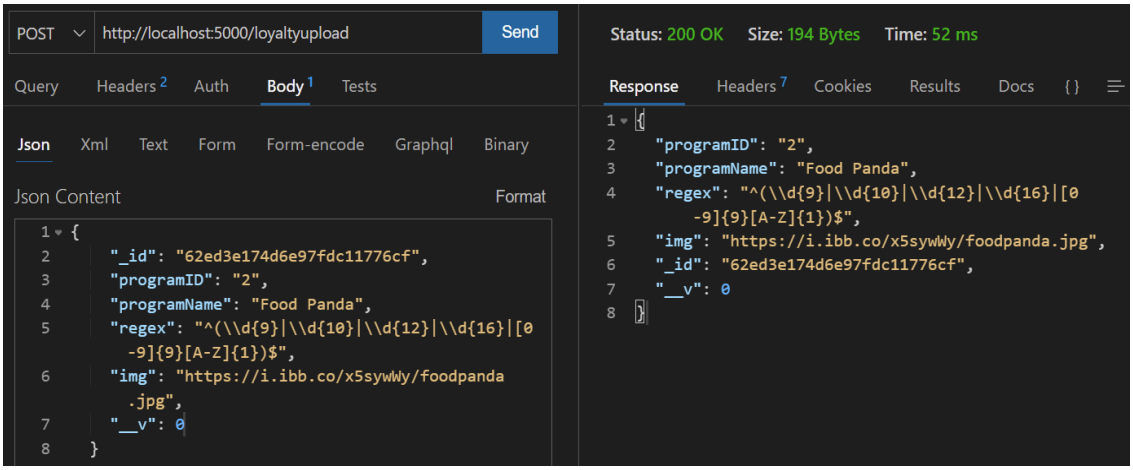
Purpose of testing our API endpoints:

By implementing API testing, we can ensure that the APIs we made perform as expected. While implementing our features, we sometimes wish to be able to immediately test the APIs we built immediately. Using Postman allows us to do so by bypassing the creation of the frontend which is to be used to call the API in the actual application. This reduces the time needed to test the application since the backend can be developed faster without having to put in more effort to create the corresponding frontend. Some examples of the backend testing of APIs are shown below.

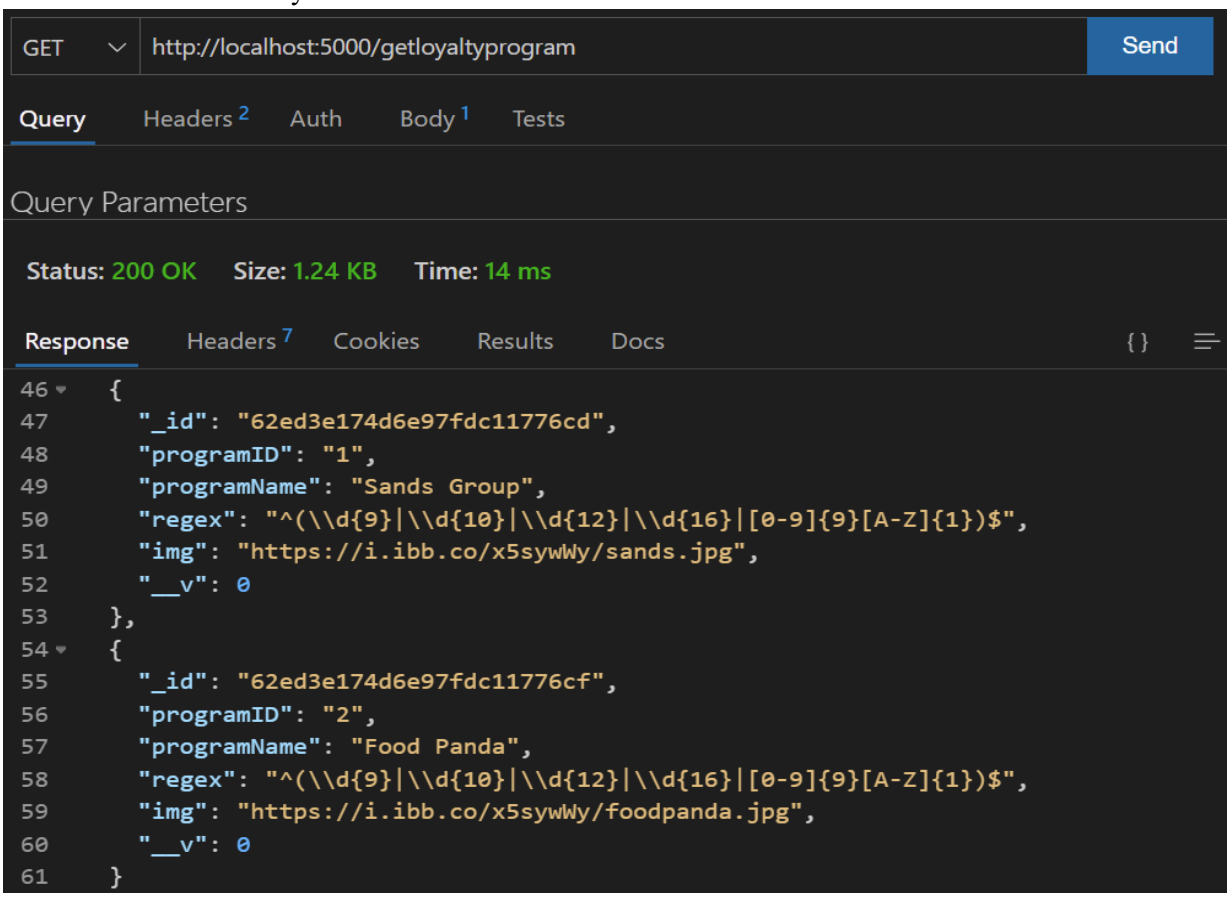
Interaction with the loyalty program database:

2 API endpoints were used in the interaction with the loyalty program database, POST - to add programs to it and GET - to get the current loyalty programs. We were able to use this to ensure that the correct loyalty programs were retrieved or inserted without having to create the frontend to do these functionalities.

To test POST functionality of the API: On success it posts new entries into the database



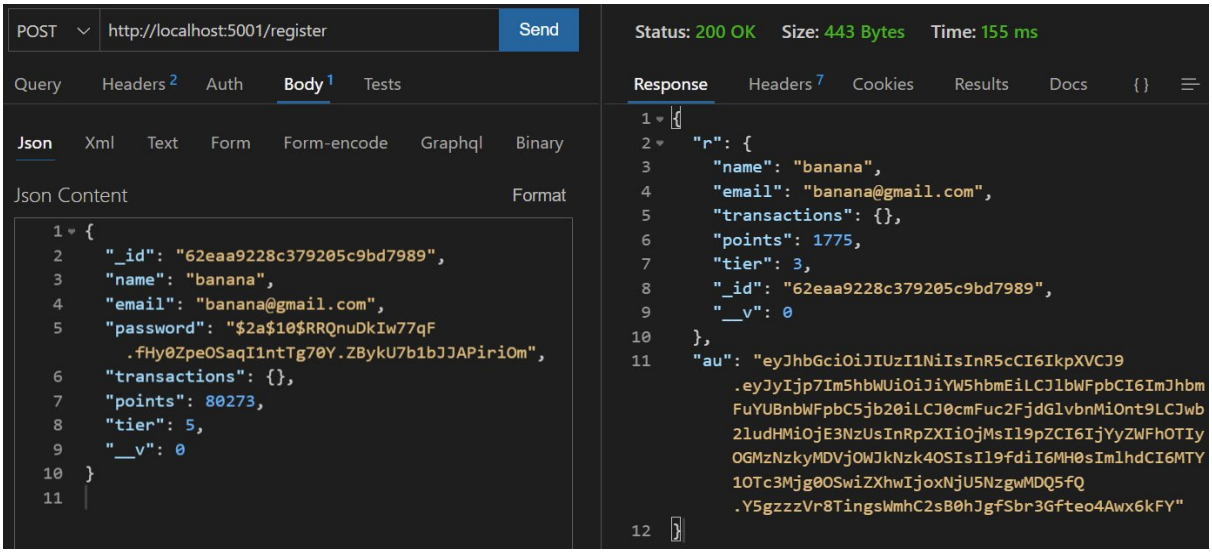
To test GET functionality of the API: On success it returns all entries in the database



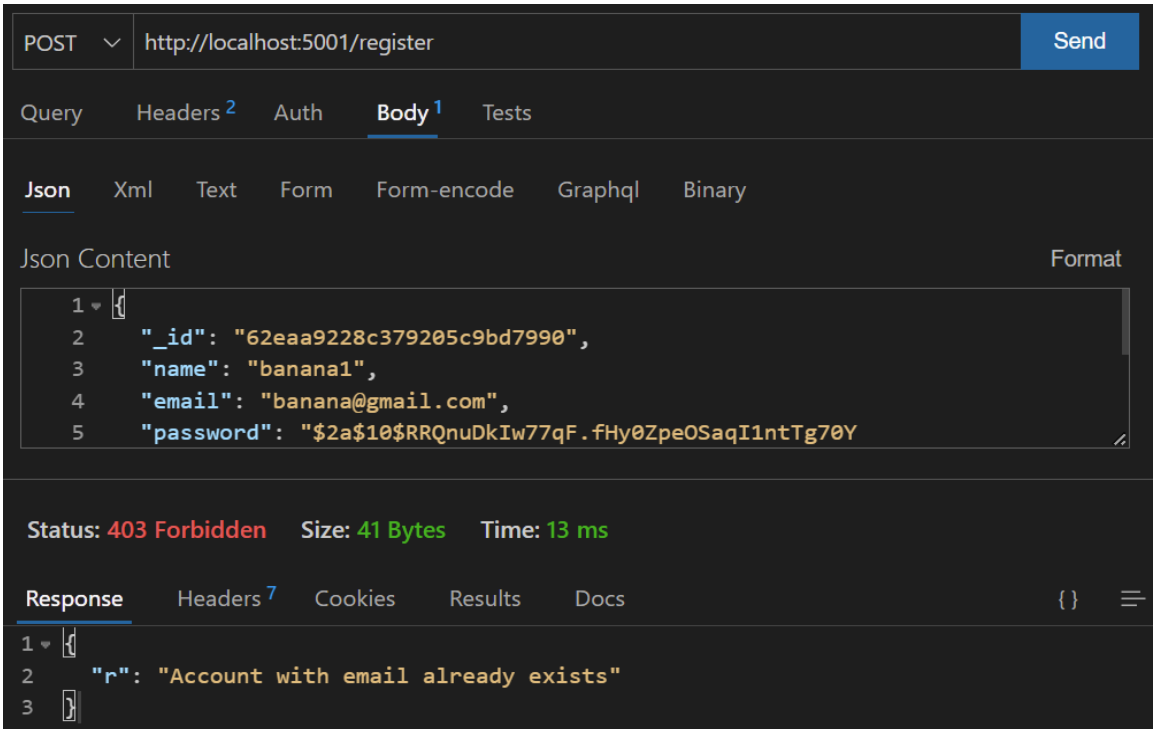
Interaction with the user database:

2 API endpoints were used in the interaction with the user's database, POST - to add users to it and GET - to get the current users. We were able to use this to ensure that the correct users were retrieved or inserted without having to create the frontend to do these functionalities.

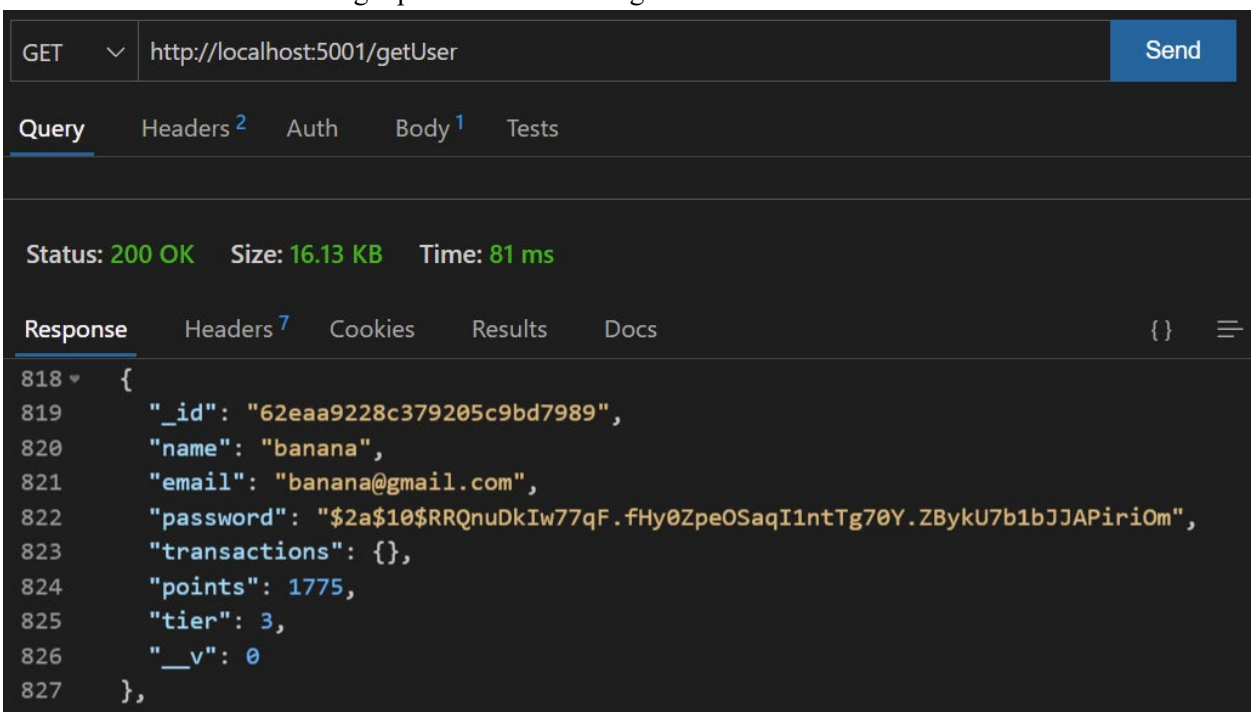
To test POST functionality: On success it creates a new user.



To ensure no duplicate emails, the test to check if the API is working as expected. The response returned is expected.



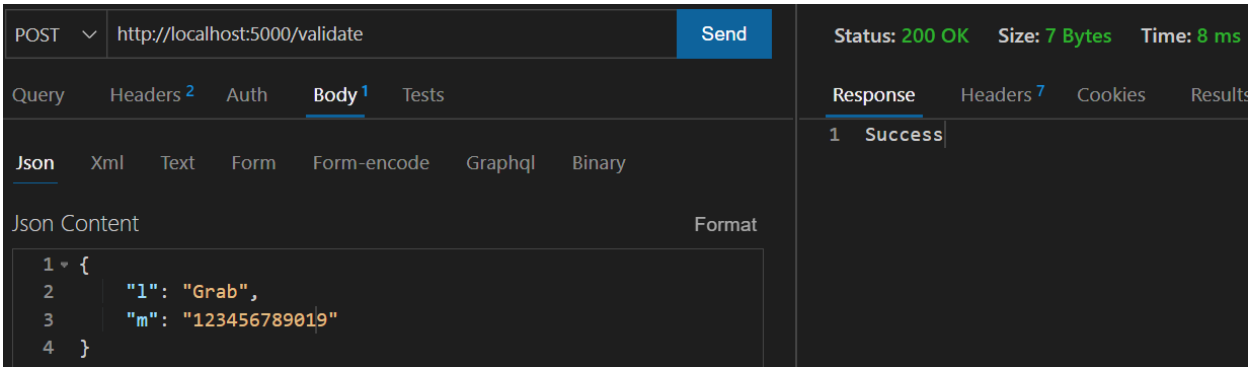
To test GET functionality: On success it returns the entered user, showing that registering users works without the need to create a signup form before testing it.

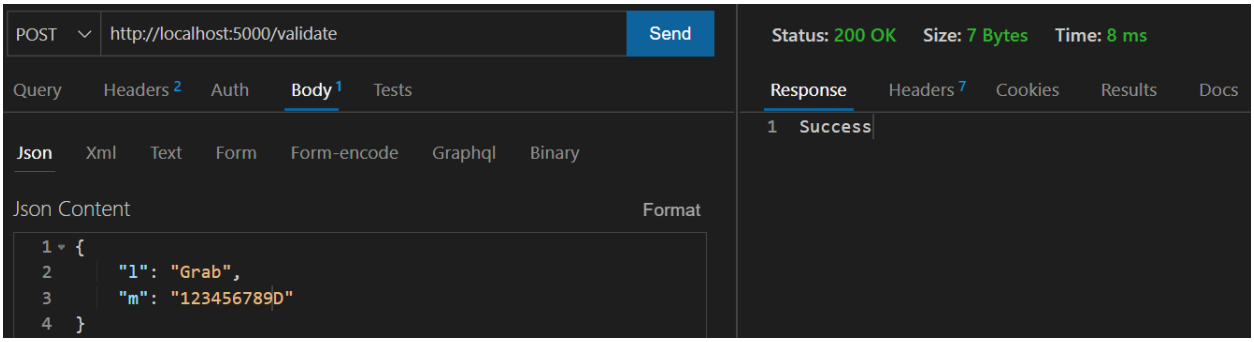


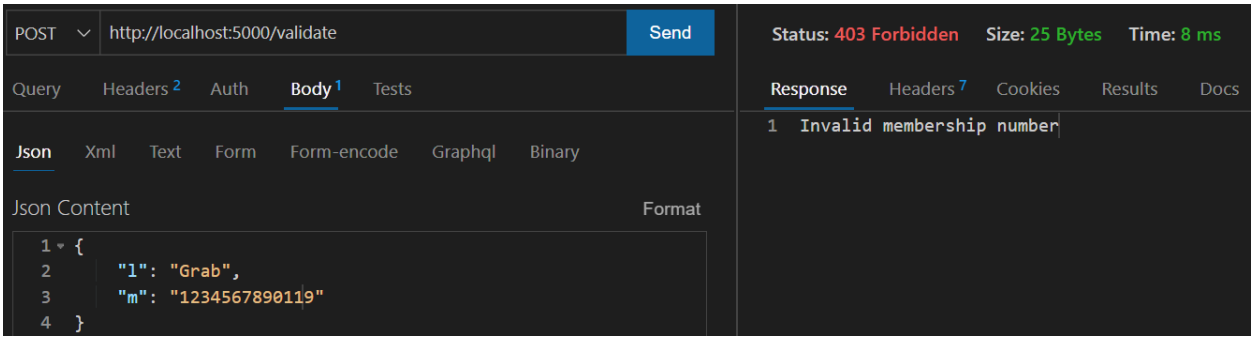
Testing of membership validation API endpoint

One of the non-functional requirements posed by the client was that our membership validation API should be as performant as possible, with the round-trip request from the bank on their frontend to the loyalty program database be ideally < 200ms. Postman was used again to validate this API endpoint, and to ensure that the API could return results in less than 200ms.

Here, 2 valid and 1 non-valid membership number denoted by “m”, is sent as POST request to the API.







This backend test successfully showed that the API only accepted certain regular expressions, with a response code of whether the request was successful or if it was an invalid format. At the same time, the time taken for the API to return a response was at maximum 10ms (8ms in the screenshot above) and is well within the performance specifications of the client.

Frontend Testing

Unit Tests

We will make use of Jest to test our individual React components (https://jestjs.io/docs/tutorial-react), specifically snapshot testing (https://jestjs.io/docs/snapshot-testing). Using snapshot testing, we aim to ensure that each webpage (e.g.: Home page, Loyalty Program page) renders correctly and consistently with the existing snapshots created.

Under the folder __tests__:

- About.test.js to test the About page
- Enquiry.test.js to test the Enquire Status page
- Home.test.js to test the landing home page
- NavBar.test.js to test the Navigation Bar of our web application
- __snapshots__ are the produced snapshots created after running npm test for the first time in the __tests__ folder

System Tests

We made use of Selenium Web Driver to implement automated tests on the frontend to simulate user interactions in cases where the user interacts with the website properly and in cases of misuse. Our aim was to ensure that our webpage will react accordingly in the way that it is supposed to.
In order to achieve this, first, for each feature, we created individual unit tests for each feature to test for a specific outcome or a certain misuse case. After ensuring that they worked as they should individually, we created tests that simulate user flow through the webpage. These tests consisted of either running through multiple features at once, to ensure that they worked when run together. Or they are a compilation of the unit test, to simulate the scenario where the user repeatedly makes a credit request, enquire their transactions, or submits invalid logins.

The files for the tests are located under the folder selenium_tests. To run these tests, cd to the folder and run node <test file name>.

Individual Tests:

Login:
To test the login page for the bank's webpage, we created two tests with Selenium. The first (login_test_success.js) with the login credentials for the existing testing account, test1. And the second test (login_test_failure.js) using credentials of a user, test2, that has yet to sign up for an account.
This was done to allow us to check that successful logins would bring the user to the bank's home page and incorrect logins would result in an error message.

Sign-Up:
We created four tests for the signup page, for cases when the user tries to submit an empty signup form (signup_test_emptyform.js), when an existing user tries to sign up again (signup_test_existinguser.js),

when the user has keyed in mismatching passwords (signup_test_passwordmismatch.js), and when the user successfully signs up (signup_test_succes.js).
This was done so that we could check that the correct error message would appear for the respective misuse case and that when signups are done correctly, that the user will be brought to the bank's homepage.

Transaction Enquiry:
When enquiring for the status of a transaction, there are 8 possible outcomes. To test each outcome, we created 8 tests, each searching for the respective reference code for the outcome, to ensure that the correct transaction appears when called with its reference code. We also created an additional test (enquiry_0098.js) to test for a successful outcome.

enquiry_0000.js → "20220000": "success",
enquiry_0001.js → "20220001": "member not found",
enquiry_0002.js → "20220002": "member name mismatch",
enquiry_0003.js → "20220003": "member account closed",
enquiry_0004.js → "20220004": "member account suspended",
enquiry_0005.js → "20220005": "member ineligible for accrual",
enquiry_0098.js → "20220098": "success",
enquiry_0099.js → "20220099": "unable to process, please contact support for more information",
enquiry_0100.js → "20220100": "pending"

Credit Request:
Two tests were created for the credit request feature, both to test for misuse cases. The first test (claimpoints_missingmembership.js), tests for when the user tries to submit the form without inputting his membership ID. The second test (claimpoints_missingpoints.js), tests for when the user tries to submit the form without inputting his membership ID.

Compiled Tests:

login_logout.js:
This test was done to test the logout button of the navigation bar and to simulate user flow when the user logins and logouts of their account

login_test_random.js
This test takes in an argument for the number of rounds of invalid login, where we made a randomizer to create random emails and passwords to be input into the login form and submitted. After the rounds of invalid login, Selenium then inputs the correct login credentials for the test1 account. The current number of iterations is 3.

signup_tests.js
This file is a compilation of tests in the signup page. The first test being when the user interacts with the 'Sign Up' and 'Sign In' buttons to navigate between the login and signup pages. Secondly, Selenium will attempt an empty signup form submission, followed by an existing user signup test, then a password mismatch test, and finally a successful signup test.

This was done to ensure that the signup page was secure under all cases.

homepage_test.js
This test runs through all the links in the homepage and navigation bar of our web application to ensure that the user will be brought to the correct pages after clicking each button. Eventually, the test logs out and returns to the login page.

claimpoints_randominput.js
In this test, Selenium navigates through login and the homepage to click the 'View Available Rewards' button to be brought to the page with the list of available loyalty programs. The test first runs through the possible misuse cases, followed by the correct inputs for the user to claim rewards. When claiming rewards, there are specific formats for the membership IDs, being a string of numbers of length 9, 10, 12 and 16, and a string of 9 numbers with a capital letter at the end. The Amount field of the claim rewards form also specifies a number more than 1.

Misuse cases:
1. Incorrect Membership ID length
2. Attempting to claim with 0 in the Amount field of the claim rewards form
3. Attempting to claim with a negative amount in the Amount field of the claim rewards form
4. Submitting the form with an empty Membership ID field
5. Submitting the form with an empty Amount field
6. Submitting the form with the amount specified being more than what your account has

Proper use cases:
1. Membership ID of 9 numbers and 1 capital letter with sufficient points for the amount specified
2. Membership ID of 9 numbers with sufficient points for the amount specified
3. Membership ID of 10 numbers with sufficient points for the amount specified
4. Membership ID of 12 numbers with sufficient points for the amount specified
5. Membership ID of 16 numbers with sufficient points for the amount specified

enquiry_everything.js
In this test, we navigate through the login and homepage to the navigation bar and click on the 'Enquire Status' button. Selenium will first sort through the transaction outcomes in descending, then ascending order by clicking the arrow button under Outcome. Then, after undoing the sorting of the transactions, Selenium will run through the 9 tests for transaction enquiry as well as searching for transactions through the outcomes of SUCCESS, FAIL, and PENDING.

Robustness Testing

Fuzzing

We created one fuzzer to be used for all the fuzzer tests under **fuzzyFuzzer.js**.

We then created a fuzzer test for every possible user input field in our web application. Each fuzzer test takes in argument for the number of rounds of fuzzed input to be submitted in the forms. The length of fuzzer input increases every 10 rounds of fuzzing.

fuzzer_creditreq.js
To fuzz the fields of the claim rewards form.

fuzzer_enquiry.js
To fuzz the transaction enquiry search fields for reference code and outcome.

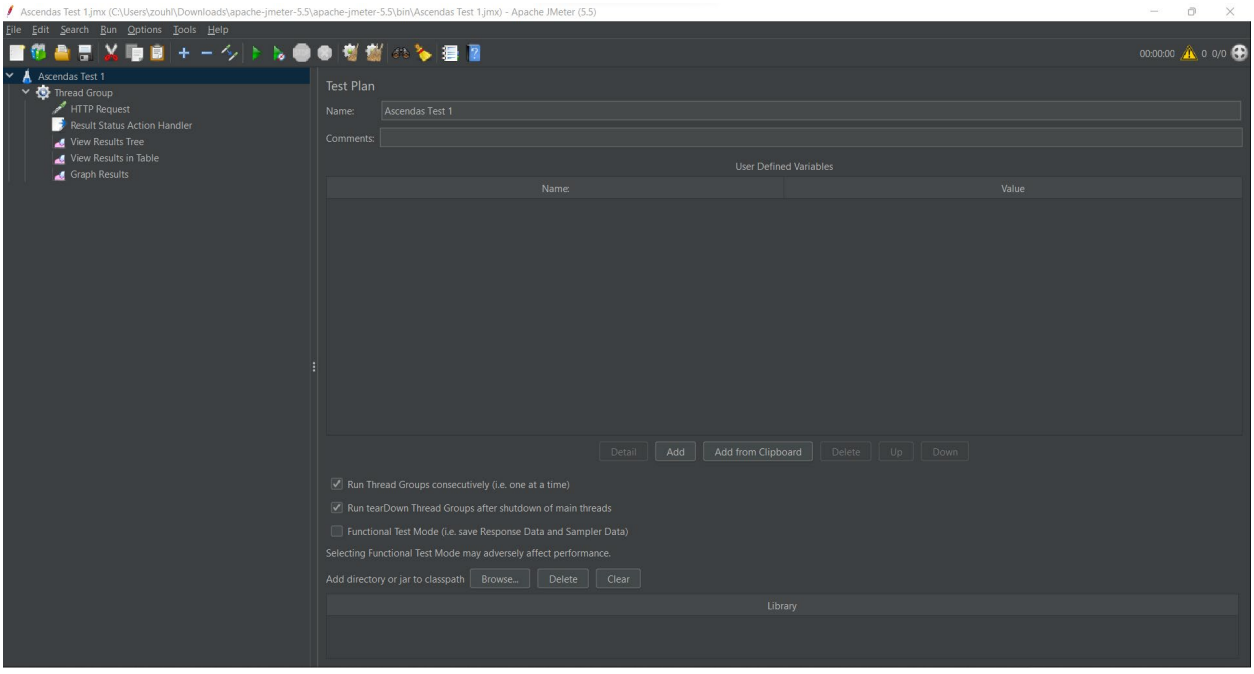
fuzzer_login.js
To fuzz the email and password fields of the login form.

fuzzer_signup.js
To fuzz the email, name, and password fields of the signup form.

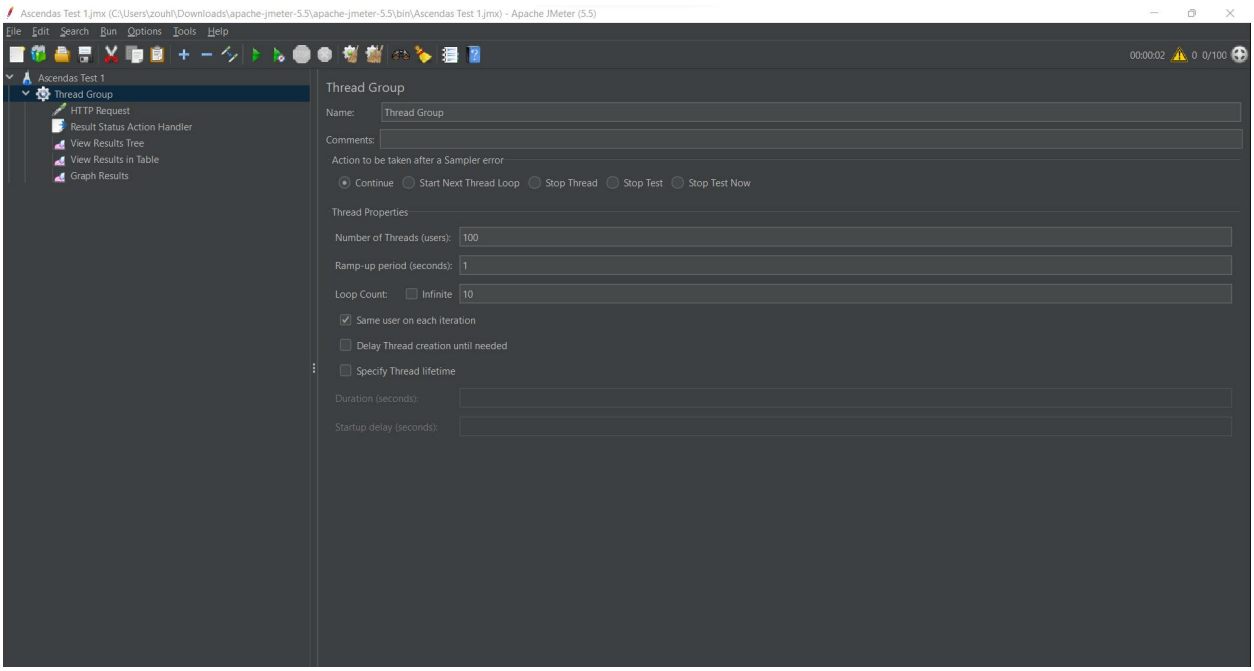
JMeter

To further test the robustness of our webpage, we also made use of the JMeter software for load testing of our web application. Load testing is the act of putting simulated demand on our software program by letting a large number of users access the application simultaneously.

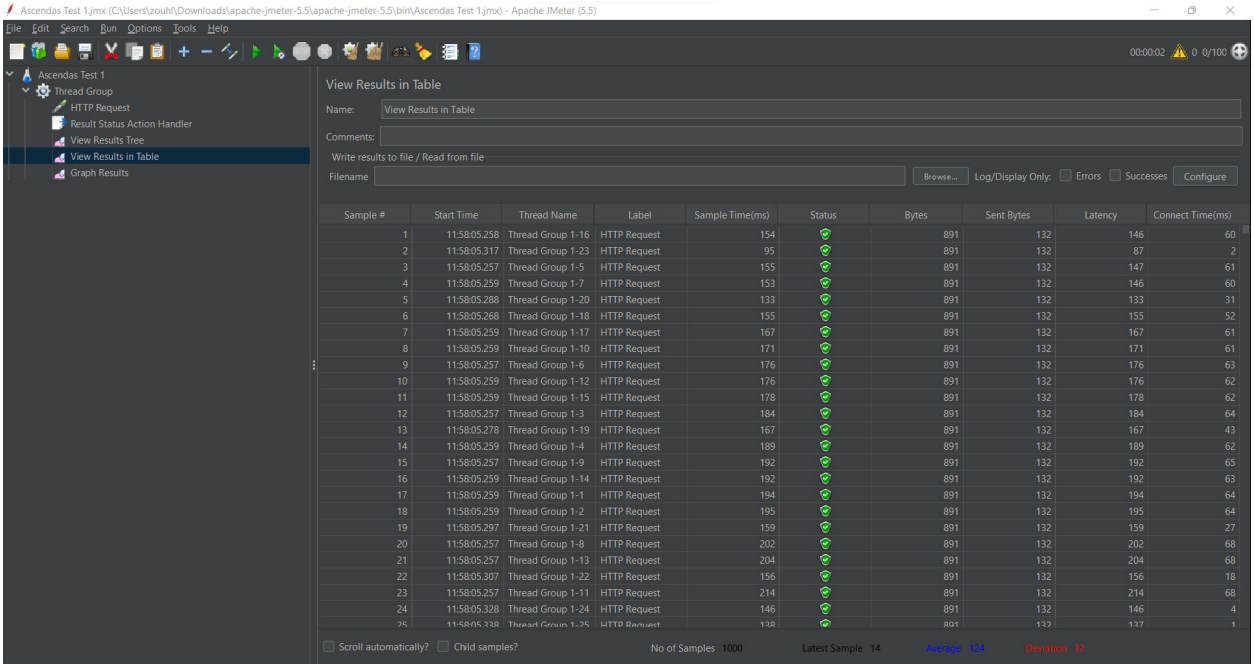
Using this, we could determine the number of concurrent users the application can support without deterioration in performance.



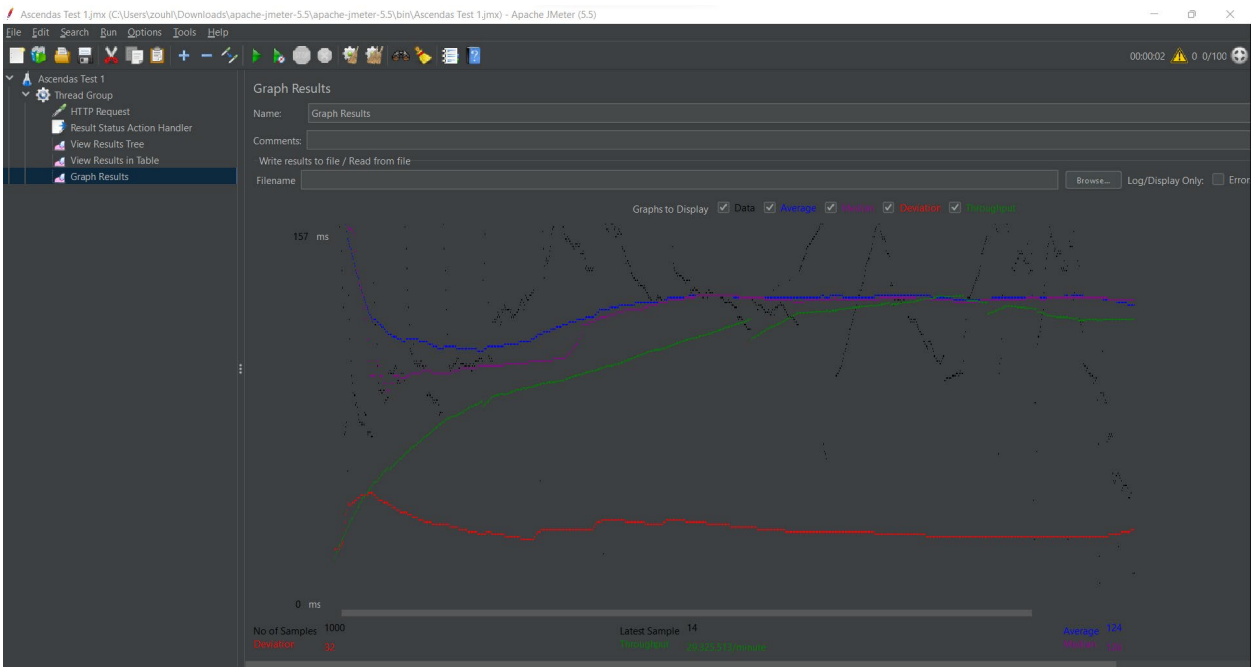
We tested for 100 threads (users) for 10 rounds:



The results in a table (first 25):



The results in a graph:



This showed that the response time for the last request sample is 157ms, and our system can be reasonably performance with 100 users on site at a time.

Regression Testing

Regression testing, a type of blackbox system testing technique was also extensively used and was key in the success of the project. During integration of new features, sometimes modification, upgrading or improvements of code could cause any existing functionality to not work as expected. Hence, our testing plan also involved using regression testing to ensure overall stability and functionality of our existing features and doing so ensured that the entire system stays sustainable under continuous improvements.

Unit testing and individual system tests were useful in testing new individual functionality, but they do not account for how compatible they are with the existing ones. With regression testing, it was the final step to ensure that the product behaves as a whole during integration of new elements and features.

Since regression testing involves a lot of specific use cases to test and can get repetitive due to the continuous testing of old features, we had to rely on automated tests developed using selenium, which greatly reduced the amount of time and effort required by us to perform these tests manually. The suite of automated tests that we developed using selenium, together with some key manual tests that cannot be automatically checked by automation were executed every time there was major refactoring of code.

Some findings were that regression system tests were very useful in pinpointing if any changes or implementation of new code features broke the existing code. This could immediately be pinpointed and rectified instantly, saving time and improving efficiency. This prevents any uncaught bugs arising from code refactoring from accumulating and eventually causing the entire system to be full of code smells, which would then be too late and troublesome to resolve bugs.

Final Remarks on Testing

Software testing was crucial in our software development journey. By conducting extensive tests to ensure that our frontend and backend was working, we could ensure that the development of our application went as smoothly as possible.

Using unit testing, we verified that the individual components function as expected. Using postman to test our API endpoints ensured that the backend was functional and ready to be integrated with the frontend.

Using robustness testing in the form of fuzzing ensured that our application was more reliable and edge cases were taken care of and load tests mimicked many users using the application at the same time.

Using system tests allowed us to ensure that individual systems were working as expected, and some were combined to test whether interacting subsystems work as expected. Using automation for this greatly reduced the amount of effort required.

Regression testing, the final and most important step after each continual upgrading of code, involved all the above tests and ensured that different systems could work together as expected, at a reasonable performance level, and functions as we expect it to be. Eventually, when the application was assembled, it was performed once more to ensure a fully functional website.

Lessons learnt

From Project Meeting 1, we stated back then that the software development process that we would apply is the agile methodology. Agile methodology ensured that we were able to focus on incremental and small updates to deliver our assigned work before the next meeting.

The main challenges faced in following the agile methodology would be preventing agile burnout. Agile burnout tends to happen when team members overcommit early on, and sometimes in order to meet commitments to the team, will sacrifice sleep for work. Since agile requires us to meet often, the time commitment is very high and demands a lot from the entire team. This could lead to members being drained and disengaged and have a negative effect on the progress of the project and mental health.

To prevent agile burnout and overcome the challenge, we did proper sprint planning to ensure that the features to be implemented can be realistically completed before the next meeting. We also met consistently to prevent work from piling up and kept meetings light-hearted, ensuring that everyone was on the same page and knew what to do. We also made sure that the deliverables assigned to everyone were not overwhelming and can be done within a reasonable amount of time.

The main takeaway would be that following closely to the agile methodology was a key factor in ensuring that the project went well. We were also aware of the challenges and demands of following the agile methodology, and it was important that we made the effort to address those challenges to ensure successful completion of the project.

From a testing perspective, one of the lessons learnt is that testing should be as automated as possible to accelerate the pace of testing when there is refactoring of code. Test automation shortens testing cycles, allows for greater test coverage, and reduces manpower needed for tests.