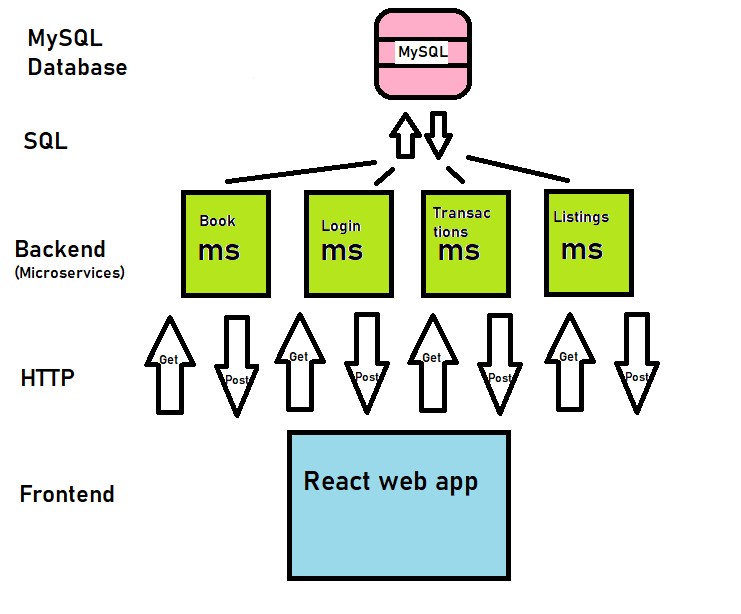
**Team 3 | Friday 10:30am | Tutor: Dipto**

**Vision Statement**

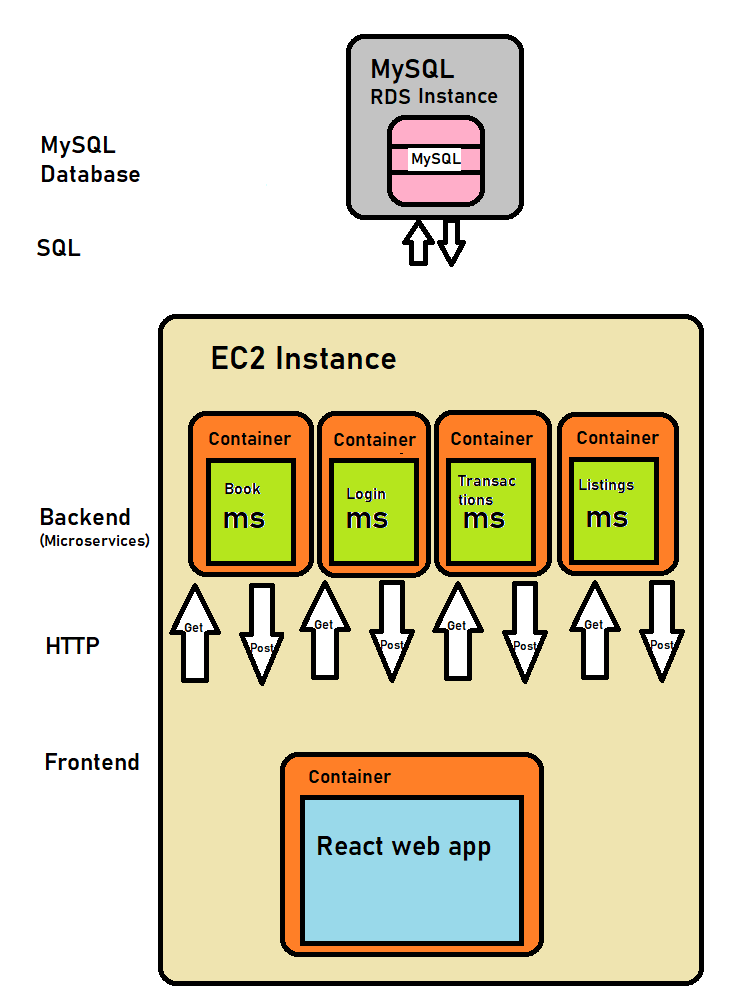
Bookeroo allows students and professionals who need to buy books to buy as well as sell books as a customer, which increases their interest in reading and studying more. Bookeroo is an online web application that allows numerous shop owners and ordinary members of the public to sell books and keep track of transactions and orders. Bookeroo also offers three separate user types: admin, shop owner, and customer. It also includes the PayPal API, which is used to process payments for books purchased by customers. Bookeroo shows all the books and allows authorised users to add new ones. It also has a function called Sell Books that allows you to sell your books based on their condition. Admin users, for example, could produce reports for transactions, books, and so on. This would allow the administrator to keep track of how the programme is performing and whether or not there are any issues. Owners of shops have their own sub-features, such as adding books, reviewing transactions, and selling new/old books. Similarly, public users are permitted to purchase books, sell old books, and examine the transactions they have completed for the books they have sold and purchased. There are other extra options, such as searching for books, classifying books, and adding reviews, that allow users to browse through them and determine which book is right for them. They can also write reviews, which allow people to express their thoughts on a variety of topics such as book condition, pricing, and content. As previously said, PayPal is utilised for checking out, and the books can be put to cart, with the buyer having the option to withdraw their selections. Then customers must provide their PayPal information, which will take money from their bank account when they have purchased the books. The programme is hosted on AWS, which provides a gratifying cloud experience that is very user-friendly. All of these aspects in an app make it well-designed and feature-rich, with the potential to help a large number of users. These characteristics make this programme both useful and efficient for the user.

**Architecture**

Application Architecture



Cloud Architecture



**Refactoring**

**Front-End**

1.

As we know, we have three types of users. Customers, Shop Owners and Admin. Initially, I had divided the Dashboard feature of each user such that different users have links to different files for dashboard. After Refactoring the code, I managed to put all the routes in a single Dashboard file. This was possible by creating a method in the actions folder which returns the current userType. Using this usertype I was able to only make the routes available which are for that specific user.

**Code-Smell Before:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**After:**

Text

Description automatically generated

2.

I originally had public routes which would make certain routes public for everyone and then I had to implement other files which would account for private and restricted route. To overcome this repetition, I created a page, on which when it goes to a private page for example, “Dashboard”, it checks whether the user is logged in or not. If the user is not logged in, then it is directed to the login page to login and then get an access to the Dashboard.

**Code-Smell Before:**

Text

Description automatically generated

**After:**

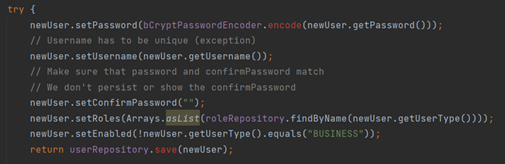
Text

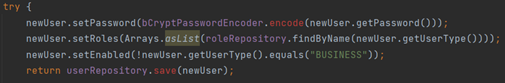
Description automatically generated

Text

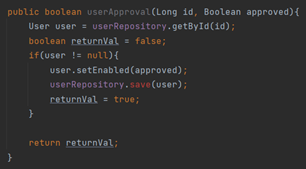
Description automatically generated

**Back-end**





The provided code block pertains to saving a user within the database. The setUsername line is completely redundant, whereas the setPassword line is also unnecessary. Given that under the user model, the confirmPassword field is tagged as @Transient, the confirmPassword field will never be persisted within the database.





Given that the approveUser and blockUser methods were practically identitical, they had been combined into the one userApproval method for the sake of eliminating code repetition. The logic remains the same, whilst a Boolean value for whether the user is approved or blocked is also passed into the method.

**Git + GitFlow**

**Branches Breakdown:**

|  |  |
| --- | --- |
| Branch | Use |
| master | Major release branch, committed only when Sprint finished |
| develop | Working branch for most recent, stable, development version of the app. |
| release-sprint-1 test-deploy | Release branches at end of sprint. |
| backend-transactions,  backend-usergroups,  bookSearch,  bookindex  cart  feature-books  frontend-dashboard  frontend-reg  listings  reviews  transactions | Feature branches. We kept these for marking purposes, but in the real world we would delete them after they are out of use. |

**GitFlow visualization**

<https://app.gfc.io/github/s3847243/Bookeroo>

We followed the GitFlow standards to collaborative development. Master was only committed to at the end of each milestone. Develop was the main branch for working on. Each member made a feature branch of develop to work on features. After the feature was complete, this was merged into develop.

**Scrum Process**

The Scrum Master is Jeffrey, the group met weekly in the tutorial, chatted via Discord, and held a meeting on Discord once a week.

Jeffrey managed the project, as well as developing the app itself. Team members caught up and discusses responsibilities during the stand-up meetings.

The team conducted a planning meeting at the start of the Sprint, where team members discussed sprint backlog items, assigning them story points. For the retrospective, the team gave constructive criticism to one another, and discussed what could be done to be a better and more efficient development team during the next sprint.

**Deployment Pipeline**

**­**Each microservice has a Dockerfile

Text

Description automatically generated

Text

Description automatically generated

React app also has a Dockerfile

Text

Description automatically generated

Repository base directory has a .circleci directory and config.yml file.

Each microservice is built into a Docker image, tested, and deployed onto DockerHub.

Text

Description automatically generated

Same goes for the frontend.

Text

Description automatically generated

The success of this process is confirmed or denied by CircleCI dashboard.

Graphical user interface, text, application

Description automatically generated

Our EC2 instance has Docker installed following,

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html>

The docker service is run.

We copy the docker-compose.yml file into the EC2 directory, then run

`docker-compose up –detach`

then ,

`docker-compose ps`

to confirm the containers are running.

**Test Plan**

**Frontend**

Last Sprint we had 31 total tests for the fontend.

Graphical user interface, text

Description automatically generated

This Sprint, we added 12 tests.

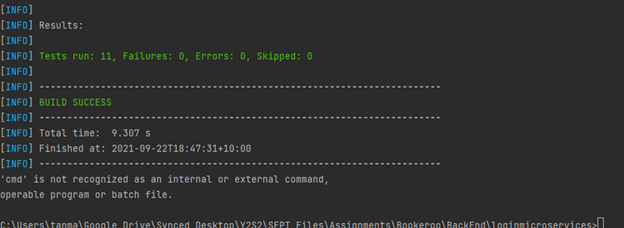
Text

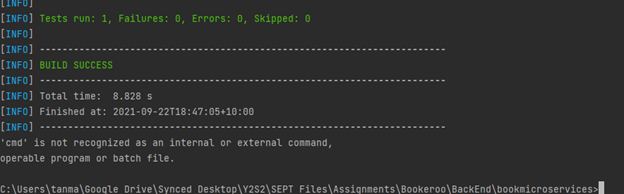
Description automatically generated

**Backend**

We added the Transactions and the Listings microservices this Sprint.

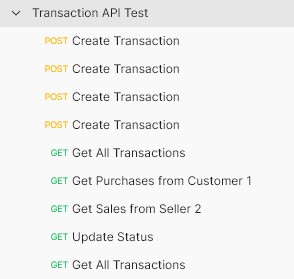
**Unit Tests**



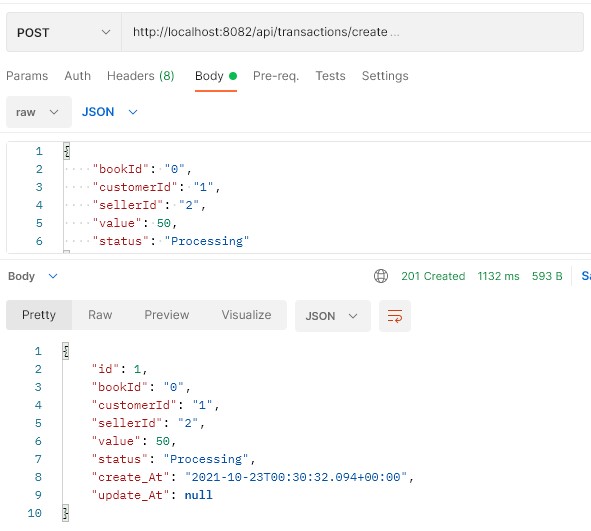


**Postman tests for transactions:**

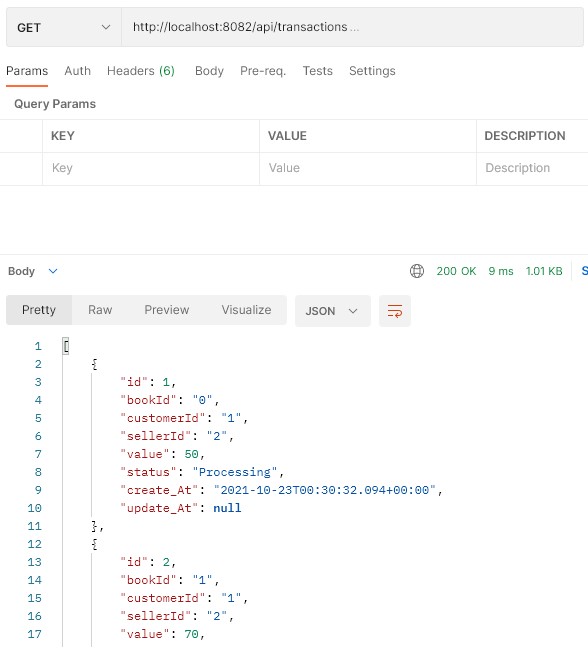
1. Plan



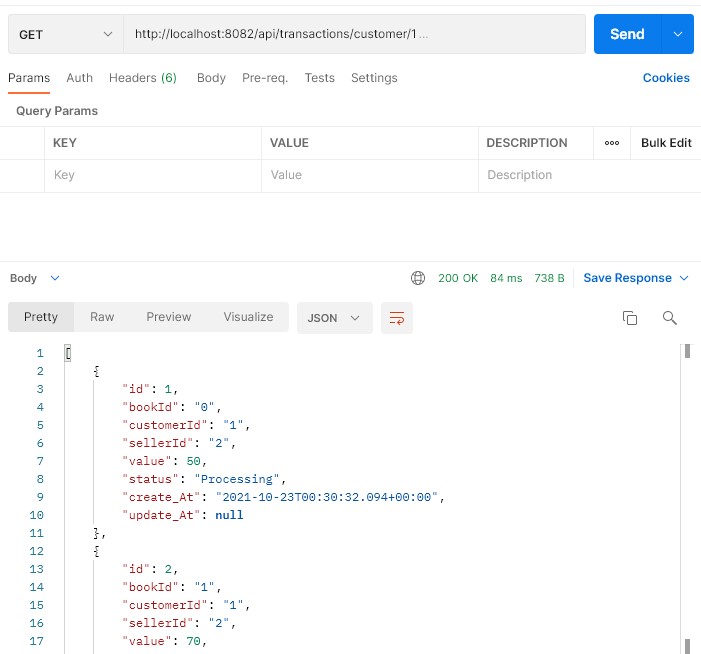
1. Create Transaction



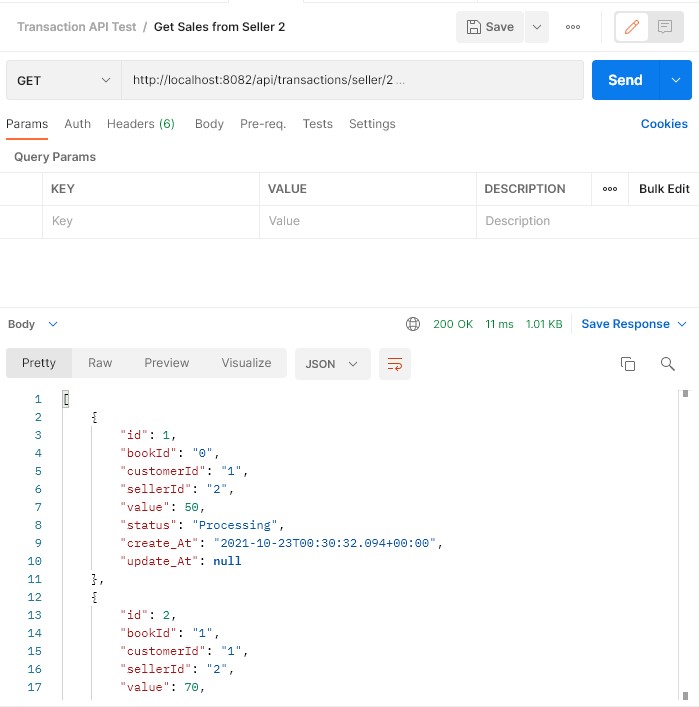
1. Get all transactions



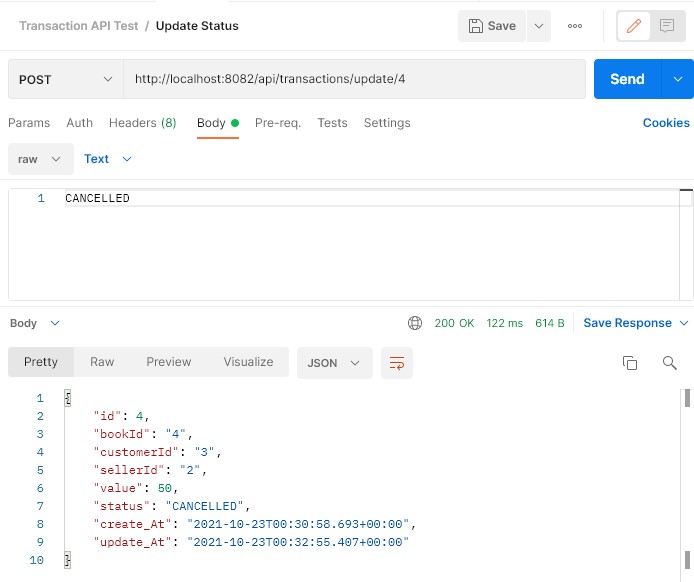
1. Get Transactions where user is customer



1. Get Transactions where user is seller

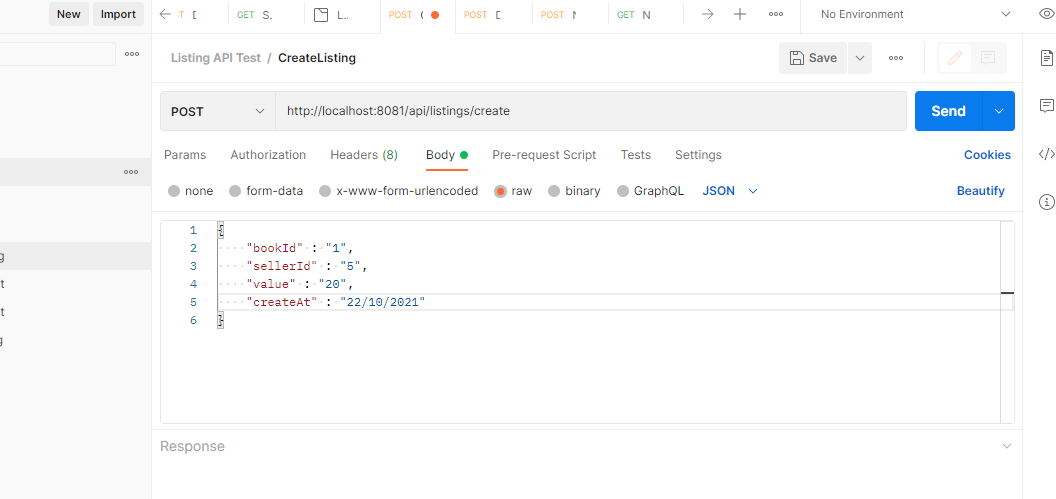


1. Update a transaction

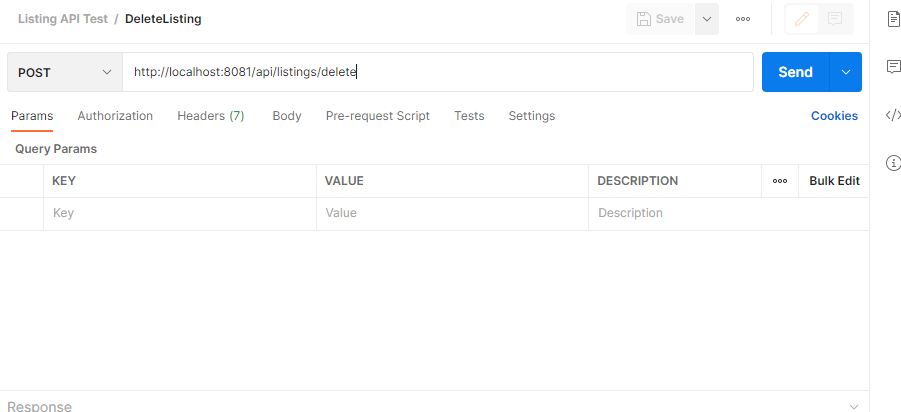


**Postman tests for Listings:**

1. Create listing



1. Delete listing



**User Acceptance Tests + Integration Tests**

Please see the file UAT Milestone 3.xlsx

**LOG4J2**

