Software Engineering: Processes and Tools (COSC2299) - Milestone 2

SEPT-MON-5.30-Ujj-Group1

Lachlan Furlong - s3722243 (20%)
Dean Bolte - s3784365 (20%)
Lucas Mellor - s3720988 (20%)
Edvin Berberovic - s3722163 (20%)
Matthew Moloney - s3717566 (20%)

Overview

Github
https://github.com/RMIT-SEPT/majorproject-8-mon-17-30-1

Trello
https://trello.com/invite/b/0IMFERnp/c2dc85c624da354b52967072b11d4272/sprint-board

AWS deployment (frontend)
http://sept-frontend.s3-website-us-east-1.amazonaws.com/

AWS deployment (backend)
sept-backend-env.eba-zmub6gjh.us-east-1.elasticbeanstalk.com:80/api/v1/

CI/CD
*have been given 2 day extension from Homy

Milestone features

Stories

A summary of new user stories that were implemented as tasks in our backlog include:

- Register as customer and admin
- Admin can register workers
- Admin can create services
- View booking history
- Admin can view services
- Admin can view workers
- Admin can view all bookings
- Customers can book services
- Customer can cancel bookings

Functionality

A summary of the functionality implemented from the user stories in

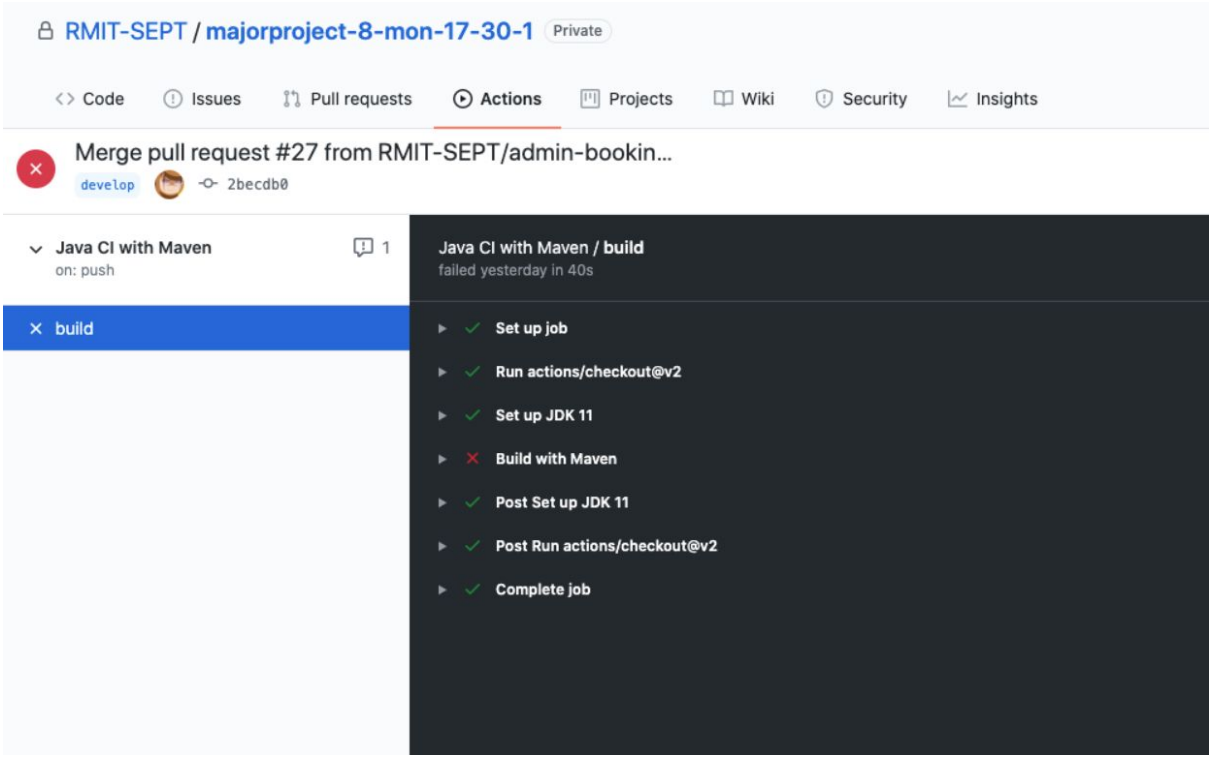
- Customer, Admin registration
- Admin functionality with registering workers and creating services
- Admin exclusive view in viewing all services, workers and bookings
- Customers can create a booking and cancel booking, then view it in their history.

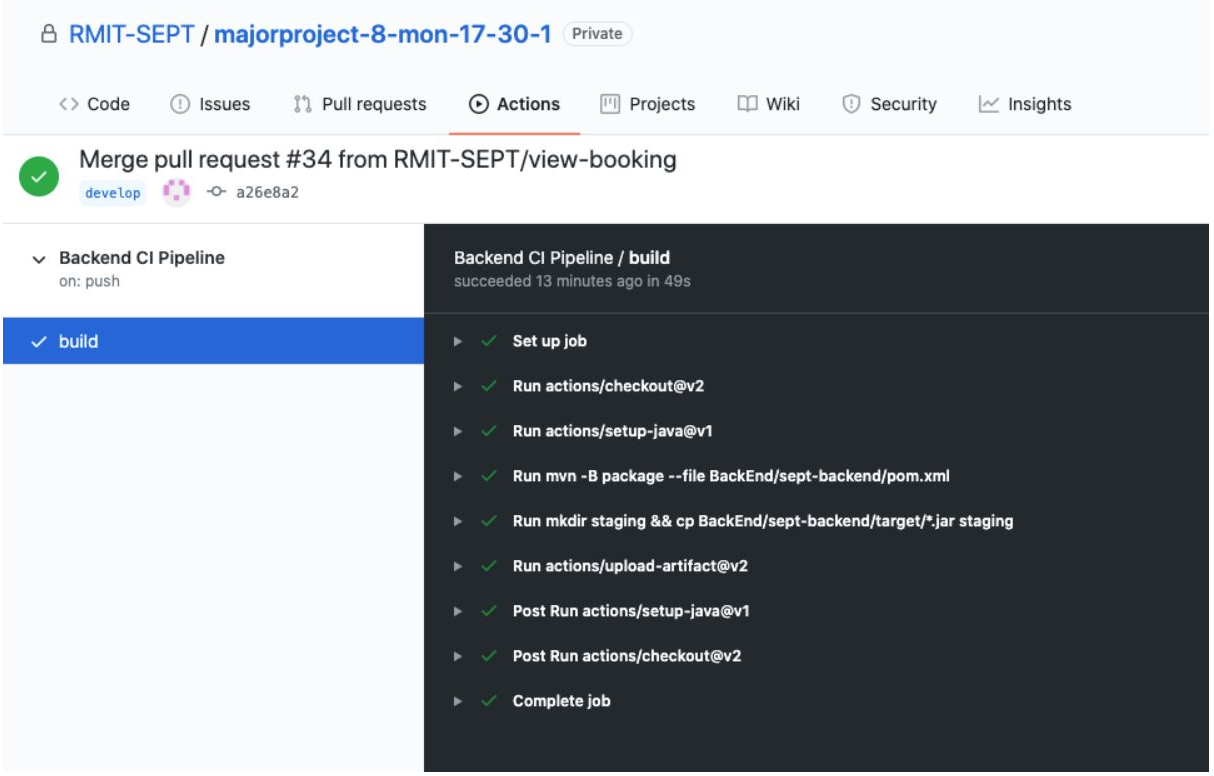
Tests

Tests are implemented for each of the new functionality. The test files include:

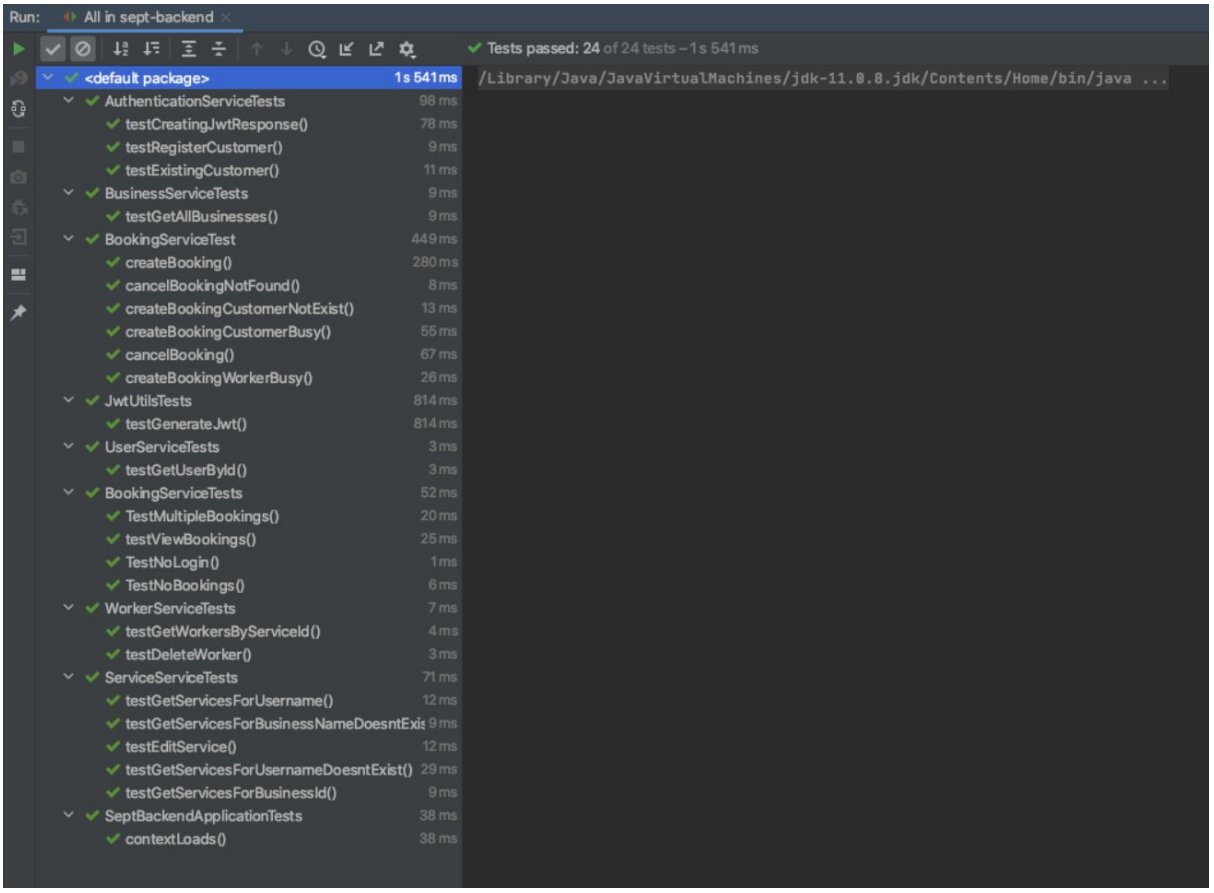
- AuthenticationServiceTests for registration and login functionality
- BookingServiceTests to test create booking and the edge cases, cancel booking, attempting to create a booking without logging in.
- ServiceServiceTests to test the repository is returning the correct information when requested with various identifiers such as businessId or workerUsername.
- UserServiceTests to test the repository is returning the right user when requested.
- WorkerServiceTests to test the repository is returning the correct workers from the serviceId identifier.

Evidence of CI





Evidence of tests



Team member duties

- Lachlan Furlong - s3722243
    - Defined the layers of the backend (service, controller, repository, etc.)
    - Redefined the relationship of the entities to match updated business requirements
    - Defined layout for testing
    - Implemented CRUD for AGME services in both frontend and backend
    - Defined axios usage in frontend
- Dean Bolte - s3784365
    - Defined layout for frontend testing with Jest
    - Implemented frontend unit testing
    - Managed asynchronous requests in axios
    - Managed issues for frontend react implementation
    - Implemented booking services in frontend
- Lucas Mellor - s3720988
    - Implemented front and backend implementation for viewing active bookings as a customer
    - Implemented front and backend implementation for deleting services as an admin
    - Implemented front and backend implementation for deleting bookings as a customer
    - Implemented backend implementation for deleting a worker as an admin
    - Wrote Unit tests for cancel booking functionality
- Edvin Berberovic - s3722163
    - Front and backend implementation for cancel Booking
    - Wrote Unit tests for cancel booking functions
    - Deployed backend service to AWS (Elastic Beanstalk)
    - Deployed frontend service to AWS (S3)
    - Connected frontend and backend on AWS using cloudfront
    - Implemented CI for project backend
- Matthew Moloney - s3717566
    - Frontend and backend implementation for viewing workers
    - Frontend and backend implementation for viewing customer booking history
    - Frontend and backend implementation for viewing all booking history for an admin
    - Frontend and backend implementation for creating a new worker
    - Frontend implementation for deleting a worker
    - Backend implementation for editing a worker
    - Unit testing for create booking functionality

Design documentation

For our overarching microservices design, we are using Spring Boot with an integrated H2 database for our backend, and React.js for our frontend.

The Spring Boot backend is where the main business logic for AGME resides, enabling persistence of data and a unified interface for performing operations. This interface is implemented using HTTP REST, allowing for create, read, update and delete (CRUD) operations. This provides the benefit of being able to run local development environments that can be harnessed with a collection of HTTP requests to test integrated functionality.

The code within the backend is structured and managed using high cohesion and low coupling, such that as different features are being built, there is a level or abstraction that allows developers to work collaboratively within creating serious conflicts in version control. For example, there is a controller layer that exposes the application to HTTP requests. This allows the outwards facing interface to be completely independent of the actual implementation within the rest of the backend. The next level of computation is the service layer, which pertains to most of the business logic. The service layer is responsible for performing operations, such creating a new customer, or determining the availability of a new booking, and persisting any data operations in the database. Using Spring JPA, we are easily able to integrate database management into the code, via a repository layer, using a vendor-agnostic approach. This means that as we migrate from the H2 database to a dedicated deployed database, such as MySQL, the repository layer and thus its integration in the service layer will remain unaffected. On top of the benefits of these layers of abstraction, it also allows for testing singular pieces of functionality, and then mocking dependent functions. We are implementing the testing using JUnit5.

The frontend is similarly structured into an application layer, a component layer and a service layer. The application layer is responsible for managing the structure of the application, such as routing endpoints to components, and managing the landing page of users. The component layer is responsible for displaying the view of the application to the user, and managing any other components within the page. As the user interacts with the page, the component makes calls to the service layer. It is the service layer that makes HTTP requests to the backend, and then filling out information in the page.

As changes propagate to the source of both of these applications, an included CI pipeline is run when a branch is merged within the main repository trunk. This pipeline is implemented using GitHub actions, where all unit and integration tests are run for both the frontend and backend, and packaged into an artifact that will be deployed using CD. We currently have a one-time deployment instantiated for our applications, hosting the frontend using a static-host in an S3 bucket with a CloudFront layer behind it, and the backend deployed to AWS Beanstalk. As we integrate Docker into our services, we will eventually be able to deploy using AWS ECS, or even AWS EKS.