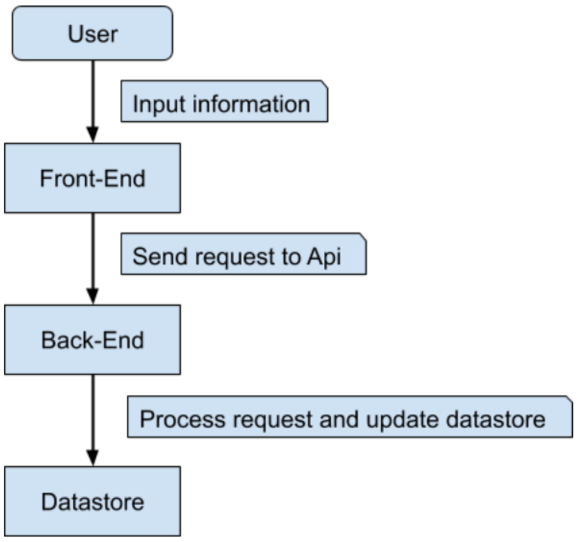
Members: Daniel Miskimmin (s3722763), Kejie Wang (s3716098), Sandra Oommen (s3794692)

Project Report

Vision

For our product, all though not fully functional overall product, both the front and back end do give a good idea of the structure of the program and what a final product will look like. The front end is clear, with the functionality of the website clear and not hidden away behind a lot of menus. The overall feel of the website is clean, with a nice flow to it and as a result, there is nothing jarring the user experience. It's important to have a website like is as it means that the user will not be discouraged from using the website and also not dread using the website. The same can be said for the backend implementation whereby the interactions with the backend are simplistic, with the combination of endpoints and HTTP request types leading to self-explanatory operations on the backend. As a result, different front ends can be added to the application easily as a direct result of this simplicity. This is a direct result of using a REST api structure for the backend. What it also means is that the program is easily scalable, which for this product is done though Kubernetes. This scalability is good for two reasons, with the first being that the program can easily increase the number of instances of the program as the demand/use of the program increases, meaning that the end user will not experience a bad experience as a result of the available resources being overwhelmed. The second benefit is the opposite to the first, whereby the program can decrease the number of instances of the program running, decreasing the running costs of the program when the use of the program is low. The low coupling between the frontend and backend mean that the program is also easily updatable with both the frontend and backend having the ability to be updated separately. It is a benefit as it allows for patches for bugs to be implemented separately as well as feature deployments to be deployed separately, and so only one needs to be updated if changes occurred to one of them.



Final System architecture

Our program, both the front end and back end did not have any refactoring occur, with there being nothing fundamentally broken or restrictive in any way. The reason for this is that the original implementation for the front and backend allowed for additional features to be added easily and with as little modification of the original code as possible. For the backend this was done by having all the functionality of the processing of the HTTP requests and modifications to the database being separated. For the front end this was done by having the different functionality of the requests to the backend as well as the functionality of the actual website being separated.

What was refactored though was the CircleCI configuration file as well as infrastructure code for AWS. This was done between milestone 2 and milestone 3 and changed the program deployment from using a manual ec2 instance with docker installed to it, to a Kubernetes deployment using helm to deploy the program in the Kubernetes cluster. The CircleCI changes were to make the workflow of the CI to be more modular, making it easier to implement things like the testing of the front end as well as the uploading of the containers created from the CI part of the CircleCI workflow. The infrastructure code needed to be changed as a lot of the infrastructure used was going to be handled instead by kubectl. The terraform configuration files therefore needed to be split into two different instances of terraform. The first sets up the s3 bucket which will be used to store the state of the Kubernetes cluster. The second config will setup the RDS which will be used by the backends of the program, using the same VPC and subnet as the Kubernetes cluster and therefore must be applied after kubectl has set up the Kubernetes cluster.

Overview of Gitflow organization

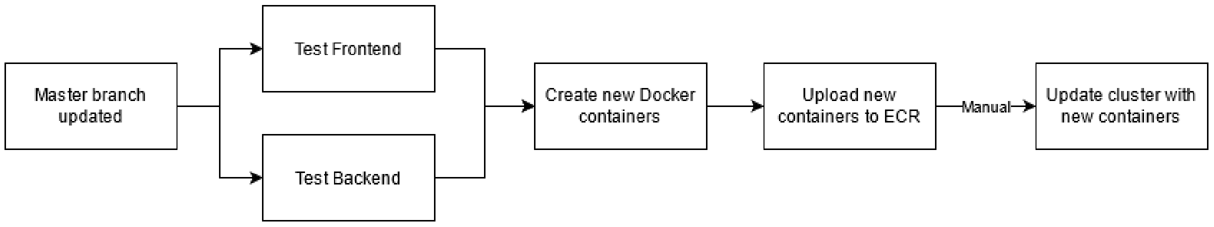
The Gitflow of the project is one of the low points of the project, with the branching inconsistent and there being pushed directly to the master branch. These are not something which should not be done in the Gitflow workflow. The commits as well were also inconsistent, where the commits to the git repository were done sporadically and usually around where the milestones occurred. This created problems in the development process as it meant that linking the front and backend together became harder as each end didn't have the other as a reference to see how they needed to work and testing for the interactions couldn't be done. Many bugs were therefore created, or in other words could be avoided but weren’t, if more commits happened more often.

Scrum Process

Our overall goal for the rate that we would meet during the week was twice a week, including the lab session on Friday. Originally this did occur, with the second meeting being on a Thursday, but we found out after a few weeks that having the second meeting time as Thursday meant that the overall coordination of the program development was reduced. We then changed the time to a Tuesday or Monday, depending on the schedules of members of the group. When our group was effectively 2 people, we did try to organise times to meet, even though the level of coordination required was less, as we worked on separate areas. This though did eventually devolve into only meeting once on Fridays. When Kejie joined near the end of the semester, we still didn't have the second meeting, but did though use Microsoft Teams to keep in contact and make sure we were doing work.

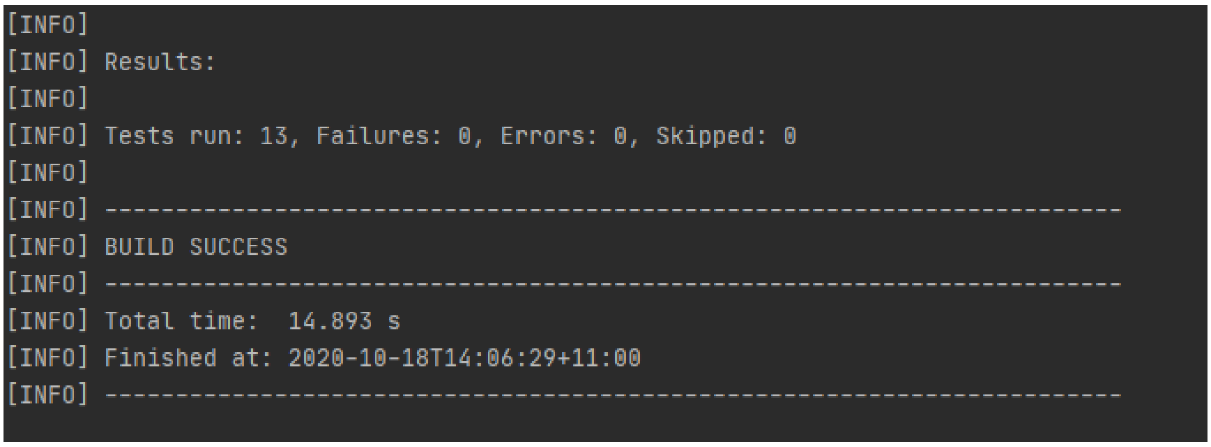
Diagram of deployment pipeline

Most of the processes and transitions between the different stages of the deployment pipeline is automatic, except the updating of the containers. This is done manually using helm.


Master branch updated
Test Frontend
Test Backend
Create new Docker containers
Upload new containers to ECR
Manual
Update cluster with new containers

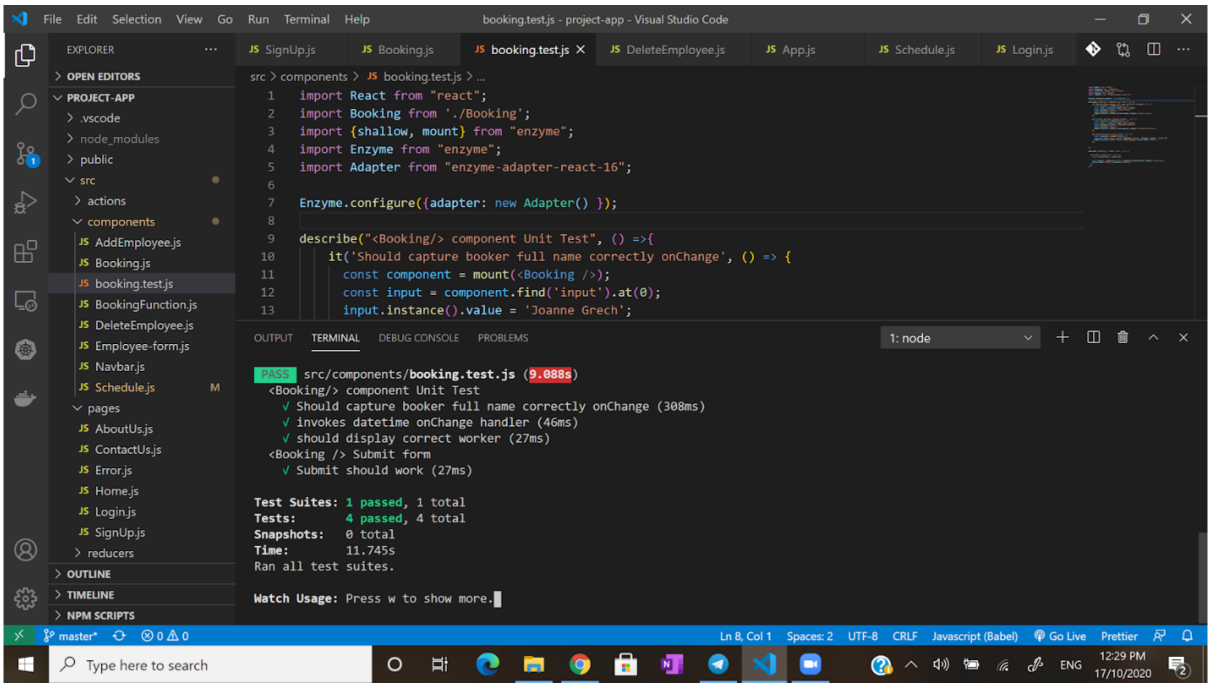
Documentation of acceptance test cases

All test cases passed - Backend and Frontend



```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14.893 s
[INFO] Finished at: 2020-10-18T14:06:29+11:00
[INFO] ------------------------------------------------------------------------
```

[Figure 1- backend test]



```
import React from "react";
import Booking from './Booking';
import {shallow, mount} from "enzyme";
import Enzyme from "enzyme";
import Adapter from "enzyme-adapter-react-16";

Enzyme.configure({adapter: new Adapter() });

describe("<Booking/> component Unit Test", () =>{
    it('Should capture booker full name correctly onChange', () => {
      const component = mount(<Booking />);
      const input = component.find('input').at(0);
      input.instance().value = 'Joanne Grech';
```

```
PASS  src/components/booking.test.js (9.088s)
  <Booking/> component Unit Test
    √ Should capture booker full name correctly onChange (308ms)
    √ invokes datetime onChange handler (46ms)
    √ should display correct worker (27ms)
  <Booking /> Submit form
    √ Submit should work (27ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        11.745s
Ran all test suites.

Watch Usage: Press w to show more.
```

[Figure 2- Frontend test]