Milestone 3

SangYeon Lee - s3742751

Mitchell Gust - s3782095

David Brown - s3785994

Vision statement

Our group has developed an exceptional product. We have built a single page React application that enables reusable components, and near immediate rendering. The application features two event driven REST API run by the Node platform that are perfect in handling the data intensity involved in our business and authentication services. We make use of NATS, an incredibly fast data streaming system to enable asynchronous communication between our microservices and our cloud database, MongoDB Atlas. Our selected server offers consistent performance, and security you can trust.

The project offers advantages for both the customer and business. We provide customers with a reliable way to reach their favourite local businesses. A search bar is provided on the central home screen to give customers an easy way to locate their requested business, and an efficient way to fast-track customers straight into the booking process.

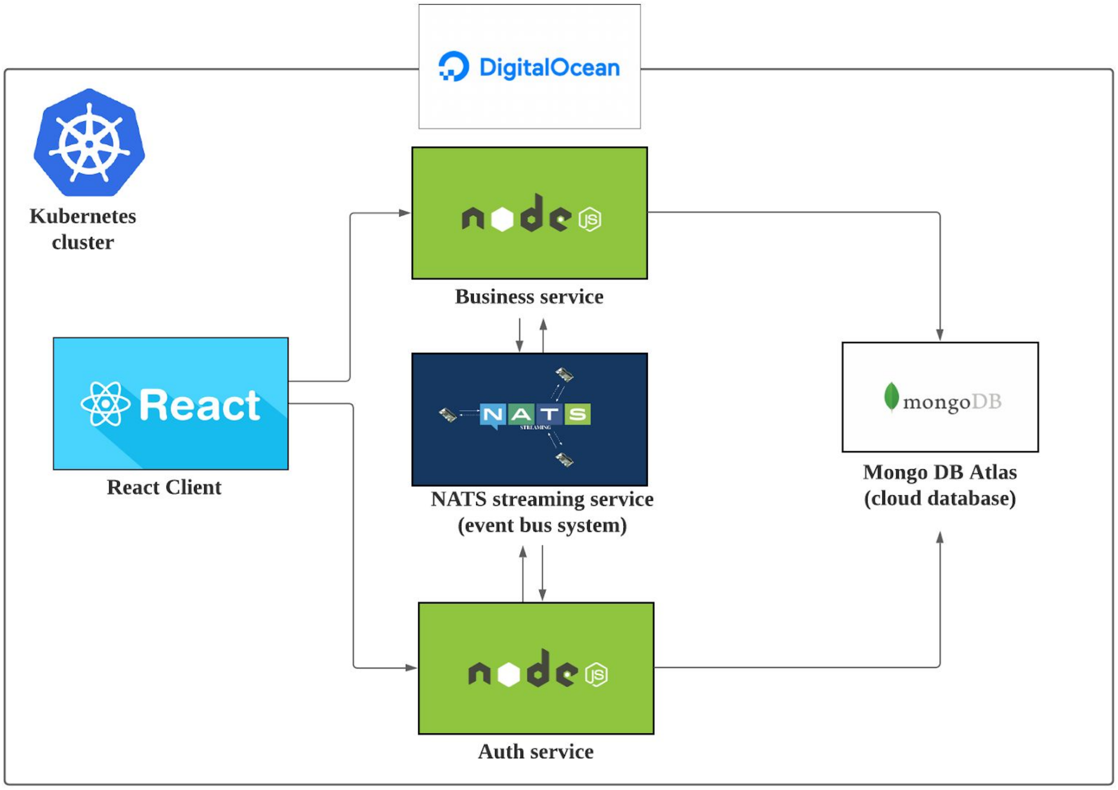
This simple implementation provides great value to the business by providing the customer an easy user-friendly way to find their desired company and to create bookings without the need to dive through multiple menus and screens.

Bookings are available for any of the listed business services and customers are given the ability to book these services with a desired worker. When a desired worker is chosen, the dates and time slots shown are appropriate to their selection and match the worker's availability.

This provides the business with an efficient and self-managed scheduler and eliminates the need to coordinate and manage their worker's tasks for the day. This feature can also highlight to the business who their most popular workers are.

We have done extensive work to deliver further value to the business by providing a way to manage their workers and schedule entirely through our application, to make it a one-stop-shop for any eager business to expand their current system. Businesses can create separate accounts for workers, allowing them to notify their manager of their availability and allowing businesses to add and edit working hours.

Design Architecture



Our team is using a **React** single page application for the client service and **Node** express REST API for the Authentication Service and Business Service.

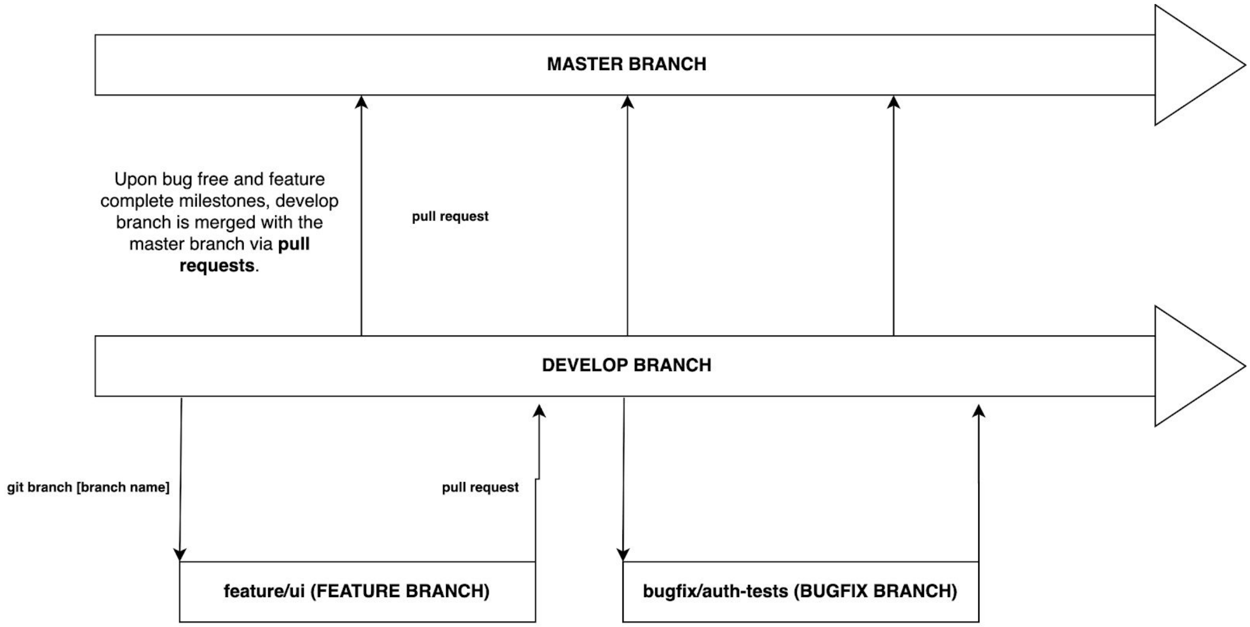
We are using **NATS** streaming server to enable asynchronous communication between microservices and cloud **mongoDB**.

Our application is running in the Kubernetes Cluster provided by **Digital Ocean**. We have chosen Digital Ocean over other cloud service providers as it is the easiest kubernete cluster to configure, and is the cheapest on the market. Digital Ocean kubernete cluster ranges from only $20-30 per month while AWS is over a hundred dollar a month

Current kubernete configuration runs only one replica per service.

Processes & tools (Skaffold & Gitflow)

Our team utilised a typical Gitflow workflow consisting of a single master branch containing stable releases for milestones, a single develop branch for feature complete (but not necessarily bug free) iterative development, and temporary feature branches used to facilitate the development of individual features such as React components or general functionality such as large numbers of unit tests.


MASTER BRANCH
Upon bug free and feature complete milestones, develop branch is merged with the master branch via **pull requests**.
pull request
DEVELOP BRANCH
git branch [branch name]
pull request
feature/ui (FEATURE BRANCH)
bugfix/auth-tests (BUGFIX BRANCH)

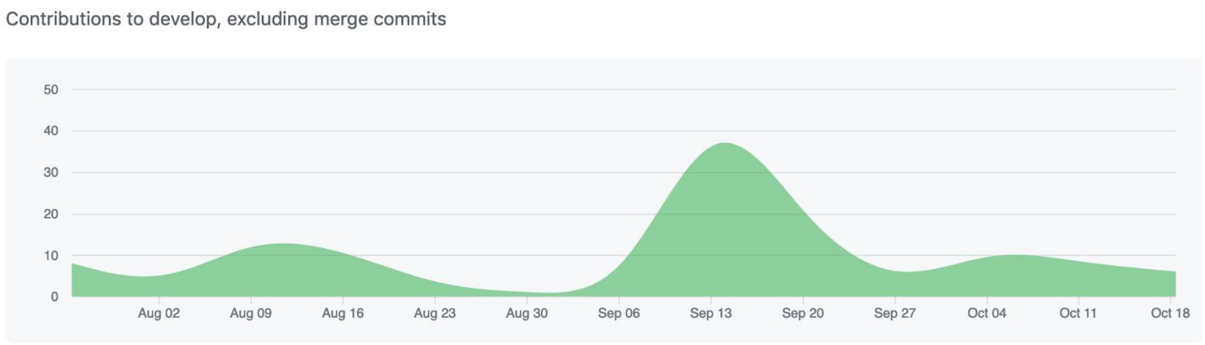
New branches are created for the implementation of functionality of the software, as well as writing tests and performing bug fixes. The branches are created using the command **git branch [branch name]**, entered into using **git checkout [branch name]**, and deleted using **git branch -d [branch name]**. **Pull requests** are used in order to merge with the develop branch.

New updated to the develop branch are periodically retreived from the develop branch by developers using **git pull** within the develop branch and merged into their develop branches using **git merge develop** while within the feature/test/bugfix branch.

Feature branches were either merged with the develop branch before being updated, or (later during development as strategies changed) merged via pull requests.

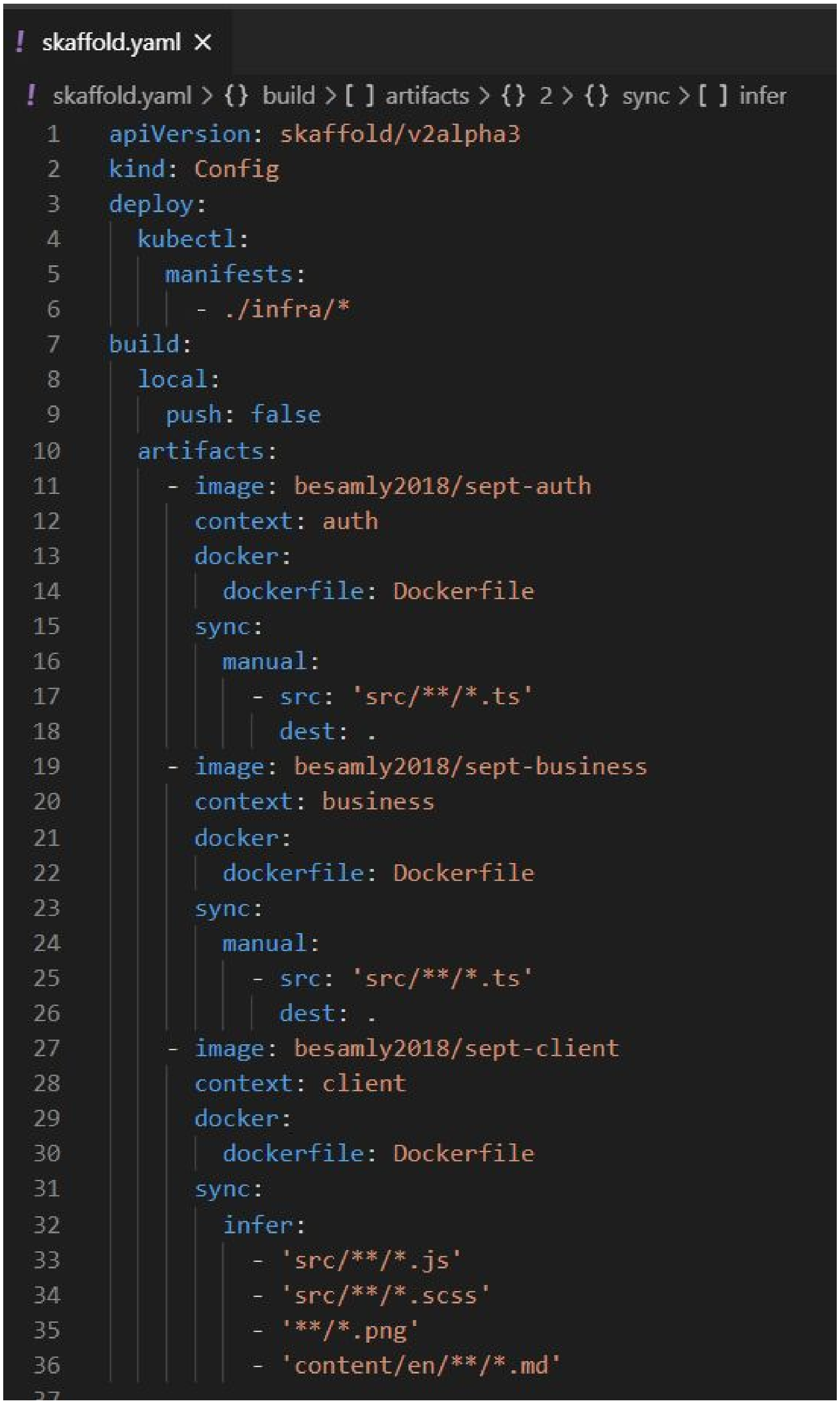
As a general rule, we labelled our branches depending on their contents - feature branches were preceded by the “feature/” suffix (e.g. “feature/ui”). The same occurred for tests branches (“test/”). This provided some context to the purpose of branches from quick observation and therefore prevented clutter.

Commit Frequency over Time



Our team used Skaffold container tool which facilitates continuous development for Kubernetes applications. It iterates on each microservice’s source code, and when there is change on the source code, it rollouts update to local Kubernetes clusters without the need of manually building an image and restarting kubernete cluster.

Skaffold configuration file

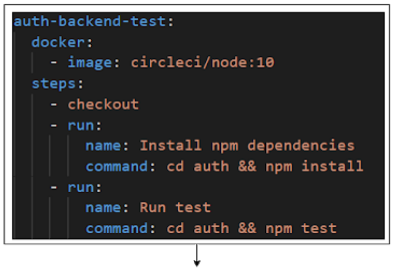


! skaffold.yaml ×
! skaffold.yaml > {} build > [ ] artifacts > {} 2 > {} sync > [ ] infer

```yaml
apiVersion: skaffold/v2alpha3
kind: Config
deploy:
  kubectl:
    manifests:
      - ./infra/*
build:
  local:
    push: false
  artifacts:
    - image: besamly2018/sept-auth
      context: auth
      docker:
        dockerfile: Dockerfile
      sync:
        manual:
          - src: 'src/**/*.ts'
            dest: .
    - image: besamly2018/sept-business
      context: business
      docker:
        dockerfile: Dockerfile
      sync:
        manual:
          - src: 'src/**/*.ts'
            dest: .
    - image: besamly2018/sept-client
      context: client
      docker:
        dockerfile: Dockerfile
      sync:
        infer:
          - 'src/**/*.js'
          - 'src/**/*.scss'
          - '**/*.png'
          - 'content/en/**/*.md'
```

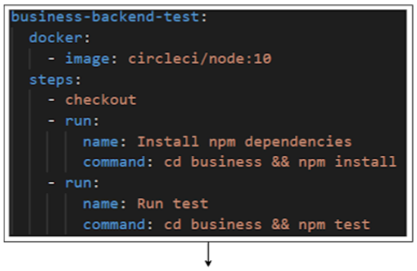
Deployment pipeline setup

1. Auth service testing



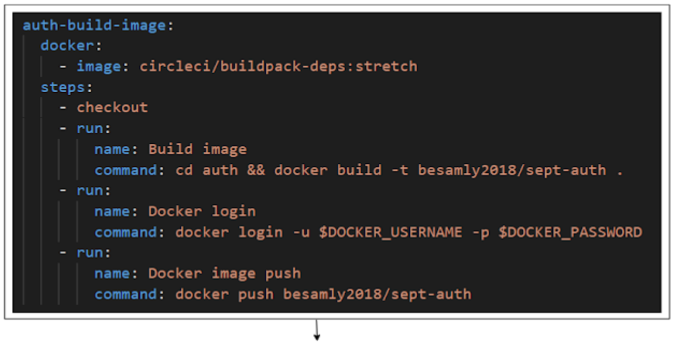
```
auth-backend-test:
  docker:
    - image: circleci/node:10
  steps:
    - checkout
    - run:
        name: Install npm dependencies
        command: cd auth && npm install
    - run:
        name: Run test
        command: cd auth && npm test
```

2. Business service testing



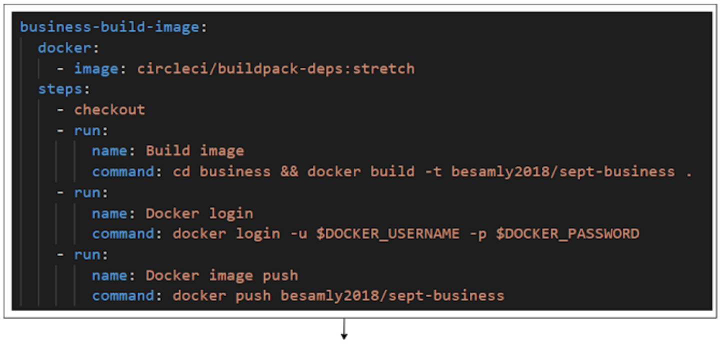
```
business-backend-test:
  docker:
    - image: circleci/node:10
  steps:
    - checkout
    - run:
        name: Install npm dependencies
        command: cd business && npm install
    - run:
        name: Run test
        command: cd business && npm test
```

3. Build auth service image



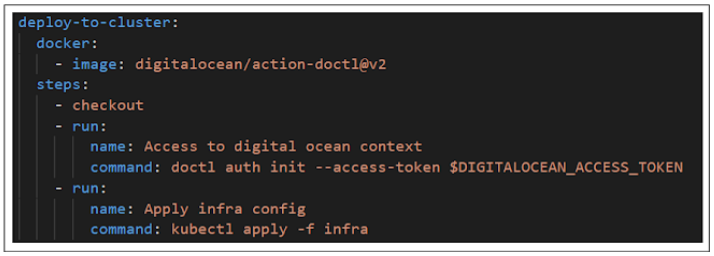
```
auth-build-image:
  docker:
    - image: circleci/buildpack-deps:stretch
  steps:
    - checkout
    - run:
        name: Build image
        command: cd auth && docker build -t besamly2018/sept-auth .
    - run:
        name: Docker login
        command: docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
    - run:
        name: Docker image push
        command: docker push besamly2018/sept-auth
```

4. Build business service image



```
business-build-image:
  docker:
    - image: circleci/buildpack-deps:stretch
  steps:
    - checkout
    - run:
        name: Build image
        command: cd business && docker build -t besamly2018/sept-business .
    - run:
        name: Docker login
        command: docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
    - run:
        name: Docker image push
        command: docker push besamly2018/sept-business
```

5. Deploy to Digital ocean kubernete cluster



```
deploy-to-cluster:
  docker:
    - image: digitalocean/action-doctl@v2
  steps:
    - checkout
    - run:
        name: Access to digital ocean context
        command: doctl auth init --access-token $DIGITALOCEAN_ACCESS_TOKEN
    - run:
        name: Apply infra config
        command: kubectl apply -f infra
```

Scrum process

Our scrum process consisted of meeting at least two times a week. Our typical week would consist of a scrum meeting on Wednesday 11:30am, and another on Sunday 11:30am. We were very effective in reflecting on what we had learnt and succeeded in redirecting our approach with our previous experiences and issues in mind. We believe the tools that Scrum implement are very useful in maintaining constant interaction with the team and supplying a framework to support motivation and good work. The Scrum Master for this sprint was Mitchell Gust, but for the majority of the project the Scrum Master role was held by previous team member, Anh Nguyen. David Brown was the team’s Product Owner.

Communication Logs

Team: 3.THURS-10-30.3

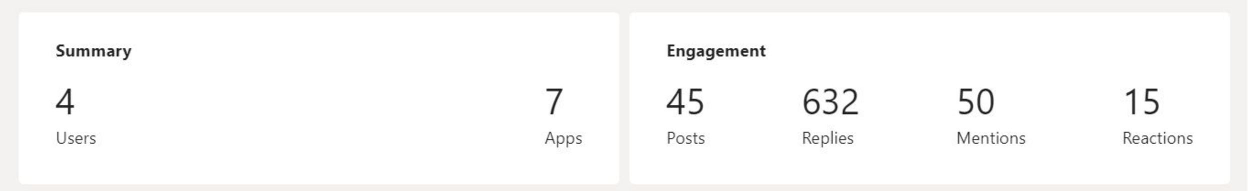
Members:

- SangYeon Lee - s3742751
  Mitchell Gust - s3782095
- David Brown - s3785994

Communication Summary:

- Meetings (Every Wednesday and Sunday 11:30am) on Microsoft Teams
- General Questions on Microsoft Teams General Chat Channel
- Tutorial (Thursday 10:30am) with Sachin
- Peer Learning Sessions (Planned when Required)

Microsoft Teams Channel Analytics:



Summary

4 Users

7 Apps

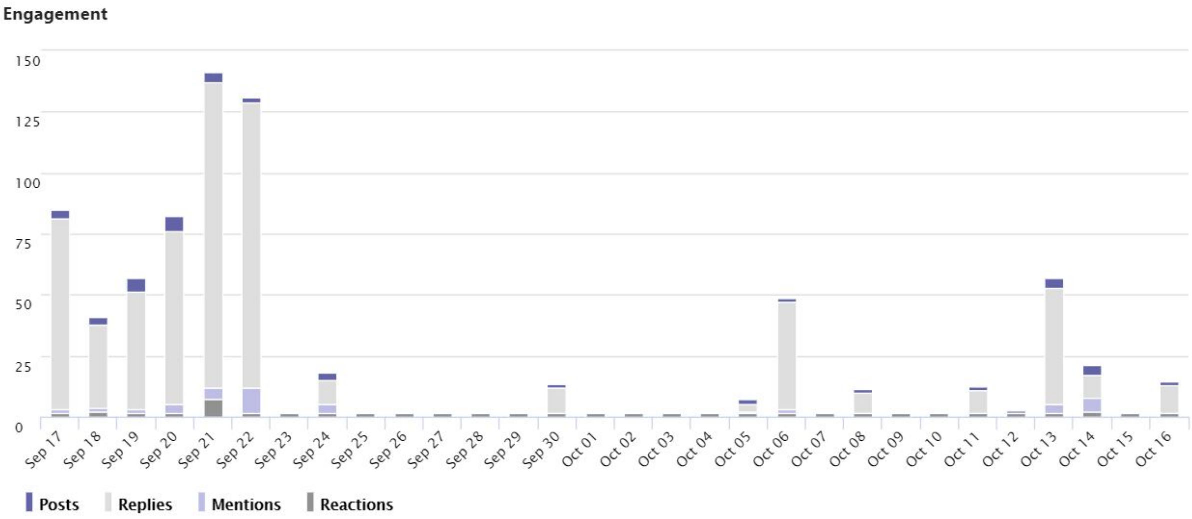
Engagement

45 Posts

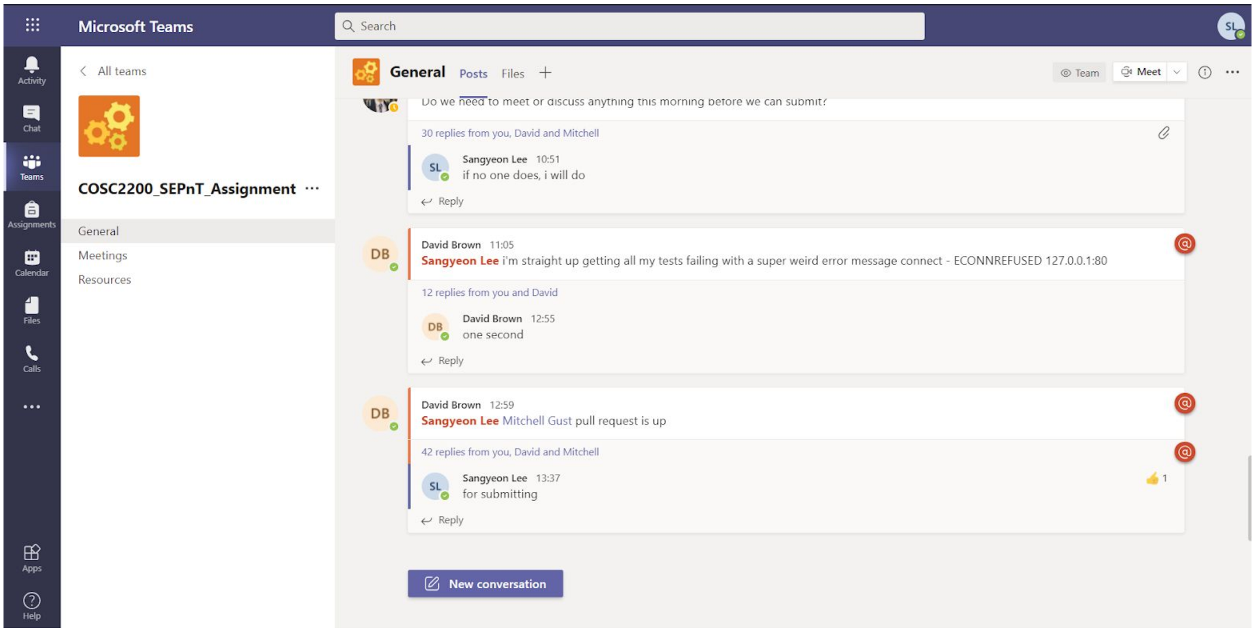
632 Replies

50 Mentions

15 Reactions



Microsoft Teams Screenshot:



Do we need to meet or discuss anything this morning before we can submit?

30 replies from you, David and Mitchell

Sangyeon Lee 10:51
if no one does, i will do

Reply

David Brown 11:05
Sangyeon Lee i'm straight up getting all my tests failing with a super weird error message connect - ECONNREFUSED 127.0.0.1:80

12 replies from you and David

David Brown 12:55
one second

Reply

David Brown 12:59
Sangyeon Lee Mitchell Gust pull request is up

42 replies from you, David and Mitchell

Sangyeon Lee 13:37
for submitting

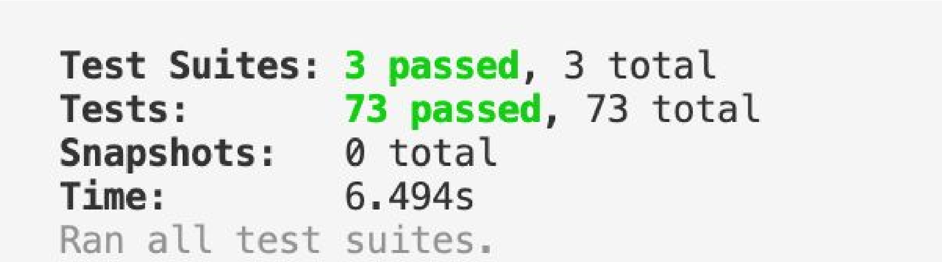
👍 1

Reply

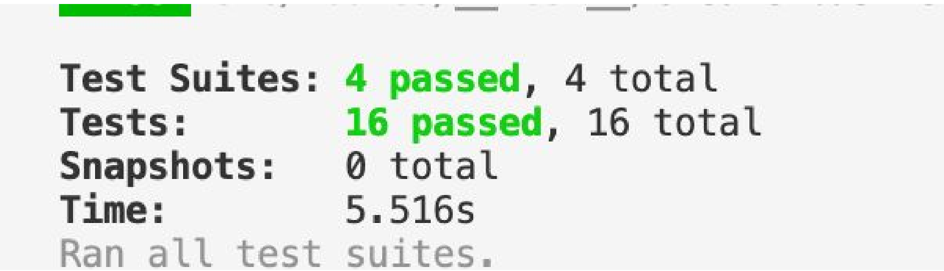
New conversation

Test Cases (Unit & Acceptance)

Evidence of Passing Unit Tests (auth and business microservices)



```
Test Suites: 3 passed, 3 total
Tests:       73 passed, 73 total
Snapshots:   0 total
Time:        6.494s
Ran all test suites.
```



```
Test Suites: 4 passed, 4 total
Tests:       16 passed, 16 total
Snapshots:   0 total
Time:        5.516s
Ran all test suites.
```

Status of Acceptance Tests

Last evaluated: 18th October

No.	Purpose	Pass (Y/N)
1	Test getting booking history with existing prior bookings for customer in system	Y
2	Test getting booking history with no prior bookings for customer in system	Y
3	Test ‘view bookings’ with an existing worker who has upcoming appointments	Y
4	Test ‘view bookings’ with an existing worker who has no upcoming appointments	Y
5	Test ‘Edit Worker Profile’ with an existing worker	Y
6	Test ‘Edit Worker Profile’ with an existing worker	Y
7	Test ‘View Worker Profile’ with an existing worker	Y
8	Test blank input for employee creation	Y
9	Test existing username for employee creation	Y

10	Test all valid inputs for employee creation	Y
11	Test invalid username for employee edit	Y
12	Test all valid inputs for employee edit	Y
13	Test invalid hours input for add shift	Y
14	Test valid input for add shift	Y
15	Test invalid hours input for edit shift	Y
16	Test valid hours input for edit shift	Y
17	Test viewing past bookings with no past bookings	Y
18	Test viewing past bookings with existing past bookings	Y
19	Test viewing upcoming bookings with no upcoming bookings	Y
20	Test viewing upcoming bookings with no upcoming bookings	Y
21	Test viewing worker availability with no available workers	Y
22	Test viewing worker availability with available workers	Y
23	Test viewing service availability with no available services	Y
23	Test viewing service availability with available services	Y
25	Test attempting booking with no available times	Y
26	Test attempting booking with unavailable services	Y
27	Test attempting booking with unavailable services	Y
28	Test the Registered User can access the home page	Y
29	Test the Unregistered User can access the home page	Y
30	Test an Admin with Admin permissions can access the Admin Dashboard	Y
31	Test the Admin is redirected to the Admin Dashboard after login	Y
32	Test a Customer without Admin permissions can not access the Admin Dashboard	Y
33	Test a Worker without Admin permissions can not access the Admin Dashboard	Y

34	Test a Worker with Worker permissions can access the Worker Dashboard	Y
35	Test the Worker is redirected to the Worker Dashboard after login	Y
36	Test a Customer without Worker permissions can not access the Worker Dashboard	Y
37	Test an Admin without Worker permissions can not access the Worker Dashboard	Y
38	Test the Customer can access the Customer Dashboard	Y
39	Test the Customer is redirected to the Customer Dashboard after login	Y
40	Test a Worker without Customer permissions can not access the Customer Dashboard	Y
41	Test an Admin without Customer permissions can not access the Customer Dashboard	Y
42	Test User with invalid credentials can not log in	Y
43	Test User can not Register with sign up fields left blank	Y
44	Test that a user can not register with an invalid email	Y
45	Test that a user can not register with an invalid phone number	Y
46	Test that a user can not register with a duplicate username	Y
47	Test that a user can register with valid details	Y
48	User can not change user name to the name that has been already taken.	Y
49	Users can not edit passwords without meeting password policy.	Y
50	Customer can not cancel booking less than 24 before the event	Y

Links to Resources

Click Label to View Resource

Deployed Application

Private Trello

Google Drive

Github

MS Teams

Postman Published Documentation