Software Engineering Process and Tools

Semester 2 2020 -Milestone3

Group: Lab-Wed-6.30-Group5

Members:

Neil Kennedy

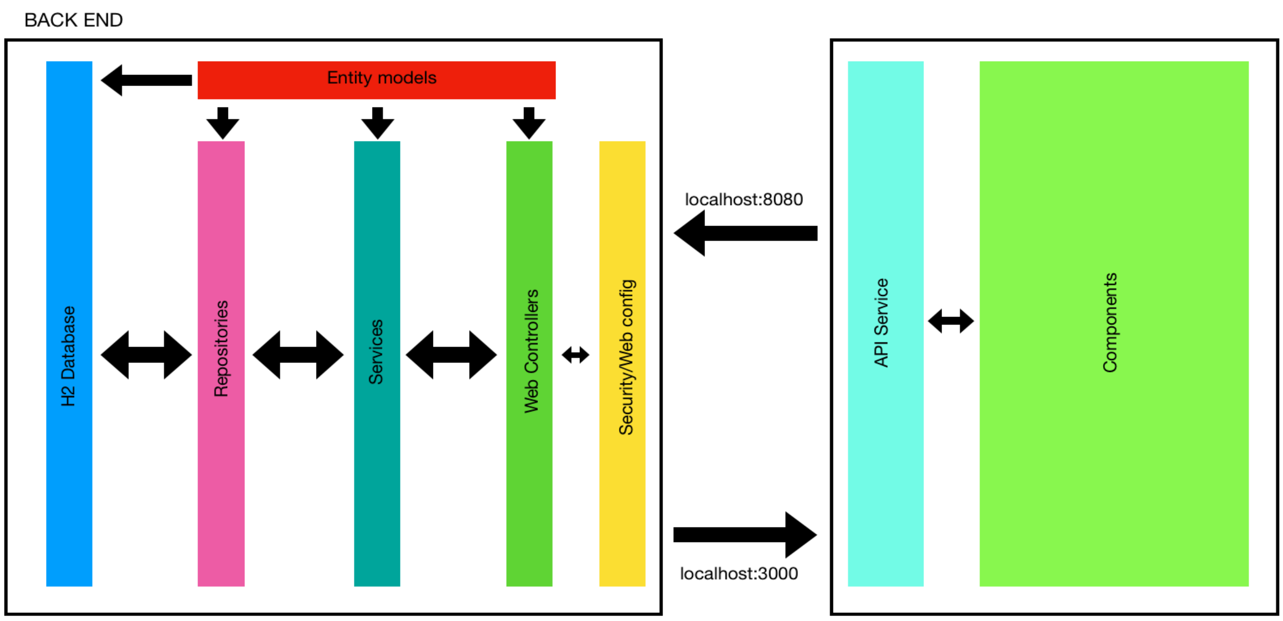
Moditha Sulakshana

Michael Anning

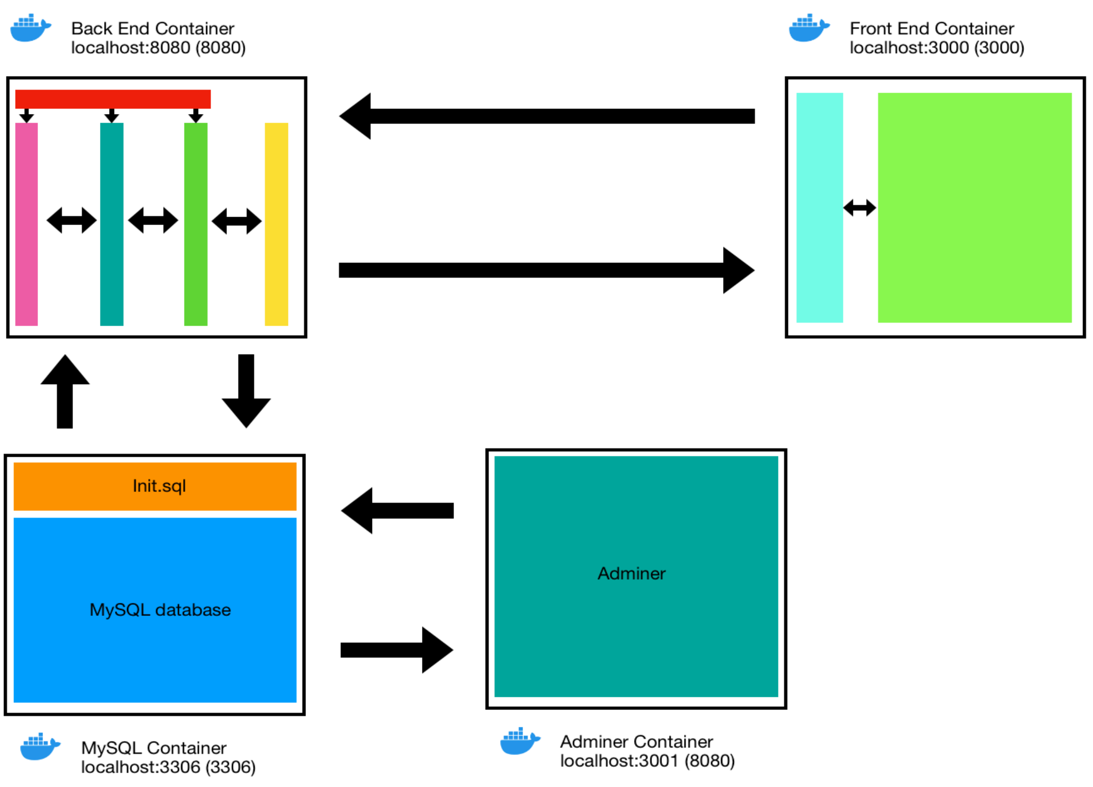
Boyan Yao

Architecture

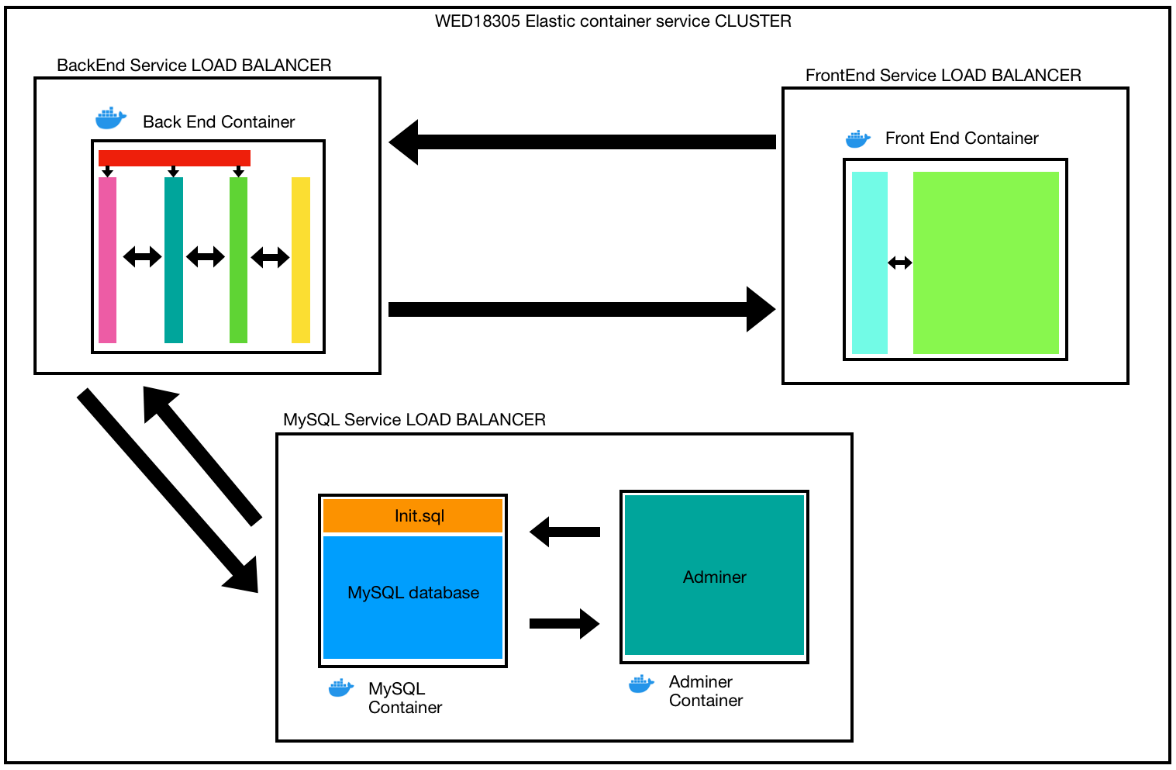
Local Development Architecture (H2 database)



Local Deployment Architecture Docker (Mysql) ***run docker-compose up -d***



AWS Deployment Architecture Docker (MySQL)


WED18305 Elastic container service CLUSTER
BackEnd Service LOAD BALANCER
Back End Container
FrontEnd Service LOAD BALANCER
Front End Container
MySQL Service LOAD BALANCER
Init.sql
MySQL database
Adminer
MySQL Container
Adminer Container

Development Reports

Neil Kennedy Development Report

As far as refactoring goes some of the biggest changes that happened was the need to edit entity models to handle many too many connections in JoinTables, most of these where added to the Entity_User allowing connections to, schedules, services and bookings.
Services was changed in the second sprint to handle time lengths, the original design had users choosing their bookings start and end time but after a review this was changed to handle service specific time lengths so that users only choose their start time.
Entity names needed to be un capitalised after the creation of a MySQL database as that was causing issues also MySQL was originally hosted locally during development using software called Ampps but was converted into a docker container as docker files where implemented for ease of deployment.

Reflecting on the current state of the software I have a list of "smells" that would I would have liked to address but ultimately ran out of time. Firstly is our request objects, these objects are used for handling input from the front end and parsing info into the correct type, while the idea is sound as the endpoints grew we started to end up with a request for every endpoint, ideally I would have liked a more generalised request objects that could be used to minimise the number of them being used and promote more code reusability.

Secondly is the command line runner inside DemoApplication.java, this was setup at the beginning of production to insert testing data easily onto the H2 database, unfortunately this becomes more of a problem when moving to MySQL for deployment and should have been a schema, although this implementation can result in user errors inserting data since it doesn't use the endpoints or service for data insertion.

Schedule entities should have had a @JoinTable reference to bookings to allow easy access to bookings on a specific schedule and vis versa, this was an unfortunate oversight in the early design and results currently in unnecessary time and cpu usage spent finding bookings for a schedule and as such is rather inefficient.

The web controllers which handle all the endpoints for specific endpoints are somewhat large, while they collapse down to a fairly small and easily readable file they could be split into sub files that handle endpoint specific feature such as create, delete, edit etc, since we can have multiple types of create, delete or edit these could be separated for easier reading.

Speaking of large files the testing classes are very large, while they handle a good amount of testing they should be split into smaller sub categories like the web controller suggestion above to improve readability.

The "Entity_UserType" has a UserTypeID enum set up(not by me) but it makes a poor assumption of Entity_UserType IDs as constants when they are currently handled by the database, this would need to be changed to not make assumptions that could easily be accidentally changed and break a lot of stuff.

Cookies and cors handling is somewhat tedious at the moment, "Access-Control-Allow-Origin" header is hardcoded in the authorisation success handle and logout success handler, also the cors access address is hardcoded in the WebMvcConfig file. Ideally this should be controlled by a spring environment variable that would allow quick adjustment at deployment time via AWS or docker.

Currently I have a MySQL docker container which is used for hosting our database data but at the moment this setup is vulnerable to data loss when stopping the container and should not be used for full release it was more of a development extension.

The AWS deployment is currently rather tedious, part of this is AWS itself and the other part is the limited features available to student accounts, you cannot add users to your account for others to access let alone implementing certain features to automate deployment.
AWS deployment could do with more work as my experience with it is limited and as such I'm not sure the final deployment is "correct" or the most efficient.

The Docker files are probably not set up optimally as a build results in hanging images being left over which can eat up hard drive space quickly.

Michael Anning Development Report

Inscribed below are the list of smells I have personally noticed:

Entity_UserType Enum: Each user has a UserType that determines specifically what they're allowed access to, as well as what backend functions are allowed to affect them. Initial data was provided that created UserTypes in a certain order. To make UserType checking reusable, the UserTypeID enum was introduced, adhering to the preset order.
However, this was poor design, as it didn't accommodate for the chance that UserTypes could be deleted, and remade in any order in a production environment. In hindsight, a much better solution would to check if a User's UserType contains a specific name (like 'Admin') and make conditions based on that. However, this addresses the next thing to change:

Security Config's Privilege Enforcing: Security Config.java handles all access privileges to backend & frontend functionality. It checks the UserType ID for a user currently being evaluated to enforce privileges. However, as discussed above, we can't always predict what order the UserTypes will be created in, so their IDs may not equal what we're currently evaluating.
To reiterate my suggestion, Security Config should check for the name of a User's UserType instead of its ID. It's a much safer alternative as we can set the value of a UserType record's 'name' variable, but not its ID. That's determined by the order of which a record's created.

Booking Class's JSON Input: We had an initial design for our Booking class where it contained a single 'user_ids' array. The idea was that if a booking could have multiple employees running it, then we should reference each employee by their ID. However, we later decided on only assigning bookings to one employee.

The actual booking design was never updated to accommodate this change. For refactoring, I would wish to change the 'users' array into an 'employee' Entity_User variable. This would make it clearer as to what the variable's intended for. Additionally, we could add another Entity_User variable called 'customer', which would indicate which customer's ordered the booking.
To make this change, we'd also need to update how booking records are created on the front end. Currently, JSON input takes an array called 'user_ids' and links said id inside to the relevant employee. We could instead change that to 'employee_id' which takes in a number. Additionally, we'd also need to update how bookings are assigned to customers, so that they're assigned to the booking's 'customer' variable.
Finally, we also must change 'create_booking' tests to instead push in a long value instead of an array of longs.

Entity Classes ID's Being of Type Long: This is a design choice that unnecessary memory risk on the program. Being an assignment for school, we can only assume that the amount of bookings created won't get anywhere near 263-1. As such, it would be more suitable to update each ID to be of type 'int' instead. It has a more reasonable maximum value of 231-1, and would take up far less memory in our program to store. We'd also need to update all get ID types to be of type int instead.

Entity Classes Protected Variables: None of the Entity classes (our model classes) are superclasses, yet they each possess protected variables. There's no reason to do when setting them to 'private' would suffice.

Entity Booking's Getters: In the Entity Booking class, there exists two getter methods called 'getCustomers' & 'getEmployees'. They check users the 'users' array for their UserTypeID, which we discussed would be invalid with our above changes. To refactor this, we would rename the methods to 'getCustomer' & 'getEmployee', and each would retrieve the proposed 'customer' & 'employee' Entity_User variables respectively.

Moditha Sulakshana Development Report

My main forte was working on the front-end development of this project. While coding the front-end I ran into several "smells" that made the functions challenging or halted for its respective milestones.

DateTimeFormat: The DateTimeFormat we choose for this project was doing everything in the UTC Date format, which was horrible when things needed to be passed to the backend and retrieve data. There was a lot of necessary functions created on each component that required a date, it wasn't reusable as some endpoints required only a date and some end points required date+time. Another problem was since I used different react calendar components which provided additional features which were needed for some of the functionalities, they handled the UTC time format differently too. Moving to something like the "POSIX" time format would have reduced the cluster of code by at least 45% and improved the efficiency of each component and also running into fewer bugs of handling date and times.

Lack of Variety in Schedule Backend API: Due to lack of time and priority needed elsewhere, the backend schedule API lacked the variety for different instances of API calls of the system. The getSchedule Endpoint returned me the employees schedule up to the next month, which is what the systems functionality is supposed to do however when I wanted to get the schedule of an employee of a particular day, it would require me to format the date selected into UTC and then traverse through the returned list of the employee schedules and find that particular day to get me the that dates schedule which could have been avoided by implementing another end-point that takes a date.

Refactoring Hooks: Previously having worked on a few React projects, I started off writing the front-end code using React Hooks. I never wrote API calls for React and only worked on the UI side of things, hence I didn't know that implementing API calls using React Hooks would be much harder than I expected. Since the tutorials in class were done using Axios, I was advised to refactor my code into class components and use Axios for my API calls which made me get rid of a lot of templates and functions I didn't know how to refactor to classes.

Changing Date Selection Component: I started creating the booking form using normal form type components, so I used two selection boxes that had type=time so that the user can select the start time and end time. However, at around Sprint 2 I realized that whenever a booking is created, I should be blocking out these time slots so that multiple bookings at the same time cannot be created. While this seemed fine in my head that it was as simple as removing these times from each time selection components, there was no way for me to link them together to create time slots and block them off together. Hence, I had to get rid of both these default time components and generate my own time component. For that I used a drop-down box, that gets populated with available time slots from the employee's schedule (start time to end time) and this would allow me to block out the time slots which are booked another customer. Hence a lot of refactoring was done to handle times and pass into the create booking Api.

Boyan Yao development report

My role for the team is a front-end developer. The following list is the smells I have noticed for this project:

React component usability:

The format of date and time we use is UTC time. The frontend submits a local date time and a process convert it to UTC time format then stored in the backend. One big challenge I have faced during development when I was creating react calendar initially to display employee upcoming bookings was that when I retrieve the data from backend, it was difficult to convert it into a local time format and insert the data to a calendar fitting every parameter. The backend problem was fixed by backend programmers, the fitting parameter was not able to be solved as the Syncfusion react calendar has different parameters for a calendar than our project can provide. Eventually, I went for an alternative and replace the syncfusion-react-calendar with react-card. As we have used cards to display all bookings, it is consistent and familiar to use. It can also be adjusted according to our Booking array and display different view for different user purposes.

Unit-testing:

Unit testing for both frontend and backend is critical. Though backend programmers have completed unit tests well, the testing for frontend was quite challenging as there were few instructions or examples. Therefore, frontend programmers had to face massive research and experiments as we are new to enzyme.

Backend API fetching data:

Initially we decided to have different endpoints for fetching booking data for different users, the process was complicated and caused conflicts sometimes. Then a UserTypeID solution was introduced and we could adjust backend to fetch bookings data for different users based on their UserTypeID with just one endpoint in API service.

GitFlow overview

GitHub: https://github.com/RMIT-SEPT/majorproject-wed-18-30-5

Originally each user just developed in their own branch(NeilK, Modi, Michael, Boyan) this was mainly because of issues relating to everybody's experience using Git, keeping us seperate was a way of limiting the damage that individuals where doing to the code base but as time went on and peoples skills improved and we shifted to a more appropriate git flow style of having branches for specific elements of development and then performing merge requests to pull back into development.

Main -> Development -> (feature branches)

This is the layout of our branch design, features are created then pull requested into development, the merge requests are approved by Neil for back end features and Modi for front end features, Neil handles pull request into Main from development mainly for milestone delivery.

Contributions can be viewed here: https://github.com/RMIT-SEPT/majorproject-wed-18-30-5/graphs/contributors

Scrum process

Scrum master duties were shared (alternating between Neil and Modi)

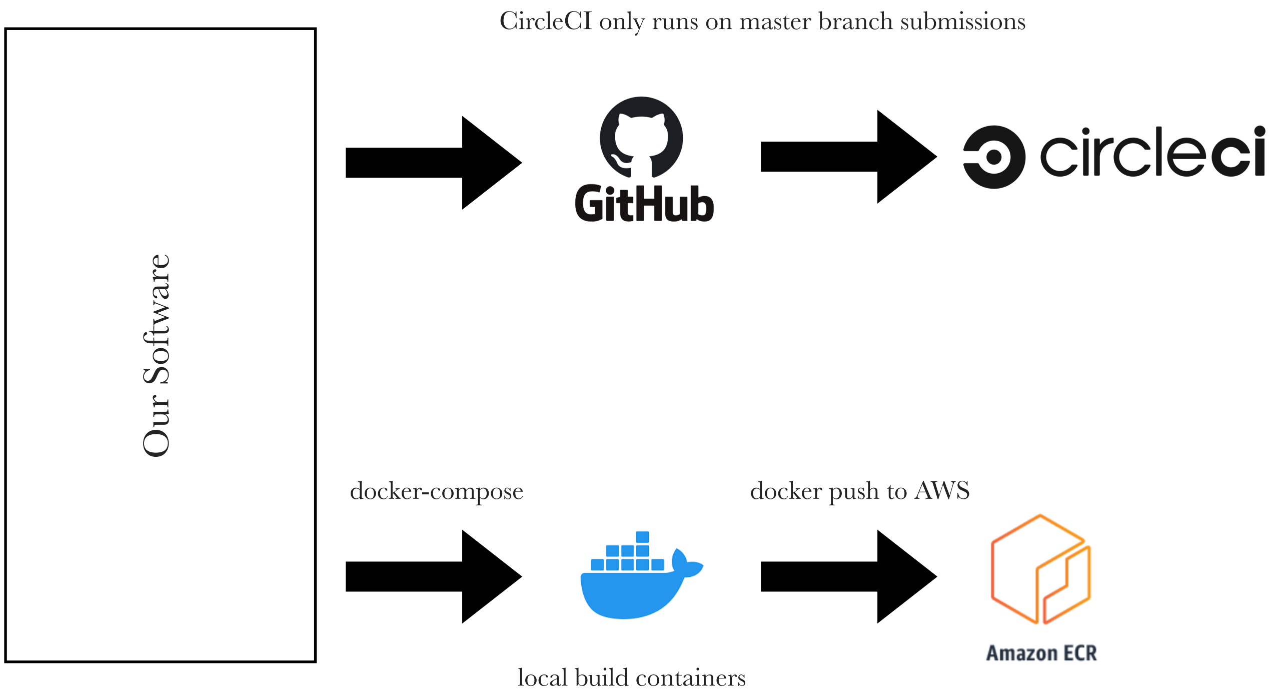
Meetings where held twice a week, Wednesday 18:30 before and after the tutorial and then on Sunday at 18:30, meetings where held on Microsoft Teams our channel is LAB-WED-6.30-Ujj-Group5.
Sprint planning was done on Sundays as well as a standup, Wednesdays consisted of either a standup or Sprint review, reviews and planning where only done at the end of a sprint which wasn't each week so stand ups where done instead.

The backlog and all tasks where assigned using click up and can be seen here: https://app.clickup.com/6915699/home/landing

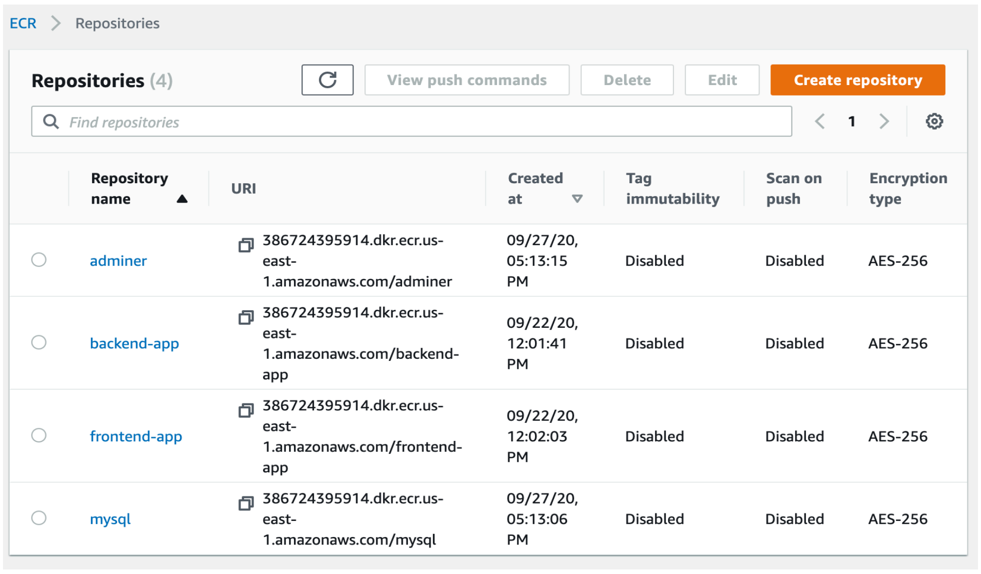
Deployment pipeline

Our software implements both CircleCI and Docker files, while we have circleCi implemented in building and testing both the front and back end it was pretty useless to our development since there was barely any compute minutes available when we went to use it and as such after its implementation it really was not used since it was never operational.

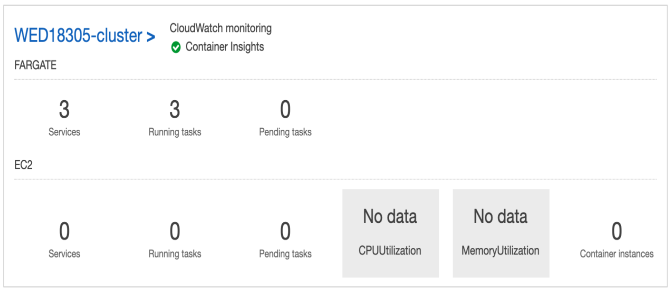
Docker was used to divide the elements of this software into containers for deployment on AWS but could also be used locally using the docker-compose file. Docker builds can be done locally and tested locally before using docker push to send the tagged containers to AWS to be stored on their amazon elastic container registry from there they are deployed using the amazon elastic container service using one Fargate cluster and three Fargate services.

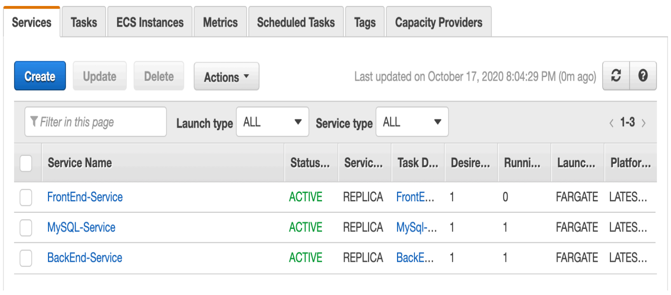


AWS deployment will not be active for reviewing as it will have run out of credits but the information below shows its setup and deployment, there is also an issue with http only access being rather tedious and requiring tweaks to google chrome to get it to work and as such isn't production ready

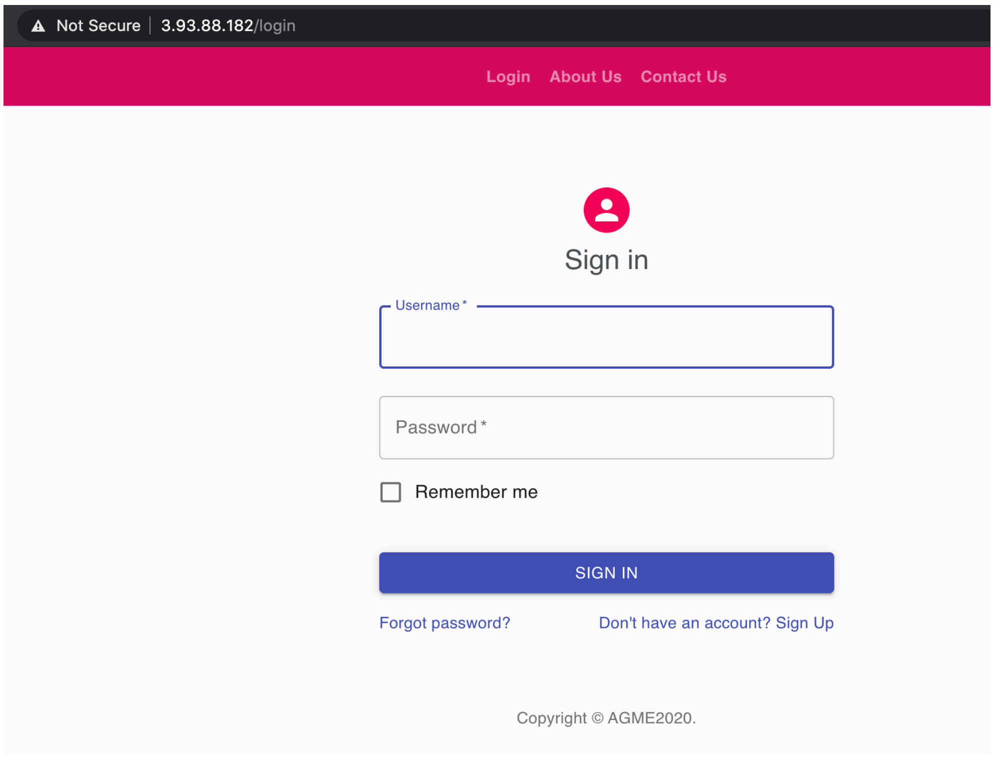


Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type
adminer	386724395914.dkr.ecr.us-east-1.amazonaws.com/adminer	09/27/20, 05:13:15 PM	Disabled	Disabled	AES-256
backend-app	386724395914.dkr.ecr.us-east-1.amazonaws.com/backend-app	09/22/20, 12:01:41 PM	Disabled	Disabled	AES-256
frontend-app	386724395914.dkr.ecr.us-east-1.amazonaws.com/frontend-app	09/22/20, 12:02:03 PM	Disabled	Disabled	AES-256
mysql	386724395914.dkr.ecr.us-east-1.amazonaws.com/mysql	09/27/20, 05:13:06 PM	Disabled	Disabled	AES-256





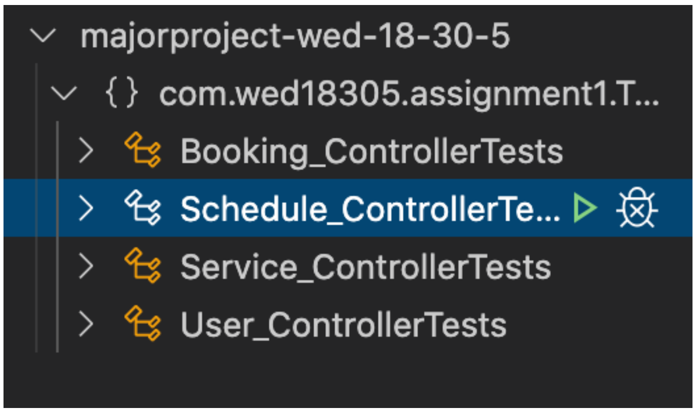
Service Name	Status...	Servic...	Task D...	Desire...	Runni...	Launc...	Platfor...
FrontEnd-Service	ACTIVE	REPLICA	FrontE...	1	0	FARGATE	LATES...
MySQL-Service	ACTIVE	REPLICA	MySql-...	1	1	FARGATE	LATES...
BackEnd-Service	ACTIVE	REPLICA	BackE...	1	1	FARGATE	LATES...



Acceptance tests

Acceptance tests can be seen in the GitHub repository within our UserStories folder
https://github.com/RMIT-SEPT/majorproject-wed-18-30-5/tree/master/docs/User%20Stories
There are some acceptance tests on the user stories pdfs and there will also be acceptance test specific pdf files in there as well.

JUnit tests for the back end within our visual studio code development

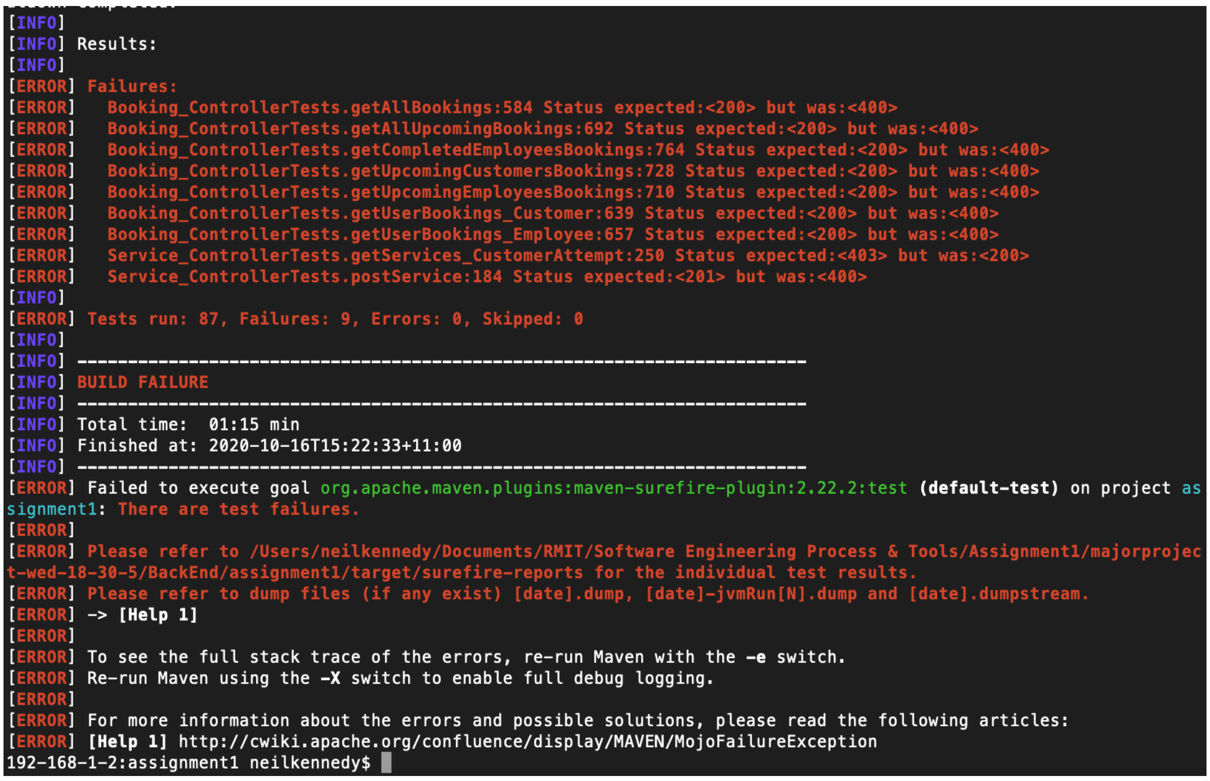


```
majorproject-wed-18-30-5
  {} com.wed18305.assignment1.T...
    Booking_ControllerTests
    Schedule_ControllerTe...
    Service_ControllerTests
    User_ControllerTests
```

These tests can be run within Visual Studio Code or can be done via Maven using

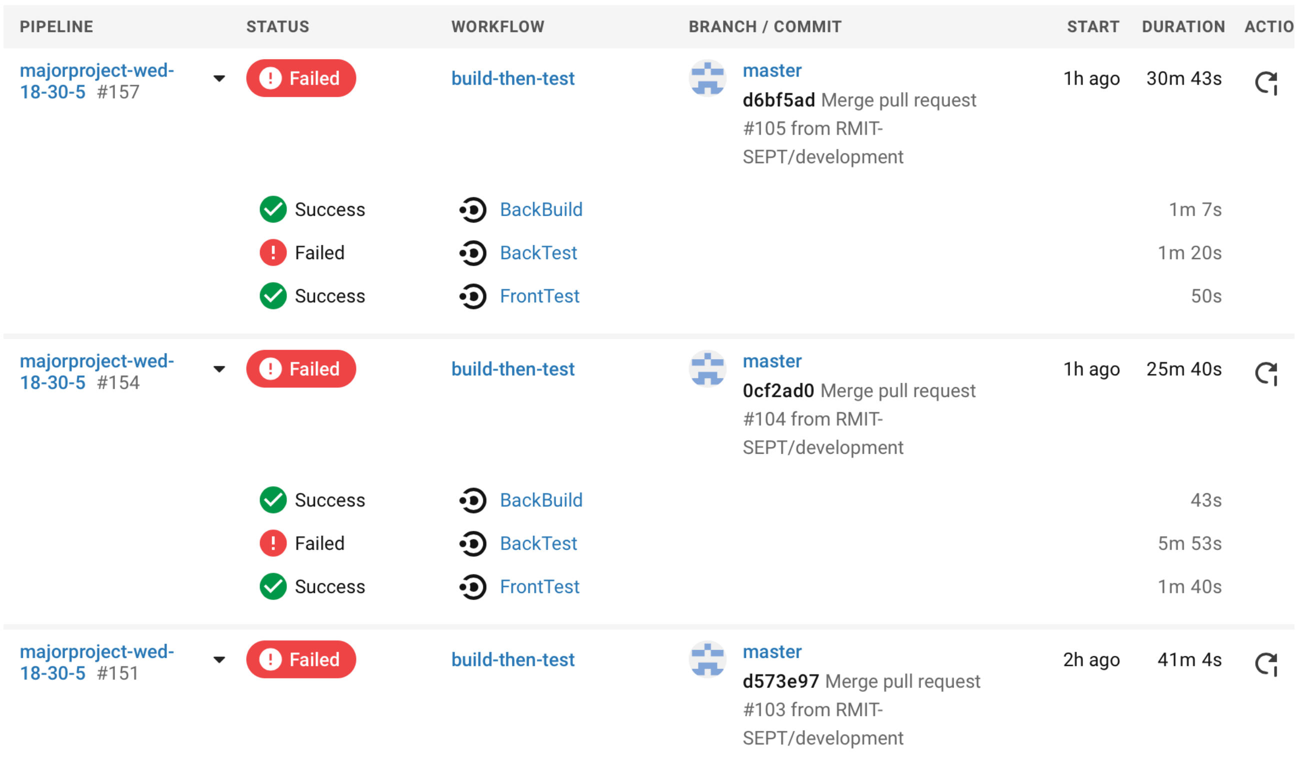
the command **./mvnw test**

Currently some of Michaels tests are not working.



```
[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   Booking_ControllerTests.getAllBookings:584 Status expected:<200> but was:<400>
[ERROR]   Booking_ControllerTests.getAllUpcomingBookings:692 Status expected:<200> but was:<400>
[ERROR]   Booking_ControllerTests.getCompletedEmployeesBookings:764 Status expected:<200> but was:<400>
[ERROR]   Booking_ControllerTests.getUpcomingCustomersBookings:728 Status expected:<200> but was:<400>
[ERROR]   Booking_ControllerTests.getUpcomingEmployeesBookings:710 Status expected:<200> but was:<400>
[ERROR]   Booking_ControllerTests.getUserBookings_Customer:639 Status expected:<200> but was:<400>
[ERROR]   Booking_ControllerTests.getUserBookings_Employee:657 Status expected:<200> but was:<400>
[ERROR]   Service_ControllerTests.getServices_CustomerAttempt:250 Status expected:<403> but was:<200>
[ERROR]   Service_ControllerTests.postService:184 Status expected:<201> but was:<400>
[INFO]
[ERROR] Tests run: 87, Failures: 9, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:15 min
[INFO] Finished at: 2020-10-16T15:22:33+11:00
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.2:test (default-test) on project as
signment1: There are test failures.
[ERROR]
[ERROR] Please refer to /Users/neilkennedy/Documents/RMIT/Software Engineering Process & Tools/Assignment1/majorprojec
t-wed-18-30-5/BackEnd/assignment1/target/surefire-reports for the individual test results.
[ERROR] Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
192-168-1-2:assignment1 neilkennedy$
```

Circle CI tests



PIPELINE	STATUS	WORKFLOW	BRANCH / COMMIT	START	DURATION	ACTIO
majorproject-wed-18-30-5 #157	Failed	build-then-test	master **d6bf5ad** Merge pull request #105 from RMIT-SEPT/development	1h ago	30m 43s	
	✓ Success	BackBuild			1m 7s	
	! Failed	BackTest			1m 20s	
	✓ Success	FrontTest			50s	
majorproject-wed-18-30-5 #154	Failed	build-then-test	master **0cf2ad0** Merge pull request #104 from RMIT-SEPT/development	1h ago	25m 40s	
	✓ Success	BackBuild			43s	
	! Failed	BackTest			5m 53s	
	✓ Success	FrontTest			1m 40s	
majorproject-wed-18-30-5 #151	Failed	build-then-test	master **d573e97** Merge pull request #103 from RMIT-SEPT/development	2h ago	41m 4s	