Project Report TUES-1830-5

Table of Contents

Vision statement

Our vision is to help the customer to have a good way to set their bookings and the events, which can help them save time when they actually want to book for anything like the gym, restaurant and the swimming pool. Also help the business to set their events to bring the benefits and the customers to get into their shop hence they can get more customers and let them become much more famous than before so that they are easier to track the other customers. We also have the admin which can help to control the number of customers and businesses so that the customers and the business who actually did something wrong or bad to the whole website will be deleted by the admin. We focus on a fair deal with both the customers and the business, hence any business and customer who are interested in our website can join, and our admin will keep a track on you guys.

Gitflow

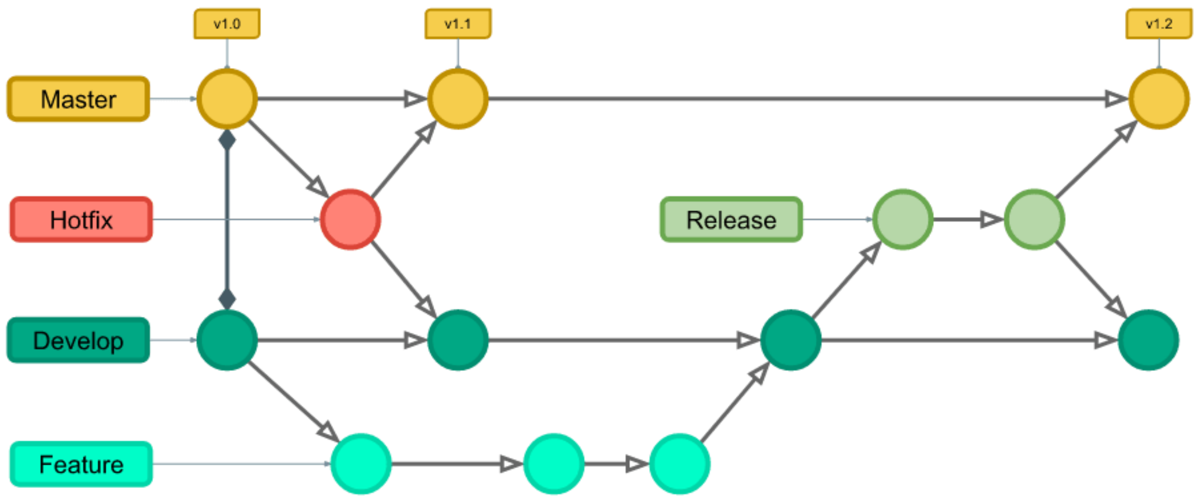
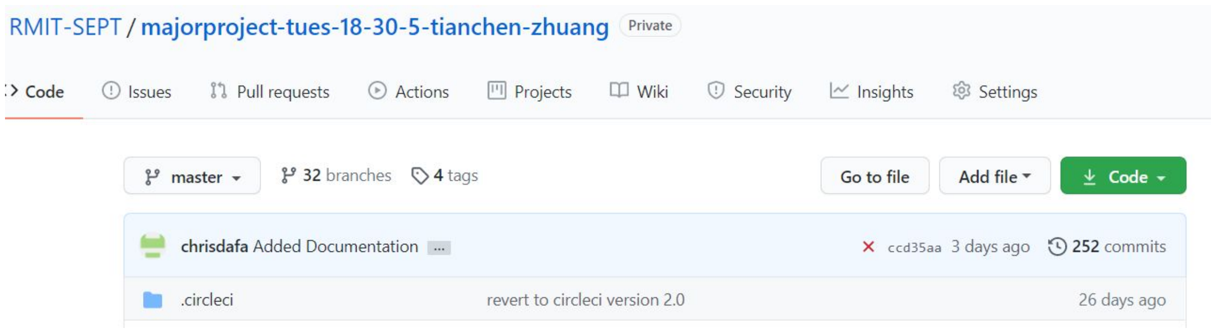


Image sourced from https://www.campingcoder.com/2018/04/how-to-use-git-flow/

The project used a standard git flow procedure which can be seen in the image above. When we first started we created a development branch named ‘dev’. From there we followed the process of creating a new feature branch for a user story ticket as they were picked up. Our project team ended up with 31 branches total but we had a couple more as we were deleting our branches after we merged them to keep the repository a bit cleaner.

Over the course of the project there have been 250 commits as well as 4 versions released. We were planning on only having 3 releases but during our second release we realised we had not merged in the frontend implementation. Below is a snapshot of the github repo as evidence to support the claims made above.



RMIT-SEPT / majorproject-tues-18-30-5-tianchen-zhuang Private

master 32 branches 4 tags

chrisdafa Added Documentation ... ✕ ccd35aa 3 days ago 252 commits

.circleci revert to circleci version 2.0 26 days ago

The repository can be accessed by the following link
https://github.com/RMIT-SEPT/majorproject-tues-18-30-5-tianchen-zhuang

Scrum process

The Scrum master throughout the project was Caspar Koutsoukis, we had initially intended to switch the role each sprint but the team didn't end up doing this. The end result of the project was a direct result of implementing Scrum processes, while it wasn't perfect execution it did mean that the team was getting together and discussing the project which meant that we never fell behind.

The Scrum process followed by the team was a traditional approach. The team was working in two week sprints with the goal of producing deliverable outcomes at the end of each sprint. The Scrum meetings the team completed were standups three times a week, sprint planning at the start of each sprint and sprint retrospectives at the end.

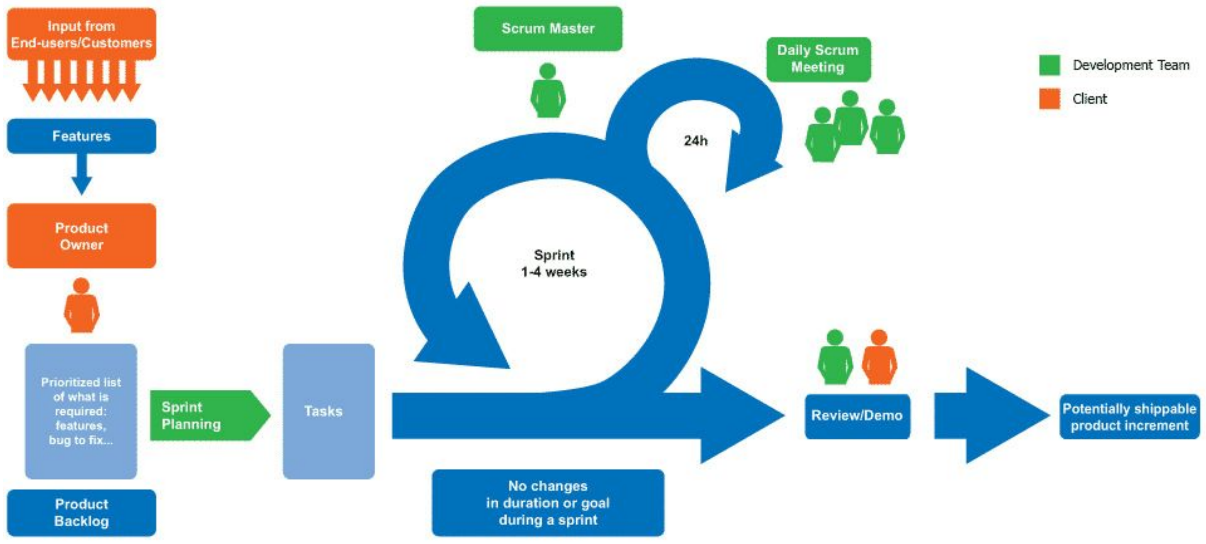

Input from End-users/Customers
Features
Product Owner
Prioritized list of what is required: features, bug to fix...
Product Backlog
Sprint Planning
Tasks
Scrum Master
Daily Scrum Meeting
24h
Sprint 1-4 weeks
No changes in duration or goal during a sprint
Review/Demo
Potentially shippable product increment
Development Team
Client

Image sourced from https://www.nutcache.com/blog/how-to-use-scrum-to-boost-teams-productivity/

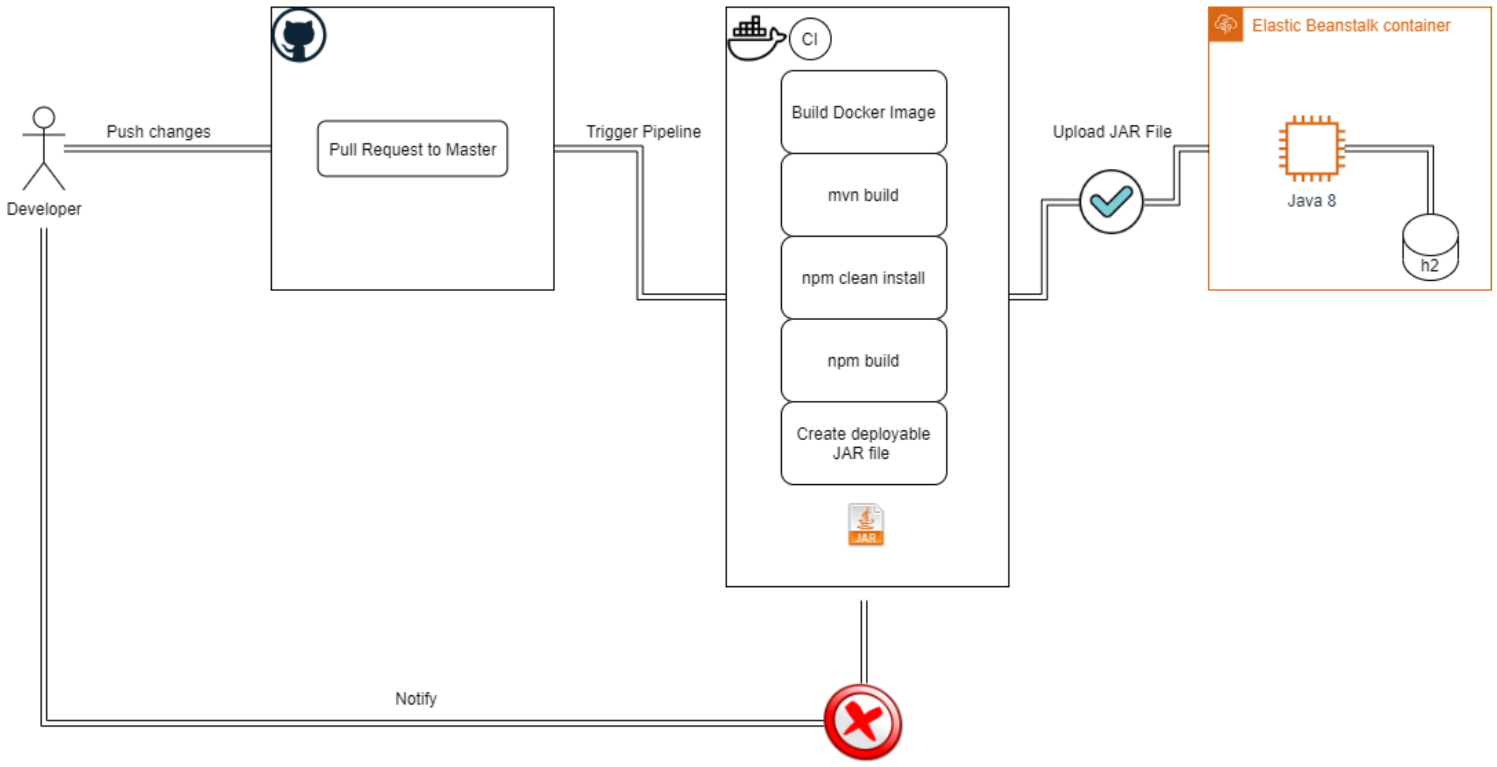
To complete the project while having deliverable outcomes after each sprint the team created user stories based on the project requirements. This method of breaking the task down into actionable items allowed the team to successfully plan the project and then delegate the work when development started. These items were placed into a product backlog that the team then prioritized and grouped together to ensure we could deliver a minimum viable product at the end of each sprint.

Using an online program called Clickup for our workflow management tool ensured that team members had visibility on what was being worked on and what could be picked up. Creating lists on clickup enhanced viability, we had one for our product backlog and then one for each sprint. During our sprint planning meetings we would discuss which items to be committed to the sprint, this was a difficult task for sprint 1 as we didn't know what the team was capable of but it got easier as the development progressed.

At various stages throughout the project the team had demos to our client which provided valuable insight into the direction of our product. The interactions with the client would happen after each iteration was released as is standard with the scrum process.

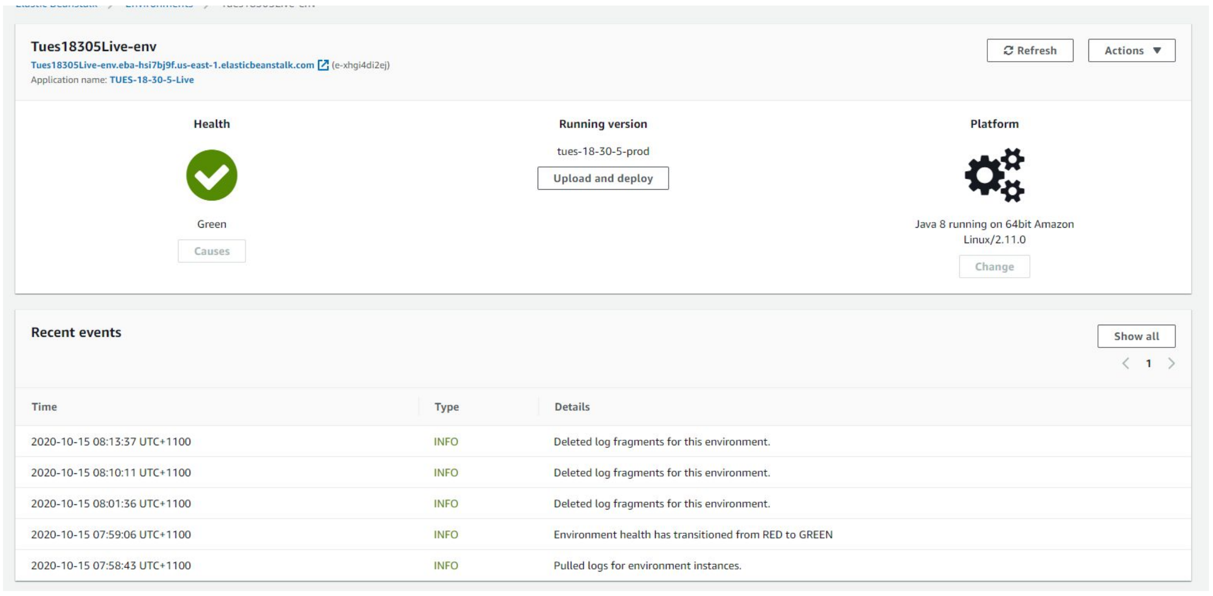
Deployment pipeline setup: CI/CD, steps in the pipeline, automation tools in the pipeline

The above diagram shows our intended deployment pipeline linking our Github repository to CircleCI so that we created PR to master which as explained in our gitflow was our release strategy. Upon triggering the pipeline CircleCI would read from our config yaml file and run basic maven and nodejs commands to install dependencies, run unit tests and finally create a deployable jar file. To deploy this jar



file in AWS we used Elastic Beanstalk to create a ec2 instance with java 8 preloaded. AWS would though automatically handle deployment.

This is mentioned as our intended pipeline is due to issues with the CircleCI account running out of credits so we could not fully complete this design. To deploy our application to AWS the jar file was created locally and then manually uploaded through the ElasticBeanstalk UI. There are screenshots below as an example of our environment and application.



Tues18305Live-env

Tues18305Live-env.eba-hsi7bj9f.us-east-1.elasticbeanstalk.com (e-xhgi4di2ej)

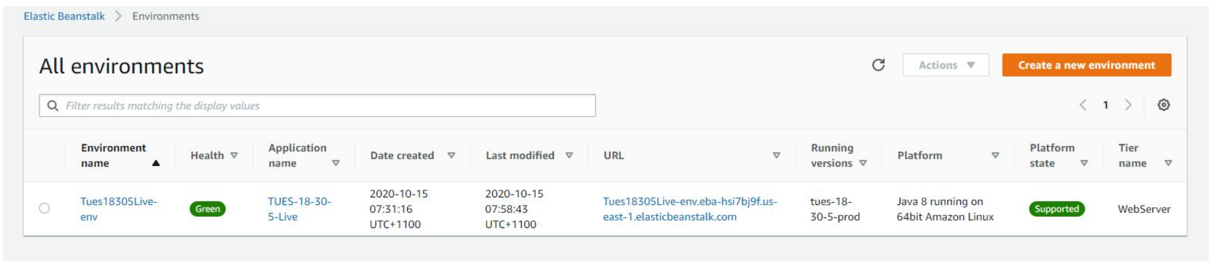
Application name: TUES-18-30-5-Live

Health	Running version	Platform
Green	tues-18-30-5-prod	Java 8 running on 64bit Amazon Linux/2.11.0

Recent events

Time	Type	Details
2020-10-15 08:13:37 UTC+1100	INFO	Deleted log fragments for this environment.
2020-10-15 08:10:11 UTC+1100	INFO	Deleted log fragments for this environment.
2020-10-15 08:01:36 UTC+1100	INFO	Deleted log fragments for this environment.
2020-10-15 07:59:06 UTC+1100	INFO	Environment health has transitioned from RED to GREEN
2020-10-15 07:58:43 UTC+1100	INFO	Pulled logs for environment instances.

ElasticBeanstalk Environment

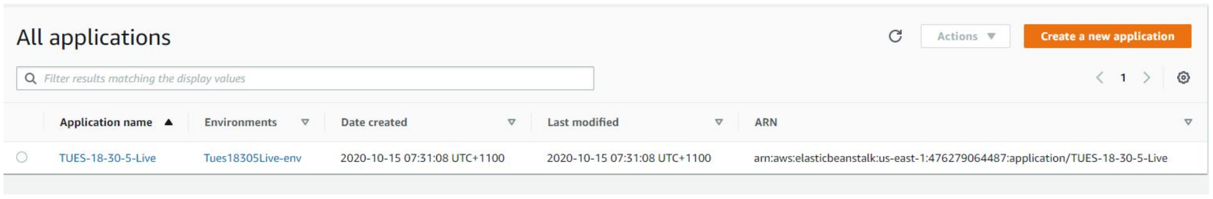


Elastic Beanstalk > Environments

All environments

	Environment name	Health	Application name	Date created	Last modified	URL	Running versions	Platform	Platform state	Tier name
○	Tues18305Live-env	Green	TUES-18-30-5-Live	2020-10-15 07:31:16 UTC+1100	2020-10-15 07:58:43 UTC+1100	Tues18305Live-env.eba-hsi7bj9f.us-east-1.elasticbeanstalk.com	tues-18-30-5-prod	Java 8 running on 64bit Amazon Linux	Supported	WebServer

Application

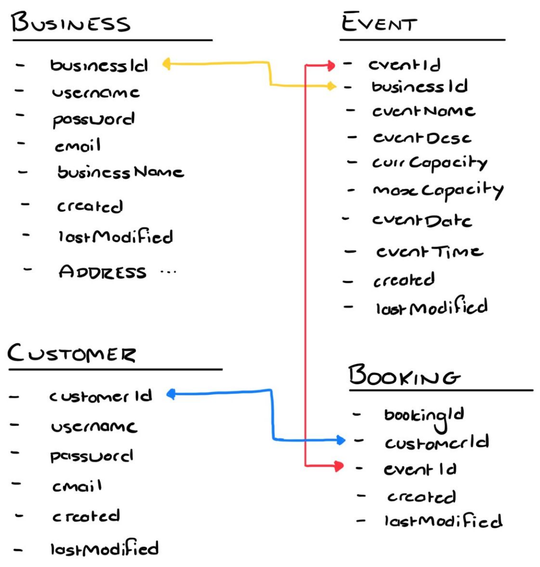


All applications

	Application name	Environments	Date created	Last modified	ARN
○	TUES-18-30-5-Live	Tues18305Live-env	2020-10-15 07:31:08 UTC+1100	2020-10-15 07:31:08 UTC+1100	arn:aws:elasticbeanstalk:us-east-1:476279064487:application/TUES-18-30-5-Live

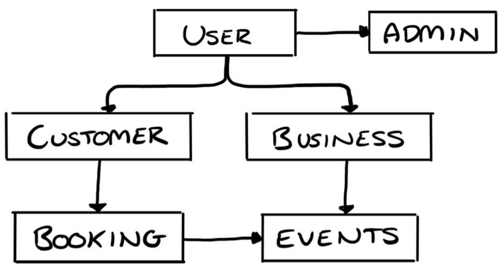
Refactoring

Around the end of development, within sprint 3, the names of classes throughout the backend did not make logical sense, therefore the backend was refactored so that the readability and understanding of the code was improved overall. Some variables within each model/class were also removed, as they were not used and no longer needed. The diagram below represents what the models look like after the refactoring.



Prior to refactoring, the ‘Customer’ model class (and therefore the API, repository and service classes where named accordingly) was named ‘User’, this did not make sense as the Business and Admin are also considered to be users to the application. Also the ‘Booking’ model originally was the ‘Event’ model where the business would create a booking and a customer would book a booking, which was to be saved in a secondary database (model) within the ‘Booking’ model. This again did not make sense and it was very difficult to understand for someone new working on the backend. It also was not the best choice for all the customers' bookings being saved as a secondary table in the primary table, as a customer could have a lot of bookings. The ‘Booking’ model was renamed to ‘Event’, which holds all events business’ make, and a new ‘Booking’ model was created, which held all bookings the customer makes to a business’ events. With the newly refactored code, the API’s for booking an event and creating events became a lot more straightforward and less confusing. A developer with no prior experience in this backend would be able to understand quickly. The refactored code overall makes it a better process and looks much cleaner.

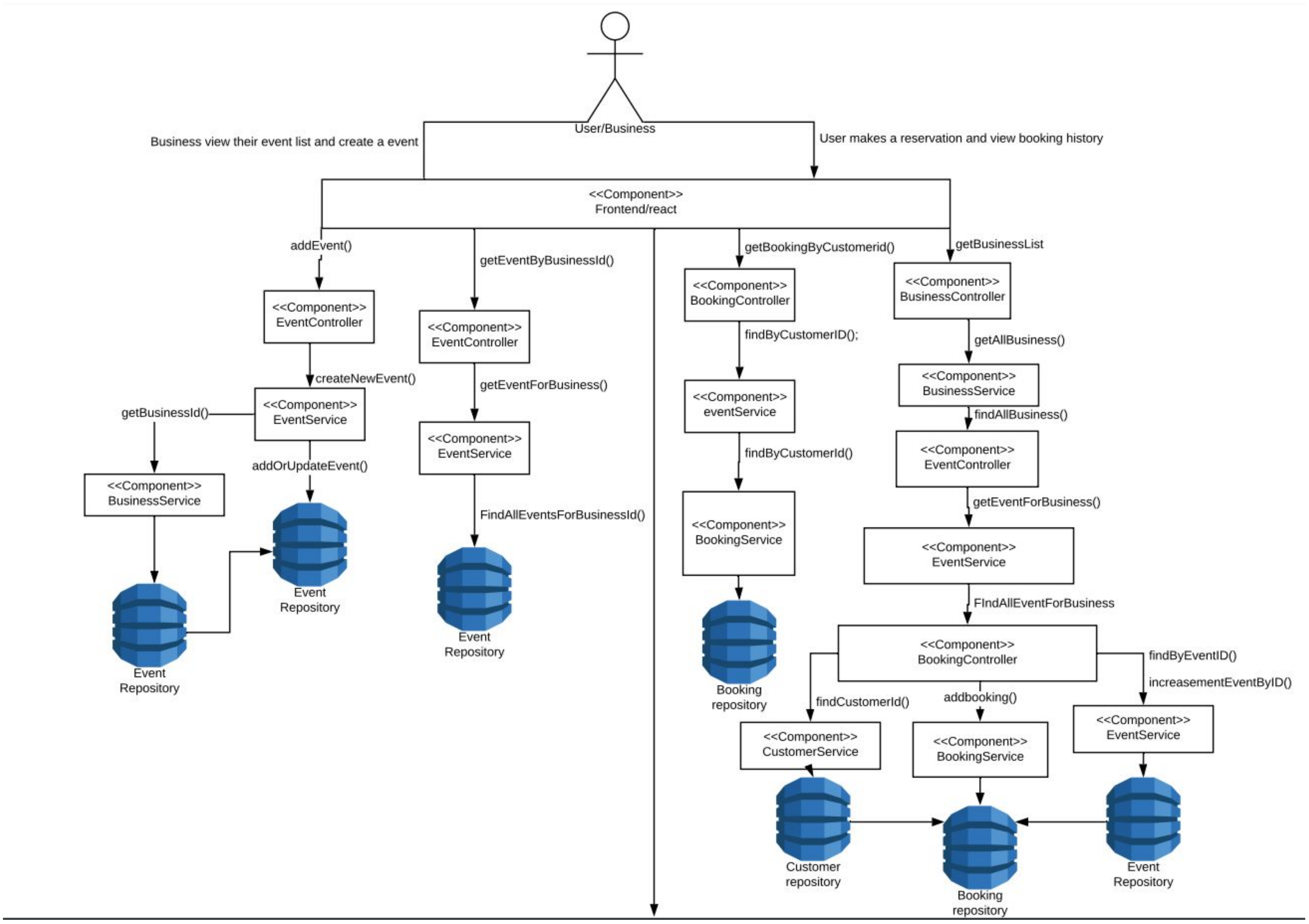
As more development has been made and the decision to include Spring Security to the application, the following diagram of the models would be ideal.



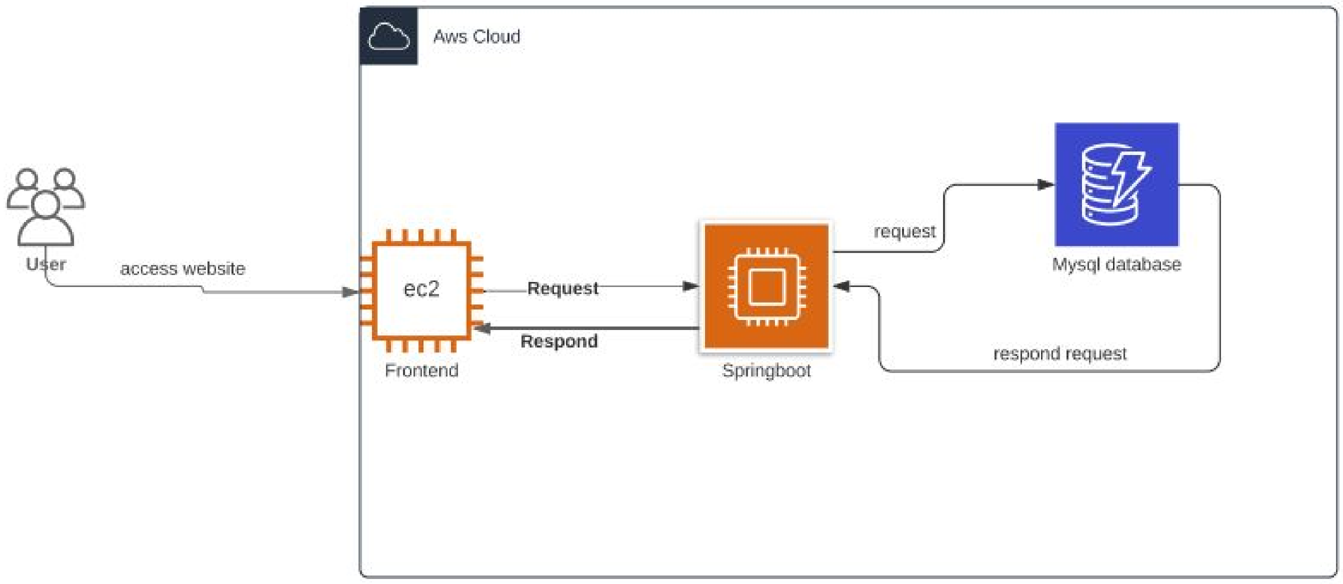
The ‘Admin’, ‘Customer’, and ‘Business’ models extending a ‘User’ model which had the ‘UserDetails’ implemented, was the goal. This was attempted in branch ‘29-Spring-Security’ on the GitHub repo. This implementation of the backend structure would have made more sense and made the code cleaner, the next refactor in backend development would be to implement this structure.

System Architecture / Design

Component diagram


User/Business
Business view their event list and create a event
User makes a reservation and view booking history
<<Component>> Frontend/react
addEvent()
getEventByBusinessId()
getBookingByCustomerid()
getBusinessList
<<Component>> EventController
createNewEvent()
<<Component>> EventService
getBusinessId()
<<Component>> BusinessService
addOrUpdateEvent()
Event Repository
Event Repository
<<Component>> EventController
getEventForBusiness()
<<Component>> EventService
FindAllEventsForBusinessId()
Event Repository
<<Component>> BookingController
findByCustomerID();
<<Component>> eventService
findByCustomerId()
<<Component>> BookingService
Booking repository
<<Component>> BusinessController
getAllBusiness()
<<Component>> BusinessService
findAllBusiness()
<<Component>> EventController
getEventForBusiness()
<<Component>> EventService
FIndAllEventForBusiness
<<Component>> BookingController
findCustomerId()
addbooking()
findByEventID()
increasementEventByID()
<<Component>> CustomerService
<<Component>> BookingService
<<Component>> EventService
Customer repository
Booking repository
Event Repository

System Architecture


Aws Cloud
User
access website
ec2
Frontend
Request
Respond
Springboot
request
Mysql database
respond request