



GymBros Booking System

13.10.2020

Jesse Osrecak

Hillson Ngo

Simon O'shaughnessy

Alexander Murphy

Berke Batmaz

Gym Bros Co

Introduction

Throughout Semester 2, our group developed a back-end and front-end api booking system, and we applied it to a scenario where customers would be needing to hire a trainer from a gym. Through four sprints, each sprint we tasked ourselves with developing an api that would be applicable and satisfactory to a real world scenario.

In this instance, we decided to call our product the GymBrosAPI™ (not exactly the most creative), to showcase how the booking system would work. Initially designed as an input form, one of our front-end experts was able to come up with a solution that simplified the overall process for our hypothetical customers: an existing time-slot that allows them to directly interact with a time-line. This visual engagement, we believe, would provide users a better 'experience' in terms of booking compared to our previous iteration.

Our Vision

Booking systems are vital to any system, especially when businesses rely on customers booking a business' services ahead of time. Our product, the GymBrosAPI, was built to satisfy the core needs for the business, by understanding how the customer thinks and feels.

The GymBrosAPI was built on the following principles:

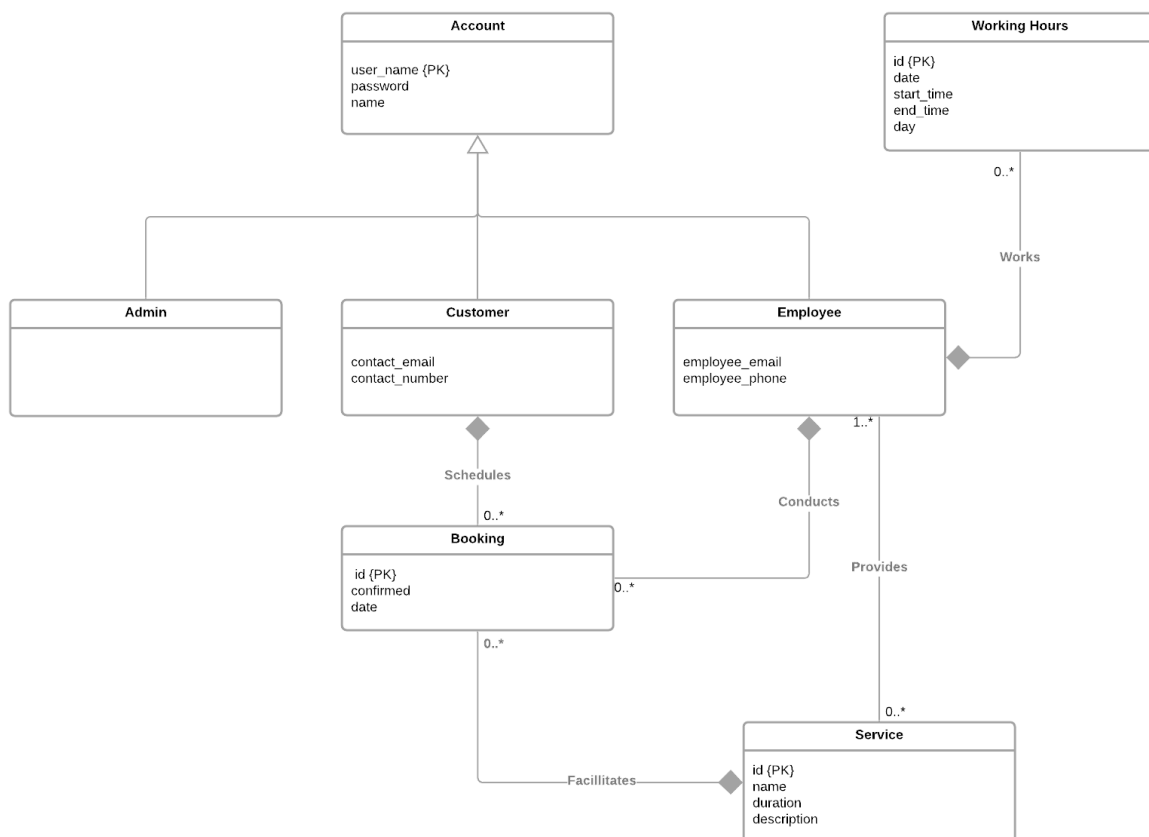
- **Simple**
 - The group knew that booking systems are complicated and take a few moments to understand the procedure.
 - By simplifying the amount of clicks, or a 'form like' appearance, users are not overwhelmed with the amount of boxes to click or fill like most systems, but instead, can just click and drag.
- **Less Is More**
 - Bookings can be overwhelming. For customers, the simplicity of the API means there are less distractions built-in, so they can quickly enter and exit from the booking situation without worry.
 - Employees can also quickly do their jobs easier as well: with our website providing a necessary highlighting of the information, all the critical information is quickly seen by them so they can perform their duties with confidence.

- **Presentation Matters**
 - Customers judge websites and their functionalities by their looks - and looks are vital in attracting their initial attention. We structured our front-end and booking systems to be easier on the eyes - not only for attractiveness, but also for usability as well.

Project Structure

I. Database

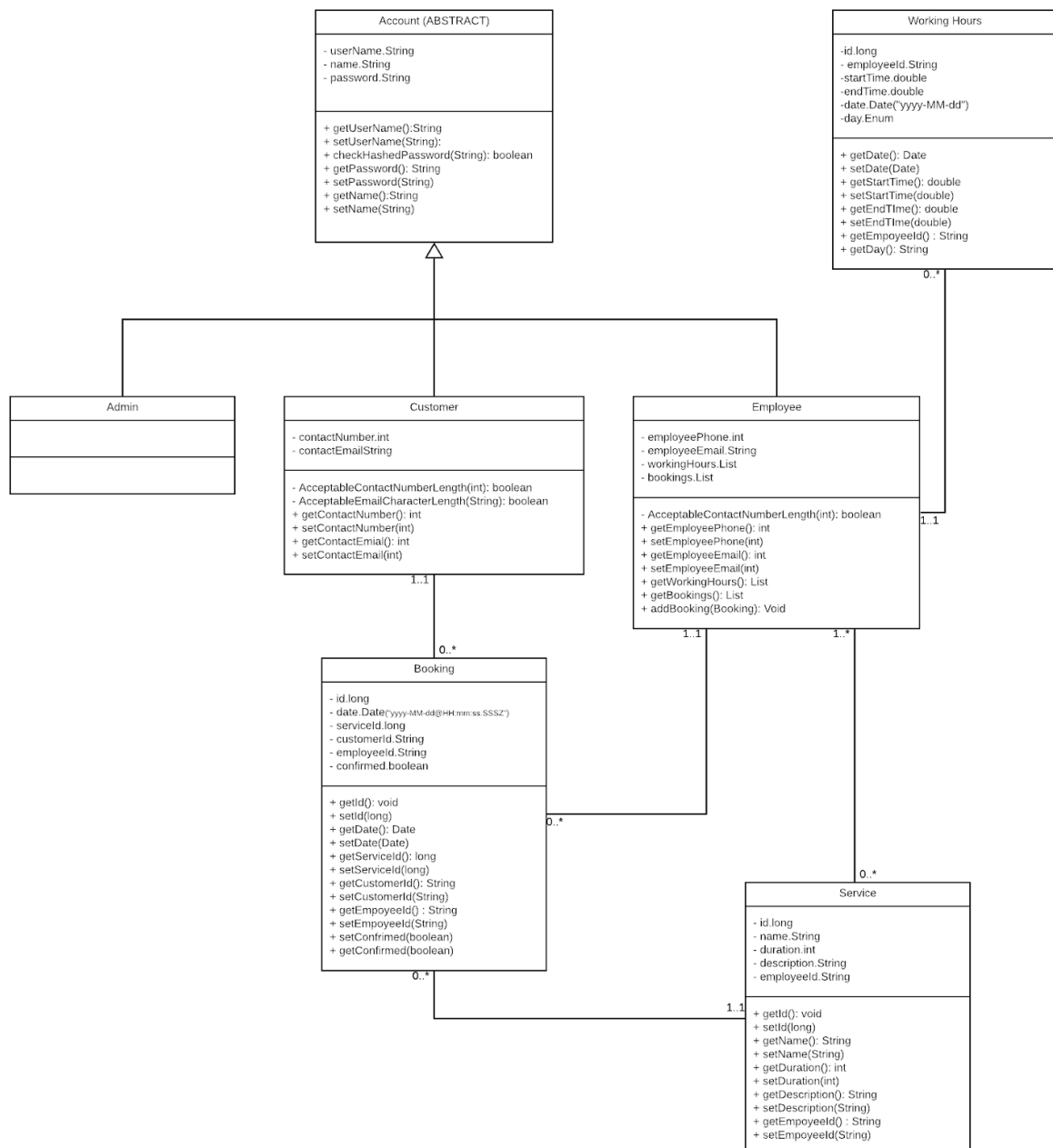
The intent for the database was to store all relevant information for each entity with minimal repetition. E.g our booking entity contains a reference to its id, date and confirmation status, as well as foreign keys that belong to associated database entries; employee, customer and service. This keeps the database simple without restricting information flow.



II. Backend

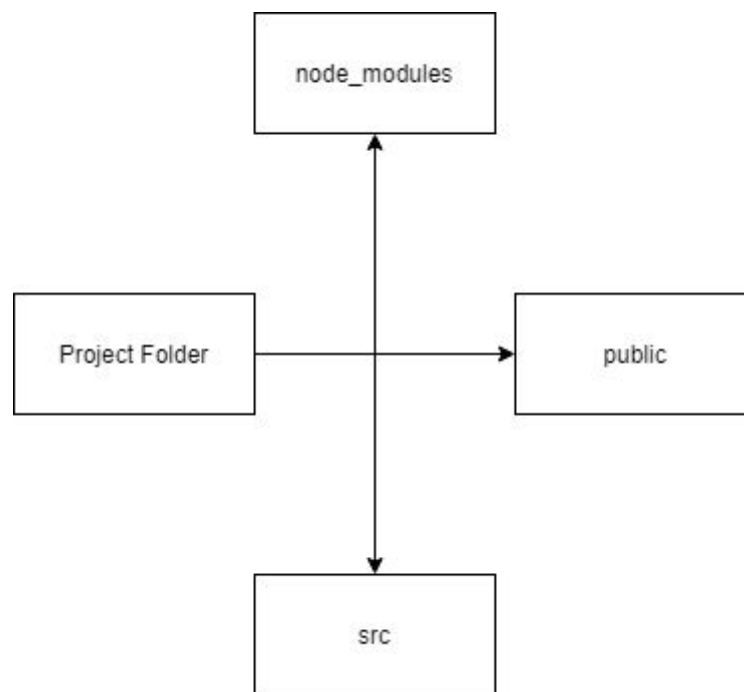
The backend holds the main logic for the API. Most models consist of simple getters and setters and the corresponding service class processes any logic requests. E.g. The BookingService contains a method that calculates and returns the remaining available unbooked time of an employee, given only the employee's username. We also vied for abstraction when it came to user accounts to increase polymorphism and reduce code repetition.

The Front-End makes calls to the backend controllers requesting information for it to display. We sought to minimize the data processing in the front end with it instead focusing on UI and page state programming.



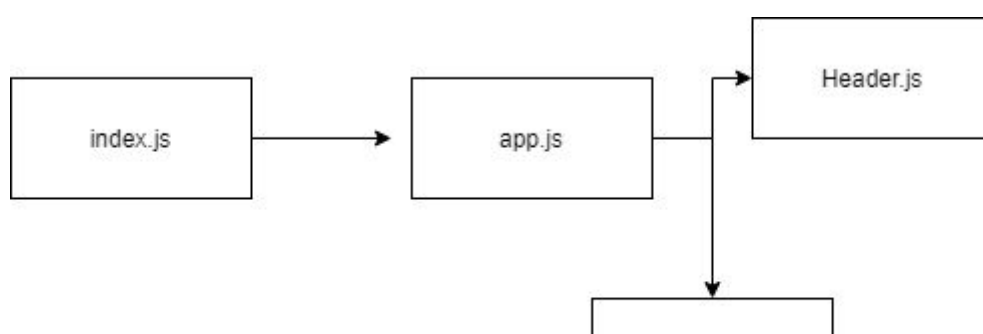
III. Frontend

The design of the frontend closely modelled the structure of a standard webpage then the designer initially created a basic react webpage, via the command, `npm create-new-app`.



All of the existing code, especially the working code would be managed under the 'src' folder. Node_modules itself contain the various libraries used. For our project, besides the main react library, we also used 'react-bootstrap' for both the aesthetic and combined functionality of bootstrap for our components. Public contained images that we used for our homepage, such as the landing page image, customer/employee image (images of individuals **who do not exist, as they are AI generated**), and the group's branding icon.

Under the `src` folder, we housed all our components which functioned separate components that either represented a specific module, or a layout management. Code-wise, `index.js` calls `App.js`, in which `App.js` then manages **both** `Header.js` and `BaseRoutes.js`. The reasons are explained under '**Refactorisation - Front-End Restructuring**'.



Refactorisation

I. Front-End Restructuring

The initial internal development of the front-end website was based on a basic, and primitive understanding of how websites were built. That is, to place elements together, one after another like an essay and hope for the best. As the development team began to further customise the website, minor changes on one end created issues of **visual inconsistency** with the other front-end pages. This would violate the group's vision of *Presentation Matters*.

Before	After
<pre>function App() { return(<div> <Route exact path ="/Customer" component = {Customer}/> <Route exact path ="/Employee" component = {Employee}/> </div>); }</pre>	<pre>function App() { return (<div className = "AppSite"> <div className = "SiteContent"> { /* Header Related Content Placed here */ } <div className = "AppHeader"> <Header/> </div> { /* Main Content Added here */ } <div className = "AppMain"> <BaseRoutes/> </div> </div> { /* Sticky Footer can be placed here */ } </div>); }</pre>

As seen in the comparison above, before the abstraction, App.js featured a 'add an ignore' mentality, which meant when visual changes were forced via the CSS, it was tedious to handle. By abstracting all the non-header related pages to **<BaseRoutes/>**, this meant that a singular change in App.css instead would impact all of their respective main-content pages under <BaseRoutes/>. This allows for efficiency for and consistency.

While it may seem to be a basic shift, there are also more numerous, undocumented changes to the code-base concerning the front-end throughout the developmental period. This includes the introduction of a new library (React-Bootstrap), where its components are used heavily throughout different sections of the website to highlight vital information.

Furthermore, as the front-end developers understood the nature of react-bootstrap and javascript with its various intricacies the group made logical code changes to how certain elements within the page should be rendered.

II. Website Navigation Restructure

At the end of Milestone 2, each of the various ‘features’ and ‘components’ required by the requirement documentation were illogically placed within the main header/navigation bar.



From an outsider’s perspective, various options often lead to curiosity, where users can accidentally access pages where they do not have certain privileges, namely account type. For example, any user was able to access the admin’s dashboard and create security issues.

With the finished result of the security token, as well as the ‘Redirect’ component from react allowed the developers to ‘push’ users to their account specific dashboard after signing into their accounts. That is, when an employee signs in, they would be automatically redirected to their dashboard, however, they can still navigate to other webpages if they wish to do so, namely the homepage and booking functionalities.



The developers believe this is a quality fixture that is beneficial to most employees and customers, since this reduces confusion on the question, “what next”.

III. UI

First implementations of many user interfaces where unintuitive forms with manual typed input, and tedious clicking through menus. Booking creation and editing employee working hours were two of the pages most in need of a revamp. Considering booking is the main purpose of the application it’s improvement was crucial to an effective user experience.

The improved UI can be seen below; rather than cumbersome forms and labels we changed the displays to communicate the information visually and compactly. We allow the user to interact with the onscreen elements with a hover or click of the mouse ultimately affecting our vision of a *simple, less is more* and well *presented* application.

Hours for Employee:

Jim_User

Scheduled **Not Scheduled** -Hover for details; Click (and drag) to toggle.

Date	0:00	2:00	4:00	6:00	8:00	10:00	12:00	14:00	16:00	18:00	20:00	22:00	24:00
10/18	[Grid of scheduled and unscheduled blocks]												
10/19	[Grid of scheduled and unscheduled blocks]												
10/20	[Grid of scheduled and unscheduled blocks]												
10/23	[Grid of scheduled and unscheduled blocks]												
10/24	[Grid of scheduled and unscheduled blocks]												

New Day: dd/mm/yyyy

Start Time: 19:30
End Time: 20:00

Submit

Create Booking

Enter your require service, preferred employee and select a available time-block.
If no times appear, please try another employee.

Service: Medical

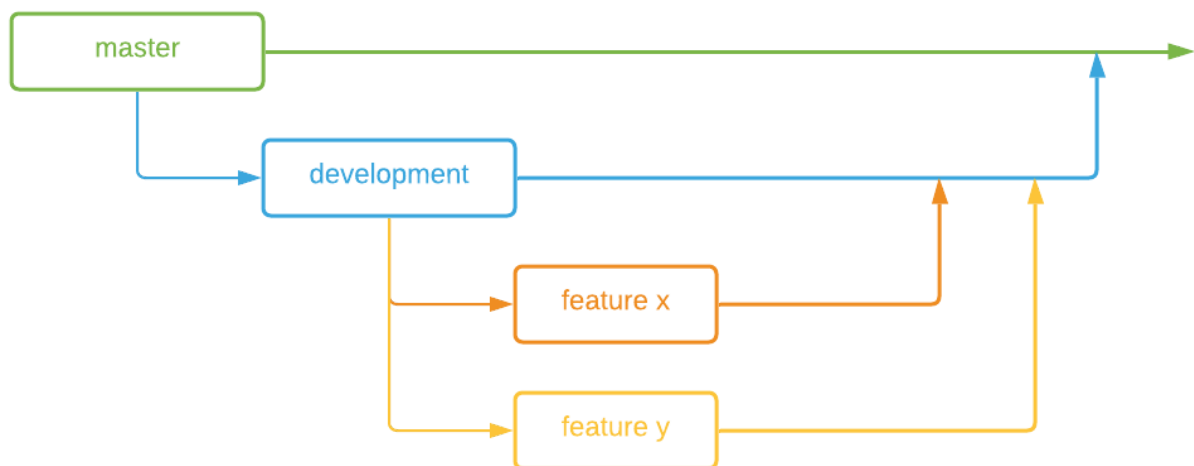
Employee: Jim #Jim_User

Available Times -Hover for details; Click to book.

Date	0:00	2:00	4:00	6:00	8:00	10:00	12:00	14:00	16:00	18:00	20:00	22:00	24:00
10/19	[Grid of available time blocks]												
10/20	[Grid of available time blocks]												
10/23	[Grid of available time blocks]												
10/24	[Grid of available time blocks]												

Gitflow

Git was heavily utilised throughout each milestone, along with our task management tools, it helped keep all our tasks and features organised. We could make sure they worked in isolation before introducing in other features that others had implemented. Each sprint starts and ends on the development branch.



Master

The master only contains an implementation that all group members are happy to submit as a milestone. No work was ever done directly on the master branch and it was only ever merged to from the development branch once everything was up to standard.

Development

At the start of a sprint, all members that wanted to add a feature would branch off of development. Development was only worked on directly occasionally, once a few features had been merged back into it. This was to ensure that the features were working together as intended and to add final touches to make a milestone ready for submissions (and merging to master).

Feature Branches

Individual feature branches were where most of the direct implementation work took place. Every time a group member wanted to work on a new major feature, they would branch off of development. If it was a minor feature that was related to an already existing feature, they would branch off of the existing feature branch, do the work, and merge back in a relatively short span of time. In multiple instances, two or more important features that were completed and had to be merged to development, would first merge together in order to sort out any merge conflicts and issues that may arise. Once they were known to be working, that branch could then be merged into development.

Each group member was not at all isolated to one feature branch that they created. All throughout the sprints members would pull from each other's branches to assist each other, as well as create multiple feature branches themselves and work on them in parallel where it was helpful to do so.

Scrum Process

For our project we undertook the scrum process with the Agile Development framework. Jesse acted as the scrum master throughout the entirety of the project, organising not only the meetings, but establishing the priority user-stories for each sprint, balancing the burndown chart, and cleaning the overall product backlog.

For our Scrum meetings, the group scheduled to meet every Monday, Wednesday, and Friday. Wednesday was a day where all the group members had no clashing or urgent classes; it was dedicated to assignment work, and longer meetings - such as Sprint planning. Friday's meeting was held after the lecture so that if any new information about the project was released, we could discuss it as a group, as well as finalising our sprint retros. Our Monday meeting was often dedicated to planning and code reviewing. Aside from these meetings we would also meet on other days if there were any complications with undergoing the project, mainly for debugging and aiding others in working towards their goals.

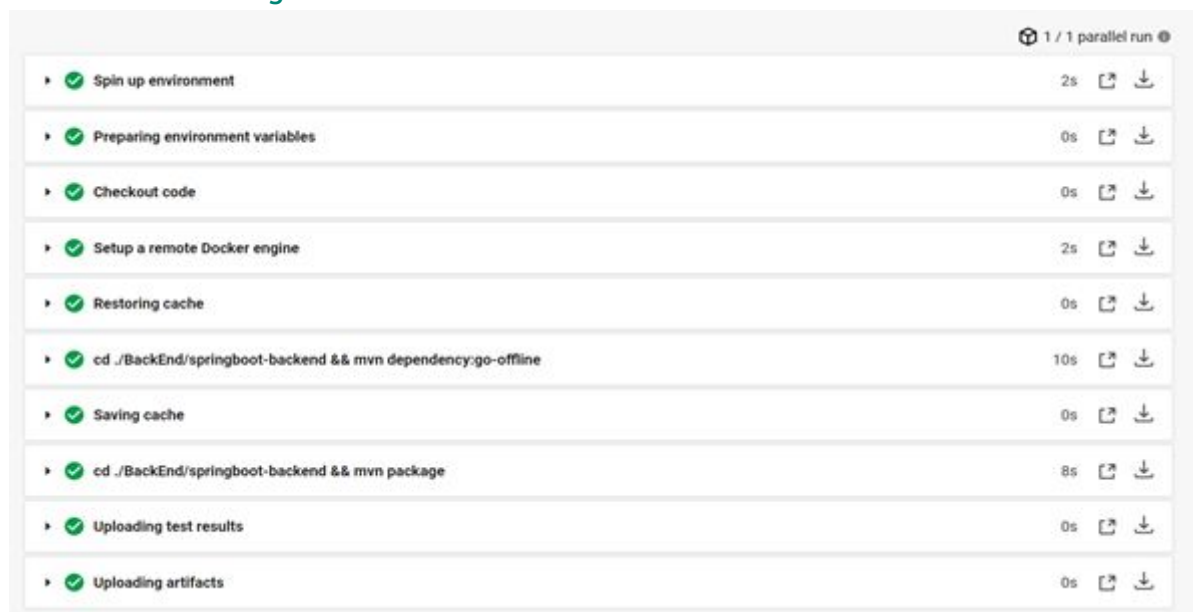
All our Stand-up meetings (Monday, Wednesday & Friday) were documented with Hilson taking notes during the meeting and Jesse Reviewing them. All of our tasks were documented using Trello so that we could easily see what people were working on and how much progress they had made towards a certain task.

Deployment

Our main development tool that we employed to streamline our project was CircleCI, for our continuous integration and continuous delivery. CircleCi proved to be very useful as it allowed us to run tests and builds automatically through Github whenever a pull request was made. This allowed for a greater focus to be placed on the coding aspect without having to manually run tests ourselves.

Due to credit running out on the university account, we needed to create a private account in order to set up the CircleCI automation process before merging it back into our main project repository to allow us to run automated tests. This caused some small issues with merging back and forth and in general added another layer of difficulty to the set-up, but overall was a rather minor issue.

Automated Testing



The screenshot displays a CircleCI job execution log for a single parallel run. The job consists of ten steps, all of which completed successfully, as indicated by green checkmarks. The steps include environment setup, code checkout, Docker engine setup, cache restoration, dependency installation, cache saving, packaging, test result upload, and artifact upload. The total duration of the job is 2s.

Step	Duration	Status
Spin up environment	2s	Success
Preparing environment variables	0s	Success
Checkout code	0s	Success
Setup a remote Docker engine	2s	Success
Restoring cache	0s	Success
cd ./BackEnd/springboot-backend && mvn dependency:go-offline	10s	Success
Saving cache	0s	Success
cd ./BackEnd/springboot-backend && mvn package	8s	Success
Uploading test results	0s	Success
Uploading artifacts	0s	Success

```
cd ./BackEnd/springboot-backend && mvn package

Hibernate: insert into service (id, description, duration, employee_id, name) values (null, ?, ?, ?, ?)
Hibernate: insert into service (id, description, duration, employee_id, name) values (null, ?, ?, ?, ?)
Hibernate: insert into working_hours (id, date, day, employee_id, end_time, start_time) values (null, ?, ?, ?, ?)
Hibernate: insert into working_hours (id, date, day, employee_id, end_time, start_time) values (null, ?, ?, ?, ?)
Hibernate: insert into working_hours (id, date, day, employee_id, end_time, start_time) values (null, ?, ?, ?, ?)
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.486 s - in com.sept.majorproject.group
2020-10-16 11:40:40.231 INFO 235 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA
2020-10-16 11:40:40.231 INFO 235 --- [extShutdownHook] .SchemaDropperImpl$DelayedDropActionImpl : H000000477: S
Hibernate: drop table if exists admin CASCADE
2020-10-16 11:40:40.234 WARN 235 --- [extShutdownHook] o.h.engine.jdbc.spi.SqlExceptionHelper : SQL Error: 9
2020-10-16 11:40:40.234 ERROR 235 --- [extShutdownHook] o.h.engine.jdbc.spi.SqlExceptionHelper : Database is
2020-10-16 11:40:40.236 WARN 235 --- [extShutdownHook] o.h.engine.jdbc.spi.SqlExceptionHelper : SQL Error: 9
2020-10-16 11:40:40.236 ERROR 235 --- [extShutdownHook] o.h.engine.jdbc.spi.SqlExceptionHelper : Database is
2020-10-16 11:40:40.236 WARN 235 --- [extShutdownHook] o.s.b.f.support.DisposableBeanAdapter : Invocation c
2020-10-16 11:40:40.236 INFO 235 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting dow
2020-10-16 11:40:40.237 INFO 235 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1
2020-10-16 11:40:40.238 INFO 235 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ sb-backend ---
[INFO] Building jar: /home/circleci/project/BackEnd/springboot-backend/target/sb-backend-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.2.RELEASE:repackage (repackage) @ sb-backend ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 7.901 s
[INFO] Finished at: 2020-10-16T11:40:41Z
[INFO]
CircleCI received exit code 0
```

Above is an example of one of our backend service's testing rundowns along with the build success dialogue. This is an automatic process that runs whenever a pull request is made. The test being successful automatically allows the pull request to go through which greatly helped in decreasing the amount of potential human errors that could arise, and therefore increasing development time.

Additional

This section features additional aspects and/or commentary to highlight topics that are not listed within the given assessment rubric.

Restrictive Testing

Previously, **before** the implementation of the security token, the group had developed initial automated Springboot integration tests for its repository, services, and controller classes. These initial tests were created for the customer component. The tests were able to pass and ran when required.

However, with the development of the Springboot Security and the implementation of its security features, the team could not further automated testing for the Springboot components despite their best efforts. The reasoning is this:

- To test the Springboot components, there was a specific dependency that needed to be added to the pom.xml file. This can be done **only** when the project is 'reloaded' under the Maven options, updating the overall versions of the projects.

- However, updating the dependencies itself meant some of the existing annotation, classes and packages were depreciated or shifted, resulting in new implementation. This created a dozen **more** issues that needed to be resolved before the team could test the project at its previous, un-upgraded version, in order to understand the original problem..

Given the limitations of the time itself, the group felt it was necessary to efficiently spend time developing and further enhancing areas that were more critical. With the use of version control the group reverted to the working, released version of Milestone 3. However, the group does believe the fundamental idea of having these aspects be implemented, especially in the working world.

Lessons

Utilise the latest versions

As mentioned in the 'Restrictive Testing' scenario, the group learned that in the long-run, one of the fundamental aspects before the implementation, is understanding the versioning of our given tools. It is important to test 'changes' to the overall versions/dependencies early so that future issues can be resolved, in order to allow the overall project to be sufficient for further activities, i.e, testing.

The group also learned that with every integration of a framework, that requires a new 'set' of annotations and syntax, thus, would influence how and when the users must test their code. While the team **did** do testing as they developed their code-base, automated testing is restricted in this given scenario.

Summary

Overall, the group tried its best to produce a product of high quality, as demonstrated in our booking system that was slowly developed and refactored overall the course of the project. We also carefully cared to know *how* to present our API, to demonstrate our understanding of 'corporate' professionalism. Without a doubt that there are errors experienced throughout the project. But these are lessons that the group will carry on to our next project, to become better software engineers.