RMIT UNIVERSITY

AGME Booking System

Major Assignment - Group 9.Mon-1530-2

Team:

Hue Phuong Le	s3687477
Vincent Villaflores	s3728807
Chhayhy Kourn	s3699618
Meng Kheang Leng	s3704080
William Bossen	s3658961

Table of Contents:

Vision statement

AGME’s Online Appointment Booking System (OABS) is a custom made versatile booking system designed to make bookings as convenient as possible.

It is designed for self service, whether it be a customer or administrator, the need for calling to book or have physical data is no longer necessary. Our system streamlines the booking process and holds important information that administrators can view whenever they wish, vastly improving business efficiency.

Calling to make an appointment is an old and time consuming way to book a service, with OABS, running out of time to book becomes a thing of the past. In just 20 seconds from login to log out you can book your favourite service for the time you want, greatly improving the customer experience that most are so complacent with.

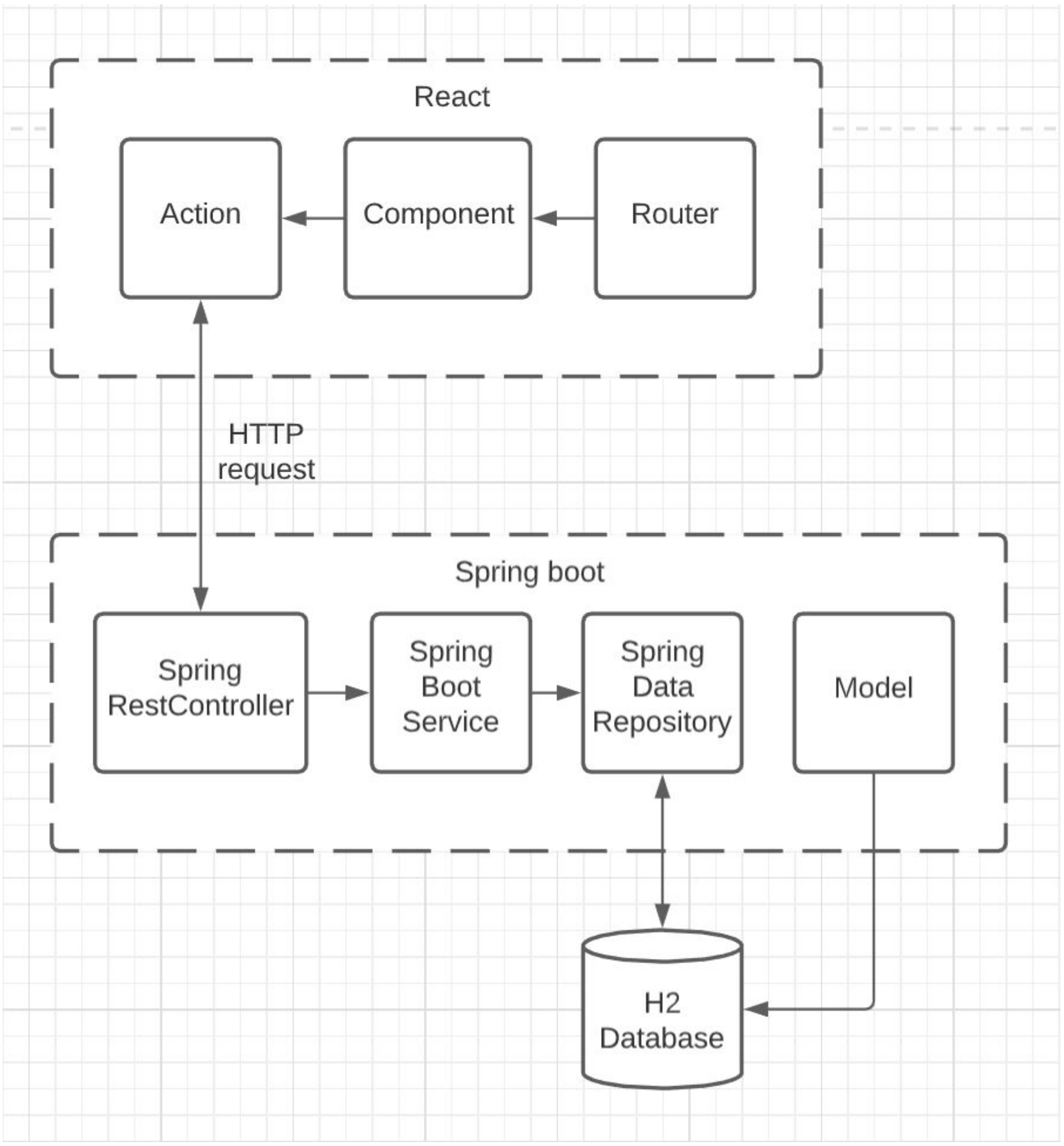
For customer usability we have combined our teams design knowledge to create a system that looks and feels as easy to use as possible. After logging in, everything a customer or admin could ever need is just one click away, no hard to find submenus and no unnecessary pages, just an easy to understand main menu that understands user behaviour.

The modern customer is no longer impressed by a “Call to book” process which has been used for so long, they’re expecting a booking system to handle their every need. This is why our booking system is ready to implement into any setting, transforming business to the next standard of professionalism.

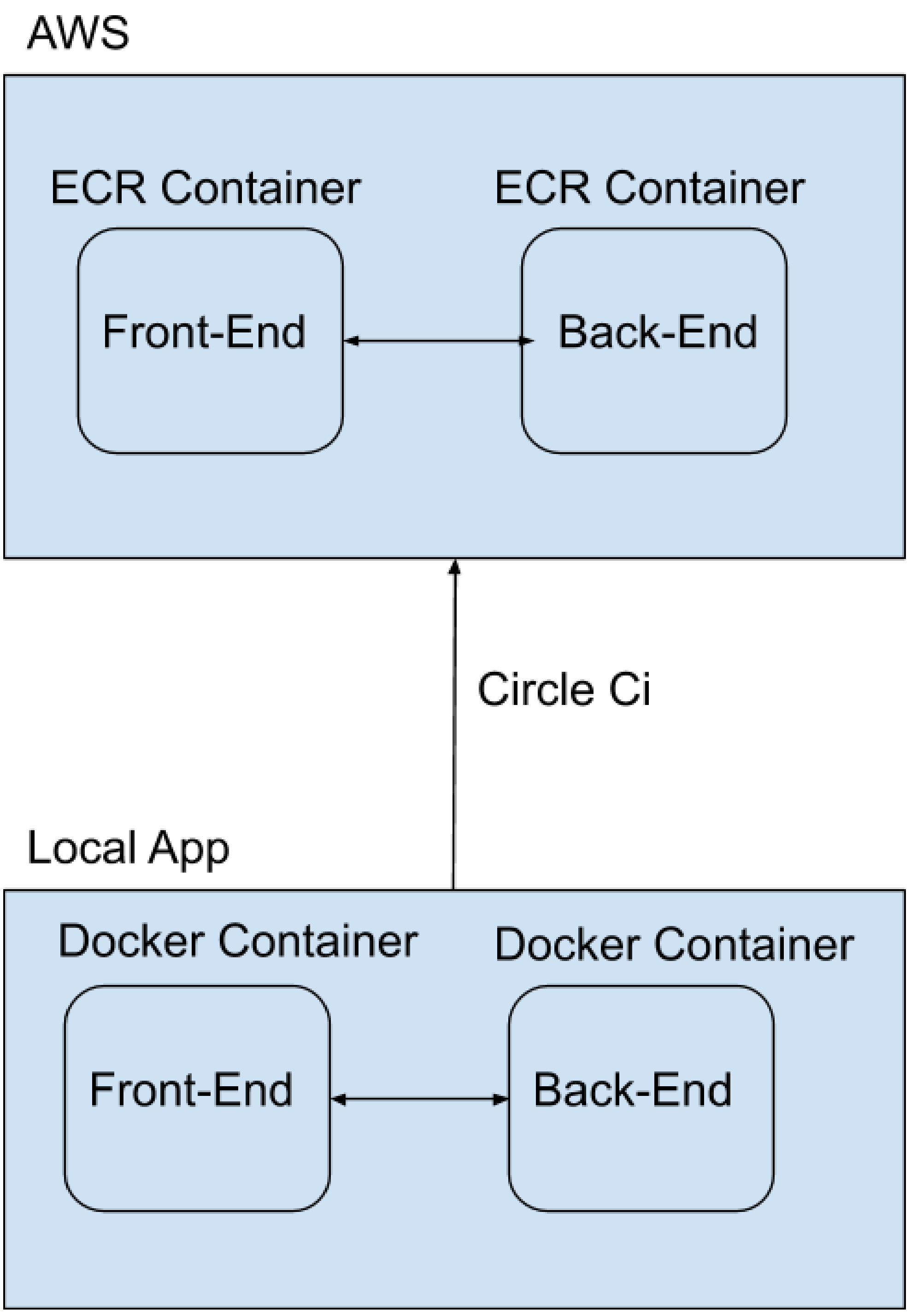
OABS created the opportunity for businesses to be ahead of the competition and implement a process that their opposition haven't even discovered yet.

Diagram

1. Diagram of final basic system architecture



2. Diagram of deployment setup



Refactoring

There are some problems found out from the backend code that have bad smells and deserve refactoring:

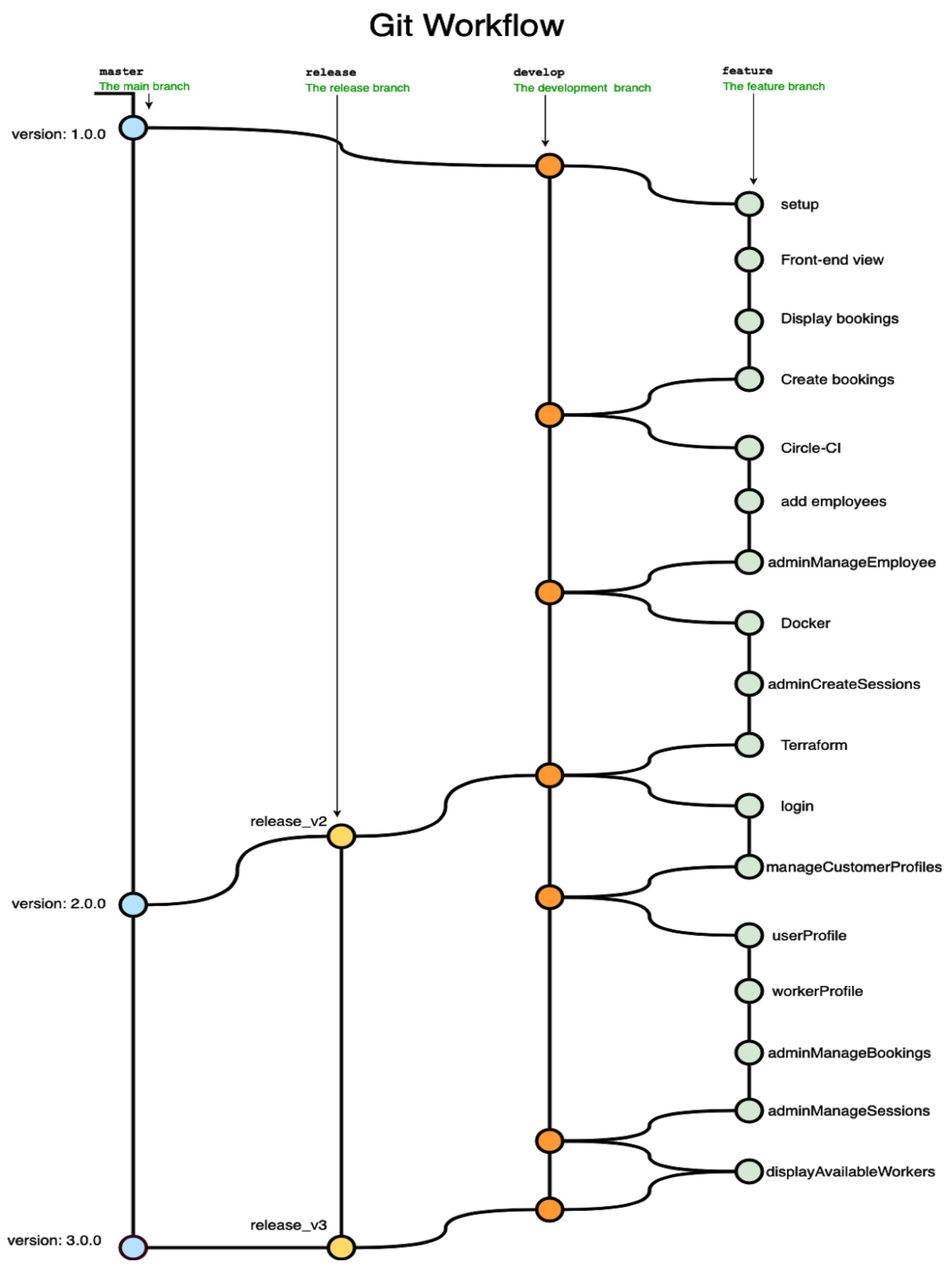
- Needless complexity: After performing debugging, there are two classes which are never used: CustomAuthenticationSuccessHandler which is never called from SecurityConfig and InvalidLoginResponse since frontend can check for the unauthorized status returned from server and display message immediately. Therefore, the two classes are removed.

- Duplicated code: two classes UserService and CustomUserService basically behave the same way, the only difference is CustomUserService implements from UserDetailsService and has to override the function to return the user when UserService does not have to. They all get user details from UserRepository class. Therefore, the class CustomUserService is discarded and UserService then implements UserDetailsService and overrides the method instead.

- UserValidator and PasswordValidator all validate the new and confirm passwords but one checks for login request and one checks for update password request. However, since each validator can only support one class, these two classes behave similarly but support different classes. Therefore, these two classes are both deleted and a function to validate passwords is added to the Utility class (the class provides all helper methods).

Consequently, after performing the refactoring, the team can ensure that there are no duplicated and unused classes or functions in particular from the code. The code now looks less confusing than previous versions, the design is improved because dirty code is all turned into clean code.

Gitflow overview



The commit in github is performed everyday with different team members. When we start implementing a new feature, we create a new feature branch (which is based on the develop branch). After finishing all the implementation for each sprint, we merge them all from feature branch to develop branch. Then the develop branch is merged to release branch and release branch is merged into master branch.

Scrum process

Meeting Frequency

Our team set a fixed schedule for regular meetings. Every week since we started the project, we had 3 meetings each week which are on Monday, Wednesday and Saturday. We had specific goals in every meeting.

Meeting Process

Every meeting on Saturday, we had sprint reviews from the last sprint and we would have sprint retro after that.

While doing sprint reviews, our team would check if all features had been implemented and are working well. We would ask each other questions to get the idea of where everyone is. After a few minutes of discussion, we would move on to do sprint retro. We would ask each other questions of what went well, what that does not went well and what to improve on. Also, we documented the answers of our team members into a file called sprint retro followed by the number of sprints.

On Monday, we would have a meeting with our client (tutor) to prioritize the features that we would have to implement next. After finishing with our client, we would extend half an hour more to discuss between our group, to divide tasks to our team members and make sure everyone understood their tasks that we were given. The extra minutes we put in were also to give team members to ask questions and find out if anyone is being left behind. If however, someone is being left behind, we would then help that person to fix their problem. Fortunately, our team members had their own set of skills, which as a result, work well together. We rarely have huge problems.

On Wednesday, we would have a brief meeting to check up on team members. If we are on tracks or if anyone needs some assistance, we would jump in and help to solve the problem immediately.

Scrum Master

Our scrum master is Vincent Villaflores.