12/10/2020

Transport Ticket System

Module: CS4125

Systems Analysis & Design

Lecturer: Anila Mjeda

Submitted by:

Eoghan Russell 17202124

Pawel Ostach 17214211

Damian Skrzypek 17217679

Darragh Kelly 17235545

Table of Contents

[Business Scenario 3](#_Toc58237110)

[Software Lifecycle 4](#_Toc58237111)

[Waterfall Model 4](#_Toc58237112)

[Agile 4](#_Toc58237113)

[Project plan 5](#_Toc58237114)

[Requirements 7](#_Toc58237115)

[Functional 7](#_Toc58237116)

[Use Case Diagrams 7](#_Toc58237117)

[Use Case Descriptions 9](#_Toc58237118)

[Non-Functional Requirements 14](#_Toc58237119)

[Technical: 14](#_Toc58237120)

[Security: 14](#_Toc58237121)

[Usability: 14](#_Toc58237122)

[Reliability & Performance: 14](#_Toc58237123)

[Quality Attributes 14](#_Toc58237124)

[Security 14](#_Toc58237125)

[Usability 14](#_Toc58237126)

[Reliability & Performance 15](#_Toc58237127)

[GUI Prototypes 15](#_Toc58237128)

[System Architecture 17](#_Toc58237129)

[Package Diagram 17](#_Toc58237130)

[Model View Controller(MVC) Discussion 18](#_Toc58237131)

[Android Studio: 18](#_Toc58237132)

[AWS database & hosting the Spring backend: 18](#_Toc58237133)

[JUnit: 18](#_Toc58237134)

[System Analysis 19](#_Toc58237135)

[Candidate Classes 19](#_Toc58237136)

[Initial identified class list: 19](#_Toc58237137)

[UML Class Diagram 20](#_Toc58237138)

[Sequence & Communication Diagrams 21](#_Toc58237139)

[Entity Relation Diagram 23](#_Toc58237140)

[Tabular Class Listing 24](#_Toc58237141)

[Code 24](#_Toc58237142)

[Added Value 24](#_Toc58237143)

[GitHub – Version Control 24](#_Toc58237144)

[Concurrency 24](#_Toc58237145)

[REST architectural pattern 24](#_Toc58237146)

[AWS Database 25](#_Toc58237147)

[Sprint Boot Deployment - Damian 25](#_Toc58237148)

[Recovered Architecture and Design Blueprints 25](#_Toc58237149)

[Critique 26](#_Toc58237150)

[Overview 26](#_Toc58237151)

[Changes between Analysis Sketches & Blueprint 26](#_Toc58237152)

[References 27](#_Toc58237153)

# Business Scenario

This project is a transport ticket booking mobile application. The app will be capable of letting its users being able to book and pay for transport tickets. It has the capabilities of letting the user once they have booked tickets to view these tickets and be able to cancel them.

This system works as follows:

The project will be based on a mobile application for the practicality of this app being used on public transport. The app will be an easy to use GUI user interface, providing a user experience that will be simple and easy to navigate for anyone.

The user will download the app and install it on their phone/smart device. They will be able to register and login into the application. A users account will contain details about them, including their name and email address. Once logged in they will be able to search the nearby routes and will be able to select the route that they desire. Once selected the user will have the option to book a ticket on this route, if they have a discount code, they will be able to use here and get money off the ticket. After booking the ticket they will be able to view this ticket and will be able to show it to the ticket inspector if they get asked.

The admin and user will be able to change their account information. If in case the admin or user forgets their password, they will have the option to change password.

Administrators of the app will have access to the same app as users, but we'll have more abilities then that of the user. If needed, Admins will have the ability to add or remove routes and be able to modify existing routes.

# Software Lifecycle

We decided on an agile approach for our project as we were most familiar with this software development cycle. However, it’s still important to discuss how we came to this decision of choosing agile over the waterfall software cycle.

## Waterfall Model

“The Waterfall SDLC model is a sequential software development process in which progress is regarded as flowing increasingly downwards (similar to a waterfall) through a list of phases that must be executed in order to successfully build a computer software”. (Bassil 2012)

The major issue we had when discussing the waterfall model was the structure and sequence of steps that must be taken. We can’t move onto the next step without the previous step being complete as well as not being able to move back a step if required. It seemed to be a poor choice to make when we all used agile methodologies on placement, and we didn’t feel we would succeed in adapting a new software lifecycle in a semester. We also assumed that there would be hiccups and changes need throughout the project and we would not get everything perfect in one iteration.

## Agile

The term agile stands for “moving quickly” (Balaji and Murugaiyan 2012). We liked the idea of having deliverables through the semester that we could discuss in our weekly scrum meetings. One of the main advantages of using an agile approach is the “ability to respond to the changing requirements of the project” (Balaji and Murugaiyan 2012). This is a bonus over waterfall as we could make changes at any point throughout the project and incorporate them.

Taking an iterative approach to our work, we could achieve our goals each week in terms of deliverables. Each build is an increment of the last in terms of features, the final build holds all the features required from the start.

With an agile approach, there was no confusion when one part was going to be completed and the next started, having weekly scrum meetings and goals meant there is no guesswork between each team member as there is continuous communication and input from everywhere.

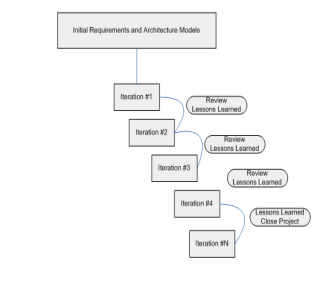


Figure 1: Agile Model Life Cycle (Balaji and Murugaiyan 2012)

# Project plan

|  |  |  |  |
| --- | --- | --- | --- |
| Deliverable | Description | Responsibility | Week |
| Business Scenario | * Concept | Darragh | 4 |
| Software Lifecycle | * Introduction * Agile | Eoghan | 4 |
| Project Plan | * Project Plan Table | Pawel | 4 |
| Established Roles | * Role Allocation Table | All | 3 |
| Requirements | * Use case Diagrams * Structured Use Case Descriptions * Detailed Use Case Descriptions * Non-functional requirements * Quality Attributes * GUI Prototypes | Darragh  Darragh  Darragh  Damian  Eoghan  Pawel | 5or6 |
| System Architecture | * Package Diagram * Decision Details | Damian | 6 |
| System Analysis | * Candidate Classes * UML Analysis Diagram * Communication(Sequence) Diagram | Damian | 6 |
| Tabular Class Listing | * Code Breakdown Table | All | 10 |
| Code | * Code Snippets * Model View/Architecture * Design Patterns * GUI/UI Screenshots and Explanation * Version Control * JUnit Testing * Code Style and Standards * Database Support * Functions is Used(e.g. Lambda) | All | 10 |
| Added Value | * GitHub * Database (SQL) | Eoghan/Damian | 11 |
| Recovered Architecture and Design Blueprints | * Architectural Diagram * Design Time Class Diagram * State Chart Diagram * Screen Correlation | Pawel | 11 |
| Critique | * Architecture * Design Patterns * Language * Framework | Eoghan | 11 |
| References | * List of Readings | All | 12 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Role** | **Description** | **Designated Team Member** |
| 1 | Project Manager | Sets up group meetings, gets agreement on the project plan, and tracks progress. | Pawel Ostach |
| 2 | Documentation Manager | Responsible for sourcing relevant supporting documentation from each team member and composing it in the report. | Eoghan Russell |
| 3 | Business Analyst / Requirements Engineer | Responsible for section 6 - Requirements. | Darragh Kelly |
| 4 | Architect | Defines system architecture | Damian Skrzypek |
| 5 | Systems Analysts | Creates conceptual class model | Damian Skrzypek |
| 6 | Designer | Responsible for recovering design time blueprints from implementation. | Pawel Ostach |
| 7 | Technical Lead | Leads the implementation effort | Darragh Kelly |
| 8 | Programmers | Each team member to develop at least 1 package in the architecture | ALL |
| 9 | Tester | Coding of automated test cases | Eoghan Russell |

# Requirements

## Functional

### Use Case Diagrams

Diagram

Description automatically generated

Diagram

Description automatically generated

### Use Case Descriptions

Use case 1 - Register

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User enters credentials and clicks register | 1. System ensures username is unique |
|  | 1. System creates new user account |

**Alternative Course:**

* Username is already taken
* System will display an error message

Use case 2 - Login

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User enters their credentials and selects the login button | 1. System checks username and password is correct |
|  | 1. User is then logged in |

**Alternative Course:**

* User enters wrong credentials
* System will display an error message

Use case 3 - Search routes

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User select the search route option from the menu. | 1. System displays the routes that are available |
| 1. User selects the route they want to choose | 1. System will display the information for the specific route |

**Alternative Course:**

* System is unable to find a route to the destination that the user wants to travel to.

Use case 4 - Book ticket

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User selects the route they want to choose | 1. Route is selected |
| 1. User selects the book option under the route | 1. System displays a confirmation message |

**Alternative Course:**

* User tries to book a ticket when the capacity of vehicle is full.
* System shows an error message

Use case 5 - Apply discount

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User enters in discount during booking | 1. System checks if valid discount |
|  | 1. System applies the discount to the price of booking |

**Alternative Course:**

* Discount code is no longer viable.
* System shows an error message

Use case 6 - Reset password

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User select into ‘Account’ option | 1. System brings them to the Account section |
| 1. User selects the ‘Change Password’ button | 1. System redirects to a change password screen |
| 1. User enters a new password and selects confirm button | 1. Passwords get changed |

**Alternative Course:**

* User tries to change password to the same password as they currently have.
* System shows an error message

Use case 7 - Change account information

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User select into ‘Account’ option | 1. System brings them to the Account section |
| 1. User selects the change account information | 1. System changes the account information |

**Alternative Course:**

* User does not enter all mandatory information fields and tries to save
* An error message will appear.

Use case 8 - Cancel ticket

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. User selects cancel ticket option in menu | 1. System shows the valid tickets that can be cancelled |
| 1. User confirms the ticket which they want to cancel and selects ‘Confirm’ | 1. System cancels the ticket |

**Alternative Course:**

* User tries to cancel ticket during transit.
* System shows an error message

Use case 9 - Add route

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. Admin selects the option to add new route | 1. New route option screen shows |
| 1. Admin enters in details about new route and selects add | 1. System add the new route as an option |

**Alternative Course:**

* The route already is at max route capacity
* System shows an error message

Use case 10 - Remove route

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. Admin selects the option to remove route | 1. Remove route option screen shows |
| 1. Admin confirms that the route is being removed | 1. System removes route from the options |

**Alternative Course:**

* Admin tries to remove route that is currently active
* System shows an error message

Use case 11- Modify route

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. Admin selects the option to modify route | 1. Modify route screen shows |
| 1. Admin changes information about the route | 1. System changes the information about route |

**Alternative Course:**

* Admin tries to modify route that is currently active
* System shows an error message

Use case 12 – Add Discount

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. Admin selects the option to add new Discount | 1. Add Discount screen appears |
| 1. Admin add new information about the discount e.g. how much, the code etc. | 1. System add the new discount code. |

**Alternative Course:**

The discount code already exists.

System shows an error.

Use case 13 – Remove Discount

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. Admin selects the option to remove Discount | 1. Remove Discount screen appears |
| 1. Admin selects the remove option on the discount in which they want to remove | 1. System removes the discount. |

**Alternative Course:**

The discount code does not exist.

System shows an error.

## Non-Functional Requirements

### Technical:

* Use Android studio for the frontend
* Must be developed in java and use AWS RDS for data base
* Spring for webservice

### Security:

* any private information about the user should be encrypted and user should be aware of what is being saved.
* password should not be visible on the screen and should be encrypted.
* User accounts should not be able to access any resources that they are not authorised to do e.g. resources meant for admins.

### Usability:

* UI should be easy to use and should not require much explanation for the user to understand what each button, textbox etc. does and how to interact with them.
* Each action that a user takes should provide feedback to the user, this will ensure the app feels responsive and allows users to quickly understand what the software is doing
* Searching for routes should be easy to do i.e. select from and to stations, date and with one click/tap the user should be able to view the results and select the one they want to buy.

### Reliability & Performance:

* Searching for different routes should be quick and should return correct results as this is the main part of the system, responses should not exceed 1s.
* Buying tickets should also be quick but making sure that the correct one and correct quantity is top priority.
* If the system is slow and/or results in incorrect behaviour/result the user is less likely to use it.

## Quality Attributes

### Security

The use of the Java programming language adds to the security of our application. Upon compilation, the source code written in java gets compiled into bytecode, which is later interpreted by the Java virtual machine. “Bytecode is resistant to tampering by external agents” (*Why Is Java Preferred to Other Languages as a Building Block?*).

We will provide security of our user’s private information through encryption. We used sha256 encoding to encode the password (one-way hashing) and never return the password back. This means that we generate a fingerprint of the input, but there is no way to get back to the original input.

The user’s will not be able to access any unauthorised information, this will be hidden to users unless they are an admin.

### Usability

The use of the android platform for our booking application contributes to the usability of the system. Android helps us implement the BYOD (Bring your Own Device) approach with ease, as android is installed on various devices. We will also utilise many android features that offer usability, the first being fragments. These are UI layouts, which can be seen as a modular section of an activity, having its own lifecycle, receiving its own events, and you can add or remove while the activity is running.

We will provide feedback to the user by using the inbuilt functionality of toasts in android. These are small popups that provide simple feedback when required.

### Reliability & Performance

Through the use of android functionality such as fragments, we will see an increase in performance. The lifecycle of an activity is intense, whereas fragments remain in the background until called. Therefore, saving performance when they are not required.

The use of the java programming language adds to the reliability of our application. “Java supports reliable exception handling that can withstand all the major types of erroneous and exception conditions without breaking the system” (*Why Is Java Preferred to Other Languages as a Building Block?*). This means that our application is less likely to crash without giving us an error.

The use of an AWS (Amazon Web Service) database adds to the reliability of our application. The database has automated features that make it reliable, crash recovery being one of those. This means that it’s designed to recover instantaneously and continue to serve our application.

## GUI Prototypes

Text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

# System Architecture

## Package Diagram

Chart, diagram

Description automatically generated

## Model View Controller(MVC) Discussion

Our system follows model-view-controller(MVC) design pattern by splitting into three layers, a user-interface(UI) layer, a business layer and a data layer.

UI is how the user interacts with our system, then the data from both the UI and the data layer is processed in the business layer.

This approach decouples the different parts of the system providing great flexibility when it comes to changing any of the components, for example if we are required to change the way our data is stored we can easily replace the implementation in data layer to the updated storage solution without breaking any functionality in the business or UI layers.

Another great advantage to using MVC and splitting the system like this is the ease of parallel development as different people can work on different parts without having dependencies on each other, of course for the system to fully work all parts need to be finished.

## Android Studio:

Android studio is the main software used in developing android apps, it provides great support for creating apps for different types of phones and versions of android. It has built in layout editor for creating GUI allowing to create nice looking and responsive apps.  
The system will be developed using Java as it is one of the main languages when developing apps for Android but also Java is an object-oriented language which will fit very well into the planned design/architecture.

## AWS database & hosting the Spring backend:

Amazon web services provides an easy to use and secure environment for developing different array of applications. Thanks to its flexibility we are able to select the appropriate parameters for this project i.e. java, MySQL database and an Linux machine for hosting the Spring webservice backend for our app. Final great advantage to mention is the fact that AWS provides a lot of choices when it comes to pricing, allowing to start of free or with very limited investment and buy more space/computational power as the software’s userbase and scope expands.

## JUnit:

For testing we will use JUnit as it is a widely adopted, efficient testing framework becoming a standard around the world when it comes to writing tests for Java programs, and it is supported by most IDE’s including Android studio.

# System Analysis

## Candidate Classes

Initially to create a class list we have looked at nouns in the functional requirements and use cases this gave us a good starting point for the possible classes required, after more in depth look we have noticed that some classes could be reduced as same functionality can be covered by one class e.g. we don’t need separate class for an Admin and can be covered by just having user type in the User class.

## Initial identified class list:

User (class)

~~Admin~~ (represented by usertype attribute)

~~Customer~~ (represented by usertype attribute)

~~Username~~ (attribute)

~~Password~~ (attribute)

~~Register~~(event/operation)

~~Login~~(event/operation)

~~Book~~(event/operation)

Booking(class)

Discount(class)

~~Account information~~ (attributes)

~~Ticket~~(same as booking)

Route(class)

~~Add~~(event/operation)

~~Remove~~(event/operation)

~~Modify~~(event/operation)

~~Cancel~~(event/operation)

~~Payment~~(became transaction class)

Further Break Down:

Entity classes(User, Booking, Discount, Route, Transaction)

Interface Services for business logic(UserService, RouteService, DiscountService, BookingService)

Interface Repository for retrieving data from webservice(UserRepository, RouteRepository, DiscountRepository, BookingRepository, PaymentRepository)

## UML Class Diagram

Diagram, schematic

Description automatically generated

## Sequence & Communication Diagrams

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

## Entity Relation Diagram

Diagram

Description automatically generated

# Tabular Class Listing

## Spring Boot

|  |  |  |  |
| --- | --- | --- | --- |
| Package | Class | Author | Lines of Code |
| com.cs4125.bookingapp.controllers | BookingController | Damian/Eoghan | 60 |
| com.cs4125.bookingapp.controllers | DiscountController | Damian/Eoghan | 67 |
| com.cs4125.bookingapp.controllers | RouteController | Damian/Eoghan | 71 |
| com.cs4125.bookingapp.controllers | UserController | Damian/Eoghan | 30 |
| com.cs4125.bookingapp.model.entities | Booking | Damian/Eoghan | 98 |
| com.cs4125.bookingapp.model.entities | Discount | Damian/Eoghan | 67 |
| com.cs4125.bookingapp.model.entities | Route | Damian/Eoghan | 76 |
| com.cs4125.bookingapp.model.entities | TransactionRecord | Damian/Eoghan | 65 |
| com.cs4125.bookingapp.model.entities | User | Damian/Eoghan | 75 |
| com.cs4125.bookingapp.model.repositories | BookingRepository | Damian/Eoghan | 14 |
| com.cs4125.bookingapp.model.repositories | DiscountRepository | Damian/Eoghan | 12 |
| com.cs4125.bookingapp.model.repositories | RouteRepository | Damian/Eoghan | 16 |
| com.cs4125.bookingapp.model.repositories | TransactionRepository | Damian/Eoghan | 12 |
| com.cs4125.bookingapp.model.repositories | UserRepository | Damian/Eoghan | 14 |
| com.cs4125.bookingapp.model | ConcreteRouteFactory | Damian | 21 |
| com.cs4125.bookingapp.model | ConcreteUserFactory | Damian | 24 |
| com.cs4125.bookingapp.model | RouteFactory | Damian | 11 |
| com.cs4125.bookingapp.model | UserFactory | Damian | 10 |
| com.cs4125.bookingapp.services | BookingService | Damian | 15 |
| com.cs4125.bookingapp.services | BookingServiceImpl | Damian | 204 |
| com.cs4125.bookingapp.services | DiscountService | Damian | 16 |
| com.cs4125.bookingapp.services | DiscountServiceImpl | Damian | 136 |
| com.cs4125.bookingapp.services | EncryptionService | Damian | 9 |
| com.cs4125.bookingapp.services | EncryptionServiceImpl | Damian | 16 |
| com.cs4125.bookingapp.services | RouteService | Damian | 17 |
| com.cs4125.bookingapp.services | RouteServiceImpl | Damian | 141 |
| com.cs4125.bookingapp.services | TransactionContext | Damian | 57 |
| com.cs4125.bookingapp.services | TransactionRecordCancelledState | Damian | 22 |
| com.cs4125.bookingapp.services | TransactionRecordCompletedState | Damian | 30 |
| com.cs4125.bookingapp.services | TransactionRecordInitialState | Damian | 31 |
| com.cs4125.bookingapp.services | TransactionRecordProgressState | Damian | 35 |
| com.cs4125.bookingapp.services | TransactionRecordState | Damian | 12 |
| com.cs4125.bookingapp.services | UserService | Damian | 10 |
| com.cs4125.bookingapp.services | UserServiceImpl | Damian | 58 |
| com.cs4125.bookingapp | BasicController | Eoghan | 138 |
| com.cs4125.bookingapp | BookingappApplication | Eoghan | 13 |
| com.cs4125.bookingapp.controller | UserTest | Eoghan | 53 |
| com.cs4125.bookingapp | BookingappApplicationTests | Eoghan | 13 |

## Android Application

## Total Code Developed

## Team Member Contribution

|  |  |
| --- | --- |
| Team Member | Lines Contributed |
| Eoghan Russell | 10,311 |
| Damian Skrzypek |  |
| Pawel Ostach |  |
| Darragh Kelly |  |

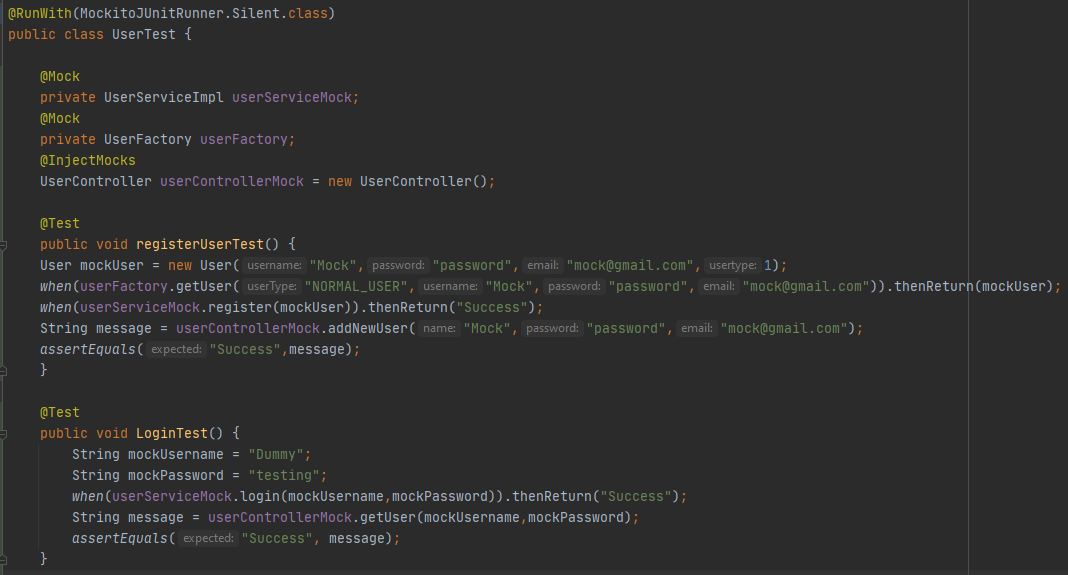
Note: Lines taken from GitHub adjusted due to .idea files committed

# Code

## Junit Testing

We decided on Junit for the testing of our application. We used Mockito along with Junit, this allowed us to create mock objects and define the output of certain method calls.

The @Mock variables are mock objects created for our unit tests. The @InjectMocks variable is the mock class that utilises the mock objects upon creation. This allows for easy expandability as nothing is hard coded and the behaviour is defined at run-time. @Test defines a unit test.



This Junit and Mockito unit testing implementation allow for easy automation and expandability. It could simply be set up to run nightly on the automation server Jenkins, testing our applications functionality to pick up on bugs if anything changes.

## GitHub – Version Control

<https://github.com/Eoghan1232/CS4125_Project1_TeamBased>

GitHub was used as a version control system for our project throughout the implementation stage. Once someone was finished or had started part of their implementation, they committed to the master branch. This helped us keep track with everyone’s progress throughout the implementation. If something was broken or wrong, we could simply revert the specific commit.

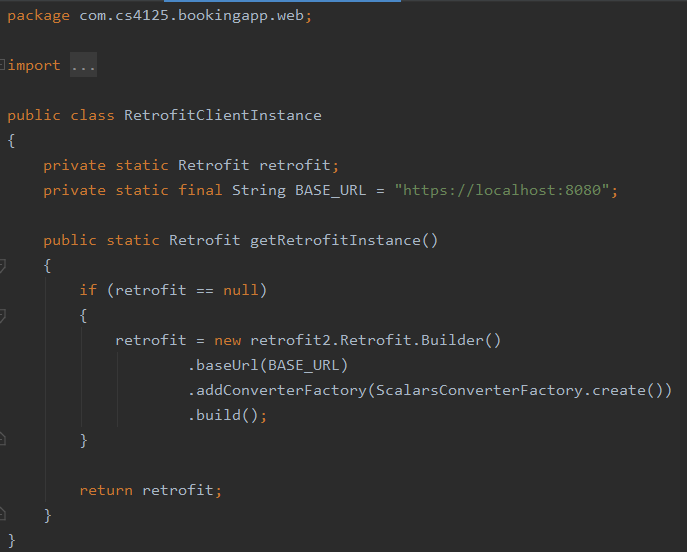
“””” NEED IMAGE OF CONTRIBUTIONS FROM GITHUB””””

# Added Value

## Concurrency

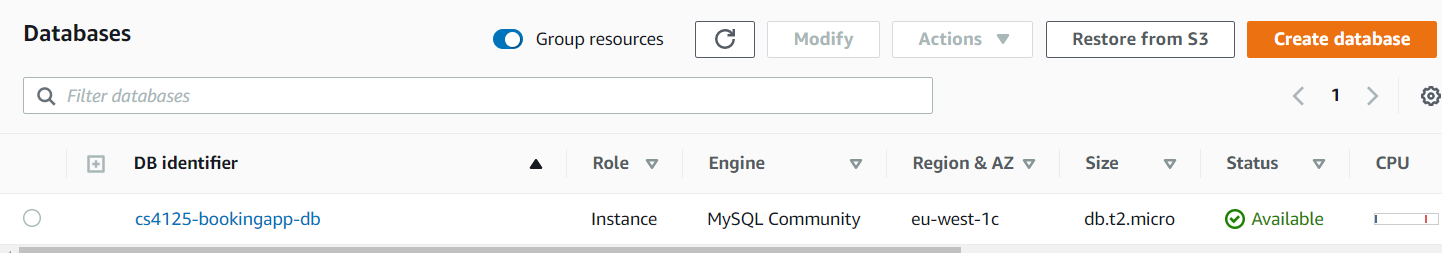
## REST architectural pattern

For our REST architectural implementation, we integrated the Retrofit API into our android application. Retrofit is a type-safe REST client, allowing us to send network requests. This connected our android application with our Spring Boot application (REST based webservice). This turned the REST API into a java interface for us to use. Retrofit uses the webservice interface we created to build the corresponding requests required.



## AWS Database

For our database, we used AWS RDS (Relational Database Service) for our project. This allowed us to easily setup a database that can be accessed globally. We opted for AWS’s free tier and created an instance of our database.



This database is in SQL, we could check the database through SQL Workbench to check the contents and tables. Spring Boot webservices was used to connect to the database, then we use Retrofit API to connect our android application to the AWS database.

## Sprint Boot Deployment - Damian

# Recovered Architecture and Design Blueprints

# Critique

## Overview

Overall, our implementation was consistent with our analysis sketches set out at the start of the project. During the implementation period, packages were added that weren’t originally planned or evident in the design phase. Our initial designs were based on the initial plan set out after the first few weeks after the project specifications were released. Naturally, this was built upon and changed during the implementation stage. The differences between the analysis sketch versus the blueprint sketches outlined below illustrate the changes that were made.

## Changes between Analysis Sketches & Blueprint

“”Compare and contrast the analysis sketches versus the blueprints afterwards””

# References

Balaji, S. and Murugaiyan, M.S. (2012) 'Waterfall vs. V-Model vs. Agile: A comparative study on SDLC', *International Journal of Information Technology and Business Management*, 2(1), 26-30.

Bassil, Y. (2012) 'A simulation model for the waterfall software development life cycle', *arXiv preprint arXiv:1205.6904*.

*Why Is Java Preferred to Other Languages as a Building Block?* , Techopedia.com, available: <https://www.techopedia.com/2/28705/development/programming-languages/why-is-java-preferred-to-other-languages-in-building-technological-blocks> [accessed 6th November].