DALHOUSIE UNIVERSITY

CSCI 5308

QUALITY ASSURANCE

PROJECT REPORT

E – RAIL

Submitted By:

Dhruvi Shah (B00812683)

Samarth Raval (B00812673)

Varun Mahagaokar (B00826634)

TABLE OF CONTENTS

FIGURE 3
INTRODUCTION 4
E-RAIL'S FEATURES 4
TECHNOLOGIES USED 12
FOLDER STRUCTURE 12
NAMING CONVENTION 13
DESIGN PATTERN IMPLEMENTATION 14
SEPARATION OF LAYERS 16
TESTING OF E-RAIL 16
REFACTORING 17
TECHNICAL DEBT 18
MEMBER'S CONTRIBUTION 18
REFERENCES: 23

FIGURE

Figure 1: Station Management....4

Figure 2: Train Management 5

Figure 3: Assign Stations....5

Figure 4: Class Management 6

Figure 5: Report Generation 6

Figure 6: User Management 7

Figure 7: User/Admin Login 8

Figure 8: Forget Username 9

Figure 9: Forget Password....9

Figure 10: Reset Password....10

Figure 11: Tickets Booking 11

Figure 12: Booking History....11

Figure 13: Folder Structure of eRail....13

Figure 14: Spring Boot REST API Architecture 16

Figure 15: JUnit TestCases....17

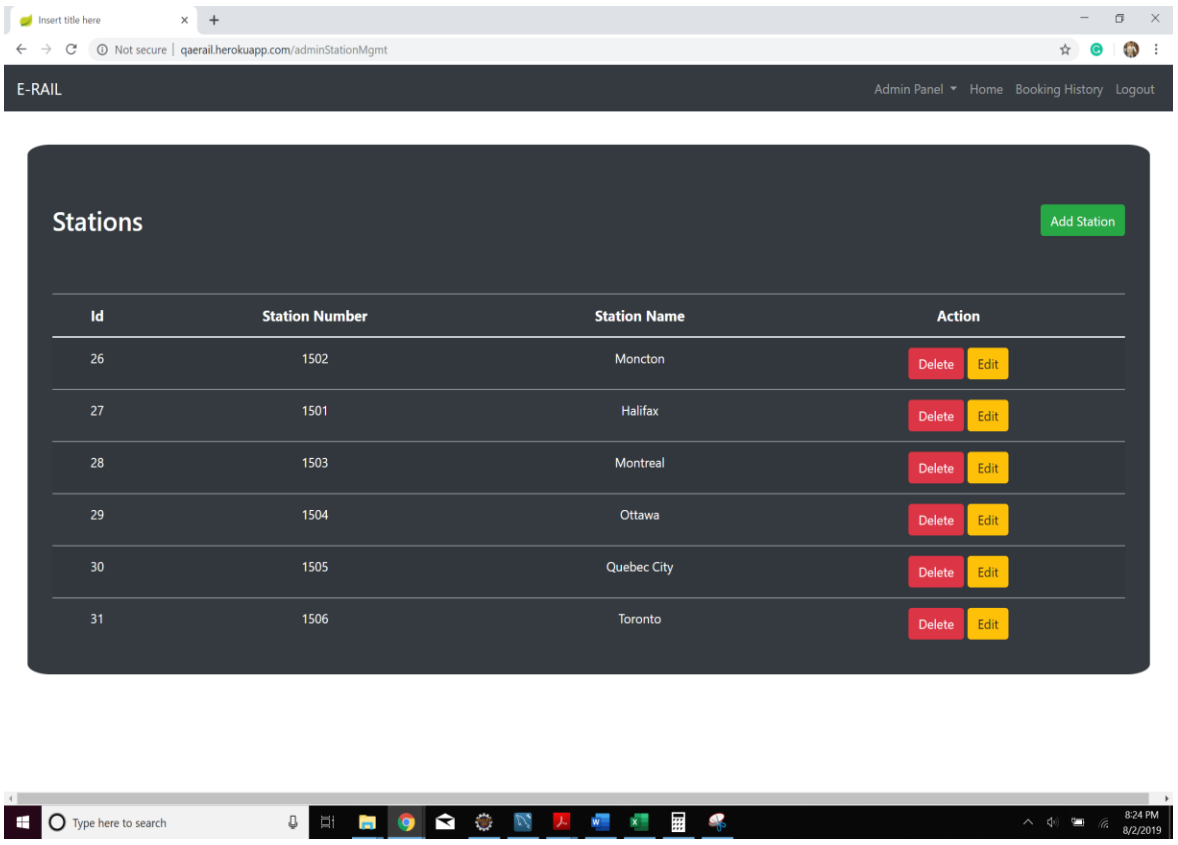
INTRODUCTION

ERail-Ticketing is a web application portal to book online tickets for train. It is one stop destination where there are two user personas- Admin, Passengers-Users. Passengers need to register with a valid email id and phone Number, Username and Password. Using the credentials passenger can login and book tickets. Passenger can search train by Source and destination or by Train Number and then can book tickets. Once ticket is booked, they can view it in booking history. Passengers can see the estimated arrival, departure and current status of their booking. Admin will be the entity for controlling details. Admin will be able to update all types of details of trains, station, Train classes. Moreover, Admin can generate reports for the booking and can create new admin.

E-RAIL'S FEATURES

Admin

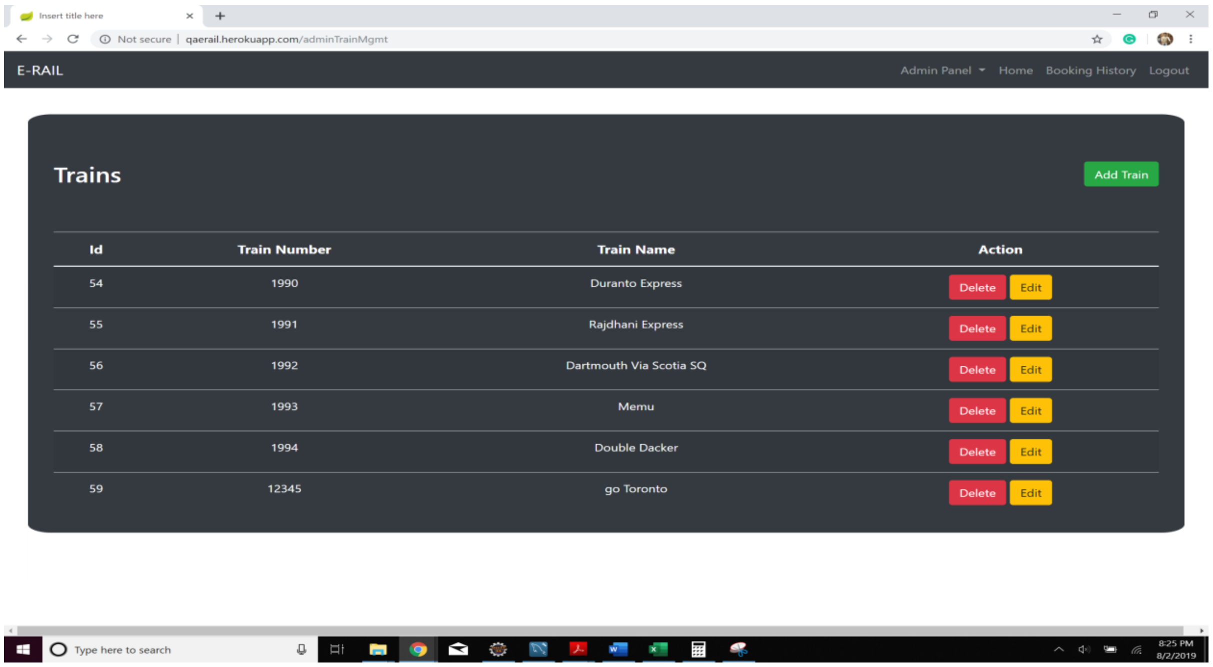
- <u>Update Train and station Data</u>



Stations

Id	Station Number	Station Name	Action
26	1502	Moncton	Delete Edit
27	1501	Halifax	Delete Edit
28	1503	Montreal	Delete Edit
29	1504	Ottawa	Delete Edit
30	1505	Quebec City	Delete Edit
31	1506	Toronto	Delete Edit

Figure 1: Station Management

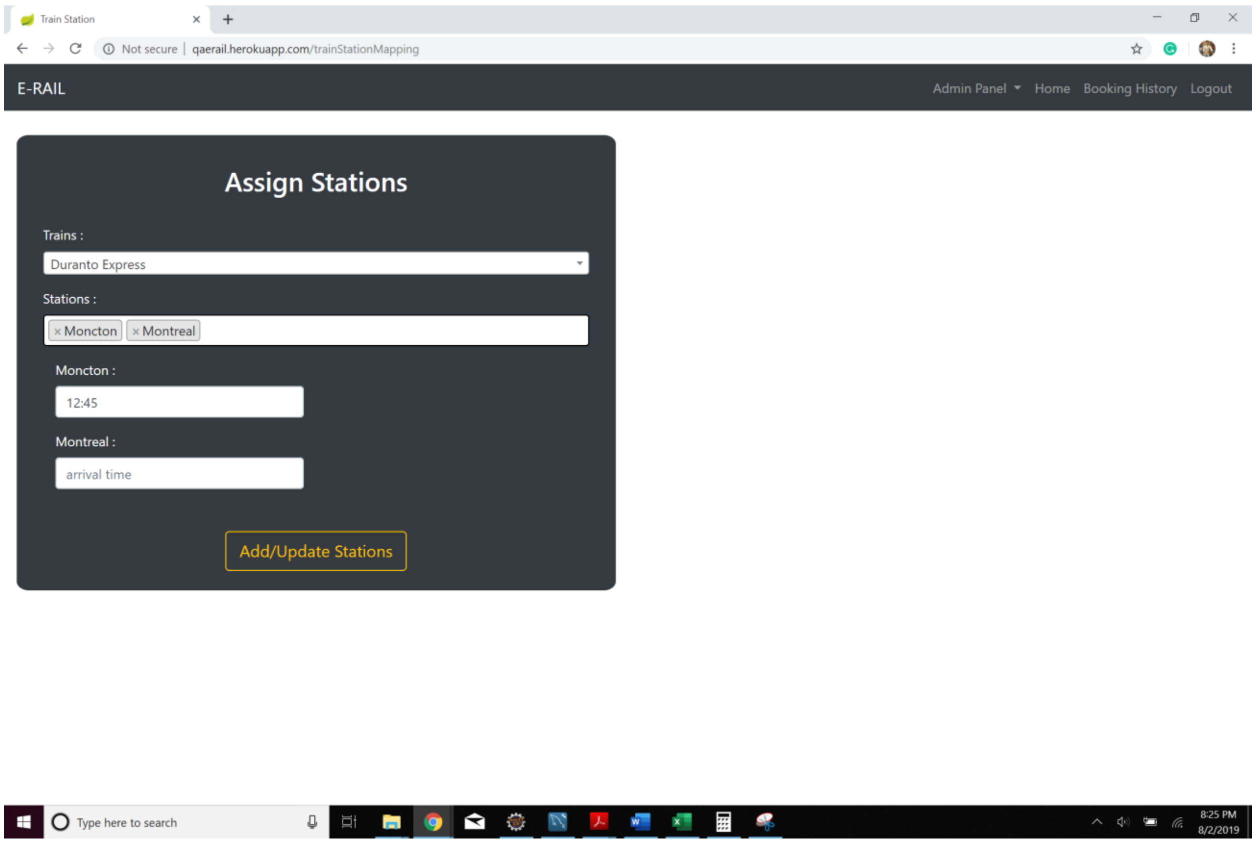


E-RAIL

Trains

Id	Train Number	Train Name	Action
54	1990	Duranto Express	Delete Edit
55	1991	Rajdhani Express	Delete Edit
56	1992	Dartmouth Via Scotia SQ	Delete Edit
57	1993	Memu	Delete Edit
58	1994	Double Dacker	Delete Edit
59	12345	go Toronto	Delete Edit

Figure 2: Train Management



E-RAIL

Assign Stations

Trains : Duranto Express

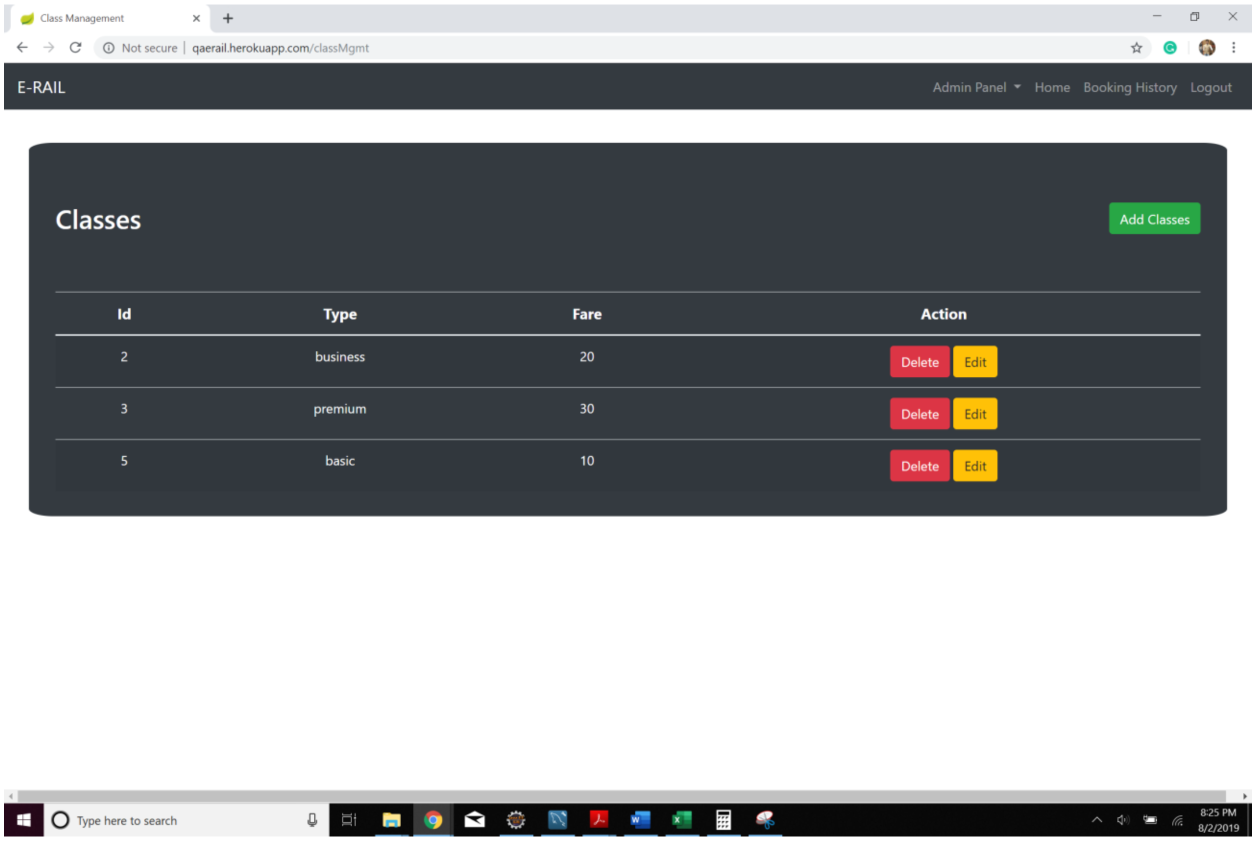
Stations : × Moncton × Montreal

Moncton : 12:45

Montreal : arrival time

Add/Update Stations

Figure 3: Assign Stations



Classes

Id	Type	Fare	Action
2	business	20	Delete Edit
3	premium	30	Delete Edit
5	basic	10	Delete Edit

Figure 4: Class Management

- Generate Reports

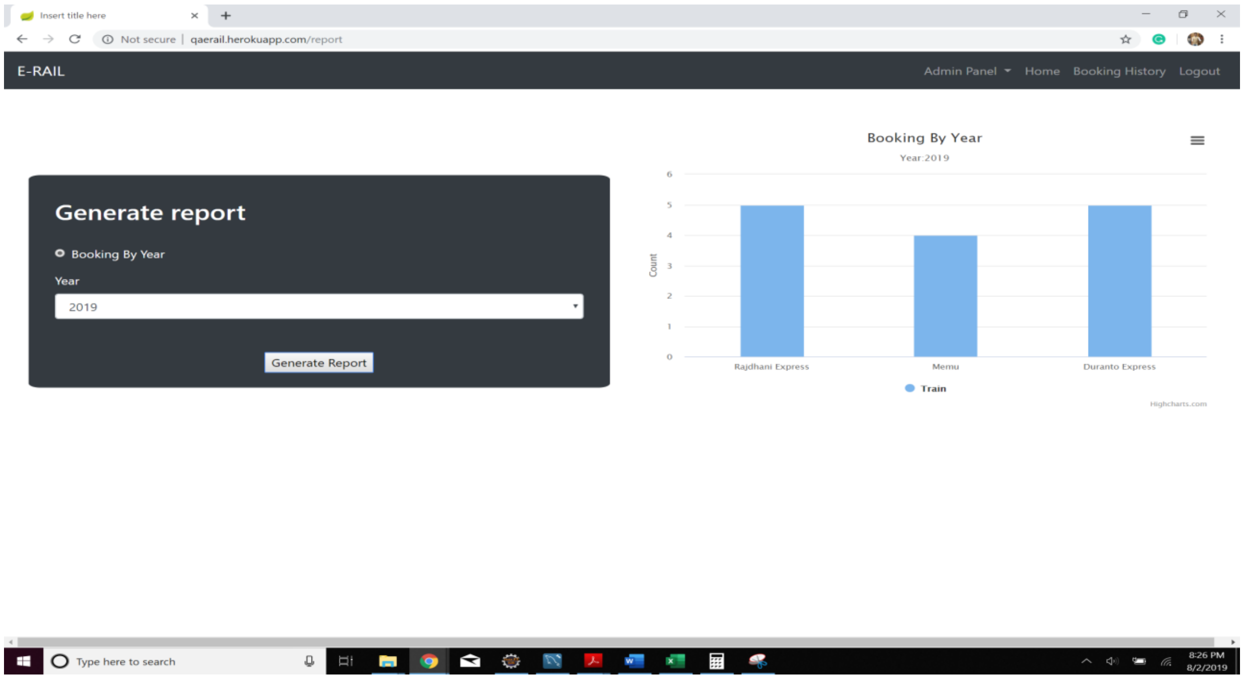
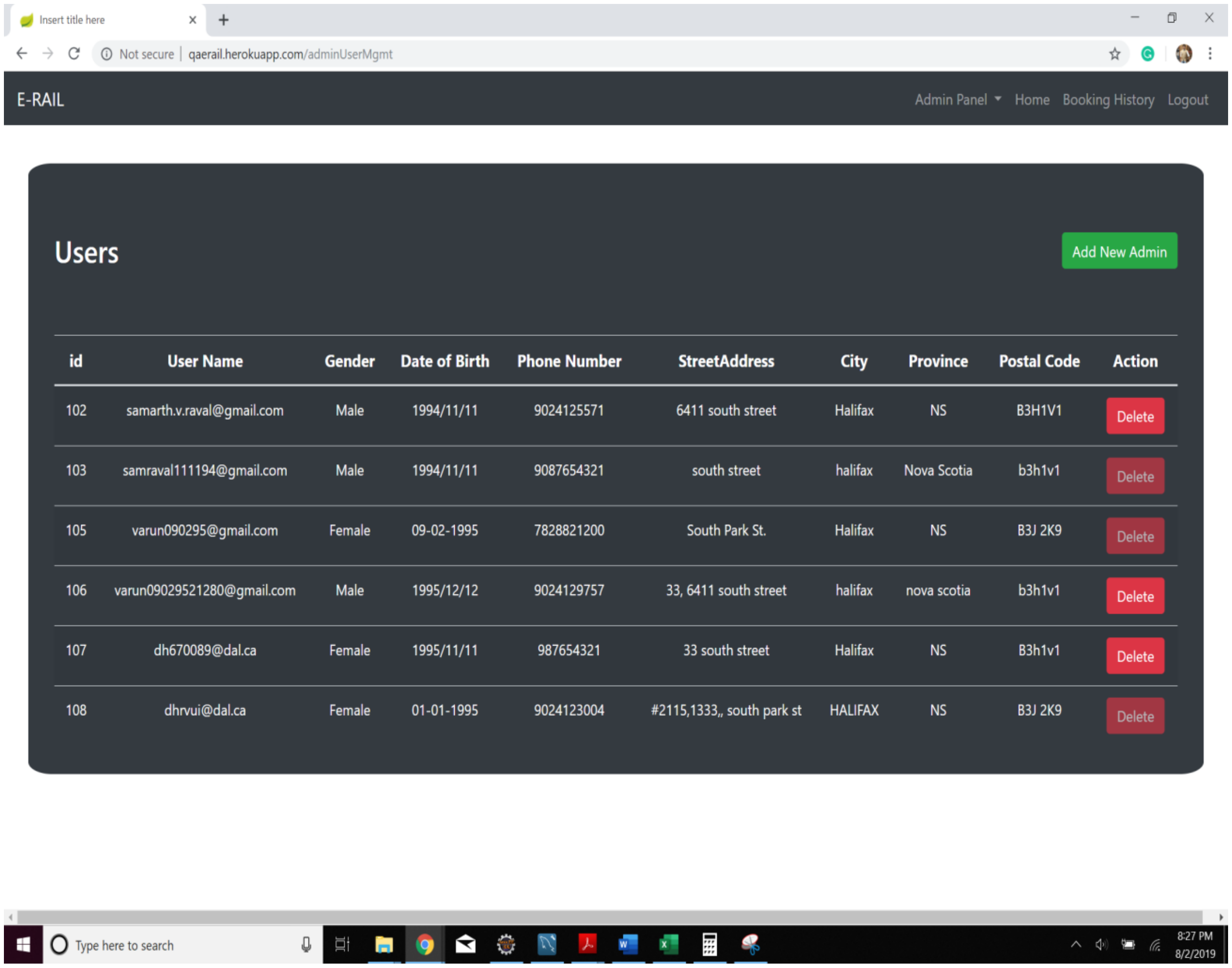


Figure 5: Report Generation

- <u>Manage Users</u>



Users

id	User Name	Gender	Date of Birth	Phone Number	StreetAddress	City	Province	Postal Code	Action
102	samarth.v.raval@gmail.com	Male	1994/11/11	9024125571	6411 south street	Halifax	NS	B3H1V1	Delete
103	samraval111194@gmail.com	Male	1994/11/11	9087654321	south street	halifax	Nova Scotia	b3h1v1	Delete
105	varun090295@gmail.com	Female	09-02-1995	7828821200	South Park St.	Halifax	NS	B3J 2K9	Delete
106	varun09029521280@gmail.com	Male	1995/12/12	9024129757	33, 6411 south street	halifax	nova scotia	b3h1v1	Delete
107	dh670089@dal.ca	Female	1995/11/11	987654321	33 south street	Halifax	NS	B3h1v1	Delete
108	dhrvui@dal.ca	Female	01-01-1995	9024123004	#2115,1333,, south park st	HALIFAX	NS	B3J 2K9	Delete

Figure 6: User Management

User

- <u>Profile Management</u>

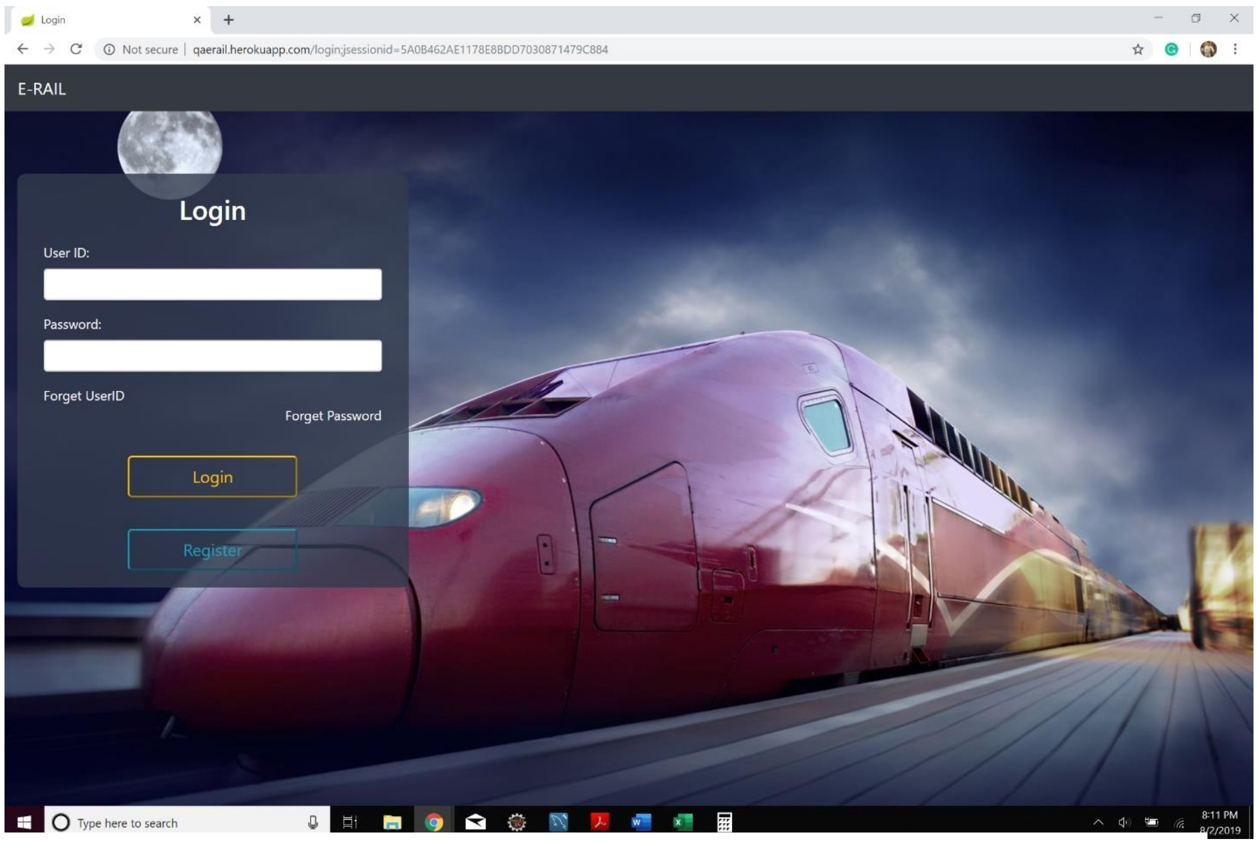


Figure 7: User/Admin Login

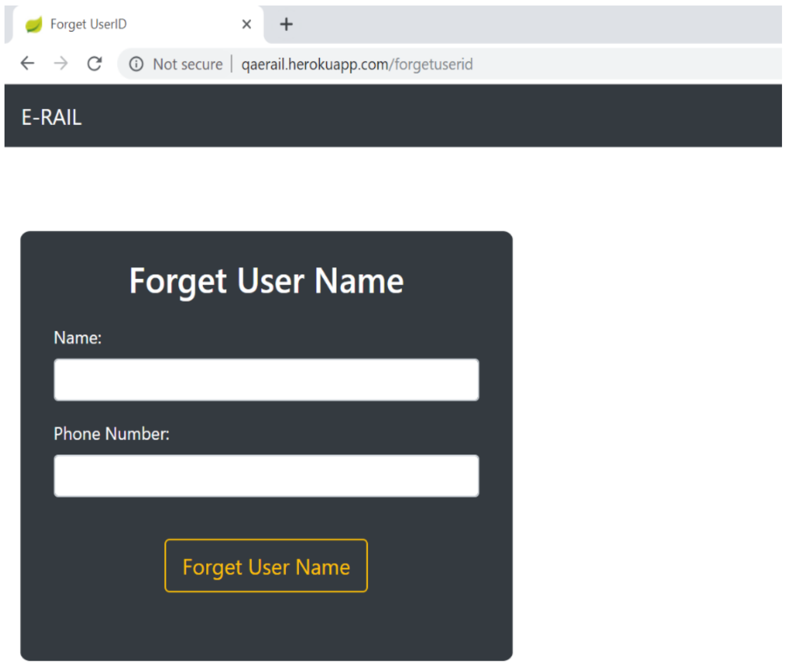


Figure 8: Forget Username

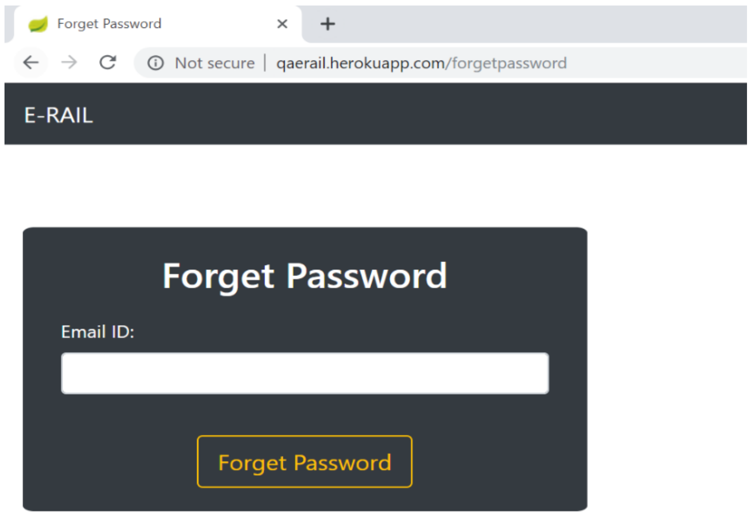
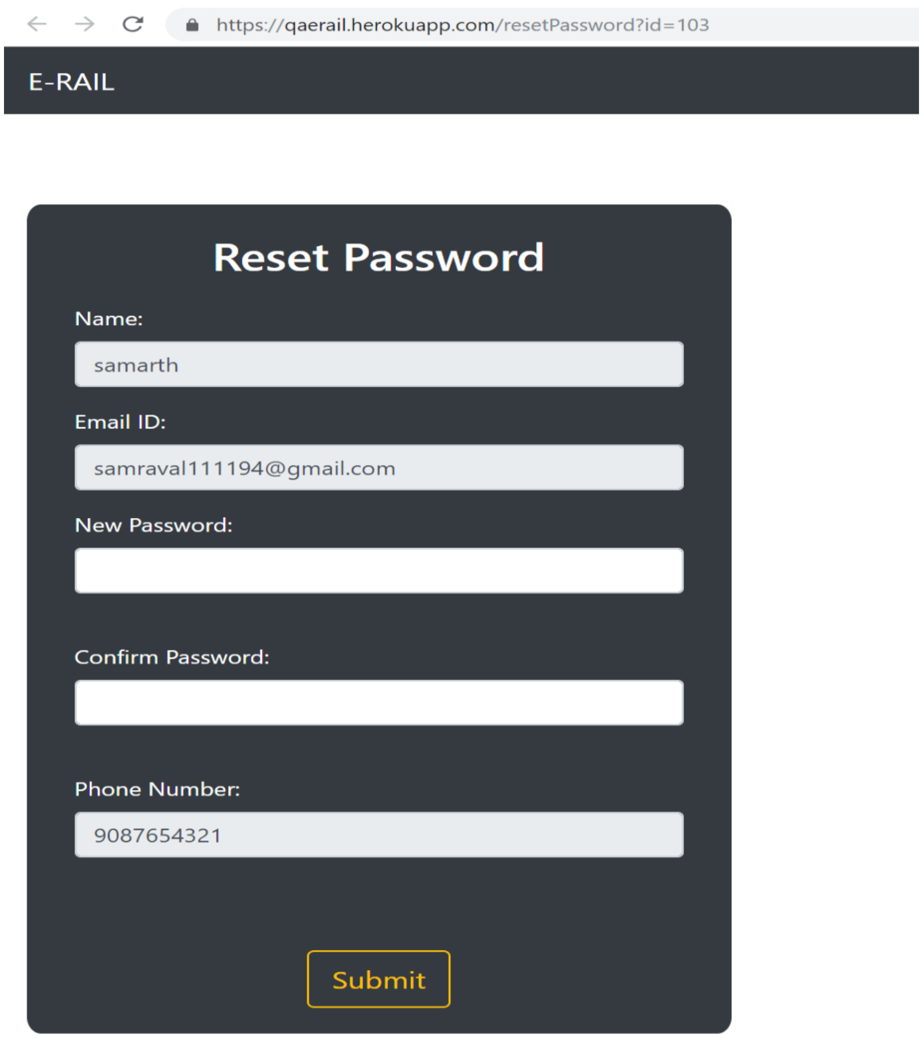


Figure 9: Forget Password



https://qaerail.herokuapp.com/resetPassword?id=103

E-RAIL

Reset Password

Name: samarth

Email ID: samraval111194@gmail.com

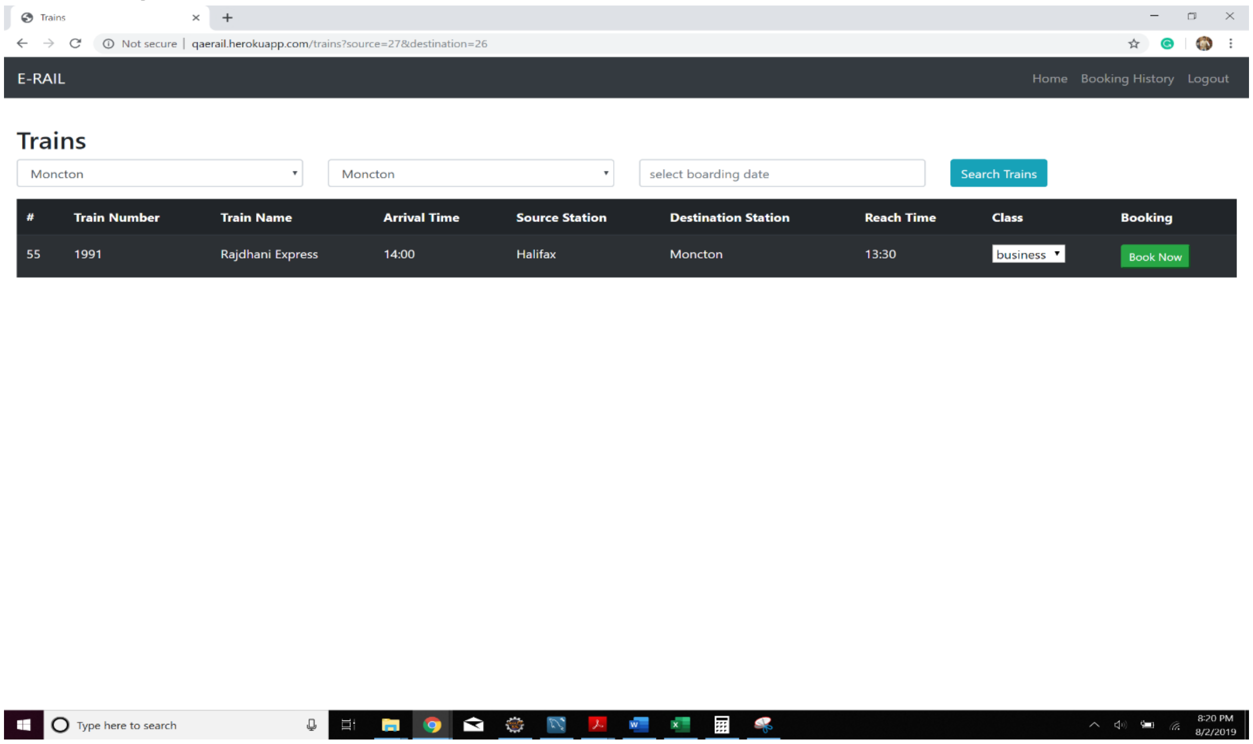
New Password:

Confirm Password:

Phone Number: 9087654321

Figure 10: Reset Password

- Booking Tickets



Trains

qaerail.herokuapp.com/trains?source=27&destination=26

E-RAIL — Home Booking History Logout

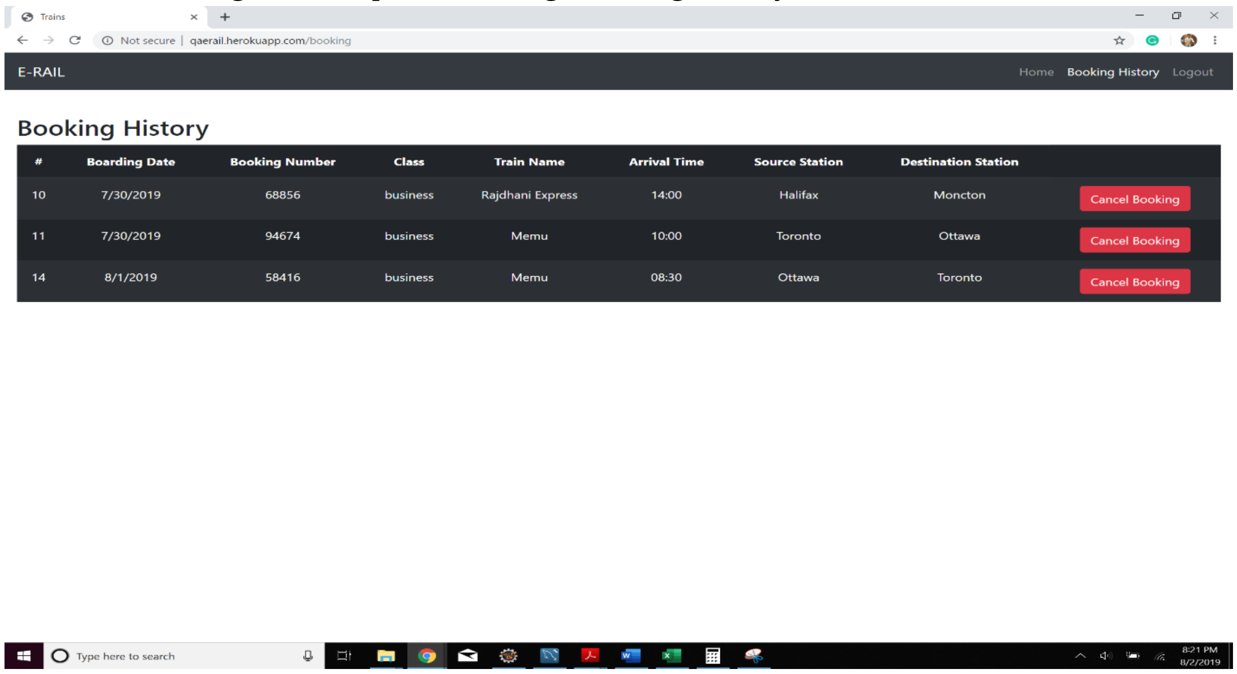
Trains

Moncton | Moncton | select boarding date | Search Trains

#	Train Number	Train Name	Arrival Time	Source Station	Destination Station	Reach Time	Class	Booking
55	1991	Rajdhani Express	14:00	Halifax	Moncton	13:30	business	Book Now

Figure 11: Tickets Booking

- Train and Booking Status, Update Booking, Booking History:



Trains

qaerail.herokuapp.com/booking

E-RAIL — Home Booking History Logout

Booking History

#	Boarding Date	Booking Number	Class	Train Name	Arrival Time	Source Station	Destination Station	
10	7/30/2019	68856	business	Rajdhani Express	14:00	Halifax	Moncton	Cancel Booking
11	7/30/2019	94674	business	Memu	10:00	Toronto	Ottawa	Cancel Booking
14	8/1/2019	58416	business	Memu	08:30	Ottawa	Toronto	Cancel Booking

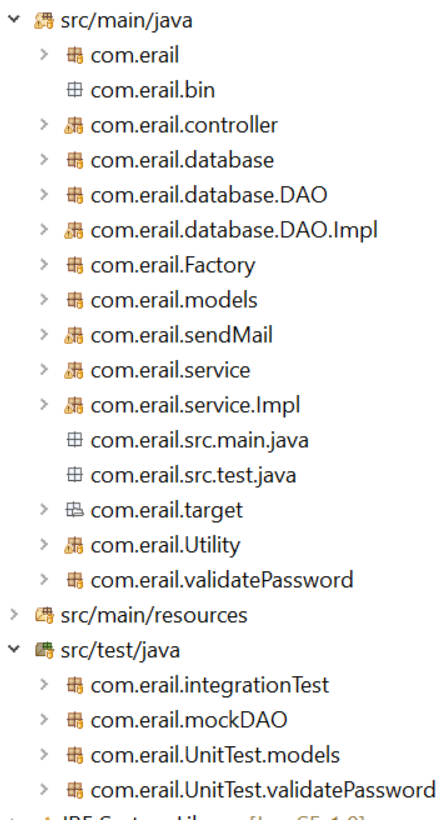
Figure 12: Booking History

TECHNOLOGIES USED

1. **Framework:**
   - Spring Boot
2. **Database:**
   - MySQL
3. **Server-Side Technology:**
   - Java
4. **Client-Side Technology:**
   - Java Server Pages
   - jQuery
   - Hyper Text Markup Language
   - Bootstrap
5. **Build Automation Tool:**
   - Maven
6. **Task Management Tool:**
   - Trello
7. **Version Control:**
   - GitHub
8. **Continuous Integration Tool:**
   - Jenkins
9. **Cloud Deployment Tool:**
   - Heroku

FOLDER STRUCTURE

It is very important when working in a team to follow a folder structure. We have followed as shown in below figure. All the functionality is separated by different packages. In addition, Interfaces are separated by inherited package for its Implementation. Test cases are also written in complement different package.



```
src/main/java
    com.erail
    com.erail.bin
    com.erail.controller
    com.erail.database
    com.erail.database.DAO
    com.erail.database.DAO.Impl
    com.erail.Factory
    com.erail.models
    com.erail.sendMail
    com.erail.service
    com.erail.service.Impl
    com.erail.src.main.java
    com.erail.src.test.java
    com.erail.target
    com.erail.Utility
    com.erail.validatePassword
src/main/resources
src/test/java
    com.erail.integrationTest
    com.erail.mockDAO
    com.erail.UnitTest.models
    com.erail.UnitTest.validatePassword
```

Figure 13: Folder Structure of eRail

NAMING CONVENTION

Naming Convention is an important part of the programming. It plays a crucial role as teamwork needs single format for the naming of the structures. It increases the readability of the code which allows users to understand code more quickly. For the naming convention, we have used “CamelCase” naming style for good code maintenance and readability of code. CamelCase are of two types. Uppercase where first letter starts with Uppercase and rest follows the same format – Example - HelloWorld. Lower Camel Case first letter of the word begins with lowercase and rest all with uppercase Example - HelloWorld.

For the spacing in our project, we have used tabs instead of space as tabs give more user readability and make the code look even. Following table shows the list of naming and coding conventions we have used in our project.

Identifier Type	**Rules for Naming**	**Example**
Packages	We have declared all our packages in lowercase Camel Case	com.erail.sendMail com.erail.service

Classes	We have declared all our classes in Upper Camel Case	class StationDAO
		class ClassTrainDAO

Interfaces	For interface, we followed with the naming convention Upper camel Case that starts with I	interface IService interface IServiceFactory

Methods	For methods we have followed with the Lower camel case structure	public Map<String,Object> createBooking ();
		public Map<String,Object> findStationByStationNumber ();

Variables	For declaration of variables we have used Lower Camel case	private IServiceFactory iServiceFactory;

DESIGN PATTERN IMPLEMENTATION

Factory Pattern:

Factory Design pattern is creational pattern which is related to the creation of the Object. In this pattern, we create the object in such a way that it won't expose the creation logic to the client. The client uses the same common interface to create new type of object. This pattern allows clients to create objects of a library in a way such that it won't have the tight coupling with the class libraries. We created a factory class named "DAOFactory.java" and "ServiceFactory.java" which implements IDAOFactory and IServiceFactory interface. [1]

Singleton Pattern

Singleton Design pattern also comes under the creational design pattern. This pattern involves a single class which is responsible to create an object which makes sure that only a single object gets created. This class gives us facility to access its only object which can be accessed directly without need to instantiate the object of the class. Singleton class has only one instance of the class at the time per JVM instance. [2]

We used singleton pattern to make sure that there is only one connection opened for database operation. To implement singleton pattern, we created “DatabaseConfig.java” which gives connection object using getConnection method.

Command Pattern

Command pattern is a data driven design pattern that comes under the behavioral pattern category. In command pattern object is used to encapsulate all the information needed to perform and action or trigger an event at a later stage. This pattern increases the productivity of the development team and quality of the software. This gives us opportunity to achieve complete decoupling between the sender and the receiver [3].

We have created an interface IRuleValidator which is acting as a command. We have concrete command classes Digits, Length, LowerCase, UpperCase and SpecialCharacter implementing IRuleValidator which will do actual command processing. A class ValidatePassword is created which acts as an invoker object.

ValidatePassword object uses command pattern to identify which object will execute which command based on the type of the command. UserController will use ValidatePassword class to demonstrate command pattern.

SEPARATION OF LAYERS

We have used Spring boot REST API architecture for separation of layers that is, presentation layer and business layer and data layer. It is used to divide application process.

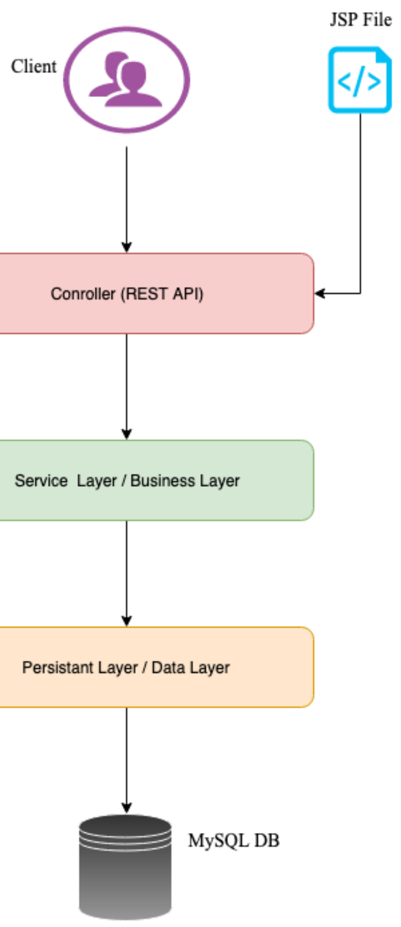
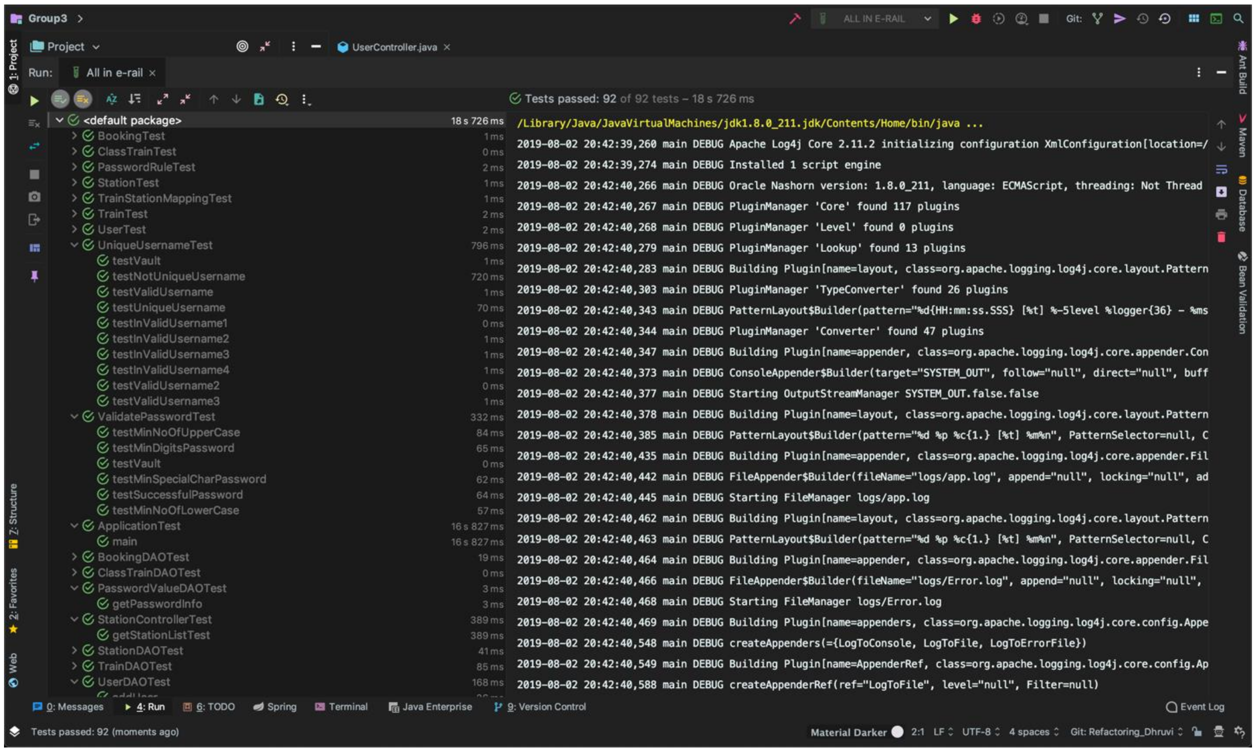


Figure 14: Spring Boot REST API Architecture

TESTING OF E-RAIL

Testing is one of the key phases of the software development cycle and the compulsory process needed to qualify the software as suitable for release. The testing guarantees that the software will work as per the expected execution. Testing helps to discover the issues in early stages of software development cycle which reduces development cost remarkably.

We have used Junit to write testcases for E-rail project. We have written 92 testes which includes logical functionality as well as testing of model class and even testing of DAO using mock DAO objects.



```
<default package>                      18 s 726 ms
  BookingTest                          1 ms
  ClassTrainTest                       0 ms
  PasswordRuleTest                     2 ms
  StationTest                          1 ms
  TrainStationMappingTest              1 ms
  TrainTest                            2 ms
  UserTest                             2 ms
  UniqueUsernameTest                   796 ms
    testVault                          1 ms
    testNotUniqueUsername              720 ms
    testValidUsername                  1 ms
    testUniqueUsername                 70 ms
    testInValidUsername1               0 ms
    testInValidUsername2               1 ms
    testInValidUsername3               1 ms
    testInValidUsername4               1 ms
    testValidUsername2                 0 ms
    testValidUsername3                 1 ms
  ValidatePasswordTest                 332 ms
    testMinNoOfUpperCase               84 ms
    testMinDigitsPassword              65 ms
    testVault                          0 ms
    testMinSpecialCharPassword         62 ms
    testSuccessfulPassword             64 ms
    testMinNoOfLowerCase               57 ms
  ApplicationTest                      16 s 827 ms
    main                               16 s 827 ms
  BookingDAOTest                       19 ms
  ClassTrainDAOTest                    0 ms
  PasswordValueDAOTest                 3 ms
    getPasswordInfo                    3 ms
  StationControllerTest                389 ms
    getStationListTest                 389 ms
  StationDAOTest                       41 ms
  TrainDAOTest                         85 ms
  UserDAOTest                          168 ms
```

```
Tests passed: 92 of 92 tests – 18 s 726 ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home/bin/java ...
2019-08-02 20:42:39,260 main DEBUG Apache Log4j Core 2.11.2 initializing configuration XmlConfiguration[location=/
2019-08-02 20:42:39,274 main DEBUG Installed 1 script engine
2019-08-02 20:42:40,266 main DEBUG Oracle Nashorn version: 1.8.0_211, language: ECMAScript, threading: Not Thread
2019-08-02 20:42:40,267 main DEBUG PluginManager 'Core' found 117 plugins
2019-08-02 20:42:40,268 main DEBUG PluginManager 'Level' found 0 plugins
2019-08-02 20:42:40,279 main DEBUG PluginManager 'Lookup' found 13 plugins
2019-08-02 20:42:40,283 main DEBUG Building Plugin[name=layout, class=org.apache.logging.log4j.core.layout.Pattern
2019-08-02 20:42:40,303 main DEBUG PluginManager 'TypeConverter' found 26 plugins
2019-08-02 20:42:40,343 main DEBUG PatternLayout$Builder(pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %ms
2019-08-02 20:42:40,344 main DEBUG PluginManager 'Converter' found 47 plugins
2019-08-02 20:42:40,347 main DEBUG Building Plugin[name=appender, class=org.apache.logging.log4j.core.appender.Con
2019-08-02 20:42:40,373 main DEBUG ConsoleAppender$Builder(target="SYSTEM_OUT", follow="null", direct="null", buff
2019-08-02 20:42:40,377 main DEBUG Starting OutputStreamManager SYSTEM_OUT.false.false
2019-08-02 20:42:40,378 main DEBUG Building Plugin[name=layout, class=org.apache.logging.log4j.core.layout.Pattern
2019-08-02 20:42:40,385 main DEBUG PatternLayout$Builder(pattern="%d %p %c{1.} [%t] %m%n", PatternSelector=null, C
2019-08-02 20:42:40,435 main DEBUG Building Plugin[name=appender, class=org.apache.logging.log4j.core.appender.Fil
2019-08-02 20:42:40,442 main DEBUG FileAppender$Builder(fileName="logs/app.log", append="null", locking="null", ad
2019-08-02 20:42:40,445 main DEBUG Starting FileManager logs/app.log
2019-08-02 20:42:40,462 main DEBUG Building Plugin[name=layout, class=org.apache.logging.log4j.core.layout.Pattern
2019-08-02 20:42:40,463 main DEBUG PatternLayout$Builder(pattern="%d %p %c{1.} [%t] %m%n", PatternSelector=null, C
2019-08-02 20:42:40,464 main DEBUG Building Plugin[name=appender, class=org.apache.logging.log4j.core.appender.Fil
2019-08-02 20:42:40,466 main DEBUG FileAppender$Builder(fileName="logs/Error.log", append="null", locking="null",
2019-08-02 20:42:40,468 main DEBUG Starting FileManager logs/Error.log
2019-08-02 20:42:40,469 main DEBUG Building Plugin[name=appenders, class=org.apache.logging.log4j.core.config.Appe
2019-08-02 20:42:40,548 main DEBUG createAppenders(={LogToConsole, LogToFile, LogToErrorFile})
2019-08-02 20:42:40,549 main DEBUG Building Plugin[name=AppenderRef, class=org.apache.logging.log4j.core.config.Ap
2019-08-02 20:42:40,588 main DEBUG createAppenderRef(ref="LogToFile", level="null", Filter=null)
```

```
Tests passed: 92 (moments ago)
```

Figure 15: JUnit TestCases

REFACTORING

Refactoring is process of restructuring of application internally such that it does not affect application external behavior. Refactoring of code makes it easier to read and maintain. There are various techniques we have followed so that code no longer "smells".

Form Template Method:

In several methods, we need to map and set a list to a model object. So, in that case we had to reputedly assign the values. So, we have refactored the code so that we can pass a result set and it can be used to map object to the desired Model Object List. For an instance. Methods like findStationByStationName and findStationByStationNumber, both the methods need the result set to be mapped in a list of Station Model Objects. So, we have used Row Mapper which implements the code to assign result to model objects and returning List as needed.

Encapsulate field:

Encapsulation is one the important concepts. We have encapsulated data in models by declaring attributes as private and utilizing it by the means of public methods. In Addition, If the data and behavior of a model are closely interrelated and are in the same place in the code, it is much easier for you to maintain and develop this component.

Preserve Whole Object:

In the updateStationDetails method of StationService.java initially, we were passing a list of arguments to update all the data of station. So, instead of a long parameter list, we are passing the whole object.

Remove Assignments to Parameters:

In some of the methods we used to assign the input parameter to a variable and used variable to manipulate the data.This approach was used because Each element of the code was responsible for only one thing. This makes code maintenance much easier going forward, since we can safely replace code without any side effects.

TECHNICAL DEBT

We could have implemented Observer pattern that will notify passengers who have booked specific train tickets whose arrival time is changed or updated by sending them an email consisting of updated arrival time and other booking data.

MEMBER'S CONTRIBUTION

Varun Mahagaokar:

	List of Classes	Methods
Varun Mahagaokar	Booking.java	
	TrainStationMapping.java	
	ClassTrain.java	
	BookingController.java	
	ClassTypeController.java	
	TrainStationController.java	
	DatabaseConfig.java	
	IBookingDAO.java	
	BookingDAO.java	
	IClassTrainDAO.java	
	ClassTrainDAO.java	
	IEmailConfigDAO.java	
	EmailConfigDAO.java	
	ITrainStationDAO.java	

	TrainStationDAO.java	
	IStationDAO.java	
	StationDAO.java	
		stationCountBetweenStations
	ITrainDAO.java	
	TrainDAO.java	
		findTrainBetweenStation
	IDAOFactory.java	
	IDAOFactory.java	
		createBookingDAO
		createClassTrainDAO
		createEmailConfigDAO
		createTrainStationDAO
	IServiceFactory.java	
	ServiceFactory.java	
		createBookingDAO
		createClassTrainDAO
		createEmailConfigDAO
		createTrainStationDAO
	IBookingService.java	
	BookingService.java	
	IClassTrainService.java	
	ClassTrainService.java	
	IEmailConfigService.java	
	EmailConfigService.java	
	ITrainStationService.java	
	TrainStationService.java	
	Utility.java	
	ApplicationTest.java	
	BookingDAOTest.java	
	ClassTrainDAOTest.java	
	StationControllerTest.java	
	ViewControllerTest.java	

	MockBookingDAO.java	
	MockClassTrainDAO.java	
	BookingTest.java	
	ClassTrainTest.java	
	TrainStationMappingTest.java	

Samarth Vyomeshchandra Raval:

Samarth Raval	LoginController.java	
	UserController.java	
		addUser
		findUserByEmail
		findUserById
		findUserByPhone
	IDAOFactory.java	
	DAOFactory.java	
		createUserDAO
		createPasswordValueDAO
	IServiceFactory.java	
	ServiceFactory.java	
		createPasswordValueService
		createUserService
	IPasswordValueDAO.java	
	IUSerDAO.java	
	PasswordValueDAO.java	
	UserDAO.java	
		addUser
		changePassword
		findUserByPhone
		getAllUsername
	User.java	
	PasswordRule.java	
	IPasswordValueService.java	
	IUserService.java	
	PasswordValueService.java	
	UserService.java	
	UniqueUsername.java	
	ValidatePassword.java	

	Digits.java	
	IRuleValidator.java	
	Length.java	
	LowerCase.java	
	SpecialCharacter.java	
	UpperCase.java	
	PasswordValueDAOTest.java	
	UserDAOTest.java	
	MockPasswordValueDAO.java	
	MockUserDAO.java	
	PasswordRuleTest.java	
	UserTest.java	
	UniqueUsernameTest.java	
	ValidatePasswordTest.java	

Dhruvi Shah:

Dhruvi Shah	ReportController.java	
	StationController.java	
	TrainController.java	
	ViewController.java	
	UserController.java	
		getUserList
		addUser
		updateUserDetails
		deleteUserById
		getTotalNonAdminUser
	Train.java	
	Station.java	
	User.java	
	ITrainService.java	
	TrainService.java	
	ReportService.java	
	IReportService.java	
	StationService.java	
	IStationService.java	
	UserService.java	
	IUserService.java	

	StationTest.java	
	TrainTest.java	
	StationDAOTest.java	
	TrainDAOTest.java	
	MockStationDAO.java	
	MockTrainDAO.java	
	StationTest.java TrainTest.java	
	IDAOFactory.java	
	DAOFactory.java	
		createReportDAO
		createStationDAO
		createTrainDAO
	IServiceFactory.java	
	ServiceFactory.java	
		createReportService
		createStationService
		createTrainService

REFERENCES:

[1] GeeksforGeeks. (2019). *Design Patterns* | *Set 2 (Factory Method) - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/design-patterns-set-2-factory-method/ [Accessed 2 Aug. 2019].

[2] GeeksforGeeks. (2019). *Singleton Design Pattern* | *Implementation - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/singleton-design-pattern/ [Accessed 2 Aug. 2019].

[3] Paranj, B. (2019). *Java Tip 68: Learn how to implement the Command pattern in Java*. [online] JavaWorld. Available at: https://www.javaworld.com/article/2077569/java-tip-68--learn-how-to-implement-the-command-pattern-in-java.html [Accessed 2 Aug. 2019].

[4] Design Patterns and Refactoring. (2019). Retrieved 2 August 2019, from https://sourcemaking.com/refactoring/refactorings