International Institute of Information Technology Bangalore

(IIIT Bangalore)

Courier Management For IIITB

In the Guidance of **Prof. B. Thangaraju**
Teaching Assistant: **Suchi (MT2021140)**

Archit Sangal
IMT2019012

Aditya Vardhan
IMT2019003

Table Of Contents

Abstract **3**
Overview **3**
Features & Functionalities 3
What and Why DevOps 4
What is DevOps? 4
Why DevOps? 6
Technologies and Tools Used **7**
System Configuration **7**
Git and GitHub 7
Assumptions 8
Maven Project 9
Jenkins Setup 10
Jenkins Item Setup 14
Jenkins Pipeline 15
WebHooks 16
MySQL 17
NPM and Node 17
Ansible 18
Software Development Life Cycle **18**
Source Control Management : Git/Github 18
Frontend 20
Login Page: 20
Forgot Password: 21
Register page: 22
Adding Courier: 23
Receiving Courier: 24
Delete Couriers: 25
Responsive Calendar: 26
Viewing Available Couriers: 27
Viewing All Couriers: 28
Searching By Roll Number or By Name: 29
Users View: 30
Backend 31
Installations: 31
MySQL: 31
Dependencies used: 31
Testing with mockito and junit: 33
App Functionalities: 35
API Documentation **35**
Available API(s) for admin: 35
Available API(s) for users: 42
Docker **50**
Automation **50**
Ansible 50
Jenkins 52
Kubernetes **55**
Logging **58**
Continuous Monitoring **59**

Scope of Future Work **60**
Challenges Faced **61**
Conclusion **61**
References **61**

Abstract

A comprehensive solution that makes it easier to receive and pick up packages on campus is the courier management application that our team created for our college, IIIT Bangalore. The application is made to meet the needs of students, faculty, staff members, and administrators who need to manage packages that need to be picked up on college property.

Users of the application can easily track packages thanks to its responsive and user-friendly interface.

Numerous features offered by the courier management application guarantee effective management of package delivery. The user's ability to track packages is one of the application's most crucial features. Users can receive notifications about the status of their packages, including when the owner or someone acting on their behalf received them as well as when the admin received them from the delivering company. The application includes automatic notifications that keep users updated on the status of their package, ensuring that users are always aware of the status of their package. Email notifications enable users to stay informed about the status of their packages at all times. Additionally, this feature reduces the possibility of packages getting misplaced or lost.

Admin has more functionality and has more control over the package information. Consequently, a feature for tracking package history is also included in the admin's side courier management application. With the help of this feature, administrators can view the history of their packages, including the times that they were picked up and delivered. Only the administrator has the ability to add packages and mark them as received. There are also numerous additional features. Scroll to features section 2.2 for more information.

Overall, the courier management software created for our college is a useful and easy-to-use application that streamlines the process of adding and receiving packages on campus. The college community can reduce errors and save time by using this application to track and deliver packages manually.

Overview

Features & Functionalities

We have included a lot of features in our application, details of the features can be found in the respective sections (e.g. Software Development Life Cycle -> Backend -> App Functionalities), here is a list of them:

1. We have implemented all the **CRUD functionalities**, with great UI. Users can view the packages while the admin can create, read, update and delete courier entries.
2. Security features like **JWT authentication** have been implemented in the application, which takes care of **authorization and authentication** as we also have user and admin roles (**role-based access**) in the backend.
3. We have **OTP verification** for new users. Also if a user wants to **update the password** for an account they need to again go through OTP verification.
4. There is also a **notification system** being implemented, so when a package is being **added and received**, the user to whom that package belongs gets notification for the same.
5. We have **search functionality** implemented also, using substring search.
6. We have **continuous monitoring** too.
7. **Ingress** has been implemented to professionally deploy the frontend and backend.

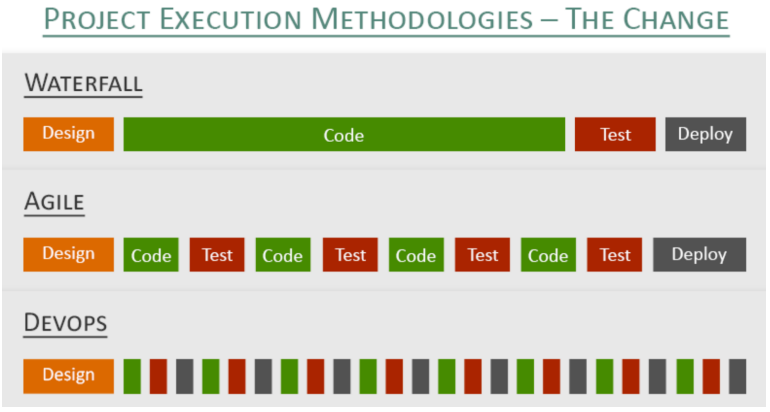
Link To the repository: https://github.com/architsangal/Courier-Management-Application (branch : main)

What and Why DevOps

What is DevOps?

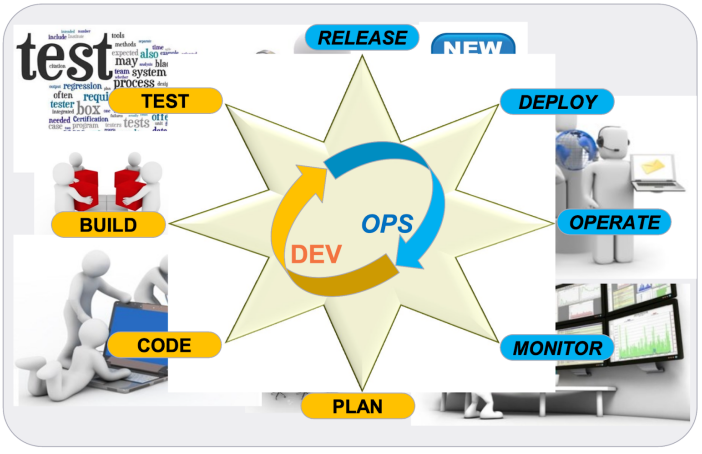
Using the DevOps methodology, software development teams and IT operations teams are encouraged to work together and communicate. DevOps aims to improve both the software's quality and the speed and efficiency with which it is developed and deployed.

Infrastructure automation, continuous integration, and continuous delivery are just a few of the techniques and tools used in DevOps. Continuous integration allows for early detection of integration errors by having developers frequently merge their code changes into a central repository. Automating the software deployment process through continuous delivery enables prompt and dependable release of new features and updates. Utilizing tools to automatically configure and manage IT infrastructure, such as servers and networks, is known as infrastructure automation.



DevOps promotes a culture of cooperation, communication, and shared accountability amongst developers, operations teams, and other stakeholders in addition to these methods and technologies. This entails regular meetings, feedback loops, and a concentration on ongoing learning and development.

Ultimately, the goal of DevOps is to eliminate barriers between development and operations teams and to establish a setting where software is built and deployed more quickly, dependably, and effectively.



➢ DevOps is a set of practices that combines software development (Dev) and IT operations

(Ops) to improve the speed and quality of software delivery. The goal of DevOps is to break down the traditional silos between development and operations teams and create a more collaborative and agile environment.

- DevOps practices include continuous integration and continuous delivery (CI/CD), infrastructure as code (IaC), monitoring and logging, and automated testing. By automating and streamlining these processes, DevOps teams can reduce the time and effort required to deliver software, while also improving the reliability and quality of the software being delivered.

- DevOps also emphasises a culture of collaboration, transparency, and continuous improvement. This involves creating cross-functional teams that include developers, operations engineers, and other stakeholders, and encouraging open communication and feedback. DevOps teams also embrace a "fail fast, learn fast" mentality, which involves testing and iterating on software quickly and continuously in order to identify and fix issues before they become major problems.

- Overall, DevOps is a key enabler of digital transformation and has become increasingly important in today's fast-paced, cloud-based software development landscape. By embracing DevOps practices, organisations can improve their agility, reduce their time to market, and deliver higher-quality software that meets the needs of their customers.

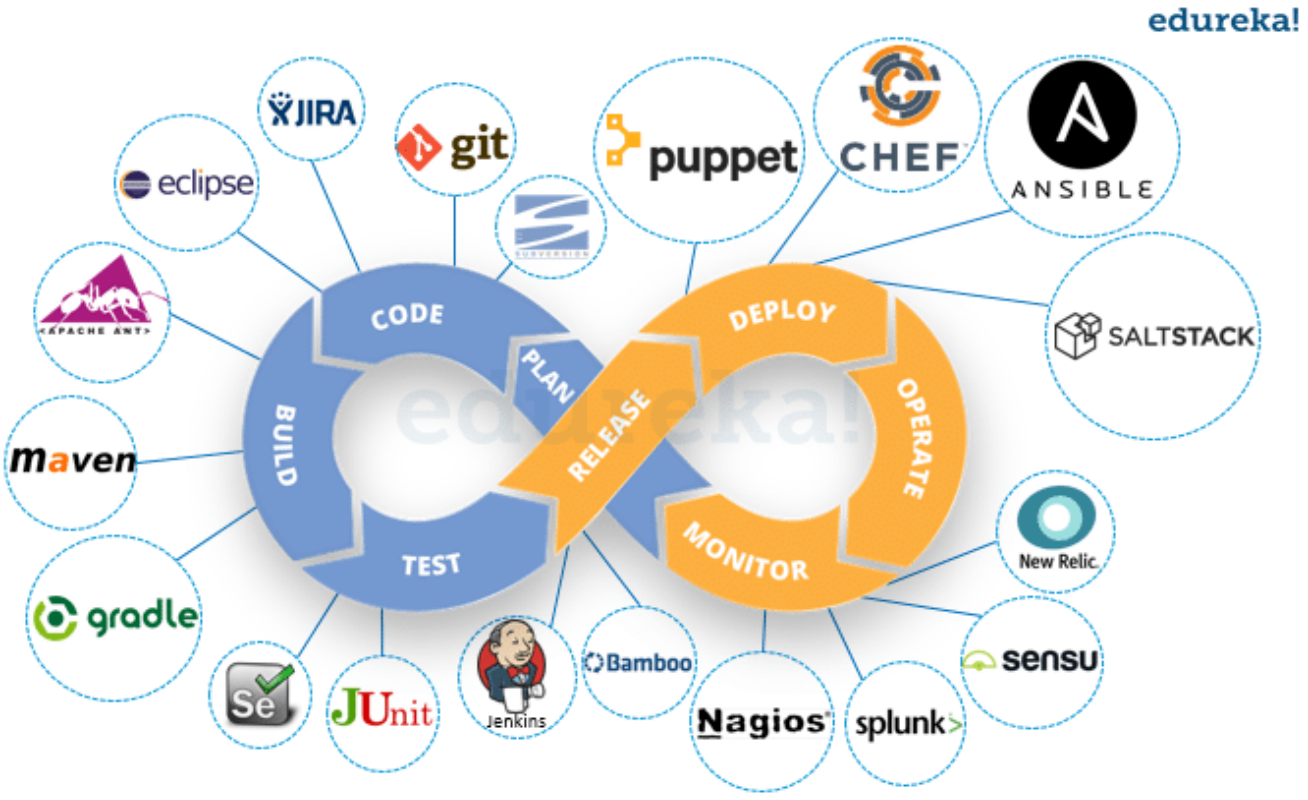


Figure: Above image is taken from edureka!

Why DevOps?

➢ DevOps should be used because it offers several benefits to organizations that can help them to become more agile, efficient, and competitive in today's fast-paced digital landscape. Here are some of the key reasons why DevOps should be used:

- Faster Time-to-Market: By automating and streamlining software delivery processes, DevOps enables organizations to release software faster, more frequently, and with fewer errors. This can give organizations a competitive edge by allowing them to deliver new features and functionality to customers more quickly.
- Improved Collaboration: DevOps promotes a culture of collaboration between development and operations teams, which helps to break down silos and improve communication. This can lead to better alignment between different parts of the organisation, which can result in faster and more efficient problem-solving.
- Increased Efficiency: DevOps practices like automation, continuous integration, and continuous delivery can help organisations to reduce the time and effort required to deliver software. This can lead to greater efficiency and productivity, and can free up resources to focus on other areas of the business.
- Improved Quality: By emphasising automated testing, monitoring, and logging, DevOps helps organisations to identify and fix issues more quickly, before they become major problems. This can lead to higher-quality software that meets the needs of customers more effectively.
- Greater Agility: DevOps enables organisations to respond more quickly to changing market conditions and customer needs. By automating and streamlining software delivery processes, organisations can become more agile and responsive, which can help them to stay ahead of the competition.

Technologies and Tools Used

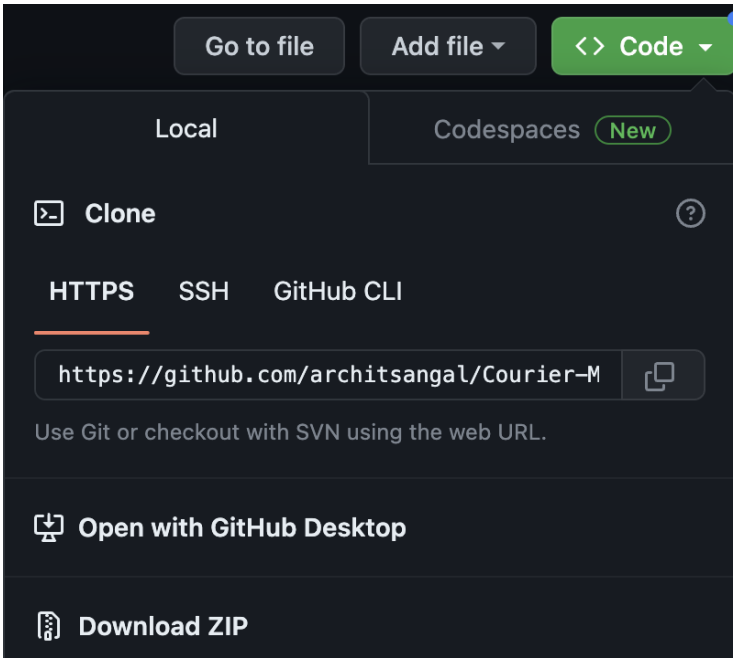
- Version Control System Used: Git and GitHub
- Testing: JUnit and Mockito Core
- CI/CD Pipeline: Jenkins
- Building/Packaging: Jenkins with docker (have explained this in detail in the respective sections)
- Containerization: Docker
- Deployment: Ansible
- Log creation: Spring Boot Logging Internal Libraries with Levels
- Analysis and Visualisation: ELK
- Continuous Monitoring: Prometheus and Grafana
- Frontend - React
- Backend - SpringBoot (a maven project)
- Database- NoSQL

System Configuration

Git and GitHub

The history of file changes, source code, and other information are tracked using source control management. It aids in keeping the ongoing project well-structured and organised in a variety of ways. It makes it possible for several programmers from the same team to work on the same project. In this project, we'll be using the cloud-based version control tool Github as our SCM tool.

Github Repository Link: https://github.com/architsangal/Courier-Management-Application/



Initialising your local project repo with git:

```
git init
git remote add origin <github repo URL>
```

Workflow:

```
git add <files>
git commit -m <Your commit message>
git pull origin main
git push -u origin main
```

Assumptions

I am assuming the following installations are already in place in your local system -

- 'git' version control system

```
$ git --version
```

```
git version 2.39.0
```

- Java 11

```
$ java --version
java 11.0.16.1 2022-08-18 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.16.1+1-LTS-1)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.16.1+1-LTS-1, mixed mode)
```

- 'Jenkins' basic setup, where you can login and create a pipeline job.

```
$ whereis jenkins-lts
jenkins-lts: /opt/homebrew/bin/jenkins-lts
```

- 'Docker' is installed in the local system.

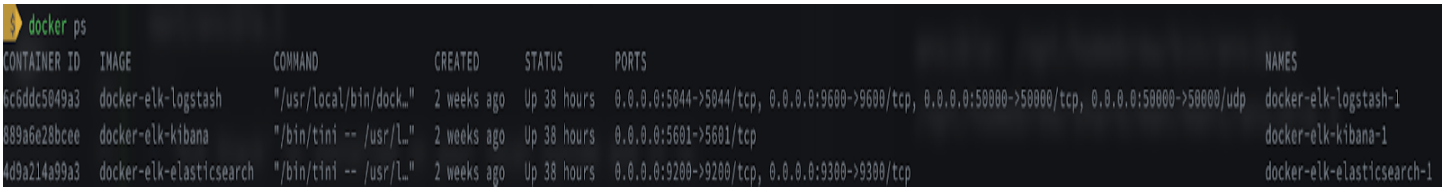
```
$ whereis docker
docker: /usr/local/bin/docker
```

- 'Ansible' is installed in your local system.

```
$ whereis ansible
ansible: /opt/homebrew/bin/ansible /opt/homebrew/share/man/man1/ansible.1
```

- 'ELK Stack' is installed in your local system and it is running.

```
$ docker ps
```



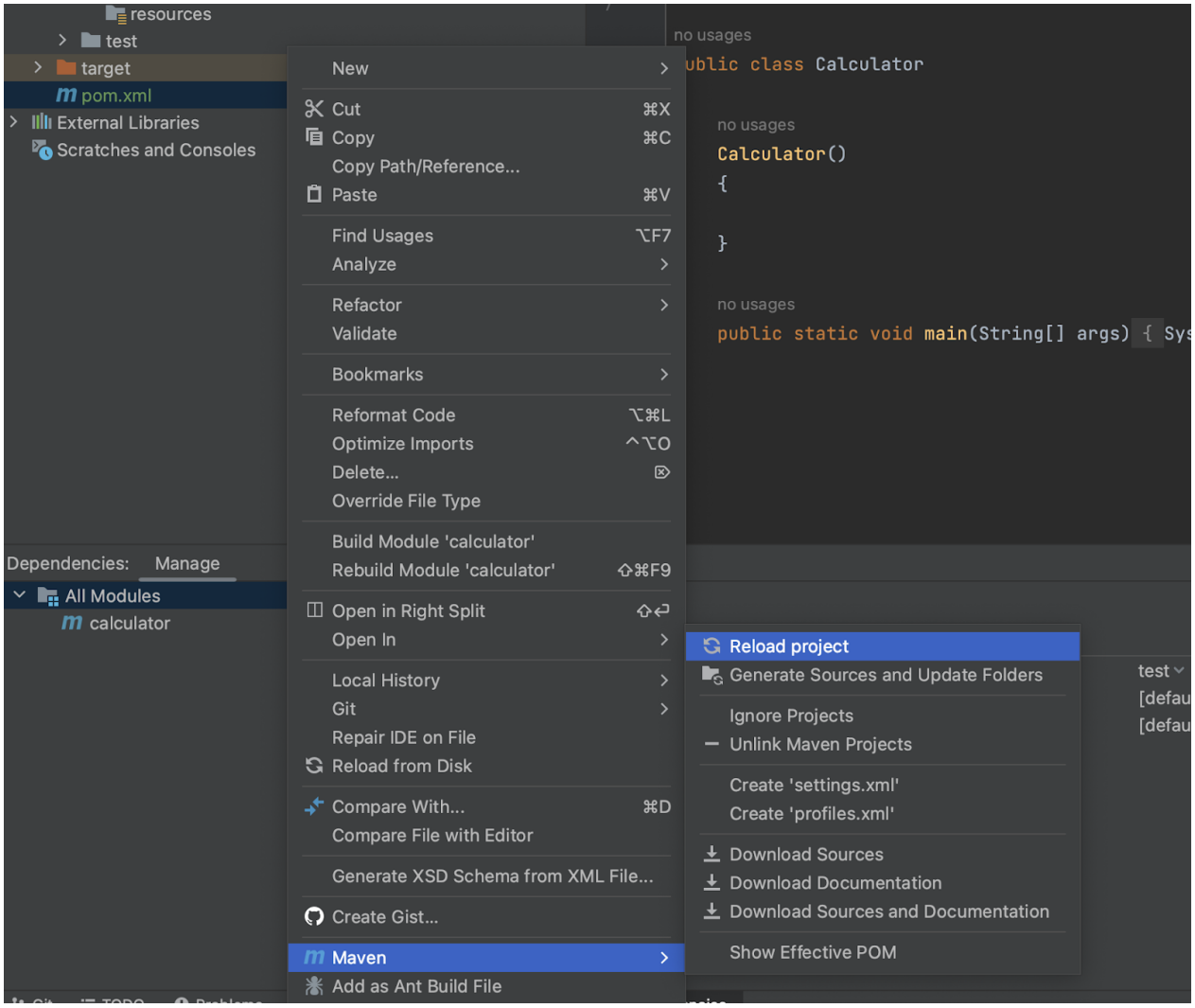
- 'IntelliJ IDEA CE' is installed in the system.

- 'ngrok' is installed on the local system.

The versions of the above softwares may vary upto some extent and depending on how you installed them, using **`brew`** or **`.app`** files, etc.

Maven Project

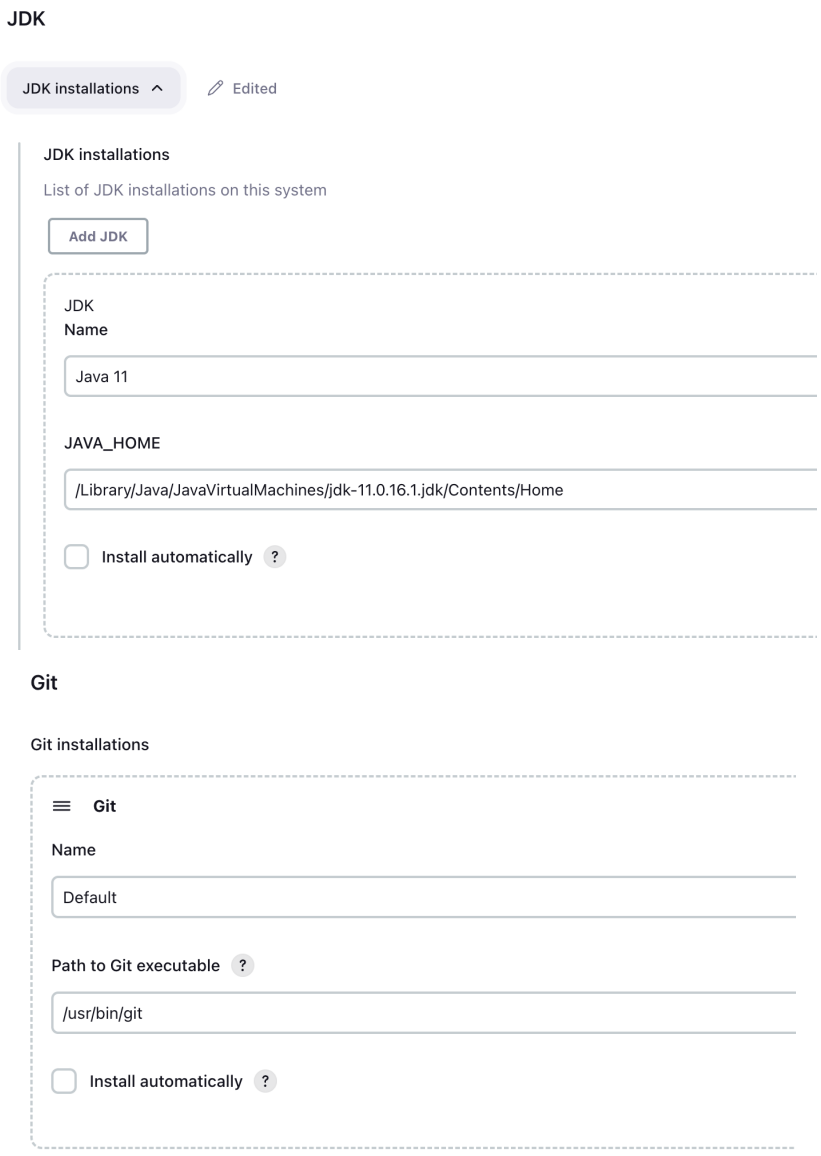
- Code is a standard maven project with a test module with pom.xml managing the dependencies. See the project directory.
- Update the projects with the dependencies which you have included in your pom.xml file using the procedure given below -

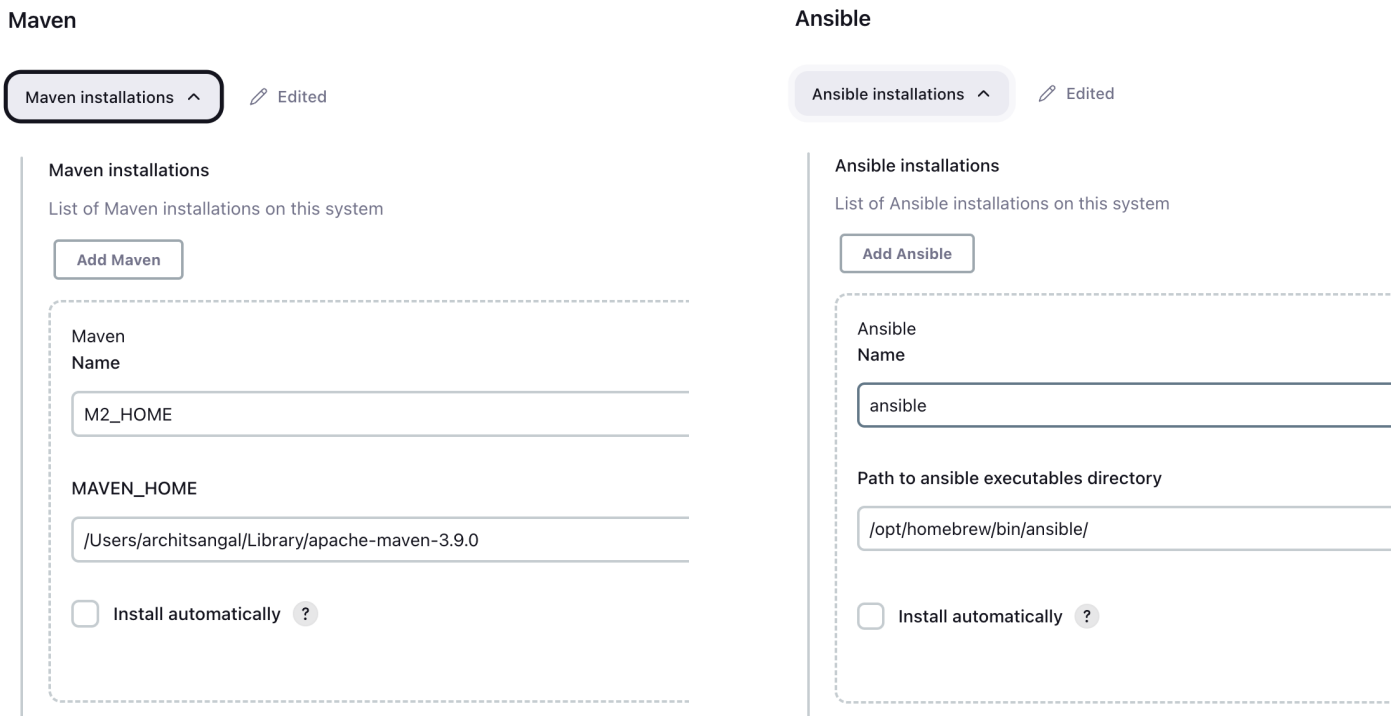


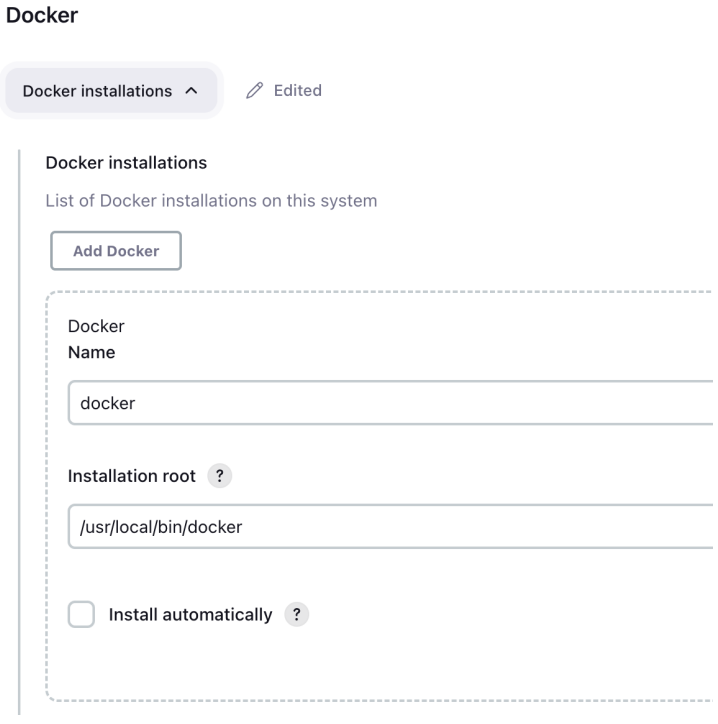
Jenkins Setup

- Login into the Jenkins.
- Follow the path: Jenkins Dashboard -> Manage Jenkins -> Configure System, and replicate the below -

- SPE Mini Project is a secret text, which is for GitHub WebHook.
- Follow the path: Jenkins Dashboard -> Manage Jenkins -> Configure Global Tools, and replicate the below -







- Go to celler repo of jenkins and add path to it's **`homebrew.mxcl.jenkins-lts.plist`** file.

```
$ cd /opt/homebrew/Cellar/jenkins-lts/
$ cd 2.375.3 #some version
$ ls
```

There you will find a file `homebrew.mxcl.jenkins-lts.plist`. Add path to it -

```
<key>EnvironmentVariables</key>
<dict>
    <key>PATH</key>
<string>/opt/homebrew/bin/:/usr/local/bin/:/usr/local/bin/docker:/usr/local/bin:/usr/bi
n:/bin:/usr/sbin:/sbin:/Applications/Docker.app/Contents/Resources/bin/:/Users/admin/Li
brary/Group\ Containers/group.com.docker</string>
</dict>
```

After the above step, `homebrew.mxcl.jenkins-lts.plist` should look something like this-

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>homebrew.mxcl.jenkins-lts</string>
    <key>LimitLoadToSessionType</key>
    <array>
        <string>Aqua</string>
        <string>Background</string>
        <string>LoginWindow</string>
        <string>StandardIO</string>
        <string>System</string>
    </array>
    <key>ProgramArguments</key>
    <array>
        <string>/opt/homebrew/opt/openjdk@17/bin/java</string>
        <string>-Dmail.smtp.starttls.enable=true</string>
        <string>-jar</string>
        <string>/opt/homebrew/opt/jenkins-lts/libexec/jenkins.war</string>
        <string>--httpListenAddress=127.0.0.1</string>
        <string>--httpPort=8080</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>EnvironmentVariables</key>
    <dict>
        <key>PATH</key>

<string>/opt/homebrew/bin/:/usr/local/bin/:/usr/local/bin/docker:/usr/local/bin:/usr/bi
n:/bin:/usr/sbin:/sbin:/Applications/Docker.app/Contents/Resources/bin/:/Users/admin/Li
brary/Group\ Containers/group.com.docker</string>
    </dict>
</dict>
</plist>
```

- Follow the path: Jenkins Dashboard -> Manage Jenkins -> Manage Credentials, add the keys and user name, mine looks like the following:

Credentials

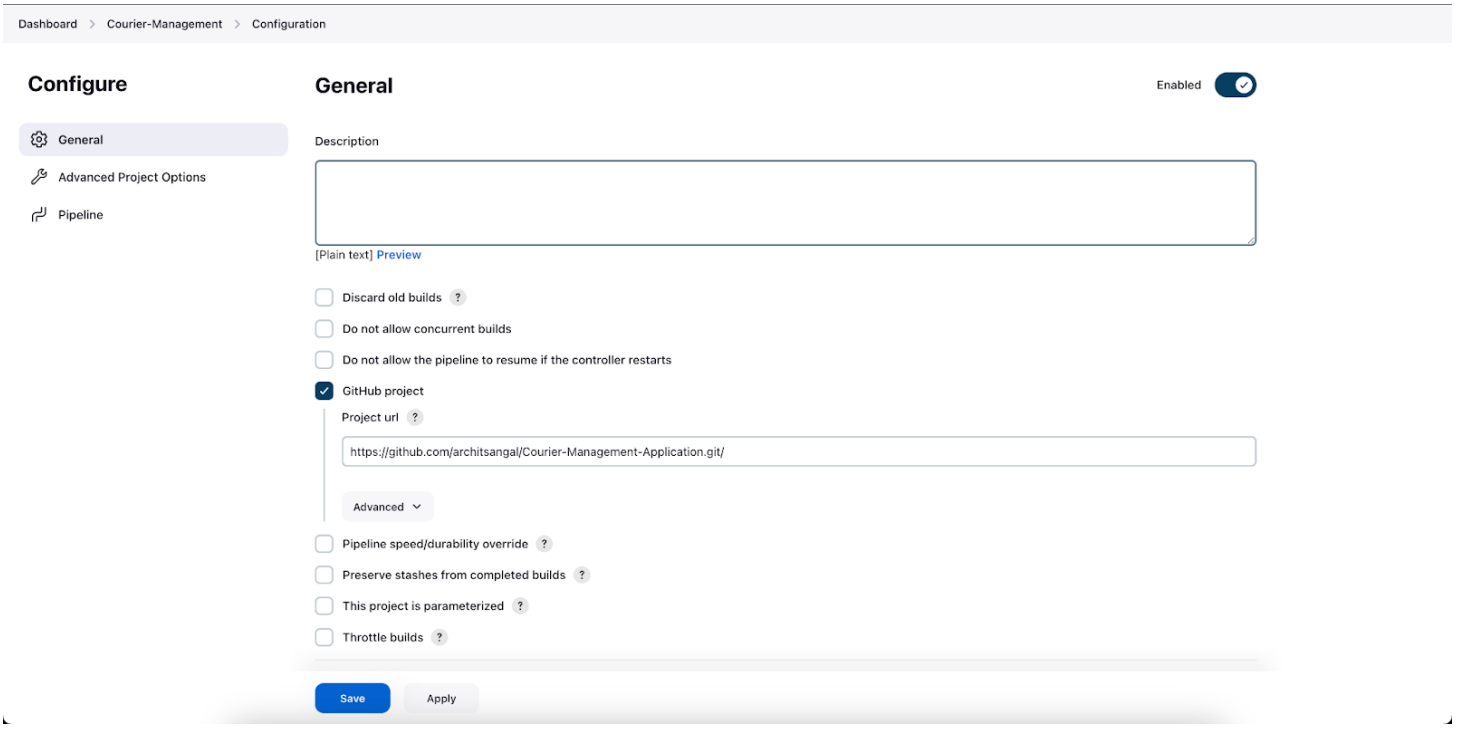
T	P	Store ↓	Domain	ID	Name
		System	(global)	49654976-14f2-4c60-aeb6-4804fc359cec	Secret text
		System	(global)	8ad63d4d-7c23-43a4-8638-47f78b28ace5	Secret text
		System	(global)	Master_Jenkins_Private_key	jenkins (Jenkins Master Private Key to Add Multiple Agents)
		System	(global)	dockerhub	architsangal/****** (DockerHub Credentials)
		System	(global)	2cc913cf-8c05-4c83-988d-42c1451aedd1	SPE Mini Project

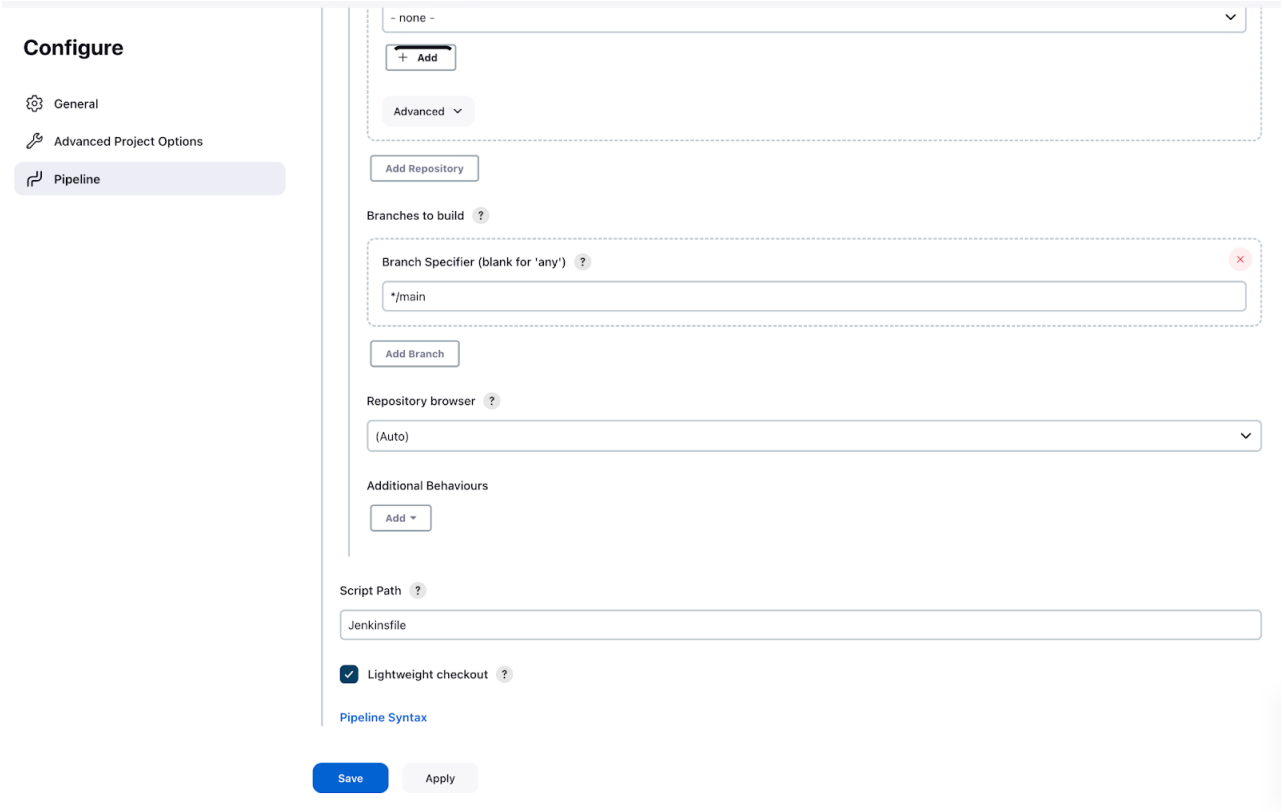
Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

Jenkins Item Setup

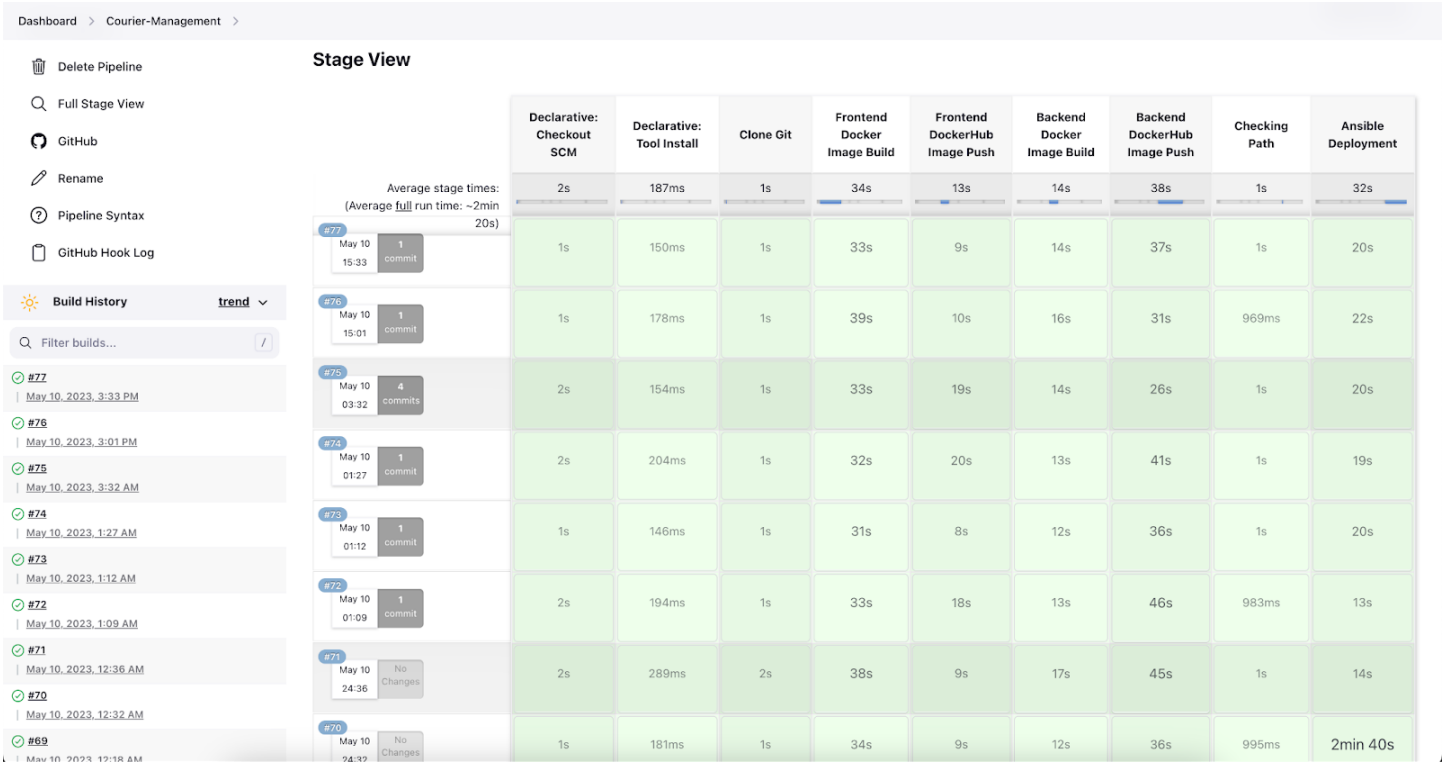
- Add a new Item of type `Pipeline`
- Go to Configure Section, and replicate the following:





Jenkins Pipeline

- start the ‘jenkins’ service.
- Click On build Now, you have your pipeline.



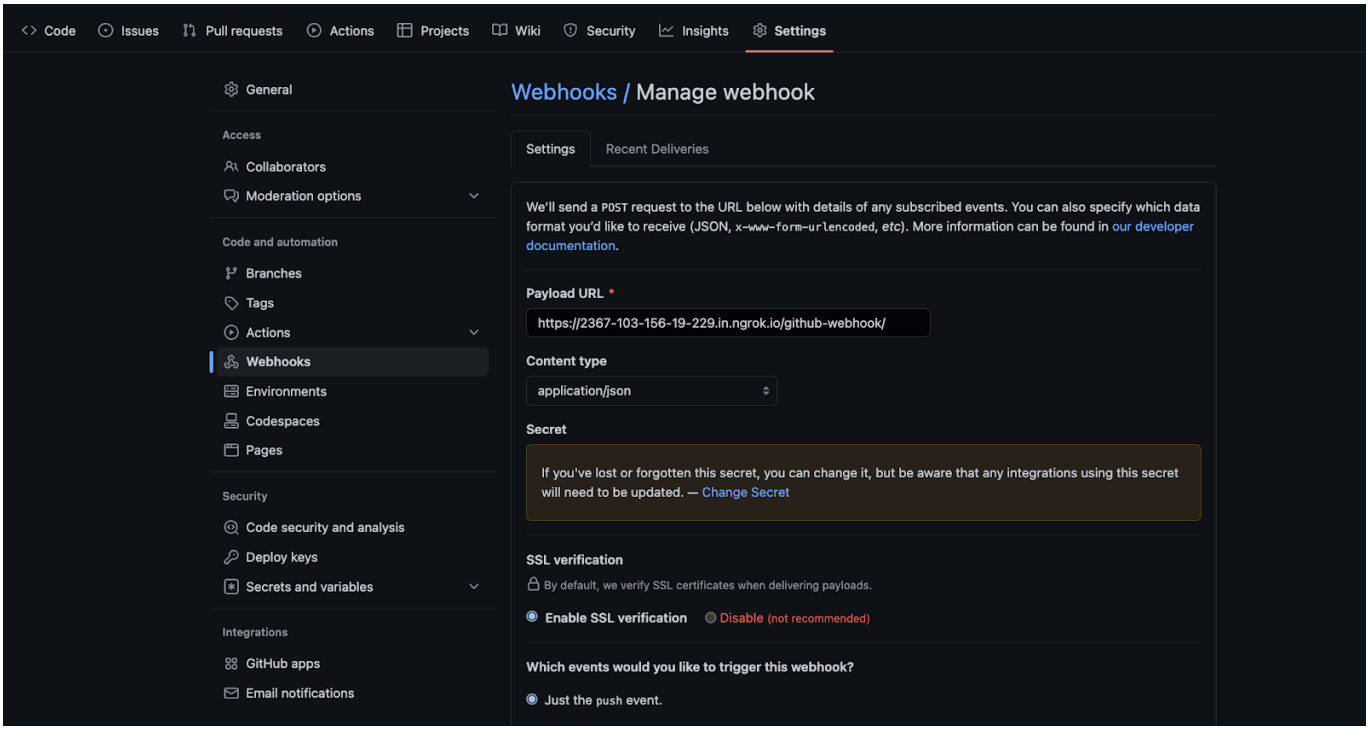
WebHooks

- Webhooks are messages that are transmitted immediately whenever there is a change in the environment. If we make any changes to the GitHub repository, the webhook will activate the Jenkins pipeline immediately, and it will do so automatically. Ngrok is able to link local servers that are protected by NATs (Network Address Translation) and firewalls to the public internet by utilising encrypted tunnels. It is equipped with a real-time web interface that gives you the ability to view any HTTP traffic that is moving through your tunnels. It gives you the capability of connecting to the internet through a web server that is operating on your local machine. Just provide ngrok with the port number on which your web server is actively listening.

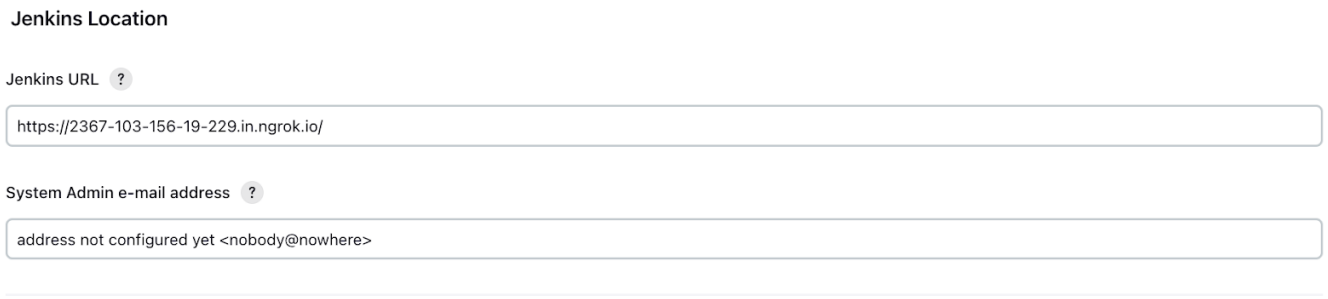
```
ngrok http 8080
```

```
ngrok                                                                    (Ctrl+C to quit)

Add OAuth and webhook security to your ngrok (its free!): https://ngrok.com/free

Session Status                online
Account                       vardhan.av2000@gmail.com (Plan: Free)
Version                       3.1.1
Region                        India (in)
Latency                       213ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://2367-103-156-19-229.in.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              5       0       0.00    0.00    30.08   59.78

HTTP Requests
-------------

POST    /github-webhook/                                                 200 OK
```

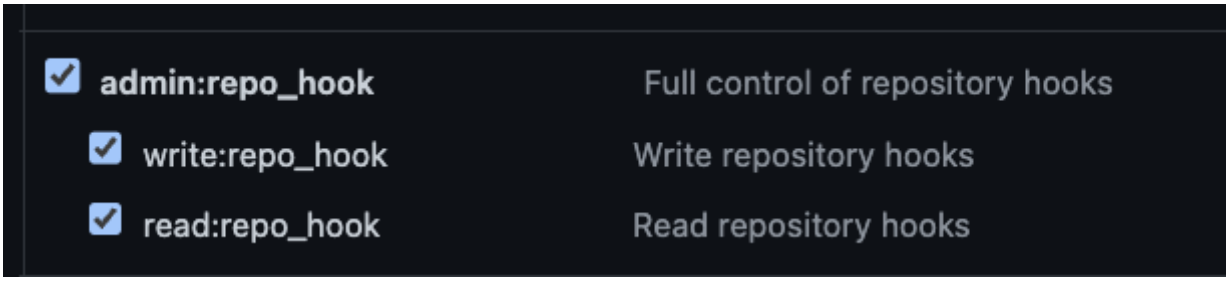
In the github repository go to settings and in Webhooks add ngrok address in payload url and the personal access token in secret.

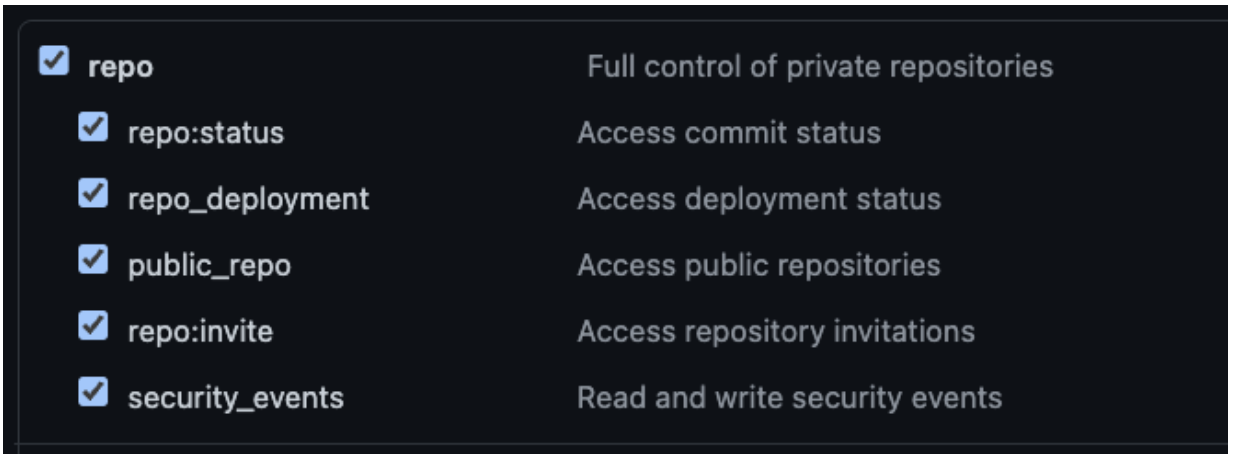


In Jenkins → Configure System



The scope of the secret key must have the following:





MySQL

If brew installed it just check if it is running.

NPM and Node

Assuming npm and node are installed, run the following:

```
npm install
npm start
```

Ansible

Ansible is an open-source automation tool that simplifies IT configuration management and application deployment across servers, networks, and cloud environments using a simple, human-readable language. It allows users to automate tasks across multiple systems, making it easier to manage and scale infrastructure. Ansible uses a push-based model, where the control machine pushes configurations and commands to remote machines, and supports a wide range of modules and plugins for managing different technologies and platforms. Ansible is used by IT teams of all sizes to automate repetitive tasks, improve efficiency, and reduce errors in the IT environment.

Just check if ansible is installed otherwise you can follow the guidelines given in documentation.

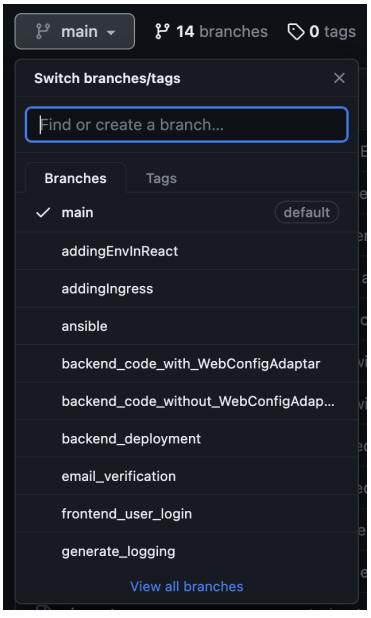
Software Development Life Cycle

Source Control Management : Git/Github

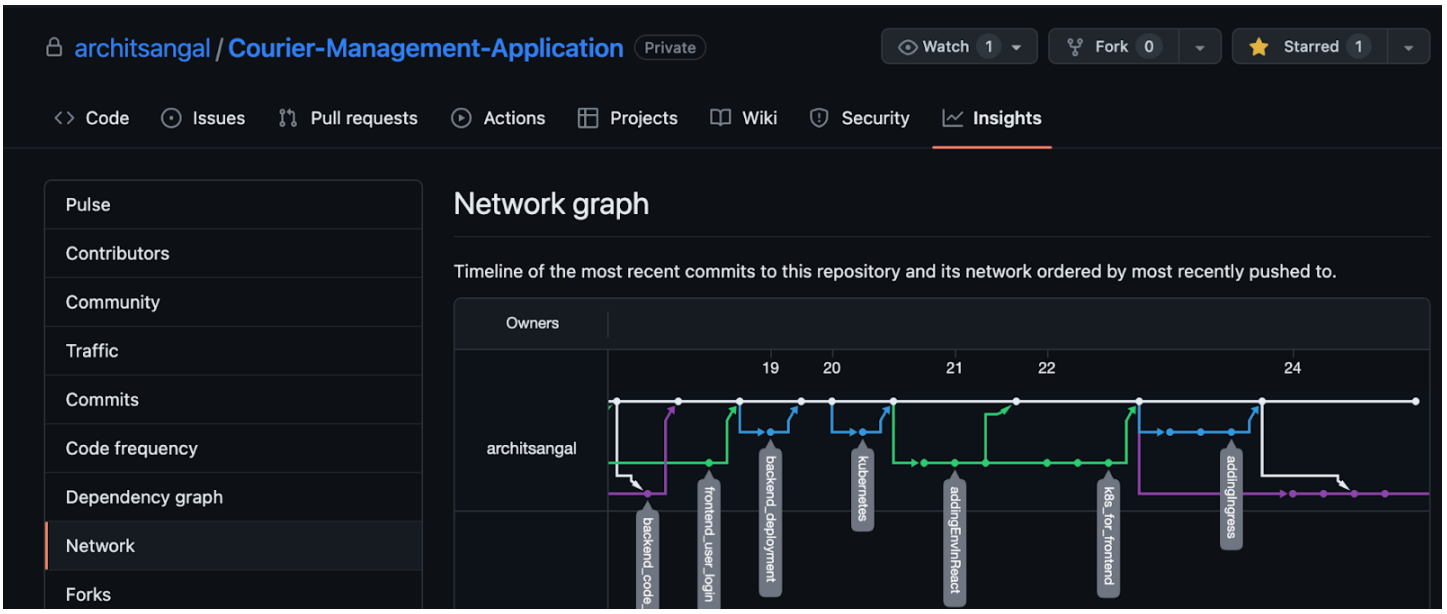
Git commit: A local repository can be updated by using the Git version control system's commit command. The changes to the files are captured in a snapshot by a commit, which also contains a note outlining the changes. Git commit improves collaboration amongst developers by allowing several contributors to work on the same codebase at once and granular control over versioning.

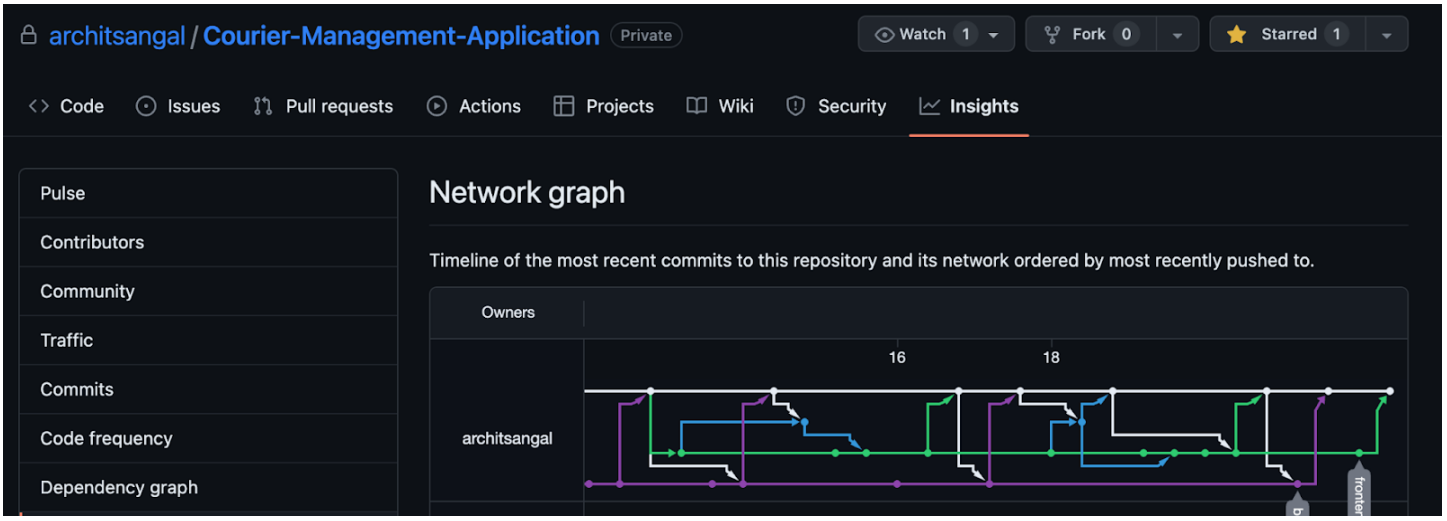
Our project contains 160+ commits.

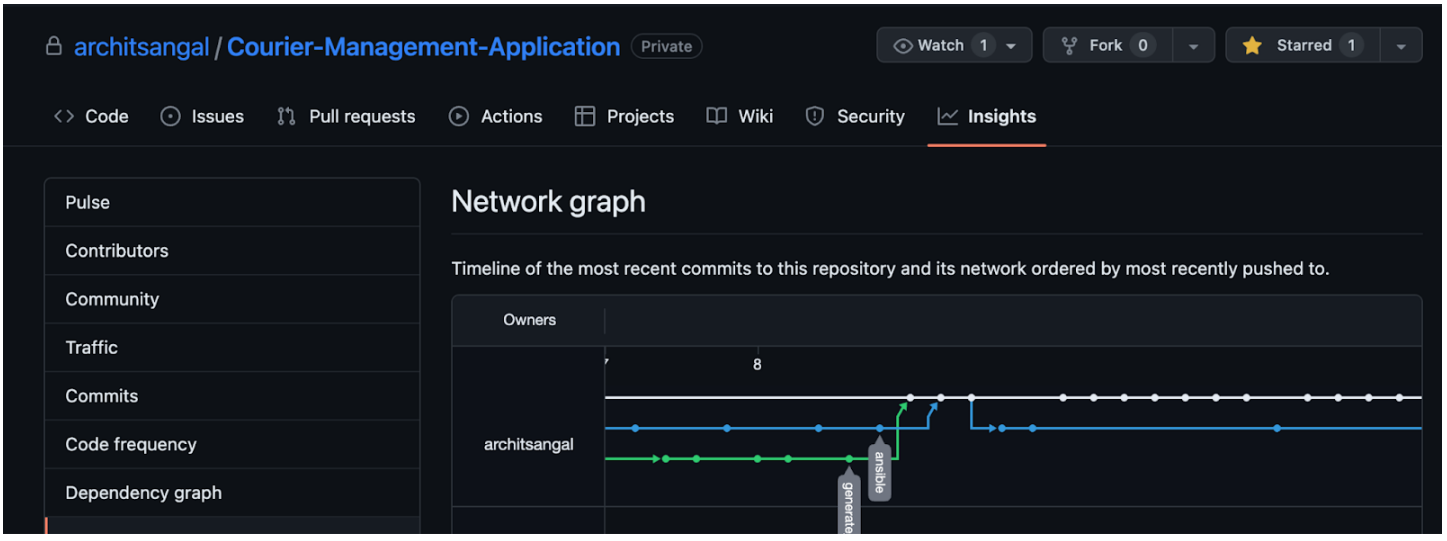
Git branch: A branch in Git is an independent line of development that enables changes to a codebase without affecting the primary codebase. Branches can be used to test out new functionality, address issues, or make other codebase modifications without interfering with the main development process. Git branch enables the creation and management of numerous branches as well as the merging of changes between branches when they are prepared to be incorporated into the main codebase. Branches can be used to separate changes made by various contributors, facilitating collaborative development in parallel.



For each new feature we extensively used the git branch. It allowed us to freely experiment with the features and modifications without worrying about disturbing the main code that is in working condition. At many places it saved us from having merge conflicts.



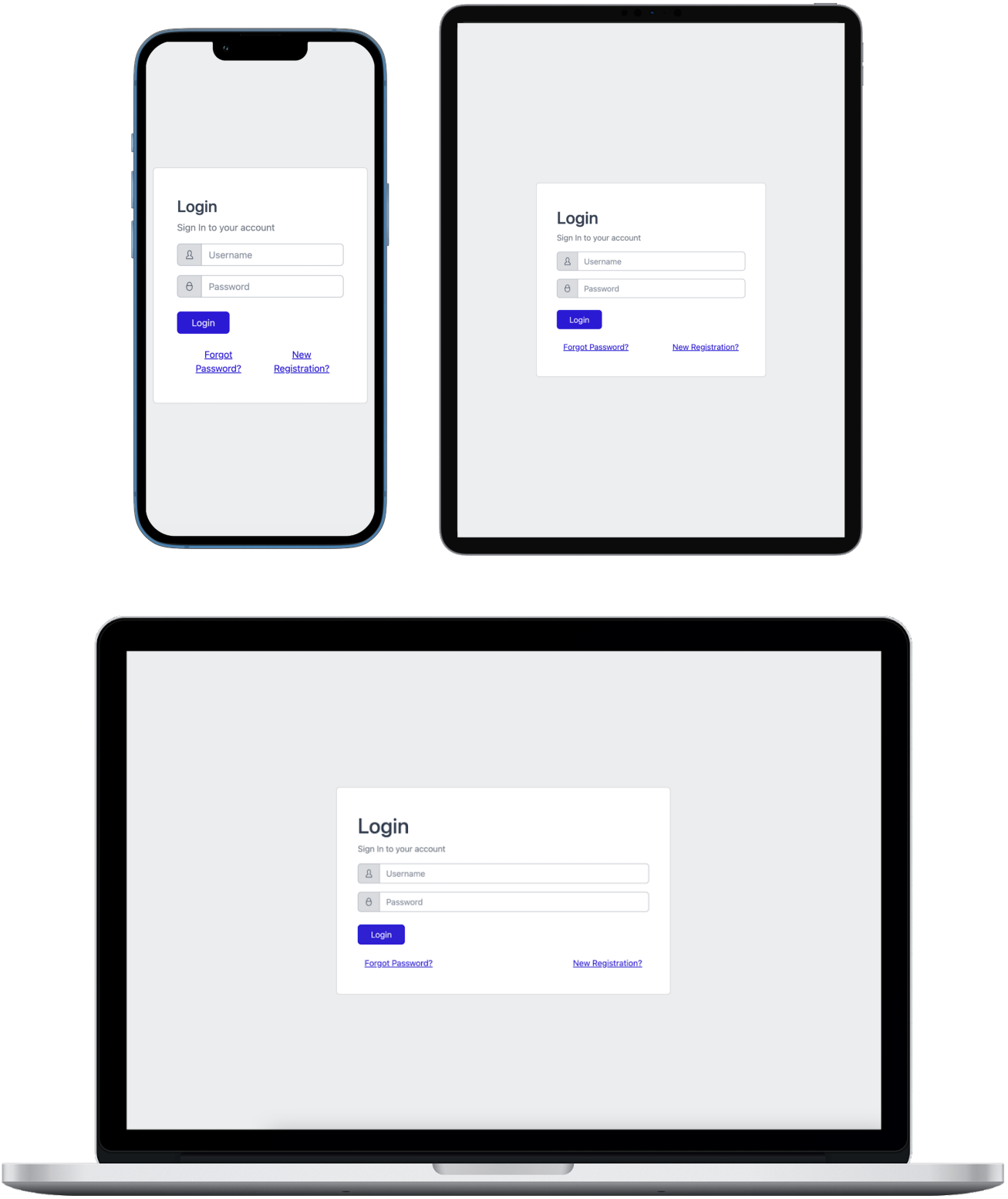




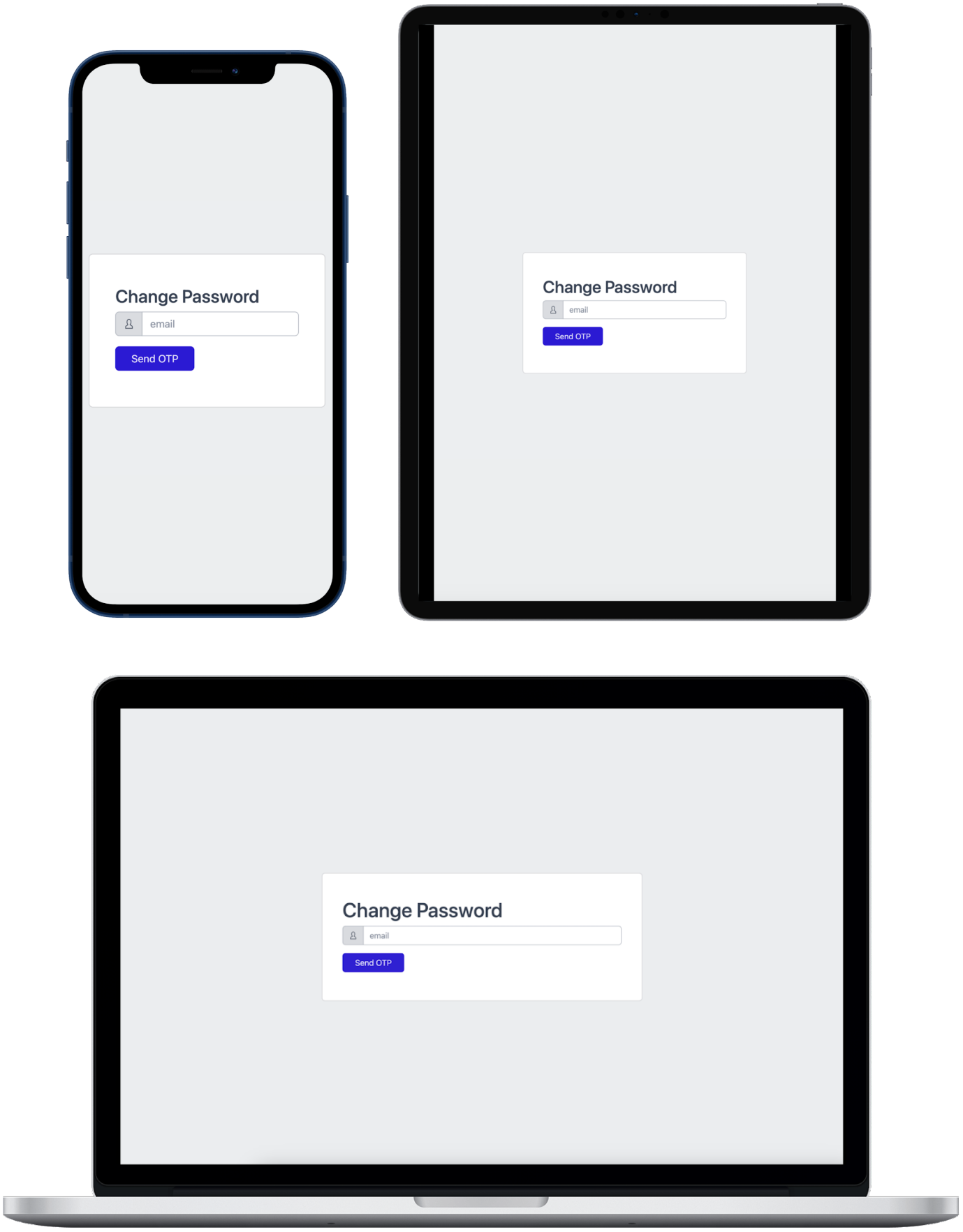
Frontend

We have created an excellent UI from the template provided by using Core UI. It is a fully responsive frontend below are the various screen shots:

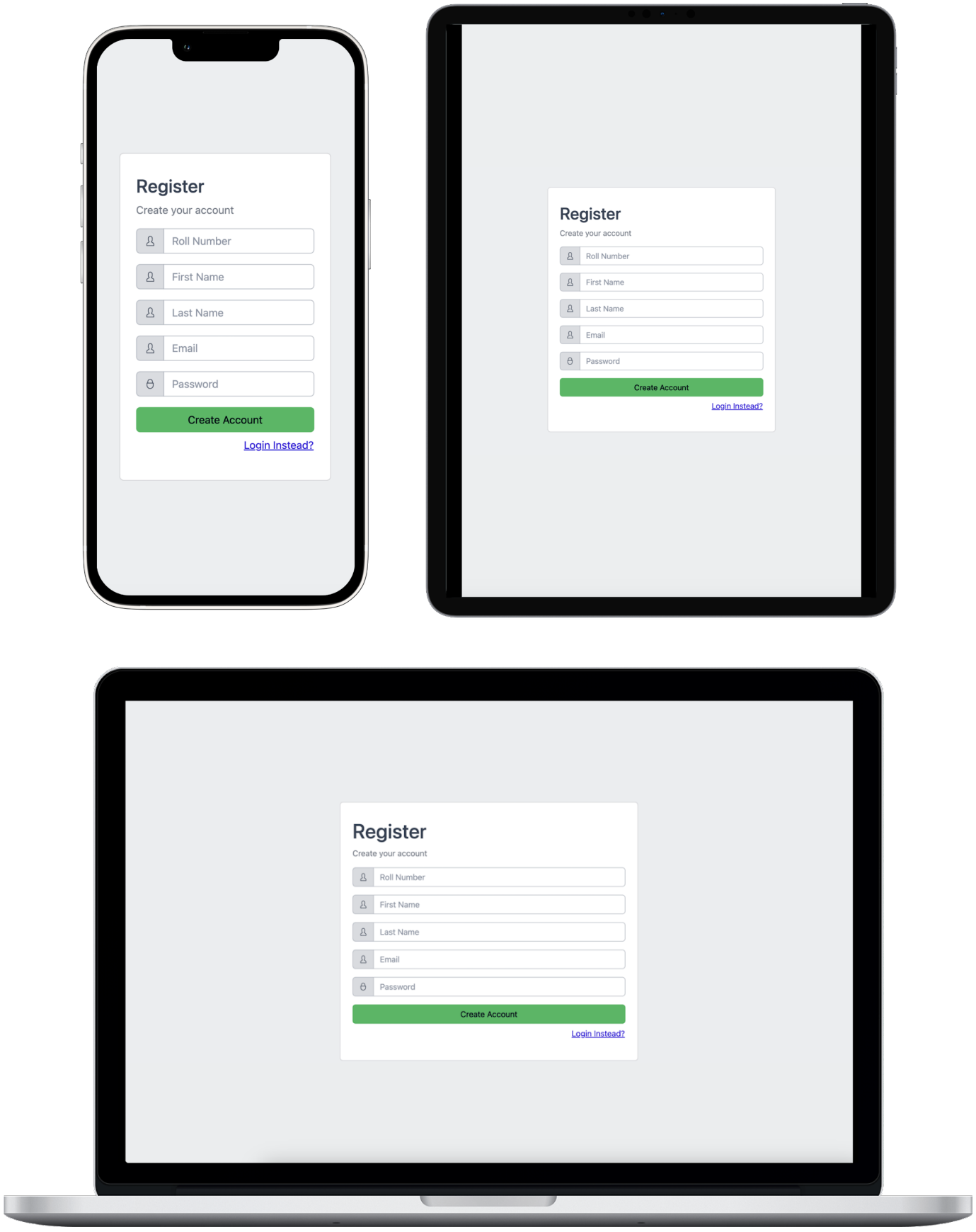
Login Page:



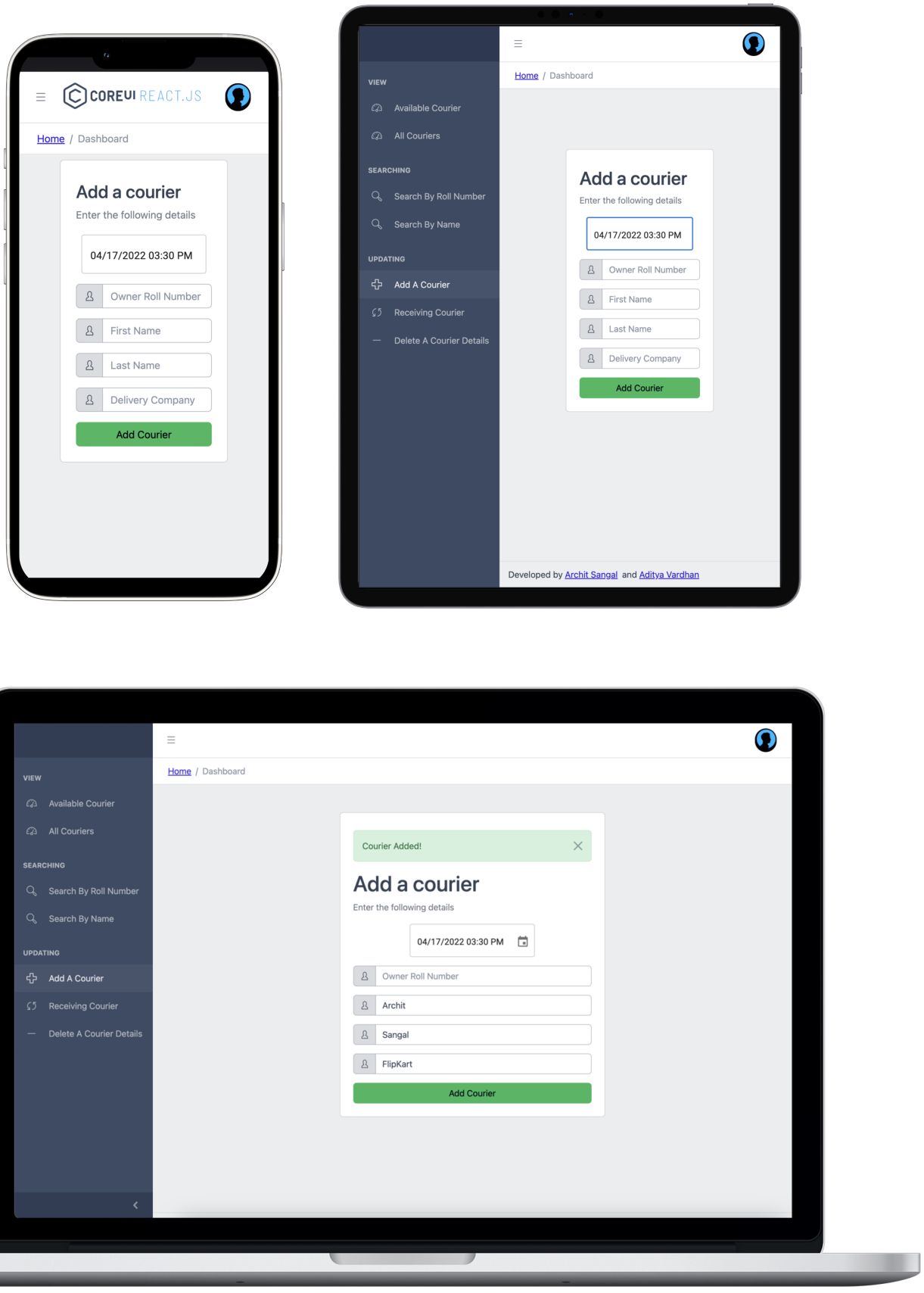
Forgot Password:



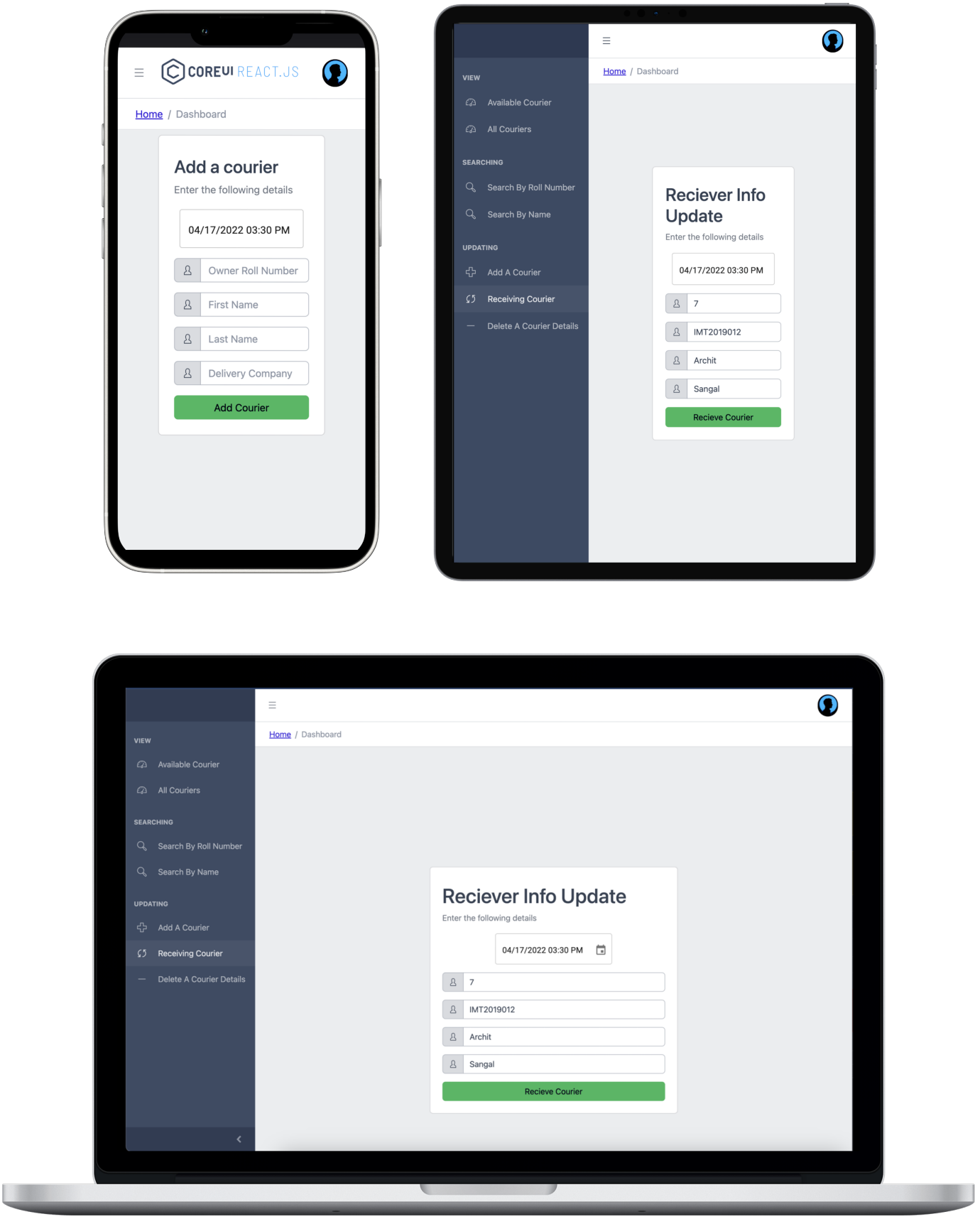
Register page:



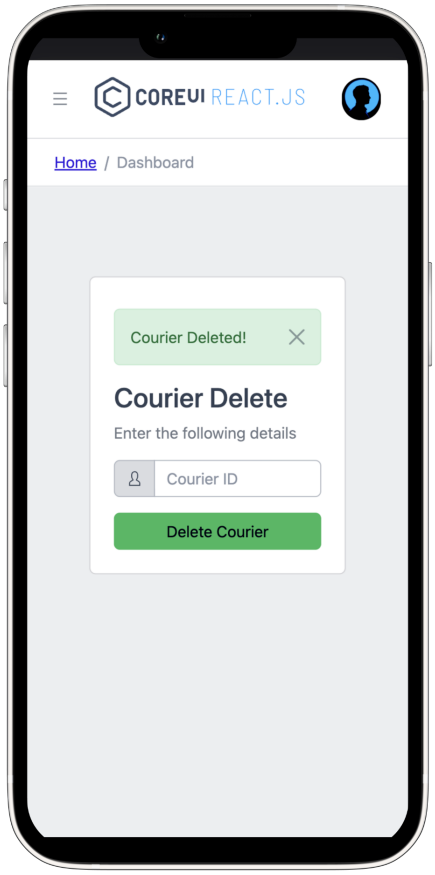
Adding Courier:

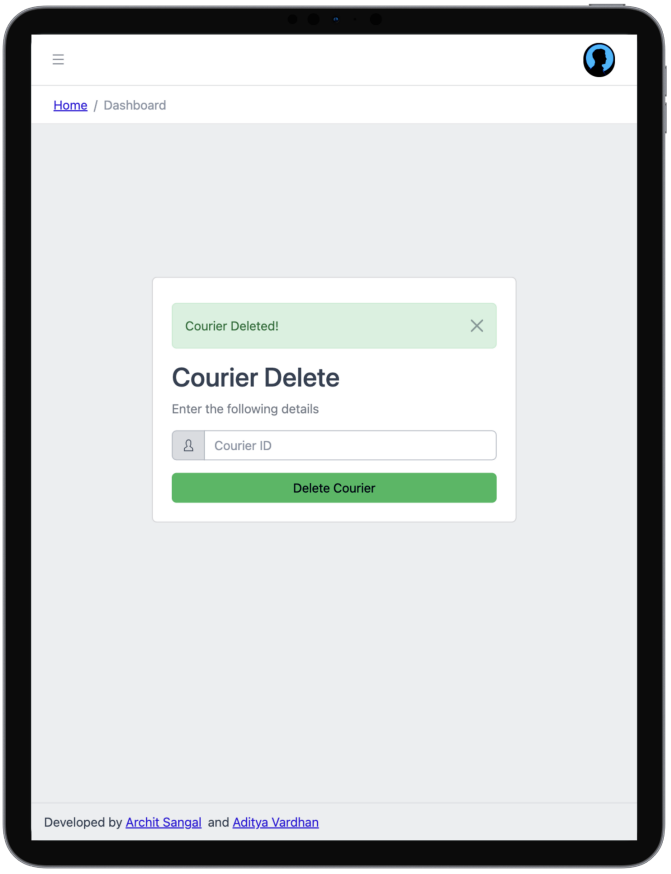


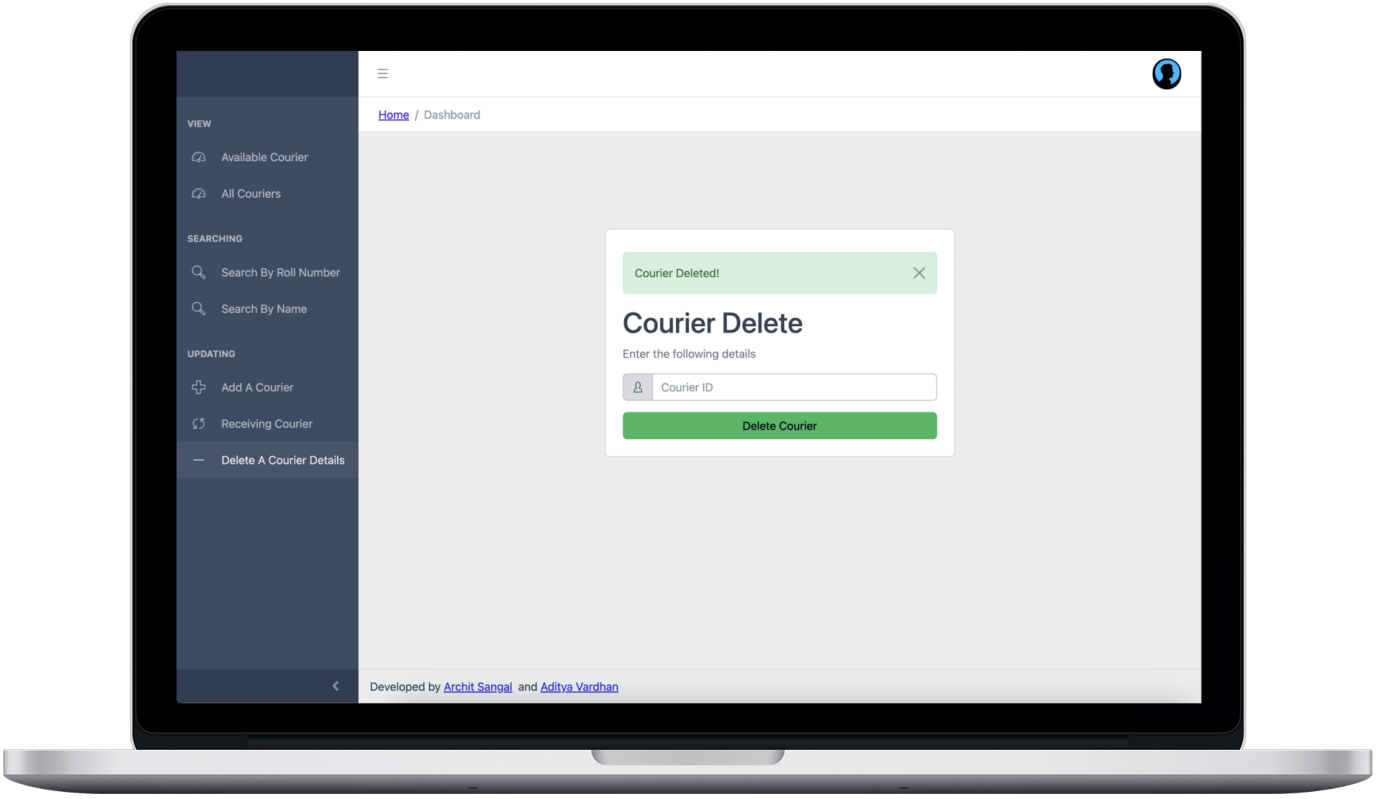
Receiving Courier:



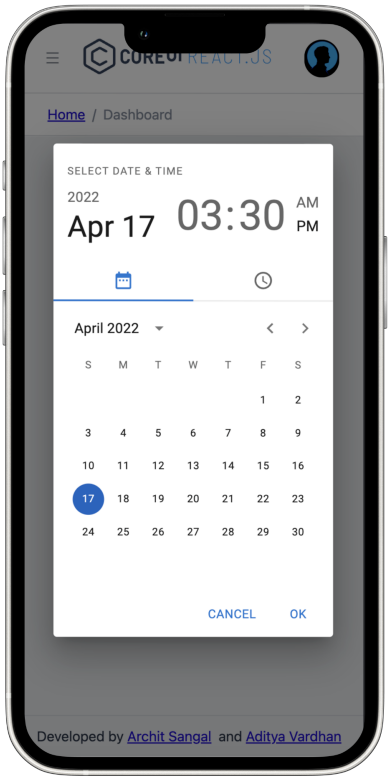
Delete Couriers:

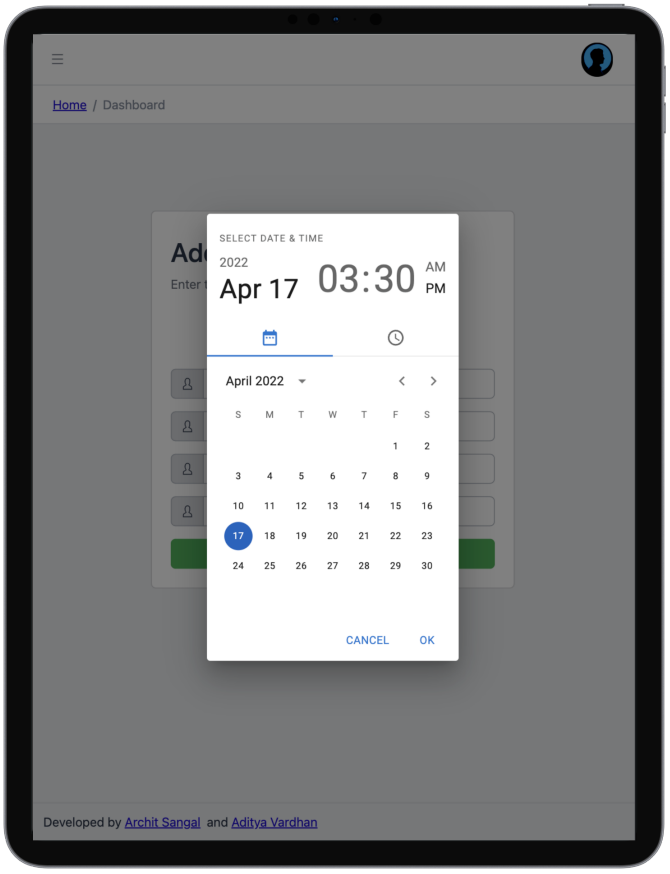


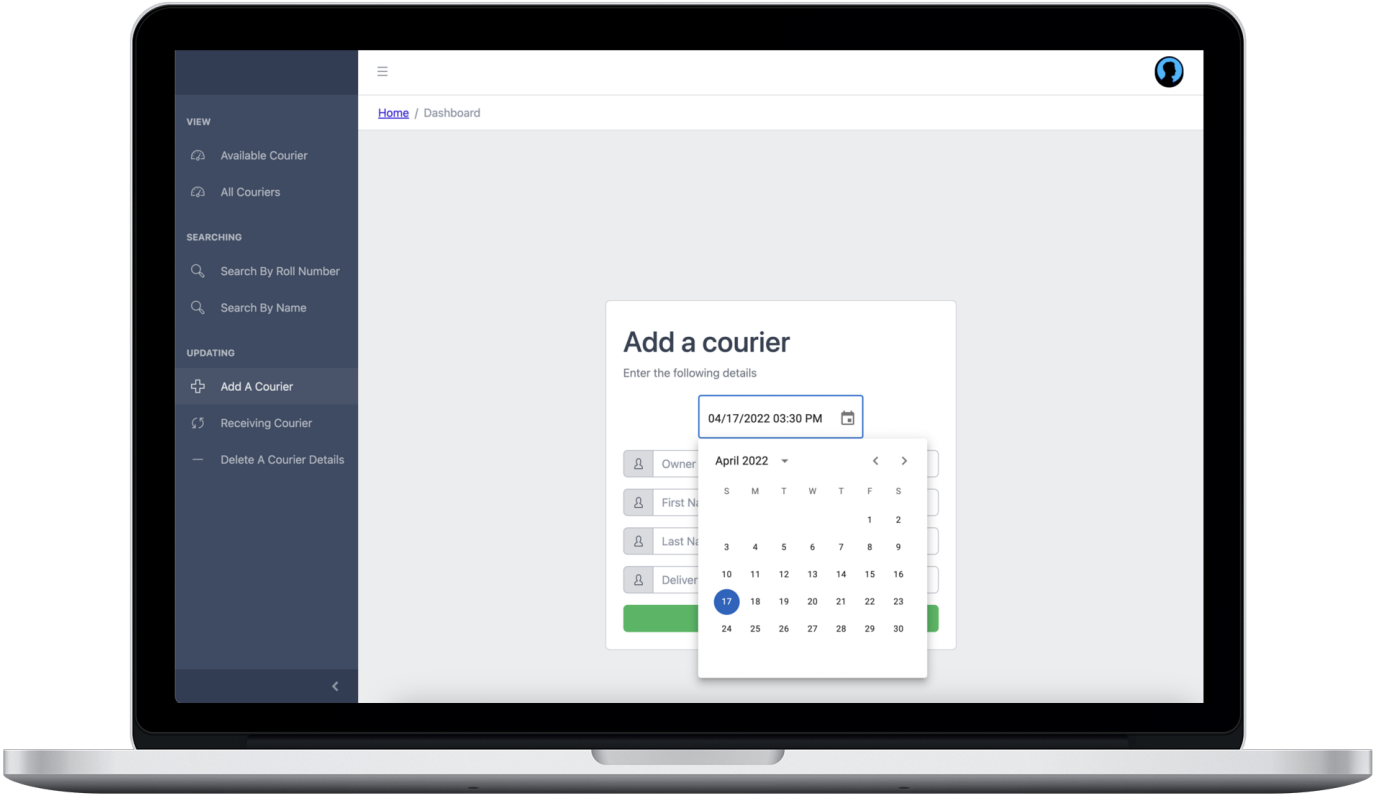




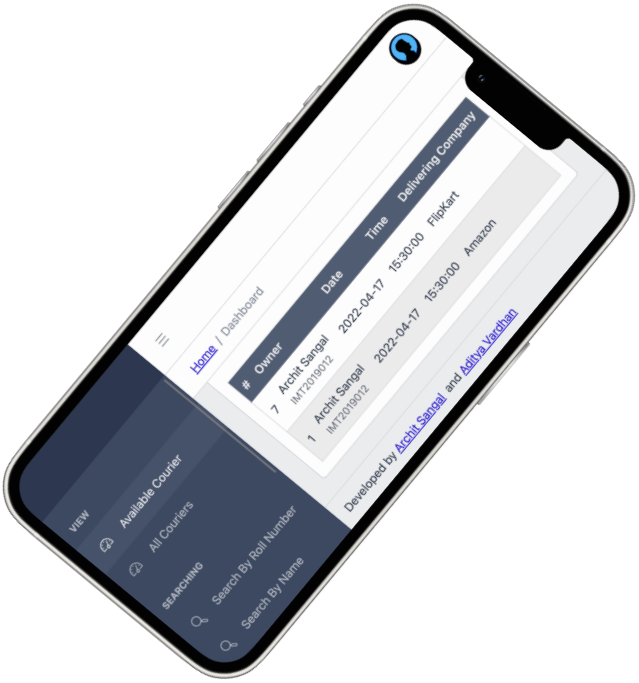
Responsive Calendar:

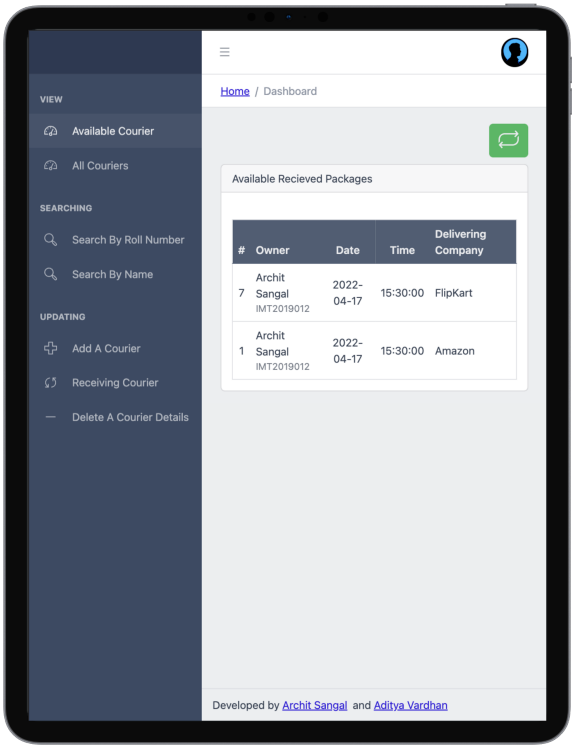


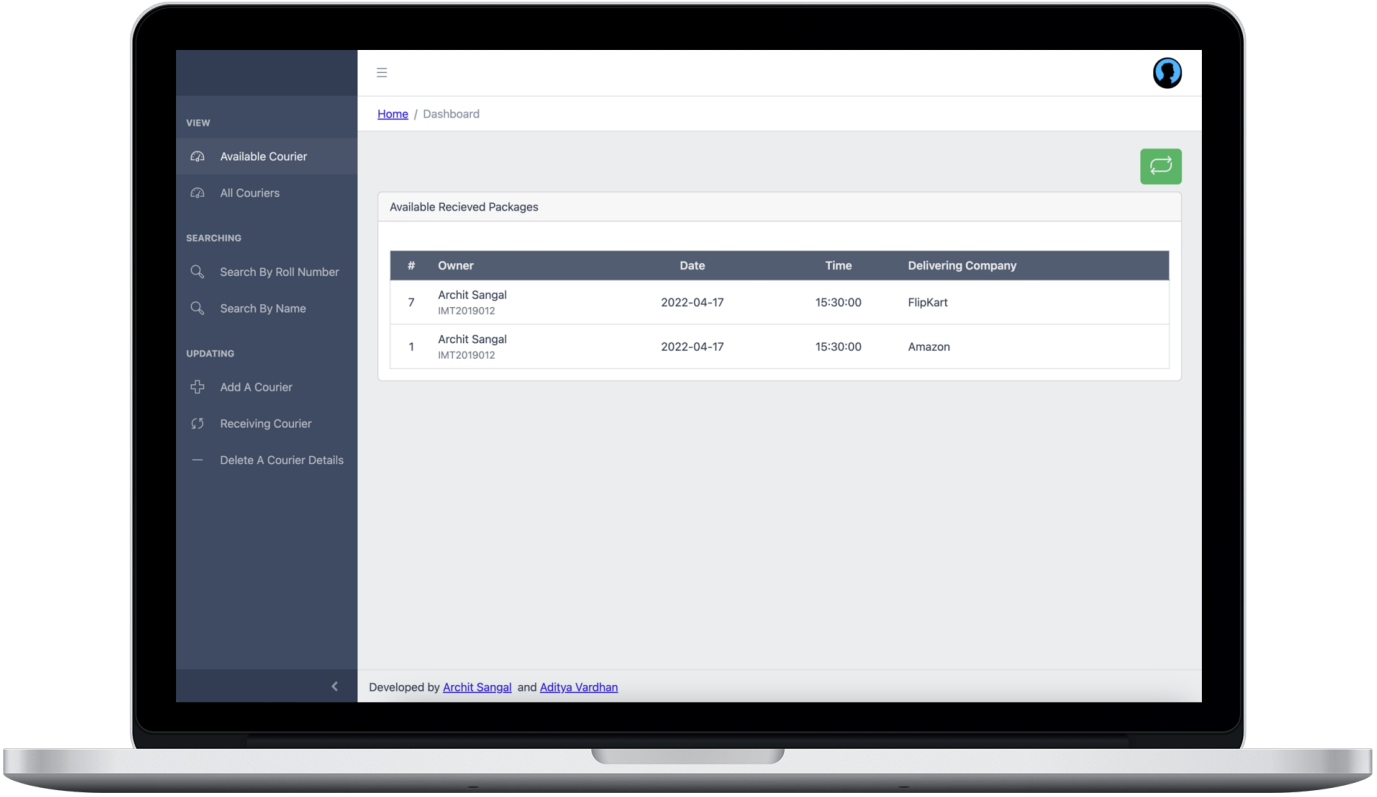




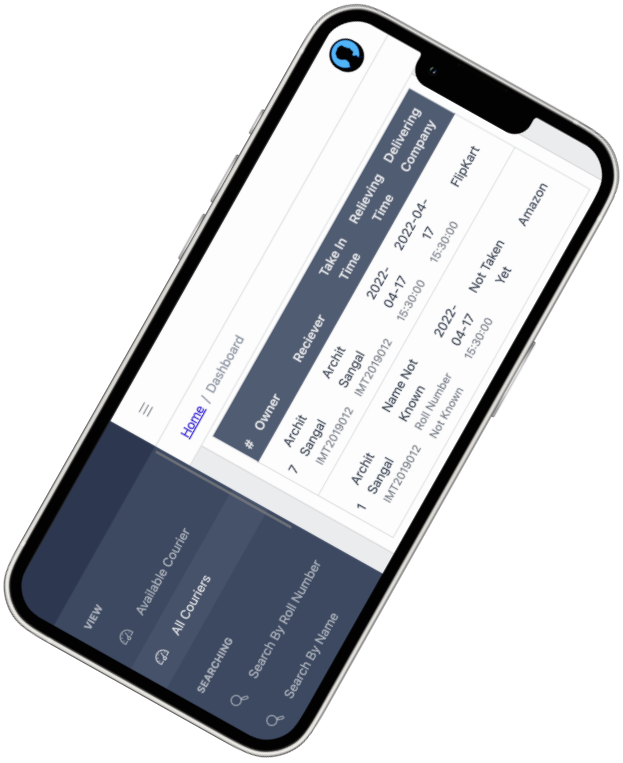
Viewing Available Couriers:

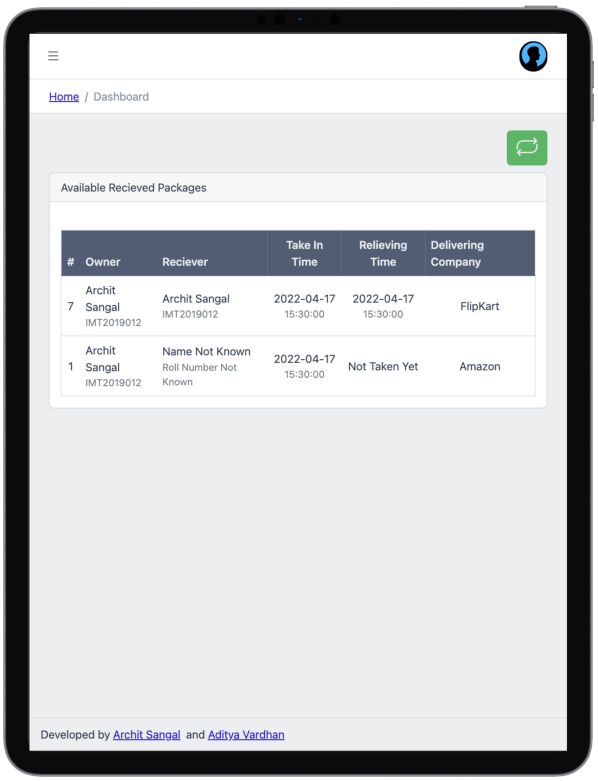


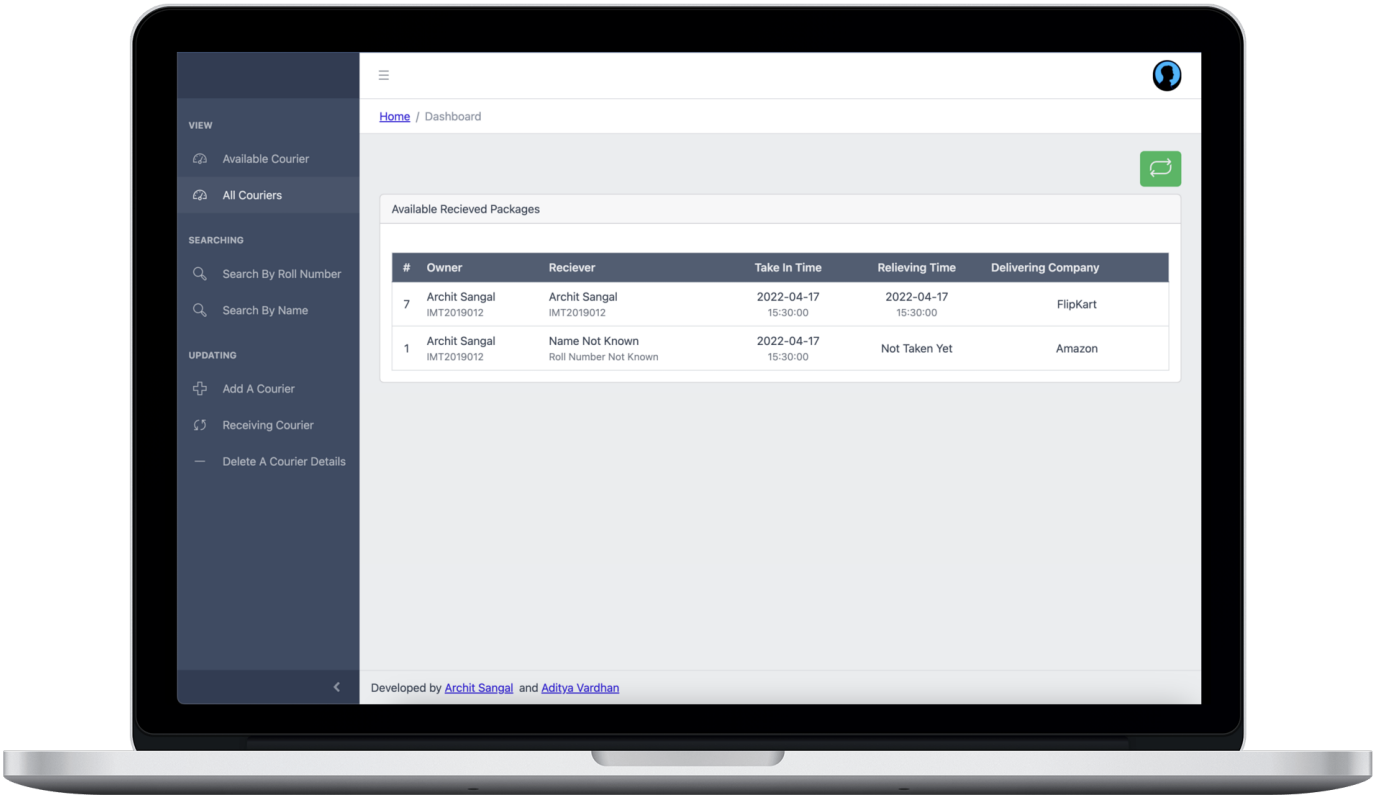




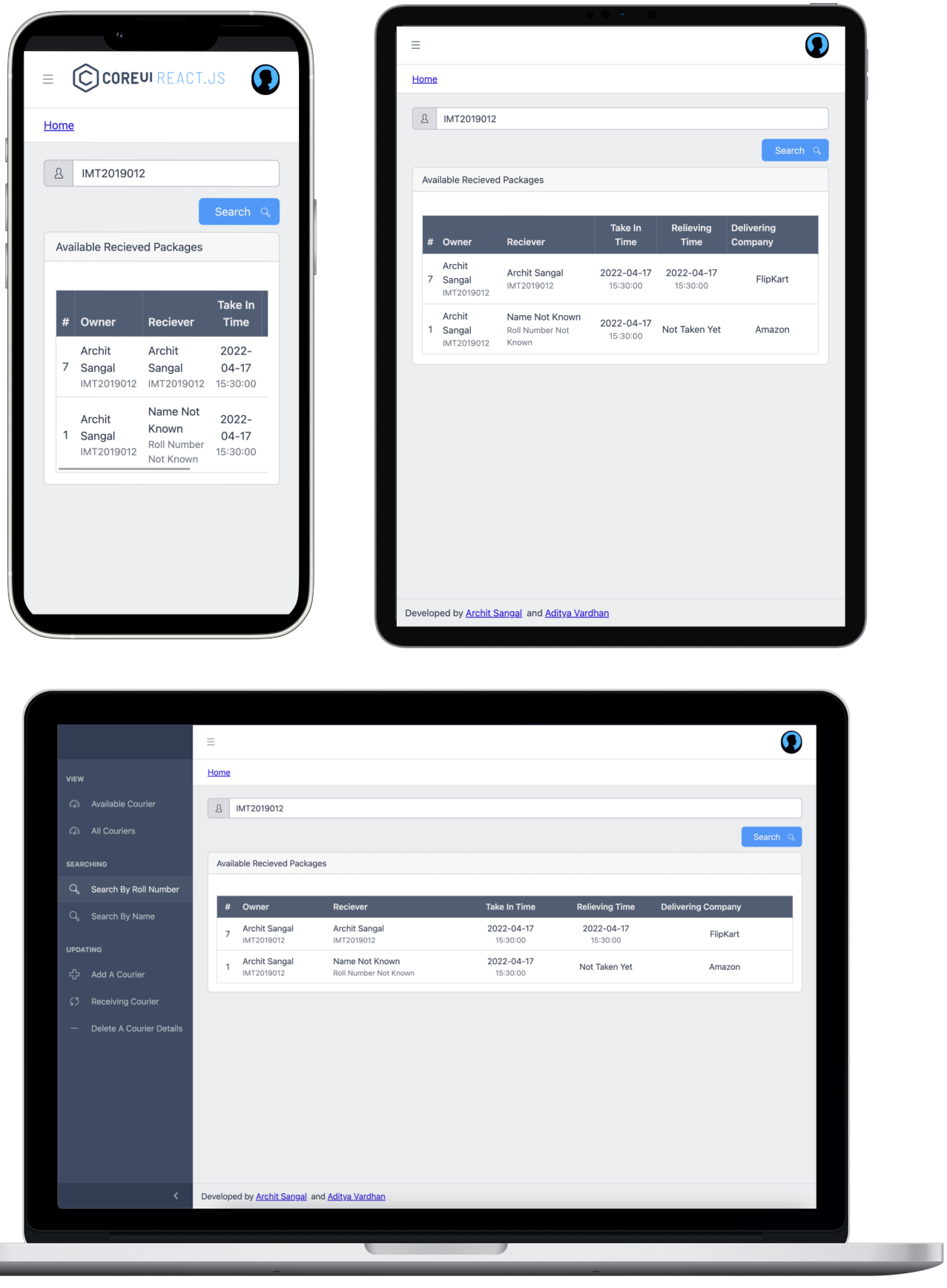
Viewing All Couriers:



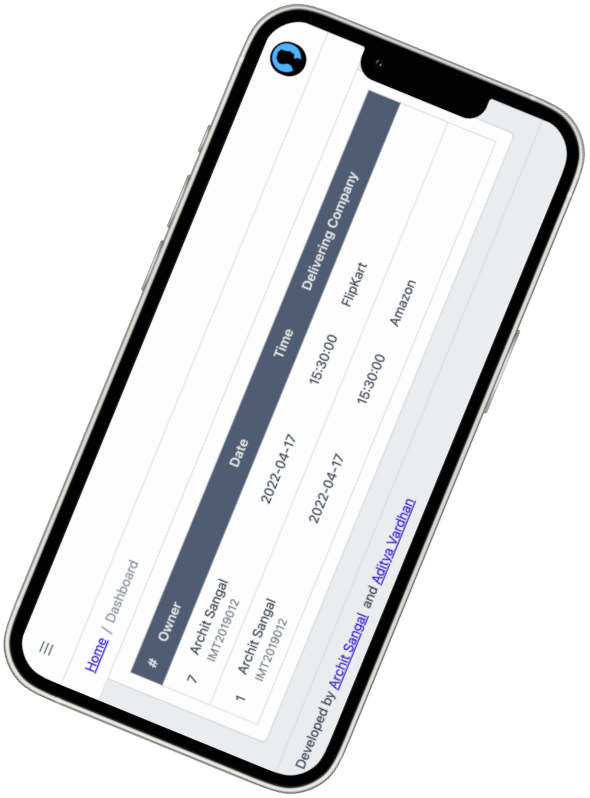


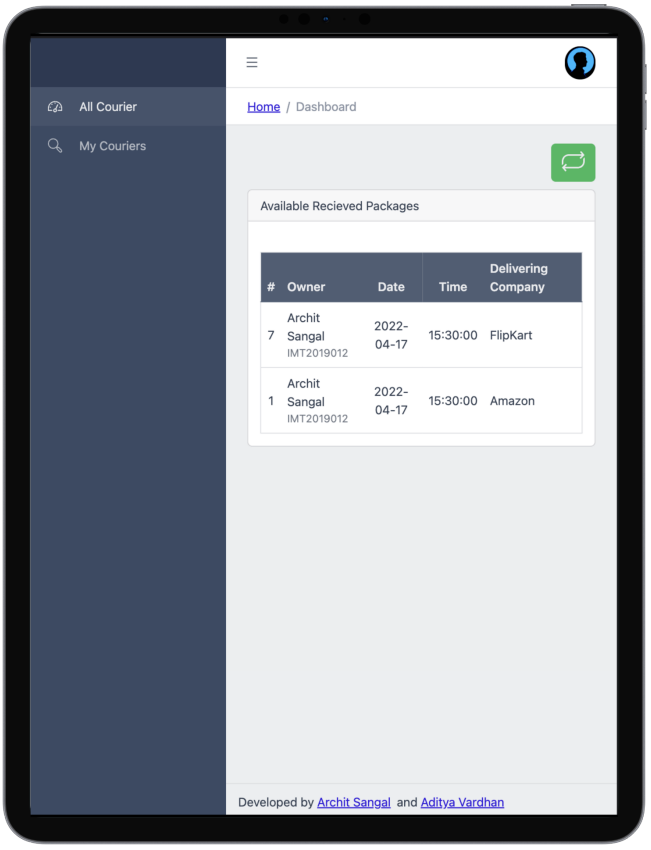


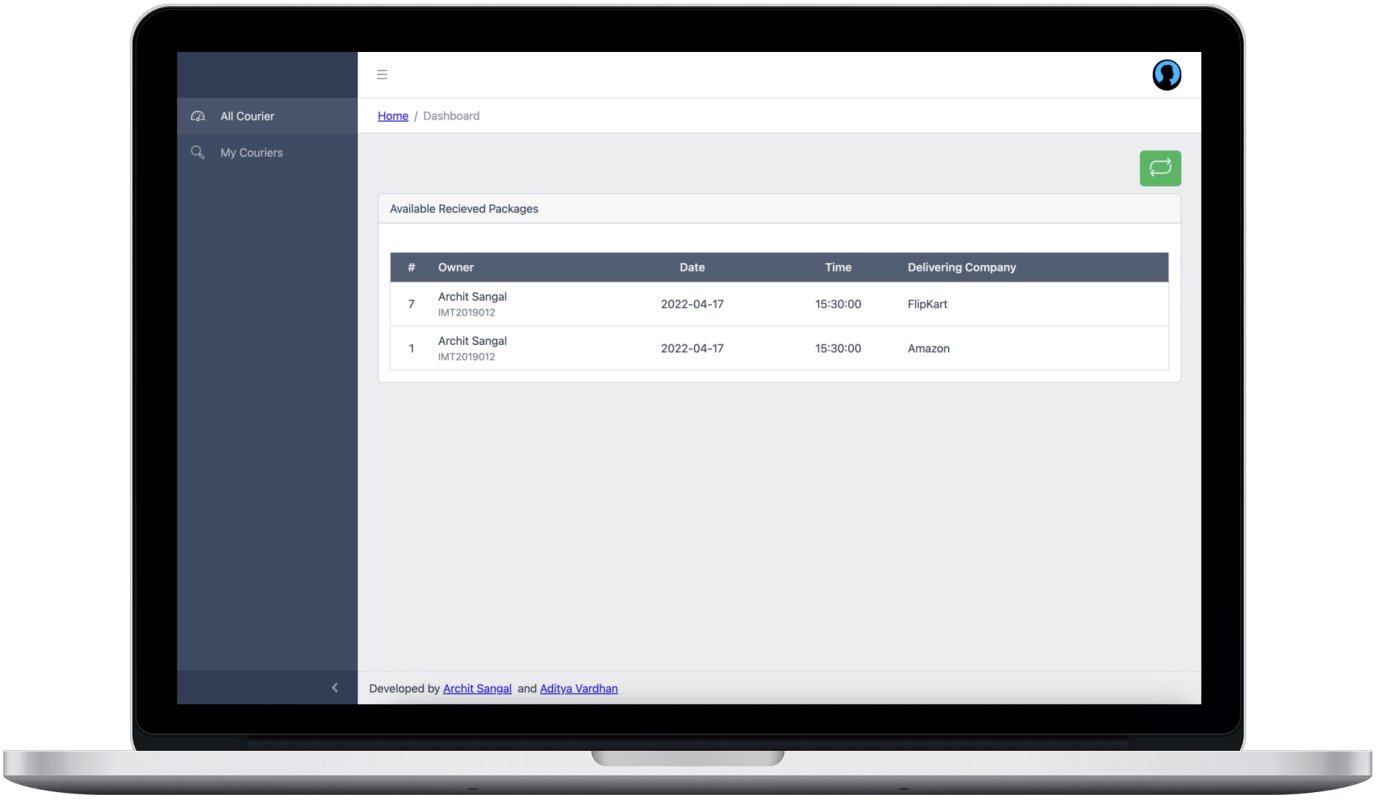
Searching By Roll Number or By Name:



Users View:


Home / Dashboard
Available Recieved Packages
Owner Date Time Delivering Company
7 Archit Sangal IMT2019012 2022-04-17 15:30:00 FlipKart
1 Archit Sangal IMT2019012 2022-04-17 15:30:00 Amazon
Developed by Archit Sangal and Aditya Vardhan


All Courier
My Couriers
Home / Dashboard
Available Recieved Packages
Owner Date Time Delivering Company
7 Archit Sangal IMT2019012 2022-04-17 15:30:00 FlipKart
1 Archit Sangal IMT2019012 2022-04-17 15:30:00 Amazon
Developed by Archit Sangal and Aditya Vardhan


All Courier
My Couriers
Home / Dashboard
Available Recieved Packages
Owner Date Time Delivering Company
7 Archit Sangal IMT2019012 2022-04-17 15:30:00 FlipKart
1 Archit Sangal IMT2019012 2022-04-17 15:30:00 Amazon
Developed by Archit Sangal and Aditya Vardhan

Backend

Installations:

MySQL:

MySQL is an open-source relational database management system that uses SQL (Structured Query Language) to manage and manipulate data. It is widely used in web applications as a backend database and can handle large amounts of data efficiently. MySQL is known for its speed, reliability, and ease of use, making it a popular choice for developers and businesses alike.

Before running the app, create a mysql database with the name “iiitb_couriers”.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| hive               |
| iiitb_couriers     |
| information_schema |
| metastore          |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
7 rows in set (0.00 sec)
```

Dependencies used:

1. A collection of pre-configured requirements are provided by the Spring Boot Starter Web dependency when creating web applications using Spring Boot. It contains the Spring MVC framework, which offers assistance for using the Model-View-Controller pattern to create online applications. It is simple to execute your application without the requirement for an external web server thanks to the embedded web server provided by the Spring Boot Starter Web dependency, which contains Tomcat. We use Tomcat for our project.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.4.5</version>
</dependency>
```

2. JUnit is a dependency that provides a testing framework for Java applications. It allows developers to write and run automated tests to verify the correctness of their code. JUnit includes several annotations and assertions that make it easy to define test cases and expected results.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
</dependency>
```

3. Spring Boot Starter Data JPA is a dependency that gives you a set of pre-configured libraries for making JPA-based data access layers in Spring Boot applications. It comes with the Hibernate ORM technology, which lets you map Java objects to tables in a relational database. The Spring Boot Starter Data JPA dependency also helps with database migrations, transaction management, and other standard data access tasks. By making this a project dependency, you can quickly set up a JPA-based data access layer without having to do a lot of setup or write a lot of boilerplate code.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.5.13</version>
</dependency>
```

4. MySQL Connector/J is a JDBC driver for connecting Java applications to MySQL databases. It allows Java developers to connect to MySQL servers and perform database operations using JDBC API.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.25</version>
</dependency>
```

5. Spring Boot Starter Security is a dependency that provides a set of pre-configured libraries for building secure web applications with Spring Boot. It includes support for authentication, authorization, and other security-related features.

   JJWT (Java JSON Web Tokens) is a dependency that provides a way to create, sign, and verify JSON web tokens. It is often used in combination with Spring Boot Starter Security to implement secure authentication and authorization. By including these dependencies in your project, you can quickly set up a secure web application with minimal configuration.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>2.5.13</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

6. Spring Boot Starter Mail is a dependency that provides a set of pre-configured libraries for sending email messages in Spring Boot applications. It includes support for popular email protocols, such as SMTP, and provides an easy-to-use API for sending email messages from your application.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

7. Mockito Core is a dependency that gives Java apps a way to test themselves. It lets writers make fake objects and fake methods to use for testing. The Mockito Core dependency includes a set of APIs and utilities for building and working with mock objects, as well as support for verifying and validating method calls.

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
</dependency>
```

Testing with mockito and junit:

Note that the only 2 methods which made sense to test in our project, were “addCourier()” and “updateCourier()” methods. Rest of the methods were either mostly using inbuilt functions or were too simple to test.

```
@ExtendWith(MockitoExtension.class)
class CourierServiceTest {

    2 usages
    @InjectMocks
    private CourierService courierService;

    3 usages
    @Mock
    private CourierDao courierDao;

    no usages
    @Mock
    private UserDao userDao;

    no usages  vardhan2000 *
    @BeforeEach
    void setUp() {
        MockitoAnnotations.initMocks( testClass: this);
    }
```

We use the Mockito Core dependency to create mock objects of the CourierDao and UserDao interfaces, which are dependencies of the CourierService.

The @ExtendWith(MockitoExtension.class) annotation is used to enable the Mockito JUnit extension, while the @InjectMocks and @Mock annotations are used to inject the mock objects into the CourierService instance and the test class, respectively.

The setUp() method is used to initialize the mock objects using the MockitoAnnotations.initMocks(this) method

```
@Test
void TestAddCourier() {
    Courier courier = new Courier();
    courier.setOwner("Aditya Vardhan");
    courier.setCourierCompany("Amazon");
    courier.setArrivalDate(Date.valueOf( s: "2023-04-14"));
    courier.setArrivalTime(Time.valueOf( s: "19:50:30"));
    courier.setOwnerRollNo("IMT2019003");

    courierService.addCourier(courier);
    verify(courierDao).save(courier);
}
```

The TestAddCourier() method is a test case that creates a new Courier instance, adds it to the CourierService using the addCourier() method, and then verifies that the save() method of the CourierDao interface is called with the Courier instance as an argument using the verify() method.

```
@Test
void updateCourier() {
    Courier courier = new Courier();
    courier.setOwner("Aditya Vardhan");
    courier.setCourierCompany("Amazon");
    courier.setArrivalDate(Date.valueOf("2023-04-14"));
    courier.setArrivalTime(Time.valueOf("19:50:30"));
    courier.setOwnerRollNo("IMT2019003");
    courier.setCourierID(1);

    Courier cour = new Courier();
    cour.setCourierID(1);
    cour.setReceiverName("Archit Sangal");
    cour.setReceiverRollNo("IMT2019012");
    cour.setDeliverDate(Date.valueOf("2023-04-14"));
    cour.setDeliverTime(Time.valueOf("22:26:00"));
    cour.setStatus("COMPLETE");

    Courier c = new Courier();
    c.setOwner("Aditya Vardhan");
    c.setCourierCompany("Amazon");
    c.setArrivalDate(Date.valueOf("2023-04-14"));
    c.setArrivalTime(Time.valueOf("19:50:30"));
    c.setOwnerRollNo("IMT2019003");
    c.setCourierID(1);
    c.setReceiverName("Archit Sangal");
    c.setReceiverRollNo("IMT2019012");
    c.setDeliverDate(Date.valueOf("2023-04-14"));
    c.setDeliverTime(Time.valueOf("22:26:00"));
    c.setStatus("COMPLETE");


    doReturn(Optional.of(cour)).when(courierDao).findById(1);
    when(courierDao.save(eq(cour))).thenReturn(c);
    Courier result = courierService.updateCourier(cour);
      System.out.println(result);
    assertEquals(result,c);
}
```

The CourierDao interface's findById() and save() methods are stubbed with the doReturn() and when() methods, respectively. The updated courier is then passed to the CourierService's updateCourier() method. The assertEquals() method is used to compare the result to what was intended. The Optional class is used in

the above code to handle the possibility of a null value being returned by the findById() method of the CourierDao interface.

Overall, this test method makes sure that the updateCourier() method of the CourierService class uses the CourierDao interface to update the courier in the database properly.

App Functionalities:

1. **CRUD functionalities:**
    a. Add courier: the admin can create a new courier instance and save it to the database.

    b. Update courier: the admin can update the details of an existing courier in the database.

    c. Delete courier: the admin can delete the details of an existing courier from the database.

    d. Read: the admin can read the details of the couriers in the database. The user can also do so but not for all couriers, only for the couriers that belong to him/her.

2. **Additional functionalities:**

    a. Email verification: A newly registered user has to verify their email by entering the OTP that will be sent to their email-id, when they signUp.

    b. Login: A user can login using their username and password. They will be given a jwt token which they can then use to use the protected functionalities (protected in terms of who can use them).

    c. Forgot password: if a user has forgotten their password, they can use this functionality to get a secured OTP to their registered mail, which they can then use to reset their password.

    d. Authorization: There are 2 roles defined in our project, namely ‘admin’ and ‘user’. Each functionality has an authorization check i.e. only a person with a certain role can use them. For example, add, delete and update couriers can only be done by someone with an ‘admin’, whereas a reset password API can only be called by someone with a ‘user’ role.

    e. Email notification: Everytime a new courier is added or updated, a notification email will be sent to the owner of the courier, if they are registered in the database.

    f. Search functionality: an admin can search the database for couriers by using the name or roll number fields. Note that the admin doesn’t need to type the whole name or roll number. Our search functionality is capable of performing partial search i.e. searching by use of substrings.

API Documentation

For now, we’ll look at the API(s) with domain name (“localhost:9090”). To customise the domain name, refer to later sections.

Available API(s) for admin:

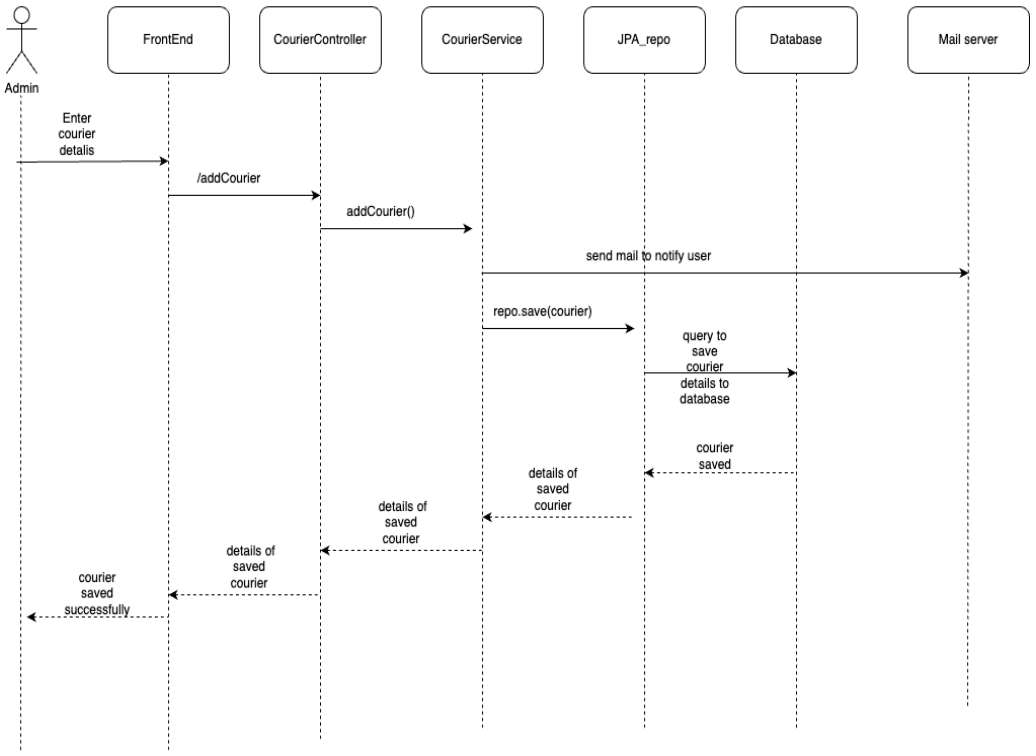
1. Add Courier:
    a. Endpoint: *localhost:9090/addCourier*
    b. HTTP method: POST
    c. Descriptions: It will add a new courier to the database.
    d. Request body :


```
{
    "owner":"Aditya Vardhan",
    "courierCompany":"Amazon",
    "arrivalDate":"2023-04-14",
    "arrivalTime":"19:50:30",
    "ownerRollNo":"IMT2019003"
}
```


    e. Response body :


```
{
    "courierID": 2,
    "owner": "Aditya Vardhan",
    "courierCompany": "Amazon",
    "arrivalDate": "2023-04-14",
    "arrivalTime": "19:50:30",
    "ownerRollNo": "IMT2019003",
    "receiverName": null,
    "receiverRollNo": null,
    "deliverDate": null,
    "deliverTime": null,
    "status": "ACTIVE"
}
```


    f. UML diagram:



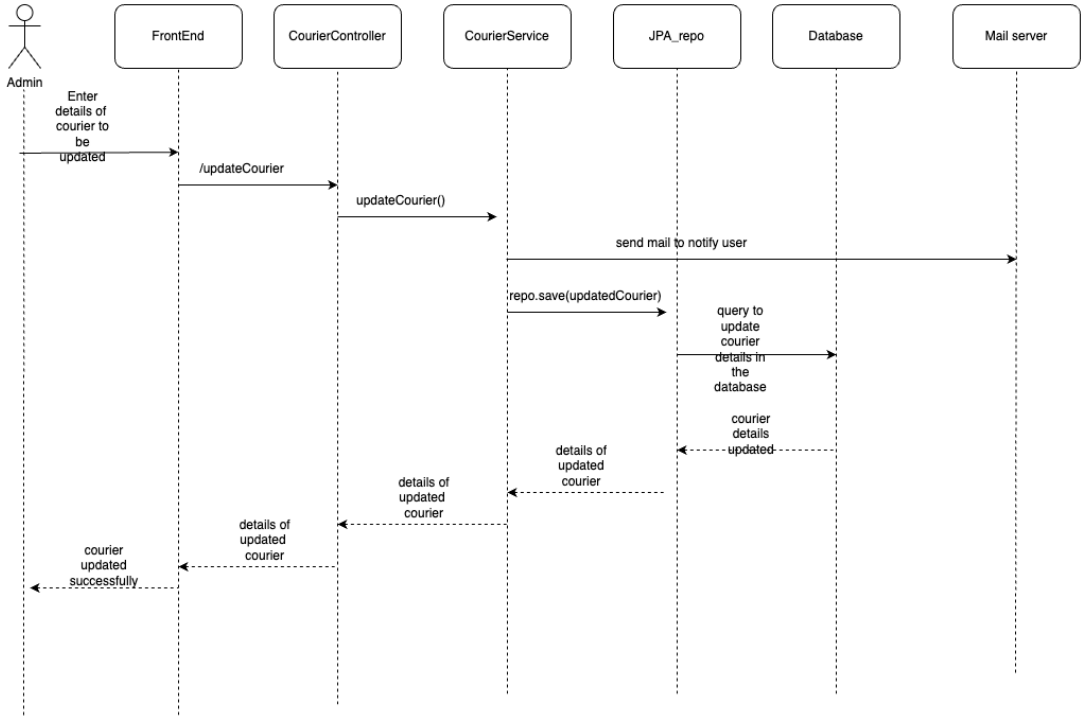
2. Update courier:
    a. Endpoint: *localhost:9090/updateCourier*
    b. HTTP method: POST
    c. Descriptions: It will update details of an existing courier in the database.
    d. Request body :


```
"courierID":"6",
"receiverName":"Archit Sangal",
"receiverRollNo":"IMT2019012",
"deliverDate":"2023-04-14",
"deliverTime":"22:26:00",
"status":"COMPLETE"
```

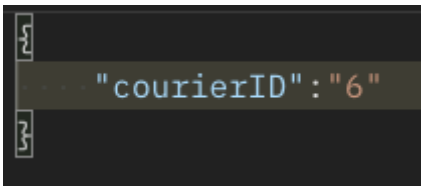

    e. Response body :


```
"courierID": 6,
"owner": "Aditya Vardhan",
"courierCompany": "Amazon",
"arrivalDate": "2023-04-14",
"arrivalTime": "19:50:30",
"ownerRollNo": "IMT2019003",
"receiverName": "Archit Sangal",
"receiverRollNo": "IMT2019012",
"deliverDate": "2023-04-14",
"deliverTime": "22:26:00",
"status": "COMPLETE"
```


    f. UML diagram:

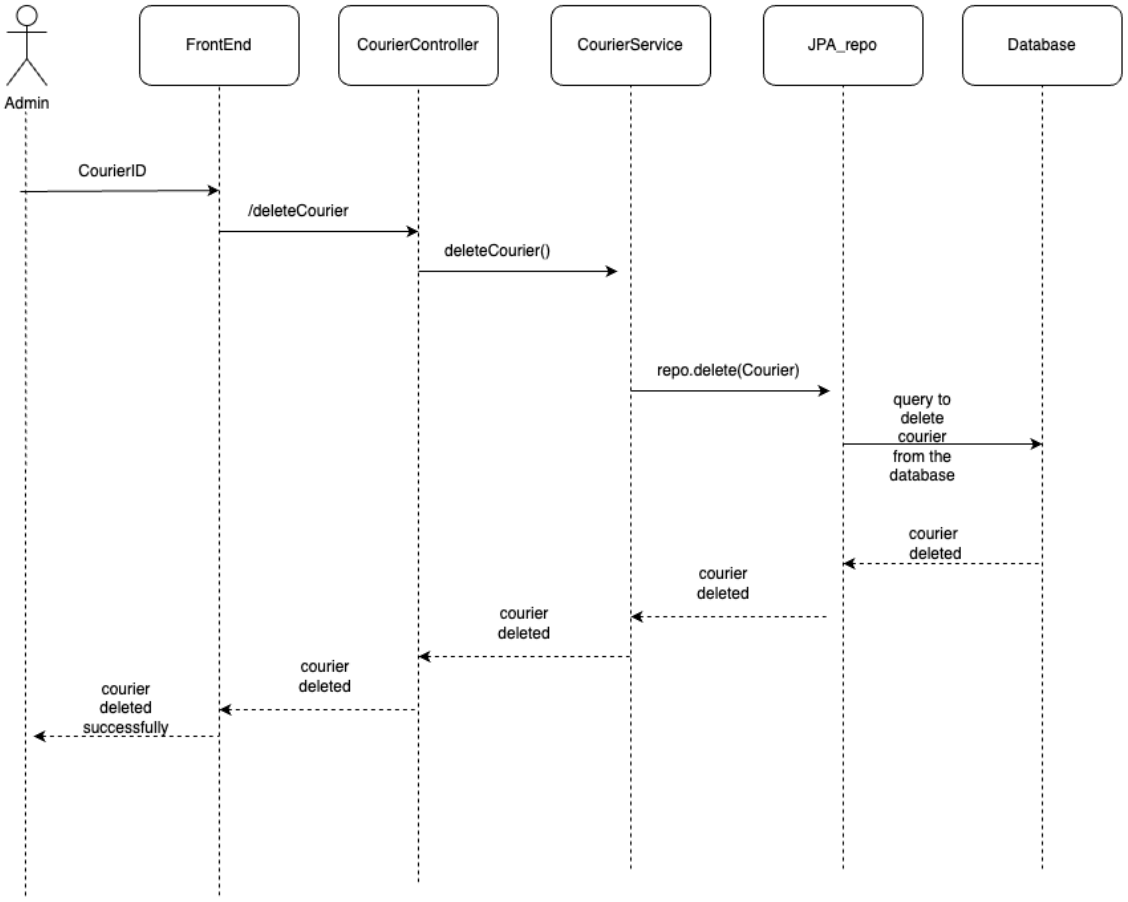


3. Delete courier:
    a. Endpoint: *localhost:9090/deleteCourier*
    b. HTTP method: POST
    c. Descriptions: It will delete the details of an existing courier in the database.
    d. Request body :



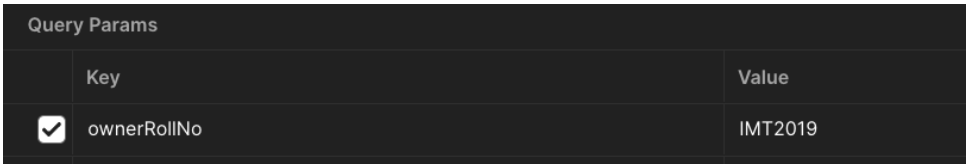
    e. Response body : none

    f. UML diagram:



4. Get couriers by roll number:
    a. Endpoint: *localhost:9090/getCouriersByRollNo?ownerRollNo=<rollNo>*
    b. HTTP method: GET
    c. Descriptions: it will get the couriers, whose ownerRollNo attribute ‘contain’ the given roll number, from the database. Note that it can perform partial search.

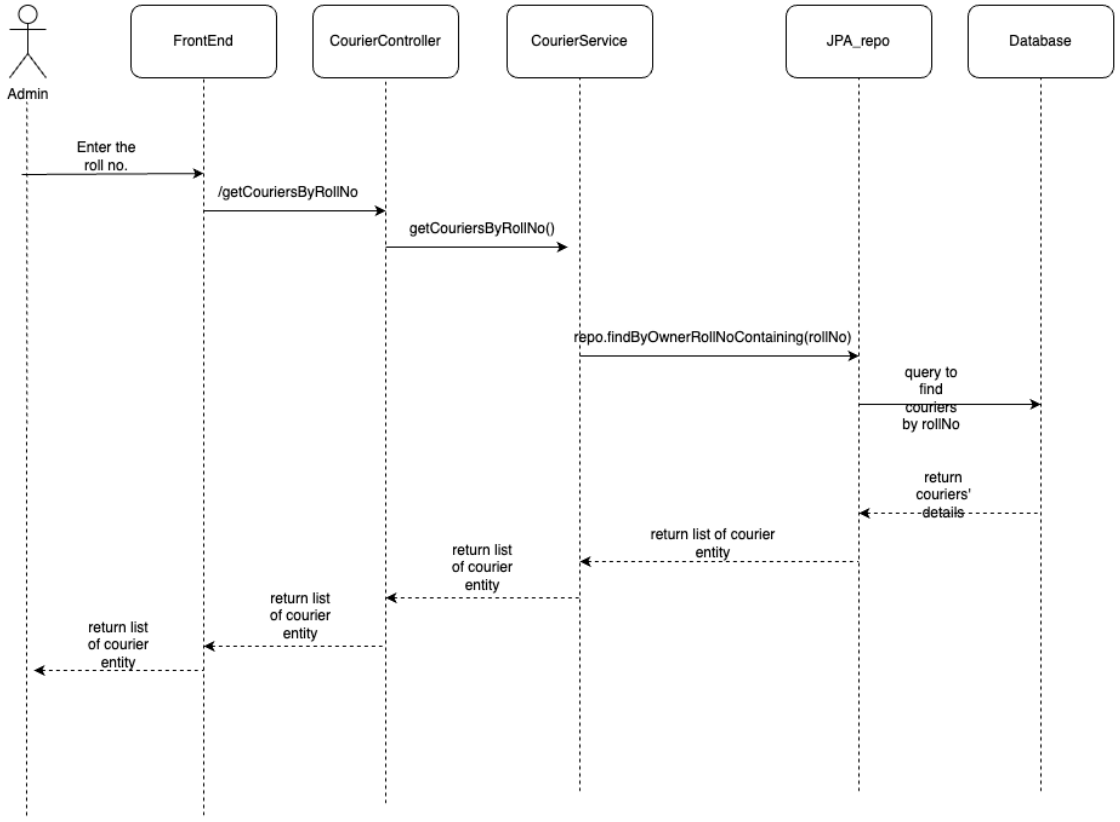
    d. Request parameters:



e. Response body: list of couriers


```
[
    {
        "courierID": 7,
        "owner": "Aditya Vardhan",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-14",
        "arrivalTime": "19:50:30",
        "ownerRollNo": "IMT2019003",
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    }
]
```


f. UML diagram:

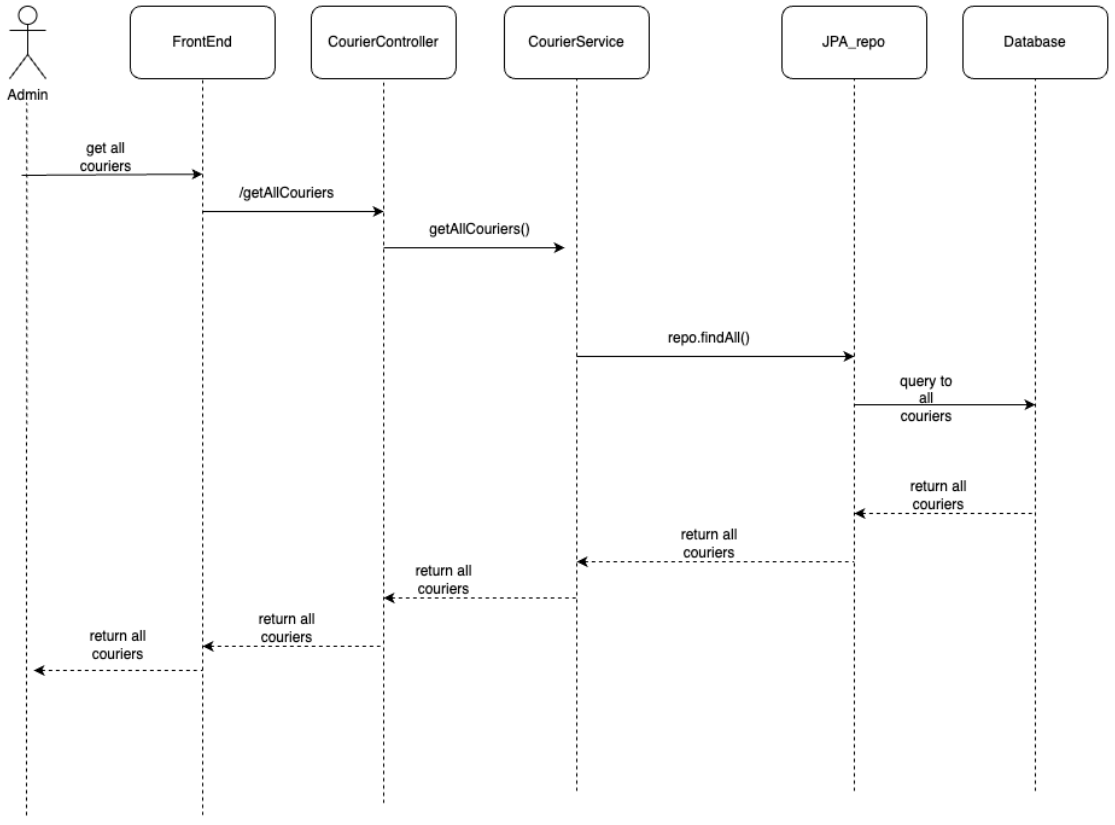


5. Get all couriers:
    a. Endpoint: *localhost:9090/getAllCouriers*
    b. HTTP method: GET
    c. Descriptions: gets all the couriers from the database.
    d. Request body : none. It makes use of the jwt token to just check authorisation.

    e. Response body :

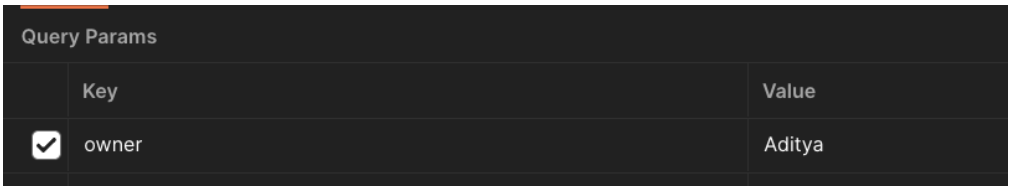
```
[
    {
        "courierID": 5,
        "owner": "Aditya Kaka",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-15",
        "arrivalTime": "19:50:30",
        "ownerRollNo": null,
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    },
    {
        "courierID": 3,
        "owner": "Aditya Kaka",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-15",
        "arrivalTime": "19:50:30",
        "ownerRollNo": null,
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    },
    {
        "courierID": 4,
        "owner": "Aditya Vardhan",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-14",
        "arrivalTime": "19:50:30",
        "ownerRollNo": "IMT2019003",
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    }
]
```


f. UML diagram:



6. Get couriers by name:
    a. Endpoint: *localhost:9090/getCouriersByName?owner=<name>*
    b. HTTP method: GET
    c. Descriptions: it will get the couriers, whose owner attribute ‘contain’ the given name, from the database. Note that it can perform partial search.

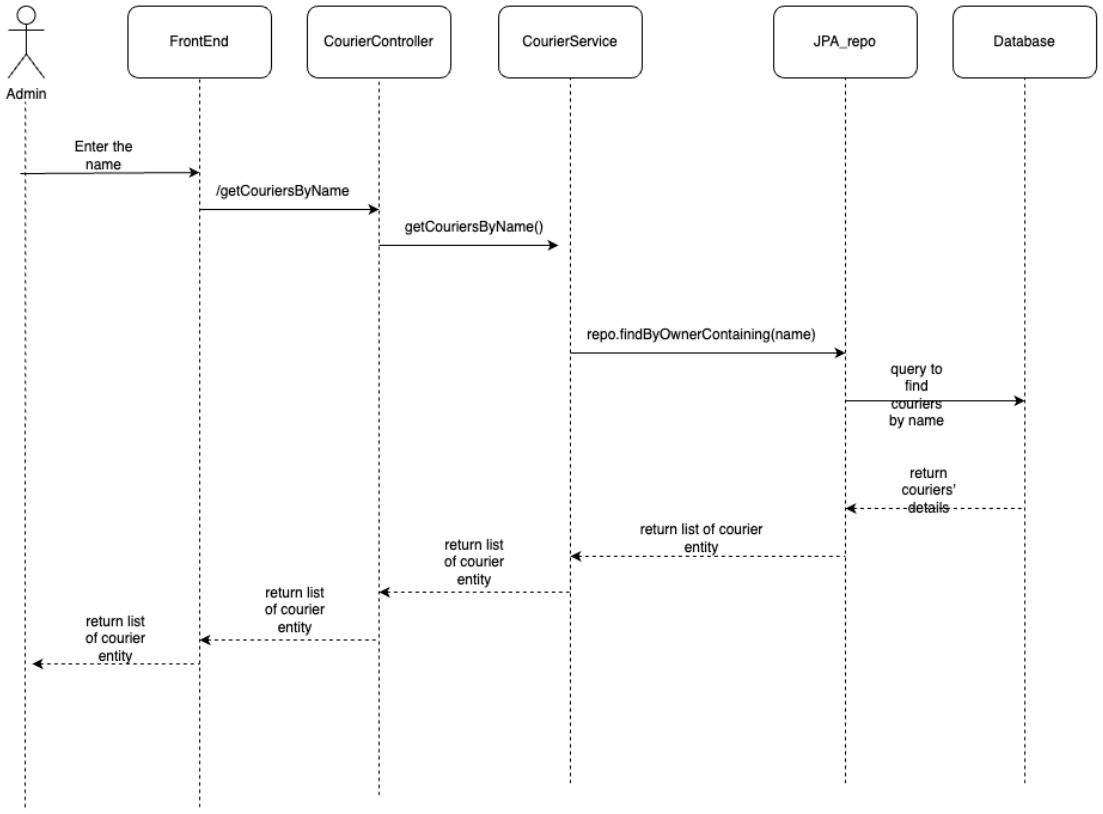
d. Request parameters:



e. Response body :


```
[
    {
        "courierID": 3,
        "owner": "Aditya Kaka",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-15",
        "arrivalTime": "19:50:30",
        "ownerRollNo": null,
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    },
    {
        "courierID": 5,
        "owner": "Aditya Kaka",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-15",
        "arrivalTime": "19:50:30",
        "ownerRollNo": null,
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    },
    {
        "courierID": 7,
        "owner": "Aditya Vardhan",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-14",
        "arrivalTime": "19:50:30",
        "ownerRollNo": "IMT2019003",
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    }
]
```

f. UML diagram:



Available API(s) for users:

1. Sign up:
    a. Endpoint: *localhost:9090/registerNewUser*
    b. HTTP method: POST
    c. Descriptions: it will add a new user to the database and send an OTP via mail to verify the input mail-id.

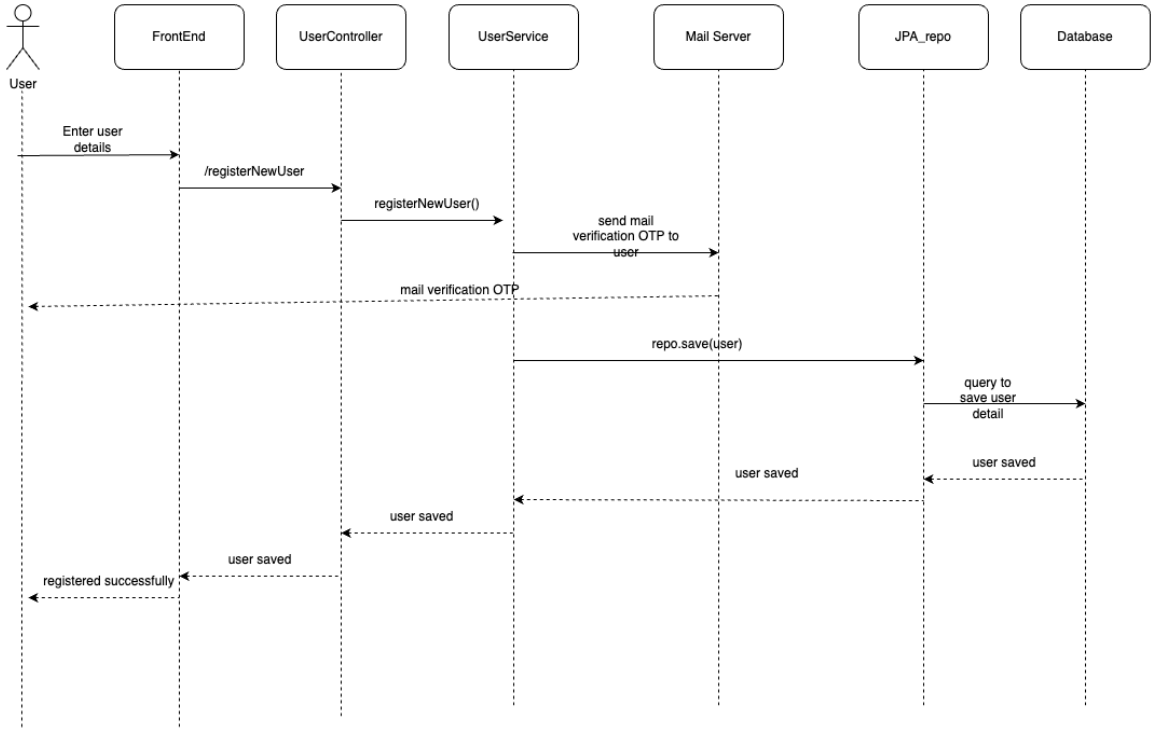
    d. Request body :

```
{
    "userName":"IMT2019003",
    "userFirstName":"Aditya",
    "userLastName":"Vardhan",
    "userPassword":"1234",
    "mailID":"vardhan.av2000@gmail.com"
}
```

    e. Response body : Note the password the otp values are exclusively set to null while returning the object for security reasons.

```
"userName": "IMT2019003",
"userFirstName": "Aditya",
"userLastName": "Vardhan",
"userPassword": null,
"role": [
    {
        "roleName": "User",
        "roleDescription": "Default role for newly created record"
    }
],
"mailID": "vardhan.av2000@gmail.com",
"status": "NOT_VERIFIED",
"otp": null
```


f. UML diagram:



2. Verify mail:
    a. Endpoint: *localhost:9090/verifyMail/?mailId=<mail>&OTP=<otp>*
    b. HTTP method: POST
    c. Descriptions: it will verify the mail-id of the user by matching the entered OTP with the saved OTP in the database.

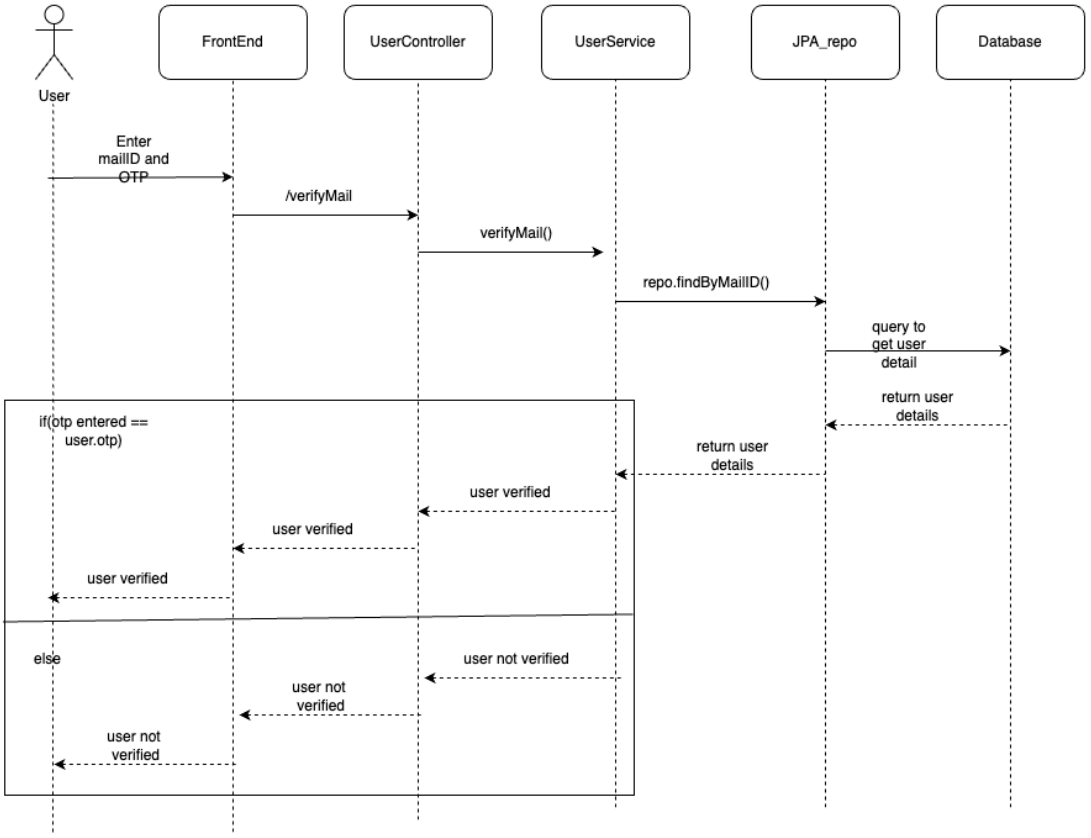
    d. Request parameters :

Query Params		
	Key	Value
☑	mailId	vardhan.av2000@gmail.com
☑	OTP	177824
	Key	Value

    e. Response body: note that the password is encoded for security reasons.

```
"userName": "IMT2019003",
"userFirstName": "Aditya",
"userLastName": "Vardhan",
"userPassword": "$2a$10$S78f3jv.LDzaPSGXApcH2O8C6CNUqs1TA4TqD7VWEJ/CPPoIdPEG6",
"role": [
    {
        "roleName": "User",
        "roleDescription": "Default role for newly created record"
    }
],
"mailID": "vardhan.av2000@gmail.com",
"status": "VERIFIED",
"otp": "177824"
```


f. UML diagram:



3. Login:
    a. Endpoint: *localhost:9090/authenticate*
    b. HTTP method: POST
    c. Descriptions: it will generate and return a valid jwt token, if the user enters correct credentials.

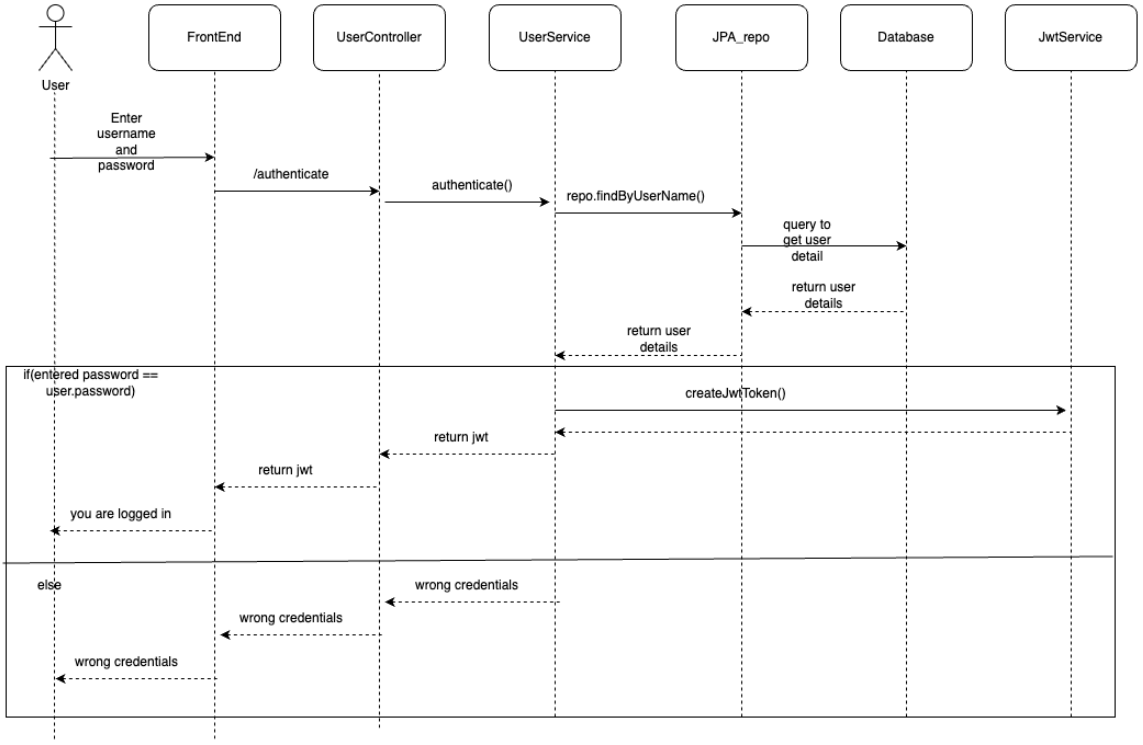
    d. Request body :


```
"userName":"IMT2019003",
"userPassword":"1234"
```


    e. Response body :

```
{
    "user": {
        "userName": "IMT2019003",
        "userFirstName": "Aditya",
        "userLastName": "Vardhan",
        "userPassword": "$2a$10$S78f3jv.LDzaPSGXApcH2O8C6CNUqs1TA4TqD7VWEJ/CPPoIdPEG6",
        "role": [
            {
                "roleName": "User",
                "roleDescription": "Default role for newly created record"
            }
        ],
        "mailID": "vardhan.av2000@gmail.com",
        "status": "VERIFIED",
        "otp": "177824"
    },
    "jwtToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJJTVQyMDE5MDAzIiwiZXhwIjoxNjgzOTc4MzUzLCJpYXQiOjE2ODM5NjAzNTN9.
        FPimtUkXZd2nTn9jYb8f5cYKf8cnz9CXwe0PPxPiJxj7F5OQ8blf_SquMLk0dE-8FwgMZxGvVfLGdaW2IAdk0Q"
}
```


f. UML diagram:



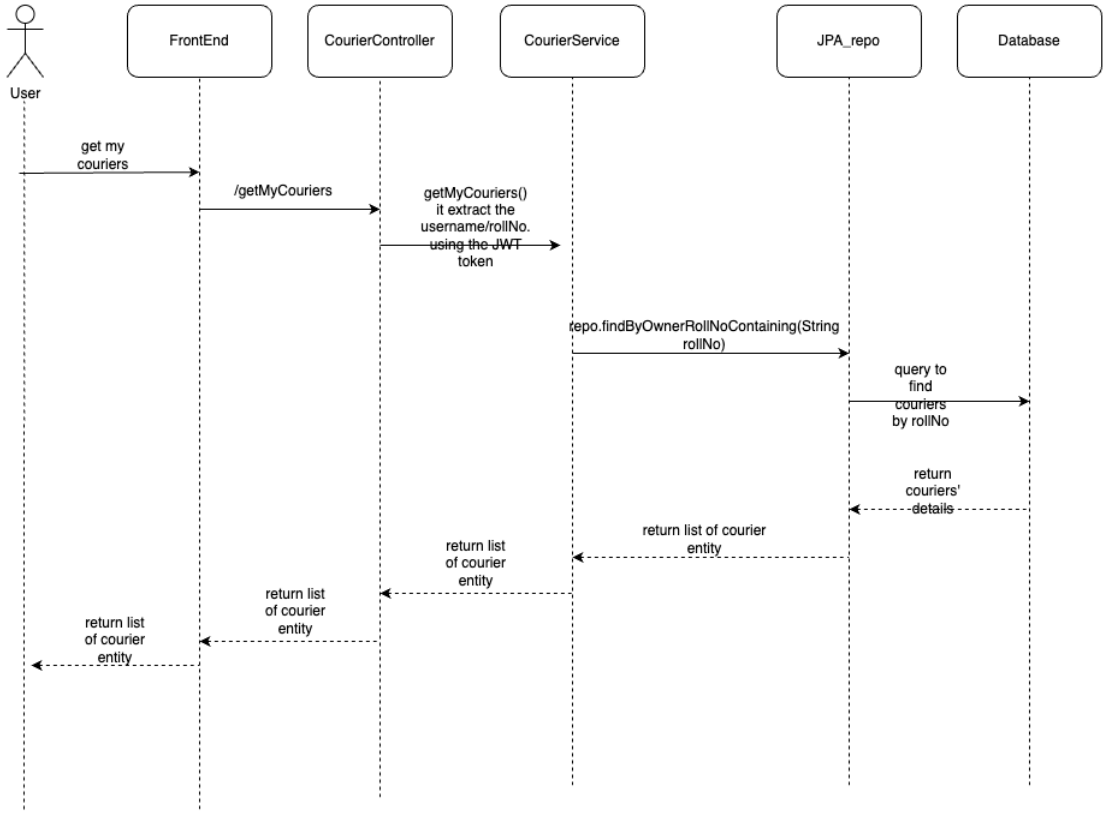
4. Get my couriers:
    a. Endpoint: *localhost:9090/getMyCouriers*
    b. HTTP method: GET
    c. Descriptions: it will extract the ‘username’ attribute of the logged in user, and then use that to get the list of the corresponding couriers.

    d. Request body: none. It makes use of the jwt token to just check authorisation.

    e. Response body :

```
[
    {
        "courierID": 6,
        "owner": "Aditya Vardhan",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-14",
        "arrivalTime": "19:50:30",
        "ownerRollNo": "IMT2019003",
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    },
    {
        "courierID": 7,
        "owner": "Aditya Vardhan",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-14",
        "arrivalTime": "19:50:30",
        "ownerRollNo": "IMT2019003",
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    }
]
```


f. UML diagram:



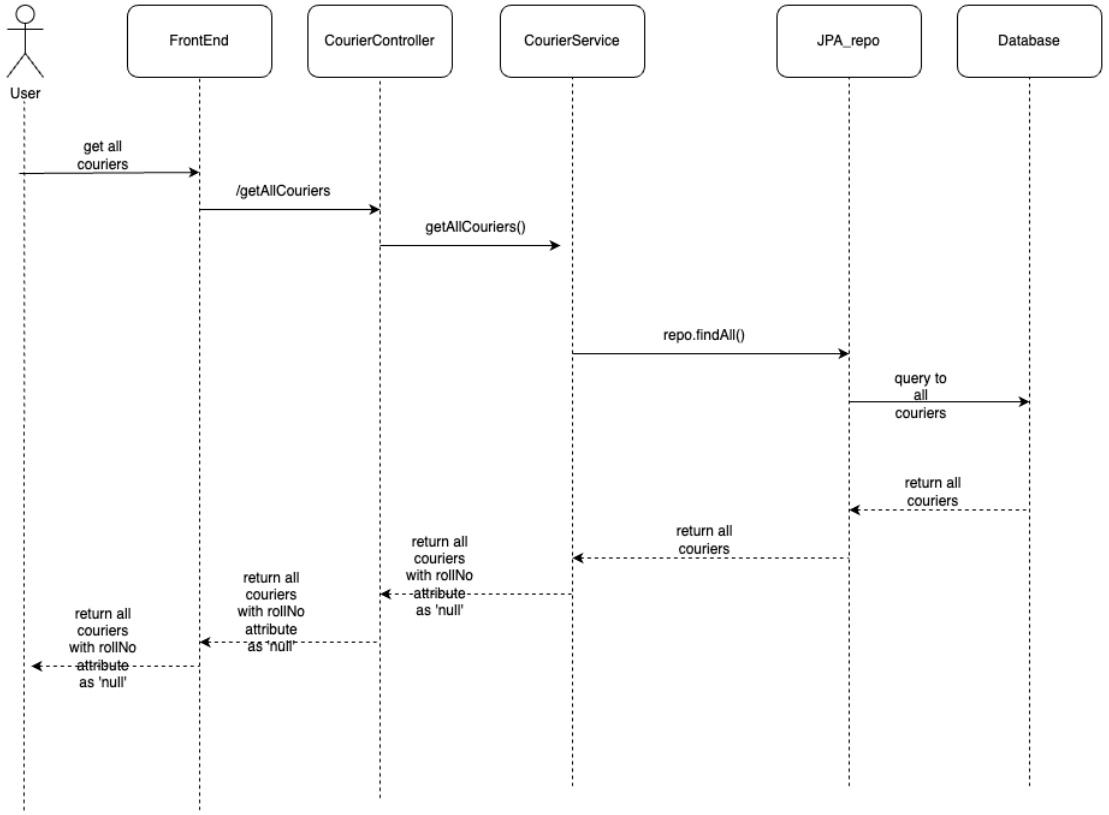
5. Get all couriers without roll number:
    a. Endpoint: *localhost:9090/getAllCouriers*
    b. HTTP method: GET
    c. Descriptions: it will get all the couriers without ‘ownerRollNo’ mentioned on it.

    d. Request body : none. It makes use of the jwt token to just check authorisation.

e. Response body : note that only those couriers are returned which don't have roll numbers on them.


```
[
    {
        "courierID": 5,
        "owner": "Aditya Kaka",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-15",
        "arrivalTime": "19:50:30",
        "ownerRollNo": null,
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    },
    {
        "courierID": 3,
        "owner": "Aditya Kaka",
        "courierCompany": "Amazon",
        "arrivalDate": "2023-04-15",
        "arrivalTime": "19:50:30",
        "ownerRollNo": null,
        "receiverName": null,
        "receiverRollNo": null,
        "deliverDate": null,
        "deliverTime": null,
        "status": "ACTIVE"
    }
]
```


f. UML diagram:



6. Forgot password:
    a. Endpoint: *localhost:9090/forgotPasswordOTP?mailId=<mail>*
    b. HTTP method: POST
    c. Descriptions: it will send an OTP to the registered email-id.

    d. Request parameters :

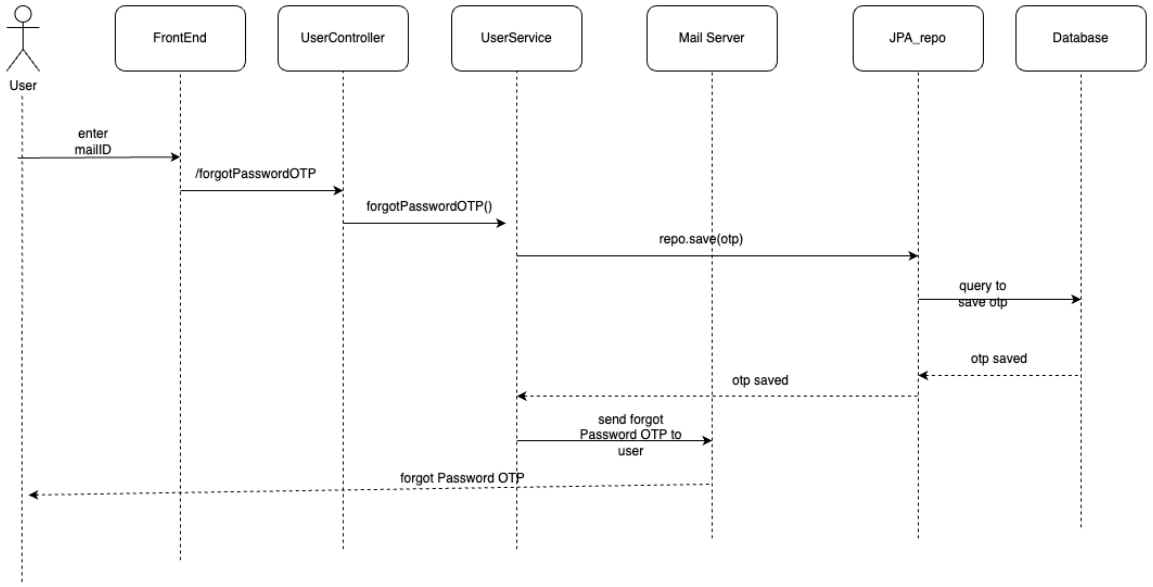
Query Params

	Key	Value
☑	mailId	vardhan.av2000@gmail.com

e. Response body : Note the password the otp values are exclusively set to null while returning the object for security reasons.


```
{
    "userName": "IMT2019003",
    "userFirstName": "Aditya",
    "userLastName": "Vardhan",
    "userPassword": null,
    "role": [
        {
            "roleName": "User",
            "roleDescription": "Default role for newly created record"
        }
    ],
    "mailID": "vardhan.av2000@gmail.com",
    "status": "VERIFIED",
    "otp": null
}
```


f. UML diagram:



7. Reset Password:
    a. Endpoint: *localhost:9090/resetPassword*
    b. HTTP method: POST
    c. Descriptions: it will update the password of the user in the database, if the user enters the correct OTP.

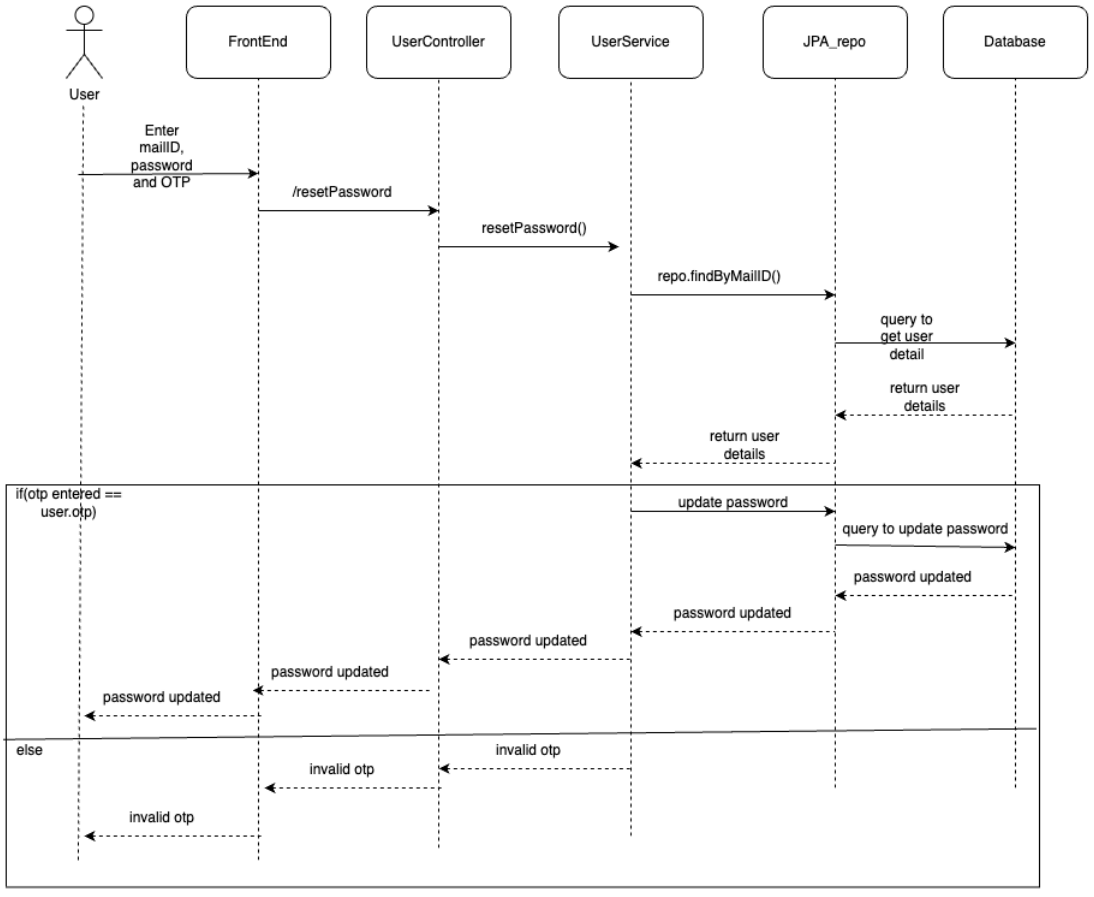
    d. Request body :

```
    "userName":"IMT2019003",
    "mailID":"vardhan.av2000@gmail.com",
    "otp":"821094",
    "userPassword":"5678"
```


e. Response body :


```
{
    "userName": "IMT2019003",
    "userFirstName": "Aditya",
    "userLastName": "Vardhan",
    "userPassword": "$2a$10$GPHArvVLvBVl4h06.TNsJemTd3/Ecw7AKkhkDEd0astYnHXgSSOlK",
    "role": [
        {
            "roleName": "User",
            "roleDescription": "Default role for newly created record"
        }
    ],
    "mailID": "vardhan.av2000@gmail.com",
    "status": "VERIFIED",
    "otp": "821094"
}
```


f. UML diagram:



Docker

This is the back-end dockerization, we just transfer the .jar file and run it like a java code.

```
FROM openjdk:17
EXPOSE 9090
ADD target/CourierManagement-0.0.1-SNAPSHOT.jar springboot.jar
ENTRYPOINT [ "java", "-jar", "/springboot.jar" ]
```

This is the frontend dockerization, we just transfer the build production level code to nginx and rest is taken care of nginx container files. It runs it on a server, with proxy settings:

```
FROM nginx:alpine
COPY build/ /usr/share/nginx/html
EXPOSE 80
```

Automation

Ansible

We made 2 ansible files:

- One for MacOS
- One for Ubuntu

I will be describing MacOS ansible files here.

First we use brew to install some packages that need to be installed and we start the minikube with 4 cpu core and 8 GB storage to run Prometheus and Grafana, and ELK.

```
---
- name: Installing Dependencies
 hosts: localhost
 tasks:
   - name: Install OpenJDK 17
     homebrew:
       name: openjdk@17
       state: present

   - name: Check Minikube status
     command: minikube status
     register: minikube_status
     changed_when: false
     failed_when: false
   - name: Printing Minikube status
     debug:
       var: minikube_status
   - name: Start Minikube
     become: false
     command: minikube start --cpus 4 --memory 8192
     when: "'Running' not in minikube_status.stdout"
   - name: Installing Addons
     become: false
     command: minikube addons enable ingress
     ignore_errors: true
```

```
 - name: Installing Addons
   become: false
   command: minikube addons enable metrics-server
   ignore_errors: true
```

Now we transfer secret and config files from local system to remote system:

```
 - name: Transfer file from controller
   synchronize:
     src: /Users/architsangal/Data/Projects/secretFiles/configFileAndSecrets.yaml
     dest: ./
 - name: LS
   shell: |
     ls
   register: ls
   changed_when: false
   failed_when: false
 - name: Debugging Logs
   debug:
     var: ls
```

Now we apply k8s files and remove secret and config file so that there is no security issue:

```
- name: Setting Config and Secrets
   shell: |
     kubectl apply -f .
     rm configFileAndSecrets.yaml
   ignore_errors: true


 - name: Applying K8s Command in Repository
   shell: |
     cd ./../kubernetesnew/
     kubectl delete -f .
     kubectl apply -f .
   register: applying_K8s
   changed_when: false
   failed_when: false
 - name: Debugging Logs
   debug:
     var: applying_K8s
```

Now we create Helm repos for continuous monitoring and we set it up:

```
- name: Checking the Helm Repos
   command: helm repo list
   register: helm_repo_list
   changed_when: false
   failed_when: false
 - name: Adding Prometheus For Monitoring If not Available In Helm Repo
   become: false
   command: helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
   ignore_errors: true
 - name: Adding Stable Repo In Helm Repo If Not Available
   become: false
   command: helm repo add stable https://charts.helm.sh/stable
   ignore_errors: true
 # We want to have a stable environment so we will not run this command
 # as it may have unintended consequences
```

```
  # - name: Updating Helm Repos
  #   become: false
  #   command: helm repo update
  - name: Getting Info about the pods running
    command: kubectl get all
    register: get_all
    changed_when: false
    failed_when: false
  - name: Uploading Prometheus Charts to Kubernetes if not there
    become: false
    command: helm install prometheus prometheus-community/kube-prometheus-stack
    ignore_errors: true
  # May Be useful if we want a more stable environment
  # - name: Uploading Prometheus Charts Fixed Version to Kubernetes if needed
  #   become: false
  #   command: helm install prometheus prometheus-community/kube-prometheus-stack --version "9.4.1"
  #   when: "'prometheus' not in helm_repo_list.stdout"

  - name: Port-Forwarding Prometheus Grafana Deployment
    command: kubectl port-forward deployment/prometheus-grafana 3000
    async: 31536000
    poll: 0
    register: progra
    ignore_errors: true
  - name: Debugging Logs
    debug:
      var: progra
```

Setup ELK in docker container:

```
  - name: ELK setup
    shell: |
      cd ./../ELK
      git clone https://github.com/deviantony/docker-elk
    ignore_errors: true

  # https://github.com/deviantony/docker-elk
  - name: ELK setup
    shell: |
      cd ./../ELK/docker-elk
      docker-compose up setup
      docker-compose up -d
    ignore_errors: true
```

Jenkins

First we set up the tool and environment like java, maven, path (Path not PATH), registry or credentials of docker hub to push the images.

```
pipeline
{
   tools {
       maven "M2_HOME"
       jdk "Java 11"
   }

   agent any
   environment
```

```
{
    Path = "/usr/local/bin/minikube:${PATH}"
    registry = "architsangal/speminiproject"
    registryCredential = "dockerhub"
    dockerImage = ""
}
```

Now we define images like git clone

```
stages
{
    stage('Clone Git')
    {
        steps
        {
            git branch: 'main',
            url: 'https://github.com/architsangal/Courier-Management-Application.git'
        }
    }
```

Now we build the front-end cloned repo by going to the correct workspace. We also push the image to docker hub, using the credentials:

```
    stage('Frontend Docker Image Build')
    {
        steps
        {
            dir("frontend/")
            {
                sh "npm install"
                sh "npm run build"
                sh "docker build -t architsangal/courier_react:latest ."
            }
        }
    }
    stage('Frontend DockerHub Image Push')
    {
        steps
        {
            script
            {
                docker.withRegistry('', registryCredential)
                {
                    sh "docker push architsangal/courier_react:latest"
                }
            }
        }
    }
```

Now we build the back-end cloned repo by going to the correct workspace. It includes maven clean install. We also push the image to docker hub, using the credentials:

```
    stage('Backend Docker Image Build')
    {
        steps
        {
            dir("backend/")
            {
                sh "mvn clean install"
                sh "docker build -t architsangal/courier_spring_boot:latest ."
```

```
                }
            }
        }
        stage('Backend DockerHub Image Push')
        {
            steps
            {
                script
                {
                    docker.withRegistry('', registryCredential)
                    {
                        sh "docker push architsangal/courier_spring_boot:latest"
                    }
                }
            }
        }
```

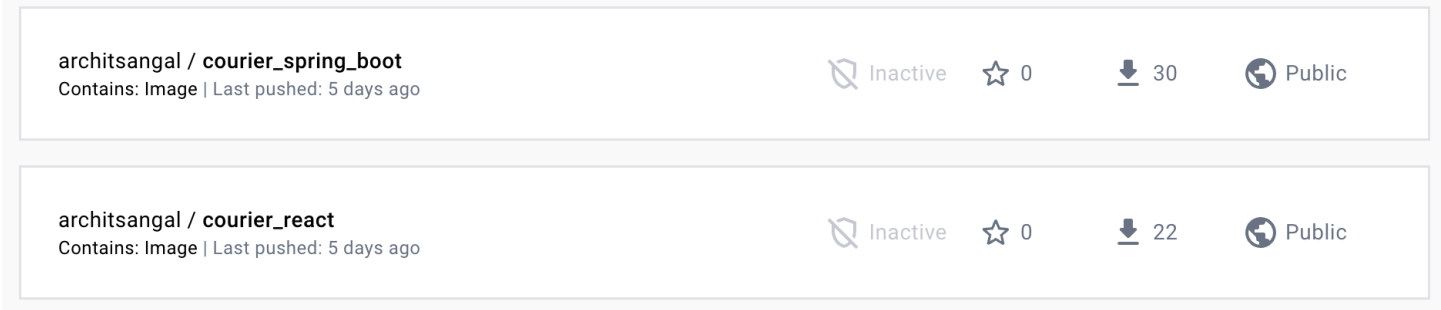
We will be using minikube and Jenkins need to refresh the path, so we print the path to refresh it:

```
        stage('Checking Path')
        {
            steps
            {
                sh 'whereis minikube'
                sh 'echo $USER'
            }
        }
```

Finally we run the ansible file described above:

```
        stage('Ansible Deployment')
        {
            steps
            {
                dir('./ansible')
                {
                    ansiblePlaybook colorized: true,
                    installation: 'Ansible',
                    // inventory: 'inventory',
                    playbook: 'playbook.yml'
                }
            }
        }
    }
}
```

We can see the pushed images on docker hub:



Kubernetes

For persistence storage we use pvc:

```
# Define a 'Persistent Volume Claim'(PVC) for MySQL Storage, dynamically provisioned by cluster
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mysql-pv-claim # name of PVC essential for identifying the storage data
 labels:
   app: mysql
   tier: database
spec:
 accessModes:
   - ReadWriteOnce #This specifies the mode of the claim that we are trying to create.
 resources:
   requests:
     storage: 1Gi #This will tell kubernetes about the amount of space we are trying to claim.
---
```

We also need a service for communication to deployment:

```
# Define a 'Service' To Expose MySQL to Other Services
apiVersion: v1
kind: Service
metadata:
 name: mysql
 labels:
   app: mysql
   tier: database
spec:
 ports:
   - port: 3306
     targetPort: 3306
 selector:
   app: mysql
   tier: database
 clusterIP: None # DNS is used, so clusterIP is not needed
---
```

Finally we have the deployment, in which we link the service and deployment, pvc, use config files for env variables, this introduces persistence storage to pod:

```
# Deployment specification begins here
apiVersion: apps/v1
kind: Deployment
metadata:
 name: mysql
 labels:
   app: mysql
   tier: database
spec:
 selector: # mySQL Pod Should contain same labels
   matchLabels:
     app: mysql
 strategy:
   type: Recreate
 template:
   metadata:
```

```
    labels: # Must match 'Service' and 'Deployment' selectors
      app: mysql
      tier: database
  spec:
    containers:
      - name: mysql
        image: mysql
        imagePullPolicy: "IfNotPresent"
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysecret
                key: Root_Pass
          - name: MYSQL_DATABASE
            valueFrom:
              configMapKeyRef:
                name: configpara
                key: DATABASE_NAME
        ports:
          - containerPort: 3306
            name: mysql
        volumeMounts:
          - mountPath: /var/lib/mysql
            name: mysql-persistance-storage
    volumes:
      - name: mysql-persistance-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

We have the deployment for spring boot which includes the env variables for communication with other pods.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: springboot
spec:
 selector:
   matchLabels:
     app: springboot
 replicas: 1
 template:
   metadata:
     labels:
       app: springboot
   spec:
     containers:
       - name: springboot
         image: architsangal/courier_spring_boot:latest
         ports:
           - containerPort: 9090
         env:   # Setting Environmental Variables
           - name: DB_HOST   # Setting Database host address
             value: mysql
           - name: DB_NAME  # Setting Database name
             valueFrom:
               configMapKeyRef:
                 name: configpara
                 key: DATABASE_NAME
           - name: DB_USERNAME  # Setting Database username
             value: root
```

```
          - name: DB_PASSWORD # Setting Database password
            valueFrom:
              secretKeyRef:
                name: mysecret
                key: Root_Pass
---
```

We need a service for communication to deployment of spring boot:

```
# Define a 'Service' To Expose the application
apiVersion: v1
kind: Service
metadata:
 name: springboot
spec:
 ports:
   - protocol: "TCP"
     port: 9090    # Port inside the cluster
     targetPort: 9090  # Port exposed to the outside
     nodePort: 30200
 selector:
   app: springboot
 type: LoadBalancer
```

We have also implemented ingress for professional deployment for:

- Spring boot

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: courier-ingress
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
 - host: "springboot.com"
   http:
     paths:
     - pathType: Prefix
       path: "/api/(.*)"
       backend:
         service:
           name: springboot
           port:
             number: 9090
```

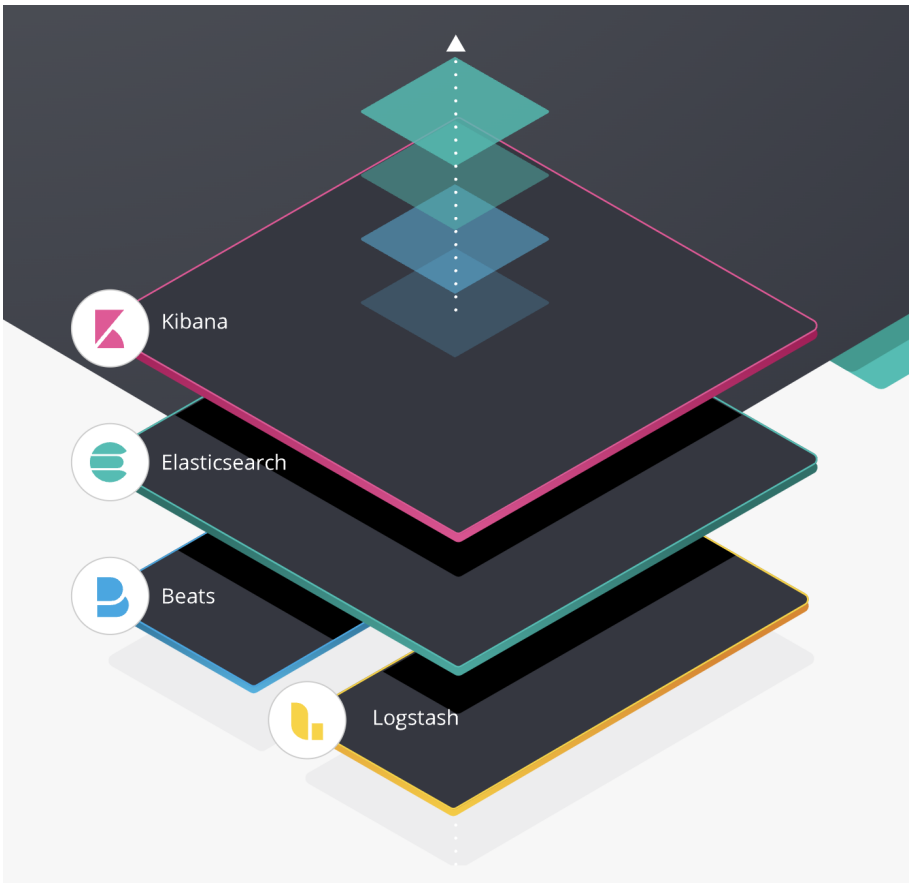
- Frontend

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: courier-ingress2
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
 - host: "courier-management"
   http:
     paths:
     - pathType: Prefix
       path: "/temp/(.*)"
       backend:
```

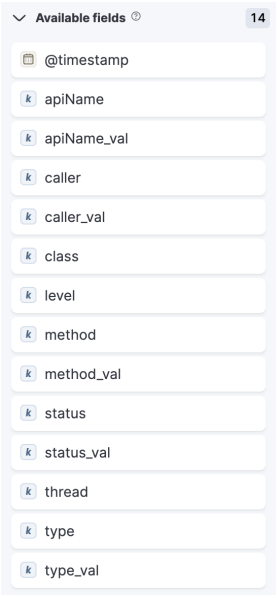
```
        service:
          name: react
          port:
            number: 80
```

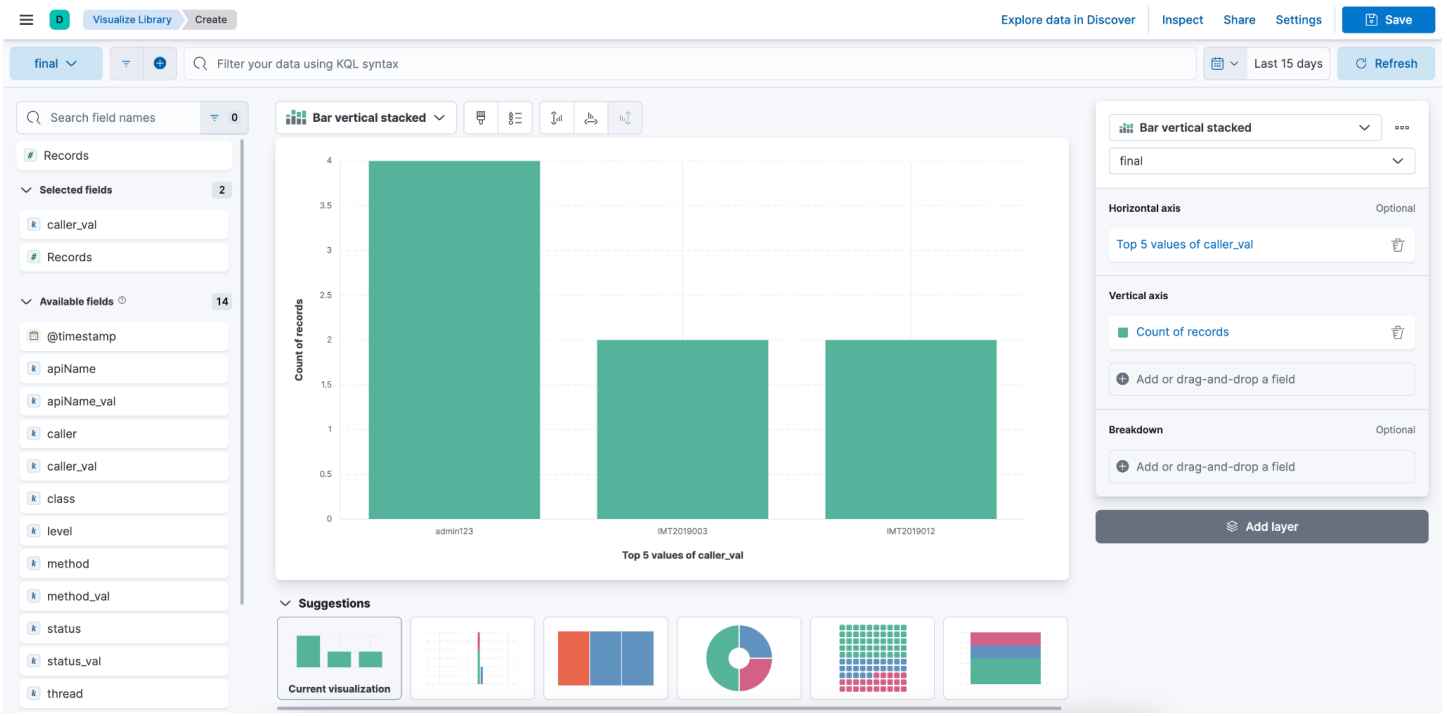
Logging

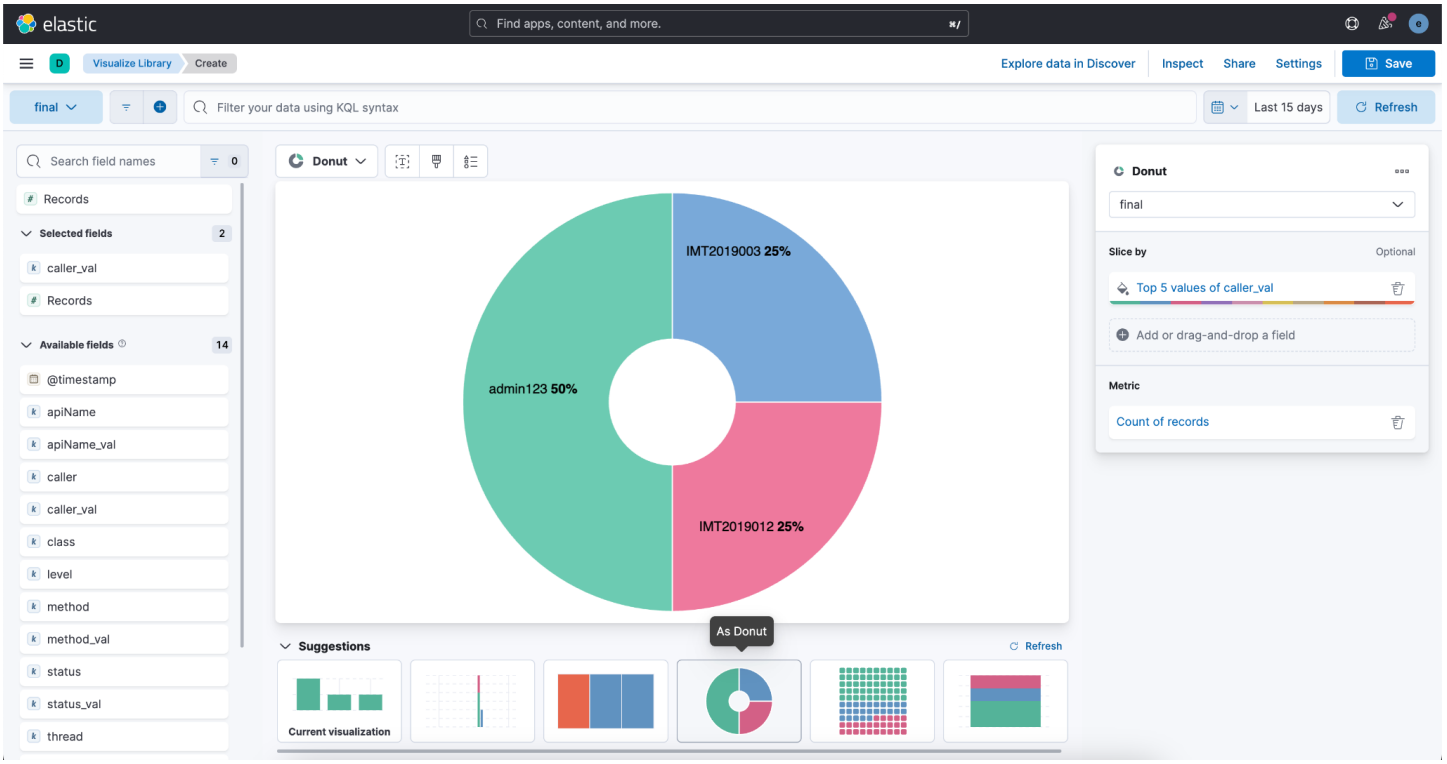
The ELK Stack is a log management platform made up of three open-source Elasticsearch, Logstash, and Kibana products. Elasticsearch indexes and saves the information that Logstash collects and parses. After Kibana creates visualisations of the data. (documentation)



Below are the available fields:



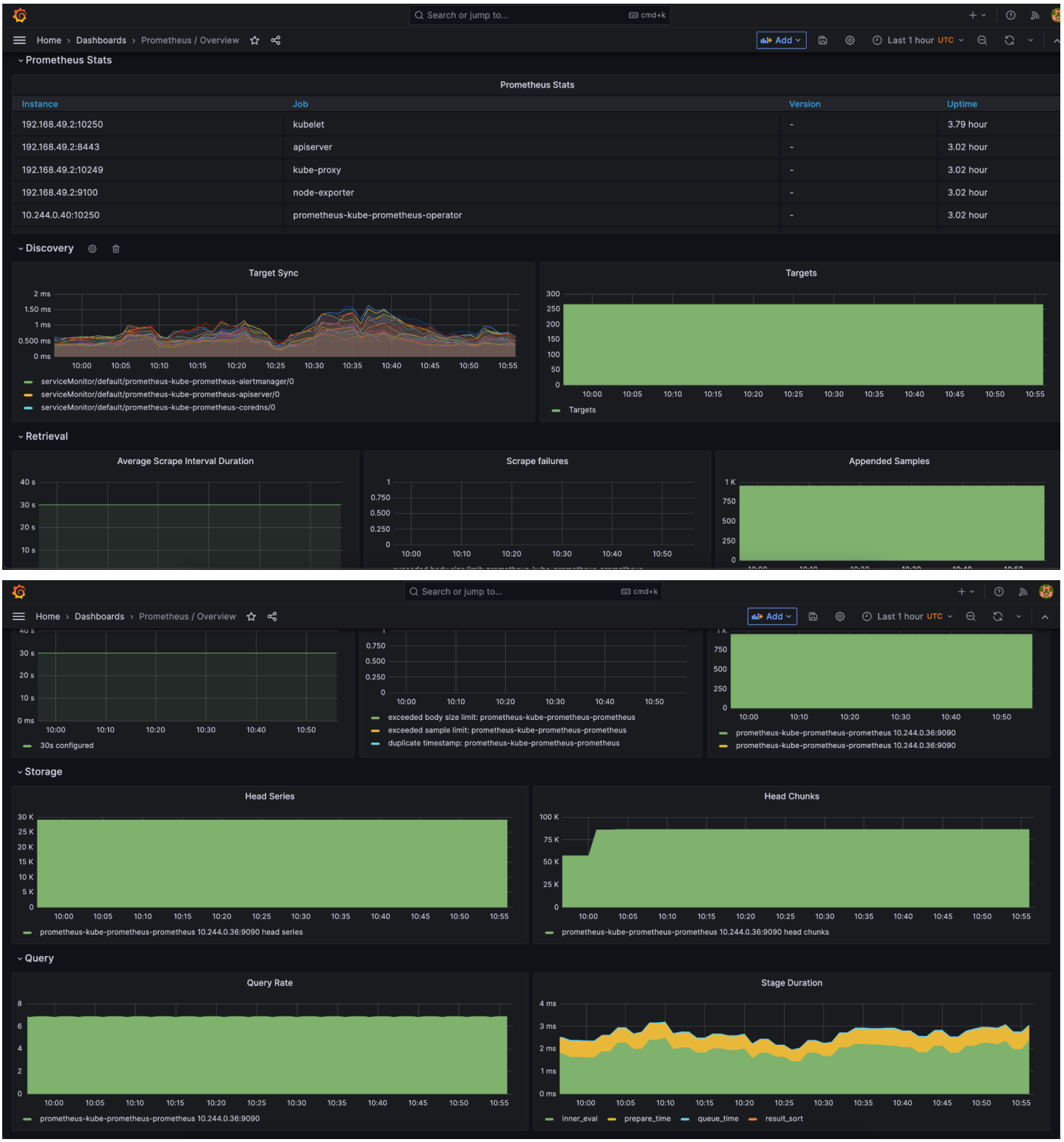




Continuous Monitoring

Prometheus Operator implements the Kubernetes Operator pattern for managing a Prometheus-based Kubernetes monitoring stack. A Kubernetes Operator consists of Kubernetes custom resources and Kubernetes controller code. Together, these abstract away the management and implementation details of running a given service on Kubernetes.

The Prometheus Operator installs a set of Kubernetes custom resources that simplify Prometheus deployment and configuration. For example, using the ServiceMonitor custom resource, you can configure how to monitor Kubernetes services in K8s YAML manifests instead of Prometheus configuration code. The Operator controller then communicates with the K8s API server to add service /metrics endpoints and automatically generate the required Prometheus scrape configurations for the configured services. Below are some screenshots:



Scope of Future Work

- We have implemented logging that is logstash, elastic search and kibana. That can be automated, using file beats. We have implemented these using HELM charts. Initially they were implemented using docker containers and later they were transferred and set up properly in kubernetes (check the commits of the logging branch on GitHub). Now, only file beats need and logstash connection needs to be handled.
- Paging can be implemented in the frontend to make the frontend scalable-ready.
- Code compiled on MacOS arm64 may not work on ubuntu x86. Problem lies with different processors M1 chips and Intel chips. This can be a area of work in future.

Challenges Faced

- Spring Boot pod was not connecting with mysql pod due to environment variable. It took a lot of time to figure this problem out.
- I tried paging for frontend but the process of introducing it was pretty long and we figured out that it is okay to not to implement it as it creates a problem only if we have more than 1000 packages reset we can filter out.
- Code compiled on MacOS arm64 may not work on ubuntu x86. So I need to compile the codes in the respective environments.

Conclusion

We were able to make a good UI for frontend and backend. We did this using continuous deployment and many automation tools described above. It really helps us reduce our work of compilation and testing. Also, once continuous deployment is done, we can deploy it almost instantly. It was a good project and we learnt a lot.

References

1. https://spring.io/guides/gs/multi-module/
2. https://kubernetes.io/
3. https://www.ansible.com/
4. https://coreui.io/demos/bootstrap/4.2/free/notifications/alerts.html
5. https://www.docker.com/
6. https://www.elastic.co/what-is/elk-stack
7. https://helm.sh/
8. https://artifacthub.io/
9. https://gitlab.com/nanuchi/youtube-tutorial-series/-/blob/master/prometheus-exporter/install-prometheus-commands.md
10. https://www.jenkins.io/
11. https://www.scaler.com/topics/spring-boot/mockito-spring-boot/