# CourtsAtHome

The application aimed at eliminating presence of litigant or lawyer in the court and adjudication of the cases related to traffic department to online. User can search the cases by giving user name/vehicle number/phone number. User can check the status of the case filed. User can close the case by paying the fine. The technologies used to create this application include Angular, SQL and Java.

# Application Scenario

1. After logging in or registering, users can search cases by phone number or vehicle number.
2. The user can view all cases associated with the entered vehicle number or mobile number by clicking the Search button.
3. The specifics of every case that has been registered will be listed in a table along with a ***Pay Now*** button. When it button is clicked, a modal popup displaying case information such as ***No of Violations***, ***Total Fine***, and ***Pay*** button will appear.
4. As soon as he hits the ***Pay*** button, the modal window will close, a toast message will appear, and the case's status will change to ***Closed***.
5. By clicking the ***Logout*** button in the top right corner of the screen, users can log out of the application.

# Instructions

* A Project with the desired folder structure has been already created. You are only permitted to alter the logical layer; creating files or folders is not permitted.
* The application includes a **database** layer and the required folders that contains the necessary code. You only need to complete the given objectives by writing your logic in the specified controller and service methods.
* You are not allowed to directly insert new data into the database.
* Do not change any schema variables

# Backend Objectives

The application uses the APIs mentioned below. where some of the controllers and services have definitions and others have blank definitions, requiring you to develop logic to fulfil the objectives.

1. We have defined the below methods in the ***Controller*** Layer. You must complete the body of it.

@GetMapping("/getCaseDetails")

**public** ResponseEntity<ResponseDto> getCaseDetails (@RequestParam (required = **false**) String mobileNo, @RequestParam (required = **false**) String vehicleNo)

@PutMapping("/updateCaseStatus")

**public** ResponseEntity<ResponseDto> updateCaseDetails (@RequestParam Integer caseId)

1. We have defined below methods in ***Service*** Layer. You must complete the body of them.

**public** List<CaseDetailsDto> getCaseDetails (String mobileNo, String vehicleNo)

**public** CaseDetailsDto updateCaseDetails (Integer caseId)

**Note**: -

You are allowed to make changes only in below files.

1. CaseDetailsController.java

2. CaseDetailsServiceImpl.java

You are not allowed to make any changes in below files.

1. application.properties
2. pom.xml

You are not allowed to make any changes in the files present in below packages.

1. com.example.courtathome.constant

2. com.example.courtathome.dto

3. com.example.courtathome.entity

4. com.example.courtathome.exception

5. com.example.courtathome.handler

6. com.example.courtathome.repository

# Build, Deploy and Run

1. Load the project in the IDE
2. Update the maven dependency
3. mvn life cycle to build & deploy
4. Run the project

### Frontend Objectives

**Note: -**You are allowed to make changes only in below mentioned files.

1. search-view-case.component.html
2. search-view-case.component.ts
3. case-details-modal.component.ts
4. case-details-modal.component.html
5. In search-view-case.component a radio will indicate based on what user is searching cases. Take a reference of that radios to determine what form control should be added or removed from “searchCasesForm”.
   1. If user selected “Mobile No.” add a form control by the name “**mobileNumber**” which should only accept number and should throw error if any entry made other than numbers.
   2. If user selected “Vehicle No.” add a form control by the name “**vehicleNo**” which should only accept alphanumeric and should throw error if any entry made other than alphanumeric.
6. In search-view-case.component, on click of search “getCaseDetails” API will be called and the response will be stored in the “caseDetails” variable. Write your logic to process and reinitialize “caseDetails” variable with modified data based on below conditions.
   1. Remove all the duplicate entries exists for same caseName on the same date (irrespective of time) in the “caseDetails” variable, but keep track of how many entries where there by adding new property “noOfVoilations” to the object.

**For example: -** let’s consider below array as scenario.

**Initial Response is as Below:** -

initialResponse = [

{caseName: ‘Signal Violation”, date:’ Mon Jan 16 2023 18:38:27’, fine:1000},

{caseName: ‘Signal Violation”, date:’ Mon Jan 16 2023 20:40:15’, fine:1000},

{caseName: ‘Helmet Violation”, date:’ Tue July 04 2023 19:36:01’, fine:1500}

]

***Result Should Be: -***

modifiedResponce = [

{caseName: ‘Signal Violation”, date:’ Mon Jan 16 2023 18:38:27’, fine:1000, noOfVoilations :2},

{caseName: ‘Helmet Violation”, date:’ Tue July 04 2023 19:36:01’, fine:1500,

noOfVoilations :1}

]

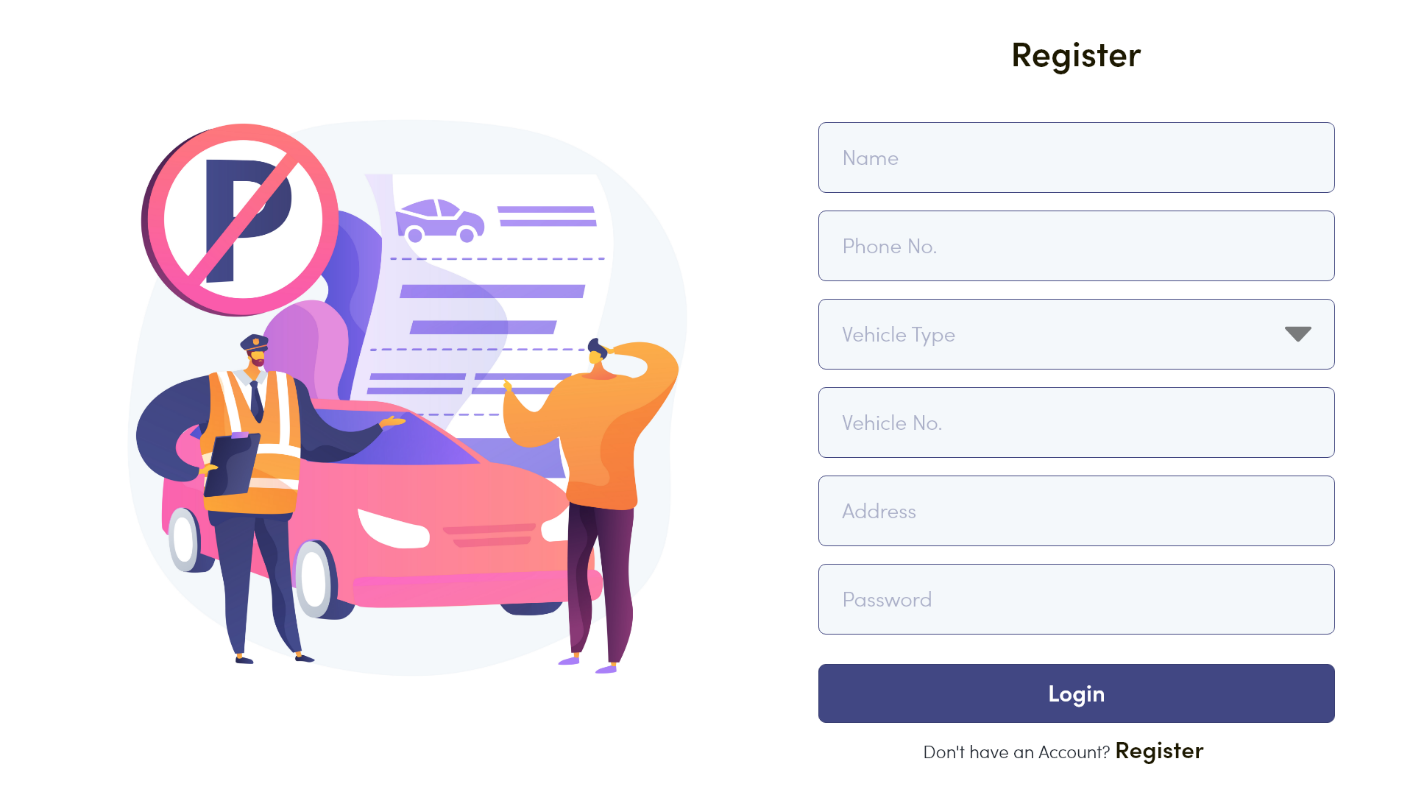
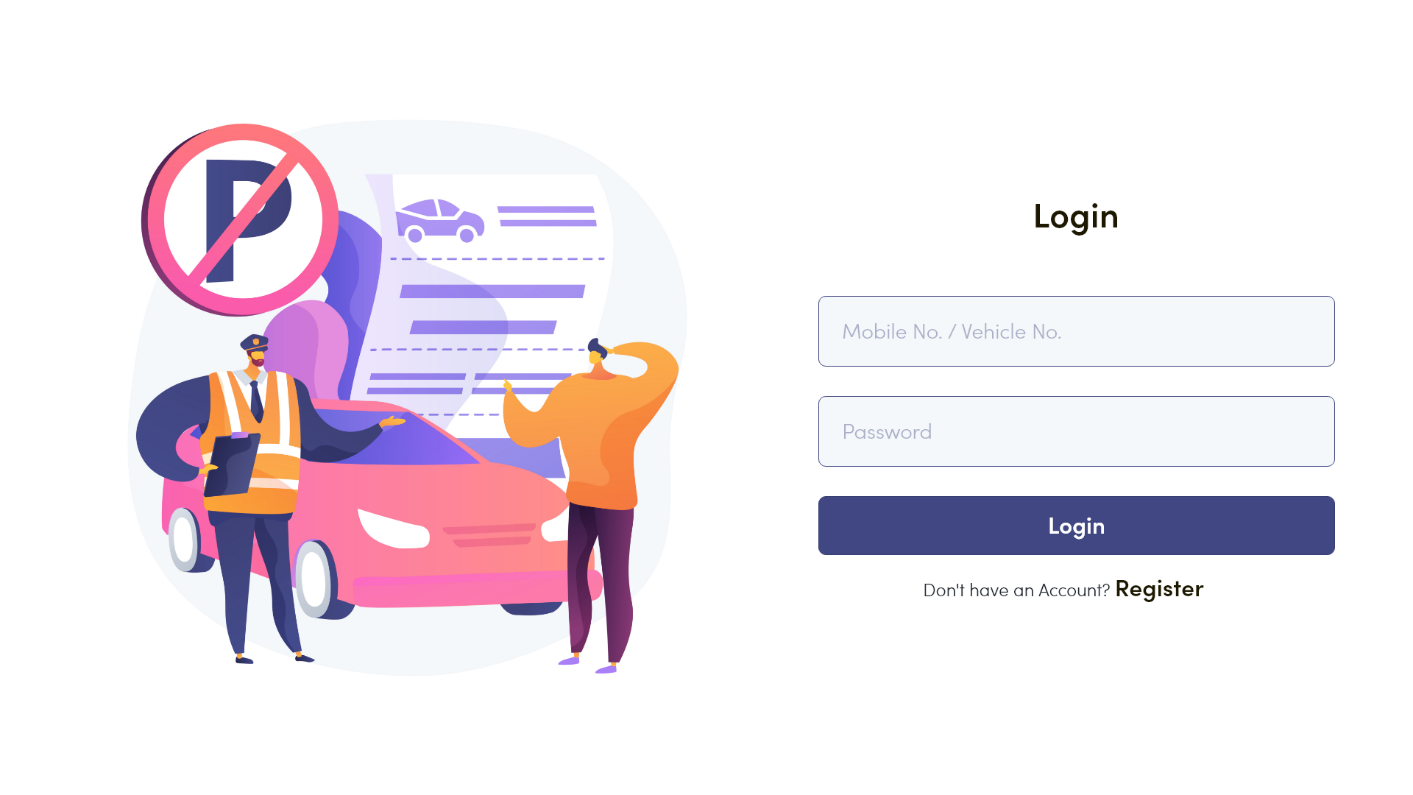
1. In case-details-modal.component, on click of pay call “**updateCaseStatus**” API to mark the “**status”** as “**Closed**” and close the modal with a proper notification about case status to User.

# Build, Run and Deploy

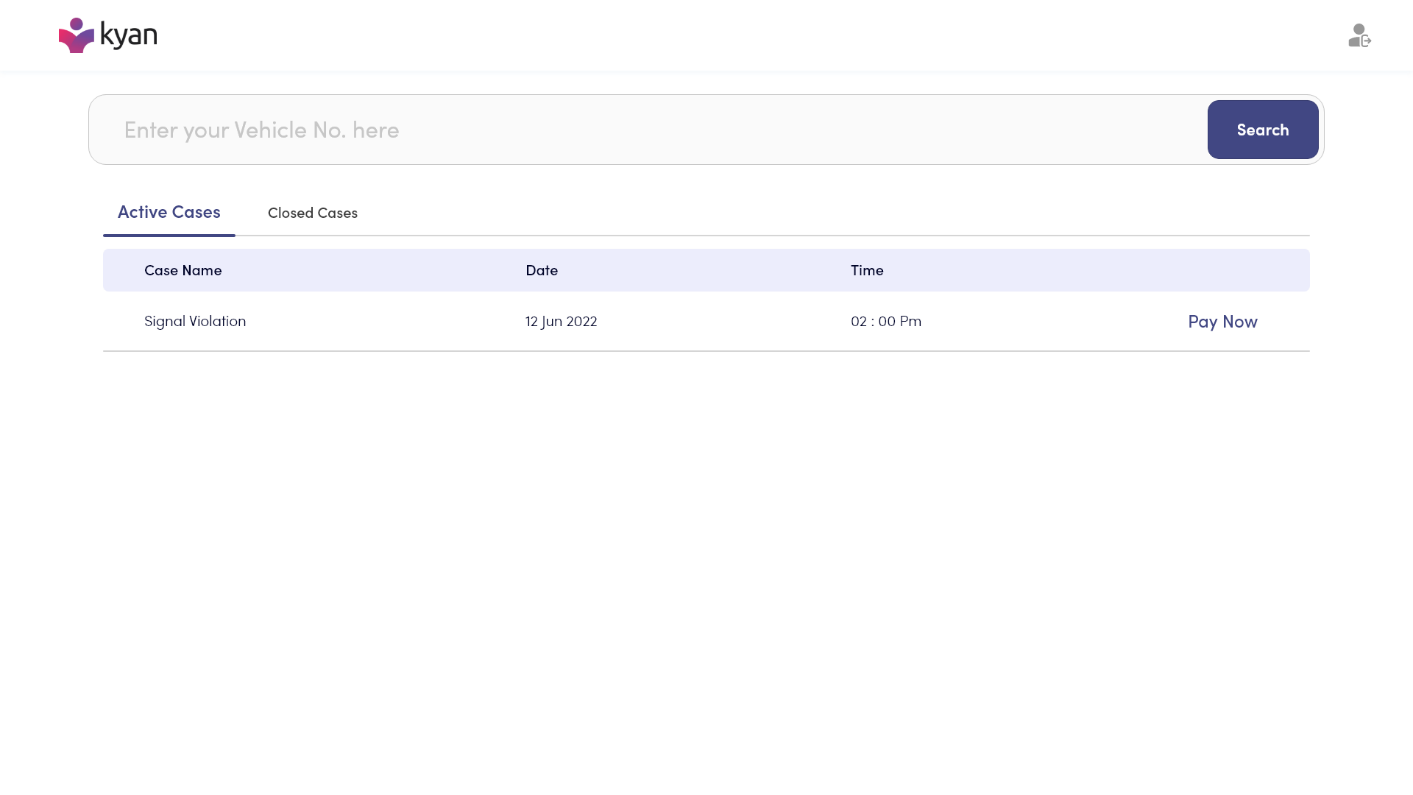
* + Load the project in VS Code.
  + Install the packages.
  + You can use the build command and deploy the application on the server.
  + Start the development server.

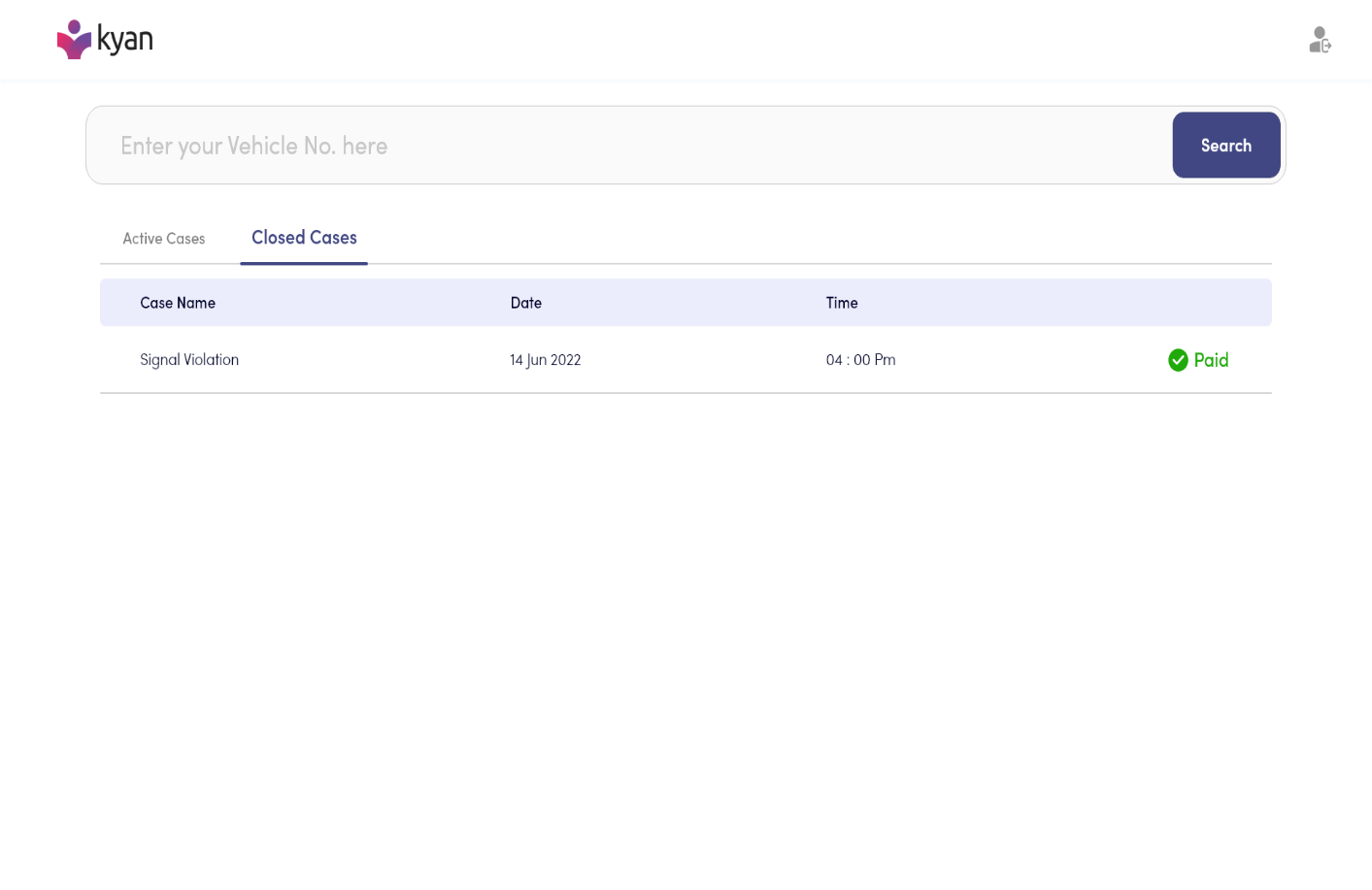
**User Interface Designs**

* Login and Register Page: -

****

* Search View Case Page: -

****

****

* Case Details Modal Page: -

