1. Project Details

1.1. Title of the Project

Auto2Auto: Online Vehicle Dealing Application

1.2. Course name

PG-DAC

1.3. PRN no. with Team Member Name

Sr.No.	Name	Roll No.
1.	Yugant Prakash Kadu	200250120113
2.	Ekta Singh	200250120029
3.	Vaibhav Prakash Dhabekar	200250120103
4.	Pritam Hemantrao Raut	200250120070

2. Working of the Project

2.1. Description

2.1.1. Auto2Auto is an Online Vehicle Dealing Application based on B2B E-Commerce, or Business to Business Electronic Commerce business model.

2.1.2. This platform establishes online business dealing of vehicle between two entities.

2.1.3. The features or functions like Authentication, Authorization, Product exploration and Payment Gateway will be implemented for development of this online dealing application.

2.2. Homepage

2.2.1. Landing page of application is homepage with tabs named as Category, Brand, Login and Register

2.3. Registration

2.3.1. Text fields are provided so as to record user information and also customerType so that application can distinguish between them as Admin/Retailer/Manufacturer.

2.4. Login

2.4.1. A User needs to login to buy a product or navigate through vehicles on a website. User needs to enter the correct credential for email and password.

2.5. Login as Manufacturer and Retailer

2.5.1. When user is authenticated successfully, user as manufacturer is redirected to Manufacturer Homepage with Header fields descriptive as manufacturer name, manufacturer type with its brand logo.

2.5.2. Description about the respective brand vehicle is provided such as productId, Product image, Brand Name, Product Name, etc.

2.5.3. Add vehicle tab is provided to add a new vehicle in the Manufacturer portfolio.

2.5.4. Profile tab show the Manufacturer details

2.5.5. Logout tab destroys the current session of the Manufacturer and redirects to the homepage.

2.6. Payment Page (Payment Receipt)

2.6.1. A Retailer needs to select a type of payment from Razorpay Checkout form and make Payment.

2.6.2. After successful payment, Retailer is redirected to Payment Receipt page which includes details like Payment id, Order Id, Payment amount, Payment mode and date.

2.7. Login as Admin

2.7.1. When user is authenticated successfully, user as Admin is redirected to Admin Homepage with Header fields descriptive as Admin name

2.7.2. Admin have the right to modify User, Category, Brand and Product details

2.7.3. In addition to this, admin can Add another Admin and Add new Brand by using respective available tabs.

2.7.4. User tab, Category tab, Brand tab, Product tab, Order tab, Payment tab shows their respective homepage with data.

3. Scenario: Logged in as Retailer (Presentation, Service and Dao layer)

3.1. Registration

3.1.1. **Presentation**

User needs to register his details on the website by selecting UserType as Retailer/Manufacturer and other details.

3.1.2. **Service**

When the user clicks on the Register button, callRegisterUserDetails() service will be called and POST request to '/user/registration' endpoint will be done.

3.1.3. **Dao**

ins() function corresponding to '/user/registration' endpoint will be invoked and user_object.save() will be executed. Resulting data will be stored in a mysql database.

3.2. Login

3.2.1. **Presentation**

User needs to login to buy a product or navigate through vehicles on a website.

Service

When the user clicks on the Login button, onLogin(form: NgForm) service will be called and a POST request to '/user/verifyCustomer' endpoint will be done.

3.2.2. **Dao**

user.login(us.getEmail(), us.getPassword()) function corresponding to '/user/verifyCustomer' endpoint will be invoked and user_object.save() will be executed. Resulting data will be stored in a mysql database.

3.3. Retail Homepage

3.3.1. **Presentation**

After Login Retailer will be navigated to Retailer Home Page . When Retailer HomePage will be open he/she can see List of vehicles with their features.

Service

When the home page is loaded invokeProductsDetails() service will be called and GET request to '/vehicle/getAllProducts' endpoint will be done.

3.3.2. **Dao**

findAll() function gets called.

3.4. Order a product

3.4.1. **Presentation**

When Retailer wants to buy an item clicks on Buy Now button, Order id is generated and passed to Payment Gateway.

3.4.2. **Service**

redirectDotnet(productData: Products) service will be called and post request to'/buy/generateOrder' endpoint will be done.

3.4.3. **Dao**

It will go to RazorPay Gateway.

3.5. Payment Page

3.5.1. **Presentation**

A Retailer needs to select a type of payment from Razorpay Checkout form and make Payment.

3.5.2. **Service**

RazorpayController() service will be called and get request to 'Index' ActionResult will be done.

3.5.3. **Dao**

Payment details like payment amount, ordered, payment date and mode of payment will be saved in Razorpay Dashboard.

4. Problem and Solution

4.1. Binding : Object Binding (Class Create)

4.1.1. Initially we were trying to bind each variable declared in .ts file with HTML ng-model which made it difficult to send variables value as post request at server side.

4.1.2. In result, we made a class for each related component to make post requests with class objects whose member where binded with data fields in HTML.

4.2. Join JPA (@anotation)

4.3. Payment

4.3.1. Razorpay Class not defined because NuGet package Razorpay 3.0.0 was not installed

4.3.2. MVC and WebApi: Since MVC project was created as Payment Gateway with integration of Razorpay, we also wanted to get data from Razorpay Dashboard. To overcome this problem, WebApi project was created to make API request to Razorpay Dashboard to get data e.g. GET endpoint /orders/:id

4.3.3. Transfer data from Angular to MVC: We tried sending Object as url parameter but it did not succeed, so we passed variables as parameters in Query String.

4.3.4. Converted unix timestamp to time in JS: The Default datatype for the timestamp of Razorpay is Unix. so we had to convert unix timestamp to Js timestamp.

4.4. Git (Merge Conflict, package in Git repository)

4.4.1. We have a Main branch and a branch for each team member in each Git Repository.

4.4.2. When Main branch pull changes from all the branches, merge conflicts occur.

4.4.3. So, to resolve merge conflict, Main branch review all the files with conflict and commit the file.

4.5. Brand According Category

4.5.1. During Registration, all the Brand available in the database were fetched and displayed on Drop Down irrespective of their related Categories.

4.5.2. This could have resulted in integrity constraint conflict.

4.5.3. So whenever a user selects a Category name, we have filtered Brand data according to Category id and displayed it on the Brand drop down menu.

4.6. Session Implementation (Authorization, Admin Panel)

4.6.1. We used Session to Authorize each user and route them to their respective homepage.

4.7. Add Vehicle (Session Id and Manufacturer Id)

4.8. Spring data JPA query with parameter properties

4.9. Joining two table entities in Spring Data JPA

4.10. Spring data jpa save can not get id

4.10.1. When order details where stored in database, in result wanted to return this orederid to be send on the Payment gateway controller.

4.10.2. To overcome this problem we referred stackoverflow solution and retrieved orderid of current saved order data.

4.11. Set auto increment initial value for MySQL table using ALTER command

5. Learn

5.1. Join two or three tables in JPA

1. A ManyToOne relationship in Spring Boot is used to refer source object to target object and return reference of object relation occurs.
@ManyToOne(cascade=CascadeType.ALL) ,@JoinColumn(name = "categoryid") used in brand class for showing Many brands have one category .

2. User Repository

```
@Query("select new auto2auto.UserProfileResult(u.userid, u.firstname,
c.name,b.name ,u.usertype) from User u Join u.categoryid c Join u.brandid
b ")
        List<UserProfileResult> getUserProfile();
```

3. Here Join clause is used to get data from query with related fields.

5.2. Admin Navigation Tabs

1. Angular router is used to navigate through different tabs available in the Admin dashboard. When user click on any tab, routename matches exact to parameterized route array in app-routing module and related tab view is generated.
2. Parameterized route is implemented in the Admin dashboard to improve code efficiency instead of creating individual components for each tab available.
3. Router made code less complex.

5.3. Authentication and Authorization

1. User goes through two layers of security during login process ie. Authentication and Authorization
2. Authentication:
    a. This service is used for logging in the user. A User gets notified when input credentials in login fields are not satisfied.
    b. Authentication service notifies subscribed components whenever user logged in or logged out.
3. Authorization:
    a. After authentication service is verified, the user is redirected to authorization service to check the user privileges.
    b. According to user privilege right permissions, the application provides access to its related components like admin, manufacturer and retailer.

5.4. Integration of External Api and Payment Gateway

1. Razorpay as Payment Gateway has been implemented in this Project. Razorpay documentation helped us to integrate Razorpay Checkout form step by step.
2. We have install Razorpay NuGet Package in .NET MVC framework for Object requirement and its implementation in Checkout form.
3. To get Payment and Order related details from Razorpay Dashboard, Razorpay APIs and all their responses are returned in JSON.