

Vehicle Loan-Hub

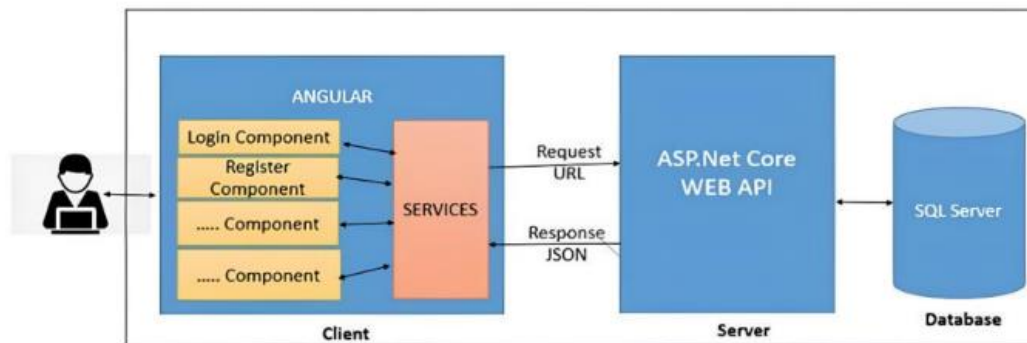
Overview:

Welcome to **VehicleLoanHub**, where applying for vehicle loans is seamless for users and efficient for administrators. Users can easily apply for loans for cars, trucks, or motorcycles through our intuitive platform, while administrators can manage applications effortlessly. With real-time updates and analytics, users can track their applications, and administrators can make informed decisions. VehicleLoanHub simplifies the financing process, ensuring a smooth experience for all involved.

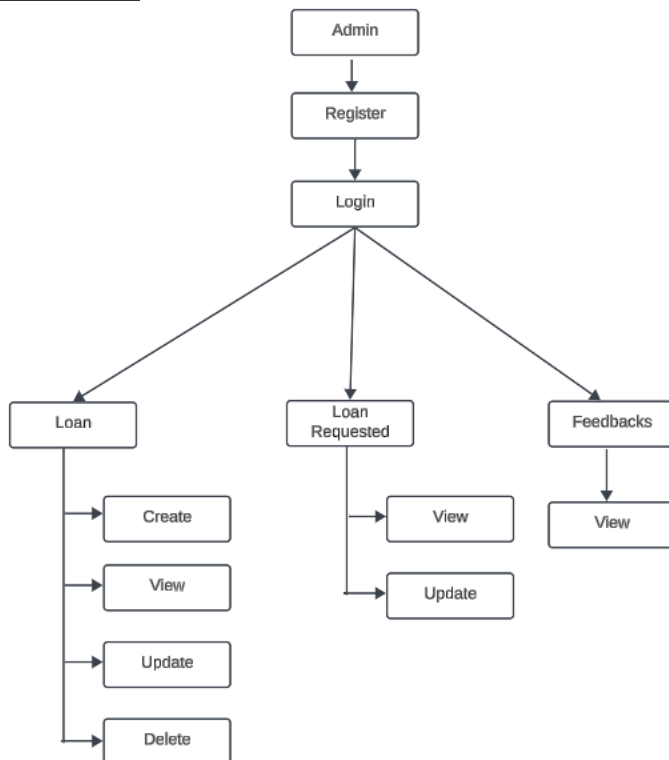
Users of the System:

1. Admin
2. User

System Architectural Diagram:



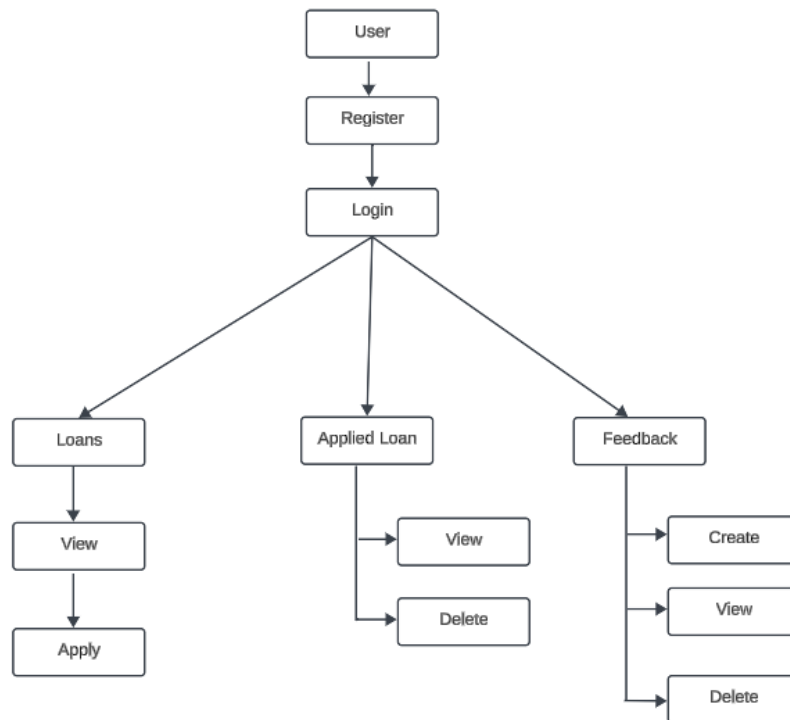
Admin Actions: Flow Diagram:



- **Loan:** Admins can view and create new loans by providing necessary information.
- **Loan Requested:** Admins can access and review applied loans, with the ability to update their status by approving or rejecting them.
- **Feedbacks:** Admins have the option to view user's feedback.

User Actions:

Flow Diagram:



- **Loan:** Users can view and apply for the list of loans.
- **Applied Loan:** Users can view the list of applied loans.
- **Feedback:** Users have the option to create and view their feedback.

Modules of the Application:

Admin:

1. Register
2. Login
3. Home
4. Loan
5. Loan Requested
6. Feedbacks

User:

1. Register
2. Login
3. Home
4. Loans
5. Applied Loan
6. Feedback

Technology Stack

Front End

Angular 10+, HTML, CSS

Back End

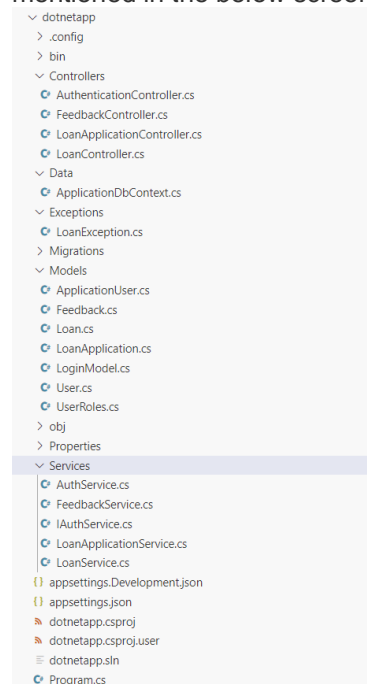
.NET Web API, EF Core, Microsoft SQL Server Database.

Application assumptions:

1. The login page should be the first page rendered when the application loads.
2. Manual routing should be restricted by implementing Auth Guard, utilizing the canActivate interface. For example, if the user enters as <http://localhost:8080/dashboard> or <http://localhost:8080/user> the page should not navigate to the corresponding page instead it should redirect to the login page.
3. Unless logged into the system, the user cannot navigate to any other pages.
4. Logging out must again redirect to the login page.

Backend Requirements:

Create folders named as **Models, Controllers, Services, Data and Exceptions** inside **dotnetapp** as mentioned in the below screenshot.



ApplicationDbContext: (/Data/ApplicationDbContext.cs)

Inside **Data** folder create **ApplicationDbContext** file with the following **DbSet** mentioned below

```
public DbSet<User> Users { get; set; }  
public DbSet<Loan> Loans { get; set; }  
public DbSet<LoanApplication> LoanApplications { get; set; }  
public DbSet<Feedback> Feedbacks { get; set; }
```

Model Classes:

Inside **Models** folder create all the model classes mentioned below.

Namespace: All the model classes should be located within the **dotnetapp.Models** namespace.

User (Models / User.cs):

This class stores the user role (Admin or User) and all user information.

Properties:

- UserId: int
- Email: string
- Password: string
- Username: string
- MobileNumber: string
- UserRole: string (**Admin/User**)

Loan (Models / Loan.cs):

This class stores information about a loan.

Properties:

- LoanId: int
- LoanType: string
- Description: string
- InterestRate: decimal
- MaximumAmount: decimal

LoanApplication (Models / LoanApplication.cs):

This class represents a loan application.

Properties:

- LoanApplicationId: int
- UserId: int
- User?: User
- LoanId: int
- Loan?: Loan
- SubmissionDate: DateTime
- Income: decimal
- Model: DateTime
- PurchasePrice: decimal
- LoanStatus: int
- Address: string
- File: string

Feedback (Models / Feedback.cs):

This class represents feedback submitted by users.

Properties:

- FeedbackId: int
- UserId: int
- User?: User
- FeedbackText: string
- Date: DateTime

LoginModel (Models / LoginModel.cs):

This class stores the email and password to authenticate the user during login.

Properties:

- Email: string
- Password: string

UserRoles (Models / UserRoles.cs):

This class defines constants for user roles.

Constants:

1. Admin: string - Represents the role of an admin user.
2. User: string - Represents the role of a regular user.

ApplicationUser (Models / ApplicationUser.cs):

This class represents a user in the application, inheriting from **IdentityUser** class.

Property:

- Name: string (Max length 30)

Exceptions: (Exceptions / LoanException.cs)

1. Inside **Exceptions** folder create the exception file named **LoanException(LoanException.cs)**.
2. **Purpose:** The **LoanException** class provides a mechanism for handling exceptions related to loan operations within the application.
3. **Namespace:** It should be located within the **dotnetapp.Exceptions** namespace.
4. **Inheritance:** Inherits from the base **Exception** class, enabling it to leverage existing exception handling mechanisms.
5. **Constructor:** Contains a constructor that accepts a message parameter, allowing to specify custom error messages when throwing exceptions.

For example, you might throw a **LoanException**

1. When attempting to **delete a loan that is referenced by a loan application**.
2. When trying to **add a loan with the same loan type** as an existing one.
3. When attempting to **apply for a loan that has already been applied** for by the same user.

Important note:

Implement database logic only in the **service file functions without using try-catch**. Use **try-catch only in the controller files** and call the service file functions inside it.

Services:

Inside "**Services**" folder create all the services file mentioned below.

Namespace: All the services file should be located within the **dotnetapp.Services** namespace.

LoanService (Services / LoanService.cs)

This service class provides methods to interact with loan data stored in the database.

Constructor:

```
public LoanService(ApplicationDbContext context)
{
    _context = context;
}
```

Functions:

1. **public async Task<IEnumerable<Loan>> GetAllLoans():**
 - a. Retrieves and returns all loans from the database.
2. **public async Task<Loan> GetLoanById(int loanId):**
 - a. Retrieves a loan from the database with the specified **loanId**.

3. **public async Task<bool> AddLoan(Loan loan):**

- a. Check if a loan with the same type already exists in the database.
- b. If a loan with the same loan type exists, throw a **LoanException** with the message "**Loan with the same type already exists**".
- c. If no loan with the same type exists, add the new loan to the database.
- d. Save changes asynchronously to the database.
- e. Returns **true** for the successfully insertion.

4. **public async Task<bool> UpdateLoan(int loanId, Loan loan):**

- a. If no loan with the specified **loanId** is found in the database, return **false**.
- b. Check if a loan with the same loan type already exists in the database.
- c. If a loan with the same loan type exists, throw a **LoanException** with the message "**Loan with the same type already exists**".
- d. If **no loan with the same type exists**, update the existing loan with the values from the provided loan object.
- e. Save changes asynchronously to the database.
- f. Return **true** for the successful update.

5. **public async Task<bool> DeleteLoan(int loanId):**

- a. Retrieve the loan from the database based on the provided **loanId**.
- b. Save changes asynchronously to the database.
- c. Check if the loan is referenced in any **LoanApplication**. If referenced, throw a **LoanException** with the message "**Loan cannot be deleted, it is referenced in loanapplication**".
- d. If the loan is not referenced by any **LoanApplication**, remove the loan from the database.
- e. Save changes asynchronously to the database.
- f. Return **true** for the successful delete.

LoanApplicationService (Services / LoanApplicationService.cs):

This service class provides methods to interact with loan application data stored in the database.

Constructor:

```
public LoanApplicationService(ApplicationDbContext context)
{
    _context = context;
}
```

Functions:

1. **public async Task<IEnumerable<LoanApplication>> GetAllLoanApplications():**

- a. Retrieves all loan applications from the database.

2. **public async Task<IEnumerable<LoanApplication>> GetLoanApplicationsByUserId(int userId):**

- a. Retrieves all loan applications associated with a specific **userId** from the database.

3. **public async Task<bool> AddLoanApplication(LoanApplication loanApplication):**

- a. Check if the user already applied for this loan. (Checks if there is any existing loan application for the same loan (**LoanId**) and user (**UserId**) combination in the database.)
- b. If such a loan application exists, it throws a **LoanException** with the message "**User already applied for this loan**".
- c. If not, add the new loan application to the database.
- d. Save changes asynchronously to the database.
- e. Return **true** for the successful insertion.

4. **public async Task<bool> UpdateLoanApplication(int loanApplicationId, LoanApplication loanApplication):**

- a. Retrieve the existing loan application from the database with the specified **loanApplicationId**.
- b. If no loan application with the specified **loanApplicationId** is found, return **false**.
- c. If found, update the loan application with the values from the provided **loanApplication** object.
- d. Save changes asynchronously to the database.
- e. Return **true** for the successful update.

5. **public async Task<bool> DeleteLoanApplication(int loanApplicationId):**

- a. Retrieve the existing loan application from the database with the specified **loanApplicationId**.

- b. If no loan application with the specified loanApplicationId is found, return **false**.
- c. If found, **delete the loan application** with the provided loanApplicationId.
- d. Save changes asynchronously to the database.
- e. Return **true** for the successful delete.

FeedbackService (Services / FeedbackService.cs):

This service class provides methods to interact with feedback data stored in the database.

Constructor:

```
public FeedbackService(ApplicationDbContext context)
{
    _context = context;
}
```

Functions:

1. public async Task<IEnumerable<Feedback>> GetAllFeedbacks():

- a. Retrieves all feedbacks from the database.

2. public async Task<IEnumerable<Feedback>> GetFeedbacksByUserId(int userId):

- a. Retrieves all feedbacks associated with a specific **userId** from the database.

3. public async Task<bool> AddFeedback(Feedback feedback):

- a. Adds new feedback to the database.
- b. Return **true** for the successful insertion.

4. public async Task<bool> DeleteFeedback(int feedbackId):

- a. Retrieve the existing feedback from the database with the specified **feedbackId**.
- b. If no feedback with the specified feedbackId is found, return **false**.
- c. If found, delete the feedback with the provided feedbackId.
- d. Save changes asynchronously to the database.
- e. Return **true** for the successful delete.

AuthService (Services / AuthService.cs):

The **AuthService** class is responsible for user authentication and authorization.

Constructor:

```
public AuthService(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole>
roleManager, IConfiguration configuration, ApplicationDbContext context)
{
    this.userManager = userManager;
    this.roleManager = roleManager;
    _configuration = configuration;
    _context = context;
}
```

Functions:

1. public async Task<(int, string)> Registration (User model, string role):

- a. Check if the email already exists in the database. If so return **"User already exists"**.
- b. Registers a new user with the provided details and assigns a role.
- c. If any error occurs return **"User creation failed! Please check user details and try again"**.
- d. Return **"User created successfully!"** for the successful register.

2. public async Task<(int, string)> Login (LoginModel model):

- a. Find user by email in the database.
- b. Check if user exists, if not return **"Invalid email"**.
- c. If the user exists, check the password is correct, if not return **"Invalid password"**.
- d. Logs in a user with the provided credentials and generates a **JWT** token for authentication.

3. private string GenerateToken(IEnumerable<Claim> claims):

- a. Generates a JWT token based on the provided claims.

IAuthService (Services / IAuthService.cs):

The **IAuthService** is an interface that defines methods for user registration and login.

Methods:

1. Task< (int, string)> Registration (User model, string role);
2. Task< (int, string)> Login (LoginModel model);

Controllers:

Inside "**Controllers**" folder create all the controllers file mentioned below.

Namespace: All the controllers file should located within the **dotnetapp.Controllers** namespace.

AuthenticationController (Controllers / AuthenticationController.cs):

This controller handles user authentication and registration requests.

Functions:

1. public async Task<ActionResult> Login(LoginModel model)

- a. Accepts login requests, validates the payload, and calls the authentication service to perform user login.
- b. It utilizes **_authService.Login(model)** method.
- c. Returns a **200 OK response** with a JWT token upon successful login.
- d. If an exception occurs during the process, it returns a **500 Internal Server Error** response with the exception message.

2. public async Task<ActionResult> Register(User model):

- a. Accepts registration requests, validates the payload. If fails, then returns error.
- b. Calls the authentication service to register a new user(**_authService.Registration(model, model.UserRole)**). Returns a **200 OK response** with success message upon successful registration.
- c. If an exception occurs during the process, it returns a **500 Internal Server Error** response with the exception message.

LoanController (Controllers / LoanController.cs):

This controller manages **loans**, interacting with the **LoanService** to perform CRUD operations.

Functions:

1. public async Task<ActionResult<IEnumerable<Loan>>> GetAllLoans():

- a. Implement the logic inside **try-catch block**.
- b. The **GetAllLoans** method is a controller action responsible for retrieving all loans.
- c. It calls the **_loanService.GetAllLoans()** method to fetch all loan from the service layer.
- d. It returns a **200 OK response** with the retrieved loan applications.

2. public async Task<ActionResult<Loan>> GetLoanById(int loanId):

- a. Implement the logic inside **try-catch block**.
- b. The **GetLoanById** method is a controller action responsible for retrieving a loan by its ID.
- c. It calls the **_loanService.GetLoanById(loanId)** method to retrieve the loan from the service layer.
- d. If the loan is not found, it returns a **404 Not Found response** with a message "**Cannot find any loan**".
- e. If the loan is found, it returns a **200 OK response with the loan data**.

3. public async Task<ActionResult> AddLoan([FromBody] Loan loan):

- a. Implement the logic inside **try-catch block**.
- b. The **AddLoan** method is a controller action responsible for adding a loan.
- c. It receives the loan data in the request body.
- d. It tries to add the loan using the **_loanService.AddLoan(loan)** method.
- e. If adding the loan is successful, it returns a 200 OK response with a success message "**Loan added successfully**".
- f. If adding the loan fails, it returns a **500 Internal Server Error response** with a failure message "**Failed to add loan**".

g. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

4. public async Task<ActionResult> UpdateLoan(int loanId, [FromBody] Loan loan):

- a. Implement the logic inside **try-catch block**.
- b. The **UpdateLoan** method is a controller action responsible for updating a loan.
- c. It receives the loan ID and updated loan data in the request body.
- d. It tries to update the loan using the **_loanService.UpdateLoan(loanId, loan)** method.
- e. If the update is successful, it returns a 200 OK response with a success message "**Loan updated successfully**".
- f. If the loan is not found, it returns a **404 Not Found response** with a message "**Cannot find any loan**".
- g. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

5. public async Task<ActionResult> DeleteLoan(int loanId):

- a. Implement the logic inside **try-catch block**.
- b. The **DeleteLoan** method is a controller action responsible for deleting a loan.
- c. It receives the loan ID to be deleted.
- d. It tries to delete the loan using the **_loanService.DeleteLoan(loanId)** method.
- e. If the deletion is successful, it returns a 200 OK response with a success message "**Loan deleted successfully**".
- f. If the loan is not found, it returns a **404 Not Found response** with a message "**Cannot find any loan**".
- g. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

LoanApplicationController (Controllers / LoanApplicationController.cs):

This controller manages loan applications, interacting with the LoanApplicationService to perform CRUD operations.

Functions:

1. public async Task<ActionResult<IEnumerable<LoanApplication>>>

GetAllLoanApplications():

- a. Implement the logic inside **try-catch block**.
- b. The **GetAllLoanApplications** method is a controller action responsible for retrieving all loan applications.
- c. It calls the **_loanApplicationService.GetAllLoanApplications()** method to fetch all loan applications from the service layer.
- d. It returns a **200 OK response** with the retrieved loan applications.

2. public async Task<ActionResult<LoanApplication>> GetLoanApplicationByUserId(int userId):

- a. Implement the logic inside **try-catch block**.
- b. The **GetLoanApplicationByUserId** method is a controller action responsible for retrieving loan applications by user ID.
- c. It calls the **_loanApplicationService.GetLoanApplicationsByUserId(userId)** method to fetch loan applications for the specified user from the service layer.
- d. It checks if any loan applications are found. If not, it returns a **404 Not Found response** with a message "**Cannot find any loan application**".
- e. If loan applications are found, it returns a **200 OK response** with the retrieved loan applications.

3. public async Task<ActionResult> AddLoanApplication([FromBody] LoanApplication loanApplication):

- a. Implement the logic inside **try-catch block**.
- b. The **AddLoanApplication** method is a controller action responsible for adding a new loan application.
- c. It receives the loan application data in the request body.
- d. It tries to add the loan application using the **_loanApplicationService.AddLoanApplication(loanApplication)** method.

- e. If adding the loan application is successful, it returns a **200 OK response with a success message “Loan application added successfully”**.
- f. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

4. public async Task<ActionResult> UpdateLoanApplication(int loanApplicationId, [FromBody] LoanApplication loanApplication)

- a. Implement the logic inside **try-catch block**.
- b. The **UpdateLoanApplication** method is a controller action responsible for updating a loan application.
- c. It receives the **loanApplicationId** and updated loan application data in the request body.
- d. It tries to update the loan application using the **_loanApplicationService.UpdateLoanApplication(loanApplicationId, loanApplication)** method.
- e. If the update is successful, it returns a **200 OK response with a success message “Loan application updated successfully”**.
- f. If the loan application is not found, it returns a **404 Not Found response with a message “Cannot find any loan application”**.
- g. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

5. public async Task<ActionResult> DeleteLoanApplication(int loanApplicationId, [FromBody] LoanApplication loanApplication):

- a. Implement the logic inside **try-catch block**.
- b. The **DeleteLoanApplication** method is a controller action responsible for deleting a loan application.
- c. It receives the **loanApplicationId** to be deleted.
- d. It tries to delete the loan application using the **_loanApplicationService.DeleteLoanApplication(loanApplicationId)** method.
- e. If the deletion is successful, it returns a **200 OK response with a success message “Loan application deleted successfully”**.
- f. If the loan application is not found, it returns a **404 Not Found response with a message “Cannot find any loan application”**.
- g. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

FeedbackController (Controllers / FeedbackController.cs):

This controller manages feedbacks, interacting with the **FeedbackService** to perform CRUD operations.

Functions:

1. public async Task<ActionResult<IEnumerable<Feedback>>> GetAllFeedbacks():

- a. Implement the logic inside **try-catch block**.
- b. The **GetAllFeedbacks** method is a controller action responsible for retrieving all feedbacks.
- c. It tries to get all feedbacks using the **_feedbackService.GetAllFeedbacks()** method.
- d. If the operation is successful, it returns a **200 OK response** with the retrieved feedbacks.
- e. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

2. public async Task<ActionResult<IEnumerable<Feedback>>> GetFeedbacksByUserId(int userId):

- a. Implement the logic inside **try-catch block**.
- b. The **GetFeedbacksByUserId** method is a controller action responsible for retrieving feedbacks by **userId**.
- c. It tries to get feedbacks by **userId** using the **_feedbackService.GetFeedbacksByUserId(userId)** method.
- d. If feedbacks are found, it returns a **200 OK response** with the retrieved feedbacks.
- e. If an exception occurs during the process, it returns a **500 Internal Server Error response** with the exception message.

3. public async Task<ActionResult> AddFeedback([FromBody] Feedback feedback):

- a. Implement the logic inside **try-catch block**.
- b. The **AddFeedback** method is a controller action responsible for adding a new feedback.
- c. It receives the feedback data in the request body.
- d. It tries to add the feedback using the **_feedbackService.AddFeedback(feedback)** method.

- e. If adding the feedback is successful, it returns a **200 OK response** with a success message **“Feedback added successfully”**.
- f. If an exception occurs during the process, it returns a **500 Internal Server Error** response with the exception message.

4. `public async Task<ActionResult> DeleteFeedback(int feedbackId):`

- a. Implement the logic inside **try-catch block**.
- b. The **DeleteFeedback** method is a controller action responsible for deleting a feedback.
- c. It receives the **feedbackId** to be deleted.
- d. It tries to delete the feedback using the **_feedbackService.DeleteFeedback(feedbackId)** method.
- e. If the deletion is successful, it returns a **200 OK response** with a success message **“Feedback deleted successfully”**.
- f. If the feedback is not found, it returns a **404 Not Found** response with a message **“Cannot find any feedback”**.
- g. If an exception occurs during the process, it returns a **500 Internal Server Error** response with the exception message.

Endpoints:

Authentication		^
POST	/api/login	▼
POST	/api/register	▼
Feedback		^
GET	/api/feedback	▼
POST	/api/feedback	▼
GET	/api/feedback/user/{userId}	▼
DELETE	/api/feedback/{feedbackId}	▼
Loan		^
GET	/api/loan	▼
POST	/api/loan	▼
GET	/api/loan/{loanId}	▼
PUT	/api/loan/{loanId}	▼
DELETE	/api/loan/{loanId}	▼
LoanApplication		^
GET	/api/loan-application	▼
POST	/api/loan-application	▼
GET	/api/loan-application/user/{userId}	▼
PUT	/api/loan-application/{loanApplicationId}	▼
DELETE	/api/loan-application/{loanApplicationId}	▼

1. Login: [Access for both Admin and User]

Endpoint name: “/api/login”

Method: POST

Request body:

```
{
  "Email": "string",
  "Password": "string"
}
```

Response:**2. Register: [Access for both Admin and User]****Endpoint name:** "/api/register"**Method:** POST**Request body:**

```
{
  "Username": "string",
  "Email": "user@example.com",
  "MobileNumber": "9876541221",
  "Password": "Pass@2425",
  "UserRole": "string"
}
```

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
400 BadRequest	JSON object containing Error message.
500	JSON object containing Error message.

3. Get all loans: [Access for both Admin and User]**Endpoint name:** "/api/loan"**Method:** GET**Response:**

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing all the loans.
500	JSON object containing Error message.

4. Add loan: [Access for only Admin]**Endpoint name:** "/api/loan"**Method:** POST**Request body:**

```
{
  "LoanType": "string",
  "Description": "string",
  "InterestRate": 10,
  "MaximumAmount": 10
}
```

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
400 BadRequest	JSON object containing Error message.
500	JSON object containing Error message.

5. Get specific loan: [Access for only Admin]

Endpoint name: "/api/loan/{loanId}"

Method: GET

Parameter: loanId

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing loan details.
404 NotFound	JSON object containing Error message.
500	JSON object containing Error message.

6. Update loan: [Access for only Admin]

Endpoint name: "/api/loan/{loanId}"

Method: PUT

Parameter: loanId

Request body:

```
{
  "LoanType": "string",
  "Description": "string",
  "InterestRate": 10,
  "MaximumAmount":10
}
```

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
404 NotFound	JSON object containing Error message.
500	JSON object containing Error message.

7. Delete loan: [Access for only Admin]

Endpoint name: "/api/loan/{loanId}"

Method: DELETE

Parameter: loanId

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
404 NotFound	JSON object containing Error message.
500	JSON object containing Error message.

8. Get all loan applications: [Access for only Admin]

Endpoint name: "/api/loan-application"

Method: GET

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing all the loan applications.
500	JSON object containing Error message.

9. Add loan application: [Access for only User]

Endpoint name: "/api/loan-application"

Method: POST

Request body:

```
{  
  "UserId": 1,  
  "LoanId": 1,  
  "SubmissionDate": "2024-05-05T12:00:42.743Z",  
  "Income": 0,  
  "Model": "2024-05-05T12:00:42.743Z",  
  "PurchasePrice": 0,  
  "LoanStatus": 0,  
  "Address": "string",  
  "File": "string"  
}
```

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
400 BadRequest	JSON object containing Error message.
500	JSON object containing Error message.

10. Get loan application specific to user: [Access for only User]

Endpoint name: "/api/loan-application/{userId}"

Method: GET
Parameter: userId
Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing loan applications details.
500	JSON object containing Error message.

11. Update loan application: [Access for both Admin and User]

Endpoint name: "/api/loan-application/{loanApplicationId}"

Method: PUT

Parameter: loanApplicationId

Request body:

```
{
  "UserId": 0,
  "LoanId": 0,
  "SubmissionDate": "2024-05-05T12:00:42.743Z",
  "Income": 0,
  "Model": "2024-05-05T12:00:42.743Z",
  "PurchasePrice": 0,
  "LoanStatus": 0,
  "Address": "string",
  "File": "string"
}
```

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
404 NotFound	JSON object containing Error message.
500	JSON object containing Error message.

12. Delete loan application: [Access for only User]

Endpoint name: "/api/loan-application/{loanApplicationId}"

Method: DELETE

Parameter: loanApplicationId

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
404 NotFound	JSON object containing Error message.
500	JSON object containing Error message.

13. Get all feedbacks: [Access for only Admin]

Endpoint name: "/api/feedback"

Method: GET

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing all feedback.
500	JSON object containing Error message.

14. Add feedback: [Access for only User]

Endpoint name: "/api/feedback"

Method: POST

Request body:

```
{  
  "UserId": 0,  
  "FeedbackText": "string",  
  "Date": "2024-07-07T12:28:56.927Z"  
}
```

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
400 BadRequest	JSON object containing Error message.
500	JSON object containing Error message.

15. Get feedback specific to user: [Access for only User]

Endpoint name: "/api/feedback/{userId}"

Method: GET

Parameter: userId

Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing feedback details specific to user.
500	JSON object containing Error message.

16. Delete feedback: [Access for only User]

Endpoint name: "/api/feedback/{feedbackId}"

Method: DELETE

Parameter: feedbackId

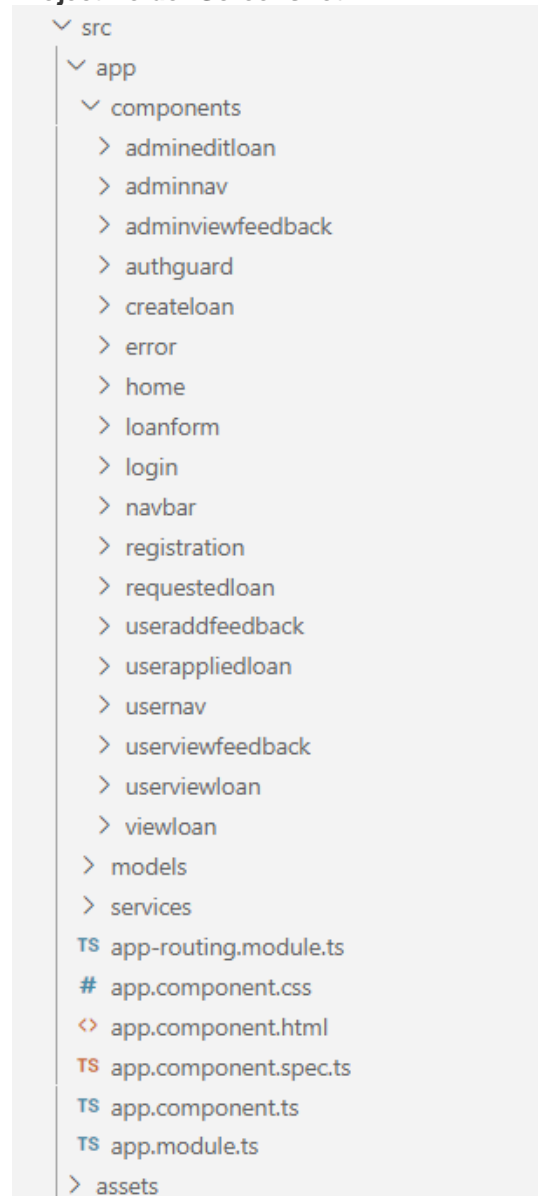
Response:

Status Code	Response body
200 (HttpStatusCode OK)	JSON object containing success message.
404 NotFound	JSON object containing Error message.
500	JSON object containing Error message.

Frontend Requirements:

- Create a folder named components inside the app to store all the components. (Refer project structure screenshots).
- Create a folder named models inside app to store all the model interface.
- Create a folder named as services inside app to implement all the services.
- Create model interface referring the backend entities (User, Loan, Loanapplication, Feedback) mentioned in the backend requirements accordingly.
- You can create your own components based on the application requirements.
- Import model files, services and components as required.

Project Folder Screenshot:



Services and Models:

- ▼ src
 - ▼ app
 - > components
 - ▼ models
 - TS feedback.model.ts
 - TS loan.model.ts
 - TS loanapplication.model.ts
 - TS login.model.ts
 - TS user.model.ts
 - ▼ services
 - TS auth.service.spec.ts
 - TS auth.service.ts
 - TS feedback.service.spec.ts
 - TS feedback.service.ts
 - TS loan.service.spec.ts
 - TS loan.service.ts
 - TS app-routing.module.ts
 - # app.component.css
 - <> app.component.html
 - TS app.component.spec.ts
 - TS app.component.ts
 - TS app.module.ts
 - > assets
 - > environments
 - TS apiconfig.ts
 - ★ favicon.ico
 - <> index.html
 - TS main.ts
 - TS polyfills.ts
 - # styles.css

Frontend Models:

User Model:

```
class User {
  UserId?: number;
  Email: string;
  Password: string;
  Username: string;
  MobileNumber: string;
  UserRole: string;
}
```

Login Model:

```
class Login {
```

```
Email: string;
Password: string;
}
```

LoanApplication Model:

```
interface LoanApplication {
  LoanApplicationId?: number;
  UserId?: number;
  UserName: string;
  LoanId?: number;
  SubmissionDate: String;
  Income: number;
  Model: string;
  PurchasePrice: number;
  LoanStatus: number;
  Address: string;
  File: string;
}
```

Loan Model:

```
interface Loan {
  LoanId?:number;
  LoanType: string;
  Description: string;
  InterestRate: number;
  MaximumAmount: number;
}
```

Feedback Model:

```
class Feedback {
  FeedbackId?: number;
  UserId: number;
  FeedbackText: string;
  Date: Date;
}
```

Frontend services:

- Declare a public property `apiUrl` to store the backend URL in all the services.
- For example, public `apiUrl` = 'http://localhost:8080'. Instead of 'localhost', replace it with the URL of your workspace port 8080 URL.
- For the API's to be used please refer the API Table.
- Authorized token to be passed in headers for all end points.

1. AuthService(auth.service.ts):

- Create a service name as **auth** inside `app/services` folder to implement the following functions.

Methods Overview:

- `register(user: User): Observable<any>`:
 - Use this method to register a new user. It sends a POST request to the '/api/register' endpoint with the user data provided as the body.
- `login(login : Login): Observable<any>`:
 - This method is used to authenticate a user by logging them in. It sends a POST request to the '/api/login' endpoint with the user's email and password. Upon successful login, it stores the JWT token in `localStorage` and updates the user's role and ID using `BehaviorSubjects`.

2. LoanService(loan.service.ts):

- Create a service name as **loan** inside `app/services` and implement the following functions in it.

Methods Overview:

- `getAllLoans(): Observable<Loan[]>`:

- Use this method to fetch all loans from the server. It sends a GET request to the '/api/loan' endpoint with the authorization token prefixed with 'Bearer' stored in localStorage.
- deleteLoan(loanId: string): Observable<void>:
• Call this method to delete a loan with the specified ID. It sends a DELETE request to the '/api/loan/:id' endpoint with the loan ID and the authorization token prefixed with 'Bearer' stored in localStorage.
- getLoanById(id: string): Observable<Loan>:
• Use this method to retrieve a loan by its ID. It sends a GET request to the '/api/loan/:id' endpoint with the loan ID and the authorization token prefixed with 'Bearer' stored in localStorage.
- addLoan(requestObject: Loan): Observable<Loan>:
• Call this method to add a new loan. It sends a POST request to the '/api/loan' endpoint with the loan data provided as the requestObject and the authorization token prefixed with 'Bearer' stored in localStorage.
- updateLoan(id: string, requestObject: Loan): Observable<Loan>:
• Use this method to update an existing loan. It sends a PUT request to the '/api/loan/:id' endpoint with the loan ID and the updated loan data provided as the requestObject, along with the authorization token prefixed with 'Bearer' stored in localStorage.
- getAppliedLoans(userId: string): Observable<LoanApplication[]>:
• This method retrieves all loan applications submitted by a specific user. It sends a GET request to the '/api/loan-application/user/:userId' endpoint with the user ID and the authorization token prefixed with 'Bearer' stored in localStorage.
- deleteLoanApplication(loanId: string): Observable<void>:
• Call this method to delete a loan application with the specified ID. It sends a DELETE request to the '/api/loan-application/:id' endpoint with the loan application ID and the authorization token prefixed with 'Bearer' stored in localStorage.
- addLoanApplication(data: LoanApplication): Observable<LoanApplication>:
• Use this method to submit a new loan application. It sends a POST request to the '/api/loan-application' endpoint with the loan application data provided as the data parameter and the authorization token prefixed with 'Bearer' stored in localStorage.
- getAllLoanApplications(): Observable<LoanApplication[]>:
• This method fetches all loan applications from the server. It sends a GET request to the '/api/loan-application' endpoint with the authorization token prefixed with 'Bearer' stored in localStorage.
- updateLoanStatus(id: string, loanApplication: LoanApplication): Observable<LoanApplication>:
• Call this method to update the status of a loan application. It sends a PUT request to the '/api/loan-application/:id' endpoint with the loan application ID, updated loan application data, and the authorization token prefixed with 'Bearer' stored in localStorage.

3. FeedbackService(feedback.service.ts):

- Create a service name as **feedback** inside app/services and implement the following functions in it.

Methods Overview:

- `sendFeedback(feedback: Feedback): Observable<Feedback>`:
Use this method to send feedback to the server. It sends a POST request to the `/api/feedback` endpoint with the feedback data provided and the authorization token prefixed with 'Bearer' stored in `localStorage`.
- `getAllFeedbacksByUserId(userId: string): Observable<Feedback[]>`:
This method retrieves all feedbacks submitted by a specific user. It sends a GET request to the `/api/feedback/user/:userId` endpoint with the user ID and the authorization token prefixed with 'Bearer' stored in `localStorage`.
- `deleteFeedback(feedbackId: string): Observable<void>`:
Call this method to delete a feedback with the specified ID. It sends a DELETE request to the `/api/feedback/:feedbackId` endpoint with the feedback ID and the authorization token prefixed with 'Bearer' stored in `localStorage`.
- `getFeedbacks(): Observable<Feedback[]>`:
This method fetches all feedbacks from the server. It sends a GET request to the `/api/feedback` endpoint with the authorization token prefixed with 'Bearer' stored in `localStorage`.

Validations:

Client-Side Validation:

- Implement client-side validation using HTML5 attributes and JavaScript to validate user input before making API requests.
- Provide immediate feedback to users for invalid input, such as displaying error messages near the input fields.

Server-Side Validation:

- Implement server-side validation in the controllers to ensure data integrity.
- Validate user input and API responses to prevent unexpected or malicious data from affecting the application.
- Return appropriate validation error messages to the user interface for any validation failures.

Exception Handling:

- Implement exception handling mechanisms in the controllers to gracefully handle errors and exceptions. Define custom exception classes for different error scenarios, such as API communication errors or database errors.
- Log exceptions for debugging purposes while presenting user-friendly error messages to users. Record all the exceptions and errors handled store in separate table "ErrorLogs".

Error Pages:

Create custom error pages for different HTTP status codes (e.g., 404 Not Found, 500 Internal Server Error) to provide a consistent and user-friendly error experience. Ensure that error pages contain helpful information and guidance for users.

Thus, create a reliable and user-friendly web application that not only meets user expectations but also provides a robust and secure experience, even when faced with unexpected situations. Error page has to be displayed if something goes wrong.

Something Went Wrong

We're sorry, but an error occurred. Please try again later.

Frontend Screenshots:

- All the asterisk (*) marked fields are mandatory in the form. Make sure to mark all the field names with * symbol followed by the validations.

Navigation Bar: (navbar component)

Component displays a title along with router links to "Register" and "Login".

Landing Page: (home component)

Vehicle Loan Hub

Vehicle Loan Hub is a digital platform for those seeking reliable and flexible vehicle loan options. At the heart of the platform is a wide and varied collection of loan offerings, each enriched with detailed descriptions, competitive interest rates, and comprehensive terms and conditions. This wealth of information enables users to make informed decisions, ensuring that their selected loan aligns seamlessly with their financial capabilities and preferences. Vehicle Loan Hub is designed to meet the diverse needs of its users, offering a seamless and user-friendly loan application experience for securing funds for their dream vehicle.

Component features a heading "Vehicle Loan Hub" accompanied by an introductory message that provides an overview of the application.

User side (Admin and User):

Registration Page: (registration component)

Clicking "Register" in the navbar displays the registration page for both roles. Please refer to the screenshots below for validations.



Registration

Username*

Email*

Password*

Confirm Password*

Mobile Number*

Role*

Select a role

Register



Registration

Username*

*Username is required

Email*

*Email is required

Password*

*Password is required

Confirm Password*

*Confirm Password is required

Mobile Number*

*Mobile number is required

Role*

Select a role

*Role is required

Register



Registration

Username*

Email*

*Please enter a valid email

Password*

*Password must include at least one uppercase letter, one lowercase letter, one digit, and one special character

Confirm Password*

*Passwords do not match

Mobile Number*

*Mobile number must be 10 digits

Role*

ADMIN

Register

VEHICLE LOAN-HUB

Register Login

Registration

Username*

Email*

Password*

Confirm Password*

Mobile Number*

Role*

ADMIN

Register

When the "Register" button is clicked, upon successful submission, the user must be navigated to the login page.

VEHICLE LOAN-HUB

Register Login

Registration

Username*

Email*

Password*

Confirm Password*

Mobile Number*

Role*

ADMIN

***User already exists**

Register

If a user attempts to register with an existing email, a message stating "User already exists" will be displayed.

Login Page (login component)

This page is used for logging in to the application. On providing the valid email and password, the user will be logged in.

VEHICLE LOAN-HUB

Register Login

Login

Email*

Password*

Login

Don't have an account? [Register here](#)

Perform validations for email and password fields.

VEHICLE LOAN-HUB

RegisterLogin

Login

Email*

*Email is required

Password*

*Password is required

Login

Don't have an account? [Register here](#)

VEHICLE LOAN-HUB

RegisterLogin

Login

Email*

demoadmin@gmail.com

*Please enter a valid email address

Password*

Login

Don't have an account? [Register here](#)

VEHICLE LOAN-HUB

RegisterLogin

Login

Email*

arjuman@gmail.com

Password*

*Invalid email or password

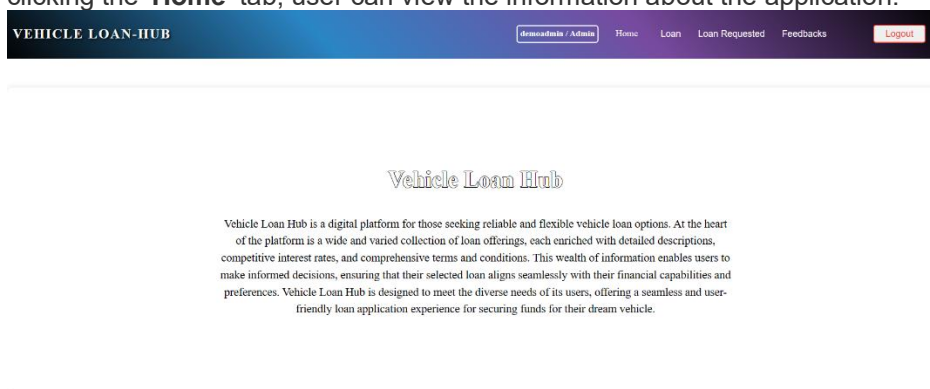
Login

Don't have an account? [Register here](#)

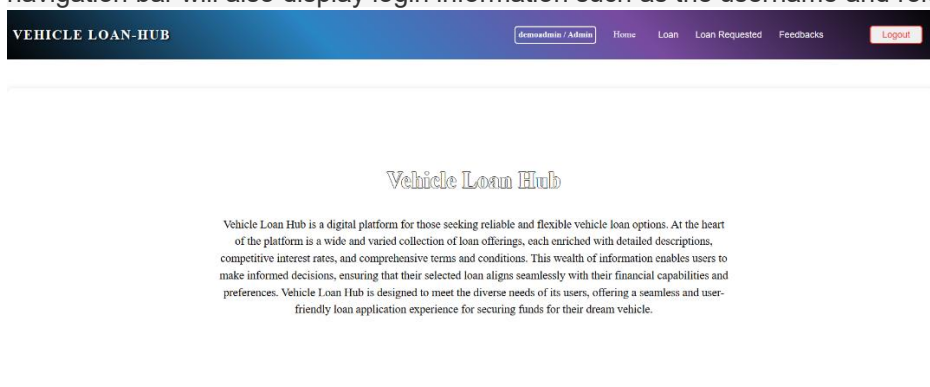
On Clicking the 'Login' button, user will be navigated to the (adminnav) based on their roles.

Admin side:

Home Component: This page is used to display the information about the loan hub application. On clicking the '**Home**' tab, user can view the information about the application.

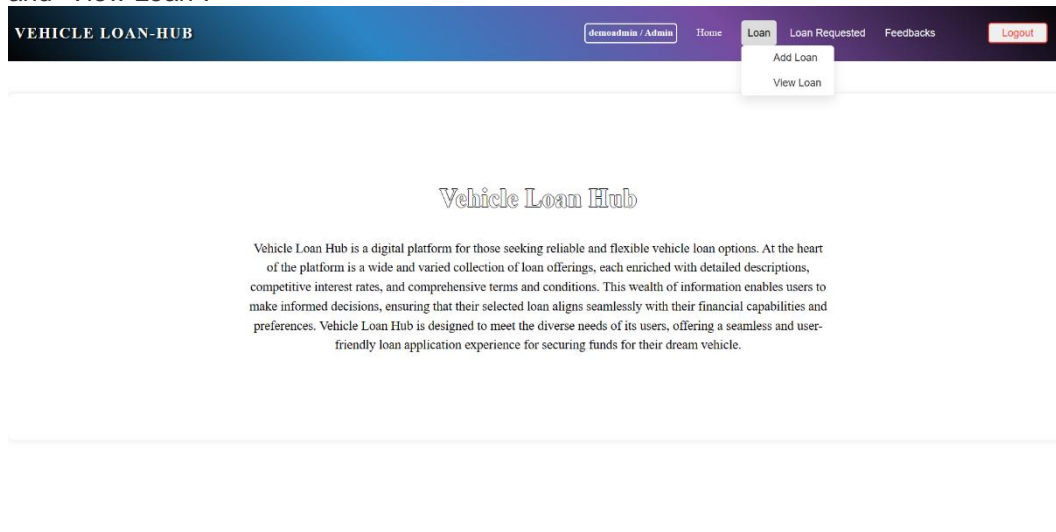


Upon successful login, if the user is an admin, the (**adminnav component**) will be displayed. If the user is a regular user, the (**usernav component**) will be displayed. Additionally, the role-based navigation bar will also display login information such as the username and role



Admins can navigate to other pages by clicking on the menu available in the navigation bar.

On hovering over the "Loan" item in the navbar, a submenu should appear with options to "Add Loan" and "View Loan".



Clicking on "Add Loan" will navigate to the **createloan component**, which displays a form with the heading "**Create New Loan**". The admin should be able to add the loan details on this page, which is used to add a new loan to the system.

The screenshot shows the "Create New Loan" form. The form has a title "Create New Loan" and four input fields: "Loan Type*", "Description*", "Interest Rate*", and "Maximum Amount*". Each field has a small red asterisk indicating it is required. Below the input fields is a blue "Submit" button.

Perform validations for all the form fields.

VEHICLE LOAN-HUB

demoadmin / Admin

Home

Loan

Loan Requested

Feedbacks

Logout

Create New Loan

Loan Type*

Loan Type

*Loan Type is required

Description*

Loan Description

*Description is required

Interest Rate*

Interest Rate

*Interest Rate is required

Maximum Amount*

Maximum Amount

*Maximum Amount is required

Submit

Clicking the "Submit" button with empty fields will display a validation message stating "All fields are required".

VEHICLE LOAN-HUB

demoadmin / Admin

Home

Loan

Loan Requested

Feedbacks

Logout

Create New Loan

Loan Type*

Loan Type

Description*

Loan Description

Interest Rate*

Interest Rate

Maximum Amount*

Maximum Amount

*All fields are required

Submit

VEHICLE LOAN-HUB

demoadmin / Admin

Home

Loan

Loan Requested

Feedbacks

Logout

Create New Loan

Loan Type*

Car

Description*

This loan is for purchasing a new car.

Interest Rate*

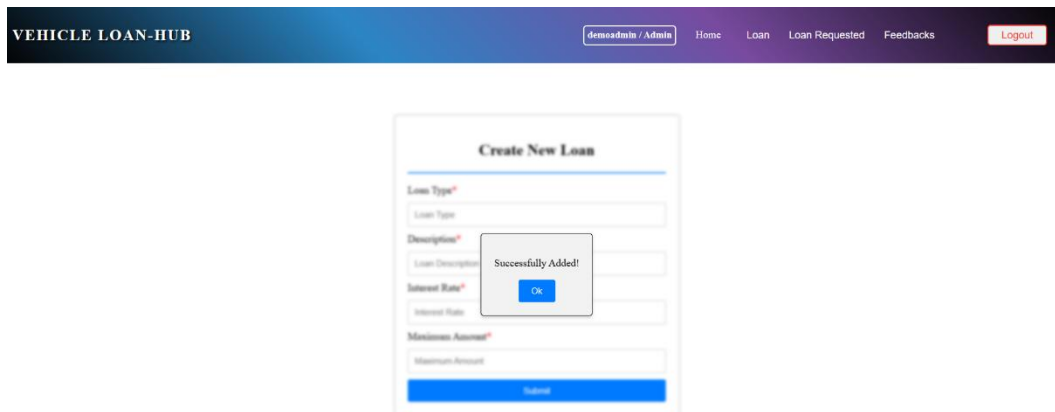
7.9

Maximum Amount*

160000

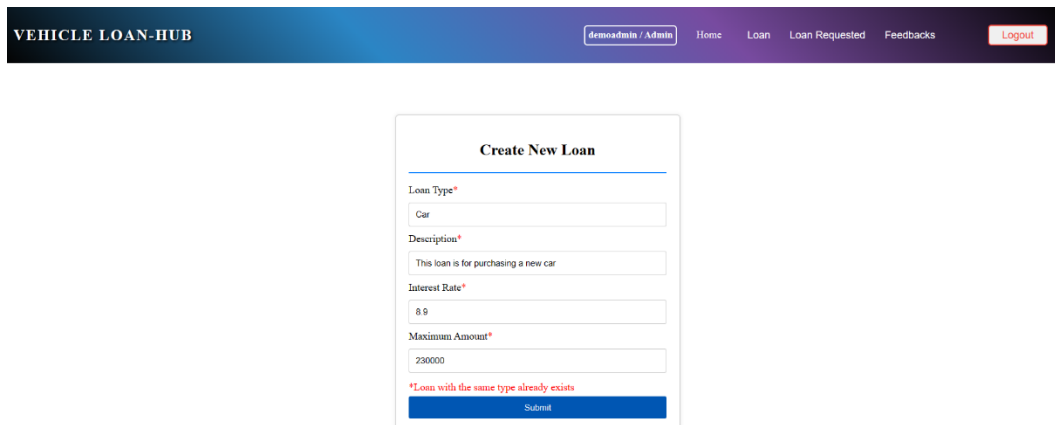
Submit

Upon clicking the "Submit" button, if the operation is successful, a popup message saying "Successfully Added!" should be displayed.



The screenshot shows the 'VEHICLE LOAN-HUB' header with a navigation bar containing 'demoadmin / Admin', 'Home', 'Loan', 'Loan Requested', 'Feedbacks', and a 'Logout' button. Below the header is the 'Create New Loan' form. The form has five input fields: 'Loan Type', 'Description', 'Interest Rate', and 'Maximum Amount'. A modal dialog box is displayed in the center of the form, containing the text 'Successfully Added!' and an 'Ok' button. The 'Submit' button is at the bottom of the form.

If the admin adds a new loan with an existing loan type, an error message stating "Loan with the same type already exists" should be displayed.



The screenshot shows the 'VEHICLE LOAN-HUB' header with the same navigation bar. Below the header is the 'Create New Loan' form. The form has five input fields: 'Loan Type' (with 'Car' entered), 'Description' (with 'This loan is for purchasing a new car' entered), 'Interest Rate' (with '8.9' entered), and 'Maximum Amount' (with '230000' entered). Below the input fields, a red error message is displayed: '*Loan with the same type already exists'. The 'Submit' button is at the bottom of the form.

On Clicking the 'Submit' button a loan is added to the system and admin can add a new loan again. To move to other pages admin can click any of the menus available in the navbar.

Admin View Loan

On hovering over the "Loan" item in the navbar, a submenu should appear with options to "Add Loan" and "View Loan".

Clicking on "View Loan" will navigate to the **viewloan component**, which displays all loan details in a table format with the heading "Vehicle Loans". Additionally, a search feature is provided to search based on LoanType and Description.

Vehicle Loans

S.No	LoanType	Maximum Amount	Interest Rate	Description	Action
1	Car	₹160000	7.9%	This loan is for purchasing a new car.	<div>EditDelete</div>
2	Truck	₹230000	6.9%	This loan is for financing a new truck.	<div>EditDelete</div>

Clicking the edit button will navigate to the **admineditloan component**, which displays an editing form with pre-populated loan data of the selected loan. This form also contains a "Back" button that will navigate to the **viewloan component**. Validations are performed for all the form fields.

If the Admin updates the loan type to match an existing loan type, an error indicating "Loan with the same type already exists" will be displayed.

Back

Edit Loan

Loan Type*

Truck

Description*

This loan is for purchasing a new car.

Interest Rate*

7.9

Maximum Amount*

160000

*Loan with the same type already exists.

Update Loan

Back

Edit Loan

Loan Type*

Car

Description*

This loan is for purchasing a new car.(Updated)

Interest Rate*

7.9

Maximum Amount*

160000

Update Loan

Upon clicking the "Update Loan" button, if the operation is successful, a popup saying "Updated successfully" will be displayed. Clicking "OK" will redirect to the viewloan component.

The screenshot shows the 'Edit Loan' form in the 'VEHICLE LOAN-HUB' application. The form fields are: Loan Type (Car), Description (This loan is for purchasing a new car.), Interest Rate (7.9%), and Maximum Amount (₹160000). A green 'Updated successfully' message box with an 'OK' button is overlaid on the form. A blue 'Update Loan' button is at the bottom of the form.

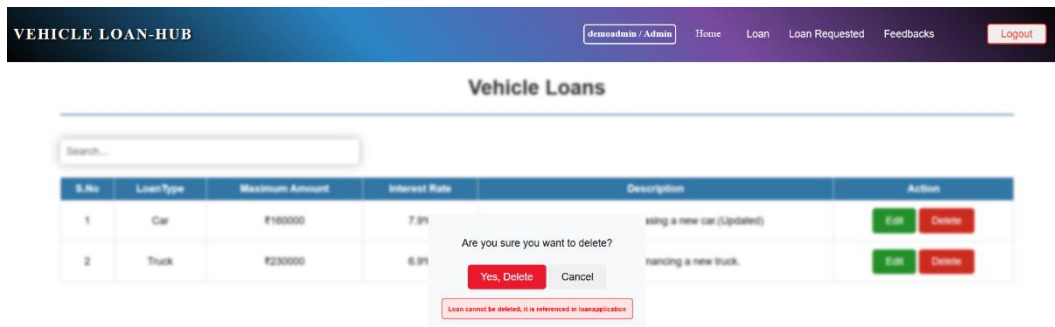
The screenshot shows the 'Vehicle Loans' table in the 'VEHICLE LOAN-HUB' application. The table has columns: S.No, LoanType, Maximum Amount, Interest Rate, Description, and Action. There are three rows of data.

S.No	LoanType	Maximum Amount	Interest Rate	Description	Action
1	Car	₹160000	7.9%	This loan is for purchasing a new car.(Updated)	Edit Delete
2	Truck	₹230000	6.9%	This loan is for financing a new truck.	Edit Delete
3	loan_78b3d250-7b48-4880-9f9e-63157a77c445	₹1000	10%	test	Edit Delete

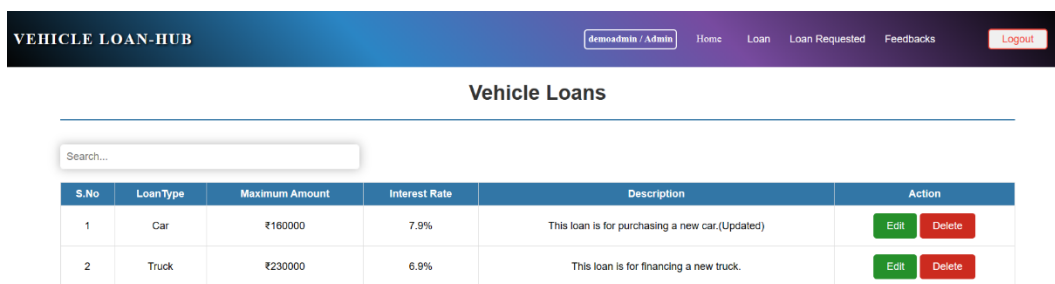
On clicking the "Delete" button, a pop-up should be displayed with confirmatory message to delete the data.

The screenshot shows the 'Vehicle Loans' table in the 'VEHICLE LOAN-HUB' application. A confirmation dialog is displayed over the table, asking 'Are you sure you want to delete?' with 'Yes, Delete' and 'Cancel' buttons.

S.No	LoanType	Maximum Amount	Interest Rate	Description	Action
1	Car	₹160000	7.9%	This loan is for purchasing a new car.(Updated)	Edit Delete
2	Truck	₹230		This loan is for financing a new truck.	Edit Delete
3	loan_78b3d250-7b48-4880-9f9e-63157a77c445	₹10		test	Edit Delete



Once a user has applied for the loan, the admin should not be able to delete the loan. Attempting to do so will display an error message stating "Loan cannot be deleted, it is referenced in loan application".



After successful deletion, the table should be refreshed as mentioned in the above images.

Admin View Loan Requested

Clicking on "Loan Requested" from the navbar will navigate to the **requestedloan component**, which displays all loans requested by all users. If no data is available, then "Oops! No records Found" should be displayed.

VEHICLE LOAN-HUB

demoadmin / Admin

Home

Loan

Loan Requested

Feedbacks

Logout

Loan Requests for Approval

Search...

Filter by Status: All

S.No	Username	Loan Type	Model	Submission Date	purchasePrice	Income	Status	Action
Oops! No records Found								

If data is available, the loans requested by all users will be displayed in table format. Additionally, features such as the admin being able to search based on Loan Type and filter the data based on Status.

VEHICLE LOAN-HUB

demoadmin / Admin

Home

Loan

Loan Requested

Feedbacks

Logout

Loan Requests for Approval

Search...

Filter by Status: All

S.No	Username	Loan Type	Model	Submission Date	purchasePrice	Income	Status	Action
1	user1	Car	2019-04-10	2024-05-10	₹900000	₹450000	Pending	<div>Show MoreApproveReject</div>
2	user2	Truck	2024-04-07	2024-05-10	₹2500056	₹230000	Pending	<div>Show MoreApproveReject</div>

The admin can approve or reject user requests by clicking on the "Approve" or "Reject" button, respectively.

VEHICLE LOAN-HUB

demoadmin / Admin

Home

Loan

Loan Requested

Feedbacks

Logout

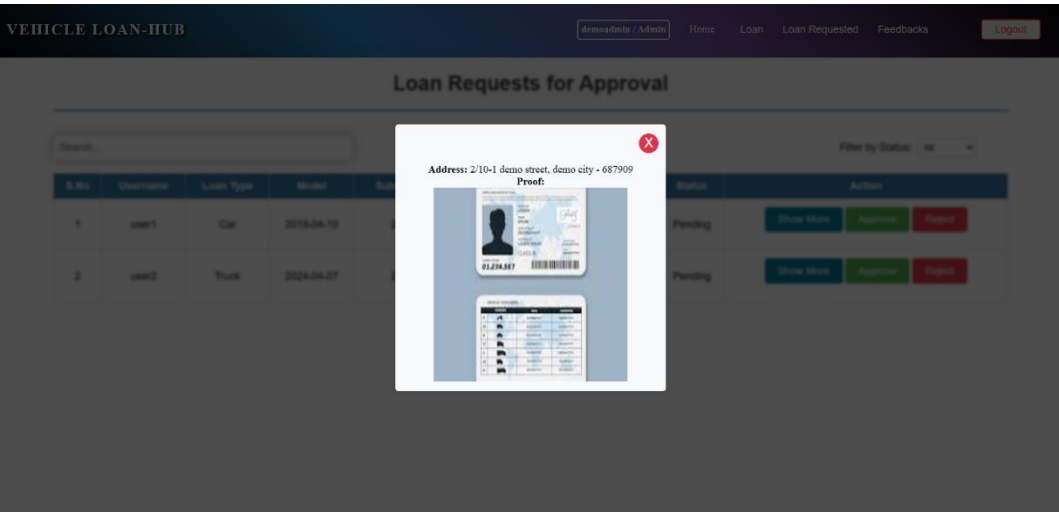
Loan Requests for Approval

Search...

Filter by Status: All

S.No	Username	Loan Type	Model	Submission Date	purchasePrice	Income	Status	Action
1	user1	Car	2019-04-10	2024-05-10	₹900000	₹450000	Rejected	<div>Show MoreApprove</div>
2	user2	Truck	2024-04-07	2024-05-10	₹2500056	₹230000	Approved	<div>Show MoreReject</div>

Clicking on "Show More" will display additional details such as Address and Proof in a popup modal.

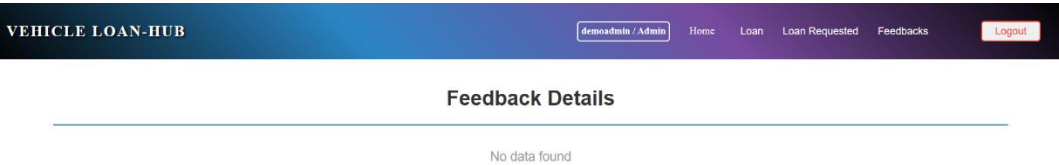


Clicking on the "X" sign will close the modal.

Admin View Feedbacks:

Clicking on "Feedbacks" from the navbar will navigate to the **adminviewfeedback component**, which displays all feedbacks posted by all users.

If no data is available, then "No data found" should be displayed.

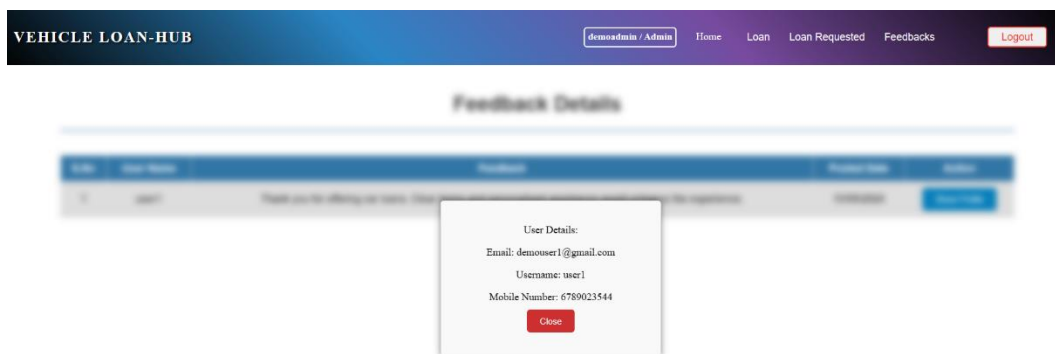


If data is available, the feedbacks posted by all users will be displayed in table format.

Feedback Details

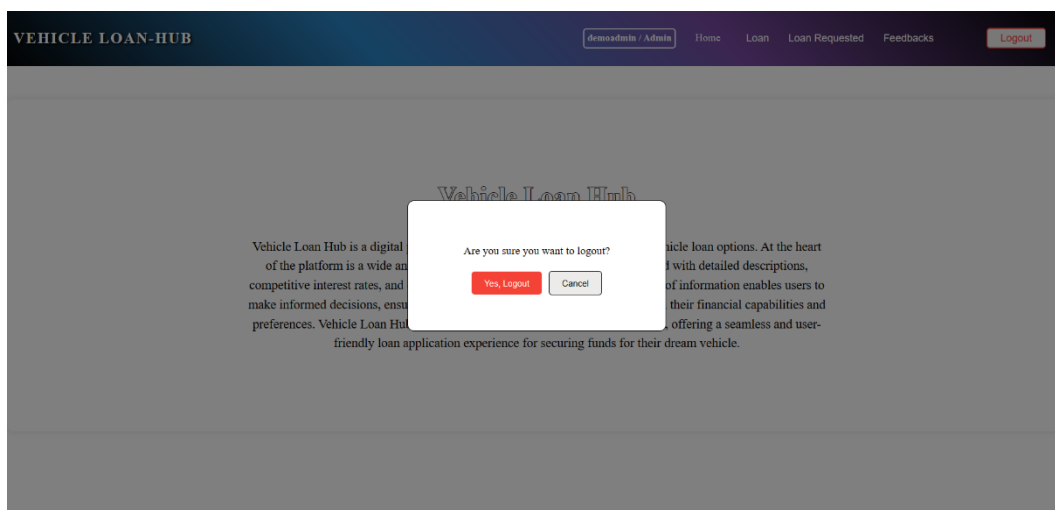
S.No	User Name	Feedback	Posted Date	Action
1	user1	Thank you for offering car loans. Clear terms and personalized assistance would enhance the experience.	10/05/2024	Show Profile

Clicking on "Show Profile" will display additional details about the user in pop-up modal.

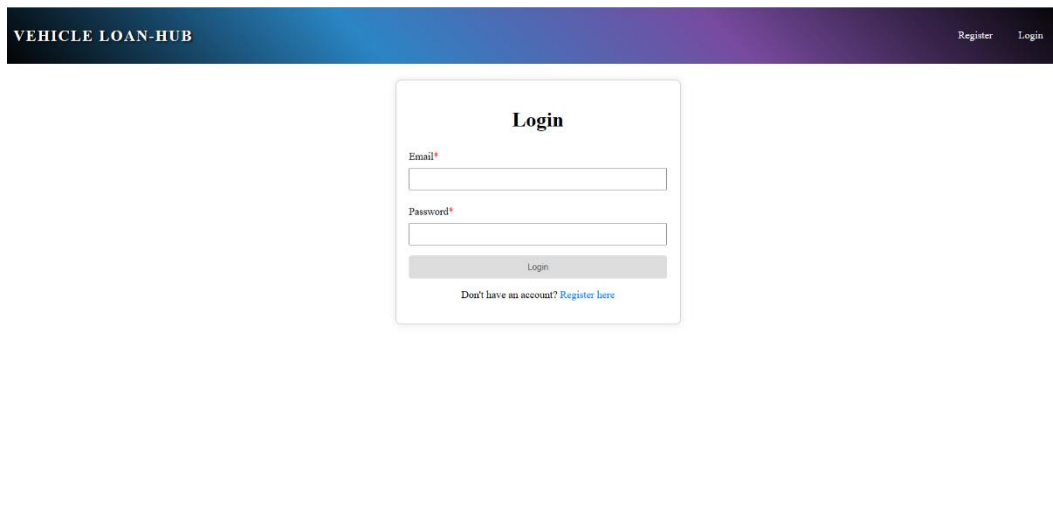


On clicking the "Close" button, will close the pop-up modal displayed.

On clicking the "Logout" button, a pop-up should be displayed with confirmatory message to logout the user.



Clicking "Yes, Logout" will navigate to the login component.



VEHICLE LOAN-HUB

Register Login

Login

Email*

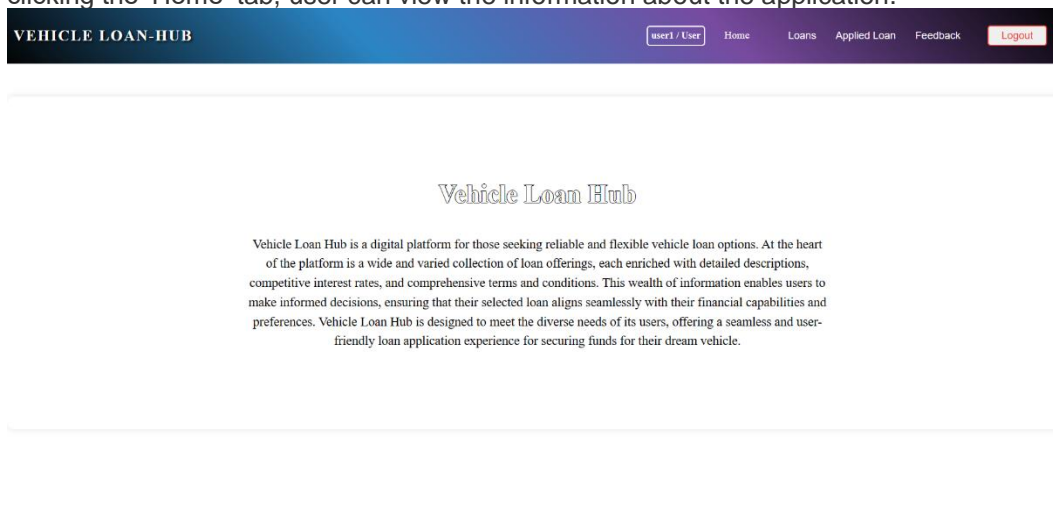
Password*

Login

Don't have an account? [Register here](#)

User side:

Home Component: This page is used to display the information about the loan hub application. On clicking the 'Home' tab, user can view the information about the application.



VEHICLE LOAN-HUB

user1 / User Home Loans Applied Loan Feedback Logout

Vehicle Loan Hub

Vehicle Loan Hub is a digital platform for those seeking reliable and flexible vehicle loan options. At the heart of the platform is a wide and varied collection of loan offerings, each enriched with detailed descriptions, competitive interest rates, and comprehensive terms and conditions. This wealth of information enables users to make informed decisions, ensuring that their selected loan aligns seamlessly with their financial capabilities and preferences. Vehicle Loan Hub is designed to meet the diverse needs of its users, offering a seamless and user-friendly loan application experience for securing funds for their dream vehicle.

Upon successful login, if the user is an admin, the **(adminnav component)** will be displayed. If the user is a regular user, the **(usernav component)** will be displayed. Additionally, the role-based navigation bar will also display login information such as the username and role.

User View Loans:

Clicking on "Loans" from the navbar will navigate to the **userviewloan component**, which displays all loans posted by admin.

If no data is available, then "Oops! No records Found" should be displayed.

If the user has applied for the loan, it will display as "Applied"; otherwise, it will display "Apply". Clicking the "Apply" button will navigate to the loanform component, which displays a form with the heading "Loan Application Form". Additionally, it includes a "Back" button, which, when clicked, moves

back to the userviewloan component

VEHICLE LOAN-HUB

user / User

Home

Loans

Applied Loan

Feedback

Logout

Available Vehicle Loans

Search...

S.No	Loan Type	Loan Description	Interest Rate	Maximum Amount	Action
1	Car	This loan is for purchasing a new car.(Updated)	7.9%	₹160000	<div>Apply</div>
2	Truck	This loan is for financing a new truck.	6.9%	₹230000	<div>Apply</div>

Clicking the "Apply" button will navigate to the **loanform component**, which displays a form with the heading "Loan Application Form". Additionally, it includes a "Back" button, which, when clicked, moves back to the **userviewloan component**

VEHICLE LOAN-HUB

user / User

Home

Loans

Applied Loan

Feedback

Logout

Loan Application Form

Back

Income*

*Income is required

Model*

dd-mm-yyyy

*Model is required

Purchase Price*

*Purchase Price is required

Address*

*Address is required

Proof*

Choose File

No file chosen

Submit

Perform validations for all the form fields.

VEHICLE LOAN-HUB

user / User

Home

Loans

Applied Loan

Feedback

Logout

Loan Application Form

Back

Income*

*Income is required

Model*

dd-mm-yyyy

*Model is required

Purchase Price*

*Purchase Price is required

Address*

*Address is required

Proof*

Choose File

No file chosen

Submit

Clicking the "Submit" button with empty fields will display a validation message stating "All fields are required".

VEHICLE LOAN-HUB

user / User

Home

Loans

Applied Loan

Feedback

Logout

Loan Application Form

Back

Income*

Model*

dd-mm-yyyy

Purchase Price*

Address*

Proof*

Choose File | No file chosen

*All fields are required

Submit

For the Model, future dates should be disabled, allowing only dates from the past to be selected.

VEHICLE LOAN-HUB

user / User

Home

Loans

Applied Loan

Feedback

Logout

Loan Application Form

Back

Income*

Model*

dd-mm-yyyy

Purchase Price*

Address*

Proof*

Choose File | No file chosen

*All fields are required

Submit

VEHICLE LOAN-HUB

user / User

Home

Loans

Applied Loan

Feedback

Logout

Loan Application Form

Back

Income*

60000

Model*

01-01-2024

Purchase Price*

450000

Address*

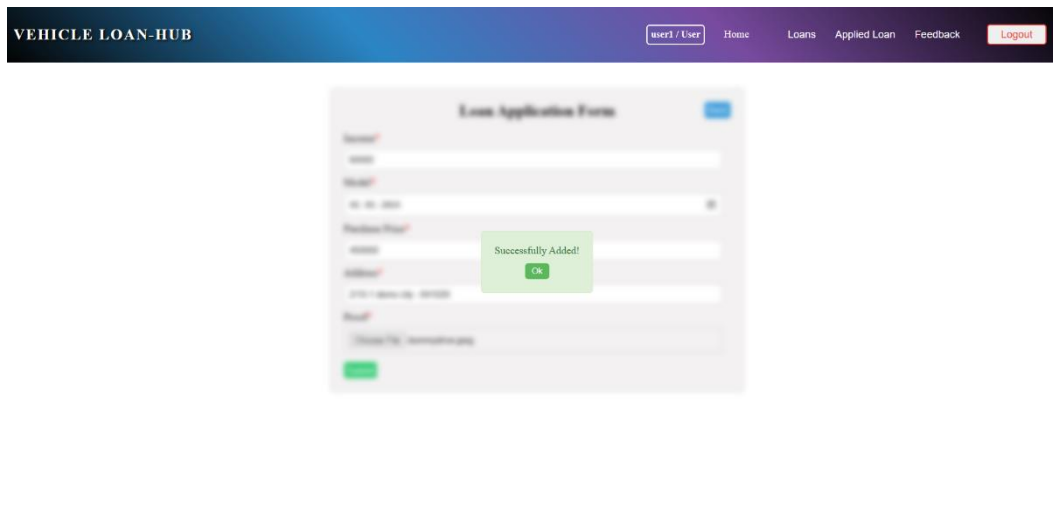
2 / 10 - 1 demo street2, demo city - 456789

Proof*

Choose File | dummydrive.jpeg

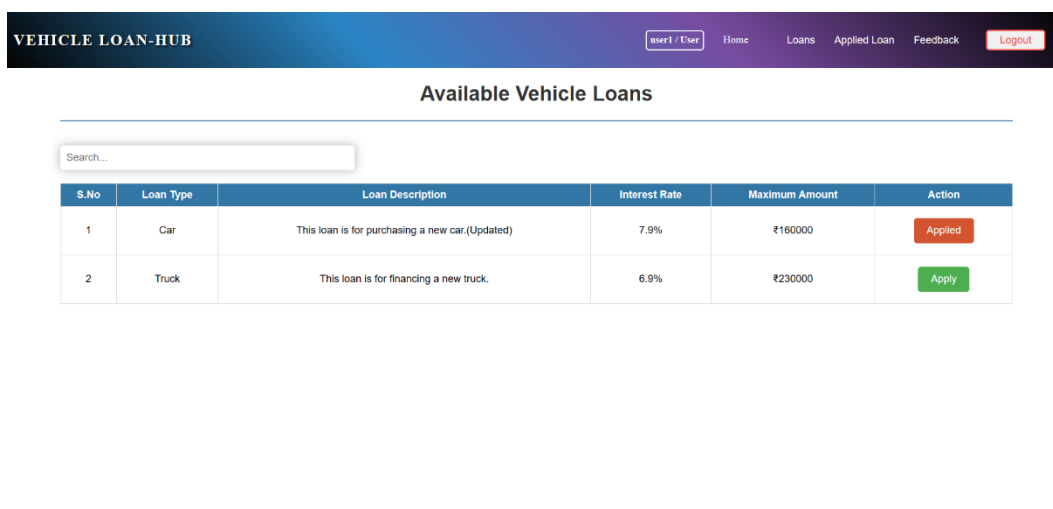
Submit

Upon clicking the "Submit" button, if the operation is successful, a popup message saying "Successfully Added!" should be displayed.



The screenshot shows the 'VEHICLE LOAN-HUB' header with navigation links: 'user1 / User', 'Home', 'Loans', 'Applied Loan', 'Feedback', and 'Logout'. Below the header is a 'Loan Application Form' with fields for Name, Address, Phone No., Email, and a Submit button. A green 'Successfully Added!' message box with an 'Ok' button is displayed over the form.

Clicking "Ok" will redirect to the **userviewloan component**. If the user has applied for the loan, it will display as "Applied"; otherwise, it will display "Apply".



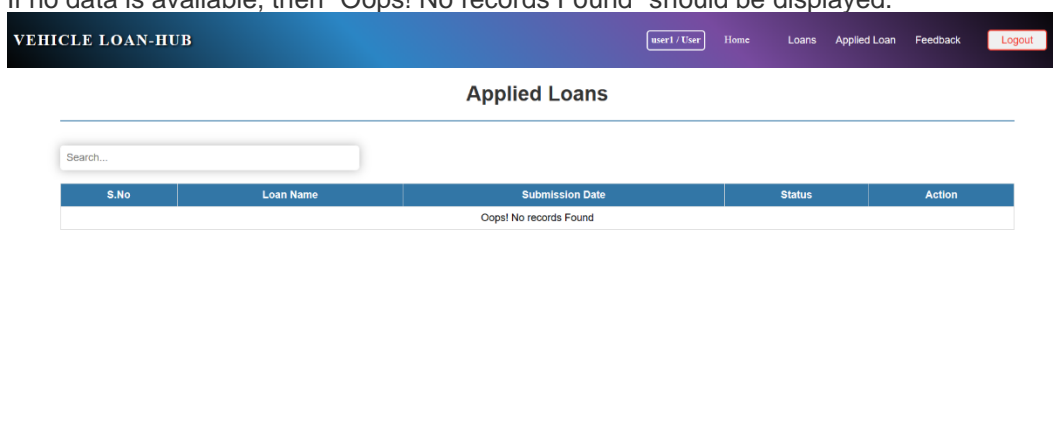
The screenshot shows the 'VEHICLE LOAN-HUB' header with navigation links: 'user1 / User', 'Home', 'Loans', 'Applied Loan', 'Feedback', and 'Logout'. Below the header is a section titled 'Available Vehicle Loans' with a search bar and a table of available loans.

S.No	Loan Type	Loan Description	Interest Rate	Maximum Amount	Action
1	Car	This loan is for purchasing a new car.(Updated)	7.9%	₹160000	Applied
2	Truck	This loan is for financing a new truck.	6.9%	₹230000	Apply

User Applied Loan:

Clicking on "Applied Loan" from the navbar will navigate to the **userappliedloan component**, which displays the user's applied loans.

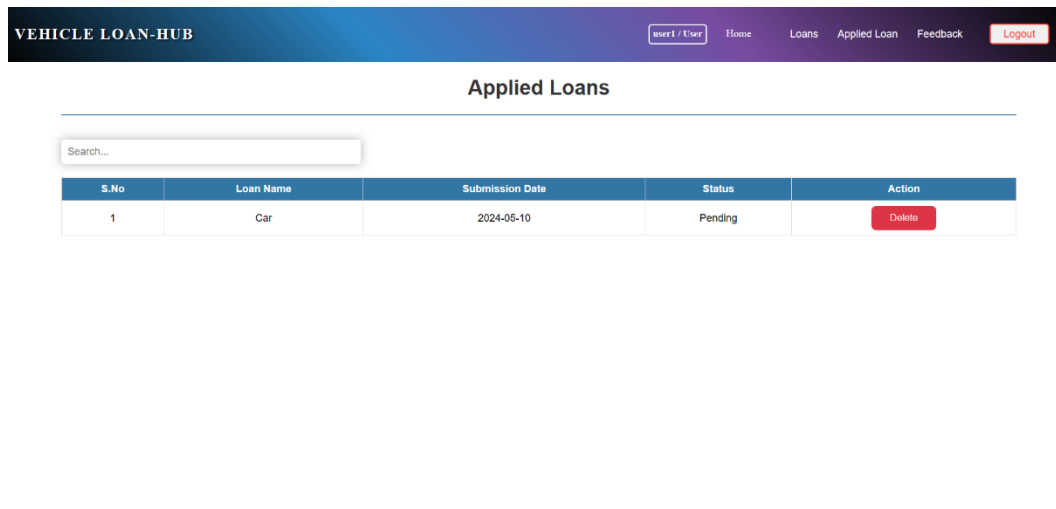
If no data is available, then "Oops! No records Found" should be displayed.



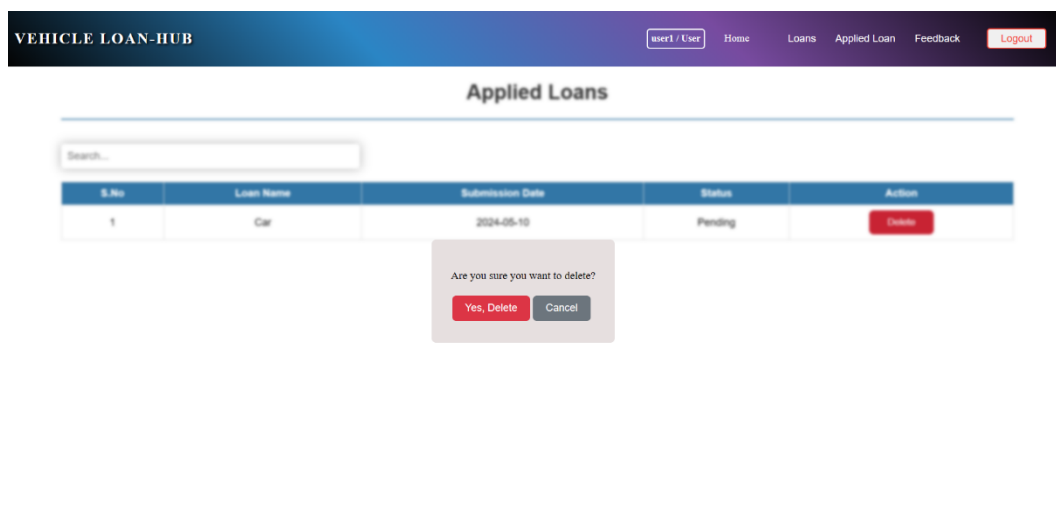
The screenshot shows the 'VEHICLE LOAN-HUB' header with navigation links: 'user1 / User', 'Home', 'Loans', 'Applied Loan', 'Feedback', and 'Logout'. Below the header is a section titled 'Applied Loans' with a search bar and a table of applied loans.

S.No	Loan Name	Submission Date	Status	Action
Oops! No records Found				

If data is available, the loans requested will be displayed in table format. Additionally, features such as the user being able to search based on Loan Name.



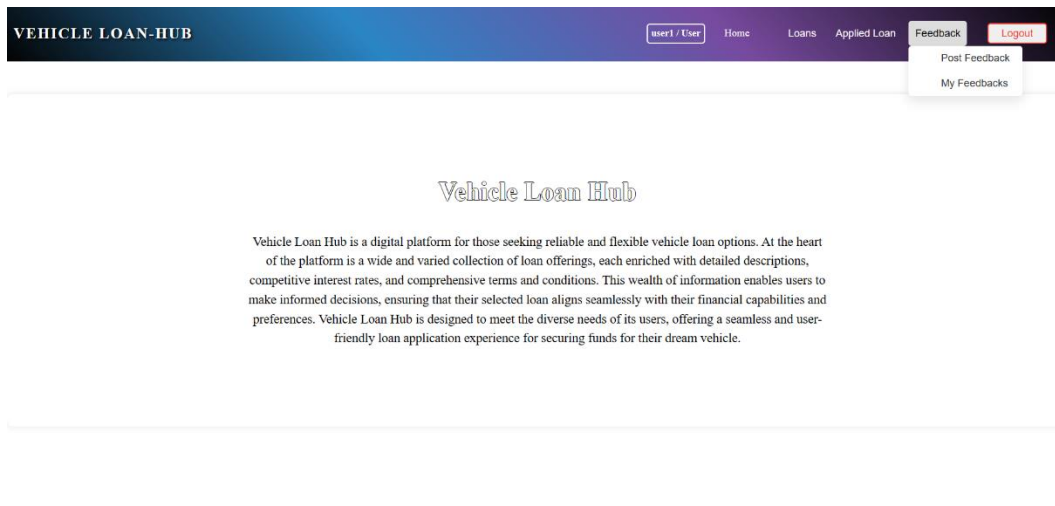
On clicking the "Delete" button, a pop-up should be displayed with confirmatory message to delete the data.



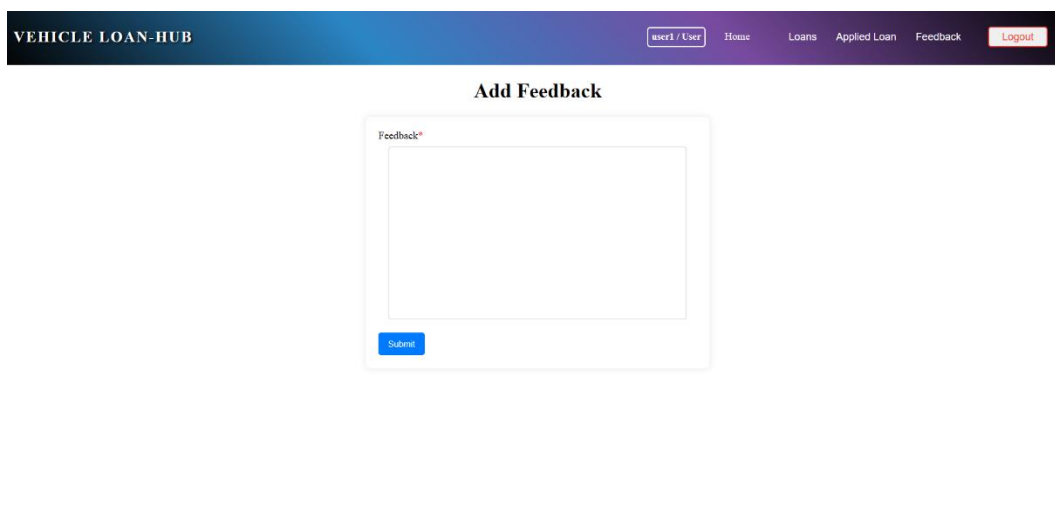
Clicking "Yes, Delete" will delete the applied loan, and the change will be automatically reflected.

User Feedback:

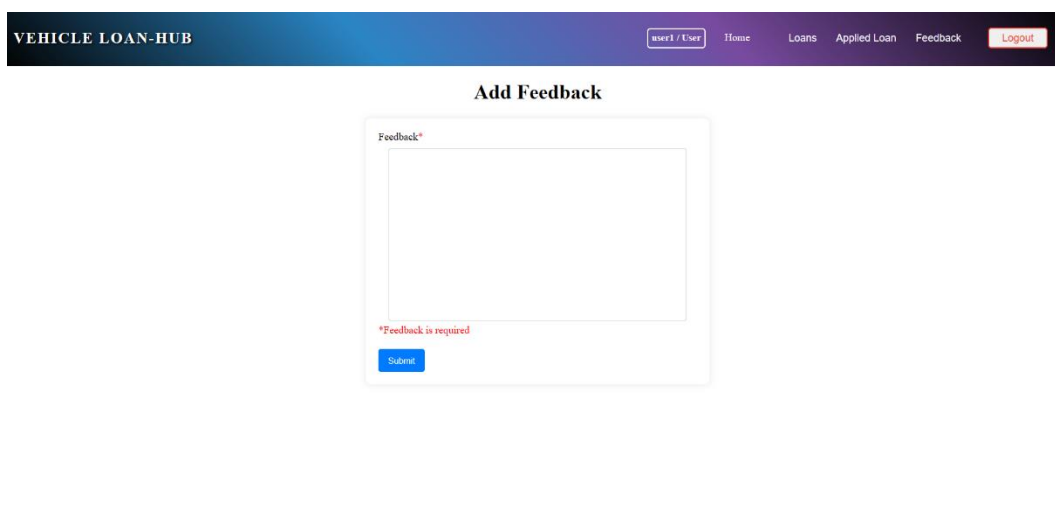
On hovering over the "Feedback" item in the navbar, a submenu should appear with options to "Post Feedback" and "My Feedbacks".



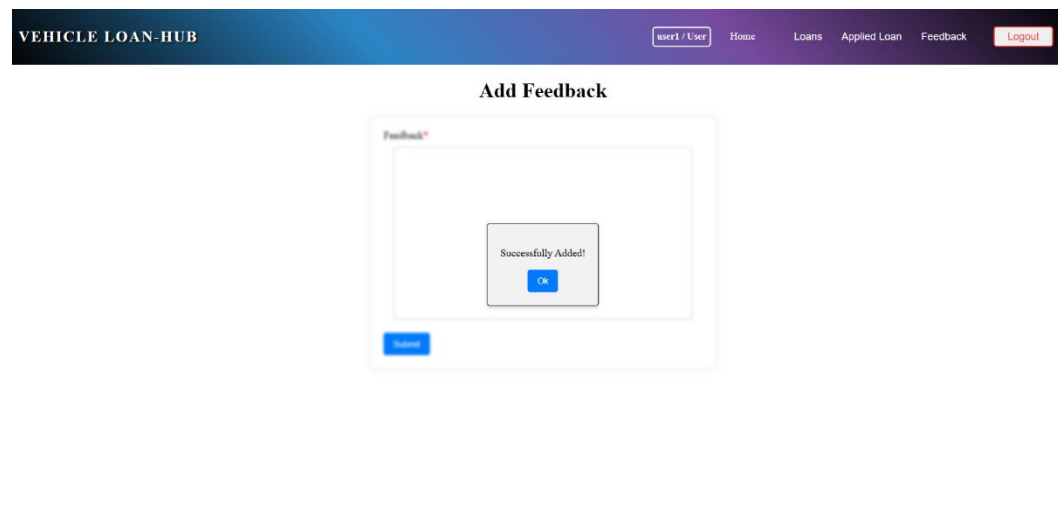
Clicking on "Post Feedback" will navigate to the **useraddfeedback component**, which displays the form to post feedback with heading as "Add Feedback"



Clicking the "Submit" button with empty textarea will display a validation message stating "Feedback is required".



Upon clicking the "Submit" button, if the operation is successful, a popup message saying "Successfully Added!" should be displayed.

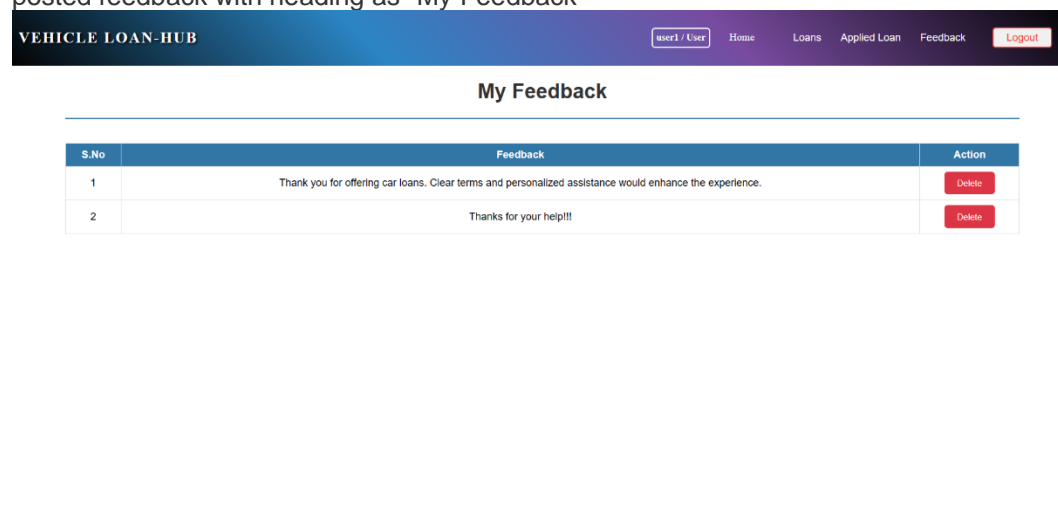


Clicking the "Ok" button will close the popup, and the same **useraddfeedback component** will be displayed.

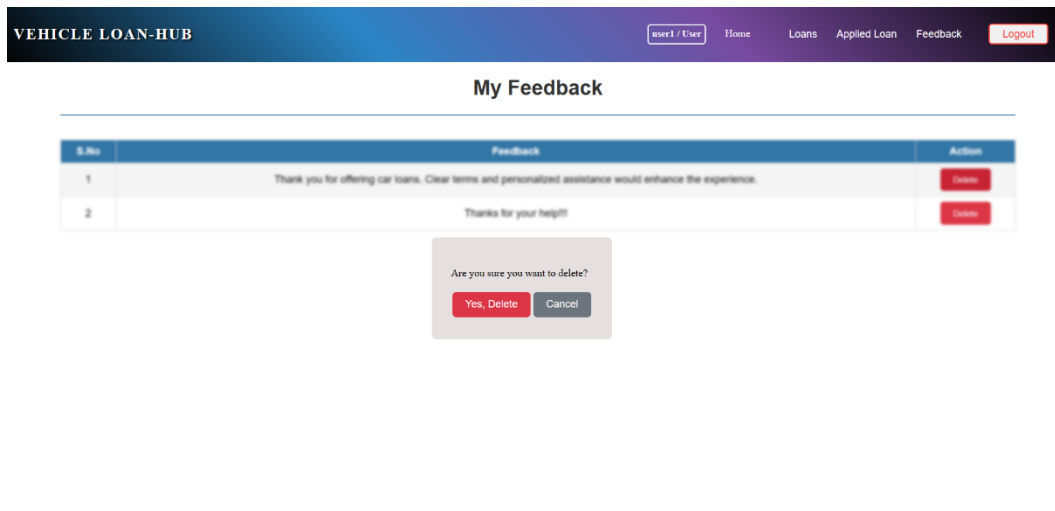
User view Feedback:

On hovering over the "Feedback" item in the navbar, a submenu should appear with options to "Post Feedback" and "My Feedbacks".

Clicking on "My Feedbacks" will navigate to the **userviewfeedback component**, which displays posted feedback with heading as "My Feedback"

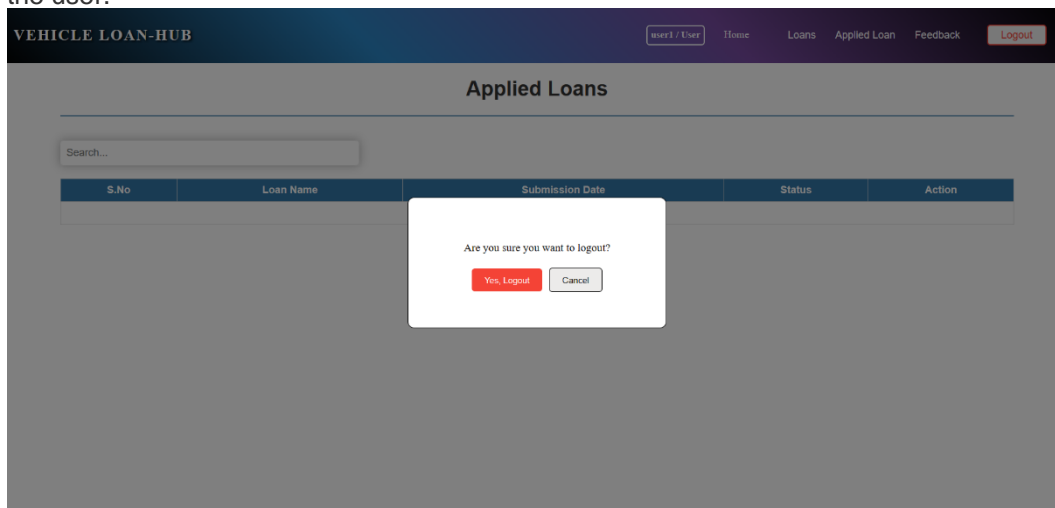


On clicking the "Delete" button, a pop-up should be displayed with confirmatory message to delete the data.

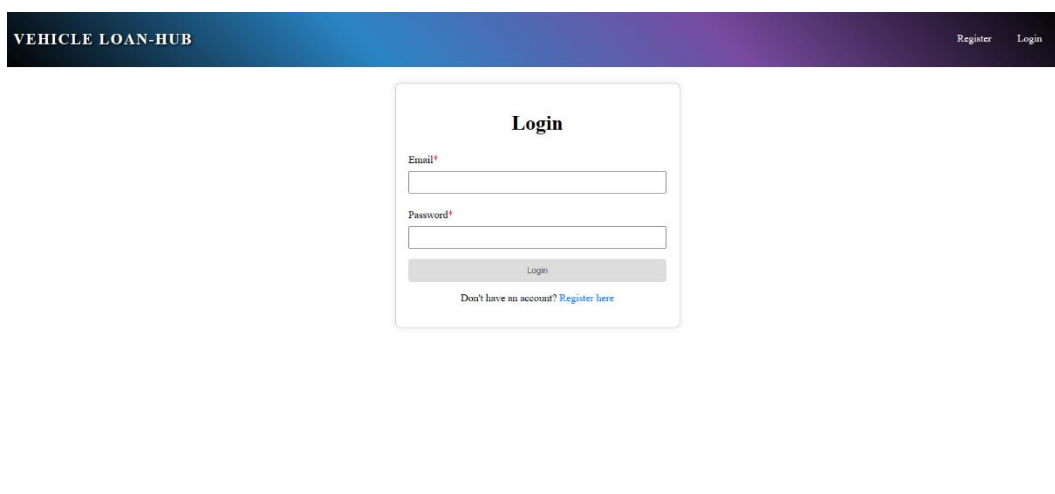


Clicking "Yes, Delete" will delete the feedback posted, and the change will be automatically reflected.

On clicking the "Logout" button, a pop-up should be displayed with confirmatory message to logout the user.



Clicking "Yes, Logout" will navigate to the login component.



error component

This page displays an error message stating "Something Went Wrong".



Something Went Wrong

We're sorry, but an error occurred. Please try again later.

Platform Prerequisites (Do's and Don'ts):

1. The angular app should run in port 8081.
2. The dotnet app should run in port 8080.
3. To incorporate .Net Security into the application, use JWT authentication within the project workspace.

Key points to remember:

1. The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.
2. Remember to check the screenshots provided with the SRS.
3. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
4. Adhere strictly to the endpoints mentioned in API endpoints section.
5. Don't delete any files in a project environment.

HOW TO RUN THE PROJECT:**BACKEND:**

Open the terminal and follow the commands below.

- **cd dotnetapp**

Select the dotnet project folder

- **dotnet restore**

This command will restore all the required packages to run the application.

- **dotnet run**

To run the application in port 8080

- **dotnet build**

To build and check for errors

- **dotnet clean**

If the same error persists clean the project and build again

To work with Entity Framework Core:

Install EF using the following commands:

dotnet new tool-manifest

dotnet tool install --local dotnet-ef --version 6.0.6

dotnet dotnet-ef --To check the EF installed or not

dotnet dotnet-ef migrations add "InitialSetup" --command to setup the initial creation of tables mentioned in DBContext

dotnet dotnet-ef database update --command to update the database

To Work with SQLServer:

(Open a New Terminal) type the below commands

sqlcmd -U sa

password: **examlyMssql@123**

>use DBName

>go

1> insert into TableName values(" ", " ", "...")

2> go

Note:

1. Please ensure that the application is running on port 8080 before clicking the "Run Test Case" button.
2. Database Name should be **appdb**
3. **Use the below sample connection string to connect the Ms SQL Server**

**connectionString = "User ID=sa;password=examlyMssql@123;
server=localhost;Database=appdb;trusted_connection=false;Persist Security
Info=False;Encrypt=False";**

FRONTEND:

Open the terminal and follow the commands below.

Step 1:

- Use **"cd angularapp"** command to go inside the angularapp folder
- Install Node Modules - **"npm install"**

Step 2:

- Write the code inside src/app folder
- Create the necessary components
- To create Service: **"npx ng g s <service name>"**
- To create Component: **"npx ng g c <component name>"**

Step 3:

- Click the **Run Test Case** button to run the test cases

Note:

- Click PORT **8081** to view the result / output.
- If any error persists while running the app, delete the node modules and reinstall them.