Online Mobile Recharge Portal

Overview:

The online mobile recharge web application is an innovative solution catering to the growing needs of users seeking a convenient and secure method to recharge their mobile services. With a user-friendly interface, the application allows seamless registration and authentication, enabling users to create accounts and log in securely. Providing a dashboard that acts as a control center, the application presents a spectrum of mobile recharge options, from selecting service providers to customizing plans and recharge amounts.

Users of the System:

1. **Customers**:

Role:

- Create and manage their customer accounts.
- Check the plans available.
- They can make payment.

2. **Administrators**:

Roles:

- Access and navigate the admin dashboard for an overview of plans.
- Manage the postpaid plans.
- Manage the Add-ons.
- Communicate with customers through the portal for additional information or updates.
- View Recharge History

Functional Requirements:

- Build an application that customers can select plans and recharge online.
- The user will have option to select the plan and recharge it.
- The user can also select the add-ons required.
- The admin can add/view/edit/delete prepaid plans.
- The admin can also add/edit/view/delete postpaid plans.
- The admin can also add add/edit/view/delete add-ons.
- The admin can also view recharges history.
- The user can make payment.
- The user can give reviews.

Non-Functional Requirements:

1. Security: The system must implement robust security measures to protect user data, including user authentication, secure data storage, and encrypted data transmission.
2. Scalability: The system should be designed to handle an increasing number of plans and customers.
3. Usability: The user interface should be intuitive and user-friendly, with responsive design for mobile and desktop users.
4. Availability: The system should be available 24/7 with minimal downtime for maintenance.
5. Logging and Auditing: Support logging and auditing of system activities for monitoring and troubleshooting.

Abstract:

The online mobile portal is an advanced digital platform designed to streamline and simplify mobile service recharges for users. This intuitive system offers a comprehensive suite of functionalities, including user-friendly registration and login processes, a dashboard providing an overview of recharge options, and a secure payment gateway for seamless transactions.

Application Flow:

Customer side:

The online mobile recharge portal begins with user registration and login, enabling account creation and secure access. Once logged in, users encounter a dashboard displaying various options for recharges, transaction history, promotions, and support. They select their mobile operator, service type, and recharge amount, proceeding to payment processing facilitated through secure gateways

offering multiple payment options. Following a successful recharge, users receive confirmation status stored in their history.

Admin side:
The administrative flow within the online recharge portal begins with administrators accessing the admin dashboard, providing a comprehensive overview of plans. Admins can also edit/delete plans, add-ons. Once the payment process is complete from the user side, the recharge history with customer details will be displayed in the admin side. Admin can view the recharge history details.

Modules of the Application:

ADMIN:
1. Register
2. Login
3. Dashboard
- Prepaid plans
- Postpaid plans
- Add-ons
- Recharge History

CUSTOMER:
1. Register
2. Login
3. Dashboard
- Plans Page
- Add-on page
- Recharge page
- Payment History

Technology Stack

Front End - React , HTML, CSS, JavaScript
Backend - Java, Spring Boot, MySQL
Authentication - JWT for User Authentication

Application assumptions:
1. The login page should be the first page rendered when the application loads.
2. Manual routing should be restricted by using Auth Guard by implementing the canActivate interface. For example, if the user enters as http://localhost:8080/signup or http://localhost:8080/home 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.
5. To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.
6. Design forgot password and forgot email buttons on the login page.

Validations:
1. Basic email validation should be performed.
2. Basic mobile number validation should be performed.
3. Basic password should be performed.

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

Server-Side Validation:

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

Exception Handling

1. Implement exception handling mechanisms in the controllers to gracefully handle errors and exceptions.
2. Define custom exception classes for different error scenarios, such as API communication errors or database errors.
3. 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:

1. 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.
2. Ensure that error pages contain helpful information and guidance for users.
3. 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.

Project Tasks:
API Endpoints:

Admin Side:

Action	URL	Method	Response
Admin Login	/auth /login	POST-Sends email ID and password	Return token
Admin SignUp	/auth /register	POST-Sends User Model data	True of false
Add plan	/api/plan	POST – Send Plan Model Data	Plan Added
View Plan	/api/plan	GET – Fetches Plan data	Retrieves all the plan details
Edit Plan	/api/plan/{planId}	PUT – Sends plan Id	Plan Edited
Delete Plan	/api/plan/{planId}	DELETE – Sends plan Id	Plan Deleted
Add add-on	/api/addon	POST – Sends Addon Model data	Add-on Added

Customer Side

Action	URL	Method	Response
View Plans	/api/customer/plan	GET – Fetches plan details	Retrieve all recharge plans
Add Recharge	/api/customer/recharge/{planId}	POST – Send Plan data	Plan Recharged
Add Addon	/api/customer/recharge/{addonId}	POST – Sent addOnid	AddOn Recharged
View add-ons	/api/customer/addon	GET – Fetches add-on details	Retrieve all add-ons plans
View Recharge	/ api/customer/{userid}	GET – Get the recharge	Retrieve my recharge
Edit Recharge	/api/customer/recharge/planId	PUT – Send plan Id	Recharge plan edited
Delete Recharge	/api/customer/recharge/planId	DELETE – Sends Plan Id	Recharge plan deleted
Make payment	/api/customer/makepayement	Post- Payment data	Payment status

Backend Requirements:

Model Class:

User:
This class stores the user type (admin or the customer) and all user information.
§ email: String
§ userId: Long
§ password: String
§ username: String
§ mobileNumber: String
§ userRole: String (ADMIN/CUSTOMER)

Customer:
§ customerId: integer
§ customerName: string
§ address: string
§ User user (One to One)
§ List<Payment> payments (One to Many)

Plan:
This class contains plan details.
§ planId: Long
§ planType: String
§ planName: String
§ planValidity: String
§ platDetails: String
§ planPrice: double
§ List<Recharge> recharges (One to Many)

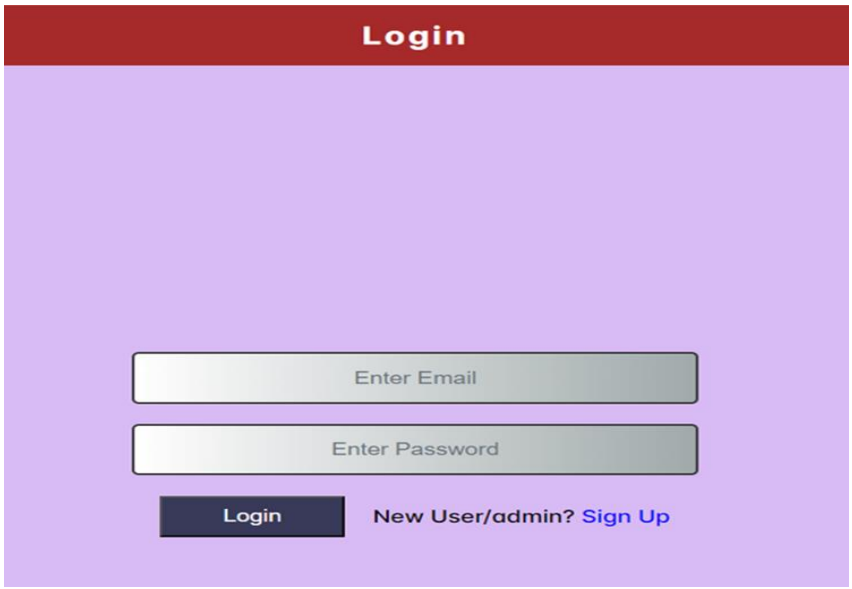
Addon:
This class contains add-on details.
§ addonId: Long
§ addonName: String
§ addonPrice: double
§ addonDetails: String
§ addonValidity: String
§ List<Recharge> recharges (One to Many)

Recharge:
This Class stores the recharge details
§ rechargeId: Long
§ rechargePrice: double
§ status: string
§ date: Date
§ Plan plan (Many to One)
§ AddOn addOn (Many to One)
§ Payment payment (One to One)

Payment:
This class stores the details of the admission.
§ paymentId: long
§ status: String
§ totalAmount: Double
§ paymentDate: Date
§ modeOfPayment: String
§ Customer customer (ManyToOne)

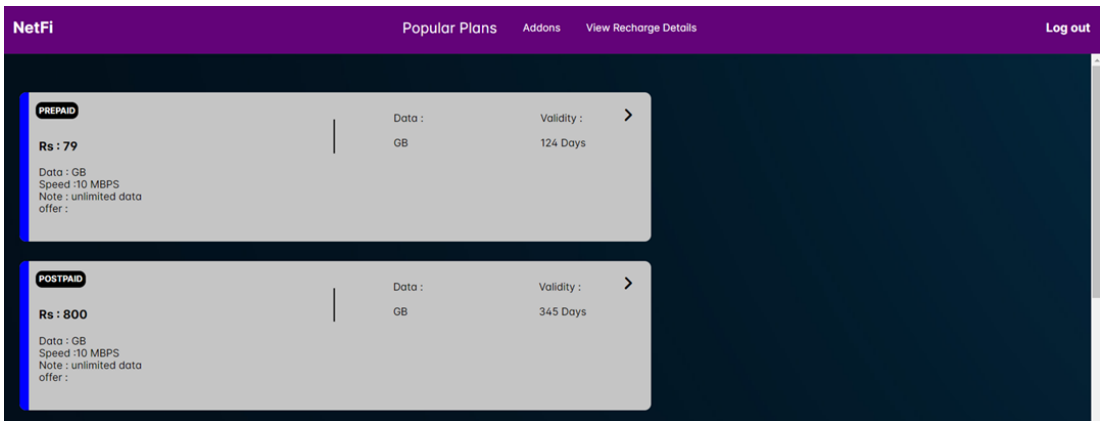
Register Page : The User can register their details and role

Login Page:

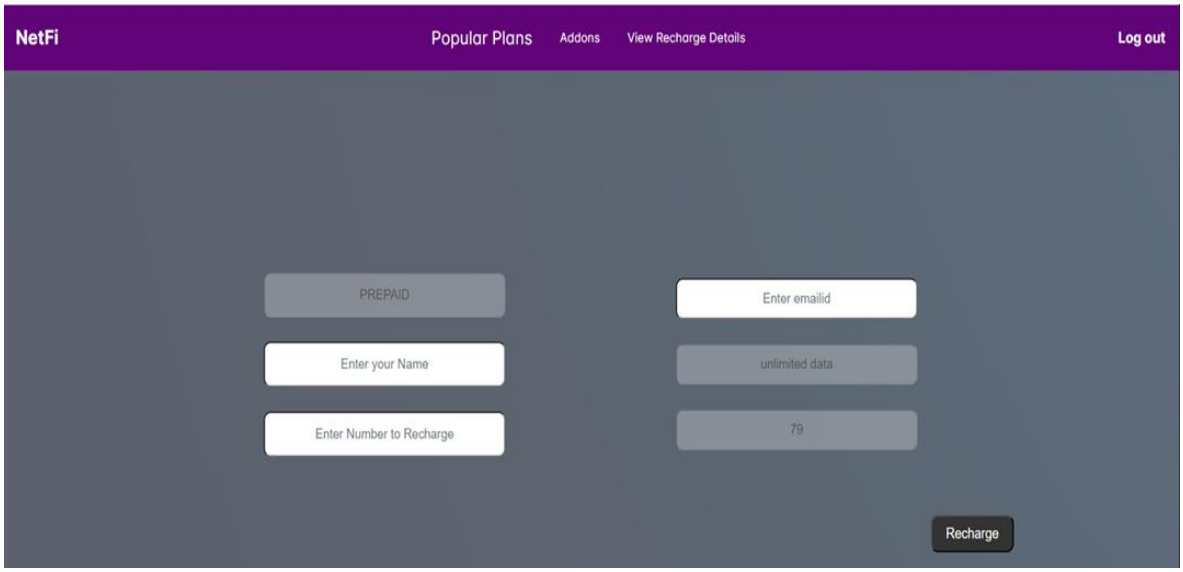


CUSTOMER Side:

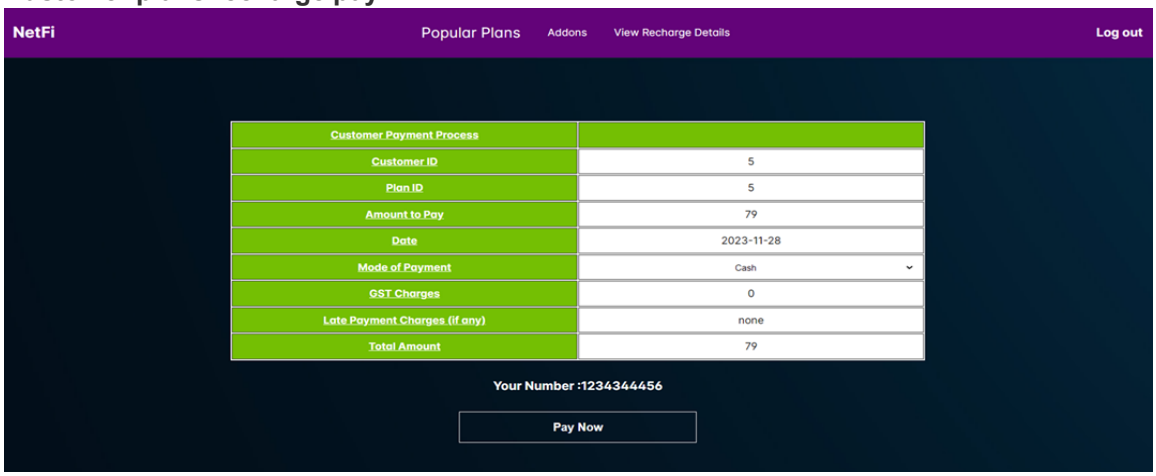
Customer plans : The Customer Can view the Recharge Plans.



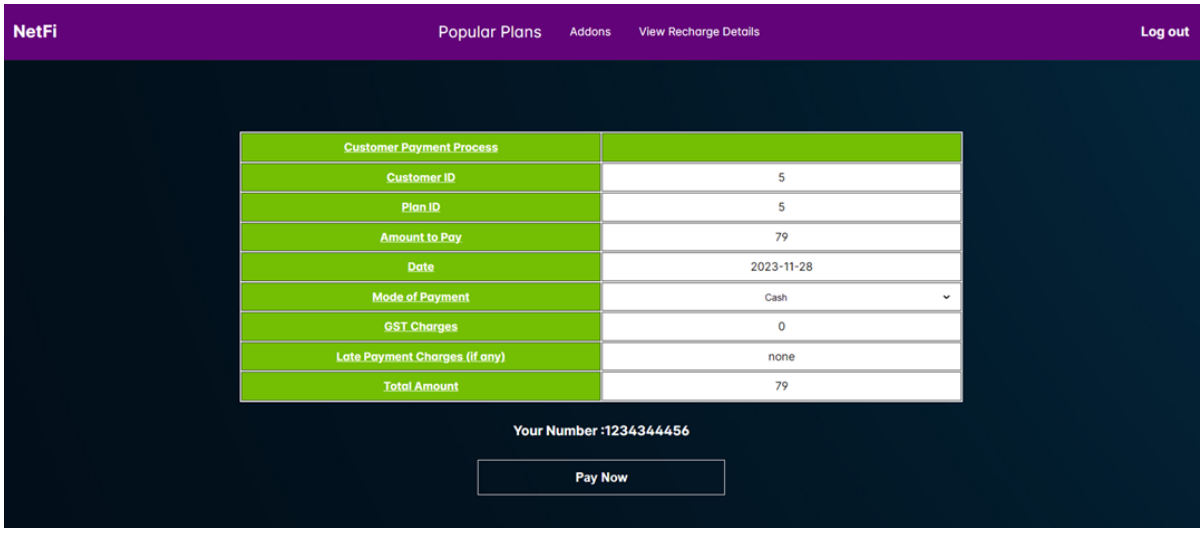
Customer plans recharge: The Customer can choose the plan and recharge



Customer plans recharge pay:

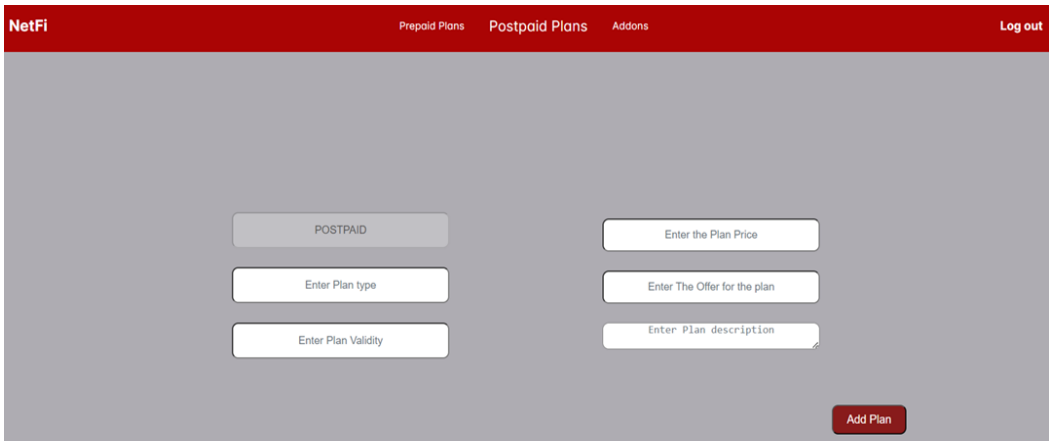


Customer Individual History: The Customers can view their own Recharge History

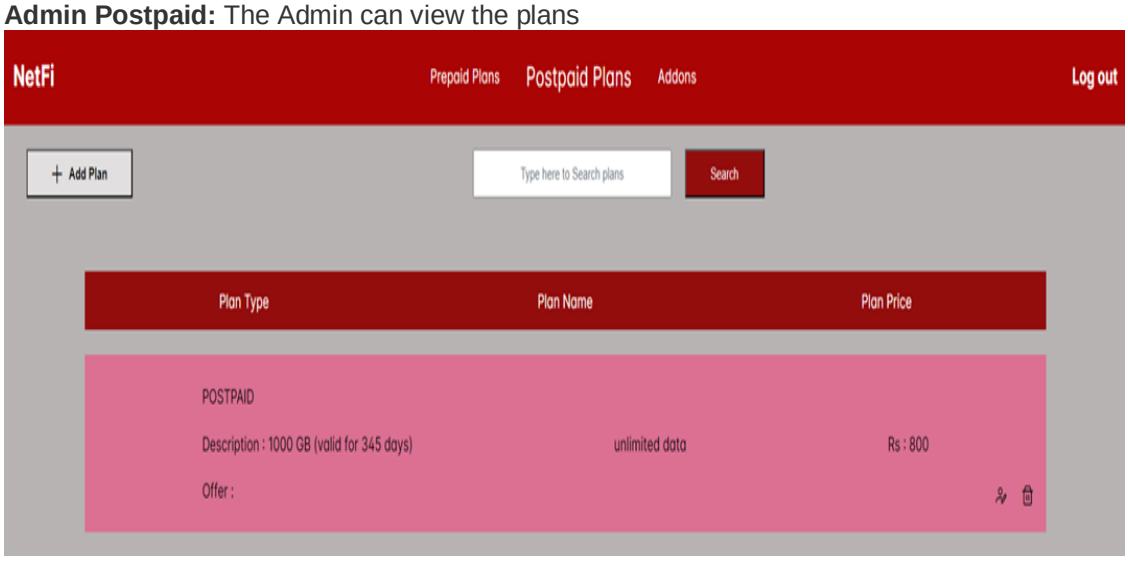


Admin

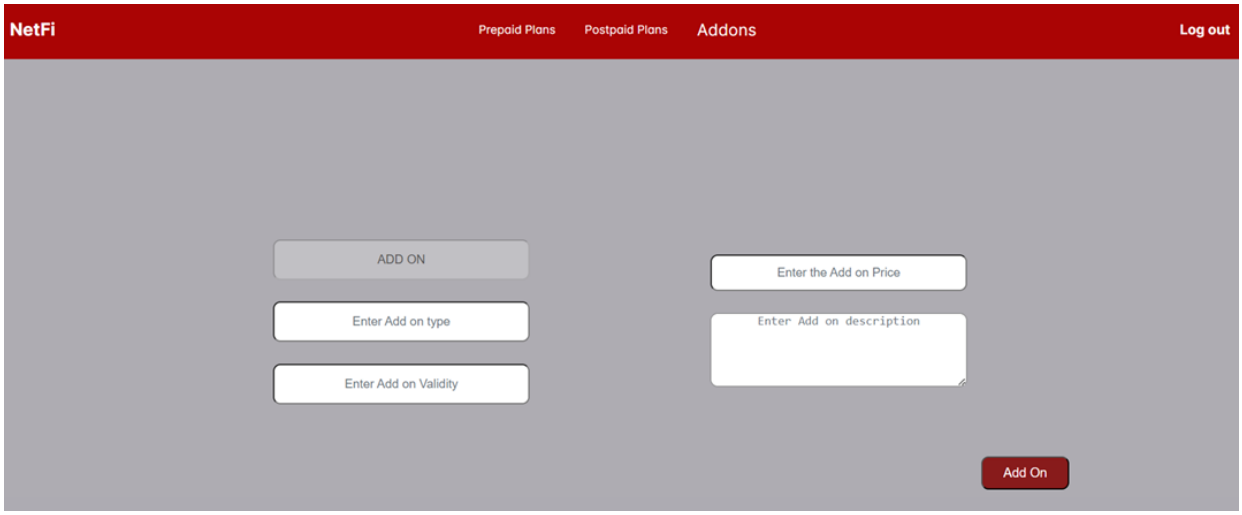
Postpaid Add Plan: Admin can add the Recharge Plans



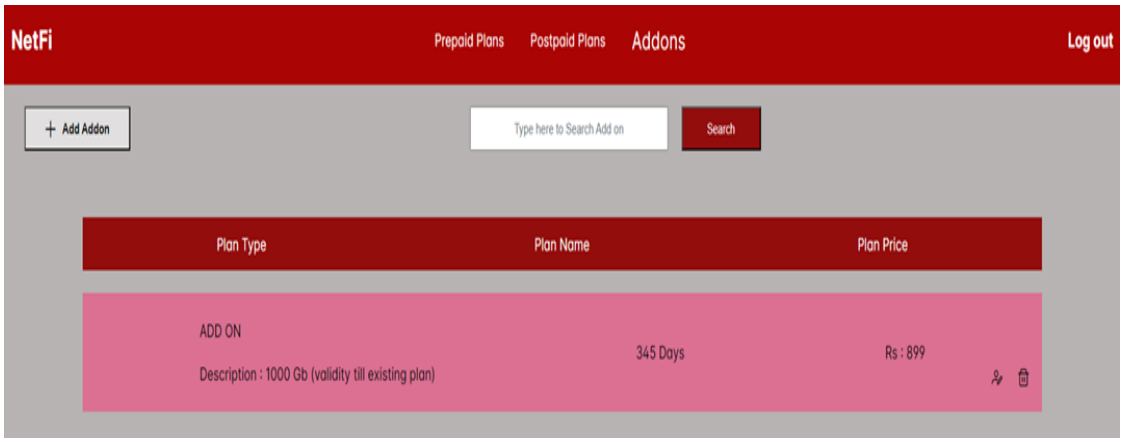
Admin Postpaid: The Admin can view the plans



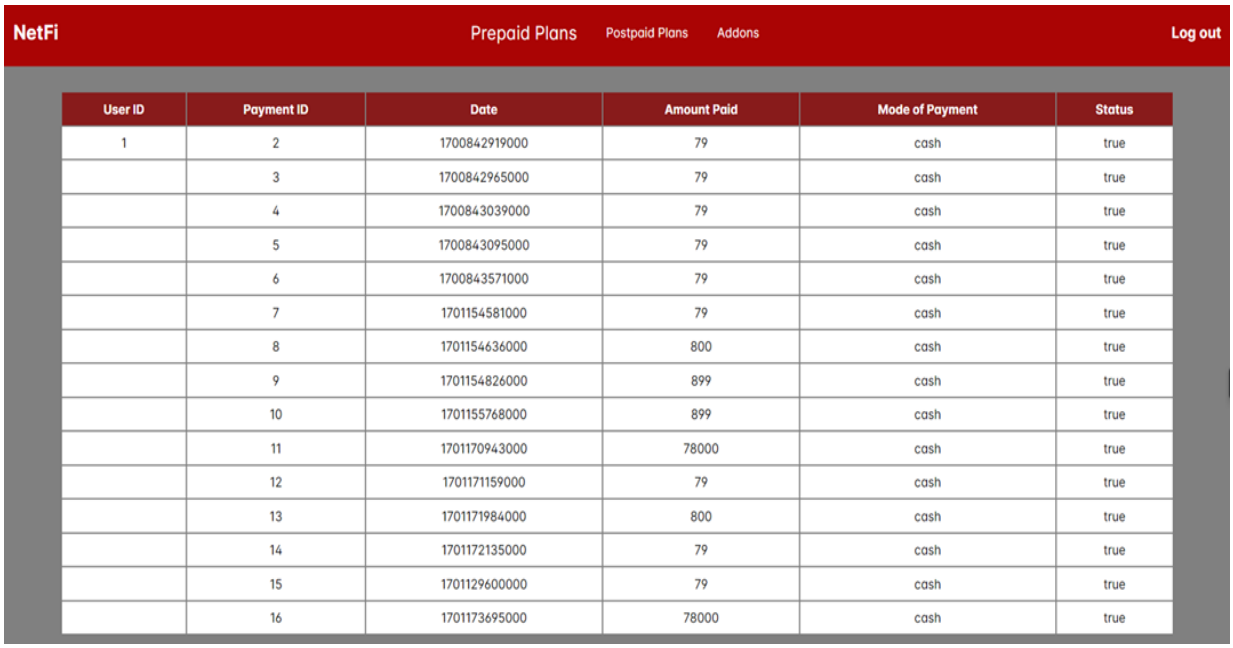
Addons Add plan: The Admin can add the Addons plans



The Admin can view the Add on plans



The Admin can view all the recharge history



User ID	Payment ID	Date	Amount Paid	Mode of Payment	Status
1	2	1700842919000	79	cash	true
	3	1700842965000	79	cash	true
	4	1700843039000	79	cash	true
	5	1700843095000	79	cash	true
	6	1700843571000	79	cash	true
	7	1701154581000	79	cash	true
	8	1701154636000	800	cash	true
	9	1701154826000	899	cash	true
	10	1701155768000	899	cash	true
	11	1701170943000	78000	cash	true
	12	1701171159000	79	cash	true
	13	1701171984000	800	cash	true
	14	1701172135000	79	cash	true
	15	1701129600000	79	cash	true
	16	1701173695000	78000	cash	true

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

Ø The react app should run in port 8081.
Ø The spring boot app should run in port 8080.

Other Important Key factors in the application:

- Should use Custom Exceptions mandatory.
- Tables should have proper relationship and keys
- Frontend Application should be menu driven.
- Proper Menu / Navigation for corresponding role
- Client side validations and server side validations are mandatory
- Error should be handled.
- Follow best programmer practice while developing
- Provide proper Naming Conventions
- Don't delete any files in a project environment.
- You should use NotFound(), NoContent(), BadRequest(), CreatedAtAction() to handle the HTTP status code as return values for the Controller methods as mentioned.
- Use Mysql database

How to run the Project

Back End
API endpoint:
8080

Platform Guidelines:
To run the command use **Terminal** in the platform.

Spring Boot:
Navigate to the springapp directory => **cd springapp**
To start/run the application '**mvn spring-boot:run**'

To Connect Database Open Terminal
Cmd:**mysql -u root –protocol=tcp -p**
Password: **examly**

Front End
Step 1:
Open the terminal
Use "nvm use 14" command to change node version to 14

Step 1:
Use "cd reactapp" command to go inside the reactapp folder
Install Node Modules **- "**npm install**"**

Step 2:
Write the code inside src folder
Create the necessary components

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