1.Introduction

Project Title:

Fundtastic : make Fundraising fun and easy!

Team Members:

VANGA NAVYA - TEAM LEADER
YOGESWARARAO BONIGALA - TEAM MEMBER 1
VEPURI SMILY - TEAM MEMBER 2
YARRAM SAGAR - TEAM MEMBER 3

2. PROJECT OVERVIEW

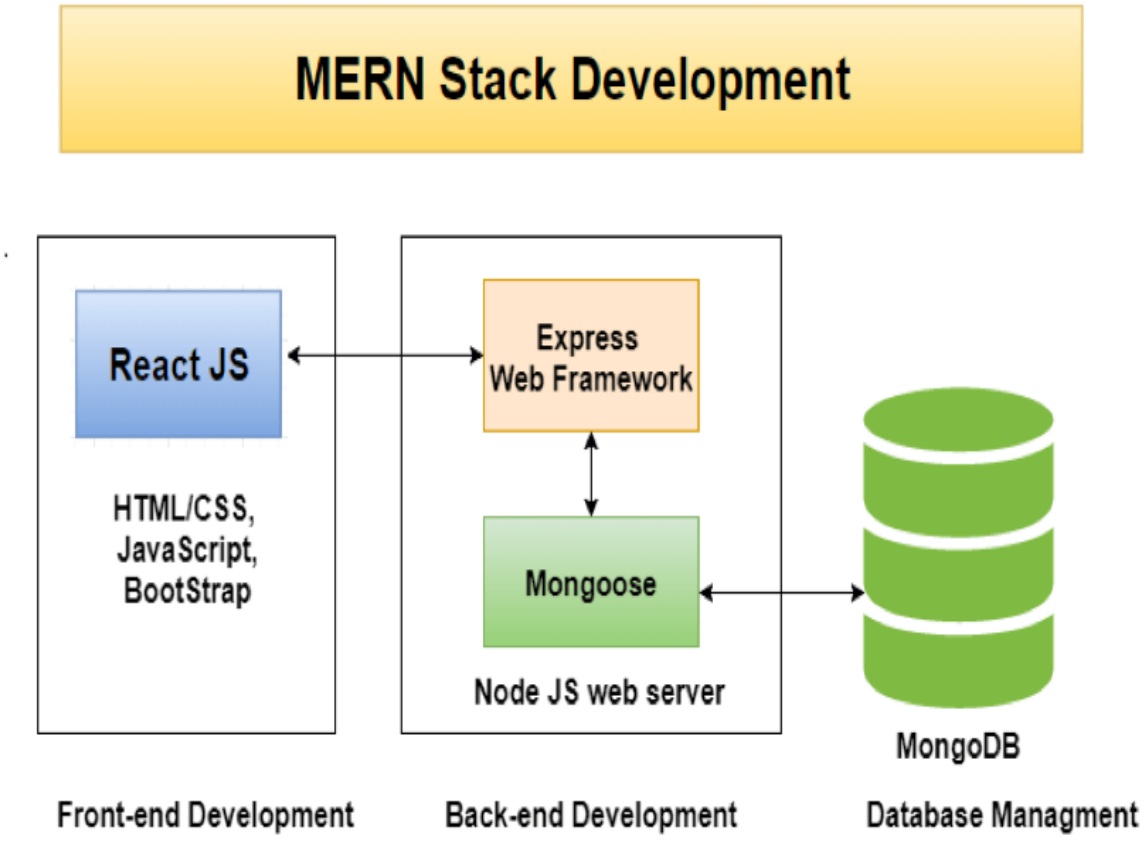
Purpose:

The purpose of the project is to develop a comprehensive Fundraising and Investment Platform using the MERN stack (MongoDB, Express.js, React.js, and Node.js) to bridge the gap between individuals seeking funds for their projects and potential investors. This platform aims to provide a user-friendly interface that facilitates efficient communication and transaction management between fundraisers and investors, thereby simplifying the process of raising and managing funds while enhancing investor engagement. By addressing key challenges such as limited visibility for fundraisers, complex investment processes, and lack of transparency in fund allocation, the platform seeks to empower entrepreneurs by offering them a dedicated space to showcase their projects and engage directly with investors. It also aims to simplify the investment process for investors by allowing them to easily discover, evaluate, and engage with promising campaigns. Additionally, the platform promotes financial transparency by providing structured details on funding goals and progress, fostering trust between fundraisers and investors. Through a technology-driven approach leveraging the MERN stack, the platform ensures real-time updates, secure transactions, and a seamless user experience. Ultimately, this project seeks to create a robust ecosystem for financial collaboration that fosters innovation, supports the growth of new ideas, and drives economic development and social impact by addressing market gaps in existing fundraising platforms.

Key Features:

- Registration and Login
- Raise Fund Request
- View Raised Requests
- Profile Management
- Logout
- Approach for Investments
- View Investments

3. Architecture



Frontend:

In The front-end of the Fundraising and Investment Platform is designed to provide a seamless and intuitive user experience for both fundraisers and investors. Built using React.js, the front-end ensures that users can easily navigate through the platform's features, perform actions efficiently, and receive real-time updates on their activities.

Key Features

- **Responsive Design**: The platform's interface is responsive, ensuring optimal usability across various devices, including desktops, tablets, and smartphones. This design approach guarantees that users have a consistent experience regardless of the device they use.
- **User-Friendly Interface**: The front-end is designed with simplicity and ease of navigation in mind. Users can quickly access key functionalities such as campaign creation, investment tracking, and profile management without encountering complex menus or interfaces.
- **Real-Time Updates**: Leveraging React's capabilities, the platform provides real-time updates to users. This feature is crucial for fundraisers to monitor their campaign progress and for investors to track their investments' status without needing to refresh or reload pages.
- **Interactive Components**: The use of interactive components such as forms, progress bars, and dashboards enhances user engagement. These elements provide immediate feedback and visual cues to guide users through various processes on the platform.
- **Secure Authentication**: The front-end integrates secure authentication mechanisms using JWT (JSON Web Tokens) to ensure that user sessions are protected. This security measure helps maintain user privacy and data integrity throughout their interactions with the platform.

Technical Stack

- **React.js**: As the primary framework for building the user interface, React.js allows for the development of dynamic and responsive components that enhance user interaction.

- **HTML5/CSS3**: These technologies are employed for structuring and styling web pages, ensuring a clean layout that adheres to modern web standards.
- **JavaScript (ES6+)**: Used extensively for client-side scripting, JavaScript enables interactive features and enhances overall user interactivity on the platform.

2. Error Handling & Notifications

Handling errors (like failed API requests) and providing feedback to the user is essential. Use libraries like **React Toastify** or custom notification components for

1. Displaying success messages when an expense is added.
2. Showing error messages when something goes wrong

UI/UX Design :

For styling and layout, you may use:

- **CSS Modules / SASS / Styled Components**: Depending on your preference for styling. CSS frameworks like Bootstrap or Tailwind CSS is been used to speed up development and create responsive, mobile-friendly layouts.
- **User Interface Elements**:
- **Tables/Lists**: To display a list of expenses.
    - **Forms**: For adding or editing expenses.
    - For confirming deletions, displaying form errors, etc.

Back End:

The back-end of the Fundraising and Investment Platform is responsible for handling all server-side logic, data management, and integration with external services. Built using Node.js and Express.js, it ensures efficient processing of requests, secure data handling, and seamless communication with the front-end.

Key Components

- **API Development**: RESTful APIs are developed to handle user authentication, campaign management, investment processing, and other

core functionalities. These APIs facilitate communication between the front-end and back-end.

- **Database Management**: MongoDB is used as the primary database to store user data, campaign details, investment records, and activity logs. Mongoose is employed to define schemas and models for structured data management.
- **Authentication and Authorization**: Secure user authentication is implemented using JSON Web Tokens (JWT), ensuring that only authorized users can access specific resources. Role-based access control (RBAC) is enforced to differentiate functionalities available to fundraisers, investors, and administrators.
- **Data Security**: Sensitive information such as passwords and financial details are encrypted using bcrypt. The platform also employs SSL/TLS protocols for secure data transmission.
- **Real-Time Updates**: WebSockets or libraries like Socket.io may be used to push real-time notifications to users about campaign updates or investment statuses.

Technical Stack

- **Node.js**: Provides a scalable environment for building fast server-side applications.
- **Express.js**: A framework for Node.js that simplifies API development and request handling.
- **MongoDB**: A NoSQL database that offers flexibility and scalability for data storage.
- **Mongoose**: An ODM library for MongoDB that helps in defining schemas and interacting with the database.

Security Measures

- **Data Encryption**: All sensitive data is encrypted both in transit and at rest.
- **Secure Transactions**: Integration with secure payment gateways ensures safe financial transactions.
- **Regular Audits**: Security audits are conducted regularly to identify and mitigate potential vulnerabilities.

Scalability and Maintenance

The back-end architecture is designed to be modular, allowing easy updates and scalability. As user demand grows, additional resources can be allocated without affecting performance. The use of cloud services like AWS or Heroku facilitates easy deployment and scaling.By focusing on these aspects, the back-end of the Fundraising and Investment Platform ensures robust performance, security, and reliability, providing a strong foundation for the platform's operations.

2. **API Layer**

The backend in a MERN stack application exposes a set of RESTful APIs to the front end. These APIs perform CRUD (Create, Read, Update, Delete) operations and handle any other requests from the client.

HTTP Methods:

- GET: Retrieves resources (e.g., list of campaigns).
- POST: Submits data to create a new resource (e.g., add a new campaign).
- PUT: Updates a resource (e.g., editing an campaign).
- DELETE: Removes a resource (e.g., delete an campaign).

Data base (MongoDB):

The database architecture for the Fundraising and Investment Platform is designed to efficiently manage and store data related to users, campaigns, investments, and activities. MongoDB, a NoSQL database, is chosen for its flexibility, scalability, and seamless integration with JavaScript-based technologies in the MERN stack.

Key Components

- **Data Models**: The platform utilizes several key data models:
    - **User Model**: Captures user details, roles (fundraiser or investor), authentication information, and associated campaigns and investments.

- **Campaign Model**: Represents fundraising campaigns with attributes like title, description, goal amount, current amount raised, status, and bank details.
- **Investment Model**: Tracks investments made by users, including the amount, associated campaign, and status.
- **Activity Model**: Records user activities within the platform for tracking actions performed by users.

Schema Design

- Embedding vs. Referencing: MongoDB schema design can utilize embedding for one-to-few relationships to reduce the need for joins and improve read performance. For one-to-many or many-to-many relationships, referencing is preferred to maintain flexibility and scalability.
- Normalization and Denormalization: Depending on the application's needs, data can be normalized to reduce redundancy or denormalized to optimize read operations.

Performance Optimization

- Indexing: Indexes are used on frequently queried fields to enhance query performance. This includes fields like user IDs and campaign statuses.
- Sharding: MongoDB's sharding capabilities allow horizontal scaling by distributing data across multiple servers. This ensures the platform can handle increased loads as it grows.

Security Measures

- Data Encryption: Sensitive data such as user credentials are encrypted both at rest and in transit using industry-standard protocols like SSL/TLS.
- Access Control: Role-based access control (RBAC) is implemented to ensure that users have permissions appropriate to their roles (e.g., fundraiser or investor).

Scalability and Flexibility

- Document Data Model: MongoDB’s document model allows for flexible schema design that can evolve with application requirements without needing complex migrations.

- Multi-Cloud Deployment: MongoDB supports deployment across various cloud platforms, providing flexibility in hosting options and ensuring high availability.

By leveraging MongoDB's strengths in scalability and flexibility, the database architecture of the Fundraising and Investment Platform is well-equipped to handle dynamic data requirements while ensuring high performance and security.

3.Setup Instructions

Prerequisites :

To develop a full-stack expense tracker app using React js, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

- Node.js and npm
- MongoDB
- Express.js
- React
- HTML,CSS and JavaScript
- Database Connectivity
- Development Environment
- Version control

Installation :

Prerequisites

- Node.js: Ensure Node.js is installed on your system. You can download it from the official Node.js website.
- MongoDB: Install MongoDB and ensure it is running. You can use MongoDB Atlas for a cloud-based solution.
- Git: Install Git for version control.

Clone the Repository

1. Open your terminal or command prompt.
2. Clone the repository using Git:
   ```bash
   git clone <repository-url>
   ```
3. Navigate into the project directory:
   ```bash
   cd <project-directory>
   ```

Install Dependencies

Backend

1. Navigate to the backend directory:
   ```bash
   cd backend
   ```
2. Install the required Node.js packages:
   ```bash
   npm install
   ```

Frontend

1. Navigate to the frontend directory:
   ```bash
   cd frontend
   ```
2. Install the required Node.js packages:
   ```bash
   npm install
   ```

Environment Configuration

1. Create a `.env` file in the backend directory.
2. Add necessary environment variables such as:
   - `MONGO_URI`: Your MongoDB connection string.
   - `JWT_SECRET`: A secret key for JWT authentication.
   - `PORT`: The port number for the server (default is 5000).

Running the Application

Backend

1. Start the backend server:
   ```bash
   npm run dev
   ```

```

**Frontend**

1. Start the frontend server:
   ```bash
   npm start
   ```

Accessing the Application

- Open your web browser and go to `http://localhost:3000` to access the application.

Additional Steps

- **Database Setup**: Ensure your MongoDB instance is running and accessible with the credentials provided in your `.env` file.
- **Testing**: Use tools like Postman to test API endpoints.
   ```

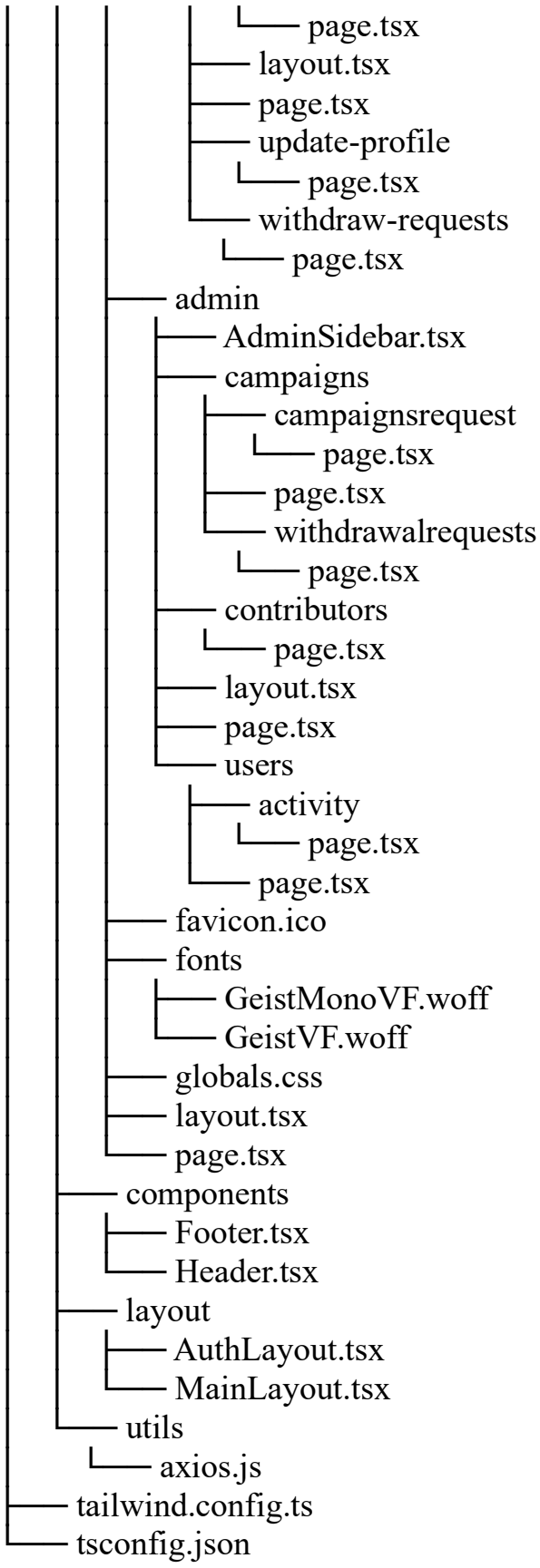
5.Folder structure

Client:

1. Directory Structure

A well-organized React frontend might have the following directory structure:

```
frontend/
├── README.md
├── next-env.d.ts
├── next.config.mjs
├── package-lock.json
├── package.json
├── postcss.config.mjs
├── src
│   ├── app
│   │   ├── (auth)
│   │   │   ├── layout.tsx
│   │   │   ├── login
│   │   │   │   └── page.tsx
│   │   │   └── register
│   │   │       └── page.tsx
│   │   ├── (user)
│   │   │   ├── about
│   │   │   │   └── page.tsx
│   │   │   ├── campaign
│   │   │   │   └── [id]
│   │   │   │       └── page.tsx
│   │   │   ├── campaigns
│   │   │   │   └── page.tsx
│   │   │   ├── contact
│   │   │   │   └── page.tsx
│   │   │   └── profile
│   │   │       ├── UserSidebar.tsx
│   │   │       ├── campaigns
│   │   │       │   ├── new
│   │   │       │   │   └── page.tsx
│   │   │       │   ├── page.tsx
│   │   │       │   └── update
│   │   │       │       └── [id]
│   │   │       │           └── page.tsx
│   │   │       ├── contributions
```


page.tsx
layout.tsx
page.tsx
update-profile
page.tsx
withdraw-requests
page.tsx
admin
AdminSidebar.tsx
campaigns
campaignsrequest
page.tsx
page.tsx
withdrawalrequests
page.tsx
contributors
page.tsx
layout.tsx
page.tsx
users
activity
page.tsx
page.tsx
favicon.ico
fonts
GeistMonoVF.woff
GeistVF.woff
globals.css
layout.tsx
page.tsx
components
Footer.tsx
Header.tsx
layout
AuthLayout.tsx
MainLayout.tsx
utils
axios.js
tailwind.config.ts
tsconfig.json

2. Main Folders and Files

The frontend directory is organized to manage different aspects of the application efficiently, utilizing Next.js for server-side rendering and React components.

Root Files

- **README.md**: Contains documentation and instructions for setting up and using the frontend.
- **next-env.d.ts**: TypeScript declaration file for Next.js, ensuring type safety.
- **next.config.mjs**: Configuration file for customizing Next.js settings.
- **package-lock.json** & **package.json**: Manage dependencies and scripts for the project.
- **postcss.config.mjs**: Configuration for PostCSS, used for processing CSS styles.
- **tailwind.config.ts**: Configuration for Tailwind CSS, enabling utility-first styling.
- **tsconfig.json**: TypeScript configuration file, specifying compiler options.

src Directory

app Directory

This directory organizes pages and layouts using Next.js routing:

- **(auth)**: Handles authentication-related pages.
    - layout.tsx: Layout component for authentication pages.
    - login/page.tsx: Login page component.
    - register/page.tsx: Registration page component.
- **(user)**: Manages user-related pages and features.
    - about/page.tsx: About page component.
    - campaign/[id]/page.tsx: Dynamic campaign detail page.
    - campaigns/page.tsx: List of all campaigns.
    - contact/page.tsx: Contact page component.

    - profile/: Contains user profile management components like:
        - UserSidebar.tsx: Sidebar component for user navigation.
        - Campaign management pages (new, update, etc.).
        - contributions/page.tsx: User contributions page.
        - Profile update and withdrawal request pages.
- **admin**: Admin-specific pages and components.
    - AdminSidebar.tsx: Sidebar component for admin navigation.
    - Campaign and user management pages.

Components

- **components/**: Reusable UI components like Footer.tsx and Header.tsx.

Layout

- **layout/**: Defines layout components such as:
    - AuthLayout.tsx: Layout for authentication pages.
    - MainLayout.tsx: Main layout for general pages.

Utils

- **utils/axios.js**: Configuration for Axios, handling HTTP requests.

Assets

- **favicon.ico**: Icon displayed in the browser tab.
- **fonts/**: Custom fonts used in the application (GeistMonoVF.woff, GeistVF.woff).
- **globals.css**: Global CSS styles applied across the application.

Server:

1. Directory Structure

A well-organized Node.js backend project might follow this structure

backend/

```
├── controller
│   ├── Campaign.js
│   ├── Investment.js
│   ├── User.js
│   └── admin
│       ├── AdminActivity.js
│       ├── AdminCampaign.js
│       ├── AdminInvestment.js
│       └── AdminUser.js
├── index.js
├── middleware
│   ├── AdminAuthorization.js
│   ├── Auth.js
│   └── LogActivity.js
├── models
│   ├── Activity.js
│   ├── Campaign.js
│   ├── Investment.js
│   └── User.js
├── package-lock.json
├── package.json
├── routes
│   ├── Campaign.js
│   ├── Investment.js
│   ├── User.js
```

```
│   └── admin
│       ├── AdminActivity.js
│       ├── AdminCampaign.js
│       ├── AdminDashboard.js
│       ├── AdminInvestment.js
│       ├── AdminUser.js
│       └── index.js
└── utils
    ├── activity.js
    ├── connectDB.js
    └── generateRandomWords.js
```

2. Main Folders and Files

The backend of the Fundraising and Investment Platform is organized to handle server-side logic, data management, and API routing efficiently. This structure ensures maintainability and scalability.

Root Files

- **index.js**: The entry point of the application, responsible for setting up the server, connecting to the database, and initializing middleware and routes.
- **package-lock.json** & **package.json**: Manage dependencies and scripts for the backend project.

controller Directory

Contains logic for handling requests and responses:

- **Campaign.js, Investment.js, User.js**: Handle CRUD operations and business logic for campaigns, investments, and users.
- **admin/**: Contains controllers specifically for admin functionalities:
    - AdminActivity.js: Manages admin-related activities.

    - AdminCampaign.js: Handles campaign management by admins.
    - AdminInvestment.js: Manages investment operations from an admin perspective.
    - AdminUser.js: Admin functionalities related to user management.

middleware Directory

Holds middleware functions for processing requests:

- **AdminAuthorization.js**: Ensures that only admins can access certain routes.
- **Auth.js**: Handles user authentication and authorization.
- **LogActivity.js**: Logs user activities for tracking purposes.

models Directory

Defines data schemas using Mongoose:

- **Activity.js, Campaign.js, Investment.js, User.js**: Define MongoDB schemas for activities, campaigns, investments, and users.

routes Directory

Manages API endpoints:

- **Campaign.js, Investment.js, User.js**: Define routes for handling requests related to campaigns, investments, and users.
- **admin/**: Contains routes specific to admin operations:
    - AdminActivity.js: Routes for admin activity management.
    - AdminCampaign.js: Routes for managing campaigns by admins.
    - AdminDashboard.js: Admin dashboard routes.
    - AdminInvestment.js: Routes for investment management by admins.
    - AdminUser.js: Admin-specific user management routes.
    - index.js: Aggregates all admin routes.

utils Directory

Contains utility functions:

- **activity.js**: Functions related to logging or processing user activities.
- **connectDB.js**: Handles database connection setup.
- **generateRandomWords.js**: Utility for generating random words, possibly for testing or seeding data.

This structured approach ensures clear separation of concerns, making the backend easy to manage and extend.

6.Running the Application

To successfully run the Fundraising and Investment Platform, you need to start both the frontend and backend servers. This requires running two separate processes in different terminal windows or tabs. Below are the detailed steps for setting up and running the application, assuming a standard directory structure:

Directory Structure

```
/fundraising-platform
  /frontend   (React frontend)
  /backend    (Node.js/Express backend)
```

Starting the Backend (Node.js/Express Server)

1. Navigate to the Backend Directory:
   - Open a terminal window.
   - Change directory to the backend folder:
     ```bash
     cd backend
     ```
2. Install Dependencies:
   - Ensure all necessary Node.js packages are installed:
     ```bash
     npm install
     ```

3. Start the Backend Server:

- Run the development server, typically using `nodemon` for automatic reloading on changes:

```bash
npm run dev
```

- This command starts the backend server on a specific port, often `http://localhost:5000`. If there is no `dev` script, you can start the server using:

```bash
node index.js
```

- Ensure your MongoDB instance is running and accessible with the credentials provided in your `.env` file.

Starting the Frontend (React App)

1. Navigate to the Frontend Directory:

- Open another terminal window or tab.

- Change directory to the frontend folder:

```bash
cd frontend
```

2. Install Dependencies:

- Install all required Node.js packages for the React app:

```bash
npm install
```

```

3. Start the React Development Server:

- Launch the frontend server with:

```bash
npm start
```

- This will start the React application on a different port, usually `http://localhost:3000`.

Accessing the Application

- Frontend Access: Open your web browser and navigate to `http://localhost:3000` to access the frontend interface.

- Backend API: The frontend will likely make API requests to the backend server at `http://localhost:5000`. Ensure that API URLs in your React code match this backend server's URL.

Additional Considerations

- Environment Variables: Ensure that environment variables are correctly set in a `.env` file in your backend directory. This includes database connection strings and any secret keys.

- Testing API Endpoints: Use tools like Postman to test API endpoints and ensure they are functioning as expected.

- Troubleshooting: If you encounter issues, check console logs for errors or warnings that can guide you in resolving them.
```

7.API Documentation

The backend of the Fundraising and Investment Platform exposes a set of RESTful API endpoints for managing users, authentication, campaigns, and investments. Below is an example of how these endpoints are structured.

Authentication Endpoints

1. **POST /api/auth/register**
   - **Description**: Registers a new user.
   - **Request Body**:
     ```
     json
     {
       "name": "John Doe",
       "email": "johndoe@example.com",
       "password": "password123"
     }
     ```
   - **Response**:
     ```
     json
     {
       "message": "User registered successfully",
       "token": "JWT_TOKEN"
     }
     ```
2. **POST /api/auth/login**
   - **Description**: Logs in a user and returns a JWT token.
   - **Request Body**:
     ```
     json
     {
       "email": "johndoe@example.com",
       "password": "password123"
     }
     ```
   - **Response**:
     ```
     json
     {
       "message": "Login successful",
       "token": "JWT_TOKEN"
     }
     ```
3. **GET /api/auth/me**
   - **Description**: Retrieves details of the currently authenticated user.

- **Headers**:
  ```
  json
  {
    "Authorization": "Bearer JWT_TOKEN"
  }
  ```
- **Response**:
  ```
  json
  {
    "id": "userId",
    "name": "John Doe",
    "email": "johndoe@example.com"
  }
  ```

Campaign Endpoints

1. **GET /api/campaigns**
   - **Description**: Retrieves a list of all campaigns.
   - **Headers**:
     ```
     json
     {
       "Authorization": "Bearer JWT_TOKEN"
     }
     ```
   - **Response**:
     ```
     json
     [
       {
         "id": "campaignId1",
         "title": "Campaign Title",
         "description": "Description of the campaign",
         "goalAmount": 1000,
         "currentAmountRaised": 500,
         "status": "active"
       }
     ]
     ```
2. **POST /api/campaigns**
   - **Description**: Creates a new campaign.
   - **Headers**:
     ```
     json
     {
       "Authorization": "Bearer JWT_TOKEN"
     }
     ```
   - **Request Body**:

```
json
{
  "title": "New Campaign",
  "description": "Details about the campaign",
  "goalAmount": 1000
}
```

- **Response**:

```
json
{
  "message": "Campaign created successfully",
  "campaignId": "newCampaignId"
}
```

Investment Endpoints

1. **GET /api/investments**
   - **Description**: Retrieves a list of all investments for the authenticated user.
   - **Headers**:

```
json
{
  "Authorization": "Bearer JWT_TOKEN"
}
```

   - **Response**:

```
json
[
  {
    "id": "investmentId1",
    "campaignId": "campaignId1",
    "amount": 100,
    "status": "confirmed"
  }
]
```

2. **POST /api/investments**
   - **Description**: Adds a new investment for the authenticated user.
   - **Headers**:

```
json
{
  "Authorization": "Bearer JWT_TOKEN"
}
```

   - **Request Body**:

```
json
```

```
{
  "campaignId": "campaignId1",
  "amount": 100
}
```

- **Response**:

```
json
{
  "message": "Investment added successfully",
  "investmentId": "newInvestmentId"
}
```

8.Authentication

Authentication and authorization are critical components of the Fundraising and Investment Platform, ensuring that only registered users can access their data and perform specific actions securely.

1. Authentication

Authentication is the process of verifying a user's identity, typically using credentials like email and password. Once authenticated, a user is issued a JWT (JSON Web Token), which allows the server to recognize the user in future requests.

How Authentication Works

- **Registration (/api/auth/register)**:
    1. **User Sign-up**: When a user signs up, they send their name, email, and password to the backend via a POST request.
    2. **Password Hashing**: The backend uses a library like bcrypt to hash the password before saving it in the database for security purposes.
    3. **User Creation**: After successful validation and hashing, the user's data (name, email, and hashed password) is stored in MongoDB.
    4. **JWT Token Generation**: Once registered, the backend creates a JWT for the user and sends it as part of the response. This token can be used for subsequent authentication (login).

```
javascript
const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, {
  expiresIn: '1h',
});
```

- **Login (/api/auth/login)**:
    1. **User Login**: When a user logs in, they provide their email and password.
    2. **Password Verification**: The backend checks if the user exists and compares the provided password with the hashed password stored in the database using bcrypt.compare().
    3. **JWT Token Issuance**: If the login credentials are valid, the backend generates a JWT token and returns it to the user.

```
Example response after login:
json
{
  "message": "Login successful",
```

```
  "token": "JWT_TOKEN"
}
```

2. Authorization

Authorization ensures that a user can only access resources (e.g., campaigns, profiles) they are permitted to access. This is enforced using the JWT token that was issued during authentication.

How Authorization Works

- **JWT Token Storage**:
  - After login, the JWT token is typically stored on the client-side (frontend) in localStorage or sessionStorage (or more securely, in a cookie).
  - The token is included in the Authorization header of every HTTP request that requires authorization (e.g., accessing user's campaigns).

    Example request with JWT:

    ```
    json
    {
      "Authorization": "Bearer JWT_TOKEN"
    }
    ```
- **Access Control**:
  - **Fetching User's Campaigns**: When a user requests to fetch their campaigns (GET /api/campaigns), middleware verifies their token, extracts their user ID, and queries the database for campaigns associated with that user ID.

    ```
    javascript
    // Get all campaigns for the authenticated user
    router.get('/campaigns', authenticateToken, async (req, res) => {
      const campaigns = await Campaign.find({ userId: req.user });
      res.json(campaigns);
    });
    ```
  - **Updating or Deleting a Campaign**: For routes like updating (PUT /api/campaigns/:id) or deleting (DELETE /api/campaigns/:id), middleware ensures that the campaign belongs to the authenticated user by checking the campaign's userId field against req.user (the authenticated user's ID).Example of protecting update logic:

    ```
    javascript
    router.put('/campaigns/:id', authenticateToken, async (req, res) => {
      const campaign = await Campaign.findById(req.params.id);
    ```

```
  if (!campaign || campaign.userId.toString() !== req.user) {
    return res.status(403).json({ message: 'Access denied' });
  }

  // Proceed with updating the campaign
  campaign.title = req.body.title;
  await campaign.save();
  res.json(campaign);
});
```

Tokens

Tokens, especially JSON Web Tokens (JWTs), are commonly used for authentication and session management.

JSON Web Tokens (JWT)

JWTs are a popular method for authentication in modern web applications, especially in full-stack MERN projects. They provide a stateless and scalable solution for managing user sessions and verifying their identity.

- **Structure of JWTs**:
  - **Header**: Contains the type of token (JWT) and the signing algorithm (e.g., HS256 for HMAC-SHA256).
  - **Payload**: Contains claims, which are typically user data, such as the user's ID, email, and any other relevant information.
  - **Signature**: A cryptographic signature used to verify that the token has not been tampered with.

    Example response after successful login:

    ```
    json
    {
      "message": "Login successful",
      "token": "JWT_TOKEN"
    }
    ```

How JWT is Used in the Platform

- **Authentication Flow**:
  1. **User Login**: The user logs in by sending their credentials (email and password) to the backend (POST /api/auth/login).

2. **Token Issuance**: If login credentials are valid, the backend generates a JWT, signs it using a secret key, and returns it to the user.
3. **Token Storage**: The frontend stores the JWT (typically in localStorage or sessionStorage). This token is included in headers of subsequent API requests.

By implementing robust authentication and authorization mechanisms using JWTs, the platform ensures secure access control while maintaining a seamless experience for users across different sessions.

10.User Interface

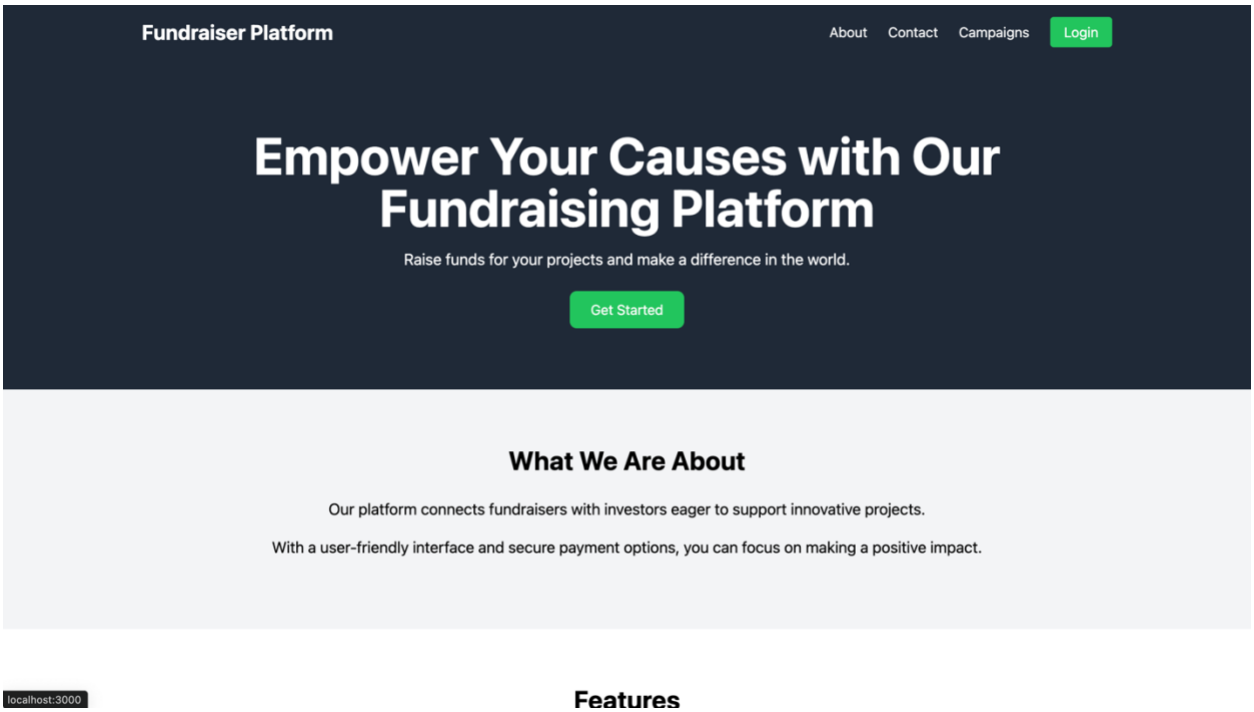


Fig 1: Home Page

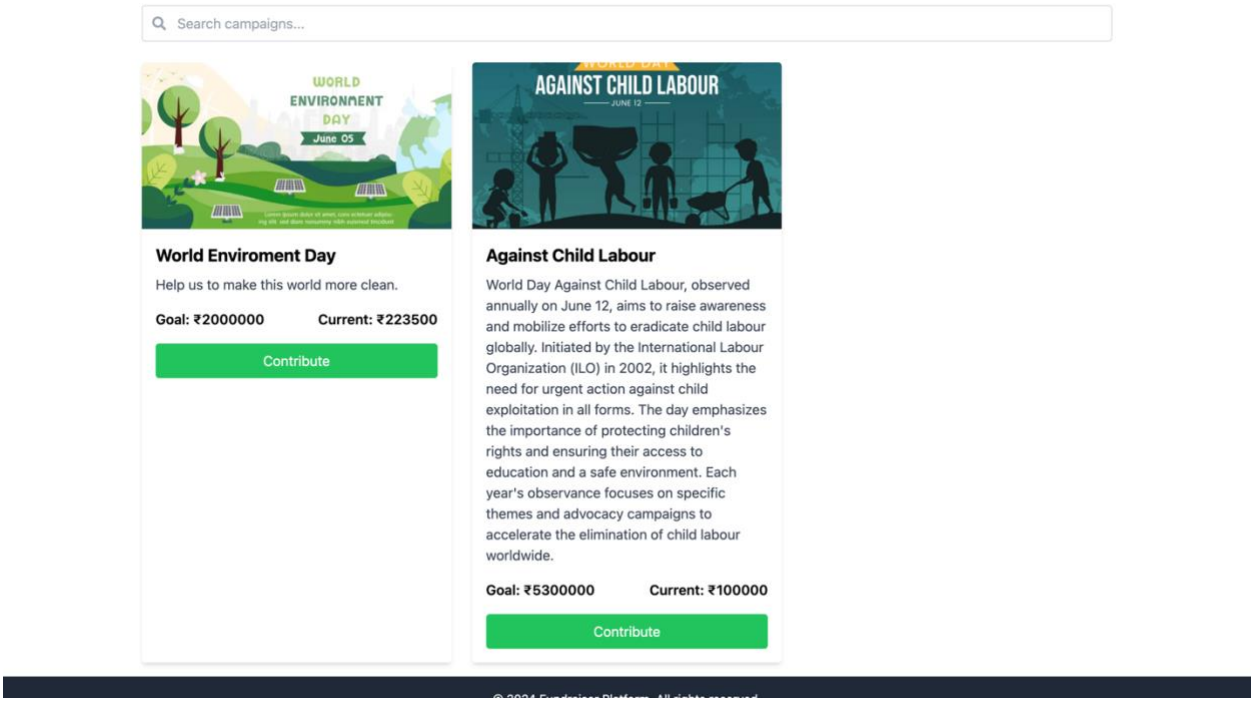


Fig 2: Campaigns Page

Fig 3: Campaign Page

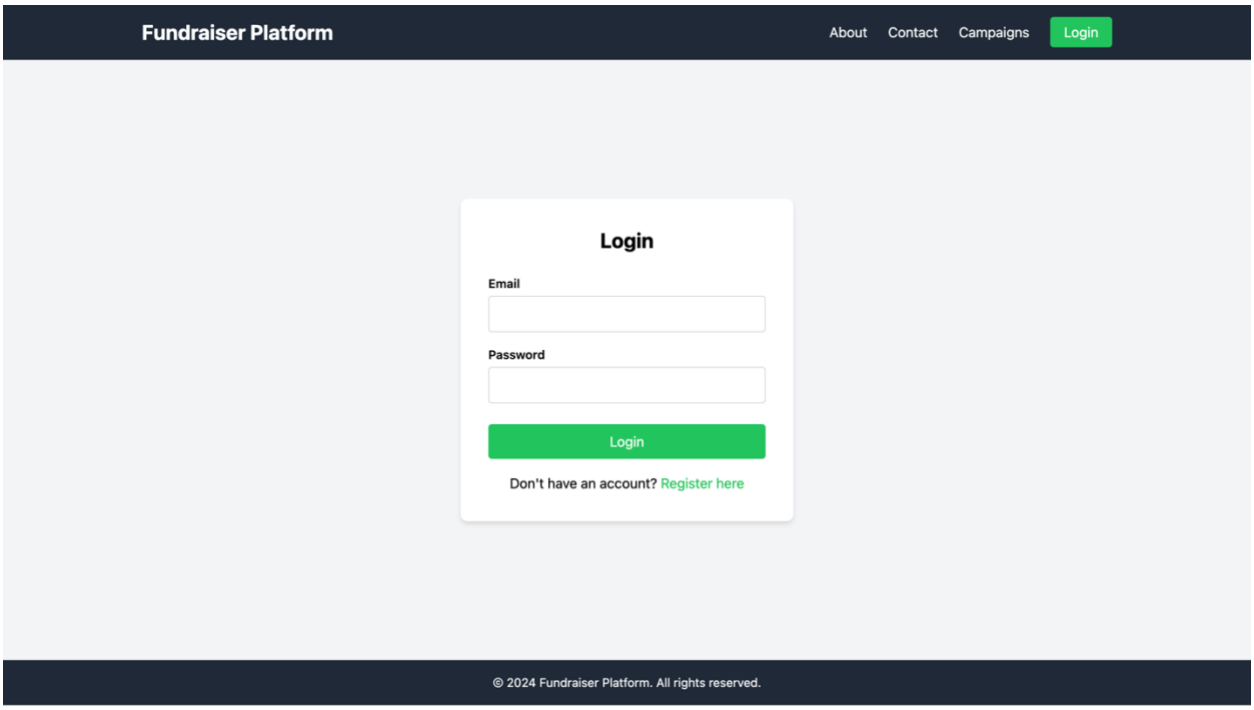


Fig 4: Login Page

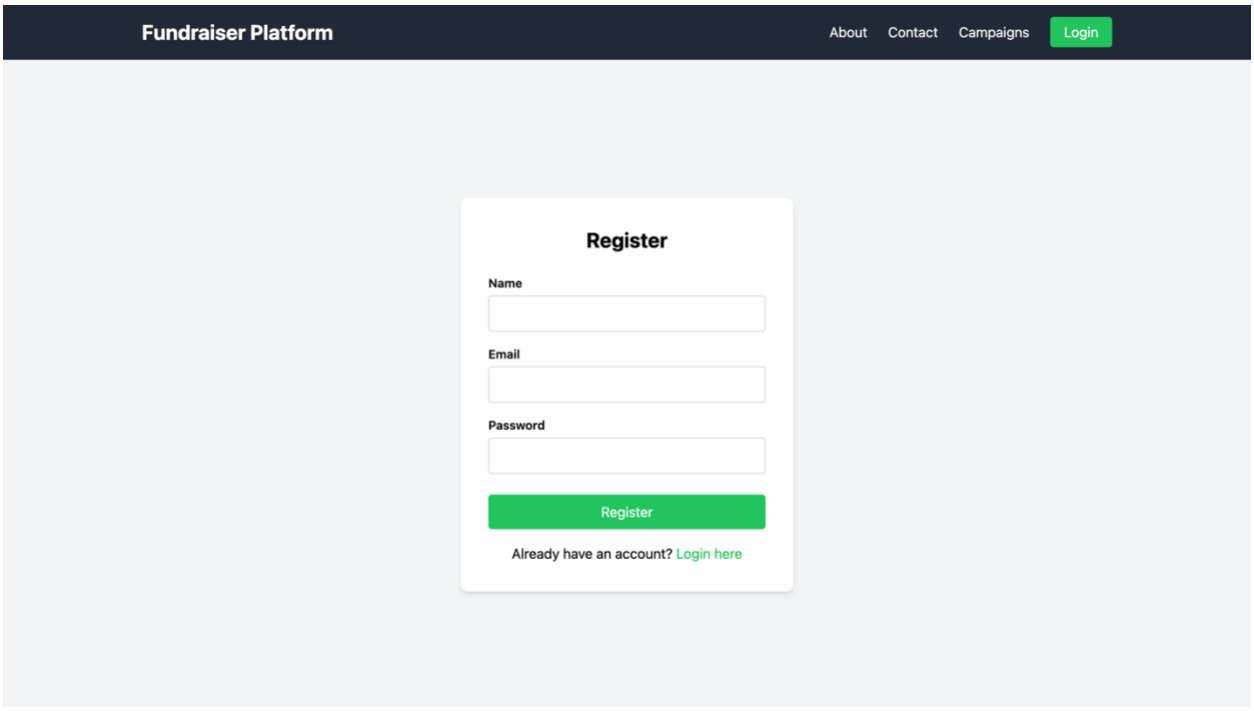


Fig 5: Register Page

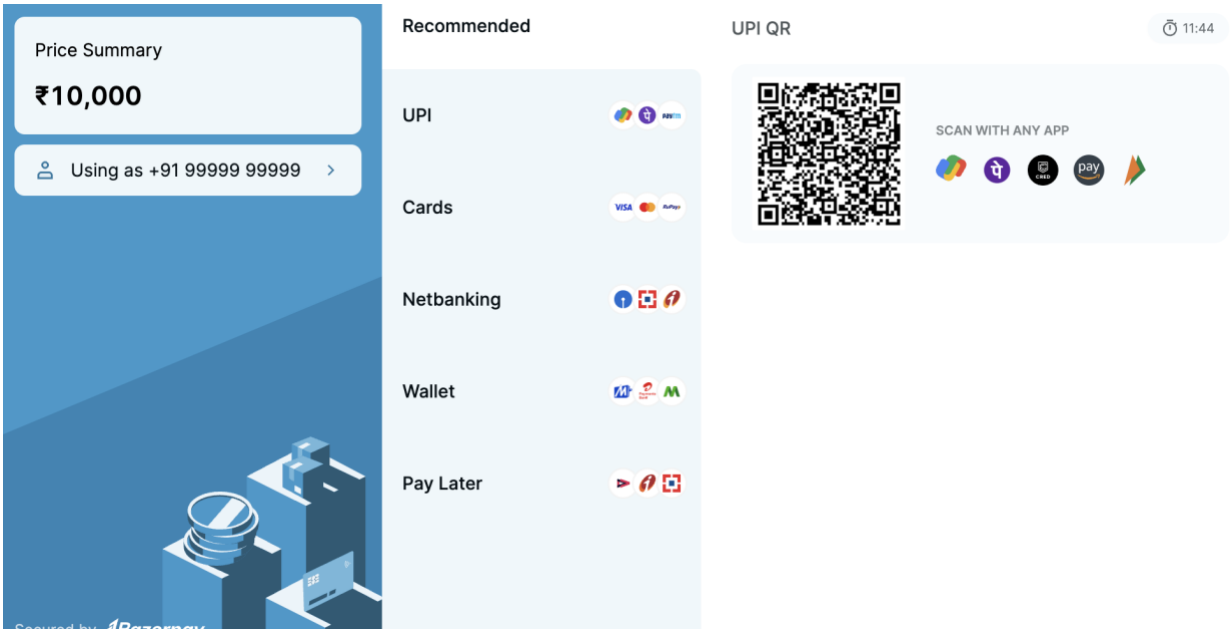


Fig 6: Razor Pay

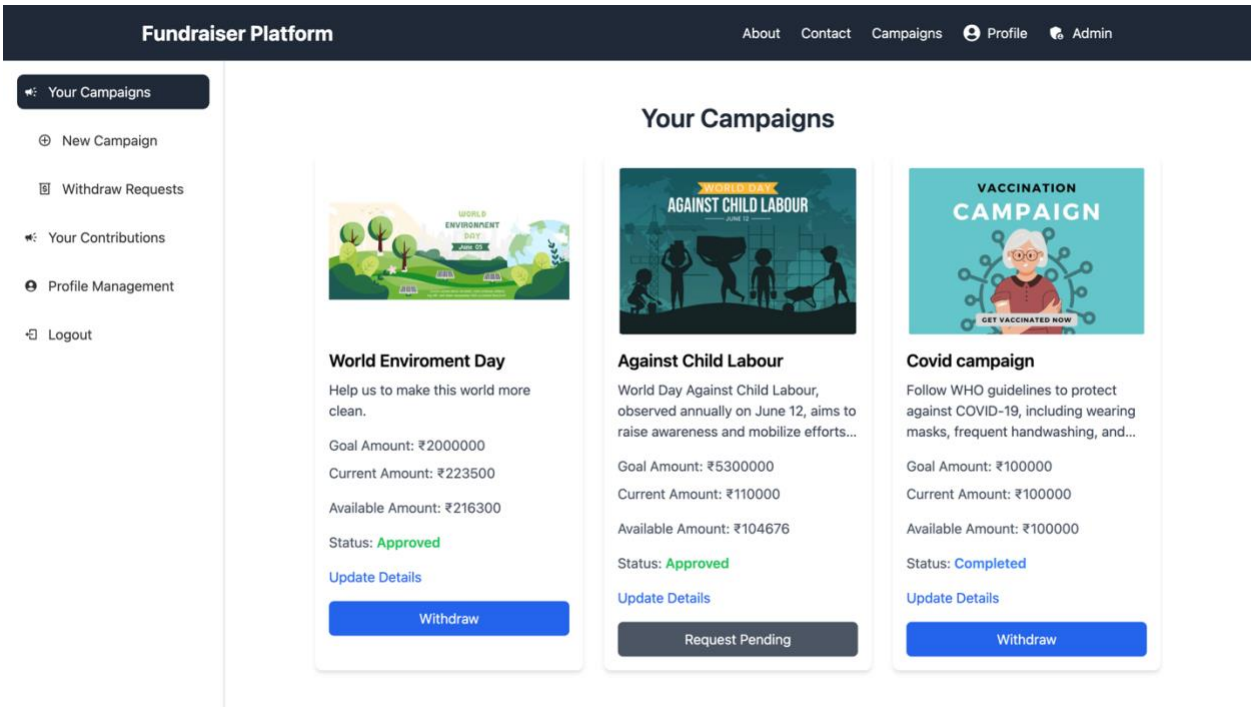


Fig 7: Your Campaigns Page

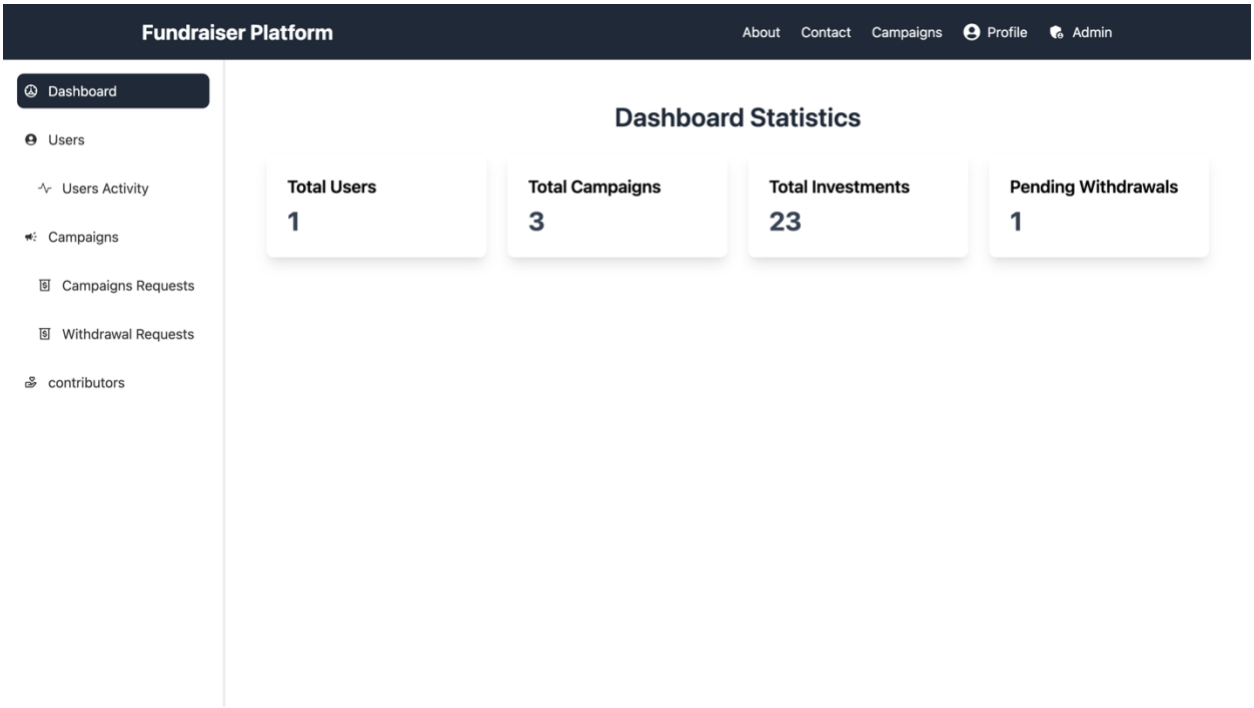


Fig 8: Admin Panel

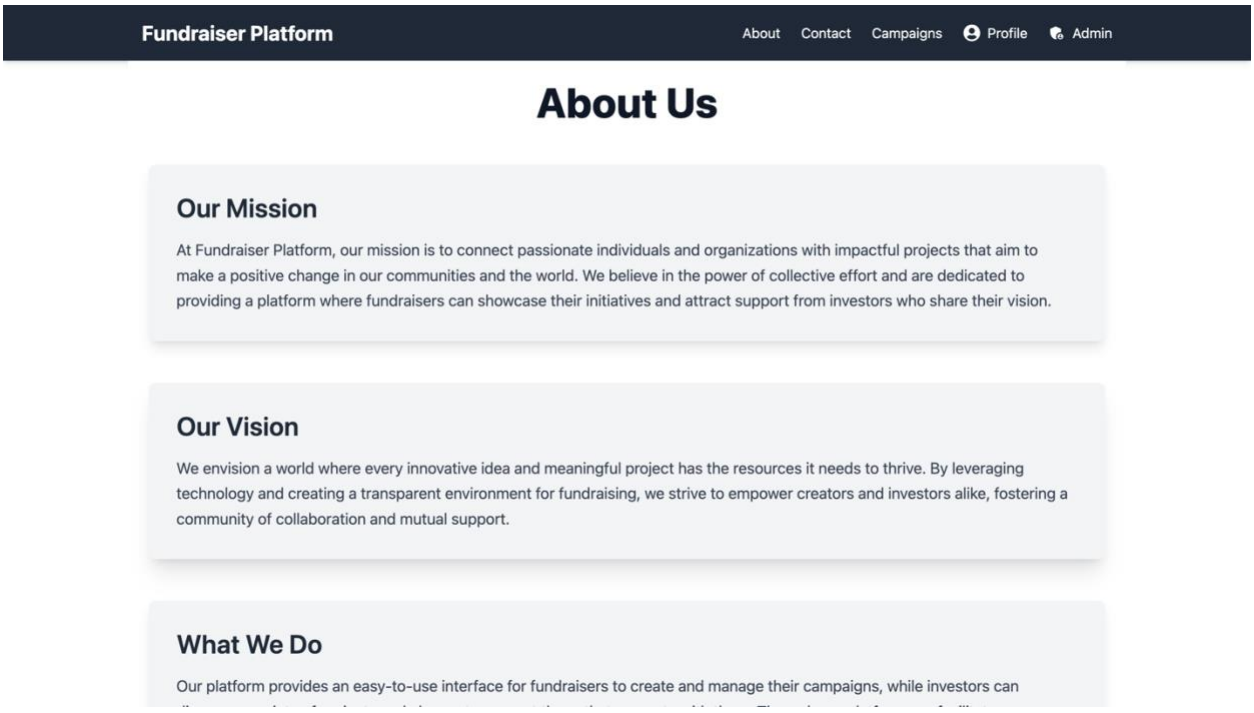


Fig 9: About us

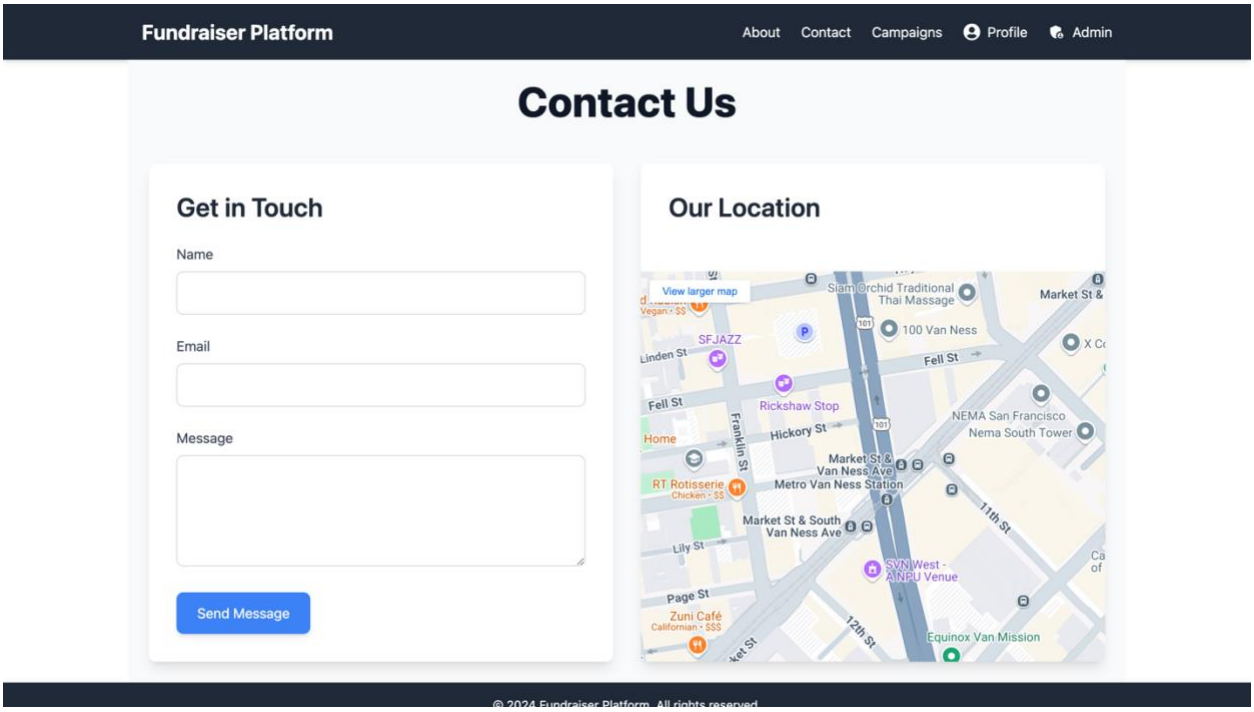


Fig 10: Contact Page

10.Testing

Testing is an essential phase in the development of the Fundraising and Investment Platform, ensuring that the application functions correctly and meets user requirements. The testing process involves both black box and white box testing methods, tailored to validate different aspects of the system.

Black Box Testing
Black box testing focuses on verifying the functionality of the application without considering its internal code structure. It validates whether the expected output matches the provided input.

Test Cases

- **Test Case 1: User Registration**
  - **Input**:
    json
    {
      "email": "testuser@email.com",
      "password": "testpassword"
    }
  - **Expected Output**:
    json
    {
      "message": "User registered successfully"
    }
- **Test Case 2: Create Campaign**
  - **Input**:
    json
    {
      "title": "New Campaign",
      "description": "This is a test campaign",
      "goalAmount": 5000
    }
  - **Expected Output**:
    json
    {
      "message": "Campaign created successfully"
    }

White Box Testing
White box testing examines the internal logic, structure, and code paths of the application. It ensures that all parts of the code are functioning as expected.

Test Cases

- **Test Case 1: Update User Profile**
  - **Input**:
    ```
    json
    {
      "email": "updateduser@email.com",
      "password": "newpassword"
    }
    ```
  - **Expected Output**:
    ```
    json
    {
      "message": "User profile updated successfully"
    }
    ```
- **Test Case 2: Edit Campaign Details**
  - **Input**:
    ```
    json
    {
      "title": "Updated Campaign Title",
      "description": "Updated description for the campaign"
    }
    ```
  - **Expected Output**:
    ```
    json
    {
      "message": "Campaign details updated successfully"
    }
    ```

Testing Methodology

- **Black Box Testing**: Focuses on user interactions and ensures that the application meets functional requirements. It is used to validate features like registration, login, and campaign creation.
- **White Box Testing**: Involves detailed examination of code logic, including loops, conditions, and data flow. It ensures that each function performs as intended and optimizes code efficiency.

11. Known Issues

Despite the robust design and implementation of the Fundraising and Investment Platform, several common issues can arise during development or user interaction. These issues may affect both developers and end-users, requiring attention to ensure a smooth experience.

1. Token Expiration and Reauthentication Issues
   - **Issue**: Users may experience unexpected logouts if the JWT access token expires without a refresh token mechanism in place.
     - **Symptoms**: Users might encounter sudden logouts or errors when trying to access protected routes after a certain period (e.g., one hour).
     - **Solution**: Implement a refresh token mechanism that allows the access token to be refreshed automatically, minimizing disruptions and avoiding the need for users to log in again frequently.

2. Inconsistent Data Between Client and Server
   - **Issue**: Data inconsistencies can occur if there is a lack of synchronization between the frontend and backend, especially when using optimistic updates (updating the UI before server confirmation).
     - **Symptoms**: Users may notice incorrect balances, duplicate entries, or items that appear deleted still showing up in lists.
     - **Solution**: Ensure that UI changes are only made after receiving successful responses from the server. Implement error handling to rollback changes if necessary.

3. Memory Leaks in React Components
   - **Issue**: Memory leaks can occur in React components when asynchronous operations (like API calls) continue after a component has unmounted.
     - **Symptoms**: Developers might see warnings such as "Can't perform a React state update on an unmounted component" in the console, or observe increasing memory usage over time.
     - **Solution**: Use cleanup functions in useEffect hooks to cancel ongoing asynchronous requests when components unmount. This helps prevent memory leaks and ensures efficient resource management.

4. Slow Performance Due to Inefficient MongoDB Queries

- **Issue**: Inefficient queries in MongoDB can lead to slow application performance, especially when handling large datasets.
    - **Symptoms**: Users may experience slow response times when fetching data, particularly with large numbers of records.
    - **Solution**: Optimize database queries by using indexes on frequently queried fields (such as user ID or date). Utilize MongoDB's aggregation framework or Mongoose queries efficiently to improve performance.

These known issues highlight areas for improvement and optimization within the platform. Addressing these challenges proactively will enhance both developer efficiency and user satisfaction, ensuring a more reliable and responsive application.

12.Future Enhancements

The Fundraising and Investment Platform is designed to be scalable and adaptable, allowing for the integration of new features and improvements over time. Here are some potential future enhancements that could further enhance the platform's functionality and user experience:

1. Crowdfunding Features

- Micro-Investments: Introduce the ability for multiple investors to contribute smaller amounts to a campaign, enabling fundraisers to reach their goals through collective micro-investments.
- Reward-Based Funding: Implement a system where fundraisers can offer non-monetary rewards to investors based on their contributions, similar to popular crowdfunding platforms.

2. Advanced Analytics and Reporting

- Investor Insights: Provide detailed analytics for investors to track their investment performance, returns, and overall portfolio health.
- Campaign Performance Metrics: Offer fundraisers comprehensive analytics on campaign performance, including engagement rates and funding trends.

3. Enhanced User Engagement

- Community Features: Develop forums or discussion boards where fundraisers and investors can interact, share insights, and build a community.
- Social Sharing Tools: Enable users to easily share campaigns on social media platforms to increase visibility and attract more investors.

4. Improved Security Measures

- Two-Factor Authentication (2FA): Implement 2FA for added security during login processes, protecting user accounts from unauthorized access.
- Fraud Detection Algorithms: Develop algorithms to detect and prevent fraudulent activities on the platform, ensuring a safe environment for all users.

5. Mobile Application Development

- Native Mobile Apps: Create dedicated mobile applications for iOS and Android platforms to provide users with seamless access to the platform's features on-the-go.

- Push Notifications: Implement push notifications to keep users updated on campaign progress, investment opportunities, and important platform announcements.

6. Integration with Blockchain Technology

- Smart Contracts: Explore the use of blockchain technology for implementing smart contracts that automate transactions and ensure transparency in fund allocation.
- Decentralized Identity Verification: Use blockchain for secure identity verification processes, enhancing trust between fundraisers and investors.

7. Personalized Investment Recommendations

- AI-Powered Suggestions: Utilize artificial intelligence to recommend investment opportunities to users based on their preferences and past activities.
- Customized Alerts: Allow users to set up alerts for specific types of campaigns or investment opportunities that match their interests.

By considering these enhancements, the Fundraising and Investment Platform can continue to evolve, meeting the changing needs of its users while maintaining a competitive edge in the market. These improvements will not only enhance user satisfaction but also contribute to building a more robust and innovative fundraising ecosystem.