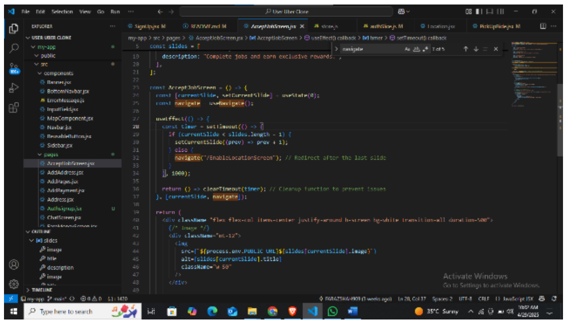
AcceptJobScreen Code:



```
const slides = [
    description: "Complete jobs and earn exclusive rewards.",
  },
];

const AcceptJobScreen = () => {
  const [currentSlide, setCurrentSlide] = useState(0);
  const navigate = useNavigate();

  useEffect(() => {
    const timer = setTimeout(() => {
      if (currentSlide < slides.length - 1) {
        setCurrentSlide((prev) => prev + 1);
      } else {
        navigate("/EnableLocationScreen"); // Redirect after the last slide
      }
    }, 1000);

    return () => clearTimeout(timer); // Cleanup function to prevent issues
  }, [currentSlide, navigate]);

  return (
    <div className="flex flex-col items-center justify-around h-screen bg-white transition-all duration-500">
      {/* Image */}
      <div className="mt-12">
        <img
          src={`${process.env.PUBLIC_URL}${slides[currentSlide].image}`}
          alt={slides[currentSlide].title}
          className="w-50"
        />
      </div>
```

What the Code Does:

1. **Slideshow Content**:
   - There are 3 slides, each with:
     - An image (e.g., "Accept a Job")
     - A title
     - A short description
2. **Automatic Slide Change**:
   - Each slide stays for 1 second (1000 milliseconds).
   - After the last slide, the app automatically sends the user to the EnableLocationScreen.
3. **User Controls**:
   - **Dots at the bottom** show which slide is currently active.
   - A **"Skip" button** lets users jump straight to the next screen without waiting.
4. **Design**:
   - The screen is centered and has a white background.
   - The image, title, and description are displayed neatly.

How It Works:

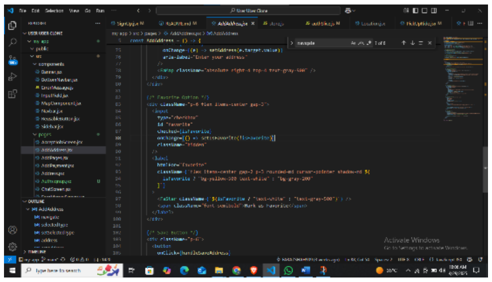
- useState keeps track of which slide is currently showing.
- useEffect handles the automatic slide change after 1 second.
- useNavigate helps move to the next screen when the slides finish.

- The slides array holds all the slide data (image paths, titles, descriptions).

Components Used:

- CustomLink: A reusable button for navigation (like the "Skip" button).
- **Simple dots** at the bottom to indicate progress.

AddAddress Screen Code:



```
const AddAddress = () => {
            onChange={(e) => setAddress(e.target.value)}
            aria-label="Enter your address"
          />
          <FaMap className="absolute right-4 top-4 text-gray-500" />
        </div>
      </div>

      {/* Favorite Option */}
      <div className="p-6 flex items-center gap-3">
        <input
          type="checkbox"
          id="favorite"
          checked={isFavorite}
          onChange={() => setIsFavorite(!isFavorite)}
          className="hidden"
        />
        <label
          htmlFor="favorite"
          className={`flex items-center gap-2 p-3 rounded-md cursor-pointer shadow-md ${
            isFavorite ? "bg-yellow-500 text-white" : "bg-gray-200"
          }`}
        >
          <FaStar className={`${isFavorite ? "text-white" : "text-gray-500"}`} />
          <span className="font-semibold">Mark as Favorite</span>
        </label>
      </div>

      {/* Save Button */}
      <div className="p-6">
        <button
          onClick={handleSaveAddress}
```

What the Code Does:

1. **Header Section**
   - A yellow header with a back button (FaArrowLeft) and the title **"Add New Address"**.
   - Clicking the back button takes the user to the previous screen.
2. **Address Type Selection**
   - Users can choose between **Home, Work, or Custom** address types.
   - Each type has an icon (FaHome, FaBriefcase, FaMapMarkerAlt).
   - The selected type is highlighted in yellow.
3. **Address Input Field**
   - A text box where users can type or search for an address.
   - Includes a map icon (FaMap) on the right side.
4. **Favorite Option**
   - A toggle button (FaStar) lets users mark the address as a **favorite**.
   - Turns yellow when selected.
5. **Save Button**
   - A black button (FaCheckCircle) saves the address.
   - The button is **disabled if no address is entered**.
   - Saved addresses are stored in **localStorage** (a browser storage system).

How It Works:

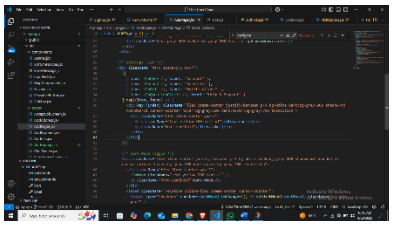
- useState tracks:

    - The selected address type (Home, Work, Custom).
    - The entered address text.
    - Whether the address is marked as a favorite.
- localStorage stores the addresses so they persist even after closing the app.
- useNavigate helps move back to the previous screen or the address list after saving.

Key Features:

Simple UI with icons and clear buttons.
Dynamic selection (selected address type is highlighted).
Data persistence (saved addresses stay even after refresh).
User-friendly validation (save button disabled if no address is entered).

Addpage Screen Code:


[illegible]

Basic Structure

1. Imports: At the top, we import React and some icons we'll use (arrow, user, lock, bell, etc.) from the "react-icons" library.
2. Dark Mode: We use useState to create a darkMode variable that can be toggled between light and dark themes.

What the Page Shows

1. Header Section (Yellow Bar at Top)

- Has a back arrow button (left side) that takes you to the previous page when clicked
- Shows "Settings" as the title in the center
- The whole bar is yellow (bg-yellow-500)

2. Profile Section

- Shows a user's profile picture (using a sample image)
- Displays the name "John Doe" and email "johndoe@email.com"
- Has a white background (or dark gray in dark mode)
- Rounded corners and shadow for nice appearance

3. Settings Options List

- Shows 4 different options with icons:
  - Account (user icon)
  - Security (lock icon)
  - Notifications (bell icon)
  - Help & Support (question mark icon)

Each option:

- Has an icon on the left (in yellow)
- Shows the option name
- Changes color slightly when you hover over it

4. Dark Mode Toggle

- Shows a moon icon and "Dark Mode" text
- Has a switch button that toggles between light and dark mode
- When you click the switch, it changes the whole page's colors

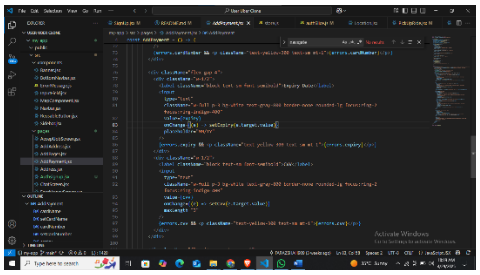
Special Features

- Dark Mode: The entire page changes colors when dark mode is turned on
    - Light mode: light gray background with black text
    - Dark mode: dark gray background with white text
- Hover Effects: Options slightly change color when you point at them
- Responsive Design: Uses Tailwind CSS classes to look good on mobile screens

How It Works

- When you click the dark mode switch, it calls setDarkMode to update the state
- This state change makes React re-render the component with the new colors
- All colors change automatically because they're based on the darkMode variable

Payment Method Code:



```
const AddPayment = () => {
            />
            {errors.cardNumber && <p className="text-yellow-300 text-sm mt-1">{errors.cardNumber}</p>}
          </div>

          <div className="flex gap-4">
            <div className="w-1/2">
              <label className="block text-sm font-semibold">Expiry Date</label>
              <input
                type="text"
                className="w-full p-3 bg-white text-gray-800 border-none rounded-lg focus:ring-2
                focus:ring-indigo-400"
                value={expiry}
                onChange={(e) => setExpiry(e.target.value)}
                placeholder="MM/YY"
              />
              {errors.expiry && <p className="text-yellow-300 text-sm mt-1">{errors.expiry}</p>}
            </div>
            <div className="w-1/2">
              <label className="block text-sm font-semibold">CVV</label>
              <input
                type="text"
                className="w-full p-3 bg-white text-gray-800 border-none rounded-lg focus:ring-2
                focus:ring-indigo-400"
                value={cvv}
                onChange={(e) => setCvv(e.target.value)}
                maxLength="3"
              />
              {errors.cvv && <p className="text-yellow-300 text-sm mt-1">{errors.cvv}</p>}
            </div>
          </div>
```

What This Code Does

1. Imports:
   - We import React and the useState hook for managing form data
   - We import two icons (ArrowLeft and CreditCard) from the "lucide-react" library
2. State Management:
   - We track all the card details (name, number, expiry, CVV) using useState
   - We track whether to save the card (saveCard) and any form errors (errors)

The Form Sections

1. Header Section

- Purple-to-blue gradient background
- Back arrow button at the top left
- Credit card icon in the center
- "Add Payment Method" title

2. Form Fields

1. **Cardholder Name:**
   - Text input for the name on the card
   - Shows error if left empty
2. **Card Number:**
   - Input for 16-digit card number
   - Shows error if not exactly 16 digits

3. **Expiry Date and CVV (side by side):**
    - Expiry date must be in MM/YY format
    - CVV must be exactly 3 digits
    - Both show errors if format is wrong
4. **Save Card Checkbox:**
    - Option to save the card for future payments
    - Can be toggled on/off

3. Submit Button

- White button that turns blue when hovered
- Submits the form when clicked

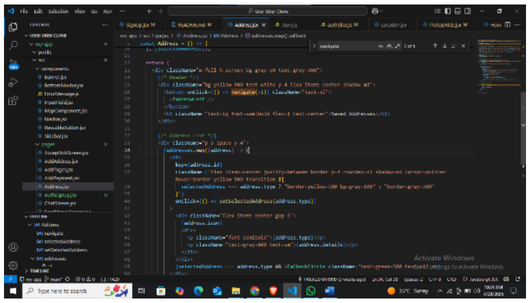
Validation Features

- **Checks that all fields are filled correctly:**
    - Name can't be empty
    - Card number must be 16 digits
    - Expiry must be MM/YY format
    - CVV must be 3 digits
- Shows yellow error messages below any invalid fields
- Only submits if all fields are valid (shows success alert)

How It Works

1. When you type in any field, it updates the corresponding state variable
2. When you click submit:
    - It checks all validations
    - Shows errors if any
    - If everything is correct, shows "Payment Submitted Successfully!" alert

Address Selection Page:



```
const Address = () => {
  }, [selectedAddress]);

  return (
    <div className="w-full h-screen bg-gray-50 text-gray-900">
      {/* Header */}
      <div className="bg-yellow-500 text-white p-4 flex items-center shadow-md">
        <button onClick={() => navigate(-1)} className="text-xl">
          <FaArrowLeft />
        </button>
        <h2 className="text-lg font-semibold flex-1 text-center">Saved Addresses</h2>
      </div>

      {/* Address List */}
      <div className="p-6 space-y-4">
        {addresses.map((address) => (
          <div
            key={address.id}
            className={`flex items-center justify-between border p-4 rounded-xl shadow-md cursor-pointer
            hover:border-yellow-500 transition ${
              selectedAddress === address.type ? "border-yellow-500 bg-gray-100" : "border-gray-300"
            }`}
            onClick={() => setSelectedAddress(address.type)}
          >
            <div className="flex items-center gap-3">
              {address.icon}
              <div>
                <p className="font-semibold">{address.type}</p>
                <p className="text-gray-600 text-sm">{address.details}</p>
              </div>
            </div>
            {selectedAddress === address.type && <FaCheckCircle className="text-green-500 text-
          </div>
```

What This Page Does

1. **Header Section (Yellow Bar at Top):**
   - Has a back arrow button (left side) that returns to the previous page
   - Shows "Saved Addresses" as the title in the center
   - The whole bar is yellow (bg-yellow-500)
2. **Address List:**
   - Shows 3 pre-saved addresses:
     - Home (with home icon)
     - Work (with briefcase icon)
     - Custom (with map marker icon)
   - Each address shows its type (bold) and full details
   - The currently selected address has:
     - Yellow border
     - Light gray background
     - Green checkmark icon
3. **Add New Address Button:**
   - Black button at the bottom with a plus icon
   - When clicked, it navigates to "/AddAddress" (though this route isn't shown in this code)

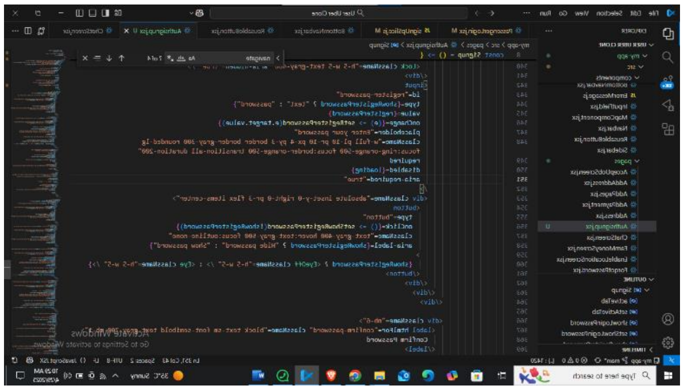
Special Features

- **Address Selection:**
    - Click any address to select it
    - Selected address gets highlighted visually
    - Selection is saved in browser storage (so it remembers your choice)
- **Icons:**
    - Home address has a yellow home icon
    - Other addresses have gray icons
    - Selected address shows green checkmark
- **Storage:**
    - Uses localStorage to remember your selected address even if you close the app
    - Saves automatically when you change selection

Technical Details

1. **State Management:**
    - selectedAddress tracks which address is chosen
    - Defaults to "Home" if nothing was saved before
2. **Effects:**
    - useEffect automatically saves the selected address whenever it changes
3. **Navigation:**
    - Uses React Router's useNavigate for the back button and "Add New" button
4. **Styling:**
    - Uses Tailwind CSS for all styling
    - Hover effects on addresses and buttons
    - Shadow and rounded corners for nice visual appearance

Signup/Login Page:



```
            <Lock className="h-5 w-5 text-gray-400" aria-hidden="true" />
          </div>
          <input
            id="register-password"
            type={showRegisterPassword ? "text" : "password"}
            value={registerPassword}
            onChange={(e) => setRegisterPassword(e.target.value)}
            placeholder="Enter your password"
            className="w-full pl-10 pr-10 px-4 py-3 border border-gray-300 rounded-lg
              focus:ring-orange-500 focus:border-orange-500 transition-all duration-200"
            required
            disabled={loading}
            aria-required="true"
          />
          <div className="absolute inset-y-0 right-0 pr-3 flex items-center">
            <button
              type="button"
              onClick={() => setShowRegisterPassword(!showRegisterPassword)}
              className="text-gray-400 hover:text-gray-500 focus:outline-none"
              aria-label={showRegisterPassword ? "Hide password" : "Show password"}
            >
              {showRegisterPassword ? <EyeOff className="h-5 w-5" /> : <Eye className="h-5 w-5" />}
            </button>
          </div>
        </div>
      </div>

      <div className="mb-6">
        <label htmlFor="confirm-password" className="block text-sm font-semibold text-gray-700 mb-2">
          Confirm Password
        </label>
```

What This Page Does

1. Dual Functionality (Tabs)

- Has two tabs at the top:
  - Sign In (for existing users)
  - Sign Up (for new users)
- Users can switch between tabs by clicking them

2. Sign In Form (Login)

- Email and password fields
- "Remember me" checkbox
- "Forgot password" link
- Password visibility toggle (eye icon)
- Orange "Sign In" button
- Link to switch to Sign Up

3. Sign Up Form (Registration)

- Full name, email, password, and confirm password fields
- Terms and conditions checkbox
- Password visibility toggle for both password fields
- Orange "Create Account" button
- Link to switch to Sign In

Special Features

1. Form Validation

- Checks that passwords match during registration
- Requires all fields to be filled
- Validates email format
- Requires accepting terms for registration

2. User Experience

- Loading states during form submission
- Password visibility toggle
- Error messages using SweetAlert (popup notifications)
- Success messages after successful actions
- Smooth transitions between tabs

3. Security

- Passwords are hidden by default
- Connection to backend API (http://localhost:5000)
- Stores authentication token in localStorage
- Timeout for API requests (5 seconds)

4. Technical Details

- Uses React hooks (useState, useEffect)
- React Router for navigation
- SweetAlert for notifications
- Tailwind CSS for styling
- Accessibility features (aria-labels)

How It Works

1. **Sign In Process:**
    - **U**ser enters email/password

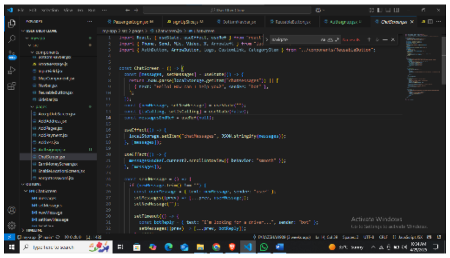
    - Clicks "Sign In"
    - If successful:
        - Stores token in localStorage
        - Shows welcome message
        - Redirects to "/HomeScreen"

2. **Sign Up Process:**
    - User fills all fields
    - Checks passwords match
    - Must accept terms
    - If successful:
        - Stores token in localStorage
        - Shows welcome message
        - Switches to Sign In tab
        - Clears form

Error Handling

- Shows clear error messages for:
    - Wrong credentials
    - Passwords not matching
    - Network issues
    - Terms not accepted
    - Server problems

Chat Screen Page:



```
import React, { useState, useEffect, useRef } from "react";
import { Phone, Send, Mic, Video, X, ArrowLeft } from "[illegible]
import { AuthButton, ArrowButton, Logo, CustomLink, CategoryItem } from "../components/ReusableButton";


const ChatScreen = () => {
  const [messages, setMessages] = useState(() => {
    return JSON.parse(localStorage.getItem("chatMessages")) || [
      { text: "Hello! How can I help you?", sender: "bot" },
    ];
  });
  const [newMessage, setNewMessage] = useState("");
  const [isCalling, setIsCalling] = useState(false);
  const messagesEndRef = useRef(null);

  useEffect(() => {
    localStorage.setItem("chatMessages", JSON.stringify(messages));
  }, [messages]);

  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
  }, [messages]);

  const sendMessage = () => {
    if (newMessage.trim() !== "") {
      const userMessage = { text: newMessage, sender: "user" };
      setMessages((prev) => [...prev, userMessage]);
      setNewMessage("");

      setTimeout(() => {
        const botReply = { text: "I'm looking for a driver...", sender: "bot" };
        setMessages((prev) => [...prev, botReply]);
```

Main Features

1. Header Section (Yellow Bar)

- Shows a back button (using the reusable ArrowButton component)
- Displays "Driver Chat" as the title
- Includes phone and video call buttons on the right side

2. Message Display Area

- Shows all chat messages in bubbles
- User messages appear on the right (yellow bubbles)
- Bot/driver messages appear on the left (white bubbles)
- Automatically scrolls to the newest message

3. Message Input Area

- Text input field for typing messages
- Send button (paper plane icon)
- Microphone button for voice messages
- Pressing Enter or clicking Send adds the message

4. Calling Overlay

- Appears when call buttons are clicked
- Shows "Calling..." text
- Has a red button to end the call

How It Works

1. **Storing Messages:**

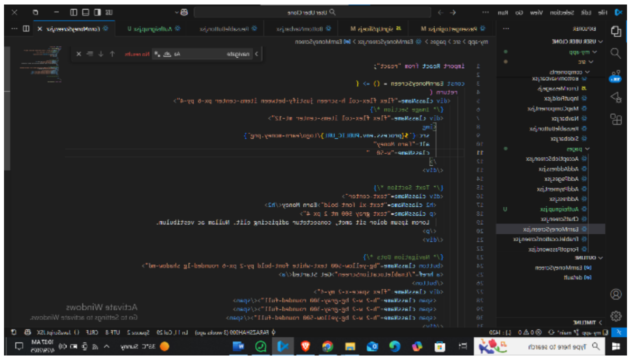
    - Messages are saved in localStorage so they don't disappear when refreshing
    - Starts with one default bot message: "Hello! How can I help you?"

2. **Sending Messages:**
    - When you type and send a message:
        - Your message appears immediately
        - After 1.5 seconds, the bot replies automatically
        - The screen scrolls down to show new messages
3. **Call Functionality:**
    - Clicking phone or video icons shows the calling overlay
    - Clicking the red X button ends the call

Technical Details

- Uses React hooks (useState, useEffect, useRef)
- Responsive design that works on mobile and desktop
- Smooth scrolling to newest messages
- Persistent chat history using localStorage
- Reusable components (ArrowButton)

Earn Money Screen:



```
import React from "react";

const EarnMoneyScreen = () => {
  return (
    <div className="flex flex-col h-screen justify-between items-center px-6 py-4">
      {/* Image Section */}
      <div className="flex flex-col items-center mt-12">
        <img
          src={`${process.env.PUBLIC_URL}/logo/earn-money.png`}
          alt="Earn Money"
          className="w-58  "
        />
      </div>

      {/* Text Section */}
      <div className="text-center">
        <h2 className="text xl font bold">Earn Money</h2>
        <p className="text gray 500 mt 2 px 4">
          Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ac vestibulum.
        </p>
      </div>

      {/* Navigation Dots */}
      <button className="bg-yellow-500 text-white font-bold py-2 px-6 rounded-lg shadow-md">
        <a href="/EnableLocationScreen">Get Started</a>
      </button>
      <div className="flex space-x-2 my-4">
        <span className="h-2 w-2 bg-gray-300 rounded-full"></span>
        <span className="h-2 w-2 bg-gray-300 rounded-full"></span>
        <span className="h-2 w-2 bg-yellow-500 rounded-full"></span>
      </div>
```

Screen Layout

1. **Image Section (Top):**
   - Displays an "earn-money.png" image from the public folder
   - Centered on the screen with margin at the top
2. **Text Section (Middle):**
   - Bold heading "Earn Money"
   - Subtitle with placeholder text (Lorem ipsum)
   - Both texts are centered
3. **Navigation Dots (Bottom):**
   - Three small dots indicating screen position
   - Third dot is yellow (active), others are gray
   - Shows this is the third screen in a sequence
4. **Get Started Button:**
   - Yellow button with white text
   - Links to "/EnableLocationScreen"
   - Has shadow effect for depth

Key Features

- Simple, clean onboarding design
- Responsive layout that works on all screen sizes

- Visual progress indicator (navigation dots)
- Clear call-to-action button
- Uses Tailwind CSS for styling

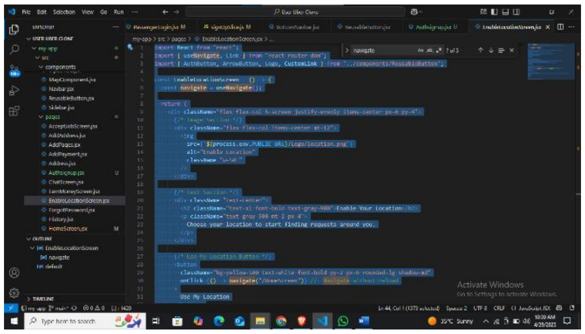
How It Works

1. Users see this screen as part of a multi-step onboarding flow
2. The yellow dot shows they're on the third step
3. Clicking "Get Started" takes them to enable location services
4. The image visually represents the earn money concept

This is a typical onboarding screen design that:

- Introduces a feature (earning money)
- Provides minimal explanatory text
- Guides users to the next step
- Shows progress in a multi-step flow

Location Permission Screen:



```
import React from "react";
import { useNavigate, Link } from "react-router-dom";
import { AuthButton, ArrowButton, Logo, CustomLink } from "../components/ReusableButton";

const EnableLocationScreen = () => {
  const navigate = useNavigate();

  return (
    <div className="flex flex-col h-screen justify-evenly items-center px-6 py-4">
      {/* Image Section */}
      <div className="flex flex-col items-center mt-12">
        <img
          src={`${process.env.PUBLIC_URL}/Logo/location.png`}
          alt="Enable Location"
          className="w-50 "
        />
      </div>

      {/* Text Section */}
      <div className="text-center">
        <h2 className="text-xl font-bold text-gray-900">Enable Your Location</h2>
        <p className="text-gray-300 mt-2 px-4">
          Choose your location to start finding requests around you.
        </p>
      </div>

      {/* Use My Location Button */}
      <button
        className="bg-yellow-500 text-white font-bold py-2 px-6 rounded-lg shadow-md"
        onClick={() => navigate("/HomeScreen")} // Navigate without reload
      >
        Use My Location
```

What This Screen Contains:

1. **Visual Element**
   - A location pin/map image (loaded from public/Logo/location.png)
   - Centered near the top of the screen
2. **Text Explanation**
   - Bold heading: "Enable Your Location"
   - Subtext explaining why: "Choose your location to start finding requests around you"
3. **Main Action Button**
   - Yellow "Use My Location" button
   - When clicked, navigates to the HomeScreen
   - In a real app, this would trigger browser/device location permission
4. **Secondary Option**
   - "Skip for now" link (gray text)
   - Also goes to HomeScreen but without enabling location
   - Uses a reusable CustomLink component

Key Features:

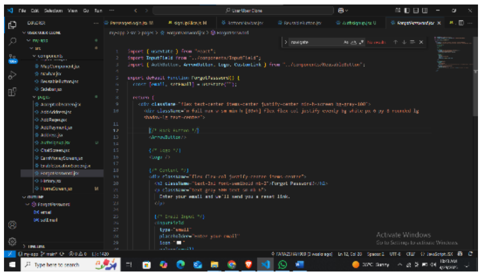
- Clean, minimalist design
- Clear call-to-action
- Alternative option for users who want to skip

- Responsive layout (works on all screen sizes)
- Uses React Router for navigation
- Reusable components (CustomLink)

How It Would Work in a Real App:

1. The "Use My Location" button would actually:
    - Request browser location permissions
    - Handle permission granted/denied cases
    - Possibly show a loading state
    - Store the location if granted
2. **The "Skip" option would:**
    - Let users proceed without location
    - Might show limited functionality

Forgot Password Screen:



```
import { useState } from "react";
import InputField from "../components/InputField";
import { AuthButton, ArrowButton, Logo, CustomLink } from "../components/ReusableButton";

export default function ForgotPassword() {
  const [email, setEmail] = useState("");

  return (
    <div className="flex text-center items-center justify-center min-h-screen bg-gray-100">
      <div className="w-full max-w-sm min-h-[80vh] flex flex-col justify-evenly bg-white px-6 py-8 rounded-lg shadow-lg text-center">

        {/* Back Button */}
        <ArrowButton/>

        {/* Logo */}
        <Logo />

        {/* Content */}
        <div className="flex flex-col justify-center items-center">
          <h2 className="text-2xl font-semibold mb-2">Forgot Password?</h2>
          <p className="text-gray-500 text-sm mb-6">
            Enter your email and we'll send you a reset link.
          </p>

          {/* Email Input */}
          <InputField
            type="email"
            placeholder="Enter your email"
            icon="✉"
```

Screen Layout

1. **Back Button (Top-left):**
   - Uses the reusable ArrowButton component
   - Allows users to navigate back to the previous screen
2. **Logo (Top-center):**
   - Displays the app logo using the reusable Logo component
   - Helps with brand recognition
3. **Main Content:**
   - Heading: "Forgot Password?" in bold
   - Subtext explaining the process: "Enter your email and we'll send you a reset link"
   - Email input field.
4. **Bottom Section:**
   - Large yellow "Sign In" button (though this should probably say "Send Reset Link")
   - Text link below saying "Remember your password? Sign In"

Key Features

- Clean, centered design with white card on gray background
- Reusable components (ArrowButton, Logo, InputField, CustomLink)
- Responsive layout that works on mobile and desktop
- Simple form handling with React state management
- Consistent styling with Tailwind CSS classes

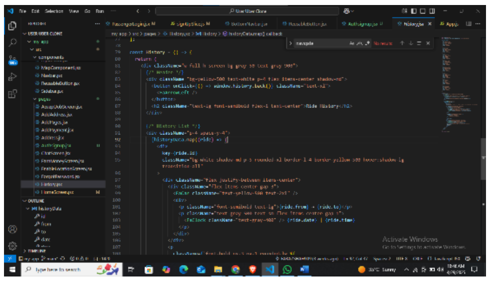
How It Should Work

1. User enters their email address
2. Clicks the "Sign In" button (which should be renamed to "Send Reset Link")
3. The app would typically:
    - Validate the email format
    - Send a password reset link to that email
    - Show a success message
    - Redirect to login or show a confirmation screen

Areas for Improvement

1. **The main button text should change from "Sign In" to "Send Reset Link"**
2. **Currently missing:**
    - Form validation
    - Actual password reset functionality
    - Loading state during submission
    - Error handling for invalid emails

Ride History Screen:



```
const History = () => {
  return (
    <div className="w-full h-screen bg-gray-50 text-gray-900">
      {/* Header */}
      <div className="bg-yellow-500 text-white p-4 flex items-center shadow-md">
        <button onClick={() => window.history.back()} className="text-xl">
          <FaArrowLeft />
        </button>
        <h2 className="text-lg font-semibold flex-1 text-center">Ride History</h2>
      </div>

      {/* History List */}
      <div className="p-4 space-y-4">
        {historyData.map((ride) => (
          <div
            key={ride.id}
            className="bg-white shadow-md p-5 rounded-xl border-l-4 border-yellow-500 hover:shadow-lg
            transition-all"
          >
            <div className="flex justify-between items-center">
              <div className="flex items-center gap-3">
                <FaCar className="text-yellow-500 text-2xl" />
                <div>
                  <p className="font-semibold text-lg">{ride.from} → {ride.to}</p>
                  <p className="text-gray-500 text-sm flex items-center gap-1">
                    <FaClock className="text-gray-400" /> {ride.date} | {ride.time}
                  </p>
                </div>
              </div>
              <p
```

Screen Layout

1. Header Section

- Yellow bar at the top with back button (left)
- "Ride History" title (center)
- Uses the FaArrowLeft icon from react-icons

2. Ride History List

- **Displays 8 sample rides from the historyData array**
- **Each ride appears as a white card with:**
  - Left yellow border
  - Car icon (FaCar)
  - Route (From → To)
  - Date and time with clock icon (FaClock)
  - Status badge (green for "Completed", red for "Cancelled")
  - Price with money icon (FaMoneyBillWave)
  - "View Details" button

Key Features

- **Interactive Elements:**
  - Back button to return to previous screen
  - Hover effects on ride cards (shadow grows)
  - "View Details" buttons for each ride
- **Visual Indicators:**

    - Color-coded status badges
    - Icons for better visual scanning
    - Consistent card design
- **Data Display:**
    - Shows origin and destination
    - Displays date and time
    - Shows price for each ride
    - Indicates ride status

Sample Data

The historyData array contains 8 example rides with:

- Unique IDs
- From/To locations
- Dates and times
- Prices
- Statuses (Completed or Cancelled)

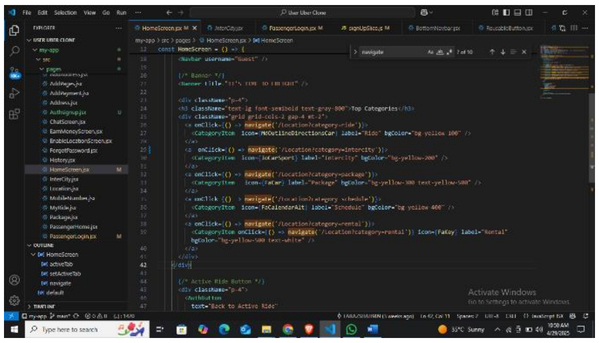
Technical Details

- Uses Tailwind CSS for styling
- React Icons (Font Awesome) for visual elements
- Responsive design that works on mobile
- Clean component structure with no state management needed (static data)

Potential Enhancements

1. Add actual API integration to fetch real user history
2. Implement pagination for long history lists
3. Add filtering options (by date, status, etc.)
4. Make "View Details" functional
5. Add loading states for data fetching

Home Screen:



```
const HomeScreen = () => {
    <Navbar username="Guest" />

    {/* Banner */}
    <Banner title="IT'S TIME TO FREIGHT" />

    <div className="p-4">
    <h3 className="text-lg font-semibold text-gray-800">Top Categories</h3>
    <div className="grid grid-cols-2 gap-4 mt-2">
      <a onClick={() => navigate('/Location?category=ride')}>
        <CategoryItem  icon={MdOutlineDirectionsCar} label="Ride" bgColor="bg-yellow-100" />
      </a>
      <a  onClick={() => navigate('/Location?category=intercity')}>
        <CategoryItem  icon={IoCarSport} label="Intercity" bgColor="bg-yellow-200" />
      </a>
      <a onClick={() => navigate('/Location?category=package')}>
        <CategoryItem   icon={FaCar} label="Package" bgColor="bg-yellow-300 text-yellow-500" />
      </a>
      <a onClick={() => navigate('/Location?category=schedule')}>
        <CategoryItem  icon={FaCalendarAlt} label="Schedule" bgColor="bg-yellow-400" />
      </a>
      <a onClick={() => navigate('/Location?category=rental')}>
        <CategoryItem onClick={() => navigate('/Location?category=rental')} icon={FaKey} label="Rental"
        bgColor="bg-yellow-500 text-white" />
      </a>
    </div>
  </div>

    {/* Active Ride Button */}
    <div className="p-4">
      <AuthButton
        text="Back to Active Ride"
```

Screen Structure

1. Header Section

- Uses the Navbar component displaying "Guest" as the username
- Shows the app's navigation bar at the top

2. Banner Section

- Displays a promotional banner with the text "IT'S TIME TO FREIGHT"
- Uses the Banner component

3. Category Grid (Main Content)

- **Title: "Top Categories"**
- **2x2 grid of service categories:**
    1. Ride (Car icon) - Navigates to '/Location?category=ride'
    2. Intercity (Sport car icon) - Navigates to '/Location?category=intercity'
    3. Package (Car icon) - Navigates to '/Location?category=package'
    4. Schedule (Calendar icon) - Navigates to '/Location?category=schedule'
    5. Rental (Key icon) - Navigates to '/Location?category=rental'
- Each category uses the reusable CategoryItem component
- Different background colors (yellow shades) for visual distinction

4. Active Ride Button

- Yellow button labeled "Back to Active Ride"
- Uses the AuthButton reusable component

- Links to '/Location' when clicked

5. Bottom Navigation

- Uses the BottomNavbar component
- Tracks active tab state with useState
- Allows switching between app sections

Key Features

- **Navigation: Uses React Router's useNavigate for page transitions**
- **Reusable Components:**
  - Navbar, Banner, BottomNavbar for layout
  - CategoryItem, AuthButton for UI elements
- **Visual Design:**
  - Consistent yellow color scheme
  - Icons for each service type
  - Responsive grid layout
- **State Management:**
  - Tracks active tab for bottom navigation
  - No complex state needed for this screen

Technical Details

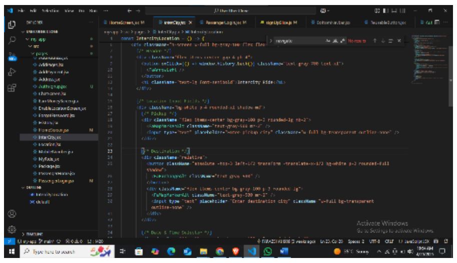
- Built with React and Tailwind CSS
- Uses react-icons for visual elements
- Clean component structure
- Mobile-friendly design (note the mb-20 for bottom navbar space)

Potential Improvements

1. Add user authentication state (replace "Guest" with actual username)
2. Implement dynamic banner content
3. Add loading states for category items

4. Include more interactive elements
5. Add recent rides/orders section

Intercity Location Selection Screen:



```
const IntercityLocation = () => {
    <div className="h-screen w-full bg-gray-100 flex flex
      {/* Header */}
      <div className="flex items-center gap-4 pb-4">
        <button onClick={() => window.history.back()} className="text-gray-700 text-xl">
          <FaArrowLeft />
        </button>
        <h1 className="text-lg font-semibold">Intercity Ride</h1>
      </div>

      {/* Location Input Fields */}
      <div className="bg-white p-4 rounded-xl shadow-md">
        {/* Pickup */}
        <div className="flex items-center bg-gray-100 p-3 rounded-lg mb-2">
          <FaMapMarkerAlt className="text-gray-500 mr-2" />
          <input type="text" placeholder="Enter pickup city" className="w-full bg-transparent outline-none" />
        </div>

        {/* Destination */}
        <div className="relative">
          <button className="absolute -top-3 left-1/2 transform -translate-x-1/2 bg-white p-2 rounded-full
          shadow">
            <FaExchangeAlt className="text-gray-600" />
          </button>
          <div className="flex items-center bg-gray-100 p-3 rounded-lg">
            <FaMapMarkerAlt className="text-gray-500 mr-2" />
            <input type="text" placeholder="Enter destination city" className="w-full bg-transparent
            outline-none" />
          </div>
        </div>

        {/* Date & Time Selector */}
```

Screen Layout

1. Header Section

- Back button (left arrow icon)
- "Intercity Ride" title
- Simple navigation back to previous screen

2. Location Input Section (White Card)

- Pickup Location Field:
  - Map marker icon
  - Text input for pickup city
  - Light gray background
- **Destination Field (with swap button):**
  - Floating swap button between fields
  - Map marker icon
  - Text input for destination city
  - Light gray background
- **Date & Time Selector:**
  - Calendar icon
  - Native datetime-local input
  - Light gray background

3. Action Buttons

- Locate on Map Button (Yellow):
  - Map marker icon

    - Full-width on left side
- **Saved Locations Button (Lighter Yellow):**
    - Bookmark icon
    - Full-width on right side

4. Recent Trips Section

- **"**Recent Trips" heading
- Placeholder message when no trips exist
- White card with subtle shadow

5. Proceed Button (Sticky Bottom)

- Large yellow button
- "Proceed" text
- Links to "PickUpRide" page
- Stays fixed at bottom of screen

Key Features

- Clean, intuitive interface for intercity bookings
- Location swapping with prominent button
- DateTime picker integrated with calendar icon
- Two main action options (map and saved locations)
- Recent trips display (empty state handled)
- Responsive design that works on mobile devices

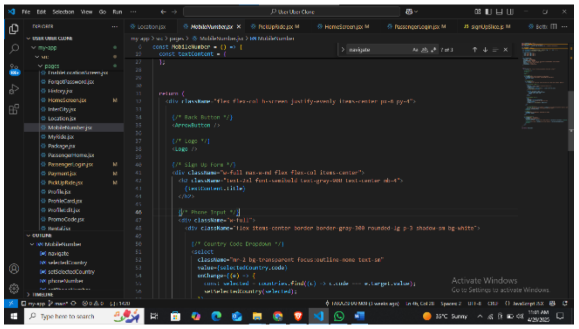
Technical Implementation

- Uses React with functional components
- React Icons (Font Awesome) for visual elements
- Tailwind CSS for styling
- Semantic HTML structure
- Simple navigation without complex state

Potential Enhancements

1. Add autocomplete for city inputs
2. Implement actual location swapping logic
3. Connect to maps API for "Locate on Map"
4. Add real recent trips data
5. Form validation before proceeding
6. Loading states for location searches

Mobile Number Verification Screen:



```
const MobileNumber = () => {
  const textContent = {
  };



  return (
    <div className="flex flex-col h-screen justify-evenly items-center px-6 py-4">

      {/* Back Button */}
      <ArrowButton />

      {/* Logo */}
      <Logo />

      {/* Sign Up Form */}
      <div className="w-full max-w-md flex flex-col items-center">
        <h2 className="text-2xl font-semibold text-gray-900 text-center mb-4">
          {textContent.title}
        </h2>

        {/* Phone Input */}
        <div className="w-full">
          <div className="flex items-center border border-gray-300 rounded-lg p-3 shadow-sm bg-white">

            {/* Country Code Dropdown */}
            <select
              className="mr-2 bg-transparent focus:outline-none text-sm"
              value={selectedCountry.code}
              onChange={(e) => {
                const selected = countries.find((c) => c.code === e.target.value);
                setSelectedCountry(selected);
```

Screen Layout

1. Header Section

- Back Button: Top-left using reusable ArrowButton component
- Logo: Centered using reusable Logo component

2. Main Content

- **Title: "Mobile Number" displayed prominently**
- **Phone Input:**
  - Country code dropdown with flags (US +1, GB +44, etc.)
  - Phone number input field
  - Combined in a single styled container

3. Action Section

- Continue Button: Yellow button linking to OTP verification
- SMS Notice: Small text about potential message charges

4. Footer

- Terms Links: "Terms of Use" and "Privacy Policy" links

Key Features

- **International Number Support:**
  - Country code selector with flags
  - Pre-populated with common codes (US, UK, Pakistan, India)
- **Reusable Components:**
  - ArrowButton, Logo, and CustomLink for consistency

    - All text content centralized in textContent object
- **User Experience:**
    - Clean, focused interface
    - Clear progression to next step (OTP verification)
    - Legal disclosures as required

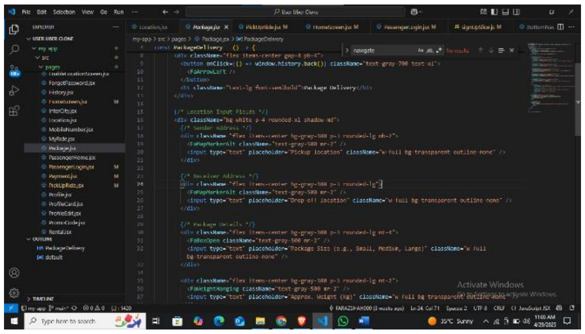
Technical Implementation

- Uses React hooks (useState, useNavigate)
- React Router for navigation
- Tailwind CSS for styling
- Accessible form elements
- Responsive design

Potential Enhancements

1. Phone number formatting/validation
2. Auto-detect country code by IP
3. Loading state during submission
4. Integration with phone verification services
5. Remember country selection preference

Package Delivery Screen:



```
const PackageDelivery = () => {
      <div className="flex items-center gap-4 pb-4">
        <button onClick={() => window.history.back()} className="text-gray-700 text-xl">
          <FaArrowLeft />
        </button>
        <h1 className="text-lg font-semibold">Package Delivery</h1>
      </div>

      {/* Location Input Fields */}
      <div className="bg-white p-4 rounded-xl shadow-md">
        {/* Sender Address */}
        <div className="flex items-center bg-gray-100 p-3 rounded-lg mb-2">
          <FaMapMarkerAlt className="text-gray-500 mr-2" />
          <input type="text" placeholder="Pickup location" className="w-full bg-transparent outline-none" />
        </div>

        {/* Receiver Address */}
        <div className="flex items-center bg-gray-100 p-3 rounded-lg">
          <FaMapMarkerAlt className="text-gray-500 mr-2" />
          <input type="text" placeholder="Drop-off location" className="w-full bg-transparent outline-none" />
        </div>

        {/* Package Details */}
        <div className="flex items-center bg-gray-100 p-3 rounded-lg mt-4">
          <FaBoxOpen className="text-gray-500 mr-2" />
          <input type="text" placeholder="Package Size (e.g., Small, Medium, Large)" className="w-full
          bg-transparent outline-none" />
        </div>

        <div className="flex items-center bg-gray-100 p-3 rounded-lg mt-2">
          <FaWeightHanging className="text-gray-500 mr-2" />
          <input type="text" placeholder="Approx. Weight (kg)" className="w-full bg-transparent outline-none" />
```

Screen Layout

1. Header Section

- Back Button: Top-left arrow icon
- Title: "Package Delivery" heading

2. Package Details Form (White Card)

- **Pickup Location:**
    - Map marker icon
    - Text input for sender's address
- **Drop-off Location:**
    - Map marker icon
    - Text input for receiver's address
- **Package Specifications:**
    - Box icon for package size (Small/Medium/Large)
    - Weight icon for approximate weight in kg
    - Calendar icon for delivery date/time selection

3. Action Buttons

- **Locate on Map Button (Yellow):**
    - Map marker icon
    - Full-width on left side
- **Saved Locations Button (Lighter Yellow):**
    - Bookmark icon

    - Full-width on right side

4. Recent Deliveries Section

- "Recent Deliveries" heading
- Placeholder message when no history exists
- White card with subtle shadow

5. Proceed Button (Sticky Bottom)

- Large yellow button
- "Proceed" text
- Links to "PickUpRide" page
- Fixed at bottom of screen

Key Features

- Clean Form Layout: Organized input fields with relevant icons
- Package-Specific Fields: Size and weight inputs
- Location Tools: Map and saved locations access
- Empty State Handling: For delivery history
- Responsive Design: Works on mobile devices

Technical Implementation

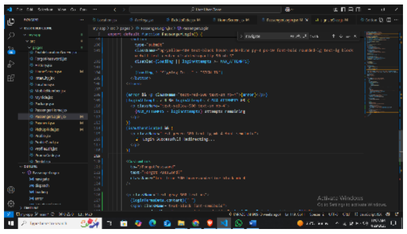
- Uses React functional components
- React Icons (Font Awesome) for visual elements
- Tailwind CSS for styling
- Simple navigation without complex state
- Semantic HTML structure

Potential Enhancements

1. Add autocomplete for location inputs
2. Implement package size/weight validation
3. Connect to maps API for location selection

4. Show actual delivery history when available
5. Add form validation before proceeding
6. Loading states for location searches

Passenger Login Screen:



```
export default function PassengerLogin() {
      type="submit"
      className="bg-yellow-500 text-black hover:underline py-4 px-16 font-bold rounded-lg text-lg block [illegible]
      disabled={loading || loginAttempts >= MAX_ATTEMPTS}
    >
      {loading ? "Signing In..." : "[illegible]"}
    </button>
  [illegible]

  {error && <p className="text-red-500 text-sm mt-2">{error}</p>}
  {loginAttempts > 0 && loginAttempts < MAX_ATTEMPTS && (
    <p className="text-yellow-500 text-sm mt-2">
      {MAX_ATTEMPTS - loginAttempts} attempts remaining
    </p>
  )}
  {isAuthenticated && (
    [illegible]
      Login Successful! Redirecting...
    </p>
  )}

  [illegible]
    to="/forgotPassword"
    text="Forgot Password?"
    [illegible]
  />

  [illegible]
    {loginFormsData.content}{" "}
    <span className="text-black font-semibold">
```

Core Functionality

1. Authentication Flow

- Redux Integration: Uses useDispatch and useSelector to manage auth state
- Login Attempt Tracking: Limits to 5 attempts (stored in localStorage)
- Success Handling: Redirects to mobile verification after successful login
- Error Handling: Displays validation and server errors

2. Security Features

- Input Sanitization: Removes special characters (<>&") from inputs
- Password Requirements: Minimum 8 characters
- Username Requirements: Minimum 3 characters
- Attempt Limiting: Prevents brute force attacks

UI Components

1. Form Elements

- Username Field: With person icon
- Password Field: With lock icon (hidden by default)
- Sign In Button: Yellow, disabled during loading/after max attempts
- Forgot Password Link: Directs to password recovery

2. Status Indicators

- Loading State: "Signing In..." text during submission
- Error Messages: Red text for validation/API errors
- Success Message: Green confirmation with redirect
- Attempt Counter: Shows remaining login attempts

3. Navigation

- Back Button: Top-left using reusable ArrowButton
- Logo: Centered branding
- Sign Up Link: For new users

Technical Implementation

1. State Management

- Local state for form data and errors
- Redux for authentication status
- localStorage for tracking login attempts

2. Validation

- Client-side validation before submission
- Real-time error clearing during typing

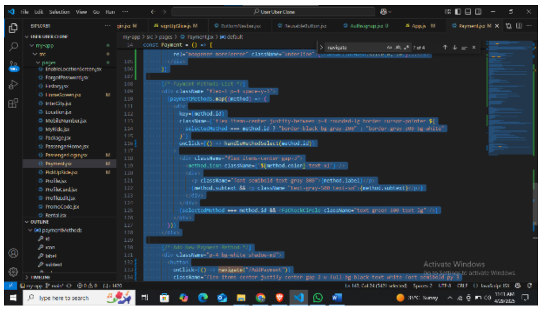
3. Routing

- React Router for navigation
- Automatic redirect on success

Potential Enhancements

1. Add CAPTCHA after failed attempts
2. Implement password strength meter
3. Add "Remember Me" functionality
4. Social login integration
5. Two-factor authentication option

Payment Methods Screen:



```
const Payment = () => {
          rel="noopener noreferrer" className="underline">{transactionHash.slice(0, 10)}...</a>
        </div>
      )}

      {/* Payment Methods List */}
      <div className="flex-1 p-4 space-y-3">
        {paymentMethods.map((method) => (
          <div
            key={method.id}
            className={`flex items-center justify-between p-4 rounded-lg border cursor-pointer ${
              selectedMethod === method.id ? "border-black bg-gray-200" : "border-gray-300 bg-white"
            }`}
            onClick={() => handleMethodSelect(method.id)}
          >
            <div className="flex items-center gap-3">
              <method.icon className={`${method.color} text-xl`} />
              <div>
                <p className="font-semibold text-gray-800">{method.label}</p>
                {method.subtext && <p className="text-gray-500 text-sm">{method.subtext}</p>}
              </div>
            </div>
            {selectedMethod === method.id && <FaCheckCircle className="text-green-500 text-lg" />}
          </div>
        ))}
      </div>

      {/* Add New Payment Method */}
      <div className="p-4 bg-white shadow-md">
        <button
          onClick={() => navigate("/AddPayment")}
          className="flex items-center justify-center gap-2 w-full bg-black text-white font-semibold py-3
```

Core Features

1. Payment Method Selection

- Credit Card: Default Visa card option
- PayPal: Digital wallet integration
- Cash: In-person payment option
- Ethereum: Crypto payments via MetaMask

2. Web3 Integration

- Auto-connects to MetaMask on load
- Handles Ethereum transactions
- Displays transaction status/hash
- Manages account changes

3. UI Components

- Clean header with back navigation
- Visual payment method cards
- Status indicators (success/error)
- "Add Payment Method" button

Security Features

1. MetaMask Validation: Checks for extension installation
2. Account Monitoring: Listens for account changes
3. Transaction Feedback: Provides visible transaction hash
4. **Error Handling:** Catches and displays payment failures

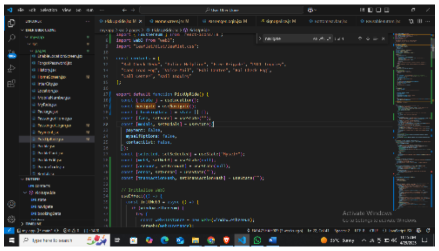
User Experience

- **Visual Feedback:**
  - Checkmarks for selected methods
  - Color-coded payment icons
  - Clear error/success messages
- **Navigation:**
  - Back button maintains flow
  - "Add Payment" links to extension
- **Mobile-Friendly:**
  - Responsive design
  - Touch-friendly elements

Enhancement Opportunities

1. Add more crypto options (Bitcoin, etc.)
2. Implement saved card management
3. Add payment amount display
4. Include transaction history
5. Add loading states
6. Implement payment confirmation

Ride Booking Screen:



```
import Web3 from "web3";
import "leaflet/dist/leaflet.css";

const contacts = [
  "Bal Check Desk", "Police Helpline", "Fire Brigade", "[illegible] Inquiry",
  "Card load Agt", "Voice Mail", "Help Center", "Bal Check Agt",
  "Call center", "Bal inquiry"
];

export default function PickUpRide() {
  const { state } = useLocation();
  const navigate = useNavigate();
  const { bookingData } = state || {};
  const [fare, setFare] = useState("");
  const [modals, setModals] = useState({
    payment: false,
    myselfOptions: false,
    contactList: false,
  });
  const [selected, setSelected] = useState("Myself");
  const [web3, setWeb3] = useState(null);
  const [account, setAccount] = useState(null);
  const [error, setError] = useState("");
  const [transactionHash, setTransactionHash] = useState("");

  // Initialize Web3
  useEffect(() => {
    const initWeb3 = async () => {
      if (window.ethereum) {
        try {
          const web3Instance = new Web3(window.ethereum);
          setWeb3(web3Instance);
```

Core Features

1. Interactive Map Interface

- Displays route between pickup and dropoff points
- Uses React-Leaflet for map rendering
- Shows markers for both locations
- Draws polyline between points

2. Booking Form

- Vehicle type selection
- Dynamic fare input with validation
- Recommended price indicator
- Payment method selection
- Passenger selection options

3. Payment Integration

- Credit Card, PayPal, Google Pay options
- Ethereum cryptocurrency payments via MetaMask
- Transaction status tracking
- Error handling for failed payments

4. User Experience

- Smooth modal animations with Framer Motion
- Contact list for alternate passengers
- Clear status messages
- Mobile-responsive design

Component Structure

1. Main Components

- PickUpRide: Primary booking interface
- MyselfOptions: Passenger selection modal
- ContactList: Saved contacts list
- PaymentModal: Payment method selector

2. UI Elements

- Interactive map with route visualization
- Form inputs with validation
- Modal dialogs with smooth animations
- Status indicators for transactions

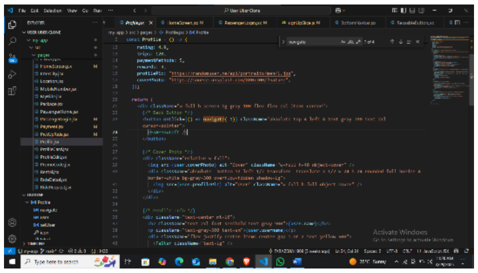
Security Features

1. MetaMask Integration: Secure wallet connection
2. Transaction Validation: On-chain verification
3. Input Sanitization: Fare amount validation
4. Error Handling: Clear user feedback

Enhancement Opportunities

1. Add real-time fare calculation
2. Implement ride tracking
3. Add driver rating system
4. Include promotional codes
5. Add scheduled booking options
6. Implement multi-stop routes

Profile Screen:



```
const Profile = () => {
    rating: 4.9,
    trips: 120,
    paymentMethods: 5,
    rewards: 4,
    profilePic: "https://randomuser.me/api/portraits/men/1.jpg",
    coverPhoto: "https://source.unsplash.com/800x300/?nature",
  });

  return (
    <div className="w-full h-screen bg-gray-100 flex flex-col items-center">
      {/* Back Button */}
      <button onClick={() => navigate(-1)} className="absolute top-4 left-4 text-gray-700 text-2xl
      cursor-pointer">
        <FaArrowLeft />
      </button>

      {/* Cover Photo */}
      <div className="relative w-full">
        <img src={user.coverPhoto} alt="Cover" className="w-full h-40 object-cover" />
        <div className="absolute -bottom-12 left-1/2 transform -translate-x-1/2 w-24 h-24 rounded-full border-4
        border-white bg-gray-300 overflow-hidden shadow-lg">
          <img src={user.profilePic} alt="User" className="w-full h-full object-cover" />
        </div>
      </div>

      {/* Profile Info */}
      <div className="text-center mt-16">
        <h2 className="text-2xl font-semibold text-gray-900">{user.name}</h2>
        <p className="text-gray-500 text-sm">{user.username}</p>
        <div className="flex justify-center items-center gap-1 mt-2 text-yellow-500">
          <FaStar className="text-lg" />
```

Screen Layout

1. Header Section

- Back Button: Top-left arrow for navigation
- Cover Photo: Full-width header image
- Profile Picture: Centered circular avatar overlapping cover photo

2. User Information

- Name: Large, bold display
- Username: Smaller, gray text
- Rating: Star icon with numerical value
- Trip Count: Number of completed rides

3. Stats Cards

- Trips: Total rides taken
- Payment Methods: Number of saved cards
- Rewards: Vouchers available
- Each stat has an icon and displays in a horizontal row

4. Quick Actions

- Edit Profile: Links to profile editing
- Settings: Links to app settings
- Both as tappable cards with icons

5. Bottom Navigation

- Fixed position tab bar

- Trips: Links to ride history
- Wallet: Links to payment methods
- Rewards: Links to vouchers

Key Features

- Visual Hierarchy: Important information emphasized
- Consistent Styling: Yellow accent color throughout
- Responsive Design: Works on mobile devices
- Interactive Elements: All buttons and links are tappable
- Data Display: Shows user statistics clearly

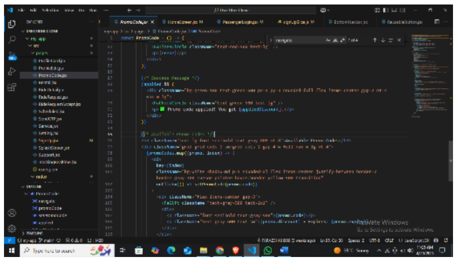
Visual Elements

- Icons: Using Font Awesome (react-icons/fa)
- Shadows: Subtle shadows for depth
- Rounded Corners: Modern UI elements
- Fixed Bottom Nav: Easy access to key sections

Potential Enhancements

1. Add real API integration
2. Implement photo upload functionality
3. Add dark mode support
4. Include recent trip preview
5. Add notification badge
6. Implement profile completion meter

Promo Code Screen:



```
const PromoCode = () => {
        <AiFillCloseCircle className="text-red-500 text-lg" />
        <p>{error}</p>
      </div>
    )}

    {/* Success Message */}
    {applied && (
      <div className="bg-green-100 text-green-800 px-4 py-3 rounded-full flex items-center gap-2 mt-4
      max-w-lg">
        <FaCheckCircle className="text-green-500 text-lg" />
        <p>✅ Promo code applied! You got {appliedDiscount}.</p>
      </div>
    )}

    {/* Available Promo Codes */}
    <h3 className="text-lg font-semibold text-gray-800 mt-8">Available Promo Codes</h3>
    <div className="grid grid-cols-1 sm:grid-cols-2 gap-4 w-full max-w-lg mt-4">
      {promoCodes.map((promo, index) => (
        <div
          key={index}
          className="bg-white shadow-md p-4 rounded-xl flex items-center justify-between border-2
          border-gray-300 cursor-pointer hover:border-yellow-500 transition"
          onClick={() => setPromoCode(promo.code)}
        >
          <div className="flex items-center gap-3">
            <FaGift className="text-gray-500 text-2xl" />
            <div>
              <p className="font-semibold text-gray-800">{promo.code}</p>
              <p className="text-gray-600 text-sm">{promo.discount} • Expires: [illegible]
            </div>
          </div>
```

Screen Layout

1. Header Section

- Back Button: Top-left navigation arrow
- Title: "Apply Promo Code" centered
- Black background with white text for contrast

2. Promo Code Input

- Input Field: Circular design with tag icon
- Apply Button: Yellow button that activates when text entered
- Visual Feedback:
  - Error messages (red)
  - Success confirmation (green)

3. Available Promo Codes

- Grid Layout: 1 column on mobile, 2 on larger screens
- Code Cards: Each shows:
  - Gift icon
  - Code name
  - Discount amount
  - Expiration date
  - "Use" button

Key Features

- Input Validation:
  - Checks for empty input

    - Verifies code against predefined list
    - Provides clear error messages
- **Persistence:**
    - Saves applied codes to localStorage
    - Retrieves on component mount
- **Interactive Elements:**
    - Click promo cards to auto-fill input
    - Hover states for buttons
    - Disabled state for empty input
- **Visual Design:**
    - Consistent yellow accent color
    - Card shadows for depth
    - Responsive grid layout

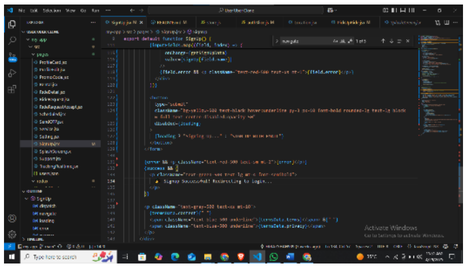
User Experience

- Instant Feedback: Clear visual indicators for success/error
- Quick Selection: Click promo cards to auto-fill
- Accessible: Color contrast meets WCAG standards
- Mobile-Friendly: Responsive design works on all devices

Enhancement Opportunities

1. Add API integration for real promo codes
2. Implement countdown timer for expiring codes
3. Add copy-to-clipboard functionality
4. Include QR code scanner option
5. Add terms & conditions modal
6. Implement promo code stacking rules.

Sign-up:



```
export default function SignUp() {
          {inputFields.map((field, index) => (
              onChange={getSignupData}
              value={signUp[field.name]}
            />
            {field.error && <p className="text-red-500 text-xs mt-1">{field.error}</p>}
          </div>
        ))}

        <button
          type="submit"
          className="bg-yellow-500 text-black hover:underline py-3 px-10 font-bold rounded-lg text-lg block
          w-full text-center disabled:opacity-50"
          disabled={loading}
        >
          {loading ? "Signing up..." : "Sign up with email"}
        </button>
      </form>

      {error && <p className="text-red-500 text-sm mt-2">{error}</p>}
      {success && (
        <p className="text-green-500 text-lg mt-4 font-semibold">
          🎉 Signup Successful! Redirecting to login...
        </p>
      )}

      <p className="text-gray-500 text-xs mt-10">
        {termsData.content}{" "}
        <span className="text-blue-500 underline">{termsData.terms}</span> &{" "}
        <span className="text-blue-500 underline">{termsData.privacy}</span>
      </p>
    </div>
```

What This Code Does

1. **Imports Needed Tools:** At the top, we bring in all the tools we need from React and other libraries to build our sign-up form.
2. **Main SignUp Function**: This is the main component that creates the sign-up page.
3. **State Management:**
   - signUp state keeps track of what the user types (username, email, passwords)
   - formErrors state stores any errors in the form (like if passwords don't match)
4. **Form Validation:**
   - Checks if the username is at least 3 characters
   - Makes sure the email looks like a real email (with @ and .)
   - Requires passwords to be at least 8 characters with special characters and numbers
   - Verifies both password fields match
5. **Handling User Input:**
   - When users type, it saves what they type and removes any potentially harmful characters
   - Clears errors when the user starts fixing them
6. **Submitting the Form:**
   - When the form is submitted, it checks for errors first
   - If no errors, it sends the sign-up data to the server
7. **Success Handling:**

    - If sign-up is successful, it shows a success message
    - After 2 seconds, it automatically sends the user to the login page

8. **The Visual Part:**
    - Shows input fields for username, email, and passwords
    - Displays error messages if something is wrong
    - Shows a loading message when waiting for the server
    - Includes a link to terms and privacy policy

Key Features

- Security: Cleans user input to prevent harmful code
- User-Friendly: Shows clear error messages to help users fix mistakes
- Responsive: Works on different screen sizes
- Visual Feedback: Shows when the form is loading or successfully submitted