

MAJOR PROJECT REPORT ON

Instant Dispatch

“Delivery Management System”

UNDERTAKEN AT

SANT SINGAJI INSTITUTE OF SCIENCE AND MANAGEMENT

BY

Nikita Yadav

SUBMITTED TO



VIKRAM UNIVERSITY UJJAIN (M.P.)

Partial fulfilment of the Requirements for the Award of the Degree Of

Bachelor of Computer

Application (B.C.A.)

Guide Name

Ms. Tanushree Soni

Sant Singaji Institute of Science and Management Sandalpur [Dewas]

Certificate

Reference No:

Date:

Project Completion Certificate

This is to certify that **Ms. Nikita Yadav** student of **BCA (Final Year)** of **Sant Singaji Institute of Science and Management**, has successfully completed the project work entitled "**Instant Dispatch - Delivery Management System**" under the guidance of **Ms. Tanushree Soni** is a bona fide piece of work carried out at **Sant Singaji Institute of Science and Management Sandalpur**.

The project entitled "**Instant Dispatch - Delivery Management System**" developed by **Nikita Yadav** in the **SSISM** and she has put at least 200 hours of laboratory work during the tenure of the project with the guide to complete this project. All the **prescribed certificates** are attached after the completion of all the formalities of the project work as pre-schedule, including internal examination.

Place

Signature of Principal

Date

Signature of HOD

Certificate of Attendance

Reference No:

Date:

This is to certify that **Ms. Nikita Yadav**, Student of BCA (**Final Year**) of **Sant Singaji Institute of Science and Management**, has successfully completed the project work entitled "**Instant Dispatch - Delivery Management System**" and he has put at least 200 hours of laboratory work during the stipulated period of the project at **Sant Singaji Institute of Science and Management, Sandalpur**.

Place

Signature of Guide

Date

Signature of Principal

Declaration

I'm **Nikita Yadav** of **Sant Singaji Institute of Science & Management** declare that the project report submitted by me under the guidance of **Ms. Tanushree Soni** is a bona fide work for the partial fulfilment of the requirement of the **BCA Final Year project work**. I have incorporated all the suggestions provided by my guide time to time.

I further declare that to the best of my knowledge this project contains my original work & does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university/Deemed university/Institute etc. Without proper citation and I shall be fully responsible for any plagiarism found at any stage.

Name & Signature of the Guide

Name & Signature of the Student

Project Approval Certificate

Reference No:

Date:

This to certify that **Ms. Nikita Yadav**, a student of **BCA (Final Year)** of **Sant Singaji Institute of Science & Management**, has successfully completed the project entitled "**Instant Dispatch - Delivery Management System**" under my guidance. I have regularly assessed the progress of the work and suggested the correction whenever required. The student has incorporated all the suggestions provided by me in this project. This project is bona fide piece of work of the standard of BCA project work carried out by the student under my supervision. Internal examination has been completed in my presence and student's performance was satisfactory and hence this project is approved for the submission and valuation thereof.

Signature of Guide

Place:

Signature of Principal

Date:

Seal of the Institute

Table of Contents

1. Introduction

- 1.1. Project Overview
- 1.2. Problems and Solutions of the Project

2. Feasibility Study

- 2.1. Technical Feasibility
- 2.2. Operational Feasibility
- 2.3. Economic Feasibility

3. System Analysis

- 3.1. Home Module
- 3.2. Audio Record Module

4. System Requirement Analysis

- 4.1. Hardware Requirement
- 4.2. Software Requirement

5. System Design

- 5.1. Module Design
- 5.2. Data Flow Diagram
- 5.3. Entity Relationship Diagram

6. System Testing and Implementation

- 6.1. Introduction
- 6.2. Software Testing
- 6.3. Unit Testing
- 6.4. Conditional Testing
- 6.5. Data Flow Testing

7. Output Screens and Code

8. Conclusion

- 8.1. Limitations
- 8.2. Future Enhancements

9. Bibliograph

Introduction

✓ Project Overview

At "**Instant Dispatch**", we understand the everyday challenges of finding reliable and fast delivery services for your parcels and packages. That's why we created this platform – to put an end to the hassle and save you precious time.

Imagine the countless hours spent searching for a reliable delivery service or a trustworthy rider. With Instant Dispatch, those challenges become a thing of the past. We're here to simplify your delivery experience, making it effortless to find the perfect delivery solution for your needs.

No more wasting time scrolling through endless options or dealing with uncertainty. Instant Dispatch is your solution to quick, reliable, and efficient delivery services. We're dedicated to solving the problems that have been a roadblock in your delivery journey, ensuring that your time is spent on what matters most – getting your packages delivered safely and on time.

Let "**Instant Dispatch**" be the key to unlocking a stress-free and time-saving approach to package delivery. Your convenience is our top priority, and we're here to make every delivery a breeze. Welcome to a world where your delivery worries are a thing of the past!

User Phase

1. Customer Phase

Discover your ideal delivery service effortlessly with Instant Dispatch's Customer Module. Browse through a diverse range of delivery options, review rider ratings to make informed choices, and confidently book the perfect delivery service for your needs. Your delivery starts with a few simple taps, ensuring a seamless and stress-free experience.

2. Rider Phase

Embark on a flexible and rewarding journey as a rider with Instant Dispatch. Register effortlessly, set your availability preferences, and accept delivery bookings that align with your schedule. Connect with customers who value your services in this user-friendly delivery network, making each delivery a positive and fulfilling experience.

3. Admin Phase

The Admin Module within Instant Dispatch is the backbone of our platform, ensuring a secure and reliable environment. Administrators wield the authority to verify and accept rider registrations, guaranteeing the quality and trustworthiness of our community. With the power to swiftly manage users, orders, and riders, admins maintain order and safety, responding promptly to reported issues or violations. The Admin Module is our guardian, upholding standards, verifying credentials, and fostering a secure space for a trustworthy and enjoyable delivery experience.

1.2 Problems and solutions of the project

1.2.1 Existing System

Individuals often face considerable challenges when searching for reliable delivery services to fulfill their shipping needs. The existing options lack a streamlined and trustworthy platform, making it time-consuming and sometimes frustrating to find a suitable delivery service. Users may encounter difficulties in assessing the reliability of both delivery services and riders, leading to concerns about the overall safety and quality of their delivery experiences. This lack of a centralized and efficient solution can result in wasted time, inconvenience, and a diminished sense of confidence when sending packages.

1.2.2 Proposed System

"Instant Dispatch" addresses this prevalent issue by providing a comprehensive platform that streamlines the process of finding the ideal delivery service and rider. By offering a user-friendly interface, detailed profiles, and a transparent rating system, Instant Dispatch ensures that users can confidently select a reliable delivery service that meets their specific requirements. The platform's commitment to safety, quality, and efficiency aims to eliminate the uncertainties associated with traditional methods of finding delivery services, offering a one-stop solution that saves time and provides peace of mind for every delivery.

Feasibility Report

Instant Dispatch's initial investigation evaluates its overall feasibility, emphasizing Technical, Operational, and Economic viability. The study assesses the potential of introducing new modules and debugging the existing system. Key aspects covered include: of the preliminary investigation:

- ✓ Technical Feasibility
- ✓ Operation Feasibility
- ✓ Economic Feasibility

2.1. Technical Feasibility

The technical issue usually raised during the feasibility stage includes:

- Does the necessary technology exist to do what is suggested?
- Can the system be upgraded if developed?
- Are there technical guarantees of accuracy, reliability, ease of access and data security?

Technical feasibility is paramount for Instant Dispatch, requiring a meticulous evaluation of its platform's robustness. This involves assessing scalability, compatibility across diverse devices, and seamless integration with various operating systems. Emphasis is placed on:

- Maintaining a stable server infrastructure
- Efficient data management
- Real-time tracking capabilities using Socket.IO
- Ability to adapt swiftly to emerging technologies

2.2. Operational Feasibility

Operational feasibility is at the forefront of Instant Dispatch's considerations, examining how seamlessly the platform integrates into existing processes. The focus is on:

- Minimizing disruptions during the transition
- Ensuring a smooth adoption process for users
- Assessing user adoption and training needs
- Understanding potential impacts on day-to-day operations
and implemented?

2.3. Economic Feasibility

Economic feasibility forms the bedrock of Instant Dispatch's strategic planning, involving a thorough cost-benefit analysis. This comprehensive examination weighs:

- Development and operational expenses
- Potential revenue streams
- User acquisition costs
- Pricing models
- Market demand analysis

System Analysis

Software Requirement Specification Overview

The need for "**Instant Dispatch**" arises from the prevalent challenges individuals face in finding reliable delivery services. With the increasing demand for convenient and trustworthy delivery solutions, there's a clear gap in the market for a platform that seamlessly connects users with reliable riders. **Instant Dispatch** addresses this need by providing a user-friendly interface where users can easily browse, book, and track deliveries, ensuring a hassle-free delivery experience.

○ **Home module**

Through the Home module, users can access various functionalities including:

- Browse available delivery services
- View rider profiles and ratings
- Access booking history
- Track active deliveries in real-time

○ **Customer Module**

In the Customer Module of Instant Dispatch, users can perform a variety of tasks:

- Browse through available riders and delivery options
- View detailed profiles of riders including ratings and reviews
- Specify delivery requirements (pickup address, delivery address, package details)
- Book delivery services
- Track deliveries in real-time
- View order history
- Receive notifications about delivery status

- **Customer Functionalities:**

- Sign In or Sign Up
- Book delivery services
- Track orders in real-time
- View order history
- Rate and review riders
- Manage profile

- **Rider Module**

In the Rider Module of Instant Dispatch, riders can:

- Manage their availability
- View booking requests
- Accept or reject delivery requests
- Update their profiles
- Navigate to pickup and delivery locations
- Update delivery status
- View earnings and performance metrics

- **Rider Functionalities:**

- Register and create profile
- Set availability status
- Accept/reject delivery requests
- Update delivery status
- View earnings
- Access navigation tools

- **Admin Module**

In the Admin Module of Instant Dispatch, administrators have comprehensive control:

- Verify and approve rider registrations
- Monitor platform performance and analytics
- Manage users, riders, and orders
- Block or unblock users as needed
- View all orders and their status
- Generate reports and analytics

- **Admin Functionalities:**

- Admin login
- Manage users (block/unblock)
- Manage riders (approve/reject, activate/deactivate)
- View all orders
- Monitor platform analytics
- Handle reported issues

System Requirement Analysis

4.1 Hardware Requirements

- Android mobile device, Computer, Laptop is needed.
- 3 GB RAM and above.
- 32 ROM and above.

4.2 Software Requirements

- Web Browser is needed.
- Operating System Compatibility.
- Internet Connection.

System Design

5.1 Module Design

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for **Instant Dispatch** or any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement has been specified and analyzed, system design is the first of the three technical activities - design, code and test that is required to build and verify software.

The significance of quality is paramount in Instant Dispatch. Design serves as the cornerstone where quality is cultivated within software development. It offers us visualizations of the platform, enabling assessment of Instant Dispatch's integrity. Design is indispensable for accurately translating user perspectives into the final product. It lays the groundwork for all subsequent software engineering processes. Without a robust design, there is a risk of constructing an unreliable system, challenging to evaluate until the final stages of development.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities – architectural design, data structure design, interface design and procedural design.

5.2 Data Flow Diagrams

Data Flow Diagrams (DFDs) are graphical tools utilized to illustrate and analyze the flow of data within the Instant Dispatch system. They serve as the central tool and foundation upon which other system components are constructed. The transformation of data from input to output, passing through various processes, is logically described independently of the system's physical components, forming the basis of logical data flow diagrams. Physical data flow diagrams depict the actual implementation and movement of data between users, departments, and workstations.

A comprehensive system description consists of a series of data flow diagrams, developed using familiar notations such as Yourdon, Gane, and Sarson. Each component in a DFD is labeled with a descriptive name, with processes further identified by a number for identification purposes within the Instant Dispatch platform. DFDs are developed across multiple levels, with each process in lower-level diagrams broken down into more detailed DFDs in subsequent levels. The top-level diagram, known as the context diagram, features a single process that is crucial for understanding the current system. Processes in the context level diagram are expanded into additional processes in the first-level DFD.

DFD Symbols

In the DFD, there are four symbols

1. A square defines a source (originator) or destination of system data.
2. An arrow identifies data flow. It is the pipeline through which the

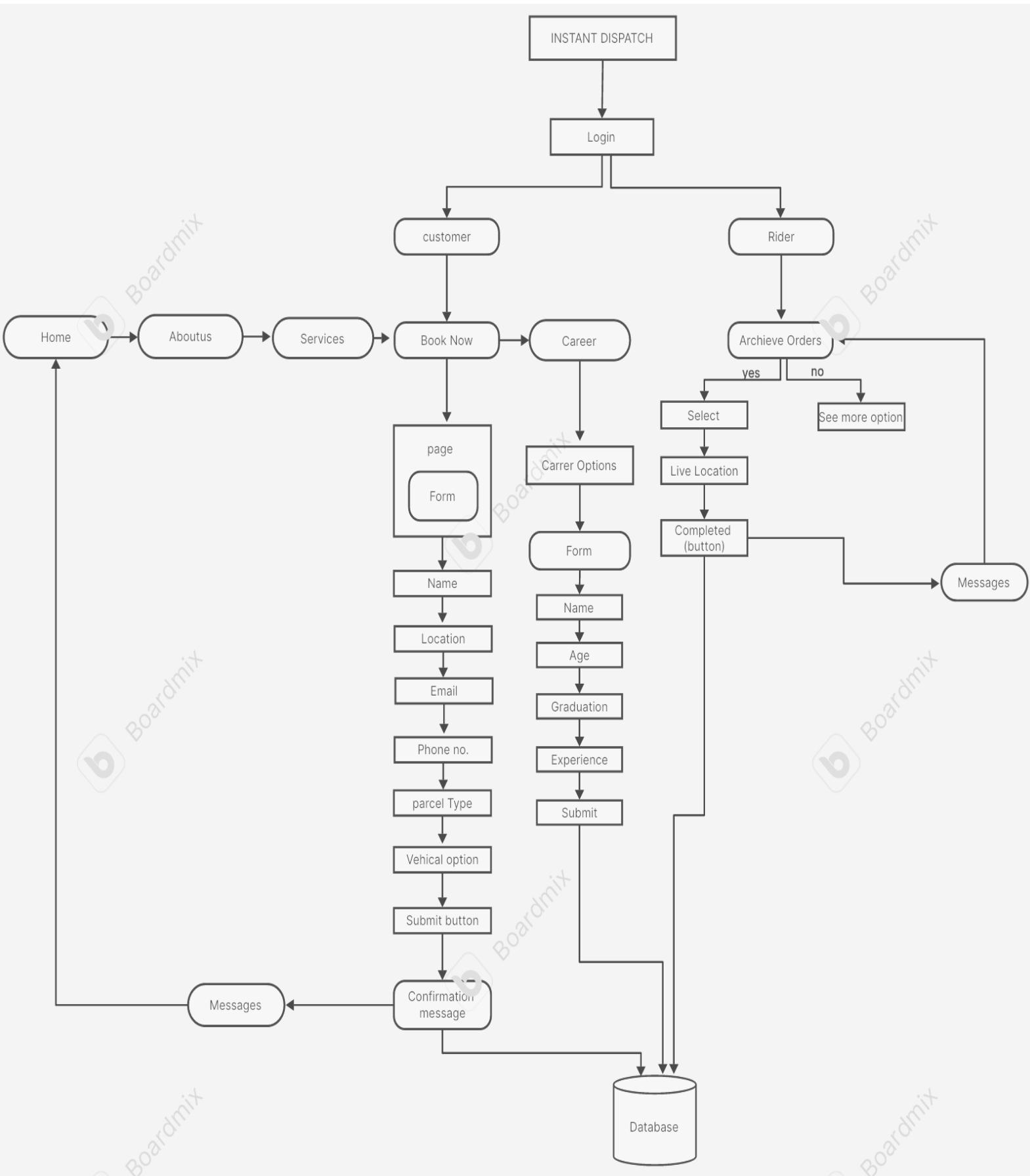
information flows.

3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.

Data Flow

1. A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The latter is usually indicated however by two separate arrows since these happen at different type.
2. A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.
3. A data flow cannot go directly back to the same process it leads. There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process.
4. A data flow to a data store means update (delete or change).
5. A data flow from a data store means retrieve or use.

A data flow has a noun phrase label more than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.



MongoDB Database

MongoDB is a modern, flexible, and open-source NoSQL database system. MongoDB stores data in flexible, JSON-like documents, making it ideal for handling unstructured data. MongoDB's flexible schema and dynamic querying capabilities allow for agile development and scalability. MongoDB is widely adopted for its high performance, scalability, and ease of use in modern application development.

Foreign Key

When a field in one table matches the primary key of another table, it is referred to as a foreign key. A foreign key is a field or a group of fields in one table whose values match those of the primary key of another table.

Aggregate queries

MongoDB's aggregate function allows you to perform complex operations on your data, such as grouping, filtering, and projecting, in a single operation. This enables you to extract valuable insights from your data efficiently and effectively.

1. Grouping documents based on specified criteria.
2. Aggregating data across multiple collections.
3. Filtering documents based on specific conditions.
4. Projecting fields to include or exclude in the output.
5. Sorting results based on one or more fields.
6. Performing mathematical and statistical calculations on the data.

System Testing and Implementation

6.1. Introduction

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for Instant Dispatch software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both large and small-scale systems.

6.2. Software Testing

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints, and validation criteria for **Instant Dispatch** software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for **Instant Dispatch** software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Taking another turn outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested

6.3. Unit Testing

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing we have is white box oriented and some modules the steps are conducted in parallel.

1. White Box Testing

This type of testing ensures that:

- All independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false side
- All loops are executed at their boundaries and within their operational bounds

2. Basic Path Testing

Established technique of flow graph with Cyclomatic complexity was used to derive test cases for **Instant Dispatch** all the functions. The main steps in deriving test cases were:

Use the design of the code and draw correspondent flow graph

Determine the Cyclometric complexity of resultant flow graph, using formula:
 $V(G)=E-N+2$ or

$V(G)=P+1$ or

$V(G)=\text{Number of Regions}$

Where $V(G)$ is Cyclomatic complexity, E is
the number of edges,
 N is the number of flow graph nodes,

P is the number of predicate nodes.

Determine the basis of set of linearly independent paths.

3. Conditional Testing

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generated on particular condition is traced to uncover any possible errors.

4. Data Flow Testing

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable was declared. The definition-use chain method was used in this type of testing. These were particularly useful in nested statements.

5. Loop Testing

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for “**Instant Dispatch**” all loops:

- All the loops were tested at their limits, just above them and just below them.
- All the loops were skipped at least once.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.
- Each unit has been separately tested by the development team itself and all the input have been validated.

User Module Output Screen

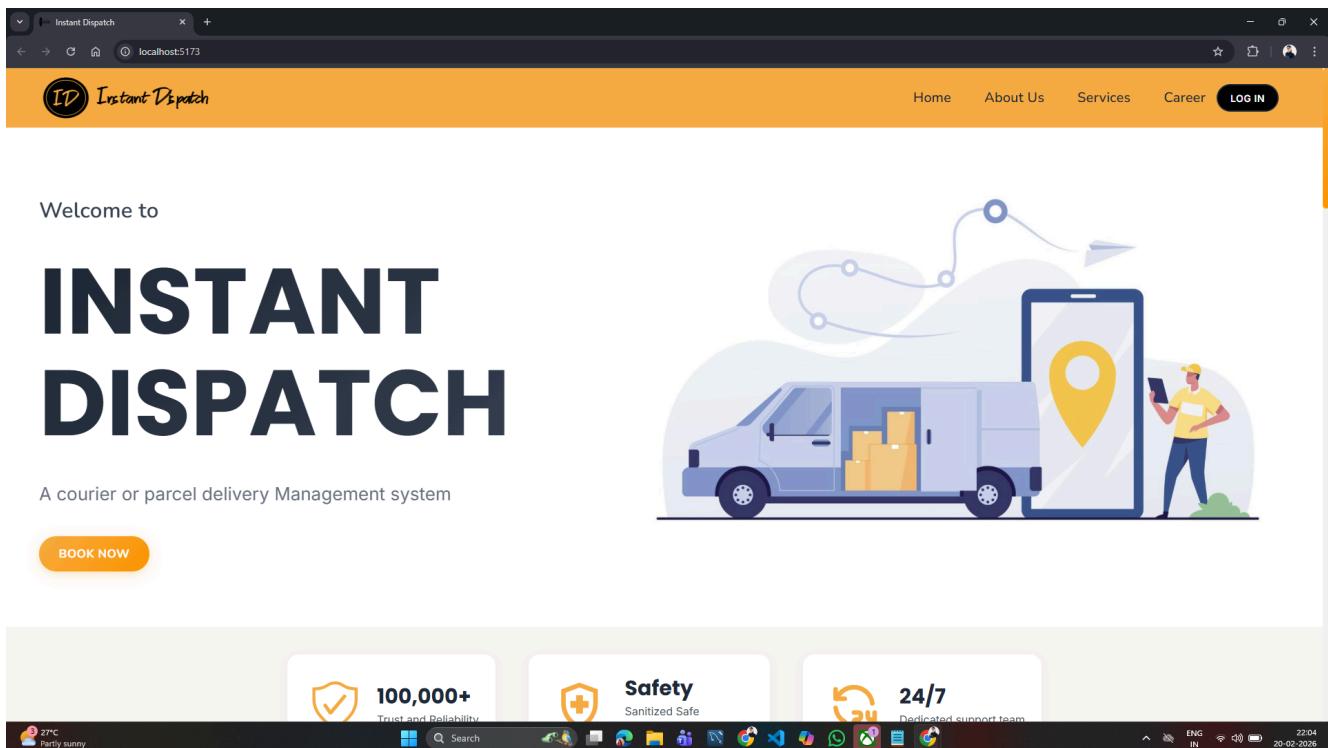


Fig. Landing Screen

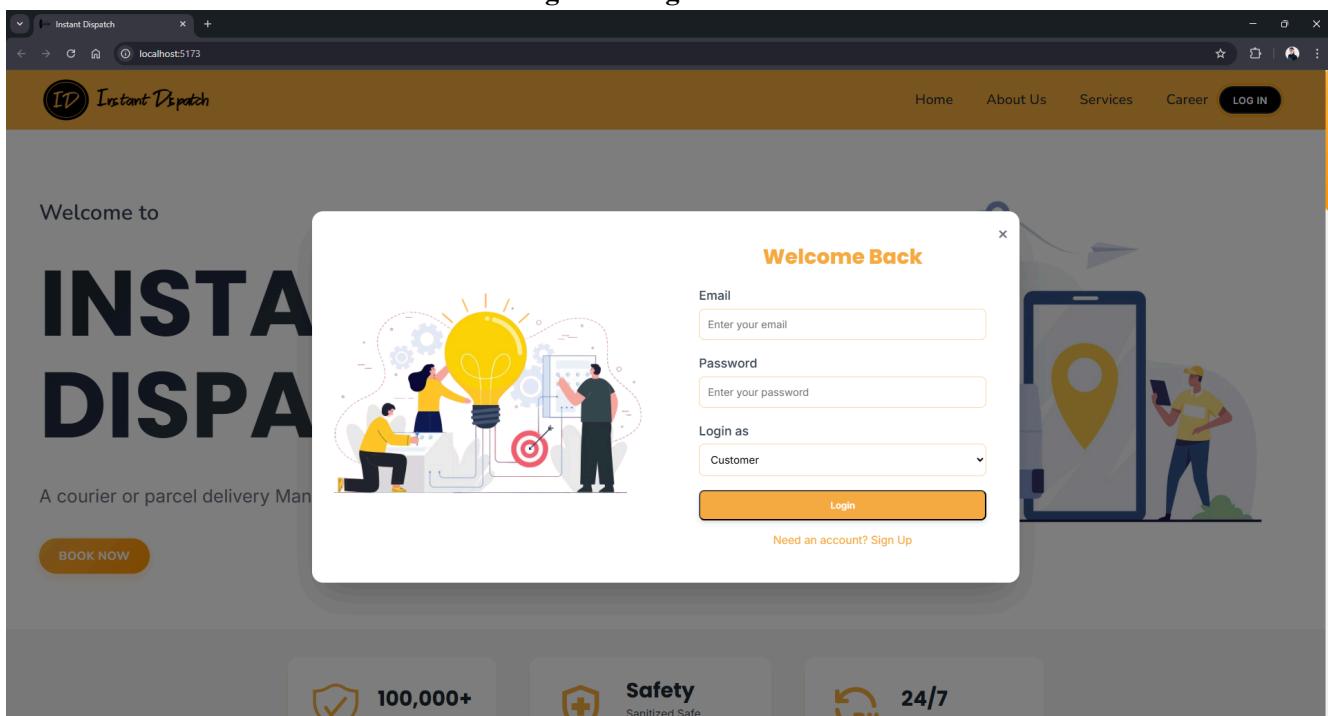


Fig. Login Screen

The screenshot shows the Visual Studio Code interface with the following details:

- File Path:** instantDispatch > src > Components > NavbarComponents > UserLogin.jsx
- Code Editor:** The current file is UserLogin.jsx, which contains functional component logic for handling user login and sign-up.
- Imports:** The code imports useState, useNavigate, RiderForm.css, and businessImg from assets.
- State Management:** It uses useState to manage isSignup, formData, and popup-related variables.
- Logic:** The component handles form submission via handleSubmit, which makes a POST request to the '/signup' endpoint with email, password, and role.
- Environment:** The status bar shows system information like battery level (20-2026), network, and time (22:08).

This screenshot is identical to the one above, showing the same code editor setup and environment. The main difference is the syntax highlighting, which uses a color scheme where comments and strings are highlighted in green, making the code easier to read.

Fig. Login Screen Code

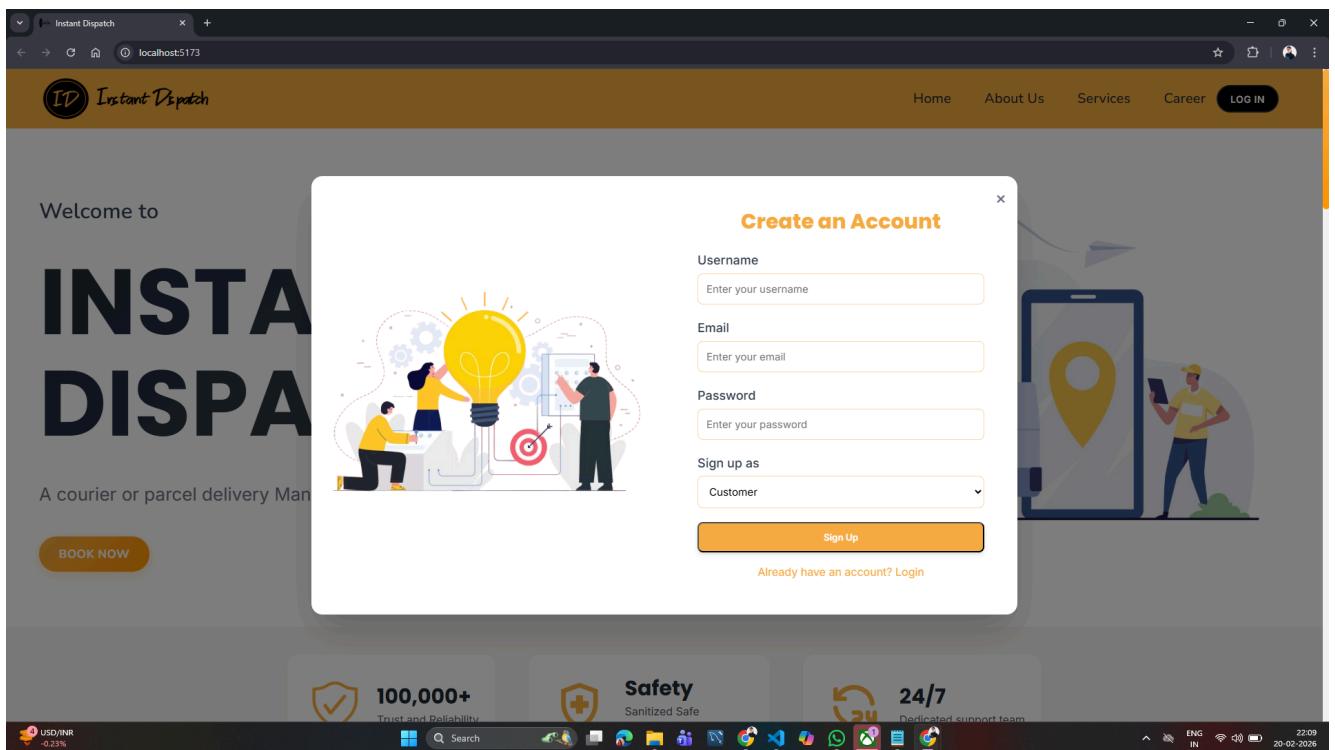


Fig. Registration Screen

```

File Edit Selection View Go Run Terminal Help < > UserSignup.jsx - Instant-dispatch - Visual Studio Code
EXPLORER
instantDispatch > src > Components > NavbarComponents > UserSignup.jsx ...
instantDispatch > src > Components > HomeComponent > safety.css
instantDispatch > src > Components > LoginComponent > index.html
instantDispatch > src > Components > NavComponents > CustomerNav.css
instantDispatch > src > Components > NavComponents > CustomerNav.jsx
instantDispatch > src > Components > NavComponents > navbar.css
instantDispatch > src > Components > NavComponents > navbar.jsx
instantDispatch > src > Components > NavComponents > NotificationList.jsx
instantDispatch > src > Components > NavComponents > Rider.css
instantDispatch > src > Components > NavComponents > RiderForm.css
instantDispatch > src > Components > NavComponents > RiderForm.jsx
instantDispatch > src > Components > NavComponents > RiderNav.jsx
instantDispatch > src > Components > UserLogin.jsx
instantDispatch > src > Components > UserSignup.jsx
instantDispatch > src > Orderlist
instantDispatch > src > redux
instantDispatch > src > servicecomponent > Profile.jsx
instantDispatch > src > servicecomponent > About.jsx
instantDispatch > src > servicecomponent > App.css
instantDispatch > src > servicecomponent > App.jsx
instantDispatch > src > servicecomponent > Book.jsx
instantDispatch > src > servicecomponent > Career.jsx
instantDispatch > src > config.js
instantDispatch > src > global-animations.css
instantDispatch > OUTLINE
instantDispatch > TIMELINE
instantDispatch > MYSQL
instantDispatch > Nikita*
instantDispatch > Launchpad
instantDispatch > Sign in to Jira
instantDispatch > No active issue
instantDispatch > Sign in to Bitbucket
instantDispatch > 0 △ 0
instantDispatch > Amazon Q
instantDispatch > Git Graph
instantDispatch > BLACKBOX Agent
instantDispatch > Open Website
instantDispatch > Generate Commit Message
instantDispatch > Windurf: Login
instantDispatch > Prettier
instantDispatch > USD/INR -0.23%
instantDispatch > Search
instantDispatch > Home
instantDispatch > About Us
instantDispatch > Services
instantDispatch > Career
instantDispatch > LOG IN
instantDispatch > 22:10
instantDispatch > ENG IN
instantDispatch > 20-02-2026

```

The screenshot shows the Visual Studio Code interface with the "UserSignup.jsx" file open in the editor. The code is a functional component for user registration. It includes logic for handling form submission, displaying error messages, and managing a registration failed state. The code uses React components like `Form`, `Input`, and `Text` along with hooks like `useState` and `useEffect`. The file is part of a larger project structure with components for navigation and user authentication.

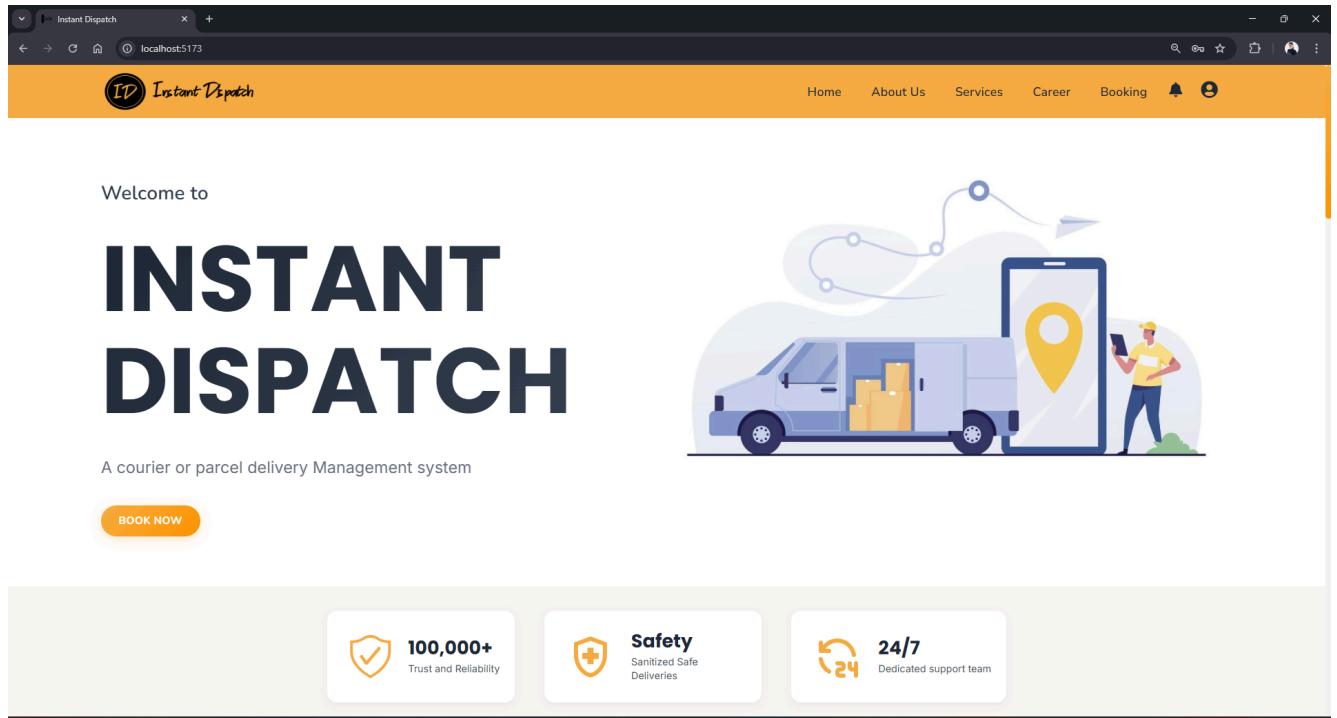
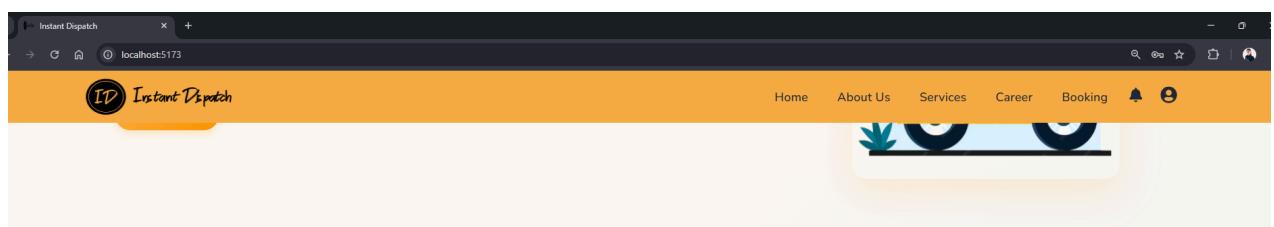


Fig. Overview Screen



How it works



Book a Rider

Start by visiting our booking page and filling out the necessary details.



Parcel with us

Your parcel is securely packed and collected from the pickup point.



Delivery

Our riders are committed to delivering your parcel on time.



Fig. Overview Screen Code

The screenshot shows a web browser window with the URL `localhost:5173/career`. The page has a yellow header bar with the logo 'Instant Dispatch'. Below the header is a light gray form titled 'Register For Job'. The form contains five input fields: 'Name' (placeholder 'Enter your name'), 'Age' (placeholder 'Enter your age'), 'Experience' (placeholder 'Enter your experience'), 'Education' (placeholder 'Enter your education'), and 'Vehicle' (a dropdown menu with placeholder 'Select vehicle'). There is also a checkbox labeled 'Do you have a license?' and a yellow 'Submit Application' button at the bottom.

Fig. Career Form page

The screenshot shows the Visual Studio Code interface with the file `CareerForm.jsx` open in the center editor tab. The code is a functional component for a career application form. It includes a handle for a submit event, a close function for a modal, and a render function that creates a form with five input fields ('name', 'age', 'experience', 'education') and a 'Vehicle' dropdown. The code uses JSX syntax with class names and state management logic.

```

const handleSubmit = async e => {
  console.error(`Submission error: ${e}`);
  setPopUpMessage('Failed to submit application');
};

const closePopUp = () => setShowPopUp(false);

return (
  <div className="career-form-container">
    <form onSubmit={handleSubmit} className="career-form w-100 p-8 rounded-lg shadow-md my-36">
      <div className="flex justify-center py-5">
        <h2>Register For Job</h2>
      </div>

      {[ 'name', 'age', 'experience', 'education'].map(field => (
        <div key={field} className="form-group mb-6">
          <label htmlFor={field} className="block text-gray-700 font-bold mb-2 text-lg">
            {field.charAt(0).toUpperCase() + field.slice(1)}
          </label>
          <input
            type={field === 'age' ? 'number' : 'text'}
            id={field}
            name={field}
            value={formData[field]}
            onChange={handleChange}
            required
            className="form-input w-full p-3 border border-gray-300 rounded-lg"
            placeholder={`Enter your ${field}`}
          />
        </div>
      ))}

      <div className="form-group mb-6">
        <label htmlFor="vehicle" className="block text-gray-700 font-bold mb-2 text-lg">Vehicle:</label>
        <select
          id="vehicle"
          name="vehicle"
        >
      
```

Fig. Career Form code

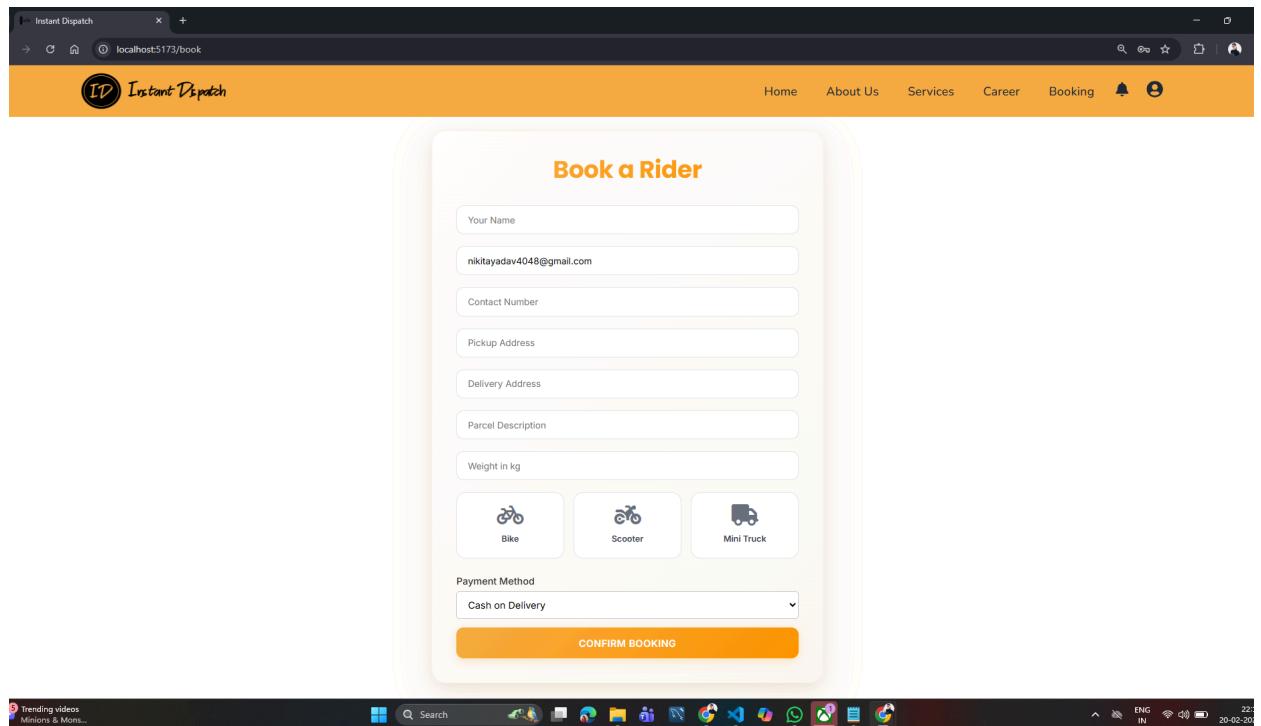


Fig. Booking Form Code

```

instantDispatch > src > Components > bookingcomponent > BookForm.jsx ...
You, 4 days ago | 2 authors (You and one other)
1 import React, { useState, useEffect } from 'react';
2 import axios from 'axios';
3 import { useDispatch } from 'react-redux';
4 import { addNotification } from '../redux/notificationSlice';
5 import './Form.css';
6
7 const BookForm = () => {
8   const dispatch = useDispatch();
9
10  const [formData, setFormData] = useState({
11    name: '',
12    email: '',
13    contact: '',
14    pickupAddress: '',
15    deliveryAddress: '',
16    parcelDescription: '',
17    weight: '',
18    vehicle: '',
19    price: 0,
20    paymentStatus: 'cod',
21  });
22
23 // Auto-fill email from logged-in user
24 useEffect(() => {
25   const user = JSON.parse(localStorage.getItem('user') || '{}');
26   if (user.email) {
27     setFormData(prev => ({ ...prev, email: user.email }));
28   }
29 }, []);
30
31 const [suggestions, setSuggestions] = useState([]);
32 const [isPickup, setIsPickup] = useState(true);
33 const [errors, setErrors] = useState({ pickupAddress: '', deliveryAddress: '' });
34 const [liveDistance, setLiveDistance] = useState(null);
35 const [calculatedDistance, setCalculatedDistance] = useState(null);
36 const [showConfirmation, setShowConfirmation] = useState(false);
37 const [finalDistance, setFinalDistance] = useState(null);
38 const [finalPrice, setFinalPrice] = useState(null);

```

Fig. Booking Form Code

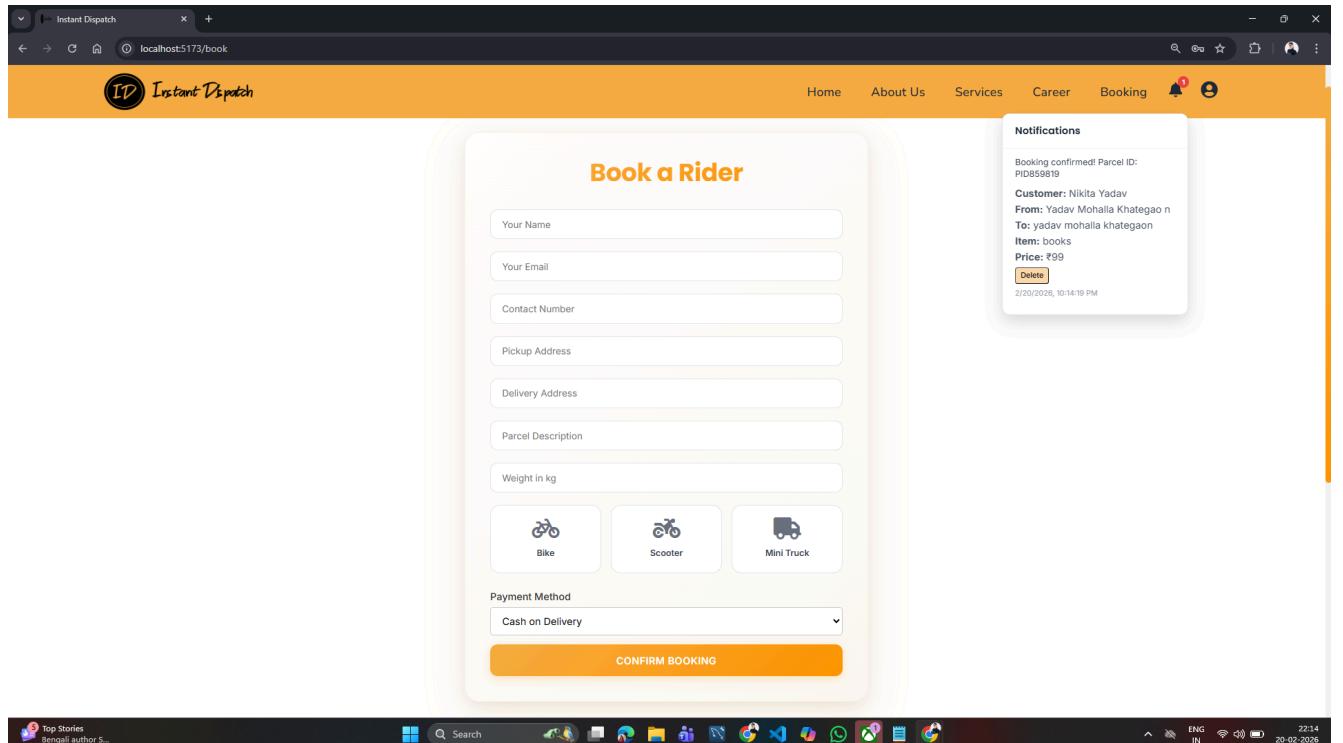


Fig. Notification section

The screenshot shows the 'NotificationList.jsx' file open in the Visual Studio Code editor. The code is a functional component that imports React, useSelector, useDispatch, and removeNotification from their respective modules. It defines a handleDelete function to remove notifications by id. The component itself uses CSS-in-JS to style its structure, including a header, a list of notifications, and a delete button for each notification item. The code ends with an export default statement for the NotificationList component.

```

File Edit Selection View Go Run Terminal Help < - > Q NotificationList.jsx - Instant-dispatch - Visual Studio Code
EXPLORER
INSTANT-DISPATCH
instantDispatch
Components
  homecomponent
    Safety.jsx
  LoginComponent
    index.html
  NavComponents
    CustomerNav.css
    CustomerNav.js
    navbar.css
    navbar.js
  NotificationList.jsx
    Rider.css
    RiderForm.css
    RiderForm.jsx
    RiderNav.jsx
    UserLogin.jsx
    UserSignup.jsx
  Orderlist
  redux
  servicecomponent
    Profile.jsx
    About.jsx
    App.css
    App.jsx
    RiderNav.jsx
    Book.jsx
    Career.jsx
    config.js
    global-animations.css
    Home.jsv
  OUTLINE
  TIMELINE
  MYSQL
Nikita* Launchpad Sign in to Jira No active issue Sign in to Bitbucket Amazon Q Git Graph BLACKBOX Agent Open Website Generate Commit Message
USD/NR -0.25% Windsor: Login Prettier ENG IN 22:26 20-02-2026

```

```

37 // export default NotificationList;
38
39 import React from 'react';
40 import { useSelector, useDispatch } from 'react-redux';
41 import { removeNotification } from './notificationSlice';
42
43 const NotificationList = () => {
44   const notifications = useSelector((state) => state.notifications.notifications);
45   const dispatch = useDispatch();
46
47   const handleDelete = (id) => {
48     dispatch(removeNotification(id));
49   };
50
51
52   return (
53     <div>
54       {notifications.length > 0 && (
55         <div className="notification-section flex flex-col items-center h-40">
56           <h2>Notifications</h2>
57           <div className="notifications">
58             {notifications.map((notification) => (
59               <li key={notification.id} className="notification">
60                 <div>
61                   <p>{notification.message}</p>
62                   <p>{notification.timestamp}</p>
63                 </div>
64                 <button onClick={() => handleDelete(notification.id)}>Delete</button>
65               </li>
66             ))}
67           </div>
68         </div>
69       )}
70     );
71   );
72 }
73
74
75 export default NotificationList;

```

Fig.Notification code

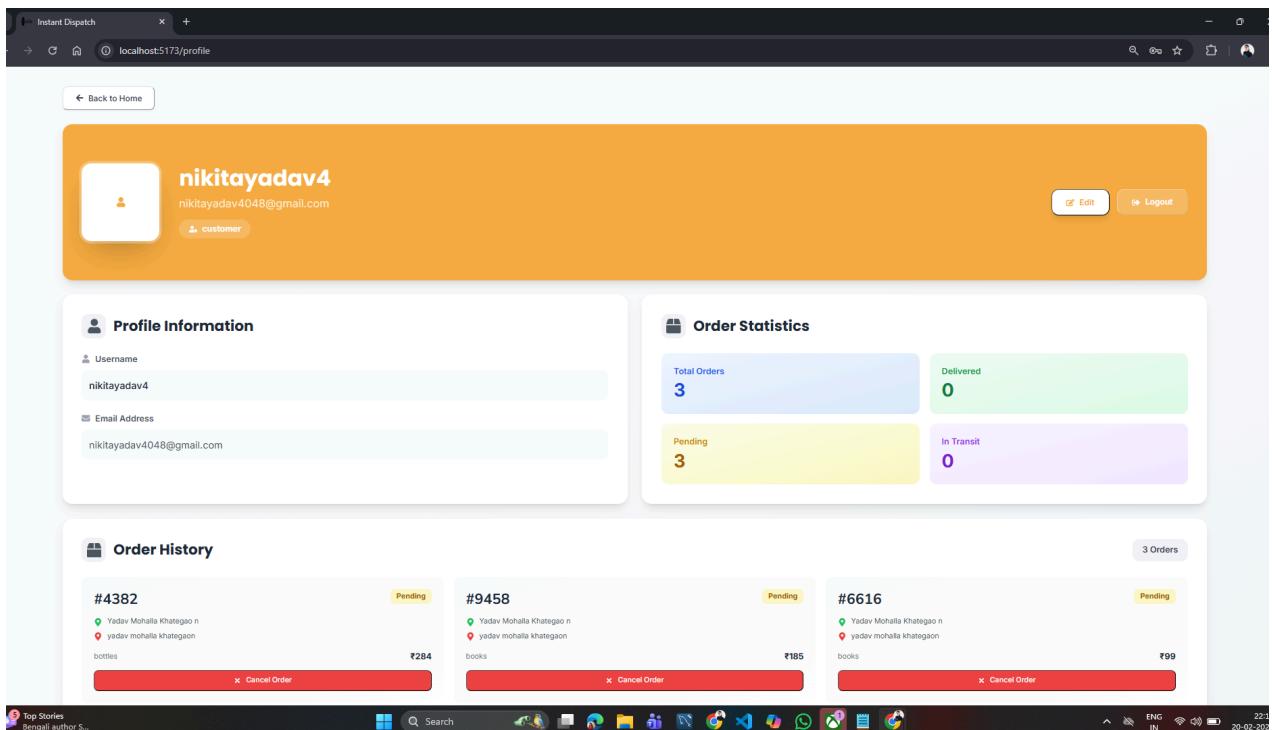


Fig. Profile screen code

```

import { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { FaUser, FaEnvelope, FaUserTag, FaEdit, FaSave, FaTimes, FaSignOutAlt, FaArrowLeft, FaBox, FaMapMarkerAlt, FaRupeeSign } from 'react-icons/fa';
import { useDispatch } from 'react-redux';
import { addNotification } from './redux/notificationSlice';
import axios from 'axios';

const Profile = () => {
  const [user, setUser] = useState(null);
  const [isEditing, setIsEditing] = useState(false);
  const [formData, setFormData] = useState({
    username: '',
    email: '',
    role: ''
  });
  const [orders, setOrders] = useState([]);
  const [loadingOrders, setLoadingOrders] = useState(false);
  const navigate = useNavigate();
  const dispatch = useDispatch();

  useEffect(() => {
    const userData = localStorage.getItem('user');
    if (userData) {
      const parsedUser = JSON.parse(userData);
      setUser(parsedUser);
      setFormData({
        username: parsedUser.username || parsedUser.name || '',
        email: parsedUser.email || '',
        role: parsedUser.role || ''
      });
      fetchOrders(parsedUser.email);
    } else {
      navigate('/login');
    }
  }, [navigate]);

  const fetchOrders = async (email) => {
    setLoadingOrders(true);
    ...
  };
}

```

Fig. Profile section code

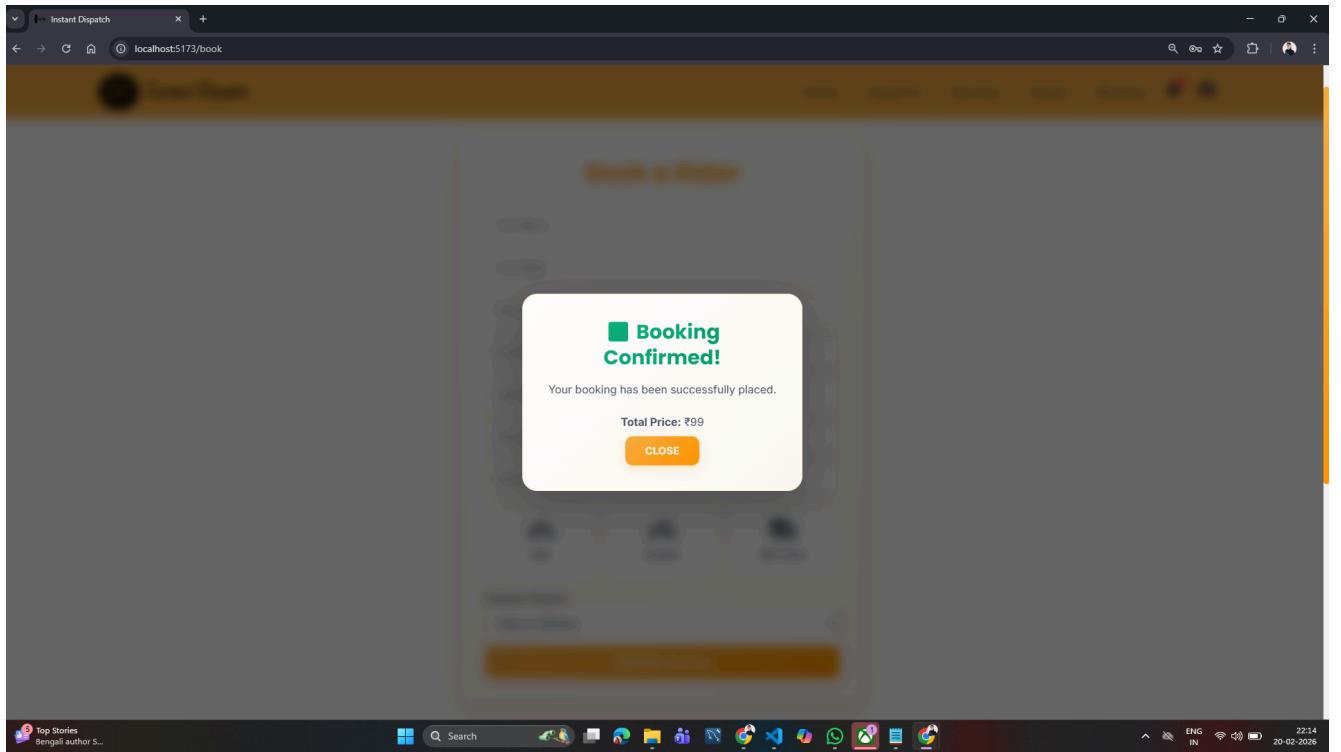


Fig. Booking Form confirmed

File Edit Selection View Go Run Terminal Help ← → Q BookForm.jsx - Instant-dispatch - Visual Studio Code - Modified D BLACKBOX

EXPLORER instantDispatch > src > Components > bookingcomponent > BookForm.jsx > ...

```

instantDispatch > src > Components > bookingcomponent > BookForm.jsx > ...
  244
  245    const BookForm = () => {
  246      247        248          // Validate vehicle selection
  249          if (!formData.vehicle) {
  250            alert('Please select a vehicle type');
  251            return;
  252          }
  253
  254        const pickupCoords = await validateAddress(formData.pickupAddress, 'pickupAddress');
  255        const deliveryCoords = await validateAddress(formData.deliveryAddress, 'deliveryAddress');
  256        if (!pickupCoords || !deliveryCoords) return;
  257
  258        const distance = haversine(pickupCoords.lng, pickupCoords.lat, deliveryCoords.lng, deliveryCoords.lat);
  259        setLiveDistance(distance);
  260        setFinalDistance(distance);
  261
  262        const totalPrice = Math.floor(vehiclePrices[formData.vehicle] * distance);
  263        setFinalPrice(totalPrice);
  264
  265        const user = JSON.parse(localStorage.getItem('user') || '{}');
  266        const bookingData = [
  267          ...formData,
  268          price: totalPrice,
  269          contact: Number(formData.contact),
  270          customerUsername: formData.email || user.email || 'Unknown User',
  271          customerEmail: formData.email,
  272        ];
  273
  274        // Check payment method
  275        if (formData.paymentStatus === 'online') {
  276          // Load Razorpay script if not loaded
  277          if (!window.Razorpay) {
  278            const script = document.createElement('script');
  279            script.src = 'https://checkout.razorpay.com/v1/checkout.js';
  280            script.onload = () => handlePayment(bookingData, totalPrice);
  281            document.body.appendChild(script);
  282          }
  283        }
  284      };
  285    };
  286  
```

OUTLINE TIMELINE MYSQL Nikita* Launchpad Sign in to Jira No active issue Sign in to Bitbucket Amazon Q Git Graph BLACKBOX Agent Open Website Generate Commit Message Windsor: Login Prettier ENG IN 22:28 20-02-2026

Fig.Booking Form confirmed code

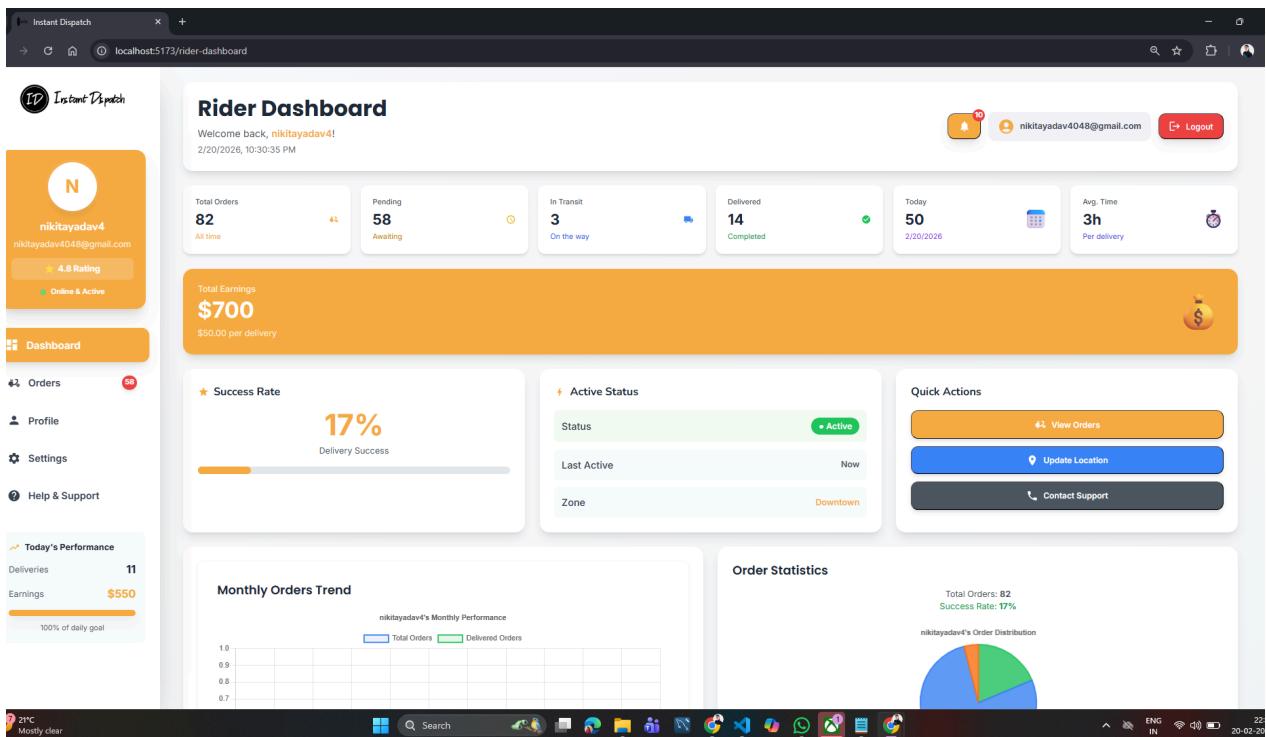


Fig. Rider Dashboard

```

File Edit Selection View Go Run Terminal Help < > Dashboard.jsx - Instant-dispatch - Visual Studio Code - Modified
instantDispatch > src > Components > Dashboard-orders > Dashboard.jsx > ...
You, 4 days ago | author (You)
1 import React, { useEffect, useState } from 'react';
2 import './Dashboard.css';
3 import './Animations.css';
4 import { MdAccountCircle, MdTrendingUp, MdDeliveryDining, MdCheckCircle, MdLogout, MdAccessTime, MdLocalShipping, MdMessage, MdNotifications, MdLoc
5 import { useSelector, useDispatch } from 'react-redux';
6 import { useNavigate } from 'react-router-dom';
7 import { fetchBookings, handleComplete } from '../redux/ordersSlice';
8 import OverviewCards from './OverviewCards';
9 import Charts from './Charts';
10 import OrdersGraph from './OrdersGraph';
11 import Notifications from './Notifications';
12 import PerformanceMetrics from './PerformanceMetrics';
13
14
15 const Dashboard = () => {
16   const dispatch = useDispatch();
17   const navigate = useNavigate();
18   const [data: orders, loading] = useSelector(state => state.orders);
19   const [user] = useState(() => JSON.parse(localStorage.getItem('user') || '{}'));
20   const [stats, setStats] = useState({ total: 0, pending: 0, delivered: 0, earnings: 0, todayOrders: 0, avgDeliveryTime: 0 });
21   const [currentTime, setCurrentTime] = useState(new Date());
22   const [notifications, setNotifications] = useState([]);
23   const [showNotifications, setShowNotifications] = useState(false);
24   const [completedOrders, setCompletedOrders] = useState(new Set());
25
26   useEffect(() => {
27     dispatch(fetchBookings());
28     // Update time every minute
29     const timer = setInterval(() => setCurrentTime(new Date()), 60000);
30
31     let previousOrders = [];
32
33     // Real-time order monitoring
34     const orderMonitor = setInterval(async () => {
35       try {
36         const response = await fetch('http://localhost:5002/api/bookings');
37         const newOrders = await response.json();
38
39         if (newOrders.length > 0) {
40           const newOrder = newOrders[0];
41           const orderIndex = previousOrders.findIndex(order => order.id === newOrder.id);
42
43           if (orderIndex === -1) {
44             previousOrders.push(newOrder);
45             dispatch(handleComplete(newOrder));
46           } else {
47             previousOrders[orderIndex] = newOrder;
48             dispatch(handleComplete(newOrder));
49           }
50         }
51       } catch (error) {
52         console.error(error);
53       }
54     }, 1000);
55
56   }, []);
57
58   return (
59     <div>
60       <h1>Welcome to the Rider Dashboard!</h1>
61       <h2>Order Summary</h2>
62       <table>
63         <thead>
64           <tr>
65             <th>Category</th>
66             <th>Value</th>
67           </tr>
68         </thead>
69         <tbody>
70           <tr>
71             <td>Total Orders</td>
72             <td>82</td>
73           </tr>
74           <tr>
75             <td>Pending</td>
76             <td>58</td>
77           </tr>
78           <tr>
79             <td>In Transit</td>
80             <td>3</td>
81           </tr>
82           <tr>
83             <td>Delivered</td>
84             <td>14</td>
85           </tr>
86           <tr>
87             <td>Today</td>
88             <td>50</td>
89           </tr>
90           <tr>
91             <td>Avg. Time</td>
92             <td>3h</td>
93           </tr>
94         </tbody>
95       </table>
96       <h2>Order Details</h2>
97       <table>
98         <thead>
99           <tr>
100             <th>Order ID</th>
101             <th>Status</th>
102             <th>Customer Name</th>
103             <th>Order Type</th>
104             <th>Delivery Time</th>
105           </tr>
106         </thead>
107         <tbody>
108           <tr>
109             <td>12345678901234567890</td>
110             <td>Delivered</td>
111             <td>John Doe</td>
112             <td>Food</td>
113             <td>2023-02-20 14:30</td>
114           </tr>
115           <tr>
116             <td>12345678901234567891</td>
117             <td>Pending</td>
118             <td>Jane Smith</td>
119             <td>Groceries</td>
120             <td>2023-02-20 15:00</td>
121           </tr>
122           <tr>
123             <td>12345678901234567892</td>
124             <td>In Transit</td>
125             <td>Mike Johnson</td>
126             <td>Food</td>
127             <td>2023-02-20 15:30</td>
128           </tr>
129         </tbody>
130       </table>
131     </div>
132   );
133 }
134
135 export default Dashboard;

```

Fig. Rider Dashboard code

The screenshot shows the 'Order Management' section of the Instant Dispatch rider dashboard. At the top, it displays the total number of orders (82), categorized by status: Out for Delivery (3), Completed (14), and Cancelled (5). Below this is a search bar and dropdown filters for customer name, all statuses, and all vehicles. The main table lists individual orders with columns for Order ID, Status, Customer, Vehicle, Weight, Amount, and Actions (View and Accept buttons). On the left sidebar, there are links for Dashboard, Orders (58), Profile, Settings, Help & Support, and Today's Performance (Deliveries: 11, Earnings: \$550). The bottom of the screen shows a Windows taskbar with various icons and system status.

Fig. Order List

The screenshot shows the 'Customer Details' page for order #9268. It includes sections for Customer Information (Customer Name: Nikita yadav, Email Address: nikitayadav4444@gmail.com, Phone Number: 9596978965) and Delivery Details (Status: Pending, Payment Status: Unpaid, Amount: ₹2392, Payment Type: Cash on Delivery). The delivery address is listed as Khaigaon, SH70, Khaygaon, Khaygaon - 450991, Madhya Pradesh, India. The package description is books, weight is 1 kg, and vehicle type is Scooter. Buttons for 'Accept Order' and 'Reject Order' are at the bottom. The left sidebar is identical to the Order List page, showing Today's Performance (Deliveries: 11, Earnings: \$550). The bottom of the screen shows a Windows taskbar.

Fig. Order Details

```

File Edit Selection View Go Run Terminal Help < -> OrderList.jsx - Instant-dispatch - Visual Studio Code - Modified
EXPLORER instantDispatch > src > Components > Dashboard-orders > OrderList.jsx ...
instantDispatch > src > Components > Dashboard-orders > OrderList.jsx ...
You 3 days ago | 1 author (You)
1 import React, { useEffect, useState } from 'react';
2 import { useDispatch, useSelector } from 'react-redux';
3 import { handleAccept, handleDetails, handleFilter, fetchBookings, setRiderOrders, updateOrderStatus } from '../redux/ordersSlice';
4 import { useNavigate } from 'react-router-dom';
5 // import socketService from '../../../../../services/socketService'; // Removed - file missing
6 import './OrderList.css';
7 import OrderListDashB from './OrderListDashB';

8 const OrdersPage = () => {
9   const dispatch = useDispatch();
10  const navigate = useNavigate();
11  const [filteredData, orders, loading] = useSelector((state) => state.orders);
12  const [searchTerm, setSearchTerm] = useState('');
13  const [statusFilter, setStatusFilter] = useState('');
14  const [vehicleFilter, setVehicleFilter] = useState('');

15  useEffect(() => {
16    const user = JSON.parse(localStorage.getItem('user') || '{}');
17    const fetchRiderOrders = async () => {
18      try {
19        console.log('Fetching rider orders from API...');
20        const response = await fetch("http://localhost:5002/api/bookings/");
21        if (!response.ok) {
22          throw new Error(`HTTP error! status: ${response.status}`);
23        }
24        const data = await response.json();
25        console.log('Fetched orders:', data.length, 'orders');
26        dispatch(setRiderOrders(data));
27      } catch (error) {
28        console.error('Error fetching rider orders:', error);
29      }
30    };
31    fetchRiderOrders();
32  }, [dispatch]);
33
34  return () => {};
35
36 }, [dispatch]);
37
38

```

Fig. Order List Code

```

File Edit Selection View Go Run Terminal Help < -> OrderDetails.jsx - Instant-dispatch - Visual Studio Code
EXPLORER instantDispatch > src > Components > Dashboard-orders > OrderDetails.jsx ...
instantDispatch > src > Components > Dashboard-orders > OrderDetails.jsx ...
You 5 months ago | 1 author (You)
1 import React from 'react';
2 import { useSelector, useDispatch } from 'react-redux';
3 import { useNavigate, useParams } from 'react-router-dom';
4 import { handleComplete, handleBack } from '../redux/ordersSlice';
5 import { MdArrowBack, MdlocationOn, MdPhone, MdEmail, MdCheckCircle, MdAccessTime } from 'react-icons/md';
6
7 const OrderDetails = () => {
8   const dispatch = useDispatch();
9   const navigate = useNavigate();
10  const { orderId } = useParams();
11  const selectedOrder = useSelector(state => state.orders.selectedOrder);
12
13  if (!selectedOrder) {
14    return (
15      <div className="min-h-screen bg-gradient-to-br from-orange-50 to-white p-6 flex items-center justify-center">
16        <div className="text-center">
17          <p>Order not found</p>
18          <button onClick={() => navigate('../orders')} className="px-6 py-3 bg-orange-50 text-white rounded-lg hover:bg-orange-600 transition-colors">
19            Back to Orders
20          </button>
21        </div>
22      </div>
23    );
24  }
25
26  const handleCompleteOrder = () => {
27    dispatch(handleComplete(selectedOrder._id));
28    navigate('../orders');
29  };
30
31  const handleBackClick = () => {
32    dispatch(handleBack());
33    navigate('../orders');
34  };
35
36
37
38

```

Fig.Order Details Code

Instant Dispatch

localhost:5173/rider-dashboard/profile

nikitayadav4's Profile

Manage your profile and view delivery performance

Personal Information

- Name: nikitayadav4
- Phone: +1234 567 890
- Email: nikitayadav4048@gmail.com
- Rider ID: Not provided
- Rating: 4.8 Rating (Online & Active)

Delivery Statistics

Total Orders	82
Completed	14
Pending	58
Earnings	\$700

Vehicle Information

- Vehicle Type: Motorcycle
- Vehicle Model: Not specified
- License Plate: AB 1234 CD
- License Number: Not provided
- Status: Active

Recent Order Activities

Last updated: 10:31:51 PM

Profile.jsx - Instant-dispatch - Visual Studio Code - Modified

```

const Profile = () => {
  const [orders, setRiderDetails] = useState({
    totalDeliveries: 0,
    completedDeliveries: 0,
    pendingDeliveries: 0
  });

  useEffect(() => {
    if (orders.length > 0) {
      const totalDeliveries = orders.length;
      const completedDeliveries = orders.filter(order => order.status === 'Delivered').length;
      const pendingDeliveries = orders.filter(order => order.status === 'Pending').length;

      setRiderDetails(prev => ({
        ...prev,
        totalDeliveries,
        completedDeliveries,
        pendingDeliveries
      }));
    }
  }, [orders]);

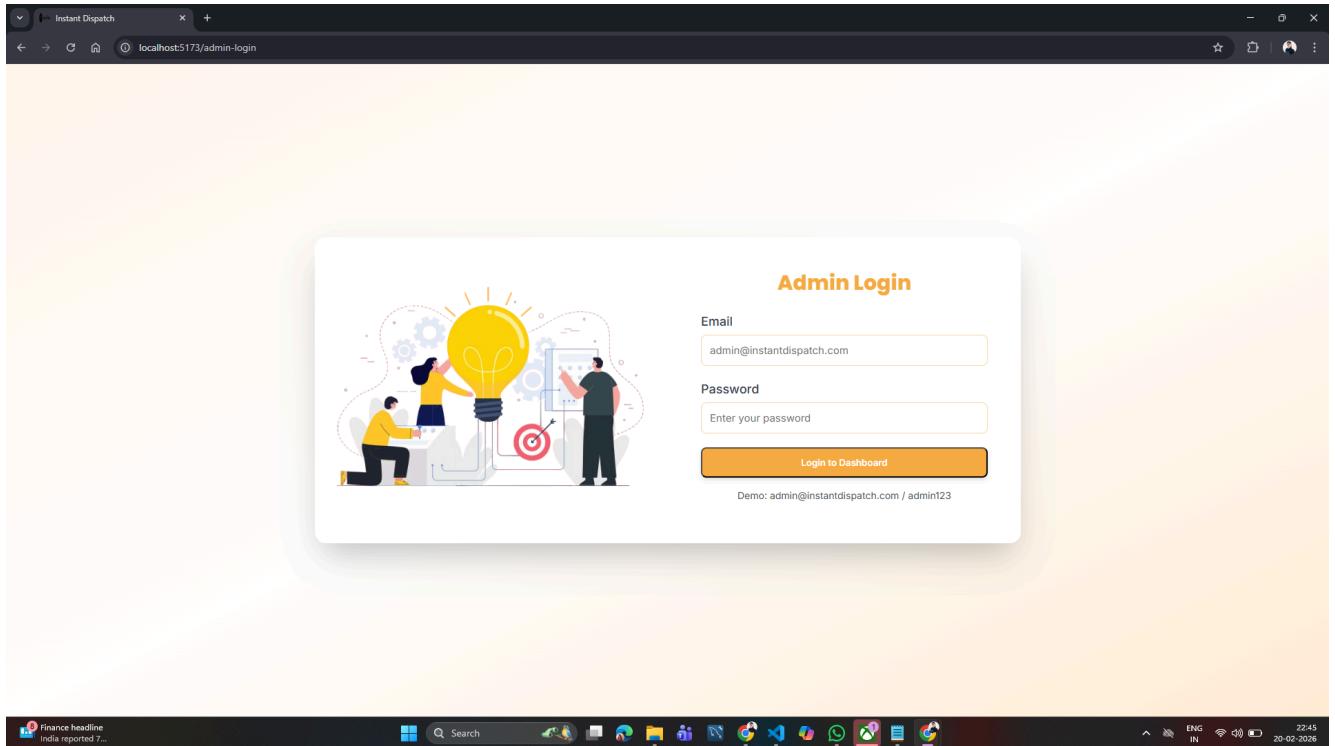
  // Real-time recent activities with live timestamps
  const recentActivities = [...orders]
    .sort((a, b) => new Date(b.createdAt || b.orderdate) - new Date(a.createdAt || a.orderdate))
    .slice(0, 5)
    .map(order, index) => {
      id: order._id || index,
      activity: `${order.status === 'Delivered' ? 'Delivered' : 'Out for Delivery'} ? 'Picked Up' : 'Received'` Order ${order.name} | time: new Date(order.createdAt || order.orderdate).toLocaleString(),
      status: order.status,
      isNew: new Date(order.createdAt || order.orderdate) > new Date(Date.now() - 30000) // New if within last 30 seconds
    });
}

const handleManualRefresh = () => {
  dispatch(fetchBookings());
  setLastUpdate(new Date());
};

const getVehicleIcon = (vehicleType) => {
  switch(vehicleType?.toLowerCase()) {
    case 'cycle':
    case 'bicycle':

```

Fig. Rider Profile



File Edit Selection View Go Run Terminal Help ← → Q AdminLogin.jsx - Instant-dispatch - Visual Studio Code - Untracked D BLACKBOX

```

const Adminlogin = () => {
  const [credentials, setCredentials] = useState({ email: '', password: '' });
  const [showPopup, setShowPopup] = useState(false);
  const [popupMessage, setPopupMessage] = useState('');
  const navigate = useNavigate();

  const handleLogin = (e) => {
    e.preventDefault();
    if (credentials.email === 'admin@instantdispatch.com' && credentials.password === 'admin123') {
      localStorage.setItem('user', JSON.stringify({ role: 'admin', email: credentials.email }));
      navigate('/admin-dashboard');
    } else {
      setPopupMessage('Invalid credentials!');
      setShowPopup(true);
    }
  };

  const closePopup = () => setShowPopup(false);

  return (
    <div className="min-h-screen flex items-center justify-center bg-gradient-to-br from-orange-50 via-white to-orange-100 p-4">
      <div className="flex w-full max-w-5xl bg-white rounded-2xl shadow-2xl overflow-hidden">
        {/* Left Side - Image */}
        <div className="hidden lg:flex lg:w-1/2 items-center justify-center p-8">
          <img src={businessImg} alt="Admin Panel" className="w-full h-auto object-contain" />
        </div>
        {/* Right Side - Form */}
        <div className="w-full lg:w-1/2 p-8 lg:p-12">
          <h2 className="text-3xl font-extrabold text-center bg-gradient-to-r from-orange-500 to-orange-400 bg-clip-text text-transparent mb-6">
            Admin Login
          </h2>
          <form onSubmit={handleLogin} className="space-y-5">
            <div>

```

Fig. Rider Profile



```

instantDispatch > src > Components > Admin-Dashboard > AdminDashboard.jsx > ...
1  import React, { useState, useEffect } from 'react';
2  import { Routes, Route, Link, useLocation } from 'react-router-dom';
3  import { MdDashboard, MdPeople, MdDirectionsBike, MdShoppingCart, MdBarChart, MdSettings, MdLogout, MdMenu, MdClose } from 'react-icons/md';
4  import logo from './assets/logo-final.png';
5  import AdminOverview from './AdminOverview';
6  import ManageUsers from './ManageUsers';
7  import ManageRiders from './ManageRiders';
8  import ManageOrders from './ManageOrders';
9  import AdminAnalytics from './AdminAnalytics';
10 import AdminSettings from './AdminSettings';

11
12 const AdminDashboard = () => {
13  const [sidebarOpen, setSidebarOpen] = useState(true);
14  const location = useLocation();
15  const [user] = useState(() => JSON.parse(localStorage.getItem('user')) || '{}');
16
17  const menuItems = [
18    { path: '/admin-dashboard', icon: MdDashboard, label: 'Overview', exact: true },
19    { path: '/admin-dashboard/users', icon: MdPeople, label: 'Users' },
20    { path: '/admin-dashboard/riders', icon: MdDirectionsBike, label: 'Riders' },
21    { path: '/admin-dashboard/orders', icon: MdShoppingCart, label: 'Orders' },
22    { path: '/admin-dashboard/analytics', icon: MdBarChart, label: 'Analytics' },
23    { path: '/admin-dashboard/settings', icon: MdSettings, label: 'Settings' },
24  ];
25
26  const handleLogout = () => {
27    localStorage.removeItem('user');
28    window.location.href = '/login';
29  };
30
31  return (
32    <div className="flex h-screen bg-gray-50">
33      {/* Sidebar */}
34      <aside className={ `${ sidebarOpen ? 'w-72' : 'w-20' } bg-white text-gray-800 h-screen shadow-2xl fixed left-0 top-0 z-40 overflow-y-auto border-r border-gray-200 transition-all duration-300 ease-in-out >
35        <div className="p-6">
36          /* Header */
37          <div className="flex items-center justify-between mb-8 pb-6 border-b border-gray-200">
38            { sidebarOpen && <img src={ logo } alt="Logo" className="h-12 w-auto" />
39            <button onClick={() => setSidebarOpen(!sidebarOpen)} className="p-2 bg-gray-100 rounded-lg text-gray-700">
```

Fig.Admin Dashboard

The screenshot shows a web application titled "Manage Users" with the subtitle "View and manage all registered users". On the left, there's a sidebar with navigation links: Overview, Users (which is selected), Riders, Orders, Analytics, Settings, and Logout. The main content area displays a table of users with columns: Name, Email, Status, and Actions. Each user row includes a small profile icon, the user's name, their email, their status (active or inactive), and a red "Block" button. A search bar at the top allows users to search by name or email.

Name	Email	Status	Actions
Priya Kumar 1	priya1@example.com	active	<button>Block</button>
Amit Kumar 2	amit2@example.com	active	<button>Block</button>
Neha Kumar 3	neha3@example.com	active	<button>Block</button>
Vikram Kumar 4	vikram4@example.com	active	<button>Block</button>
Anjali Kumar 5	anjali5@example.com	active	<button>Block</button>
Rohan Kumar 6	rohan6@example.com	active	<button>Block</button>
Sneha Kumar 7	sneha7@example.com	active	<button>Block</button>
Arjun Kumar 8	arjun8@example.com	active	<button>Block</button>
Pooja Kumar 9	pooja9@example.com	active	<button>Block</button>

The screenshot shows the code for the "ManageUsers.jsx" component in a code editor. The code is written in JSX and includes imports from "react", "SearchBar", "Table", and "User". It defines a functional component "ManageUsers" that returns a div containing a search bar and a table. The search bar has a placeholder "Search users by name or email..." and an onChange event handler. The table has a header with columns for Name, Email, Status, and Actions. The body of the table maps over an array of users, creating a tr element for each user with a td for each column value. The code uses various CSS-in-JS classes like "p-6", "mb-6", "text-gray-800", etc., to style the components.

```

const ManageUsers = () => {
  return (
    <div className="p-6">
      <div className="mb-6">
        <h1 className="text-3xl font-bold text-gray-800 mb-2">Manage Users</h1>
        <p className="text-gray-600">View and manage all registered users</p>
      </div>

      /* Search Bar */
      <div className="bg-white rounded-xl p-4 shadow-sm border border-gray-200 mb-6">
        <div className="relative">
          <input className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400 size[24]" type="text" placeholder="Search users by name or email..." value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} className="w-full pl-12 pr-4 py-3 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-orange-500" />
        </div>
      </div>

      /* Users Table */
      <div className="bg-white rounded-xl shadow-sm border border-gray-200 overflow-hidden">
        <div className="overflow-x-auto">
          <table className="w-full">
            <thead className="bg-orange-50">
              <tr>
                <th className="px-6 py-4 text-left text-sm font-semibold text-gray-800">Name</th>
                <th className="px-6 py-4 text-left text-sm font-semibold text-gray-800">Email</th>
                <th className="px-6 py-4 text-left text-sm font-semibold text-gray-800">Status</th>
                <th className="px-6 py-4 text-left text-sm font-semibold text-gray-800">Actions</th>
              </tr>
            </thead>
            <tbody className="divide-y divide-gray-200">
              {filteredUsers.map((user) => (
                <tr key={user._id} className="hover:bg-gray-50">
                  <td className="px-6 py-4">
                    <div className="flex items-center gap-3">
                      ...
                    </div>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </div>
    </div>
  );
}

```

Fig. Admin User Management

The screenshot shows the 'Manage Riders' section of the Instant Dispatch Admin Dashboard. On the left, there's a sidebar with navigation links: Overview, Users, Riders (which is selected), Orders, Analytics, and Settings. A 'Logout' button is also present. The main area has a search bar and a dropdown for 'All Status'. Below is a grid of rider profiles:

Name	Vehicle Type	Email	Phone	Status
Rahul Kumar	Motorcycle	rahul@instantdispatch.com	+91 9876543210	Approved
Priya Sharma	Scooter	priya@instantdispatch.com	+91 9876543211	Approved
Amit Singh	Bike	amit@instantdispatch.com	+91 9876543212	Approved
Neha Patel	Cycle	neha@instantdispatch.com	+91 9876543213	Approved

```

const ManageRiders = () => {
  return (
    <div className="p-6">
      <div className="text-3xl font-bold text-gray-800 mb-2">Manage Riders</h1>
      <p className="text-gray-600">Approve, reject, and manage rider registrations</p>
    </div>

    /* Search and Filter */
    <div className="bg-white rounded-xl p-4 shadow-sm border border-gray-200 mb-6">
      <div className="flex flex-col md:flex-row gap-4">
        <div className="relative flex-1">
          <MuSearch className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400" size={24} />
          <input type="text" placeholder="Search riders..." value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} className="w-full pl-12 pr-4 py-3 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-orange-500" />
        </div>
        <select value={filter} onChange={(e) => setFilter(e.target.value)} className="px-4 py-3 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-orange-500">
          <option value="all">All Status</option>
          <option value="pending">Pending</option>
          <option value="approved">Approved</option>
          <option value="rejected">Rejected</option>
        </select>
      </div>
    </div>

    /* Riders Grid */
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
      {filteredRiders.map((rider) => (
        <div key={rider._id} className="bg-white rounded-xl p-6 shadow-sm border border-gray-200 hover:shadow-md transition-shadow">
          ...
        </div>
      ))
    </div>
  )
}

```

Fig. Admin Rider Management

Manage Orders

View and track all delivery orders

Order ID	Customer	Route	Status	Date
#53cd2e	5005	Indore, India → Indore, India	Delivered	Invalid Date
#53cd30	5161	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Out for Delivery	Invalid Date
#53cd32	6872	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Delivered	Invalid Date
#53cd34	8107	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Rejected	Invalid Date
#53cd36	7877	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Delivered	Invalid Date
#53cd38	3829	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Pending	Invalid Date
#53cd3a	4980	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Delivered	Invalid Date
#53cd3c	3421	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Delivered	Invalid Date
#53cd3e	3081	Chamba, Pukhri - 176319, Himachal Pradesh, India → Cha, Zanskar, India	Delivered	Invalid Date

```

const ManageOrders = () => {
  if (loading) return <div className="p-6"><p>Loading...</p></div>;
  return (
    <div className="p-6">
      <div className="mb-6">
        <h1 className="text-3xl font-bold text-gray-800 mb-2">Manage Orders</h1>
        <p className="text-gray-600">View and track all delivery orders</p>
      </div>

      /* Search and Filter */
      <div className="bg-white rounded-xl p-4 shadow-sm border border-gray-200 mb-6">
        <div className="flex flex-col md:flex-row gap-4">
          <div className="relative flex-1">
            <MdSearch className="absolute left-1/2 top-1/2 transform -translate-y-1/2 text-gray-400" size={24} />
            <input type="text" placeholder="Search by order ID or customer name..." value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} className="w-full p-1-2 pr-4 py-3 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-orange-500" />
          </div>
          <select value={statusFilter} onChange={(e) => setStatusFilter(e.target.value)} className="px-4 py-3 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-orange-500">
            <option value="all">All Status</option>
            <option value="Pending">Pending</option>
            <option value="Out for Delivery">Out for Delivery</option>
            <option value="Delivered">Delivered</option>
          </select>
        </div>
      </div>

      /* Orders Table */
      <div className="bg-white rounded-xl shadow-sm border border-gray-200 overflow-hidden">
        <div className="overflow-x-auto">

```

Fig. Admin Order Management

Conclusion

Project Summary

Instant Dispatch successfully provides a comprehensive platform for delivery management, connecting customers with reliable riders. The platform offers:

- User-friendly interface
- Real-time tracking
- Secure authentication
- Efficient order management
- Admin control panel

7.1 Limitations

In our "Instant Dispatch" we are still improving and trying to add new functionalities:

- Limited payment methods (currently Cash on Delivery and basic online payment)
- No multi-language support yet
- Limited to specific geographic regions
- Basic analytics dashboard
- No mobile application (web-only)

7.2 Future Enhancements

While Instant Dispatch is committed to continuous improvement, planned features include:

- Mobile Applications: Native iOS and Android apps
- Advanced Payment Options: Multiple payment gateways, wallet integration
- AI-based Route Optimization: Smart routing for faster deliveries
- Multi-language Support: Support for regional languages

- Advanced Analytics: Detailed reports and insights
- Rating System Enhancement: Detailed feedback mechanism
- Scheduled Deliveries: Book deliveries in advance
- Bulk Booking: Multiple deliveries at once
- Loyalty Programs: Rewards for frequent users
- Live Chat Support: Real-time customer support
- Push Notifications: Mobile and web push notifications
- Document Verification: Automated rider verification
- Insurance Integration: Package insurance options

Bibliography

1. <https://react.dev/learn/>
2. <https://nodejs.org/en/docs/>
3. <https://expressjs.com/>
4. <https://mongoosejs.com/docs/>
5. <https://socket.io/docs/>
6. <https://redux-toolkit.js.org/>
7. <https://stackoverflow.com/>
8. <https://github.com/>
9. <https://tailwindcss.com/docs>
10. <https://www.mongodb.com/docs/>

