Technician Service Assigning System

A Project Report for Industrial Training and Internship

submitted by

Ankit Das
Aditya Saha
Antara Biswas
Abhishek Datta
Animesh Ghosh

In the partial fulfillment of the award of the degree of

BCA & B.TECH

In the department of

CSE

Of

JIS UNIVERSITY

At

Ardent Computech Pvt. Ltd.

CERTIFICATE FROM SUPERVISOR

This is to certify that **"Aditya Saha (23CS2061006), Ankit Das (23CS2061022), Antara Biswas (23CS2061027), Animesh Ghosh (23CS2061015),** and **Abhishek Datta (23CS201103)"** have completed the project titled **"Technician Service Assigning System"** under my supervision during the period from **"16-06-2025"** to **"05-07-2025"**, which is in partial fulfillment of requirements for the award of the **BCA & B.TECH** degree and submitted to the Department of **"CSE"** of **"JIS UNIVERSITY"**.

_____

Signature of the Supervisor

Date(DD/MM/YYYY):

Name of the Project Supervisor:

BONA FIDE CERTIFICATE

Certified that this project work was carried out under my supervision

"Technician Service Assigning System" is the bona fide work of

Name of the student: Ankit Das ***Signature:*** Ankit Das

Name of the student: Aditya Saha ***Signature:*** Aditya Saha

Name of the student: Antara Biswas ***Signature:*** Antara Biswas

Name of the student: Animesh Ghosh ***Signature:*** Animesh Ghosh

Name of the student: Abhishek Datta ***Signature:*** Abhishek Datta

SIGNATURE :

Name :

PROJECT MENTOR:

SIGNATURE :

Name:

EXAMINERS:

Ardent Original Seal

ACKNOWLEDGEMENT

The achievement that is associated with the successful completion of any task would be incomplete without mentioning the names of those people whose endless cooperation made it possible. Their constant guidance and encouragement made all our efforts successful.

We take this opportunity to express our deep gratitude towards our project mentor, ***Subhojit Santra,*** for giving such valuable suggestions, guidance and encouragement during the development of this project work.

Last but not the least, we are grateful to all the faculty members of **Ardent Computech Pvt. Ltd.** for their support.

CONTENT PAGE

1. COMPANY PROFILE

ARDENT (Ardent Computech Pvt. Ltd.), formerly known as Ardent Computech Private Limited, is an **ISO 9001:2015** certified Software Development and Training Company based in India. Operating independently since 2003, the organisation has recently undergone a strategic merger with ARDENT Technologies, enhancing its global outreach and service offerings.

ARDENT Technologies

ARDENT Technologies delivers high-end IT services across the UK, USA, Canada, and India. Its core competencies lie in the development of customised application software, encompassing end-to-end solutions including system analysis, design, development, implementation, and training. The company also provides expert consultancy and electronic security solutions. Its clientele spans educational institutions, entertainment companies, resorts, theme parks, the service industry, telecom operators, media, and diverse business sectors.

ARDENT Collaborations

ARDENT Collaborations, the Research, Training, and Development division of ARDENT (Ardent Computech Pvt. Ltd.), offers professional IT-enabled services and industrial training programs. These are tailored for freshers and professionals from B.Tech, M.Tech, MBA, MCA, BCA, and MSc backgrounds. ARDENT (Ardent Computech Pvt. Ltd.) provides Summer Training, Winter Training, and Industrial Training to eligible candidates. High-performing students may qualify for stipends, scholarships, and additional benefits based on performance and mentor recommendations.

Associations and Accreditations

ARDENT (Ardent Computech Pvt. Ltd.) is affiliated with the National Council of Vocational Training (NCVT) under the Directorate General of Employment & Training (DGET), Ministry of Labour & Employment, Government of India. The institution upholds strict quality standards under ISO 9001:2015 certification and is dedicated to bridging the gap between academic knowledge and industry skills through innovative training programs.

2. INTRODUCTION

In today's technology-driven world, the demand for efficient and responsive service management has significantly increased across industries. Traditional methods of assigning and tracking technical service requests are often time-consuming, disorganised, and lack transparency. To address these challenges, our project, the Technician Service Assigning System, has been developed as a comprehensive and automated solution to streamline the process of technician deployment and service tracking.

Powered by the MERN stack, our system provides a user-friendly and role-based interface for customers, technicians, and administrators. Customers can raise service requests, technicians can view and update task statuses, and administrators can assign jobs and monitor real-time progress. The system ensures prompt response, efficient scheduling, and effective communication between all stakeholders, reducing delays and human errors in service delivery.

By digitising and automating the entire technician assignment lifecycle, this project enhances operational efficiency, improves customer satisfaction, and ensures optimal use of technical manpower. It is an ideal solution for organisations looking to modernise their field service operations and build a reliable support system tailored to real-world service demands.

2A. OBJECTIVE:

The primary objective of the Technician Service Assigning System is to modernize the traditional manual method of assigning service requests by providing a responsive, role-based, and automated platform. This system aims to ensure a seamless flow between service request generation, technician assignment, progress tracking, and final resolution — all within a centralized digital environment.

The application seeks to optimize technician resource allocation, minimize service delays, and improve user satisfaction by enabling real-time monitoring and efficient communication between users, technicians, and administrators. By bridging the gap between service demand and delivery, the system helps organizations manage their field operations in a structured and accountable manner.

Ultimately, the Technician Service Assigning System empowers organizations and end-users by offering an intelligent and transparent service management experience—anytime, anywhere.

2B. SCOPE:

Our project focuses on creating a web-based application that simplifies and automates the technician service assignment workflow for organisations across various industries.

1. **User Registration & Login**:
   Secure sign-up/login functionality for users, technicians, and admins.

2. **Service Request Management**:
   An interface for users to raise service requests and track progress.

3. **Technician Dashboard**:
   A dedicated panel for technicians to view assigned jobs, update status, and mark tasks as complete.

4. **Admin Panel**:
   Tools for the admin to manage user roles, assign technicians to requests, and generate performance reports.

5. **Multiplatform Access**:
   Accessible from desktops, laptops, and modern mobile browsers.

3. SYSTEM ANALYSIS

3A. IDENTIFICATION OF NEED:

System analysis is a crucial phase in the development of our project, Technician Service Assigning System, aimed at building an efficient, reliable, and user-friendly platform to manage technician dispatch and service tracking.

With the rapid digitalization of service-based industries and the growing expectations for prompt support, there is an urgent need for a structured and real-time platform that streamlines the technician assignment process. Traditional models suffer from inefficiencies, manual errors, and delayed response times. The Technician Service Assigning System is designed to resolve these pain points by introducing automation, centralized control, and role-based access for administrators, technicians, and customers.

Key Needs Identified:

1. Accessibility to Service Management:

- Customers require a transparent and easy-to-use portal to raise service requests.
- Businesses need centralized access to technician availability, status, and task progress from anywhere.

2. Interactive and Real-Time Assignment:

- Manual assignment leads to delays and miscommunication.
- There is a need for real-time technician-job matching and status tracking to ensure timely responses.

3. Efficient Task Coordination and Reporting:

- Admins must be able to assign, reassign, and monitor jobs effectively.
- Digital logs and automated updates reduce errors and help track service timelines and technician performance.

3B. FEASIBILITY STUDY:

The feasibility study of the **Technician Service Assigning System** indicates that the project is viable and holds promise across several dimensions of implementation.

From a **technical perspective**, the application can be developed using the **MERN stack**, which is widely supported and scalable. The backend with Node.js and MongoDB offers efficient data handling for task records and user roles, while React.js ensures a responsive frontend interface.

Economically, the system can reduce operational costs associated with manual dispatching and help scale service operations without proportionate increases in manpower. For businesses that rely on frequent service calls, it enhances ROI through faster resolutions and improved customer retention.

Operationally, the interface is designed to be intuitive, requiring minimal training for users. The role-based access ensures security and reduces complexity.

Legally, the system is designed to comply with general data privacy standards. Since it deals with service data and user roles, sensitive customer data can be protected through standard authentication and authorization protocols.

The estimated development and testing cycle for the working prototype is around **6 to 8 weeks**, depending on module complexity. Overall, the project is both **achievable and impactful**, offering a strong technological foundation for field service automation in the digital era.

3C. WORKFLOW:

This document plays a vital role in the **s**oftware **d**evelopment **l**ife **c**ycle **(SDLC)** as it describes the complete requirements of the system. It is meant for use by the developers and will be the basis during the testing phase. Any changes made to the requirements in the future will have to go through a formal change approval process.

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin, and there is no overlapping in the phases.

The waterfall model is the earliest SDLC approach that was used for software development. The waterfall Model illustrates the software development process in a linear sequential flow; hence, it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In the waterfall model, phases do not overlap.

- **Waterfall Model Design:**

The waterfall approach was the first SDLC Model to be used widely in Software Engineering to ensure the success of the project. In the "Waterfall Model" approach, the whole process of software development is divided into separate phases. In the Waterfall model, typically, the Outcome of one phase acts as the input for the next phase sequentially.

- **Iterative Waterfall Model Design:**

  Definition: The Iterative Waterfall Model is a variation of the traditional Waterfall model, which is a linear and sequential software development methodology. In the Iterative Waterfall Model, the development process is divided into small, manageable cycles, allowing for the revisiting and refinement of phases before progressing to the next stage. It combines the systematic structure of the Waterfall model with the flexibility of iterative development.

The sequential phases in the Iterative Waterfall model are:

➢ **Requirement Gathering and Analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a system requirements specification (SRS) document.

➢ **System Design:** The requirement specifications from the first phase are studied in this phase, and the system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining the overall system architecture.

➢ **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

➢ **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing each unit. Post-integration, the entire system is tested for any faults and failures.

➢ **Deployment of the system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.

➢ **Maintenance:** Some issues come up in the client environment. To fix those issues, patches are released. Also, to enhance the product, some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in progress and are seen as flowing steadily downwards (like a waterfall) through the phases. The next phase starts only after the defined set of goals is achieved for the previous phase, and it is signed off, so the name is “Iterative Waterfall Model”. In this model, phases do not overlap.

➔ **Advantages:**

◆ **Flexibility:** Iterations permit adjustments based on feedback.

◆ **Early Delivery:** Partial systems can be delivered incrementally.

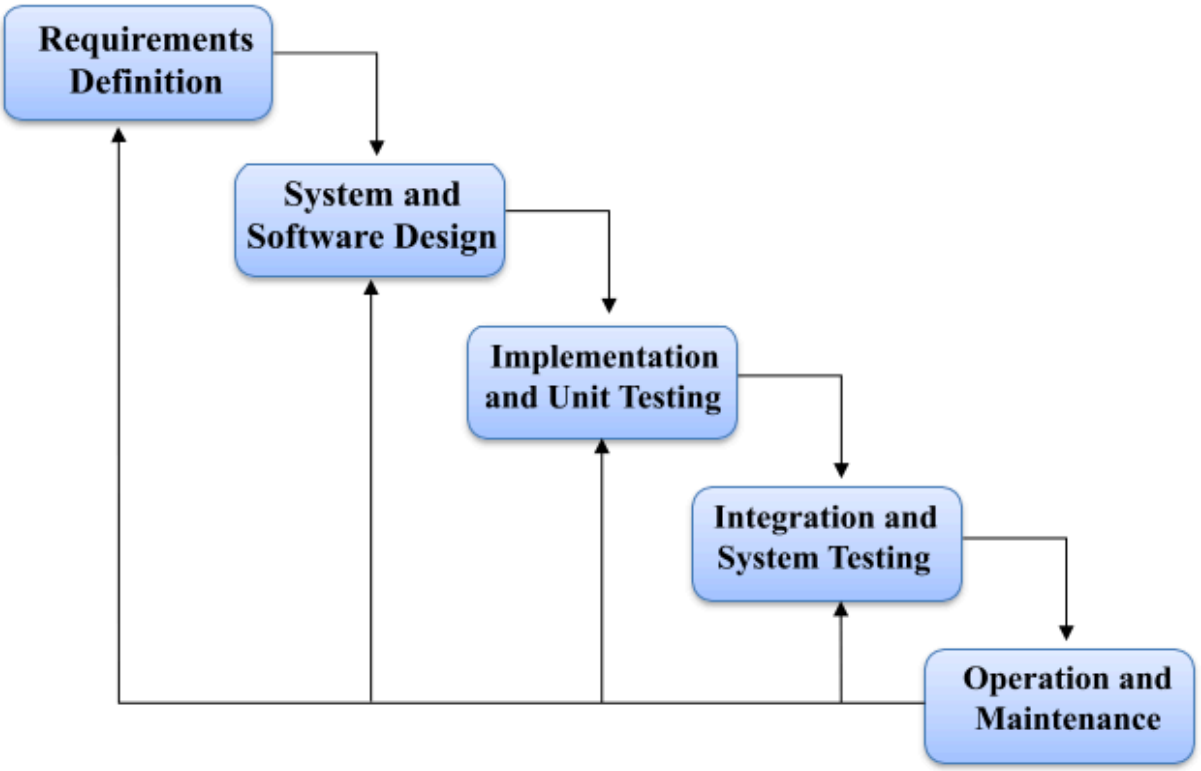
◆ **Risk Management:** Identifying and addressing issues early in the process.

➔ **Disadvantages:**

◆ **Increased Complexity:** The iterative nature can make the process more complex.

◆ **Potential for Scope Creep:** Frequent iterations may lead to scope changes.

◆ **Resource Intensive:** Continuous revisiting of phases may demand more resources.



➔ **Applications:**

The Iterative Waterfall Model is suitable for projects with evolving or unclear requirements. It is commonly used in software development projects where regular feedback and refinement are essential.

Additionally, it is applicable in scenarios where partial system delivery is beneficial, allowing stakeholders to assess progress and make adjustments.

3D. STUDY OF THE SYSTEM:

Modules: The modules used in this software are as follows:

● **Register:**

1. **User Register**:
   Customers can register on the platform to raise service requests and track their status.

2. **Technician Register**:
   Technicians register to receive service assignments based on availability and specialisation.

3. **Admin Register**:
   The administrator account is created to manage users, assign tasks, and oversee the entire workflow.

● **Login:**

1. **User Login**:
   Registered users log in to raise service tickets, view the progress of their requests, and submit feedback.

2. **Technician Login**:
   Technicians log in to view assigned jobs, update status, and complete tasks.

3. **Admin Login**:
   Admin logs in to manage service requests, assign technicians, and monitor overall system activities.

● **Home:**

This is the landing page where users can access general information, system announcements, or recent updates. It may also showcase recent technician ratings or feedback highlights.

● **Service Requests:**

1. **User Interface**:
   Allows users to create a new service request, provide details (issue description, location, preferred time), and track the status of previous requests.

2. **Admin Interface**:
   Admin can view all incoming requests, assign them to available technicians, and update status manually if needed.

● **Task Management:**

1. **Technician Interface**:
   Displays the list of tasks assigned to the logged-in technician. Allows updating task progress (e.g., "In Progress", "Completed") and adding service notes.

2. **Admin Interface**:
   Enables the admin to monitor technician activity, reassign tasks, and generate service logs.

● **About:**

This page provides background information about the project, its purpose, and the development team involved.

- **Account:**

1. **User Interface**:

   a) **My Profile**: Displays user details such as name, email, contact info, and request history.

   b) **Dashboard**: Summarizes all past and ongoing service requests raised by the user.

2. **Technician Interface**:

   a) **Technician Profile**: Shows technician details, assigned jobs, and performance summary.

   b) **Technician Dashboard**: Allows technicians to view and manage tasks and timelines.

3. **Admin Interface**:

   a) **Admin Profile**: Contains admin details and access logs.

   b) **Admin Dashboard**: Centralized panel to manage users, technicians, service requests, and overall system performance.

3E. INPUT AND OUTPUT:

The main inputs, outputs, and the major functional interactions in the **Technician Service Assigning System** are as follows:

INPUT:

1. **User Login**:
   Users (customers, technicians, and admins) enter their credentials on the login page to access the system.

2. **Service Request Form**:
   Customers input issue details, preferred service time, and contact information to raise a service request.

3. **Technician Task Updates**:
   Technicians provide task status updates such as “In Progress” or “Completed” along with optional remarks.

4. **Admin Inputs**:
   The admin adds new technician profiles, assigns jobs to available technicians, and updates overall service records.

OUTPUT:

1. **Customer Dashboard**:
   Displays the list of raised service requests, their current status (Pending, In Progress, Completed), and any admin or technician updates.

2. **Technician Dashboard**:
   Shows the list of tasks assigned to the technician along with customer details, job descriptions, and deadlines.

3. **Admin Panel**:
   Provides access to a centralized system displaying all users, service requests, technician availability, and real-time updates for efficient task assignment and management.

3F. SOFTWARE REQUIREMENT SPECIFICATIONS

The Software Requirements Specification (SRS) provides a structured and comprehensive overview of the **Technician Service Assigning System**. It defines both the functional and non-functional requirements of the system to be developed. This document serves as a foundational reference for developers, testers, and stakeholders, ensuring that the system meets expectations and performs as intended.

The SRS is prepared after thorough communication with the project team and understanding of end-user needs. It outlines what the system should do, how it should behave, and the environment in which it will operate.

Developer Responsibilities:

- To design and develop the system according to the specifications outlined in the SRS.
- To demonstrate the system's functionality and deploy it after successful acceptance testing.
- To submit detailed user documentation, including usage manuals and interface explanations, for smooth system operation.

Functional Requirements:

A. User Registration and Authentication

1. Users (customers, technicians, admins) should be able to securely create and manage accounts.
2. The system must authenticate users and maintain secure login sessions.

B. Service Request Management

1. Registered users should be able to raise new service requests with detailed descriptions.
2. Admins should have the ability to assign these requests to available technicians.

C. Technician Dashboard

1. Technicians should be able to view their assigned jobs and update task statuses.

2. Technicians should be able to view job details, including client location and description.

D. Admin Controls

1. Admin should be able to add/edit/delete users, assign jobs, and monitor technician activity.

2. Admin should be able to generate reports on completed and pending service requests.

Hardware Requirements:

1. Intel Core i3 Processor (or higher)

2. Minimum 8 GB RAM

3. SSD Drive (recommended for faster performance)

Software Requirements:

1. Windows 11 Operating System

2. Visual Studio Code (IDE for development)

3. MongoDB Atlas (Cloud-based database)

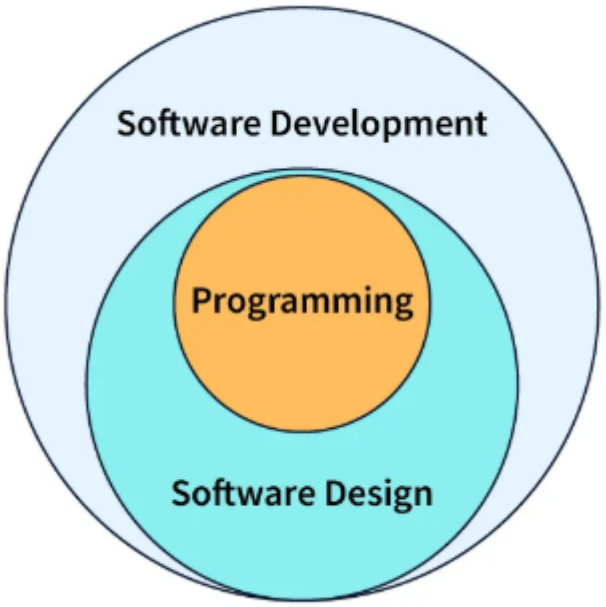
4. Node.js & npm (Backend environment)

5. Postman (for API testing and validation)

6. Google Chrome / Firefox (Browser for frontend testing)

3G. SOFTWARE ENGINEERING PARADIGM APPLIED

Software engineering paradigms refer to the structured methods and steps used throughout the software development life cycle. These paradigms guide how the software is conceived, designed, implemented, and maintained. The choice of paradigm plays a critical role in ensuring the software meets both user expectations and system reliability.



In the case of the **Technician Service Assigning System**, the development process follows a layered approach:

- The **Programming Paradigm** focuses on implementation techniques using JavaScript in the MERN stack.
- The **Software Design Paradigm** outlines how different modules (login, dashboard, request management) are structured and interact.
- The **Software Development Paradigm** governs the full project workflow using the Iterative Waterfall Model.

Together, these paradigms support a disciplined, structured, and iterative approach to development, ensuring both clarity and efficiency.

Reliability Considerations:

Reliability in software development is assessed at two levels:

1. **Requirement Reliability**:
   Ensuring that the right problem is being solved through accurate and complete requirement gathering.

2. **Delivery Reliability**:
   Ensuring the final software product performs as expected under real-world conditions.

To maintain high system reliability, the following approaches were adopted:

Three Approaches to Reliability:

1. **Error Avoidance**:
   Implementing coding standards, input validations, and well-structured module separation to prevent errors at the source.

2. **Error Detection and Correction**:
   Logging unexpected behaviors, using try-catch blocks in backend logic, and providing fallback messages to users when something goes wrong.

3. **Error Tolerance**:
   Allowing the system to continue functioning even in partial failure (e.g., if the technician assignment fails, the request still logs and alerts the admin).

4. SYSTEM DESIGN

4A. DATA FLOW DIAGRAM:

A **Data Flow Diagram (DFD)** is a graphical tool used to represent the flow of data through an information system, capturing how input is transformed into output through various processes. It helps in modeling the process-oriented aspects of the system while abstracting away implementation details.

For the **Technician Service Assigning System**, the DFD serves as an essential component of structured system design. It illustrates:

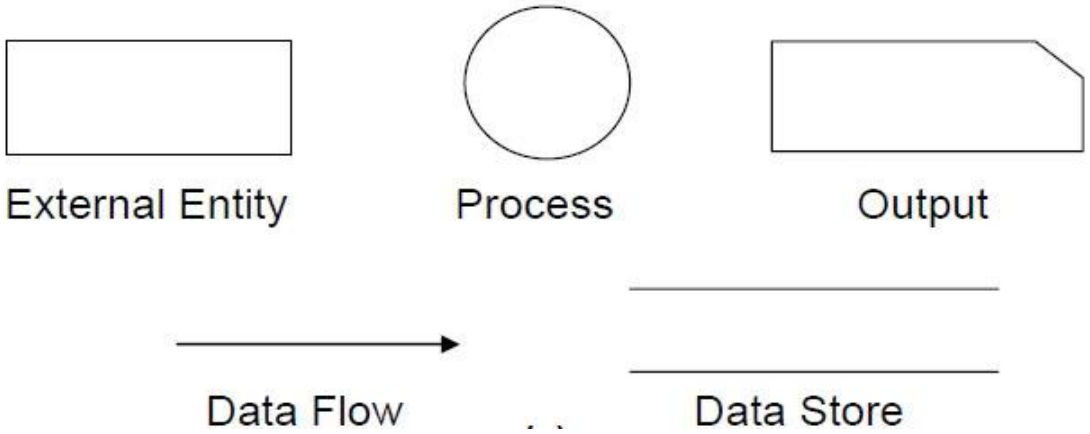
- The **sources and destinations** of data,
- The **processes** that transform data,
- The **data stores** used within the system,
- And how data moves between users, technicians, admins, and the database.

Purpose of DFD in This Project:

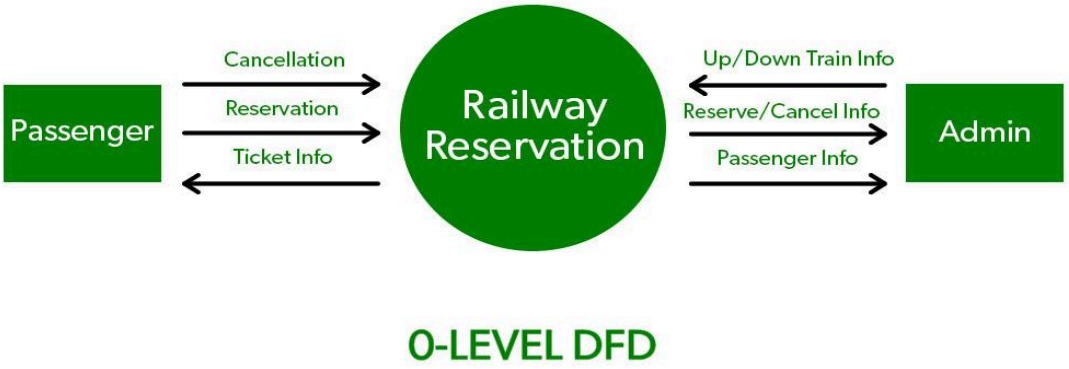
- To provide a **clear visualization** of how data is processed in the system.
- To serve as a **preliminary model** for further system design and development.
- To **identify bottlenecks** or redundancies early in the planning phase.
- To help stakeholders understand the functionality and workflow without needing technical knowledge of the underlying code.

How any system is developed can be determined through a data flow diagram model. In the course of developing a set of leveled data flow diagrams, the analyst/designer is forced to address how the system may be decomposed into component sub-systems and to identify the transaction data in the data model. Data flow diagrams can be used in both the **Analysis** and **Design** phases of the **SDLC.** There are different notations to draw data flow diagrams. Defining different visual representations for processes, data stores, data flow, and external entities.

DFD Notation:



DFD Example:



0-LEVEL DFD

Steps to Construct a Data Flow Diagram:

Four Steps are generally used to construct a DFD.

- The process should be named and referred to for easy reference. Each name should be representative of the reference.
- The direction of flow is from top to bottom and from left to right.
- When a process is distributed into lower-level details, they are numbered.
- The names of data stores, sources, and destinations are written in capital letters.

Rules for constructing a Data Flow Diagram:

- ❖ Arrows should not cross each other.
- ❖ Squares, Circles, and Files must bear a name.
- ❖ Decomposed data flow squares and circles can have the same names.
- ❖ Draw all data flow around the outside of the diagram.

DFD Context Level (Level 0):

The **Context-Level Data Flow Diagram** (Level 0) provides a high-level overview of the *Technician Service Assigning System* by representing the entire system as a **single process**. It outlines how the system interacts with its **external entities** and the major **data flows** involved.

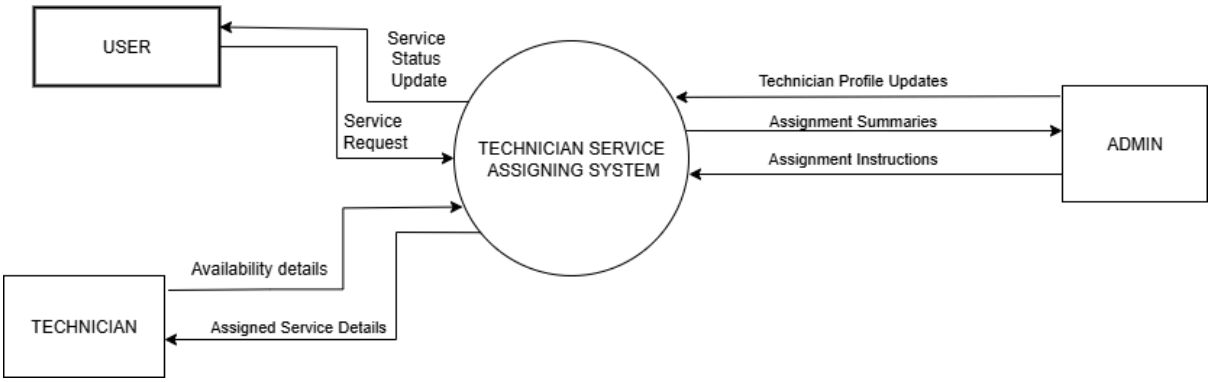
External Entities:

- **User (Customer):**
  Initiates service requests and receives service status updates from the system.
- **Technician:**
  Receives assigned service tasks and provides availability and completion updates.
- **Admin:**
  Manages technician profiles, assigns service tasks, and monitors system operations through summaries and reports.

Major Data Flows:

- **Service Request Submission** – from the User to the system.
- **Job Assignment** – from the Admin to the system and then to the Technician.
- **Task Updates** – status and completion information between the Technician and the system, which is then relayed to the User.
- **Profile Information** – Technician profile management handled by Admin.
- **System Reports** – assignment summaries and performance feedback are sent to the Admin for monitoring and improvement.

LEVEL 0 DFD OR CONTEXT DIAGRAM:



DFD Level 1:

The **Level 1 Data Flow Diagram** expands the central process of the *Technician Service Assigning System* into multiple subprocesses to illustrate the internal workings of the system. It shows how data flows between modules, external entities, and internal data stores, offering a detailed view of the system's functional decomposition.

Core Subprocesses:

- **1.0 User Management**
  Handles user information, login/authentication (if any), and profile data required for initiating requests.

- **2.0 Service Request Handling**
  Manages the intake of service requests submitted by users and stores relevant details for assignment.

- **3.0 Technician Assignment**
  Retrieves available technician data and assigns tasks based on system logic or admin input.

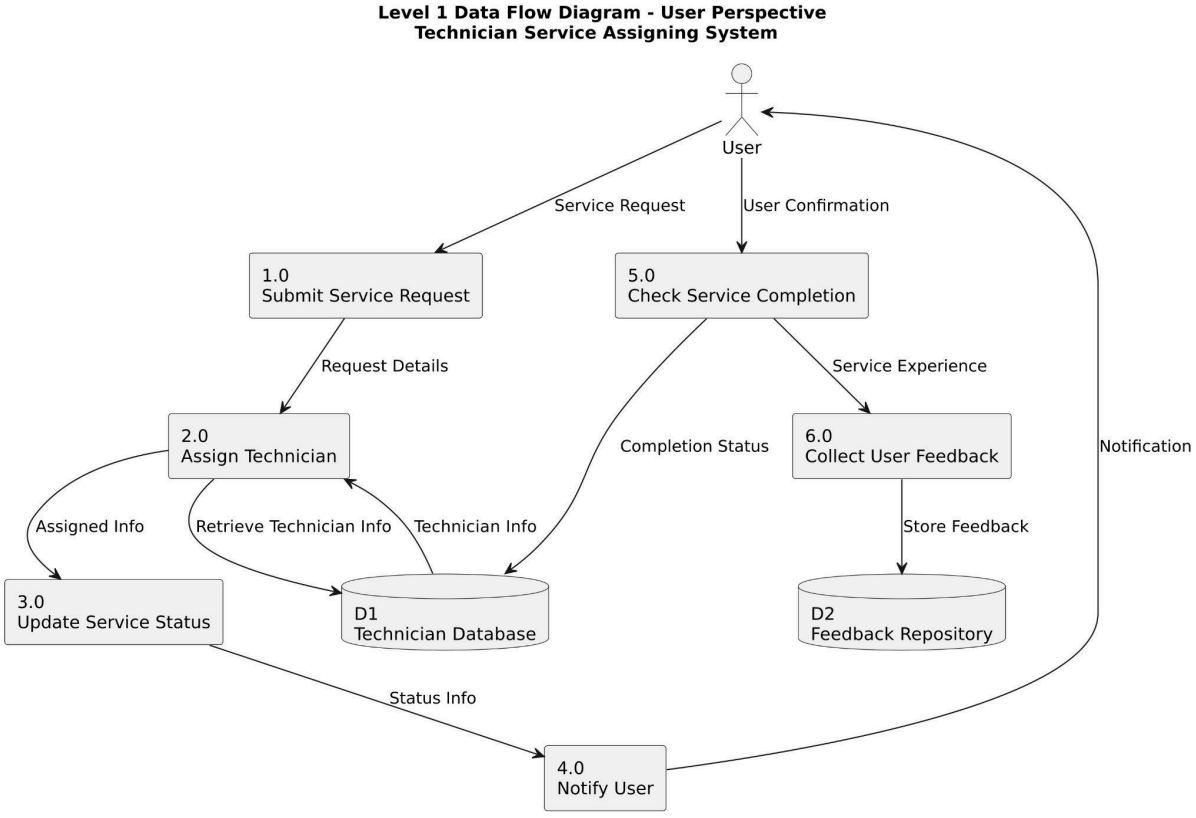
- **4.0 Task Tracking and Updates**
  Monitors progress of ongoing service tasks, updates task status, and provides real-time notifications to users.

- **5.0 Reporting and Administration**
  Generates summaries, performance data, and administrative insights for backend monitoring and improvements.

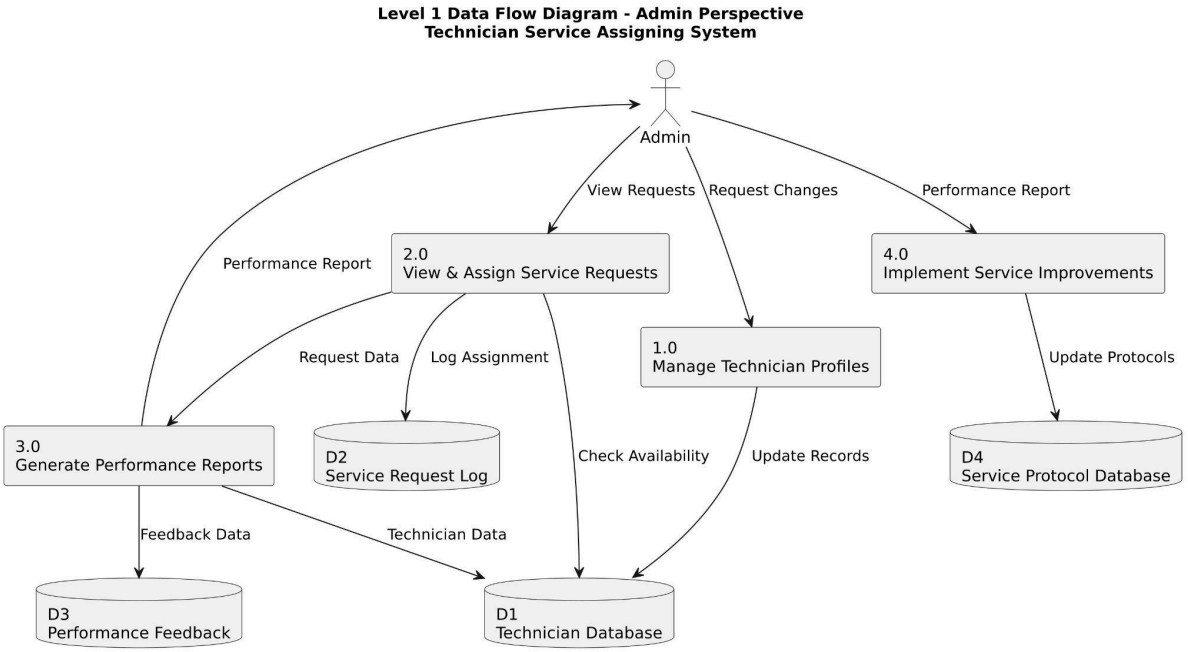
Internal Data Stores:

- **User Database** – stores user account and profile details.
- **Request/Job Data** – holds service request and job-specific information.
- **Technician Profiles** – contains technician availability, assignments, and performance history.
- **Task Status Logs** – records the progress, completion, and updates of tasks over time.

LEVEL 1 DFD OR CONTEXT DIAGRAM: USER PERSPECTIVE



LEVEL 1 DFD OR CONTEXT DIAGRAM: ADMIN PERSPECTIVE



Summary:

Data Flow Diagrams (DFDs) are a critical part of the ‘structured systems’ analysis and design method (SSADM). They assist developers and stakeholders in visualizing system operations and understanding **what** data is involved, **how** it's processed, and **where** it's stored. The DFD serves as a communication bridge between technical and non-technical stakeholders throughout all stages of system evolution.

4B. SEQUENCE DIAGRAM:

A **sequence diagram** is a type of interaction diagram that models the flow of logic within a system in a visual, time-ordered manner. It captures how objects or components interact with one another and the sequence in which these interactions occur.

In a sequence diagram, **vertical lines** (lifelines) represent different system entities that exist simultaneously, while **horizontal arrows** indicate the messages exchanged between them in chronological order. This provides a clear view of how functionality unfolds during system execution.

Sequence diagrams are particularly useful for modeling **use case realizations**, allowing developers and stakeholders to visualize the dynamic behavior of a system. They are also known as **event diagrams** or **event scenarios**.

Purpose of This Project

The **Sequence Diagram** for the Technician Service Assigning System visually represents the complete lifecycle of a service request — from initiation by the user to administrative review and process refinement. The diagram is organized into **three main phases**, each capturing the essential interactions between system actors and components.

1. Service Request Phase

- The **User** initiates the process by submitting a service request.
- The **System** captures the request, retrieves technician data, and assigns an available technician based on criteria or admin input.

2. Service Execution Phase

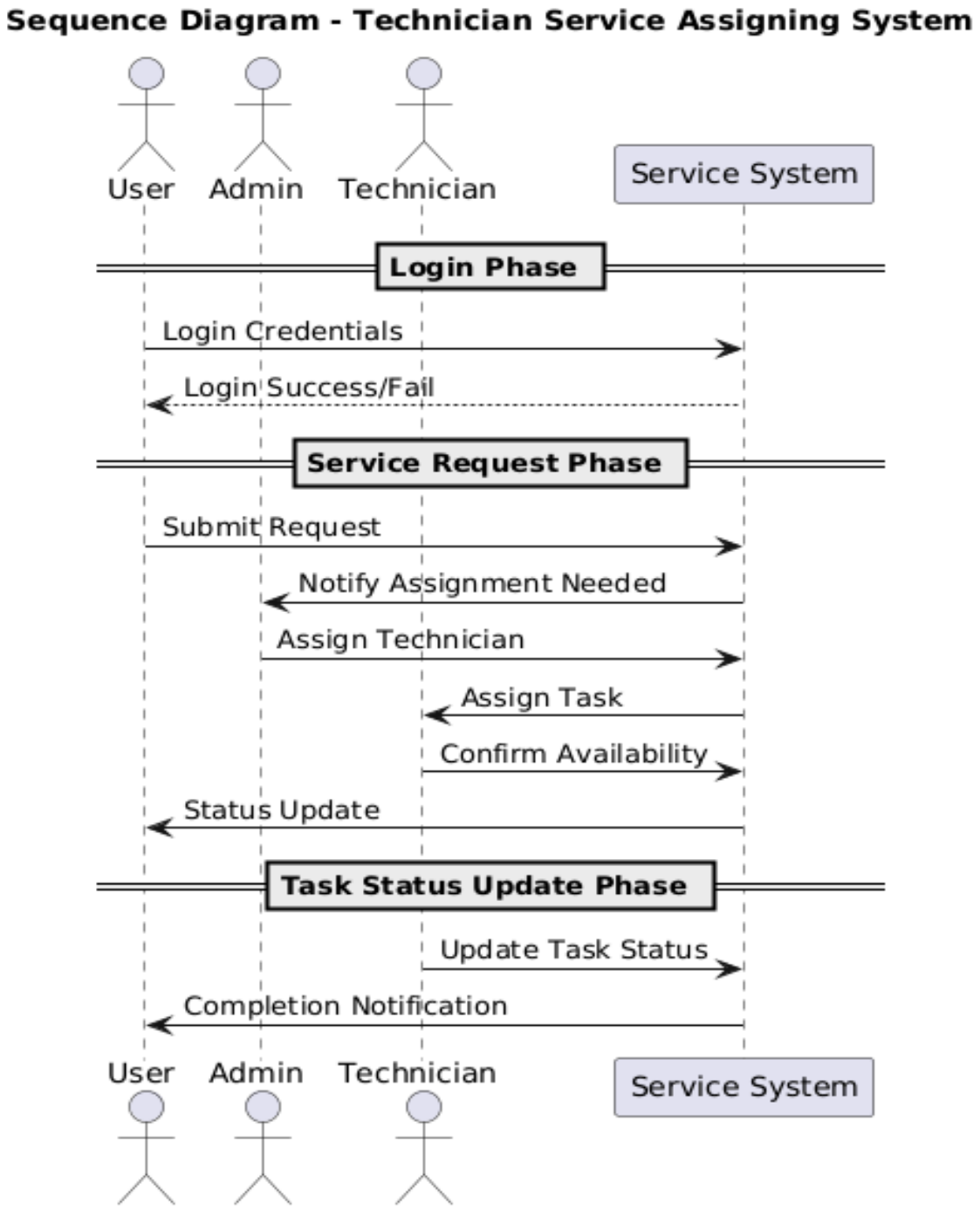
- The **Technician** performs the assigned task and updates the service status (e.g., In Progress, Completed).
- The **System** logs each update in the task tracking module.
- The **User** receives notifications and is given the option to submit feedback upon completion.

3. Admin Review & Improvement Phase

- The **Admin** accesses performance reports, technician logs, and user feedback.
- Based on this analysis, the admin may revise technician profiles or update service protocols to improve overall efficiency and quality.

Key Elements Represented in the Diagram:

- **Lifelines:** Represent major actors (User, Technician, Admin) and system components (Service System, Technician Database, Feedback Repository, etc.).
- **Messages:** Illustrated with directional arrows to show the sequence and direction of interaction between participants.
- **Phases:** Communication is logically grouped to reflect real-world operation segments, from request to review.
- **Databases:** Direct interaction with internal storage components such as the Technician Database, Service Logs, Feedback Repository, and Service Protocol Database is shown explicitly to highlight system dependencies.



This diagram effectively demonstrates the **end-to-end functionality** and **real-time interaction** among the components involved in the Technician Service Assigning System.

4C. USE CASE DIAGRAM:

A **Use Case Diagram** is a graphical representation that illustrates the interaction between the **users (actors)** and the **functionalities (use cases)** of a system. It models the **dynamic behavior** of the system from the user's perspective, highlighting how external and internal agents interact with the system to achieve specific goals.

1. Overview

The Use Case Diagram offers a functional outline of the Technician Service Assigning System. It visually represents how different actors interact with the system's core functionalities. The diagram focuses on user registration, request handling, technician assignment, task completion, and admin oversight.

2. Key Actors

- **User**: A customer who interacts with the system to register, log in, and raise service requests.
- **Admin**: The system manager is responsible for assigning technicians, marking tasks as complete, and monitoring service workflows.

3. Core Use Cases

- **Register**: Allows the user to create an account in the system.
- **Login**: Enables both the user and the admin to access the system securely.
- **Raise Service Request**: User submits a request for technician assistance.
- **Assign Technician**: Admin assigns a suitable technician to the raised request.
- **Mark Task Complete**: Admin verifies task completion and updates the system.
- **View Assignment Summary**: Admin checks the current status of technician assignments.

4. Relationships

- The associations between actors and use cases are shown using simple lines.
- Each connection indicates direct interaction between the actor and the functionality.
- No include or extend relationships are used, keeping the system logic simple and direct.

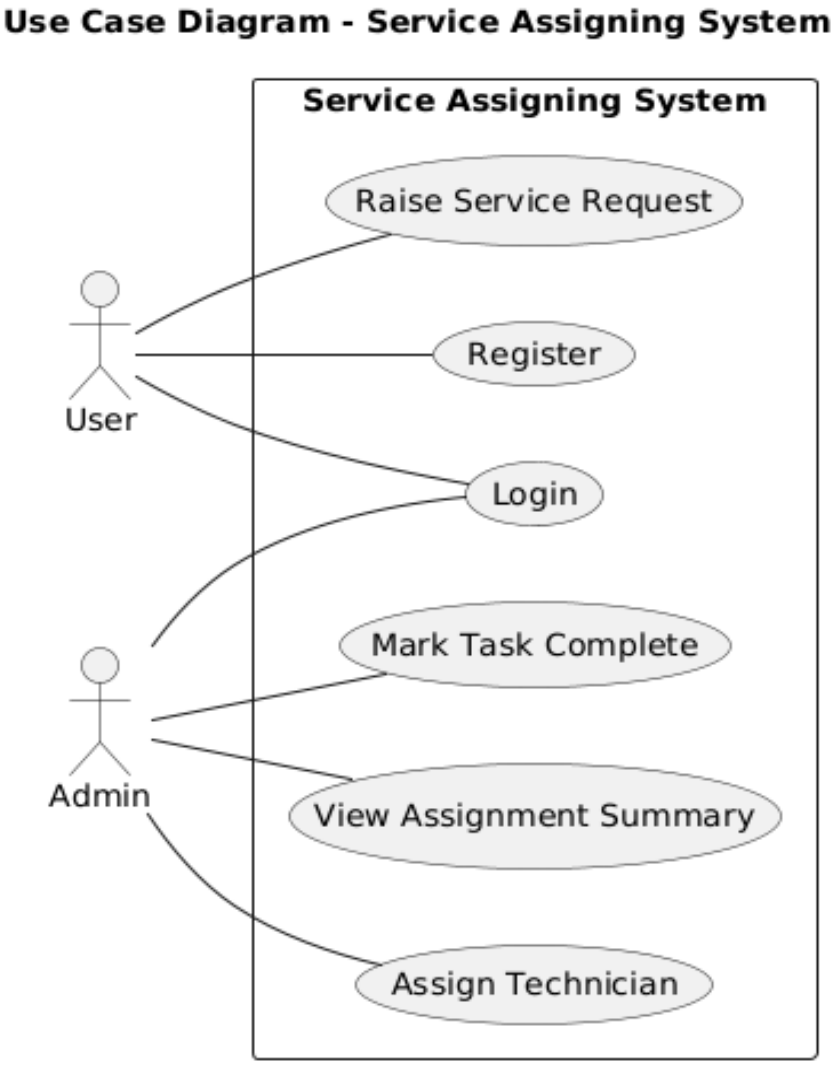
5. Purpose of the Use Case Diagram

- To capture high-level functional requirements of the system.
- To provide a clear external view of system behavior without detailing internal logic.
- To support communication between stakeholders, developers, and analysts.
- To serve as a foundation for further system design, including sequence and class diagrams.

6. Guidelines Followed in the Diagram

- Each use case is named clearly and describes a complete user intention.
- Actors are labeled based on their functional role.
- Complexity is avoided to ensure readability and relevance.
- System boundaries are marked by enclosing all use cases in a system box.

7.



8. Conclusion

This use case diagram is designed to match the current simplified implementation of the Technician Service Assigning System. It includes only essential functions and avoids unnecessary features. It highlights the main actor-system interactions and serves as a practical tool for guiding the next stages of development.

4D. SCHEMA OR CLASS DIAGRAM:

A **Class Diagram** is a UML static structure diagram that illustrates the structure of a system by showing its **classes**, their **attributes**, **methods (functions)**, and the **relationships** among them.

In the context of the **Technician Service Assigning System**, the class diagram represents the logical blueprint of how data and operations are structured within the system.

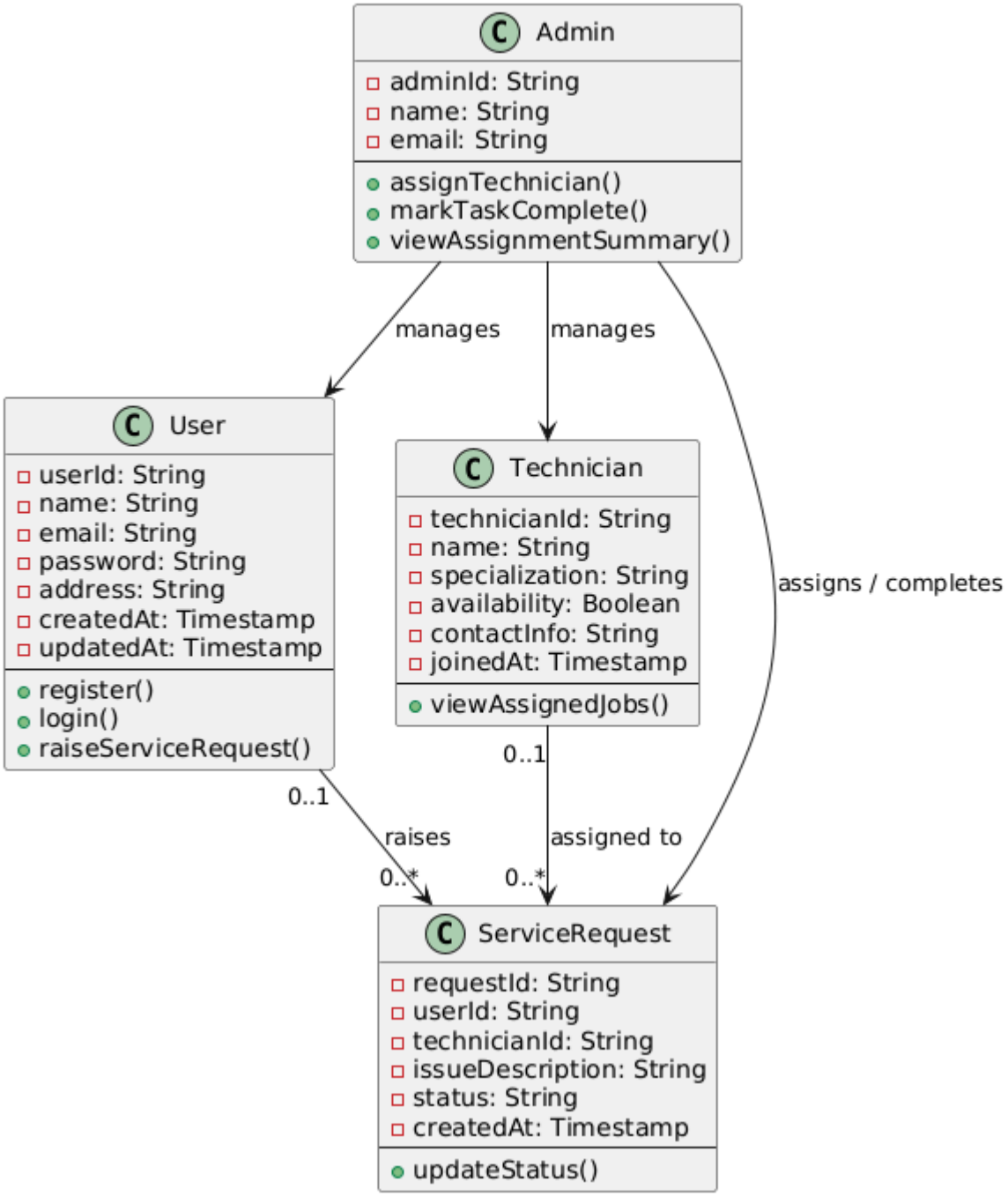
Just like a database schema defines the categories of data and their inter-relationships, a class diagram organizes system data into **object-oriented entities (classes)** and defines their **relationships (association, inheritance, dependency)**.

Each class models a real-world entity like **User**, **Technician**, **Admin**, or **ServiceRequest**, helping developers and analysts gain a clear understanding of the system's structure and logic.

The class diagram enables efficient design and development by:

- Defining **attributes** that represent stored data,
- Providing **methods** for behavior/actions,
- Establishing **relationships** like one-to-many or inheritance between classes.

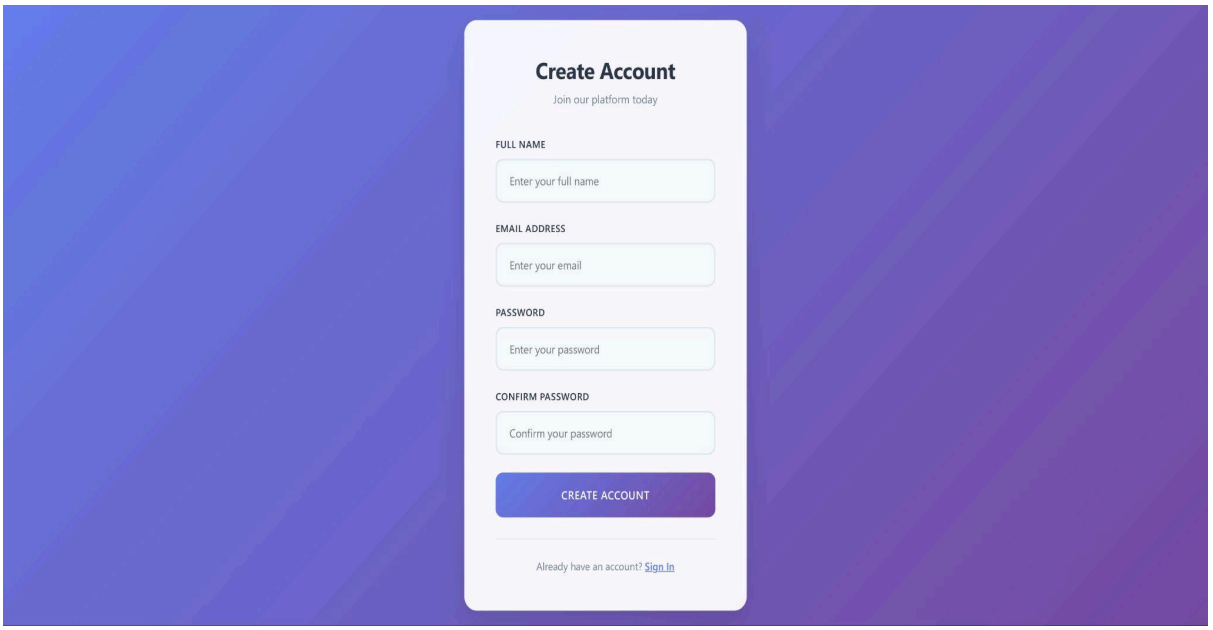
Class Diagram - Technician Service Assigning System



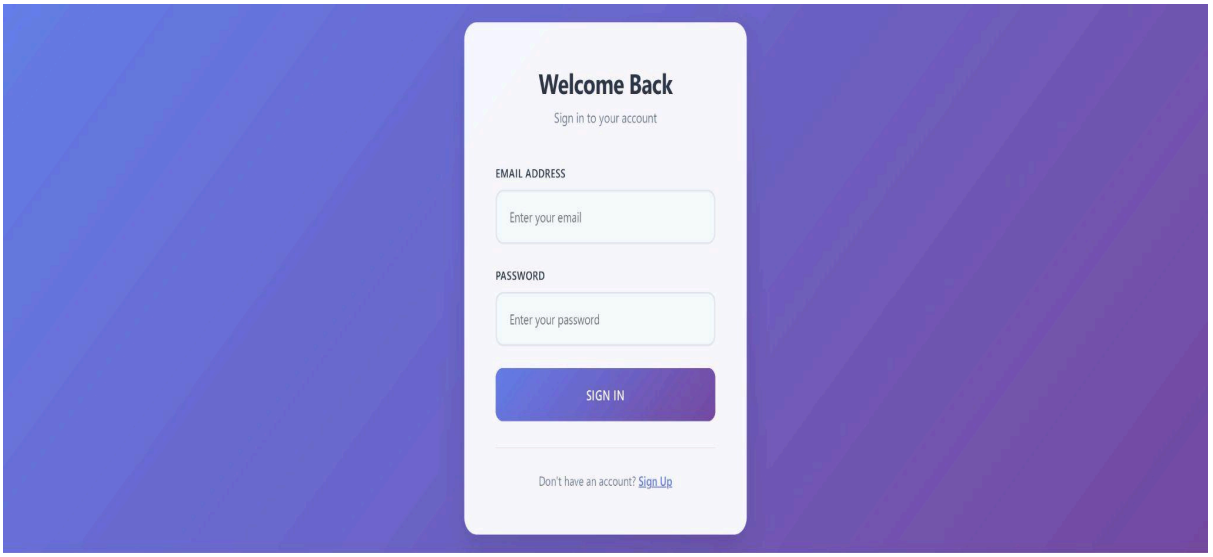
5. UI SNAPSHOT

5A. FRONTEND:

1. **A. Registration Page:**



B. Login Page:



Code:

```
import React, { useState } from 'react';
```

```
import api from '../services/api';
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
import '../styles/SignupPage.css';

const SignupPage = () => {
  const navigate = useNavigate();
  const { login } = useAuth();

  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: ''
  });

  const [errors, setErrors] = useState({});
  const [loading, setLoading] = useState(false);
  const [isLogin, setIsLogin] = useState(false);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prev) => ({
      ...prev,
      [name]: value
```

```
  }));
  if (errors[name]) {
    setErrors((prev) => ({
      ...prev,
      [name]: ''
    }));
  }
};

const validateForm = () => {
  const newErrors = {};
  if (!isLogin && !formData.name.trim()) {
    newErrors.name = 'Name is required';
  }
  if (!formData.email.trim()) {
    newErrors.email = 'Email is required';
  } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
    newErrors.email = 'Email is invalid';
  }
  if (!formData.password) {
    newErrors.password = 'Password is required';
  } else if (formData.password.length < 6) {
    newErrors.password = 'Password must be at least 6 characters';
  }
  if (!isLogin && formData.password !== formData.confirmPassword) {
```

```
    newErrors.confirmPassword = 'Passwords do not match';
  }
  return newErrors;
};

const handleSubmit = async (e) => {
  e.preventDefault();
  const formErrors = validateForm();
  if (Object.keys(formErrors).length > 0) {
    setErrors(formErrors);
    return;
  }

  setLoading(true);
  setErrors({});
  try {
    const endpoint = isLogin ? '/auth/login' : '/auth/register';
    const payload = isLogin
      ? { email: formData.email.trim(), password: formData.password }
      : {
          name: formData.name.trim(),
          email: formData.email.trim(),
          password: formData.password
        };
```

```
    console.log('🚀 Submitting payload:', payload);

    const response = await api.post(endpoint, payload);

    if (response.data.token) {
      login(response.data.token, response.data.role);

      if (response.data.role === 'admin') {
        navigate('/admin');
      } else {
        navigate('/home');
      }
    }
  } catch (error) {
    console.error('❌ API error:', error);
    if (error.response?.data?.message === 'User already exists') {
      setErrors({ email: 'Email already exists' });
    } else {
      setErrors({
        general:
          error.response?.data?.message ||
          `${isLogin ? 'Login' : 'Signup'} failed.`
      });
    }
  } finally {
```

```
    setLoading(false);
  }
};

return (
  <div className="signup-container">
    <div className="signup-card">
      <div className="signup-header">
        <h1>{isLogin ? 'Welcome Back' : 'Create Account'}</h1>
        <p>{isLogin ? 'Sign in to your account' : 'Join our platform today'}</p>
      </div>

      {errors.general && <div className="error-alert">{errors.general}</div>}

      <form onSubmit={handleSubmit} className="signup-form">
        {!isLogin && (
          <div className="form-group">
            <label htmlFor="name">Full Name</label>
            <input
              type="text"
              id="name"
              name="name"
              placeholder="Enter your full name"
              value={formData.name}
              onChange={handleChange}
```

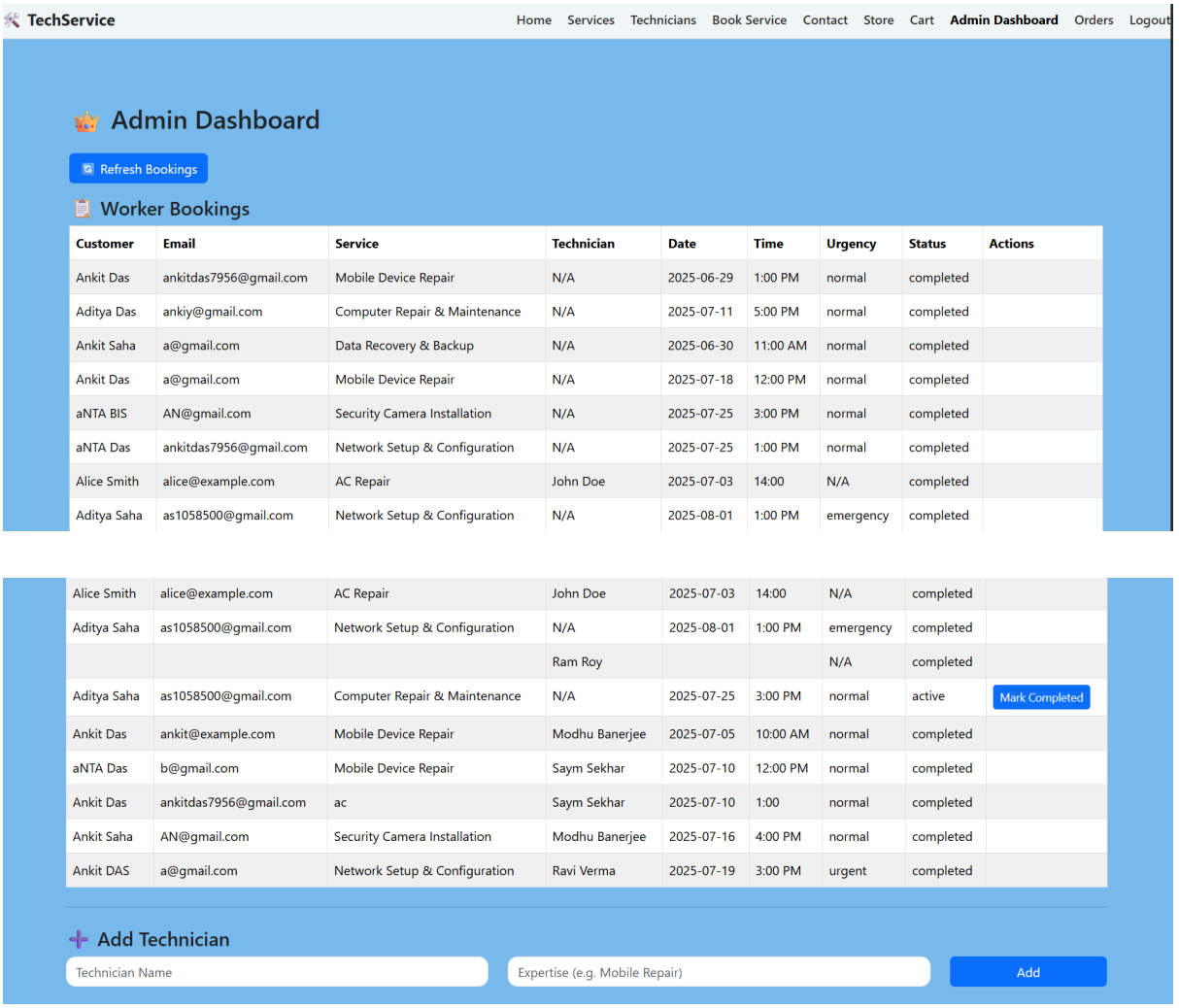
```
      className={errors.name ? 'error' : ''}
    />
    {errors.name && (
      <span className="error-text">{errors.name}</span>
    )}
  </div>
)}

<div className="form-group">
  <label htmlFor="email">Email Address</label>
  <input
    type="email"
    id="email"
    name="email"
    placeholder="Enter your email"
    value={formData.email}
    onChange={handleChange}
    className={errors.email ? 'error' : ''}
  />
  {errors.email && (
    <span className="error-text">{errors.email}</span>
  )}
</div>

<div className="form-group">
```

```
  <label htmlFor="password">Password</label>
  <input
    type="password"
    id="password"
    name="password"
    placeholder="Enter your password"
    value={formData.password}
    onChange={handleChange}
    className={errors.password ? 'error' : ''}
  />
  {errors.password && (
    <span className="error-text">{errors.password}</span>
  )}
</div>

{!isLogin && (
  <div className="form-group">
    <label htmlFor="confirmPassword">Confirm Password</label>
    <input
      type="password"
      id="confirmPassword"
      name="confirmPassword"
      placeholder="Confirm your password"
      value={formData.confirmPassword}
      onChange={handleChange}
```

```
        className={errors.confirmPassword ? 'error' : ''}
      />
      {errors.confirmPassword && (
        <span className="error-text">
          {errors.confirmPassword}
        </span>
      )}
    </div>
  )}

  <button type="submit" className="submit-btn" disabled={loading}>
    {loading ? (
      <span className="loading-spinner"></span>
    ) : isLogin ? 'Sign In' : 'Create Account'}
  </button>
</form>

<div className="form-footer">
  <p>
    {isLogin
      ? "Don't have an account? "
      : 'Already have an account? '}
    <button
      type="button"
      className="toggle-btn"
```

```
            onClick={() => {
              setIsLogin(!isLogin);
              setErrors({});
              // Do NOT clear formData on toggle so user doesn't lose input
            }}
          >
            {isLogin ? 'Sign Up' : 'Sign In'}
          </button>
        </p>
      </div>
    </div>
  </div>
 );
};

export default SignupPage;
```

2. Admin Dashboard:



Code:

```
import React, { useEffect, useState } from 'react';
import api from '../services/api';
import { Container, Table, Button, Form, Row, Col } from 'react-bootstrap';


const AdminPage = () => {
  const [bookings, setBookings] = useState([]);
  const [loading, setLoading] = useState(true);
  const [techForm, setTechForm] = useState({ name: '', expertise: '' });
```

```
const [techMessage, setTechMessage] = useState('');

const fetchBookings = async () => {
  try {
    const resBookings = await api.get('/worker-bookings');
    setBookings(resBookings.data);
  } catch (err) {
    console.error(err);
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  fetchBookings();
  const interval = setInterval(() => {
    fetchBookings();
  }, 5000);
  return () => clearInterval(interval);
}, []);

const markCompleted = async (id) => {
  try {
    await api.put(`/worker-bookings/${id}/complete`);
    fetchBookings();
```

```
  } catch (err) {
    console.error(err);
  }
};

const handleTechInput = (e) => {
  const { name, value } = e.target;
  setTechForm(prev => ({ ...prev, [name]: value }));
};

const handleAddTechnician = async (e) => {
  e.preventDefault();
  try {
    const res = await api.post('/technicians', techForm);
    setTechMessage(`✅ Technician ${res.data.name} added!`);
    setTechForm({ name: '', expertise: '' });
    setTimeout(() => setTechMessage(''), 4000);
  } catch (err) {
    setTechMessage('❌ Failed to add technician.');
    console.error(err);
  }
};

if (loading) return <p className="m-5">Loading admin dashboard...</p>;
```

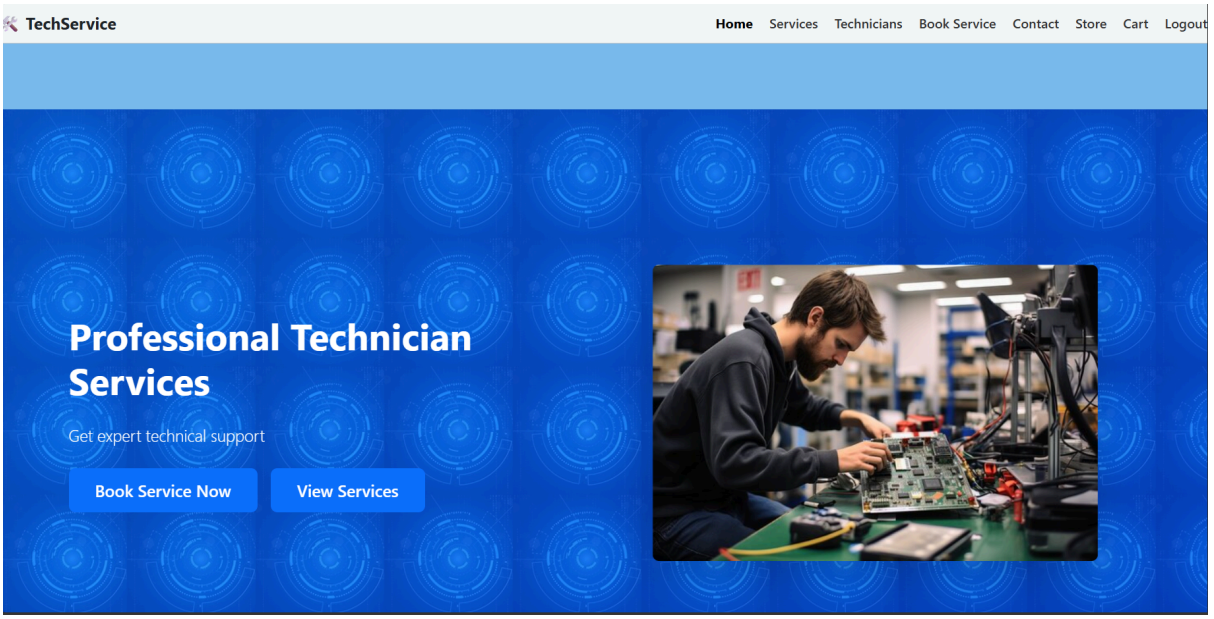
```
return (
  <Container className="mt-5">
    <h2 className="mb-4">👑 Admin Dashboard</h2>

    <Button onClick={fetchBookings} className="mb-3">
      🔄 Refresh Bookings
    </Button>

    <h4>📋 Worker Bookings</h4>
    {bookings.length === 0 ? (
      <p>No bookings found.</p>
    ) : (
      <Table striped bordered hover>
        <thead>
          <tr>
            <th>Customer</th>
            <th>Email</th>
            <th>Service</th>
            <th>Technician</th>
            <th>Date</th>
            <th>Time</th>
            <th>Urgency</th>
            <th>Status</th>
            <th>Actions</th>
          </tr>
```

```
    </thead>
    <tbody>
      {bookings.map((b) => (
        <tr key={b._id}>
          <td>{b.firstName}  {b.lastName}</td>
          <td>{b.email}</td>
          <td>{b.service}</td>
          <td>{b.technician?.name || 'N/A'}</td>
          <td>{b.date}</td>
          <td>{b.time}</td>
          <td>{b.urgency || 'N/A'}</td>
          <td>{b.status}</td>
          <td>
            {b.status === 'active' && (
              <Button size="sm" onClick={() => markCompleted(b._id)}>
                Mark Completed
              </Button>
            )}
          </td>
        </tr>
      ))}
    </tbody>
  </Table>
)}
```

```
<hr className="my-4" />

<h4>➕ Add Technician</h4>
<Form onSubmit={handleAddTechnician} className="mb-4">
  <Row>
    <Col md={5}>
      <Form.Control
        type="text"
        placeholder="Technician Name"
        name="name"
        value={techForm.name}
        onChange={handleTechInput}
        required
      />
    </Col>
    <Col md={5}>
      <Form.Control
        type="text"
        placeholder="Expertise (e.g. Mobile Repair)"
        name="expertise"
        value={techForm.expertise}
        onChange={handleTechInput}
        required
      />
    </Col>
```

```
          <Col md={2}>
            <Button type="submit" className="w-100">Add</Button>
          </Col>
        </Row>
      </Form>

      {techMessage && <p>{techMessage}</p>}
    </Container>
  );
};

export default AdminPage;
```

3. Home Page:



Code:

```
import React from 'react'

import { Container, Row, Col, Button, Card } from 'react-bootstrap'

import { Link } from 'react-router-dom'

import { FaBroom, FaTemperatureHigh, FaWifi, FaCar, FaCog, FaShieldAlt, FaClock,
FaStar } from 'react-icons/fa'

import '../styles/HomePage.css'



const HomePage = () => {

  const stats = [

    { icon: FaStar , label: 'Happy Customers' },

    { icon: FaCog , label: 'Expert Technicians' },

    { icon: FaClock, label: 'Support Available' },

    { icon: FaShieldAlt, label: 'Satisfaction Guarantee' },

  ]
```

```
const featuredServices = [
  {
    icon: FaBroom,
    title: 'House Cleanning',
    description: 'Professional House and Garden cleaner',
    color: 'primary'
  },
  {
    icon:FaTemperatureHigh ,
    title: 'AC Repair',
    description: 'Professional AC repair ',
    color: 'success'
  },
  {
    icon: FaWifi,
    title: 'Network Setup',
    description: 'Home and office network installation and configuration',
    color: 'info'
  },
  {
    icon: FaCar,
    title: 'Car Repair',
    description: 'Complete Car repair with professionals',
    color: 'warning'
  },
```

```
  ]

  return (
    <>
      <section className="hero-section">
        <Container>
          <Row className="align-items-center" style={{ paddingTop: '125px', paddingBottom:
'125px' }}>

            <Col lg={6} className="hero-content">
              <div className="fade-in">
                <h1 className="display-4 fw-bold mb-4">
                  Professional Technician Services
                </h1>
                <p className="lead mb-4">
                  Get expert technical support

                </p>
                <div className="d-flex gap-3">
                  <Button
                    as={Link}
                    to="/booking"
                    size="lg"
                    className="btn-custom"
```

```
        >
          Book Service Now
        </Button>
        <Button
          as={Link}
          to="/services"
          size="lg"
          className="btn-custom"
        >
          View Services
        </Button>
      </div>
    </div>
  </Col>
  <Col lg={6} className="text-center">
    <div className="pulse">
      <img
        src="./src/assets/computerRepair_pic.jpg"
        alt="Technician at work"
        className="img-fluid rounded-3 shadow-lg"
        style={{maxHeight: '400px'}}
      />
    </div>
  </Col>
</Row>
```

```
  </Container>
</section>
```

```
<section className="py-5 bg-light">
  <Container>
    <Row>
      {stats.map((stat, index) => (
        <Col md={6} lg={3} key={index} className="mb-4">
          <div className="stats-card text-center">
            <stat.icon className="text-primary fs-1 mb-3" />
            <span className="stats-number">{stat.number}</span>
            <p className="text-muted mb-0">{stat.label}</p>
          </div>
        </Col>
      ))}
    </Row>
  </Container>
</section>
```

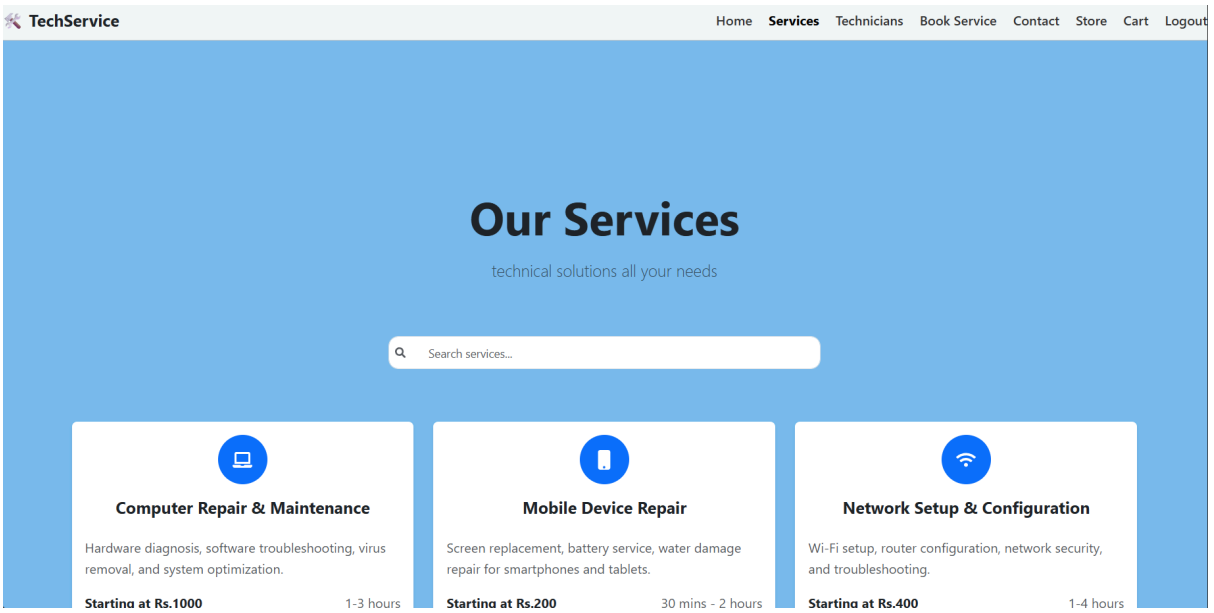
```
<section className="py-5">
  <Container>
    <Row className="mb-5">
      <Col className="text-center">
        <h2 className="display-5 fw-bold mb-3">Our Featured Services</h2>
        <p className="lead text-muted">
          We provide  technical solutions for homes and businesses
        </p>
      </Col>
    </Row>

    <Row>
      {featuredServices.map((service, index) => (
        <Col md={6} lg={3} key={index} className="mb-4">
          <Card className="service-card h-100">
            <Card.Body className="text-center p-4">
              <div className={`service-icon bg-${service.color} mb-3`}>
                <service.icon />
              </div>
              <h5 className="fw-bold mb-3">{service.title}</h5>
              <p className="text-muted">{service.description}</p>
            </Card.Body>
          </Card>
        </Col>
```

```
      ))}
    </Row>

    <Row className="mt-5">
      <Col className="text-center">
        <Button
          as={Link}
          to="/services"
          size="lg"
          className="btn-custom"
        >
          View All Services
        </Button>
      </Col>
    </Row>
  </Container>
</section>


<section className="py-5 bg-light">
  <Container>
    <Row className="align-items-center">
      <Col lg={6} className="mb-4">
        <img
          src="./src/assets/homeServices_pic.jpg"
```

```
    alt="Professional technician team"
    className="img-fluid"
   />
  </Col>
  <Col lg={6}>
   <h2 className="display-5 fw-bold mb-4">Why Choose TechService Pro?</h2>
   <div className="mb-3">
    <h5 className="fw-bold">Expert Technicians</h5>
    <p className="text-muted">Certified professionals with years of experience</p>
   </div>
   <div className="mb-3">
    <h5 className="fw-bold">Fast Response</h5>
    <p className="text-muted">Same-day service available for urgent repairs</p>
   </div>
   <div className="mb-3">
    <h5 className="fw-bold">Low Pricing</h5>
    <p className="text-muted">No hidden fees</p>
   </div>
   <div className="mb-3">
    <h5 className="fw-bold">Warranty Guarantee</h5>
    <p className="text-muted">All repairs come with warranty</p>
   </div>
   <Button
    as={Link}
    to="/technicians"
```

```
              className="btn-custom">
              Meet Our Team
            </Button>
          </Col>
        </Row>
      </Container>
    </section>
  </>
 )
}

export default HomePage
```

4. **Services Page:**



Code:

```
import React, { useState } from "react";
import {
  Container,
  Row,
  Col,
  Card,
  Button,
  Badge,
  Form,
} from "react-bootstrap";
import { Link } from "react-router-dom";
import {
  FaLaptop,
  FaMobile,
```

```
  FaWifi,
  FaTv,
  FaPrint,
  FaCamera,
  FaBroom,
  FaCar,
  FaSearch,
  FaTemperatureHigh,
  FaHouseDamage,
  FaBolt,
  FaBrush,
} from "react-icons/fa";

const ServicesPage = () => {
  const [searchTerm, setSearchTerm] = useState("");
  const [selectedCategory, setSelectedCategory] = useState("all");

  const services = [
    {
      id: 1,
      icon: FaLaptop,
      title: "Computer Repair & Maintenance",
      description:
        "Hardware diagnosis, software troubleshooting, virus removal, and system
optimization.",
```

```
  price: "Starting at Rs.1000",
  duration: "1-3 hours",
  category: "computer",
},
{
  id: 2,
  icon: FaMobile,
  title: "Mobile Device Repair",
  description:
    "Screen replacement, battery service, water damage repair for smartphones and tablets.",
  price: "Starting at Rs.200",
  duration: "30 mins - 2 hours",
  category: "mobile",
},
{
  id: 3,
  icon: FaWifi,
  title: "Network Setup & Configuration",
  description:
    "Wi-Fi setup, router configuration, network security, and troubleshooting.",
  price: "Starting at Rs.400",
  duration: "1-4 hours",
  category: "network",
},
{
```

```
  id: 4,
  icon: FaTv,
  title: "Smart Home Installation",
  description:
    "Smart TV setup, home automation, security systems, and IoT device configuration.",
  price: "Starting at Rs.1500",
  duration: "2-6 hours",
  category: "House Service",
},
{
  id: 5,
  icon: FaPrint,
  title: "Printer & Scanner Service",
  description:
    "Printer setup, troubleshooting, cartridge replacement, and maintenance.",
  price: "Starting at Rs.200",
  duration: "30 mins - 1 hour",
  category: "printer",
},
{
  id: 6,
  icon: FaCamera,
  title: "Security Camera Installation",
  description:
    "CCTV installation, IP camera setup, monitoring system configuration.",
```

```
  price: "Starting at Rs.800",
  duration: "3-8 hours",
  category: "security",
},
{
  id: 7,
  icon: FaBroom,
  title: "House Cleaner",
  description: "professional house cleaner",
  price: "Starting at Rs.500",
  duration: "1-3 hours",
  category: "cleaning",
},
{
  id: 8,
  icon: FaCar,
  title: "Car Repair",
  description: "professional car repair . we done all types of work",
  price: "Starting at Rs.220",
  duration: "2-24 hours",
  category: "car",
},
{
  id: 9,
  icon: FaBolt,
```

```
    title: "Electrician",
    description: "professional electrician",
    price: "Starting at Rs.220",
    duration: "2-24 hours",
    category: "electrical",
  },

  {
    id: 12,
    icon: FaTemperatureHigh,
    title: "AC Technician",
    description: "Professional AC technician",
    price: "Starting at Rs.1120",
    duration: "2-6 hours",
    category: "ac",
  },
];

const categories = [
  { value: "all", label: "All Services" },
  { value: "computer", label: "Computer" },
  { value: "mobile", label: "Mobile" },
  { value: "network", label: "Network" },
  { value: "smart-home", label: "Smart Home" },
  { value: "printer", label: "Printer" },
```

```
  { value: "security", label: "Security" },
  { value: "cleaning", label: "Cleaner" },
  { value: "car", label: "Car" },
  { value: "electrical", label: "Electrician" },
  { value: "ac", label: "AC" },
];

const filteredServices = services.filter((service) => {
  const matchesSearch = service.title
    .toLowerCase()
    .includes(searchTerm.toLowerCase());
  const matchesCategory =
    selectedCategory === "all" || service.category === selectedCategory;
  return matchesSearch && matchesCategory;
});

return (
  <Container style={{ paddingTop: "100px", paddingBottom: "50px" }}>
    {/* Header */}
    <Row className="mb-5">
      <Col className="text-center">
        <h1 className="display-4 fw-bold mb-3">Our Services</h1>
        <p className="lead text-muted">technical solutions all your needs</p>
      </Col>
    </Row>
```

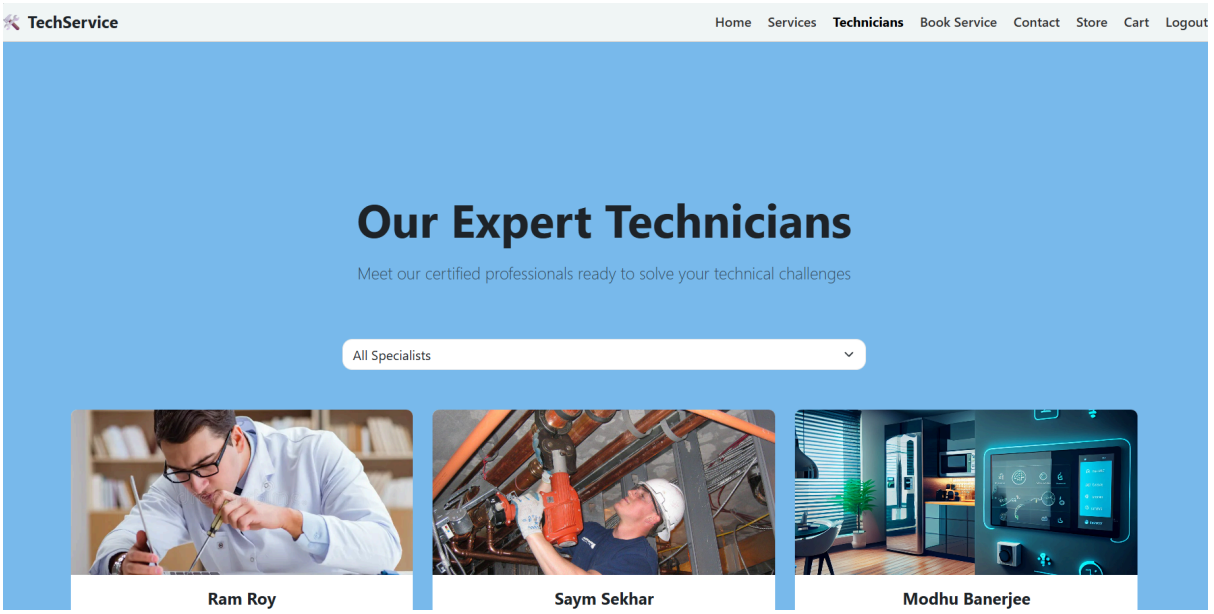
```
{/* Search */}
<Row className="mb-5 justify-content-center">
  <Col md={5} className="mb-3">
    <div className="position-relative">
      <FaSearch
        className="position-absolute top-50 start-0 translate-middle-y ms-2 text-muted"
        style={{ fontSize: "13px" }}
      />
      <Form.Control
        type="text"
        placeholder="Search services..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        className="ps-5"
        style={{
          height: "40px",
          width: "100%",
          fontSize: "14px",
          paddingLeft: "2rem",
        }}
      />
    </div>
  </Col>
</Row>
```

```
{/* No Data Message */}
{filteredServices.length === 0 && (
  <Row className="mb-5 justify-content-center">
    <Col md={8} className="text-center">
      <p className="text-muted fs-5">No data found</p>
    </Col>
  </Row>
)}



      {/* Services Grid */}
      <Row>
        {filteredServices.map((service) => (
          <Col md={6} lg={4} key={service.id} className="mb-4">
            <Card className="service-card h-100 rounded - 3 shadow-sm">
              <Card.Body className="d-flex flex-column">
                <div className="text-center mb-3">
                  <div className="service-icon bg-primary mb-3">
                    <service.icon />
                  </div>
                  <div className="d-flex justify-content-center gap-2 mb-2">
                    <h5 className="fw-bold mb-0">{service.title}</h5>
                    {service.popular && (
                      <Badge bg="warning" text="dark">
                        Popular
```

```
      </Badge>
    )}
  </div>
</div>

<p className="text-muted mb-3 flex-grow-1">
  {service.description}
</p>

<div className="service-details mb-3">
  <div className="d-flex justify-content-between mb-2">
    <span className="fw-bold text-dark">
      {service.price}
    </span>
    <span className="text-muted">{service.duration}</span>
  </div>
</div>

<Button
  as={Link}
  to="/booking"
  state={{ selectedService: service.title }}
  className="btn-custom btn-primary-custom w-100 rounded-pill"
  style={{
    backgroundColor: '#1f1f1f',
```

```
                  border: "none",
                  padding: "10px 20px",
                }}
                // <-- ADDED HERE
              >
                Book This Service
              </Button>
            </Card.Body>
          </Card>
        </Col>
      ))}
    </Row>
  </Container>
 );
};

export default ServicesPage;
```

5. Technician Page:



Code:

```
import React, { useState } from 'react';

import { Container, Row, Col, Card, Badge, Button, Form } from 'react-bootstrap';

import { FaStar, FaMapMarkerAlt, FaPhone, FaEnvelope } from 'react-icons/fa';


// ✅ Image imports

import i1 from '../assets/i1.png';

import i2 from '../assets/i2.png';

import i3 from '../assets/i3.png';

import i4 from '../assets/i4.png';

import i5 from '../assets/i5.png';

import i6 from '../assets/i6.png';

import i7 from '../assets/i7.png';

import i8 from '../assets/i8.png';

import smartHousePic from '../assets/smartHouse_pic.jpg';
```

```
import plumberPic from '../assets/plumber_pic.jpg';


const TechniciansPage = () => {
  const [selectedSpecialty, setSelectedSpecialty] = useState('all');


  const technicians = [
    {
      id: 1,
      name: 'Ram Roy',
      specialty: 'Computer Hardware',
      image: i8,
      rating: 4.7,
      reviews: 89,
      experience: '8 years',
      location: 'Dum Dum',
      phone: '11111111111',
      email: 'ram@gmail.com',
      specialties: ['Hardware Repair', 'System Building', 'Upgrades'],
      bio: 'Specialized in computer hardware diagnostics and repair with extensive experience
in custom builds.'
    },
    {
      id: 2,
      name: 'Saym Sekhar',
      specialty: 'Plumber',
```

```
    image: plumberPic,
    rating: 4.7,
    reviews: 89,
    experience: '6 years',
    location: 'Agarpara',
    phone: '22222222222',
    email: 'saym@gmail.com',
    specialties: ['Leakage Repair', 'Water Damage Fix'],
    bio: 'Expert plumber with a strong focus on resolving water-related issues quickly and efficiently.'
  },
  {
    id: 3,
    name: 'Modhu Banerjee',
    specialty: 'Smart House',
    image: smartHousePic,
    rating: 4.7,
    reviews: 89,
    experience: '10 years',
    location: 'Sodepur',
    phone: '33333333333',
    email: 'modhu@gmail.com',
    specialties: ['Network Setup', 'Security Configuration', 'Troubleshooting'],
    bio: 'Smart home automation expert specializing in integrated, user-friendly home technology solutions.'
  },
```

```
  {
    id: 4,
    name: 'Amit Kumar',
    specialty: 'Computer Technician',
    image: i3,
    rating: 4.5,
    reviews: 120,
    experience: '6 years',
    location: 'Kolkata',
    phone: '9876543210',
    email: 'amit.tech@gmail.com',
    specialties: ['Hardware Repair', 'Software Installation', 'Troubleshooting'],
    bio: 'Experienced computer technician skilled in hardware and software troubleshooting for desktops and laptops.'
  },
  {
    id: 5,
    name: 'Priya Singh',
    specialty: 'Mobile Repair Expert',
    image: i1,
    reviews: 95,
    experience: '4 years',
    location: 'Delhi',
    phone: '9123456789',
    email: 'priya.mobifix@gmail.com',
```

```
    specialties: ['Screen Replacement', 'Battery Issues', 'OS Flashing'],
    bio: 'Mobile phone repair expert with expertise in fixing Android and iOS devices.'
  },
  {
    id: 6,
    name: 'Ravi Verma',
    specialty: 'Network Engineer',
    image: i2,
    rating: 4.7,
    reviews: 150,
    experience: '7 years',
    location: 'Mumbai',
    phone: '9988776655',
    email: 'ravi.networkpro@gmail.com',
    specialties: ['Router Setup', 'LAN/WAN', 'Firewall Configuration'],
    bio: 'Professional network engineer helping homes and businesses set up secure and fast networks.'
  },
  {
    id: 7,
    name: 'Kavita Joshi',
    specialty: 'Smart Home Installer',
    image: i4,
    rating: 4.6,
    reviews: 110,
```

```
    experience: '5 years',
    location: 'Bengaluru',
    phone: '9090909090',
    email: 'kavita.smarthome@gmail.com',
    specialties: ['Automation Setup', 'Voice Control Integration', 'IoT Troubleshooting'],
    bio: 'Smart home automation expert focused on simplifying everyday living with technology.'
  },
  {
    id: 8,
    name: 'Sandeep Das',
    specialty: 'Printer Technician',
    image: i5,
    rating: 4.2,
    reviews: 60,
    experience: '6 years',
    location: 'Kolkata',
    phone: '9234567890',
    email: 'sandeep.printerfix@gmail.com',
    specialties: ['Inkjet and Laser Repairs', 'Driver Installation', 'Paper Jam Solutions'],
    bio: 'Printer repair technician with in-depth experience across HP, Canon, and Epson devices.'
  },
  {
    id: 9,
    name: 'Neha Sharma',
```

```
  specialty: 'Security System Installer',
  image: i6,
  rating: 4.8,
  reviews: 130,
  experience: '8 years',
  location: 'Pune',
  phone: '9000001111',
  email: 'neha.securetech@gmail.com',
  specialties: ['CCTV Setup', 'Door Access Systems', 'Smart Locks'],
  bio: 'Certified professional for residential and commercial security system installations.'
},
{
  id: 10,
  name: 'Anjal Mehra',
  specialty: 'AC Technician',
  image: i7,
  rating: 4.5,
  reviews: 105,
  experience: '6 years',
  location: 'Chennai',
  phone: '9444444444',
  email: 'anjali.coolcare@gmail.com',
  specialties: ['Gas Refill', 'Cooling Issues', 'Installation'],
  bio: 'Expert in servicing and repairing window, split, and inverter ACs.'
}
```

```
];

const specialties = [
  { value: 'all', label: 'All Specialists' },
  ...Array.from(new Set(technicians.map(t => t.specialty))).map(spec => ({
    value: spec,
    label: spec
  }))
];

const filteredTechnicians = technicians.filter(
  tech => selectedSpecialty === 'all' || tech.specialty === selectedSpecialty
);

const renderStars = (rating) => {
  const stars = [];
  const fullStars = Math.floor(rating);
  const hasHalfStar = rating % 1 !== 0;

  for (let i = 0; i < fullStars; i++) {
    stars.push(<FaStar key={i} className="text-warning" />);
  }

  if (hasHalfStar) {
    stars.push(<FaStar key="half" className="text-warning opacity-50" />);
```

```
    }

    return stars;
  };

  return (
    <Container style={{ paddingTop: '100px', paddingBottom: '50px' }}>
      <Row className="mb-5">
        <Col className="text-center">
          <h1 className="display-4 fw-bold mb-3">Our Expert Technicians</h1>
          <p className="lead text-muted"> Meet our certified professionals ready to solve your technical challenges</p>
        </Col>
      </Row>

      <Row className="mb-5">
        <Col md={6} className="mx-auto">
          <Form.Select
            value={selectedSpecialty}
            onChange={(e) => setSelectedSpecialty(e.target.value)}>
            {specialties.map((spec) => (
              <option key={spec.value} value={spec.value}>
                {spec.label}
              </option>
            ))}
```

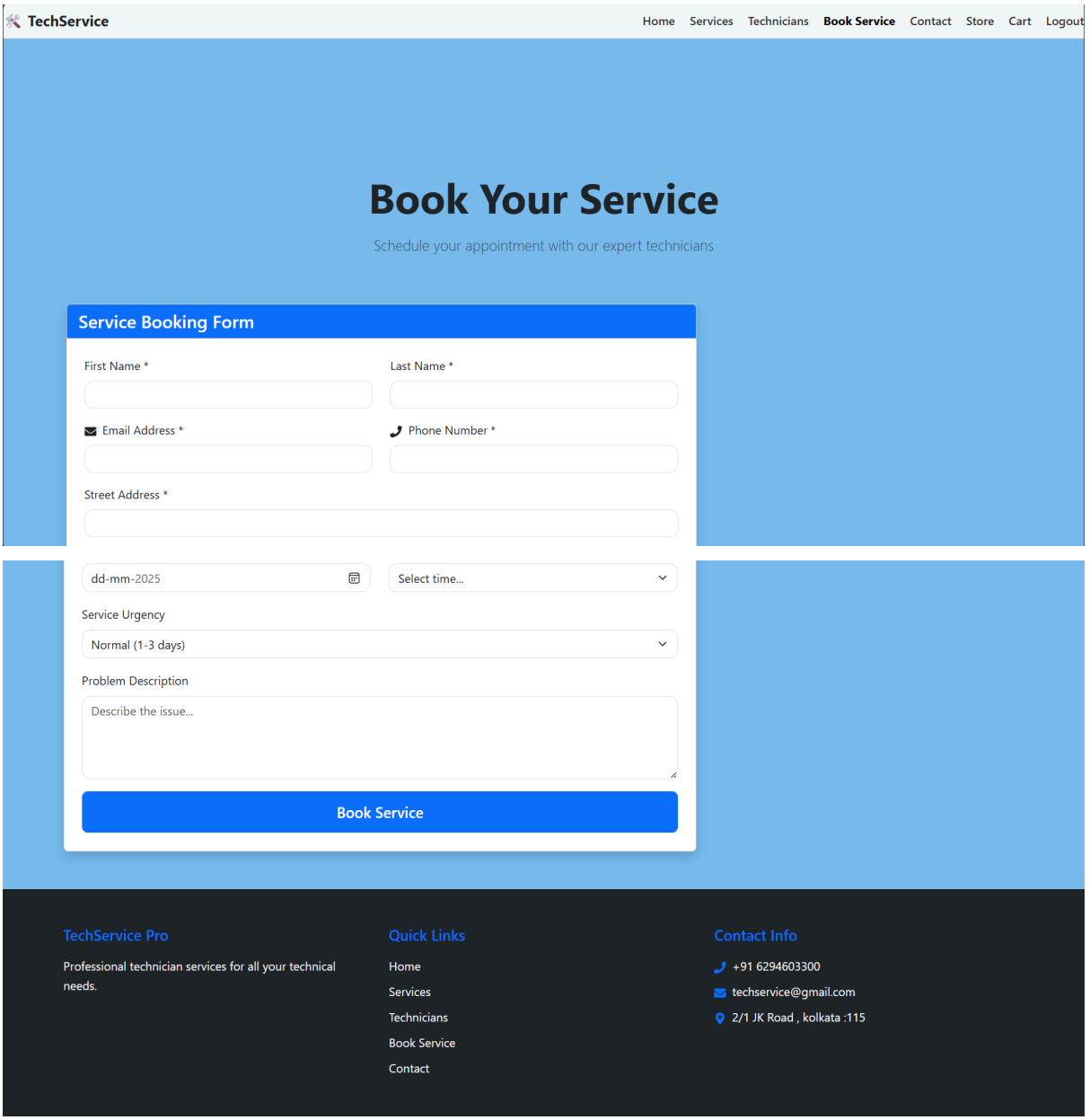
```
    </Form.Select>
  </Col>
</Row>

<Row>
  {filteredTechnicians.map((technician) => (
    <Col md={6} lg={4} key={technician.id} className="mb-4">
      <Card className="technician-card h-100">
        <img
          src={technician.image}
          alt={technician.name}
          className="technician-image card-img-top" />
        <Card.Body className="d-flex flex-column">
          <div className="text-center mb-3">
            <h5 className="fw-bold mb-1">{technician.name}</h5>
            <p className="text-primary mb-2">{technician.specialty}</p>
            <div className="d-flex justify-content-center align-items-center mb-2">
              {renderStars(technician.rating)}
              <span className="fw-bold ms-2">{technician.rating}</span>
              <span className="text-muted ms-1">({technician.reviews} reviews)</span>
            </div>
          </div>

          <div className="mb-3">
            <div className="d-flex justify-content-between mb-2">
```

```
    <span className="text-muted">Experience:</span>
    <span className="fw-bold">{technician.experience}</span>
  </div>
  <div className="d-flex justify-content-between mb-2">
    <span className="text-muted">
      <FaMapMarkerAlt className="me-1" />
      Location:
    </span>
    <span>{technician.location}</span>
  </div>
</div>

<div className="mb-3">
  <p className="text-muted small">{technician.bio}</p>
</div>

<div className="mb-3">
  <h6 className="fw-bold mb-2">Specialties:</h6>
  <div className="d-flex flex-wrap gap-1">
    {technician.specialties.map((spec, index) => (
      <Badge key={index} bg="light" text="dark" className="small">
        {spec}
      </Badge>
    ))}
  </div>
```

```
</div>


<div className="mt-auto">
  <div className="d-flex gap-2 mb-3">
    <Button
      variant="outline-primary"
      size="sm"
      className="flex-fill"
      href={`tel:${technician.phone}`}>
      <FaPhone className="me-1" />
      Call
    </Button>
    <Button
      variant="outline-secondary"
      size="sm"
      className="flex-fill"
      href={`mailto:${technician.email}`}>
      <FaEnvelope className="me-1" />
      Email
    </Button>
  </div>
  <Button className="w-100 btn-primary">
    Book Appointment
  </Button>
</div>
```

```
            </Card.Body>
          </Card>
        </Col>
      ))}
    </Row>
  </Container>
 );
};

export default TechniciansPage;
```

6. Book Service:



Code:

```
import React, { useState, useEffect } from 'react'
import { Container, Row, Col, Form, Button, Card } from 'react-bootstrap'
import { FaPhone, FaEnvelope } from 'react-icons/fa'
import '../styles/BookingPage.css'


const BookingPage = () => {
```

```
const [formData, setFormData] = useState({
  firstName: '', lastName: '', email: '', phone: '', address: '',
  city: '', zipCode: '', service: '', technician: '', date: '', time: '',
  urgency: 'normal', description: '', agreeTerms: false
});
const [technicians, setTechnicians] = useState([]);
const [showAlert, setShowAlert] = useState(false);

useEffect(() => {
  fetch('http://localhost:5000/api/technicians/available')
    .then(res => res.json())
    .then(data => setTechnicians(data))
    .catch(err => console.error('Failed to fetch technicians', err));
}, []);

const services = [
  'Computer Repair & Maintenance',
  'Mobile Device Repair',
  'Network Setup & Configuration',
  'Smart Home Installation',
  'Printer & Scanner Service',
  'Security Camera Installation',
  'Gaming Console Repair',
  'Data Recovery & Backup'
];
```

```
const timeSlots = [
  '9:00 AM', '10:00 AM', '11:00 AM', '12:00 PM',
  '1:00 PM', '2:00 PM', '3:00 PM', '4:00 PM', '5:00 PM'
];

const handleInputChange = (e) => {
  const { name, value, type, checked } = e.target;
  setFormData(prev => ({
    ...prev,
    [name]: type === 'checkbox' ? checked : value
  }));
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const payload = {
      ...formData,
      technicianId: formData.technician
    };
    delete payload.technician;

    const response = await fetch('http://localhost:5000/api/worker-bookings', {
      method: 'POST',
```

```
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    });

    const data = await response.json();

    if (response.ok) {
      setShowAlert(true);
      setTimeout(() => setShowAlert(false), 5000);
      setFormData({
        firstName: '', lastName: '', email: '', phone: '', address: '',
        city: '', zipCode: '', service: '', technician: '', date: '',
        time: '', urgency: 'normal', description: '', agreeTerms: false
      });
    } else {
      alert(data.error || 'Something went wrong.');
    }
  } catch (error) {
    console.error(error);
    alert('Server error.');
  }
};

const getTomorrowDate = () => {
  const tomorrow = new Date();
```

```
    tomorrow.setDate(tomorrow.getDate() + 1);
    return tomorrow.toISOString().split('T')[0];
  };

  const getMaxDate = () => {
    const maxDate = new Date();
    maxDate.setDate(maxDate.getDate() + 30);
    return maxDate.toISOString().split('T')[0];
  };

  return (
    <Container style={{ paddingTop: '100px', paddingBottom: '50px' }}>
      <Row className="mb-5">
        <Col className="text-center">
          <h1 className="display-4 fw-bold mb-3">Book Your Service</h1>
          <p className="lead text-muted">Schedule your appointment with our expert technicians</p>
        </Col>
      </Row>

      <Row>
        <Col lg={8}>
          <Card className="shadow">
            <Card.Header className="bg-primary text-white">
              <h4 className="mb-0">Service Booking Form</h4>
```

```
        </Card.Header>

        <Card.Body className="p-4">

          {showAlert && <div className="alert alert-success">Your booking has been
submitted!</div>}


          <Form onSubmit={handleSubmit}>

            <Row>

              <Col md={6} className="mb-3">

                <Form.Label>First Name *</Form.Label>

                <Form.Control type="text" name="firstName" value={formData.firstName}
onChange={handleInputChange} required />

              </Col>

              <Col md={6} className="mb-3">

                <Form.Label>Last Name *</Form.Label>

                <Form.Control type="text" name="lastName" value={formData.lastName}
onChange={handleInputChange} required />

              </Col>

            </Row>


            <Row>

              <Col md={6} className="mb-3">

                <Form.Label><FaEnvelope className="me-1" /> Email Address
*</Form.Label>

                <Form.Control type="email" name="email" value={formData.email}
onChange={handleInputChange} required />

              </Col>

              <Col md={6} className="mb-3">
```

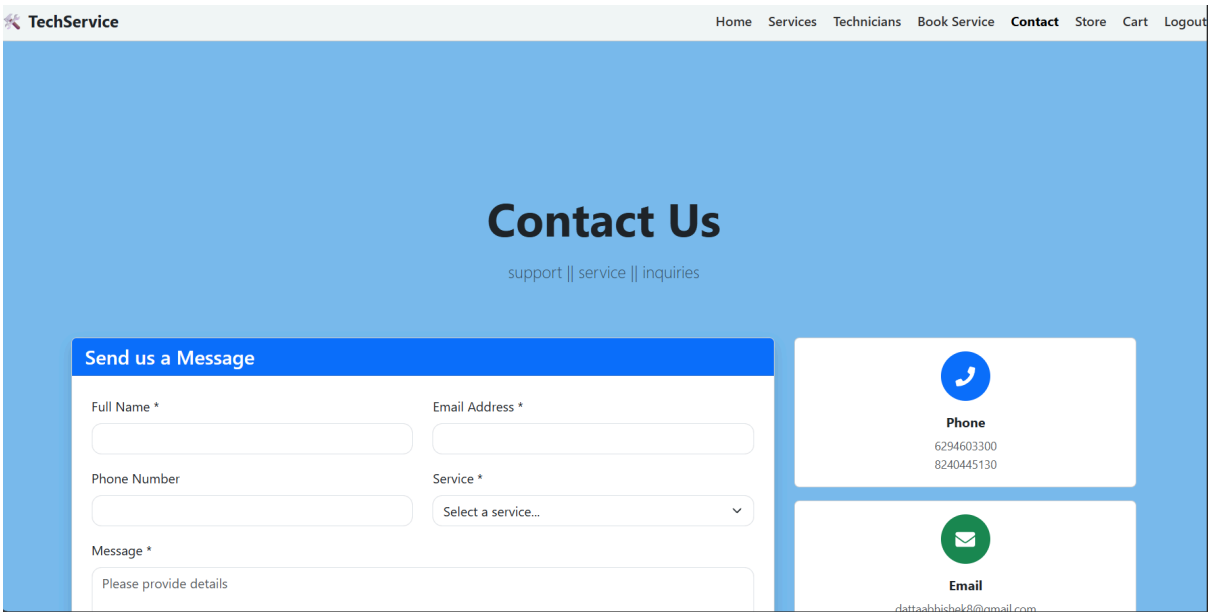
```
            <Form.Label><FaPhone className="me-1" /> Phone Number *</Form.Label>

            <Form.Control type="tel" name="phone" value={formData.phone}
onChange={handleInputChange} required />

          </Col>

        </Row>


        <Form.Group className="mb-3">

          <Form.Label>Street Address *</Form.Label>

          <Form.Control type="text" name="address" value={formData.address}
onChange={handleInputChange} required />

        </Form.Group>


        <Row>

          <Col md={8} className="mb-3">

            <Form.Label>City *</Form.Label>

            <Form.Control type="text" name="city" value={formData.city}
onChange={handleInputChange} required />

          </Col>

          <Col md={4} className="mb-3">

            <Form.Label>PIN Code *</Form.Label>

            <Form.Control type="text" name="zipCode" value={formData.zipCode}
onChange={handleInputChange} required />

          </Col>

        </Row>


        <Row>

          <Col md={6} className="mb-3">
```

```
                <Form.Label>Select Service *</Form.Label>

                <Form.Select name="service" value={formData.service}
onChange={handleInputChange} required>

                  <option value="">Choose a service...</option>

                  {services.map((service, index) => (

                    <option key={index} value={service}>{service}</option>

                  ))}

                </Form.Select>

              </Col>

              <Col md={6} className="mb-3">

                <Form.Label>Preferred Technician</Form.Label>

                <Form.Select name="technician" value={formData.technician}
onChange={handleInputChange} required>

                  <option value="">Choose an available technician</option>

                  {technicians.map((tech) => (

                    <option key={tech._id} value={tech._id}>{tech.name} -
{tech.expertise}</option>

                  ))}

                </Form.Select>

              </Col>

            </Row>


            <Row>

              <Col md={6} className="mb-3">

                <Form.Label>Preferred Date *</Form.Label>
```

```
                <Form.Control type="date" name="date" value={formData.date}
onChange={handleInputChange} min={getTomorrowDate()} max={getMaxDate()} required
/>

              </Col>

              <Col md={6} className="mb-3">

                <Form.Label>Preferred Time *</Form.Label>

                <Form.Select name="time" value={formData.time}
onChange={handleInputChange} required>

                  <option value="">Select time...</option>

                  {timeSlots.map((time, index) => (

                    <option key={index} value={time}>{time}</option>

                  ))}

                </Form.Select>

              </Col>

            </Row>


            <Form.Group className="mb-3">

              <Form.Label>Service Urgency</Form.Label>

              <Form.Select name="urgency" value={formData.urgency}
onChange={handleInputChange}>

                <option value="normal">Normal (1-3 days)</option>

                <option value="urgent">Urgent (Same day)</option>

                <option value="emergency">Emergency (ASAP)</option>

              </Form.Select>

            </Form.Group>


            <Form.Group className="mb-3">
```

```
                <Form.Label>Problem Description</Form.Label>

                <Form.Control as="textarea" rows={4} name="description"
value={formData.description} onChange={handleInputChange} placeholder="Describe the
issue..." />

              </Form.Group>


              <Button type="submit" size="lg" className="w-100 btn-custom
btn-primary-custom">Book Service</Button>

            </Form>

          </Card.Body>

        </Card>

      </Col>

    </Row>

  </Container>

 );

};


export default BookingPage;
```

7. **Contact:**



Code:

```
import React, { useState } from 'react'
import { Container, Row, Col, Form, Button, Card, Alert } from 'react-bootstrap'
import { FaPhone, FaEnvelope, FaMapMarkerAlt } from 'react-icons/fa'


const ContactPage = () => {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    phone: '',
    subject: '',
    message: ''
  });

  const [showAlert, setShowAlert] = useState(false);
```

```
const handleInputChange = (e) => {
  const { name, value } = e.target;
  setFormData(prev => ({
    ...prev,
    [name]: value
  }));
};

const handleSubmit = async (e) => {
  e.preventDefault();

  try {

    if (!formData.name || !formData.email || !formData.subject || !formData.message) {
      alert('Please fill in all fields.');
      return;
    }

    const res = await fetch('http://localhost:5000/api/contact', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
```

```
    });

    const data = await res.json();
    console.log(data);

    if (data.success) {
      setShowAlert(true);
      setTimeout(() => setShowAlert(false), 5000);

      setFormData({
        name: '',
        email: '',
        phone: '',
        subject: '',
        message: ''
      });
    } else {
      alert('Error sending message.');
    }
  } catch (err) {
    console.error(err);
    alert('Server error.');
  }
};
```

```
const contactInfo = [
  {
    icon: FaPhone,
    title: 'Phone',
    details: ['6294603300', '8240445130'],
    color: 'primary'
  },
  {
    icon: FaEnvelope,
    title: 'Email',
    details: ['dattaabhishek8@gmail.com', '23cs2011003jis@gmail.com'],
    color: 'success'
  },
  {
    icon: FaMapMarkerAlt,
    title: 'Address',
    details: ['Agarpara, Nilgunj Road', 'JIS University, 700000'],
    color: 'warning'
  },
];

const locations = [
  {
    name: 'Dum Dum Office',
    address: '7-Roymallick Colony, Kundubagan, 740000',
```

```
    phone: '6294603300',
    manager: 'Abhishek Datta'
  },
  {
    name: 'Agarpara Office',
    address: 'Agarpara, Kamarhati, 750000',
    phone: '9999999999',
    manager: 'Antara Biswas'
  },
  {
    name: 'Barrackpore Office',
    address: 'Barrackpore Road, Dada Boudi Biriyani, 760000',
    phone: '1000000000',
    manager: 'Ankit Das'
  }
];

return (
  <Container style={{ paddingTop: '100px', paddingBottom: '50px' }}>
    {/* Header */}
    <Row className="mb-5">
      <Col className="text-center">
        <h1 className="display-4 fw-bold mb-3">Contact Us</h1>
        <p className="lead text-muted">
          Support  ||  Service  ||  Inquiries
```

```
    </p>
  </Col>
</Row>

<Row>
  {/* Contact Form */}
  <Col lg={8} className="mb-4">
    <Card className="shadow">
      <Card.Header className="bg-primary text-white">
        <h4 className="mb-0">Send us a Message</h4>
      </Card.Header>
      <Card.Body className="p-4">
        {showAlert && (
          <Alert variant="success">Message sent successfully!</Alert>
        )}

        <Form onSubmit={handleSubmit}>
          <Row>
            <Col md={6} className="mb-3">
              <Form.Label>Full Name *</Form.Label>
              <Form.Control
                type="text"
                name="name"
                value={formData.name}
                onChange={handleInputChange}
```

```
      required
    />
  </Col>
  <Col md={6} className="mb-3">
    <Form.Label>Email Address *</Form.Label>
    <Form.Control
      type="email"
      name="email"
      value={formData.email}
      onChange={handleInputChange}
      required
    />
  </Col>
</Row>

<Row>
  <Col md={6} className="mb-3">
    <Form.Label>Phone Number</Form.Label>
    <Form.Control
      type="tel"
      name="phone"
      value={formData.phone}
      onChange={handleInputChange}
    />
  </Col>
```

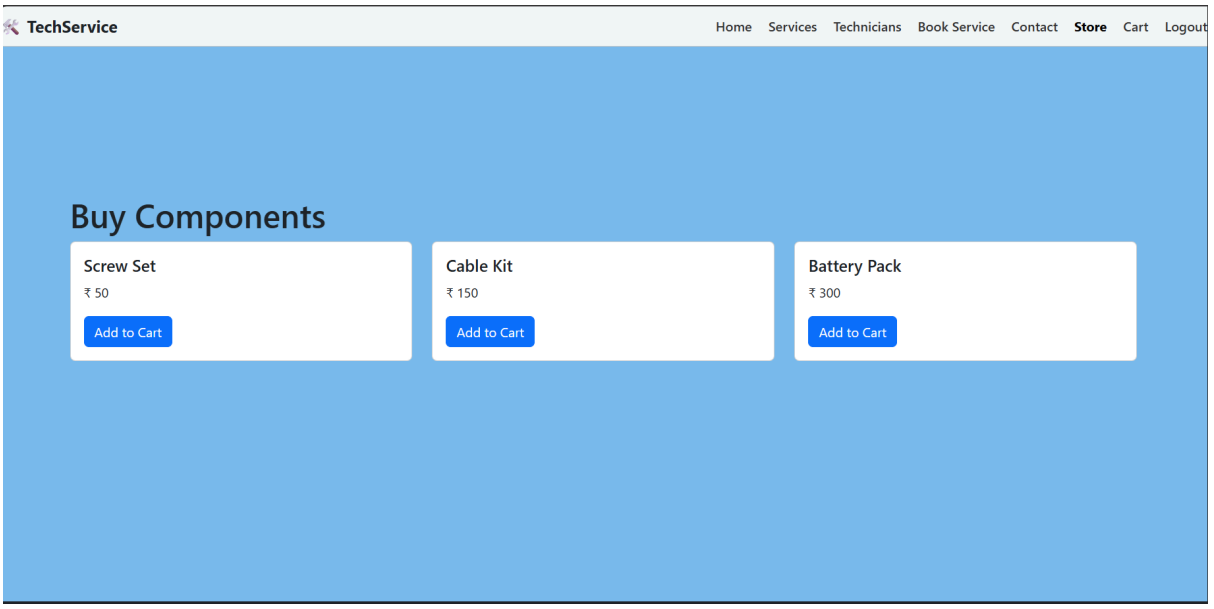
```
  <Col md={6} className="mb-3">
    <Form.Label>Subject *</Form.Label>
    <Form.Select
      name="subject"
      value={formData.subject}
      onChange={handleInputChange}
      required
    >
      <option value="">Select a subject...</option>
      <option value="general">General Inquiry</option>
      <option value="service">Service Question</option>
      <option value="support">Technical Support</option>
      <option value="complaint">Complaint</option>
      <option value="compliment">Compliment</option>
      <option value="other">Other</option>
    </Form.Select>
  </Col>
</Row>

<div className="mb-3">
  <Form.Label>Message *</Form.Label>
  <Form.Control
    as="textarea"
    rows={6}
    name="message"
```

```
            value={formData.message}
            onChange={handleInputChange}
            placeholder="Please provide details"
            required
          />
        </div>

        <Button
          type="submit"
          size="lg"
          className="w-100 btn-custom btn-primary-custom"
        >
          Send Message
        </Button>
      </Form>
    </Card.Body>
  </Card>
</Col>

{/* Contact Information */}
<Col lg={4}>
  <Row>
    {contactInfo.map((info, index) => (
      <Col sm={6} lg={12} key={index} className="mb-3">
        <Card className="h-100 text-center">
```

```
            <Card.Body>
              <div className={`service-icon bg-${info.color} mb-3`}>
                <info.icon />
              </div>
              <h6 className="fw-bold">{info.title}</h6>
              {info.details.map((detail, i) => (
                <p key={i} className="text-muted mb-0 small">{detail}</p>
              ))}
            </Card.Body>
          </Card>
        </Col>
      ))}
    </Row>
  </Col>
</Row>

{/* Service Locations */}
<Row className="mt-5">
  <Col>
    <h2 className="text-center mb-4">Our Service Locations</h2>
    <Row>
      {locations.map((location, index) => (
        <Col md={4} key={index} className="mb-3">
          <Card className="h-100">
            <Card.Header className="bg-light">
```

```
                <h6 className="mb-0 fw-bold">{location.name}</h6>
              </Card.Header>
              <Card.Body>
                <p className="mb-2">
                  <FaMapMarkerAlt className="text-primary me-2" />
                  {location.address}
                </p>
                <p className="mb-2">
                  <FaPhone className="text-success me-2" />
                  {location.phone}
                </p>
                <p className="mb-0">
                  <strong>Manager:</strong> {location.manager}
                </p>
              </Card.Body>
            </Card>
          </Col>
        ))}
      </Row>
    </Col>
  </Row>
 </Container>
 );
};
export default ContactPage
```

8. Store:

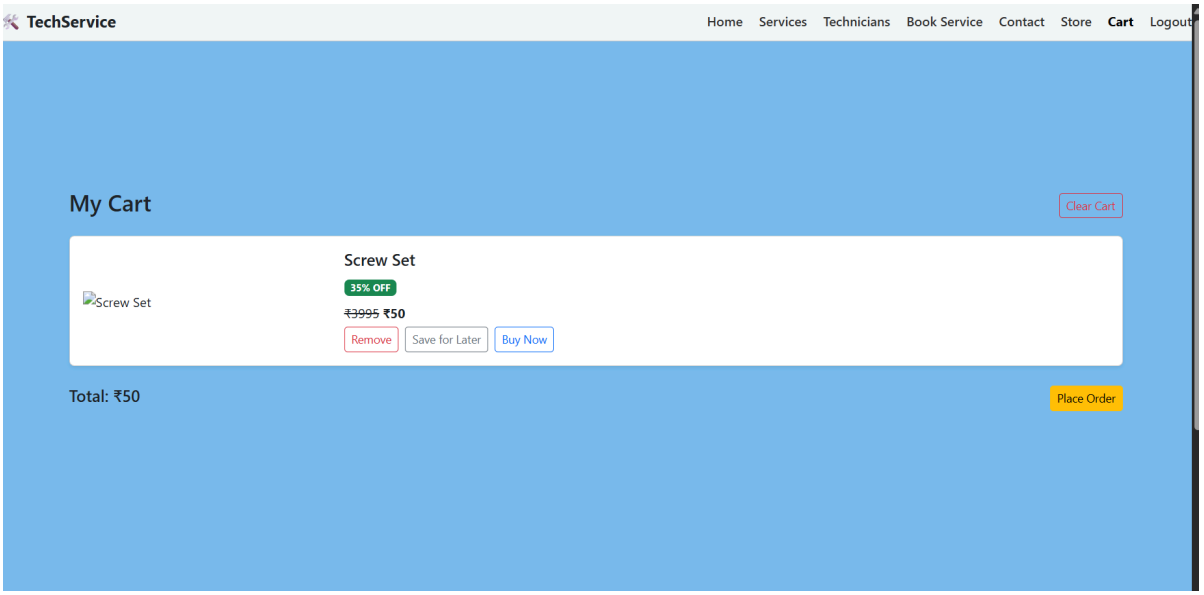


Code:

```
import React, { useState, useContext } from 'react';
import { Container, Row, Col, Card, Button, Alert } from 'react-bootstrap';
import { CartContext } from '../context/CartContext';


const StorePage = () => {
  const { addToCart } = useContext(CartContext);
  const [message, setMessage] = useState('');


  const products = [
    { id: 1, name: 'Screw Set', price: 50 },
    { id: 2, name: 'Cable Kit', price: 150 },
    { id: 3, name: 'Battery Pack', price: 300 },
  ];
  const handleAddToCart = (product) => {
```

```
    addToCart(product);
    setMessage(`${product.name} added to cart!`);
    setTimeout(() => setMessage(''), 2000);
  };
  return (
    <Container style={{ paddingTop: '100px' }}>
      <h1>Buy Components</h1>
      {message && <Alert variant="success">{message}</Alert>}
      <Row>
        {products.map((p) => (
          <Col md={4} key={p.id} className="mb-4">
            <Card>
              <Card.Body>
                <Card.Title>{p.name}</Card.Title>
                <Card.Text>₹ {p.price}</Card.Text>
                <Button onClick={() => handleAddToCart(p)}>Add to Cart</Button>
              </Card.Body>
            </Card>
          </Col>
        ))}
      </Row>
    </Container>
  );
};
export default StorePage;
```

9. Cart:



Code:

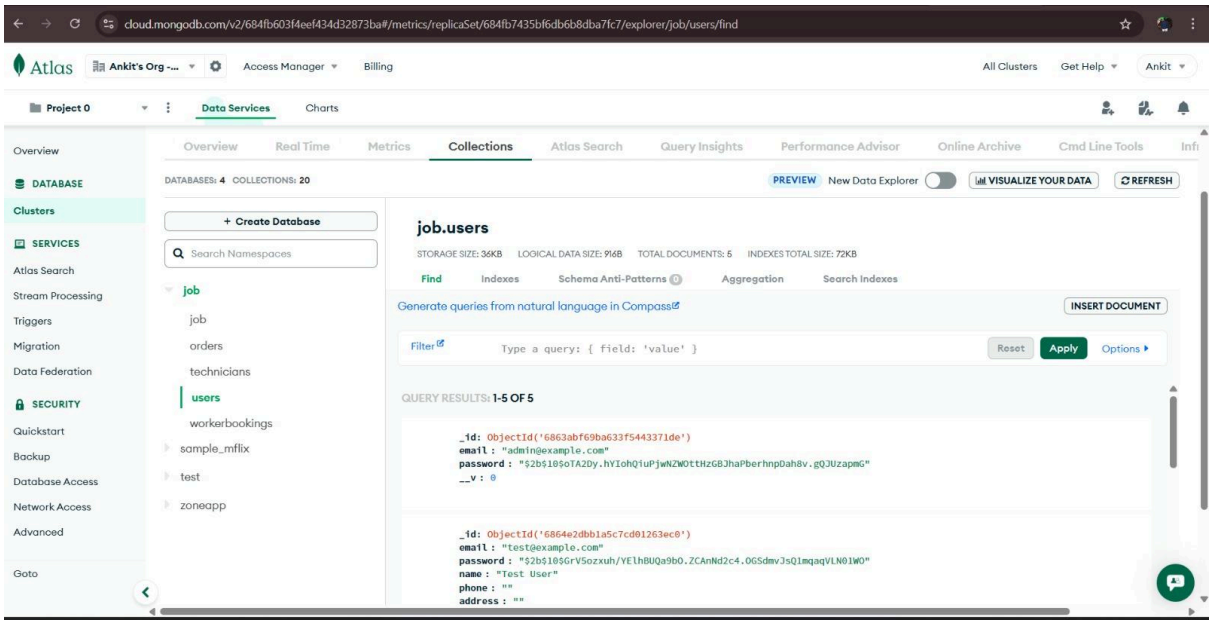
```
import React, { useContext } from 'react';
import { Container, Card, Button, Row, Col, Badge } from 'react-bootstrap';
import { CartContext } from '../context/CartContext';


const CartPage = () => {
  const { cart, removeFromCart, clearCart } = useContext(CartContext);
  // Calculate total
  const totalPrice = cart.reduce((sum, item) => sum + item.price, 0);
  return (
    <Container style={{ paddingTop: '100px' }}>
      <div className="d-flex justify-content-between align-items-center mb-3">
        <h3>My Cart</h3>
        {cart.length > 0 && (
          <Button variant="outline-danger" size="sm" onClick={clearCart}>
            Clear Cart
```

```
    </Button>
  )}
</div>
{cart.length === 0 ? (
  <p>No items in cart.</p>
) : (
  <>
    {cart.map((item, idx) => (
      <Card key={idx} className="mb-3 p-3 shadow-sm">
        <Row className="align-items-center">
          <Col xs={3}>
            {/* Placeholder image */}
            <img
              src="https://via.placeholder.com/100"
              alt={item.name}
              className="img-fluid"
            />
          </Col>
          <Col xs={9}>
            <h5>{item.name}</h5>
            <Badge bg="success" className="mb-2">35% OFF</Badge>
            <p className="mb-1">
              <del>₹3995</del> <strong>₹{item.price}</strong>
            </p>
            <div className="d-flex gap-2">
```

```
                <Button variant="outline-danger" size="sm" onClick={() =>
removeFromCart(item.id)}>
                  Remove
                </Button>
                <Button variant="outline-secondary" size="sm">
                  Save for Later
                </Button>
                <Button variant="outline-primary" size="sm">
                  Buy Now
                </Button>
              </div>
            </Col>
          </Row>
        </Card>
      ))}
      <div className="d-flex justify-content-between align-items-center mt-4">
        <h5>Total: ₹{totalPrice}</h5>
        <Button variant="warning" size="sm">Place Order</Button>
      </div>
    </>
  )}
 </Container>
 );
};
export default CartPage;
```

5B. BACKEND:

1. Database:

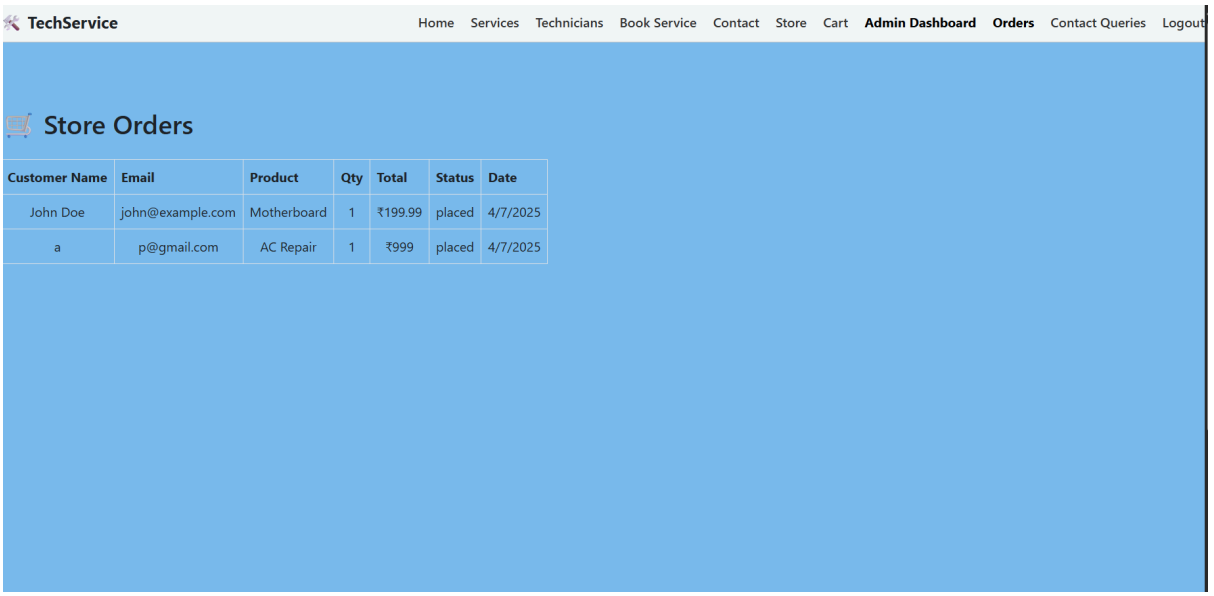


Code:

```
// backend/server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

require('dotenv').config();

const app = express();

// ✅ Middlewares

app.use(cors());

app.use(express.json());


// ✅ MongoDB connection

mongoose.connect(process.env.MONGO_URI)
```

```
  .then(() => console.log('✅ MongoDB connected'))
  .catch(err => console.error('❌ MongoDB connection error:', err));
// ✅ Routes
const workerBookingRoutes = require('./routes/workerBookings');
app.use('/api/worker-bookings', workerBookingRoutes);
const authRoutes = require('./routes/auth');
app.use('/api/auth', authRoutes);

const adminRoutes = require('./routes/admin');
app.use('/api/admin', adminRoutes);

const technicianRoutes = require('./routes/technicians'); // ✅ NEW
app.use('/api/technicians', technicianRoutes);          // ✅ NEW

const contactRoutes = require('./routes/contact');
app.use('/api/contact', contactRoutes);

const orderRoutes = require('./routes/order'); // ✅ ADD THIS
app.use('/api/orders', orderRoutes);

// ✅ Start server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`🚀 Server running on http://localhost:${PORT}`));
```

2. Store Order:



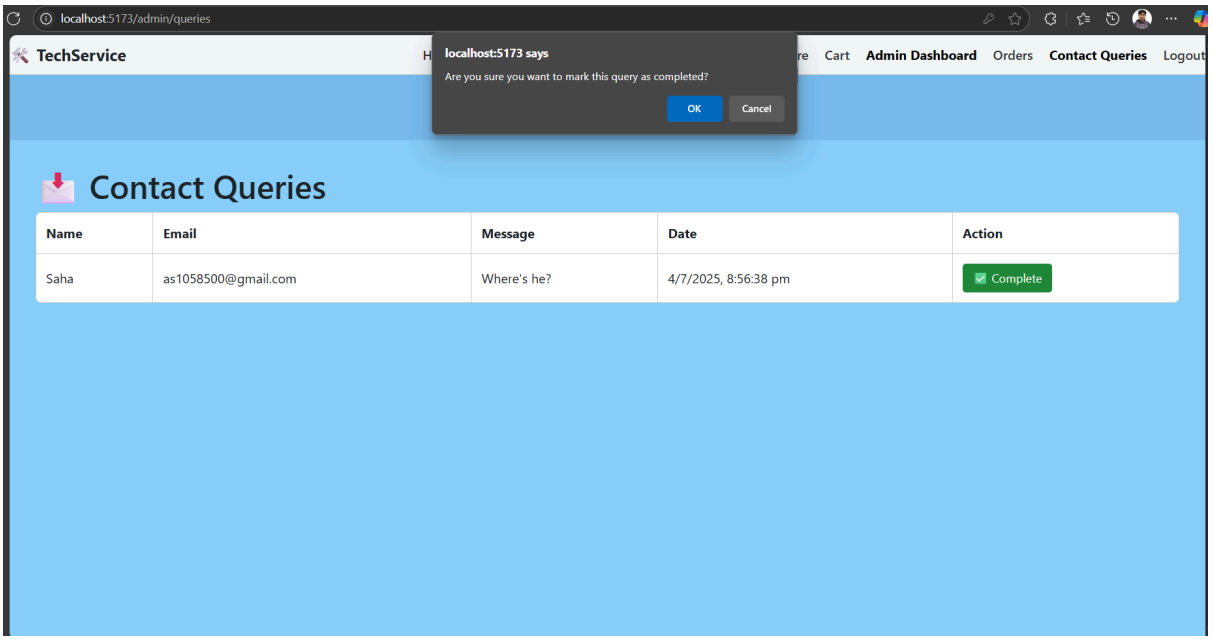
Code:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const AdminOrdersPage = () => {
  const [orders, setOrders] = useState([]);

  useEffect(() => {
    fetchOrders();
  }, []);

  const fetchOrders = async () => {
    try {
      const res = await axios.get('http://localhost:5000/api/orders/admin/all');
      setOrders(res.data);
    } catch (error) {
```

```
    console.error('Error fetching orders:', error);
  }
};

return (
  <div className="p-6">
    <h2 className="text-2xl font-semibold mb-4">🛒 Store Orders</h2>
    {orders.length === 0 ? (
      <p>No orders found.</p>
    ) : (
      <table className="min-w-full border border-gray-300">
        <thead className="bg-gray-100">
          <tr>
            <th className="p-2 border">Customer Name</th>
            <th className="p-2 border">Email</th>
            <th className="p-2 border">Product</th>
            <th className="p-2 border">Qty</th>
            <th className="p-2 border">Total</th>
            <th className="p-2 border">Status</th>
            <th className="p-2 border">Date</th>
          </tr>
        </thead>
        <tbody>
          {orders.map((order, index) => (
            <tr key={index} className="text-center">
```

```
                <td className="p-2 border">{order.customerName}</td>
                <td className="p-2 border">{order.customerEmail}</td>
                <td className="p-2 border">{order.product}</td>
                <td className="p-2 border">{order.quantity}</td>
                <td className="p-2 border">₹{order.totalPrice}</td>
                <td className="p-2 border capitalize">{order.status}</td>
                <td className="p-2 border">{new
Date(order.createdAt).toLocaleDateString()}</td>
              </tr>
            ))}
          </tbody>
        </table>
      )}
    </div>
  );
};

export default AdminOrdersPage;
```

3. Contact Queries:



Code:

```
import React, { useEffect, useState } from 'react';
import './AdminQueriesPage.css';


const AdminQueriesPage = () => {
  const [messages, setMessages] = useState([]);


  useEffect(() => {
    const fetchMessages = async () => {
      try {
        const res = await fetch('http://localhost:5000/api/contact');
        const data = await res.json();
        setMessages(data);
      } catch (err) {
        console.error('Failed to fetch messages:', err);
```

```
    }
  };
  fetchMessages();
}, []);


const handleConfirm = async (id) => {
  const confirmed = window.confirm('Are you sure you want to mark this query as completed?');
  if (confirmed) {
    try {
      const res = await fetch(`http://localhost:5000/api/contact/${id}`, {
        method: 'DELETE',
      });
      const result = await res.json();
      if (res.ok) {
        setMessages(prev => prev.filter(msg => msg._id !== id));
      } else {
        alert('Failed to delete message: ' + result.message);
      }
    } catch (err) {
      console.error('Error deleting message:', err);
      alert('Something went wrong!');
    }
  }
};
```

```
return (
  <div className="admin-queries">
    <h1>📩 Contact Queries</h1>
    <table className="contact-table">
      <thead>
        <tr>
          <th>Name</th><th>Email</th><th>Message</th><th>Date</th><th>Action</th>
        </tr>
      </thead>
      <tbody>
        {messages.map(msg => (
          <tr key={msg._id}>
            <td>{msg.name}</td>
            <td>{msg.email}</td>
            <td>{msg.message}</td>
            <td>{new Date(msg.createdAt).toLocaleString()}</td>
            <td>
              <button className="confirm-btn" onClick={() => handleConfirm(msg._id)}>
                ✅ Complete
              </button>
            </td>
          </tr>
        ))}
      </tbody>
```

```
      </table>
    </div>
  );
};

export default AdminQueriesPage;
```

6. CONCLUSION

The **Technician Service Assigning System** offers an efficient and structured solution for managing service requests, technician assignments, and task tracking in a streamlined digital environment. By leveraging modern web technologies, the system minimizes delays, reduces manual intervention, and enhances transparency across all service operations.

Through features like real-time status updates, user feedback, and admin oversight, the system ensures accountability and improves overall service quality. It empowers users to raise requests with ease, enables technicians to receive and update tasks efficiently, and equips administrators with tools to monitor and manage the entire workflow.

As service-based industries continue to digitize their operations, this system stands as a practical and scalable platform to meet growing expectations for fast, reliable, and organized service management.

7. FURTHER SCOPE AND FURTHER ENHANCEMENTS

7A. Future Scope:

The **Technician Service Assigning System** holds immense potential for scalability and innovation in the coming years. As service industries become more tech-driven, this platform can evolve to offer smarter, faster, and more intuitive solutions. Key areas of future development include:

- **AI-Based Technician Matching**: Implementing artificial intelligence to intelligently assign technicians based on availability, location, past performance, and specialization.

- **Real-Time Location Tracking**: Integration of GPS modules to track technician routes, arrival estimates, and optimize service response times.

- **Predictive Maintenance & Scheduling**: Using analytics and machine learning to predict recurring service needs, allowing preventive interventions before a breakdown occurs.

- **Mobile App Deployment**: Launching cross-platform mobile applications (Android/iOS) to allow users and technicians to interact on the go.

- **Voice-Assisted Service Requests**: Allowing customers to log service requests through voice assistants or IVR-based systems for improved accessibility.

- **Multilingual Interface**: Offering the app in multiple languages to cater to users in various regions and enhance usability.

❖7B. Further Enhancements:

To improve user satisfaction and operational efficiency, the following enhancements can be introduced in future versions:

1. User Experience Improvements

- **Role-Based Dashboards**: Tailor dashboards for users, technicians, and admins to show relevant insights and controls.
- **Accessibility Support**: Introduce dark mode, larger fonts, and screen reader compatibility for inclusive usage.
- **Push Notifications & Alerts**: Send real-time updates for task assignments, completion, and service feedback.

2. Task & Service Management

- **Multi-Level Task Escalation**: Introduce escalation matrices for unaddressed or delayed service requests.
- **Recurring Service Plans**: Allow users to subscribe to routine maintenance services on a weekly/monthly basis.
- **Digital Signature & Work Logs**: Capture digital proof of task completion with e-signatures and technician logs.

3. Communication & Feedback

- **In-App Chat/Support**: Enable live communication between users and assigned technicians for better issue resolution.
- **Star Ratings & Reviews**: Let users rate technicians and provide comments post-service.
- **Feedback Analytics**: Aggregate user feedback into performance dashboards for admins.

4. Data & Analytics

- **Service History Dashboard**: Show complete logs of past service requests and resolutions.
- **Technician Performance Tracking**: Generate reports on efficiency, response times, and customer satisfaction.
- **Predictive Insights**: Suggest optimization strategies using historical service data.

5. Technology Integrations

- **Cloud Storage & Syncing**: Ensure data reliability, automatic backups, and real-time syncing across platforms.
- **Offline Request Logging**: Allow users to raise requests without internet — auto-sync when reconnected.
- **Integration with CRM/ERP Systems**: Seamless flow of customer and service data into enterprise platforms.

8. BIBLIOGRAPHY

➢ W3Schools – https://www.w3schools.com

➢ YouTube – https://www.youtube.com

➢ Pexels (Stock Images) – https://www.pexels.com

➢ CodePen – https://www.codepen.io

➢ Google Search – https://www.google.com

➢ Google Fonts – https://fonts.google.com

➢ React Official Documentation – https://react.dev

➢ Node.js Documentation – https://nodejs.org

➢ Express.js Guide – https://expressjs.com

➢ MongoDB Atlas – https://www.mongodb.com/cloud/atlas

➢ Postman API Platform – https://www.postman.com

➢ GitHub – https://github.com

➢ Visual Studio Code – https://code.visualstudio.com

➢ Coding World – https://www.codingworld.com