Zintargo: e-commerce website

A Project Report for Industrial Training and Internship

submitted by

Manish Kumar Das

In the partial fulfillment of the award of the degree of

B.Tech

in the

Computer Science & Engineering

Of

Sandip University

At

Ardent Computech Pvt. Ltd.

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

ISO 9001:2015 CERTIFIED

UDYAM REGISTRATION
MSME

N·E·A·T
National Educational Alliance for Technology

CERTIFICATE FROM SUPERVISOR

This is to certify that **Manish Kumar Das, PRN:-230205131122** have completed the project titled Zintargo : e-commerce websiteRental Platform under my supervision during the period from “03-07-2025” to “02-08-2025” which is in partial fulfillment of requirements for the award of the **B.Tech** degree and submitted to the Department of **Computer Science & Engineering** of **Sandip University**.

____

Signature of the Supervisor

Date:dd/mm/yy

Name of the Project Supervisor:

22 YEARS OF EXCELLENCE

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

CERTIFIED ISO 9001:2015 CERTIFIED

UDYAM REGISTRATION

N·E·A·T
National Educational Alliance for Technology

BONAFIDE CERTIFICATE

Certified that this project work was carried out under my supervision

Zintargo: e commerce website is the bonafide work of

Name of the student:	***Manish kumar Das***
Name of the student:	***Ashish kumar***
Name of the student:	***Navnnet kumar***

SIGNATURE

Name :

PROJECT MENTOR

SIGNATURE Name:

EXAMINERS

Ardent Original Seal

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

CERTIFIED ISO 9001:2015 CERTIFIED

UDYAM REGISTRATION
MSME

N·E·A·T
National Educational Alliance for Technology

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 organization 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 customized 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. <u>INTRODUCTION</u>

In today's fast-paced world, the need for flexible, affordable, and convenient shopping is more critical than ever. With increasing urbanization and the rising costs of traditional retail, people are seeking smarter commerce solutions. This project, **Zintargo: E-commerce Website**, aims to address these needs by providing an efficient and user-friendly system for buying and selling products online.

Powered by the MERN (MongoDB, Express.js, React.js, Node.js) stack, Zintargo is designed to offer a seamless digital experience for users to browse, purchase, and manage orders directly from their devices. The platform includes features like real-time product availability, user authentication, order history, and secure online payments through Razorpay integration.

By connecting sellers and buyers through a single platform, this project promotes digital commerce, reduces dependency on physical stores, and supports a convenient shopping experience. Whether for daily essentials, fashion, electronics, or more, Zintargo provides a reliable solution that empowers users to shop freely—anytime, anywhere.

2A. OBJECTIVE

The primary objective of **Zintargo: E-commerce Website** is to revolutionize the way people shop by offering a smart, reliable, and accessible digital marketplace. The platform is designed to cater to individuals who seek convenient and flexible online shopping options—whether for daily essentials, occasional purchases, or leisure—without the limitations of traditional retail.

Zintargo aims to bridge the gap between consumer needs and affordability by providing a user-friendly system that allows users to browse, compare, and purchase products with ease. Through real-time product availability, secure payment systems, and seamless checkout experiences, the platform empowers users to make confident shopping decisions.

Ultimately, Zintargo seeks to promote digital commerce, reduce the friction of in-person shopping, and enhance the overall consumer experience—enabling users to shop smart, save more, and buy conveniently—anytime, anywhere.

2B.SCOPE

Our project focuses on creating a web-based application that offers a wide range of products to meet the shopping needs of users from different backgrounds and age groups

1. **User Registration & Log**
   Secure sign-up/login for Users and admins

2. **Interactive Content:**
   Easily browse products place orders,and track purchase history.

3. **User Profile Dashboard :**
   View personal details,saved items,and manage orders.

4. **Admin Profile :**
   Controls for product management,user oversight order approvals, reports, and analytics.

5. **Platform Access :**
   Available via all major web browsers with responsive design.

6. **Payment Integration :**
   Secure payment integration, one-time payments, using Razorpay

3.SYSTEM ANALYSIS

3A. IDENTIFICATION OF NEED

System analysis plays a crucial role in the development of our project, Zintargo: E-commerce Website, with a focus on building an efficient, user-friendly, and scalable platform to meet modern shopping needs. With the rapid growth of digital commerce, increasing consumer expectations, and the shift toward online shopping, there is a clear demand for a flexible and reliable e-commerce solution. Traditional retail experiences often lack convenience, product variety, and 24/7 accessibility—especially for users in remote or busy urban areas.

Zintargo addresses these challenges by offering a smart, web-based platform that allows users to browse, purchase, and manage a wide range of products easily. The system is designed to deliver a seamless shopping experience that is both convenient and accessible, supporting secure payments, efficient order tracking, and personalized user interactions.

Key Needs Identified:

Affordable and Flexible Shopping:

- Many customers look for budget-friendly products and flexible purchasing options.
- Users need access to a variety of products with options for one-time purchases, discounts, and seasonal offers.

2. Real-Time Product Availability and Ordering:

- The platform should display real-time product stock levels and allow instant ordering and cancellations.
- Seamless navigation, filtering, and location-based delivery options are essential features.

3. Secure Transactions and User Management:

- A secure and user-friendly payment system is essential for smooth transactions.
- Both users and admins require dashboards to manage orders, payment history, and product inventory effectively.

4. Convenient and Customer-Friendly Shopping Experience:

- The platform promotes easy access to products, timely deliveries, and hassle-free returns to enhance customer satisfaction

1.

3B.Project Feasibility Analysis

The feasibility study of our Zintargo: E-commerce Website indicates that the concept is both practical and promising across several key dimensions.
Technical Feasibility:
The platform can be effectively developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), providing a scalable and responsive web application framework. Essential features like real-time product updates, secure user authentication, and payment integration via Razorpay are readily achievable with current technologies.
Economic Feasibility:
Zintargo has strong revenue potential through product sales, subscription services for premium users, and partnerships with vendors and suppliers. The operational costs are sustainable, with multiple avenues for monetization and expansion.
Operational Feasibility:
The platform is designed for ease of use and scalability, meeting the needs of customers and administrators alike. Features such as order management, inventory controls, and user/admin dashboards ensure smooth operations and an excellent user experience.
Legal Feasibility:
Zintargo complies with relevant e-commerce regulations, data protection laws such as the IT Act, and includes clear terms of service to address user rights, privacy, and liability concerns.
Timeline:
The expected development and launch timeline is approximately 4 to 6 months, making the project achievable within a practical timeframe and budget.

3C.WORKFLOW

This document plays a crucial role in the **Software Development Life Cycle (SDLC)** as it clearly outlines the complete system requirements. It is intended for developers and will serve as the foundation during the testing phase. Any modifications to these requirements in the future must undergo a formal change approval process.

The **Waterfall Model** was the first process model introduced in software engineering. Also known as the **linear-sequential life cycle model**, it is straightforward and easy to use. In this model:

- Each phase must be fully completed before the next phase begins.
- There is no overlap between phases.
- The output of one phase acts as the input for the next phase.

The Waterfall Model illustrates the software development process as a **linear sequential flow**. This means that work on a new phase only starts once the previous phase is fully completed, ensuring a clear and structured approach.

☐ **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" 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 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 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 requirement specification doc.

➢ **System Design:** The requirement specifications from the first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining 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 is started only after the defined set of goals are achieved for the previous phase and it is signed off, so the name "Iterative Waterfall Model". In this model, phases do not overlap.

- **<u>Advantages:</u>**

1 . Flexibility: Iterations permit adjustments based on feedback.

2 . Early Delivery: Partial systems can be delivered incrementally.

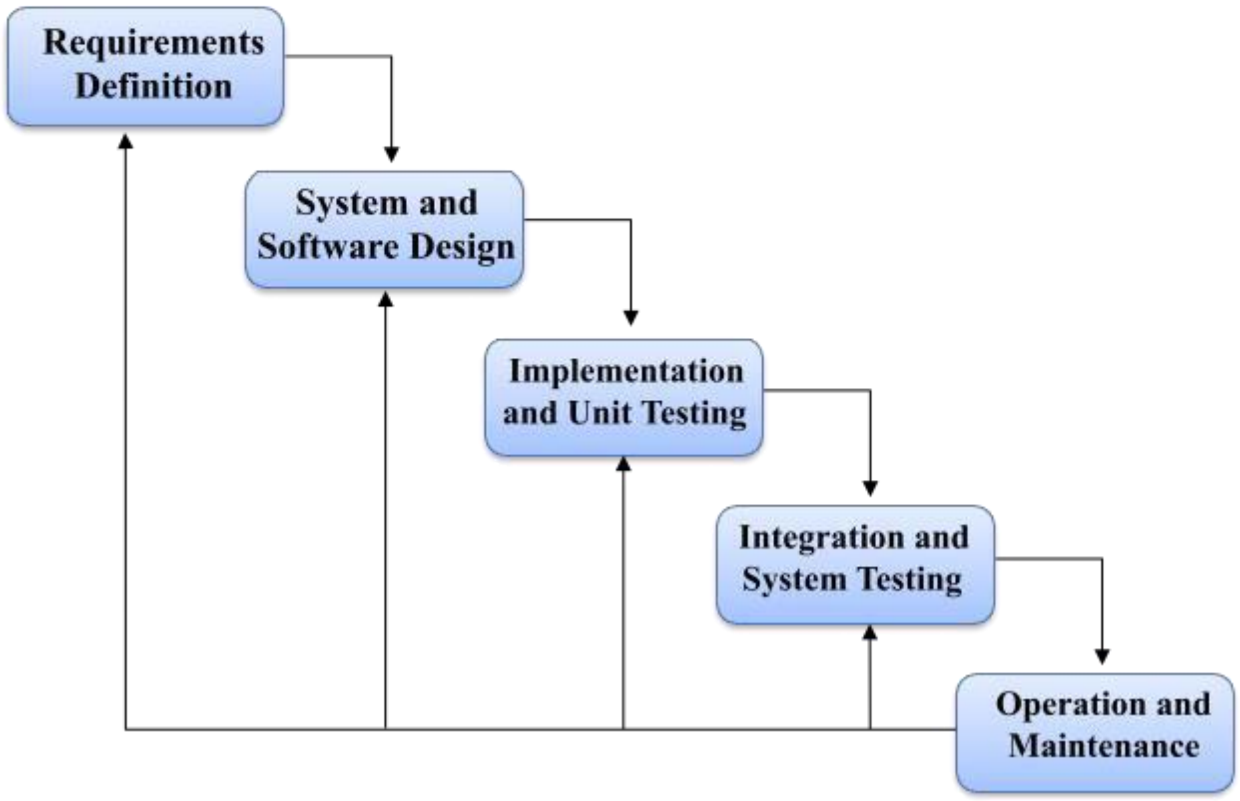
3 . Risk Management: Identifying and addressing issues early in the process.

- **<u>Disadvantages</u>:**

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

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

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


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- **<u>Applications</u>:**

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**: Here, the user will register to buy any vehicles.
  2 **. Admin Register:** Here, the admin will register to handle all the databases.

- **Verify Account:** Without registration and login user not access any content of website

- **Login:**
  1 **. User Login:** Here users will login to see available vehicles and rent any vehicle.
  2 . **Admin Login:** Here admin will log in to handle all the data.

- **Home:** This page is to see all vehicles received from users.

- **Profile:**
  1 **User interface:** This page shows the user profile and booking history ,
  2 **Admin interface:** In this page, the Admin can view or delete vehicles.

- **About:** This page will show the details about the website developers.
- **Contact:** This page will show the contact details.

- **Account:**
  1 **User Interface:**
    a) **Home:** Here, the user can see the details after login.
    b) **Profile:** Here, the user can see the vehicles he/she has rented.
  2 **Admin Interface:**
    a) **Admin Profile:** Here, the Admin can see the details after login.
    b) **Admin Dashboard:** Here the Admin can manage vehicles and user roles.

3E.INPUT AND OUTPUT

The primary inputs, outputs, and core functionalities of the **Zintargo – E-Commerce Website** are as follows:

□ **Input**

1. **User Registration/Login:**
   - Customers can create an account or log in by providing credentials such as name, email, password, and contact details on the registration or login page.
2. **Product Search & Filters:**
   - Users can enter search parameters such as product name, category, price range, brand, and ratings to find suitable products.
3. **Admin Management Functions:**
   - Admins can input new product details, update inventory, modify pricing, manage user accounts, and process order statuses via the admin panel.

□ **Output**

- **For Users:**
  - View product listings matching applied filters.
  - Place orders, view order confirmations, and receive payment details.
  - Access order history, profile information, and order delivery status in real time.
- **For Admins:**
  - Access a centralized dashboard displaying user accounts, active orders, inventory status, and sales summaries.
  - Generate reports for sales performance, product trends, and payment records.

3F.SOFTWARE REQUIREMENT SPECIFICATIONS

Software Requirements Specification provides an overview of the entire project. It is a description of a software system to be developed, laying out functional and non-functional requirements. The software requirements specification document enlists enough necessary requirements that are required for the project development. To derive the requirements we need to have a clear and thorough understanding of the project to be developed. This is prepared after detailed communication with the project team and the customer.

The developer is responsible for:

- Developing the system, which meets the SRS and solves all the requirements of the system.
- Demonstrating the system and installing the system at the client's location after acceptance testing is successful.
- Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.

Functional Requirements:

A. **User Registration and Authentication:**
    1. Users should be able to create accounts securely.
    2. The system should authenticate users and manage login sessions.

B. **Browse and Search:**
    1. Users should be able to browse and search for vehicle.

.

C. **Vehicles Display:**
    1. Each Student should have detailed and up-to-date items with prices.
    2. Users should be able to view vehicles.

Hardware Requirements:

1 . Computer has Intel I3 Processor

2 . 8 GB RAM

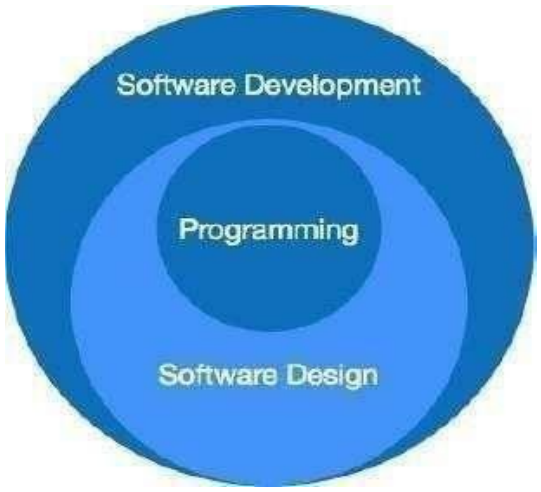
3. SSD-ROM Drive

Software Requirements:

1. Kubuntu Linux OS
2. Visual Studio Code
3. Mongo DB Atlas

3G.SOFTWARE ENGINEERING PARADIGM APPLIED

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in software engineering these paradigms stand. These can be combined into various categories, though each of them is containedin one another.



The programming paradigm is a subset of Software design paradigm which is further a subset of the Software development paradigm.

There are two levels of reliability. The first is meeting the right requirements. A careful and thorough systems study is needed to satisfy this aspect of reliability. The second level of systems reliability involves the actual work delivered to the user. At this level, the system's reliability is interwoven with software engineering and development.

There are three approaches to reliability.

1. **Error avoidance:** Prevents errors from occurring in software.
2. **Error detection and correction:** In this approach, errors are recognized whenever they are encountered, and correcting the error by the effect of the error of the system does not fail.
3. **Error to lerance:** In this approach, errors are recognized whenever they occur, but enables the system to keep running through degraded performance or Applying values that instruct the system to continue process.

4. SYSTEM DESIGN

4A. DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated.

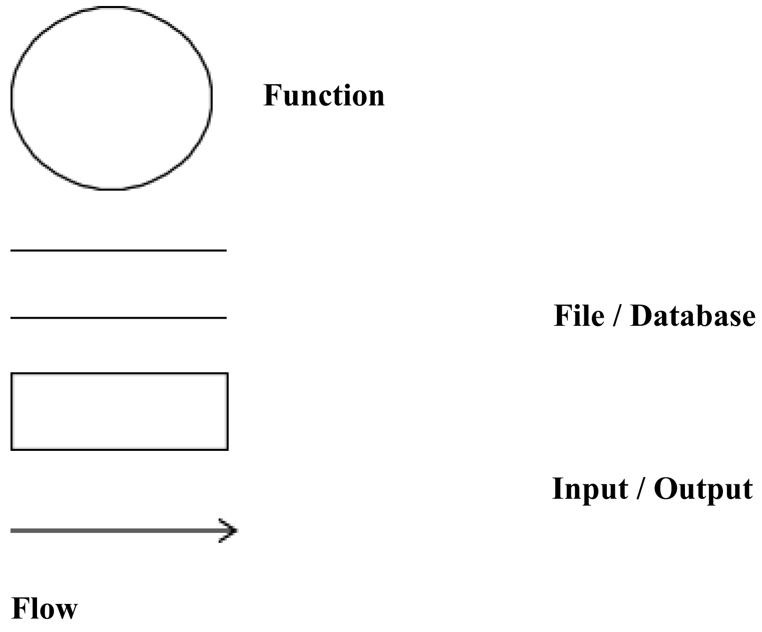
DFD can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of the process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present for the system to do its job and shows the flow of data between the various parts of the system.

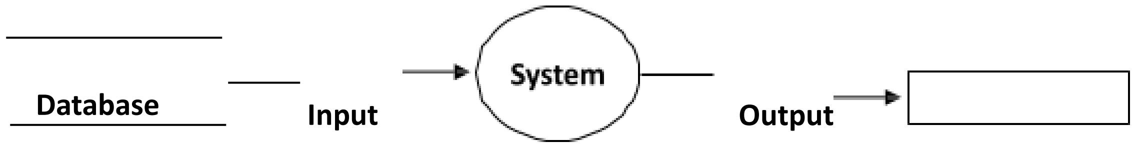
Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data-flow diagrams can be drawn up and compared with.

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 phase 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:



Steps to Construct Data Flow Diagram:

Four Steps are generally used to construct a DFD.

- ☐ Process should be named and referred for easy reference. Each name should be representative of the reference.
- ☐ The destination 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:

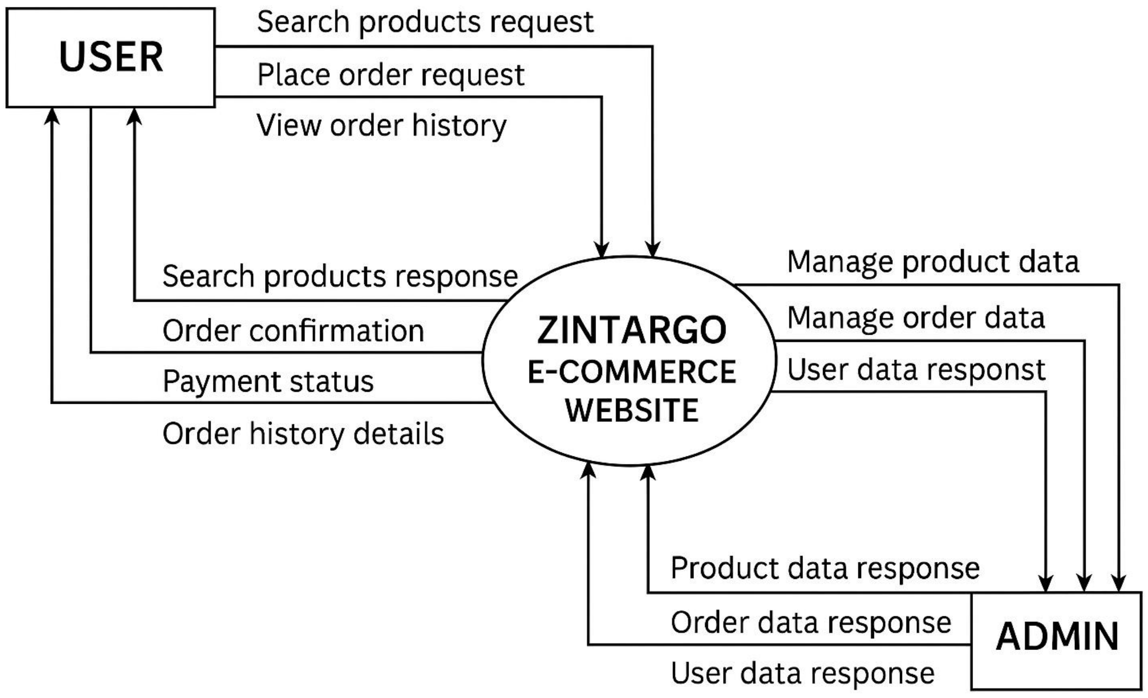
☐ Arrows should not cross each other.

☐ Squares, Circles, and Files must bear a name.

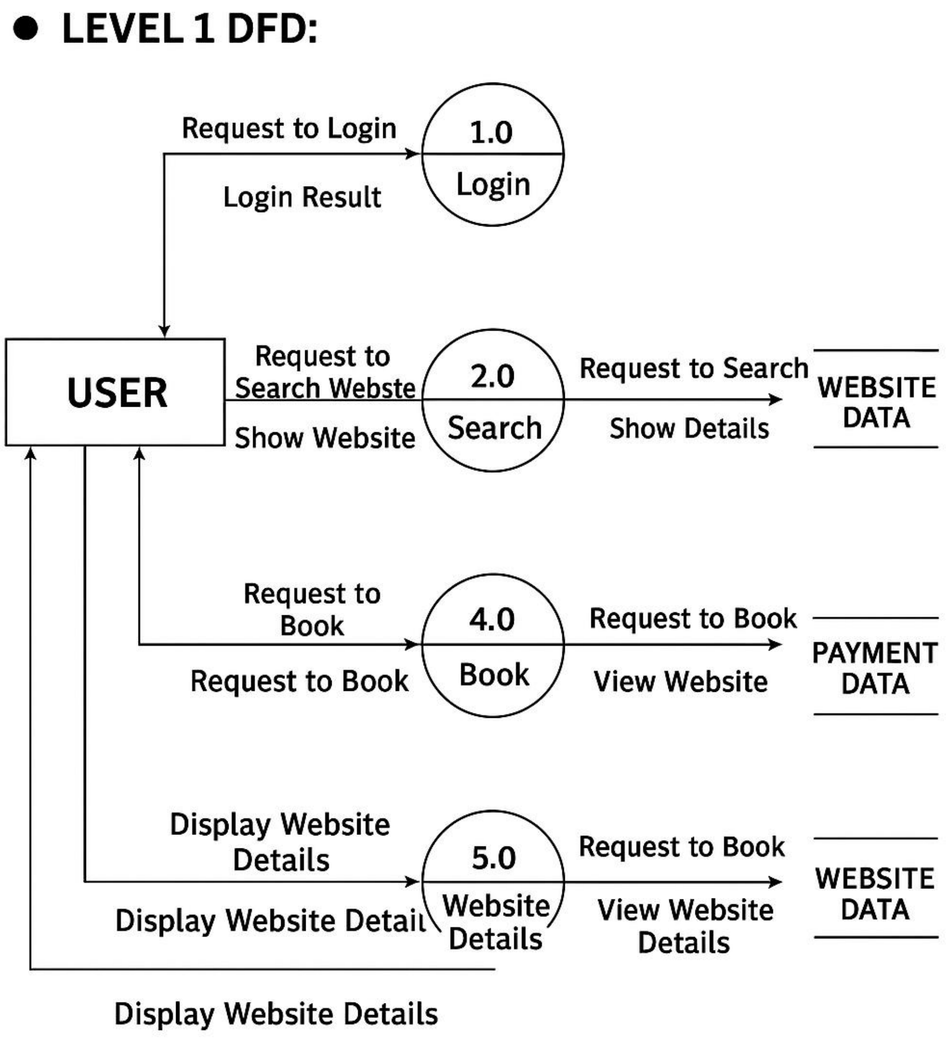
☐ Decomposed data flow squares and circles can have the same names.

☐ Draw all data flow around the outside of the diagram.

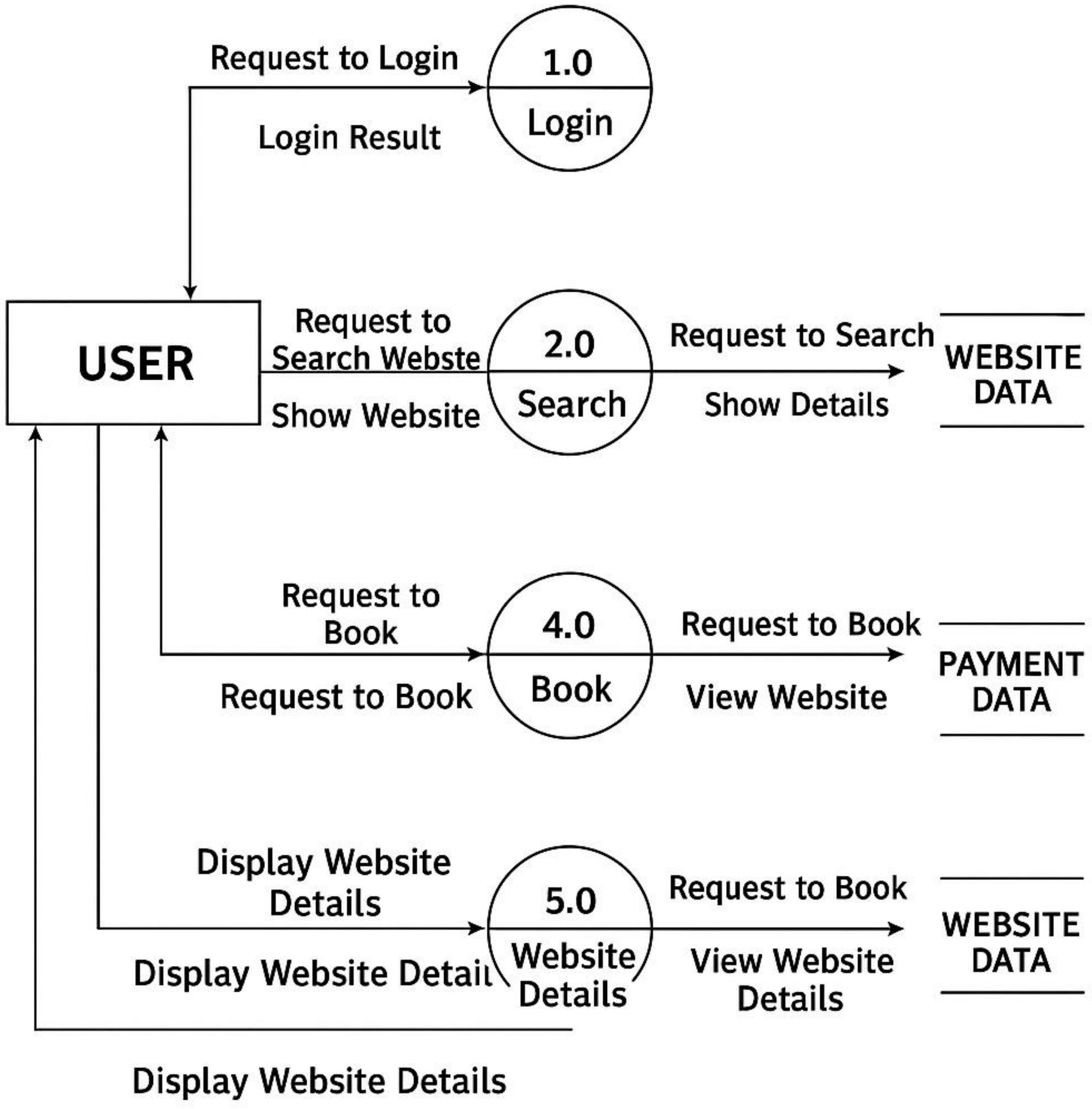
● **LEVEL 0 DFD OR CONTEXT DIAGRAM:**



● **LEVEL 1 DFD:**


● LEVEL 1 DFD:
Request to Login
1.0
Login
Login Result
USER
Request to Search Webste
2.0
Search
Show Website
Request to Search
Show Details
WEBSITE DATA
Request to Book
4.0
Book
Request to Book
Request to Book
View Website
PAYMENT DATA
Display Website Details
5.0
Website Details
Display Website Detail
Request to Book
View Website Details
WEBSITE DATA
Display Website Details

- **LEVEL 1 DFD:**



Request to Login
1.0
Login
Login Result
USER
Request to Search Webste
2.0
Search
Show Website
Request to Search
Show Details
WEBSITE DATA
Request to Book
4.0
Book
Request to Book
Request to Book
View Website
PAYMENT DATA
Display Website Details
5.0
Website Details
Display Website Detail
Request to Book
View Website Details
WEBSITE DATA
Display Website Details

4B.SEQUENCE DIAGRAM

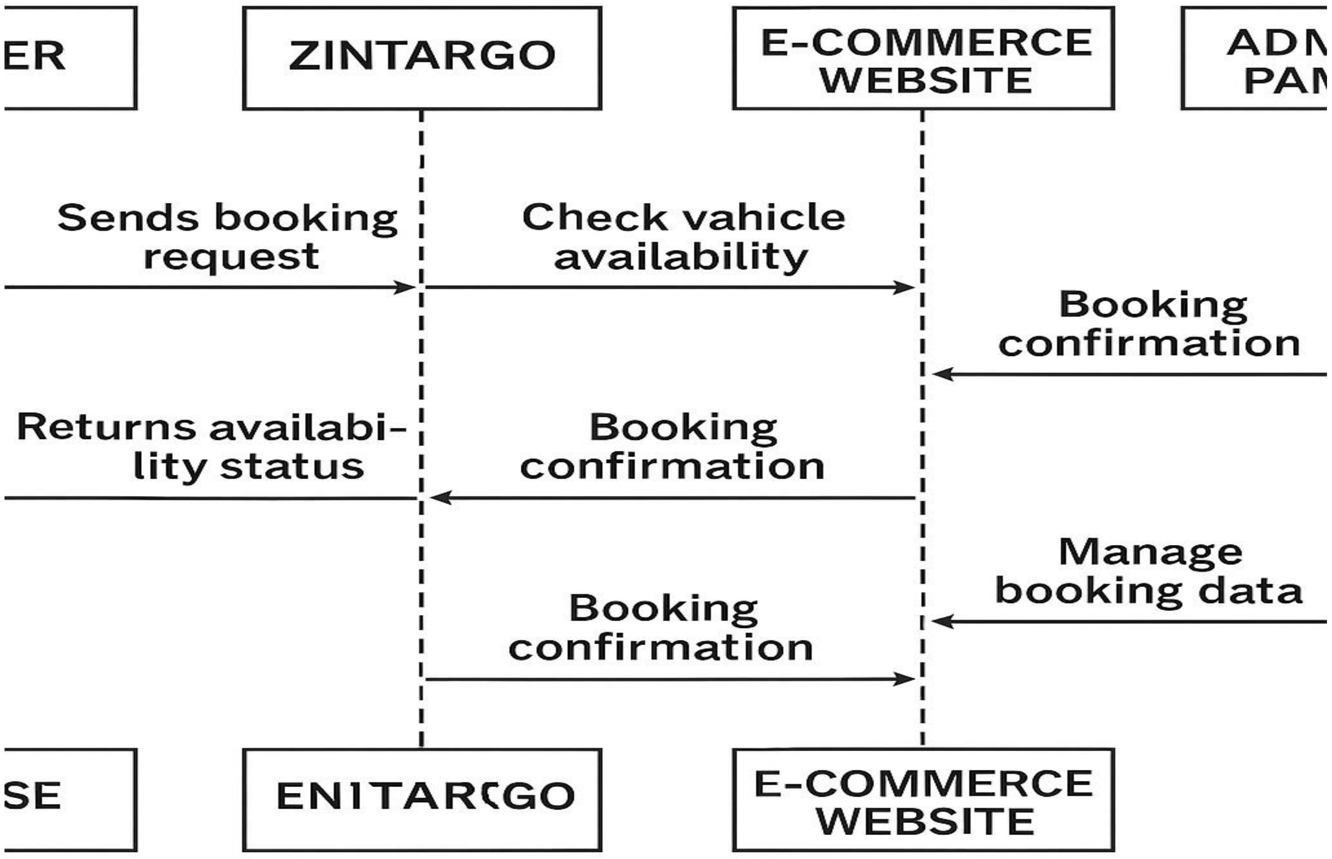
A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.

Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

A sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between several life lines .A sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

The following nodes and edges are typically drawn in a UML sequence diagram: lifeline, execution specification, message, fragment, interaction, state invariant, continuation, and destruction occurrence.



A Use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the use rand the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and a use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So, use case diagrams consist of actors, use cases, and their relationships. The diagram is used to model the system/subsystem of an application. A single-use case diagram captures a particular functionality of a system.

So, to model the entire system numbers of use case diagrams are used. The purpose of a use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because the other four diagrams (activity, sequence, collaboration, and State chart) are also having the same purpose. So, we will look into some specific purpose that will distinguish it from the other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So, when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modeled to present the outside view. So, in brief, the purposes of use case diagrams can be as follows:

- ☐ Used togather requirements of a system.
- ☐ Used to get an outside view of a system.
- ☐ Identify external and internal factors influencing the system.

How to draw Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analyzed, the functionalities are captured in use cases.

So, we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

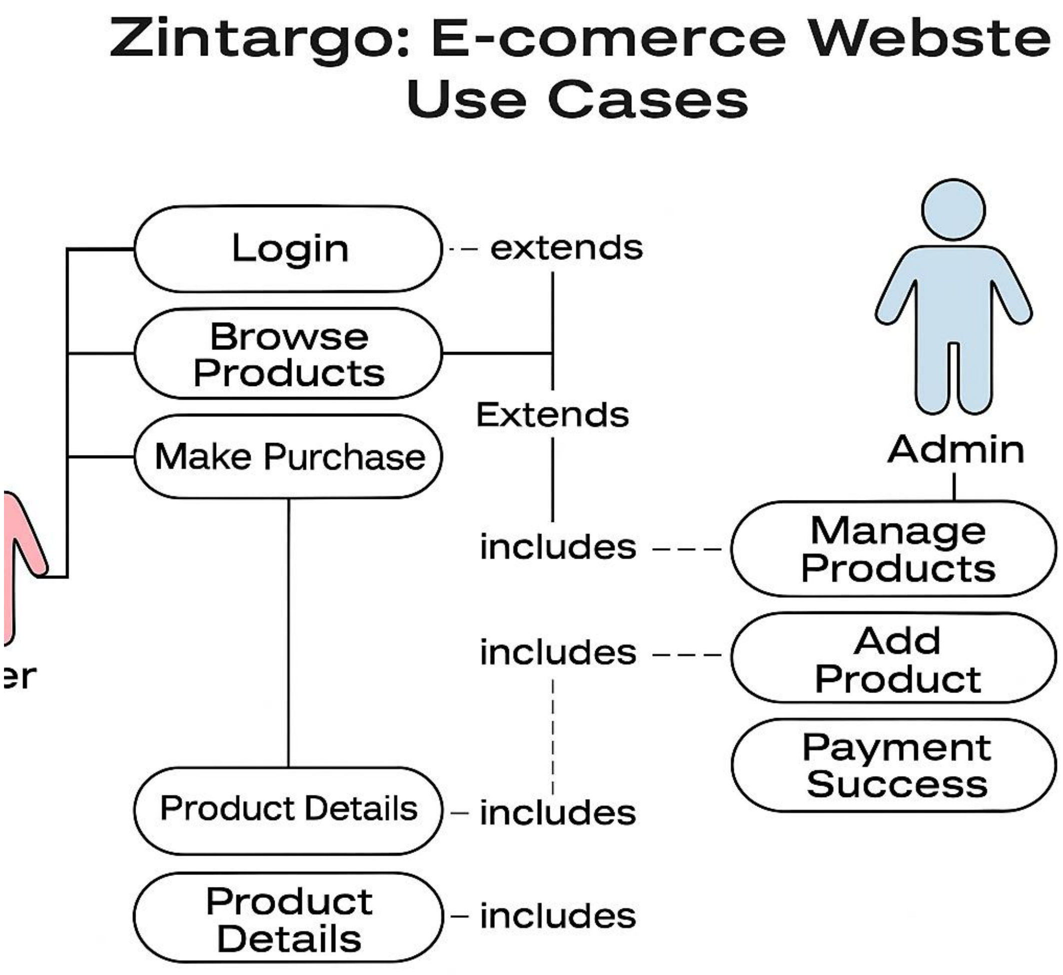
The actors can be human user, some internal applications or may be some external applications. So, in a brief when we are planning to draw use case diagram, we should have the following items identified.

- ☐ Functionalities to be represented as a use case
- ☐ Actors
- ☐ Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So, after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

- ☐ The name of a use case is very important. So, the name should be chosen in such a way so that it can identify the functionalities performed.
- ☐ Give a suitable name for actors.
- ☐ Show relationships and dependencies clearly in the diagram.
- ☐ Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
- ☐ Use note whenever required to clarify some important point

● **USE CASE DIAGRAM:**

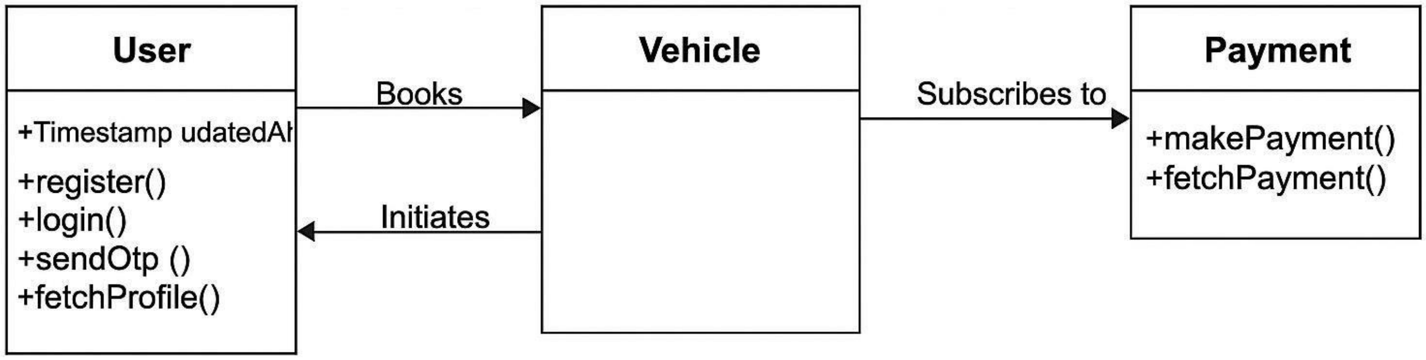


4D.SCHEMA DIAGRAM

The schema is an abstract structure or outline representing the logical view of the database as a whole. Defining categories of data and relationships between those categories, database schema design makes data much easier to retrieve, consume, manipulate, and interpret.

DB schema design organizes data into separate entities, determines how to create relationships between organized entities, and influences the applications of constraints on data. Designers create database schema to give other database users, such as programmers and analysts, a logical understanding of data.

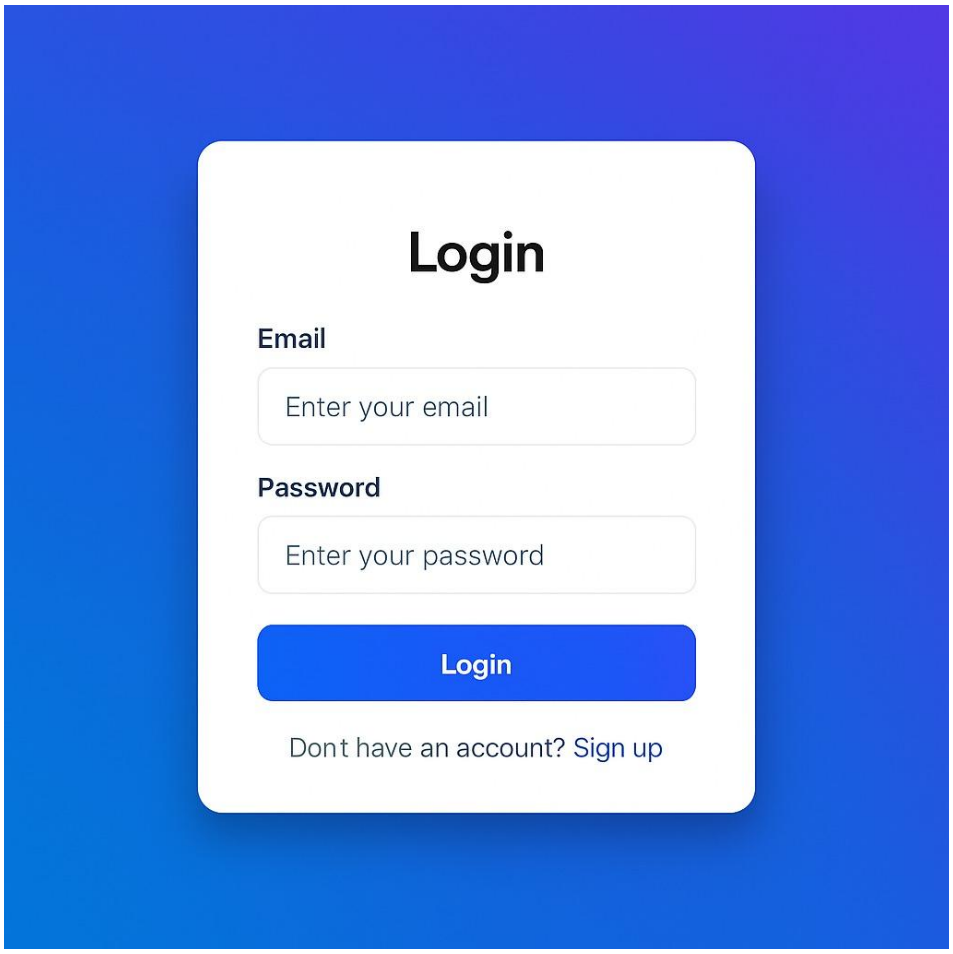
- **SCHEMA DESIGN:**



UI SNAPSHOT

❖ **FRONTEND : -**

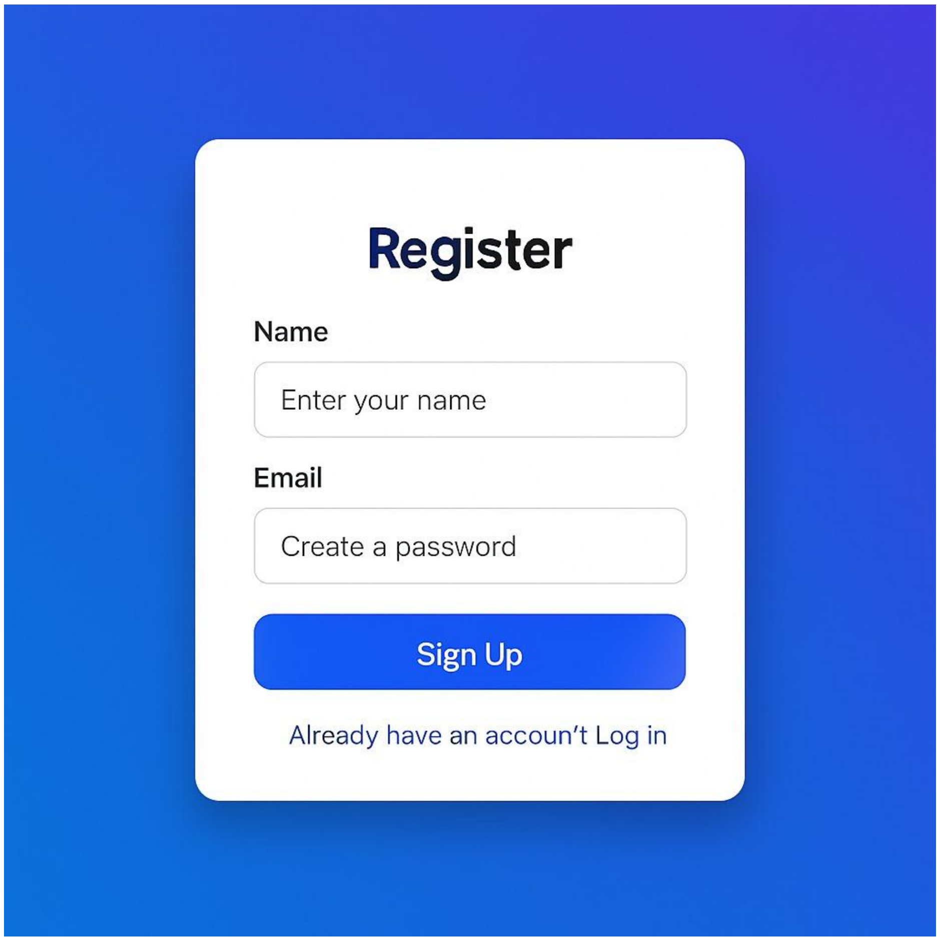
1) <u>Login Page:</u>



✓ <u>CODE</u>

```
import CommonForm from "@/components/common/form";
import { useToast } from "@/components/ui/use-toast";
import { loginFormControls } from "@/config";
import { loginUser } from "@/store/auth-slice";
import { useState } from "react";
import { useDispatch } from "react-redux";
import { Link } from "react-router-dom";

const initialState = {
  email: "",
  password: "",
};

function AuthLogin() {
  const [formData, setFormData] = useState(initialState);
  const dispatch = useDispatch();
  const { toast } = useToast();

  function onSubmit(event) {
    event.preventDefault();

    dispatch(loginUser(formData)).then((data) => {
      if (data?.payload?.success) {
        toast({
          title: data?.payload?.message,
        });
      } else {
        toast({
          title: data?.payload?.message,
          variant: "destructive",
        });
      }
    });
  }

  return (
    <div className="mx-auto w-full max-w-md space-y-6">
      <div className="text-center">
        <h1 className="text-3xl font-bold tracking-tight text-foreground">
          Sign in to your account
        </h1>
        <p className="mt-2">
          Don't have an account
          <Link
            className="font-medium ml-2 text-primary hover:underline"
            to="/auth/register"
          >
            Register
          </Link>
        </p>
      </div>
      <CommonForm
        formControls={loginFormControls}
        buttonText={"Sign In"}
        formData={formData}
        setFormData={setFormData}
        onSubmit={onSubmit}
      />
    </div>
  );
}

export default AuthLogin;
```

Register Page:



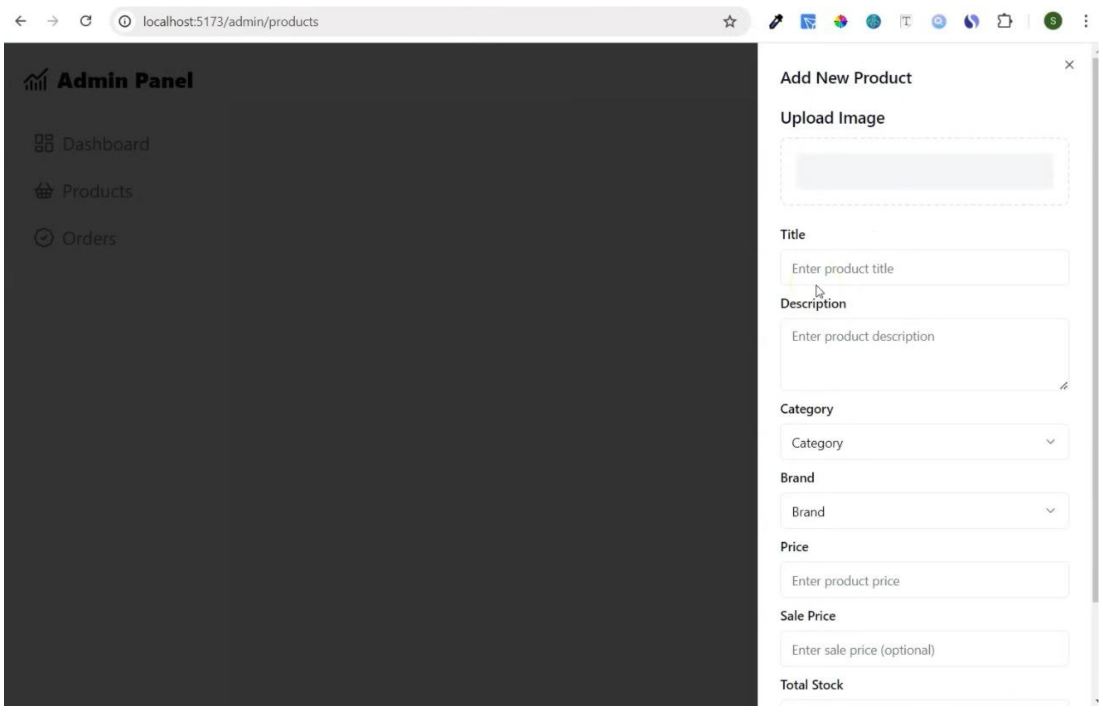
✓ **CODE**

```
import CommonForm from "@/components/common/form";
import { useToast } from "@/components/ui/use-toast";
import { registerFormControls } from "@/config";
import { registerUser } from "@/store/auth-slice";
import { useState } from "react";
import { useDispatch } from "react-redux";
import { Link, useNavigate } from "react-router-dom";

const initialState = {
  userName: "",
  email: "",
  password: "",
};
```

```
function AuthRegister() {
  const [formData, setFormData] = useState(initialState);
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const { toast } = useToast();

  function onSubmit(event) {
    event.preventDefault();
    dispatch(registerUser(formData)).then((data) => {
      if (data?.payload?.success) {
        toast({
          title: data?.payload?.message,
        });
        navigate("/auth/login");
      } else {
        toast({
          title: data?.payload?.message,
          variant: "destructive",
        });
      }
    });
  }

  console.log(formData);

  return (
    <div className="mx-auto w-full max-w-md space-y-6">
      <div className="text-center">
        <h1 className="text-3xl font-bold tracking-tight text-foreground">
          Create new account
        </h1>
        <p className="mt-2">
          Already have an account
          <Link
            className="font-medium ml-2 text-primary hover:underline"
            to="/auth/login"
          >
            Login
          </Link>
        </p>
      </div>
      <CommonForm
        formControls={registerFormControls}
        buttonText={"Sign Up"}
        formData={formData}
        setFormData={setFormData}
        onSubmit={onSubmit}
      />
    </div>
```

```
      );
    }

export default AuthRegister;
```

2) USER ADD PAGE(for admin):


Add New Product

Upload Image

Title

Enter product title

Description

Enter product description

Category

Category

Brand

Brand

Price

Enter product price

Sale Price

Enter sale price (optional)

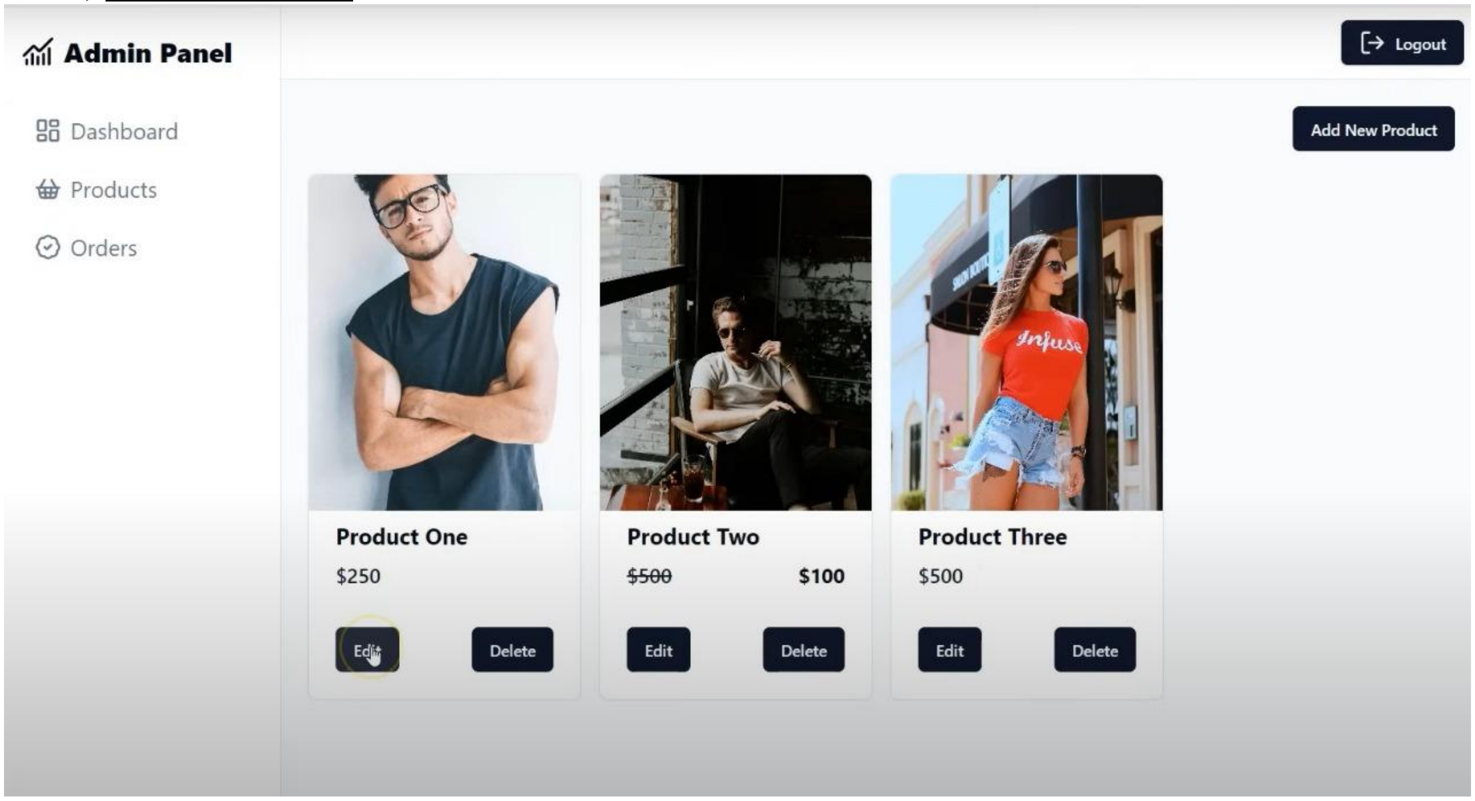
Total Stock

CODE:-

```
 import ProductImageUpload from "@/components/admin-view/image-upload";
import AdminProductTile from "@/components/admin-view/product-tile";
import CommonForm from "@/components/common/form";
import { Button } from "@/components/ui/button";
import {
  Sheet,
  SheetContent,
  SheetHeader,
  SheetTitle,
} from "@/components/ui/sheet";
import { useToast } from "@/components/ui/use-toast";
import { addProductFormElements } from "@/config";
import {
  addNewProduct,
  deleteProduct,
  editProduct,
  fetchAllProducts,
} from "@/store/admin/products-slice";
import { Fragment, useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";

const initialFormData = {
  image: null,
  title: "",
  description: "",
  category: "",
  brand: "",
  price: "",
  salePrice: "",
  totalStock: "",
  averageReview: 0,
};

function AdminProducts() {
  const [openCreateProductsDialog, setOpenCreateProductsDialog] =
    useState(false);
  const [formData, setFormData] = useState(initialFormData);
  const [imageFile, setImageFile] = useState(null);
  const [uploadedImageUrl, setUploadedImageUrl] = useState("");
  const [imageLoadingState, setImageLoadingState] = useState(false);
  const [currentEditedId, setCurrentEditedId] = useState(null);

  const { productList } = useSelector((state) => state.adminProducts);
  const dispatch = useDispatch();
  const { toast } = useToast();

  function onSubmit(event) {
    event.preventDefault();

    currentEditedId !== null
      ? dispatch(
          editProduct({
            id: currentEditedId,
            formData,
          })
        ).then((data) => {
          console.log(data, "edit");

          if (data?.payload?.success) {
            dispatch(fetchAllProducts());
            setFormData(initialFormData);
            setOpenCreateProductsDialog(false);
            setCurrentEditedId(null);
          }
        })
      : dispatch(
          addNewProduct({
            ...formData,
            image: uploadedImageUrl,
          })
        ).then((data) => {
          if (data?.payload?.success) {
            dispatch(fetchAllProducts());
            setOpenCreateProductsDialog(false);
            setImageFile(null);
            setFormData(initialFormData);
            toast({
```

```
          title: "Product add successfully",
        });
      }
    });
}

function handleDelete(getCurrentProductId) {
  dispatch(deleteProduct(getCurrentProductId)).then((data) => {
    if (data?.payload?.success) {
      dispatch(fetchAllProducts());
    }
  });
}

function isFormValid() {
  return Object.keys(formData)
    .filter((currentKey) => currentKey !== "averageReview")
    .map((key) => formData[key] !== "")
    .every((item) => item);
}

useEffect(() => {
  dispatch(fetchAllProducts());
}, [dispatch]);

console.log(formData, "productList");

return (
  <Fragment>
    <div className="mb-5 w-full flex justify-end">
      <Button onClick={() => setOpenCreateProductsDialog(true)}>
        Add New Product
      </Button>
    </div>
    <div className="grid gap-4 md:grid-cols-3 lg:grid-cols-4">
      {productList && productList.length > 0
        ? productList.map((productItem) => (
            <AdminProductTile
              setFormData={setFormData}
              setOpenCreateProductsDialog={setOpenCreateProductsDialog}
              setCurrentEditedId={setCurrentEditedId}
              product={productItem}
              handleDelete={handleDelete}
            />
          ))
        : null}
    </div>
    <Sheet
      open={openCreateProductsDialog}
      onOpenChange={() => {
        setOpenCreateProductsDialog(false);
        setCurrentEditedId(null);
        setFormData(initialFormData);
      }}
     >
      <SheetContent side="right" className="overflow-auto">
        <SheetHeader>
          <SheetTitle>
            {currentEditedId !== null ? "Edit Product" : "Add New Product"}
          </SheetTitle>
        </SheetHeader>
        <ProductImageUpload
          imageFile={imageFile}
          setImageFile={setImageFile}
          uploadedImageUrl={uploadedImageUrl}
          setUploadedImageUrl={setUploadedImageUrl}
          setImageLoadingState={setImageLoadingState}
          imageLoadingState={imageLoadingState}
          isEditMode={currentEditedId !== null}
        />
        <div className="py-6">
          <CommonForm
            onSubmit={onSubmit}
            formData={formData}
            setFormData={setFormData}
            buttonText={currentEditedId !== null ? "Edit" : "Add"}
            formControls={addProductFormElements}
            isBtnDisabled={!isFormValid()}
          />
        </div>
      </SheetContent>
     </Sheet>
    </Fragment>
  );
}

export default AdminProducts;
```

4) **DashBoard Page:**

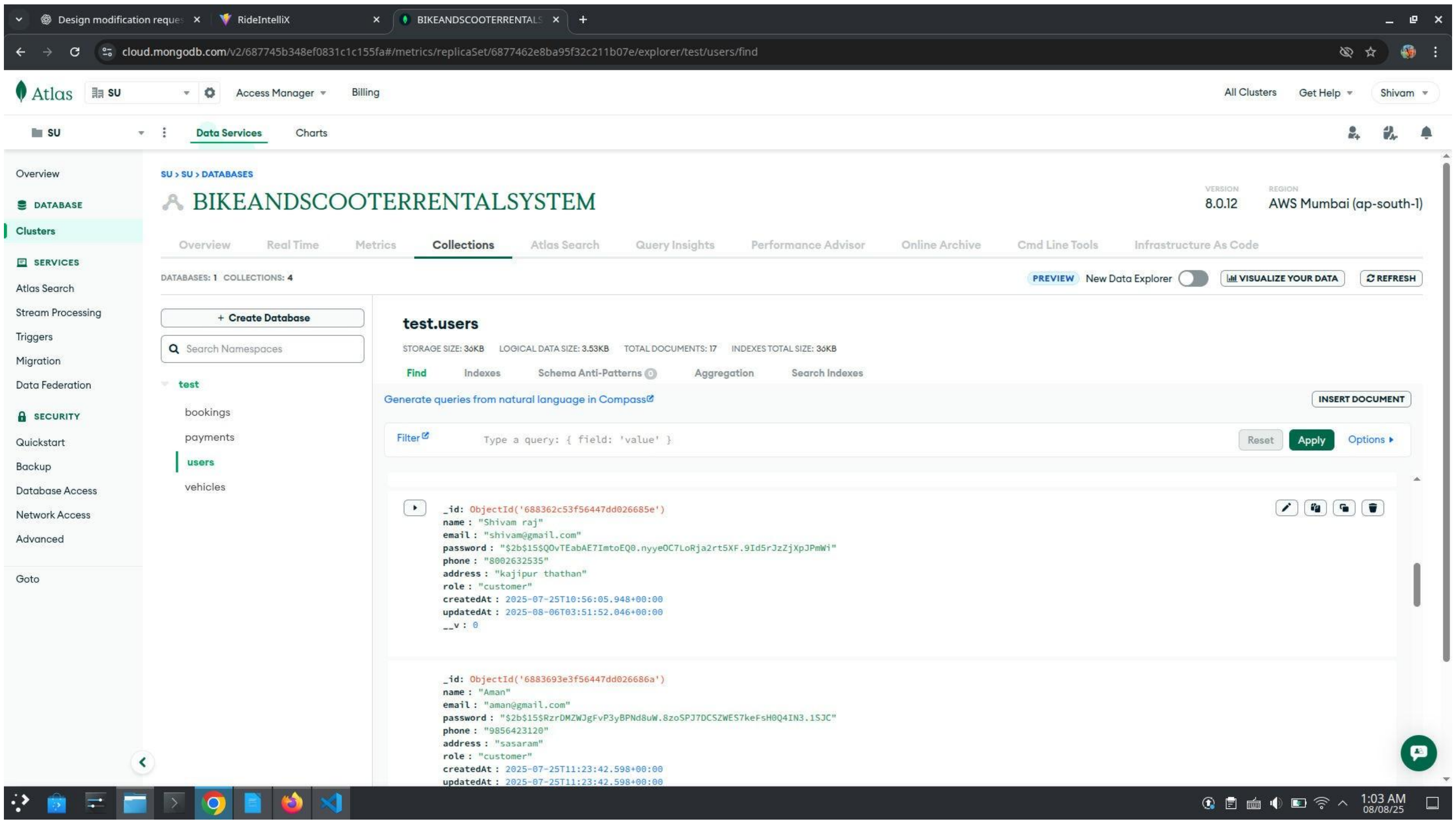


✓ CODE

```
import ProductImageUpload from "@/components/admin-view/image-upload";
import { Button } from "@/components/ui/button";
import { addFeatureImage, getFeatureImages } from "@/store/common-slice";
import { useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";

function AdminDashboard() {
  const [imageFile, setImageFile] = useState(null);
  const [uploadedImageUrl, setUploadedImageUrl] = useState("");
  const [imageLoadingState, setImageLoadingState] = useState(false);
  const dispatch = useDispatch();
  const { featureImageList } = useSelector((state) => state.commonFeature);

  console.log(uploadedImageUrl, "uploadedImageUrl");

  function handleUploadFeatureImage() {
    dispatch(addFeatureImage(uploadedImageUrl)).then((data) => {
      if (data?.payload?.success) {
        dispatch(getFeatureImages());
        setImageFile(null);
        setUploadedImageUrl("");
      }
    });
  }

  useEffect(() => {
    dispatch(getFeatureImages());
  }, [dispatch]);

  console.log(featureImageList, "featureImageList");

  return (
    <div>
      <ProductImageUpload
        imageFile={imageFile}
        setImageFile={setImageFile}
        uploadedImageUrl={uploadedImageUrl}
        setUploadedImageUrl={setUploadedImageUrl}
        setImageLoadingState={setImageLoadingState}
        imageLoadingState={imageLoadingState}
        isCustomStyling={true}
        // isEditMode={currentEditedId !== null}
      />
      <Button onClick={handleUploadFeatureImage} className="mt-5 w-full">
        Upload
      </Button>
      <div className="flex flex-col gap-4 mt-5">
        {featureImageList && featureImageList.length > 0
          ? featureImageList.map((featureImgItem) => (
              <div className="relative">
                <img
                  src={featureImgItem.image}
                  className="w-full h-[300px] object-cover rounded-t-lg"
                />
              </div>
            ))
          : null}
      </div>
    </div>
  );
}

export default AdminDashboard;
```

❖ BACKEND:-

1) USER AND ADMIN DATA:

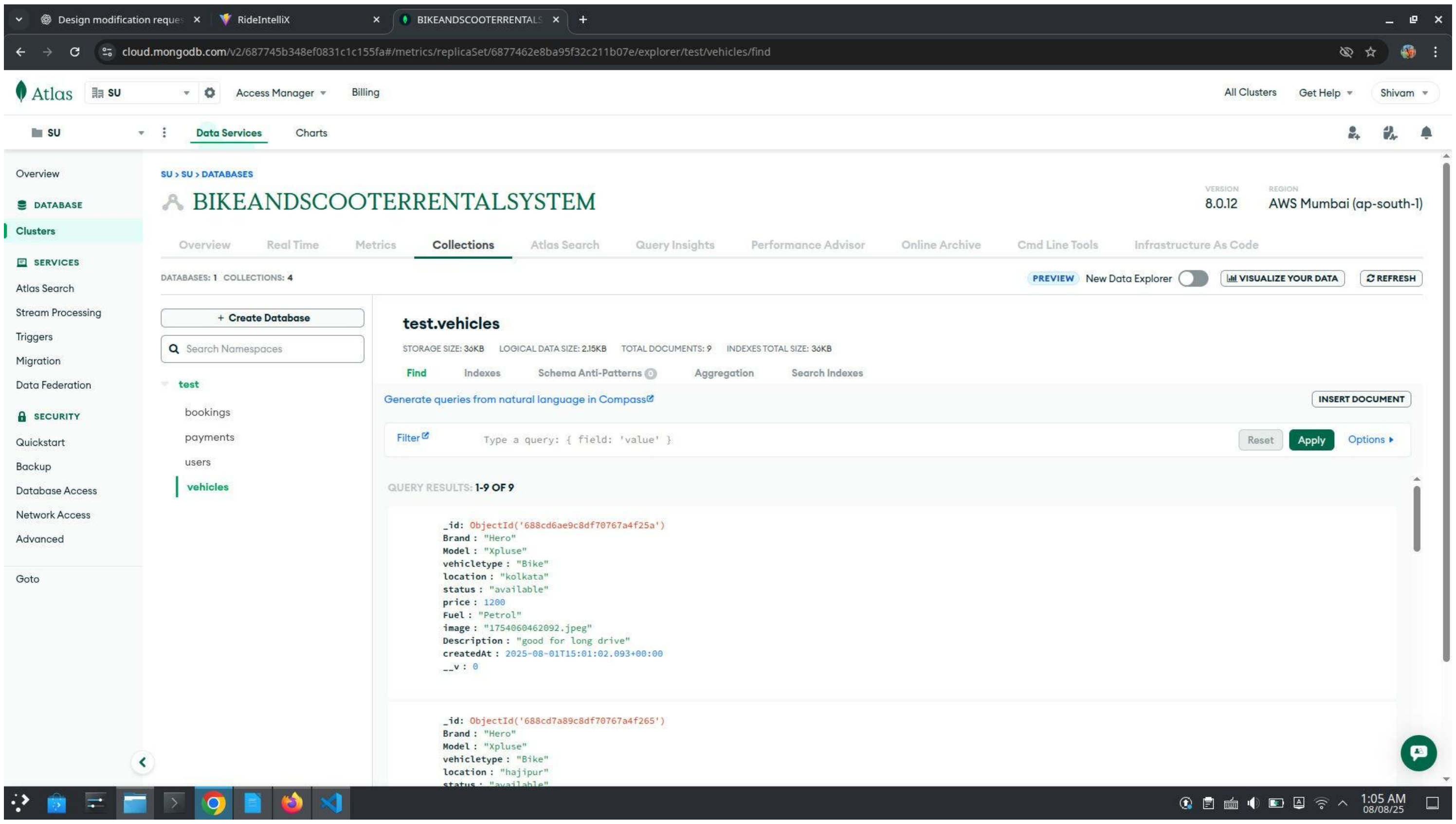


```
_id: ObjectId('688362c53f56447dd026685e')
name : "Shivam raj"
email : "shivam@gmail.com"
password : "$2b$15$QOvTEabAE7ImtoEQ0.nyyeOC7LoRja2rt5XF.9Id5rJzZjXpJPmWi"
phone : "8002632535"
address : "kajipur thathan"
role : "customer"
createdAt : 2025-07-25T10:56:05.948+00:00
updatedAt : 2025-08-06T03:51:52.046+00:00
__v : 0

_id: ObjectId('6883693e3f56447dd026686a')
name : "Aman"
email : "aman@gmail.com"
password : "$2b$15$RzrDMZWJgFvP3yBPNd8uW.8zoSPJ7DCSZWES7keFsH0Q4IN3.1SJC"
phone : "9856423120"
address : "sasaram"
role : "customer"
createdAt : 2025-07-25T11:23:42.598+00:00
updatedAt : 2025-07-25T11:23:42.598+00:00
```

● **<u>Database - db.js:</u>**

```
const mongoose = require("mongoose");


const connectdb = async () => {
try {
        await mongoose.connect(process.env.MONGO_URL);
console.log("MongoDB connected");
    } catch (err) {
        console.log("MongoDB connection failed:", err.message);
    }
};



module.exports = connectdb;
```

2) VEHICLES DATA:



cloud.mongodb.com/v2/687745b348ef0831c1c155fa#/metrics/replicaSet/6877462e8ba95f32c211b07e/explorer/test/vehicles/find

SU > SU > DATABASES

BIKEANDSCOOTERRENTALSYSTEM

VERSION 8.0.12 REGION AWS Mumbai (ap-south-1)

DATABASES: 1 COLLECTIONS: 4

test: bookings, payments, users, vehicles

test.vehicles

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 2.15KB TOTAL DOCUMENTS: 9 INDEXES TOTAL SIZE: 36KB

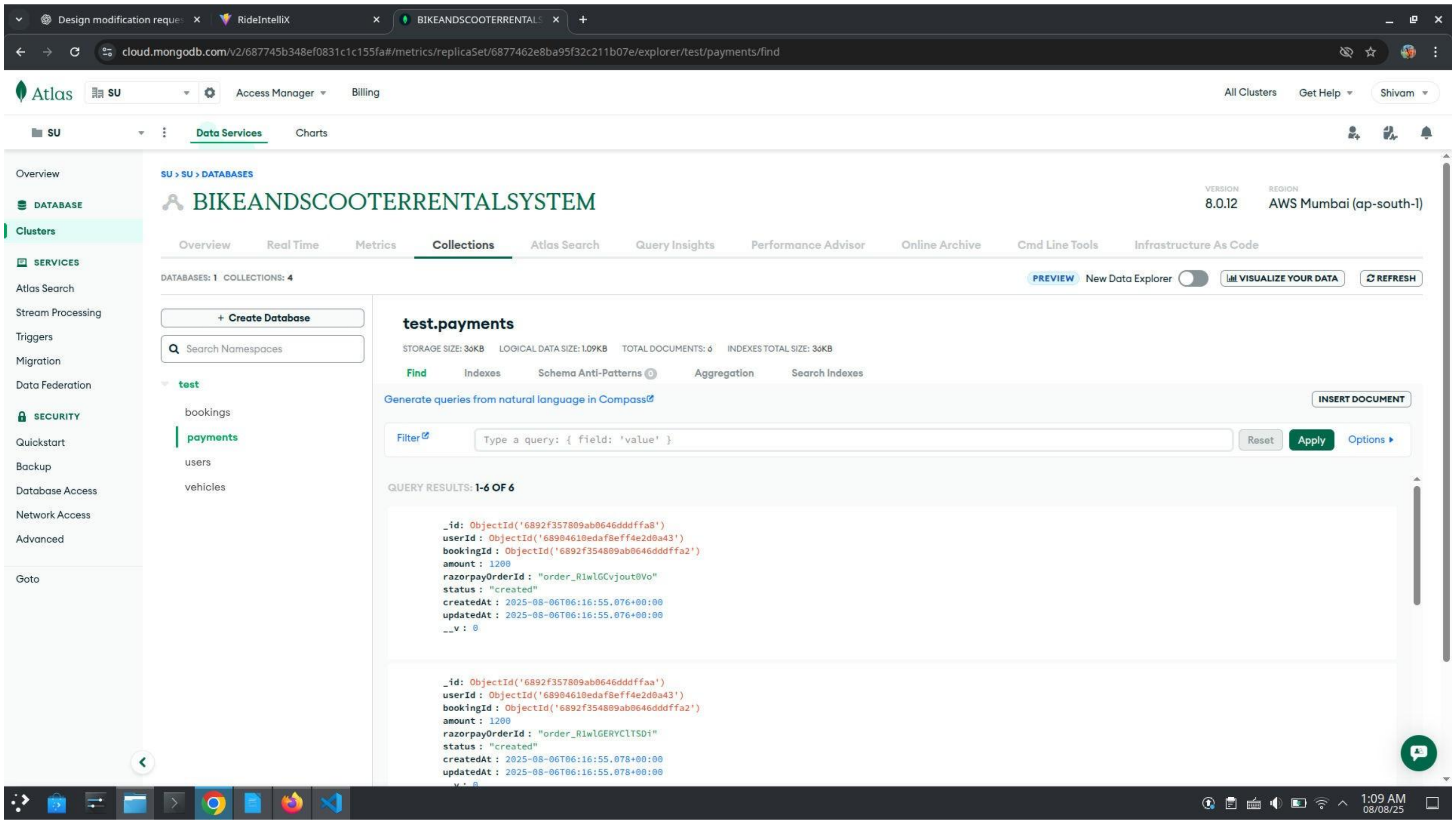
QUERY RESULTS: 1-9 OF 9

```
_id: ObjectId('688cd6ae9c8df70767a4f25a')
Brand : "Hero"
Model : "Xpluse"
vehicletype : "Bike"
location : "kolkata"
status : "available"
price : 1200
Fuel : "Petrol"
image : "1754060462092.jpeg"
Description : "good for long drive"
createdAt : 2025-08-01T15:01:02.093+00:00
__v : 0

_id: ObjectId('688cd7a89c8df70767a4f265')
Brand : "Hero"
Model : "Xpluse"
vehicletype : "Bike"
location : "hajipur"
status : "available"
```

● **model – website.js :-** const mongoose =

```
  const mongoose =
  require("mongoose");

const ProductSchema = new
  mongoose.Schema({
   name: {
     type: String,
     required: true
   },
   brand: {
     type: String
   },
   category: {
     type: String,
      required: true
    },
    location: { // If you have
  warehouses or shop locations
      type: String
    },
    status: {
      type: String,
      enum: ['in-stock', 'out-of-
  stock', 'discontinued'],
      default: 'in-stock'
    },
    price: {
      type: Number,
      required: true
    },
    image: {
      type: String,
      default: 'default-product.jpg'
    },
    description: {
      type: String,
      required: true
    },
    createdAt: {
      type: Date,
      default: Date.now
    }
});

module.exports =
  mongoose.model("Product",
  ProductSchema);
```

3) **PAYMENT DATA**



test.payments

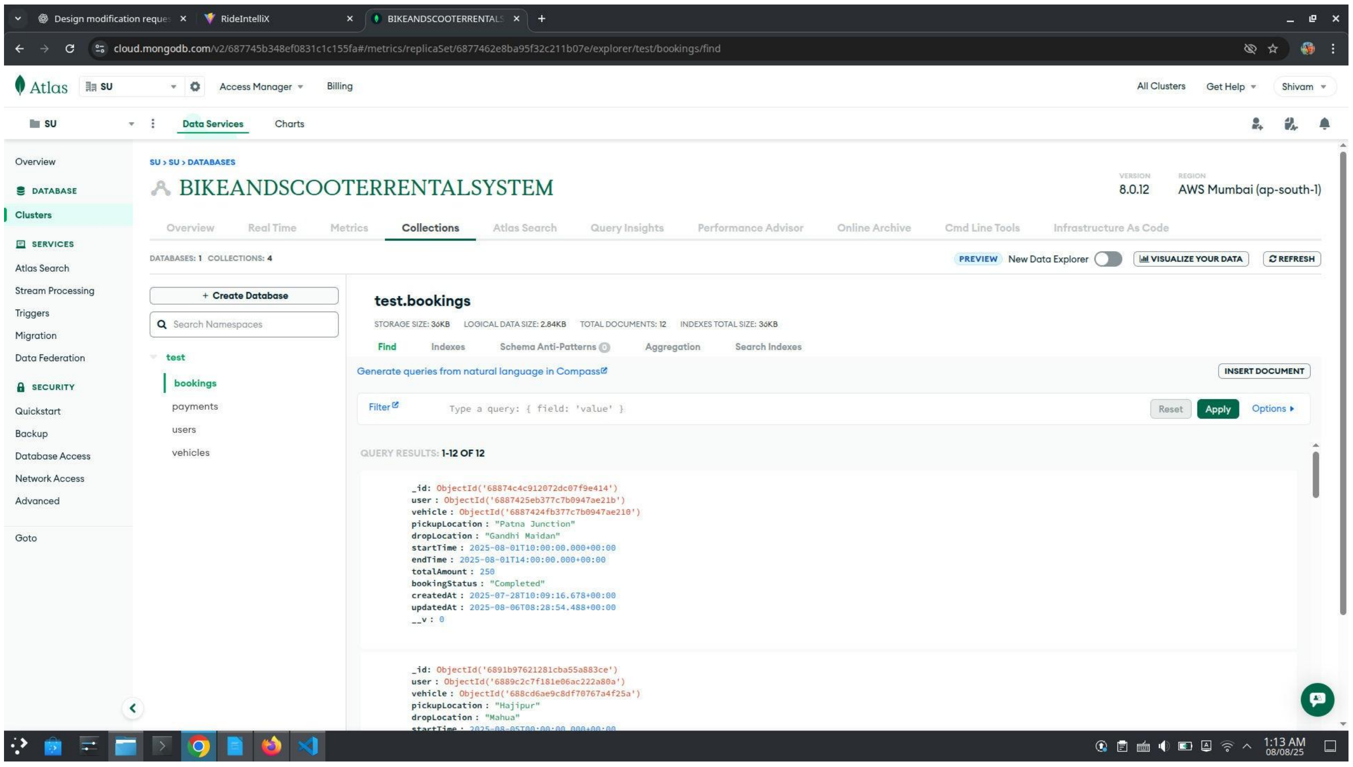
QUERY RESULTS: 1-6 OF 6

```
_id: ObjectId('6892f357809ab0646dddffa8')
userId : ObjectId('68904610edaf8eff4e2d0a43')
bookingId : ObjectId('6892f354809ab0646dddffa2')
amount : 1200
razorpayOrderId : "order_R1wlGCvjout0Vo"
status : "created"
createdAt : 2025-08-06T06:16:55.076+00:00
updatedAt : 2025-08-06T06:16:55.076+00:00
__v : 0

_id: ObjectId('6892f357809ab0646dddffaa')
userId : ObjectId('68904610edaf8eff4e2d0a43')
bookingId : ObjectId('6892f354809ab0646dddffa2')
amount : 1200
razorpayOrderId : "order_R1wlGERYClTSDi"
status : "created"
createdAt : 2025-08-06T06:16:55.078+00:00
updatedAt : 2025-08-06T06:16:55.078+00:00
```

● **model – Payment.js :-**

```
// model – Payment.js

const mongoose = require("mongoose");

const paymentSchema = new mongoose.Schema(
  {
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true
    },
    orderId: { // Links to e-commerce Order
      type: mongoose.Schema.Types.ObjectId,
      ref: "Order",
      required: true
    },
    amount: {
      type: Number,
      required: true
    },
    razorpayOrderId: String,
    razorpayPaymentId: String,
    status: {
      type: String,
      enum: ["created", "paid", "failed"],
      default: "created"
    },
    paymentMethod: { // Optional: UPI, Card, Wallet, etc.
      type: String
    }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Payment", paymentSchema);
```

4) SHOPINNG DATA



```
_id: ObjectId('68874c4c912072dc07f9e414')
user : ObjectId('6887425eb377c7b0947ae21b')
vehicle : ObjectId('6887424fb377c7b0947ae210')
pickupLocation : "Patna Junction"
dropLocation : "Gandhi Maidan"
startTime : 2025-08-01T10:00:00.000+00:00
endTime : 2025-08-01T14:00:00.000+00:00
totalAmount : 250
bookingStatus : "Completed"
createdAt : 2025-07-28T10:09:16.678+00:00
updatedAt : 2025-08-06T08:28:54.488+00:00
__v : 0

_id: ObjectId('6891b97621281cba55a883ce')
user : ObjectId('6889c2c7f181e06ac222a80a')
vehicle : ObjectId('688cd6ae9c8df70767a4f25a')
pickupLocation : "Hajipur"
dropLocation : "Mahua"
startTime : 2025-08-05T00:00:00.000+00:00
```

// model – Order.js

```
const mongoose = require('mongoose');


const orderSchema = new mongoose.Schema({
 user: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
 },
 products: [ // Array of purchased products
  {
   product: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Product',
    required: true
   },
   quantity: {
    type: Number,
    required: true,
    default: 1
   },
   price: {
    type: Number,
    required: true
```

```
      }
    }
  ],
  shippingAddress: {
    type: String,
    required: true
  },
  billingAddress: {
    type: String
  },
  orderDate: {
    type: Date,
    default: Date.now
  },
  totalAmount: {  module.exports = mongoose.model('Booking',
bookingSchema);
```

6. CONCLUSION

Zintargo emerges as a comprehensive and innovative e-commerce platform, delivering a seamless and user-friendly online shopping experience. It bridges the gap between sellers and customers by offering a diverse range of products across multiple categories, ensuring convenience, variety, and quality for every shopper.

With features like secure ordering, integrated digital payment options (via Razorpay), real-time product availability, and robust user authentication, Zintargo provides a reliable and efficient shopping environment. The platform's intuitive design and smooth navigation make it easy for customers to find and purchase products, while empowering sellers to manage inventory, orders, and sales effectively.

As online shopping continues to expand globally, Zintargo positions itself as a scalable and future-ready solution, catering to the growing demand for fast, secure, and personalized e-commerce experiences. By embracing innovation and customer-centric strategies, Zintargo aims to play a pivotal role in shaping the future of digital retail.

7. FUTURE SCOPE & FURTHER ENHANCEMENTS

❖ Future scope:-

The future of Zintargo holds vast potential for expansion, innovation, and integration of advanced technologies to further elevate the online shopping experience. As e-commerce trends evolve and consumer demands grow, the platform can scale to serve a wider audience while delivering faster, more personalized, and secure shopping solutions. Key areas for future development include:

- **AI-Powered Product Recommendations:**
  Leverage artificial intelligence to analyze user preferences, browsing history, and purchase behavior to deliver highly personalized product suggestions.
- **Dynamic Pricing & Discounts:**
  Introduce time-based offers, seasonal discounts, flash sales, and loyalty-based rewards to enhance user engagement and drive more sales.
- **Augmented Reality (AR) Shopping:**
  Allow customers to virtually "try" products, such as apparel, accessories, or home décor, before making a purchase.
- **Voice-Enabled Shopping:**
  Integrate voice assistants to make product searches and purchases faster and more accessible.
- **Multi-Warehouse & Same-Day Delivery:**
  Expand logistics with multiple fulfillment centers to enable same-day or even two-hour delivery in high-demand regions.
- **Global Expansion:**
  Scale Zintargo to serve international markets with multi-currency, multilingual support, and region-specific product catalogs.

❖ Further enhancement:-

- ❖ **User Experience Enhancements:**
  - ● **Personalized Dashboards:** Display recent orders, browsing history, wishlist items, exclusive offers, and personalized product recommendations on the home screen.
  - ● **Dark Mode & Accessibility Options:** Add dark mode, larger font options, and voice-assistive features for better accessibility.
  - ● **One-Tap Reorder:** Allow returning customers to quickly reorder frequently purchased products with a single click.
- ❖ **Security & Payment:**
  - ● **KYC Integration for Sellers:** Verify seller identities using Aadhaar/PAN to ensure trust and compliance.
  - ● **Multiple Payment Gateways:** Support UPI, credit/debit cards, Paytm, Google Pay, Amazon Pay, PayPal, etc., alongside Razorpay.
  - ● **Refund & Return Policy:** Implement clear policies with automated refund and return processing for faster resolutions.
- ❖ **Product & Inventory Management:**
  - ● **Smart Stock Tracking:** Notify admins when products are low in stock or out of stock to ensure timely restocking.
  - ● **Automated Price Updates:** Enable bulk updates for seasonal sales, discounts, and price changes.
- ❖ **Admin Panel Enhancements:**
  - ● **Analytics Dashboard:** Provide insights on top-selling products, user behavior, revenue tracking, and sales trends.
  - ● **Multi-Admin Roles:** Define access levels such as super admin, inventory manager, and customer service representative.
- ❖ **Community & Support Features:**
  - ● **In-App Chat or Support Bot:** Offer real-time help through customer chat support or AI chatbot assistance.
  - ● **Multilingual Support:** Enable the platform in multiple languages to cater to diverse audiences.
  - ● **PWA Support (Progressive Web App):** Allow Zintargo to function like a native mobile app with offline access and fast loading times.

8. BIBLIOGRAPHY

1) www.w3schools.com
2) www.youtube.com
3) www.pexels.com
4) www.codepen.io
5) www.google.com
6) www.googlefont.com
7) www.react.our
8) www.codingworld.com