Laptop Store Web Application

Team Members: Shokanov Dias
Course Title: Advanced Databases (NoSQL)
Instructor's Name: Shynar Akhmetzhanova

Aim and Goals

PURPOSE:

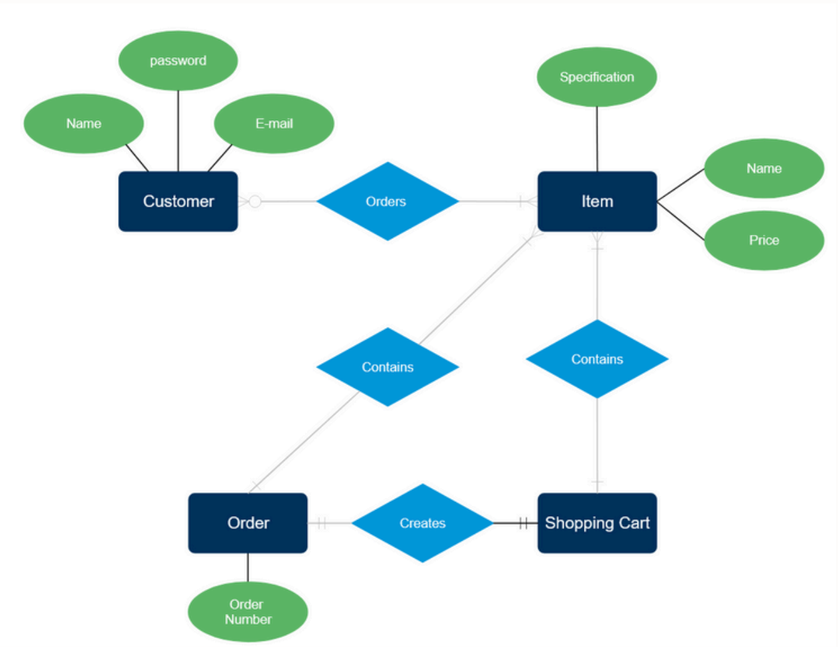
Develop a full-stack web application that simulates a laptop store.

MAIN OBJECTIVES:

- Build a robust backend using Node.js, Express, and MongoDB.
- Design an optimized, well-indexed database with clear relationships and nested documents.
- Implement comprehensive CRUD operations with RESTful API endpoints.
- Secure the application with JWT-based authentication and role-based authorization.
- Create an intuitive frontend interface for both regular users and administrators.

Relevance

- IMPORTANCE OF THE PROJECT:
  - Provides a realistic e-commerce scenario for learning and demonstration.
  - Illustrates the integration of modern web technologies and best practices.
- PROBLEMS SOLVED:
  - Manages product inventory (laptops) with efficient querying and indexing.
  - Ensures secure user interactions through authentication and authorization.
  - Handles order management and user profile updates in an organized manner.



System Architecture

Overview of Components:

- Frontend: Responsive UI using HTML, Bootstrap, and jQuery (or a modern framework for future improvements).
- Backend: Node.js and Express RESTful API.
- Database: MongoDB with a well-designed schema.

UML Diagrams:

Use-Case Diagram:

- Actors: User, Admin.
- Use Cases: Register, Login, Browse Laptops, Manage Orders, Manage Laptops (admin-only).

Sequence Diagram:

- Example: Order placement process showing the interaction between the client, server, and database.

Data Schema

Laptops Collection:

- Fields: Brand, Model, Price, Specifications (embedded: processor, RAM, storage, graphics, display), Availability.

```
const laptopSchema = new mongoose.Schema({
  brand: { type: String, required: true, index: true },
  model: { type: String, required: true, index: true },
  price: { type: Number, required: true },
  specifications: {
    processor: String,
    ram: String,
    storage: String,
    graphics: String,
    display: String
  },
  available: { type: Boolean, default: true },
  createdAt: { type: Date, default: Date.now }
});
```

Users Collection:

- Fields: Username, Email, Hashed Password, Role (user/admin), Addresses (embedded).

```
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true, index: true },
  email: { type: String, required: true, unique: true, index: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['user', 'admin'], default: 'user' },
  addresses: [{
    street: String,
    city: String,
    state: String,
    zip: String,
  }]
});
```

Orders Collection:

- Fields: User Reference, Laptops Ordered (array with laptop reference and quantity), Total Price, Order Date, Status.

```
const orderSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  laptops: [{
    laptop: { type: mongoose.Schema.Types.ObjectId, ref: 'Laptop', required: true },
    quantity: { type: Number, default: 1 }
  }],
  totalPrice: { type: Number },
  orderDate: { type: Date, default: Date.now },
  status: { type: String, enum: ['pending', 'shipped', 'delivered'], default: 'pending' }
});
```

Data Collection

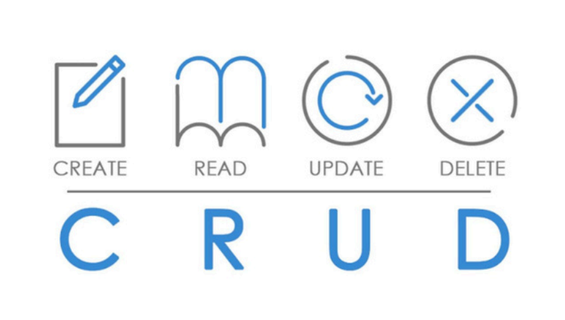
PROCESS OVERVIEW:

Initial Data Seeding: Use scripts or admin interfaces to populate laptop data.

User Input: Registration and profile update forms collect user data.

Order Placement: Data is captured through frontend order forms and processed by the backend.

Application Features



- Create: Adding new laptops, registering users, and placing orders.
- Read: Retrieving lists of laptops, user profiles, and order histories.
- Update: Modifying laptop details (admin), updating user profiles, and editing pending orders.
- Delete: Removing laptops (admin) and canceling orders (when permissible).

- Indexes on key fields (e.g., laptop brand, model; user email) to speed up queries.
- Optimization techniques applied in MongoDB queries for improved performance.

- Authorization: Role-based access control ensures that only admins can manage laptops.
- Data Protection: Passwords are hashed (using bcrypt) and sensitive data is secured.

Conclusion

KEY FINDINGS:

- Successful integration of MongoDB, Express, and a responsive frontend to build a complete web application;
- Effective use of CRUD operations, indexing, and security best practices.

FUTURE IMPROVEMENTS:

- Enhance the frontend with modern frameworks for better scalability;
- Integrate advanced error handling and logging mechanisms;
- Implement additional features such as payment gateway integration, real-time notifications;