**COMPUTER SCIENCE DEPARTMENT**

# Total Marks: 7.5

**Obtained Marks:**

LAB: Database System

**Project Report: University Management System**

**Submitted To: Mam Kashia Riaz**

**Date of Submission: 30-December-2024**

**Project Members**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Student Name** | **Reg.no** |
| **01** | **Ubaid Bin Waris** | **2212416** |
| **02** | **Mushahid Hussain** | **2212408** |
| **03** | **Muhammad Salman** | **2212400** |
| **04** | **Jehanzeb Khalid** | **2212391** |

Table of Contents

**Front Page01**

**Table of Content02**

**Introduction03**

**Specific Features03**

Database Feature 03

UI Frontend Features (Next.js)05

**Database Schema06**

**Schema Relationship and Constraints07**

**Database Connectivity With frontend (Next.js)08**

Setting up API routes in Next.js 08

Database Connection08

Fetching Data in Next.js Pages08

Relational Vs Non SQL Database Integrity09

Data flow and User Interaction09

**Data Manipulating10**

**ER Diagram13**

**Conclusion15**

**Introduction**

The **University Management System (UMS)** is a modern web-based application designed to enhance the efficiency and effectiveness of university administrative processes. This project integrates a robust relational database with a dynamic front-end interface built using **Next.js**, providing an interactive and seamless user experience for students, faculty, and administrators. The system manages essential university data, including student records, course details, faculty assignments, classroom schedules, and examination timetables. By utilizing a relational database for backend management and Next.js for front-end development, the project ensures data integrity, efficient retrieval, and user-friendly access to information. The University Management System supports functionalities such as adding, updating, and deleting records, as well as generating real-time reports. This project aims to simplify decision-making, reduce manual workloads, and improve overall data accessibility, making it a valuable tool for universities in the digital age.

**Specific Features**

# ****Database Features****

1. **Departments Management**  
   The **Departments** table stores information about academic departments within the university. Each department is uniquely identified by **department\_id**, with a name and the faculty head (**head\_faculty\_id**), ensuring department leadership is clearly defined.
2. **Students Management**  
   The **Students** table tracks essential student information, such as **first\_name**, **last\_name**, **date\_of\_birth**, **email**, and **phone\_number**. It also includes the **department\_id** field, linking each student to a specific department. This structure helps in organizing students by their departments for better academic and administrative management.
3. **Faculty Management**  
   The **Faculty** table holds faculty member details, including **first\_name**, **last\_name**, **email**, **phone\_number**, and **hire\_date**. It also has a **department\_id** to associate each faculty member with a department. This table allows for the management of faculty assignments to courses and schedules. The unique constraint on **department\_id** and **faculty\_id** ensures that faculty are only assigned to one department.
4. **Course Management**  
   The **Courses** table stores the essential course information, such as **course\_name**, **credits**, and **department\_id**. This allows courses to be linked to their respective departments and enables efficient course management, scheduling, and enrollment.
5. **Enrollments Management**  
   The **Enrollments** table is a weak entity dependent on both **Students** and **Courses**, capturing which student is enrolled in which course. It includes an **enrollment\_date** and **grade**, and the **student\_id** and **course\_id** form a unique combination, ensuring a student cannot enroll in the same course more than once.
6. **Classrooms Management**  
   The **Classrooms** table stores information about university classrooms, including **room\_number**, **building**, and **capacity**. This helps in managing classroom usage and facilitates scheduling of classes according to room availability.
7. **Schedules Management**  
   The **Schedules** table manages class timings by linking **courses**, **classrooms**, and **faculty** members to specific days and time slots. The **day** field is constrained to the days of the week, and **time\_slot** ensures that courses are scheduled in appropriate timeslots to avoid conflicts. A unique constraint on **course\_id**, **room\_id**, **day**, and **time\_slot** ensures that no class is scheduled more than once in the same room at the same time.
8. **Fees Management**  
   The **Fees** table tracks students' fee payments, including **amount**, **due\_date**, and **status** (Paid/Unpaid). This allows the university to efficiently manage fee collection and follow up with students regarding unpaid fees.
9. **Library Management**  
   The **Library** table tracks books available in the university library, with details such as **title**, **author**, **status** (Available/Loaned), and **isbn**. The **loaned\_to\_student\_id** field links a student to a book they have checked out, allowing the system to manage library loans effectively.

# ****Key Database Operations and Benefits:****

* **CRUD Operations**: All tables support basic CRUD operations (Create, Read, Update, Delete), ensuring smooth data management across different aspects of the university.
* **Data Integrity**: Foreign key constraints ensure data integrity across relationships, such as between students and departments, faculty and courses, and enrollments.
* **Efficient Data Retrieval**: By organizing data into specific tables with clear relationships, the system allows for fast and efficient data queries, enabling administrative staff to generate reports and manage records with ease.

**UI Frontend Features(Next.js)**

1. **Student Dashboard**  
   Provides a clear, user-friendly interface for students to manage their academic information, such as viewing enrolled courses, checking grades, reviewing schedules, and tracking fee payments. The design is intuitive and organized, ensuring students can easily access important academic details.
2. **Faculty Dashboard**  
   Displays faculty members’ courses, class schedules, and student enrollments. Faculty can manage their timetable, post grades, and communicate with students via the platform. The dashboard is designed to streamline faculty tasks such as assigning grades and handling student queries.
3. **Course Registration and Enrollment**  
   Allows students to browse available courses, view course details (e.g., schedule, credits, prerequisites), and enroll in or drop courses. The front-end interface provides real-time course availability updates, helping students make informed decisions when registering.
4. **Class Schedule Management**  
   Features an interactive calendar where students and faculty can view, edit, or print their full academic schedule. The schedule updates in real-time to reflect changes and ensure no conflicts in class timings or room assignments.
5. **Fee Payment and Financial Management**  
   Enables students to view their fee balance, make payments online, and track the status of their payments (Paid/Unpaid). Integration with payment gateways ensures secure online transactions. The UI sends notifications about upcoming due dates and allows students to view their payment history.
6. **Library Management**  
   Provides a searchable catalog of books available in the university library. Users can search by title, author, or ISBN and check book availability. The system allows students and faculty to reserve or check out books and tracks overdue items, sending reminders about due dates.
7. **Responsive Design**  
   Built with Next.js, the system’s front-end is fully responsive and adapts seamlessly to devices of all sizes, including desktops, tablets, and smartphones. This ensures that users can access the system anywhere, anytime, and from any device, providing flexibility and ease of use.

These **UI Front-End Features** are designed to enhance the user experience and ensure smooth interaction with the University Management System, providing students, faculty, and administrators with efficient and intuitive tools to manage their tasks.

**Database Schema**

The database schema for your University Database System would outline the structure of each table, including primary keys, foreign keys, and relationships between tables. Here’s an example schema based on the features

1. **Departments**: department\_id (PK), department\_name, head\_faculty\_id (FK).
2. **Students**: student\_id (PK), first\_name, last\_name, date\_of\_birth, email (Unique), phone\_number, enrollment\_date, department\_id (FK)
3. **Faculty**: faculty\_id (PK), first\_name, last\_name, email (Unique), phone\_number, hire\_date, department\_id (FK), UNIQUE(department\_id, faculty\_id)
4. **Courses**: course\_id (PK), course\_name, credits, department\_id (FK)
5. **Enrollments**: enrollment\_id (PK), student\_id (FK), course\_id (FK), enrollment\_date, grade, UNIQUE(student\_id, course\_id)
6. **Classrooms**: room\_id (PK), room\_number (Unique), building, capacity
7. **Schedules**: schedule\_id (PK), course\_id (FK), room\_id (FK), faculty\_id (FK), day (ENUM), time\_slot, UNIQUE(course\_id, room\_id, day, time\_slot)
8. **Fees**: fee\_id (PK), student\_id (FK), amount (DECIMAL), due\_date, status (ENUM)
9. **Library**: book\_id (PK), title, author, status (ENUM), isbn (Unique), loaned\_to\_student\_id (FK).

**Schema Relationships and Constraints Summary**

* **Departments and Students**: One-to-Many relationship (One department can have many students, each student belongs to one department).
* **Departments and Faculty**: One-to-Many relationship (One department can have many faculty members, each faculty member belongs to one department).
* **Departments and Courses**: One-to-Many relationship (One department offers multiple courses).
* **Students and Enrollments**: One-to-Many relationship (A student can enroll in multiple courses).
* **Courses and Enrollments**: One-to-Many relationship (A course can have multiple students enrolled).
* **Courses and Schedules**: One-to-Many relationship (One course can have multiple schedules).
* **Faculty and Schedules**: One-to-Many relationship (One faculty member can be assigned to multiple course schedules).
* **Classrooms and Schedules**: One-to-Many relationship (A classroom can be used in multiple schedules).
* **Students and Fees**: One-to-Many relationship (A student can have multiple fee payments).
* **Students and Library**: One-to-Many relationship (A student can borrow multiple books from the library).

**Constraints**

* **Primary Keys** Each table has a primary key (department\_id, student\_id, faculty\_id, course\_id, etc.) to uniquely identify records.
* **Foreign Keys** Foreign key constraints ensure that relationships between tables (like department\_id in Faculty and Students) maintain data integrity.
* **Unique Constraints** Unique constraints, such as on email in Students and Faculty, ensure that certain attributes remain distinct across records.
* **Enumerated Data Types**: For attributes with limited values (e.g., status in Library and Fees tables), ENUM data types improve data accuracy and constraints.

This schema provides a comprehensive structure for managing university-related data, from departments and faculty to courses, classrooms, schedules, fees, and library books. Each table is organized to support the logical relationships between different entities within the university system.

**Database Connectivity With Frontend (Next.js)**

To establish database connectivity with the front-end in a Next.js application, you generally use an API route in Next.js to connect with the database, retrieve or manipulate data, and then send it to the front-end for rendering. The following outlines the steps involved in setting up database connectivity with Next.js:

# ****Setting Up API Routes in Next.js****

# Next.js allows you to create API routes inside the pages/api directory. These routes serve as backend endpoints for handling HTTP requests (like GET, POST, PUT, DELETE). For example, you could have an endpoint to fetch data from the database or send new data to be inserted. When a request is made to these endpoints, the API route connects to the database, performs the necessary operations, and returns the result back to the front-end.

# ****Database Connection****

To connect to a database, you need to configure a connection utility. This utility allows the API routes to access and interact with the database. Depending on the type of database used (SQL or NoSQL), the connection process will vary, but the goal is to ensure a seamless connection between the database and the API routes, ensuring that data can be queried and manipulated.

# ****Fetching Data in Next.js Pages****

Once the database connection is set up, the data can be fetched within the Next.js pages. There are two primary ways to fetch data:

1. **Server-Side Rendering (SSR)**: With getServerSideProps, data is fetched on the server whenever the page is requested. This method ensures that data is preloaded and rendered on the page before it is sent to the client.
2. **Client-Side Rendering (CSR)**: Using hooks like useEffect along with fetch, data can be fetched after the page is loaded, making it suitable for real-time updates or dynamic content that doesn't need to be preloaded.

# ****Relational vs. NoSQL Database Integration****

Depending on the structure of your data, you may choose to work with either a relational database (such as MySQL or PostgreSQL) or a NoSQL database (such as MongoDB). Relational databases store structured data in tables with predefined relationships, while NoSQL databases are more flexible, allowing for unstructured data storage. The process of connecting these databases to Next.js is handled by database-specific drivers and connectors.

**For example:**

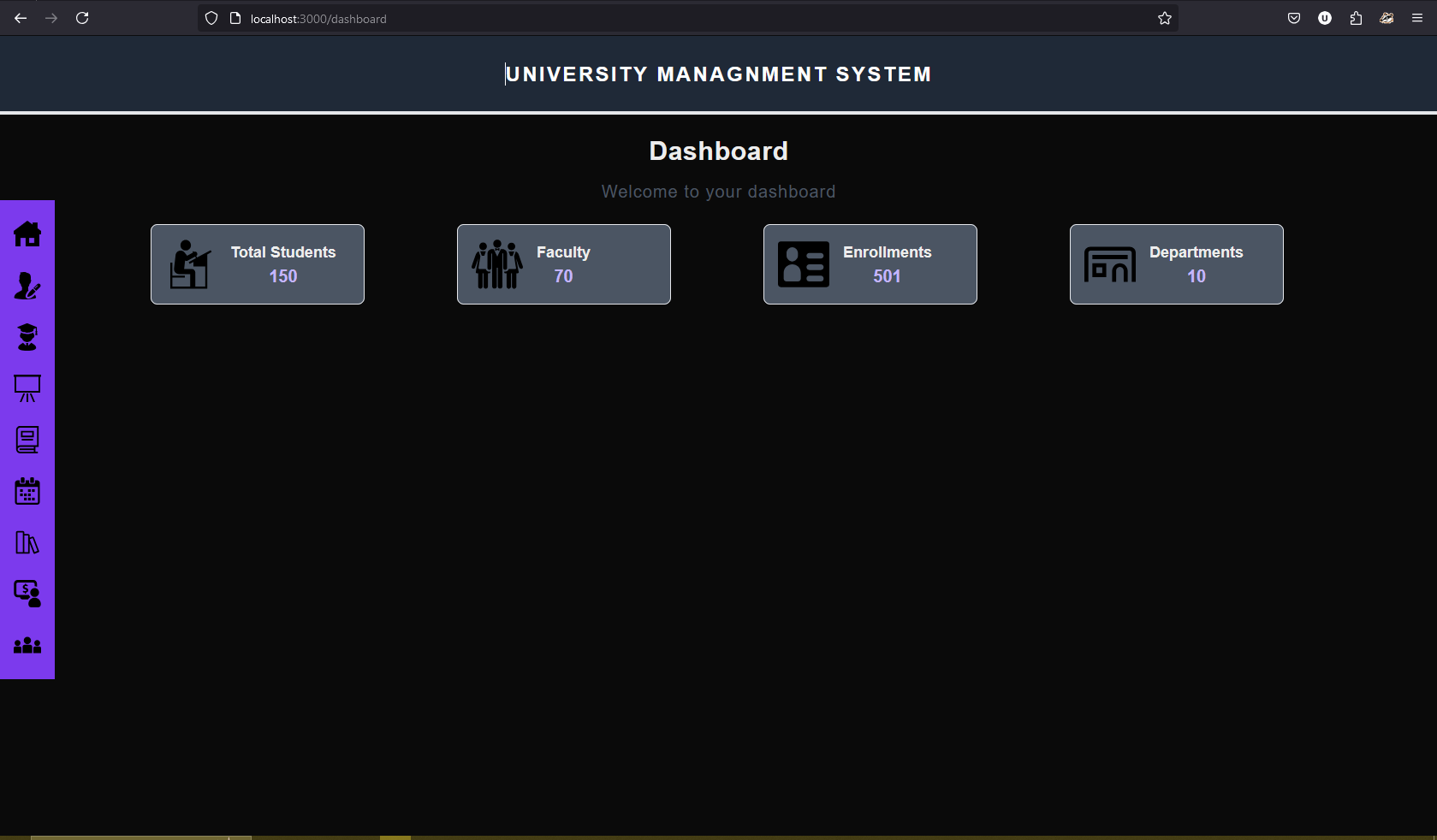
1. **Relational Databases (SQL)**: For databases like MySQL or PostgreSQL, you would use specific libraries to establish the connection and query data.
2. **NoSQL Databases (MongoDB)**: For MongoDB, you'd use MongoDB's native client to establish the connection and manage data.
3. **Data Flow and User Interaction**

Once the API routes are set up and connected to the database, the front-end (the Next.js application) can request data from the back-end. Users interact with the front-end to view or update information, such as enrolling in courses, checking grades, or viewing class schedules. The front-end sends these requests to the corresponding API routes, which interact with the database and return the required data.

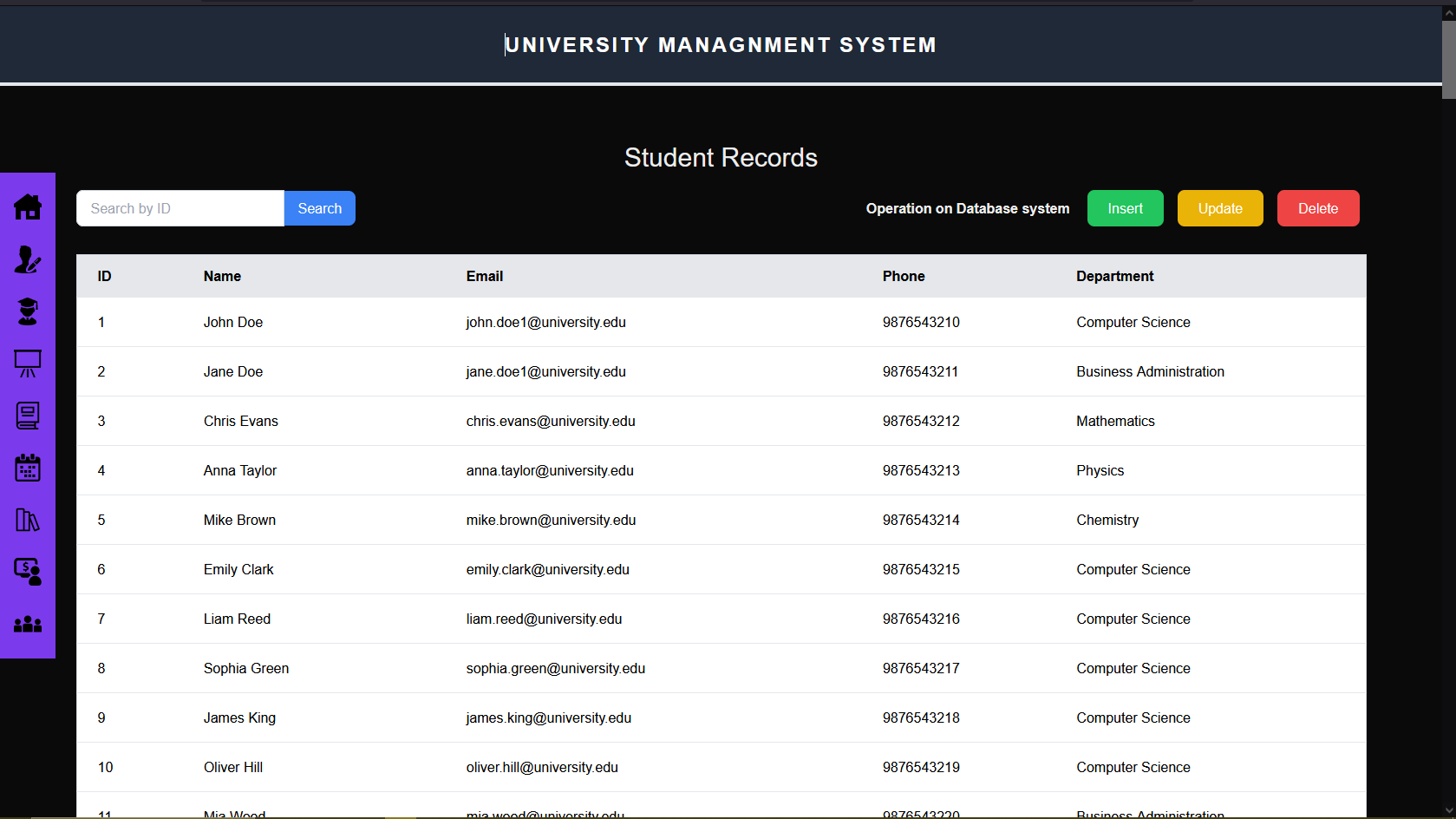
In summary, Next.js provides a powerful framework for building both the front-end and API routes that connect to a database. The interaction between the front-end (Next.js) and the database is typically handled through API routes that manage HTTP requests and responses. Whether using a relational or NoSQL database, the goal is to ensure that data is fetched, updated, and displayed efficiently, offering users a seamless experience. By setting up authentication and managing role-based access, you can ensure security and privacy while enabling smooth database connectivity for your Next.js application.

**Data Manipulation**

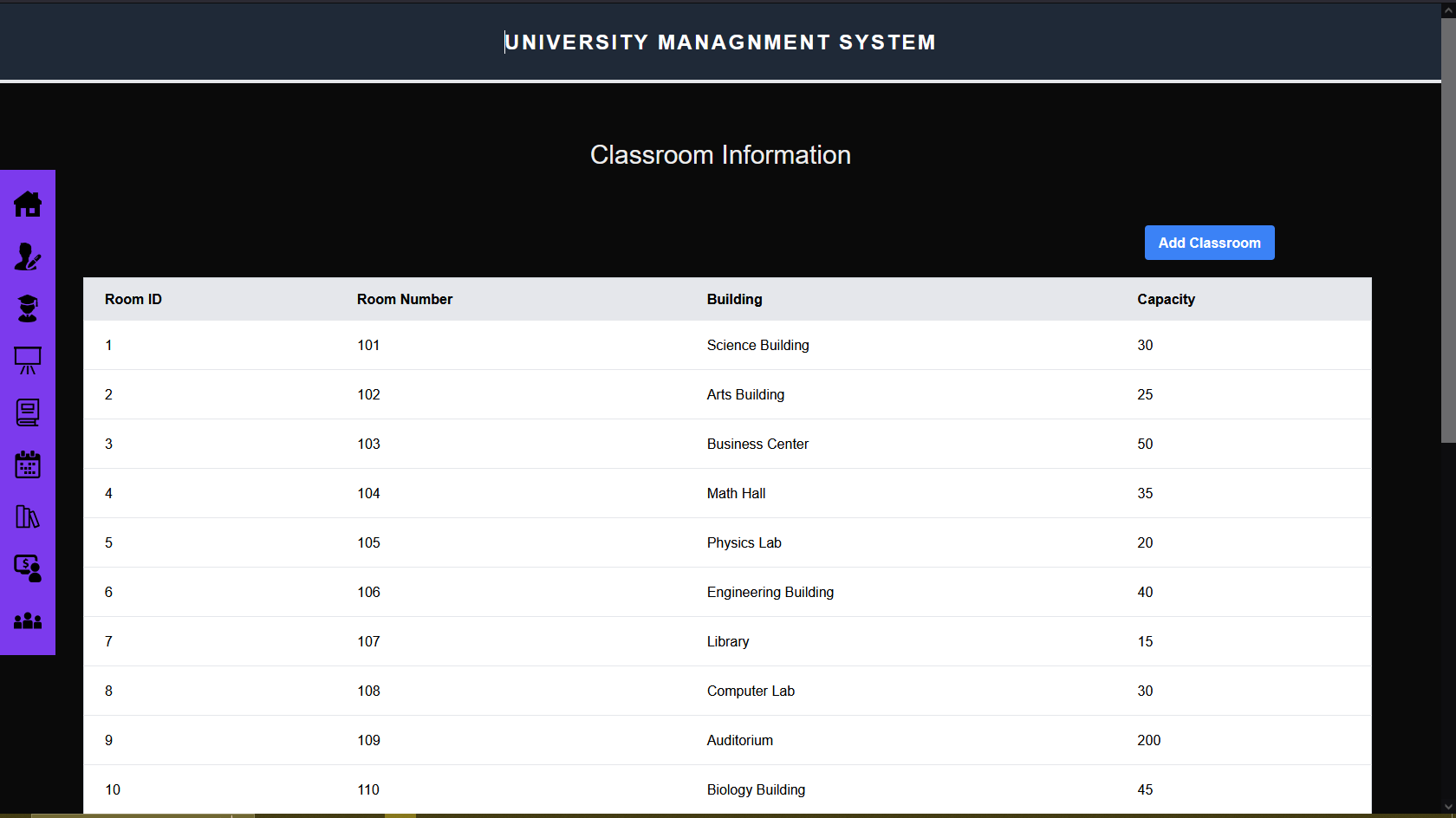
**Dashboard**



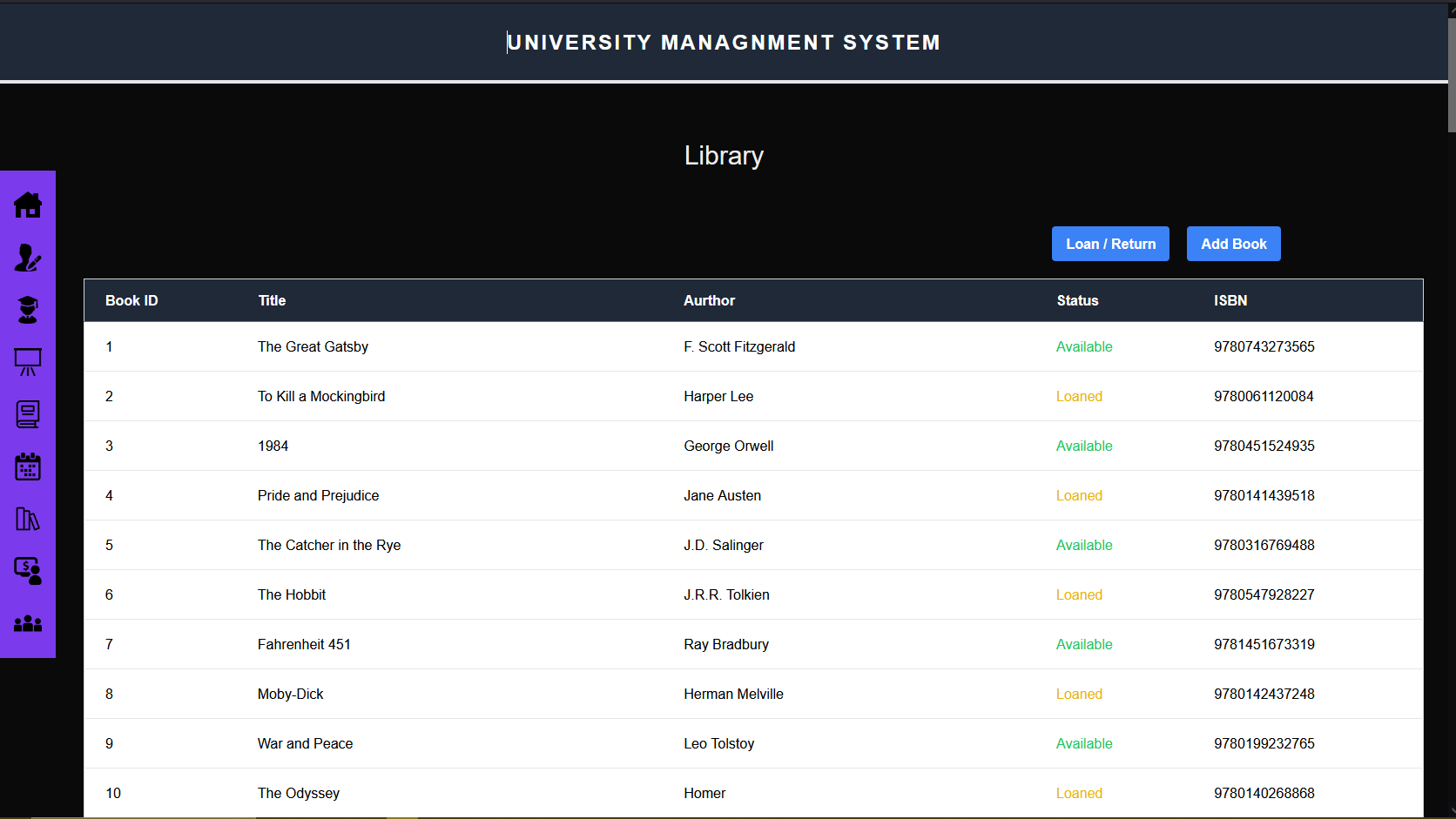
**Student Record**



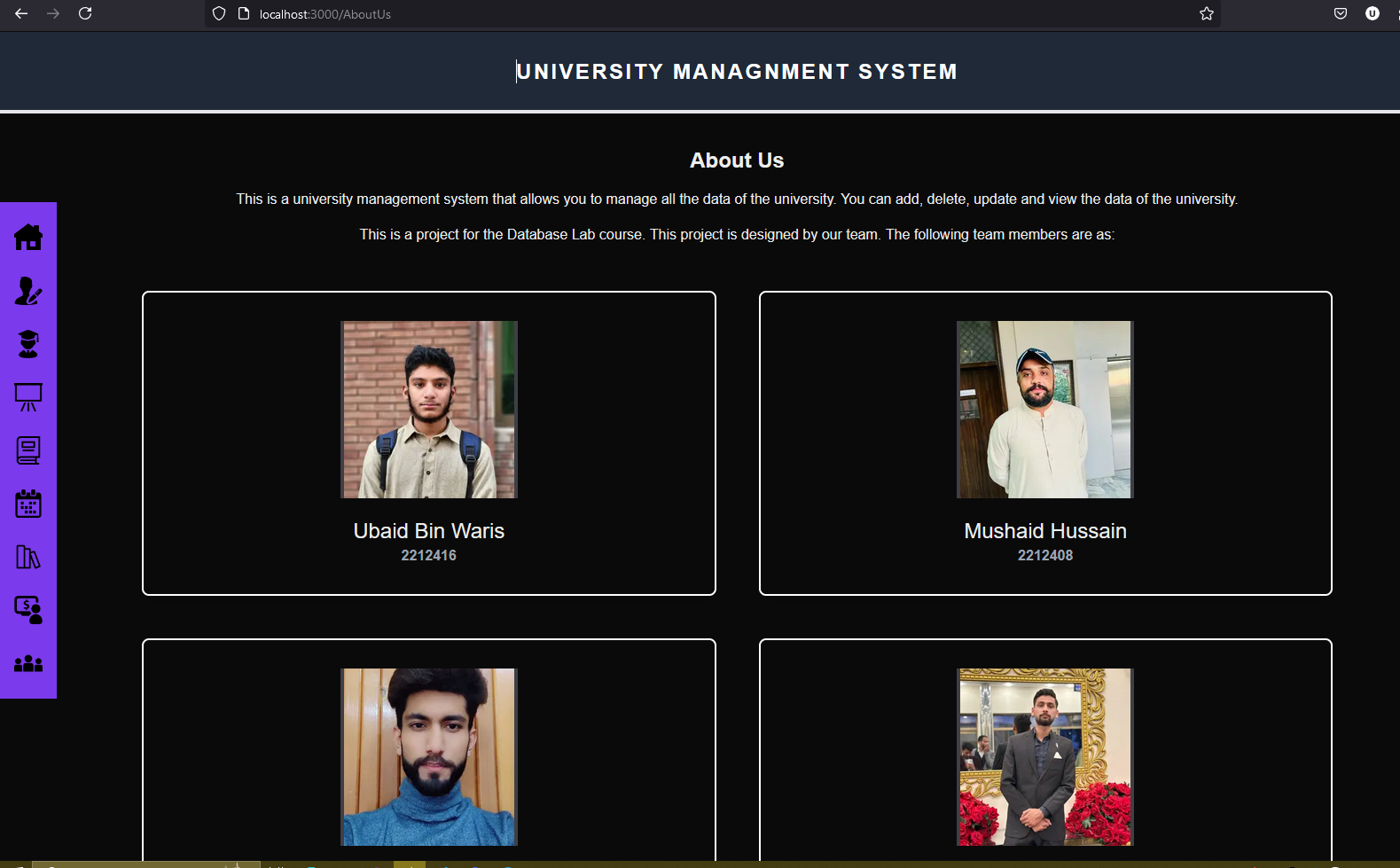
**Classroom Information**



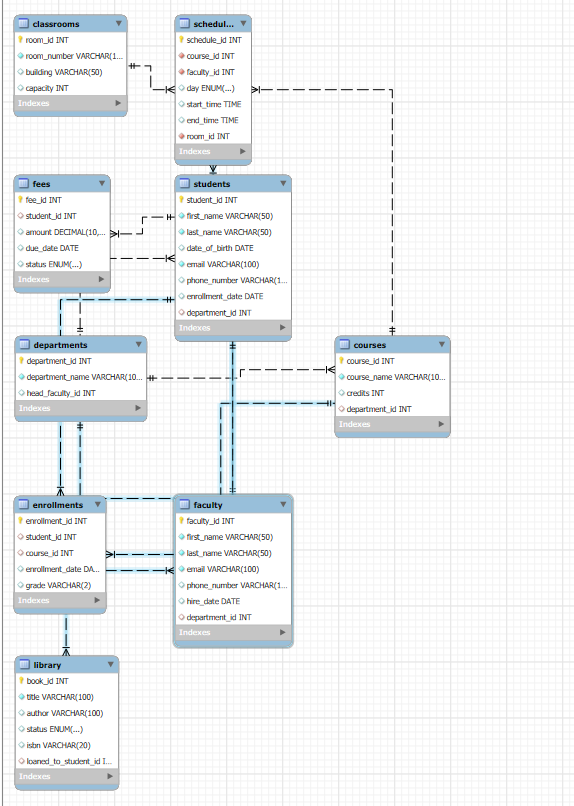
**Library**

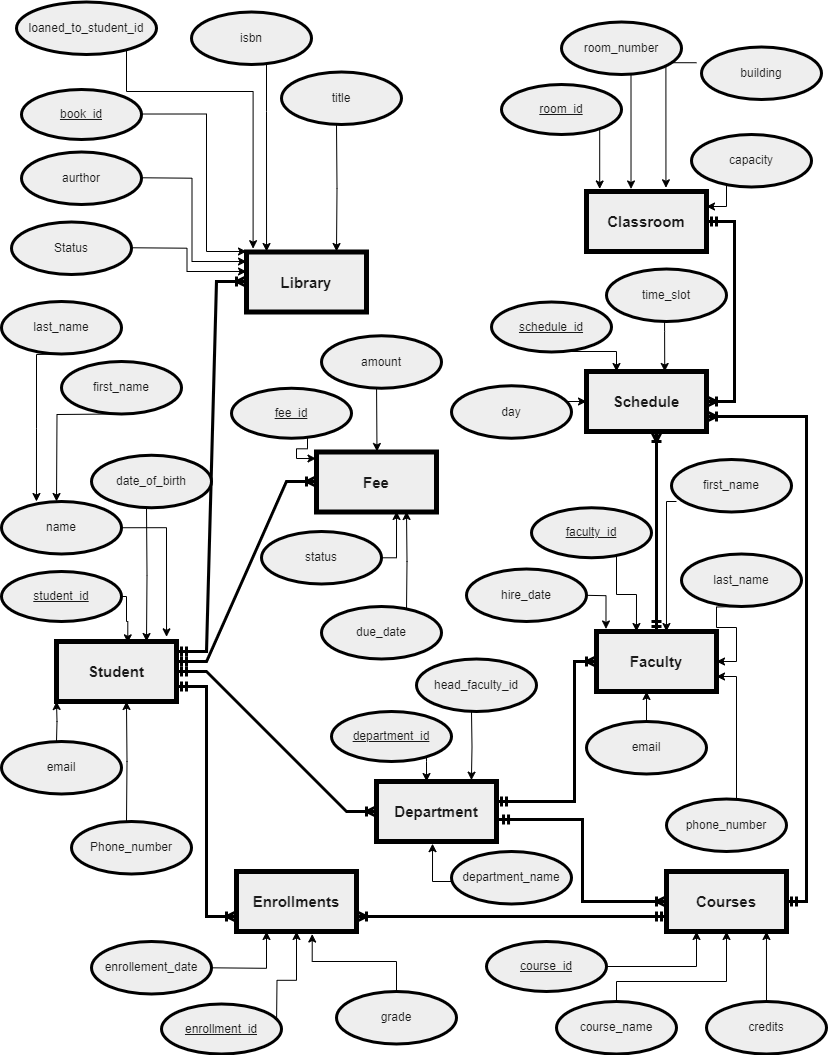


**About Us**



**ER Diagram**





**Conclusion**

The University Management System project effectively integrates a relational database and a user-friendly front-end interface to streamline the administration and management of various university operations. The database schema is designed to ensure data integrity, efficient retrieval, and secure storage of essential information, including departments, students, faculty, courses, enrollments, schedules, fees, and library records. The use of foreign keys and unique constraints strengthens the system's consistency and prevents errors or redundancies in data management.

On the front-end, Next.js enhances the user experience with responsive and intuitive dashboards for students, faculty, and administrators, ensuring seamless interaction with the system across different devices. The system's modular design supports various functionalities, including course registration, timetable management, fee tracking, and library management, making it a comprehensive solution for university administration.

By providing real-time data updates, easy access to information, and efficient management tools, this system significantly improves administrative workflows, reduces manual effort, and enhances overall operational efficiency. The project not only meets the current needs of university management but also lays the foundation for future scalability and further integration with other tools or systems.