**Implementation of Software for University Entrance Exams and Admissions Process**

A Project Report

**Submitted**

*In partial fulfillment of the requirements for the award of the degree*



## BACHELOR OF TECHNOLOGY

**In**

**COMPUTER SCIENCE and ENGINEERING**

### By

Vempati Ajay 211FA04235

Yeluri Aswith 211FA04369

Reethika Jarugula 211FA04468

Shivam Agarwal 211FA04469

Marisetti Nandini 211FA04642

Under the Guidance of

**Mr. Bhanu Prasad**

** Senior Software Engineer, VFSTR**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY & RESEARCH**

**(Deemed to be University) Vadlamudi, Guntur -522213, INDIA.**

**May, 2025**



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

***CERTIFICATE***

This is to certify that the project report entitled **“Implementation of Software for University Entrance Exams and Admissions Process”** has been submitted by **Vempati Ajay (211FA04235), Yeluri Aswith (211FA04369), Reethika Jarugula (211FA04468), Shivam Agarwal (211FA04469), Marisetti Nandini (211FA04642)** in partial fulfillment of the requirements for the Major Project course, as part of the academic curriculum of the B.Tech. CSE Program**, Department of Computer Science and Engineering (CSE**) **at VFSTR Deemed to be University.**

|  |  |
| --- | --- |
| Project Guide | HoD, CSE |
| External Examiner | |

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

# DECLARATION

I/We hereby declare that the project work entitled **"Web Based Erp Project"** submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology (B.Tech)** in **Computer Science and Engineering at VFSTR Deemed to be University** is a record of my/our original work. This project has been carried out under the supervision of the Department of Computer Science and Engineering, VFSTR Deemed to be University. The work embodied in this thesis has not been submitted previously, in part or full, to any other University or Institution for the award of any degree or diploma. I/We have duly acknowledged all sources of information and data used in the preparation of this project report and shall abide by the principles of academic integrity and ethical guidelines.

By Vempati Ajay(211FA04235)

Yeluri Aswith(211FA04369)

Reethika Jarugula(211FA04468)

Shivam Agarwal(211FA04469)

Marisetti Nandini(211FA04642)

Date:15-05-2025

# 

# ACKNOWLEDGEMENT

We would like to take this moment to sincerely thank our Project Guide, **Mr. Bhanu Prasad**, for allowing us to pursue this project and for his continuous support, insightful guidance, and motivation throughout the entire journey of our work.

We are sincerely thankful to **Dr. Ravi Sekhar**, Dean TD,VFSTR, **Arun Kumar, Suman, Kumar, Pavan**, Critical river software limited, Hyderabad, for the consistent assistance, Motivation, timely advice, and cooperation, which were instrumental in the successful completion of our project.

We are truly grateful to **Dr. S. V. Phani Kumar**, Head of the Department of Computer Science and Engineering at VFSTR Deemed to be University, for offering us the opportunity and all the necessary facilities to carry out our project effectively.

# We extend our sincere appreciation to all the faculty members, programmers,and technical staff of the Department of Computer Science and Engineering for their valuable support, knowledge sharing, and assistance throughout our academic journey.

# Finally, we are deeply thankful to our family members for their unconditional love, constant support, and encouragement, which were crucial in sustaining our efforts and successfully completing this project.

# 

# 

# With Sincere regards,

Vempati Ajay(211FA04235)

Yeluri Aswith(211FA04369)

Reethika Jarugula(211FA04468)

Shivam Agarwal(211FA04469)

Marisetti Nandini(211FA04642)

# 

# ABSTRACT

The rapid evolution of educational institutions necessitates the implementation of integrated and scalable systems to manage academic and administrative processes. This project presents a comprehensive, web-based ERP system developed using the MERN (MongoDB, Express.js, React, Node.js) stack, designed specifically to enhance the efficiency of educational institutions. The primary objective of this system is to streamline the admission process and facilitate entrance examination management through robust, user-friendly portals. The **Entrance Exam Portal** allows prospective students to register, appear for online exams, and receive results in real time. The portal is built to handle a high volume of concurrent users, ensuring a smooth and secure testing experience. The **Admission Portal** automates the application process, enabling applicants to submit their details, track application status, and receive admission offers seamlessly. The system ensures data integrity and security through role-based access control and JWT authentication. Utilizing the MERN stack offers a dynamic and responsive user experience, while MongoDB ensures efficient data handling and scalability. The application’s modular architecture and RESTful API integration make it adaptable to evolving institutional requirements. This ERP solution not only reduces manual workload but also enhances decision-making through real-time data analysis, thereby contributing significantly to the digital transformation of educational administration.

.

**TABLE OF CONTENTS**

**1.Introduction**

a) Brief project goal and purpose.

b) Scope of the project.

**2. Technologies Used**

2.1 Programming languages

2.2 Frameworks (e.g., React).

2.3 Databases (e.g., MySQL, MongoDB).

2.4 Tools (e.g., Git).

**3. Existing Work (Literature Survey)**

3.1 Overview of current solutions.

3.2 Limitations of existing systems.

**4. Requirements**

4.1 Software Requirements

4.2 OS (Windows).

4.3 Development tools (VS Code).

4.4 Libraries/Dependencies.

4.5 Hardware Requirements

4.6 Processor, RAM, Storage.

4.7 Server/Cloud specifications.

**5. Implementation Details**

5.1 Enhanced UI/UX

5.2 Improvements in design and usability.

5.3 System Integration

5.4 APIs, third-party services, or institutional systems.

5.5 Automated Decision Support

5.6 Database Structure

5.7 Tables & Relationships.

5.8 Architecture

**6. Implementation**

6.1 Code Snippets

6.2 Key algorithms/functions.

**7. Testing**

7.1 Unit, Integration, System Testing.

7.2 Identified issues & fixes.

**8. Output Screens / results**

8.1 Final product screens with explanations.

**9. Conclusion & Future Work**

9.1 Summary of achievements.

9.2 Possible enhancements.

**10. References**

10.1 References.

**CHAPTER 1:**

**INTRODUCTION**

## 1. INTRODUCTION

In the rapidly evolving world of modern web development, **practical experience through hands-on projects** is no longer optional—it is essential. With the increasing demand for dynamic, scalable, and interactive web applications, theoretical knowledge alone cannot equip developers to handle real-world challenges. Understanding this, we embarked on a comprehensive and immersive journey into full-stack development using the **MERN stack** MongoDB, Express.js, React.js, and Node.js. This stack is known for its JavaScript uniformity across the entire application stack and is widely adopted for building modern, high-performance applications.

Our project made extensive use of **React.js** combined with **Bootstrap** for creating an aesthetically pleasing and responsive frontend. Meanwhile, **Express.js** was employed to handle backend routing and API creation, and **MongoDB** served as the NoSQL database for storing and managing application data in a flexible and scalable manner. To further enhance code structure, maintainability, and future extensibility, we followed the **Model-View-Controller (MVC)** architectural pattern and incorporated **TypeScript** during the second phase of development. TypeScript, with its static typing and clear structure, helped enforce best coding practices and significantly improved our debugging efficiency and scalability potential.

The project development lifecycle was divided into two major phases:

**Phase 1: Exploratory Learning Through Foundational Tasks**

The **first phase** focused on building a strong foundation by completing **seven key tasks**, each designed to expose us to a specific feature or challenge commonly encountered in full-stack development.

1. **Student Registration Form**  
   Our journey began with building a user-friendly student registration form. This involved designing forms with input validation, dropdown menus, and proper layout styling using Bootstrap. The data entered by the students was stored in MongoDB through backend integration. This task gave us our first exposure to CRUD operations and form handling in React.
2. **Password Setup and Login System**  
   Security being a crucial aspect of any application, the next task was to build a secure password setup and login module. Here, we implemented password hashing using libraries like bcrypt and managed sessions or JWT tokens to authenticate users. This task helped us understand how to implement **authentication and authorization** mechanisms that are essential for protecting user data.
3. **International Student Registration Form**  
   Recognizing the need for inclusivity and customization, we extended the registration form to cater specifically to **international students**. This involved conditionally rendering fields based on user selection and adding backend filters to differentiate between domestic and international entries. It taught us about the importance of flexibility in user interface design and dynamic backend querying.
4. **Displaying International Student Data**  
   Once international students were registered, we created a module to display their data selectively. This involved sending API requests, fetching student records, and presenting them neatly using components in React. It was a crucial step in understanding **asynchronous data fetching, state management**, and component lifecycle methods.
5. **Certificates Management Page**  
   This module allowed students to upload, view, update, or delete academic certificates. Implementing this feature required mastering file handling on both the client and server sides. We used tools such as Multer for file uploads, and implemented role-based access control to ensure that only authenticated users could manage their files. It also required setting up appropriate data models and routes to support **file storage and retrieval in MongoDB or the file system**.
6. **Quiz Exam Page**  
   One of the more complex tasks in this phase, the quiz module, required fetching dynamically generated questions from the database, displaying them sequentially, and evaluating the answers. We created an **Instructions Page** to guide the users, a **Quiz Interface** to handle the questions, and a **Results Page** to show the final score. We implemented timers, state tracking, and score calculation logic. This task significantly boosted our understanding of **dynamic rendering, state control, conditional logic**, and interaction design in React.

**Phase 2: Advanced Modular Implementation Using MVC and TypeScript**

Having completed the foundational phase, we transitioned into the **second and more**

**structured phase**, where we built upon our learning to implement core application features using a well-organized, scalable codebase. Here, we adopted the **MVC pattern**, ensuring a clear separation between data models, business logic, and user interface layers. This allowed for cleaner code, easier maintenance, and better collaboration between team members.

The switch to **TypeScript** played a pivotal role in improving the robustness of our application. With features like static type checking, interfaces, and type inference, TypeScript made our code more reliable and easier to debug, especially as the project grew in complexity. It also aligned us with current industry practices, as many large-scale applications now use TypeScript for backend and frontend development.

In this phase, we began integrating the initial tasks into a cohesive full-stack application. We enhanced user authentication with advanced features like **session persistence**, **password reset**, and **multi-role access** (e.g., student vs. admin). Our backend services were modularized into **controllers, services, and routes**, and our frontend was divided into reusable **React components** for scalability.

**1.1. Brief project goal and purpose.**

The primary objective of our final year project was to design and develop a **comprehensive, scalable, and user-centric academic portal** tailored for modern educational institutions. Recognizing the increasing complexity of administrative processes in colleges and universities, our mission was to craft a solution that not only meets academic project requirements but also provides tangible value to educational organizations by addressing real-world operational challenges.

Traditional academic management practices are often plagued by inefficiencies such as delays, human error, limited accessibility, and lack of data integrity. These issues become even more significant as institutions scale in size and complexity. With this in mind, our proposed system focuses on **digitizing and automating** critical academic workflows—ranging from student registration and admissions to examination management, document verification, and seat allotment. By transitioning from manual to digital processes, the platform significantly reduces the administrative burden on staff while enhancing the overall experience for students.

The platform is designed to serve two major user groups: students and institutional staff. From the student’s perspective, the portal offers an **intuitive, user-friendly interface** to facilitate quick access to academic records, examination details, hall tickets, and admission documents. Students can seamlessly register, update personal information, apply for concessions, view seat allotments, and even participate in interactive activities like quizzes and surveys. On the other hand, the administrative panel provides staff members with tools to efficiently manage admissions, allot seats based on pre-defined criteria, verify documentation, and handle requests with minimal manual intervention.

To ensure robustness and maintainability, the project was structured using the **Model-View-Controller (MVC) architecture**, along with **TypeScript** and modern development frameworks. This approach ensured that the system is modular and cleanly divided into distinct components, making it easier to debug, scale, and enhance in the future. Using MVC also allowed us to simulate a professional software development environment, reinforcing best practices in code separation, state management, and interface design.

One of the notable achievements of our system is the **automation of academic workflows**. Features such as question paper generation, visual representation of seat allotment, dynamic form validation, and real-time document tracking help streamline day-to-day operations. In addition, the portal includes smart features such as real-time data synchronization, role-based access control, and customizable dashboards, which are critical for maintaining control and ensuring data confidentiality across departments.

Our academic portal also includes an embedded document management system for storing and verifying student certificates, ID proofs, admission forms, and exam-related materials. This not only reduces paper usage and storage overhead but also makes it easier to audit and retrieve records when needed. The document verification process includes status indicators and administrative approval workflows, offering a transparent and accountable system for both students and staff.

Furthermore, by implementing real-world project scenarios such as form validation, quiz handling, user authentication, and backend services integration, we were able to **mimic enterprise-level academic systems**. This experience enriched our understanding of full-stack development and taught us how to integrate technologies like MongoDB, Express, React, and Node.js within an MVC framework to create a seamless user experience.

Security and scalability were two pillars of our design philosophy. We ensured that all user inputs are validated and sanitized to prevent unauthorized access or injection attacks. Additionally, the portal can be scaled horizontally to support increasing data volume and concurrent users, making it suitable for deployment in mid-to-large-sized institutions.

* 1. **. Scope of the project.**

The scope of this project is extensive and spans across multiple functional areas of a higher education system. To organize our efforts effectively, we structured the project into two main modules following the initial set of learning tasks.

The **first module** focused on academic content management and student data handling. It included the preparation of 10 subject-wise question papers, modeled after previous exam patterns in subjects such as **Accounting, Biology, Business, Chemistry, Economics, English, History, Mathematics, Physics, and Political Science**. These question papers are maintained in structured Excel sheets and can be dynamically retrieved and managed through the web interface. Alongside, we also managed an extensive dataset comprising **6000 student records**, each containing multiple fields: Admission Number, Batch Number, Full Name, Registration Number, Gender, Stream, District, Phone Number, and Educational Qualification. The scope of this module includes not only the entry and validation of data but also bulk uploads, updates, and backend verification for academic administrative use.

The **second module** revolves around the **Seat Allotment System**, a key part of college admissions and post-admission processes. It includes five critical interfaces:

1. **Provisional Seat Allotment Order Generation Page**: This page captures complete student admission details, including the course they are admitted into, applicable fees, scholarships granted, and instructions for confirming or canceling the seat. This serves as an official provisional document for the student.
2. **Allotment Order Visual Representation (M.Tech Admission)**: This feature visualizes M.Tech seat allotments, showing admission numbers, scholarship status, and institutional terms for seat confirmation.
3. **Ph.D. (External) Program Allotment Page**: Specialized for Ph.D. candidates, this page outlines their program structure, joining date, and admission policies—especially useful for handling external candidate requirements.
4. **Student Seat Allotment View (DEO Login)**: This role-based page is intended for Data Entry Operators (DEOs) to verify and track allotment by Admission Number, enabling institutional staff to streamline updates.
5. **Concession Modify Screen**: This screen is vital for administrators to adjust scholarship or fee concession details post-admission. It includes access controls and version tracking to avoid misuse or inconsistency.

The scope of our project extends beyond just data entry and display. It includes role-based access, dynamic content rendering, real-time backend integration using Express.js, and robust frontend interactions with React.js and Bootstrap. We also used Excel files as primary data sources for offline access and easy validation, ensuring that institutions not fully digitized can still leverage the application with minimal training.

Moreover, by using the **MVC architecture with TypeScript**, the project achieves a high level of maintainability. Each component—models for schema and data, views for UI rendering, and controllers for logic handling—is decoupled for better debugging and future enhancement. TypeScript adds static type-checking and reduces bugs during development, enhancing the reliability and readability of the codebase.

**CHAPTER 2:**

**TECHNOLOGY USED**

**2. TECHNOLOGY USED**

The development of this project leveraged a rich blend of modern and widely adopted technologies that align with industry standards. These tools and platforms were strategically selected to ensure seamless performance, scalability, maintainability, and user-friendly experiences throughout both the initial tasks and the final modules of the project. The core technology stack revolves around the MERN stack—MongoDB, Express.js, React.js, and Node.js—a powerful combination that facilitates full-stack JavaScript development. This stack provides an end-to-end development environment using a single language (JavaScript/TypeScript), simplifying integration and speeding up development cycles.

**2.1. Programming languages:**

* **TypeScript**: The frontend of the application is built using **TypeScript**, a statically typed superset of JavaScript. TypeScript enhances code reliability by enforcing **strict type safety**, allowing developers to catch errors during development rather than at runtime. This is especially beneficial in **large-scale applications** where numerous components and team members are involved. TypeScript supports **modern JavaScript features** such as ES6+ syntax while adding features like **interfaces**, **enums**, **generics**, and **type inference**, which improve code clarity and maintainability.Using TypeScript in the frontend helps ensure consistent data structures when working with complex forms and API responses. It integrates seamlessly with popular frameworks like **React**, and enables better tooling support including **intelligent code completion**, **navigation**, and **refactoring capabilities** in modern IDEs like VSCode. In a project involving multistep forms, file uploads, and context-based state management, TypeScript significantly reduces the chance of runtime bugs and unexpected behavior.
* **JavaScript**: **JavaScript** serves as the backbone of the backend in this application, primarily through its implementation in **Node.js** and **Express.js**. It is used to handle all **server-side logic**, including request handling, authentication, form validation, routing, and database operations. JavaScript's **asynchronous nature** using **Promises** and **async/await** makes it highly suitable for building scalable REST APIs that can handle multiple concurrent client requests without blocking.On the server side, JavaScript interacts with the **MongoDB** database using libraries like **Mongoose**, facilitating schema definitions, data validation, and complex querying. Additionally, JavaScript is used in writing utility functions, middleware for logging or authentication, and in implementing custom business logic — such as aggregating data, processing uploaded files, or managing conditional logic for forms.
* **JSON**: **JSON** is the standard format used for **data interchange** between the frontend and backend of the application. It provides a **lightweight**, **human-readable**, and easily parsable format that enables seamless communication between client and server components. When a user fills out a form or uploads a file, the data is serialized into JSON and sent via **HTTP POST** requests to RESTful endpoints.On the backend, Express parses incoming JSON payloads using middleware like express.json(). The parsed data is then validated and stored in the MongoDB database. JSON is also used in server responses — whether to return success messages, error details, or queried data back to the frontend for rendering. This standardization simplifies development, debugging, and integration with third-party services or future microservices.Moreover, JSON is used to structure **application state**, **configuration files**, and even **context providers** in frontend state management (e.g., React Context or Redux). Its universal support across web technologies makes it the ideal choice for consistent and efficient data handling.

**2.2.Frameworks:**

**React.js:** React.js is a widely adopted JavaScript library for building fast, interactive, and dynamic user interfaces. Developed by Facebook, React is built around the concept of components, which allows developers to break down the UI into modular, reusable pieces. In our project, we leveraged React with TypeScript to ensure type safety, which significantly enhances developer productivity and code maintainability.By using React, we implemented a single-page application (SPA) that delivers a seamless user experience without full-page reloads. The Virtual DOM in React ensures efficient rendering by only updating parts of the UI that have changed. We structured the project using both functional components and React Hooks (useState, useEffect, useContext) to manage local and global state across different modules of the application. This was particularly useful in our multi-step forms (e.g., Personal Details, Course Details, Travel Details) where the data needed to persist between form steps and components.Furthermore, React Router was utilized for client-side routing to allow navigation between different views such as form pages, preview pages, and verification dashboards. This contributed to a better user experience and improved maintainability by cleanly separating the logic for different pages.

**Bootstrap:** Bootstrap is a powerful front-end framework used to create responsive and aesthetically pleasing web interfaces. In our application, Bootstrap CSS was used extensively to design a clean, mobile-friendly UI with minimal effort. The use of Bootstrap allowed our team to focus on functionality while maintaining consistent styling across the platform.We adopted Bootstrap’s grid system to build responsive layouts that adapt seamlessly to different screen sizes, including desktops, tablets, and smartphones. The utility classes helped streamline the development process, enabling quick implementation of margins, paddings, typography, and alignments.Additionally, pre-built components such as modals, forms, buttons, and tables were integrated and customized to align with our application’s branding. This reduced development time and ensured that UI components were both accessible and user-friendly.

**Express.js:** Express.js is a minimalist and flexible web application framework for Node.js, used to build the backend of our application. Express was crucial in setting up RESTful APIs that acted as the bridge between the frontend and the MongoDB database.Express allowed us to define a clear routing structure to handle various HTTP requests like GET, POST, PUT, and DELETE. These routes covered functionality such as form submission, file uploads, record updates, and certificate management. We used Express middleware such as express.json() and multer for parsing incoming JSON and handling file uploads, respectively.

Express also provided a robust foundation for implementing custom authentication logic, validation rules, and error handling. Middleware functions were used to maintain clean and modular logic. For example, we used separate middleware for logging, request validation, and access control.

**Node.js:**Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript engine, and it was the backbone of our backend environment. With Node.js, we were able to write server-side logic using JavaScript, ensuring consistency in language across the full stack of the application.Node’s event-driven, non-blocking I/O model made it ideal for handling real-time data requests, large file uploads, and concurrent API calls without performance bottlenecks. In our case, this architecture was essential for managing multiple form submissions and certificate uploads simultaneously.We used Node.js in conjunction with libraries such as Mongoose (for MongoDB interaction), Multer (for file handling), and Cors (for handling cross-origin requests) to enhance the capabilities of our backend. Node.js also provided a convenient way to organize our codebase into modules using require and module.exports, promoting clean architecture.

**MVC Architecture:** For the Student Entrance Exam and Admission modules, we adopted the Model-View-Controller (MVC) architecture, a well-known design pattern that separates an application into three main components:

**Model:** Represents the data structure and defines how the data is stored and retrieved from the database. We used Mongoose models to define schemas for collections like student\_iradm\_details, certificates\_status, and choice\_filling. These schemas ensured consistent data representation and provided powerful querying capabilities.

**View:** In our application, the view was handled by React components. These components render the data retrieved from the backend and offer user interaction interfaces. Each form step was a part of the View layer, dynamically displaying content based on the user’s actions and input.

**Controller:** Controllers acted as the intermediaries between Models and Views. They contained the business logic for handling incoming requests, performing validations, processing data, and sending appropriate responses. Controllers made use of services and utility functions to keep the logic modular and testable.

**2.3.Database:**

**MongoDB: MongoDB** is a powerful, open-source, **NoSQL document-oriented database** that played a central role in the backend architecture of our application. It was used to store and manage various types of data including user profiles, exam registration details, application forms, seat allocation preferences, uploaded certificates, and admission status records. One of the key reasons for choosing MongoDB was its **schema-less** structure, which gave us the flexibility to evolve the database schema as the application requirements changed during development.

Unlike traditional relational databases that use tables and rows, MongoDB stores data in **JSON-like documents** (BSON format), which are much more natural to work with in JavaScript-based environments like Node.js. This allowed for a smooth integration between the backend logic and the database, enabling faster development cycles. Each document can have a unique structure, which meant we could accommodate different sets of fields for different modules (e.g., personal details, course preferences, file uploads) without needing complex joins or rigid schema definitions. To enhance the structure and enforce consistency, we used **Mongoose**, a popular **Object Data Modeling (ODM) library** for MongoDB and Node.js. Mongoose allowed us to define strict schemas and validation rules for collections like student\_iradm\_details, certificates\_status, and choice\_filling, while still taking advantage of MongoDB’s flexibility. This hybrid approach helped us maintain data integrity without sacrificing agility.In our application, MongoDB collections were designed to represent real-world entities such as students, their application forms, uploaded files, and admission decisions. For instance, the certificates\_status collection was used to store binary file data (e.g., PDFs, images) representing certificates submitted during the application process. These files were linked to student admission numbers (adm\_no) to ensure accurate data association.

We also leveraged MongoDB’s **powerful querying capabilities** to implement features like filtering applications based on eligibility, retrieving seat allocations by branch preference, and dynamically updating student records. The ability to perform **complex queries, updates, and aggregations** using MongoDB’s native operators enabled us to handle backend logic efficiently.Additionally, MongoDB's support for **scalability and high availability** makes it suitable for growing educational systems. As the number of student records increases, MongoDB can scale horizontally across multiple nodes, ensuring performance is maintained.

**2.4.Tools:**

* **Git**: **Git** is a distributed version control system that played an essential role in our development workflow. It allowed us to **track code changes**, **maintain a complete history** of our project, and **collaborate effectively** within a team. Git made it possible for multiple developers to work simultaneously on different features or modules without interfering with one another’s work.We followed a **branching strategy**, using separate branches for features, bug fixes, and releases. This approach ensured that the main or production branch always remained stable, while development happened in isolated environments. Developers would create branches like feature/form-validation, fix/api-response, or hotfix/file-upload to organize their work. Once changes were completed and tested, we used Git to **merge** them into the main codebase via pull requests.Git also enabled **conflict resolution**, allowing us to resolve overlapping code changes thoughtfully, ensuring nothing was lost or overwritten. Regular commits with descriptive messages allowed us to maintain a **transparent and understandable project history**, which helped greatly during debugging or when reverting changes.
* **GitHub**: **GitHub** served as our central **code repository** and collaboration platform. We used GitHub to store, version, and share the application source code, leveraging features such as **pull requests**, **code reviews**, and **issue tracking** to maintain a well-organized development process.Pull requests enabled **peer reviews** before merging new code into the main branch. This not only ensured higher code quality but also fostered knowledge sharing among team members. Reviewers would suggest improvements or identify potential bugs, leading to a more robust final product.GitHub’s **issue tracker** was used to document bugs, enhancements, and development tasks. Each issue was assigned to team members, prioritized using labels, and linked to commits or pull requests for traceability. The GitHub Projects feature helped us create Kanban-style boards to manage development sprints and track progress visually.We also used **GitHub Actions** for basic CI/CD (Continuous Integration and Deployment), automating tasks such as code linting and test execution. This enhanced the reliability of our code before deployment.
* **Postman**: **Postman** is a powerful tool for **API testing**, and it was crucial in validating the backend endpoints of our application. During development, Postman allowed us to simulate **frontend requests** to the backend APIs, ensuring they behaved as expected before integration.We created organized **collections of API requests**, grouping them by functionality (e.g., user registration, student detail submission, file uploads, concession updates). These collections helped test different HTTP methods—**GET, POST, PUT, DELETE**—with appropriate request headers, body payloads, and parameters.
* **Visual Studio Code (VS Code)**: **Visual Studio Code**, or **VS Code**, was the **primary development environment** for our frontend and backend code. It is a lightweight yet powerful IDE with a vast ecosystem of extensions and built-in support for JavaScript, TypeScript, Node.js, and MongoDB.We used VS Code’s **IntelliSense** to get smart autocompletion, code navigation, and error highlighting, especially useful when working with TypeScript. Its support for **debugging** Node.js applications allowed us to set breakpoints, inspect variables, and step through server-side code with ease.Popular extensions such as **Prettier**, **ESLint**, and **Bracket Pair Colorizer** helped enforce consistent coding styles and detect syntax issues early. For Git operations, VS Code’s integrated **Source Control panel** allowed us to stage, commit, and merge changes without leaving the editor.We also benefited from the integrated **terminal**, where we could run commands like npm start, nodemon, MongoDB shell commands, and git operations, all within the same interface.VS Code’s **workspace settings** allowed our team to standardize development setups across machines, ensuring consistency in formatting, linting, and build configurations.
* **npm (Node Package Manager)**: **npm**, the **Node Package Manager**, was used to **install, manage, and update** all third-party packages and dependencies across both the frontend and backend of our project. It provided access to a vast ecosystem of libraries that accelerated development and enabled modern features without building everything from scratch.In the frontend, we used npm to manage packages like:

react, react-dom: Core React libraries for building the UI.

react-router-dom: Client-side routing.

axios: HTTP client for API requests.

bootstrap: UI styling and responsive design.

typescript: Type-checking for better code quality.

* In the backend, we relied on packages like:

express: Web framework for routing and middleware.

mongoose: MongoDB ODM for data modeling and queries.

cors: Handling cross-origin requests.

multer: File upload handling.

dotenv: Loading environment variables securely.

* npm also managed **scripts** in package.json for tasks like starting servers (npm start), running development servers with hot reloading (npm run dev), and running tests. Regular npm audit and npm outdated checks helped us maintain secure and up-to-date dependencies.
* **MongoDB Compass MongoDB Compass** is a **GUI (Graphical User Interface)** tool provided by MongoDB that we used extensively to visualize and manipulate our database collections. While our application interacted with MongoDB programmatically via Mongoose, Compass allowed us to **inspect collections manually**, perform **ad hoc queries**, and **debug issues** quickly.Using Compass, we could:View documents in collections like student\_iradm\_details, certificates\_status, and choice\_filling.Run complex filter queries using MongoDB’s syntax to test data logic before writing code.Create, update, or delete documents directly to simulate different test scenarios.Monitor schema variations across documents, which was particularly useful given MongoDB’s schema-flexible nature.Analyze document counts, index usage, and query performance.This tool significantly helped our QA and debugging processes, especially when tracking down data mismatches or verifying the results of update and aggregation operations
* **Excel**: Although our application was web-based and stored data in MongoDB, **Excel** played a supportive role in certain workflows. In scenarios such as **importing hall ticket records**, **exporting student data for reports**, or **tracking admission lists temporarily**, Excel integration proved valuable.
* We used **npm packages like xlsx and exceljs** to implement this functionality:
* **xlsx** helped us **read Excel files** uploaded by administrators. This allowed us to bulk import hall ticket or eligibility data into MongoDB without manual entry.
* **exceljs** enabled **exporting MongoDB data into structured Excel sheets**, which administrators could download for review or reporting purposes.
* These capabilities were especially helpful in bridging traditional administrative workflows with the digital application. Instead of re-entering data, users could upload .xlsx files with hundreds of rows, and our backend parsed and stored them appropriately. Likewise, for audit or offline analysis, Excel downloads gave users portable versions of dynamic database content.

**CHAPTER 3:**

**EXISTING WORK**

### 

### 3. EXISTING WORK

### 3.1 Overview of current solutions.

In recent years, educational institutions have increasingly adopted digital systems to manage student data, academic records, online examinations, and admission processes. Several solutions have emerged that aim to streamline student information management, including enterprise resource planning (ERP) systems such as TCS iON, Fedena, and CampusNexus. These platforms typically offer centralized student databases, admission workflows, document uploads, and result publishing capabilities. Similarly, learning management systems (LMS) like Moodle and Canvas facilitate course delivery, quiz generation, and student progress tracking. Some institutions also use custom-built solutions using Java or PHP, often backed by relational databases like MySQL or PostgreSQL.

Additionally, there are isolated web portals designed specifically for examination and seat allotment purposes. For example, government university portals often host online counselling processes for entrance exams, including displaying seat allotment results, provisional admission letters, and fee payment guidelines. These platforms are typically built with older technologies, sometimes using plain PHP or ASP.NET frameworks, which may be functional but lack flexibility, modularity, or modern design principles.

At the technical level, many of these systems are monolithic and not built using modern JavaScript-based frameworks like React or Express. They often rely on server-side rendering, and their architecture does not allow for real-time data updates, scalable microservices, or reusable component structures. Moreover, static HTML interfaces dominate, offering limited interactivity, minimal responsiveness, and poor user experience across devices.

### 3.2 Limitations of existing systems.

Despite the widespread availability of student management and examination systems, there are notable limitations in current implementations. One of the major issues is **lack of modularity and extensibility**. Traditional systems are often rigid, and introducing a new feature—such as a separate module for international students, or a dynamic quiz generation tool—requires significant codebase modifications. This results in high maintenance costs and a greater risk of introducing bugs.

Another limitation is the **absence of real-time, interactive user interfaces**. Many systems use outdated frontend technologies that fail to provide a seamless user experience. Forms are often lengthy, poorly validated, and not mobile-friendly. In contrast, modern solutions like ours leverage React.js and Bootstrap to offer dynamic interfaces that respond to user input in real-time and adapt gracefully to various devices.

**Data management and scalability** present further challenges. Legacy systems typically use relational databases with rigid schemas, making it difficult to handle varied or evolving data structures—such as dynamically changing question formats or varied student qualification details. NoSQL databases like MongoDB, as used in our project, offer better scalability and schema flexibility, allowing us to store a wide range of student and exam data without reengineering the entire backend.

Another common shortcoming is the **limited integration between modules**. For example, in many existing setups, the examination module operates independently of the admission or seat allotment systems. This leads to data redundancy, increased processing time, and a fragmented user experience. In contrast, our system follows a unified architecture using the MVC pattern, ensuring that all modules—student registration, certificate uploads, quiz handling, and seat allotment—are seamlessly connected and maintain a shared data model.

Security and role-based access control are also often lacking or underdeveloped in older systems. For instance, a Data Entry Operator (DEO) or administrator might not have a separate login or interface, leading to potential data integrity issues. In our solution, we’ve taken care to include role-based functionalities, such as a DEO login to handle specific admission number queries, ensuring that sensitive operations are performed by authorized personnel only.

Lastly, existing systems often fall short in **customization and personalization**, particularly in areas like provisional seat allotment orders. These are usually static documents generated manually or semi-automatically, without taking into account different programs (e.g., M.Tech or Ph.D.), scholarships, or individual instructions. Our system, in contrast, dynamically generates these orders with precise formatting, varied fields, and editable components for maximum flexibility.

### 3.3 Comparison Tables.

### Features Comparison Table

|  |  |  |
| --- | --- | --- |
| Feature | Existing System | Proposed System |
| Registration Process | Partially manual | Fully automated |
| Slot Booking | Limited options | Flexible and modifiable |
| Result Processing | Manual and time-consuming | Automated and efficient |
| Receipt Generation | Not available | Automatically generated |
| CRUD Operations | Limited functionality | Comprehensive features |

### Efficiency Comparison Table

|  |  |  |
| --- | --- | --- |
| Parameter | Existing System | Proposed System |
| Time for Verification | High (Manual) | Low (Automated) |
| Error Rate | High (Human errors) | Low (System accuracy) |
| User Experience | Basic | Intuitive and user-friendly |

**CHAPTER 4:**

### REQUIREMENTS

### 

### 4. REQUIREMENTS

Before embarking on the development of the Student Entrance Exam and Admission Portal, it was essential to identify the necessary software and hardware resources required to build, test, and deploy the application. This section outlines both the software and hardware specifications required to ensure the smooth development and execution of the project, while also taking into account scalability, performance, and ease of development.

**4.1. Software Requirements**

The project is built using the MERN stack—MongoDB, Express.js, React.js (with TypeScript), and Node.js. This combination offers a complete JavaScript-based solution from front-end to back-end and database handling. To ensure proper project setup and smooth development experience, several key software components were required:

* **Node.js (v18+)** – For running backend services and managing packages through npm.
* **MongoDB (Community or Atlas)** – NoSQL database for storing user information, registration data, questions, slot details, and admission records.
* **Express.js** – Lightweight web framework for building RESTful APIs that connect the frontend to the backend logic and database.
* **React.js (with TypeScript)** – For building responsive and modular front-end components, making the user interface clean and interactive.
* **Bootstrap 5** – CSS framework to ensure consistent and mobile-responsive design across all devices.
* **Postman** – For API testing and validation during backend development.
* **ExcelJS / XLSX Libraries** – Used for reading, writing, and exporting Excel files for hall ticket generation and admin reporting.
* **Mongoose** – Object Data Modeling (ODM) library for MongoDB that simplifies schema creation and data queries in Node.js.
* **Dotenv** – To manage environment variables securely.
* **CORS & Body-Parser Middleware** – To handle cross-origin requests and payload processing.

**4.2. OS (Windows/Linux).**

The development environment was designed to be platform-independent, but the actual work was primarily carried out on the following operating systems:

* **Windows 10/11** – Main development OS for both frontend and backend services.

**4.3. Development tools (VS Code, Eclipse).**

A set of reliable and developer-friendly tools were chosen to ensure efficient coding, debugging, and version control. These tools were essential for both frontend and backend development.

* **Visual Studio Code (VS Code)** – The primary code editor used for writing TypeScript, JavaScript, and JSX code. It supports Git, linting, auto-completion, and extensions for React and Node.js.
* **MongoDB Compass** – GUI tool for viewing and querying the MongoDB database locally.
* **Git & GitHub** – For version control and team collaboration, managing commits, branches, and pull requests.
* **Terminal/Command Prompt** – For executing Node, npm, and Git commands during development.
* **Google Chrome Developer Tools** – For inspecting frontend rendering, debugging, and testing responsiveness.

**4.4 Libraries/Dependencies.**

The project made use of several open-source libraries and dependencies to implement specific functionalities with ease and reduce development time. The major ones include:

* **React Router DOM** – For navigating between pages in the React application.
* **Axios** – For making HTTP requests from the frontend to the backend.
* **React Hook Form & Yup** – For form validation and management.
* **React Icons / FontAwesome** – For incorporating visual icons across the UI.
* **Moment.js / Day.js** – For handling date and time formatting, especially in exam scheduling and hall ticket generation.
* **Bcrypt.js / JWT** – For password hashing and implementing secure login and authentication flows.

**4.5 Hardware Requirements**

The project was designed to run efficiently on moderate hardware typically available to students and developers. The following are the minimum and recommended configurations:

* **Processor**: Intel i3 or AMD equivalent (dual-core)
* **RAM**: 4 GB
* **Storage**: 10 GB free disk space
* **Display**: 1366 x 768 resolution

**4.6 Processor, RAM, Storage.**

* **Processor**: Intel i5 or higher (quad-core) for smoother development and faster builds
* **RAM**: 8 GB or higher
* **Storage**: SSD with at least 20 GB free space for tools, dependencies, and MongoDB data
* **Graphics**: Basic integrated graphics sufficient for web development

**4.7 Server/Cloud specifications.**

For testing and potential deployment, a basic cloud server environment is suitable. Depending on traffic and database size, the following configurations are suggested:

* **Cloud Provider**: Heroku (for small-scale deployment), Render, Railway, or DigitalOcean / AWS EC2 for production-level deployment
* **Server Configuration**:
  + **Processor**: 1 vCPU minimum
  + **RAM**: 512 MB to 1 GB for development; 2 GB+ recommended for production
  + **Storage**: 10 GB SSD (MongoDB and static file storage)
  + **OS**: Ubuntu 20.04 LTS or Amazon Linux
* **MongoDB Atlas**: Cloud-hosted MongoDB with backup, scaling, and monitoring capabilities.
* **Domain & SSL**: (Optional) To ensure secure HTTPS connections using providers like Namecheap or Let’s Encrypt.

**CHAPTER 5:**

### SYSTEM DESIGN

**5. SYSTEM DESIGN**

System design is the backbone of any software project, as it lays the groundwork for how individual components interact, function, and respond to user input. Our MERN stack-based student portal emphasizes both functionality and user experience. This section details the design thinking that shaped the user interface, back-end systems, API connectivity, and overall system architecture for modules like **Student Entrance Exam** and **Admission Management**.

The system developed has different actors which plays different role in the system and also has different functionalities in the system. Below is a description of each:

**1. Roles, Stages, and Models**

**Actors and Their Roles**

**1. Students**

* Register and fill out their profile.
* Book exam slots.
* Attempt entrance examinations.
* Upload required certificates.
* Apply for academic seats.
* Download admission letters and fee receipts.

**2. Administrators**

* Manage student data and perform CRUD operations.
* Oversee exam management.
* Upload exam questions and create sets.
* Modify or manage slot availability.
* Process and publish results.
* Oversee fee management.

**3. Data Entry Operators (DEO)**

* Verify student profiles and certificates.
* Validate uploaded documents and data entry accuracy.
* Coordinate between students and the dean’s office.

**4. Dean**

* Allocate seats based on eligibility and verification.
* Approve final admissions.
* Authorize document and fee approvals.

**2. Methods and Processes**

**Student-Facing Functionalities**

**1. Registration and Profile Completion**

* Students create an account using basic information (email, phone, etc.).
* Profile includes personal, academic, and travel details.
* All fields are validated before submission.

**2. Slot Booking**

* Students choose an available time slot for the entrance exam.
* The system disables slots once capacity is reached.

**3. Exam Management**

* Students log in during their selected slot to access exam content.
* The system tracks time and auto-submits upon expiry.

**4. Certificate Uploads**

* Students upload required certificates (e.g., SSC, Inter, B.Tech).
* Files are stored as blobs in MongoDB and tagged by student ID.
* Supported formats: JPG, PNG, PDF.

**5. Seat Application**

* Students select their preferred branches/programs.
* Required documents must be uploaded before applying.
* DEO is notified upon application completion.

**Admin/DEO-Facing Functionalities**

**1. Verification & Dean Allocation**

* DEO verifies the student’s profile, certificates, and eligibility.
* Upon verification, applications are sent to the Dean.
* The Dean allocates seats or marks students as ineligible.

**2. Question Upload and Set Creation**

* Admin uploads individual questions or bulk CSV files.
* Question sets are created by grouping questions into predefined patterns.
* Each set is uniquely tagged for tracking.

**3. Result Processing**

* After the exam, answers are auto-evaluated based on answer keys.
* Marks are stored and results published in student dashboards.
* Cutoffs and eligibility are auto-calculated based on predefined rules.

**4. Slot Modifications**

* Admin can manually open/close slots.
* Students receive notifications if rescheduling is needed.

**5. CRUD Operations**

* Admin can create, update, or delete:
  + Student profiles
  + Question banks
  + Seat allocation data
  + Branch/program information

**6. Fee Payment and Receipts**

* Once a seat is allocated, students must complete fee payment.
* Payment gateway or offline submission is integrated.
* On payment success, system generates:
  + Admission letter (PDF)
  + Fee receipt

**3. Detailed Process by Actor**

**A. Student Stages**

**1. Registration and Profile Filling**

* Register with email, phone, and password.
* Fill forms for:
  + Personal details (Name, DOB, Address)
  + Travel details (Bus pass, local station)
  + Educational details (10th, 12th, UG marks)

**2. Slot Booking for Exams**

* Choose from available dates and times.
* System confirms slot and locks it.

**3. Writing Exams**

* Log in during booked slot.
* Start the test timer.
* Submit answers or system auto-submits after time ends.

**4. Seat Application and Certificate Upload**

* Apply for branches via ranked choices.
* Upload mandatory certificates.
* Wait for DEO verification and Dean’s decision.

**5. Downloading Admission Letter and Receipts**

* Post verification and seat allocation:
  + Pay fees
  + System auto-generates:
    - Admission letter
    - Receipt with payment details

**B. Admin Stages**

**1. Verification of Student Profiles**

* View unverified profiles.
* Cross-check submitted details and documents.
* Mark as “Verified” or “Rejected.”

**2. Question Upload and Set Creation**

* Upload individual questions or import via Excel.
* Use the UI to group questions into sets for each program.

**3. Result Processing and Slot Modifications**

* View automatic result summaries.
* Manually intervene in case of discrepancies.
* Modify slot status if needed.

**4. Seat Allocation and Fee Handling**

* Filter verified and eligible candidates.
* Allocate branches based on availability and merit.
* Confirm allocation and notify students for fee payment.

**5. CRUD Operations**

* Admin can:
  + Edit/delete question banks.
  + Modify student or DEO data.

**5.1. Enhanced UI/UX**

User Interface (UI) and User Experience (UX) were central to our system design goals. The application was built with **React.js** and styled using **Bootstrap CSS** to provide a clean, responsive, and intuitive user experience for both students and administrative users.

* **Consistent Design Language**: Bootstrap was used to maintain uniform styling, color schemes, and spacing.
* **Responsive Layouts**: All pages are mobile-friendly, supporting students accessing the portal from tablets or smartphones.
* **Interactive Elements**: React's component-based architecture allowed the use of dynamic forms, modals, spinners, and tooltips to enhance engagement.

**5.2. Improvements in design and usability.**

Compared to basic CRUD apps or manual admission processes, this system brought major usability upgrades:

* **Multi-step Forms**: Forms were broken into multiple screens (e.g., personal details, travel, qualifications) to avoid overwhelming users.
* **Progress Tracking**: Visual indicators showed how far the user was in completing the admission process.
* **Validation Feedback**: Real-time form validation was implemented using React Hook Form and Yup, preventing incorrect entries early.
* **Action Buttons with Tooltips**: Admin interfaces include hover-based hints for faster decision-making.

**5.3. System Integration**

The project follows the **MVC architecture**, allowing clear separation of concerns:

* **Model**: Handles MongoDB collections using Mongoose schemas.
* **View**: Frontend built with React.js + TypeScript.
* **Controller**: Express.js routes, business logic, and middleware.

React communicates with Express-based APIs through **Axios**, ensuring modular integration between frontend and backend layers.

**5.4. APIs, third-party services, or institutional systems.**

The system uses RESTful APIs for all interactions between the frontend and backend. Some examples include:

* POST /api/register – Register a new student.
* GET /api/exam-slots – Fetch available entrance exam slots.
* POST /api/certificates/upload – Upload graduation certificates.
* PUT /api/admissions/verify – DEO updates student admission status.

**Third-party tools/services**:

* **ExcelJS**: For generating hall tickets and exporting seat data to Excel.
* **Cloudinary or Firebase (optional)**: Can be used for image/file storage if scaled up.
* **JWT**: For secure session management and route protection.

**5.5. Automated Decision Support**

The application integrates logic-driven decision-making, reducing the need for manual processing:

* **Slot Availability Logic**: Users can only see available slots and modify if still within time limits.
* **Admission Filtering**: DEO can filter students using parameters (year, program, branch) with auto-calculated results.
* **Conditional Button States**: Accept/reject buttons are shown only when a student’s data is complete and valid.

These automated features help streamline workflows for both students and administrative staff.

**5.6 Database Structure**

The project utilizes MongoDB as the primary database to store user information, exam data, admission records, and payment details. The database follows a NoSQL structure, offering flexibility in data modelling and scalability.

* + Collections: Users: Stores profile details, authentication credentials, and role-based access.
  + Exams: Contains exam schedules, subjects, and candidate registrations.
  + Applications: Holds admission form data and status updates.
  + Payments: Tracks transaction history and fee receipts.
  + Results: Stores examination outcomes, including grades and remarks.

Data is structured in a document-oriented format, ensuring efficient storage and fast query execution.

**5.7 ER Diagram**

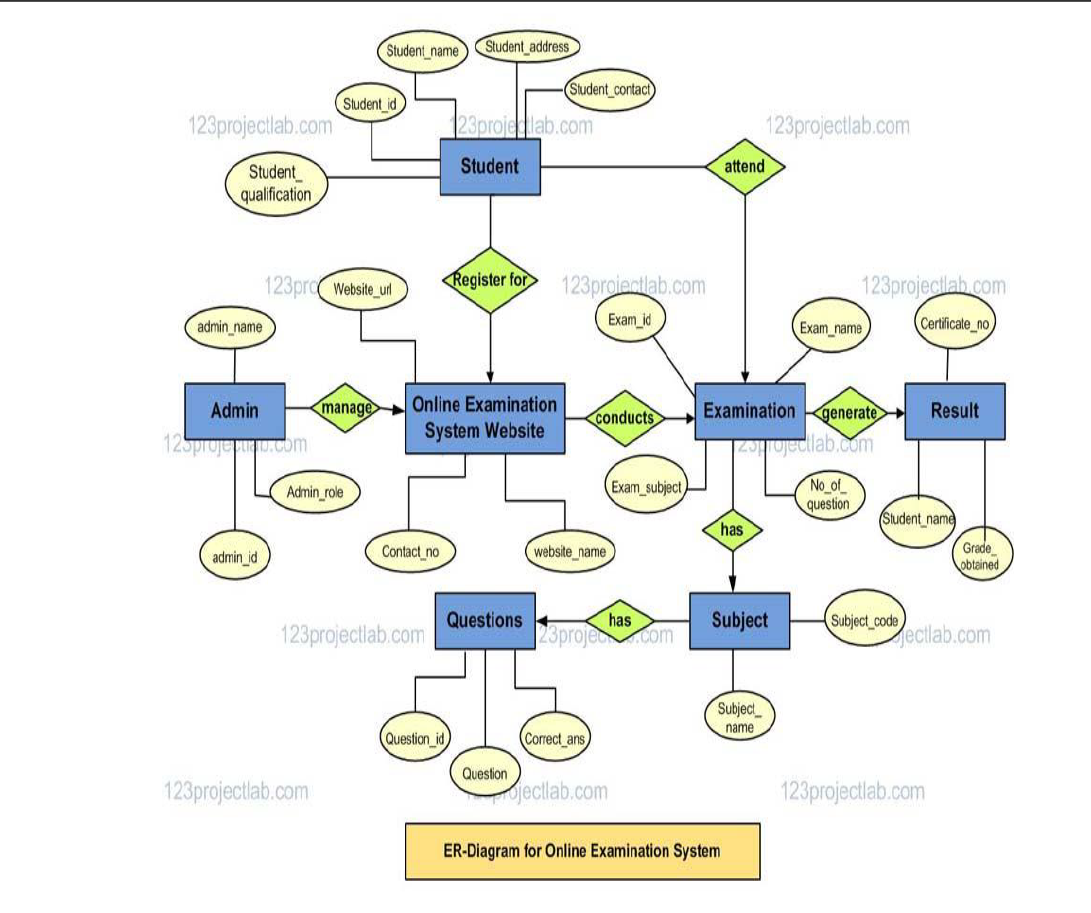
****

Figure: 5.7.1

**5.8 Tables & Relationships**

* **Students**
  + \_id, name, email, passwordHash, role
* **EntranceExams**
  + \_id, studentId, slotTime, questions[], result
* **Questions**
  + \_id, questionText, options[], correctAnswer
* **Certificates**
  + \_id, studentId, fileURL, type, uploadedAt
* **Admissions**
  + \_id, studentId, nriInfo, travelInfo, qualifications
* **Seats**
  + \_id, branch, caste, year, count

Each collection is linked via **ObjectId** references (similar to foreign keys in relational databases).

**5.9 Architecture**

The architecture follows a **Three-Tier Structure**:

1. **Presentation Layer** (Frontend)
   * Built with React.js + TypeScript.
   * Interacts via REST APIs using Axios.
   * Handles routing, state, and user interface logic.
2. **Application Layer** (Backend)
   * Built on Node.js and Express.
   * Implements business logic, middleware, API endpoints.
   * Validates requests and returns responses.
3. **Data Layer** (Database)
   * MongoDB database handles data storage.
   * Collections interact via Mongoose.
   * Ensures scalability and schema flexibility.

**5.10 Flowcharts / UML Diagrams**

**A diagram of a company

AI-generated content may be incorrect.**

Figure 5.10.1

**CHAPTER-6:**

### IMPLEMENTATION

**6. Implementation**

**6.1 Code Snippets**

**a. Student Registration Form(Task-1)**

src/components/RegistrationForm.js

 const handleChange = (e) => {

        const { name, value } = e.target;

        setFormData({ ...formData, [name]: value });

    };

    const handleSubmit = async (e) => {

        e.preventDefault();

        try {

            const response = await axios.post(

                "http://localhost:5000/api/register",

                formData,

                {

                    headers: {

                        "Content-Type": "application/json",

                    },

                }

            );

            alert(response.data.message);

        } catch (error) {

            console.error("Error:", error.response?.data || error.message);

            alert("Error occurred during registration.");

        }

    };

**b. Student password setup and LoginForm(Task-2)**

**frontend/src/components/Password.jsx**export default function PasswordSetup() {

  useEffect(() => {

    if(!localStorage.getItem("userToBeCreated")){

      window.location.href = "/";

    }

    const email = localStorage.getItem("userToBeCreated");

    setEmail(email);

    console.log(email);

  }, []);

  const validateForm = () => {

    if (password === "" || confirmPassword === "") {

      setError("Please fill all the fields!");

      return false;

    }

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    // if(emailRegex.test(email)){

    //   setError("Invalid Email");

    //   console.log(email)

    //   return false;

    // }

    if (password === "" || confirmPassword === "") {

      setError("Please fill all the fields!");

      return false;

    }

    // validate password

    const passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[a-zA-Z]).{8,}$/;

    if (!passwordRegex.test(password)) {

      setError("Password should be minimum 8 characters, at least one upper case, at least one lower case, at least one number and at least one Special characters which includes ~@#$%&\*");

      return false

    }

    return true;

  }

  const handleSubmit = async(e) => {

    e.preventDefault();

    setError("");

    if (!validateForm()) {

      return;

    }

    console.log(password);

    // Call the API to create a new user

    const response = await fetch(`${serverurl}/api/createuser`, {

      method: "POST",

      headers: {

        "Content-Type": "application/json",

      },

      body: JSON.stringify({ email, password }),

    });

    const data = await response.json();

    if (response.status === 201) {

      localStorage.removeItem("userToBeCreated");

      window.location.href = "/";

    } else {

      console.log(data.msg);

      setError("Error in creating user!");

    }

  };

**C. InternationForm(Task-3)**

**frontend/src/components/irForm.jsx**

 const handleChange = (e) => {

        const { name, value } = e.target;

        setFormData({ ...formData, [name]: value });

    };

    const handleSubmit = async (e) => {

        e.preventDefault();

        try {

            const response = await fetch(`${serverurl}/api/addirform`, {

                method: "POST",

                headers: {

                    Authorization: `Bearer ${localStorage.getItem("token")}`,

                    "Content-Type": "application/json",

                },

                body: JSON.stringify(formData),

            });

            if (response.ok) {

                alert("Form submitted successfully!");

                //   setFormData({ ...initialState }); // Clear form

            } else {

                alert("Failed to submit form!");

            }

        } catch (error) {

            console.error("Error:", error);

        }

    };

**d. Certificates Upload(Task-4)**

    useEffect(() => {

        const getuser = async () => {

            try {

                const response = await fetch(`${serverurl}/api/getuser`, {

                    headers: {

                        Authorization: `Bearer ${localStorage.getItem(

                            "token"

                        )}`,

                    },

                });

                if (!response.ok) {

                    throw new Error("Failed to fetch user");

                }

                const data = await response.json();

                console.log(data);

                setStudentDetails({

                    name:

                        data.firstName + " " + data.lastName || "Unknown User",

                    branch: data.specialization,

                    phone: data.mobile,

                    programme: data.program,

                });

            } catch (error) {

                console.error("Error fetching user:", error);

            }

        };

        getuser();

    }, []);

    // Helper function to validate file type and size

    const validateFile = (file, type) => {

        if (type === "Photo" || type === "Signature") {

            if (!file.type.match(/image\/(jpeg|jpg|png)/)) {

                alert(

                    "Only JPG/PNG images are allowed for Photo and Signature"

                );

                return false;

            }

            if (file.size > 1048576) {

                alert("File size exceeds 1MB limit for Photo/Signature");

                return false;

            }

        } else {

            if (file.type !== "application/pdf") {

                alert("Only PDF files are allowed for certificates");

                return false;

            }

        }

        return true;

    };

**e. Question Bank Upload (Task-5)**

**frontend/src/components/admin/PreviewQuestionBank.jsx**

  useEffect(() => {

    axios

      .get(`${serverurl}/api/admin/question-banks/${id}`, {

        headers: {

          Authorization: `Bearer ${localStorage.getItem("token")}`,

        },

      })

      .then((response) => {

        setQuestionBank(response.data);

        setLoading(false);

      })

      .catch(() => {

        setError("Failed to fetch question bank");

        setLoading(false);

      });

  }, [id]);

   useEffect(() => {

    if (questionBank) {

      const fetchImages = async () => {

        let images = {};

        for (let key in currentQuestion) {

          if (typeof currentQuestion[key] === "object" && currentQuestion[key]?.ref) {

            const imageUrl = await handleViewImage(currentQuestion[key]);

            images[key] = imageUrl;

          }

        }

        setImageUrl(images);

        console.log(images);

      };

      fetchImages();

    }

  }, [questionBank, currentQuestionIndex]);

  if (loading) return <div className="text-center mt-4">Loading...</div>;

  if (error) return <div className="alert alert-danger">{error}</div>;

  if (!questionBank || questionBank.questions.length === 0)

    return <div className="alert alert-warning text-center">No Questions Found</div>;

  const questions = questionBank.questions;

  const currentQuestion = questions[currentQuestionIndex];

  const getDisplayText =  (field) => {

    if(field === "manual\_image"){

      return "manual\_image";

    }

    if(typeof field === "object" && field !== null ){

    }

    return typeof field === "object" && field !== null ? field.originalName : field;

  };

  const handleNext = () => {

    if (currentQuestionIndex < questions.length - 1) {

      setCurrentQuestionIndex((prev) => prev + 1);

    }

  };

  const handleBack = () => {

    if (currentQuestionIndex > 0) {

      setCurrentQuestionIndex((prev) => prev - 1);

    }

  };

 const handleViewImage = async (value) => {

    if (!value?.ref) return;

    try {

      const response = await fetch(`${serverurl}/api/admin/question-banks/image/${value.ref}`, {

        headers: { Authorization: `Bearer ${localStorage.getItem("token")}` },

      });

      if (response.ok) {

        const data = await response.blob();

        return URL.createObjectURL(data);

      }

    } catch (error) {

      console.error("View error:", error);

    }

  };

  const renderImage = (field) => {

    console.log("field",typeof currentQuestion[field] === "object" && currentQuestion[field] !== null );

    if(typeof currentQuestion[field] === "object" && currentQuestion[field] !== null ){

    return imageUrl[field] ? <img src={imageUrl[field]} alt="Uploaded" fluid className="mt-2" style={{width:"100vh",height:"200px"}} /> : null;

    }

  };

  const handleFileChange = async (event, field) => {

    const file = event.target.files[0];

    if (!file) return;

    if (!file.type.startsWith("image/")) {

      alert("Only image files are allowed.");

      return;

    }

    const formData = new FormData();

    formData.append("file", file);

    formData.append("field", field);

    formData.append("questionId", currentQuestion.qno);

    try {

      await axios.post(`${serverurl}/api/admin/question-banks/upload-image/${currentQuestion.\_id}/${field}`, formData, {

        headers: {

          Authorization: `Bearer ${localStorage.getItem("token")}`,

          "Content-Type": "multipart/form-data",

        },

      });

      alert("File uploaded successfully!");

      window.location.reload();

    } catch (error) {

      alert("Failed to upload file");

    }

  };

  const handleDeleteImage = async (field) => {

    try {

      await axios.delete(`${serverurl}/api/admin/question-banks/delete-image/${currentQuestion.\_id}/${field}`, {

        data: { field },

        headers: {

          Authorization: `Bearer ${localStorage.getItem("token")}`,

        },

      });

      alert("Image deleted successfully!");

      window.location.reload();

    } catch (error) {

      alert("Failed to delete image");

    }

  };

**f. Online Quiz exam (Task-6)**

**frontend/src/components/QuizExam.jsx**

export default function QuizExam() {

    const [questions, setQuestions] = useState([]);

    const [examType, setExamType] = useState("");

    const [subjects, setSubjects] = useState([]);

    const [currentSubject, setCurrentSubject] = useState("physics");

    const [currentIndex, setCurrentIndex] = useState(0);

    const [answers, setAnswers] = useState({});

    const [status, setStatus] = useState({});

    const navigate = useNavigate();

    useEffect(() => {

        const fetchQuestions = async () => {

            try {

                const response = await fetch(

                    `${serverurl}/api/user/questions`,

                    {

                        headers: {

                            Authorization: `Bearer ${localStorage.getItem(

                                "token"

                            )}`,

                        },

                    }

                );

                if (!response.ok) throw new Error("Failed to fetch questions");

                const data = await response.json();

                setQuestions(data);

                setExamType(data[0]?.examtype || ""); // Set exam type

                setSubjects([...new Set(data.map((q) => q.quesubject))]); // Extract unique subjects

                setStatus(

                    Object.fromEntries(data.map((q) => [q.\_id, "notVisited"]))

                );

            } catch (error) {

                console.error("Error fetching questions:", error);

            }

        };

        fetchQuestions();

    }, []);

    const filteredQuestions = questions.filter(

        (q) => q.quesubject === currentSubject

    );

    const currentQuestion = filteredQuestions[currentIndex];

    // Calculate total time left for the selected subject

    const totalTimeLeft = filteredQuestions.reduce(

        (sum, q) => sum + q.qtimesec,

        0

    );

    const handleAnswerChange = (qId, option) => {

        setAnswers({ ...answers, [qId]: option });

        setStatus({ ...status, [qId]: "answered" });

    };

    const handleNavigation = (index) => setCurrentIndex(index);

    const handleStatusUpdate = (qId, newStatus) => {

        setStatus({ ...status, [qId]: newStatus });

    };

    // \*\*🚀 Calculate Scores Before Submitting\*\*

    const totalQuestions = filteredQuestions.length;

    const answered = Object.values(answers).filter(Boolean).length;

    const notAnswered = totalQuestions - answered;

    // Calculate obtained marks by checking answers

    const obtainedMarks = filteredQuestions.reduce((sum, q) => {

        return answers[q.\_id] === q.answer ? sum + q.qmarks : sum;

    }, 0);

    const handleSubmit = () => {

        // Use all questions instead of only the filtered ones

        const allQuestions = questions;

        const totalQuestions = allQuestions.length;

        const answered = Object.values(answers).filter(Boolean).length;

        const notAnswered = totalQuestions - answered;

        // Calculate obtained marks correctly for all subjects

        const obtainedMarks = allQuestions.reduce((sum, q) => {

            return answers[q.\_id] === q.answer ? sum + q.qmarks : sum;

        }, 0);

        navigate("/result", {

            state: {

                answers,

                questions: allQuestions, // Send all questions

                totalQuestions,

                answered,

                notAnswered,

                obtainedMarks,

            },

        });

    };

**h. Admission Module (Module-2) Student Role & DEO Role.**

1) Student Role:

Admission letter for B.Tech, M.Tech, PHD, all..

code:

**/frontend/src/pages/AdmissionLetter.tsx:**

    return (

            <Container className="text-center my-5">

                <Spinner animation="border" variant="success" />

            </Container>

        );

    }

    return (

        <Container className="my-5 mx-auto p-4 bg-light rounded shadow">

            <div className="text-center mb-3">

                <img

                    src="/vignan/images/vignanlogo.svg"

                    alt="Vignan's Logo"

                    style={{ height: "80px" }}

                />

            </div>

            <Card className="p-5 border-black mb-3 gap-2 rounded shadow">

                <h5 className="mt-2 text-white text-center my-4 p-2 bg-success rounded">

                    Provisional Seat Allotment Order

                </h5>

                <Card body className="border-light mb-3 p-4">

                    <Row>

                        {/\* Left column for details \*/}

                        <Col md={8}>

                            {details.map((item, index) => (

                                <div

                                    className="d-flex"

                                    style={{ marginLeft: "60px" }}

                                    key={index}

                                >

                                    <div

                                        className="p-1"

                                        style={{ width: "220px" }}

                                    >

                                        <strong>{item.label}</strong>

                                    </div>

                                    <div className="px-2">:</div>

                                    <div>{item.value}</div>

                                </div>

                            ))}

                        </Col>

                        {/\* Right column for profile photo \*/}

                        <Col>

                            <div

                                className="border rounded"

                                style={{ width: "150px", height: "180px" }}

                            >

                                <img

                                    src={"/profile\_photo.jpg"}

                                    alt="Profile"

                                    style={{

                                        width: "100%",

                                        height: "100%",

                                        objectFit: "cover",

                                    }}

                                />

                            </div>

                        </Col>

                    </Row>

                </Card>

                <div

                    className="text-end text-muted"

                    style={{ marginRight: "30px" }}

                >

                    <strong>Dean Admissions</strong>

                </div>

                <div className="text-start my-3 p-4">

                    <ol>

                        <li>

                            This is only a provisional admission offer and all

                            original certificates should be submitted to permit

                            to attend classwork.

                        </li>

                        <li>

                            Ratification and allocation of seat in the requested

                            branch will be confirmed after payment of first

                            semester tuition fee in 30 days or 30th June 2025,

                            whichever is earlier.

                        </li>

                        <li>

                            If not paid by due date, the provisional allocated

                            seat deemed to be vacant and will be added to pool

                            of vacant seat in the respective branch.

                        </li>

                        <li>

                            Prior to closing of admissions and after

                            commencement of classwork, tuition fee is refundable

                            with proportionate deduction of monthly fee.

                        </li>

                        <li>

                            To continue the scholarship from 2nd year 1st

                            semester student should get 7.0 CGPA from first year

                            and should not have any backlogs. Subsequently, the

                            student should get 7.0 SGPA in every semester and

                            without any backlogs for continuation of

                            scholarship.

                        </li>

                        <li>

                            For cancellation please send an email to

                            admissions@vignan.ac.in with following attachments:

                            <ul>

                                <li>

                                    Front page photo of bank account book (Clear

                                    Picture).

                                </li>

                                <li>Seat allotment letter copy.</li>

                                <li>Challan copy (Fee payment receipt).</li>

                            </ul>

                        </li>

                    </ol>

                </div>

                <Row className="my-4">

                    <Col className="text-start">

                        <strong>Signature of the parent</strong>

                    </Col>

                    <Col className="text-end">

                        <strong>Signature of candidate</strong>

                    </Col>

                </Row>

            </Card>

            <div className="text-center my-3">

                <Button

                    variant="success"

                    className="mx-2"

                    onClick={() => window.print()}

                >

                    Print

                </Button>

                <Button variant="secondary" className="mx-2">

                    Back

                </Button>

            </div>

        </Container>

    );

};

export default AdmissionLetter;

**/backend/src//services/admissionService.js:**  
const admissionDAO = require('../dao/admissionDAO');

const admissionService = {

  async getAdmissionLetter(admissionNumber) {

    const studentProfile = await admissionDAO.getStudentProfile(admissionNumber);

    if (!studentProfile) {

      throw new Error('Student not found');

    }

    const admissionForm = await admissionDAO.getAdmissionForm(admissionNumber);

    const programme = admissionForm?.prgmm\_id ? await admissionDAO.getProgramme(admissionForm.prgmm\_id) : null;

    const branch = admissionForm?.prgmm\_id ? await admissionDAO.getBranch(admissionForm.prgmm\_id) : null;

    const seatAllocation = await admissionDAO.getSeatAllocation(admissionNumber);

    const educationProfile = await admissionDAO.getEducationProfile(admissionNumber);

    const feeDetails = await admissionDAO.getFeeDetails(admissionNumber);

    const prescribedTotalFee = (feeDetails?.adm\_fee || 0) + (feeDetails?.tuition\_fee || 0);

    const concessionDetails = await admissionDAO.getConcessionDetails(admissionNumber);

    const feePerYear = (feeDetails?.tuition\_fee || 0) - (feeDetails?.discount\_fee || 0);

    return {

      admission\_number: admissionNumber,

      student\_name: studentProfile.stduent\_name || "",

      father\_name: studentProfile.father\_name || "",

      course\_name: programme?.programme\_short\_name || "",

      branch\_name: branch?.branch\_short\_name || "",

      vu\_id: seatAllocation?.vu\_id || "",

      plus2\_hall\_ticket: educationProfile?.pluse\_2\_reg\_no || "",

      plus2\_marks: educationProfile?.pluse\_2\_earend\_marks\_cgpa || "",

      plus2\_percentage: educationProfile?.pluse\_2\_percentage || "",

      seat\_allotment\_type: "",

      marks: admissionForm?.exam\_marks ||"",

      parent\_no: studentProfile.father\_mobile\_number !== 0 ? studentProfile.father\_mobile\_number : studentProfile.mother\_mobile\_number || "",

      date\_of\_birth: studentProfile.dob ? studentProfile.dob.split(' ')[0] : "" || "",

      joining\_date: "01-08-2025",

      seat\_allotment\_category: feeDetails?.category\_id || "",

      admission\_fee: feeDetails?.adm\_fee || 0,

      prescribed\_tuition\_fee: feeDetails?.tuition\_fee || 0,

      prescribed\_total\_fee: prescribedTotalFee,

      scholarship\_per\_year: feeDetails?.discount\_fee || 0,

      concession\_type: concessionDetails?.concession\_name || "",

      fee\_per\_year: feePerYear

    };

  }

};

module.exports = admissionService;

**2) DEO Role:**

**i) Slot Allotment View**

**/frontend/src/pages/SlotAllotmentView.tsx:**

  return (

    <div className="container mt-5">

      <h6 className="text-center text-primary mb-3">

        Admission Number: <a href="#">{admissionNumber || 'N/A'}</a>

      </h6>

      <h3 className="text-center fw-bold mb-4">

        Student Seat Allotment View (DEO Login)

      </h3>

      <div className="d-flex justify-content-center mb-4">

        <label className="me-2 mt-1">Enter Admission Number:</label>

        <input

          type="text"

          className="form-control w-auto me-2"

          value={admissionNumber}

          onChange={(e) => setAdmissionNumber(e.target.value)}

          onKeyDown={handleKeyDown}

          placeholder="e.g. 20250000040"

        />

        <button className="btn btn-success" onClick={fetchData}>

          Search

        </button>

      </div>

      {studentData && (

        <>

          <table className="table table-bordered">

            <tbody>

              <tr><th>Adm No</th><td>{studentData.admission\_number}</td></tr>

              <tr><th>Student Name</th><td>{studentData.student\_name}</td></tr>

              <tr><th>VU ID</th><td>{studentData.vu\_id}</td></tr>

              <tr><th>Program</th><td>{studentData.program}</td></tr>

              <tr><th>Branch</th><td>{studentData.branch}</td></tr>

              <tr><th>Fees Category</th><td>{studentData.fees\_category}</td></tr>

              <tr><th>Campus</th><td>{studentData.campus}</td></tr>

              <tr><th>User Concession names</th><td>{studentData.user\_concession\_names}</td></tr>

              <tr><th>User Concession percentage %</th><td>{studentData.user\_concession\_percentage}</td></tr>

              <tr><th>Academics Scholarship Name</th><td>{studentData.academics\_scholarship\_name}</td></tr>

              <tr><th>Academics Scholarship percentage %</th><td>{studentData.academics\_scholarship\_percentage}</td></tr>

            </tbody>

          </table>

          <table className="table table-bordered">

            <tbody>

              <tr>

                <th>Total Scholarship Names</th>

                <td>{studentData.total\_scholarship\_names}</td>

              </tr>

              <tr>

                <th>Total Scholarship percentage %</th>

                <td>{studentData.total\_scholarship\_percentage}</td>

              </tr>

            </tbody>

          </table>

        </>

      )}

    </div>

  );

};

export default SlotAllotmentView;

**/backend/src/services/slotAllotmentService.js:**  
const slotAllotmentDAO = require('../dao/slotAllotmentDAO');

const slotAllotmentService = {

  async getStudentDetails(admissionNumber) {

    const personalProfile = await slotAllotmentDAO.getPersonalProfile(admissionNumber);

    if (!personalProfile) {

      throw new Error('Student not found');

    }

    const seatLog = await slotAllotmentDAO.getSeatLog(admissionNumber);

    const admissionForm = await slotAllotmentDAO.getAdmissionForm(admissionNumber);

    const feeDetails = await slotAllotmentDAO.getFeeDetails(admissionNumber);

    const concessions = await slotAllotmentDAO.getConcessions(admissionNumber);

    const programme = admissionForm?.prgmm\_id ? await slotAllotmentDAO.getProgramme(admissionForm?.prgmm\_id) : null;

    const branch = admissionForm?.prgmm\_id ? await slotAllotmentDAO.getBranch(admissionForm?.prgmm\_id) : null;

    const campus = admissionForm?.campus\_id ? await slotAllotmentDAO.getCampus(admissionForm?.campus\_id) : null;

    const totalScholarshipNames = [

      concessions?.concession\_names || "",

      concessions?.acad\_schltshp\_names || "",

    ].filter(Boolean).join(" + ");

    const totalScholarshipPercentage =

      (concessions?.tot\_concession\_prntg || 0) +

      (concessions?.acad\_schltshp\_prntg || 0);

    return {

      admission\_number: admissionNumber,

      student\_name: personalProfile?.stduent\_name || "",

      vu\_id: seatLog?.vu\_id || "",

      program: programme?.programme\_short\_name || "",

      branch: branch?.branch\_short\_name || "",

      fees\_category: feeDetails?.category\_id || "",

      campus: campus?.campus\_venue\_code || "",

      user\_concession\_names: concessions?.concession\_names || "",

      user\_concession\_percentage: concessions?.tot\_concession\_prntg || 0,

      academics\_scholarship\_name: concessions?.acad\_schltshp\_names || "",

      academics\_scholarship\_percentage: concessions?.acad\_schltshp\_prntg || 0,

      total\_scholarship\_names: totalScholarshipNames,

      total\_scholarship\_percentage: totalScholarshipPercentage,

    };  } };

module.exports = slotAllotmentService;

**/backend/src//dao/slotAllotmentDAO.js:**

const { Long } = require('mongodb');

const mongoose = require("mongoose");

const dbAdProcess = mongoose.connection.useDb("ad\_process");

const dbAdMaster = mongoose.connection.useDb("ad\_master");

const dbAdFinance = mongoose.connection.useDb("ad\_finance")

const slotAllotmentDAO = {

  getPersonalProfile: (admissionNumber) =>

    dbAdProcess.collection("student\_personal\_profile").findOne({ admission\_number: Long.fromString(admissionNumber) }),

  getSeatLog: (admissionNumber) =>

    dbAdProcess.collection("seats\_allocation\_log").findOne({ admission\_no: Long.fromString(admissionNumber) }),

  getAdmissionForm: (admissionNumber) =>

    dbAdProcess.collection("admission\_form").findOne({ admission\_no: Long.fromString(admissionNumber) }),

  getProgramme: (programmeId) =>

    dbAdMaster.collection("programme\_ms").findOne({ programme\_id: programmeId }),

  getBranch: (programmeId) =>

    dbAdMaster.collection("prog\_branch\_ms").findOne({ pgrm\_programme\_id: programmeId })

  getCampus: (campusId) =>

    dbAdMaster.collection("campus\_ms").findOne({ campus\_id: campusId }),

  getFeeDetails: (admissionNumber) =>

    dbAdFinance.collection("std\_fee\_details").findOne({ admission\_no: Long.fromString(admissionNumber) }),

  getConcessions: (admissionNumber) =>

    dbAdProcess.collection("concession\_details").findOne({ adm\_no: Long.fromString(admissionNumber) }),

};

module.exports = slotAllotmentDAO;

**ii) Concession Modify:**

**/frontend/src/pages/ConcessionModify.tsx:**

  return (

    <div className="container mt-5 p-6 bg-light shadow-md rounded-lg">

      <h1 className="text-center mb-3 text-2xl font-bold mb-4">Concession Modify</h1>

      {/\* <div className="flex items-center mb-4 gap-2">

        <input

          type="text"

          placeholder="Enter Admission Number"

          value={admissionNo}

          onChange={(e) => setAdmissionNo(e.target.value)}

          className="border p-2 rounded"

        />

        <button onClick={fetchConcessions} className="btn btn-primary bg-blue-600 text-white px-4 py-2 rounded">

          Fetch

        </button>

      </div> \*/}

      <form

        onSubmit={(e) => {

          e.preventDefault(); // prevent page reload

          fetchConcessions();

        }}

        className="flex items-center mb-4 gap-2"

      >

        <input

          type="text"

          placeholder="Enter Admission Number"

          value={admissionNo}

          onChange={(e) => setAdmissionNo(e.target.value)}

          className="border p-2 rounded"

        />

        <button type="submit" className="btn btn-primary bg-blue-600 text-white px-4 py-2 rounded">

          Fetch

        </button>

      </form>

      {loading && <p>Loading...</p>}

      {message && <p className="text-red-500 mb-4">{message}</p>}

      {concessions.length > 0 && (

        <div>

          <table className="min-w-full border border-gray-300">

            <thead>

              <tr>

                <th className="border p-2">Concession Type</th>

                <th className="border p-2">Apply</th>

                <th className="border p-2">Concession Percentage</th>

              </tr>

            </thead>

            <tbody>

              {concessions.map((concession, index) => (

                <tr key={concession.concession\_subid}>

                  <td className="border p-2">{concession.concession\_name}</td>

                  <td className="border p-2 text-center">

                    <input

                      type="checkbox"

                      checked={concession.applied}

                      onChange={() => handleCheckboxChange(index)}

                    />

                  </td>

                  <td className="border p-2">

                    <input

                      type="number"

                      value={concession.concession\_prntg}

                      onChange={(e) => handlePercentageChange(index, Number(e.target.value))}

                      className="border p-1 rounded w-full"

                    />

                  </td>

                </tr>

              ))}

            </tbody>

          </table

          <button

            onClick={submitConcessions}

            className="mt-4 bg-green-600 text-white px-6 py-2 rounded btn btn-success"

          >

            Update

          </button>

        </div>

      )}

    </div>

  );

}

**/backend/src/services/concessionService.js:**

const SeatAllocation = require("../models/seatAllocation");

const ConcessionType = require("../models/concessionTypes");

const ConcessionDetail = require("../models/concessionDetails");

const mongoose = require("../dao/MongooseConfig").mongoose;

exports.fetchConcessionDetails = async (admissionNo) => {

    const seat = await SeatAllocation.findOne({ admission\_no: admissionNo });

    if (!seat) throw new Error("Seat allocation not found.");

    const concessions = await ConcessionType.find({ prog\_id: seat.prgmm\_id });

    const appliedConcessions = await ConcessionDetail.find({

        adm\_no: admissionNo,

    });

    const appliedMap = new Map();

    appliedConcessions.forEach((concession) => {

        appliedMap.set(concession.concession\_subid, concession);

    });

    return concessions.map((concession) => ({

        concession\_subid: concession.concession\_subid,

        concession\_name: concession.consess\_desc,

        applied: appliedMap.has(concession.concession\_subid),

        concession\_prntg:

            appliedMap.get(concession.concession\_subid)?.concession\_prntg || 0,

    }));

};

exports.updateConcessionDetails = async (admissionNo, updatedConcessions) => {

    const session = await mongoose.startSession();

    session.startTransaction();

    try {

        const existingConcessions = await ConcessionDetail.find({

            adm\_no: admissionNo,

        }).session(session);

        const existingMap = new Map();

        existingConcessions.forEach((c) =>

            existingMap.set(c.concession\_subid, c)

        )

        // Build concession\_names string

        const selectedConcessionNames = updatedConcessions

            .filter((c) => c.applied)

            .map((c) => c.concession\_name)

            .join(" + ");

        const totalConcessionPrntg = updatedConcessions

            .filter((c) => c.applied)

            .reduce((sum, c) => sum + c.concession\_prntg, 0);

        for (const concession of updatedConcessions) {

            const existing = existingMap.get(concession.concession\_subid);

            if (concession.applied) {

                if (existing) {

                    existing.concession\_prntg = concession.concession\_prntg;

                    existing.concession\_names = selectedConcessionNames;

                    existing.tot\_concession\_prntg = totalConcessionPrntg;

                    await existing.save({ session });

                } else {

                    const newConcession = new ConcessionDetail({

                        concession\_id: "C2500052", // find out how to change dynamically

                        adm\_no: admissionNo,

                        concession\_subid: concession.concession\_subid, // make this dynamic , take concession\_subid from concession\_types collection

                        concession\_name: concession.concession\_name,

                        concession\_prntg: concession.concession\_prntg,

                        adm\_year: 2025, // make this dynamic , take admission\_year from admission\_form collection

                        campus\_id: 1, // make this dynamic , take admission\_year from admission\_form collection

                        log\_userid: 1129, // make this dynamic from, take admission\_year from admission\_form collection

                        log\_ipaddress: "127.0.0.1", // current IP address

                        log\_timestamp: new Date(), // current timestamp

                        tot\_concession\_prntg: totalConcessionPrntg,

                        concession\_names: selectedConcessionNames,

                        acad\_schltshp\_id: 3, // make this dynamic

                        acad\_schltshp\_names: "v\_sat\_rank\_based", // make this dynamic

                        acad\_schltshp\_prntg: 25, // make this dynamic

                    });

                    await newConcession.save({ session });

                }

            } else {

                if (existing) {

                    await existing.deleteOne({ session });

                }           }

        }

        await ConcessionDetail.updateMany(

            { adm\_no: admissionNo.toString() },

            { $set: { concession\_names: selectedConcessionNames } },

            { session }

        );

        await session.commitTransaction();

    } catch (err) {

        await session.abortTransaction();

        console.error("Transaction error:", err);

        throw err;

    } finally {

        session.endSession();

    }

};

**6.2 Key algorithms/functions.**

Our MERN stack project implements a range of essential algorithms and functions to handle core operations across various modules efficiently. These functions ensure the application remains responsive, secure, and user-friendly for students and administrators.

**Form validation** is a critical feature present on both frontend and backend. React-based forms use libraries like Formik and Yup to validate fields such as name, registration number, and phone number, while backend validation with Mongoose ensures data integrity during API interactions.

**Authentication** is handled via password hashing using bcryptjs, and login sessions are managed with **JWT tokens**, enabling secure route access and user identity verification.

The **certificate upload module** employs Multer for file handling, supporting upload, update, and deletion operations while storing metadata in MongoDB. Restrictions are placed on file size and format to ensure safety and performance.

The **quiz module** includes algorithms for fetching questions from the database, capturing user responses, and calculating results based on correct answers. Upon completion, a score report is generated and stored.

The **seat allotment system** generates Provisional Seat Allotment Orders dynamically by processing student details, course selection, and applicable scholarships. It outputs structured admission documents and allows real-time updates through the “Concession Modify” feature.

Data filtering and search functionality help DEOs and admins efficiently find student records or certificates using MongoDB query filters with regular expressions.

A batch-wise sorting algorithm processes bulk Excel uploads (6,000 students), categorizing them by stream and assigning batches logically.

Lastly, **JWT-based middleware** secures protected routes, verifying token validity and restricting unauthorized access.

Together, these key functions form the operational core of the system, enabling a robust, secure, and interactive student management platform built with modern web technologies.

**CHAPTER 7:**

### TESTING

**7. TESTING**

**7.1 Test Cases**

To ensure the functionality and reliability of the academic portal, each module and feature was rigorously tested using well-defined test cases. Each test case included clearly stated input conditions, expected outputs, and actual outcomes to identify discrepancies and verify accuracy. For example:

* In the Student Registration Form, test cases focused on:
  + **Required fields:** Ensuring fields like name, email, registration number, and phone number could not be left blank.
  + **Input validation:** Testing for invalid formats (e.g., alphabets in phone number, improper email syntax).
  + **Database integrity:** Verifying that the entered data was correctly stored in MongoDB without duplication.
* In the Quiz Module, test cases covered:
  + **Question rendering**: Ensuring multiple-choice questions loaded dynamically from the backend.
  + **Answer selection tracking:** Validating that selected answers were correctly registered per question.
  + **Result computation:** Checking whether the score matched the actual number of correct answers.
  + **Navigation flow:** Ensuring the flow from instructions to quiz to results page worked without interruption.

**7.2 Unit, Integration, and System Testing**

To verify the robustness and modularity of the application, we followed a layered testing approach that included unit, integration, and system testing:

**• Unit Testing:**

Unit testing focused on verifying the correctness of individual components in isolation:

* In the frontend, React components like InputField, FormWrapper, and QuizCard were tested for rendering with correct props and state changes.
* In the backend, individual API endpoints and controller functions were tested using tools like Postman and Jest, ensuring they returned expected status codes and responses under different scenarios (e.g., successful vs failed login).

**• Integration Testing:**

Integration testing ensured the correct interaction between modules:

* For example, the certificate upload feature was tested end-to-end from the file selection in the React frontend, through the Express.js middleware (for file handling), and finally the successful storage in MongoDB GridFS.
* The login module was tested to ensure that once a student logged in, their session was maintained properly and subsequent requests (e.g., accessing the quiz or documents) were authenticated.

**• System Testing:**

This level of testing involved the complete end-to-end workflows:

* The admission process was tested from student registration to data verification, seat allotment generation, and document handling.
* We simulated real-time user actions to evaluate performance, responsiveness, and overall user experience.
* Attention was paid to cross-browser compatibility, mobile responsiveness, and API stability under load conditions.

**7.3 Bug Reports**

Throughout the testing phase, various bugs were encountered and documented systematically. Each bug was categorized, prioritized, and assigned a resolution timeline. A few examples include:

* Bug: Registration form crashed on invalid email input  
  Cause: Improper validation handling in the form schema  
  Fix: Integrated Yup schema validation with proper regex to enforce email format and added inline error messages.
* Bug: Certificate upload failed when uploading image files (e.g., PNG,JPEG)  
  Cause: File type mismatch with backend expectations  
  Fix: Updated the file uploader to restrict uploads to PDF format only and provided error prompts for unsupported file types.
* Bug: Quiz questions would not load under slow network conditions  
  Cause: API timeout without fallback handling  
  Fix: Implemented loading skeletons and retry mechanisms for failed fetch requests.
* Issue: Seat allotment module failed to filter students by department and category  
  Cause: Backend filter logic did not align with MongoDB schema structure  
  Fix: Refactored the controller to use accurate query operators ($and, $in) and validated input filters before processing.
* Issue: Quiz result calculations were incorrect when unanswered questions were left blank  
  Cause: Unselected answers were being interpreted as incorrect  
  Fix: Adjusted the evaluation logic to differentiate between unanswered vs incorrect answers, reflecting more accurate scoring.
* Issue: International student data was mixed with domestic entries on the dashboard  
  Cause: Improper role-based filtering on the backend  
  Fix: Introduced an isInternational flag and updated API routes to filter based on student category.
* Issue: Login sessions expired prematurely  
  Cause: Session timeout misconfiguration  
  Fix: Extended session duration using Express-session settings and added auto-refresh mechanisms.

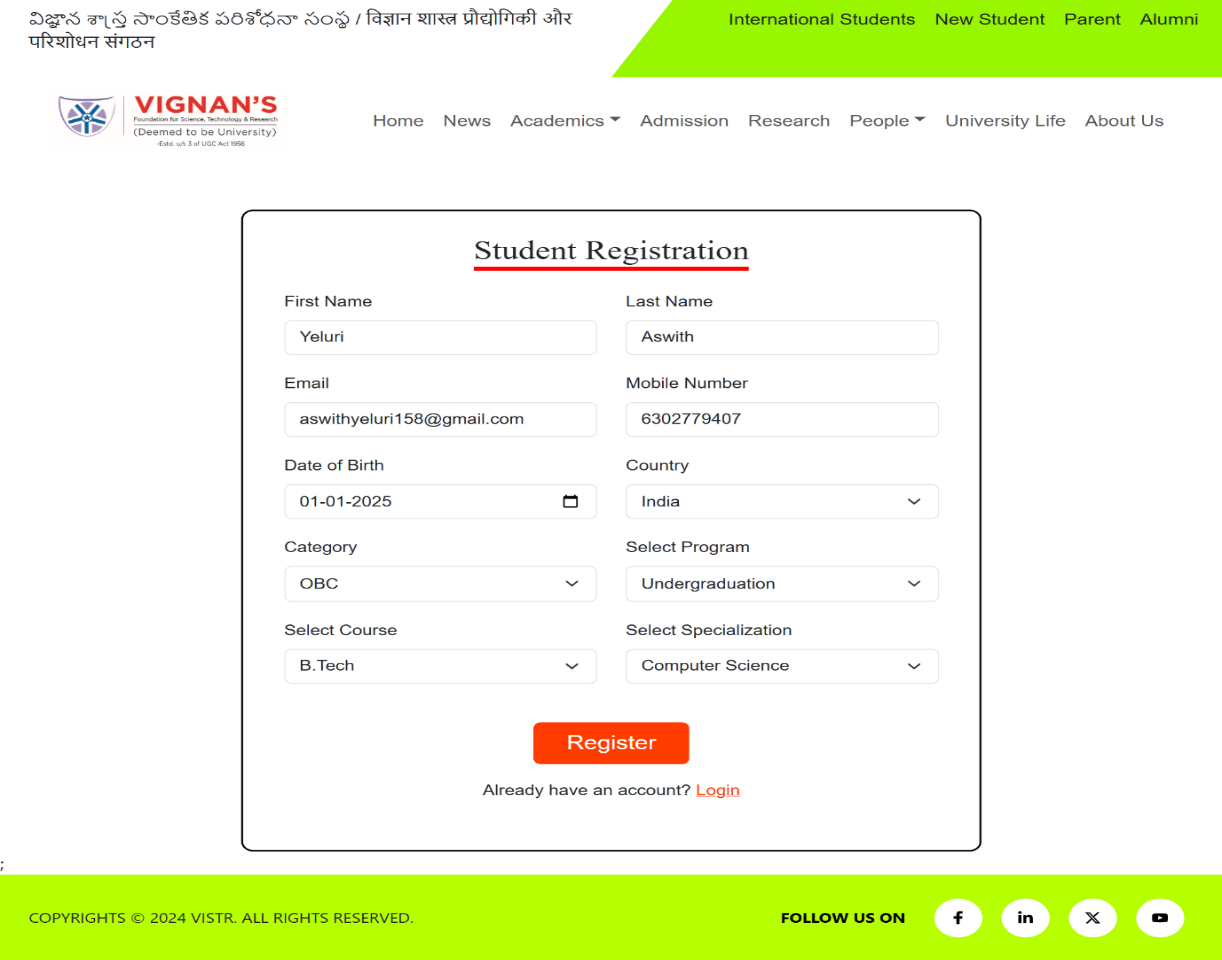
**CHAPTER 8:**

**OUTPUT SCREENS**

**8. OUTPUT SCREENS/RESULTS**

**8.1 Final product screens with explanations.**

**Task-1: Student Registration Form**



**Student Registration Screen:**

This screen allows new students to register by entering their personal, academic, and contact details. Fields include name, email, mobile number, date of birth, and course information. Dropdowns help users select category, course, program, and specialization. Once filled, students click "Register" to proceed.

**Task-2:** **Password Setup and Login**

A screenshot of a login screen

AI-generated content may be incorrect.

### ****Password Setup Screen****

### After registration, students are prompted to set a secure password. They must enter and confirm the password to complete their account setup. This step ensures login credentials are securely created. The "Submit" button finalizes the setup.

A screenshot of a computer

AI-generated content may be incorrect.

### ****Student Login Screen****

### Registered students can log in using their email and password. This screen validates the user’s credentials against stored data. On successful authentication, access is granted to the student portal. The design is minimal and user-friendly.

A screenshot of a computer

AI-generated content may be incorrect.

### ****Welcome Screen****

### Upon successful login, a personalized message welcomes the student by name. It confirms that the login was successful and the student has access to university resources. The page is simple, focusing on user acknowledgment. It enhances the user experience with a friendly tone.

**Task-3:** **International Student Form**

A screenshot of a computer

AI-generated content may be incorrect.

**Personal Details Form**

This screen captures the student’s basic and personal information, such as name, phone number, height, address, and religion.Fields for passport, visa details, and contact information are also present.A “Submit” button is provided at the bottom to save the entered data.

A screenshot of a student registration

AI-generated content may be incorrect.

**Student Registration Details:**

This page displays a summary of the student’s submitted personal details in a clean, tabular format.It confirms data such as phone numbers, countries, passport/visa information, and email.This screen serves as a read-only overview for verification purposes.

A screenshot of a computer

AI-generated content may be incorrect.

**Welcome Page**  
A simple greeting screen welcoming the student to Vignan University.It also includes a prominent link to the International Admission Portal for further actions.The design is minimal, focusing user attention on the next step.

**Task-4:** **Certificate Upload**

A screenshot of a computer

AI-generated content may be incorrect.

**Document Upload Screen**

This interface allows students to upload their photo, signature, and academic certificates.  
It displays uploaded files in a table with options to view, edit, delete, and download each item.A dropdown helps filter file types, and a message confirms successful uploads.

**Task-5:** **Question Bank Upload**

A screenshot of a computer

AI-generated content may be incorrect.

**Admin Dashboard – Question Bank**  
This admin panel allows administrators to manage the question bank.They can view or upload new questions easily using two clearly labeled buttons.The interface is clean, user-friendly, and emphasizes core admin actions.

A screenshot of a computer

AI-generated content may be incorrect.

**Question Bank – Individual Question View**

Displays a single exam question along with relevant metadata like subject, difficulty level, marks, and time.It also includes an embedded image, multiple-choice options, and the correct answer highlighted.Useful for reviewing or editing individual entries in the question bank.

**Task-6:** **Online Quiz Module**

A screenshot of a computer

AI-generated content may be incorrect.

This is a student welcome page for Vignan’s Foundation. It provides quick links for international student registration and online quiz exams. The page includes a navigation bar for accessing different university sections.

A screenshot of a computer

AI-generated content may be incorrect.

This page displays a mock test interface for the VSAT exam. It allows students to answer multiple-choice questions across subjects like Physics, Chemistry, and Maths. The interface includes options to mark questions for review, clear responses, and track the time left. A summary panel shows question status and navigation.

A screenshot of a computer

AI-generated content may be incorrect.

This page shows the **Exam Summary** after completing a mock test. It displays the total number of questions, along with how many were answered and not answered. Users have the option to generate results or return to the home page. The interface is simple and provides a quick overview of performance.

A screenshot of a computer

AI-generated content may be incorrect.

This page displays the **final exam results** after a mock test. It shows the total number of questions, the number of answered and unanswered questions, and the **marks obtained**. A "Go to Home" button allows users to return to the main page. It provides a clear summary of the user's performance.

**Module -1**

**1) Subject wise Question paper:**

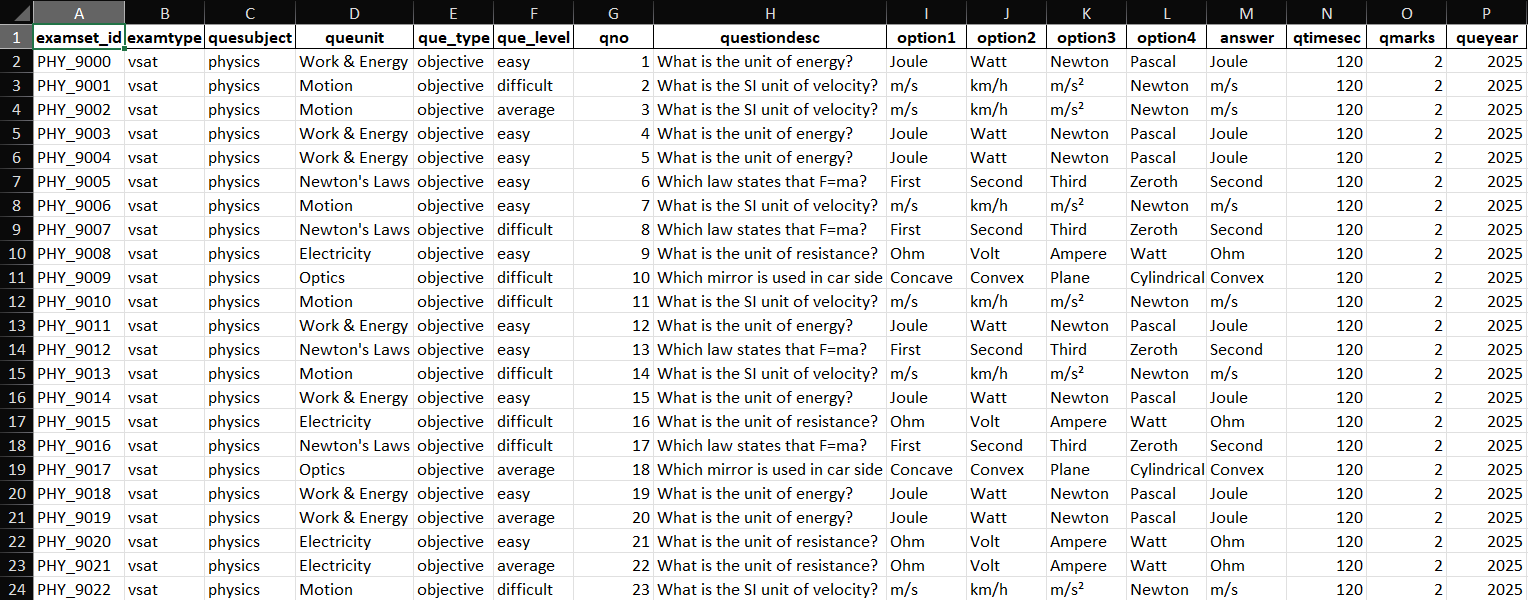
**i) Maths :**

A screen shot of a computer

AI-generated content may be incorrect.

This page displays a spreadsheet containing mock test questions of maths for a VSAT exam. It includes metadata such as exam type, subject, question type, difficulty level, question text, multiple-choice options, correct answers, and associated marks. The table is well-structured to support quiz generation and assessment analysis.

**ii) Physics:**



This page displays a spreadsheet containing mock test questions of physics for a VSAT exam. It includes metadata such as exam type, subject, question type, difficulty level, question text, multiple-choice options, correct answers, and associated marks. The table is well-structured to support quiz generation and assessment analysis.

**iii) Chemistry:**

A screenshot of a computer

AI-generated content may be incorrect.

This page displays a spreadsheet containing mock test questions of chemistry for a VSAT exam. It includes metadata such as exam type, subject, question type, difficulty level, question text, multiple-choice options, correct answers, and associated marks. The table is well-structured to support quiz generation and assessment analysis

**iv)English:**

A screenshot of a computer

AI-generated content may be incorrect.

This page shows a spreadsheet containing multiple-choice English questions for a VSAT exam. The questions cover topics like grammar, poetry, literature, and comprehension, each tagged with a difficulty level. It includes question text, four answer options, the correct answer, time limits, and marks. The dataset is well-organized for exam preparation or digital mock test generation.

**v) Biology:**

A screenshot of a computer

AI-generated content may be incorrect.

This spreadsheet presents a set of objective-type biology questions, also for a VSAT exam. The questions span genetics, microbiology, ecology, and cell biology, categorized by difficulty level. Each row details the question, answer options, correct answer, time per question, and marks assigned. It's designed to support structured testing or quiz system implementations.

**2) 6000 student lists:**

A screenshot of a computer

AI-generated content may be incorrect.

This table displays student records parsed from uploaded Excel files. It enables filtering, searching, and batch-based categorization.

**Module -2:**

A screenshot of a computer

AI-generated content may be incorrect.

This is a provisional seat allotment order from Vignan's University. It contains the details of a student admitted to the B.Tech program in the Bioinformatics branch, including their admission number, marks, category, and fees. The page also outlines important instructions related to fee payment, scholarship, and the required documents for completing the admission process.

A screenshot of a computer

AI-generated content may be incorrect.

This page is a **Provisional Seat Allotment Order** from Vignan's Foundation for Science, Technology & Research. It confirms the allotment of a seat to **Yeluri Bala Siva Durga Aswith** in the **M.Tech (Machine Design)** program under **CATEGORY-B**. The total fee for the first year is ₹2,20,000, with a GATE-based merit scholarship of ₹42,000. The allotment is provisional and subject to the submission of original certificates and tuition fee payment.

A screenshot of a computer

AI-generated content may be incorrect.

This page is a **Student Seat Allotment View (DEO Login)** for **Vempati Ajay**, who has secured admission in **B.Tech (Bioinformatics)** at Vignan’s University, **Valdamudi campus**, under **CATEGORY-A**. The student has received a total scholarship of **40%**, combining **user concession (15%)** and **academic scholarship (25%)** based on VSAT rank. The page details the specific scholarship sources and percentages.

A screenshot of a computer

AI-generated content may be incorrect.

This page is a **Concession Modify** interface for admission number **20250000123** at Vignan’s University. It displays various **concession types** such as *GATE merit*, *Alumni*, *Chairman*, and *Staff reference*, with options to apply them and input their respective **concession percentages**. For this student, a **5% concession** is applied under *Vignan\_student\_Alumni\_std* and **10% under GATE\_merit**, totaling a 15% fee concession.

**CHAPTER 9:**

**CONCLUSION AND FUTURE SCOPE**

**9. CONCLUSION AND FUTURE SCOPE**

**9.1 Summary of achievements.**

This project marks the successful end-to-end development of a **MERN (MongoDB, Express.js, React.js, Node.js) stack-based student management platform**, tailored to streamline academic workflows through modern web technologies. The application was conceptualized and developed with a focus on solving real-world challenges faced by academic institutions, providing a centralized, user-friendly digital solution.

**Foundational Modules & Learning Outcomes:**

The development process began with hands-on implementation of **seven key foundational modules**, which offered practical exposure to full-stack development practices. These modules included:

* **Student Registration and Login**: Form-based registration integrated with real-time validation and secure login using **JWT (JSON Web Tokens)**.
* **Password Setup and Reset Mechanism**: Enabled encrypted password storage using **bcrypt** and robust reset flows.
* **Certificate Uploads**: File upload system implemented using **Multer**, ensuring secure storage of academic documents.
* **Quiz Module**: Timed quizzes with dynamic question rendering, real-time answer tracking, and automatic score calculation.
* **Dynamic Data Displays**: Interactive tables and dashboards using React.js to view student and academic data.

The more detailed description of each module and their features are listed down below:

1.**Student Registration and Profile Management**   
The registration module supports both domestic and international students, allowing for conditional rendering of input fields and dynamic backend validation. Profiles include detailed segments such as travel, academic history, and document upload fields—each backed by strict validation and authentication checks. This ensures accuracy from the point of entry and simplifies the verification chain.  
  
2.**Slot Booking and Online Examination Interface**  
The slot booking feature intelligently manages exam schedules and prevents overbooking by disabling filled slots. The exam interface itself includes subject-wise navigation, countdown timers, real-time answer submission, and automated result processing—creating a seamless digital examination experience that mimics professional testing standards.  
  
3.**Certificate Upload and Document Verification**  
Students can upload certificates like SSC, Inter, UG degree, etc., which are stored in MongoDB as binary objects. The backend restricts file types to PDF and images, while role-based access ensures that only authorized users can upload or validate them. DEOs then verify these documents through a dedicated dashboard, which flags any discrepancies and alerts the Dean for final decision-making.  
  
4.**Seat Application and Allotment Management**  
This module represents one of the most critical aspects of the system. Students select preferred branches and fill out choice forms. Once their documentation is verified, the DEO forwards applications to the Dean, who assigns seats based on merit, availability, and pre-defined filters. The backend generates Provisional Seat Allotment Orders dynamically in PDF format, reflecting concessions, joining dates, and confirmation deadlines.  
  
5.**Admission Letters, Fee Receipts, and Post-Admission Tracking**  
Once a student confirms their seat, the system generates formal admission letters and fee receipts automatically. These documents are downloadable and stored securely with version history. Students can later track their academic progression, pending approvals, or concession modifications through a clean dashboard view.  
  
6.**Role-Specific Dashboards and Workflow Optimization**  
The application’s strength lies in its compartmentalization. Admins can upload question banks in bulk, DEOs manage verification queues, and the Dean approves or rejects applications—all through dedicated, clutter-free dashboards. This modular separation ensures that no workflow overlaps and that accountability is always maintained.  
  
7.**Automation Features and Security Controls**  
Key automation elements include inactivity-based logout, real-time certificate status updates, quiz evaluation, and receipt/admission letter generation. All APIs are protected via JWT-based authentication and CORS, ensuring data integrity and preventing unauthorized access. Validation is done on both frontend and backend layers, reinforcing the security model.  
  
8.**Scalability and Technological Stack**  
Built entirely on the MERN stack with TypeScript, the system supports over 6000 concurrent records without performance degradation. Its integration with tools like Postman, ExcelJS, and MongoDB Compass further enhances data manipulation and admin control. The modular architecture ensures that new modules like hostel allotment, fee management, or alumni tracking can be integrated with minimal disruption.

**Core System Architecture & Advanced Features:**

As the project matured, development transitioned to a structured architecture using **MVC (Model-View-Controller)** principles with **TypeScript**, enhancing code maintainability, clarity, and type safety. Major achievements in this phase include:

* **Question Paper Generator**: A module that generates question papers categorized by subject, stored in a modular schema in MongoDB. This tool allows educators to manage assessments digitally and reduces manual overhead.
* **Bulk Student Data Integration**: Through the integration of **Excel (.xlsx) parsing using libraries like SheetJS**, admin users can upload and manage large volumes of student records with ease.
* **Provisional Seat Allotment System**: A robust system supporting the allocation of academic seats for multiple programs such as **M.Tech**, **Ph.D.**, and more. This includes filters by department, category, and eligibility, allowing fine-grained control during allotment.
* **Admin Panel and User Access Management**: Separate modules allow administrative users to view registrations, monitor seat allotments, and manage backend data in real-time.

**Technology Highlights:**

* **Frontend**: Developed using **React.js** for building modular components and **Bootstrap** for responsive and consistent design. This combination ensured cross-device compatibility and a smooth user experience.
* **Backend**: Powered by **Node.js and Express.js**, allowing scalable RESTful API development. MongoDB provided a flexible NoSQL database structure, suitable for varied data types.
* **Security & Validation**: Integrated **JWT for secure session handling**, **Yup and Express-validator** for frontend/backend validation, and **Multer** for file safety.
* **Testing & Debugging**: Applied a layered testing approach—unit, integration, and system testing—to verify the robustness of individual modules and the entire system workflow. Identified bugs, such as data filtering mismatches and file handling errors, were resolved methodically.
* **Version Control & Collaboration**: The project was managed using **Git and GitHub**, enabling streamlined collaboration, version tracking, and issue management.

**9.2 Possible enhancements.**

**1. Role-Based Access Control (RBAC)**

* **Objective**: Differentiate system privileges based on user roles such as *Admin*, *Student*, *Faculty*, and *Department Head*.
* **Benefits**: Ensures data privacy and functionality segregation. For example, only faculty can upload question papers, and only admins can manage seat allotments.
* **Implementation Approach**: Extend JWT payload with user roles and implement middleware to validate access to specific routes.

**2. Email Notification System**

* **Objective**: Automatically notify users upon key actions—such as successful registration, seat allotment results, and quiz scores.
* **Benefits**: Improves user engagement and provides confirmation receipts.
* **Implementation Tools**: Use **Nodemailer** with an SMTP server or email services like **SendGrid** to dispatch templated emails.

**3. Data Analytics Dashboard**

* **Objective**: Visualize platform data using charts, graphs, and KPIs.
* **Examples**:
  + Bar charts showing registration statistics per department
  + Pie charts for seat distribution across programs
  + Line graphs tracking quiz performance trends
* **Libraries**: Incorporate **Chart.js**, **Recharts**, or **D3.js** within React components.

**4. Cloud Deployment & CI/CD Integration**

* **Objective**: Host the application on cloud platforms such as **AWS**, **Heroku**, or **Vercel** for broader access and automated deployment.
* **Benefits**: Ensures high availability, scalability, and professional-grade hosting.
* **Future Scope**:
  + Use **Docker** for containerized deployment.
  + Integrate **GitHub Actions** or **Jenkins** for automated testing and deployment pipelines.

**5. Mobile Application Integration**

* **Objective**: Develop a cross-platform mobile companion app for students and faculty.
* **Benefits**: Enhances accessibility and allows users to perform tasks (view results, upload documents, receive alerts) on the go.
* **Technology Stack**: Use **React Native** with shared business logic from the web app.

**6. Advanced Authentication Features**

* **Enhancements**:
  + **Multi-Factor Authentication (MFA)** for admins and faculty
  + **OAuth Integration** with institutional login systems or Google accounts

**7. Backup and Recovery System**

* **Objective**: Ensure secure and automated backups of MongoDB data and uploaded files.
* **Implementation Tools**: Use **MongoDB Atlas backups**, or custom cron jobs for database snapshots and file storage redundancy.

**8. Internationalization (i18n) Support**

* **Objective**: Support multiple languages to cater to diverse user groups.
* **Implementation Tools**: Use libraries like **react-i18next** and maintain language-specific translation files.

## CONCLUSION

## The development of this ERP-based entrance exam and admission management system represents a transformative approach to how academic institutions manage their critical workflows. With the growing demand for efficiency, transparency, and scalability in higher education, our solution provides a comprehensive and future-ready digital framework that addresses the shortcomings of legacy systems. Traditional admission processes often suffer from inefficiencies such as manual data entry, delayed result generation, poor tracking of student records, and a lack of integration between different administrative units. These bottlenecks not only slow down operations but also diminish the user experience for both students and institutional staff.

## Our system reimagines these workflows by introducing a role-specific, modular platform powered by the MERN stack—MongoDB, Express.js, React.js, and Node.js—augmented with TypeScript and the MVC architectural pattern for maintainability and scalability. The project spans the entire academic journey, starting from the student registration module that supports both national and international students with adaptive input logic and validation. Students can book examination slots in real time, write online quizzes through a well-structured exam interface, and receive results through automated evaluation systems.

## A key innovation lies in the certificate upload and verification process, where students upload essential academic documents, which are then validated by Data Entry Operators (DEOs) and finalized by the Dean. This structured and secure flow eliminates delays and enables transparent decision-making. Once documents are approved, students can fill out seat preference forms, and the backend system dynamically allocates seats based on predefined criteria and availability. This data then powers the generation of personalized Provisional Seat Allotment Orders, complete with program details, scholarship status, and confirmation instructions.

## The project also includes automatic generation of admission letters and fee receipts, which are downloadable through a secure dashboard. Role-based dashboards for students, admins, DEOs, and deans create a compartmentalized interface tailored to specific functions, reducing clutter and increasing task efficiency. Admins have the ability to upload question banks, modify slot statuses, manage result processing, and update student records—all through a centralized, intuitive UI.

## From a technical standpoint, the application is fortified with modern web development best practices. It employs JWT-based authentication for secure access, file validation for document integrity, and structured API design for reliable communication between the frontend and backend. Real-time feedback, session persistence, and error handling enhance both usability and robustness. The use of MongoDB’s flexible schema structure, along with Excel and Postman integrations, enables the platform to support large-scale data without compromising performance.

## In summary, this ERP solution provides a reliable and intelligent alternative to outdated academic workflows. It automates complex operations, reduces dependency on manual processes, enhances accuracy, and improves decision-making across departments. It not only fulfills the current academic requirements but also offers a solid foundation for future extensions such as AI-powered document scanning, mobile app interfaces, cloud hosting, and institution-wide analytics. By adopting this system, academic institutions are better equipped to scale their operations, improve administrative oversight, and deliver a seamless, transparent, and digitally enhanced experience to every stakeholder.

## CHAPTER 10

## REFERENCES

## 10. REFERENCES

## SoftLoom: ERP Software for Admission Process Efficiency and Success

## • Discusses how ERP software automates admission processes, from online applications to document uploads and data entry.

## • Read more : [Softloom+1Academia ERP+1](https://softloom.com/erp-software-for-the-admission-process/?utm_source=chatgpt.com)

## ReferenceGlobe: Enterprise Resource Planning (ERP)

## • Details the Admission Management Module, handling inquiries, online forms, merit-based sorting, and confirmations.

## • Explore here : [Ellucian+4ReferenceGlobe+4O2B Technologies+4](https://referenceglobe.com/sdc/erp_modules.php?utm_source=chatgpt.com)

## Academia ERP: Simplifying Admission Processes

## • Highlights how automated ERP systems minimize admission processing time and reduce paperwork.

## • Learn more [Academia ERP](https://www.academiaerp.com/blog/admission-process-can-be-carried-out-in-a-simple-and-error-free-manner-using-erp-software/?utm_source=chatgpt.com)

## Ellucian: Understanding ERP Systems in EdTech

## • Explains how ERP systems integrate administrative and academic functions, including admissions and class registrations.

## • Read the article [LinkedIn+6Ellucian+6IJCST+6](https://www.ellucian.com/blog/understanding-erp-systems-edtech?utm_source=chatgpt.com)

## Clobas: How College Management Software Aids Online Admissions

## • Discusses the benefits of ERP software in streamlining admission tasks and improving operational efficiency.

## • Discover more [LearnQoch+4Clobas Private Limited+4Softloom+4](https://www.clobas.com/why-use-erp-software-for-the-admission-process/?utm_source=chatgpt.com)

## Classter: School ERP Systems as Educational Backbones

## • Explores how ERP systems like Classter Core support the complete student lifecycle, including admissions.

## • Find out more[Classter](https://www.classter.com/blog/edtech/school-management-systems/school-erp-systems-the-backbone-of-educational-institutions/?utm_source=chatgpt.com)

## Proctur: Importance of ERP for Educational Institutions

## • Highlights how ERP simplifies extensive admission processes through automation.

## • Read the blog[Focus Softnet+9Proctur+9Academia ERP+9](https://proctur.com/blog/importance-of-educational-erp-software-for-effective-management-of-educational-institutions/?utm_source=chatgpt.com)

## Deskera: ERP for Educational Institutes - The Complete Guide

## • Provides insights into digital payment and admissions modules within ERP systems.

## • Explore the guide[Deskera](https://www.deskera.com/blog/erp-for-educational-institutes/?utm_source=chatgpt.com)

## OpenSIS: Cloud-Based Student Information System

## • An open-source platform offering tools for student data management, including admissions and applicant processing. 80

## • [Learn about OpenSISWikipedia+5Wikipedia+5Clobas Private Limited+5](https://en.wikipedia.org/wiki/OpenSIS)

## Sampoorna: Kerala's School Management System

## • A project by the Government of Kerala to automate school processes, including admissions, using the open-source ERP Fedena.

## • Read more about [Sampoorna](https://en.wikipedia.org/wiki/Sampoorna)