A

PROJECT REPORT

ON

Weekly Assessment Test (WAT)

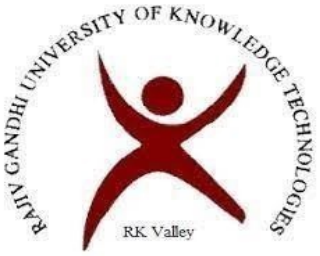
Submitted By

S. PUSHPALATHA (R200049)
C. GEETHA REDDY (R200380)

Under The Guidance of

Dr. MUMMELA MUNI BABU

M. Tech, Ph. D, Assistant Professor
Department of Computer Science and Engineering



Department of Computer Science and Engineering
Rajiv Gandhi University of Knowledge and Technologies
(RGUKT), R.K.Valley, Kadapa, Andhra Pradesh, 516 330
Accredited by 'NAAC' with 'B+' Grade

Rajiv Gandhi University of Knowledge and Technologies (RGUKT),
RK Valley, Kadapa, Andhra Pradesh, 516330

DECLARATION

We hereby declare that the project report entitled “WEEKLY ASSESSMENT TEST” submitted to the Department of COMPUTER SCIENCE AND ENGINEERING in partial fulfilment of requirements for the award of the degree of BACHELOR OF TECHNOLOGY. This project is the result of our own effort and that it has not been submitted to any other University or institution for the award of any degree or diploma other than specified above.

WITH SINCERE REGARDS
S.PUSHPALATHA– R200049
C. GEETHA REDDY– R200380

Rajiv Gandhi University of Knowledge and Technologies (RGUKT),
RK Valley, Kadapa, Andhra Pradesh, 516330

CERTIFICATE

This is to certify that the project work titled “WEEKLY ASSESSMENT TEST” is a bonafied project work submitted by S. PUSHPALATHA – R200049, C. GEETHA REDDY – R200380 in the department of COMPUTER SCIENCE AND ENGINEERING in partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering for the year 2024-2025 carried out the work under the supervision.

GUIDE
Dr. MUMMELA MUNI BABU
Assistant Professor
RGUKT, RK VALLEY

HEAD OF THE DEPARTMENT
Dr. CH. RATNA KUMARI
Assistant Professor
RGUKT, RK VALLEY

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion ofany task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success.

I am extremely grateful to our respected Director, **Dr. A V S S KUMARA SWAMI GUPTA** for fostering an excellent academic climate in our institution.

I also express my sincere gratitude to our respected Head of the Department **Dr. CH. RATNA KUMARI** for her encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project.

I would like to convey thanks to our guide at college **Dr. MUMMELA MUNI BABU** for his guidance, encouragement, co-operation and kindness during the entire duration of the course and academics.

My sincere thanks to all the members who helped me directly and indirectly in the completion of project work. I express my profound gratitude to all our friends and family members for their encouragement.

TABLE OF CONTENTS

S.NO	INDEX	PAGE NO
	Abstract	i
	List of Figures	ii
1.	Introduction	1
	1.1 Problem Statement	2
	1.2 Motivation	2
	1.3 Objective	3
	1.4 Impact	3
2.	Requirement Analysis	4
	Module-1: Funtional Requirements	4
	Module-2: Performance Requirements	5
	Module-3: Technical Requirements	6
	Module-4: Non Funtional Requirements	7
	Module-5: Third Party Packages and Libraries	7
3.	System Design and Architecture	10
	Module-1: System Overview	10
	Module-2: System Achitecture	10
	Module-3: High Level Design	12
	Module-4: Low Level Design	14
4.	User Interface Overview	19
5.	Conclusion and Future Enhancements	26
6.	References	28

ABSTRACT

The project presents **WAT (Weekly Assessment Tests)**, a full-stack web-based platform designed to automate the process of conducting weekly assessments for students and streamlining scoring and evaluation tasks for faculty. This system addresses the challenges of manual test management, delayed feedback, and administrative workload by providing a seamless, digital environment for test creation, participation, evaluation, and result tracking.

The architecture of the platform is based on the MERN stack (MongoDB, Express.js, React.js, Node.js), featuring a responsive and intuitive frontend developed using React and Tailwind CSS, with Redux Toolkit for efficient state management. The backend is powered by Node.js and Express, while MongoDB Atlas serves as the primary database. User authentication and secure access control are implemented using JWT tokens to ensure data privacy and system integrity.

Core functionalities include faculty-driven test creation with customizable question banks, automated test scheduling, instant scoring for objective-type questions, manual evaluation support for subjective responses, and dynamic dashboards for both faculty and students. Students can view upcoming tests, take assessments within defined timeframes, and immediately access their results where applicable. Faculty members benefit from automated result aggregation, easy performance analysis, and efficient management of multiple test cycles.

The platform underwent thorough module testing, validation handling, and user experience (UX) evaluations to ensure its stability, security, and responsiveness. Results indicate that WAT provides a reliable and user-friendly system for managing assessments, significantly reducing manual effort and enhancing the feedback loop between faculty and students.

LIST OF FIGURES

Figure No.	Title	Page No
Figure 2.1	Third Packages Integration	9
Figure 3.1	Three Tier Architecture of System	10
Figure 3.2	Architecture of MERN Stack	12
Figure 3.3	Schemas	14
Figure 3.4	Modules	16
Figure 3.5	Work Flow Diagram	18
Figure 4.1	User Interface Images	19

CHAPTER 1

INTRODUCTION

Assessment is a critical component of the learning process, enabling both students and faculty to measure understanding, monitor academic progress, and identify areas needing improvement. Traditionally, weekly assessments have been conducted manually, requiring significant time and effort from faculty members to prepare, administer, and evaluate tests. This manual approach often results in delays in result processing, increased errors, inconsistencies, and an overall rise in administrative workload.

To address these challenges, Weekly Assessment Testing (WAT) has been developed as a full-stack, web-based platform aimed at automating, streamlining, and enhancing the process of conducting weekly assessments. The platform empowers faculty members to efficiently create, schedule, and manage tests, while providing students with a convenient, secure environment to attempt assessments within defined timeframes, ensuring a smooth and transparent examination process.

Unlike traditional paper-based methods or fragmented digital solutions, WAT offers a centralized, structured, and scalable system for managing assessments. The platform provides the following key features:

- Automated test creation and scheduling.
- Secure user authentication and role-based access.
- Real-time scoring for objective-type questions and simplified evaluation workflows for subjective-type questions.
- Dynamic dashboards for both faculty and students to view test schedules, submissions, and performance reports.
- Immediate result generation wherever applicable.
- Transparent performance analytics and penalty mechanisms for late submissions.

WAT is built using modern web technologies, leveraging the MERN (MongoDB, Express.js, React.js, Node.js) stack, which ensures scalability, flexibility, and strong security standards. The platform has been designed with future extensibility in mind, supporting planned enhancements such as:

- Advanced performance tracking and graphical reporting.
- Integration of real-time notifications and alerts.

• AI-based mechanisms for proctoring and cheating detection.

• Deployment of a mobile application for enhanced accessibility.

By implementing WAT, educational institutions can significantly improve the efficiency, accuracy, transparency, and user experience of their assessment processes, ultimately creating a more engaging and supportive academic environment for students and faculty alike.

1.1 Problem Statement

Weekly Assessment Tests (WATs) at the college face issues like delayed start times, inconsistent test durations, and lack of immediate results, causing frustration for students and inefficiency for teachers. Additionally, cheating is a concern due to easy access to external resources and lack of secure authentication. The solution is to implement a web platform that automates scheduling, enforces strict time limits, secures login, and provides instant feedback, ensuring fairness and efficiency.

1.2 Motivation

The motivation behind Weekly Assessment Testing (WAT) stems from the challenges educational institutions face in managing weekly assessments. Traditional manual processes are often inefficient, prone to errors, and time-consuming for both faculty and students. These issues result in delayed results, inconsistencies in grading, and increased administrative workload.

WAT aims to automate and streamline the entire assessment process, ensuring a seamless and efficient experience. The platform offers real-time result generation, structured workflows, and dynamic performance tracking, addressing the inefficiencies of traditional systems.

Key motivations include:

- **Improving efficiency** through automation of test creation, administration, and evaluation.
- **Ensuring accuracy** with automated grading and real-time results.
- **Providing transparency** via real-time dashboards for students and faculty.
- **Reducing faculty workload**, allowing more focus on teaching.
- **Enabling scalability** to accommodate large student bases and future enhancements.

Inspired by digital solutions like Google Classroom and online testing platforms, WAT is uniquely designed to handle **weekly assessments** with features such as automated scoring, subject-specific question management, and detailed performance analytics.

By adopting WAT, institutions can create a more **efficient, transparent, and performance-driven academic environment**.

1.3 Objective

The primary objective of this project is to develop a scalable, secure, and user-friendly web platform that automates and streamlines the process of conducting weekly assessments. The platform aims to empower faculty members to efficiently create, schedule, and manage assessments, while providing students with a seamless and secure environment to take tests.

Key objectives include:

- Enabling faculty to easily create, schedule, and grade assessments.
- Automating the test lifecycle, including real-time scoring for objective questions and simplified evaluation for subjective questions.
- Ensuring secure user authentication and role-based access through JWT.

1.4 Impact

The potential impact of **WAT** lies in its ability to transform the assessment process in educational institutions. By automating manual tasks, the platform not only reduces administrative workload but also ensures real-time result processing, increased accuracy, and enhanced transparency.

Key impacts include:

- Improving efficiency in assessment creation, administration, and result processing.
- Providing students with timely feedback, fostering a more engaging and supportive learning environment.
- Introducing real-time scoring and performance analytics to track progress.

WAT aims to elevate the quality and experience of assessments, ultimately benefiting both faculty and students while supporting academic growth and performance.

CHAPTER 2

REQUIREMENT ANALYSIS

Module 1: Functional Requirements

User Authentication:

- **Registration and Login:** Users (faculty and students) should be able to register and log in to the platform using their email address and password.
- **Session Management:** Token-based authentication (JWT) will be used for maintaining secure sessions across the platform.

Test Creation and Scheduling:

- **Test Creation:** Faculty members should be able to create and design tests, including adding questions (objective and subjective types), defining test duration, and setting deadlines.
- **Scheduling:** Faculty members should have the ability to schedule tests on specific dates and times, and send notifications to students regarding upcoming tests.
- **Test Availability:** Tests should only be accessible to students during the scheduled time window.

Test Attempt and Submission:

- **Test Taking:** Students should be able to take the test within the given time frame.
- **Question Types:** The platform should support both objective (multiple choice, true/false) and subjective (short-answer/essay) questions.
- **Timer:** A countdown timer will be displayed to students while taking the test to ensure time management.
- **Auto-Submission:** If the student does not submit the test before the timer runs out, the test should be automatically submitted.

Real-Time Scoring:

- **Objective Question Scoring:** Scores for objective questions should be calculated in real-time, immediately after the student submits the test.
- **Subjective Question Evaluation:** Subjective questions will be manually graded by faculty members, and results should be visible to students once graded.

Result Generation and Feedback:

- **Instant Results for Objective Questions:** Once the student submits the test, results for objective questions should be instantly calculated and displayed.
- **Manual Feedback for Subjective Questions:** Faculty members should be able to provide personalized feedback for subjective responses.
- **Result Analytics:** The system should provide detailed performance analytics for both faculty and students, including overall scores, strengths, weaknesses, and comparison with class averages.

Penalty and Late Submission Management:

- **Late Submission Penalty:** If a student submits the test after the due time, penalties should be automatically applied based on predefined rules.
- **Test Extension Requests:** Students should be able to request test extensions, which can be reviewed and approved by faculty members.

Profile Management:

- **Student Profile:** Students should be able to view and edit their profile information, including personal details, enrolled courses, and past assessments.
- **Faculty Profile:** Faculty members should be able to manage their profile, view upcoming assessments, and track student performance.
- **Test History:** Users should be able to track past test attempts, view results, and see any feedback provided by faculty.

Module 2: Perfomace Requirements

- **Response Time:**
  - The platform should respond to user requests (e.g., test creation, result retrieval, test submission) in under 3 seconds.
- **Scalability:**
  - The platform should be able to handle up to 10,000 concurrent users (students and faculty) without significant degradation in performance.
- **Database Efficiency:**
  - Database queries should be optimized for fast retrieval of test schedules, student performance data, and test results, even as the data grows.
- **High Availability:**
  - The system should be designed for minimal downtime with redundancy mechanisms in place. Regular backups should be conducted for the

database, and a failover mechanism should be available to ensure service continuity.

- **Error Handling and Logging:**
    - The system should gracefully handle errors such as failed test submissions, database connection issues, or unauthorized access attempts.

Module 3: Technical Requirements

Technical requirements outline the tools and technologies that will be used in the development of the WAT platform:

- **Frontend:**
    - **React.js** for building dynamic and interactive user interfaces for students and faculty.
    - **Tailwind CSS** for creating a fast, responsive, and customizable UI design.
    - **Redux** for managing application state and ensuring seamless data flow across the application.
    - **Axios** for making API calls to the backend for fetching test data, submissions, and results.
- **Backend:**
    - **Node.js** and **Express.js** for implementing server-side logic, handling requests, and managing routes for test-related activities.
    - **MongoDB** (with **Mongoose**) for efficient data storage, including test information, user profiles, results, and logs.
    - **JWT (JSON Web Tokens)** for secure user authentication and managing sessions.
    - **Nodemailer** for sending OTPs and notifications (e.g., test reminders or result announcements).
- **Hosting and Deployment:**
    - **Heroku** or **Vercel** for deploying and hosting the web application, ensuring scalability and availability.
    - **MongoDB Atlas** for hosting the cloud-based MongoDB database.
- **Version Control:**
    - **Git** for version control to manage code and updates.
    - **GitHub** for hosting the project repository and facilitating collaboration.

- **Testing:**
    - **Jest** and **Mocha** for unit testing and integration testing of the application's core functionality.
    - **Postman** for testing API endpoints to ensure correct functionality and error-free operation.

Module 4: Non Functional Requirements

Non-functional requirements specify the criteria that judge the operation of a system rather than the behaviors. For **WAT**, the non-functional requirements might include:

- **Usability:**
    - The platform should have an intuitive user interface that is easy to navigate, with clear instructions for users to borrow and lend books.
- **Security:**
    - User passwords should be securely hashed before being stored.
    - JWT tokens should be used to ensure secure and authorized access to protected routes.
    - The system should implement HTTPS and data encryption for sensitive data.
- **Maintainability:**
    - The codebase should follow best practices for maintainability (e.g., clean code, modular structure).
    - Documentation should be provided for both the frontend and backend for future developers.
- **Reliability:**
    - The system should be robust and recover from failures automatically (e.g., database failover, retry logic).
    - Backup mechanisms should ensure that data is not lost in case of failure.

Module 4: Third Part Packages and Libraries

Frontend:

1. **React Router**:
    - **Purpose**: React Router enables dynamic routing in React applications, allowing users to navigate between pages (e.g., Home, Login, Book Details, etc.) without reloading the entire page.

- **Why Used**: It simplifies navigation within the single-page application (SPA) and helps manage routes for different views efficiently.

2. **React Hook Form**:
    - **Purpose**: A form handling library that works with React's hooks API for managing forms with minimal re-rendering.
    - **Why Used**: This package allows you to handle form validation and submission without unnecessary re-renders, providing better performance and a smoother user experience, especially for forms like registration, login, and book creation.
3. **React Toastify**:
    - **Purpose**: React Toastify is used to display toast notifications for events like success or error messages.
    - **Why Used**: This package provides a simple and customizable way to show toast notifications, such as when OTP verification is successful or when a book request is made.
4. **Formik**:
    - **Purpose**: Formik is another form-handling library that simplifies form validation, submission, and state management.
    - **Why Used**: Although React Hook Form is preferred, Formik can also be used for more complex forms that require intricate validation logic, such as book creation and user profile editing.
5. **React Select**:
    - **Purpose**: A flexible and customizable select input control for React.
    - **Why Used**: This library can be used for implementing dropdowns (e.g., selecting book genres, rental price range, etc.), offering users a better UX with search and multi-select options.

Backend:

1. **Bcrypt.js**:
    - **Purpose**: Bcrypt is a password hashing library used for securely hashing passwords before storing them in the database.
    - **Why Used**: It provides a secure method for hashing user passwords, preventing them from being stored in plain text.
2. **Joi**:
    - **Purpose**: Joi is a powerful data validation library used to validate input data, such as user registration details, book details, and other user-provided content.

- **Why Used**: It ensures that the data being processed is correct and adheres to expected formats, reducing the chances of errors or malicious inputs.

3. **CORS (Cross-Origin Resource Sharing)**:
    - **Purpose**: CORS is a middleware package that allows or restricts cross-origin requests, enabling secure API communication across different domains.
    - **Why Used**: Used to allow frontend requests from different domains (e.g., React frontend and Express backend) while ensuring security.
4. **Multer**:

Purpose: Multer is a middleware for handling multipart/form-data, used for uploading files, such as book cover images or user profile images.

- Why Used: This package is crucial for handling image uploads in the book creation process, allowing users to upload book images easily.

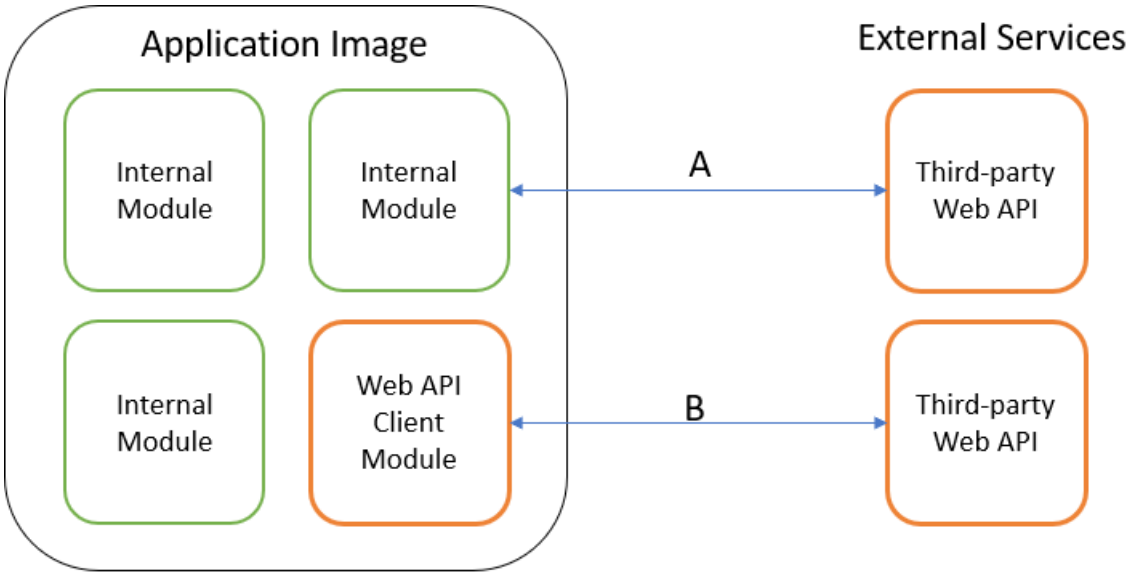


Fig 2.1: Third Packages Integration

CHAPTER 3

SYSTEM DESIGN AND ARCHITECTURE

Module 1: System Overview

The Weekly Assessment Test (WAT) platform is designed to automate and streamline the entire assessment lifecycle for educational institutions. The system provides a comprehensive digital solution for test creation, administration, evaluation, and result analysis. Key functionalities include faculty-driven test creation, automated scheduling, secure test-taking environment, instant scoring for objective questions, and detailed performance analytics.

The system follows a modular and scalable architecture, separating concerns into independent layers to ensure maintainability, performance, and security while supporting role-based access for faculty and students.

Module 2: System Architecture

- **WAT** is built using a **three-tier architecture** comprising:
- **Presentation Layer** (Frontend)
- **Application Layer** (Backend API)
- **Data Layer** (Database)

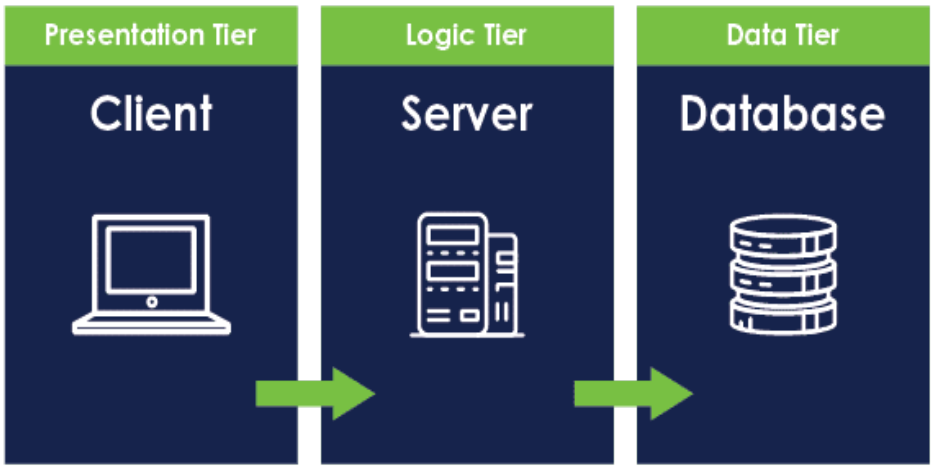


Fig 3.1: Three tier Architecture of System

Presentation Layer

- Built with **React.js** and styled using **Tailwind CSS**.

- Utilizes **Redux Toolkit** for managing application state.
- Handles UI components such as Login, Register, Dashboard, Test Creation wizard, Student Test Interface, and User Profile Management.

Application Layer:

- Built with **Node.js** and **Express**.
- Contains modular route handlers and controllers for:
    - Test creation and Management
    - Student Assessment Handling
    - Automated grading
    - Performance Analytics
- Middleware used for:
    - JWT Authentication
    - Role based authorization
    - Request Validation
    - Error handling

Data Layer:

- Powered by **MongoDB Atlas**, a cloud-based NoSQL database.
- Schemas are defined for:
    - Students
    - Faculty
    - Admin
    - Wat
    - Wat submissions

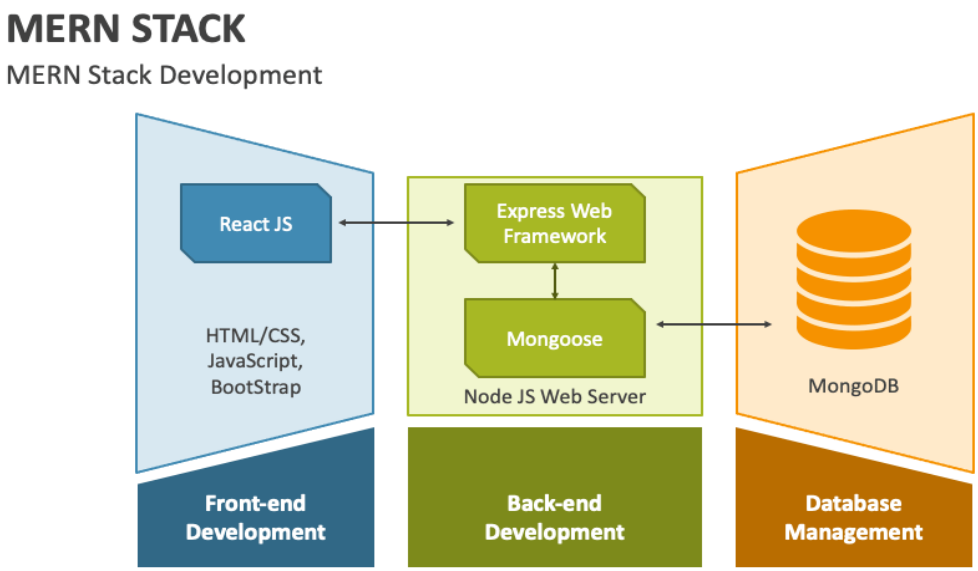


Fig 3.2: Architecture of MERN Stack

Module 3:High Level Design

. Authentication Module:

- Handles user registration, login, and OTP-based verification.
- Uses **JWT (JSON Web Token)** for secure token-based authentication and session management.

2. **User Module:**

- Manages user profiles, including the ability to view and update information.
- Displays user-specific data like **test schedules**, **test results**, **performance analytics**, and **feedback received**.
- Tracks user roles (student or faculty) and grants appropriate access to platform features.

3. **Test Management Module:**

- Allows faculty to create, update, and schedule assessments for students.
- Supports multiple question types (e.g., multiple-choice, descriptive) and auto-grading for objective-type questions.

- Allows faculty to assign time limits, review periods, and display results after test completion.

4. **Assessment Module:**
   - Facilitates students in attempting the assessments within defined time frames.
   - Tracks test progress, ensures real-time submission, and handles timeouts.
   - Implements mechanisms for handling partial submissions, auto-saving of answers, and penalties for late submissions.

5. **Grading and Feedback Module:**
   - Automatically grades objective-type questions (e.g., MCQs, true/false).
   - Simplifies faculty workflows for grading subjective-type questions by providing a grading rubric.
   - Provides instant feedback for objective questions and manual feedback for subjective answers.
   - Displays the overall score and detailed performance analytics on the student's dashboard.

6. **Result and Analytics Module:**
   - Generates performance reports for students and faculty, including test scores, average marks, and comparative analysis with peers.
   - Displays detailed analytics for faculty to identify trends, gaps, and areas of improvement in student performance.
   - Provides real-time visualizations of performance such as graphs and charts for both individual and class-level assessments.

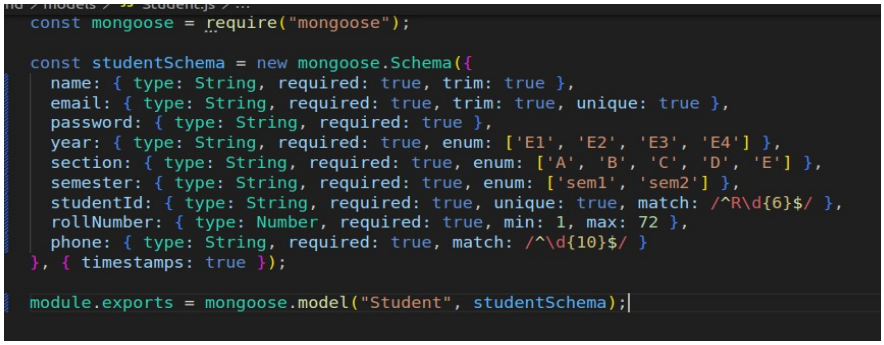
7. **Notification Module:**
   - Sends notifications for upcoming tests, deadlines, and new results via email or on-platform alerts.
   - Sends reminders for pending submissions, corrections, and penalties.
   - Enables students to receive alerts when results are published or if any tests are rescheduled.

8. **Admin/Utility Module:**
   - Manages user roles and permissions, allowing for the creation and management of student and faculty profiles.
   - Handles administrative tasks such as system backups, data migration, and platform monitoring.

- Allows for the removal of inactive or incomplete tests and requests after a certain period to maintain data cleanliness.

Module 4: Low Level Design

Student Schema:



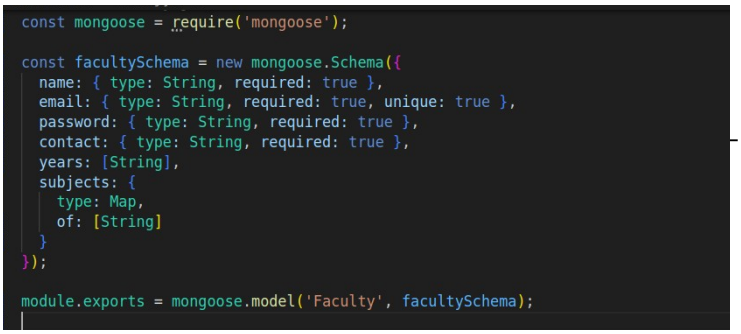
```
const mongoose = require("mongoose");

const studentSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true },
  email: { type: String, required: true, trim: true, unique: true },
  password: { type: String, required: true },
  year: { type: String, required: true, enum: ['E1', 'E2', 'E3', 'E4'] },
  section: { type: String, required: true, enum: ['A', 'B', 'C', 'D', 'E'] },
  semester: { type: String, required: true, enum: ['sem1', 'sem2'] },
  studentId: { type: String, required: true, unique: true, match: /^R\d{6}$/ },
  rollNumber: { type: Number, required: true, min: 1, max: 72 },
  phone: { type: String, required: true, match: /^\d{10}$/ }
}, { timestamps: true });

module.exports = mongoose.model("Student", studentSchema);
```

Fig 3.3: Student Schema

Faculty Schema:



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

const facultySchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  contact: { type: String, required: true },
  years: [String],
  subjects: {
    type: Map,
    of: [String]
  }
});

module.exports = mongoose.model('Faculty', facultySchema);
```

Fig 3.4: Faculty Schema

Admin *Schema:*

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

const adminSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true },
  contactNumber: { type: String, required: true, match: /^[6-9]\d{9}$/ },
  email: { type: String, required: true, unique: true, lowercase: true, match: /^\S+@\S+\.\S+$/ },
  password: { type: String, required: true, minlength: 6 },
  assignedSubjects: { type: Map, of: [String], default: {} }
}, { timestamps: true });

module.exports = mongoose.model('Admin', adminSchema);
```

Fig 3.5: Admin Schema

Wat Schema:

```
const mongoose = require('mongoose');
const questionSchema = new mongoose.Schema({
  questionText: { type: String, required: true },
  options: [String],
  correctAnswer: String
});
const watSchema = new mongoose.Schema({
  facultyId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'Faculty' },
  courseName: { type: String },
  year: { type: String, required: true },//ex:E1,E2,E3
  semester: { type: String, required: true },
  watNumber: { type: String, required: true },
  subject: { type: String, required: true },
  startTime: { type: Date, required: true },
  endTime: { type: Date, required: true },
  questions: { type: [questionSchema], required: true },
  status: { type: String, enum: ['created', 'generated', 'published'], default: 'created' },
});
const WAT = mongoose.model('WAT', watSchema);
module.exports = WAT;
```

Fig 3.6: Wat Schema

Wat Submisson Schema:

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

const watSubmissionSchema = new mongoose.Schema({
  watId: { type: mongoose.Schema.Types.ObjectId, ref: 'WAT', required: true },
  studentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Student', required: true },
  answers: [{
    questionId: { type: mongoose.Schema.Types.ObjectId, required: true },
    selectedOption: { type: String, required: true }
  }],
  score: { type: Number, required: true },
  submittedAt: { type: Date, default: Date.now }
});

watSubmissionSchema.index({ watId: 1, studentId: 1 }, { unique: true });

module.exports = mongoose.model('WatSubmission', watSubmissionSchema);
```

Fig 3.7: WatSubmission Schema

Api Routes

```
app.use("/student",studentRoutes);
app.use("/faculty", facultyRoutes);
app.use("/admin", adminRoutes);
app.use("/auth",userAuth);
app.use('/api/wats', watRoutes);
app.use('/api/mcqs',mcqRoutes);
app.use('/api/subjects', subjectRoutes);
```

Fig 3.8: Api Routes

User Role Modules

Student Module

Routes:

POST /register - Register new student
POST /login - Student login
GET /login - Student login test (protected)
GET /profile - Get student profile
PUT /update - Update student profile
GET /:year - Get students by year and section
GET /students/count - Get student count by year/section
GET /:studentId - Get student details with WAT marks

Faculty Module

Routes:

POST /register - Register new faculty
POST /login - Faculty login
GET /profile - Get faculty profile
PUT /update - Update faculty profile
GET /all - Get all faculty
GET /assigned-years - Get assigned years for faculty
GET /students/:year/:section - Get students by year and section
GET /students-by-year/:year - Get all students by year
GET /wats - Get all WATs for faculty
GET /wats/:year - Get WATs by year
GET /:facultyId/years - Get faculty assigned years

Admin Module

Routes:

POST /register - Register new admin
POST /login - Admin login
GET /login - Admin login test (protected)
GET /profile - Get admin profile
PUT /update - Update admin profile

Wat Module

Routes:

GET /wats/by-year/:year - Get WATs by academic year
GET /wats/active/by-year/:year - Get active WATs for year
GET /wats/display/:id - View WAT details

POST /wats/create - Create new WAT
POST /wats/submit - Submit WAT answers
GET /wats/submissions/:studentId - Get student submissions
GET /wats/by-year-faculty/:year/:facultyId - Get faculty's WATs
GET /wats/by-year-subject - Get WATs by subject
PUT /wats/:id - Update WAT details

Work Flow Diagram

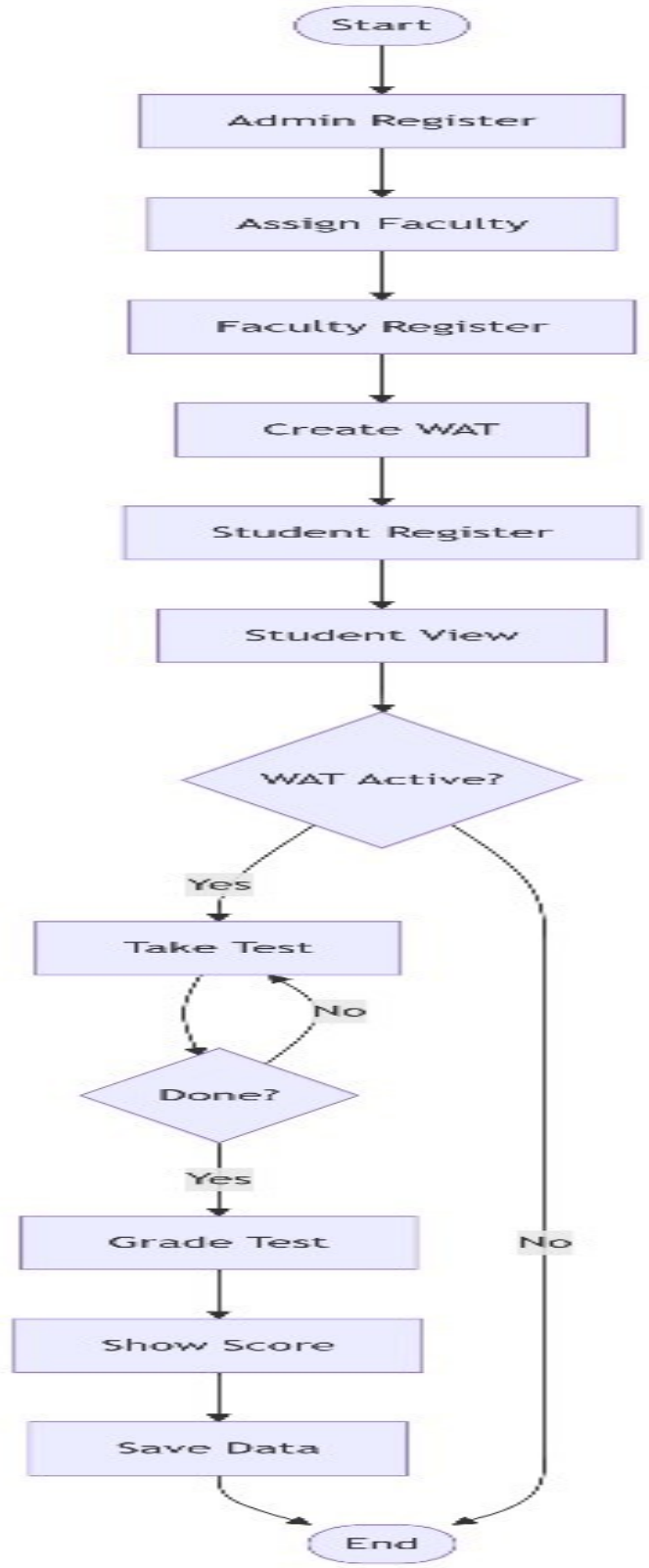


Fig 3.10: Work Flow

CHAPTER 4

USER INTERFACE OVERVIEW

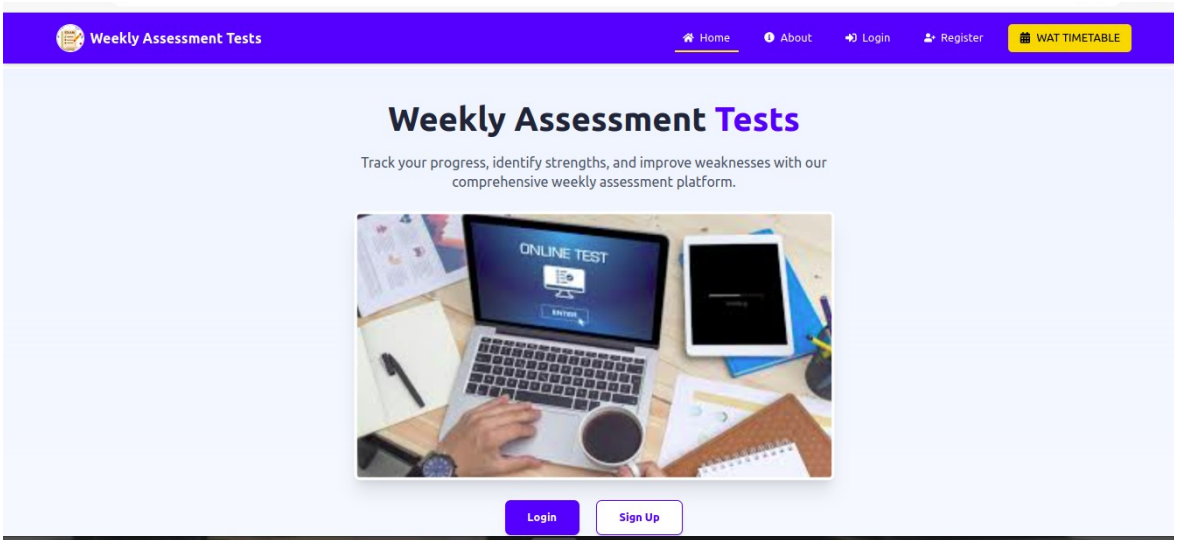


Fig 4.1: Home Page

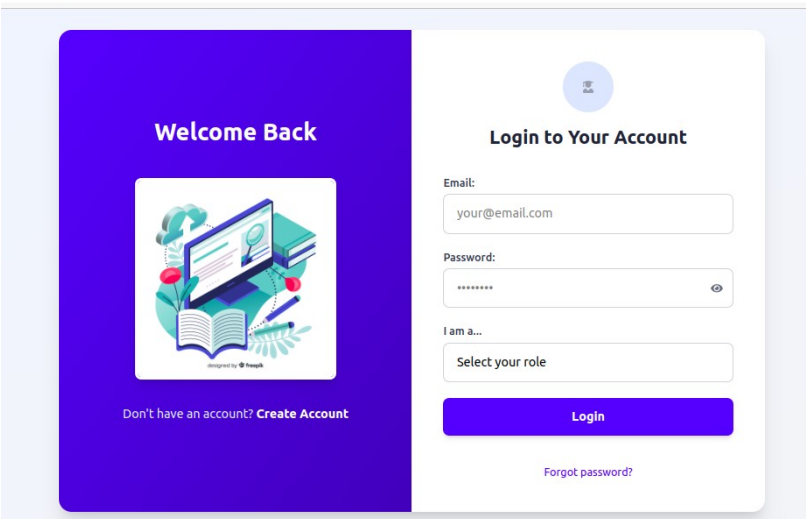


Fig 4.2: Login Page

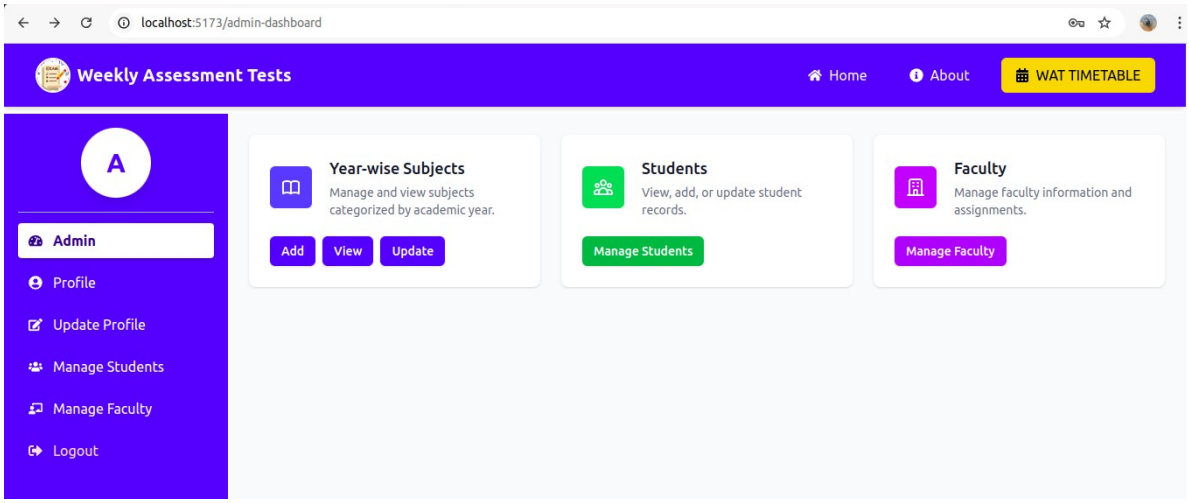


Fig 4.3: Admin Dash Board

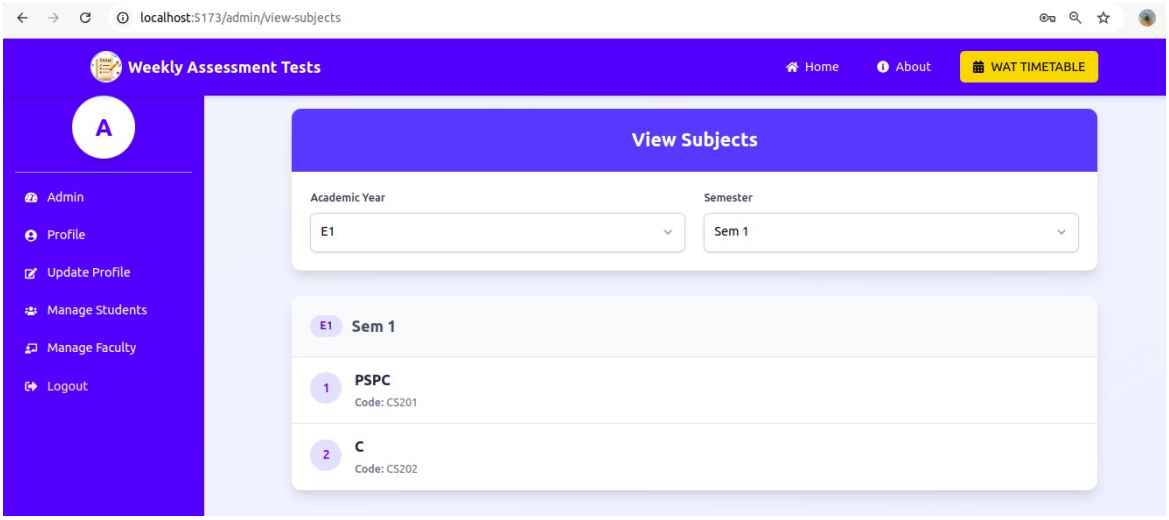


Fig 4.4: View Year wise Subjects page

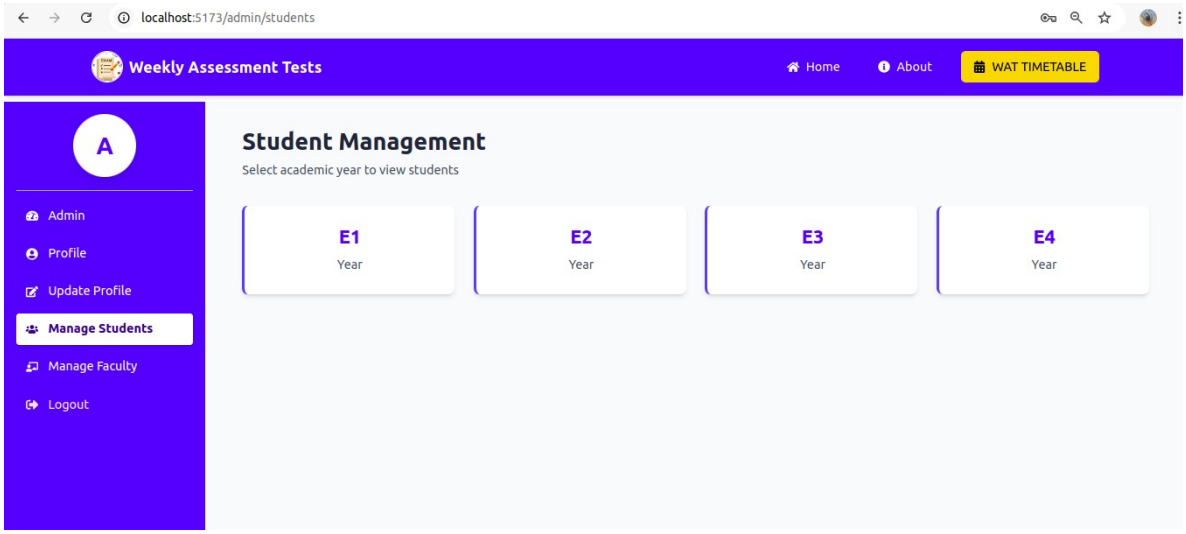


Fig 4.5: View Students Details page

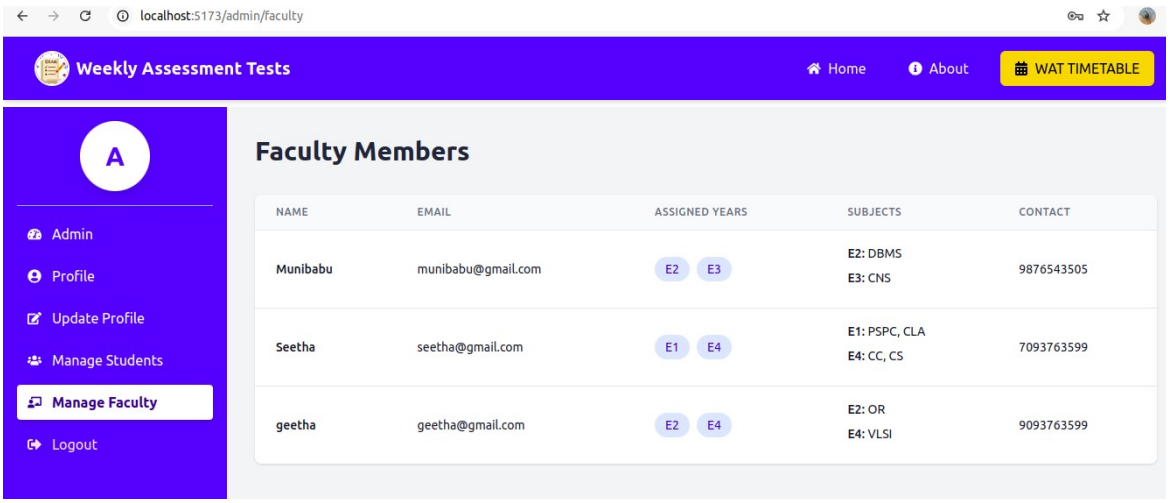


Fig 4.6: View Faculty Details Page

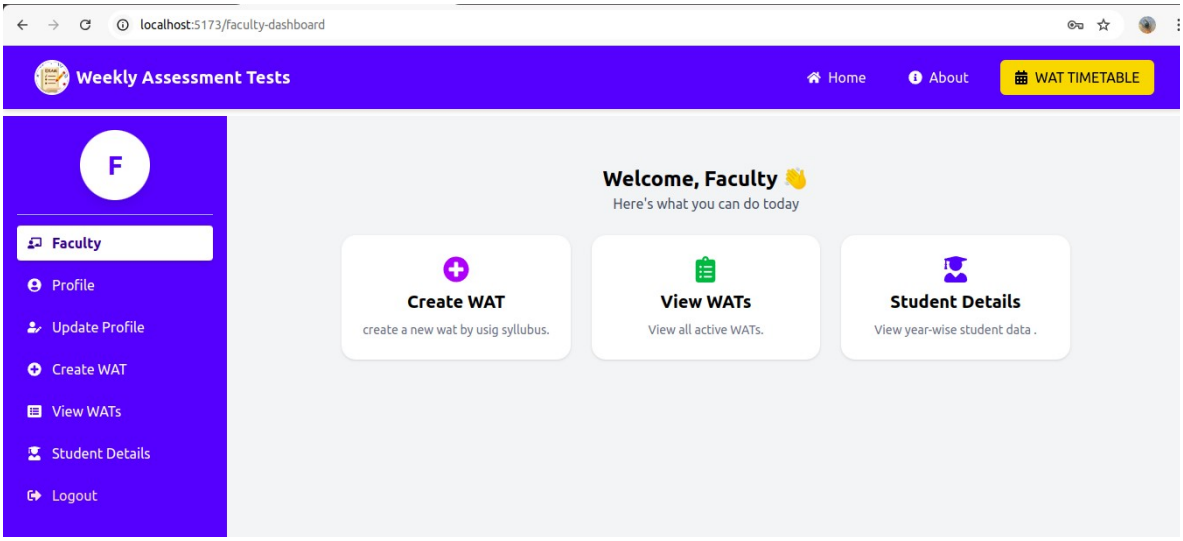


Fig 4.7: Faculty DashBoard

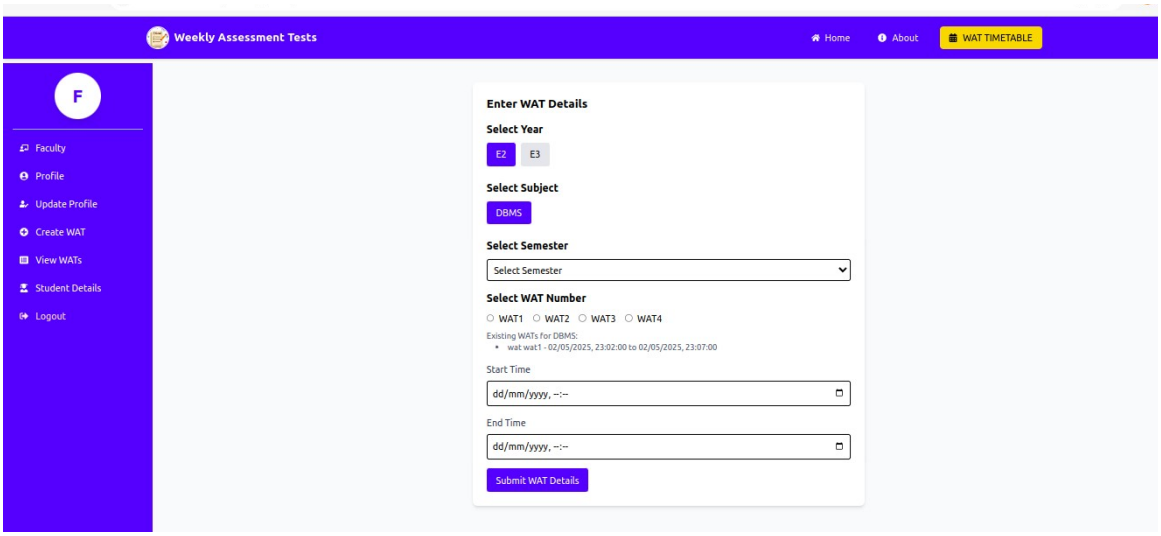


Fig 4.8: Wat Creation page

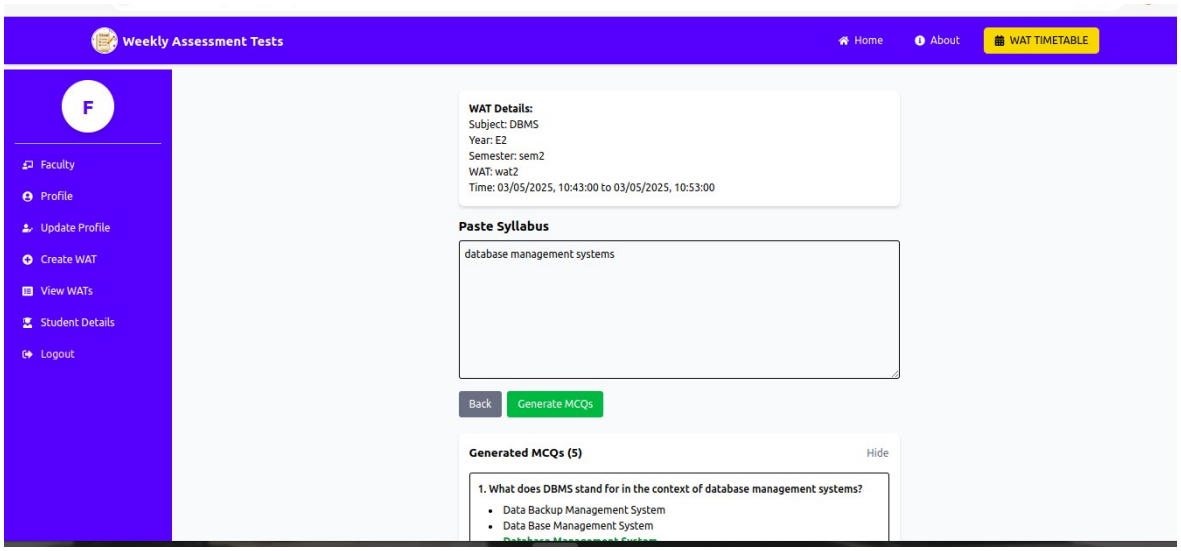


Fig 4.9. Mcqs Generation page

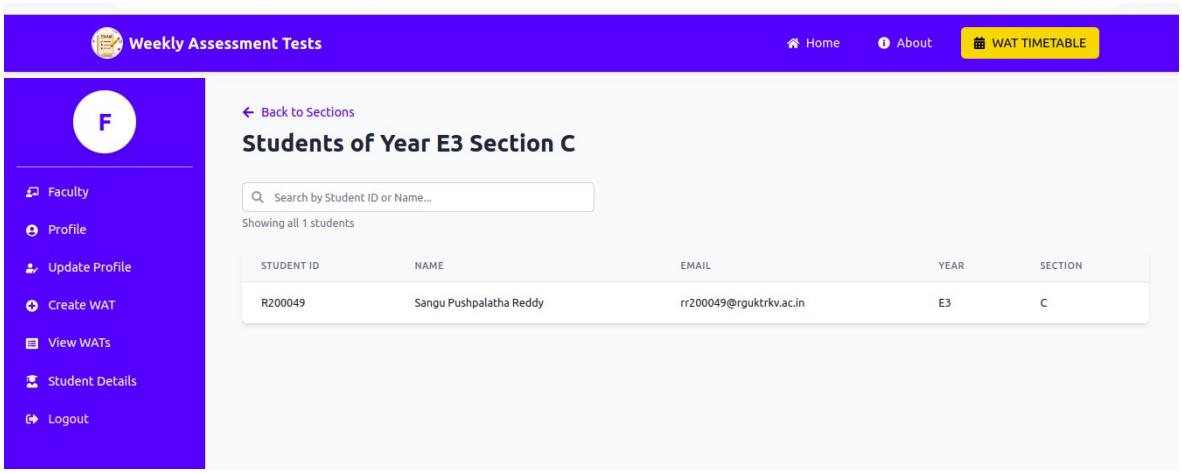


Fig 4.10: View Student Details

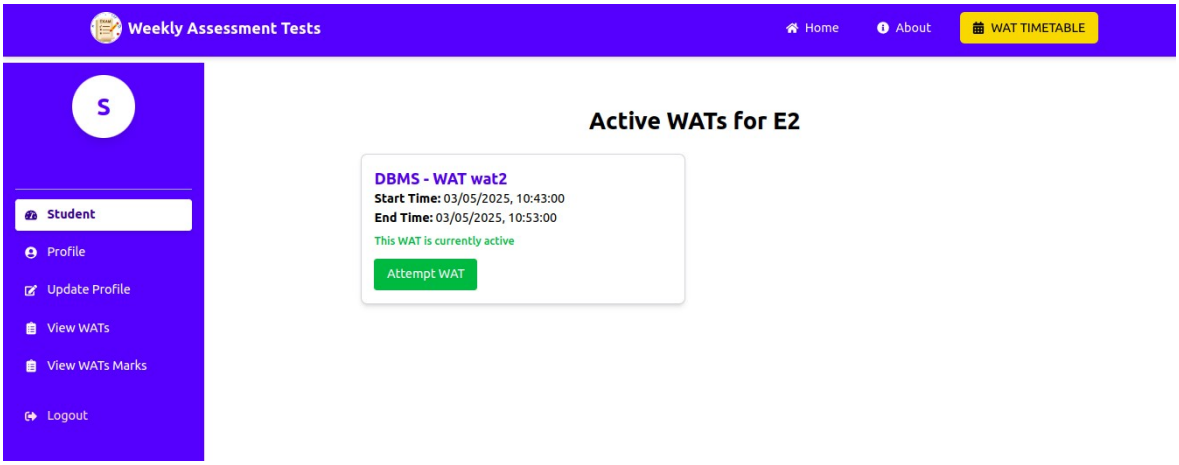


Fig 4.11: Student Dashboard

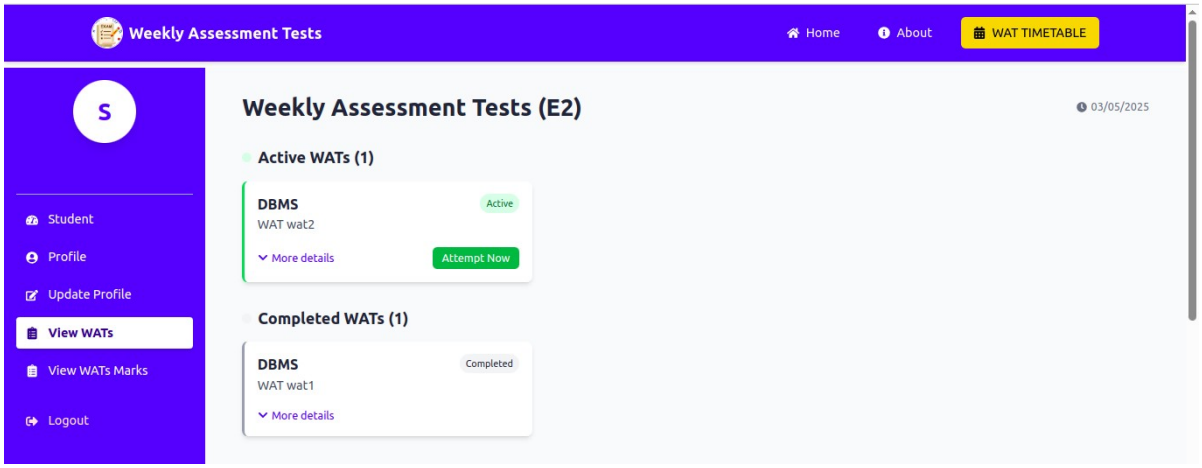


Fig 4.12: View Wats Page

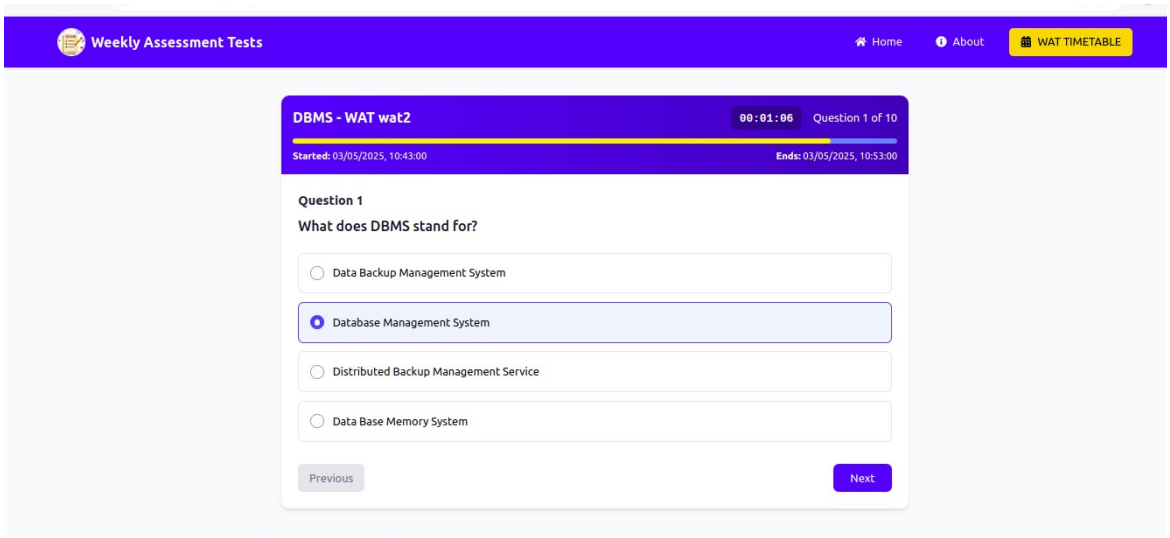


Fig 4.13: Wat Attempt Page

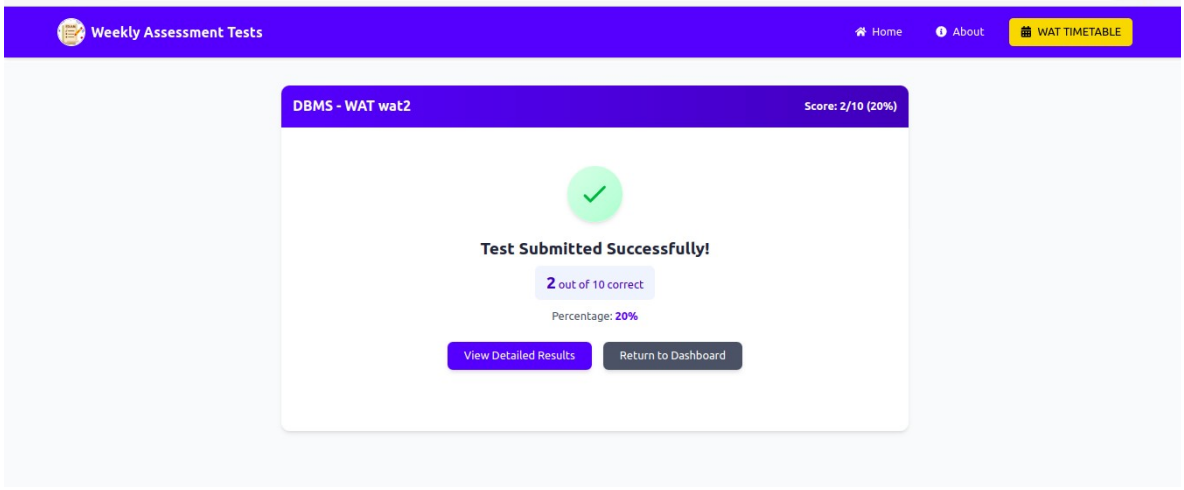


Fig 4.14: Score Display Page

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion:

The **WAT** platform began as an academic initiative to modernize traditional assessment methods and has evolved into a comprehensive digital solution for educational institutions. Through systematic design and robust implementation, the platform enables faculty to create, administer, and evaluate assessments efficiently while providing students with a seamless test taking experience.

This journey allowed me to explore and apply critical development concepts such as **centralized error handling**, **Joi-based data validation**, **image base64 processing**, and a **modular RESTful API architecture**. I enhanced my front-end skills using **React and Redux Toolkit**, styling interfaces with **Tailwind CSS**, and maintaining consistent state and data flow across the application. The backend, built using **Node.js and Express**, connected seamlessly with **MongoDB Atlas**, supporting efficient data modeling and performance.

Building WAT Platform has significantly sharpened my skills in both **frontend and backend development**, deepened my understanding of **full-stack integration**, and cultivated a problem-solving mindset grounded in real-world application needs. This project represents not only a technical milestone but also a meaningful contribution toward creating **sustainable, community-driven solutions**.

Future Enhancement:

1. Enhanced Analytics & Reporting

Your WAT system could benefit from **advanced analytics** to provide deeper insights into student performance. Implementing **data visualization dashboards** (using libraries like Chart.js or D3.js) would allow faculty and admins to track trends, identify weak areas, and compare performance across batches. Features like **automated PDF reports**, **subject-wise toppers**, and **class average trends** would help in data-driven decision-making. Additionally, integrating **predictive analytics** to flag at-risk students based on past performance could enable early interventions.

2. Mobile App & Offline Support

A **Progressive Web App (PWA) or dedicated mobile app** would improve accessibility, allowing students to take tests on smartphones with **offline mode support** (syncing answers when reconnected). Push notifications for **upcoming tests, deadline reminders, and result announcements** would keep users engaged. Features like **QR-code-based test entry** or **biometric authentication** could further streamline exam security.

3. Integration with LMS & Scalability Improvements

To make the system more robust, consider **integrating with Learning Management Systems (LMS)** like Moodle or Google Classroom for centralized course management. Moving to **microservices architecture** (using Docker/Kubernetes) would improve scalability, especially during peak exam periods. Adding **multi-language support** (for regional languages) and **accessibility features** (screen-reader compatibility) would make the platform more inclusive. Finally, **automated backup systems** and **disaster recovery plans** would ensure data security in case of failures.

REFERENCES

Official Documentation & Guides

1. **MongoDB Official Docs**
   https://www.mongodb.com/docs/
   - Best for learning NoSQL database structure, queries, and Atlas (cloud DB).
2. **Express.js Official Docs**
   https://expressjs.com/
   - Covers middleware, routing, and REST API development.
3. **React Official Documentation**
   https://react.dev/
   - Updated guide for React hooks, components, and state management.
4. **Node.js Official Docs**
   https://nodejs.org/en/docs
   - Essential for backend JavaScript runtime.

Free Courses & Tutorials

5. **MERN Stack Full Guide – FreeCodeCamp**
   https://www.freecodecamp.org/news/fullstack-react-beginner-to-advanced/
   - Hands-on project-based learning.

Books & Advanced Learning

6. **"Full-Stack React Projects" (Shama Hoque)**
   https://www.packtpub.com/product/full-stack-react-projects-second-edition/9781801073249
   - Covers MERN + Redux, JWT Auth, and deployment.

Deployment & Best Practices

7. **Deploying MERN on Vercel/Heroku**
   https://dev.to/divofred/how-to-deploy-a-mern-stack-app-on-heroku-3kmj
   - Step-by-step deployment guide.