Educational Platform Project Documentation

Project Title: Educational Platform for University

Team Members:

Bakytzhan Akhmukhamedov IT-2305

Instructor Name: Bekzat Adzhan
Submission Date: February 25, 2025

Project Overview

Objective:
The Educational Platform was developed to streamline academic task management and enhance the educational experience for university students and teachers. The primary goal was to create a centralized web application that simplifies task creation, submission, and performance tracking, addressing flaws in traditional academic workflows. By providing tools for teachers to manage assignments and for students to submit work and monitor progress, the platform aims to improve communication, convenience, and engagement in the learning process.
Description:
The completed Educational Platform is a fully functional web application featuring distinct interfaces for teachers and students. Teachers can create and manage tasks through taskcreate.html, while students can view and submit assignments via task.html. Additional features include user profiles (profile.html), grade tracking (grades.html), and interactive quiz/exam modules (quiz.html and exam.html). The front end is built using HTML, CSS (styled with Bootstrap for responsive design), and JavaScript for dynamic interactions. The back end is powered by Node.js with Express.js, handling API requests, and MongoDB serves as the NoSQL database, storing user data, tasks, submissions, grades, and quiz/exam results. File uploads are managed using multer, and JWT authentication ensures secure access for all users.

Target Audience:
The platform serves university students and teachers. Students benefit from an organized system to manage assignments, track grades, and participate in quizzes/exams, while teachers gain a tool to efficiently create tasks, monitor submissions, and assess student performance.

Why I choose this topic:
I chose the Educational Platform topic because it evolved from a series of ideas that grew in scope before I narrowed it down to fit the project timeline. Initially, I aimed to build a testing system for schools, but this concept expanded into a broader platform. I then considered adding an anti-cheat system to detect copied work by analyzing log activity, envisioning a website that could serve schools comprehensively. However, due to time constraints, I had to drop the anti-cheat feature. Ultimately, I focused on creating a convenient interface that notifies teachers and students about recent activities—like homework tasks, grades, and calendar updates—ensuring a practical and user-friendly tool for academic management.

Project Features and Implementation

Final Product:
The Educational Platform will be accessible on Heroku and its accompanied by detailed documentation and a demo video showcasing its functionality. The documentation includes the system architecture, API endpoint specifications, and user guides for both teachers and students. The demo video highlights key features, such as task creation, submission, profile management, grade tracking, and quiz/exam participation

Key Features:

- **Task Creation for Teachers (taskcreate.html)**: Teachers can select a subject and week, input task details (title, description, deadline, requirements, notes), and upload a requirements file. The /api/teacher/subject/:subjectId/tasks endpoint saves tasks in the subjectcontents collection, with uploaded files stored in the uploads/ directory (e.g., Lecture1-123456789.pdf).
- **Task Submission for Students (task.html)**: Students can view tasks, download requirements files, and submit assignments. The /api/subject/task/:taskId/submit endpoint handles submissions, storing them in the UserSubmission collection, and the interface disables resubmissions after the first upload.
- **User Profiles (profile.html)**: Both teachers and students can view and update their personal details, with data fetched via the /api/me endpoint.
- **Grades Tracking (grades.html)**: Students view their grades for tasks, quizzes, and exams via /api/grades, while teachers access class performance through /api/teacher/grades.
- **Quizzes and Exams (quiz.html, exam.html)**: Teachers create assessments with multiple-choice, true/false, or open-ended questions, stored in MongoDB and accessed via /api/quizzes and /api/exams. Students can attempt these, with automatic grading for objective questions and manual review for subjective ones.
- **Security**: JWT authentication ensures secure access, with role-based restrictions (teachers manage tasks, students submit them).
- **Responsive Design**: Bootstrap ensures the platform is accessible on desktops and mobile devices.

Implementation Details:
The back end uses Express.js to define RESTful API endpoints, such as /api/teacher/subject/:subjectId/tasks for task creation and /api/subject/task/:taskId/submit for submissions. MongoDB stores data in collections like users, subjectcontents, and usersubmissions. File uploads are handled by multer, preserving original filenames and extensions. The front end leverages Bootstrap for a responsive layout, with JavaScript managing dynamic rendering and form submissions. Error handling is implemented throughout, with notifications for failed actions (e.g., missing task title) and fallbacks for unavailable content.

Technology Stack

- **Back-End Framework:** Node.js with Express.js – Chosen for its lightweight nature and strong community support, ideal for rapid API development.
- **Database:** MongoDB – Selected for its flexibility in handling unstructured data like tasks and quiz responses.

- **Version Control:** GitHub – Used for version control, enabling regular commits and backups throughout development.
- **Deployment Tools:** Heroku – Provided a simple deployment pipeline, suitable for hosting a small-scale educational app.
- **Front-End Tools:** HTML, CSS (Bootstrap), JavaScript – Bootstrap ensured a responsive design, while JavaScript enabled dynamic functionality like form validation and content rendering.

  These tools were selected for their compatibility and ease of use, allowing me to focus on feature development while ensuring a professional, deployable application.

Challenges and Solutions

Throughout the development of the Educational Platform project, I encountered a multitude of challenges that tested my resilience and problem-solving abilities, particularly around JWT authentication, environment file configuration, JavaScript front-end development, and specific bugs that arose during implementation. One significant hurdle was managing JWT authentication, where I struggled with token validation issues due to improper configuration in the .env file; the environment variables for the JWT secret were not loading correctly, causing authentication errors like "Invalid token" and unexpected logouts after saving content, as the back-end route /api/me failed to verify the token, redirecting users to the login page. Additionally, I faced persistent bugs in the JavaScript front end, an area I found particularly frustrating because I don't enjoy building front-end interfaces—rendering dynamic content on task.html often resulted in blank pages when the server response was malformed, and the toggle buttons on the earlier teacher-subject.html page only worked after adding content due to event listener attachment issues, requiring me to rewrite the event delegation logic multiple times. File handling posed another major challenge; uploaded files lacked proper extensions (e.g., downloading as 123456789 instead of Lecture1.pdf), which I eventually resolved by adjusting the multer configuration to preserve original filenames, but not before hours of debugging and testing. I also dealt with third-party interference from ritrag.com, which caused net::ERR_CONNECTION_TIMED_OUT errors and blank pages, forcing me to test in incognito mode and remove adware from my system. Furthermore, the front-end rendering issues were exacerbated by my discomfort with JavaScript, as I struggled to manage asynchronous fetch calls in fetchContent(), leading to unhandled promise rejections that crashed the page after saving tasks, which I fixed by adding try-catch blocks and better error logging. These challenges, while daunting, ultimately deepened my understanding of full-stack development, though they reinforced my preference for back-end work over front-end design due to the constant debugging required for dynamic UI elements.