**Internship/Academic Project: Exam Result System**

**a. What did the system do?**

The Exam Result System is a web application designed for faculty members to manage and declare student results efficiently. Faculty members can authenticate themselves, after which they are authorized to add, view, update, or delete student details and records. Additionally, they can declare results for individual students or the entire class. The system leverages a full-stack solution, with a front end in HTML, CSS, Bootstrap, and Vue.js, and a backend built with Node.js, Express.js, and MongoDB. MongoDB serves as the primary database to store and manage student records and result information.

**b. What other systems have you seen in the wild like that?**

Similar systems are often found in educational institutions in the form of Student Management Systems (SMS) or Learning Management Systems (LMS). These systems, like Google Classroom, Blackboard, or Moodle, offer various features for faculty to manage student data, assign tasks, and track performance. Commercial products often add functionalities such as attendance tracking, grading, and even student analytics, which gives institutions insights into academic progress. While these products offer comprehensive solutions, the Exam Result System is more focused on result declaration and student management.

**c. How do you approach the development problem?**

My approach to developing this system involved breaking down the problem into manageable components:

1. **Frontend Design**: I began by creating a user-friendly interface using HTML, CSS, and Bootstrap to ensure responsive and accessible layouts. Vue.js was integrated to manage dynamic parts of the frontend, particularly for handling student records and updating information without page reloads.
2. **Backend and API Design**: Using Node.js and Express, I designed RESTful APIs for the CRUD operations required by faculty. Each operation corresponds to specific user actions like adding or deleting a student record.
3. **Database Integration**: MongoDB was chosen for its flexibility with dynamic schema, which was beneficial as student records could vary and be updated frequently.
4. **Security and Authentication**: To secure the system, I implemented faculty authentication and authorization. By restricting access to specific endpoints, I ensured only authorized faculty members could access sensitive features.

**d. What were interesting aspects where you copied code from Stack Overflow?**

**Password Hashing with bcrypt.js**: For secure password storage, I needed to hash passwords before saving them to the database. I found a code snippet on Stack Overflow that demonstrated how to use bcrypt.js to hash passwords during user registration and compare hashed passwords during login. By following this example, I learned to securely integrate bcrypt.js for authentication and ensure sensitive data was protected.

**e. What did you learn from some very specific copy-paste? Mention explicitly some of them.**

**JWT Authentication**: I needed a secure way to manage authentication and authorization, especially for handling sessionless authentication in APIs. From Stack Overflow, I learned how to implement JWTs in a Node.js and Express setup. This included generating a token upon successful login and attaching it to user sessions, allowing for secure, stateless authentication across endpoints. This taught me how to structure token-based authentication in a scalable way, improving both security and user experience.Top of Form

Bottom of Form