**Software Requirements Specification Summary**

**Requirements:**

- The system must provide students with the ability to view a comprehensive list of all their enrolled modules.

- Students must be able to search for modules using criteria such as course codes, title, or department

- Students must be able to register and deregister for modules.

- Administrators should be able to manage modules, student lectures, and other system settings within the application through an admin dashboard.

- Prospective students can submit applications for student status. Administrators are responsible for reviewing these applications and approving or denying them accordingly.

**Security Requirements:**

- Implement secure authentication mechanisms (e.g., password hashing, encryption) to protect user credentials.

- Implement a password hashing abstraction layer to ensure the system can adapt to future password requirements while supporting legacy passwords, in accordance with security by design principles. (Find reference for this)

- Encrypt sensitive data (e.g., user information, module registrations) both in transit and at rest.

- Utilize the AES/CBC/PKCS5Padding encryption algorithm with a 128-bit key to implement column-level encryption for sensitive data (names, contact details etc) stored in the database. (Find reference for this)

- Implement role-based access control (RBAC) to restrict access to system functionalities based on user roles [ADMIN/MODULE\_ADMIN/STUDENT]. (find reference for this)

- Use HTTPS protocol to encrypt data transmission between the client and the server to prevent data interception and tampering.

- Implement input validation mechanisms to prevent common security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks. (find reference for this)

- Implement logging mechanisms to track user activities and system events for audit and forensic analysis. (NGINX)

- Implement micro-segmentation within Docker containers to enhance network security by isolating and segmenting communication between containers (find reference for this)

**Technologies:**

* Vue.js (frontend framework)
* Nginx (proxy)
* Spring Boot (
* Postgres/Neon
* Docker
* AWS EC2

**AUTHOURIZATION / ACCESS CONTROL:**

For this application I implemented a hybrid solution of Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC).

Since the application is rather simple roles have implicit privileges instead dynamic privileges, more over to align with the principles of Least Privilege and Separation of Concerns a user may only be assigned a single role:

APPLICANT: an applicant can only view their profile details and application status.

STUDENT: A student can do what an applicant can and can also register/deregister for modules and view their modules.

MODERATOR: A moderator can create and remove modules as well as post content to the modules they are moderator of.

ADMIN: an admin has the authority to access all functionality of the application.

There is noticeably a separate role hierarchy namely ADMIN > MODERATOR and ADMIN > STUDENT > APPLICANT. This is done to support Least Privilege.

ABAC is used to implement access control for the modules where students are only granted access (student access) to modules they are registered for and moderator’s access (moderator access) to modules they are moderators of.

**AUTHORIZATION / ACESSS CONTROL**

I implemented a hybrid access control solution combining Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC). Users are assigned a single role for simplicity and to adhere to principles of Least Privilege and Separation of Concerns:

* APPLICANT: Can view profile details and application status.
* STUDENT: Can perform applicant tasks and can register/deregister for modules, view module details.
* MODERATOR: Can create/remove modules, post content to assigned modules.
* ADMIN: Has full access to all application functions.

A role hierarchy is established as ADMIN > MODERATOR and ADMIN > STUDENT > APPLICANT to enforce Least Privilege. ABAC is utilized for module access control, granting students access only to registered modules and moderators’ access only to assigned modules. Spring Boot was used to accomplish this through filtering and request pattern matching.

**AUTHENTICATION:**

In the application I implemented MFA for the application process with ReCAPTCHA and email verification. And then there is the use of username and password authentication for login but additional email confirmation for logins from different devices. JWT tokens are used for subsequent authentication purposes.

*ReCAPTCHA*

*Email Verification*

*Username and password*

Usernames must be unique and have a length of (3-30) and are restricted to alphanumeric characters and a subset of special characters. Passwords are also subject to similar restrictions but must have a length of at least 8 characters.

*JWT Tokens*

For authentication I implemented token-based authentication, namely JWT token authentication. The token only contains content that isn’t PII (Personally Identifiable Information) but besides their username for identification and authorization purposes it also contains the user’s role (this isn’t used for access control purposes but merely for the front end to decide what to display), the issuer (backend service) and the id of the intended audience. This information will be kept confidential in transit (HTTPS).

**DATA INTEGRITY:**

**LOGGING AND MONITORING:**

**SECURITY TESTING:**