(CAPS – Comprehensive Assessment and SPreparation System) Technical Design Decisions

1. Frontend Framework: React with Vite and Tailwind CSS

• **Rationale**: React was selected for its component-based architecture and mature ecosystem, allowing reusable and testable user interface components.

• **Vite** was used for its fast development build process and efficient hot-module reloading, enhancing developer productivity.

• **Tailwind CSS** provides utility-first styling, enabling consistent design patterns and faster UI development with minimal custom CSS.

2. Backend Framework: Laravel (PHP)

• **Rationale**: Laravel offers a clean MVC architecture, but we used the MMA (Modular Monolithic Architecture), built-in routing, authentication features, and robust support for RESTful API development.

• Its integration with Eloquent ORM, middleware, and command-line tooling via Artisan accelerates development and maintenance.

3. Database: MySQL with phpMyAdmin

• **Rationale**: MySQL was chosen for its performance, wide support, and compatibility with Laravel.

• **phpMyAdmin** was added to provide a web-based interface for easy database management, especially during QA and development phases.

4. Role-Based Access Control (RBAC)

• **Rationale**: CAPS requires strict access control for multiple user roles, including Dean, Program Chair, Instructor, and Student.

• Laravel's Gate and Policy system enforces secure, role-specific permissions across all modules, including question approval workflows, exam generation, and content modification.

5. Student Practice and Exam Features

• **Rationale**: The student interface is designed for simplicity and minimal cognitive load, enabling focused exam preparation.

• Includes features such as a configurable timer, immediate or delayed feedback, and a clean UI for practicing questions.

6. Containerized Development with Docker Compose

• **Rationale**: Docker was used to ensure consistent and isolated development environments across the team.

• **Key services containerized:**

  o **(caps-frontend)** – React frontend

  o **(caps-backend)** – Laravel backend

  o **(caps_mysql)** – MySQL database

  o **(phpmyadmin)** – Web-based database viewer

• A shared Docker network enables seamless inter-service communication and simplifies setup for new developers and QA testers.

7. NGINX Deployment Strategy

• **Local NGINX (Static Frontend Serving)**:

  o NGINX is used locally to serve the React frontend's production build for testing the deployment view.

• **AWS NGINX (Reverse Proxy)**:

  o On the AWS server, NGINX acts as a reverse proxy, routing external HTTPS requests to both the frontend and backend services.

  o Let's Encrypt SSL certificates are used to enforce secure HTTPS communication.

  o This setup ensures centralized request handling and domain-based routing.

8. QA Testing Strategy

• **Rationale**: The QA process is aligned with modern testing practices and automation tools:

  o **Frontend**: Cypress for end-to-end UI automation

  o **API Testing**: Postman with Newman for automated API test runs

9. Staging Environment

• **Rationale**: The staging environment is now hosted on AWS cloud infrastructure to support scalable, real-world testing conditions.

• This full-stack environment mirrors the production setup using Dockerized services for the frontend, backend, and database.

• AWS provides reliable uptime, easier collaboration for remote teams, and better integration with tools for CI/CD, testing, and monitoring.

10. Security and Data Integrity

• **Rationale**: Security is integrated at multiple layers:

  o **HTTPS** enforced via Let's Encrypt

  o **RBA**C controls access to sensitive features

  o Sensitive environment variables are stored securely, and Docker network isolation limits database exposure