Project Title: Accreditation and data management assistant

Name: Vidya Sinha 21, Dhruvi Patel 29

Deployment Process

The deployment of our system was carried out on a cloud-based infrastructure to ensure scalability, availability, and ease of management. For this project, the Flask backend was deployed on Render using Docker containerization, while the React/TypeScript frontend was deployed on Vercel.

The backend was containerized with a Dockerfile specifying Python and dependencies from requirements.txt, then pushed to the Render service for automatic builds and deployment. Environment variables such as Supabase keys and Flask secrets were securely configured in Render's dashboard, ensuring seamless connectivity with the database and storage services. A custom domain was also mapped through Render, making the backend accessible at a user-friendly URL.

Similarly, the frontend was deployed on Vercel via GitHub integration, with environment variables pointing to the backend API endpoints. Deployment was verified through live URLs, with the backend accessible at https://madms-bounceback-backend.onrender.com and the frontend at **https://madms-assistant.vercel.app/** .

Challenges faced during deployment included environment variable management, CORS errors, and initial database connectivity issues, which were resolved through secure secret storage, proper CORS configuration, and close reference to Render and Vercel documentation.

Monitoring Strategy

To ensure operational visibility and reliability, a structured monitoring strategy was established. The backend was instrumented using Prometheus to scrape custom Flask metrics via exporters, and Grafana dashboards were configured to visualize key indicators. For the frontend, Vercel Analytics provided insights into user interactions and system availability. Key performance indicators (KPIs) tracked included average API response time for critical endpoints such as document uploads and academic performance queries, system uptime percentages, and error rates measured through HTTP 4xx and 5xx responses.

Alerts were configured to notify the team in case of prolonged downtime or sudden increases in error rates. This monitoring setup provided a real-time overview of system health, enabling rapid

detection and resolution of performance bottlenecks. Evidence of monitoring included screenshots of Grafana dashboards and Prometheus graphs showing latency distribution, uptime statistics, and error trends.

Maintenance Plan

A maintenance plan was also implemented to ensure system stability over time. Weekly tasks included automated Supabase database backups and log reviews to detect anomalies. Monthly maintenance involved updating dependencies for both Python and Node.js ecosystems, along with security audits such as reviewing access logs and rotating secrets.

A quarterly performance review was scheduled to assess scalability requirements, resource usage, and potential upgrades. Routine optimizations, such as caching frequent queries, indexing databases, and leveraging Render's autoscaling features, ensured responsiveness under growing workloads. Risks such as hardware failures were mitigated by Render's cloud redundancy and backup strategies, while dependency vulnerabilities were managed by pinning versions and using automated tools for alerts.

Challenges.

=> The project also faced several operational challenges. Early in deployment, environment variable misconfigurations led to database connection errors, while improper CORS setup temporarily blocked frontend-backend communication.

=> These were resolved through consistent testing across staging and production environments and consultation of Render's deployment guidelines. Despite these hurdles, the deployment and operations strategy ultimately delivered a reliable, scalable, and maintainable system for institutional use.