



**COLLEGE CODE:** 9623

**COLLEGE NAME:** Amrita college of engineering and technology

**DEPARTMENT:** Computer Science Engineering

**STUDENT NM ID:** AB0F2B3EBEDF8BDB2EC44A024D951EA7

**ROLL NO:** 962323104092

**DATE:** 06-10-2025

**Completed the project named as Phase 4**

**TECHNOLOGY PROJECT NAME :**

**Angular JS with SQL Integration**

**SUBMITTED BY,**

**NAME :K.J.SHAHAMBHARI**

**MOBILE NO:8072123291**

# PHASE 4: Enhancements & Deployment

## 1. Additional Features

**Goal:** Extend functionality based on user feedback or backlog items.

**Ideas:**

- **Search & Filter Functionality** for tables or data lists.
- **Pagination** for large datasets (especially from SQL).
- **Role-Based Access Control (RBAC)** for admin/user dashboards.
- **Export Data** to CSV or PDF from the AngularJS frontend.
- **Email Notifications** using backend integration.

## 2. UI/UX Improvements

**Goal:** Improve the design and interactivity of the application.

**Tasks:**

- Refactor UI using a CSS framework (Bootstrap, Tailwind CSS).
- Add **loading spinners**, **skeleton screens**, and **toast notifications**.
- Improve **form validation** and input feedback.
- Ensure **responsive design** (mobile/tablet compatibility).
- Conduct a **UX audit** for confusing navigation or flows.

## 3. API Enhancements

**Goal:** Improve the backend communication layer.

**Tasks:**

- Add **input validation & sanitization** on the backend.
- Use **pagination parameters** (`limit`, `offset`) in API responses.

- Improve **error messages** and use standardized response formats (e.g., JSON with status, message, data).
- Add **authentication endpoints** (JWT, session-based, etc.).
- Optimize SQL queries to prevent over-fetching data.

## 4. Performance & Security Checks

**Goal:** Ensure app is optimized and secure before going live.

### Performance:

- Enable **lazy loading** in AngularJS where possible.
- Minify and compress JS/CSS files.
- Use **caching** strategies for static content.

### Security:

- Prevent **SQL Injection** by using parameterized queries.
- Apply **CORS policies** properly.
- Sanitize user input on both frontend and backend.
- Use HTTPS for all endpoints.
- Add **rate limiting** or throttling on critical APIs.

## 5. Testing of Enhancements

**Goal:** Ensure new features and changes are stable.

### Testing Types:

- **Unit Testing** for AngularJS components/services.
- **Integration Testing** for API + SQL.
- **Manual Testing** for UI flows, forms, and edge cases.
- Use tools like **Postman** to test backend endpoints.
- Create **test cases** for any new feature added.

## 6. Deployment (Netlify, Vercel, or Cloud)

**Goal:** Make the application accessible to users.

## Frontend (AngularJS)

- Build production-ready files using:
- `ng build --prod`
- Deploy to:
  - **Netlify:** Drag and drop the `dist/` folder or use CLI.
  - **Vercel:** Use Git integration and set build settings.
  - **Firebase Hosting** (optional): for simple deployment.

## Backend (SQL + API)

- Host using:
  - **Render, Railway, Heroku, or VPS/cloud server (AWS, GCP).**
  - Ensure SQL database (MySQL/PostgreSQL) is connected and accessible.

## Environment Configuration

- Use `.env` files for API keys, database credentials.
- Ensure **build & deployment scripts** are in place (CI/CD optional)