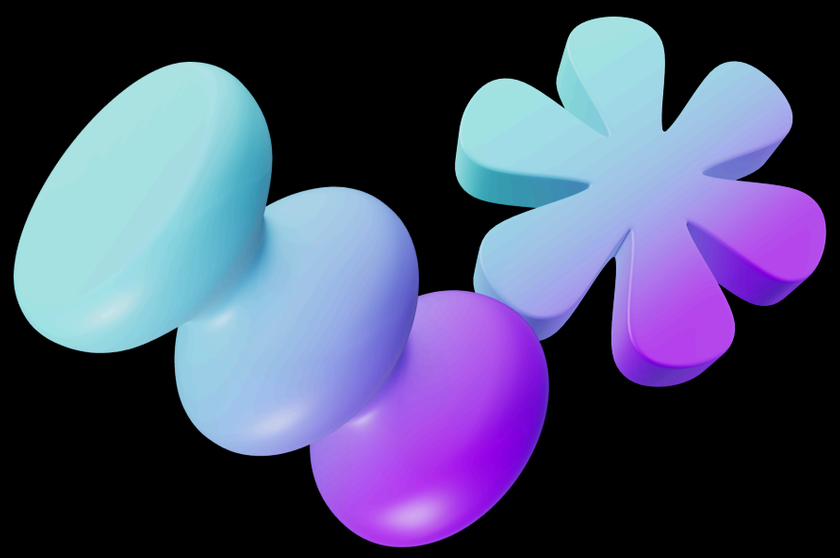


PERSONAL NOTES ORGRANISER

READ MORE

PROJECT OVERVIEW

Personal Notes Organiser is a full-stack web application designed to help users create, manage, and organize personal notes efficiently using a scalable backend and a modern, responsive frontend. The application supports complete CRUD (Create, Read, Update, Delete) operations and ensures seamless interaction between the frontend and backend through RESTful APIs.

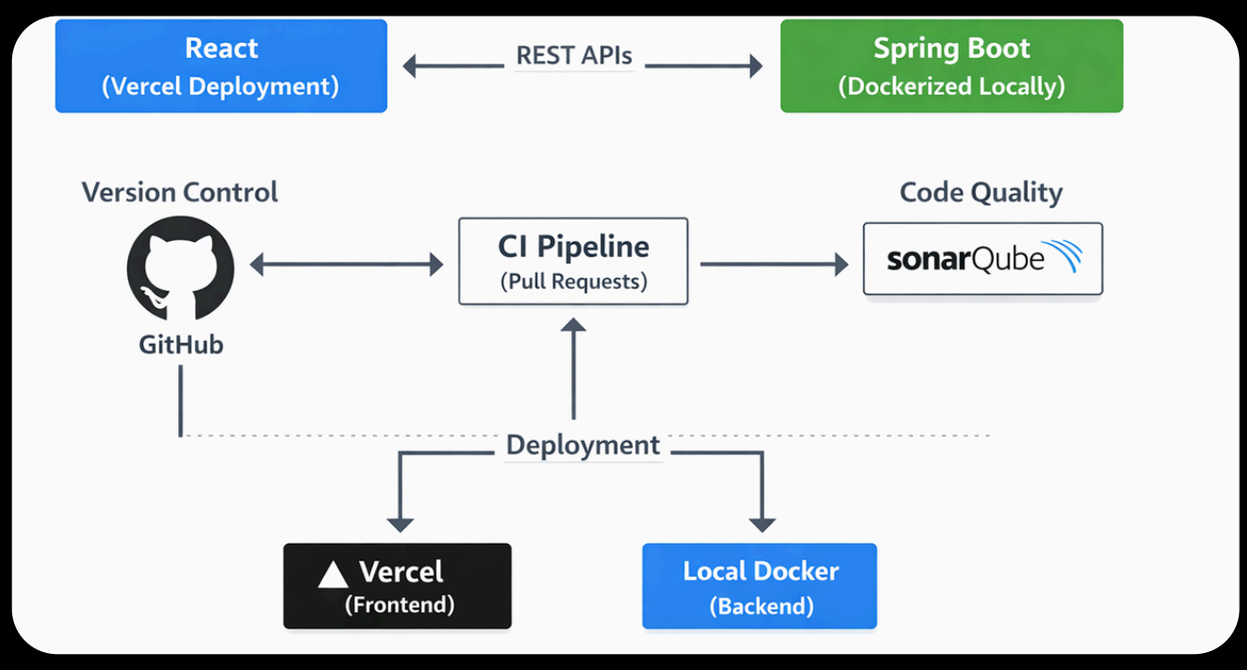
OBJECTIVE

- Modular backend architecture
- Responsive frontend
- DevOps best practices
- Code quality and automation

System Architecture

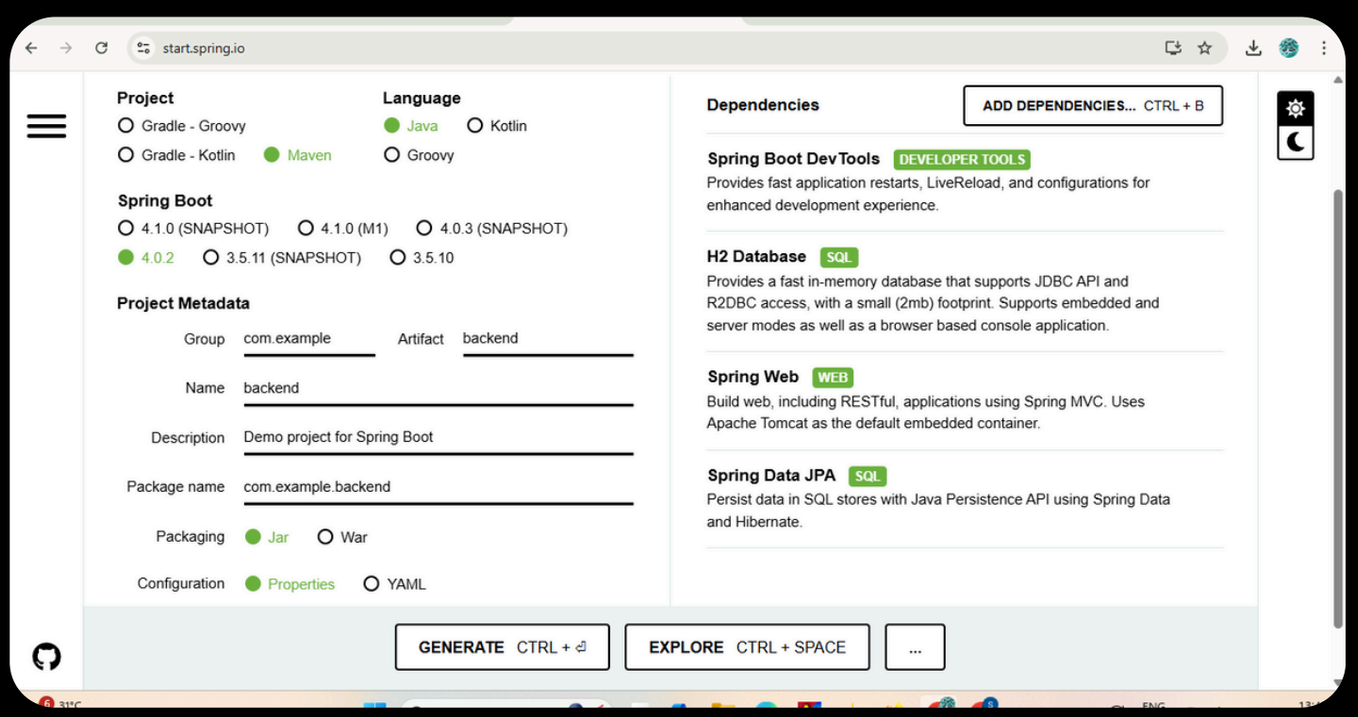
Architecture Flow:

React frontend communicates with Spring Boot backend via REST APIs GitHub used for version control Pull Request–based CI pipeline SonarQube for code quality checks Backend dockerized locally Frontend deployed on Vercel



Backend Development

- Project initialized using Spring
- Initializr Maven-based Spring
- Boot application Layered
- architecture: Controller
- Service Repository Entity REST
- APIs using JSON

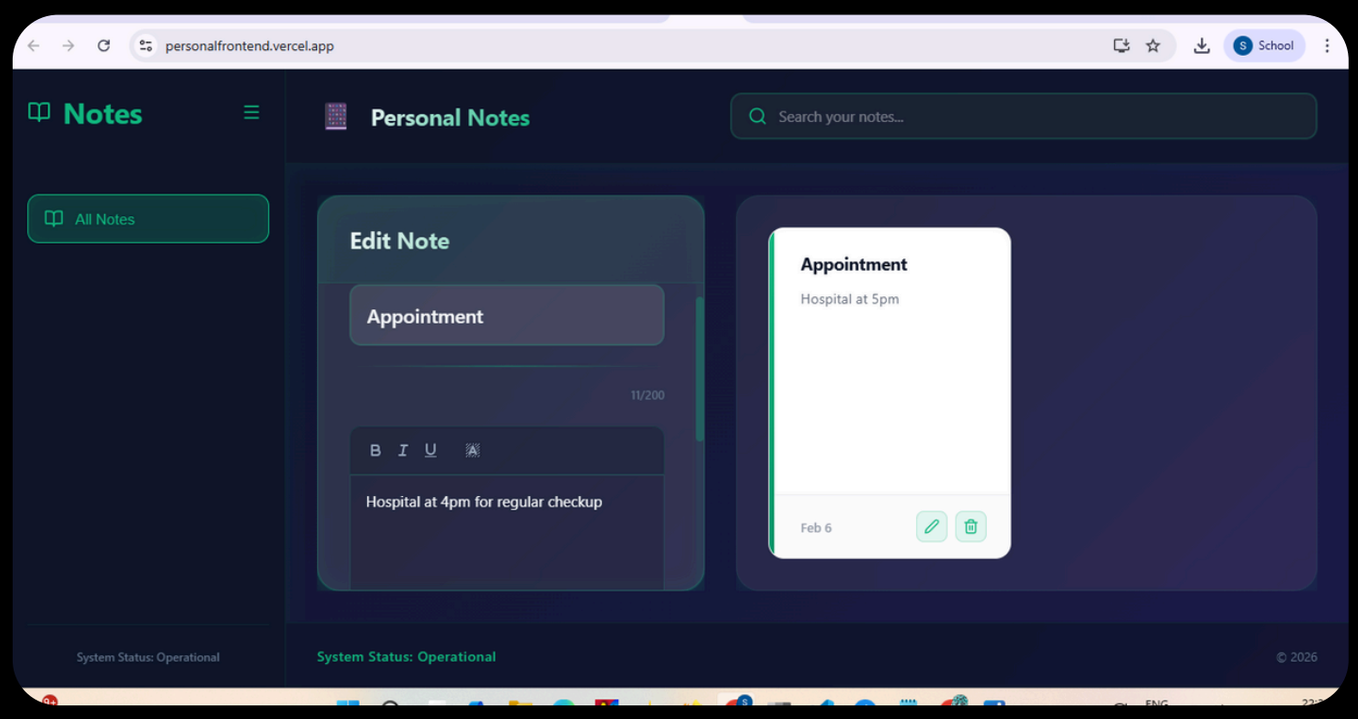


Backend Execution

- Backend started using mvn spring-boot:run
- Spring Boot runs on http://localhost:8080
- Root URL (/) shows Whitelabel Error Page (no mapping)
- REST APIs accessed using browser
- Example API endpoint: /noteservice

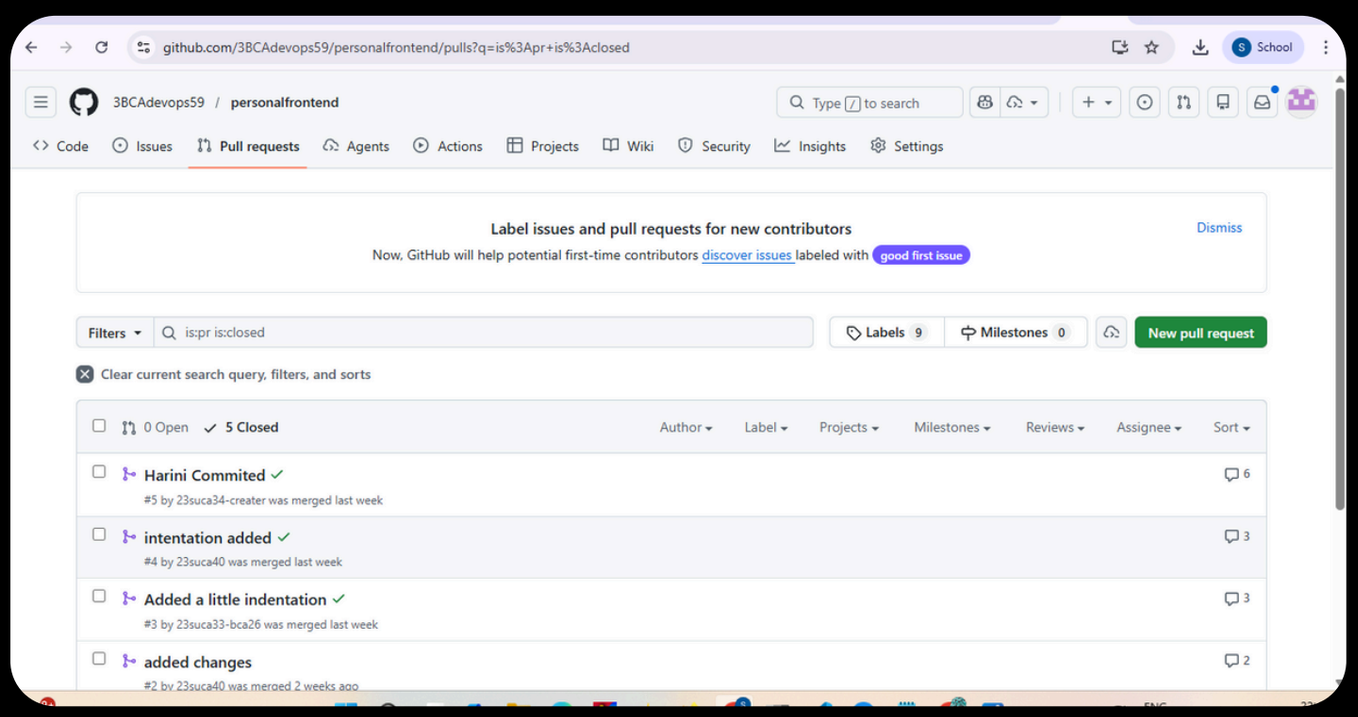
Frontend Development

- Frontend developed using React.js
- Uses component-based architecture for modular design
- Integrates backend APIs using Axios / Fetch
- Local execution using npm start
- Runs on http://localhost:3000
- Source code maintained in GitHub repository (CRM Frontend)
- Integrated into CI pipeline for build and deployment
- Part of DevOps internal project presentation



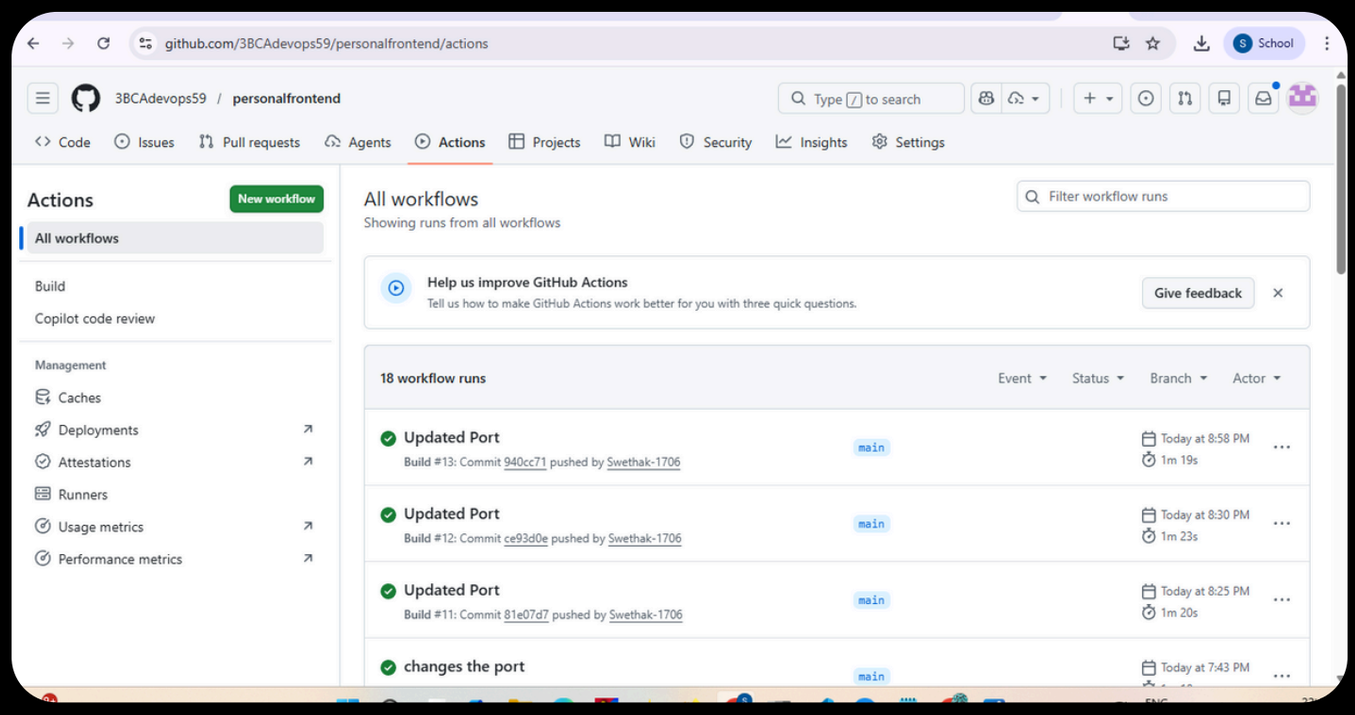
GitHub-Based Development Workflow

- GIT USED FOR SOURCE CODE VERSION CONTROL
- GITHUB REPOSITORY WITH BRANCH-BASED WORKFLOW
- FEATURE DEVELOPMENT DONE IN SEPARATE BRANCHES
- PULL REQUESTS (PRS) USED TO MERGE CHANGES
- CI PIPELINE TRIGGERED AUTOMATICALLY ON PR CREATION
- AUTOMATED BUILD AND VALIDATION EXECUTED IN CI



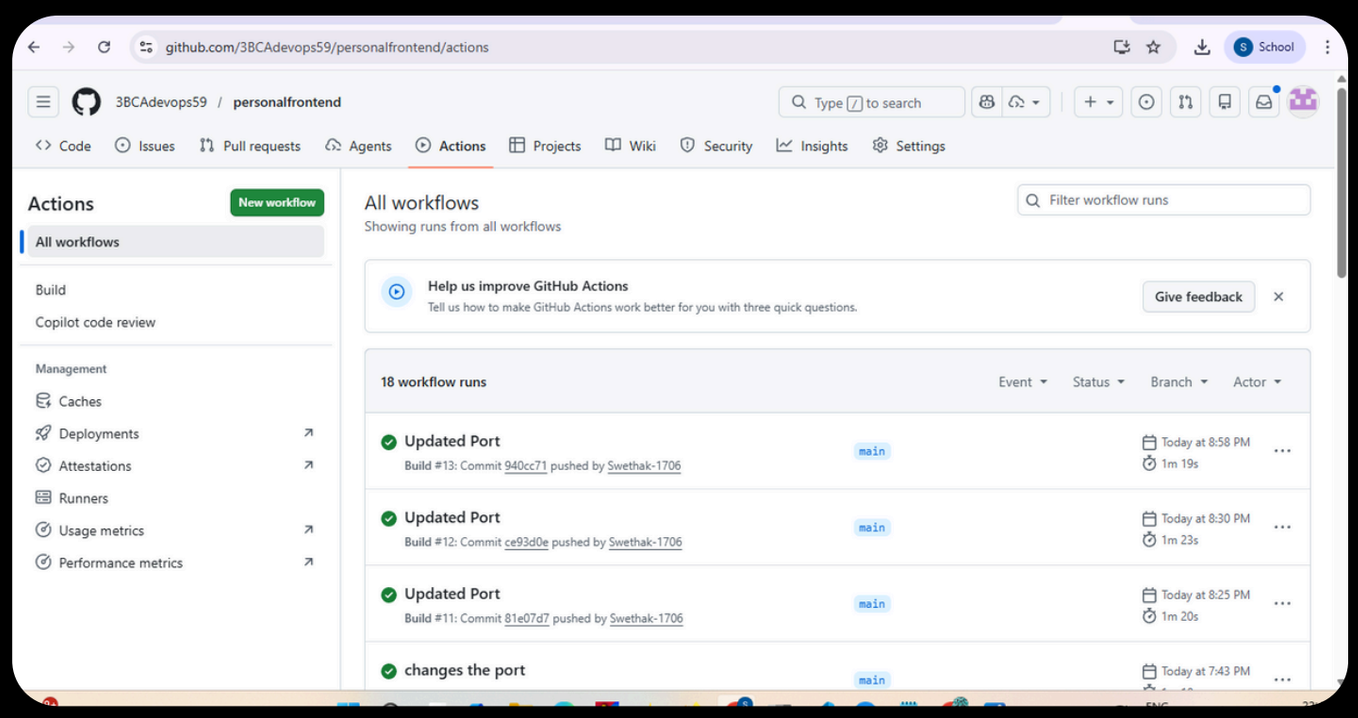
CI & SonarQube Integration

- SonarQube is integrated into the CI pipeline to automatically analyze code quality during builds
- Static code analysis is executed for every Pull Request before merging into the main branch
- The Quality Gate must be successfully passed to allow the Pull Request to be merged
- Bugs are detected to identify functional and logical errors in the code
- Code smells are identified to highlight poor design and maintainability issues



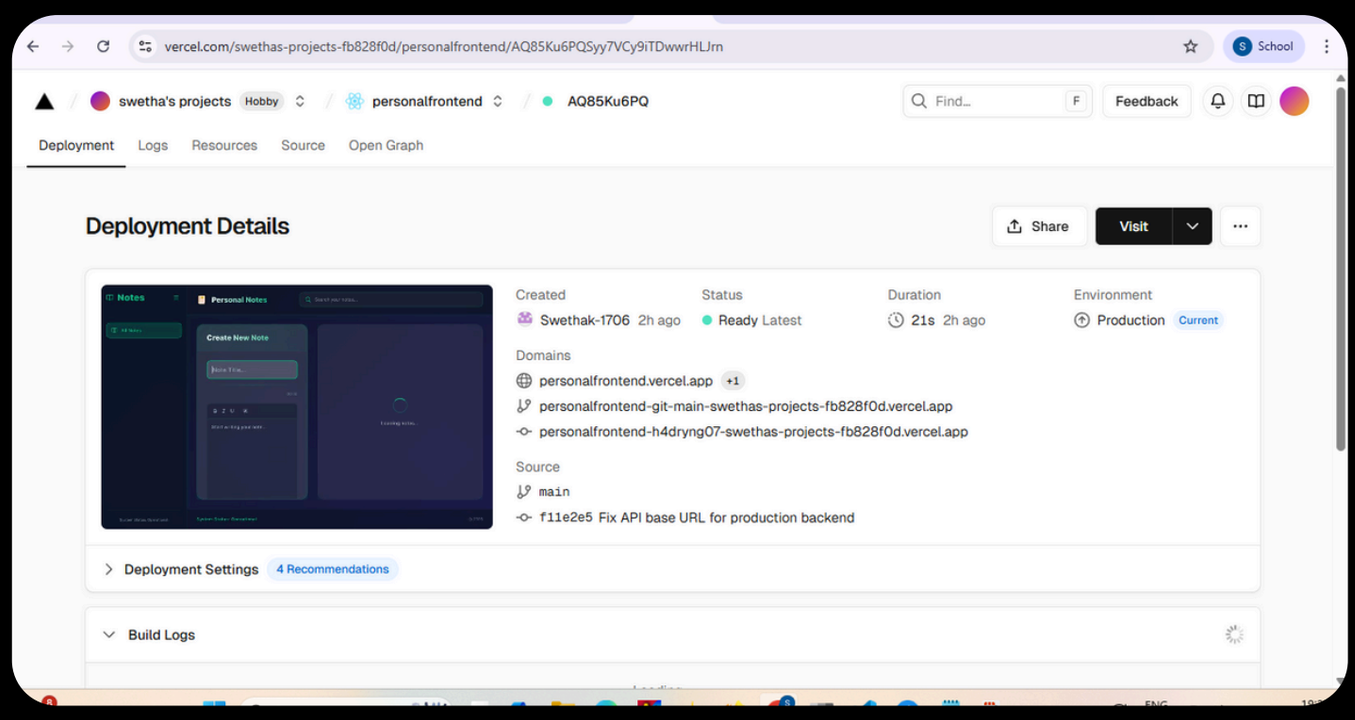
Backend Dockerization

- Spring Boot backend developed for managing personal notes using CRUD operations
- Backend application dockerized using a Dockerfile with OpenJDK base image
- Docker container used to ensure consistent runtime environment and easy deployment
- Backend successfully built and tested inside Docker container using mapped ports
- Docker used for local testing before deploying the backend to cloud (Render)



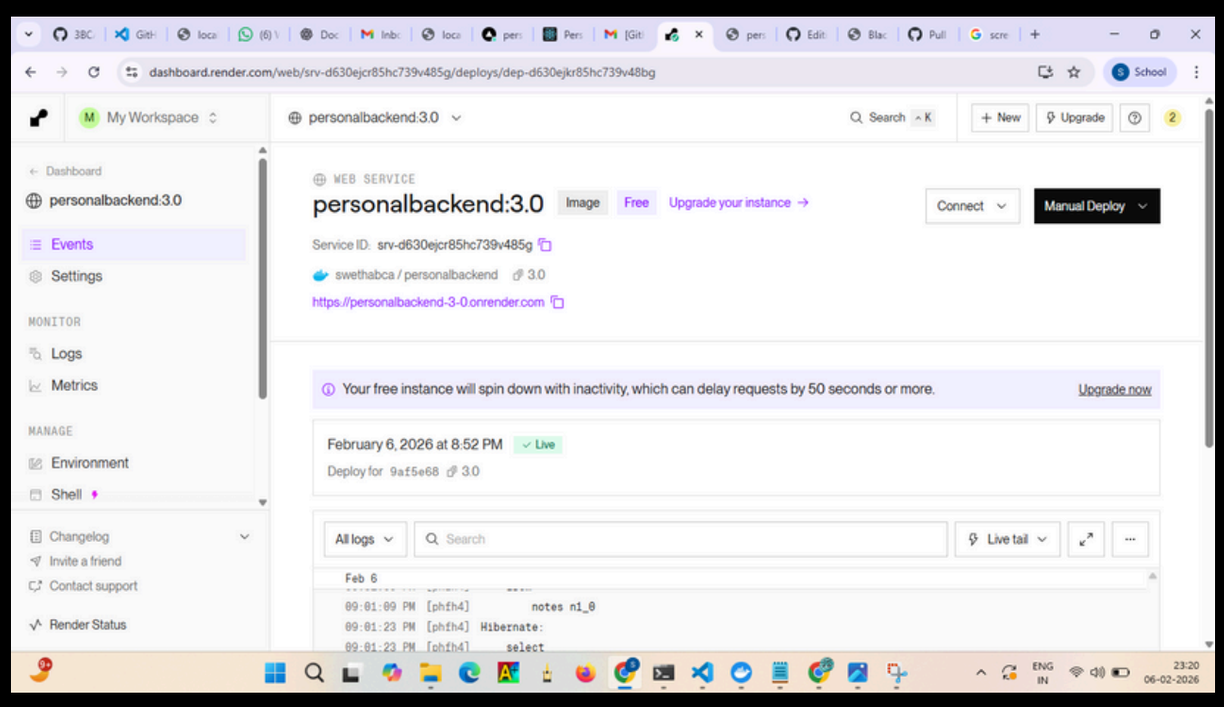
Frontend Deployment

- React-based frontend developed for the Personal Notes Organiser application
- Frontend deployed using Vercel for fast and reliable hosting
- GitHub repository connected to Vercel for automatic build and deployment
- Environment variables configured for backend API integration
- Live frontend accessible through a public Vercel deployment URL



Backend Deployment

- Spring Boot backend developed to handle CRUD operations for personal notes
- Backend dockerized and deployed using Render cloud platform
- GitHub repository connected to Render for automatic build and deployment
- Backend service exposed through a public REST API endpoint
- Live backend accessible via Render deployment URL and integrated with frontend



Issues Faced & Solutions

- Docker daemon and port binding issues during backend containerization
  → Resolved by properly starting Docker Desktop, freeing occupied ports, and remapping container ports

- Container name conflicts while running multiple backend versions
  → Fixed by stopping and removing existing containers before redeployment

- Frontend–backend integration issues after cloud deployment
  → Solved by configuring backend API URL through environment variables in Vercel

- GitHub Pull Request conflicts and branch management issues
  → Resolved by creating separate feature branches, committing changes correctly, and merging via pull requests

- SonarQube code quality issues (code smells and warnings)
  → Fixed by refactoring code, improving naming conventions, removing unused variables, and following best practices


THANK YOU