DHIVYADHARSHINI K

COURIER TRACKING

COURIERTRACKING.VERCEL.APP

PROJECT OVERVIEW

The Courier Tracking System is a full-stack web application designed to manage and track courier parcels efficiently throughout their delivery lifecycle. It provides a structured and scalable solution for creating, updating, monitoring, and managing parcel information in real time.The system is built using a Spring Boot backend and a React-based frontend, following a layered architecture and RESTful communication principles.

🎯OBJECTIVES

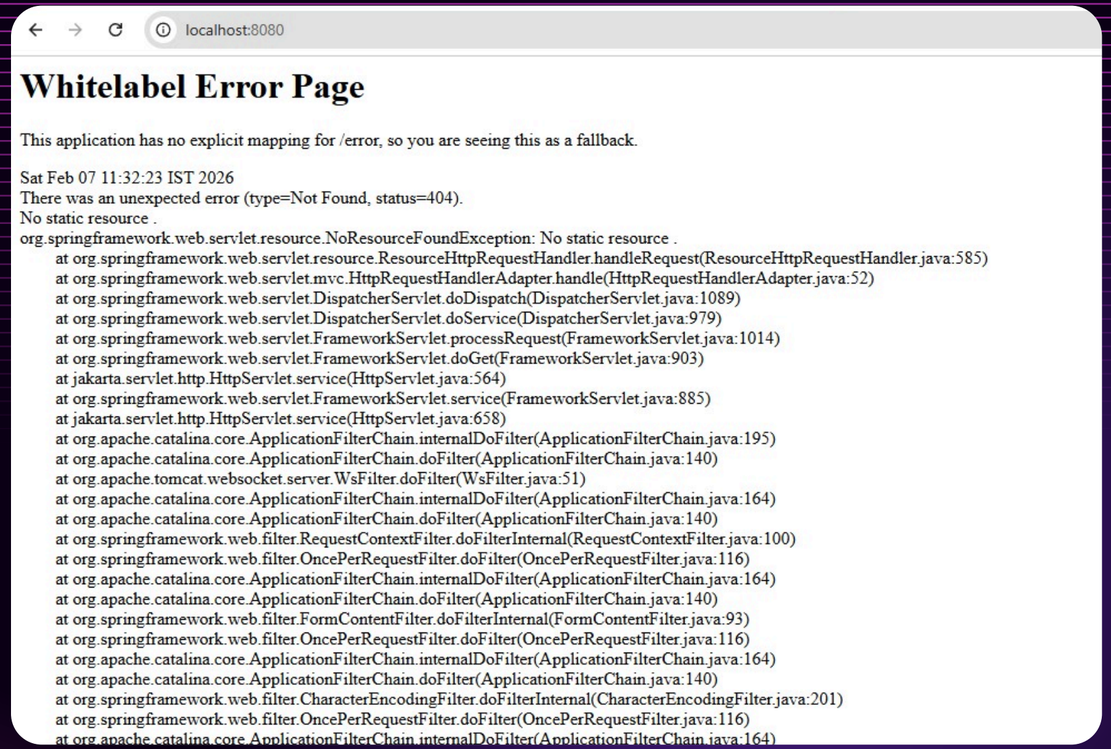
- To provide a centralized system for courier parcel management
- To enable real-time parcel status tracking
- To ensure clean separation of frontend and backend responsibilities
- To implement a scalable and maintainable architecture
- To demonstrate modern DevOps and deployment practices

🏗️ 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

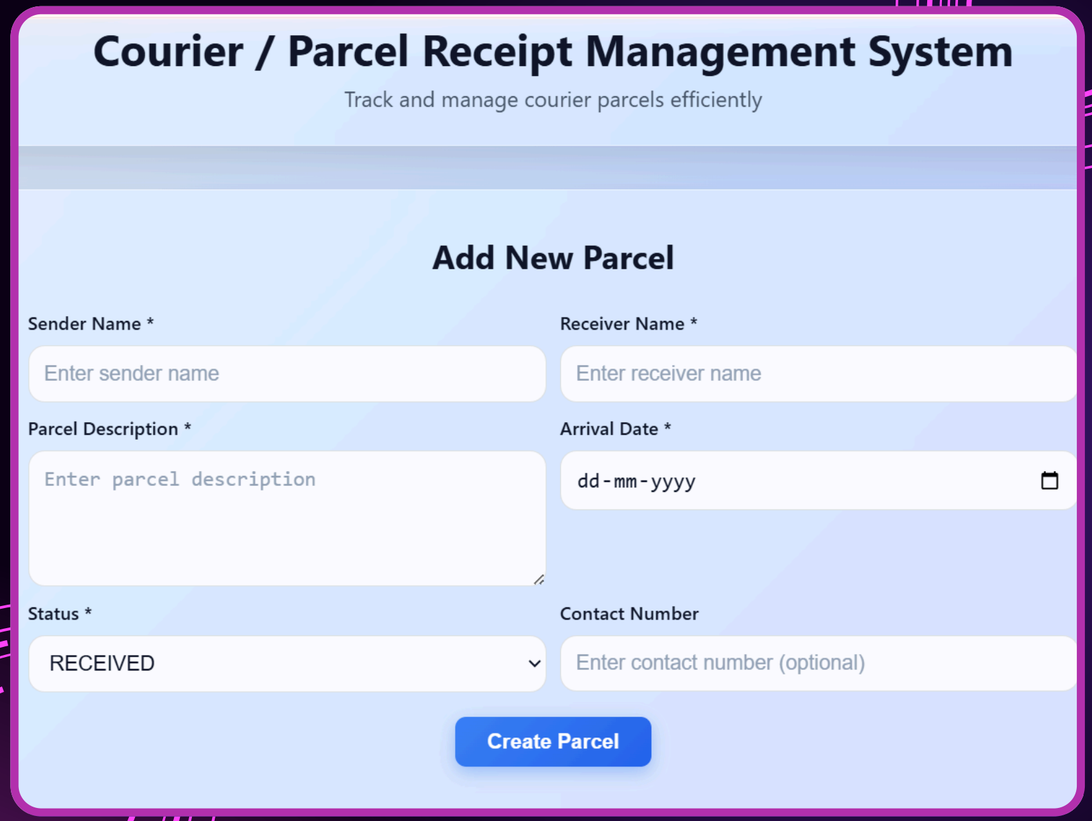


BACKEND EXECUTION

- Application executed using: mvn spring-boot:run
- Backend accessible at: http://localhost:8080
- APIs tested using browser For Example : /api/parcels

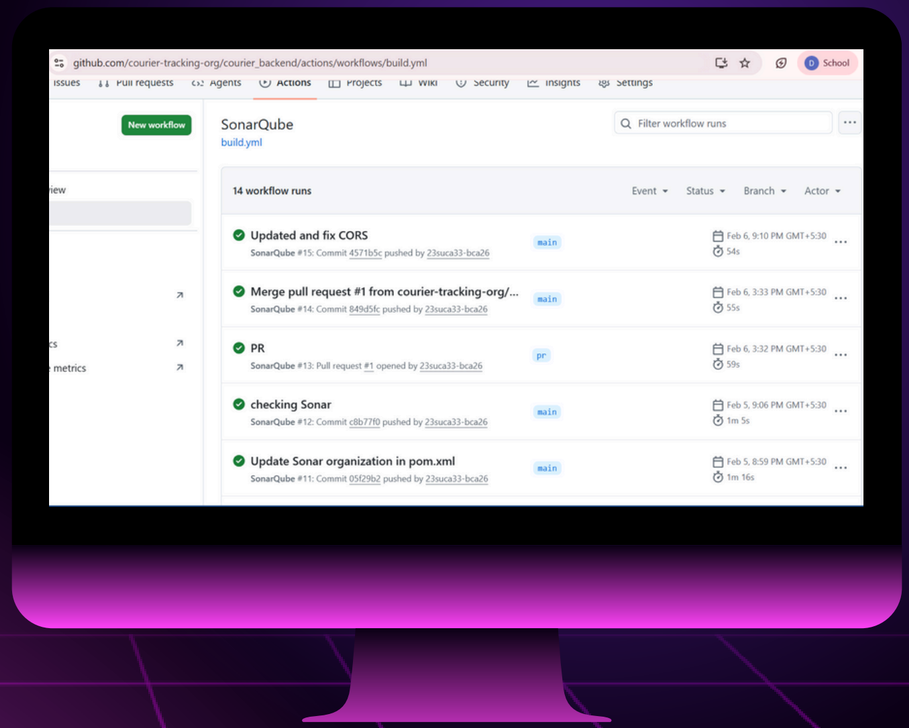
FRONTEND DEVELOPMENT

- Frontend developed using React
- Component-based architecture
- API integration using Axios / Fetch
- Local execution: npm start
- Runs on: http://localhost:3000



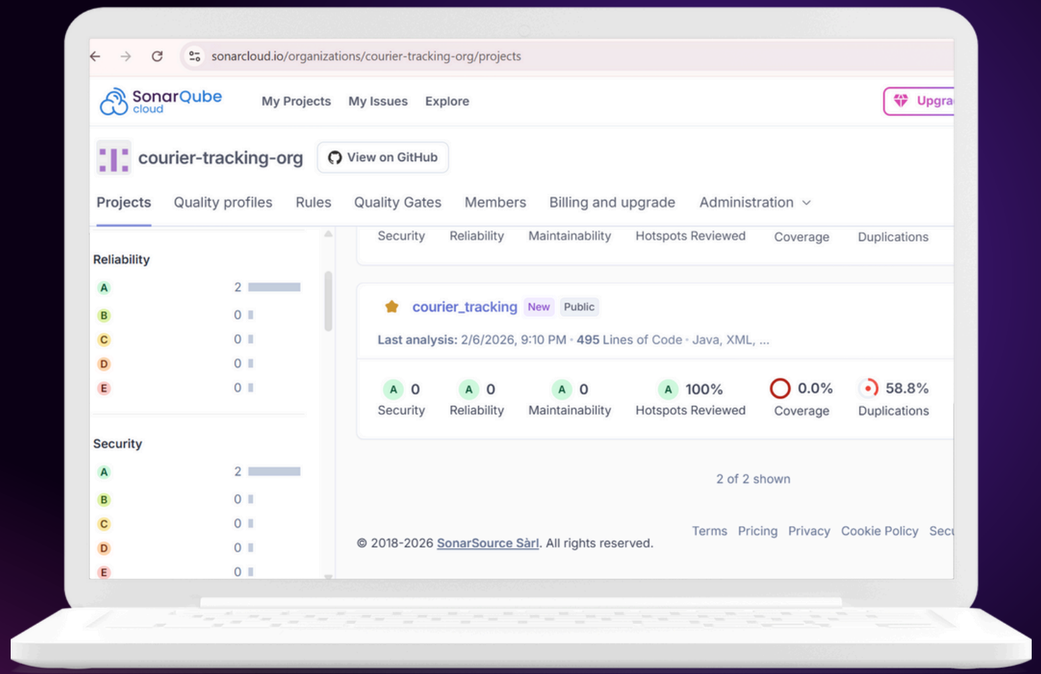
VERSION CONTROL & CI PIPELINE

- The Courier Tracking System uses Git and GitHub for version control to manage and track source code changes.
- All development is maintained on the main branch for stable and production-ready builds.
- A CI pipeline is triggered automatically on every push to the repository.
- The frontend is auto-built and deployed using Vercel with global CDN support.
- The backend is built using Maven and Docker to ensure consistent execution.
- SonarCloud is integrated to monitor code quality and maintain best practices.



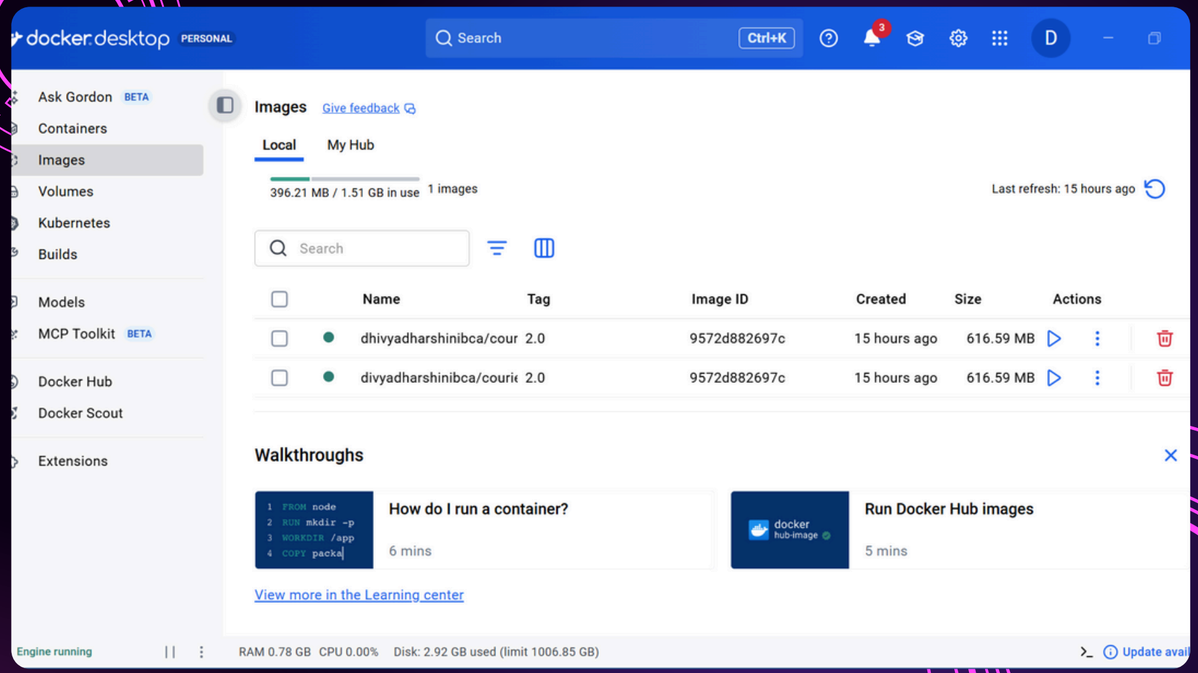
CI & SONARQUBE INTEGRATION

- Continuous Integration (CI) is implemented using GitHub Actions to automate the build process.
- Every push or pull request triggers the CI workflow automatically.
- The backend is built using Maven with Java 17 inside the pipeline.
- SonarCloud (SonarQube) is integrated to perform static code analysis.
- It checks for code smells, bugs, vulnerabilities, and maintainability issues.
- This ensures high-quality, secure, and maintainable code throughout development.



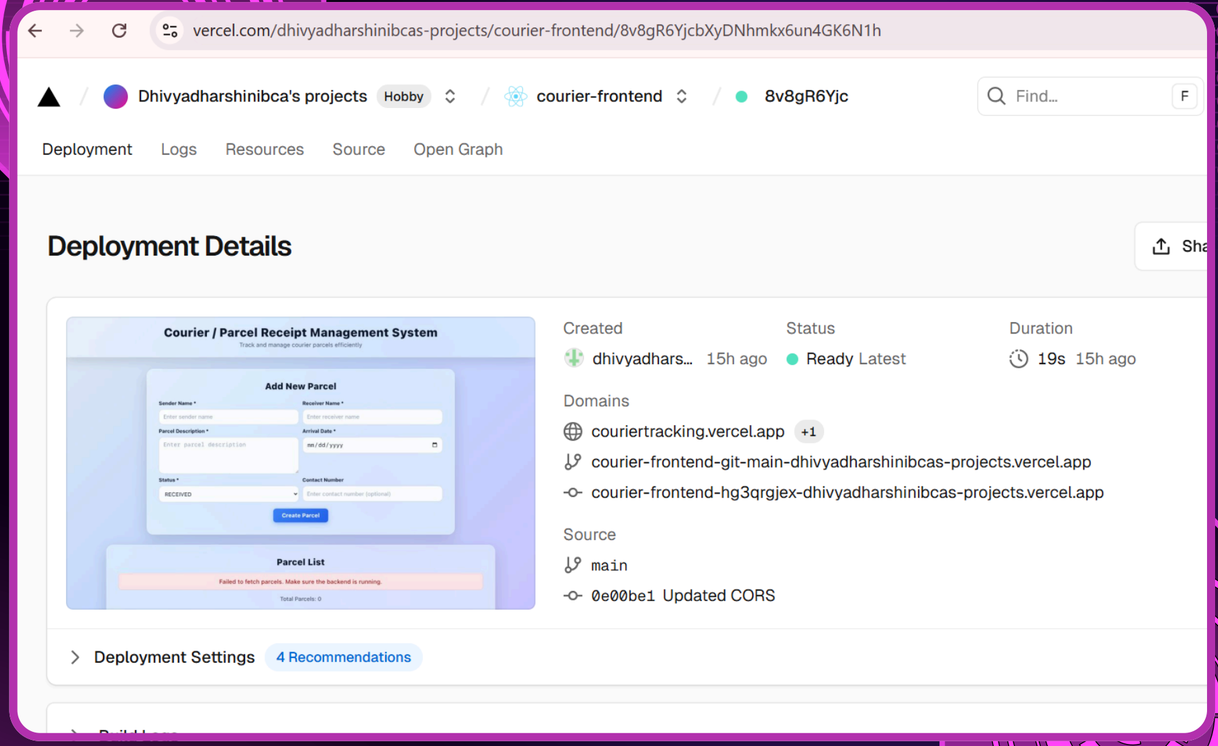
BACKEND DOCKERIZATION

- Spring Boot backend dockerized for local testing
- Dockerfile created using OpenJDK base image
- Backend verified inside Docker container
- Docker commands used
- docker build -t courier_backend:2.0
- docker run -d -p 8006:8006 --name courier_backend_container courier_backend:2.0



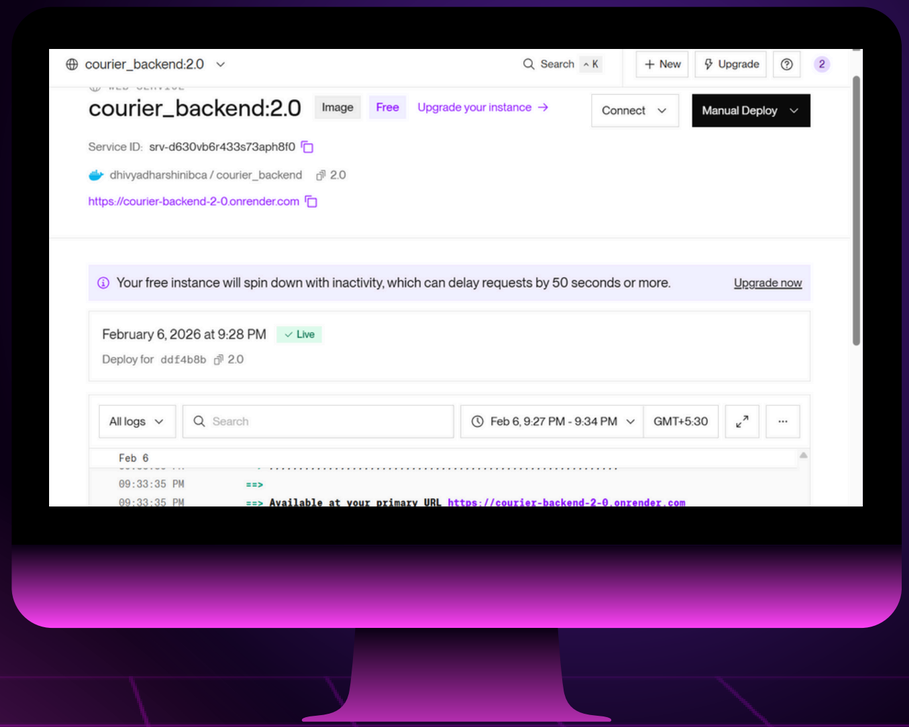
FRONTEND DEPLOYMENT

- The frontend application is deployed using Vercel.
- The GitHub repository is directly connected to Vercel.
- Each push to the main branch automatically triggers a new build.
- Vercel handles build, deployment, and hosting seamlessly.
- The application is served through Vercel's global CDN.
- The live frontend is accessible via a public Vercel URL.



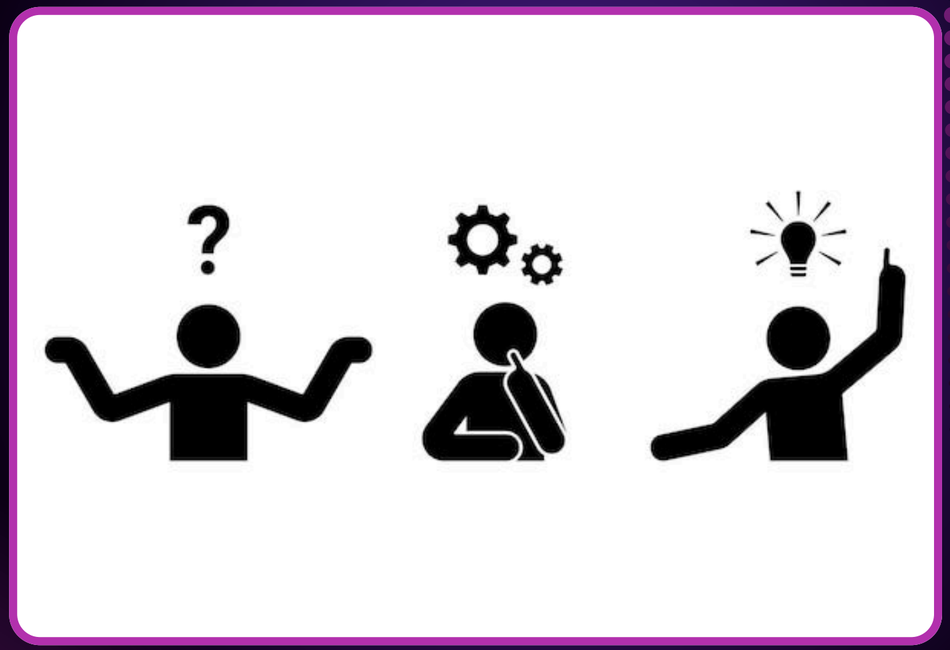
BACKEND DEPLOYMENT

- The backend application is deployed using Render cloud platform.
- The Spring Boot GitHub repository is connected to Render.
- Each commit to the main branch automatically triggers deployment.
- Render builds the project using Maven and Java 17.
- The backend is accessible through a public API endpoint.
- Application logs and status are monitored via the Render dashboard.



ISSUES FACED & SOLUTIONS

- Faced port conflicts while running the backend locally; resolved by configuring a custom server port.
- Encountered Maven not found errors in some environments; fixed using the Maven Wrapper.
- CORS issues occurred during frontend-backend integration; resolved by enabling CORS in Spring Boot.
- Experienced Docker build failures due to Java version mismatch; solved by using Java 17 base images.
- Frontend showed no data when backend was down; handled by validating backend availability and API URLs.
- CI pipeline initially failed SonarQube checks; fixed by correcting project keys and configuration files.



THANK YOU!

courier Tracking

couriertracking.vercel.app

github.com/courier-tracking-org