Scenario 2 – Optimize docker image for production

As the DevOps engineer at a tech startup, you're preparing a todo management application for deployment. The application has grown complex, with a Java backend and a React frontend, each requiring its own build process. To ensure smooth and efficient deployments, your task is to streamline the Docker build process using multi-stage builds. By separating the stages for building, testing, and packaging, you'll create a reliable production-ready image that meets the company's high standards for quality and performance.

Requirements

You are tasked with refactoring the existing Dockerfile to use multi-stage builds for both the backend and frontend components. This will include setting up separate stages for building, testing, and packaging the application, ensuring that the final Docker image is optimized for production use.

The application is a todo management system consisting of a **Java-based REST API backend** and a **React-based frontend**. It utilizes environment variables for database connectivity and requires proper configuration to run successfully.

Important: The project's README contains crucial information about the project structure, environment variables, and build steps. **You must read the README carefully before starting this exercise.** It provides detailed instructions on how to set up the environment, build the application, and run the backend and frontend services. Skipping this step may result in confusion or errors during the exercise.

- **Environment Variables**: The project requires specific environment variables (DB_URL, DB_USERNAME, DB_PASSWORD) for database connectivity.
- **Build Instructions**: The application is built using Maven for the backend and npm for the frontend. There are specific steps to package and run the application, including how to serve the frontend through the backend.
- **Testing**: The application includes unit tests that must pass before creating the final Docker image.

Refer to the README for detailed instructions on all of the above.

Resources

- Docker Multi-Stage Builds
- Dockerfile Reference
- Best Practices for Writing Dockerfiles
- Project GitHub Repository: Docker Mastery. **Please use *MULTI-STAGE-BUILD* branch for this exercise.**

Acceptance Criteria

1. **Multi-Stage Dockerfile Created**: The Dockerfile must be refactored to use multi-stage builds for both the backend and frontend.
2. **Tests Passed**: The application must pass all tests in the dedicated test stage before the final image is created.
3. **Optimized Final Image**: The final Docker image must be optimized for size and contain only the necessary components to run the application.
4. **Application Runs**: The application must run correctly in the final Docker container, with both the backend API and frontend UI accessible on port 8080.

Hints

- **Define Multiple Stages**: Create separate stages in your Dockerfile for building the backend, building the frontend, running tests, and packaging the final production image.
- **Use Minimal Base Images**: In the production stage, use a minimal base image like *eclipse-temurin:21-jre-alpine* for the backend and for serving the frontend.
- **Test and Verify**: Ensure that all stages are correctly implemented and that the final container runs both the backend and frontend components as expected.