Assignment Brief: .NET URL Shortener Service with CI/CD

Overview

A URL shortener is a tool that converts long URLs into more manageable, shorter versions. This is useful for sharing links on platforms with character limits or improving user experience by reducing clutter. Two popular URL shorteners are Bitly and TinyURL.

In this assignment, you will design and implement a URL shortener using .NET Core. You will learn how to apply the principles of system design and API design to create a scalable, performant, and user-friendly application. A key focus will be on implementing a professional Continuous Integration and Continuous Deployment (CI/CD) pipeline to automate the entire release process.

Scenario

You are a team of software engineers (3 members/team maximum) working for a startup company that wants to launch a new URL-shortening service. Your task is to design and develop the backend system and API for the service. The company follows modern DevOps principles, and a core requirement is that the entire process from code commit to production deployment must be fully automated, reliable, and cloud-native.

Requirements

Application Features:

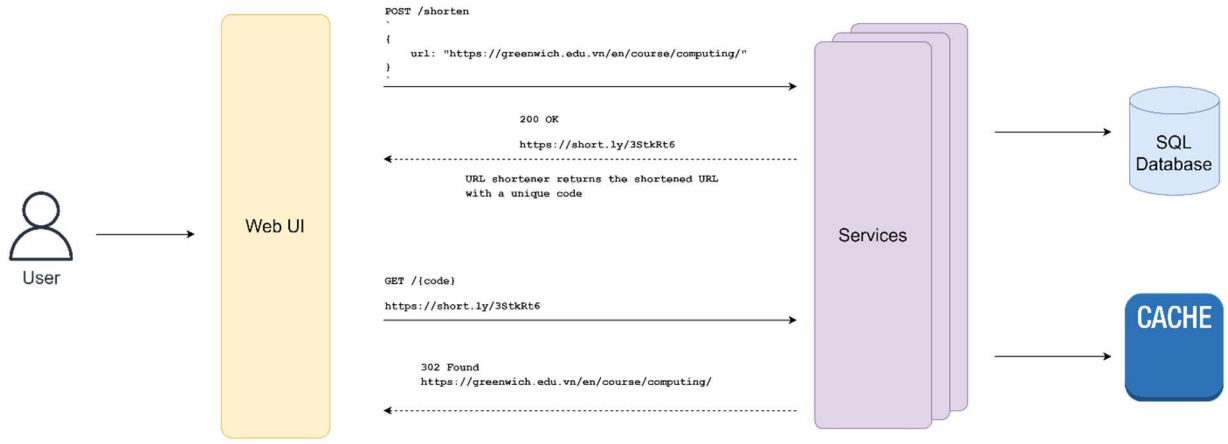
- Generate a unique code for a given URL.
- Redirect users who access the short link to the original URL.
- Validate the input URL and handle errors gracefully.
- Store the shortened URLs and their metadata in a database.
- Expose a RESTful API for creating and retrieving shortened URLs.
- Create a simple front-end web application (Vue/React) that allows users to interact with your service.

DevOps & Deployment Features:

- The application **must be containerized** using a Dockerfile.

- A complete **CI/CD pipeline** must be implemented using a platform like GitHub Actions.
- The pipeline must automatically **build** the Docker image on every push to the main branch.
- Upon successful testing, the pipeline must **push the container image** to a public registry (e.g., Docker Hub) as a versioned artifact.
- The final step of the pipeline must **automatically deploy** the application to a PaaS platform (e.g., Render) by triggering a deploy hook.

You can consider the following architecture for the starting point



Deliverables

You will submit the following deliverables:

- **Group Presentation (70%):** You will present your system design and implementation to the class. You will explain the architecture, technologies, and trade-offs of your solution. Your presentation should include slides, source code, and critically, a live demonstration and detailed explanation of your automated CI/CD pipeline, from code push to live deployment.
- **Individual Report (30%):** You will write a self-evaluation report of the project. You will reflect on your contribution, challenges, and learning outcomes. You will also provide feedback on your team members and the assignment.

- **Source Code:** Your full source code must be submitted in a Git repository, including the application code, Dockerfile, and all CI/CD configuration files (e.g., .github/workflows/*.yml).

Assessment

You will be assessed based on the following criteria:

Mark	Description
Pass: 5 – 6.5 points	You have completed the basic design and implementation of the URL shortener service. Your application is **fully containerized and deployed via an automated CI/CD pipeline** that builds, tests, pushes the image, and deploys the service. Your application must contain Unit Testing and a simple web UI (Vue/React) to interact with your system. Your presentation and report are clear and concise.
Merit: 7 – 8.5 points	You have added caching to your service to improve read performance using a distributed cache like Redis. You have also **optimized your deployment process**, for example, by using **multi-stage Docker builds** to create smaller production images or by adding a **linting/static analysis step** to your CI pipeline. You have explained the benefits of these optimizations in your presentation and report. Your project must have Integration Testing.

Distinction: 9 – 10 points	You have implemented a microservices architecture for your service, separating the users' service from the URL shortening service. Crucially, you have implemented **independent CI/CD pipelines for each microservice**, demonstrating true independent deployability. You have used a message broker like RabbitMQ to communicate between the services. You have discussed the advantages and challenges of both microservices and managing multiple pipelines in your presentation and report.