URL Shortener WebService

DevOps Implementation & Automation Project

Team Member:

1. Omar Mohsen
2. Marwa El-Zoghby
3. Marwan Ayman
4. Sara Darwish
5. Ahmed Anas

Table Of Contents

1. Project Planning & Management

1.1. Project Proposal – Overview, Objectives, Scope

1.1.1. Project Overview:
This project implements a fully functional URL Shortener Web Service built using FastAPI, SQLite, Docker, Prometheus, and Grafana. The system allows users to shorten long URLs, store them in a persistent database, and use short codes for redirection. The project also focuses heavily on DevOps practices such as containerization, service orchestration, monitoring, visualization, and metrics exposure**.**

1.1.2. Objectives**:**

1.1.2.1.
1.1.2.2. Build a working URL shortener service with REST APIs.
1.1.2.3. Containerize the backend, database, Prometheus, and Grafana.
1.1.2.4. Expose application metrics for monitoring and analysis.
1.1.2.5. Create Grafana dashboards to visualize system performance.
1.1.2.6. Demonstrate DevOps principles (observability, logging, metrics, dashboards).

1.1.3. Scope:

1.1.3.1. URL shorten + redirect functionality
1.1.3.2. Persistent SQLite storage
1.1.3.3. Custom Prometheus metrics
1.1.3.4. Grafana visualization
1.1.3.5. Docker Compose orchestration

1.2. Project Plan

1.2.1. Week 1: Build app + Docker
1.2.2. Week 2: Prometheus integration
1.2.3. Week 3: Grafana Dashboard
1.2.4. Week 4: Alerts + Persistence + Docs

1.3. Task Assignment & Roles

Team Member	Responsibilities
Omar Mohsen	Backend & API Development, Containerization, Provisioning Grafana, Prometheus
Ahmed Anas	CD Pipeline & Cloud Deployment & Kubernetes implementation
Marwa Elzoghby	Monitoring & Visualization integration using Prometheus & Grafana
Sara Darwish	Frontend & Alerting Integration
Marwan Ayman	Database Schema Design, CI pipeline Sanity test, Project documentation

1.4. Risk Assessment & Mitigation

Risk	Impact	Mitigation
Database loss	Medium	Use Docker volumes
Prometheus is not scraping metrics	High	Validate /metrics endpoint
Code errors during build	Medium	CI linting + testing
Security concerns	Medium	Validate URLs, use HTTP codes

1.5. KPIs

1.5.1. API Response Time: < 200ms
1.5.2. Redirect success rate: ≥ 98%
1.5.3. Metrics coverage: 100% Prometheus scraping
1.5.4. Dashboard completeness: 100% monitoring widgets

2. Literature Review

The literature review examines previous work, existing technologies, and established practices related to URL shortening systems, web service design, containerized application development, and monitoring solutions. This review

provides the theoretical foundation upon which the project design and implementation were built

3. Requirements Gathering

3.1. Stakeholder Analysis

Stakeholder	Needs
End User	Shorten URLs easily
Admin	View and manage stored URLs
Developers	A maintainable, scalable codebase
DevOps Engineers	Observable system with metrics

3.2. User Stories

3.2.1. As a user, I want to enter a long URL and get a shortened version.

3.2.2. As a user, I want to click the short link and be redirected immediately.

3.2.3. As an admin, I want to view all stored URLs.

3.2.4. As a DevOps engineer, I want to monitor system metrics.

3.3. Functional Requirements

3.3.1. Shorten any valid URL.

3.3.2. Store URLs in SQLite.

3.3.3. Redirect short codes to original URLs.

3.3.4. Expose metrics at `/metrics`.

3.4. Non-Functional Requirements

3.4.1. **Performance:** Fast responses (<200ms).

3.4.2. **Usability:** Simple UI, dark mode.

3.4.3. **Reliability:** Persistent storage + Docker volumes.

3.4.4. **Monitoring:** Prometheus integration.

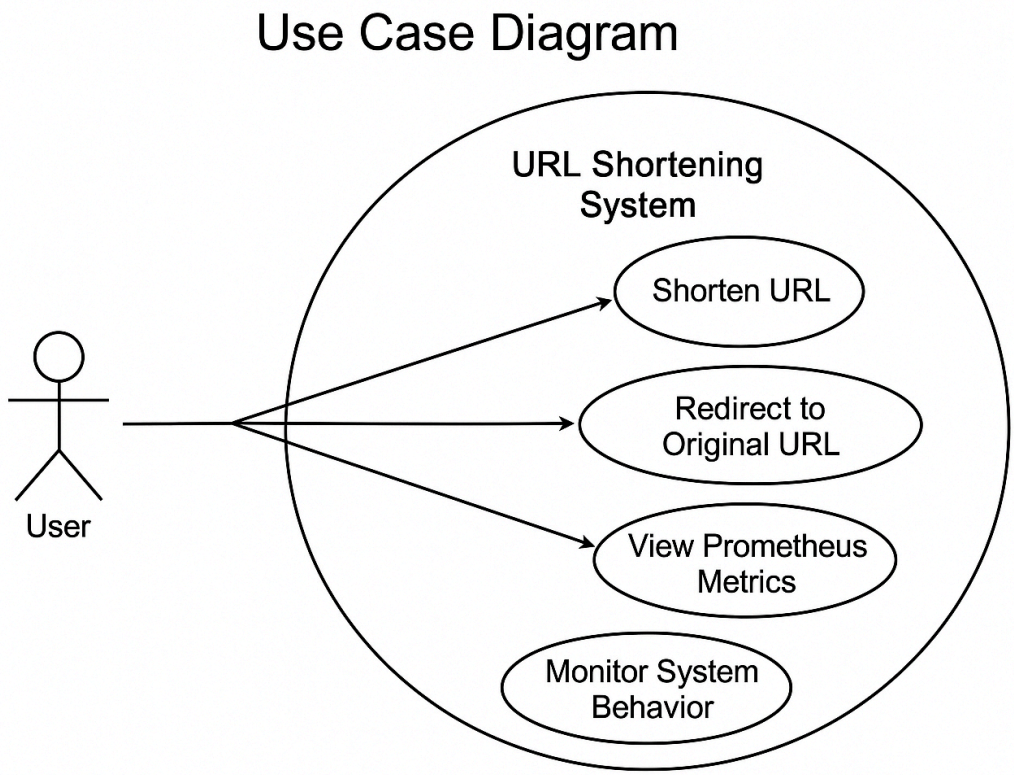
3.4.5. **Portability:** Docker Compose environment.

4. System Analysis & Design

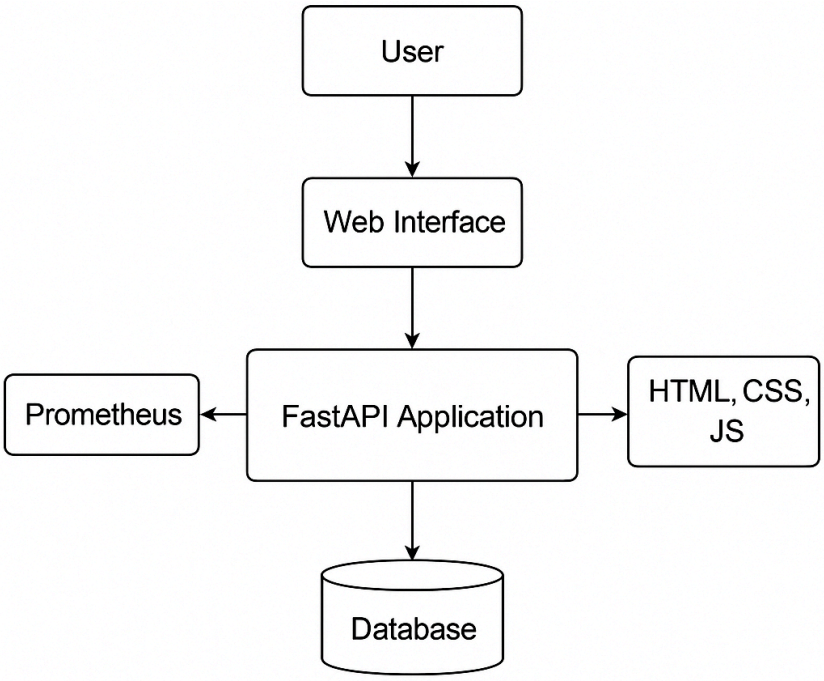
4.1. Problem Statement

Long URLs are difficult to share and manage. This project provides a lightweight web service to shorten URLs while monitoring performance using DevOps tools.

4.2. Use Case Diagram



4.3. Software Architecture



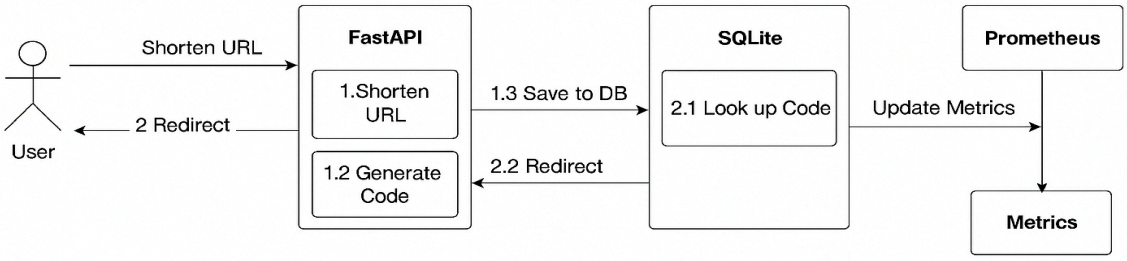
Software Architecture

4.4. Database Design (ERD)

URL		
id	Integer	PK
short_code	String(10)	
long_url	String(512)	
clicks	Integer	
created_at	DateTime	

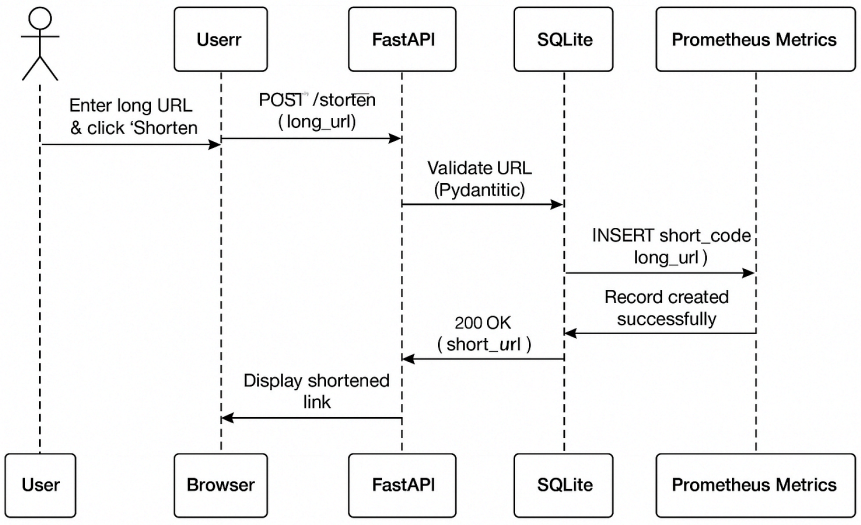
4.5. Data Flow & System Behavior

4.5.1. Data Flow Diagrams

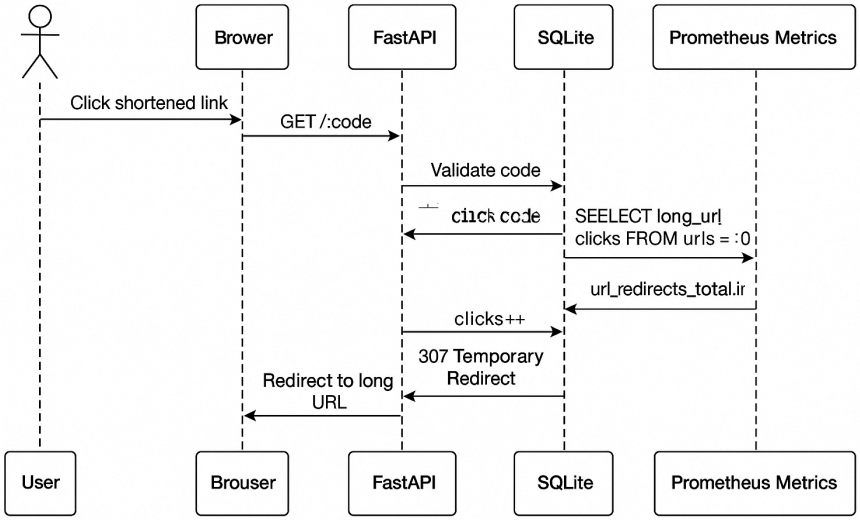


4.5.2. Sequence Diagram

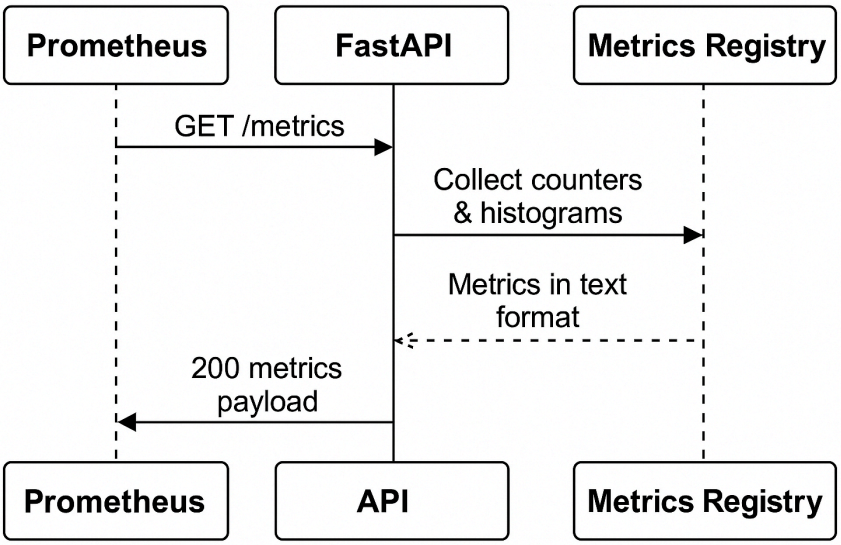
4.5.2.1. Shorten URL Sequence



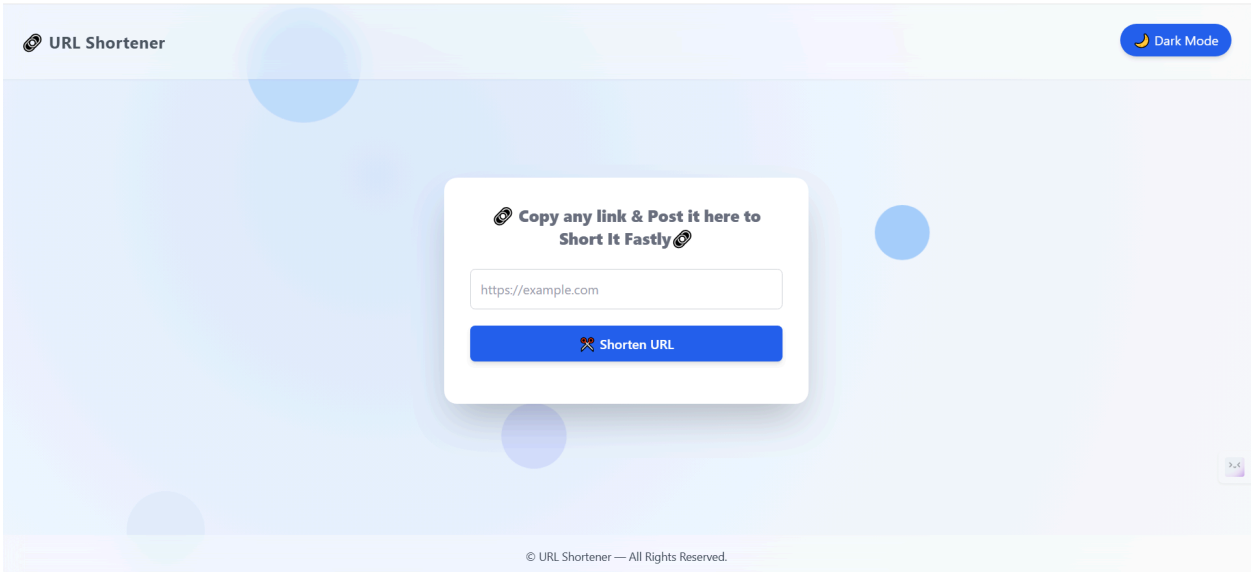
4.5.2.2. Redirect Sequence



4.5.2.3. Prometheus Scraping Sequence



4.6. UI/UX Prototyping

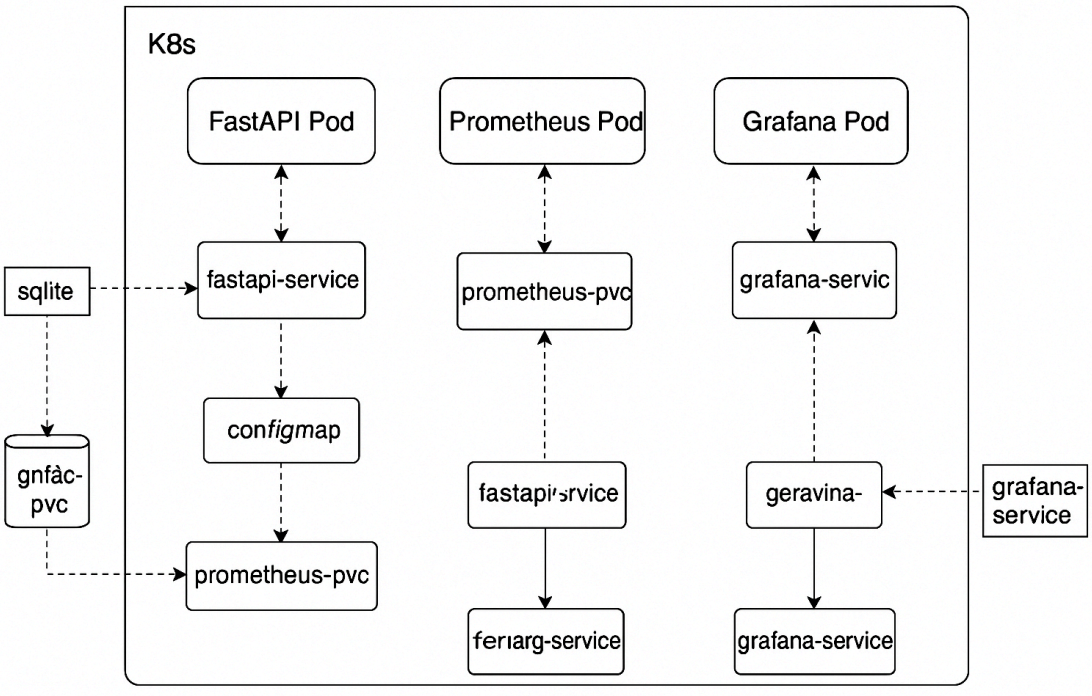


4.7. System Deployment Diagram

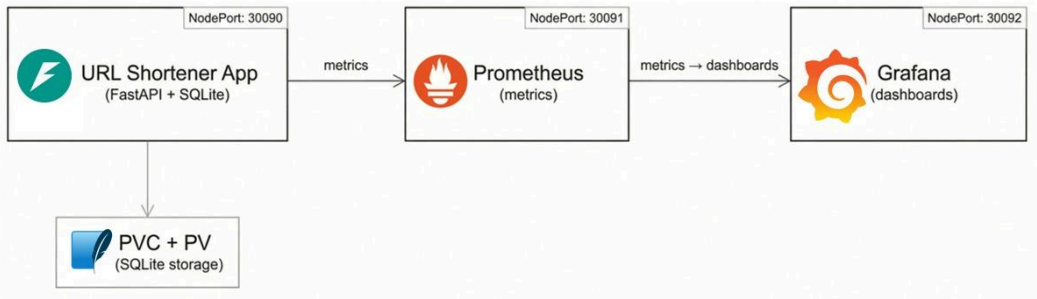
4.7.1. Tech Stack:

FastAPI, SQLAlchemy ORM, SQLite Database, HTML5 + TailwindCSS, Prometheus, Grafana

4.7.2. Deployment Diagram



4.7.3. Component Diagram



4.8. Additional Deliverables

4.8.1. API Documentation (LINK)

5. Implementation

5.1. Source Code Summary

5.1.1. Structured & Well-Commented Code

The project follows a clean and organized structure that separates backend logic, database access, models, frontend resources, and monitoring functionality into dedicated files

5.1.1.1. `url_shortener.py` – Main FastAPI application logic

5.1.1.2. **`models.py`** – SQLAlchemy ORM model definitions

5.1.1.3. **`database.py`** – Database engine, session, and dependency management

5.1.1.4. **`static/` (index.html, style.css, script.js)** – User interface assets

5.1.1.5. **`prometheus.yml`** – Monitoring configuration

5.1.2. Coding Standards & Naming Conventions

5.1.2.1. Consistent naming conventions

5.1.2.1.1. Snake_case for variables and functions → `generate_short_code()`, `get_db()`

5.1.2.1.2. PascalCase for Pydantic/ORM models → `URL`, `UrlBase`, `UrlResponse`

5.1.2.1.3. Clear and descriptive variable names → `short_code`, `long_url`, `redirect_success_count`

5.1.3. Modular Code & Reusability

The project is decomposed into reusable, maintainable modules

5.1.3.1. Database Module (`database.py`)

This ensures all routes share the same database session pattern.

5.1.3.1.1. Provides a reusable `SessionLocal` factory

5.1.3.1.2. Encapsulates DB access via the `get_db()` dependency

5.1.3.2. Models Module (`models.py`)

5.1.3.2.1. Defines a single reusable SQLAlchemy model shared across routes:

5.1.3.2.1.1. `URL` table with fields: `id`, `short_code`, `long_url`, `clicks`, `created_at`

5.1.3.3. Prometheus Metrics:

Metrics counters are declared once and reused throughout the app via imports.

5.1.3.4. Utility Function

5.1.3.4.1. Reusable function for code generation, following the single-responsibility principle.
`def generate_short_code(length=6):`

5.1.3.5. Frontend Reuse

5.1.3.5.1. `script.js` handles all client-side logic

5.1.3.5.2. Utility functions for copy-to-clipboard and dark-mode toggle

5.1.4. Security & Error Handling

5.1.4.1. Input Validation

5.1.4.1.1. The project uses **Pydantic models** to validate incoming URLs to ensures that only correct `http://` / `https://` URLs enter the system

```
class UrlBase(BaseModel):
    long_url: HttpUrl = Field(..., max_length=512)
```

5.1.4.2. Exception Handling

5.1.4.2.1. The redirect endpoint safely handles invalid codes preventing (Non-existent codes, Broken redirects, Unexpected crashes)

```
if not entry:
    lookup_failed_count.inc()
    raise HTTPException(status_code=404, detail="Short URL not found")
```

5.1.4.3. Database Safety

5.1.4.3.1. SQLAlchemy ORM prevents SQL injection

5.1.4.3.2. SQLite engine uses safe connection config (`check_same_thread=False`)

5.1.4.4. Security-aware Defaults

5.1.4.4.1. Short codes are randomly generated using a secure character set

5.1.4.4.2. Click tracking and admin endpoints are protected through server-side logic

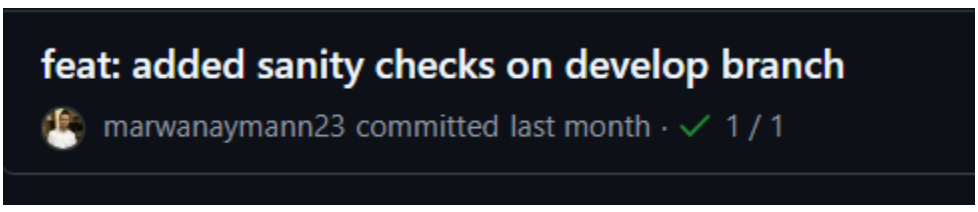
5.2. Version Control

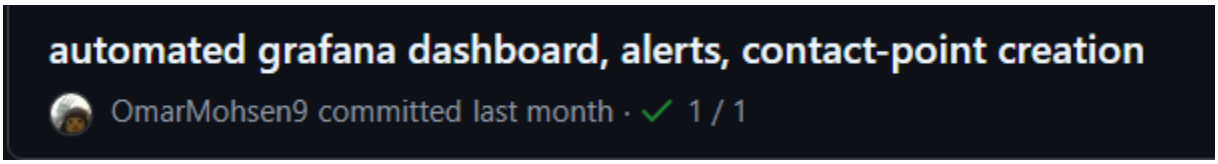
5.2.1. Repository hosted on GitHub (URL_Shortner_Repo)

5.2.2. Feature-branch workflow used

5.2.3. Commit messages are meaningful

The project maintains a consistent, meaningful commit history following widely accepted conventions.





6. Testing & Quality Assurance

6.1. Test Cases & Test Plan

A comprehensive testing process was applied to ensure system reliability, correctness, and performance

6.1.1. TC01: Shorten valid URL - Returns `200 OK` + valid `short_url`

6.1.2. TC02: Unique short code - Each request generates a different code

6.1.3. TC03: Redirect valid code - Returns `307 Temporary Redirect`

6.1.4. TC04: Redirect invalid code - Returns `404 Short URL not found`

6.1.5. TC05: Click tracking - `clicks` value increments in DB

6.1.6. TC06: Metrics exposure - Returns Prometheus metrics text

6.1.7. TC07: Admin list - Returns JSON list of all URL entries

6.2. Automated Testing (CI Sanity Checks)

The project uses a Continuous Integration workflow to validate every push and pull request automatically.

CI Workflow used: **develop-sanity-checks.yml**

6.2.1. Runs sanity checks on the project structure

6.2.2. Performs automated validations and checks APIs

6.3. Bug Reports

6.3.1. Reported Bug #1 – Redirect Not Found

Issue: Redirecting an unknown short code caused unexpected behavior.

Cause: No entry returned from the database.

Fix: Added condition and raised HTTPException:

```
if not entry:
    lookup_failed_count.inc()
    raise HTTPException(status_code=404,
detail="Short URL not found")
```

6.3.2. Reported Bug #2 – Duplicate Short Codes

Issue: Occasional collisions during short code generation.

Fix: Added a loop to regenerate codes until they are unique

```
short_code = generate_short_code()
while db.query(URL).filter(URL.short_code ==
short_code).first():
    short_code = generate_short_code()
```

6.3.3. Reported Bug #3 – Request Latency Measurement

Issue: Prometheus histogram not updating properly.
Fix: Added custom middleware:

```
latency = time.time() - start
request_latency.labels(endpoint=request.url.path)
.observe(latency)
```

Bug Tracking Summary

Most issues were tracked informally during development and resolved immediately as part of iterative testing cycles.

7. Final Presentation & Reports

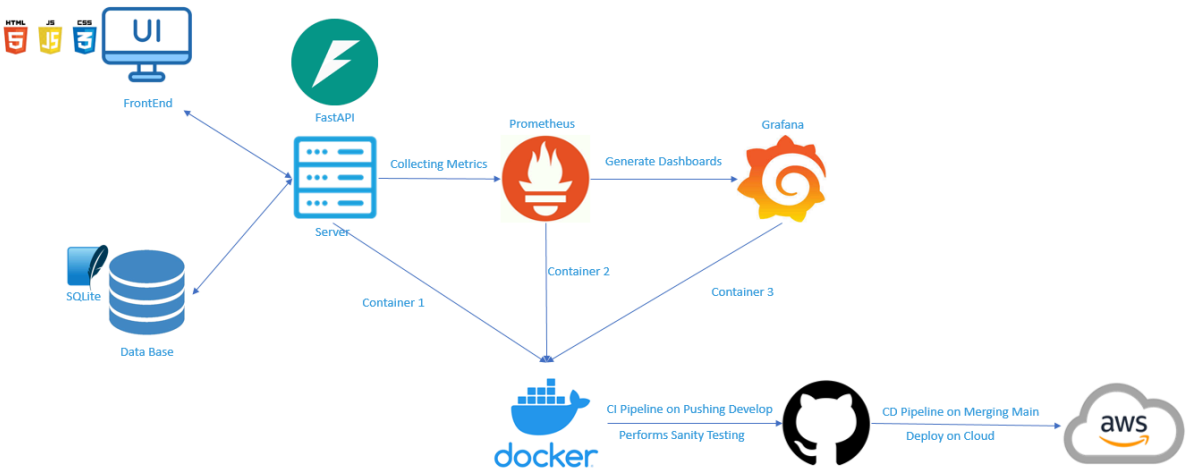
7.1. User Manual

[Usermanual.md]

7.2. Technical Documentation

7.2.1. System Architecture

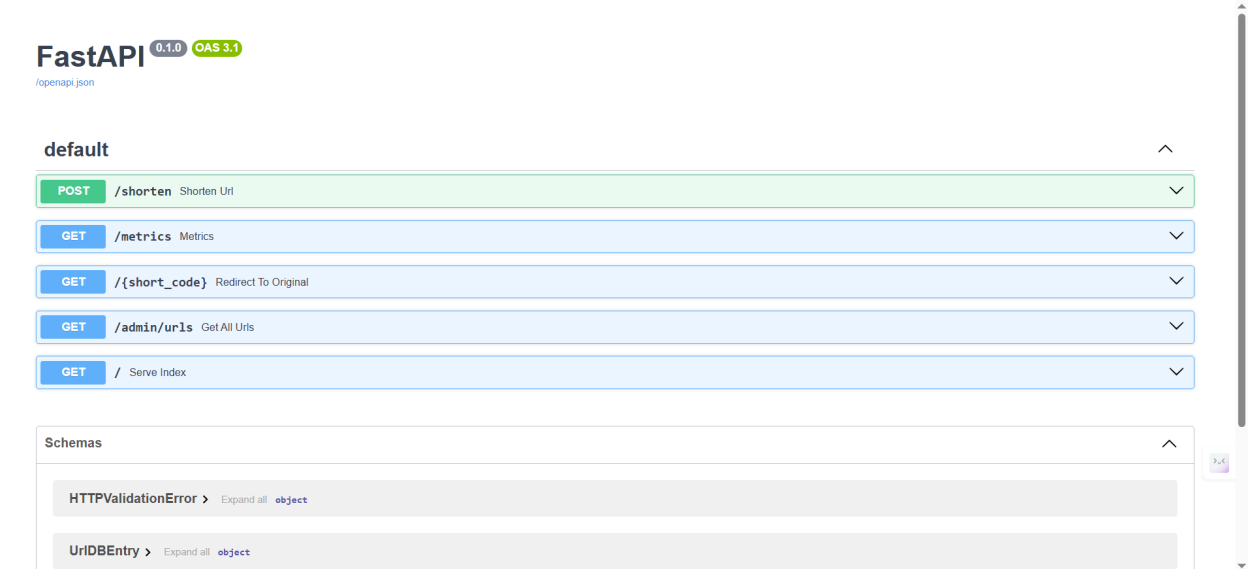
System Architecture



7.2.2. Database Schema

URL		
id	Integer	PK
short_code	String(10)	
long_url	String(512)	
clicks	Integer	
created_at	DateTime	

7.2.3. API Documentation



7.3. Project Presentation (LINK)