

Project Documentation Template

Table of Contents

1. Introduction
2. Project Structure
3. Installation
4. Usage
5. Configuration
6. Testing
7. Deployment
8. Contributing
9. License
10. Acknowledgments
11. System Architecture

Introduction

Provide a brief overview of the project, its purpose, and key features.

Project Structure

Explain the directory structure and the purpose of each folder/file.

```
/itu-minitwit
  .github/
    workflows
      build-and-test.yml      # GitHub Action workflows
      build-release.yml       # Automated build and test
      continuous-deployment.yml # Creates release on push with a tag
      lint-and-format-check.yml # Deployment to dig
      scheduled-release.yml   # Automated linter and formatting checks
      sonarcube.yml           # Automated weekly release
      sonarcube.yml           # Automated Sonarcube checks
  logging/
    docker-compose.yml        # Logging configuration files
    nginx.conf                 # Starts ELK stack and nginx containers
  logstash/
    nginx.conf                 # Reverse proxy with authentication
  remote_files/
    # Logstash configuration
  report/
    # Files used remotely on the minitwit server for
  src/
    # Report files
    minitwit.core/            # Source code
    minitwit.infrastructure/   # Domain Layer - Domain models
    minitwit.web/              # Infrastructure Layer - Data access
    Program.cs                 # Presentation Layer - Web app & API entry point
  terraform/
    # Program entrypoint
    # Terraform configurations for provisioning
```

files/	# Files used by terraform
modules/	
minitwit_logging/	# Terraform code for logging infrastucture
minitwit_server/	# Terraform code for minitwit infrastucture
main.tf	# Terraform module definitions
terraform.tfvars	# Terraform variables
variables.tf	# Terraform variables declarations
tests/	# Test cases
minitwit.tests/	
minitwit.tests.cs	# API tests
playwright.test.cs	# UI tests
docker-compose.yml	# For running the program locally
Dockerfile	# Application Dockerfile
itu-minitwit.sln	# Project solution file

Installation

Step-by-step guide on how to set up the project locally.

```
# Clone the repository
git clone https://github.com/username/repo-name.git

# Navigate to the project directory
cd repo-name

# Install dependencies
<insert installation commands>
```

Usage

Instructions on how to run and use the project.

```
# Example command to start the project
<insert usage commands>
```

Configuration

Details about configuration files and environment variables.

Process Perspective

This perspective should clarify how code or other artifacts come from idea into the running system and everything that happens on the way.

In particular, the following descriptions should be included:

- A complete description of stages and tools included in the CI/CD chains, including deployment and release of your systems.
- How do you monitor your systems and what precisely do you monitor?
- What do you log in your systems and how do you aggregate logs?
- Brief results of the security assessment and brief description of how did you harden the security of your system based on the analysis.
- Applied strategy for scaling and upgrades.

CI/CD Chains

Our CI/CD Pipelines are built for the purpose of easier maintainness and security of our deployment to servers. ensuring that our infrastructure are up regardless of maintainness or failures for newly pushed commits.

```
/root
  workflows
    build-and-test.yml
    build-release.yml
    continous-deployment.yml
    lint-and-format-check.yml
    scheduled-release.yml
    sonarcube.yml
```

The following workflows are implemented to ensure a robust CI/CD pipeline:

1. **Build and Test Workflow**
This workflow automates the build process and runs all unit and integration tests to ensure code quality.
2. **Build Release Workflow**
Automatically creates a release when a new tag is pushed to the repository.
3. **Continuous Deployment Workflow**
Deploys the application to the production server upon successful completion of tests and builds.
4. **Lint and Format Check Workflow**
Ensures that the code adheres to the project's linting and formatting standards.
5. **Scheduled Release Workflow**
Automates weekly releases to ensure regular updates and maintenance.
6. **SonarQube Workflow**
Performs static code analysis using SonarQube to identify potential bugs and vulnerabilities.

Each workflow is defined in the `.github/workflows` directory and is triggered based on specific events such as pushes, pull requests, or scheduled intervals.

Deployment Chain

The deployment process follows a structured chain format to ensure reliability and minimize downtime. The steps are as follows:

1. **Linting and Code Quality Checks**

The code is first analyzed for adherence to linting and formatting standards. This ensures that the codebase remains clean and maintainable.

2. **Integration Testing**

Once the linting checks pass, the commit undergoes rigorous integration testing to validate that all components work together as expected.

3. **Deployment with Rolling Updates**

If the commit successfully passes all previous stages, the deployment process begins. Rolling updates are utilized to ensure a seamless transition. This approach guarantees that if the deployment encounters any issues, an unaffected backup server remains operational to handle the workload while the problem is resolved.

This deployment strategy ensures high availability and minimizes the risk of service disruption during updates.

```
# Run tests
<insert test commands>
```

Deployment

Steps to deploy the project to production.

Contributing

Guidelines for contributing to the project.

License

This Project Itu_Minitwit is licensed and distributed under the MIT license

Acknowledgments

Credit individuals or resources that helped in the project.

SystemArchitecture

```
# Dependency List:
1. Microsoft.EntityFrameworkCore.Design - Version: 9.0.1
2. Microsoft.Extensions.Configuration - Version: 9.0.2
3. Microsoft.Extensions.Configuration.EnvironmentVariables - Version: 9.0.2
4. Microsoft.Extensions.Configuration.UserSecrets - Version: 9.0.2
```

5. Npgsql.EntityFrameworkCore.PostgreSQL - Version: 9.0.4
6. prometheus-net - Version: 8.2.1
7. Serilog.AspNetCore - Version: 9.0.0
8. Serilog.Sinks.Console - Version: 6.0.0
9. Microsoft.AspNetCore.Identity - Version: 2.3.1
10. Microsoft.EntityFrameworkCore.Sqlite - Version: 9.0.1
11. Microsoft.AspNetCore.Identity.EntityFrameworkCore - Version: 9.0.1
12. Microsoft.AspNetCore.Identity.UI - Version: 9.0.1
13. Microsoft.EntityFrameworkCore.Tools - Version: 9.0.0
14. Microsoft.VisualStudio.Web.CodeGeneration.Design - Version: 9.0.0
15. prometheus-net.AspNetCore - Version: 8.2.1
16. Serilog - Version: 4.2.0
17. Serilog.Formatting.Compact - Version: 3.0.0
18. Serilog.Sinks.Elasticsearch - Version: 10.0.0
19. Serilog.Sinks.Network - Version: 2.0.2.68
20. Serilog.Sinks.Async - Version: 1.5.0
21. coverlet.collector - Version: 6.0.4
22. Microsoft.AspNetCore.Mvc.Testing - Version: 9.0.2
23. Microsoft.NET.Test.Sdk - Version: 17.13.0
24. Microsoft.Playwright.NUnit - Version: 1.50.0
25. xunit - Version: 2.9.2
26. xunit.runner.visualstudio - Version: 3.0.0
27. Postgres - Version: 16.9
28. Kibana - Version: 8.12.1
29. logstash - Version: 8.12.1
30. elasticsearch - Version: 8.12.1
31. Nginx - Version: 1.27.0
32. Dotnet_SDK - Version: 9.0.0
33. org.Sonarcube - Version: 6.1.0

sequenceDiagram

```

    participant minitwit_Simulator
    participant Minitwit_Application
    participant DataBase

    minitwit_Simulator->>Minitwit_Application: Http Post (api/fllws/{username}) unfollow {us
    Note right of Minitwit_Application: FollowRequest
    Minitwit_Application-->>minitwit_Simulator: http statuscode 200
    Note left of minitwit_Simulator: Succesfull Response
    loop BatchInsert
    Minitwit_Application->>Minitwit_Application: Insert into batch queue
    end
    Minitwit_Application->>DataBase: Get UserId <user1> <user2>
    DataBase->>Minitwit_Application: UserId1 UserId2
    Note left of Minitwit_Application: checks if both users exists
    Minitwit_Application->>DataBase: Does User1 Follow User2
    DataBase->>Minitwit_Application: Bollean
  
```

Note left of Minitwit_Application: If true
Note right of DataBase: Unfollow Sql Command
Minitwit_Application->>DataBase: Put user1 unfollow user2
DataBase->>Minitwit_Application: Status Response