MSc_group_c

ITU MiniTwit Report Skeleton

May 2024

1 Introduction

2 System perspective

This section presents the system.

2.1 Design and architecture

The system is refactored with the Go programming language.

2.1.1 Module diagram

An overview of the modules of the codebase is presented by the following package diagram. The package diagram models the internal structure of the codebase from the src folder (not infrastructure). Though it should be noted that within the handlers folder, the classes auth.go, message.go and user.go are presented, and its dependencies. This is to depict the complexity of that modules (since it is the biggest and most central module in the system).

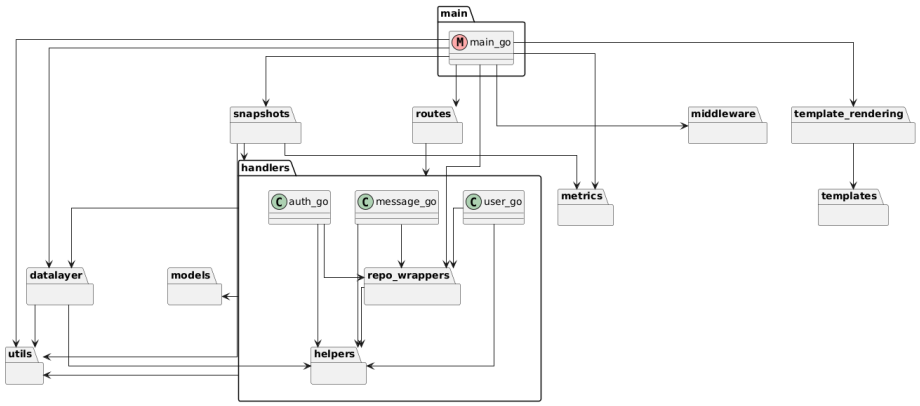

Figure 1: Module diagram

In the diagram it can be seen, that the main.go file orchestrates the system. It (in this context) has the responsibility for: 1. Render the template (frontend) 2. Initialize a new instance of the database object 3. Setup middleware 4. Setup routes, which have the responsibility of exposing the endpoints that further orchestrates to the handlers module for the logic of the API.

2.1.2 Sequence diagrams

Below, two sequence diagrams showcase how the different parts of the system interact when procecssing a "Follow" request. The first version shows the processes involved when the request is sent via. the UI, whereas the second version shows the processes involved when sent via. the API.

Note that while both versions use the same API, they use different endpoints.

2.2 Dependencies

2.3 System interactions

2.4 Current state of the system

2.4.1 SonarQube analysis summary

The following table summarizes key code quality metrics from SonarQube analysis:

Metric	Value
Lines of Code (LOC)	1,591
Code Duplication	4.1%
Security Hotspots	8
Overall Rating	A (Excellent quality)
Cyclomatic Complexity	216 (handlers: 151)
Technical Debt	~1 hour 7 minutes

2.4.2 Code climate

The following table summarizes key code quality metrics from Code Climate analysis:

Metric	Value
Lines of Code (LOC)	1,912
Code Duplication	0 %
Overall Rating	A (Excellent quality)
Complexity	299 (handlers: 196)
Technical Debt	~1 day 2 hours

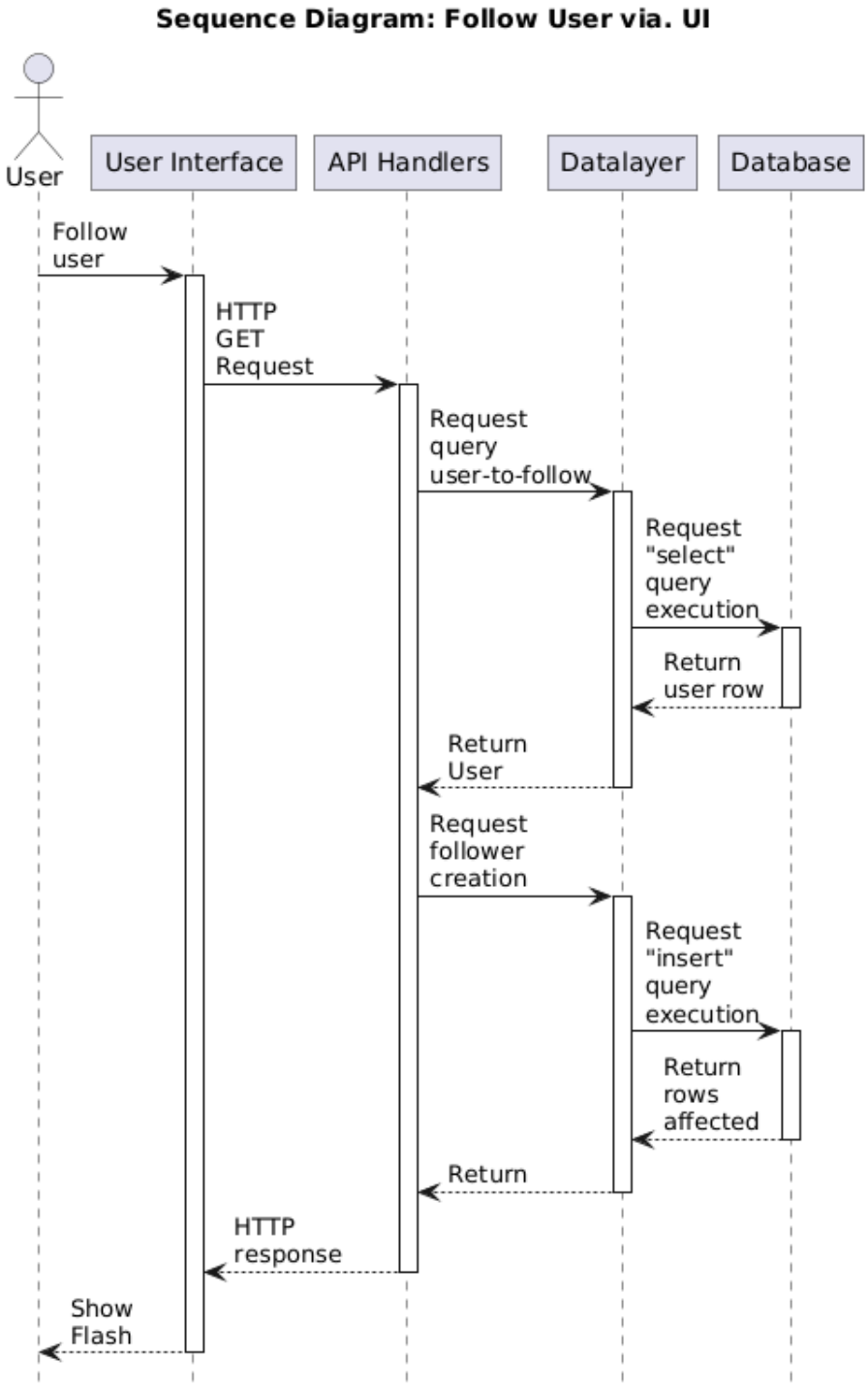

Figure 2: Sequence diagram - Follow request via UI

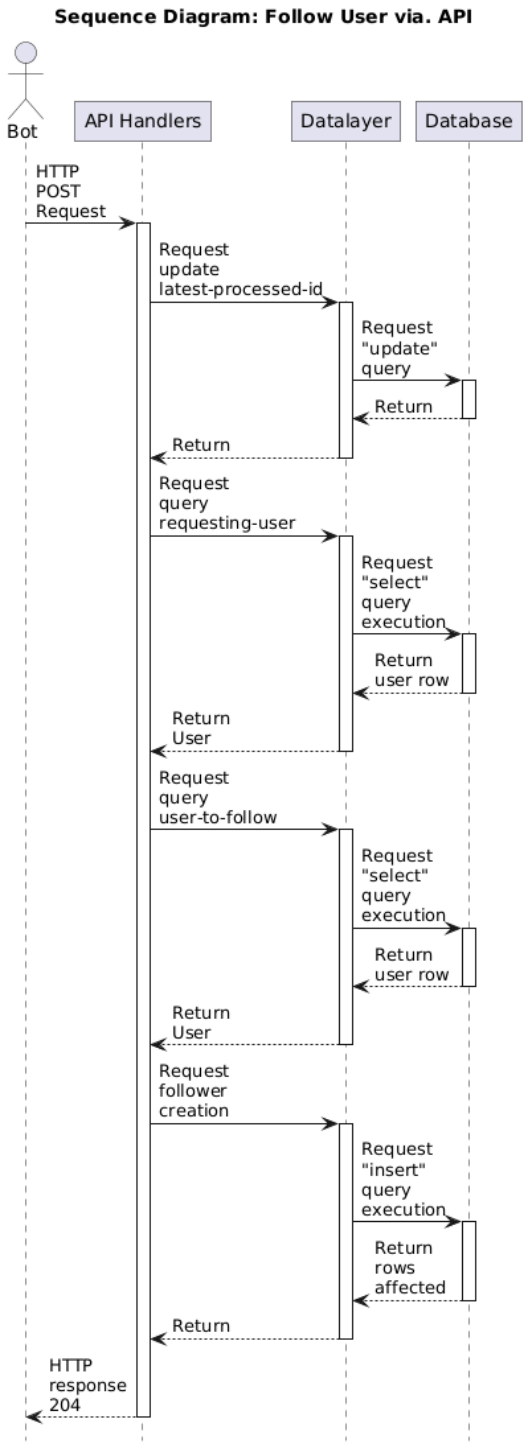

Figure 3: Sequence diagram - Follow request via API

2.4.3 Overall assesment

Both tools show a high complexity in the handlers module

2.5 Orchestration

To streamline the deployment of the program, Docker, docker-compose, Docker Swarm, and Terraform are used.

The Dockerfile copies all source code from the `src` package to a binary image of the program.

There are two docker-compose files, `docker-compose.yml` and `docker-compose.deploy.yml`. Both define the six central services of the system: app, prometheus, alloy, loki, grafana, and database.

`docker-compose.yml` is needed for local deployment. It uses localhost IP-adresses and has default usernames and passwords.

`docker-compose.deploy.yml` is used for remote deployment. It builds on `docker-compose.yml`, but replaces information where relevant. It specifies the configuration of a Docker Swarm with one manager and two workers: The app runs on two worker replicas, while logging and monitoring services are constrained to only run on the manager node (though alloy collects logs from everywhere). This enables horizontal scaling.

Infrastructure-as-Code is used simplify the setup of the Docker Swarm remotely. Terraform files can be found in `.infrastructure/infrastructure-as-code`. Automatic deployment via. Terraform works as illustrated in the sequence diagram below.

2.6 Deployment

2.6.1 Allocation viewpoint

3 Process perspective

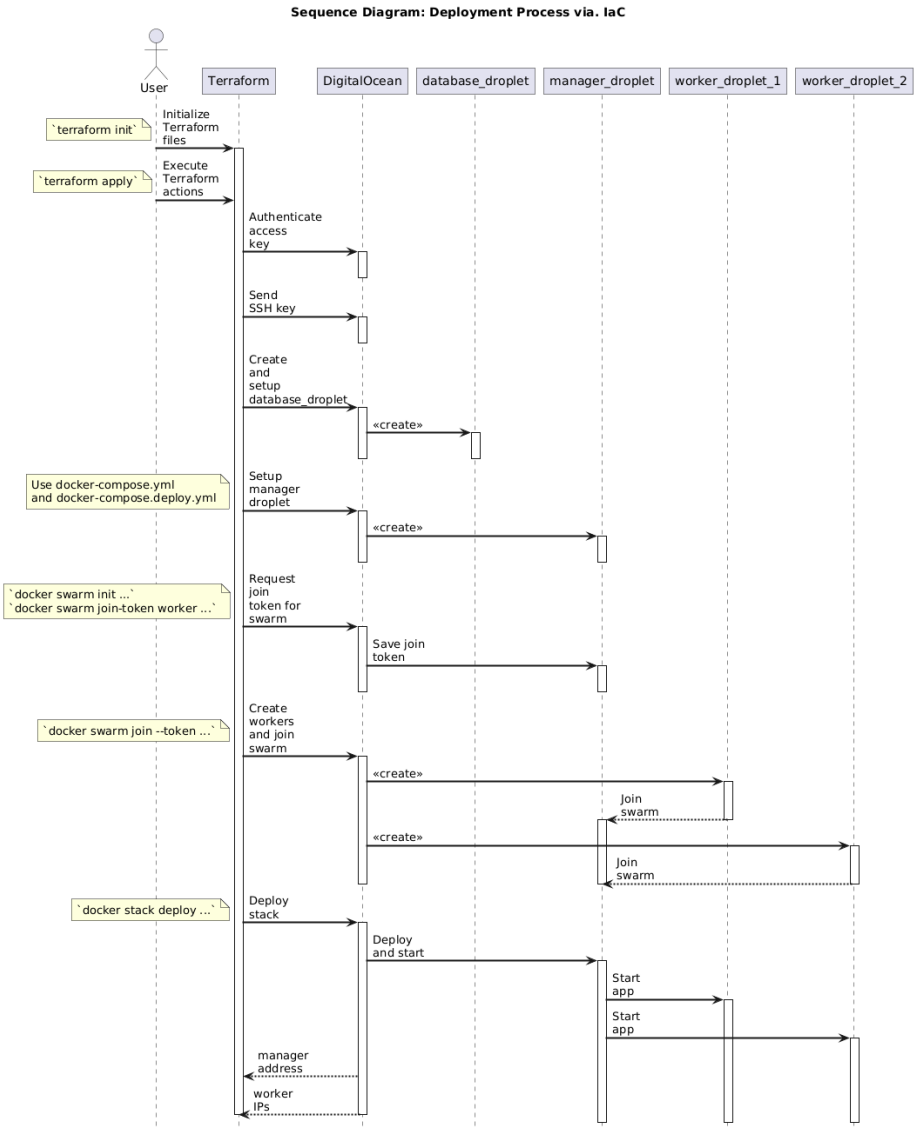

Figure 4: Sequence diagram of IaC

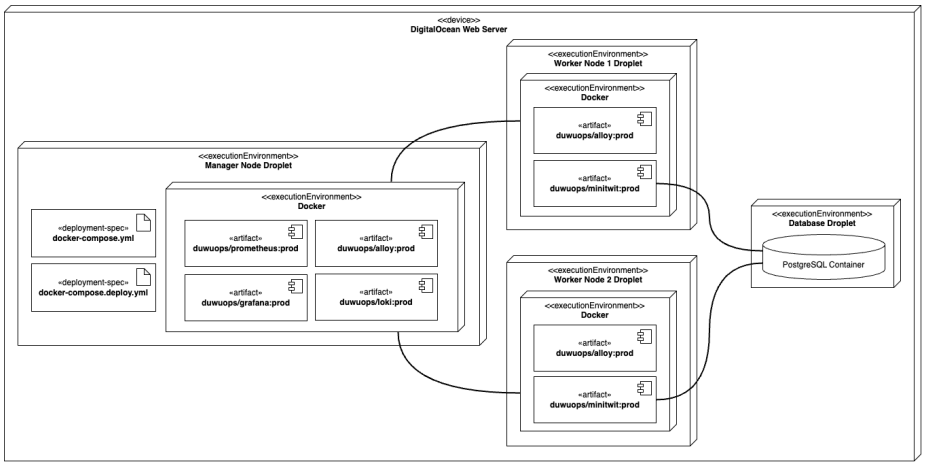

Figure 5: Deployment diagram