# The Report

# Introduction

During the course DevOps, Software Evolution and Software Maintenance an application stemming from an earlier course has been subjected to a simulation simulating real world usage by users. This has forced us to make the application more durable, more performant, and more maintainable. This has taught us to develop more performant applications, implement robust CI/CD workflows, and to use metrics, monitoring and logging to monitor, and keep our application healthy.

# System’s Perspective[MADS, MAGNUS]

## Design and Architecture of ITU-MiniTwit Systems

## Dependencies of ITU-MiniTwit Systems

- ASP.NET Core: The foundation of our application, used for implementing the web server, user authorization and identification, and our minimal web API.
- Entity Framework: Used for the database abstraction layer and all CRUD operation logic.
- Playwright: A testing library enabling end-to-end (E2E) tests.
- Prometheus: Monitoring system and time-series database for collecting and querying metrics.
- Xunit: A testing library for integration and unit testing.
- Coverlet: A tool for determining code coverage in tests.
- Moq: A test library for generating mock objects for unit testing.

## Important Interactions of Subsystems

- For example, via an illustrative UML Sequence diagram that shows the flow of information through your system from user request in the browser, over all subsystems, hitting the database, and a response that is returned to the user.
- Similarly, another illustrative sequence diagram that shows how requests from the simulator traverse your system.

## Current State of Systems

# Process' Perspective [OLIVER, MIKKEL]

## CI/CD Chains

# System Monitoring

# System Logging

We initially tried deploying the ELK stack for logging and monitoring but faced difficulties. We then chose New Relic, which logs everything written to a the node's console, similar to local debugging. It logs stack traces, exception messages, and preceding events when requests fail, allowing us to monitor API access, track server actions, and identify errors.

The aggregated logs where sent to New Relic and were accessible through their propriotary dashboard.

We configured the New Relic agent in our Dockerfile, ensuring consistent logging across all application nodes.

# Security Assessment

**OLIVER**

# Scaling and Upgrade Strategy

For scaling the application, Horizontal scaling with Docker Swarm was applied, as it offers numerous benefits, including improved application availability, load balancing, and fault tolerance. By distributing containers across multiple nodes, Docker Swarm ensures that the application can handle increased traffic and continue operating even if some nodes fail. This scalability allows for seamless expansion of resources in response to the growing demands during the course. Given the existing use of Docker in the application, adopting Docker Swarm was a logical choice, leveraging the teams familiarity with Docker's ecosystem while enhancing their ability to manage and scale containerized applications efficiently.

We chose rolling updates for our deployment strategy as it is the default method in Docker Swarm and aligns well with our existing infrastructure. This approach provides continuous availability by updating services incrementally, minimizing downtime without requiring additional resources. The alternative for this strategy is the Blue-Green upgrade strategy, but the additional ressources and implementation time was what additionally made the team favour Rolling Updates. ([szulik_2017])

# Use of AI-Assistants

In this project these AI-assistants were used:

- OpenAI's ChatGPT version 3.5, 4.0.
- GitHub Copilot

The AI-assistants were mainly used for:

- Breaking down code logic. I.g. In order to recreate the Python API controller provided by the course, the code needed to be translated into C# and modified to fit our application. ChatGPT

was a great tool for understanding each endpoint and what data would be included in a call and a response.

- Code completions. GitHub Copilot acted as an extension of IntelliSense, in the sense that it could auto-complete simple pieces of code, such as loops, if-statements, and method signatures.
- Research. ChatGPT was also used to provide a secondary explanation when researching new technologies, in situations were the documentation either was difficult to understand, or if subsidary information was needed.
- Stacktrace breakdowns. ChatGPT was used to breakdown stracktraces, summarizing the information as well as providing a more user-friendly format to read.
- Identify functions that could be made more performant.

Downsides of using AI-assistants:

- Both ChatGPT and Github Copilot are flawed, which makes them unreliable tools. Sometimes it would take as much time to doublecheck the output of an assistant as would have to complete the task without it, which defeats the purpose of using them.
- If used without careful inspeciton of the provided code, the LLM is likely to introduce bugs into the application. This is due to the fact, that LLM's have a difficult time understanding the context in which the requested code is supposed to operate.

# Lessons Learned Perspective [EVERYONE]

## Evolution and Refactoring

## Operation

## Maintenance

# DevOps Style of Work

As the entire team has been taking the course "Second Year Project: Software Development in Large Teams" which introduces working by the Agile principles and with Scrum as a framework, it's only natural that some elements have been taken into the project especially since these frameworks align well with the DevOps style of work as shown in table 7 of ([jabbari_2016]).

The effects of learning Scrum seeped into the working style of the team, not by introducing scrum events and the like, but by using the 3 pillars of Scrum; Adaptation, Transparency, and Inspection ([scrum_guide_2020]) as guidelines. Each Friday the team held physical meetings, where the state of the project was discussed, keeping each member up to date while answering questions any member might have. Breaking down the work each week, increased understanding of the project, transparency, and ensured openness amongst the team. GitHub allowed for fine-grained inspection through peer-reviewed code inspections facilitated with Pull requests. GitHub also provided a Kanban board to showcase the backlog, as well as the status of ongoing work.

In the same way the agile principles were introduced to the project. Of the twelve principles; "Welcome changing requirements" ([agile_principles_2001]), was the most prevalent as new requirements were added almost weekly. Furthermore how to meet those requirements wasn't set in stone. In situations where the team would find a better way to fulfill a task, there would be little resistance to incorporating it into the project.

Another vital principle was; "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation". To implement this principle, the team had both the weekly physical meeting, but would routinely also hold pair-/ or mob-programming sessions. The latter part, contributed to increasing the ownership of the codebase, generally raises the quality of the code produced, and minimises the time spent on code inspections. **FIND STUDY THAT SHOWS BENEFITS OF PAIRPROGRAMMING**

# Conclusion

# References

Szulik, Maciej. "Colorful deployments: An introduction to blue-green, canary, and rolling deployments." *Opensource.com*, 2 May 2017. https://opensource.com/article/17/5/colorful-deployments

Scrum guide, 2020. https://scrumguides.org/scrum-guide.html

The Agile Manifesto, 12 principles, 2001. https://agilemanifesto.org/principles.html

Jabbari, Ramtin. "What is DevOps? A Systematic Mapping Study on Definitions and Practices.", 2016 https://www.researchgate.net/publication/308857081_What_is_DevOps_A_Systematic_Mapping_Study_on_Definitions_and_Practices