DevOps report

Write names here

May 17, 2021

1 System's Perspective

1.1 Design of the system

[ALL, intially JACOB AND FREDERIK]

1.2 Architecture of the system

ALL, initially ALBERT AND RASMUS]

1.3 Dependencies

[ALEKXANDER AND JACOB]

1.3.1 Software dependencies

The dependencies of the system is derived via GitHub from the `mvc-minitwit.csproj` and

`HomeControllerTests.csproj` project files. These correspond to the MiniTwit and testing project respectively. The dependency graph can be found on https://github.com/albertbethlowsky/DevOpsGroupH/network/dependencies. A small snippet of the current graph for both projects can be seen below:

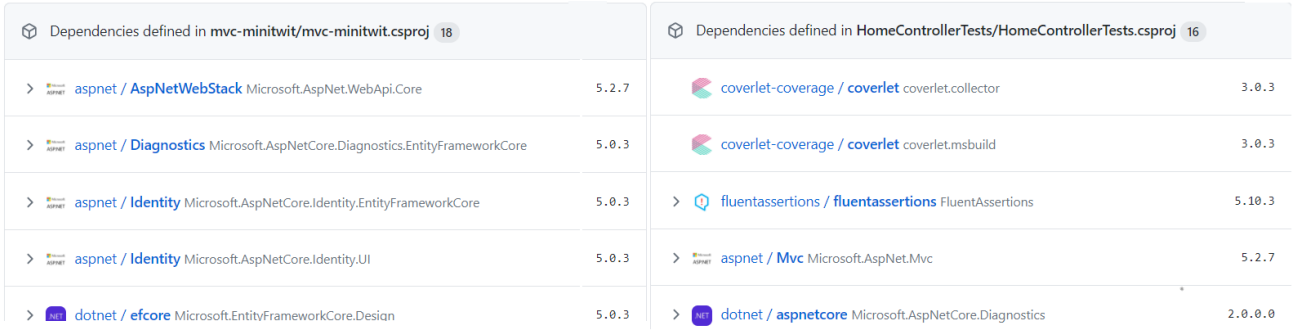

Figure 1: Snippet of dependency graphs of both `mvc-minitwit.csproj` and

`HomeControllerTests.csproj`

1.3.2 Cloud dependencies

These dependencies are related to the services hosted on Azure, where MiniTwit also is deployed.

Name	Service	Provider	Description
neutrals-minitwit	App Service	Microsoft Azure	Hosting of web applications (.NET application)
minitwit-neutrals	App Service	Microsoft Azure	Hosting of web applications (prometheus, grafana)
neutralsseq	App Service	Microsoft Azure	Hosting of web applications (Datalust - Seq)
minitwit-neutrals	SQL Server	Microsoft Azure	Hosting of SQL database
minitwitDb (minitwit-neutrals)	SQL database	Microsoft Azure	SQL database
neutralsminitwit.azurecr.io	Docker container	Azure Container Registry	Containerizing of applications

1.3.3 Important interactions of subsystems

[FREDERIK, JACOB, ALEKXANDER]

1.4 Current state of the system

[ALEKXANDER AND FREDERIK] With the help of different static analysis tools, it's possible to get a sense of the state of the system, including an estimation of the technical debt. Their respective results are briefly presented:

SonarCloud

Based on the results from SonarCloud the systems seems to be in a satisfactory state, however there are a few caveats to point out. The auto-generated contents of `Migrations/`, and the `wwwroot/lib` folder has been excluded from all analysis since the latter contains deprecated jQuery libraries. These were given from the beginning, and were thus deemed out of scope to fix with time available.

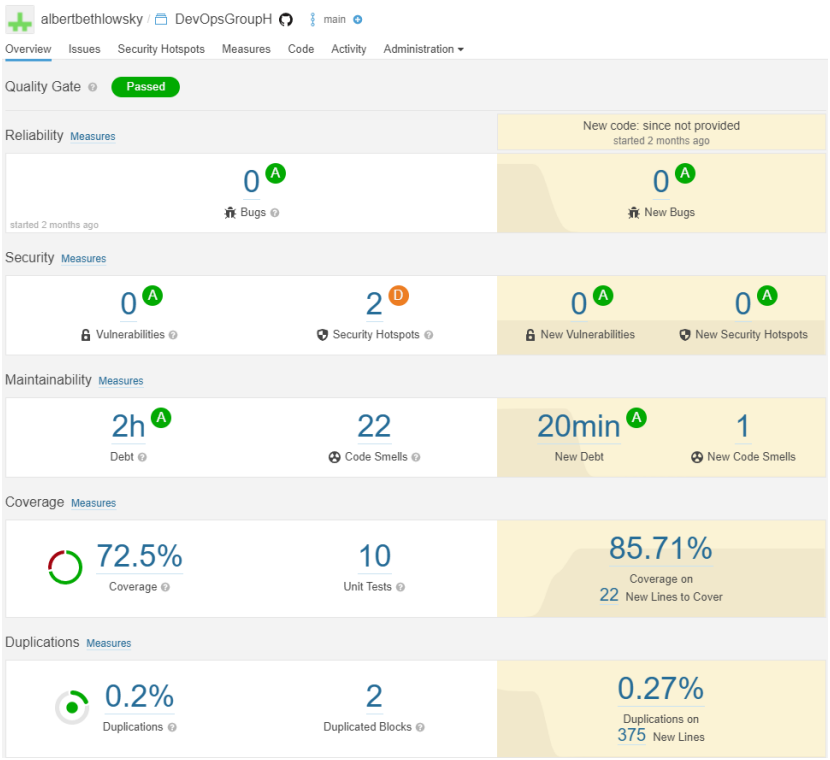

Figure 2: Results from SonarCloud and its different categories, with the Quality Gate passed

The boiler plate, initialization files `Program.cs`, `Startup.cs` are excluded from test coverage analysis. `HomeController.cs` is also excluded from test coverage due to project scope and challenges with mocking cookies. From the maintainability category, there's "2 hours" of debt, but it's worth mentioning that it's a relative estimate based on "Code smells", uncovered and duplicated lines. This estimate should be compared with the estimates from the other tools.

Code Climate

The repository has a maintainability grade of "B", which with their estimates would be about 2 days of technical debt. There are 19 issues in total, split into 8 of duplication and 11 code smells. The file with the lowest maintainability score is `HomeController.cs` and has a "C", which would be the file to prioritize first for refactoring. The same files and folders have been excluded as with SonarCloud.

As a side note a big source of the issues in both `HomeController.cs` and `APIController.cs` comes in the shape of *"Avoid too many return statements"*. This is an issue that can be taken with a grain of salt, since it relates to the debate of "clean OOP" approach, which is harder to implement in server/API application.

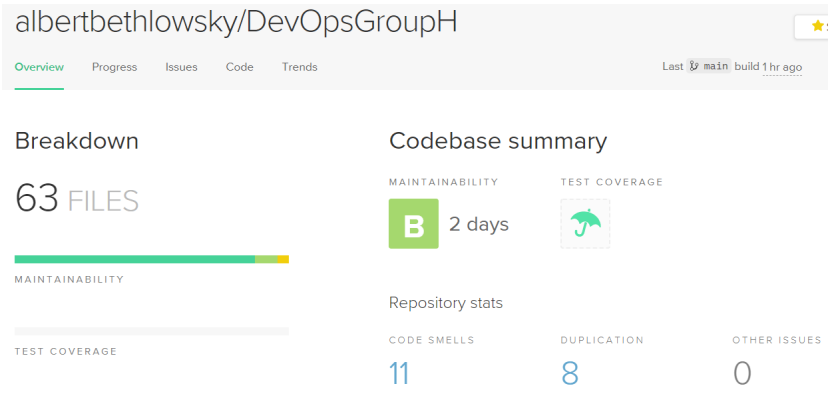

Figure 3: Code Climate summary and overview

Better Code

[Albert]

InferSharp

At the top of the `README.md`, badges of the different tools have been added to help give a quick overview of their results. It's worth mentioning that technical debt is a relative term, and as seen between the tools, can vary quite a lot (e.g. 2 days to 2 hours).

1.5 Licenses and compatibility

[ALBERT AND RASMUS]

2 Process perspective

2.1 Interactions between developers

[RASMUS ALBERT]

2.2 Team organization

[RASMUS ALBERT]

2.3 Description of CI/CD pipelines, stages and tools

[ALEKXANDER FREDERIK] Azure DevOps is used to manage the CI/CD pipelines for the project. See azure-pipelines.yml for the stages and tools utilized in our pipeline. Terraform will be applied to the pipeline in future iterations such that we have automated the infrastructure (IaC) maintenance.

2.4 Repository arrangement/organization

[FREDERIK JACOB]

2.5 Applied Branching Strategy

[ALEXANDER FREDERIK] Throughout the project a "Long-Running Branches" strategy has been used. The *master* branch has primarily been used for code releases or `.yml` pipeline changes. The *development* branch was used to merge all the features back into, so that it in turn becomes can be merged into *master* as the next stable release. Finally the individual feature branches correspond to the issues on the Task board on GitHub. Each feature branch is named after their issue number,

e.g. `fea34APIController`. Pull requests to merge the branches has been used, but not with absolute consistency.

2.6 Applied development process and related tools

[JACOB FREDERIK]

2.6.1 Simple Kanban board

[FREDERIK RASMUS] We have utilized github's projects to manage our tasks. The board consists of the traditional kanban setup with 'tasks', 'in progress', 'parked', and 'done'.

2.7 Monitoring

[JACOB ALEKXANDER]

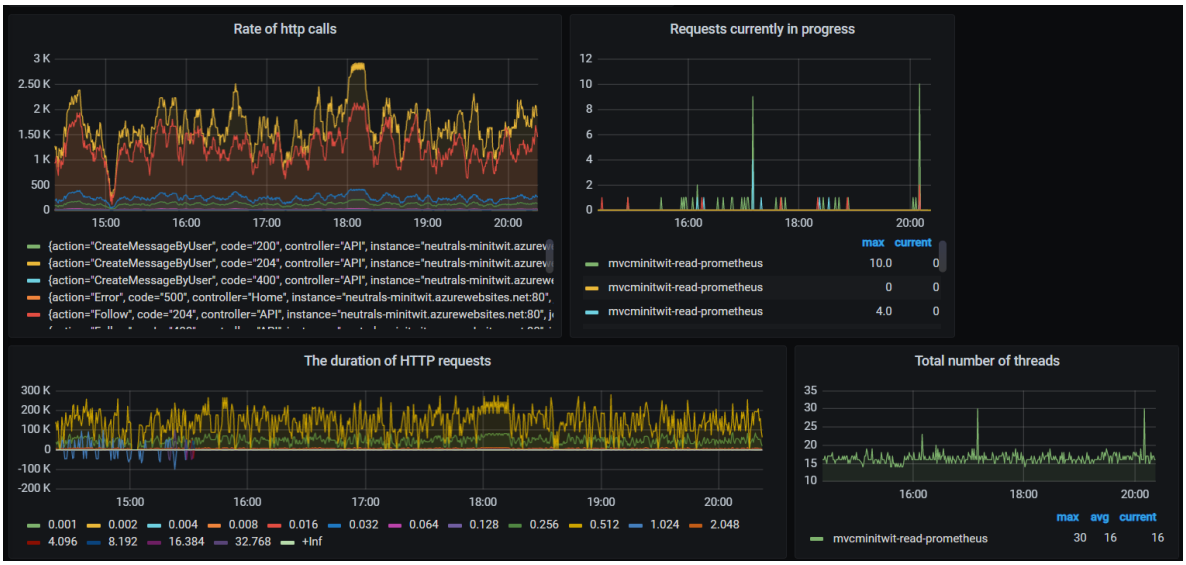

Figure 4: Grafana dashboards

2.8 Logging and log aggregation

[JACOB ALEKXANDER]

2.9 Brief security assessment

[RASMUS ALBERT]

2.10 Applied scaling and load balancing strategy

[FREDERIK AND ALEKXANDER] Due to different challenges with Azure Kubernetes Service on Azure and Azure DevOps it was not possible to implement scaling within the given time frame. We did however implement azure's 'scale out' service which is a built-in feature that helps applications perform their best when demand changes.

3 Lessons learned perspective

3.1 Evolution and refactoring

[ALL, init JACOB AND RASMUS]

3.2 Operation

[ALL, init ALEKXANDER AND ALBERT]

3.3 Maintenance

ALL, init FREDERIK and JACOB]