Chirp! Project Report

ITU BDSA 2023 Group `15`

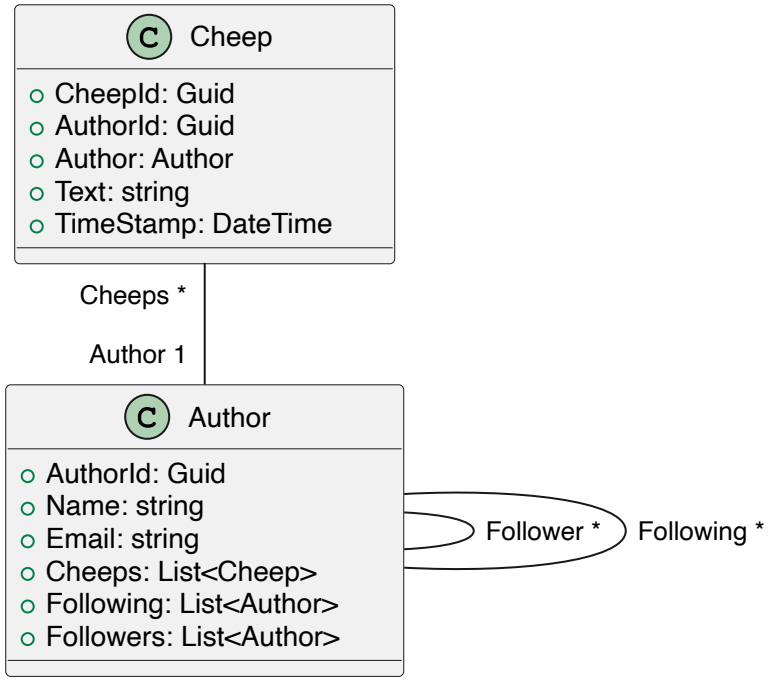
Daniel Sølvsten Millard dmil@itu.dk
Frederik Lund Rosenlund frlr@itu.dk
Jacob Vortscir Pærregaard jacp@itu.dk
Mads Christian Nørklit Jensen macj@itu.dk
Rasmus Lundahl Nielsen raln@itu.dk

- Design and Architecture of *Chirp!*
  - Domain model
  - Architecture — In the small
  - Architecture of deployed application
  - User activities
  - Sequence of functionality/calls trough *Chirp!*
- Process
  - Build, test, release, and deployment
  - Team work
  - How to make *Chirp!* work locally
  - How to run test suite locally
    - Unit and integration tests
    - UI tests (E2E)
- Ethics
  - License
  - LLMs, ChatGPT, CoPilot, and others

1 Design and Architecture of *Chirp!*

1.1 Domain model

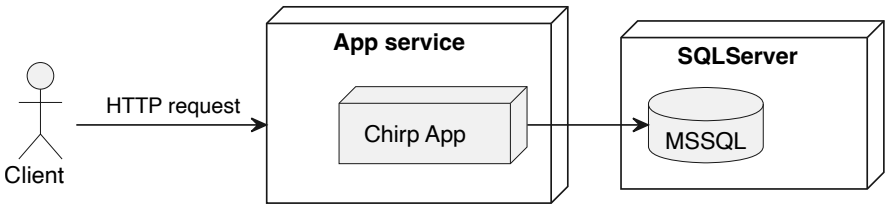
Here comes a description of our domain model.



1.2 Architecture — In the small

1.3 Architecture of deployed application

"Illustrate the architecture of your deployed application. Remember, you developed a client-server application. Illustrate the server component and to where it is deployed, illustrate a client component, and show how these communicate with each other."



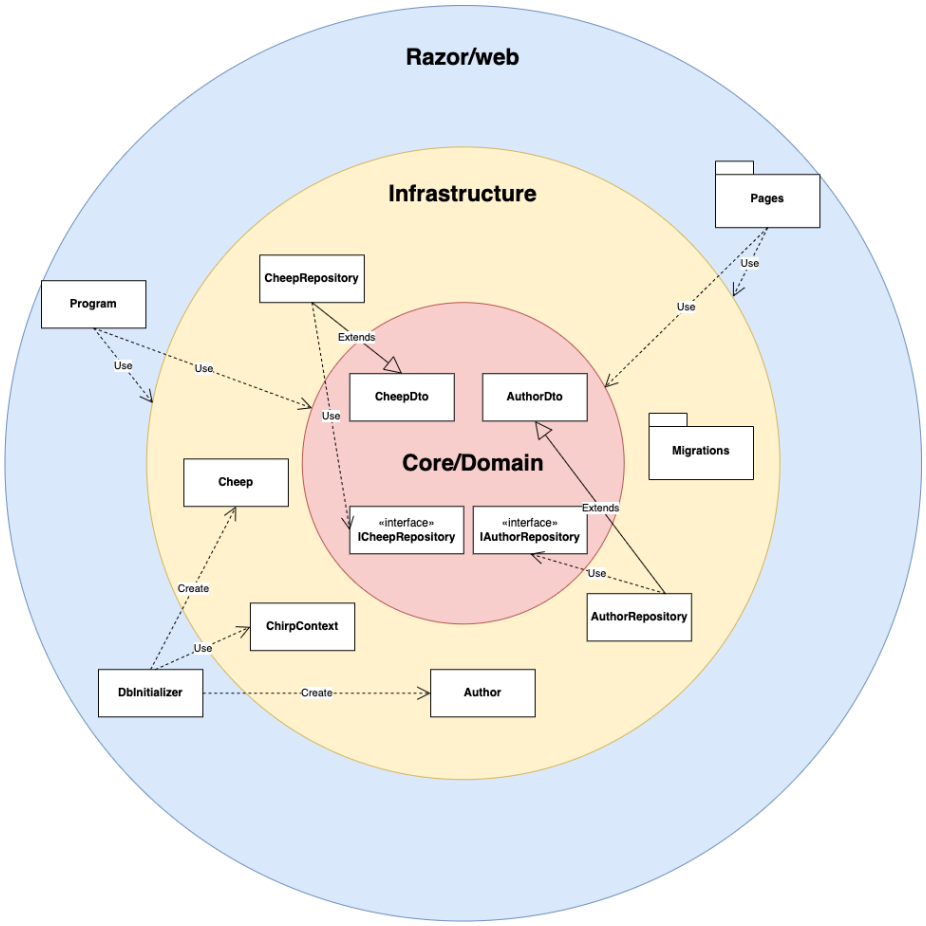
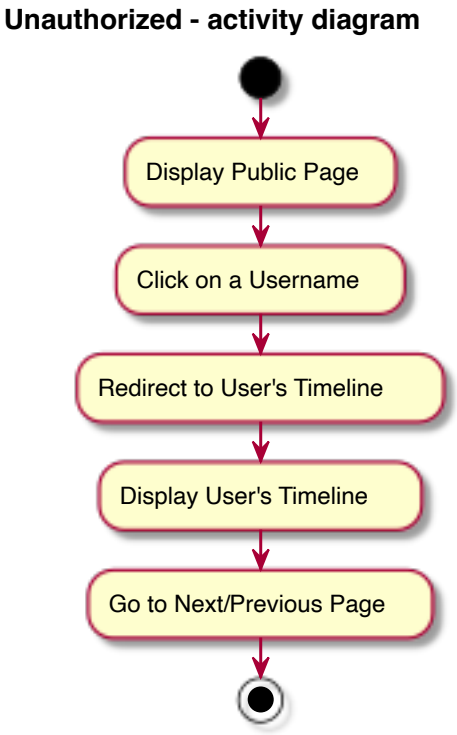
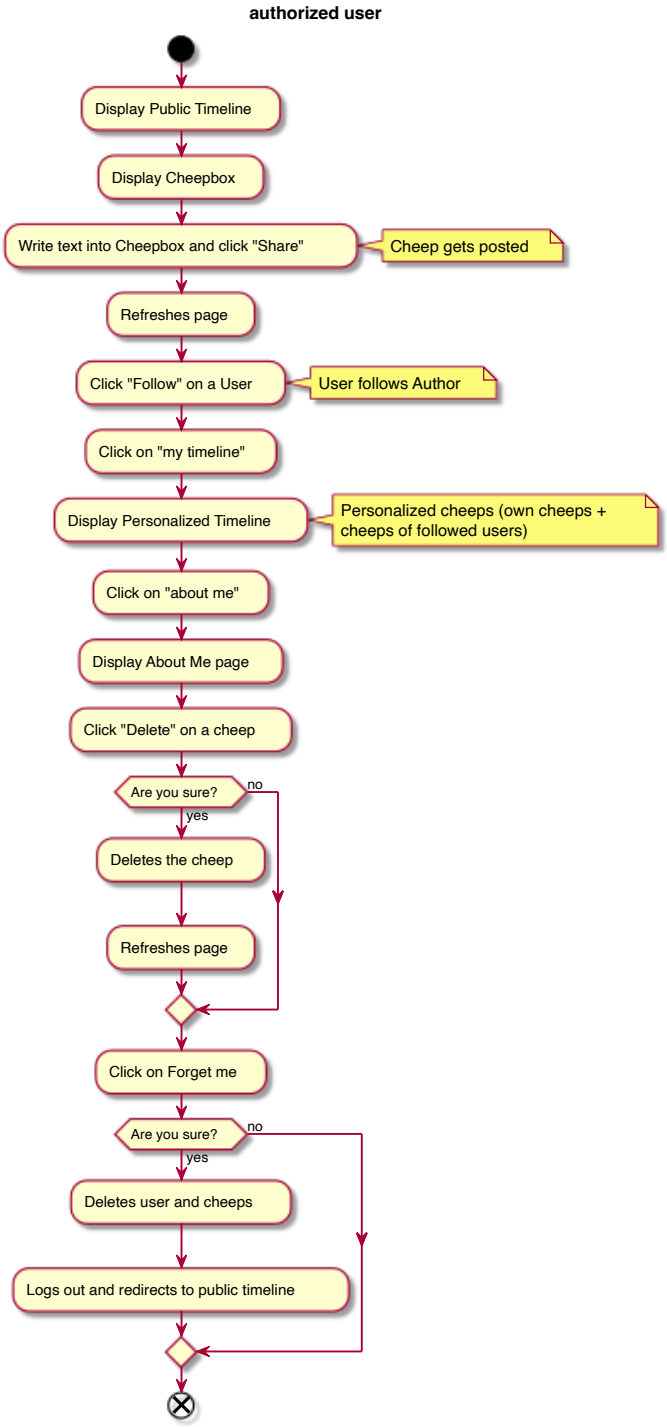


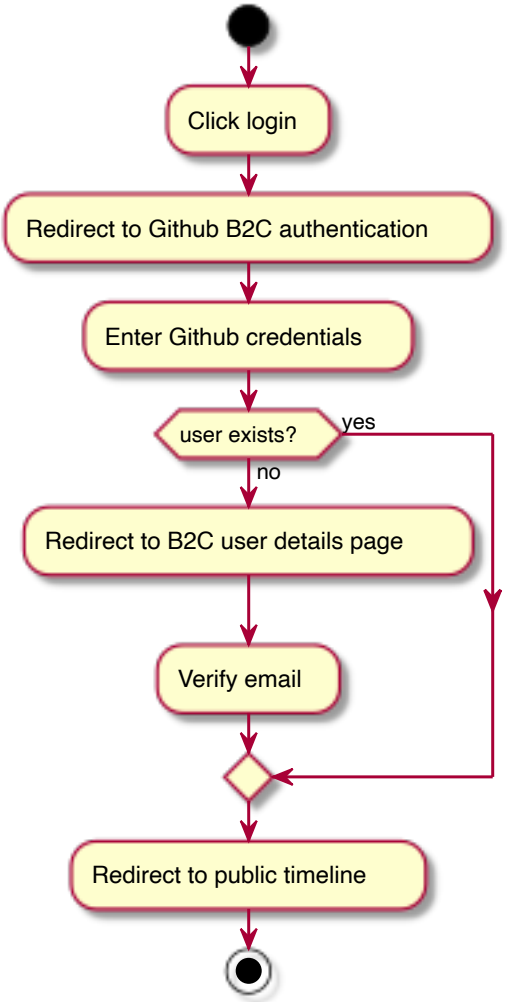
Figure 1: onion_architecture

1.4 User activities




authorized user
Display Public Timeline
Display Cheepbox
Write text into Cheepbox and click "Share"
Cheep gets posted
Refreshes page
Click "Follow" on a User
User follows Author
Click on "my timeline"
Display Personalized Timeline
Personalized cheeps (own cheeps + cheeps of followed users)
Click on "about me"
Display About Me page
Click "Delete" on a cheep
Are you sure?
no
yes
Deletes the cheep
Refreshes page
Click on Forget me
Are you sure?
no
yes
Deletes user and cheeps
Logs out and redirects to public timeline

Login - activity diagram

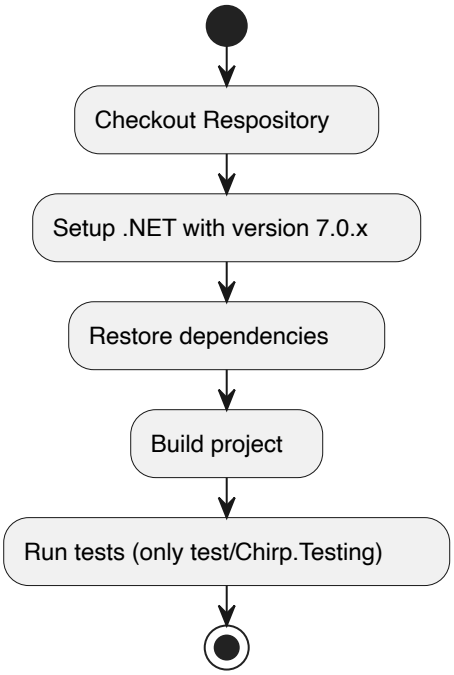


1.5 Sequence of functionality/calls trough *Chirp!*

2 Process

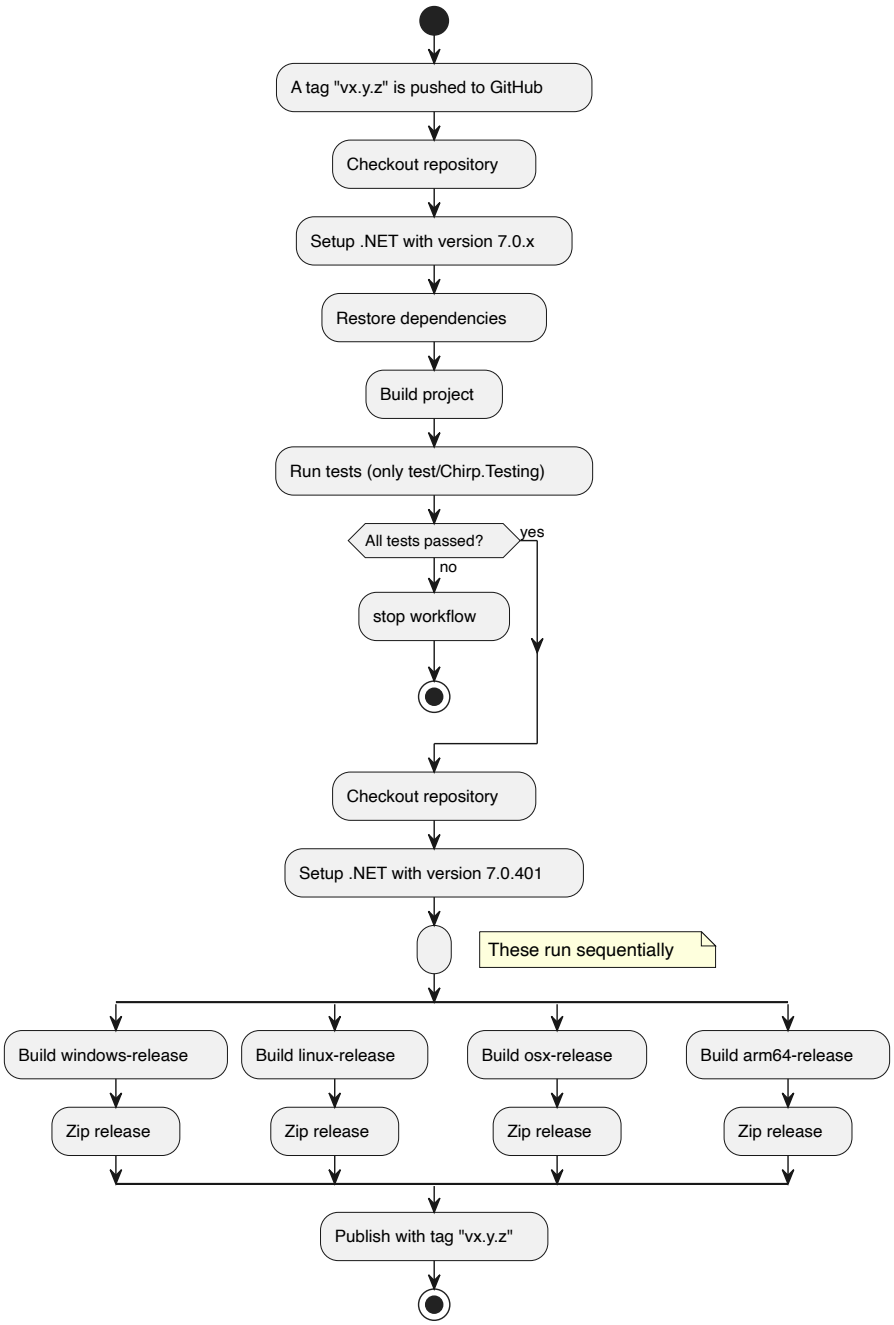
2.1 Build, test, release, and deployment

2.1.1 Building and testing the application



Whenever a push is made to GitHub, a workflow will start testing the application. This is the case for every branch and makes sure that whenever we make something - even if it is a new feature - the tests will run and the program won't fail when merged to main. This only includes the Chirp.Testing-folder as we had issues with GitHub-login for the UI-testing-workflow.

2.1.2 Automatiting GitHub releases

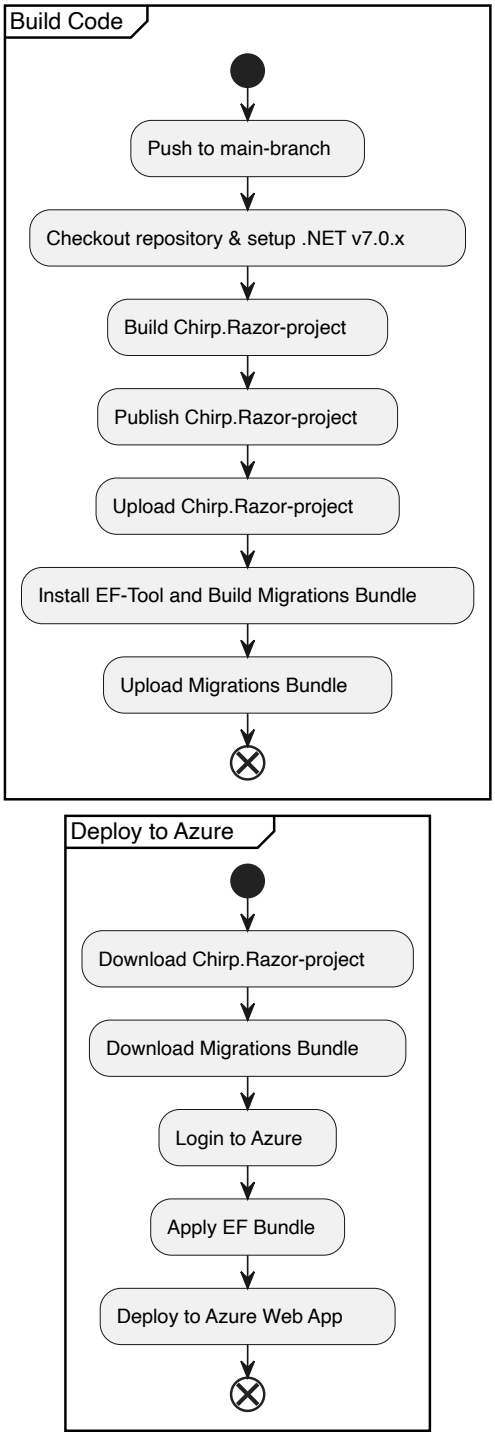


When a tag is pushed to GitHub using the syntax “vx.y.z”, a workflow is started

that first tests the application (same as building and testing the application). If the tests passes a new release is made where the versio number (tag) is the title. The workflow then builds the application for windows, linux, macOS and macArm separately, zips them and uploads them to the release-page.

We use semantic versioning as our "guide" on how to determine the versionnumber.

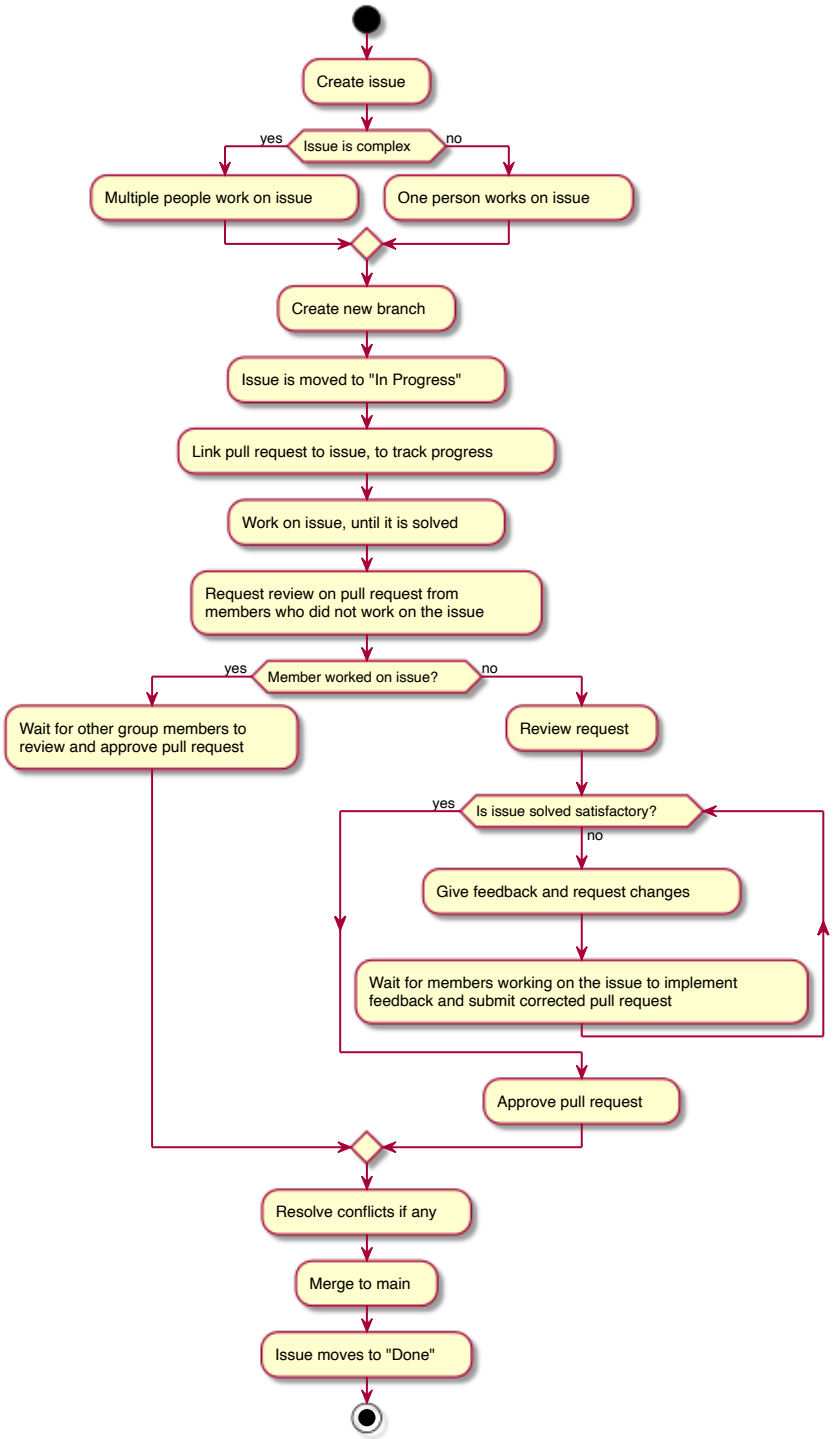
2.1.3 Automating Azure deployment



When a push is made to the main-branch a workflow deploying the application to our Azure server is started. The workflows first tasks builds the application, installs ef-tool and creates a migration bundle with the ef-tool. This is then uploaded.

A new task is then created that downloads the application and migrations bundle. The workflow then logs in to Azure and applies the bundle to our application through a connectionstring. Lastly the application is deployed to Azure using an Azure-webapp-deploy-action.

2.2 Team work


Create issue
Issue is complex
yes
no
Multiple people work on issue
One person works on issue
Create new branch
Issue is moved to "In Progress"
Link pull request to issue, to track progress
Work on issue, until it is solved
Request review on pull request from members who did not work on the issue
Member worked on issue?
yes
no
Wait for other group members to review and approve pull request
Review request
Is issue solved satisfactory?
yes
no
Give feedback and request changes
Wait for members working on the issue to implement feedback and submit corrected pull request
Approve pull request
Resolve conflicts if any
Merge to main
Issue moves to "Done"

When a new issue is created, it is automatically assigned to the "new" column in the project board. Members of the team can then assign themselves to the issue, with the amount of people working on it, being mainly dependent on the complexity of the issue. When a member assigns themselves to an issue, they move the issue to the "in progress" column. A new branch is created to work on the issue, and a pull request is linked to the issue, to track the progress on it. When the issue is considered completed, by the working members, the pull request is reviewed by the other members of the team. Members then review if they find the solution satisfactory. If the solution is not found satisfactory, they provide feedback, throughout their review and await the working members to consider feedback and submit a corrected pull request for review. If the solution is found satisfactory, the pull request is merged into the main branch. The issue is then moved to the "done" column.

2.3 How to make *Chirp!* work locally

To run our Chirp application locally it is required that you clone the project, which can be done by running the following command in the terminal:

```
git clone https://github.com/ITU-BDSA23-GROUP15/Chirp.git
```

As we are using a MSSQL database, it is required that you have docker installed on your machine. If you do not have docker installed, you can find a guide on how to install it here.

To download a Docker image, create a container for it and run it, run the following command in the terminal:

```
sudo docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=<YourStrong@Passw0rd>" \
   -p 1433:1433 --name sql1 --hostname sql1 \
   -d \
   mcr.microsoft.com/mssql/server:2022-latest
```

Replace `<YourStrong@Passw0rd>` with a password of your choice. This password will be used to access the database, so make sure to remember it.

We now need to set a Connection string with the password to the docker, that you have just set in the previous command. This is aved in the user secrets, which can be done by running the following command from a terminal at the root of the project:

```
dotnet user-secrets set "ConnectionStrings:ChirpDb" "Server=localhost;Database=ChirpDb;User
```

From the root of the project run the following command to run the project locally:

- dotnet run –project src/Chirp.Razor/
- Open a browser and go to http://localhost:5273/
- Do we need to do any migrations?

You should now expect to see the public timeline, stating at the top of the site, that you need to login to cheep, a login button should be available at the top right. The rest of the features on the site, will only become avaibalbe after logging in, which is done using your github account.

2.4 How to run test suite locally

2.4.1 Unit and integration tests

This project contains two test suites, as we have seperated our UI test, into a seperate test suite. To run our unit/integration tests, make sure docker is running, open a terminal at the root of the project and run the command:

```
dotnet test test/Chirp.Testing/
```

As our tests are run in a docker container, coupled with the fact that our program relies on a database, every method we have requires a read from the database when testing it. Therefore our unit tests are also integration tests, as they test the integration between our code and the database.

We did not manage to create enough tests for full code coverage, with some methods lacking a corresponding test. The methods that are tested are *CreateAuthor, CreateCheep* and *GetCheeps*.

2.4.2 UI tests (E2E)

Our UI test acts as an end to end test, as it tests the UI and functionality of our program as a whole by simulating user input. A few things is required to run the UI test suite. As our UI test is made with Playwright, the a supported browser for this needs to be installed. This can be ensured by running the following command:

```
npx playwright install
```

Furthermore, since an authenticated user is required to access most of the functionality of the program, and since our only means of authenticating is through Github, user secrets needs to be set up with a valid Github account. This can be done by running the following command:

```
dotnet user-secrets set "UserName" "<Insert your Github username>" --project test/Chirp.UI.T
```

Followed by:

```
dotnet user-secrets set "Password" "<Insert your Github password>" --project test/Chirp.UI.T
```

To run the UI test suite, you need to have a local instance of the program running. This can be done from a terminal at the root of the project by entering:

```
dotnet build
```

followed by:

```
dotnet run --project src/Chirp.Razor/
```

In another terminal at the root of the project, enter the following command to run the UI test suite:

```
dotnet test test/Chirp.UI.Testing/
```

If you have two factor authentication enabled on your Github account, the test suite needs to be run in headless mode, to allow for you to complete the authentication, when the test tries to log in. This can be done by running the following command instead of the one above:

```
dotnet test test/Chirp.UI.Testing/ -- Playwright.BrowserName=chromium Playwright.LaunchOptio
```

The SlowMo value can be adjusted to your liking, to slow down or speed up the test suite, to make it easier to follow what is happening.

The UI/E2E-test that this test suite contains, tests the overall functionality of our program along with the UI elements and the navigation between the URLs.

3 Ethics

3.1 License

We've chosen the MIT License, which is a permissive free software license, because of its limited restriction on reuse. In this project we wanted to encourage reuse of our code, and therefore we've chosen a license that allows for this. The MIT License is also a very common license, which makes it easy for others to understand the terms of the license.

3.2 LLMs, ChatGPT, CoPilot, and others

"State which LLM(s) were used during development of your project. In case you were not using any, just state so. In case you were using an LLM to support your development, briefly describe when and how it was applied. Reflect in writing to which degree the responses of the LLM were helpful. Discuss briefly if application of LLMs sped up your development or if the contrary was the case."

- Co-pilot: Has been useful for auto completing code that has already been made previously, especially the repository database functions.
- Co-pilot: Was not good at creating new code with new logic, almost always faulty and spent more time debugging autocompleted code than what it benefitted.
- ChatGPT: Has been used a lot as a starting point in debugging, when everything seemed overwhelming.
- ChatGPT: General questions about project topics, that helped narrow down the scope of the task and therefore researching became a lot easier.

- ChatGPT & Co-pilot: Has been used with varying succes, to understand errors provides by the compiler. Sometimes it was helpful, other times it simply did not understand the error and provided wrong suggestions.