Chirp! Project Report

ITU BDSA 2024 Group 4

Allan Petersen allp@itu.dk Bergur Davidsen berd@itu.dk
Lucas Venø Hansen lucasveha@gmail.com
Mikkel Clausen mikcl@itu.dk Victor Sforzini visf@itu.dk

0.1 Design and architecture

0.1.1 Domain model

0.1.2 Architecture — In the small

Below a diagram can be seen, showing the onion architecture of the Chirp program. The outer circles depend on the inner circles:

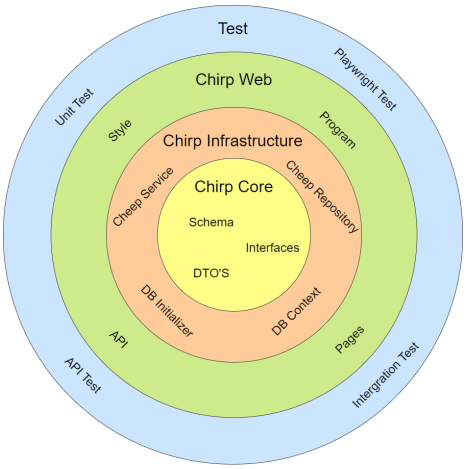

Figure 1: Onion architecture of Chirp program

In the **center** of the onion you find the Chirp Core, this is where the most primitive code lies, like objects and interfaces.

A step out of the Chirp Core, you find the Chirp Infrastructure. This is where the handling of the database is done, this includes retrieving, deleting

and updating data. Defining the database, and giving it some initial data is also done in Chirp Infrastructure.

In the **third layer** of the diagram, Chirp Web lies. This is where the webpage HTML is found, along with all the styling. The API which the web pages communicate with lies here as well.

Lastly there is the **outer layer**, naturally the tests are positioned here. The test suite includes Unit-, integration- and end2end tests. The end2end tests are done using Playwright.

0.1.3 Architecture of deployed application

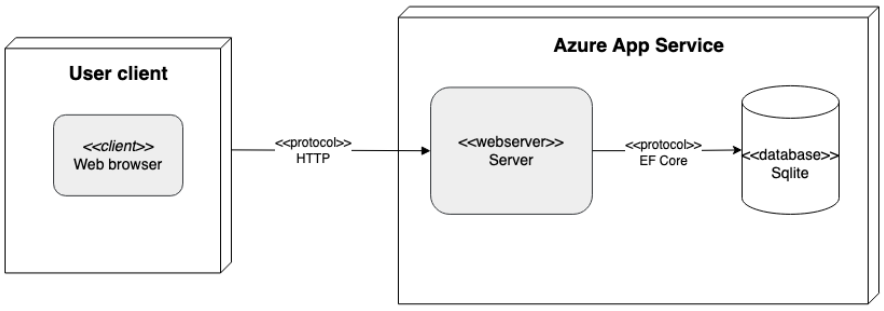

Figure 2: Architecture of deployed application

The deployed application follows the client-server architecture. The client communicates with the server through HTTP requests. The server is hosted on the Azure App service and the database is sqlite. Communication between the server and the database is done through Entity Framework Core.

0.1.4 User activities

The goal of this chapter is to show some core interactions from both an **unauthenticated user** and **authenticated user**. We make use of UML activity diagrams, these will visualize the states triggered by a user's actions.

First off we want to show what an unauthenticated user can do, and how the journey is for such a user to register.

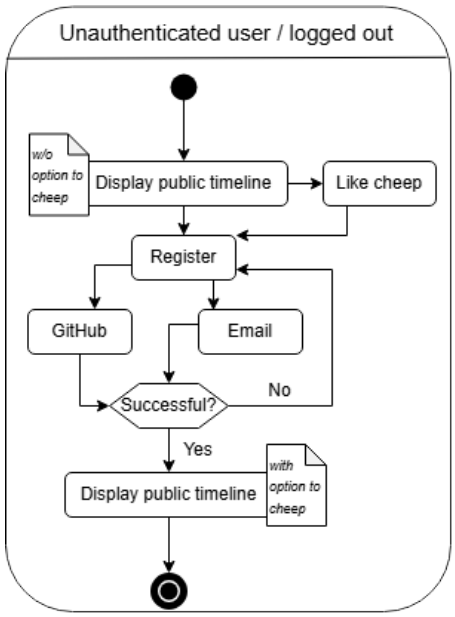

Figure 3: Unauthenticated user journey and register

This diagram show that a user can authenticate with both Email, and GitHub. Also, if you like a cheep from a user on the public timeline. It will simply not like it, but instead put you on the register page. Registering this way will give the same result as just navigating to the register page using the navigation bar.

When you are authenticated / logged in, we have 4 primary actions a user can do, respectively: Cheep, Like, Follow and Delete the account from the Chirp service.

The process of cheeping is shown in this diagram:

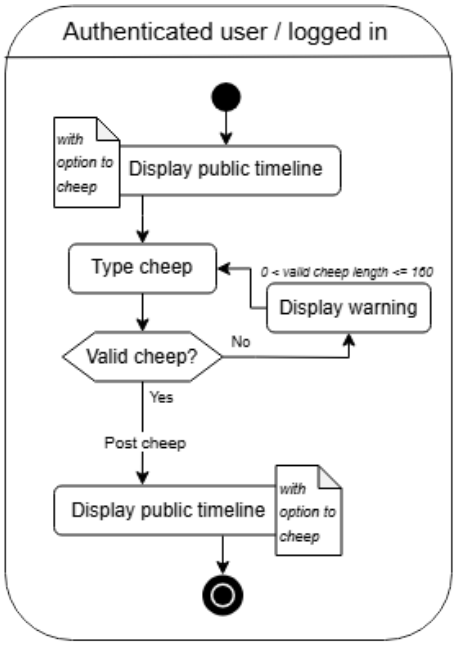

Figure 4: Cheeping journey and validation of cheep

A cheep is valid if its length, as show in the diagram, is between 0 and up to and including 160 characters. If you were to click the share button, with and empty text field, a warning will pop up. A warning pop up won't explicitly be shown to the user for cheeps longer than 160 characters, we simply show the length counter on screen, and don't allow for more characters, in both front- and backend.

The users also need to like cheeps, for that action we have this diagram:

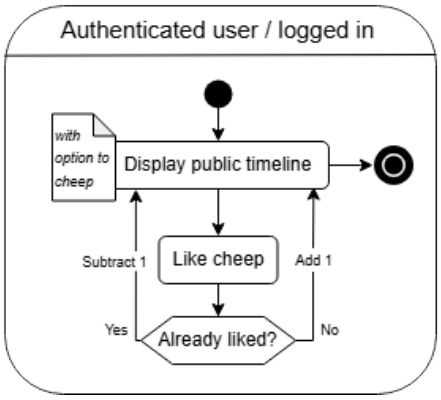

Figure 5: CLiking cheeps

The 'heart' button we have besides each cheep is essentially a toggle for likes on the given cheep. And as showed in the diagram, each user can only like any given cheep once. It is important to note, as of now the liking of a cheep will result in the page redirecting you to the root page (page 1), even though you might be on for instance page 6. There is an obvious room for improvement, and the task is currently a task in the project board.

Next up we want to show the journey of a user following another user.

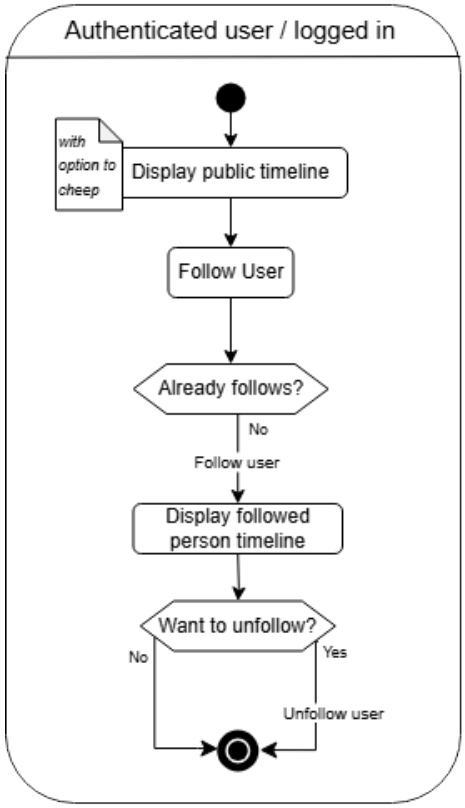

Figure 6: Following users

The journey of following a user, is close to the same as liking cheeps, as both are 'toggles'. The only difference is that we decided to show the newly followed users profile after the follow action. Which eliminates the issue we are having with liking cheeps far down on the public timeline, and wanting to scroll beyond that point afterwards. This does then create the issue with wanting to continue scrolling after following. But this navigation to the private timeline of the newly followed user, is a conscious decision.

Lastly it is important for us to show how the user can delete, and see the data we have gathered.

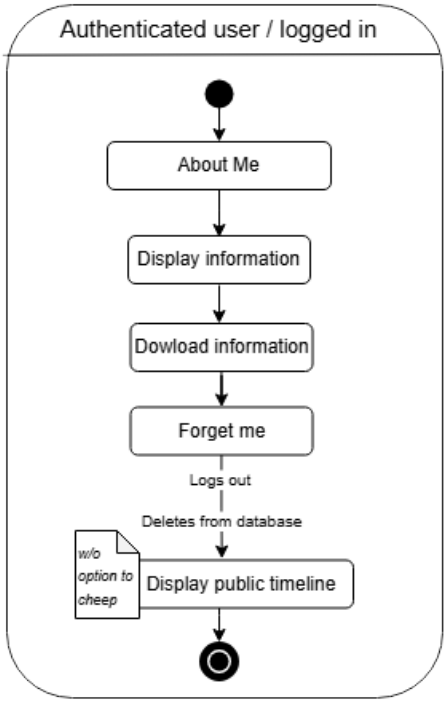

Figure 7: Deleting the user and download data

The linear diagram is pretty much self-explanatory, but we feel it is important to show either way, since this is last key feature for a user to experience.

0.1.5 Sequence of functionality/calls trough *Chirp!*

0.2 Process

0.2.1 Build, test, release, and deployment

All of the build, test, release, and deployment is done using GitHub Actions.

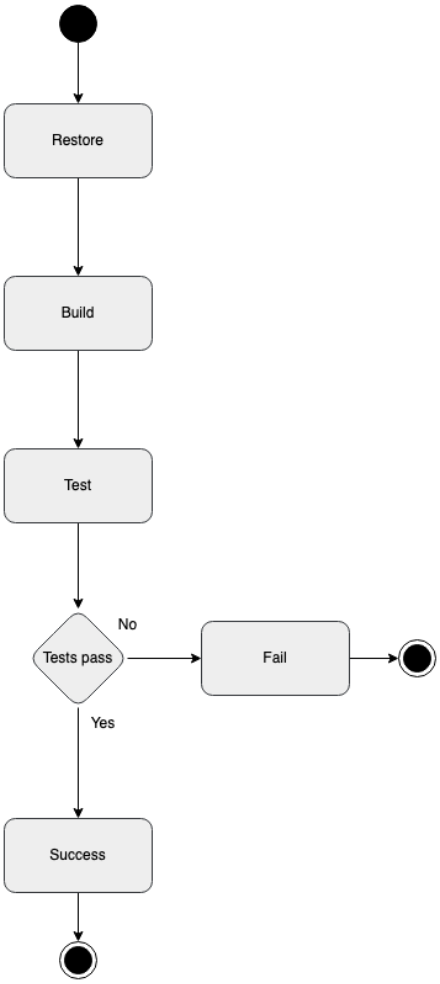

Figure 8: Build and test flow

The build and test flow is one out of two flows, that run when a pull request is made to the main branch. This flow will build and run the test suite, and if the test suite passes, the flow will be marked as successful. If the test suite fails, the flow will be marked as failed.

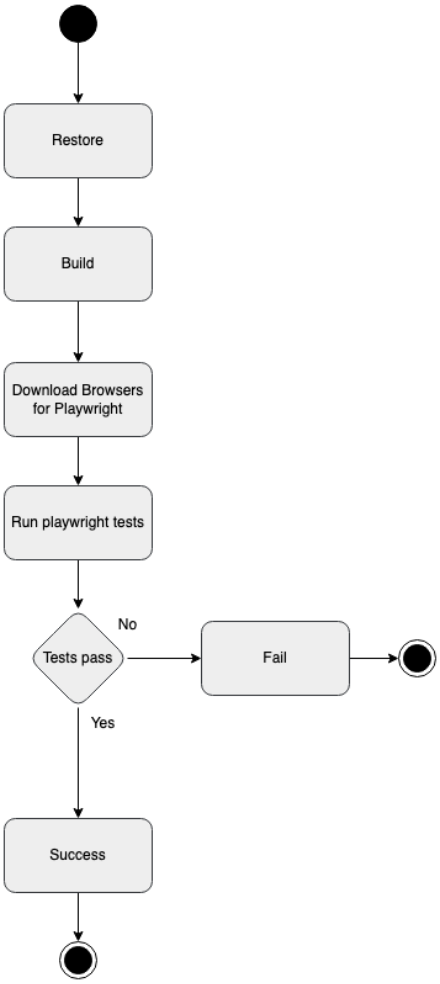

Figure 9: Playwright test flow

The Playwright test flow is the second flow that run when a pull request is made to the main branch. This flow will build and run the UI tests and end2end test with Playwright, and if the test suite passes, the flow will be marked as successful. If the test suite fails, the flow will be marked as failed.

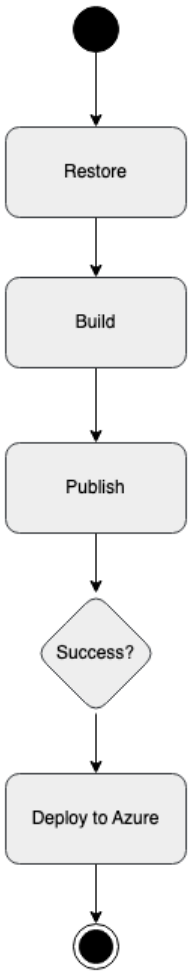

Figure 10: Deployment flow

The deployment flow is the flow that runs when a pull request is merged into the main branch. This flow will build the project, run the `dotnet publish` command, and deploy the project to the Azure App Service.

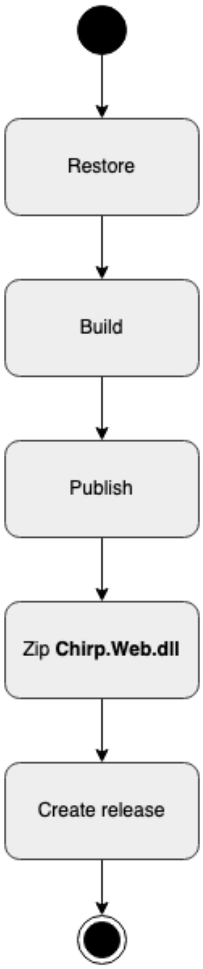

Figure 11: Release flow

The release flow is the flow that runs when a version is tagged in the repository. This flow will build the project, run the `dotnet publish` command, and then create a zip folder with the *Chirp.Web.dll* file. This zip folder is then uploaded to the GitHub release page, under the tag that was created.

0.2.2 Team work

As of writing the report we have 7 issues that are still waiting to be resolved. Most of these are not features that changes functionalities for the end-user.

The most prominent issue we are working on to be resolved, is shown under 'In

Progress'. We need to establish more tests that fully tests the service.

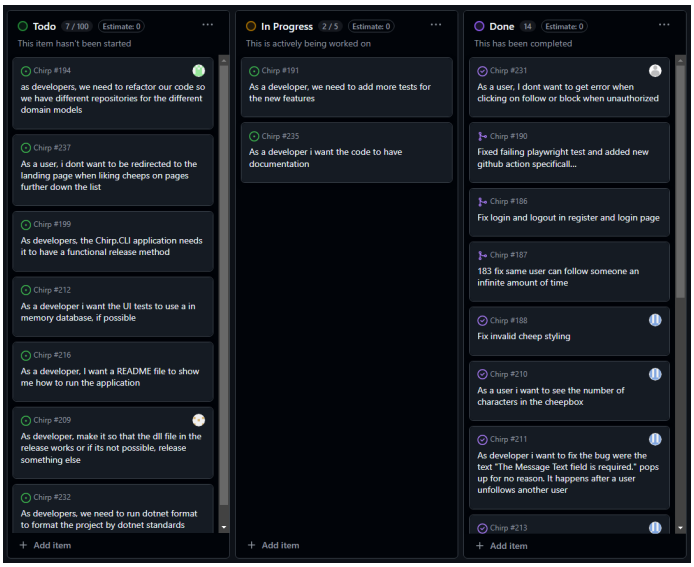

Figure 12: Flow of activites, issue to merge

Most if not all the development was done in the way the diagram below shows. Some things to point out is we mob programmed being 3-5 people together in the beginning. This was a decision we made since we needed some individual tasks to be done in order to continue. That way the 'code review' part of the pipeline became more or less just an obstacle instead of a tool.

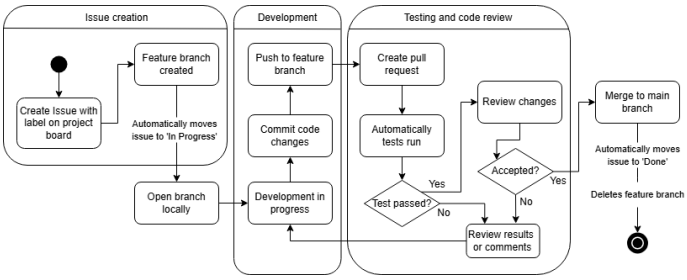

Figure 13: Flow of activites, issue to merge

Later on it became a tool we used and relied on. Mostly because we instead of mob programmed worked in pairs, or even sometimes on our own. So this diagram and code review mostly represents our late work on the project. 'Issue creation' and 'Development' has stayed the same since the beginning.

0.2.3 How to make *Chirp!* work locally

0.2.4 How to run test suite locally

0.3 Ethics

0.3.1 License

We decided to go ahead with and use the **MIT License**

0.3.2 LLMs, ChatGPT, CoPilot, and others

In the development of our project we used ChatGPT, and when we did so, we made sure to add ChatGPT as a co-author in our git commit message like so:

```
ChatGPT <>
```

ChatGPT was very helpful when used to create simple code parts and debug some. On the other hand the ChatGPT was not helpful with complex code questions. Therefore, we ended up finding it mostly useful for us to understand parts of the code and guide us on where to start on complex tasks.

However, we also experienced some negatives when using ChatGPT. It could sometimes go in a spiral, in cases like this we would look at the slides and on the web for help. We also gave Gemini some use sometimes when ChatGPT was not helpful, we did however never use any of the provided code, so it never got to be a co-author.

For the most part the use of LLMs sped up our development, however sometimes they were sent into a spiral and hallucinated, which could confuse us more. So we experienced the limitations of LLMs and got to learnt how to use them more efficiently.