

Chirp! Project Report

ITU BDSA 2023 Group 15

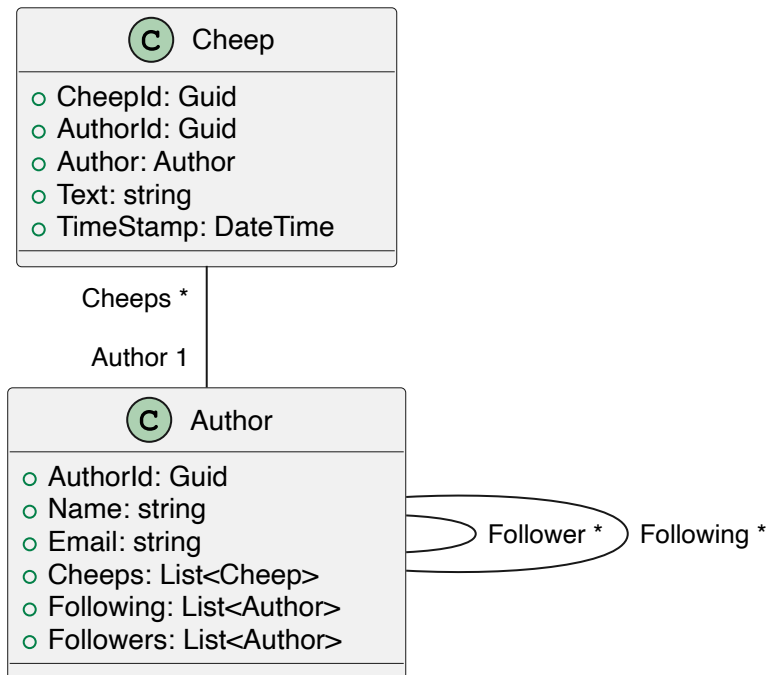
Daniel Sølvsten Millard dmil@itu.dk
Frederik Lund Rosenlund frlr@itu.dk
Jacob Vortscir Pærregaard jacp@itu.dk
Mads Christian Nørklit Jensen macj@itu.dk
Rasmus Lundahl Nielsen raln@itu.dk

- Design and Architecture of *Chirp!*
 - Domain model
 - Architecture — In the small
 - Architecture of deployed application
 - User activities
 - Sequence of functionality/calls through *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.”

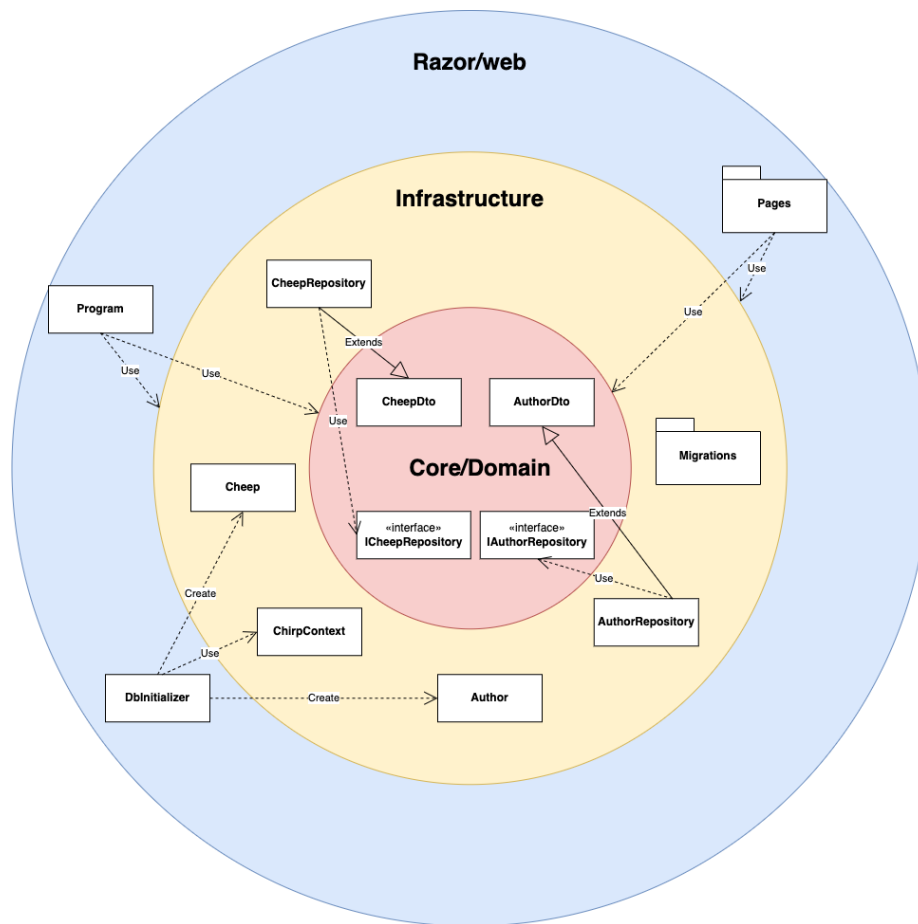
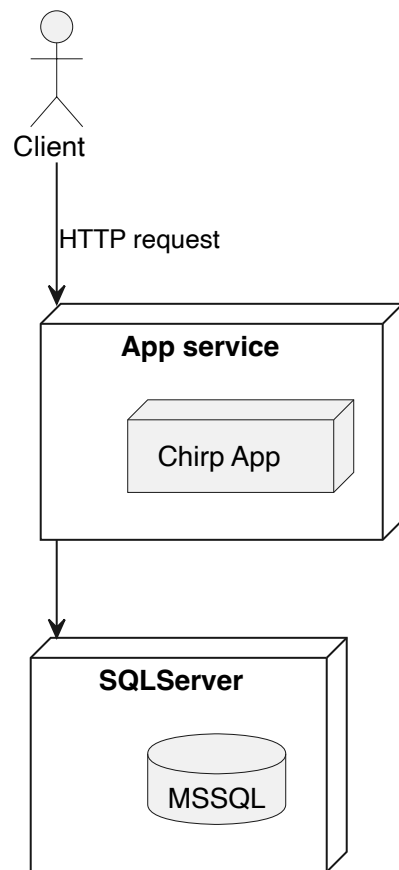
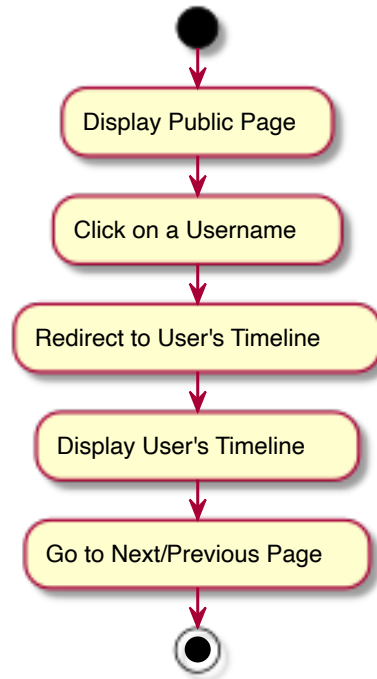


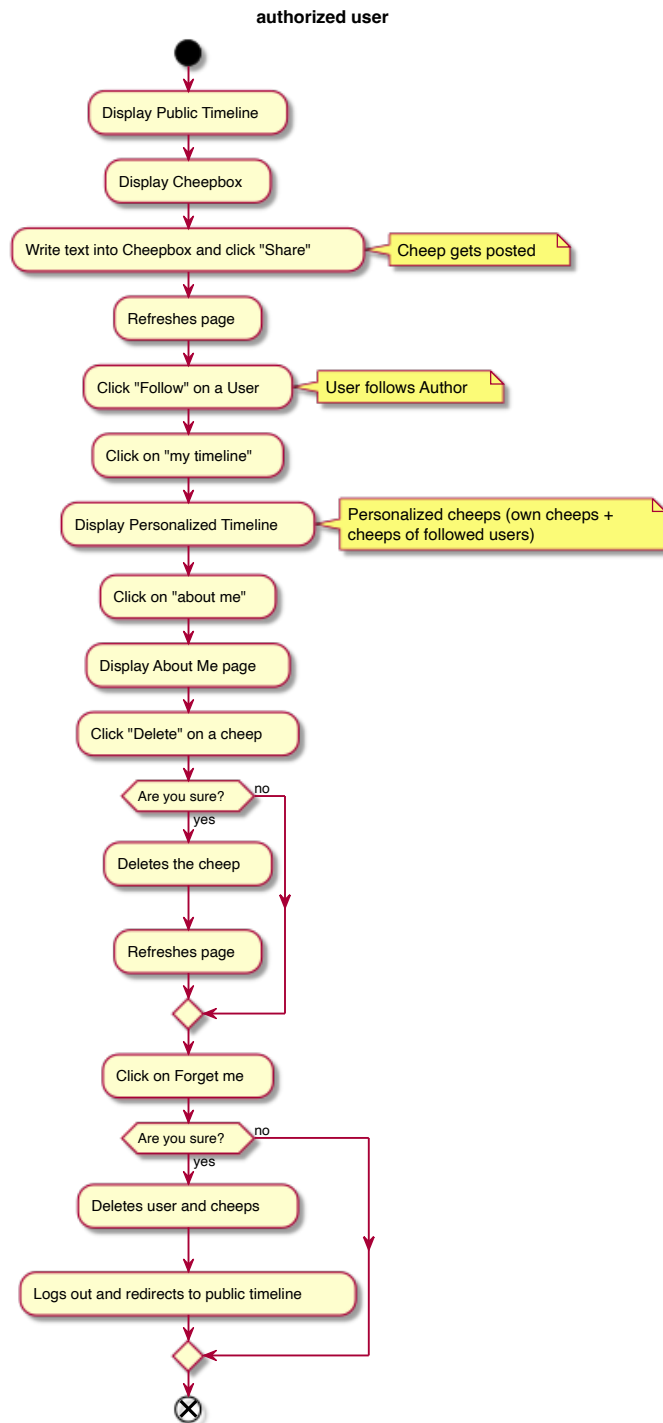
Figure 1: onion_architecture



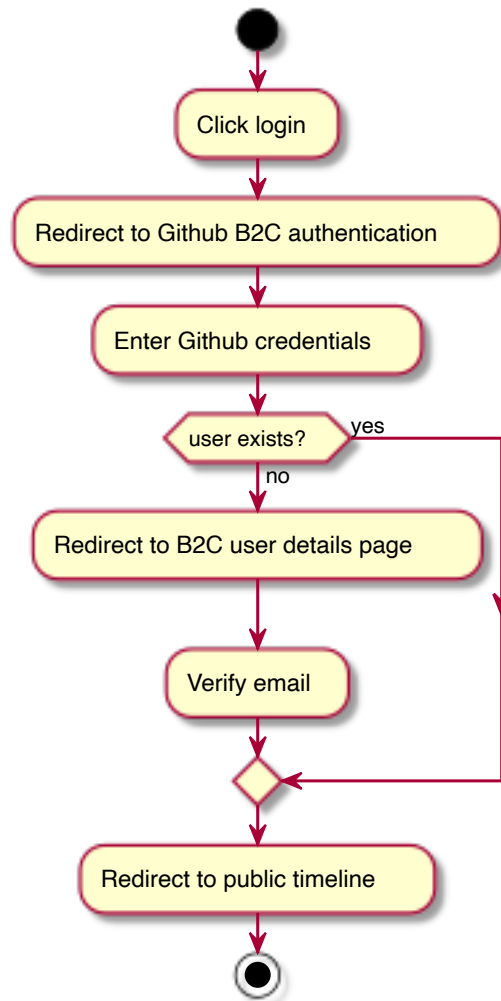
1.4 User activities

Unauthorized - activity diagram





Login - activity diagram

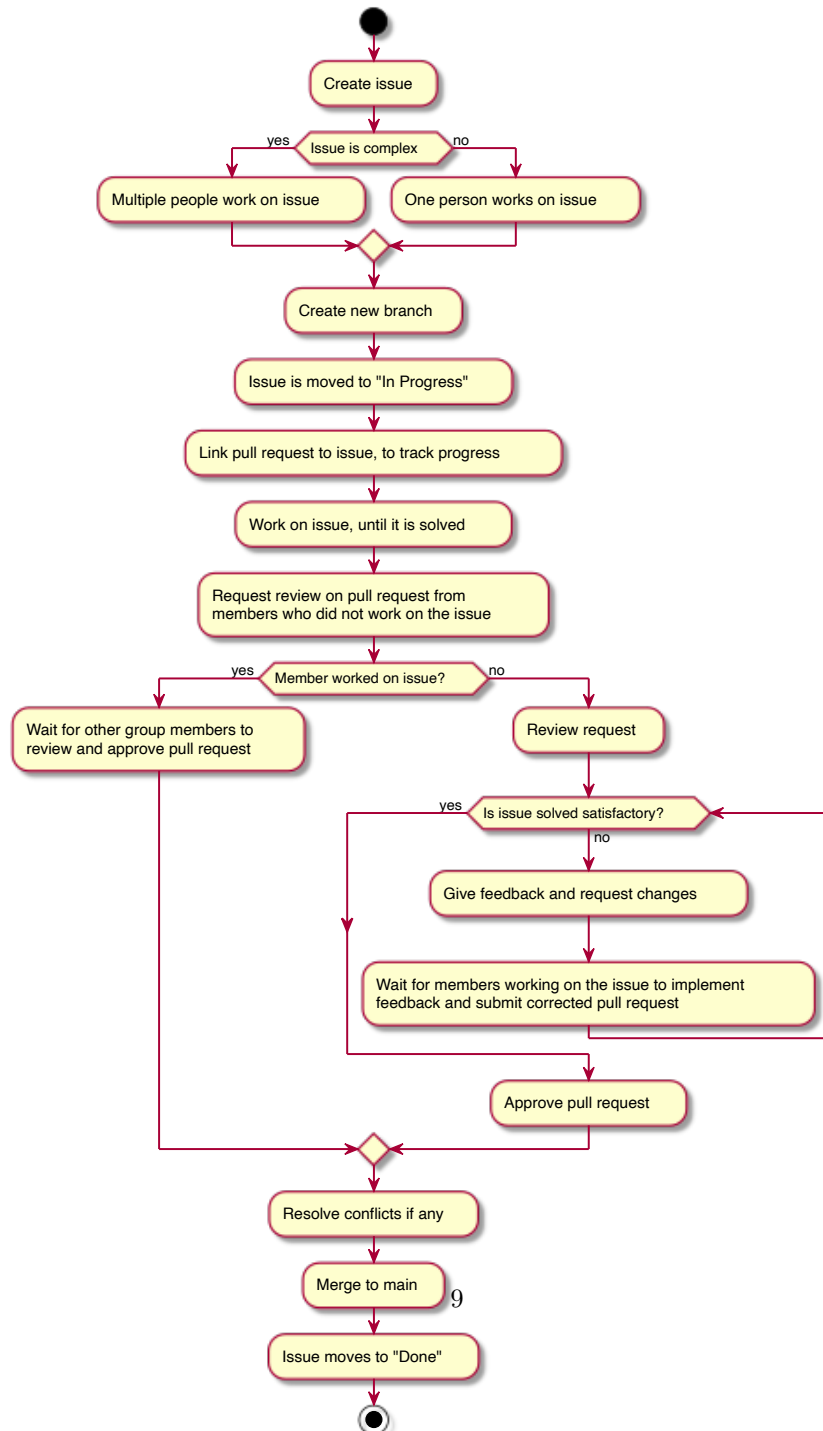


1.5 Sequence of functionality/calls through *Chirp!*

2 Process

2.1 Build, test, release, and deployment

2.2 Team work



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

- Probably something about docker? The program cant run without docker I think?

To clone the project run the following command in the terminal:

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

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 avaialbe 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 simply 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.7
```

Followed by:

```
dotnet user-secrets set "Password" "<Insert your Github password>" --project test/Chirp.UI.7
```

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.LaunchOpti
```

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.