Chirp! Project Report

ITU BDSA 2024 Group 2

Anders Georg Frølich Hansen ageh@itu.dk
Mads Østrup moes@itu.dk
Mathias Bindslev Hansen bimh@itu.dk
Nikolai Tilgreen Nielsen nitn@itu.dk
Rasmus Mygind Poulsen rpou@itu.dk

1 Introduction to Chirp

2 Design and Architecture of *Chirp!*

2.1 Domain model

Here comes a description of our domain model.

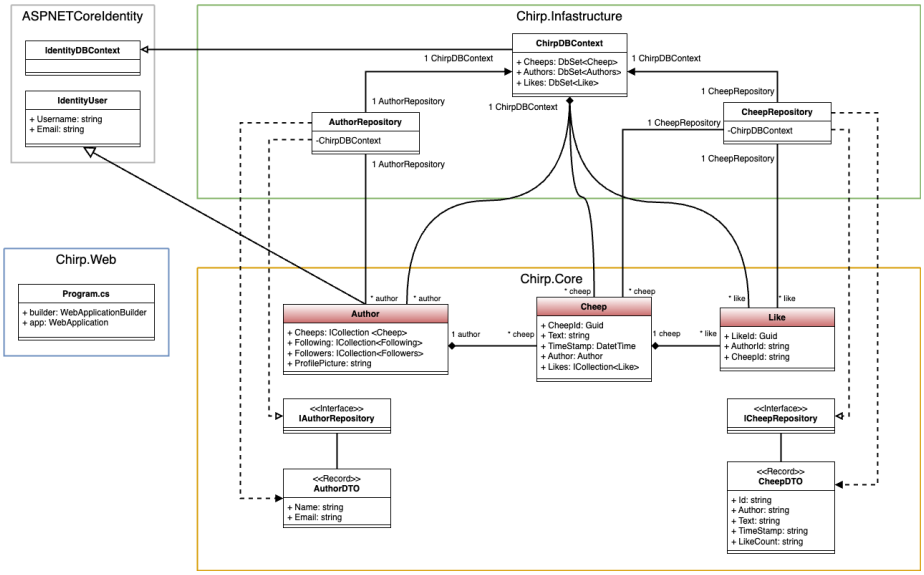

Figure 1: Illustration of domain model

2.2 Architecture — In the small

Onion Architecture

The Chirp! project implements the onion architecture, which is seen in the composition of the code. The code is split into three different layers: * A chirp **core** layer, that is the domain of the project, and how it differentiates from other programs. * A chirp **infrastructure** layer, that is responsible for manipulation and retrieval of data. True to the onion architecture, this layer is built upon the **core** layer, which means that **infrastructure** depends on **core**. * A chirp **web** layer, that is responsible for the UI of the project. Again, true to onion architecture this layer depends on **core** and **infrastructure**.

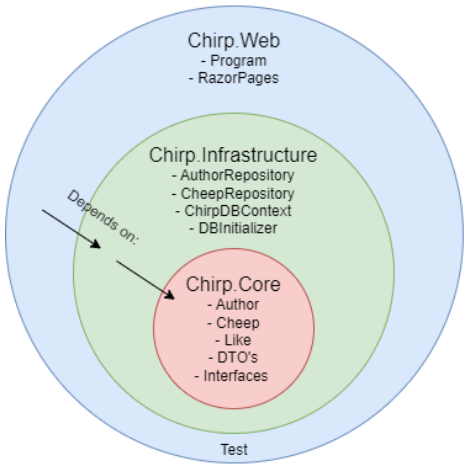

Figure 2: Organization of codebase

Having the project split up into separate layers is optimal for testing. Since they are loosely coupled the core of the project can be tested independently without the other layers. This makes a foundation for good testing. In addition this means that the outer layers can be modified without affecting the inner layers. This results in easy scalability and maintainability. All in all this architecture greatly benefits the project in the long run.

2.3 Architecture of deployed application

The application follows a client-server architecture. The server is a web application deployed on Azure App Service. It provides the necessary interface and API endpoints for communication with the client.

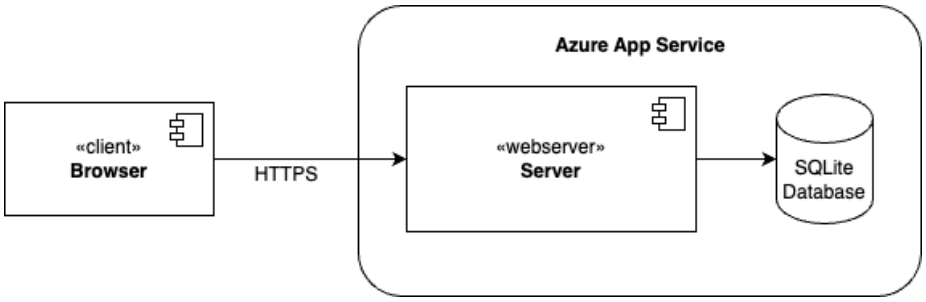

Figure 3: Deployed application

2.4 User activities

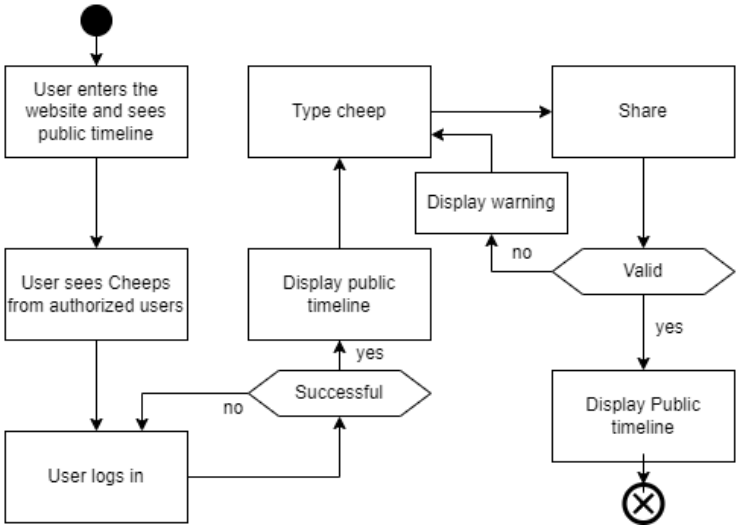

Figure 4: User Story for making a cheep

2.5 Sequence of functionality/calls through *Chirp!*

The diagram of sequences shown below illustrates a sequence of calls in the Chirp application initiated by the user, for both an unauthenticated user and an authenticated user.

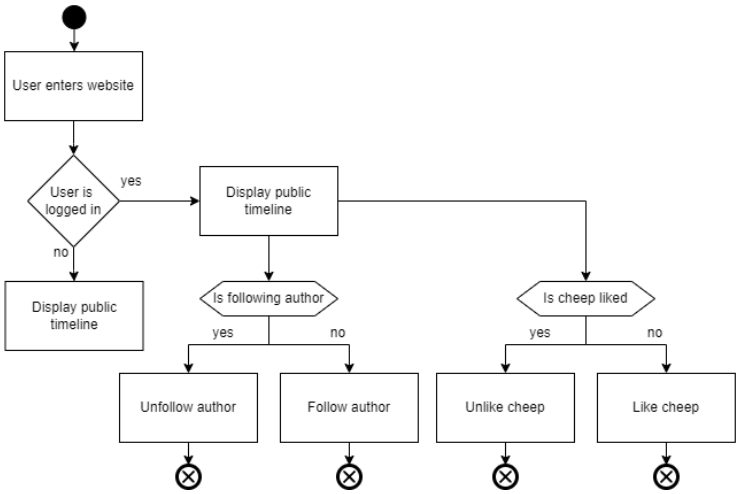

Figure 5: User story for following and liking

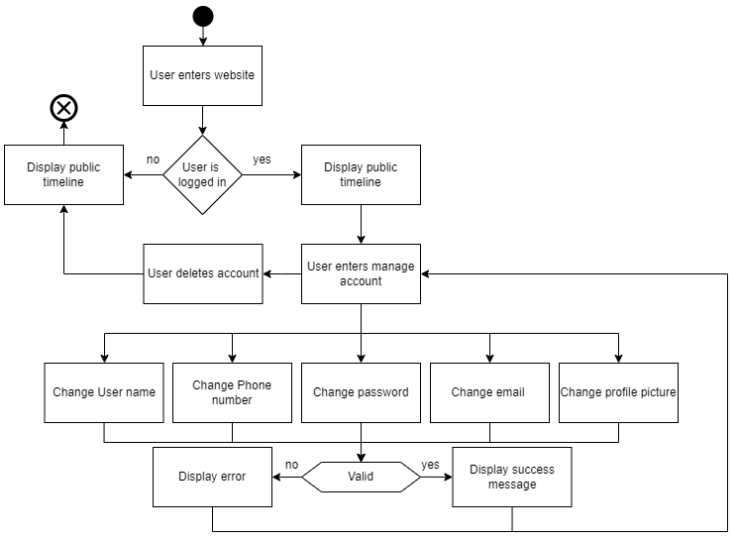

Figure 6: User story for managing personal information

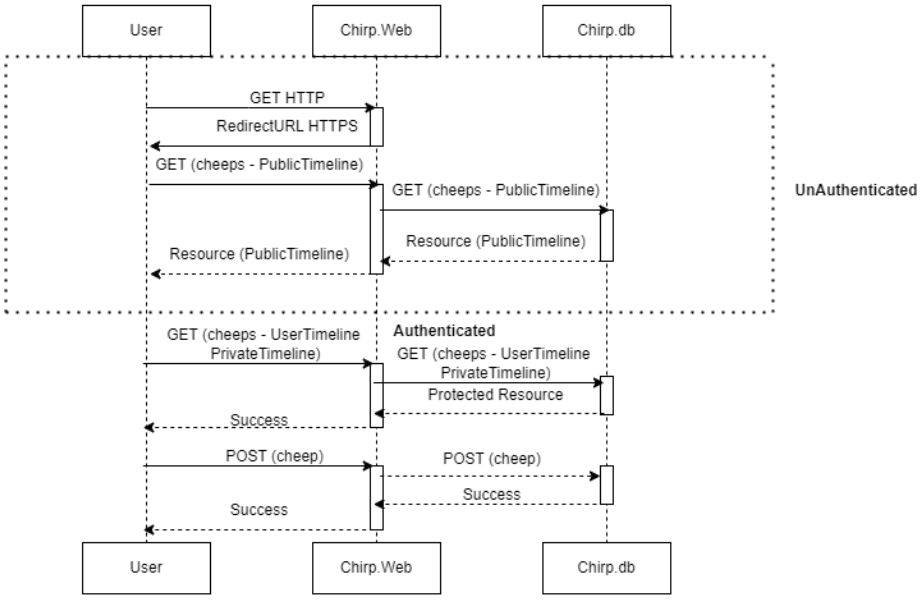

Figure 7: Flow of messages and data through Chirp

3 Process

3.1 Build, test, release, and deployment

Build and test The image below illustrates what happens when either a push is committed or a pull request is made. * `dotnet restore`, this command restores the dependencies and tools of the project. * `dotnet build`, which then builds the project and its dependencies. * The yml installs playwright, which is necessary for running test on GitHub. * `dotnet test`, where all of the tests will run and show if any test will fail and which succeed.

Deployment When pushing to main, the build and test flows are run. It also logs in to Azure by using the Azure secrets and deploys Chirp to the Azure web service.

Release When pushing with a new tag, a release is made to GitHub: * The application is published for Windows, Linux and MacOS. * The artifacts for each publish is zipped to its own zip file * The zip files are released to GitHub under a new release.

3.2 Team work

All issues related to the mandatory project work as well as our own extra features are done.

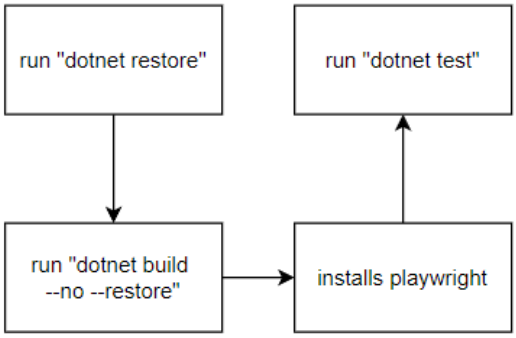

Figure 8: Build and test workflow

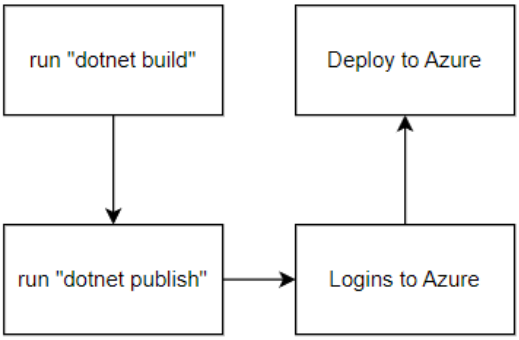

Figure 9: Deployment workflow

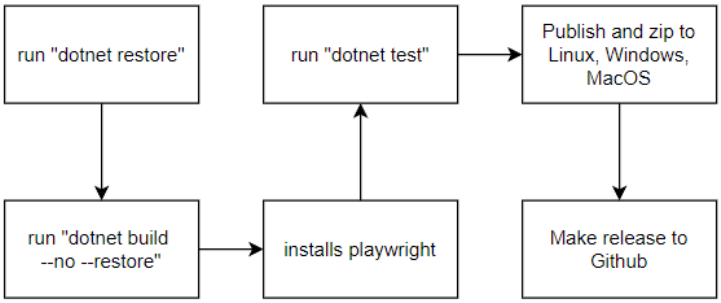

Figure 10: Release workflow

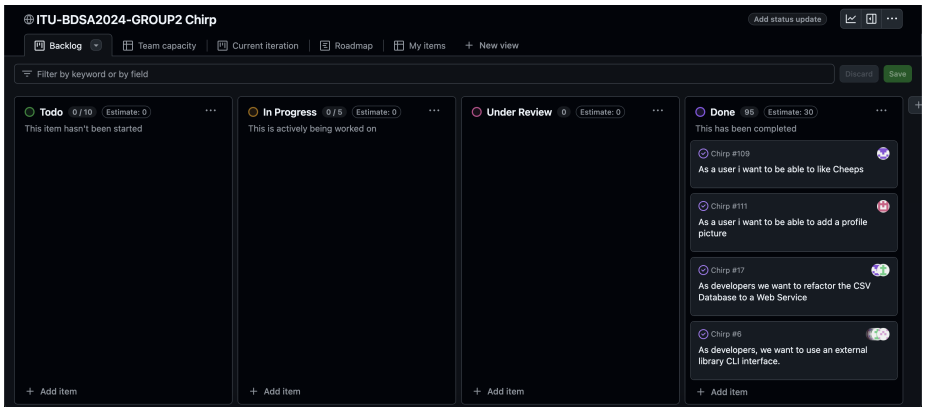

Figure 11: Project board before hand-in.

From issue to main All new features and enhancements are added as issues to the `ITU-BDSA2024-GROUP2 Chirp` backlog in GitHub. Issues follow the workflow as depicted below until they are merged to the main branch and deployed to Azure.

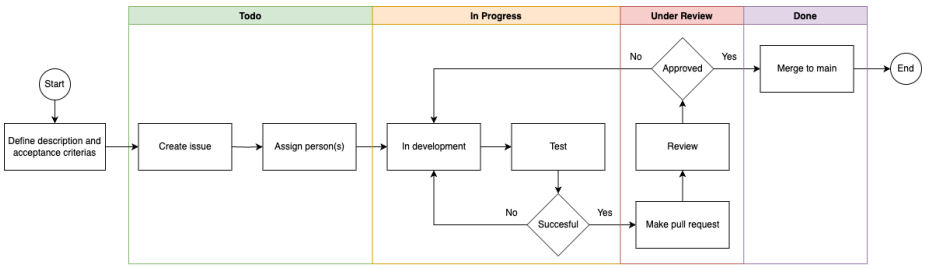

Figure 12: Development workflow.

Collaboration Depending on the extend of the task, each issue is assigned to the responsible person(s). When developing in teams we have made extensive use of pair programming as well as code reviews on an external monitor.

Code reviews To ensure software quality and participation, commits have undergone a review process from a team member who have not worked on the issue. We have used pull requests for this purpose.

3.3 How to make *Chirp!* work locally

Running *Chirp!* locally

1. Clone the repository by running the following command `git clone https://github.com/ITU-BDSA2024-GROUP2/Chirp.git`.

2. Setup program secrets.

   2.1. Go to the root of the project `/Chirp`.

   2.2. Type the following command:

ClientID: `dotnet user-secrets set "authentication_github_clientId" "Ov23li6lCKKhGGXefnEf"`

ClientSecret: `dotnet user-secrets set "authentication_github_clientSecret" "460047215cdea005fd386c508c0ae3dc1412c20d"` 3. Cd into the folder `Chirp/src/Chirp.Web`. 4. Type `dotnet run`.

Note: You have to use dotnet 8 for the program to function properly.

3.4 How to run test suite locally

Running *Chirp!* tests locally

1. Cd into the folder `Chirp/src/Chirp.Web`.
2. Run the following command `pwsh bin/Debug/net8.0/playwright.ps1 install --with-deps`.
3. Go to the root of the project `/Chirp`.
4. Type `dotnet test`.

Note: If some tests are failing, try deleting the database from the `src/Chirp.Web` folder. Additionally check if there is a .db file in the `test/Chirp.API.Tests/bin`. If there is one, delete that too. Then run the tests again.

There are four kinds of tests: * **UNIT tests** are testing 1 method or one feature. This could be testing changing profile picture. * **INTEGRATION tests** are testing a combination of methods, or a component of the website. This could be testing the register form. * **END TO END tests** are testing a full user experience. This could be making a profile, sending a cheep and entering the private timeline. * **UI tests** were made using playwright, for easy navigation through the UI elements.

4 Ethics

4.1 License

Chirp uses the MIT License.

4.2 LLMs, ChatGPT, CoPilot, and others

During the preparation of Chirp we have used ChatGPT to assist our development and learning process. In most cases we have been cautious with our usage of ChatGPT and always consulted the official documentation, TAs or websites like Stack Overflow first.

The primary goal and intend of using an LLM is to improve our understanding of the course material and frameworks used. We think that the responses from ChatGPT were especially helpful in understanding complex concepts. Contrary, responses from ChatGPT were not very helpful in speeding up development as it overcomplicates many aspects and does not have a deep understanding of our domain model like we do ourselves.