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 Design and Architecture of *Chirp!*

1.1 Domain model

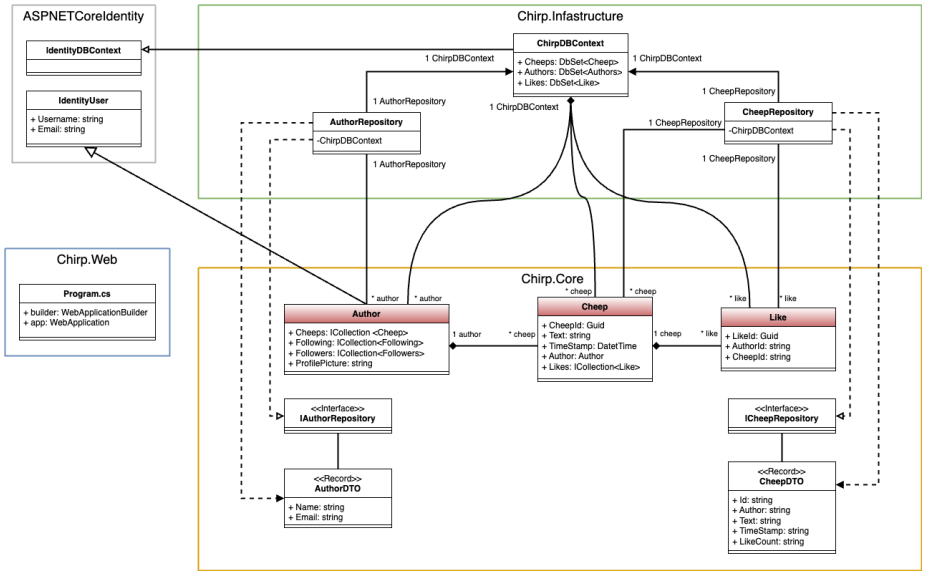


Figure 1: Illustration of domain model (highlighted in red).

1.2 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 contains the domain model of the project.
- 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**.

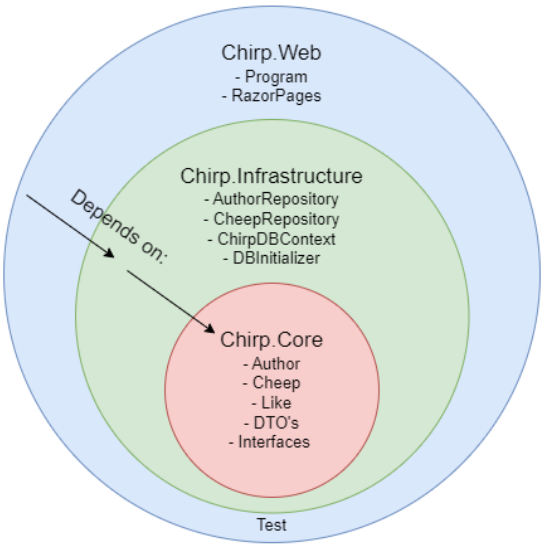


Figure 2: Organization of codebase.

Having the project split up into separate layers is optimal for testing. Since the layers are loosely coupled the core of the project can be tested independently. 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.

1.3 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.

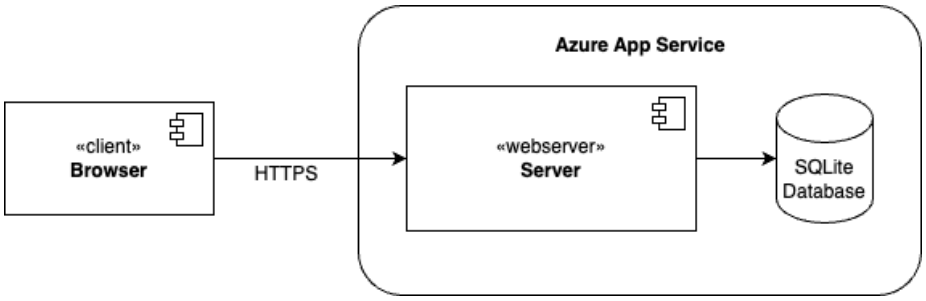


Figure 3: Deployed application.

1.4 User activities

Below are illustrated examples of different user activities.

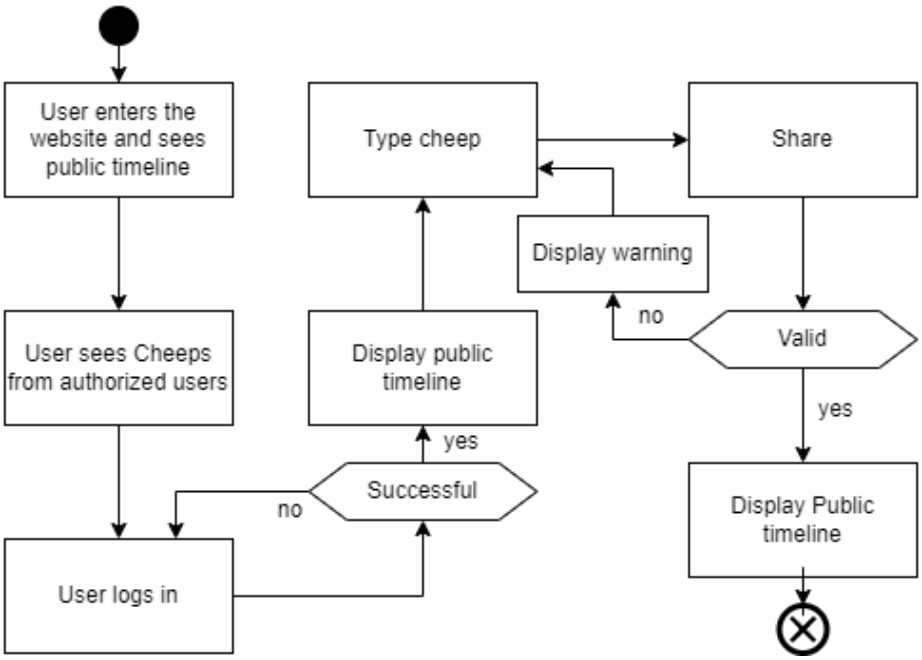


Figure 4: User Story for making a cheep.

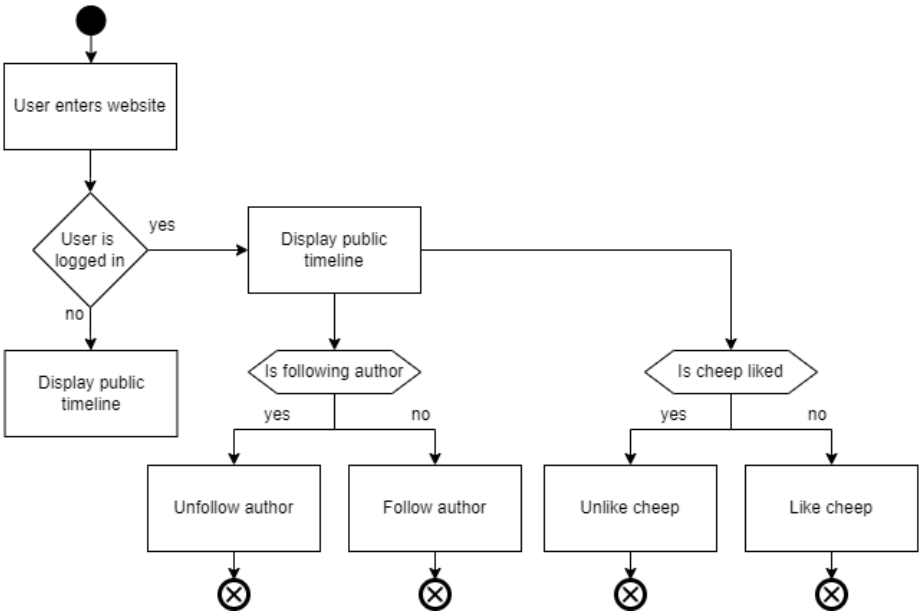


Figure 5: User story for follow and like.

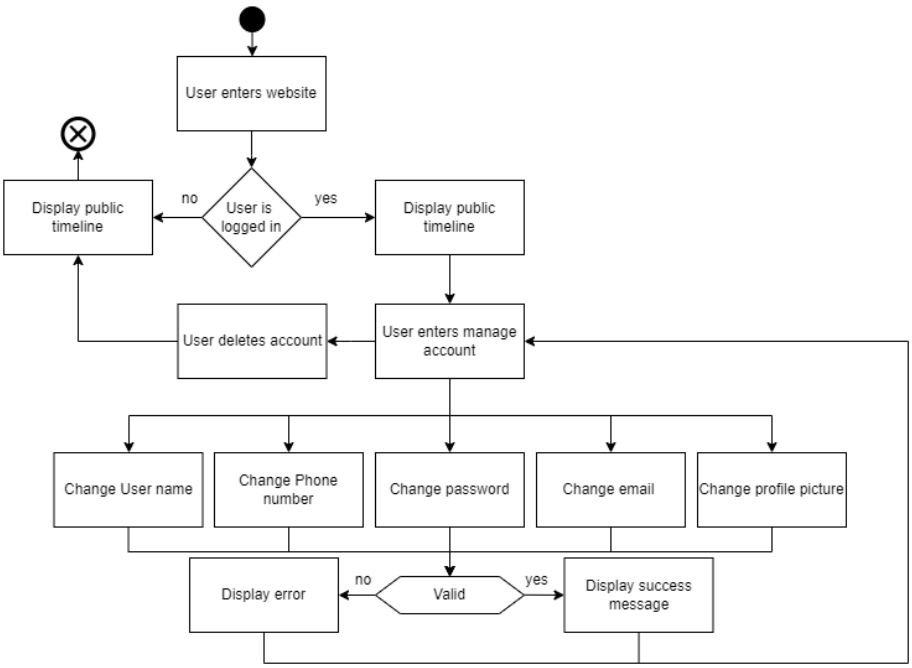


Figure 6: User story for managing personal information.

1.5 Sequence of functionality/calls through *Chirp!*

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

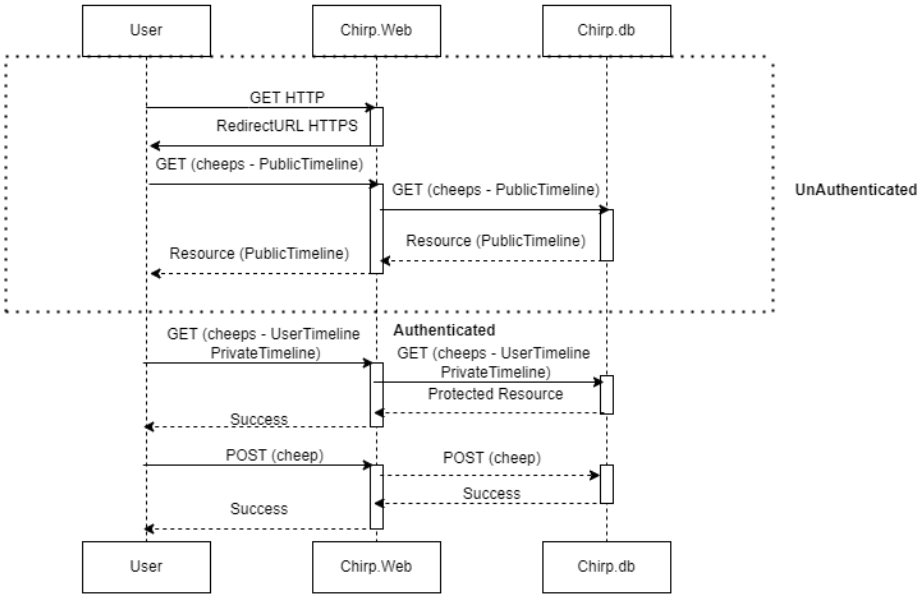


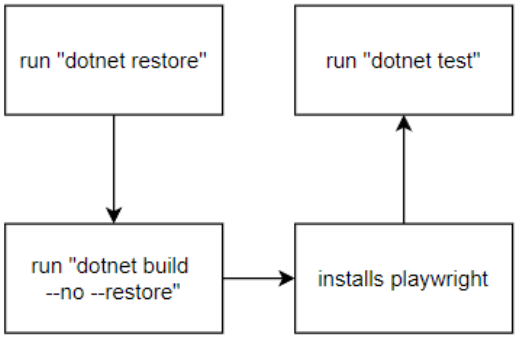
Figure 7: Flow of calls and data through Chirp!

2 Process

2.1 Build, test, release, and deployment

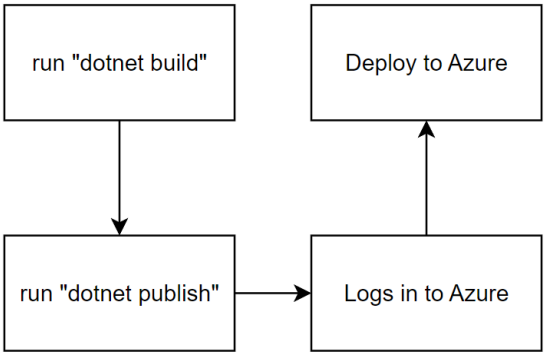
Build and test

When a push or pull request is made, a build and test workflow is triggered. This workflow sets up the .NET environment, restores dependencies, builds the project and installs Playwright browsers necessary for UI tests. Lastly it runs all the tests.



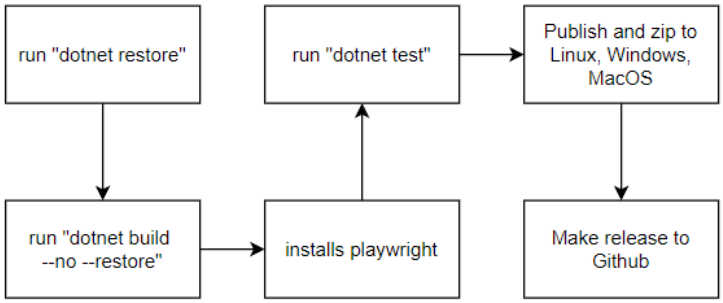
Deployment

When a push is made to the main branch, a deployment workflow is triggered. This workflow builds the app, uploads the build artifacts, and deploys them to Azure using provided secrets.



Release

When a new tag is pushed to the repository, a release workflow is triggered. This workflow builds the app for multiple platforms (Linux, Windows, macOS x64, and macOS ARM), zips the artifacts, and publishes them as part of a GitHub release corresponding to the new version.



2.2 Team work

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

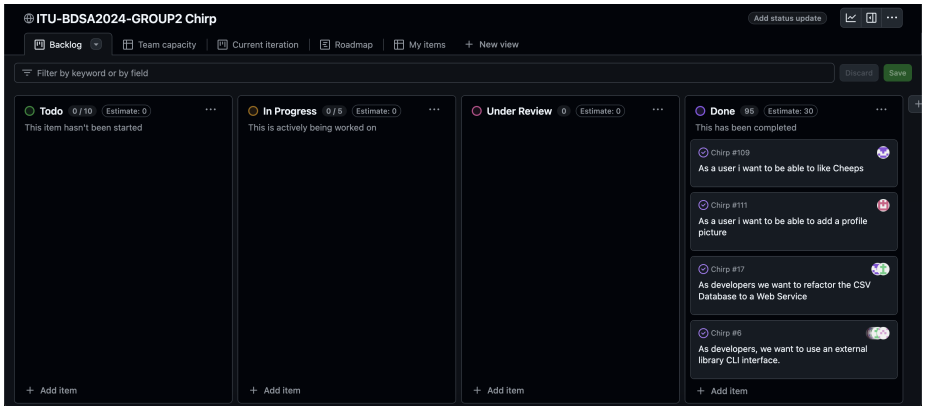


Figure 8: Project board before hand-in.

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.

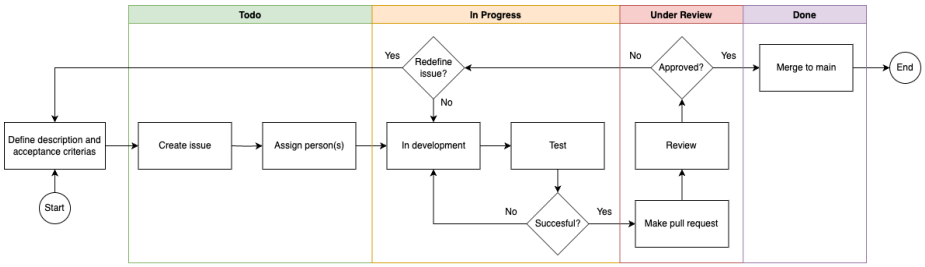


Figure 9: Development workflow.

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.

2.3 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 commands:

   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.

2.4 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 fail the database file might be deprecated. To fix this delete the database file 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.

The test suites contain the following tests:

Unit tests

Testing individual methods in isolation. E.g. creating a new cheep or author in the database.

Integration tests

Testing a combination of methods or a component of the website. E.g. testing that liking a cheep stores the like in the database.

End-to-end tests

Testing a complete user journey through the application. E.g. logging in, making a cheep, and having the cheep displayed in the respective authors private timeline.

UI tests

Made using Playwright. Testing and simulating user interactions and navigation through the user interface. E.g. testing that the submit button is accessible when sharing a cheep. These test are mostly designed as integration or end-to-end tests.

3 Ethics

3.1 License

Chirp! uses the MIT License.

3.2 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. It overcomplicates many aspects and does not have a deep understanding of our domain model like we do ourselves.