Chirp! Project Report

ITU BDSA 2024 Group `10`

Christoffer Grünberg gryn@itu.dk
Rasmus Rosenmejer Larsen rarl@itu.dk
Mathias Labori Olsen mlao@itu.dk
Alex Tilgreen Mogensen alext@itu.dk
Anthon Castillo Hertzum acah@itu.dk
Bryce Raj Karnikar brka@itu.dk

Contents

1 Design and Architecture of *Chirp!*

1.1 Domain model

The *Chirp!* domain model is centered around the Author class. Authors inherit traits for account management from IdentityUser. Authors can create Cheeps and interact with them through Likes or Comments. Each Author maintains a list of Likes and Comments, enabling the logging of which Authors have interacted with which Cheeps. Furthermore, Authors can follow other Authors, storing a list of the Authors they follow and the Authors who follow them.

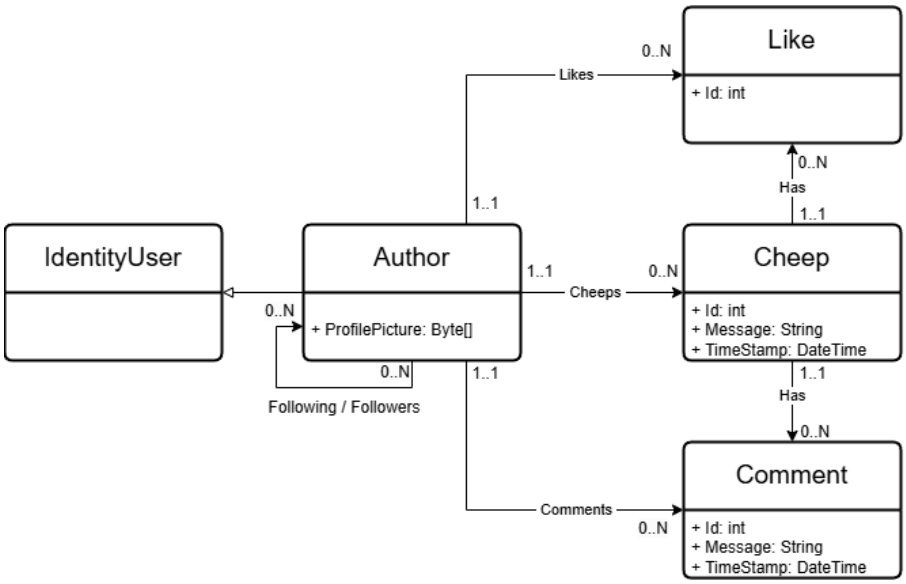

Figure 1: Illustration of the *Chirp!* data model as UML class diagram.

1.2 Architecture — In the small

Due to the application's size, each layer consists only of a single project, as highlighted in bold. **Chirp.Web** references **Chirp.Infrastructure**, which deviates from the Onion architecture for two reasons:

1. **Program.cs** requires it to configure services.
2. ASP.NET Identity uses it for user registration and verification.

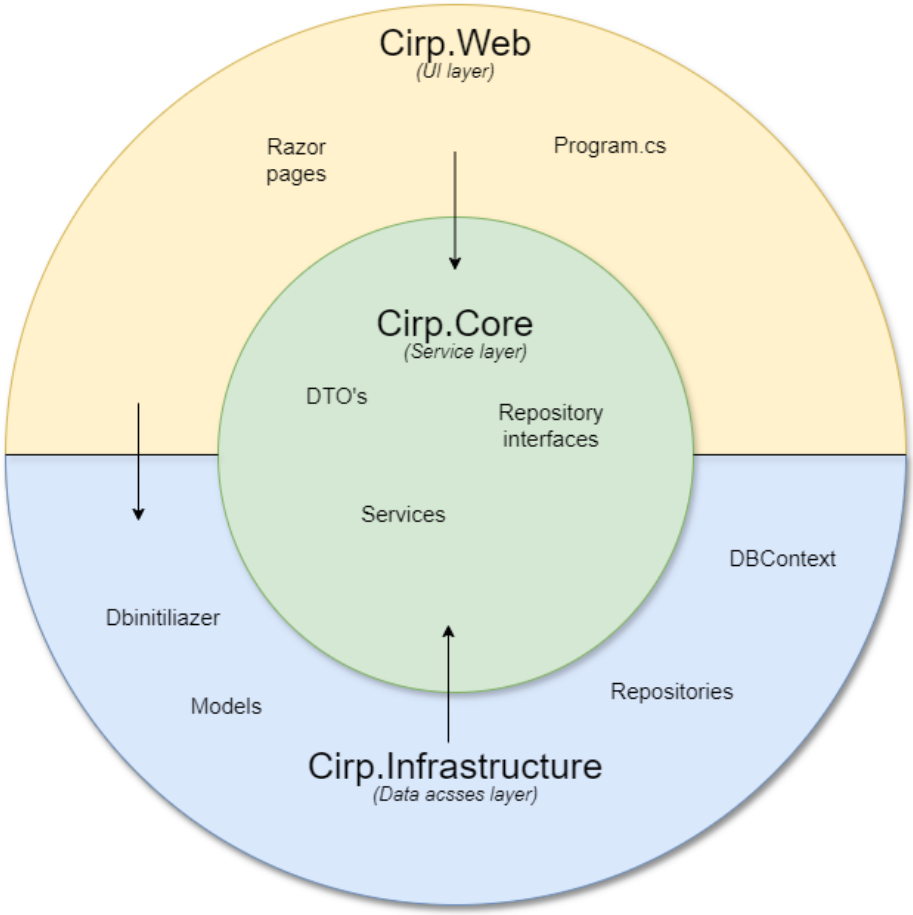

Figure 2: Illustration of the *Chirp!* program architecture.

1.3 Architecture of deployed application

The *Chirp!* application is deployed to Microsoft Azure App Service as a complete component, consisting of Chirp.Web for the GUI, Chirp.core for exposing the business services, and Chirp.Infrastructure for managing the domain model and repositories. Users connect to Chirp.Web through Azure. For read and write requests, the Azure Web App communicates with the deployed SQLite server. If users attempt to login or register using OAuth via GitHub, Chirp.Web sends authentication requests to GitHub.

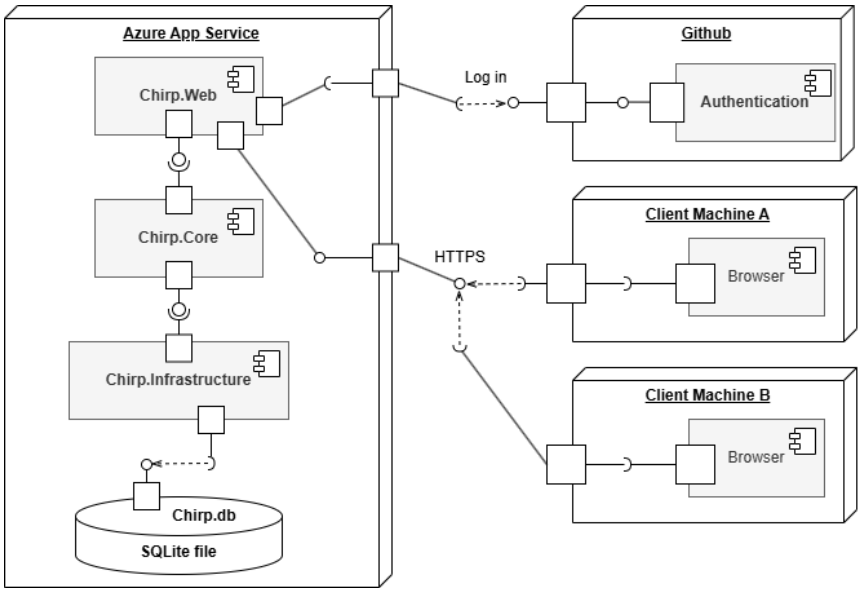

Figure 3: Illustration of the *Chirp!* deployment architecture of the application.

1.4 User activities

To enhance the readability of the UserActivities diagram, the complete diagram has been divided to display activities based on whether the user is signed in or not. Common for the whole application is.

- On every page you can go to home(the public timline)
- On every page where you can read cheeps. 2 videos are playing on both sides.
- On every page where you can read cheeps, you can switch between pages, with the exception of about me

The full diagram can be seen under *docs/images/UserActivitiesDiagram.pngT*

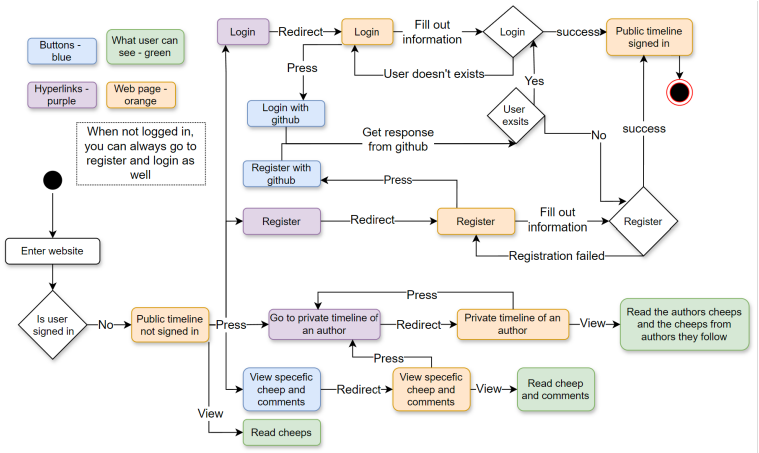

Figure 4: Illustration of the *Chirp!* functionality while signed out.

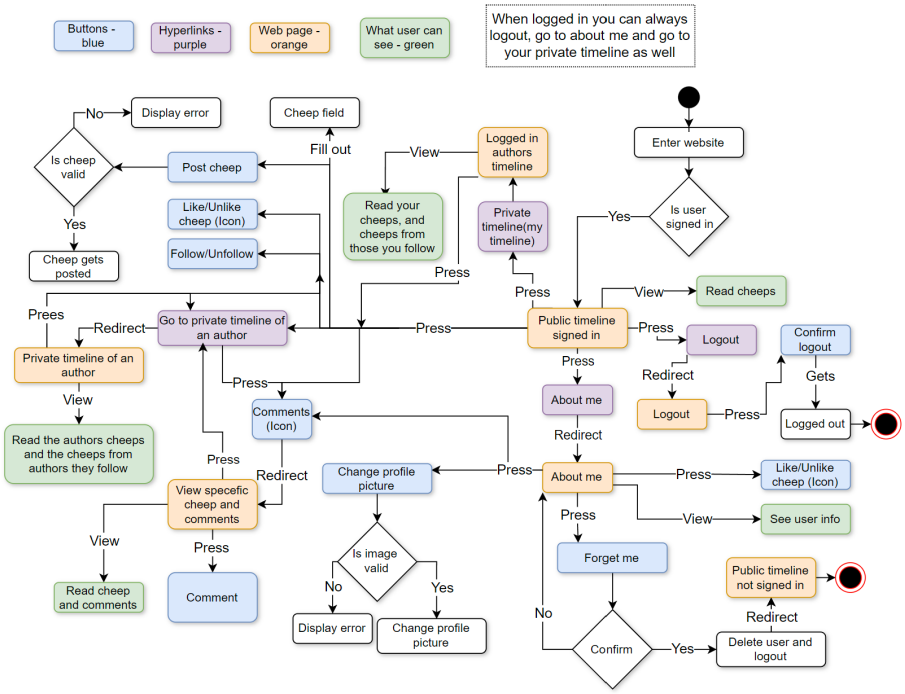

Figure 5: Illustration of the *Chirp!* functionality while signed in.

1.5 Sequence of functionality/calls trough *Chirp!*

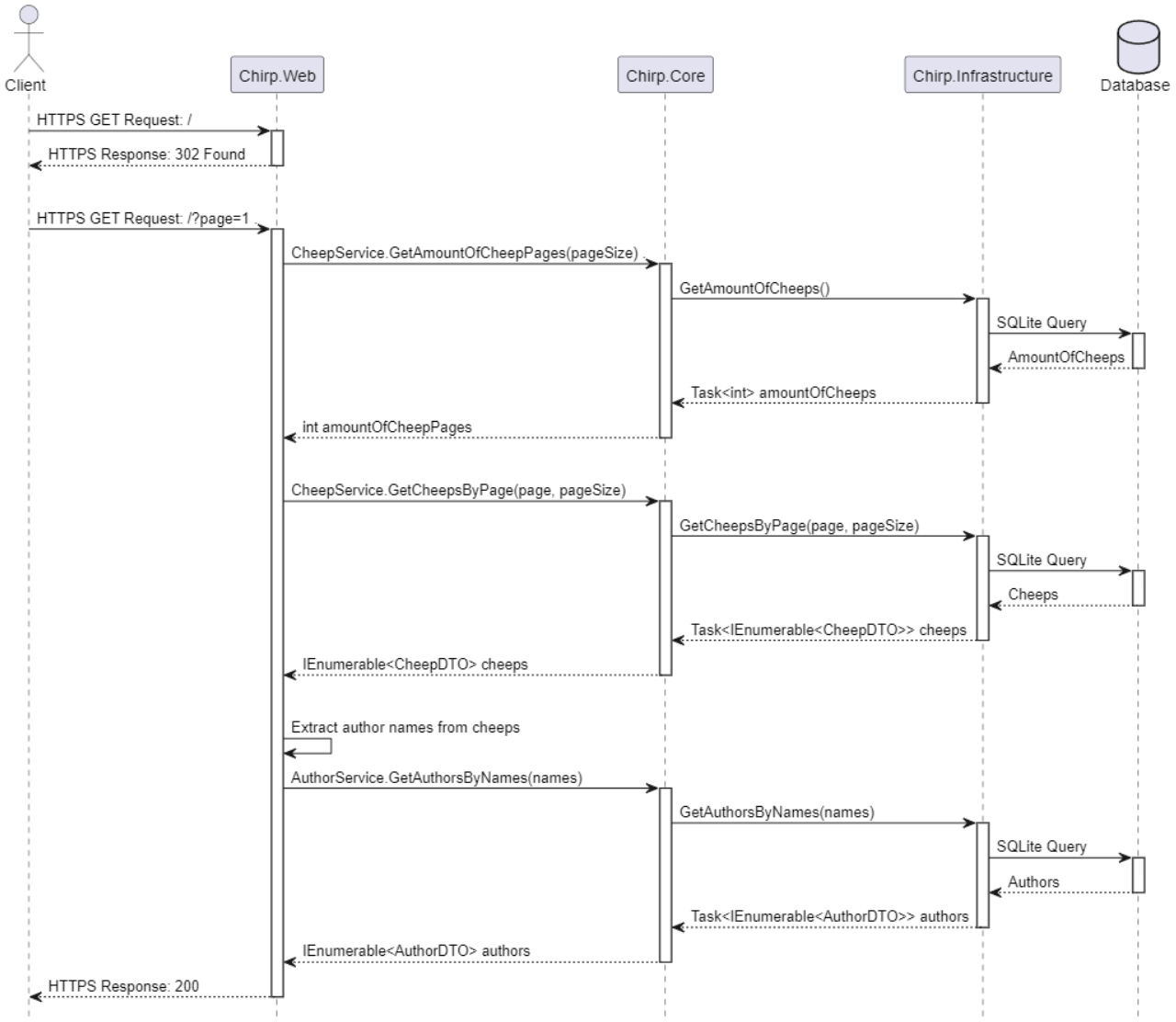

Figure 6: Sequence diagram of a request to the root endpoint *Chirp!* application.

2 Process

2.1 Build, test, release, and deployment

Figure 5 below illustrates the workflows used for building and deploying the *Chirp!* application. The process starts when a pull request is merged into the main branch. The blue boxes represents workflows.

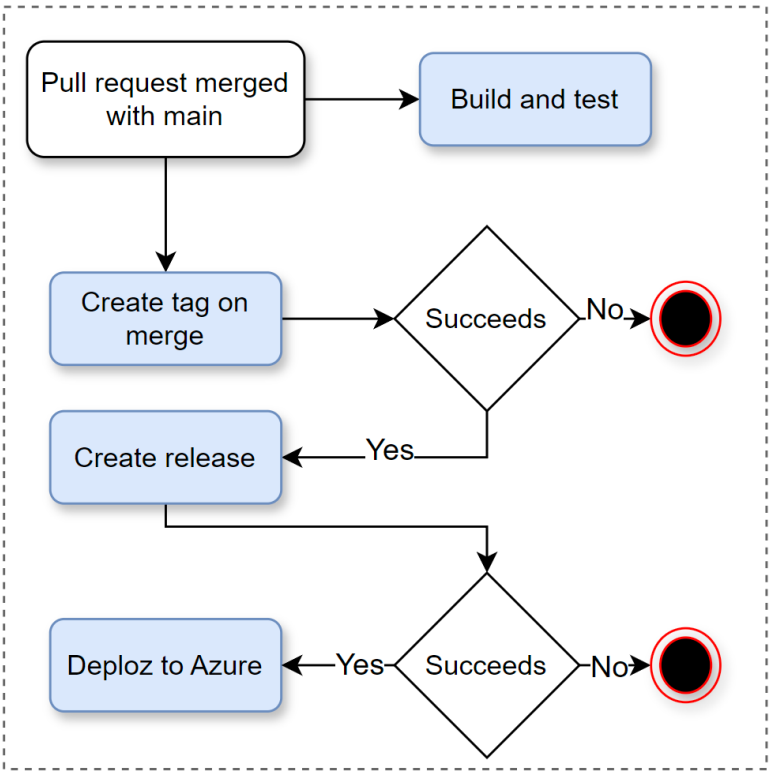

Figure 7: Illustration of github workflows for building and deploying the *Chirp!* application.

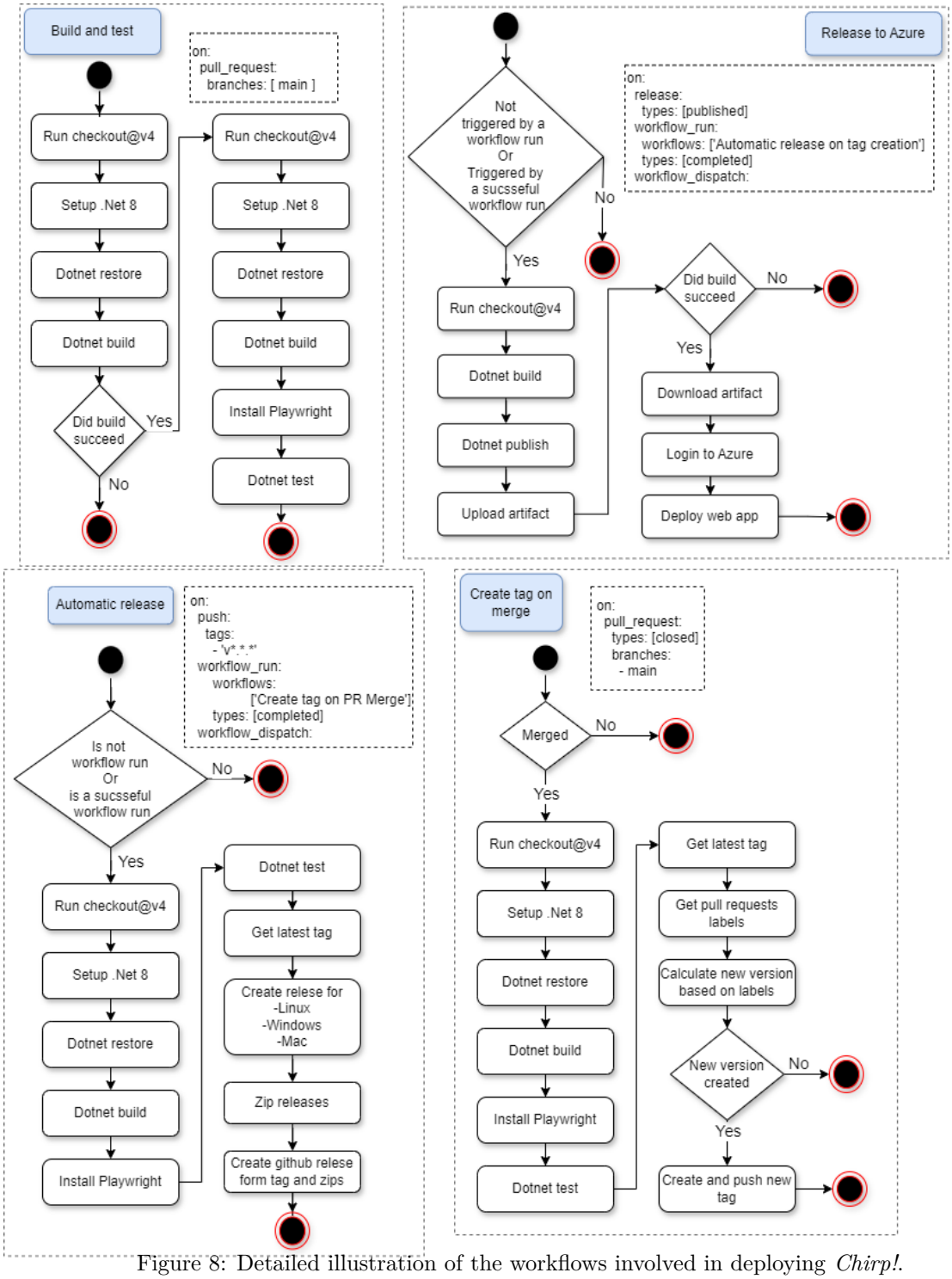

Figure 8: Detailed illustration of the workflows involved in deploying *Chirp!*.

2.2 Pull Requests

To validate pull requests and ensure that only code meeting the following criteria:

- Contains no warnings
- Successfully builds
- Passes all tests

is merged into the main branch. The following workflow structure was established.

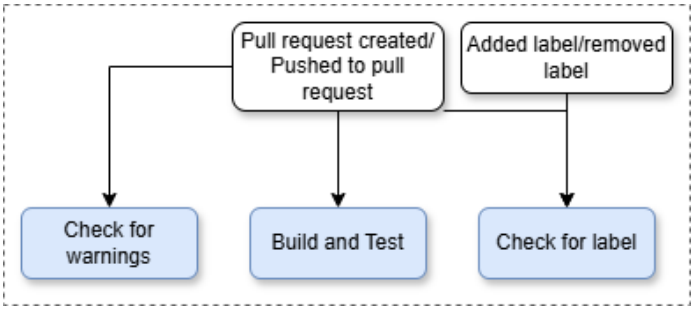

Figure 9: Illustration of github workflows for pull requests into the main branch of *Chirp!*

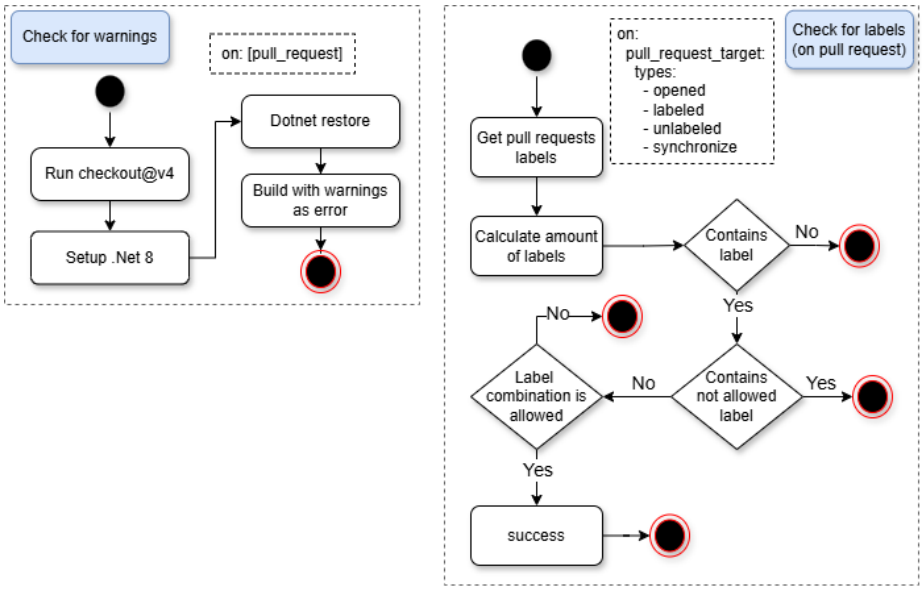

Figure 10: Workflow structure for pull requests into the main branch of *Chirp!*

2.3 Team work

2.3.1 Project Board

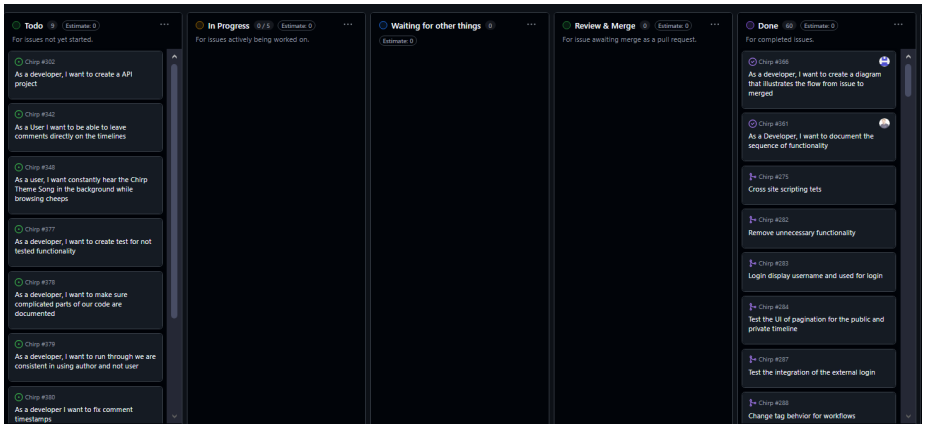

Figure 11: Project board

2.3.2 Unclosed Issues

Some issues remain open in the Todo column. These include a combination of features that the group found interesting but were unable to implement within the project timeframe, as well as bugs, refactoring tasks, and documentation updates.
To better mimic the functionality of *X* (formerly known as Twitter), users should be able to leave comments directly on the timeline pages. This feature was intended to be implemented with a popup window that appears when a user clicks on a Cheep, allowing them to leave comments. However, handling and displaying message format errors proved challenging, so the feature was given an "Extra" tag and left open.

The group also planned a significant refactor, which involved moving as much database access as possible to an API project. Since ASP.NET Identity is used for user registration and verification, a local database would still be required for the web project to store user information. The main purpose of the API project was to decouple data access from the web application, making it easier to build additional features, such as a mobile app, by enabling shared data access across projects. While a centralized database could achieve similar results, an API is more future-proof as it abstracts the database layer, allowing database switching without impacting the projects using the API.

Some additional features were also left incomplete, such as a theme song, improved data display for users, and fixing a bug related to timestamps on comments.

The remaining issues involve refactoring for better code quality, improving in-code documentation, and ensuring consistent naming conventions throughout the codebase.

For the Command-Line-Interface (CLI) version of Chirp!, an error with the end-to-end tests persists. These tests pass when the database file contains the expected Cheep and the tests are run on Windows. However, the group was unable to make the test work in isolation from the actual database and cross-platform. As a result, the end-to-end-test branch remains open.

2.3.3 Issue Progression

The illustration below shows how the group worked with issues during the project. Steps highlighted in blue show issue creation, red boxes show the development process and green how issues are merged from the feature branch into main.

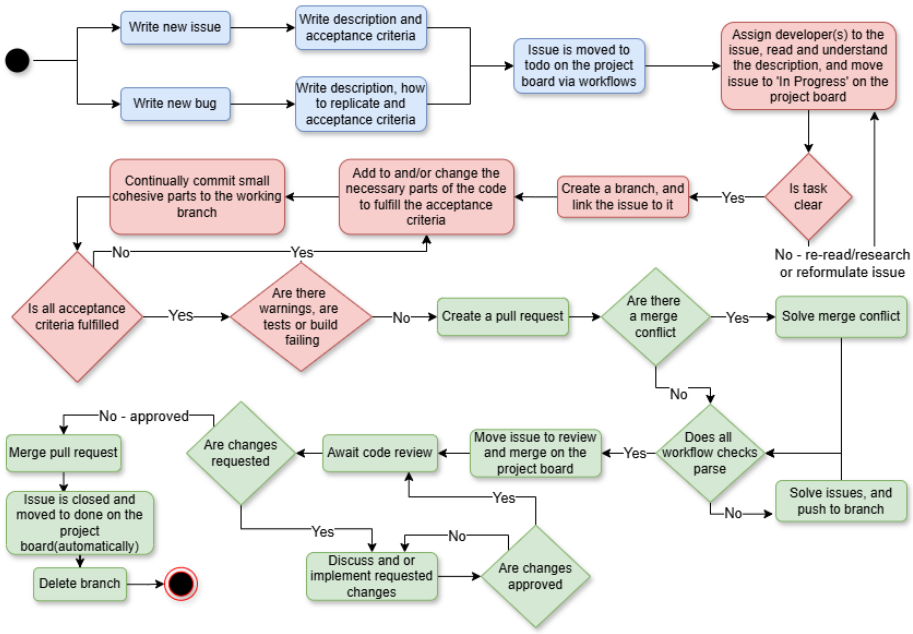

Figure 12: Illustration of the *Chirp!* issue progression from creation to merge.

2.4 How to make *Chirp!* work locally

First clone the repository to your machine with:

```
git clone https://github.com/ITU-BDSA2024-GROUP10/Chirp.git
```

In order for the program to work, you will need to configure the user-secrets. To do this navigate to */Chirp/Chirp*, in the terminal, and run:

Windows:
```
dotnet user-secrets init --project .\src\Chirp.Web\
dotnet user-secrets set "authentication:github:clientId" "Ov23lisGJEMdXORhzpDr"
--project .\src\Chirp.Web\

dotnet user-secrets set
"authentication:github:clientSecret" "a9229ceee8bb014070dc9abe892cf07d7aba4d0d"
--project .\src\Chirp.Web\
```

MacOs & Linux:
```
dotnet user-secrets init --project ./src/Chirp.Web/
dotnet user-secrets set "authentication:github:clientId" "Ov23lisGJEMdXORhzpDr"
--project ./src/Chirp.Web/

dotnet user-secrets set
"authentication:github:clientSecret" "a9229ceee8bb014070dc9abe892cf07d7aba4d0d"
--project ./src/Chirp.Web/
```

Next navigate to */Chirp/Chirp/src/Chirp.Web* and in your terminal do either:

```
dotnet watch
dotnet run
```
The *OpenIdConnect* button, for registering and log in, is not meant to be used with the application. It is the Identity Server and are only used for testing purposes.

2.5 How to run test suite locally

In order to run the UI-Tests, make sure that your system has Playwright installed. Next, navigate to */Chirp/Chirp* and in your terminal do

```
dotnet test
```

2.6 Our test structure

We have three kinds of test

- Unit
- UI
- End to end

Since our services are essentially return statements calling our repository, we found integration tests for these to be of lesser value compared to other aspects of the project. However, if the project continues, testing these services would be beneficial to ensure their functionality remains intact as they evolve.

All of the unit tests focus on the repository methods, as this is where the majority of the computation takes place.

Our UI tests are designed to be quite general. We don't mock any dependencies; instead, we use an in-memory database and validate only whether the UI behaves as expected. Additional tests could have been implemented to further isolate the UI, such as mocking the behavior of the service methods used. However, given the size of the application, we prioritized other tasks.
We also have some end-to-end tests, though more could be added. Since our UI tests are so general, they somewhat serve the role of end-to-end tests as well.

3 Ethics

3.1 License

This program is licensed with the GNU GENERAL PUBLIC LICENSE Version 3. For the dependincies used, they either have an MIT or an Apache-2.0 license. And since we only use Duende Identity server for testing purposes, a license is not required, as stated at the bottom of, https://duendesoftware.com/products/communityedition

3.2 LLMs, ChatGPT, CoPilot, and others

3.2.1 CoPilot

Github CoPilot has been used during the development of this project. It has been a great tool for speeding up development, as it is quicker to read through the code it recommends than to write it. Not everything it recommends is usable or as desired, but it can also help when learning a new language or framework to introduce new methods and structure.

3.2.2 ChatGPT

ChatGPT was used primarily for the three following things.

- Understanding and debugging error messages
- Writing some HTML and CSS code
- Understanding and discussing code

Understanding and debugging error messages. An LLM is ideal here since error messages can be very long and contain a lot of information. Sometimes, they can also be harder to understand if one doesn't have a lot of knowledge of the framework being used.
Writing some HTML and CSS code. Since HTML and CSS is time consuming, we used ChatGPT to help write some of the UI code that could then be fine tuned by hand.

Understand and discussing code. This can be very helpful when others are not available, especially when finding code online, for example from Stack Overflow or documentation. When learning a new language or framework, this can especially be helpful, since you don't know a lot of the tricks yet.

Overall, the use of LLM's sped up our development process, and helped us get a better understanding of C# and .NET. While using LLM's can help speed up debugging and development processes, it is still important to learn how to work independent of AI-assistance. The data centers running the models also consume large amounts of energy, and as a developer you need to be conscious of the impact of this technology.