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

1 Design and Architecture of *Chirp!*

1.1 Domain model

The *Chirp!* domain model is setup around the Author class. Authors inherit traits for account management from IdentityUser. Authors are able to create Cheeps and interact with them with Likes or Comments. Each Author keeps a list of Likes and Comments enabling logging of which Authors have interacted with which Cheeps.

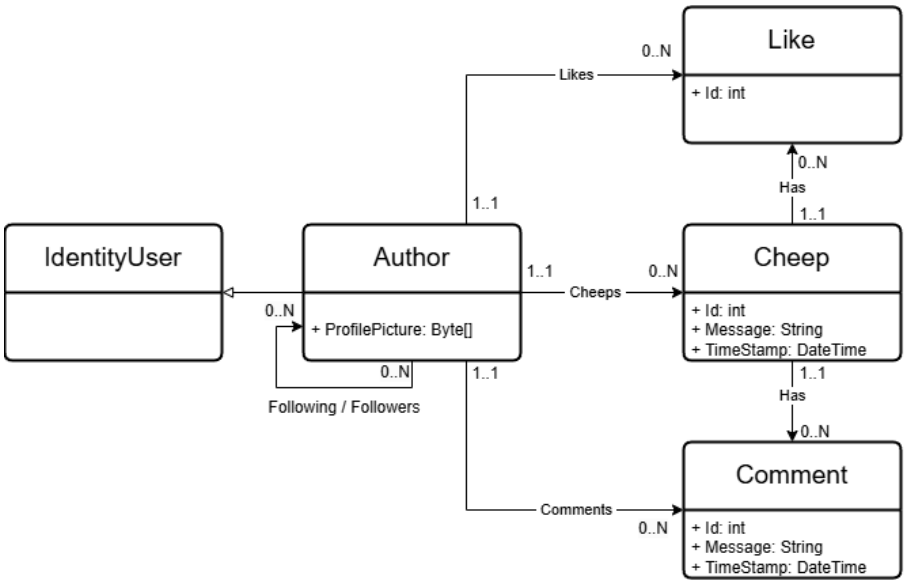

Figure 1: Illustration of the *Chirp!* data model as UML class diagram.

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. Microsoft Identity uses it for user registration and verification.

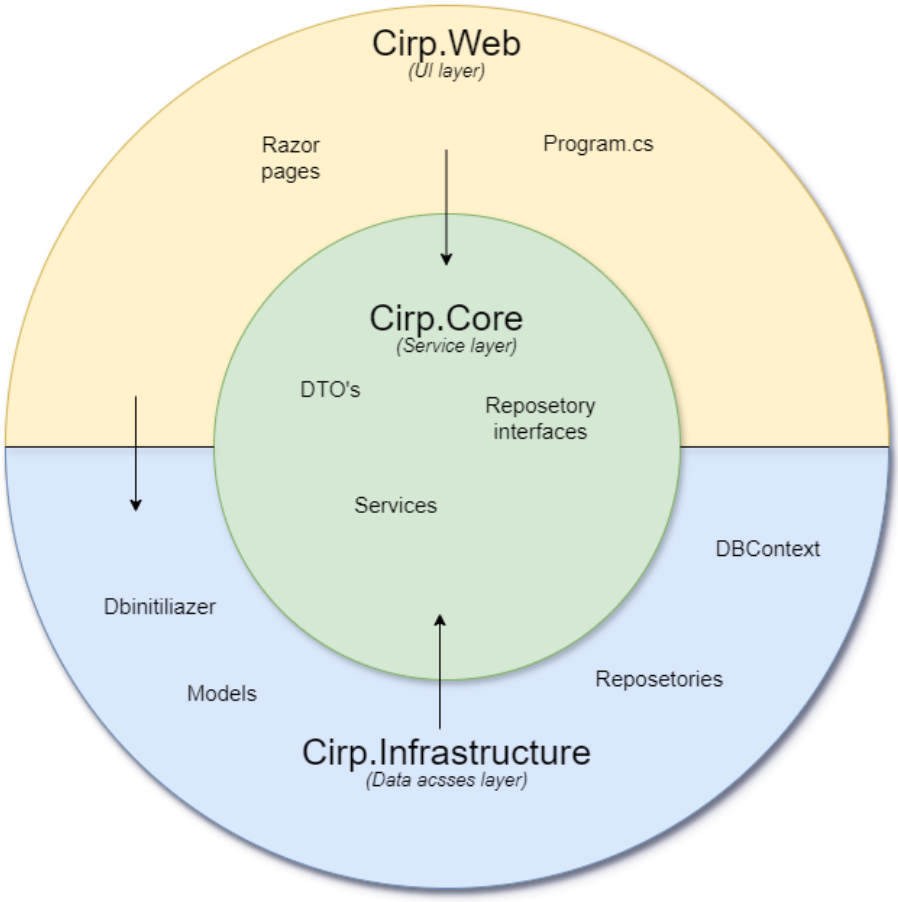

Figure 2: Illustration of the *Chirp!* program architecture.

1.3 Architecture of deployed application

The *Chirp!* application is deployed to the Microsoft Azure App Service as a complete component consisting of Chirp.Web for the GUI, Chirp.Infrastructure handling the domain model and repositories. The User connects to Chirp.Web through Azure. On read and write requests the Azure Web App will make calls to the deployed SQLite server. If users attempt to login or register with OAuth via github Chirp.Web will make calls to GitHub Authentication.

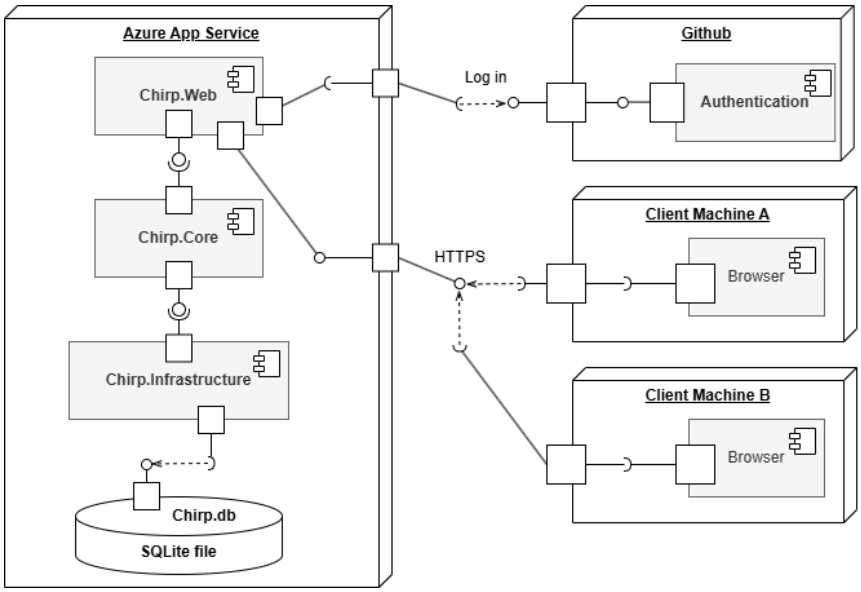

Figure 3: Illustration of the *Chirp!* deployment architecture of the application.

1.4 User activities

In order to increase the readability of the UserActivities diagram, the total diagram has been decomposed to show activities depending on whether the User is signed in or not.

The total diagram can be under *docs/images/UserActivitiesDiagram.png*

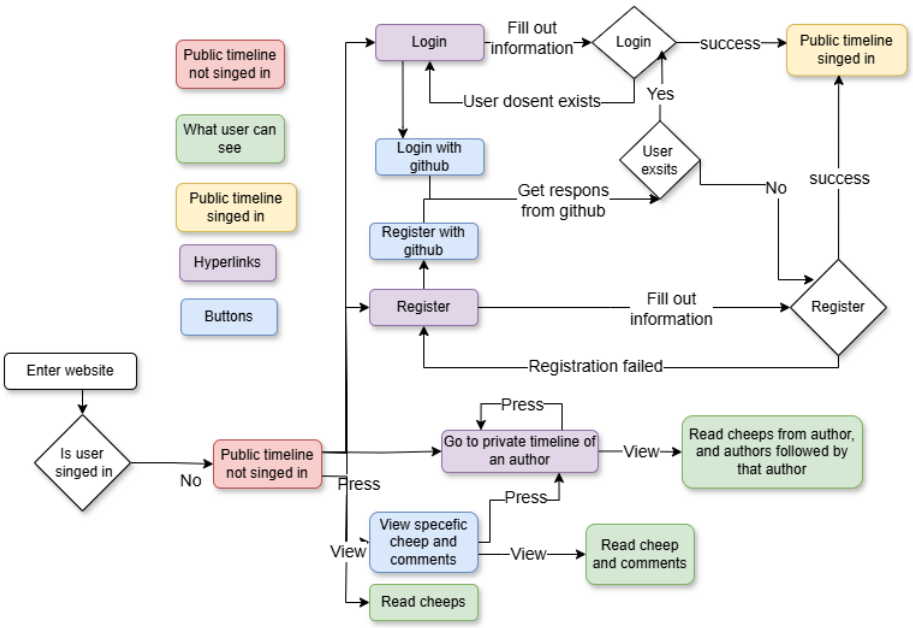

Figure 4: Illustration of the *Chirp!* functionality while signed out.

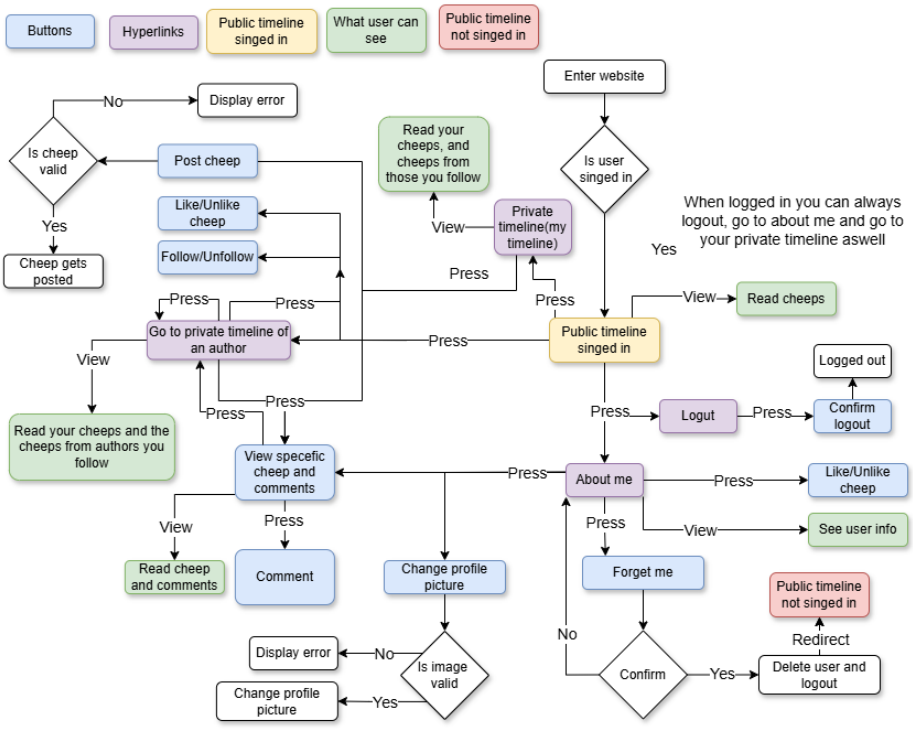

Figure 5: Illustration of the *Chirp!* functionality while signed in.

1.5 Sequence of functionality/calls trough *Chirp!*

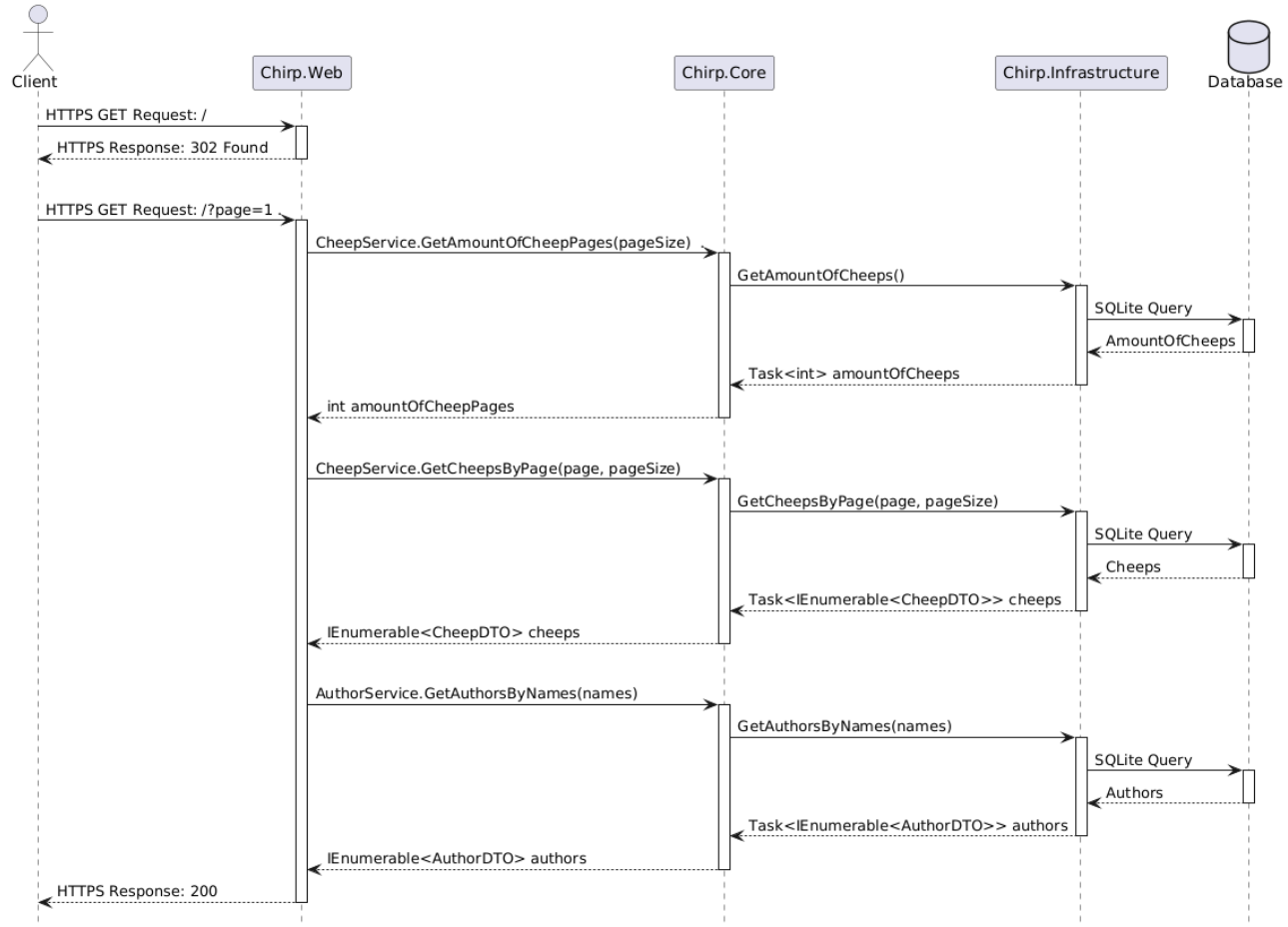

Figure 6: Sequence diagram of the flow of messages through the *Chirp!* application.

2 Process

2.1 Build, test, release, and deployment

The figure 5 below illustrates the workflows used for building and deploying the *Chirp!* application. The process stars, when pull request is merged into the main branch. The blue boxes are workflows

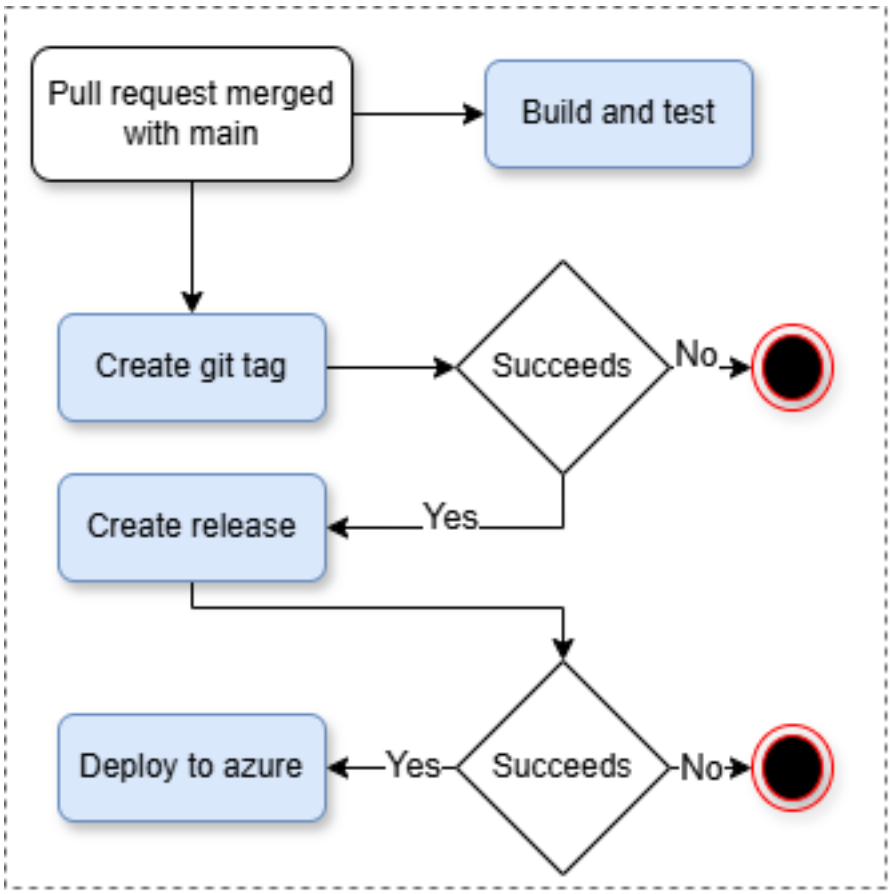

Figure 7: Illustration of github workflows for building and deploying the *Chirp!* application.

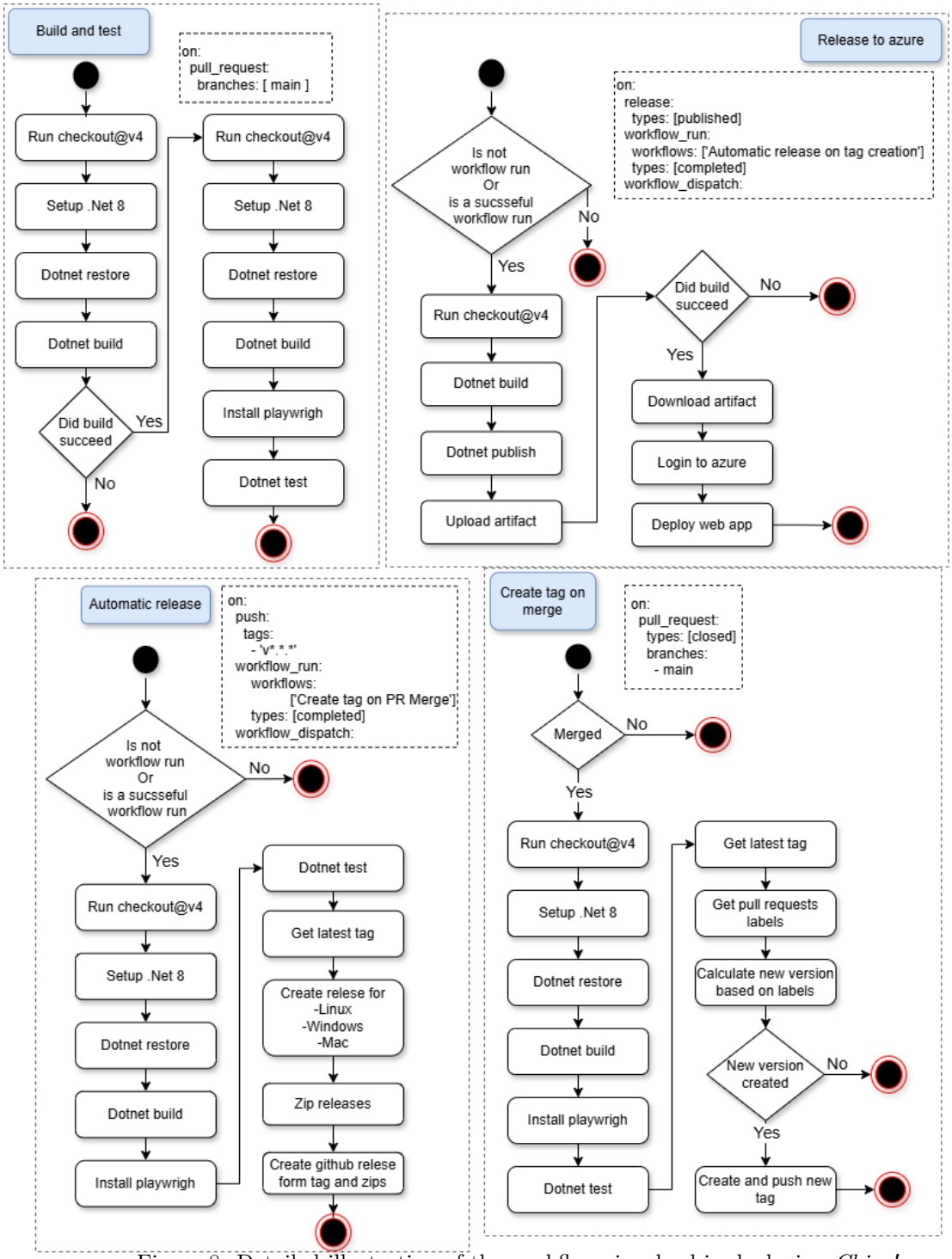

Figure 8: Detailed illustration of the workflows involved in deploying *Chirp!*.

2.2 Pull Requests

The group also had workflows setup to make sure pull requests. In order to make sure only code, where the build succeeded and no tests failed, was pulled into main the following workflow structure was setup.

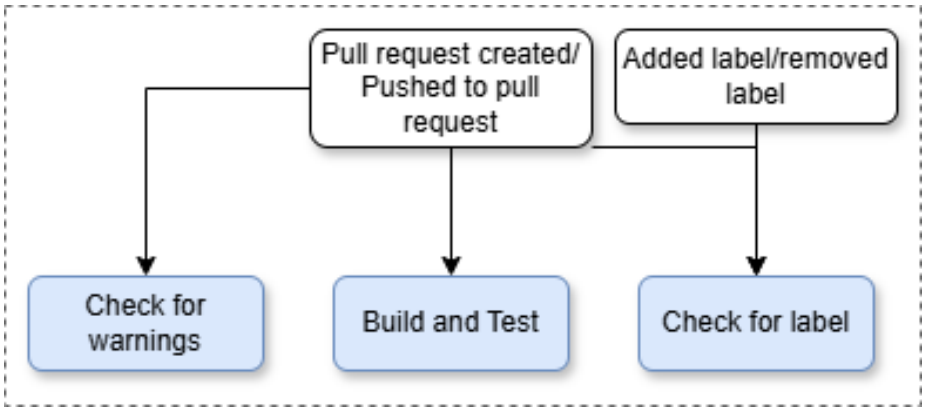

Figure 9: Illustration of github workflows for building and deploying the *Chirp!* application.

2.3 Team work

2.3.1 Project Board

2.3.2 Unclosed Issues

Some issues still remain open in the Todo column, these are extra features that the group found interesting but did not get to implement within the time frame of the project work.
In order to better mimic the functionality of *X* (f.k.a *Twitter*), users should be able to leave comments directly on the timeline pages. This would be implemented by having a popup window appear, where users could leave comments, when clicking a Cheep. However getting this to work while handling and displaying message-format-errors proved to be an issue, and the feature was given an *Extra* tag and left open.
We also wanted to make a big refactor, which involved moving what database access we could to an API project. Since we use Microsoft Identity for user registration and verification, a local database would still be required for the web project to store user information. The main reason for the API project is to decouple data access from the web application, making it easier to build additional features, such as a mobile app, by enabling shared data across projects. While a centralized database could achieve similar results, an API is more future-proof, as it abstracts the database layer, making the switching of the database, have no impact on the projects using the API.

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 purple how issues are merged from the feature branch into main.

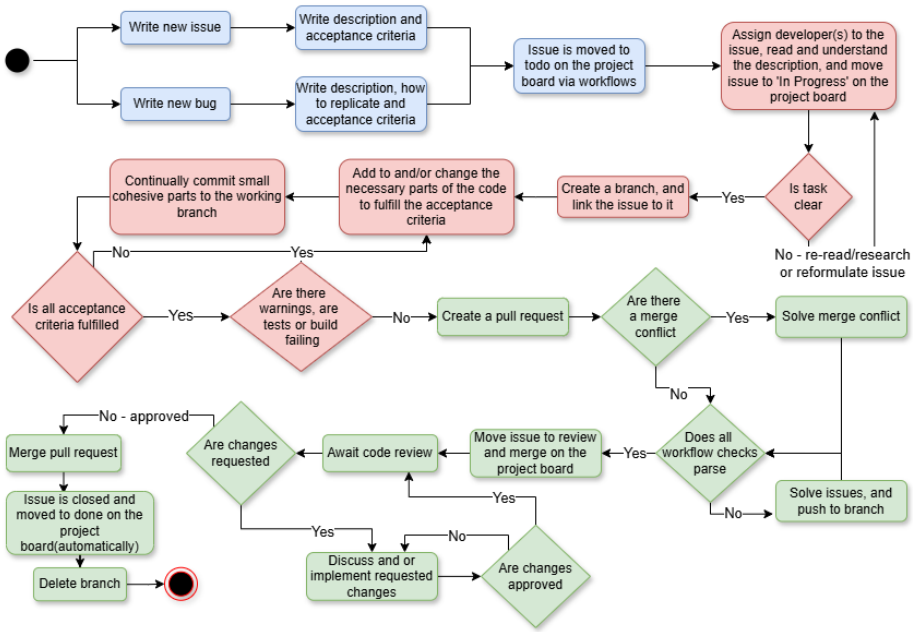

Figure 10: Illustration of the *Chirp!* issue progression from creation to merge.

2.4 How to make *Chirp!* work locally

Navigate to */Chirp/Chirp/src/Chirp.Web* and in your terminal do *dotnet run* or *dotnet watch*

2.5 How to run test suite locally

Navigate to */Chirp/Chirp* and in your terminal do *dotnet test*.

3 Ethics

3.1 License

This program is licensed with the GPL-2.0 License

3.2 LLMs, ChatGPT, CoPilot, and others

ChatGPT was used to understand and debug error messages, and write some html for the razor pages.