Chirp! Project Report
ITU BDSA 2023 Group 9

Anton Dørge Friis anlf@itu.dk   Johan Sandager jsbe@itu.dk
Oline Scharling Krebs okre@itu.dk
Jonas Kramer kram@itu.dk
Clara Walther Jansen clwj@itu.dk
Lauritz Andersen lana@itu.dk

1 Design and Architecture of *Chirp!*

1.1 Domain model

In our domain model, we capture entities and relationships to provide a high-level abstraction of the system's static structure, thereby centering the focus on the business logic of our Chirp!-system. We provide a complete diagram below:

At the highest level in this diagram, we have Chirp.Core, Chirp.Infrastructre, Chirp.Web, wherein certain classes are contained. Apart from this, we have references to external libraries such as AspNetCoreIdentity and FluentValidation.

1.1.1 Brief Description of Classes inside Chirp.Core

Chirp.Core contains all the core functionality. In Chirp.Core, we have Data Transfer Objects (DTOs) for Cheeps and Authors, as well as interfaces for Author and Cheep Repositories. Nothing inside Chirp.Core points to anything outside of Chirp.Core. There is an associative relationship between the DTOs and the

repositories, since the repositories use the DTOs to create cheeps and authors, get cheeps and name of authors, follow authors etc. The interface repositories provide an interface contract for the repository implementation that will transfer structured data between different layers of the application and use DTOs as a standardized communication contract in this regard.

1.1.2 Brief Description of Classes inside Chirp.Infrastructure

Key elements within Chirp.Infrastructure include CheepRepository and Author-Repository, which are concrete implementations of the Repository interfaces. These repositories use ChirpContext to interact with the database, managing data for authors and cheeps. Each repository handles specific queries related to either authors or cheeps, working with Author and Cheep models. However, they return only Data Transfer Objects (DTOs) to maintain separation of concerns. The Author and Cheep models, stored in the database, are queried from the dbsets of the dbcontext. The composition relationship between Author and Cheep classes and ChirpContext ensures data integrity, as removing the context also removes the associated data models from the program.

The Author class inherits from identity user to streamline authentication and authorization for a Chirp author. There is a composite relationship between Author and Cheep, indicating that an Author can own 0 or more cheeps and that any existing cheep is owned by a unique Author.

1.1.3 Brief Description of Classes inside Chirp.Web

Inside of Chirp.Web, we have our Program.cs, which is the class the program is run from. It has the fields WebApplicationBuilder builder and WebApplication app, which are used for encapsulating the app's services and middleware and for building the web application, setting up authentication and services necessary to make the application run.

1.2 Architecture — In the small

The diagram above depicts the Onion-architecture of our code-base. The different layers and their position shows which code they have access to. 'Chirp.Core' only know about itself, whereas 'Chirp.Infrastucture' has access to the code in 'Chirp.Core' but not the outer layers - except for the database as depicted with the arrow in the diagram.

1.3 Architecture of deployed application

We will briefly discuss the architecture of the deployed application, which is based on a client-server-model.

In the illustration below, the Web Server, hosted on Azure, manages user requests and serves the website, while the SQL Server, also on Azure, stores structured
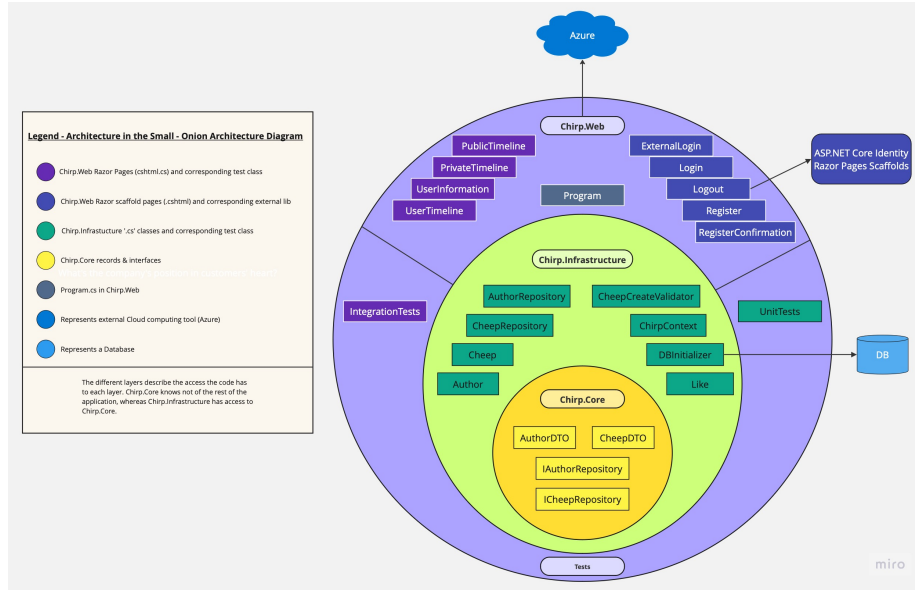
Figure 1: Illustration of the *Chirp!* data model as UML class diagram.

data like user information and chirps. The Client browser communicates with the Web Server, which contains the webpage artifact. The Web server, in turn, interacts with the SQL Server, which contains the SQL_Database artifact, for database operations. In this manner, the Client doesn't directly connect to the SQL Server; but instead communicates with the Web Server, which handles the interaction with the database.

1.4   User activities

We will outline a few different user journeys to showcase the capabilities of Chirp! users. This includes showcasing actions for an unauthorized user, guiding through registration and login processes, and demonstrating a typical user journey within the Chirp! app when logged in.

1.4.1   Un-authorised user-journey

For an unauthorized user, typical actions might involve viewing cheeps on the public timeline or accessing specific details about an author, such as past cheeps, total cheeps, and other information, which would be accessible through the author's private timeline. This process is depicted in a simple User Activity diagram, where the unauthorized user navigates the webpage to view public cheeps and then explores a specific author's timeline on Chirp! for more details.
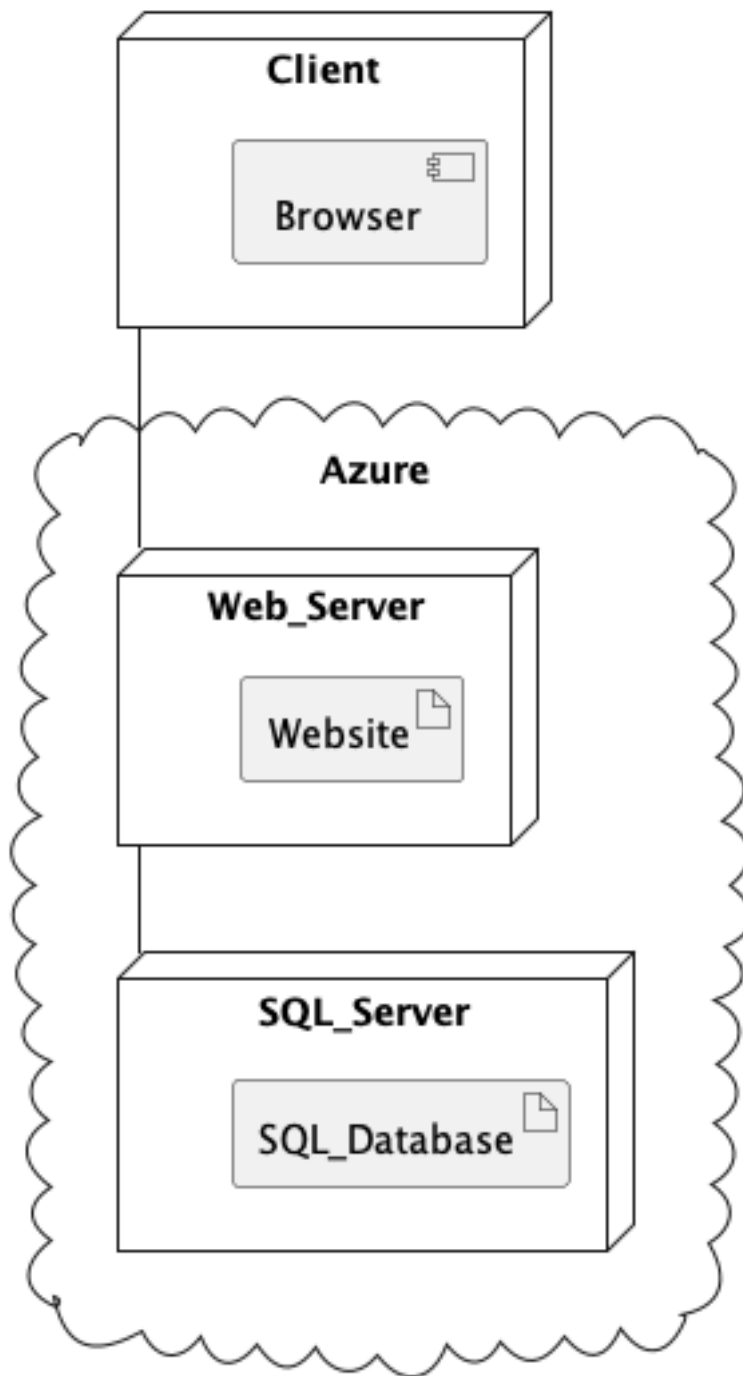
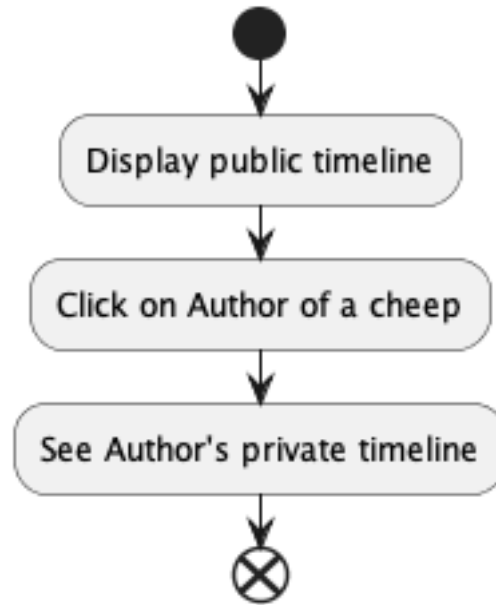Figure 2: Illustration of the *Chirp!* deployed application

Figure 3: Illustration of Unauthorised user journey

1.4.1.1   Registration and Login processes: For the full Chirp! experience, authorized users can personalize their interaction by posting, liking cheeps, and following authors. Let's illustrate the registration and log-in processes to demonstrate how users can achieve this.

1.4.1.2   Registration   To register, users click the "Register" button and choose either GitHub, where they will be authorized if they have an account in Github, or they can enter their password, username, and email, and then click "Register." After registration, users are redirected to the public timeline, where they can start following other users, liking their posts, and also posting cheeps.

1.4.1.3   Log-in   To log in, users click the login button, choose either GitHub or enter their password and username. Upon successful login, users are redirected to their private timeline if following someone or to the public timeline if not following anyone yet.

1.4.1.4   Cheeping and Following Authors   When using Chirp, users primarily write cheeps or follow authors and like their cheeps. The upcoming descriptions will illustrate user journeys centered around these actions. In both of the chapters below, we assume the user have already been through the log-in or registration process. to engage in these functionalities.
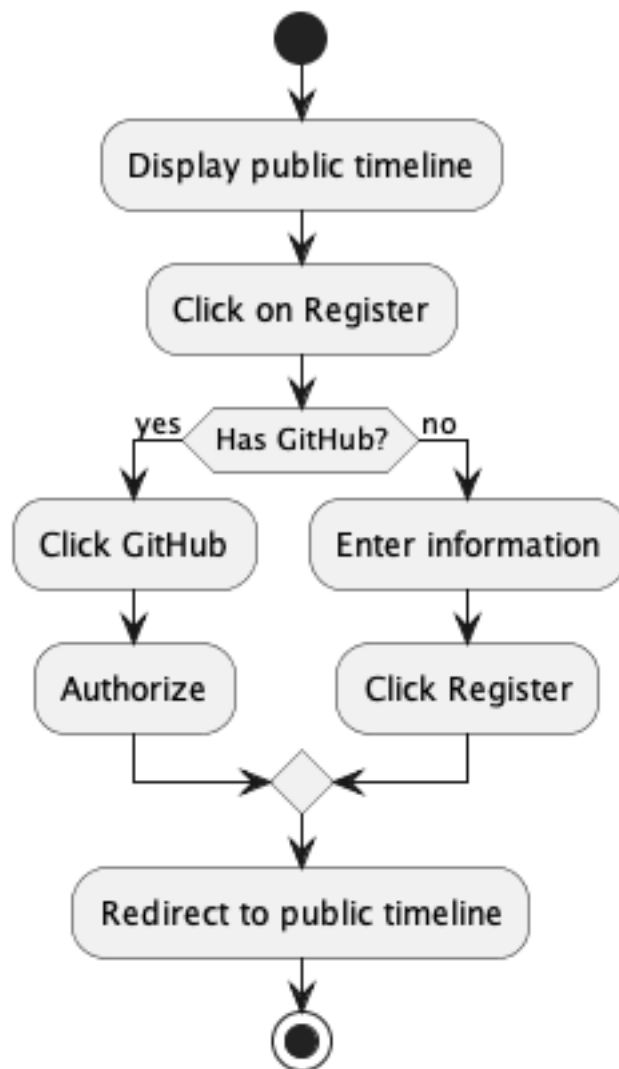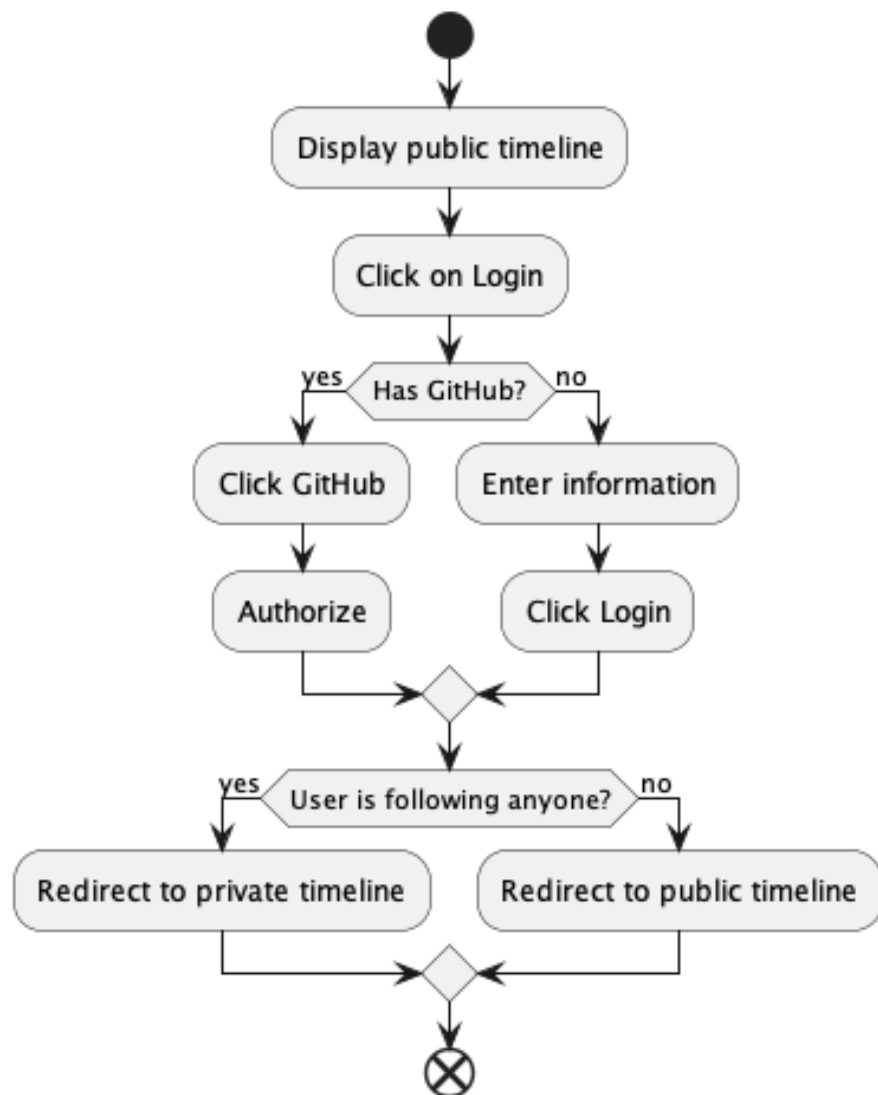
Figure 4: Illustration of Registering to Chirp

Figure 5: Illustration of Log-in process

1.4.1.5   Following   The user starts by visiting the Public timeline. They click on author x to view that author's timeline. If not already following, they hit "follow," go to their private timeline, and check what author x has posted. If already following, the user unfollows the author, then checks their private timeline to ensure author x's cheeps are no longer visible.

1.4.1.6   Cheeping   We assume, we have an user, who is already following other users. When this is the case, a user sees their private timeline, when logged in, where they see their own posts and posts from the people they follow. They then share short cheeps (up to 160 characters) and then look at their private timeline again to see what they've posted.

1.5   Sequence of functionality/calls through *Chirp!*

We see the sequence of calls that is made both internally by the program and externally, from a unauthorized user, sending a simple GET request to the root endpoint (acessing bdsagroup9chirprazor.azurewebsites.net).

2   Process

2.1   Build, test, release, and deployment

Here we have the build and test workflow, that ensures that the code passes all tests before merging a pull request to main.

Here we see the workflow that creates a release of the program to Github. It is triggered when a tag of the format v* is pushed to github.

This Github workflow is triggered after a push to main, and releases main to our production environment.

2.2   Team work

We set up a KanBan board to handle our issues and give us an overview of the process an issue had to go through.

- The issue is created and put into the 'Backlog'.
- When the issue is ready with a description, acceptance criteria and has no dependencies pending it is moved to then coloumn 'Ready'.
- When we start working on the issues it is moved to the 'In progress' coloumn.
- If the issue relates to the code-base and we have determined it is done, it is moved to 'Create tests'.
- When tests have been made (if necessary) a PR is created and the issues is moved to the coloumn 'In review'.
- If the PR for the issues gets approved, it is merged into the main-branch. If this is the case the issue can be closed and moved to the 'Done' coloumn.
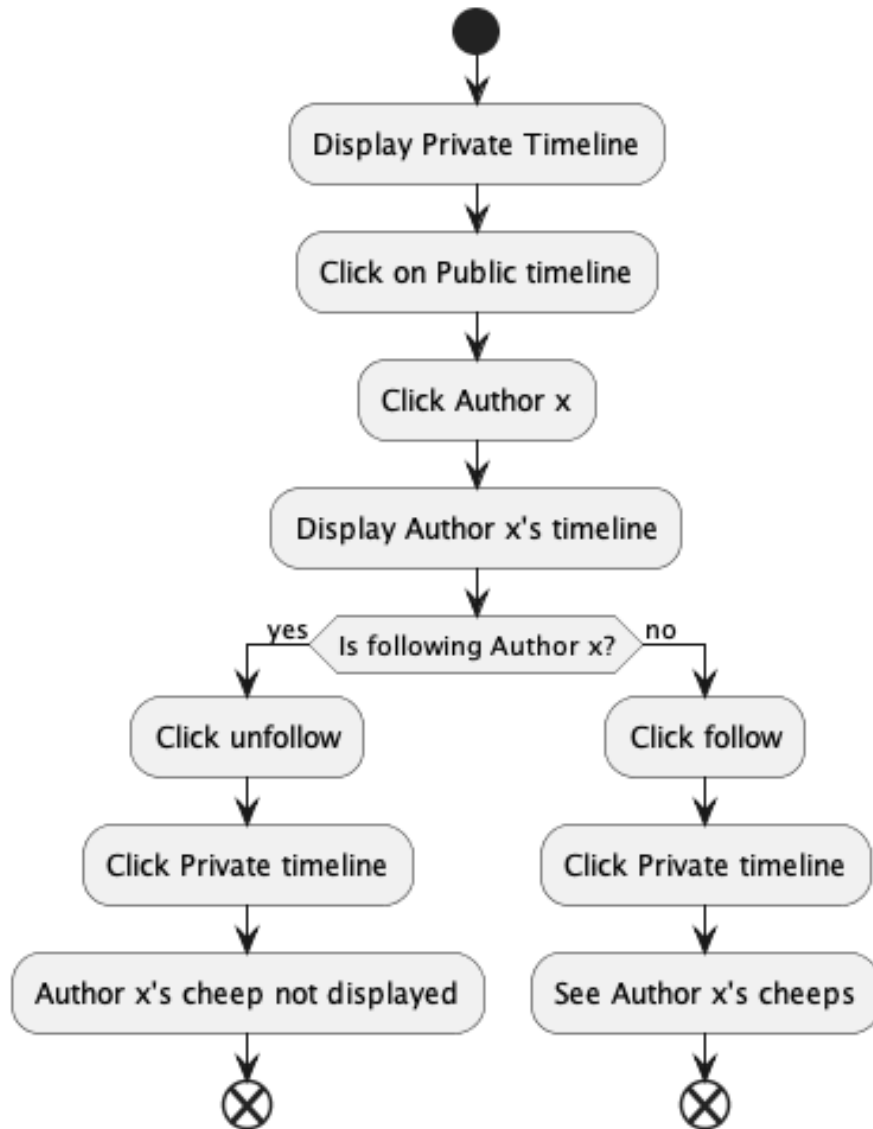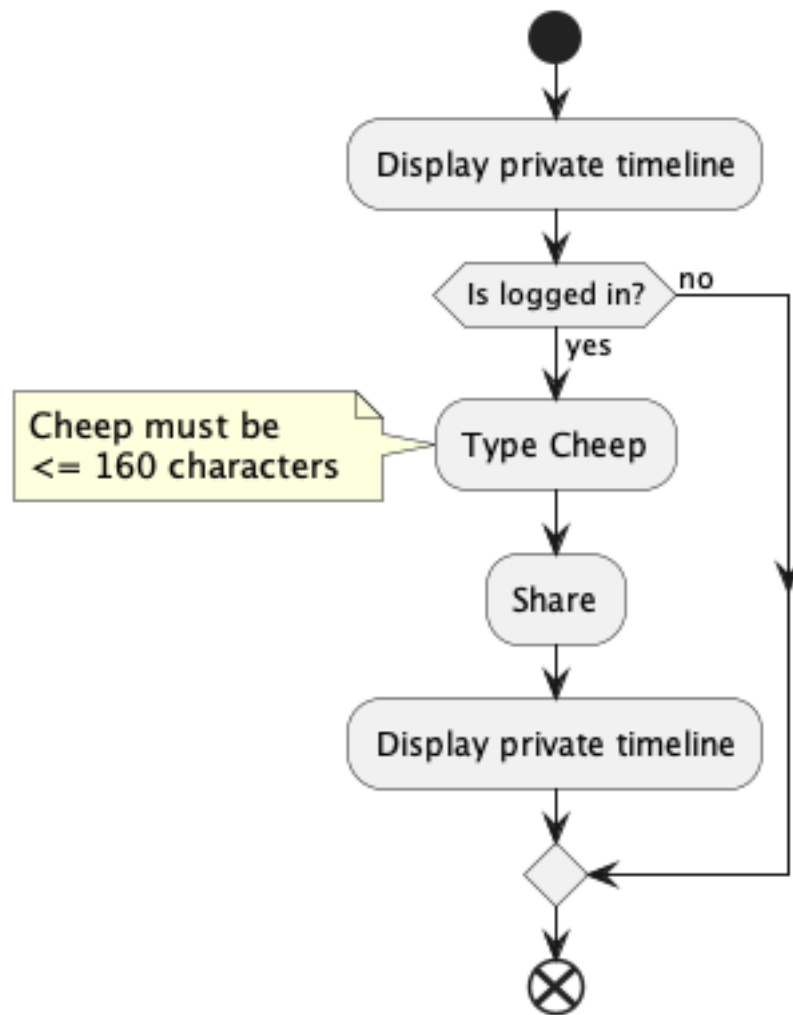
Figure 6: Illustration of Following other users
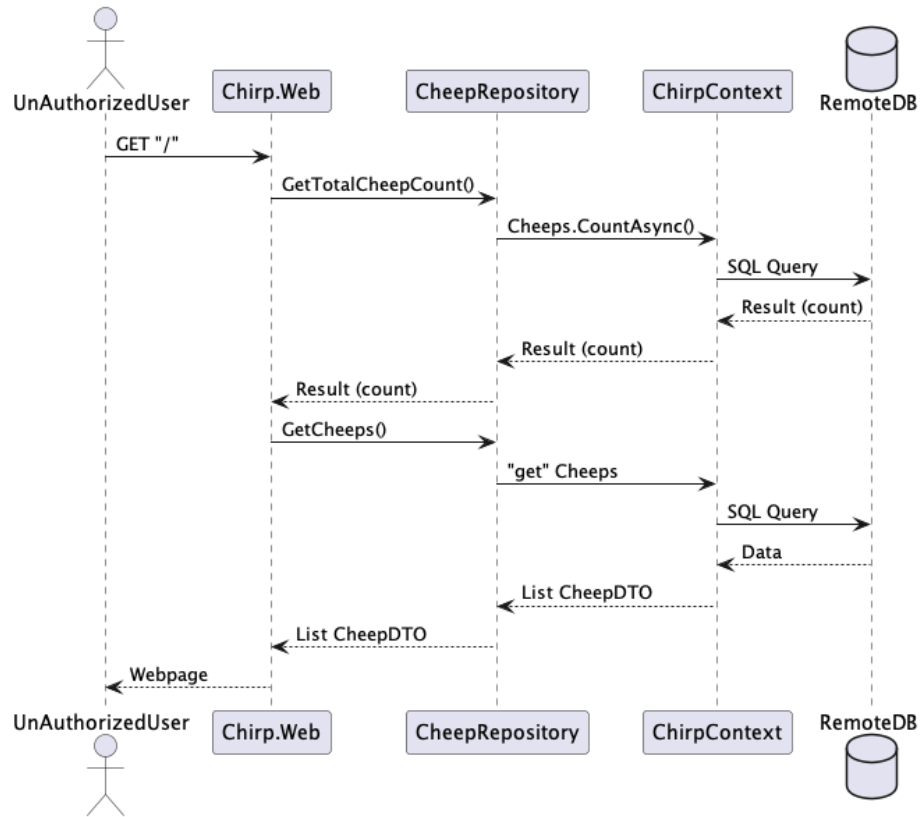
Figure 7: Illustration of Cheeping

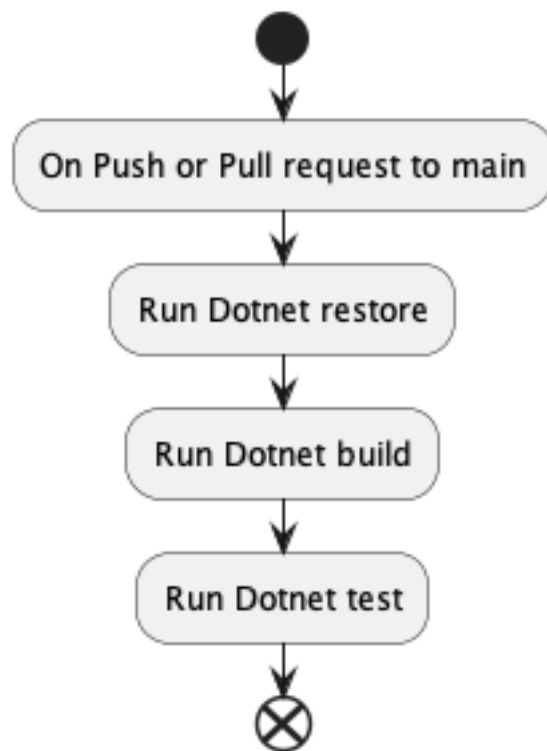Figure 8: Sequence of calls thorugh Chirp for an unauthorized user to root
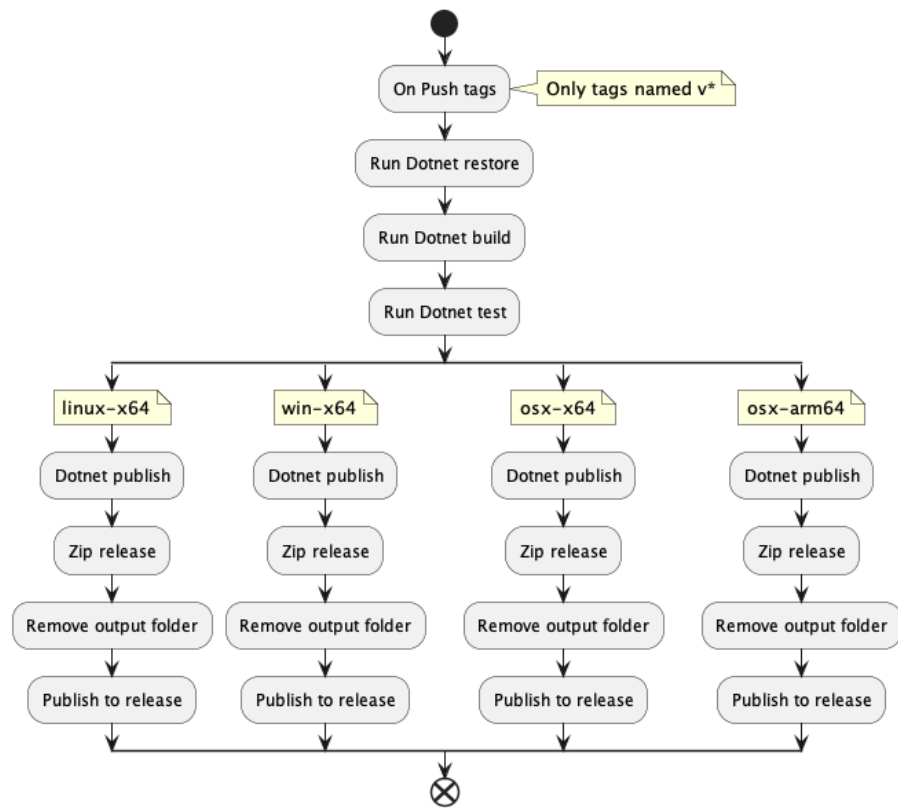
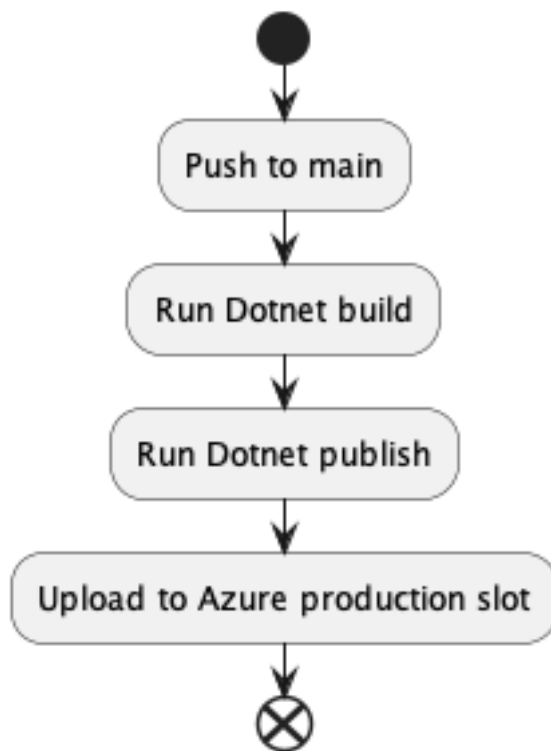Figure 9: Build and test github workflow

Figure 10: Release github workflow

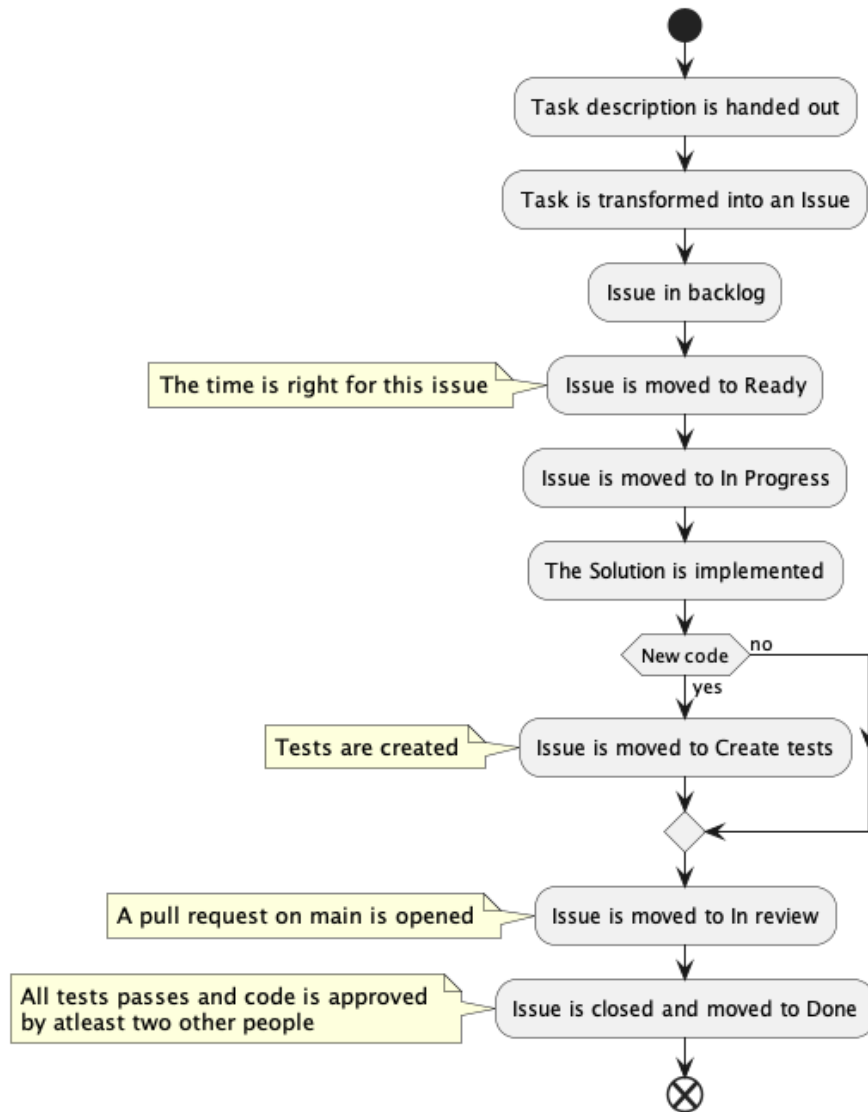Figure 11: Deployment gihub workflow

Figure 12: Workflow from Task to Finish

2.3 How to make *Chirp!* work locally

Firstly, open a command prompt. From here, navigate to the folder in which you want the project to be, and run the command: `git clone https://github.com/ITU-BDSA23-GROUP9/Chirp.git`. Afterwards, navigate to the Chirp folder by running the command `cd Chirp`. Afterward, navigate to src/Chirp.Web by running the command `cd src/Chirp.Web/`, and then start the application by running the command `dotnet run`.

2.4 How to run test suite locally

Firstly, open a command prompt. Navigate to the folder in which the Chirp! application is. When standing in the root folder, simply run the command `dotnet test` to run all tests. If you want to run tests for individidual parts of the system, run the command `cd test` to go to the test directory, and then go into an individual directory, for example by running `cd Chirp.Core.Tests/`, and then running the command `dotnet test`.

3 Ethics

3.1 License

We have chosen the MIT License for our application. Mainly due its simplicity, which makes it easy for students to collaborate and simplifies licensing issues. In addition, it allows for lots of people to use our webpage, and if we potentialy wanted to commercialize Chirp!, the license is really flexible and allows this.

3.2 LLMs, ChatGPT, CoPilot, and others

In this report we have only made use of the LLM's: ChatGPT and CoPilot.

ChatGPT: has been used to understand the theory behind some of the features which we have implemented, and generally not to generate code, unless explictly specified as co-writer. In some cases, it has been used for debugging purposes, indicated by including it as a co-writer in commit messages.

CoPilot: was only used in the later stages of the process to speed up code-writing-proces when writing generic code. (We decided in the beginning that we would follow Rasmus' reccomendation of writing code without CoPilot, when we were learning the basics of C#.)