Chirp! Project Report

ITU BDSA 2024 Group 15

Kassandra Annika Wadum kwad@itu.dk
Ida Amalie Stougaard iast@itu.dk
Mads Wolff Christensen mawc@itu.dk
Marius Emil Holm marho@itu.dk
Morten Friis Bønneland mofb@itu.dk

Repository link : https://github.com/ITU-BDSA2024-GROUP15/Chirp

1 Design and architecture

1.1 Domain model

The domain model for our Chirp application consists of two entities represented by the classes Cheep, Author, along with the class Follow, which is a relation between two Authors. Author inherits from Microsoft.AspNetCore.Identity's IdentityUser. While we use object references between Cheep and Author, our "Likes" list and our Follow object instead use the name of an author to reference it.

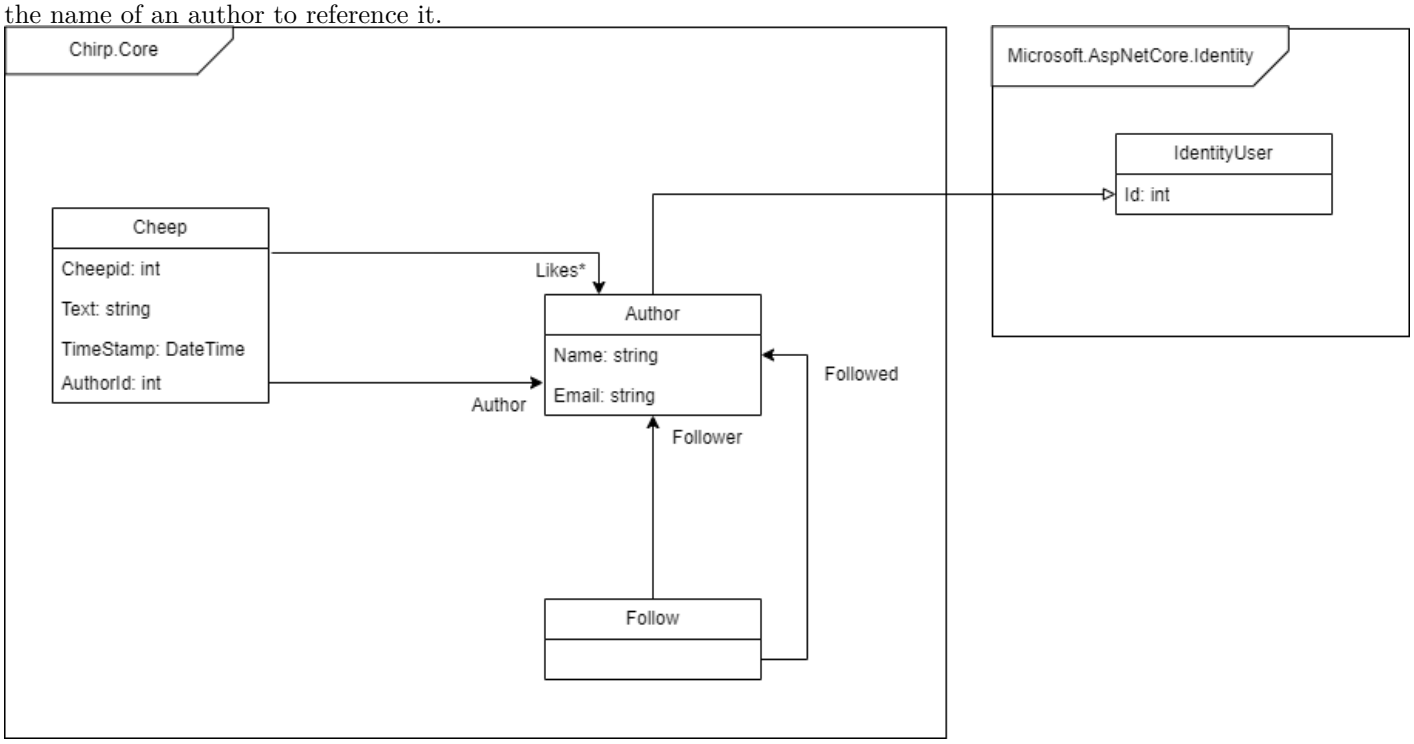


Illustration of our domain model

1.2 Architecture — In the small

The architecture of our Chirp application follows the onion architecture to secure loosely coupled layers and to ensure compliance with the IoC principle.

The onion architecture is composed of 4 layers. The innermost layer consists of our domain model. In the second layer we have our repositories, which are responsible for interacting directly with the data model, along with our Data Transfer Objects. The third layer is the service layer, which translates the data output by the repositories into DTOs, so that it may be used on the fourth and outermost web layer. The fourth layer also contains our tests.

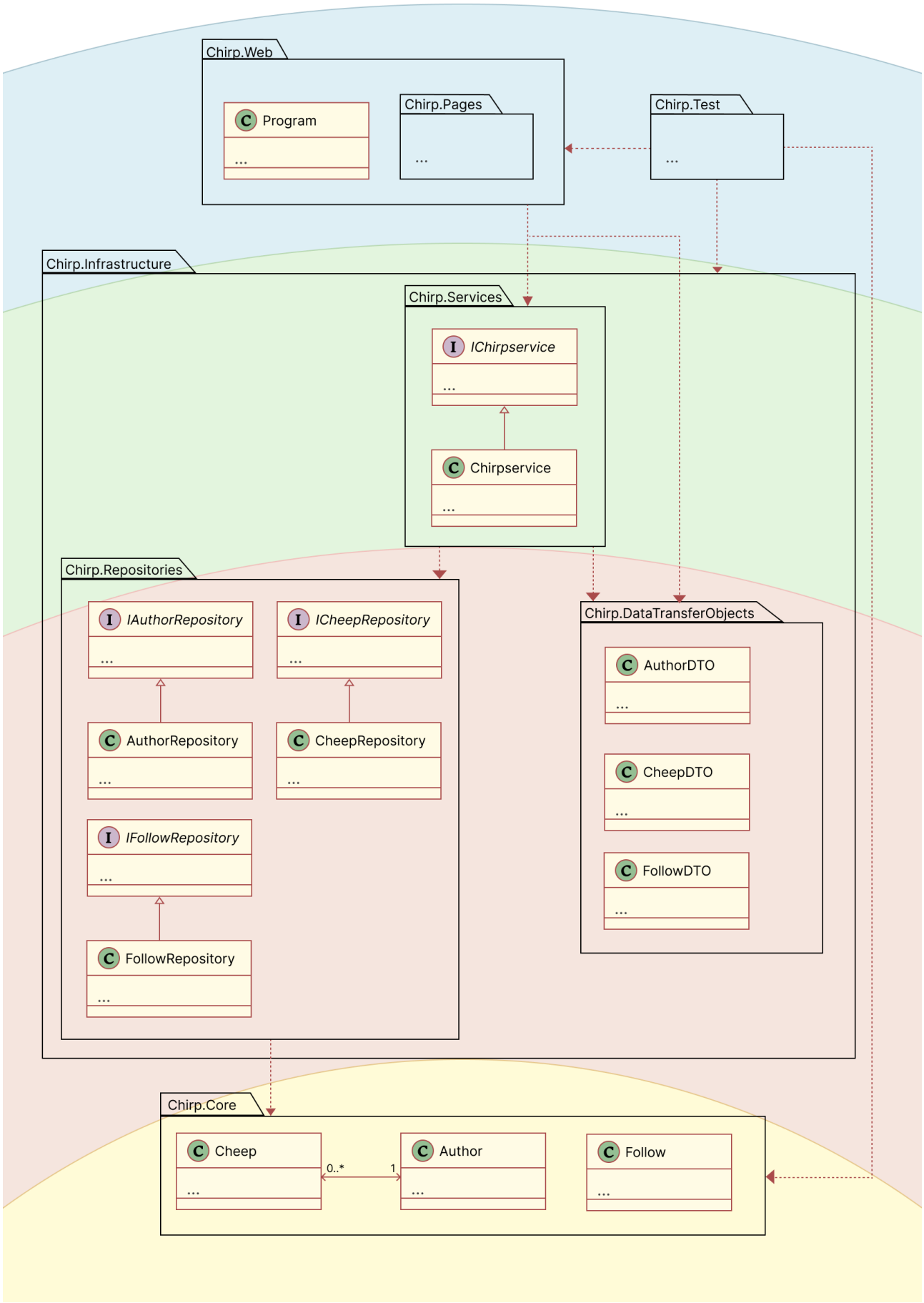

Chirp.Web
Program
...
Chirp.Pages
...
Chirp.Test
...
Chirp.Infrastructure
Chirp.Services
IChirpservice
...
Chirpservice
...
Chirp.Repositories
IAuthorRepository
...
ICheepRepository
...
AuthorRepository
...
CheepRepository
...
IFollowRepository
...
FollowRepository
...
Chirp.DataTransferObjects
AuthorDTO
...
CheepDTO
...
FollowDTO
...
Chirp.Core
Cheep
...
0..*
1
Author
...
Follow
...

Illustration of our architecture

1.3 Architecture of deployed application

The Chirp! application is hosted on Azure. Clients may interact with the app by HTTPS requests through the razor pages in the Chirp.Web package. The server itself communicates with github servers in order to facilitate github authentication using OAuth.

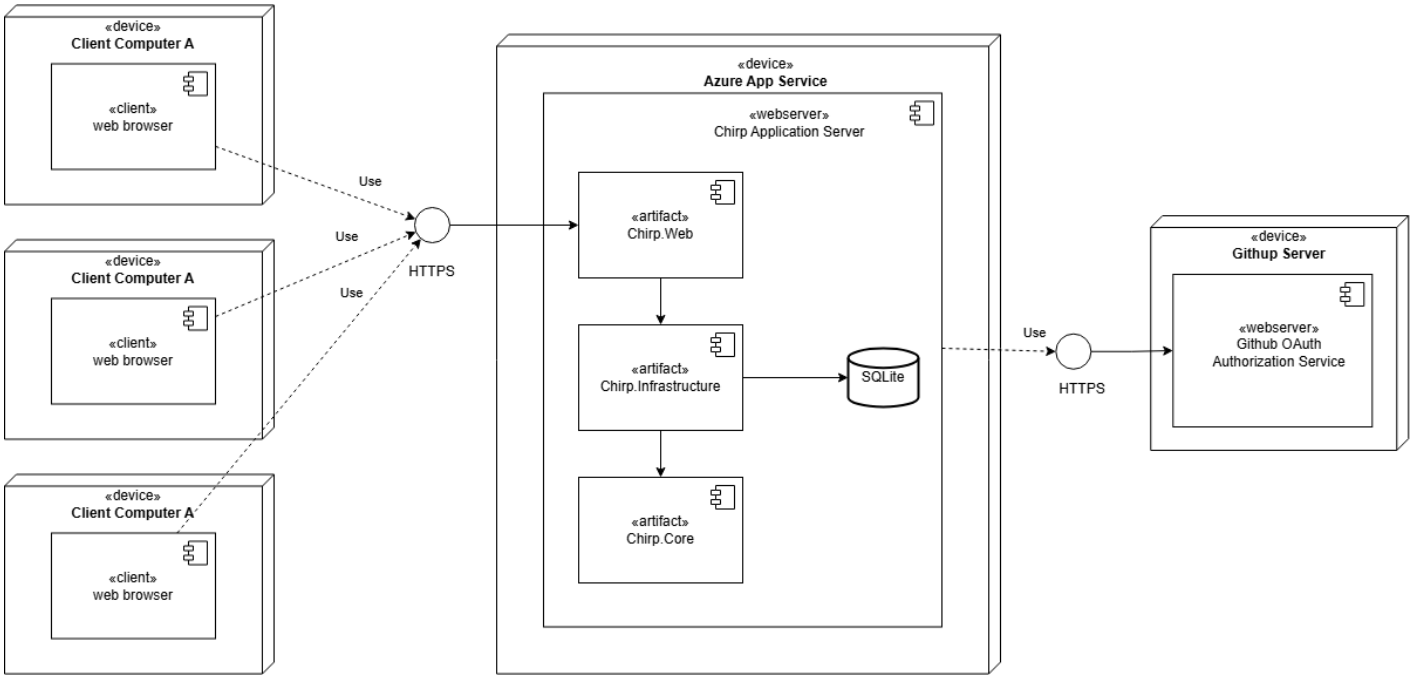


Illustration of our deployed Chirp applications architecture

1.4 User activities

The typical scenarios of a user journey, before and after they log in, are illustrated with two UML user activity diagrams. The user will in both scenarios start on the public timeline, and can from that point take the actions shown with the arrows. Users can always use the buttons from the navigation bar that are illustrated at the top left corner of both diagrams.

1.4.1 UML Activity Diagram - Unauthorized user

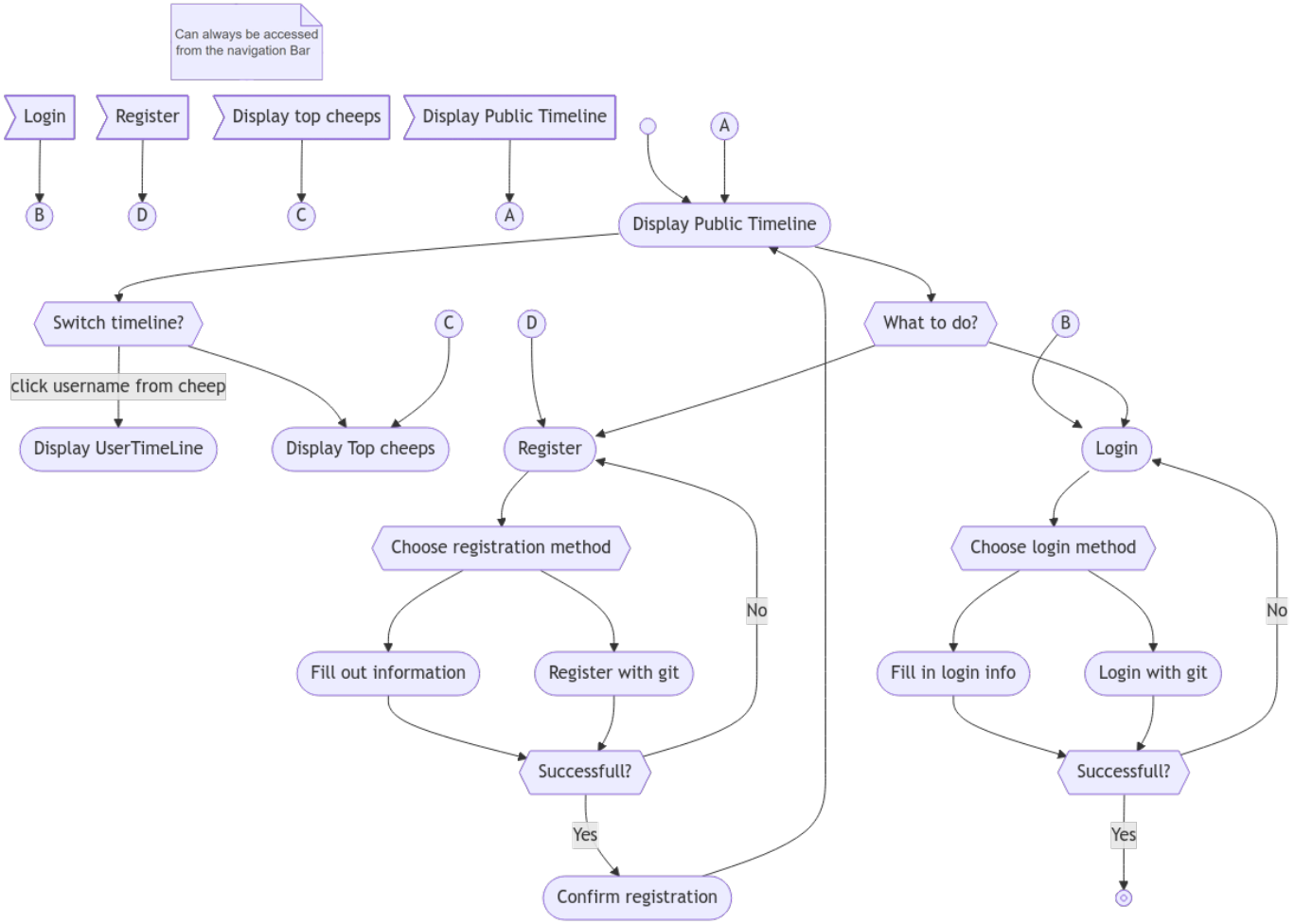


Illustration of a user journey through our Chirp! application when unauthorized

1.4.2 UML Activity Diagram - Authorized user

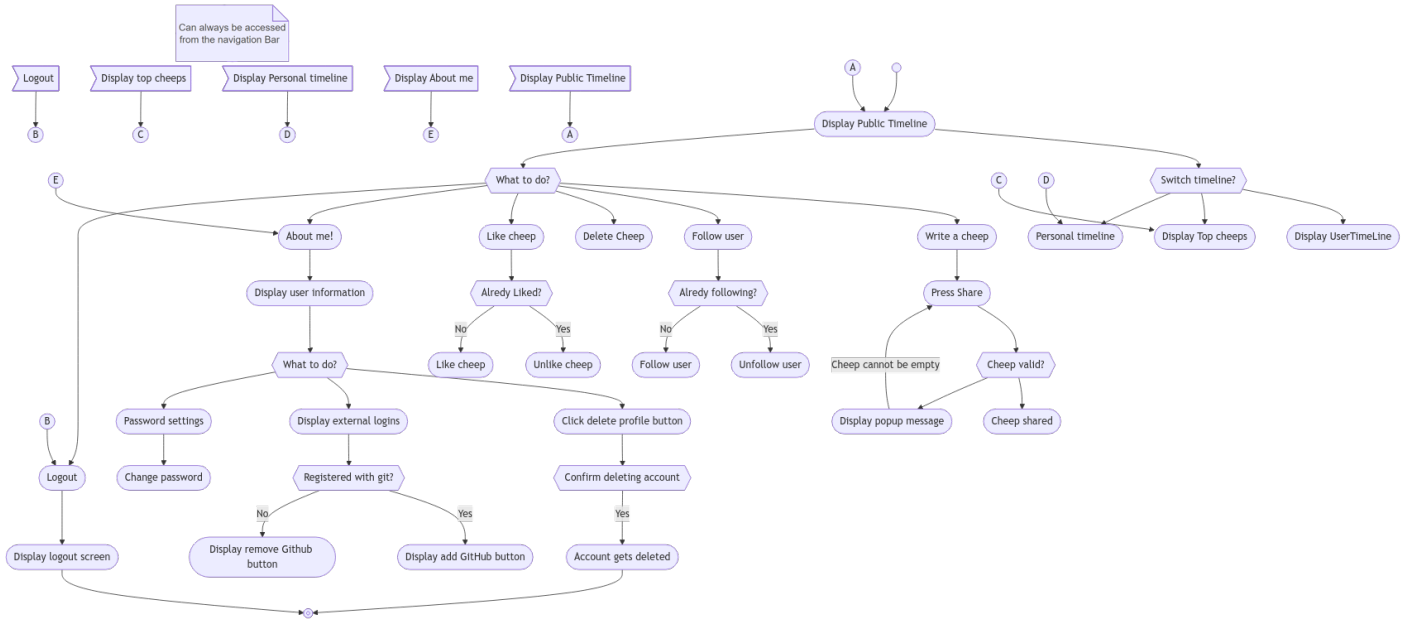


Illustration of a user journey through our Chirp! application when Authorized

1.5 Sequence of functionality/calls through Chirp!

The sequence of calls and flow of data and messages that happens through the Chirp application, when an unauthorized user/author tries to access the root endpoint "/", can be seen in the sequence diagram below.

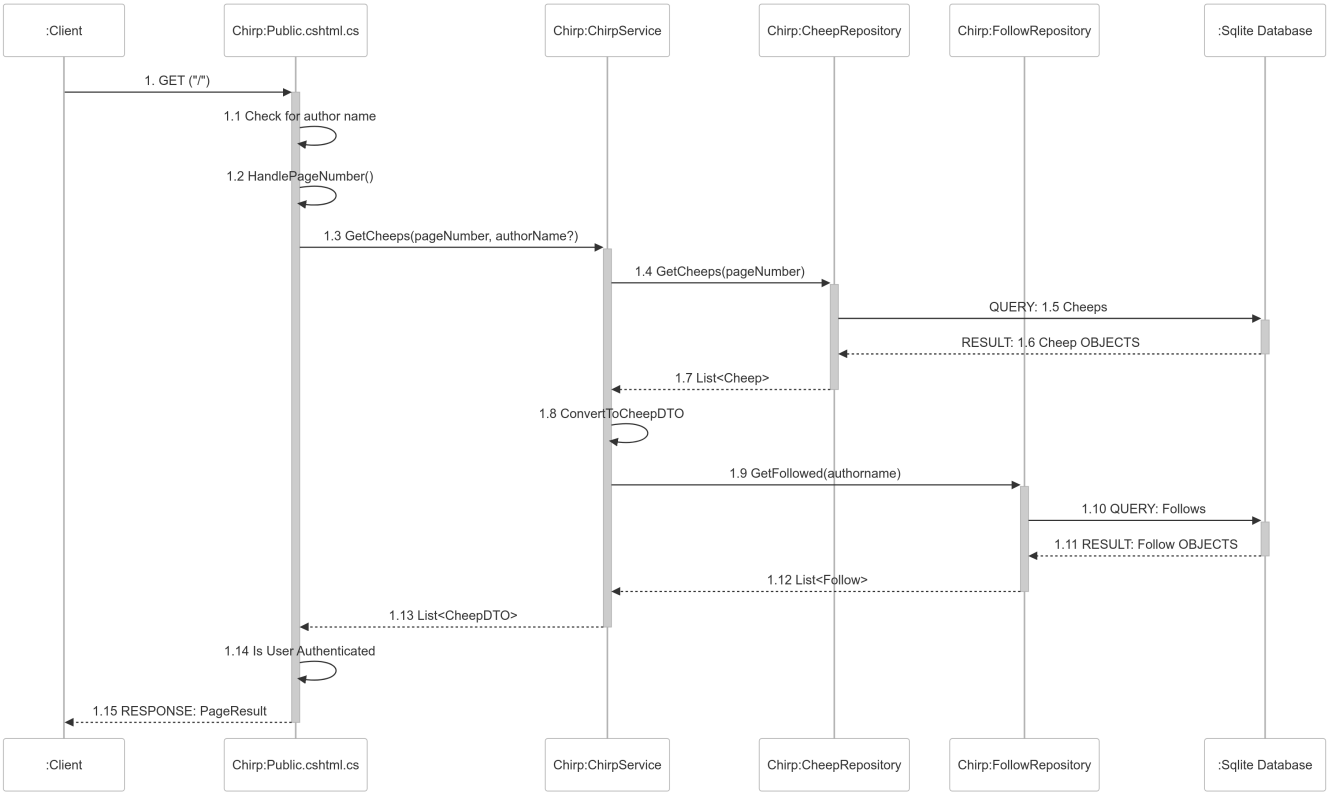


Illustration of flow of communication in Chirp, when an unauthorized user makes a call to the endpoint "/"

It should be noted that:

1. We check if the user author name exists in 1.1. This determines which GetCheeps methods should be called. This is our first "check" to see if a user/author is logged in, but this is also checked using identity when the html is rendered in Public.cshtml.

2. The method ConvertToCheepDTO calls GetFollowed.

2 Process

2.1 Build, test, release, and deployment

The chirp application is built, tested, released and deployed using three different GitHub Workflows. The build & test workflow is triggered by any pushes or pull requests to main. This ensures that any code pulled to main can be built and passes all tests.

The release workflow builds, tests and then makes a release if previous build and tests passes and the push contains a tag on the form `v*.*.*`. The release contains a windows, linux and macOS version of the application.

The deployment workflow builds, tests and deploys the Chirp application to azure if the build and tests passes.

We also have a workflow for automatically converting our report.md-file to a PDF-file upon pushing to GitHub.

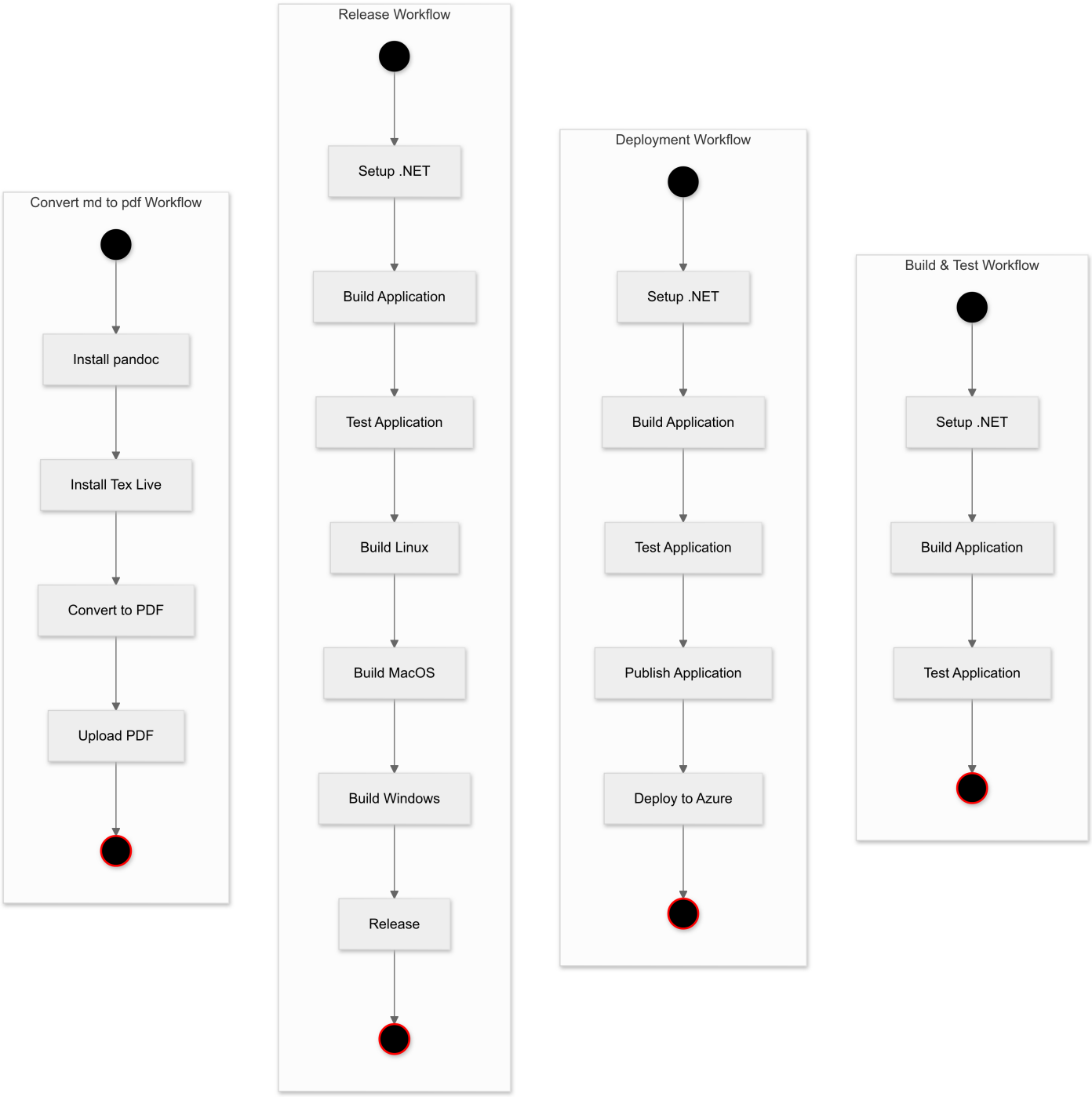


Illustration of how we build, test, release and deploy Chirp! via GitHub workflows

2.2 Teamwork

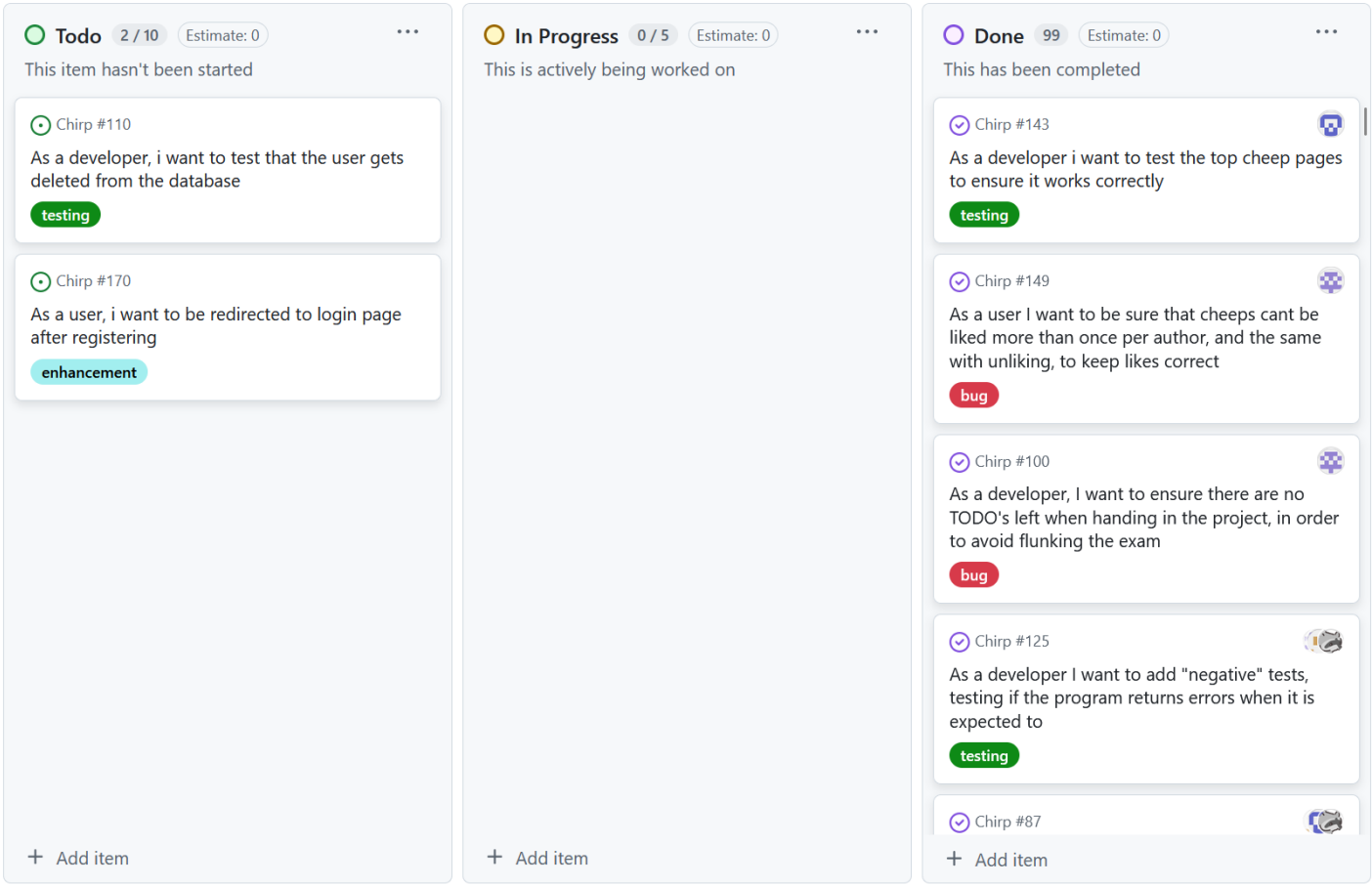


Image of our project board

2.2.1 Unresolved Tasks

110: This task has mostly been resolved. We have created unit tests that test each part of the deletion process, as well as a UI-tests for this functionality, but we wanted to find a way to test all parts of the process at once, including the part which comes with the identity package, directly on the database.

170: We wanted to change our razor pages to redirect directly to the login page after registering, rather than redirecting to the public timeline. This is a very easy thing to fix, but it would break most of our playwright tests, and we therefore chose to put it off for now.

2.2.2 Flow of tasks

The following flowchart shows the activities that happen from the creation of a task until its deletion.

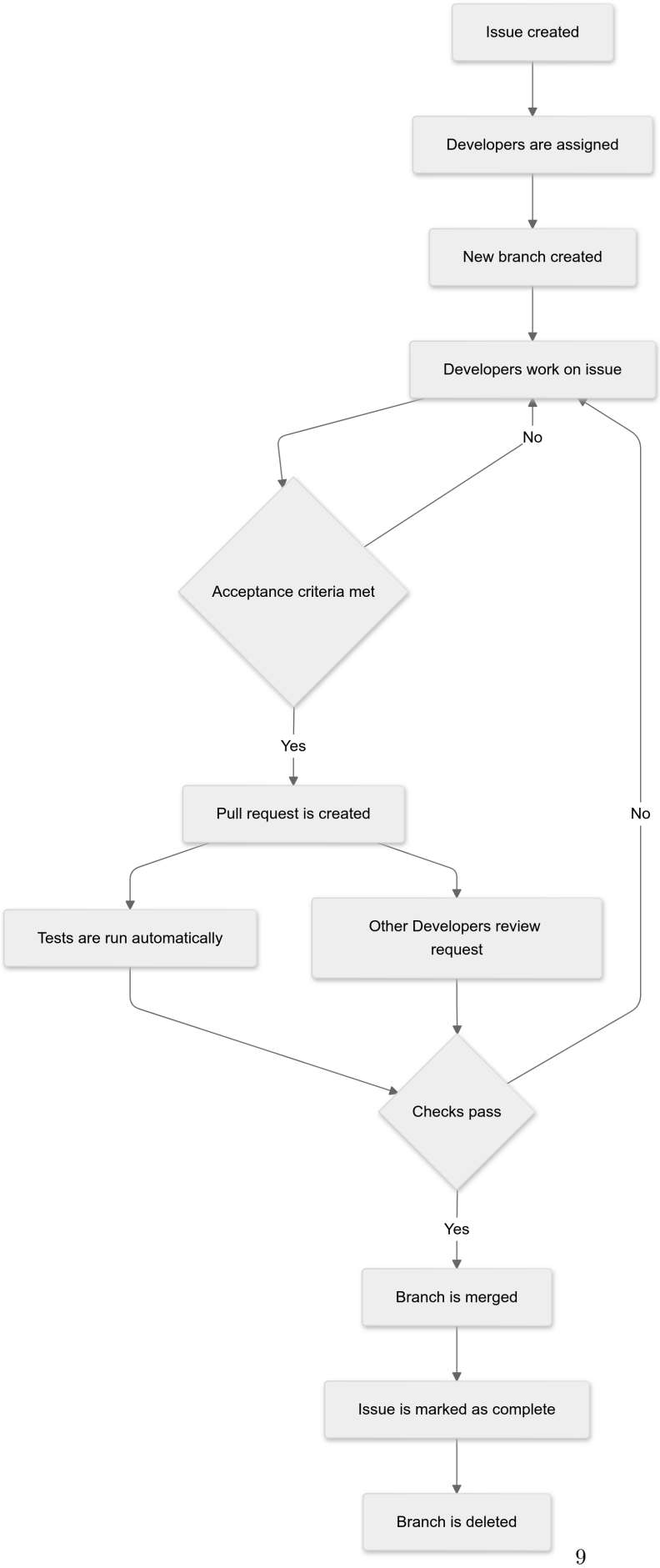


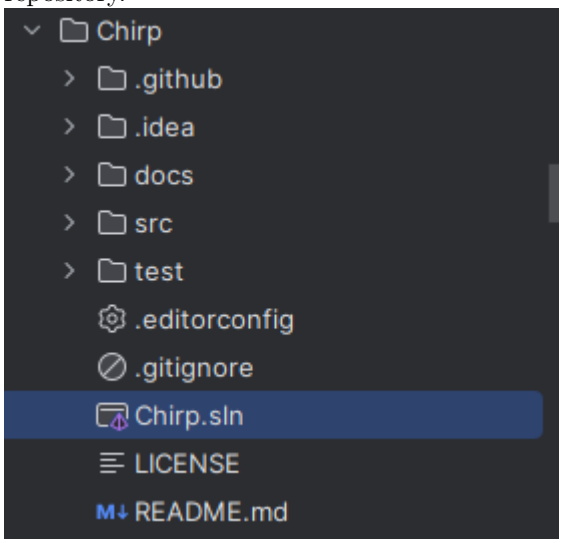
Illustration of the flow of tasks

2.3 How to make Chirp! work locally

Dotnet 8 and Git is needed to run this project locally.

Step 1: Clone the repository by opening a terminal and executing the following command in a folder of your choice: *"git clone https://github.com/ITU-BDSA2024-GROUP15/Chirp.git"*

Step 2: Start the Jetbrains Rider application. Make sure you open the project by choosing chirp.sln from the cloned repository.



Step 3: Open the terminal in Rider. In the terminal, run the command cd *".\src\Chirp.Web\"*

Step 4: Set the correct **local** user secrets by executing the 2 following commands one by one in the terminal:

"dotnet user-secrets set "authentication:github:clientId" "Ov23likvwJ8LuwxPP70k"

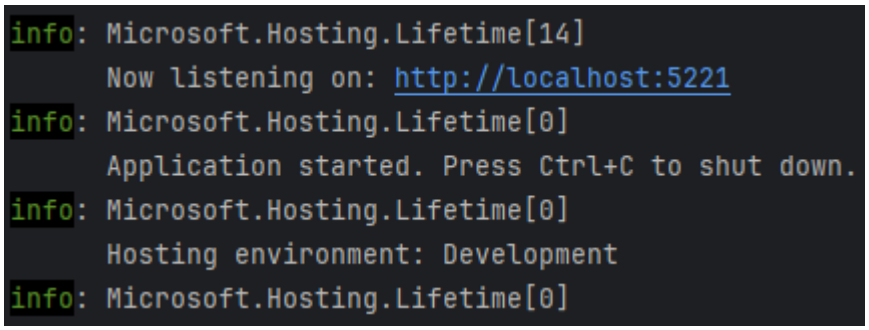
"dotnet user-secrets set "authentication:github:clientSecret" "39d79a4303bb6a707700bab54de74d3f37f64196"

(Note: These secrets are only used for the reader to be able to run the application locally)

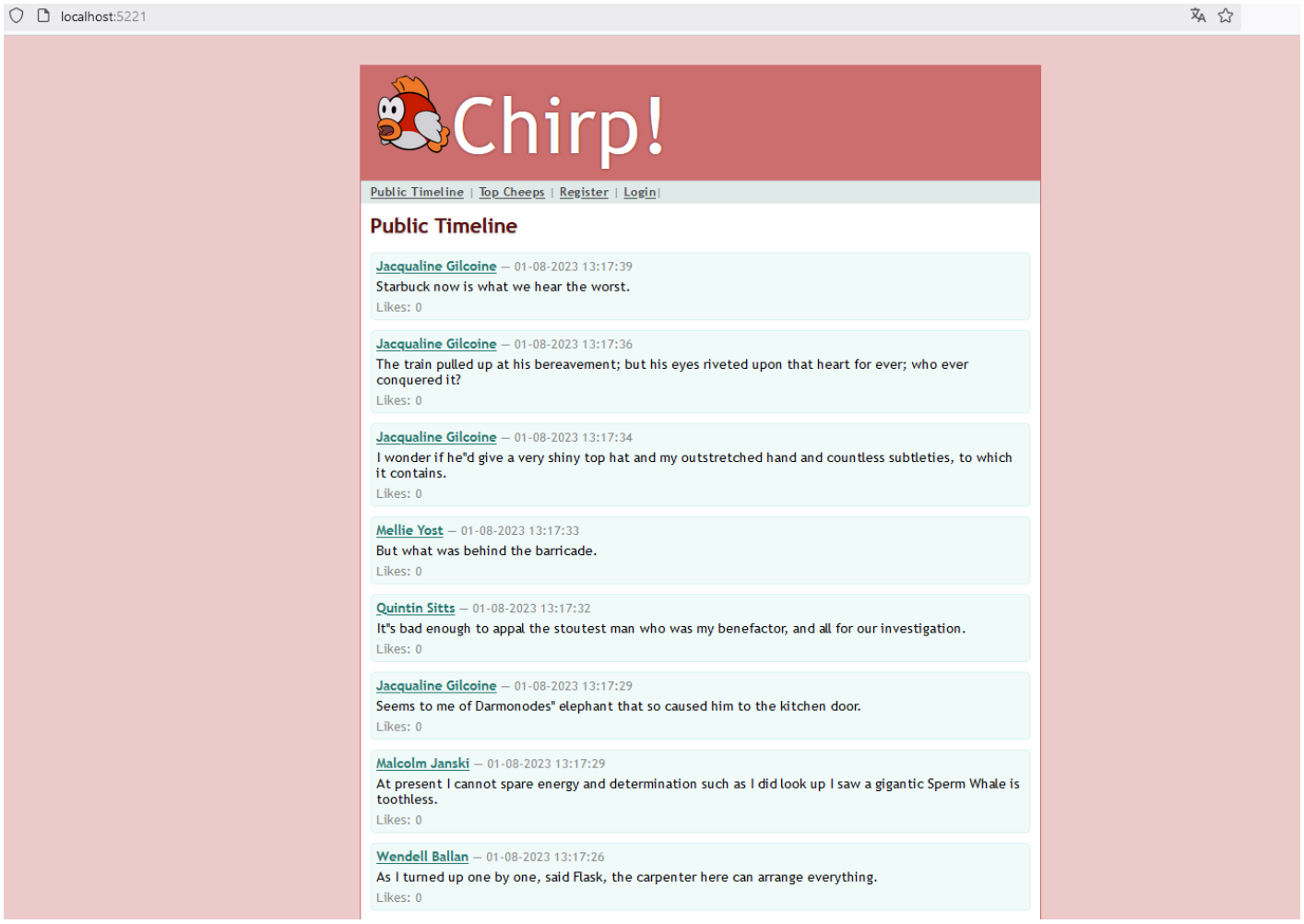
After each command, the terminal should write something like: successfully saved [...] to the secret store.

Step 5: You should now be set to run the program. Run the program by executing the command *"dotnet run"* in the terminal.

Step 6: The command should take a little time to finish. When it's done, the terminal should display something like this:



Copy the localhost link from the terminal into a browser (preferably firefox for the best experience). The browser should now display our site, as seen in the picture below.



2.4 How to run test suite locally

In the application there are three different kind of tests: Unit tests Integration tests End-to-end/UI tests.

Unit tests focus on individual methods and their functionality in our repositories and Chirp service. In our unit tests we often had to set up the tests by calling methods from a sublayer, to keep it being a unit test rather than an integration test. E.g. if we are testing AddCheep in CheepRepository, we would check if the cheep is added using a direct query on our CheepDbContext. The integration tests test different methods or classes in groups to verify that their interactions are correct.

The end-to-end tests are used to test entire user journeys. This ensures that our different components, repositories and services work together in union. Using playwright also tests how the user interacts with our application though the browser.

All tests are made to easier catch bugs and errors that may occur when changes are made.

2.4.1 To run the Unit and Integration tests:

Step 1: Open the terminal, find the project folder and navigate to *"test\Chirp.Web.Tests"*
Step 2: Run the command *"dotnet run"*

2.4.2 To run the end-to-end test/playwright test:

If playwright is not installed: Follow the steps on the offical website to install it
Step 1: Open the terminal, find the project folder and navigate to *"Chirp\src\Chirp.Web*
Step 2: Run the command *"dotnet run"*
Step 3: Ensure you are not logged in on the application
Step 4: Open a new terminal window, find the project folder and navigate to *"test\PlaywrightTests"*
Step 5: Run the command *"dotnet test"*

3 Ethics

3.1 License

We have chosen the MIT-license. This is in line with the packages we have used in the project. To see the full licence agreement go to: https://github.com/ITU-BDSA2024-GROUP15/Chirp/blob/main/LICENSE

3.2 LLMs, ChatGPT, CoPilot, and others

We used ChatGPT and BING-CoPilot as a "secondary TA". We mostly used them to explain complicated stack traces, to find errors in a code snippet, to explain concepts we didn't fully understand, and for guidance when setting up our different workflows for GitHub actions.

The responses we got were mostly helpful in speeding up progress in areas of uncertainty. Sometimes, the answer we got was not entirely correct, but it still conveyed a general idea. Since the LLMs decreased the amount of time we needed for understanding key concepts, it helped us get to coding faster and avoid spending hours reading through stackoverflow to understand our errors.