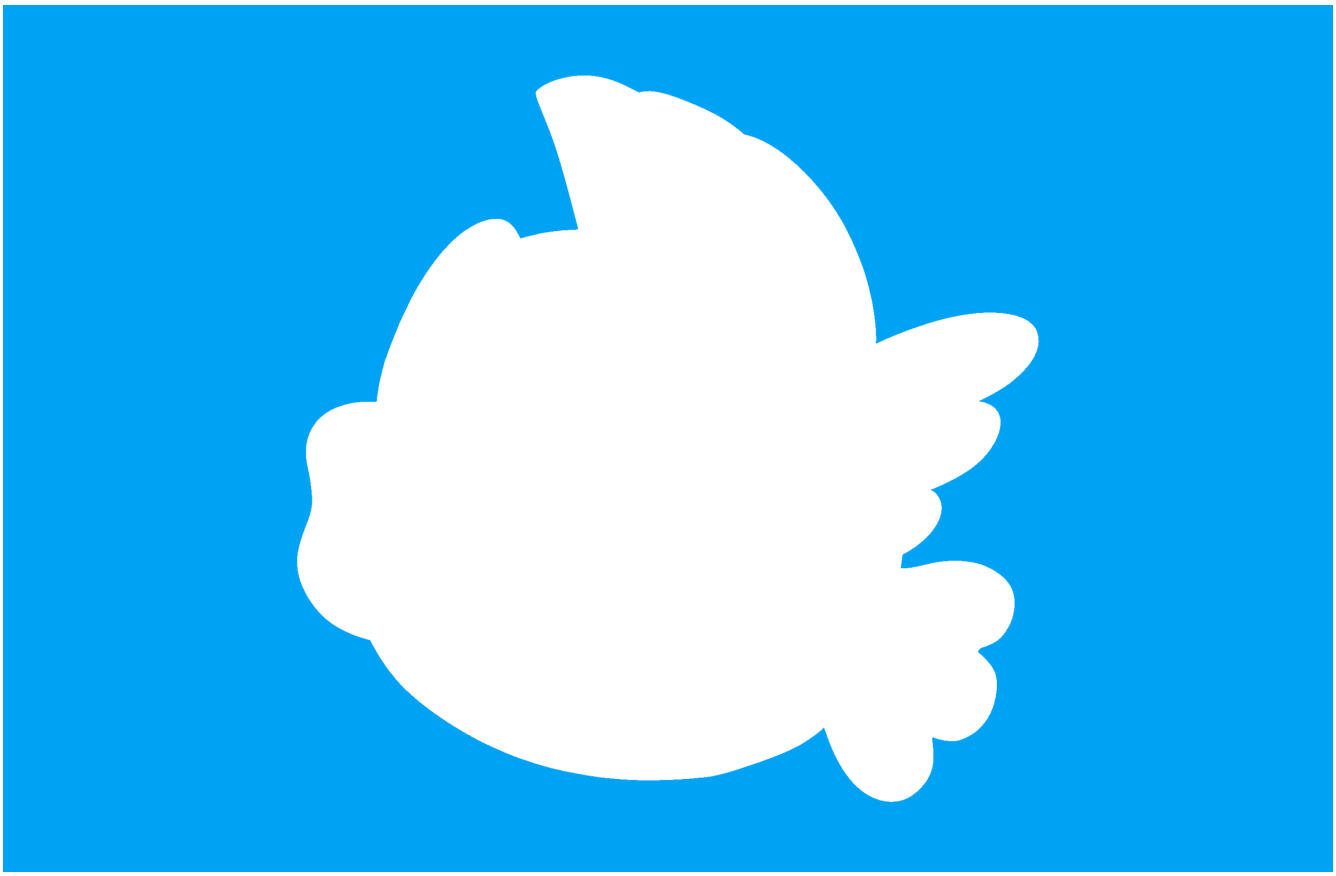
Chirp!

Group 20

Alexander alefr@itu.dk Simon simoh@itu.dk Sif esja@itu.dk Radmehr rads@itu.dk



IT UNIVERSITY OF COPENHAGEN

Course code	BSANDSA1KU
Name of course	Analysis, Design and Software Architecture
Course manager	Helge Pfeiffer - ropf@itu.dk
Project title	Chirp!
Group number	20
Date	December 19 2024
School	IT-University of Copenhagen

1 Table of contents:

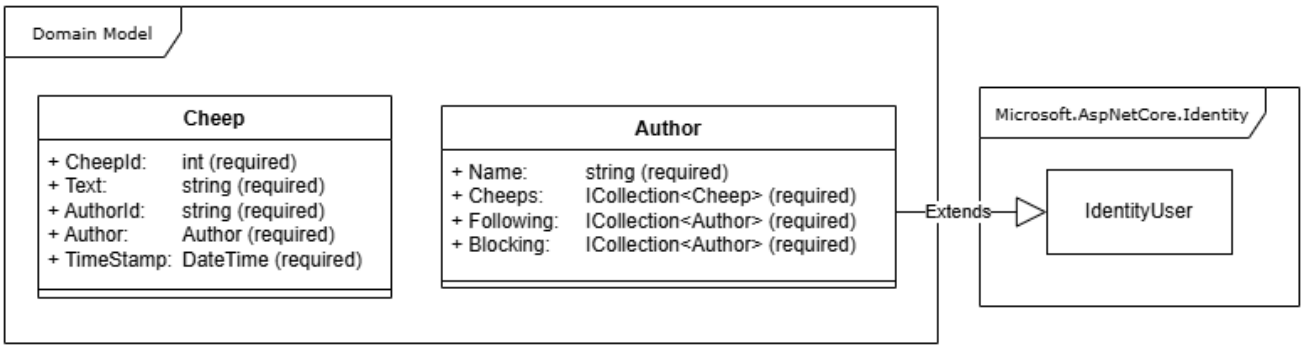
2 Design and Architecture of Chirp!

2.1 Domain model

Our domain model for our application: Chirp. It is designed with the purpose to simulate 'Twitter' or 'X' as it is called now. It consists of two parts, a Cheep class and an Author class that are closely tied.

The Cheep has some fields where it stores all the information about a cheep. ***CheepId***: A unique number assigned to the cheep. ***Text***: Store the written message, with a limit of 160 characters. ***AuthorId/Author***: Referencing the Author that wrote said message, such that the messages and authors are connected. ***TimeStamp***: To get the time the cheep are posted.

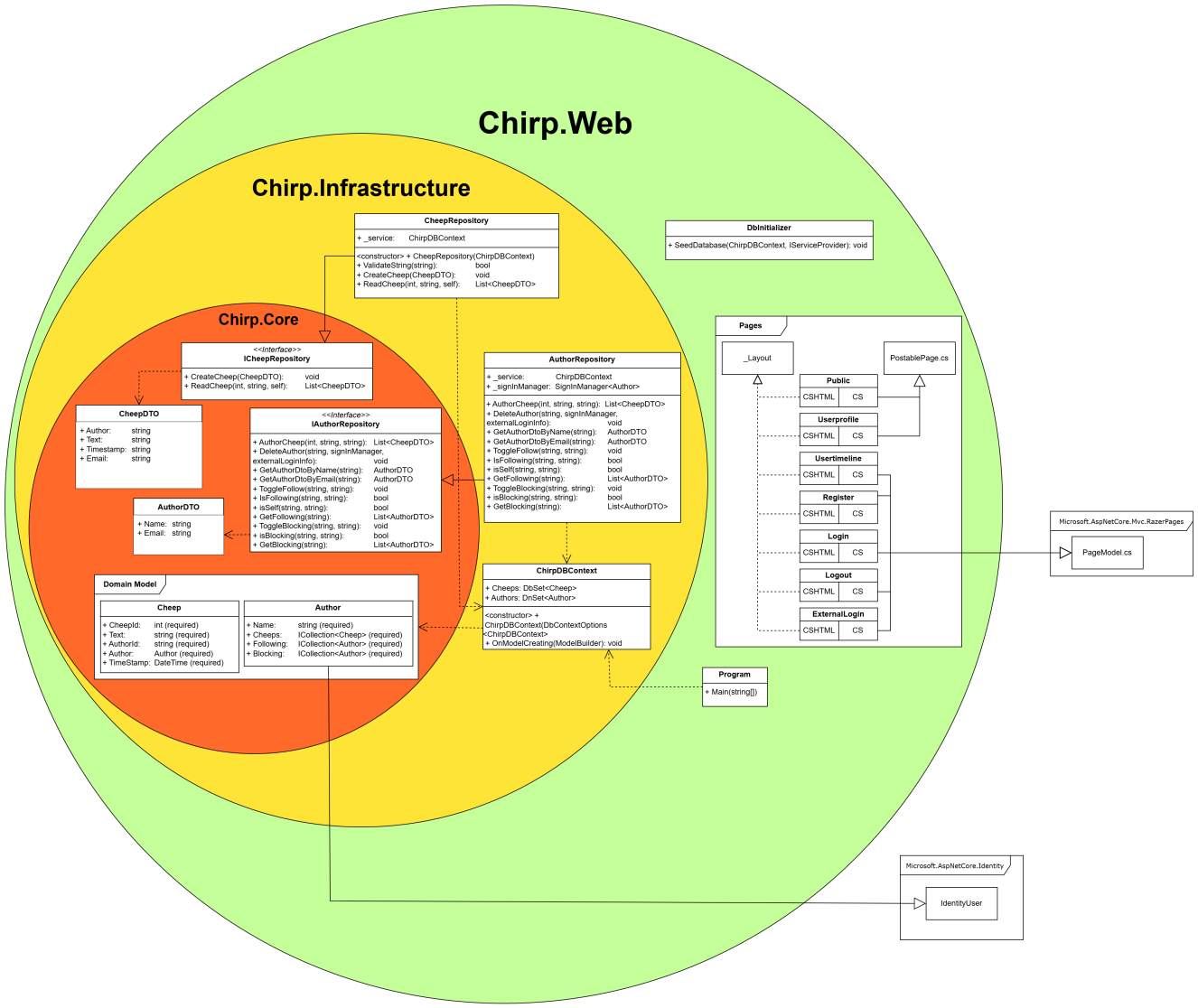
The Author has also some fields where it stores the information about a user. ***Name***: The name of the author, which is limited to 50 characters. ***Cheeps***: Stores all cheeps written by the author. ***Following***: Stores all other authors that the users has chosen to follow, which get added to their timeline and About Me page. ***Blocking***: Stores all other authors that the users has chosen to block, which get added to their About Me page and filtered out everywhere else. Author extends *Microsoft.AspNetCore.Identity*. It also provides a hashed password and secures proper email verification.



UML class diagram of our domain model

2.2 Architecture In the small

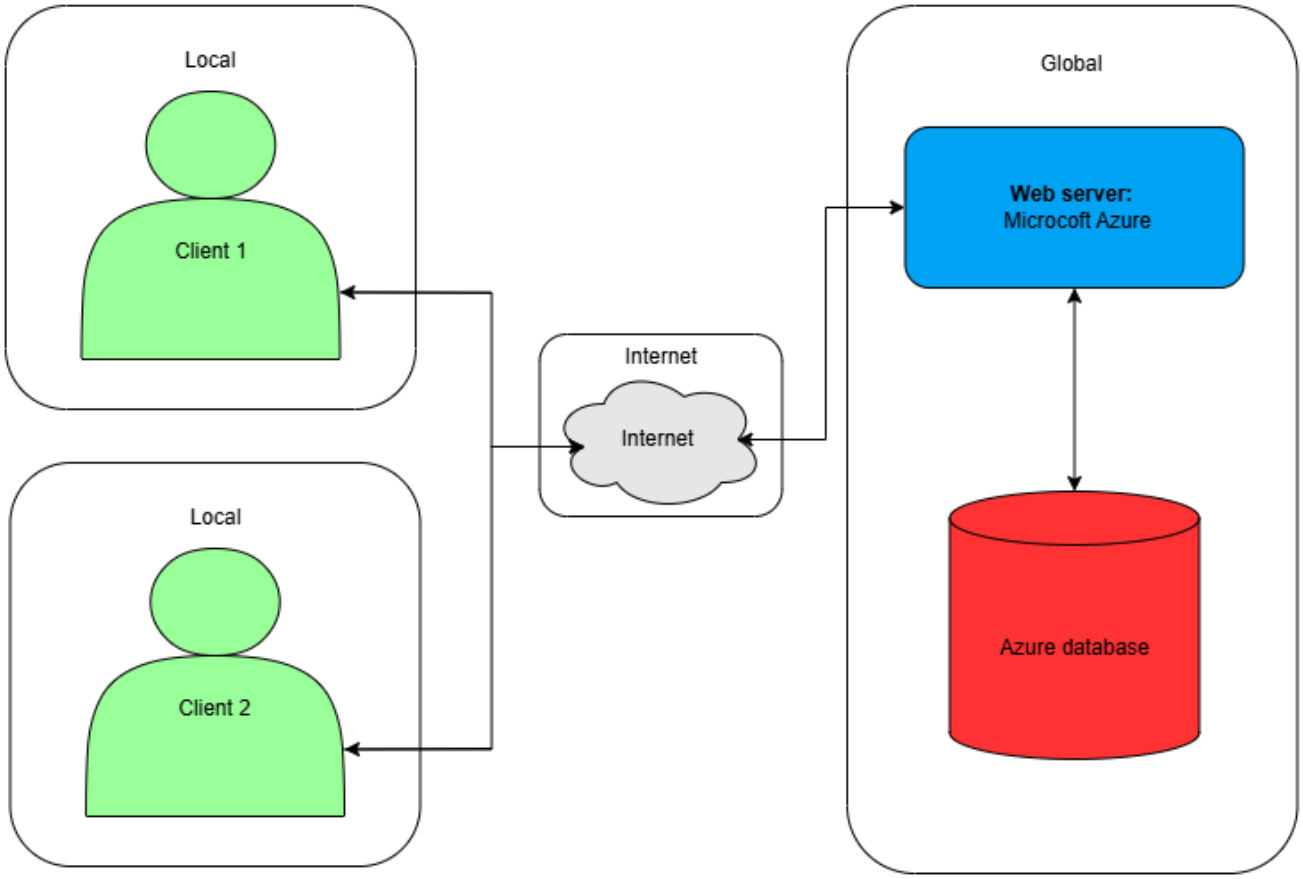
The diagram below illustrates our chirp program in Onion Architecture. The architecture has three main layers: - The inner layer - Chirp.Core: Contains our DTOs classes ***AuthorDTO*** and ***CheepDTO*** along with interfaces for ***AuthorRepository*** and ***CheepRepository*** and the ***Domain Model*** - The middle layer - Chirp.Infrastructure: Contains ***AuthorRepository***, ***CheepRepository*** as well as ***ChirpDBContext***. - The outer layer - Chirp.Web: Represents the web application as it contains ***Program*** as to run the program, our ***DbInitializer*** aka our database. The folder ***Pages*** uses the Microsoft package '*Microsoft.AspNetCore.Mvc.RazorPages*', as they are responsible for the layout.



Onion architecture of the Chirp application

2.3 Architecture of deployed application

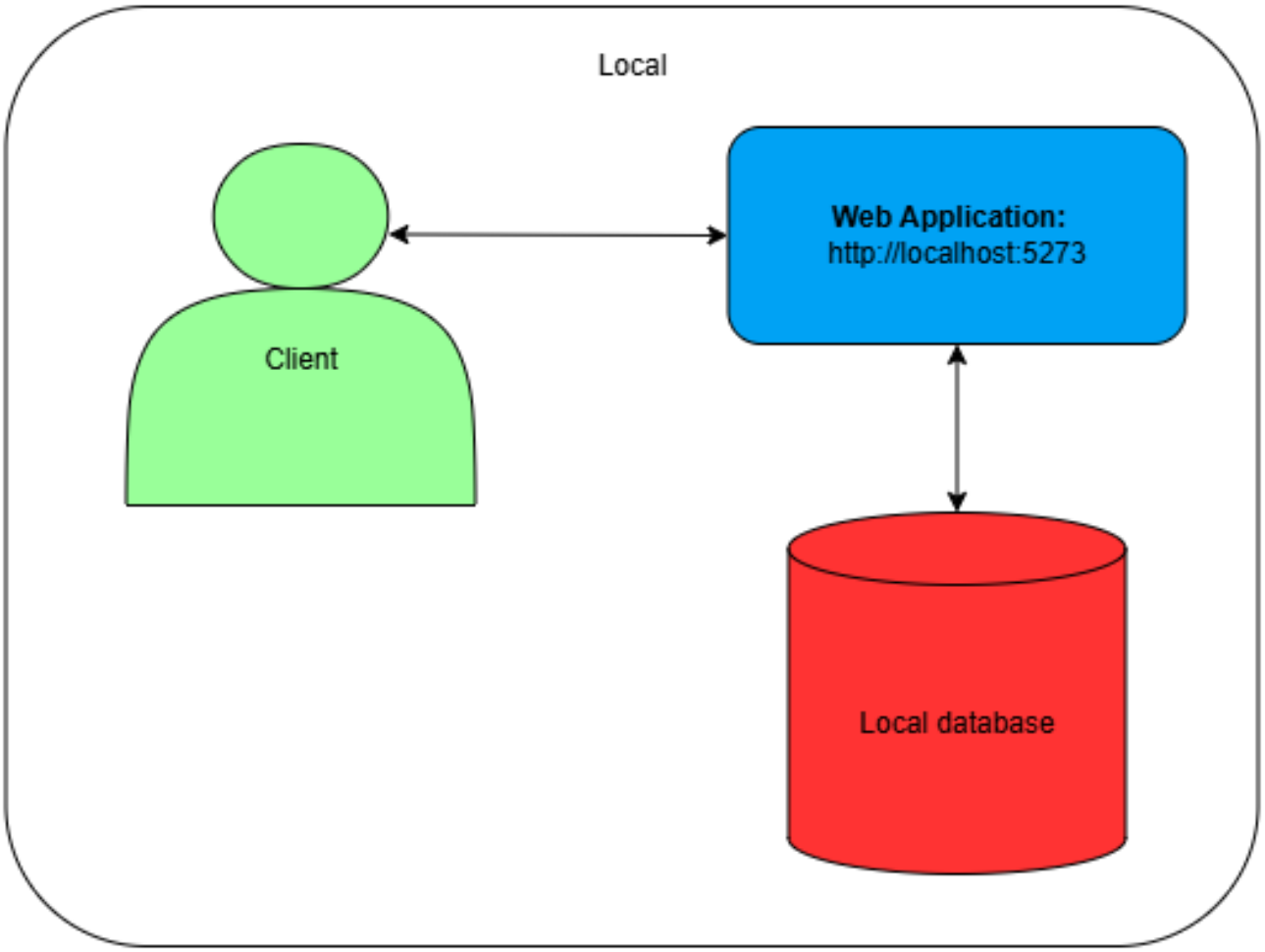
2.3.1 Client-server application



Architecture Illustration of our client-server application

On the left side there are some clients (there can be a number of clients, but we have only shown 2 clients) and they communicate with the internet. They are interacting with web servers located on Microsoft Azure server. The web server communicates to the database, as it needs to store or retrieve the data the client should use, and then it gets sent back.

2.3.2 Local client application



Architecture Illustration of our local client application

On the left side there are our local clients. The client interacting with web server There is a local server: localhost:5273. The server communicates with 'mychirp.db', as it needs to store or retrieve the data the client should use, and then it gets sent back.

2.4 User activities

Here is a flowchart that illustrates the different paths there are in our program from the three 'Login' options 'login with authorized user', 'login with GitHub' or 'login with non-registered user'. After the log-in process, all the different things there are many different things that can be done For example 'writing a cheep', 'block a user', and 'look at the "***About Me***" page'. And our user journey ends with clicking "***log out***" or clicking on the "***Forget me***" buttons.

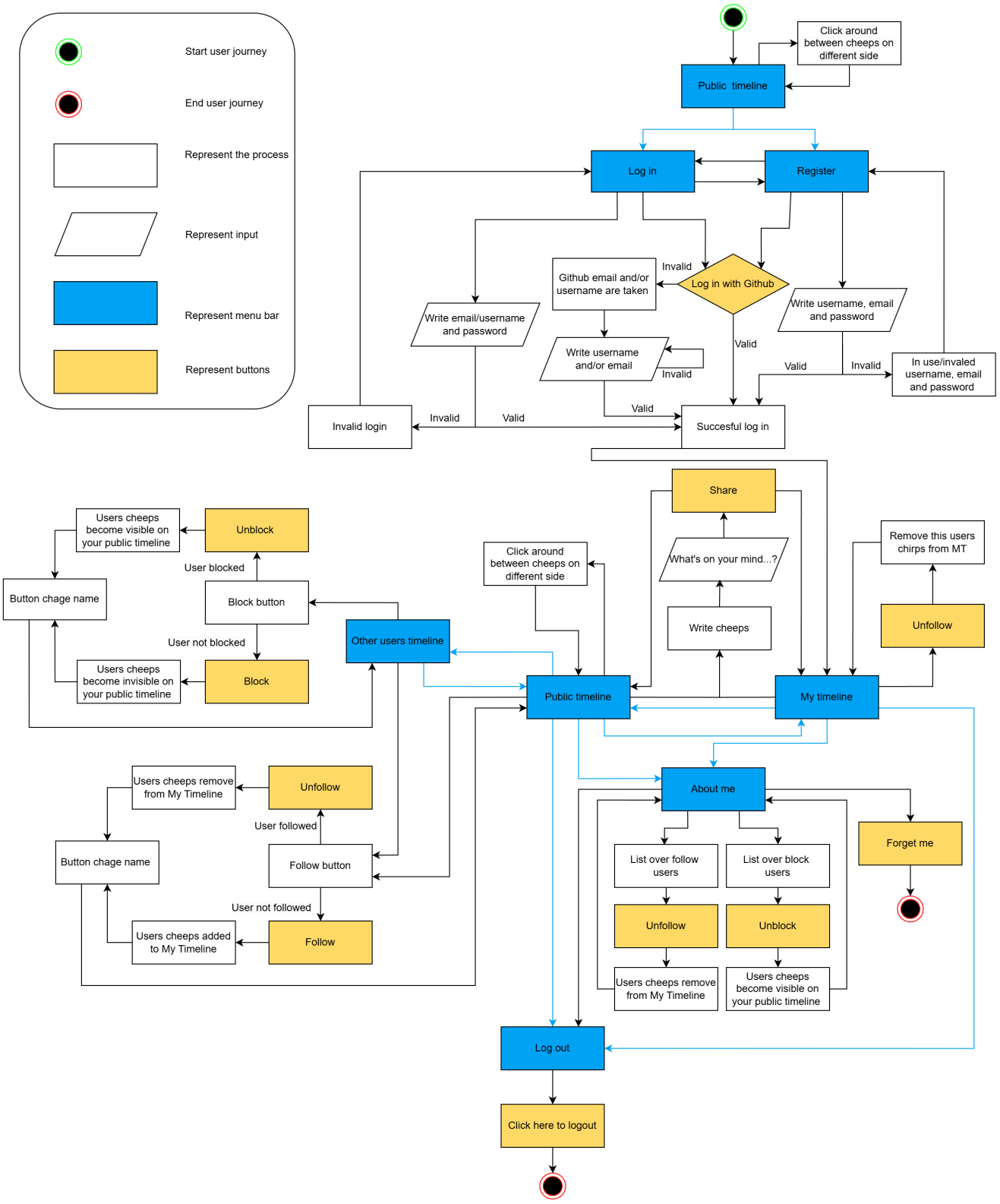
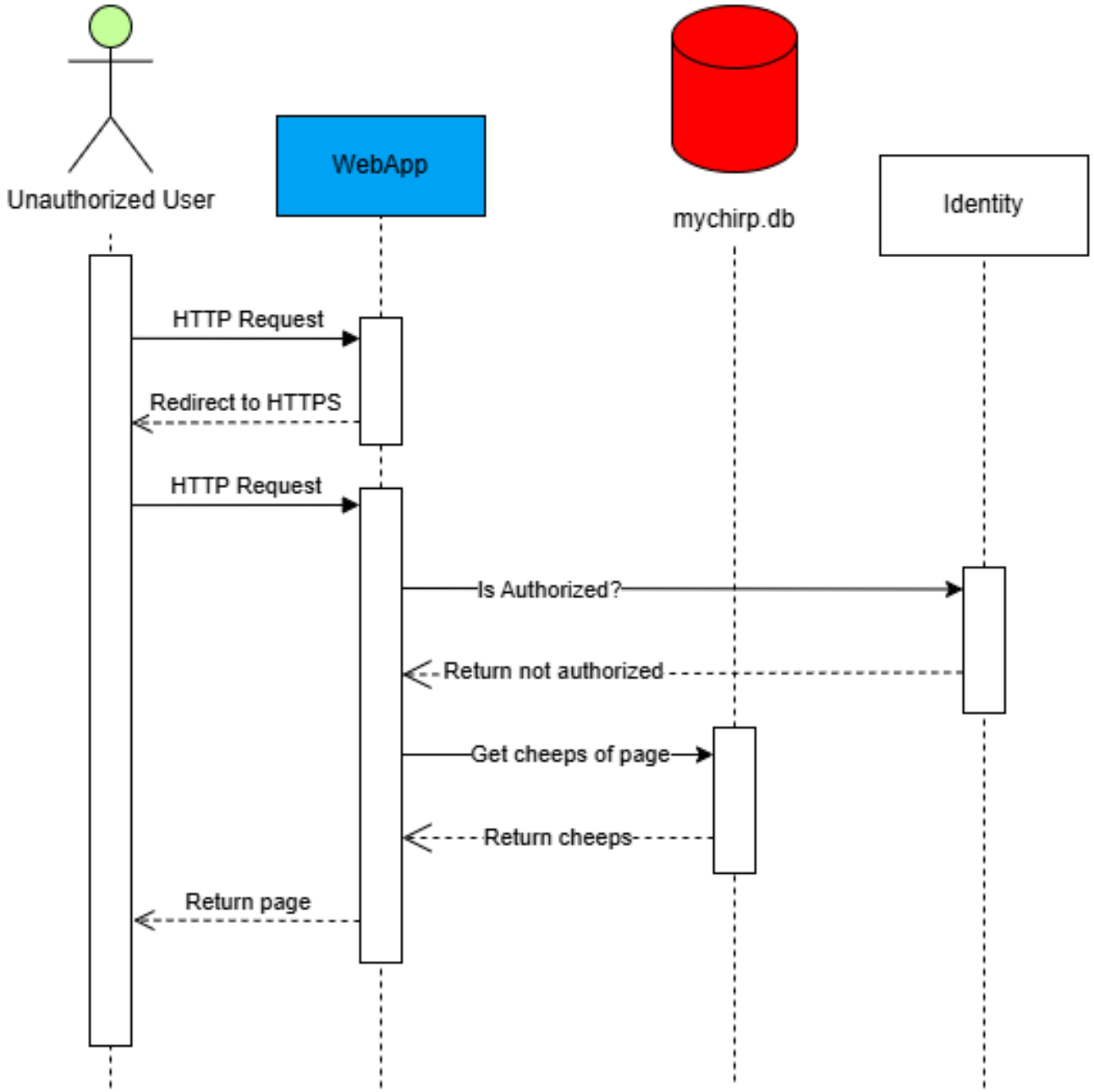


Illustration of flowchart of a non-authorized user

2.5 Sequence of functionality/calls through Chirp!



UML sequence diagram from HTTP request to root endpoint

The sequence diagram shown above shows a singular unauthorized user process for being shown the homepage when first finding the website.

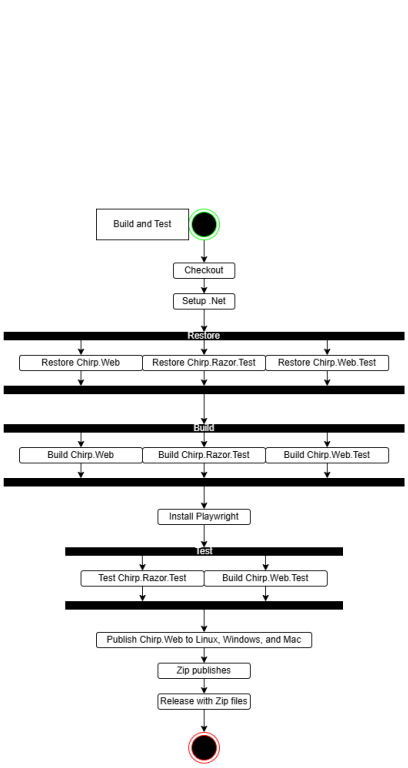
Standard protocol when getting a site is requesting its HTTP version, which is not stable and can easily be compromised, which is why Chirp! then redirects to a HTTPS version and stores a cookie for the user, which makes the user always connect to the HTTPS version when requesting the site.

Afterward, Microsoft Identity attempts to check whether the user is authorized (logged in) or not. Due to this user being completely new and having no prior cookies, it is determined that they are unauthorized.

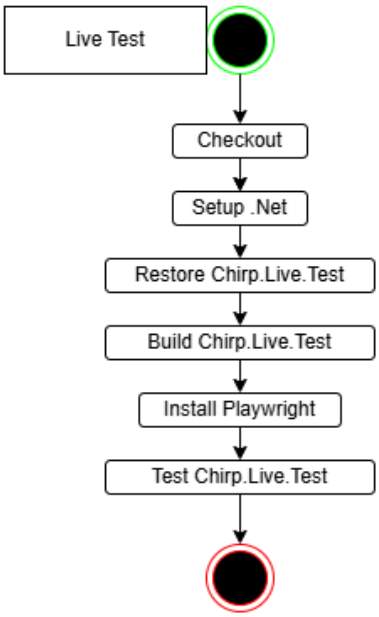
Then, the Chirp! homepage gets loaded, meaning the page has to get all available cheeps that can be shown on one page, which is then sent to the user. They are now on a static page.

3 Process

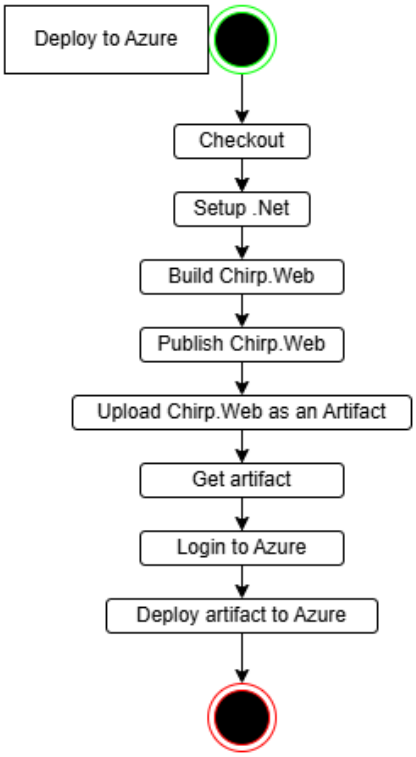
3.1 Build, test, release, and deployment



Build and Test



Live Test



Deploy to Azure

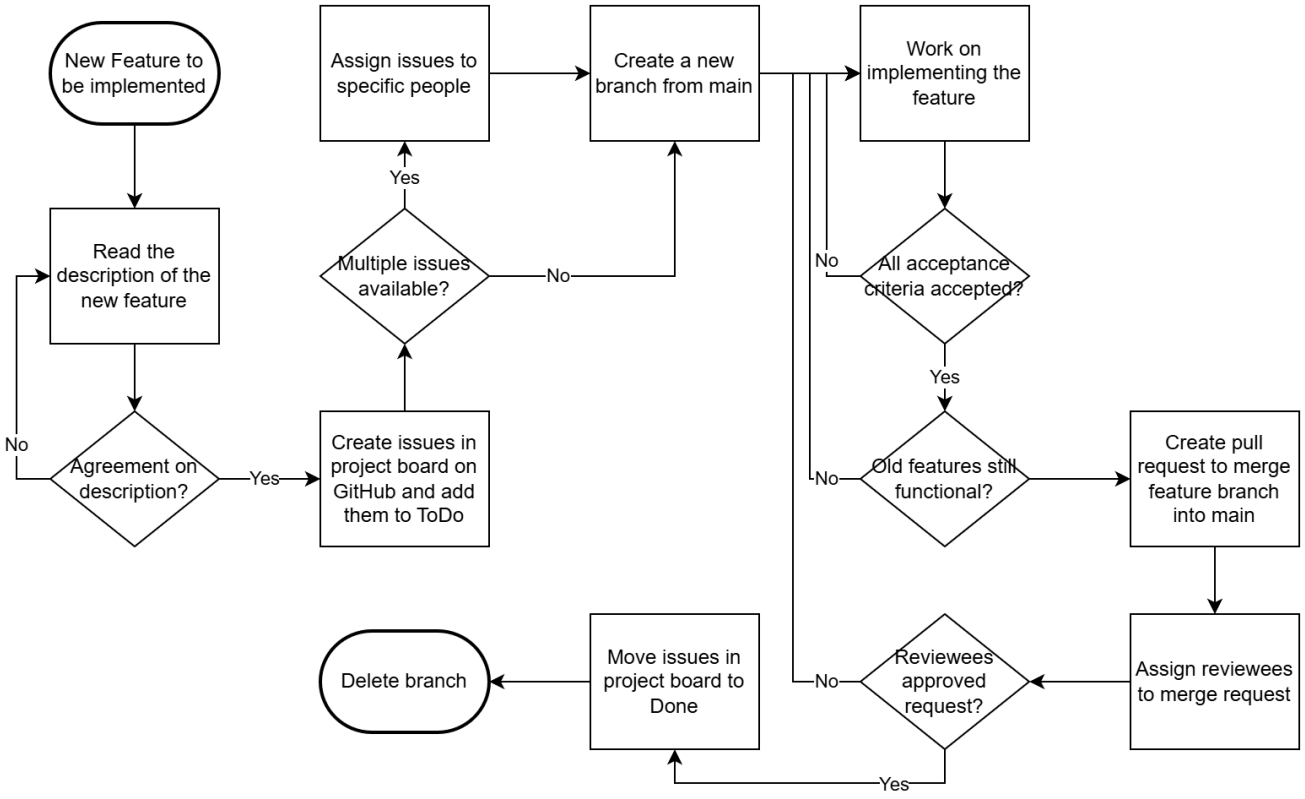
Build and Test were made to create a release of the newest version-tagged build, as a compressed zip-file for Mac, Linux and Windows. It also runs through all unit tests and UI tests.

Live Test runs every 6 hours to check if our server is still alive and currently running.

Deploy to Azure is run whenever a build is version-tagged. It builds and then publishes it and sends it to Azures server to override the previous build.

Both Live Test and Deploy to Azure can also be activated via the GitHub actions interface, if deemed necessary. Build and Test can also be activated this way but will always fail.

3.2 Team work



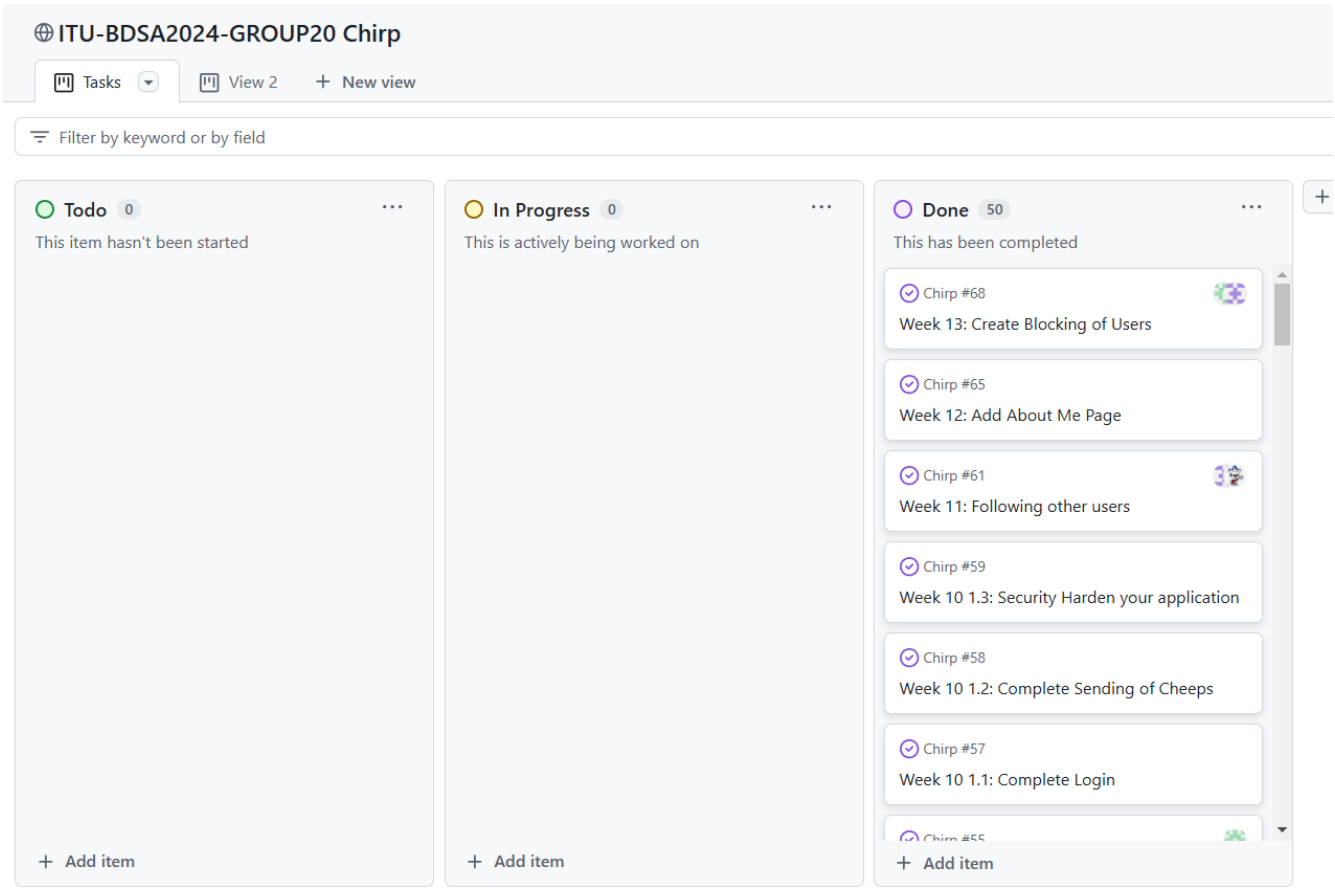
Flowchart of feature development

Whenever we were assigned a new feature, we made sure to properly understand the assignment and first then create an issue on our GitHub project board, which is placed in the ToDo section, with plenty of acceptance criteria. While we weren't entirely consistent in assigning issues to people in the team, we still tried to uphold it whenever possible.

At this point, we were ready to create a new branch and begin working on implementing the feature. Only after confirming that the implementation meets all acceptance criteria and does not impede previous features, we create a pull request to merge the branch into main.

Pull requests has to be peer-reviewed and accepted by other team members before it can be resolved, after which, the issues will be moved to the Done section of the project board and the feature branch will automatically be deleted.

Throughout the project we made a project board where we took the weekly assignment on 'README_PROJECT' and thereafter made them into as small a task as we could. We attempted to do it as structured as we could with naming them after the week they came from and the part of the week, like "Week 9 1.1) . . . ". And with this structure we were able to keep up the workload in an okay timeframe, so now all the tasks are in 'Done'. But we were not perfect at assigning the task on the board.



Github Project board in the end of development

Like there can be seen on the screenshot, every task from 'Todo' and 'In Progress' are moved to 'Done'. But in week 13 we made a list of things (a brainstorm of sorts) so we could add the most interesting thing like 'Block user', 'liking cheep', 'Mobile friendliness', and 'comment on cheeps', ... But we did not add these things on the board, since we were unsure of how many extra features we could add in the allotted timeframe. So, we added our main feature, 'Block a user' and then if someone got the time more could be added.

3.2.1 Our team work

While we did initially start out as a team of five, during development the group did deteriorate a little bit. After a few weeks, Mathias dropped out of ITU, leaving us down to four people. When starting out, we gave Radmehr a decent bit of leeway in his productivity, however, during this entire semester, he has not contributed to the project nor report at all, except for a co-authored commit back on the 4th of September. This problem with Radmehr was brought up to our TA during the second session.

3.3 How to make Chirp! work locally

3.3.1 Web-app

There is a running version at this link

3.3.2 Locally

In order to run Chirp there are 2 options - Clone the repository - Run the release

3.3.2.1 Cloned Repository

To run this locally from a cloned repository, please do the following:

Make sure you have dotnet 8.0 installed; see download

Clone the repository with this git command:

```
git clone https://github.com/ITU-BDSA2024-GROUP20/Chirp.git
```

Then go to the Chirp.Web directory with:

```
cd .\src\Chirp.Web\
```

Now run these commands inside the directory

```
dotnet user-secrets set "authentication_github_clientId"
"Ov23liGbRbgORjmb9wUp"
```

```
dotnet user-secrets set "authentication_github_clientSecret"
"0293ae8fdb1f1b046f42ab98234b11469648708e"
```

You should now be able to run Chirp with:`dotnet run` and access it at http://localhost:5273 when it is running

3.3.2.2 Release

To run the release, first go to the main page of the Repository and click on the release section.

Find the latest version and download one of the following files depending on your operations system: - Chirp-Win.zip, for Windows users - Chirp-Mac.zip, for Mac users - Chirp-Linux.zip , for Linux users

When the file has been downloaded please unzip it. Then open a terminal and navigate to one of the following directories depending on your operations system: - Chirp-Win\artifact\win, for windows - Chirp-Mac\artifact\mac, for mac - Chirp-Linux\artifact\linux, for linux

Now run the following commands in the terminal:

```
dotnet dev-certs https -t
```

```
./Chirp.Web  --urls="http://localhost:5273"
--"authentication_github_clientId" "Ov23liGbRbgORjmb9wUp"
--"authentication_github_clientSecret" "0293ae8fdb1f1b046f42ab98234b11469648708e"
--development
```

When running the application, and it is done starting up, a popup will appear in your terminal indicating which port it is running on “http://localhost:5273”

3.4 How to run test suite locally

3.4.1 Unit tests

To run the unit test for this program you will need to open a terminal and navigate to the `Chirp.Razor.Test` directory which can be done from the root of the repository with:

```
cd .\test\Chirp.Razor.Test\
```

Now run `dotnet test`

3.4.2 Integration and end-to-end tests

Before running these tests, please delete the mychirp.db file in the `\src\Chirp.Web` directory and do the same when the tests are done running.

To run integration and end-to-end tests there are some prerequisites.

To install Playwright, navigate to the root of Chirp, then run:

```
cd .\test\Chirp.Web.Test\
```

Build the program with:

```
dotnet build
```

Then run

```
pwsh bin/Debug/net8.0/playwright.ps1 install
```

Followed by

```
npx playwright install --with-deps
```

When Playwright is installed, open a separate terminal and navigate to the Chirp repository root and then run the following command:

```
cd .\src\Chirp.Web
```

And then run:

```
dotnet run
```

When the program is running, go to the terminal that is in `.\test\Chirp.Web.Test` and run the test with

```
dotnet test
```

Now, as the test are running you should see a browser popup where things are happening. That is the tests running.

At some point, a GitHub window may appear where it asks you to authorize, please press the button, otherwise the test will fail.

3.4.3 Tests

The unit test we have go over all the different functions we have in the Author Repository and Cheep Repository checking if they are working as intended.

The integration tests go over all the different actions a user can take while on the web app and checks if the things that should show up when doing those actions are there.

The end-to-end test goes over all the different features in one go, like logging in, cheeping, following, blocking, and so on.

4 Ethics

4.1 License

We choose to use a MIT License since it allows others to download and change it however they want without restrictions.

4.2 LLMs, ChatGPT, CoPilot, and others

During this project, we agreed that we would limit ourselves to use LLMs as little as possible. We used LLMs like ChatGPT and Gemini when certain problems would arise that a normal search would not help. However, the code written by them were never useful except as a hint at what direction to go with the actual code. We always ended up finding the answer to our problem somewhere else. Therefore, we recommend not using LLMs for this type of project, as doing so slowed down development.