Chirp! Project Report

ITU BDSA 2023 Group 3

Albert Ross Johannessen alrj@itu.dk
Benjamin Juul Jensen bejj@itu.dk
Felix Anton Andersson fela@itu.dk
Marius Würtz Søgaard Nielsen mawn@itu.dk
Natalie Clara Petersen natp@itu.dk

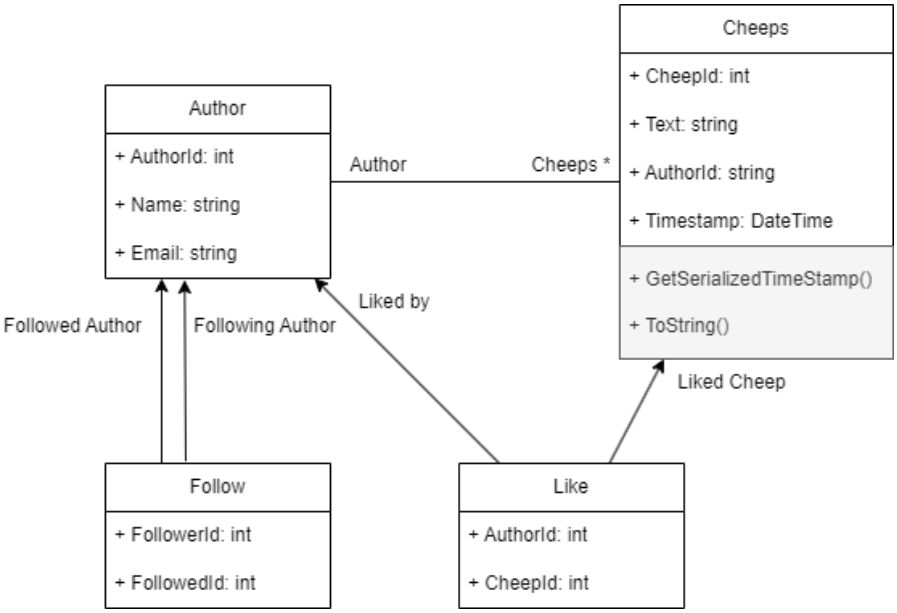
1 Introduction

This is a report for the course Analysis, Design and Software Architecture. This report will go into depth with our architectural decisions, technologies used, how our application works and how we worked together. This is part of the formal requirements of the course, and will function as the final hand-in.

2 Design and Architecture of *Chirp!*

2.1 Domain Model

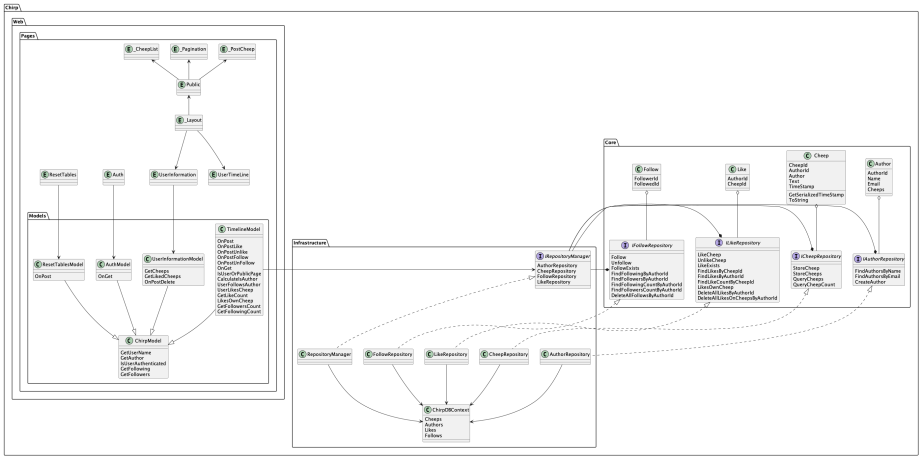
Our domain model consists of Entity classes for Authors, Cheeps, Follows, and Likes. These entities dictate the fundamental structure of our application and how we can model and interact with our database.



Note the absence of Date Transfer Objects (DTOs). Ideally, we would have implemented it, but since the scope of the project was not large enough to merit the conversion of Entities to DTOs, we decided to use the Entities directly instead.

2.2 UML Diagram

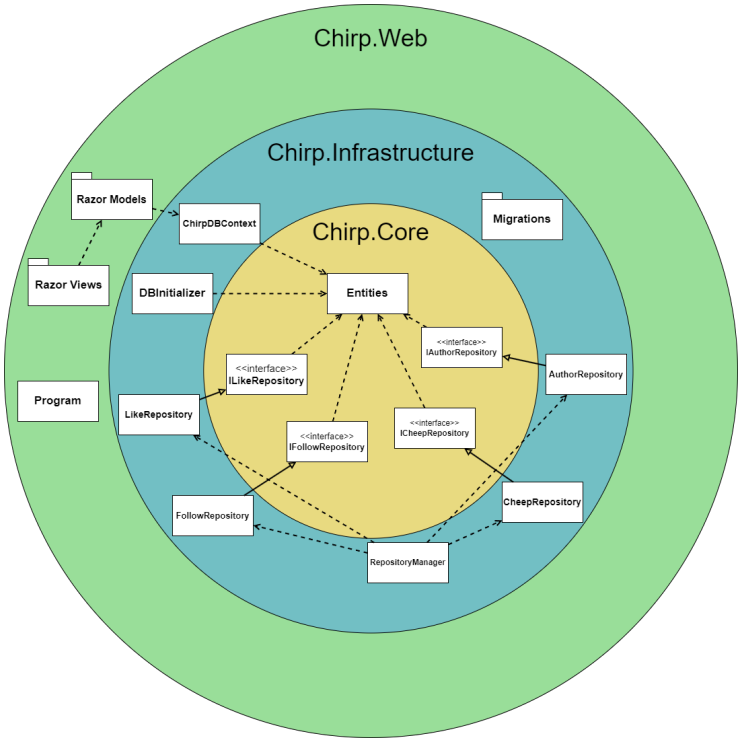
Below is a UML diagram of the entire application that shows the many relationships between projects, interfaces and classes in depth:



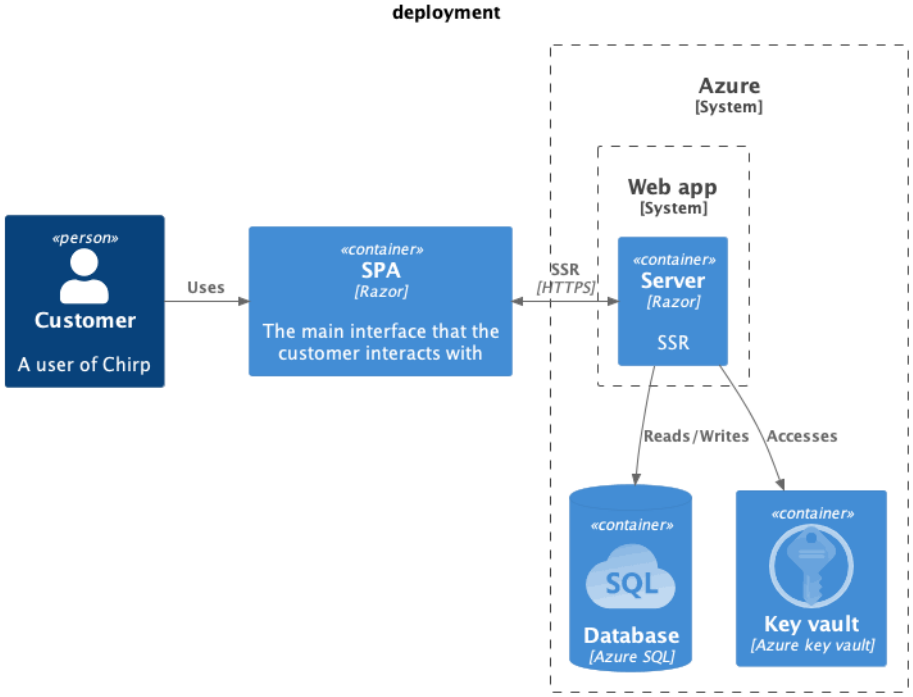
2.3 Architecture — In the small

Our code architecture is modeled after the Onion architectural pattern, where each package of our application corresponds to one 'layer' of the onion. Arrows never go from one layer to a layer further out, which is in line with the methodology of the pattern.

If there was a prospect of implementing additional features requiring complex queries, we would consider using service classes aswell to distinctly seperate the business logic from the Core and Infrastructure layers.



2.4 Architecture of deployed application



2.4.1 Razor

In this diagram we have taken into consideration that Razor utilizes server side rendering, which means that it generates JavaScript and HTML from the code written and sends that to the user. This means for our diagram that each user gets a page with some JavaScript, which can send HTTP requests to our server when the user want to interact with our application.

2.4.2 Database

The database consists of an Azure SQL database running MsSQL in the cloud. There are also other choices for a managed databases like Azure Cosmos DB, Firebase, Cloud SQL(google), Amazon RDS... and many more. Though there are problems with not using Azure SQL, which include: 1. Not all offer a relational model. 2. It is easier to not spread your service providers. 3. Azure offers free credits on a student subscription. 4. GDPR wise it is better to use Azure or AWS since they can guarantee data to be stored in EU. See Azure and AWS. 5. - It should be noted that industry standards *(Atleast in Denmark)* seem to tend towards Azure, since they are more trusted. ### Key Vault We needed a way to get the connection to our database and that can quickly become problematic since a connection string should **NOT** be placed in the codebase,

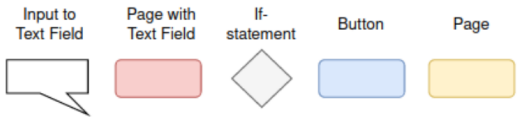
nor anywhere in the GitHub repository.

We decided to use an environment variable to contain the connection string. There are two possible issues with this: 1. Security - If a malicious actor gets access to the running application *(Which is running inside a container)* they would also be able to get into our database. 2. Internally vulnerable - Everyone who gets access to the web app also has the connection string.

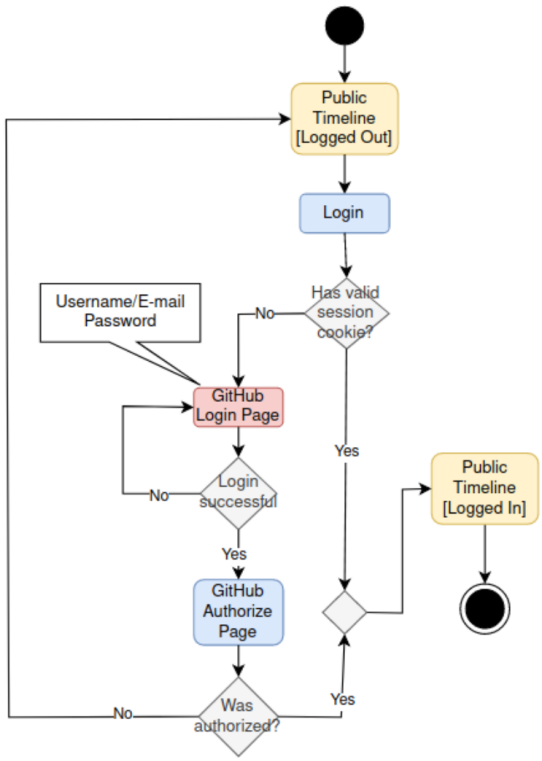
These are prevented when using an Azure Key Vault, since you can manage access more easily and safely.

2.5 User Activities

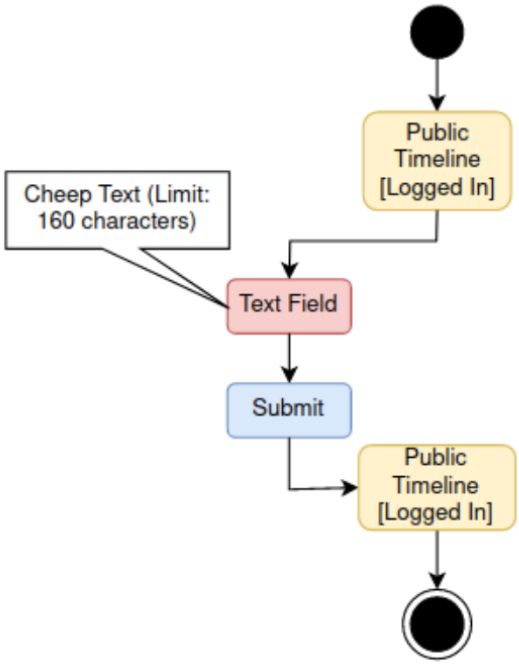
This section will describe the user journey through a couple of different common scenarios. That is, not all the functionality is described here, but the primary ones. The picture below shows the different legends which will be used in this section.



The first undertaking most users will take is logging in. This is done using GitHub OAuth. The below user activity diagram will visualize this process.

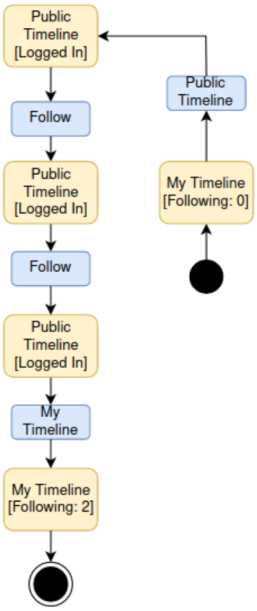


After authentication most of the functionality of the Cheep application is now accessible. This includes sending Cheeps, as can be seen below:

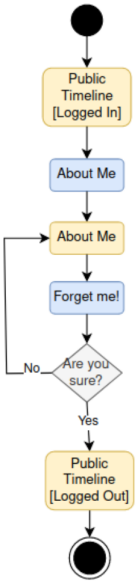


Here the state between the first `Public Timeline` and the last should be seen as different, as there will now be an additional Cheep in the last state.

It is also possible for a user to see how many people they are following, along with following other Cheepers. This can be seen in the activity diagram below:



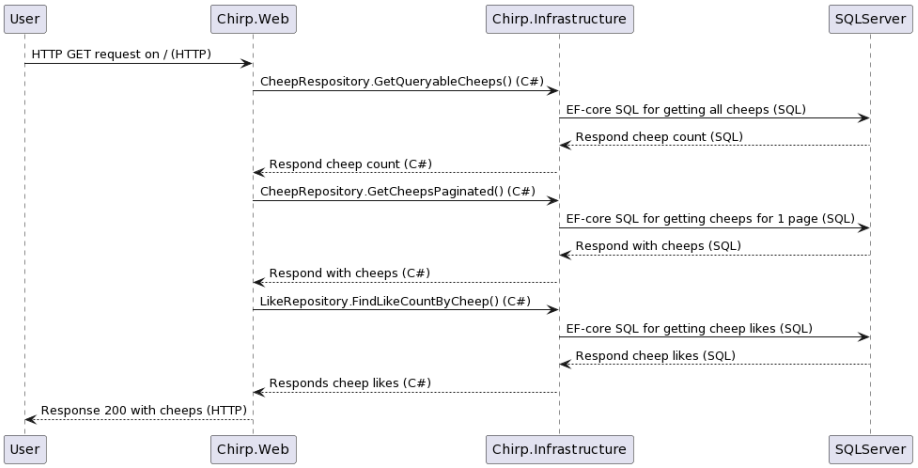
Finally if a user is tired of using the Cheep application, and wishes to delete their user, it is possible to do so using the `Forget me!` feature. This can be seen below:



It is worth noting that we have two additional features which only work when the application is running in a development instance; it is possible to login without Github OAuth using a Development Login, and it is possible to reset the database tables for more accurate end-to-end testing and debugging.

2.6 Sequence Diagram

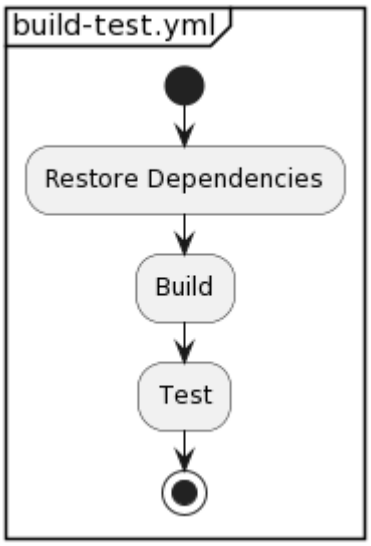
The sequence diagram below visualizes the flow of our application when an unauthenticated user sends a HTTP GET request to the root endpoint of our application (/) until the server responds with a fully rendered HTML page:



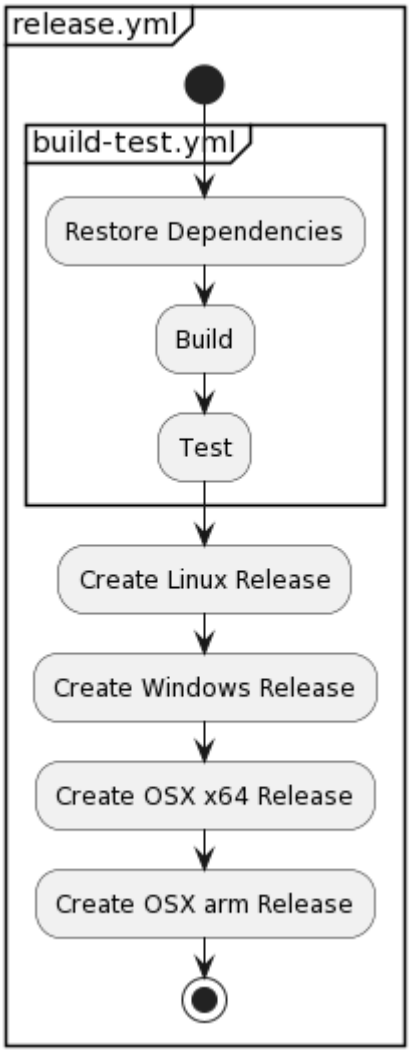
3 Process

3.1 Build, Test, Release, and Deployment

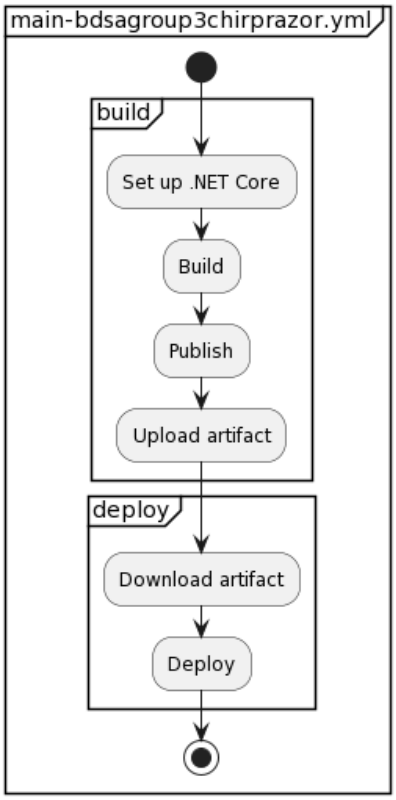
Throughout our development process we used different workflows to automatize certain repetitive tasks. This had different purposes, including application durability and accessibility. When our application was a simple CLI app, the need for automated testing to minimize the risk of bugs making it to the main branch arose. We achieved this using a simple Github Actions workflow with the following structure:



At this time, we also needed a way to release our application as separate executables for each major operating system. This was done by implementing a workflow that triggers only on tags being pushed using the standard versioning syntax. The workflow works by running the testing workflow first, then building the releases:



Finally, once we changed our app to be a razor web app, we needed automatic deployment on pushes to the main branch. The resulting workflow works by building and publishing the app which can then be deployed to our web app:

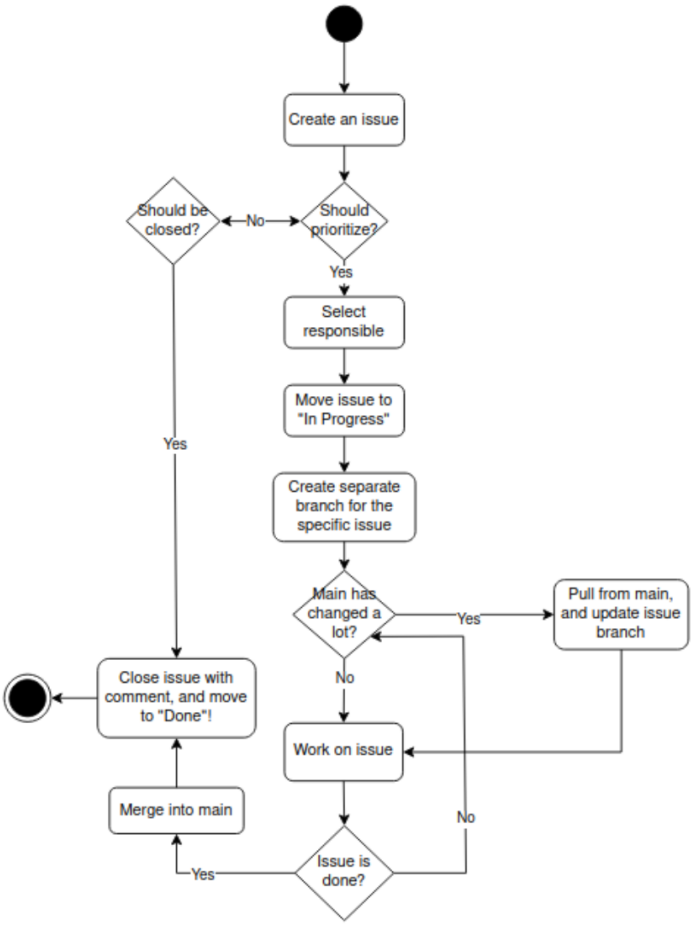


3.2 Team Work

When we started the project we agreed on a methodology for team work. We agreed to primarily meet on-site at the IT-University of Copenhagen, and have done so on every occasion with few exceptions.

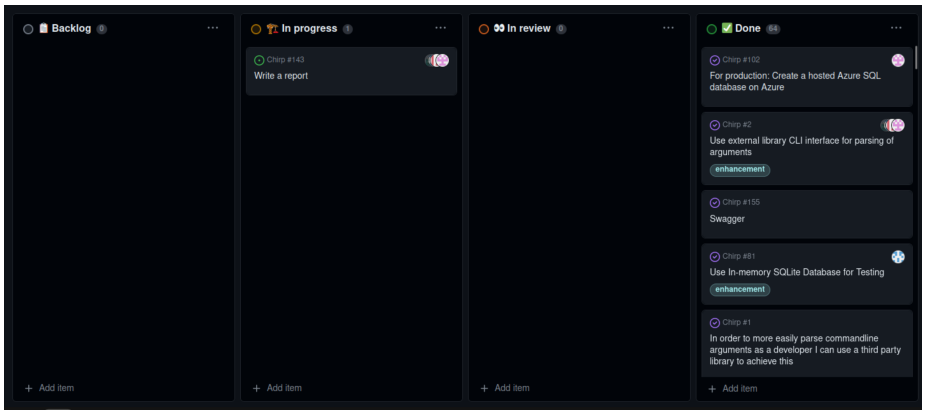
We also agreed that we would create pull requests, and try to always have atleast two other group members review them before merging to main. We tried to emphasize that it was important to give constructive feedback, as well as complimenting atleast one element of the submitted pull request, especially when it was not approved immediately.

We have made an activity diagram to visualize our team's workflow.



Additionally at the end of an issue, we did our best to remove branches, which we knew would not be used anymore.

This was the team's project board, right before hand-in:



3.3 How to make *Chirp!* work locally

3.3.1 Step One [Prerequisites]

First install `Git` and clone this repository

```
git clone https://github.com/ITU-BDSA23-GROUP3/Chirp
```

[Note: If you are on Windows we recommend using WSL]

3.3.2 Step Two [Dependencies]

To run the application you need the `dotnet-runtime` package, `dotnet-sdk` and `aspnet-runtime`.
[Note: Remember to install `Dotnet 7` and not a different version]

On `Debian`-based systems this can be done `apt`:

```
sudo apt install dotnet-sdk-7.0
```

You can also just download it via the website.

You also need Entity Framework Core:

```
dotnet tool install --global dotnet-ef --version 7.0.14
```

You might need to add dotnet tools to your PATH. This can be done in bash by adding this line to the bottom of your `~/.bashrc`.

```
export PATH="$PATH:$HOME/.dotnet/tools/"
```

Afterwards restart your terminal.

3.3.3 Step Three [Github OAuth]

Now we will create a Github OAuth app

Click `New OAuth app` and fill in the details. The homepage URL should be: `http://localhost:1339` and the callback URL should be `http://localhost:1339/signin-github`.

You should now take note of the `client ID`, you will need this in a second. This can be found on the apps page.

Now you need to generate a secret, take note/copy of the `client secret` as well, this is done on the same page.

We need this such that we can set our Development Secrets.

3.3.4 Step Four [Secrets]

```
dotnet user-secrets --project src/Chirp.Web init
```

Set secrets with these names

```
dotnet user-secrets --project src/Chirp.Web \
    set "development:authentication:github:clientId" "<client id>"
dotnet user-secrets --project src/Chirp.Web \
    set "development:authentication:github:clientSecret" "<secret id>"
```

3.3.5 Step Five [Running it locally!]

[Note: Remember to replace and with the respective values from the prior section]

We have set it up to work with Docker which can be downloaded here.

If you are using a windows machine you can also choose to use SQL server, though you are going to have to change the connection string in `src/Chirp.Web/appsettings.Development.json`.

If you still want to use Docker you have to do the following:

Make sure you have installed `Docker`, opened Docker desktop, and run

```
sudo docker pull mcr.microsoft.com/mssql/server:latest
```

Now you are almost ready! You can run the MsSQL, using the following command:

```
sudo docker run \
    -e "ACCEPT_EULA=Y" \
    -e "MSSQL_SA_PASSWORD=Adsa2023" \
    -p 1433:1433 \
    --name sqlpreview \
    --hostname sqlpreview \
    -d mcr.microsoft.com/mssql/server:2022-latest
```

[Note: We recommend modifying this default password]

Make sure that you are in the root directory of Chirp and run

```
dotnet ef migrations add InitialMigrations \
    --project src/Chirp.Infrastructure/ \
    --startup-project src/Chirp.Web/
```

Now you can update the database and run the application:

```
dotnet ef database update --project src/Chirp.Web
dotnet run --launch-profile Localhost --project src/Chirp.Web
```

[Note: If you change the database structure, you might want to remove the database container, and follow the steps from step five again]

3.4 How to run test suite locally

You should be able to run all tests except the E2E-tests following the guide above. This section is specifying how to run our E2E-tests.

We are using Selenium Grid to automate our testing. This makes headless end-to-end testing really easy across platforms. Ensure that you have `docker` and run:

```
sudo docker run --net="host" -d -p 4444:4444 \
    -v /dev/shm:/dev/shm selenium/standalone-chrome
```

[Note: This might not work on non-UNIX compliant systems]

Ensure that the application is running by doing:

```
dotnet run --launch-profile Localhost
```

Now we can run

```
dotnet test
```

It is possible to utilize Virtual Network Computing (VNC) to see the tests running. These will by default be running on `http://localhost:7900/` using noVNC with the default password `secret`.

4 Ethics

4.1 GDPR and Privacy

To meet GDPR laws, we have added a feature that lets users completely remove their accounts from our app. This deletes all their personal data. We also follow the 'Privacy By Default' principle, meaning we only keep the user data we need and do not store anything redundant.

4.2 License

The MIT License was selected because it is easy to understand and allows a lot of freedom. It lets people use, change, and share software freely, asking only to keep the original copyright notice. This approach helps more people use and contribute to the software, and avoids complicated legal rules, making it great for open-source projects such as our Cheep project.

4.3 LLMs, ChatGPT, CoPilot, and others

Some members of our group have used ChatGPT and CoPilot for writing small sections of code. These tools have been helpful for quick coding tasks or solving specific simple C# problems. Our primary use case has been writing code faster when we already knew what to write. For that reason, the code generated by these AI tools has not been included in our commits as `Co-authored-by:`, as we believed our prompts to be so specific, that credit can not be given to ChatGPT or Copilot. However, when discussing this in hindsight, we agreed that we should probably have included it regardless, but upon inspection of the code, we found it difficult to determine when such tools had been used.