Project *Chirp!*

BDSA Group 17

Burak Özdemir buoe@itu.dk
Hanan Choudhary Hadayat abha@itu.dk
Joshua James Medilo Calba jcal@itu.dk
Julius Dalsgaard Bertelsen jube@itu.dk
Tan Dang tanda@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	17
Date	December 21 2023
School	IT-University of Copenhagen

0.1 Table of contents:

1 Design and architecture
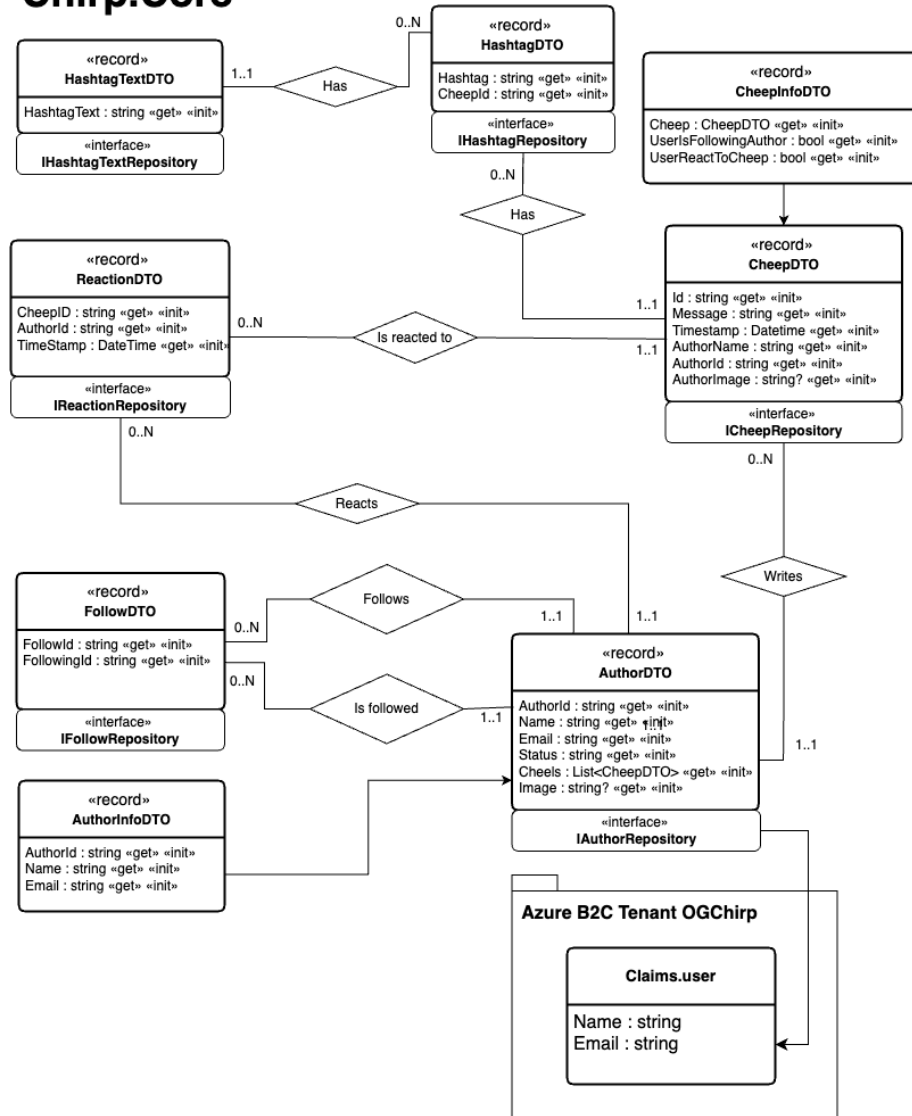
1.1 Domain model

The domain model reflects the entities and their relationship of the Chirp application, as well as the interfaces of the repositories that allow for accessing and manipulating these different entities and their related data. These entities form the foundation of the business logic of the application.

The diagram below shows the domain model of the Chirp application.

Chirp.Core

«record» HashtagTextDTO

HashtagText : string «get» «init»

«interface» IHashtagTextRepository

1..1 — Has — 0..N

«record» HashtagDTO

Hashtag : string «get» «init»
CheepId : string «get» «init»

«interface» IHashtagRepository

0..N — Has

«record» CheepInfoDTO

Cheep : CheepDTO «get» «init»
UserIsFollowingAuthor : bool «get» «init»
UserReactToCheep : bool «get» «init»

«record» ReactionDTO

CheepID : string «get» «init»
AuthorId : string «get» «init»
TimeStamp : DateTime «get» «init»

«interface» IReactionRepository

0..N — Is reacted to — 1..1 / 1..1

«record» CheepDTO

Id : string «get» «init»
Message : string «get» «init»
Timestamp : Datetime «get» «init»
AuthorName : string «get» «init»
AuthorId : string «get» «init»
AuthorImage : string? «get» «init»

«interface» ICheepRepository

0..N — Reacts

0..N — Writes — 1..1

«record» FollowDTO

FollowId : string «get» «init»
FollowingId : string «get» «init»

«interface» IFollowRepository

0..N — Follows — 1..1
0..N — Is followed — 1..1

«record» AuthorDTO

AuthorId : string «get» «init»
Name : string «get» «init»
Email : string «get» «init»
Status : string «get» «init»
Cheels : List<CheepDTO> «get» «init»
Image : string? «get» «init»

«interface» IAuthorRepository

«record» AuthorInfoDTO

AuthorId : string «get» «init»
Name : string «get» «init»
Email : string «get» «init»

1..1

Azure B2C Tenant OGChirp

Claims.user
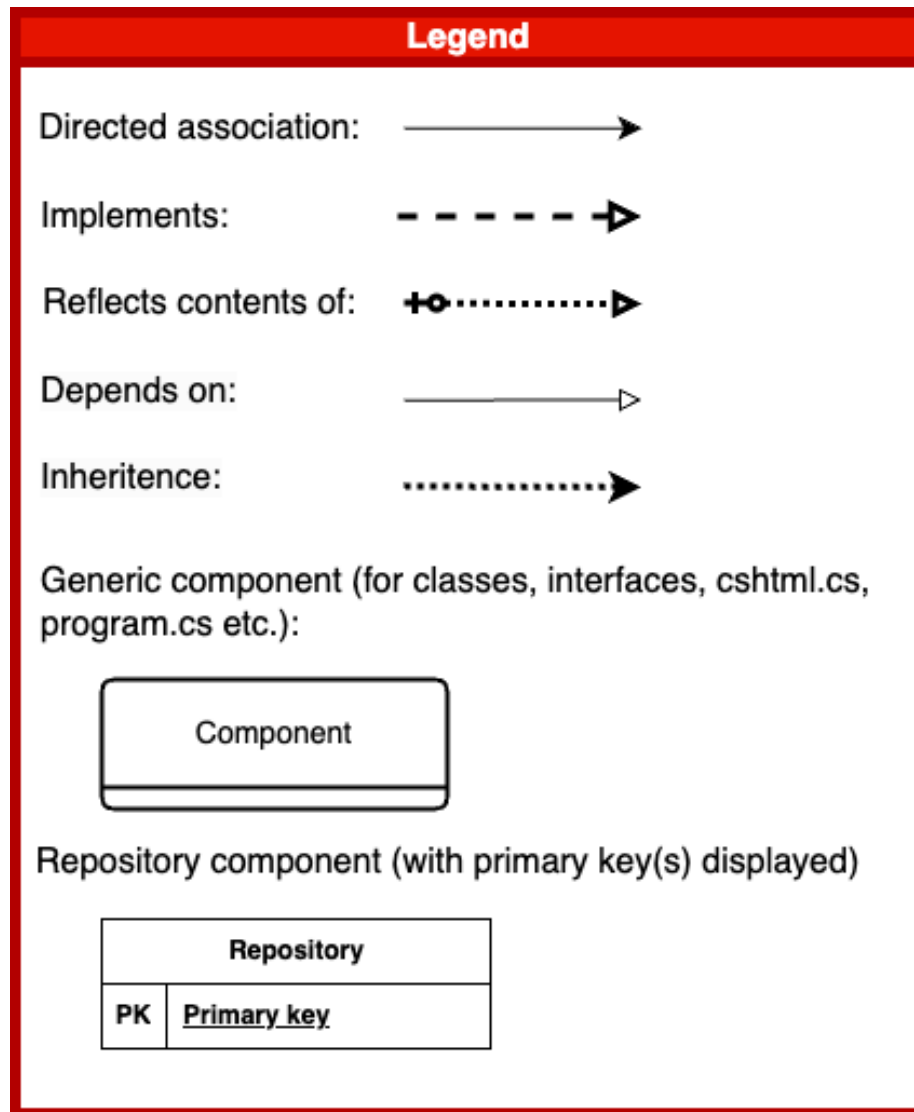
Name : string
Email : string
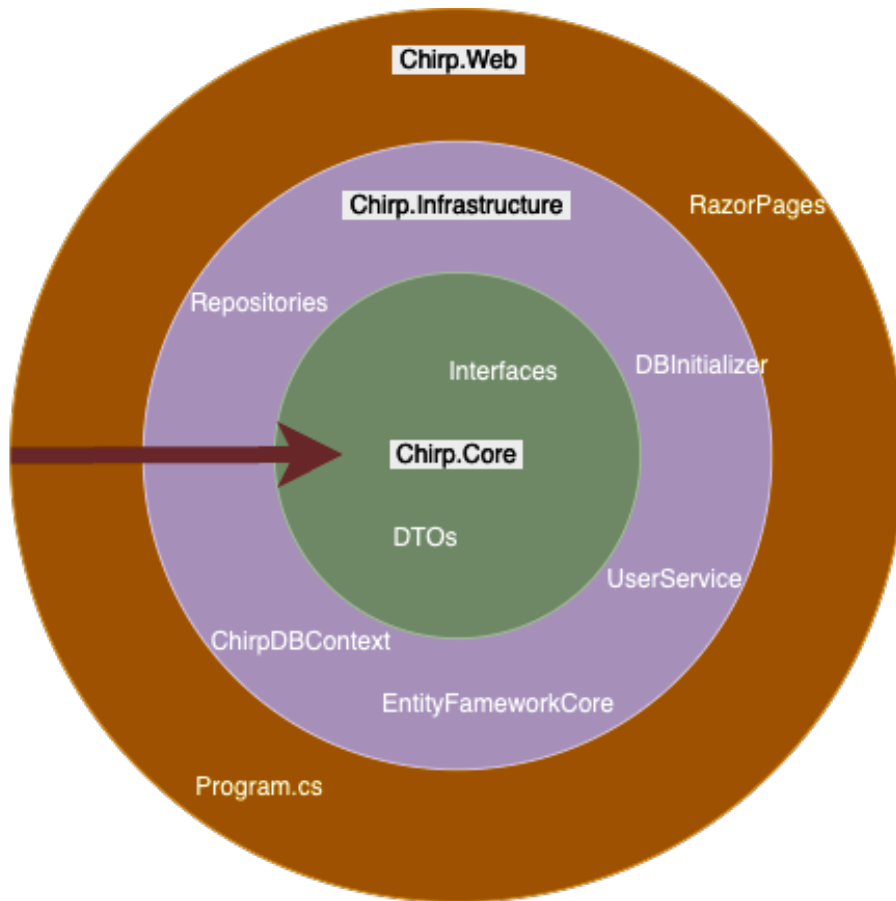
1.2 Architecture — In the small

The application Chirp has been designed with Onion architecture in mind.

The diagram below shows the organization of the code of the Chirp application, showcasing the relationships and dependencies of different components of the program, highlighting how the code of the program is organised into different layers of the onion architecture design.

The diagram shows the inward flow of dependencies in accordance with Onion architecture, so that the inner layers remain independent of external dependencies.

Some relationships have been omitted to improve the readability of the diagram.

Legend

Directed association:

Implements:

Reflects contents of:

Depends on:

Inheritence:

Generic component (for classes, interfaces, cshtml.cs, program.cs etc.):

Component

Repository component (with primary key(s) displayed)

Repository	
PK	Primary key

The figure below shows a more simplistic view of the onion architecture structure of the components of the application.

## 1.3 Architecture of deployed application

The Chirp application is deployed on Microsoft Azure, utilizing Azure services with an Azure SQL database.

The diagram below shows the interaction between the client component with a user interface allowing the user to make requests and the application deployed on Microsoft Azure as the server component. The diagram also shows the interaction of this deployed application with the Azure SQL Database, as well as signing in through a social account (GitHub).
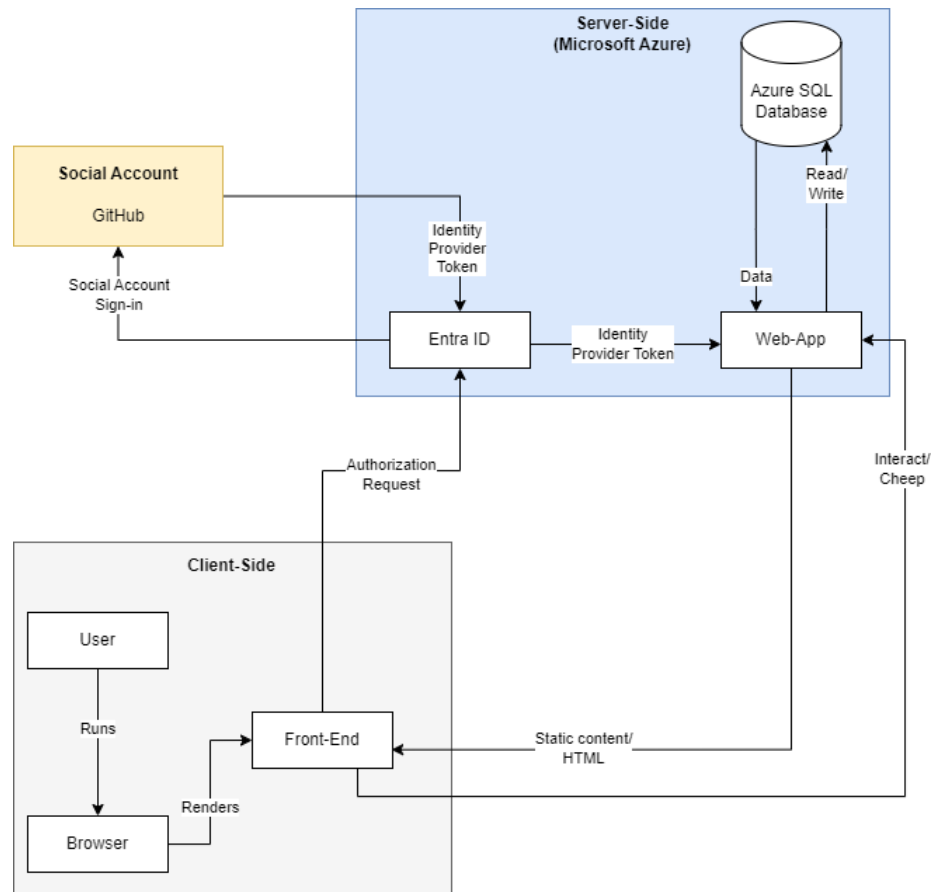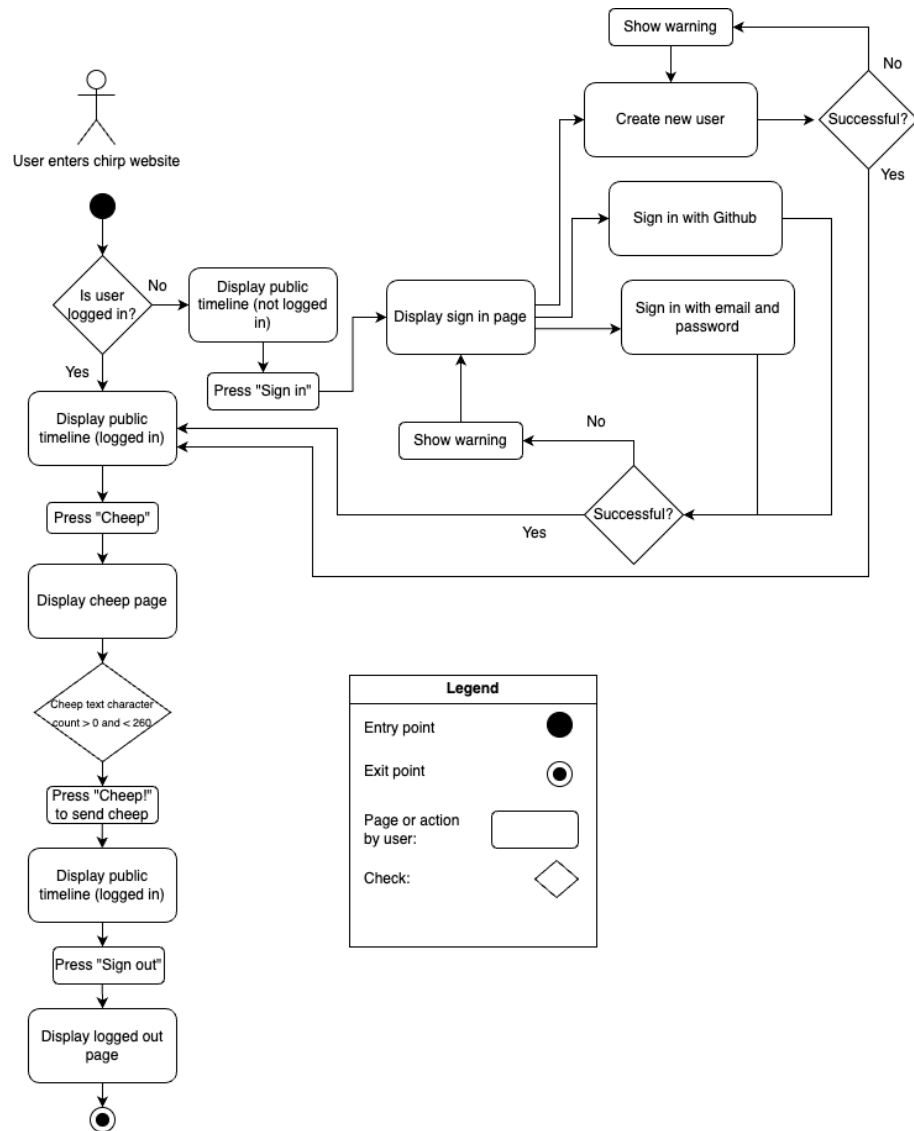
1.4 User activities

The diagram below shows a series of typical user activities through the Chirp application while remaining unauthorized and after logging in and becoming authorized.
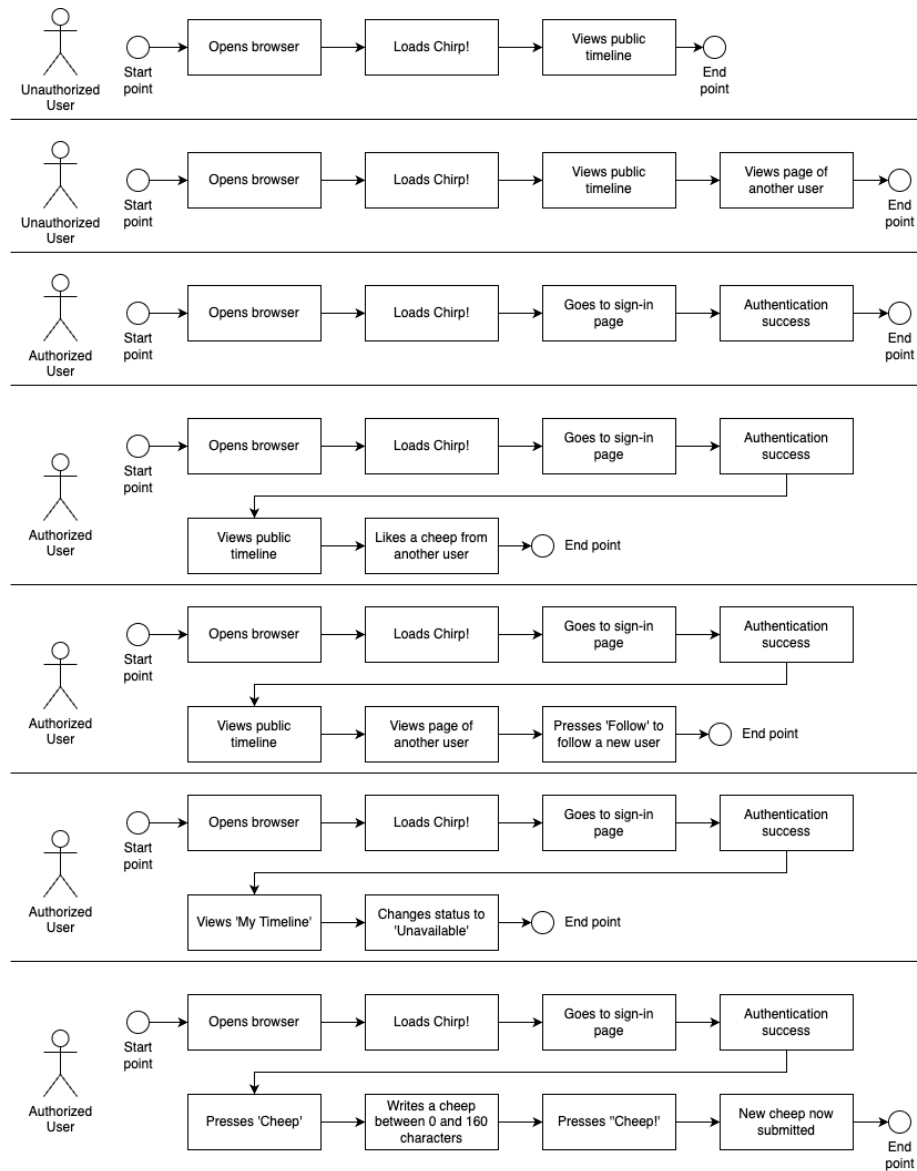
The diagram below shows a slightly more detailed view of possible scenarios of a user journey, in which a user enters the chirp website, logs in or creates a profile if necessary, sends a cheep, and then logs out.

User enters chirp website

Show warning

Is user logged in?

No → Display public timeline (not logged in)

Yes

Create new user

Successful?

No

Yes

Display public timeline (logged in)

Press "Sign in"

Display sign in page

Sign in with Github

Sign in with email and password

Show warning

No

Successful?

Yes

Press "Cheep"

Display cheep page

Cheep text character count > 0 and < 260

Press "Cheep!" to send cheep

Display public timeline (logged in)

Press "Sign out"

Display logged out page

Legend	
Entry point	●
Exit point	◉
Page or action by user:	
Check:	◇

9

1.5   Sequence of functionality/calls trough *Chirp!*

The diagrams below illustrates the flow of messages and data through the Chirp
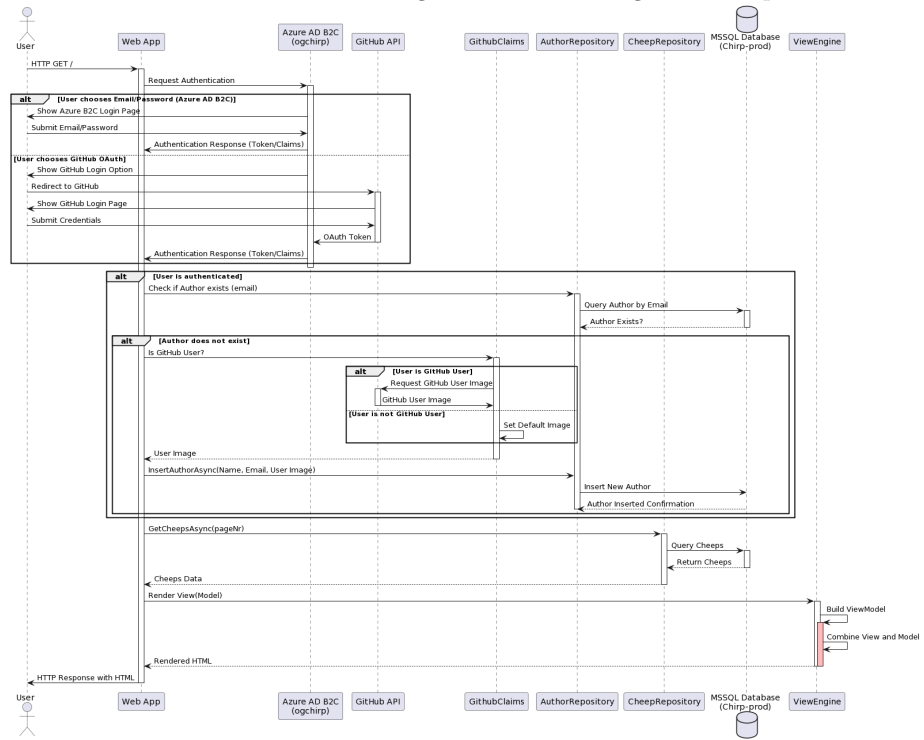
application.
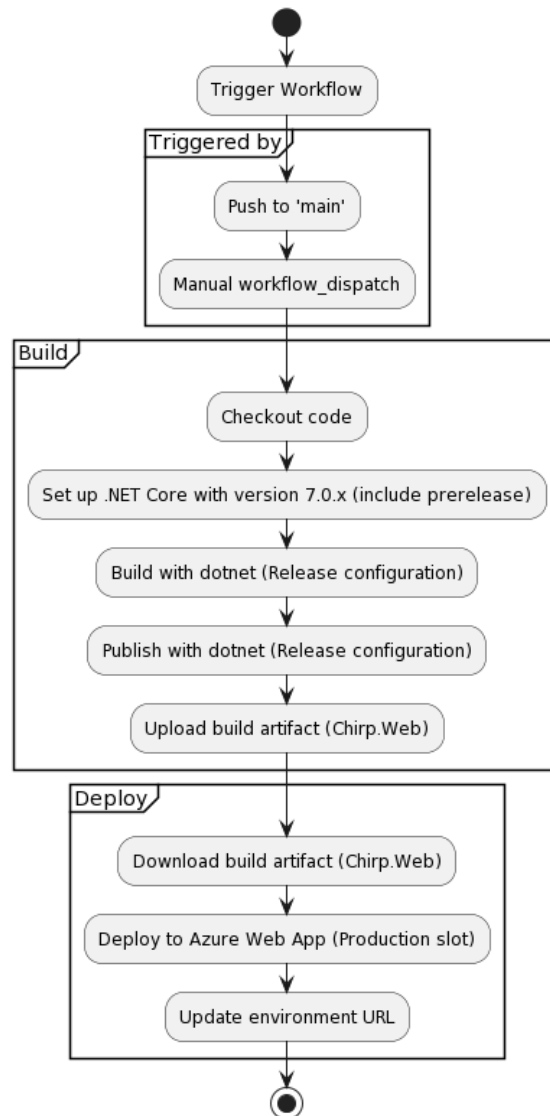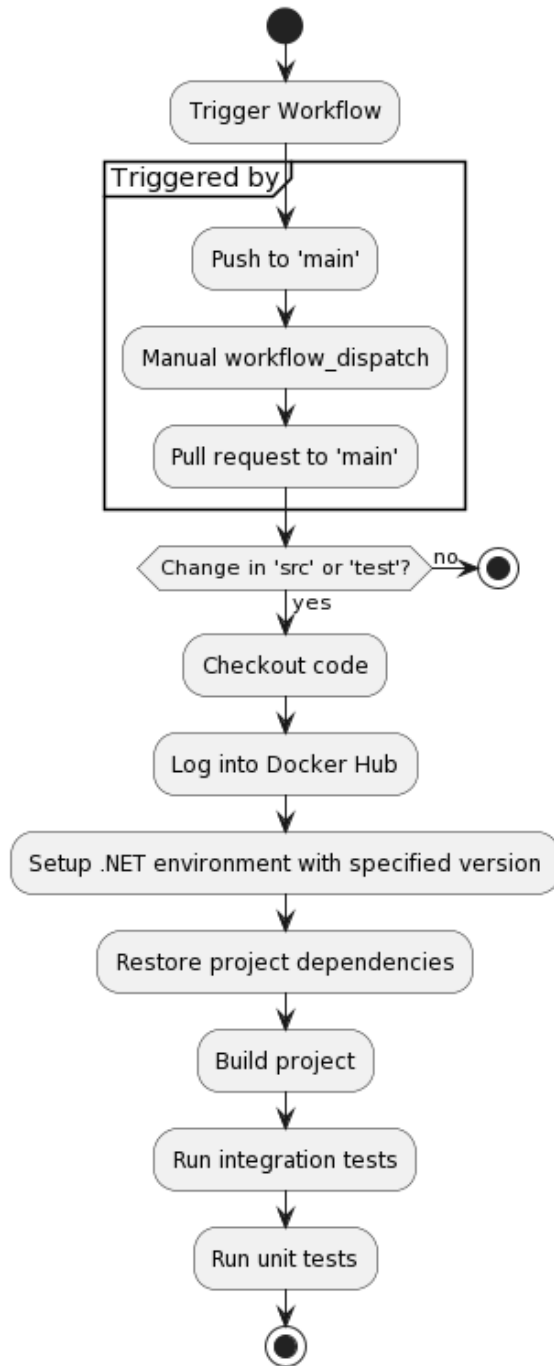
2 Process

2.1 Build, test, release, and deployment

This workflow is the deploy workflow for azure

.NET Build and Test Workflow

Trigger Workflow

Triggered by

Push to 'main'

Manual workflow_dispatch

Pull request to 'main'

Change in 'src' or 'test'? — no

yes

Checkout code

Log into Docker Hub

Setup .NET environment with specified version

Restore project dependencies

Build project

Run integration tests

Run unit tests

This workflow is Build and Test workflow

Workflow to Publish Release Artifacts for Linux, Windows, and macOS
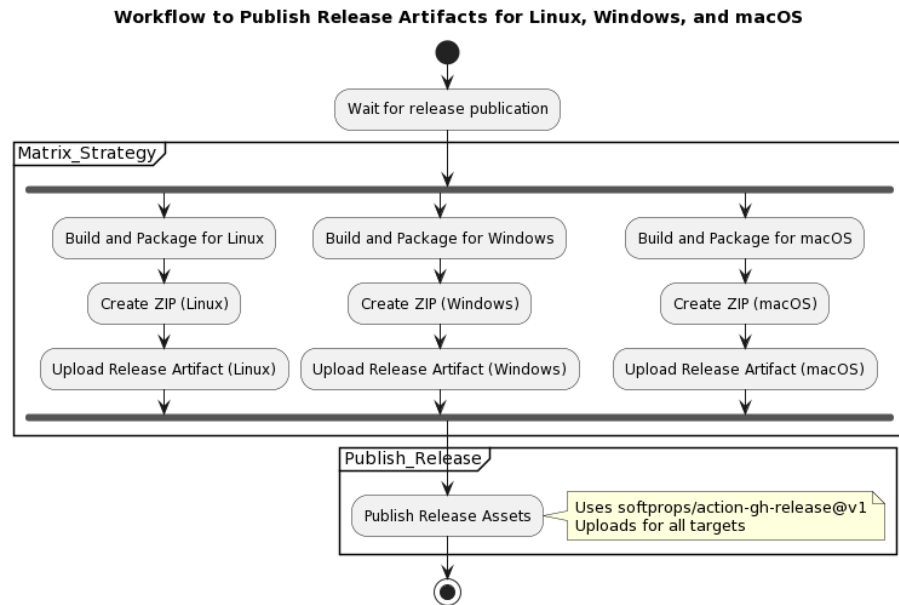
This is release workflow

2.2 Team work

Overall, we managed to complete all the features we wanted for the application. These include all the features specified by the requirements of the project and some extra features, such as some UI changes and hashtags.

We have one unresolved task in our project board: To make the email unique in the the Authors table, since there was a possibility that an Author could appear twice or more in the table with same name and email but with different id. The reason we did not resolve it, is because of the low priority.

The image below shows the project board just before hand-in, with the remaining unresolved issue.

13
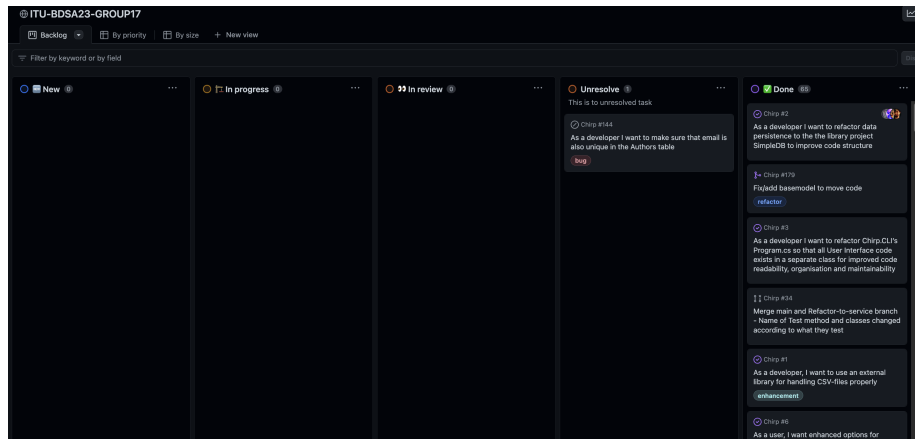
There are of course many more features we could have implemented given time. Some ideas, which were discussed during development but not prioritized include the ability to comment on a cheep and sharing a cheep to name a few.

The diagram below shows the lifecycle of a GitHub issue from it's creating until it is closed and resolved.
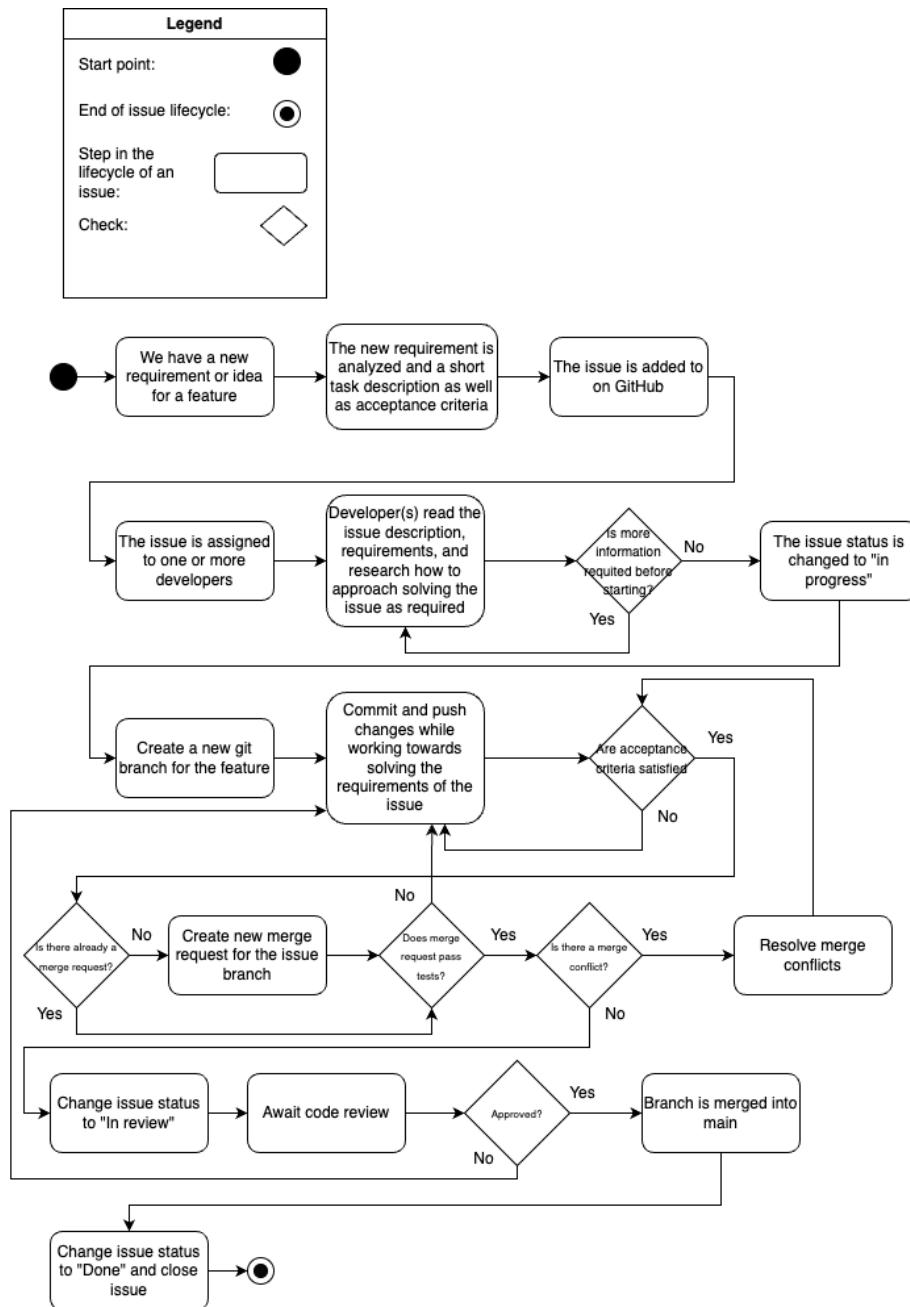
2.3 How to make *Chirp!* work locally

In order to run the application locally, you can either

1. Clone this repository
2. Run the release version

2.3.0.1 Pre requirements You need to have dotnet 7.0 installed see download

You need to have setup AzureB2C Tenant see the guide

You have docker installed see docker

2.3.0.1.1 Optional pre requirements Tenant read and write permission for API Guide

2.3.1 Clone the repository

In order to run the application locally by cloning the repository, please do as follows:

Clone the repository using this git command:

`git clone https://github.com/ITU-BDSA23-GROUP17/Chirp.git`

Change directory into

`cd src/Chirp.Web`

Inside the directory, run one of the following commands:

`dotnet watch --clientsecret [your-secret]`

`dotnet run --clientsecret [your-secret]`

You should now have access to a localhost with a specific port, in which this web-app can be accessed

2.3.2 Run the release

In order to run the release versions, please do as follows:

On the main page of this repository, click on the **_Releases_** section There will be a few assets available (including source code), but only one of the following three will be relevant for us:

- Chirp-win-x64.zip, for Windows users
- Chirp-osx-x64.zip, for Mac users
- Chirp-linux-x64.zip, for Linux users

Please install and unzip one of the three folders, depending on your operating system Now, there should be the following application available in the extracted folder:

- Chirp.Web.exe for Windows users
- Chirp.Web for Mac and Linux users

16

Now, you have a runnable (as described in step 4). Depending on your operating system, you can run the web-app as follows:

Run the following commands:

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

```
./Chirp.Web --urls="https://localhost:7102;http://localhost:5273"
--clientsecret [your-secret]
```

Upon running the application, a terminal will pop up, indicating in which port (in the localhost) the web-app is up and running

2.4 How to run test suite locally

Ensure that Docker is running on your machine, and in the root folder, run this command to test all the tests.

```
dotnet test
```

The following test have been implemented

2.4.1 Unit test

The unit tests are designed to test each individual component of our application by itself.

We have designed a series of unit tests to verify that our DTOs correctly encapsulate data. These tests confirm that each DTO retains and accurately represents the data passed to its constructor.

To run only the unit tests, use the following command in the root folder of the project:

```
dotnet test --filter Category=Unit
```

2.4.2 Integration test

The integration tests are designed to test how different parts of the application interacts with eachother. These tests involves instances of the database containers and checks if the application does the CRUD operations as expected.

The integration tests test that the repositories classes are able to correctly recieve and modify the relevant data in the database.

Before running integration tests, make sure Docker is running as the tests rely on `Testcontainers.MsSql` to create a containerized MS SQL Server instance.

To run only the integration tests, use the following command in the root folder of the project:

```
dotnet test --filter Category=Integration
```

2.4.3 End to end test

The playwright can be going into the folder in which the test is saved:

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

And then run the build command

```
dotnet build
```

After this you need to install the browser:

```
pwsh bin/Debug/net7.0/playwright.ps1 install
```

If you are on Linux or do not have Powershell you can use https://nodejs.org/en

Refer to the given link for installation guide https://docs.npmjs.com/downloading-and-installing-node-js-and-npm

Then run

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

Then in the project you can run:

```
dotnet test
```

Which should start the test

3 Ethics

3.1 License

We chose to use the MIT license for our Chirp application, since it allows other developers to distribute, use and copy our software without imposing significant restrictions.

3.2 LLMs, ChatGPT, CoPilot, and others

On one hand, using LLMs, particularly ChatGPT for was at times helpful for gaining a basic understanding of frameworks like Entity Framework core, Docker, and Onion architecture through questioning. On the other hand, relying on LLMs for this kind of information and understanding quickly proved disadvantageous in other ways, since the accuracy of the responses is always uncertain and should be approached with a level of skepticism, often necessitating validation from more reliable sources.

The use of LLMs in generating or helping with the writing of code has also been only partially helpful. Sometimes the work with debugging code which relied on a lot of help from LLMs ended up being more work than just researching and properly understanding the problem ourselves. ChatGPT was mostly for explaining errors or explaining the code, and did prove helpful in the debugging

process in this regard. GitHub co-pilot was used in a limited extent when writing simple repeating and predictable code, e.g. when we write insert methods in to our database in `DbInitializer.cs`.

Ultimately, LLMs are very capable and have been useful but their utility is also limited. Overall they are just another addition for a developers toolbox