Chirp! Project Report

1 *Chirp!* Project Report

1.1 ITU BDSA 2023 Group 7

1.1.1 Course title: Analysis, Design and Software Architecture

1.1.2 Course code: BSANDSA1KU

1.1.3 Authors

Casper Storm Frøding - csfr@itu.dk

Daniel Fich - dafi@itu.dk

Line Juul Kabbeltved Præstegaard - ljni@itu.dk

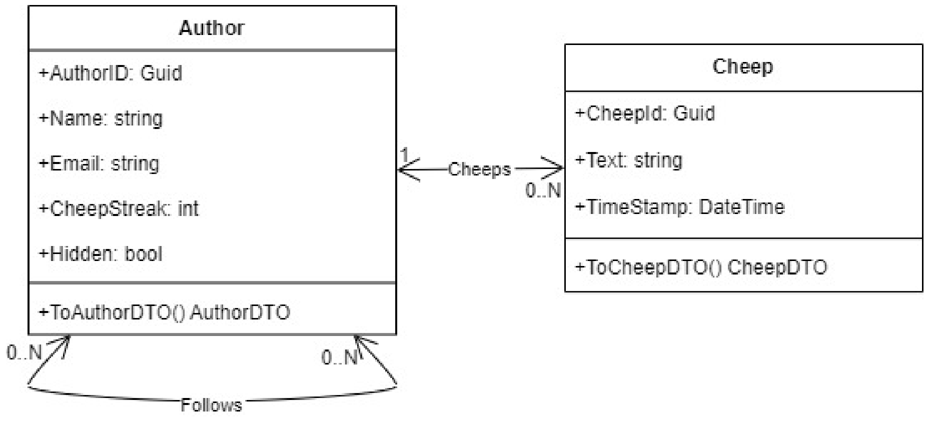
Max Brix Koch - mbko@itu.dk

Sebastian Cloos Hylander - sehy@itu.dk

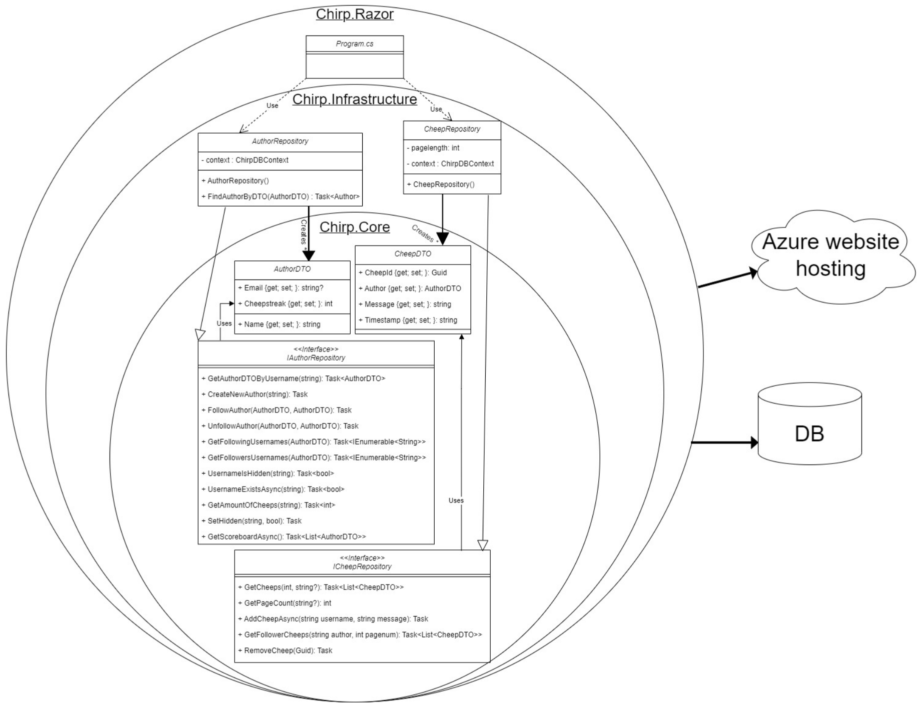
2 Design and Architecture of Chirp!

2.1 Domain model

Our domain model consists of the `Author`- and `Cheep` class. Each author can have multiple cheeps and each cheep has a reference to one author. Each author can follow and be followed by multiple other authors.



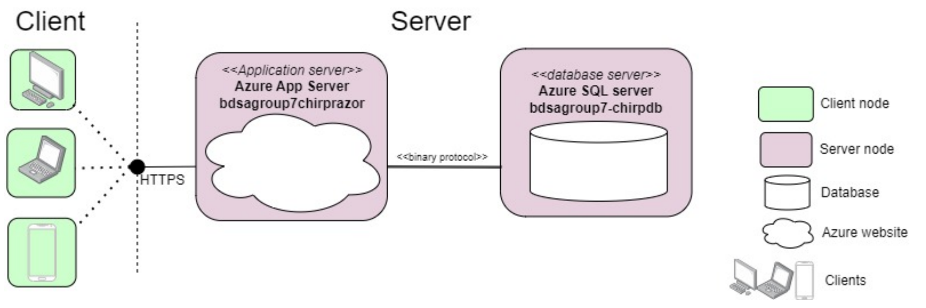
2.2 Architecture — In the small


Chirp.Razor
Program.cs
Use
Use
Chirp.Infrastructure
AuthorRepository
- context : ChirpDBContext
+ AuthorRepository()
+ FindAuthorByDTO(AuthorDTO) : Task<Author>
CheepRepository
- pagelength: int
- context : ChirpDBContext
+ CheepRepository()
Creates *
Creates *
Chirp.Core
AuthorDTO
+ Email {get; set; }: string?
+ Cheepstreak {get; set; }: int
+ Name {get; set; }: string
CheepDTO
+ CheepId {get; set; }: Guid
+ Author {get; set; }: AuthorDTO
+ Message {get; set; }: string
+ Timestamp {get; set; }: string
Uses
<<Interface>>
IAuthorRepository
+ GetAuthorDTOByUsername(string): Task<AuthorDTO>
+ CreateNewAuthor(string): Task
+ FollowAuthor(AuthorDTO, AuthorDTO): Task
+ UnfollowAuthor(AuthorDTO, AuthorDTO): Task
+ GetFollowingUsernames(AuthorDTO): Task<IEnumerable<String>>
+ GetFollowersUsernames(AuthorDTO): Task<IEnumerable<String>>
+ UsernameIsHidden(string): Task<bool>
+ UsernameExistsAsync(string): Task<bool>
+ GetAmountOfCheeps(string): Task<int>
+ SetHidden(string, bool): Task
+ GetScoreboardAsync(): Task<List<AuthorDTO>>
Uses
<<Interface>>
ICheepRepository
+ GetCheeps(int, string?): Task<List<CheepDTO>>
+ GetPageCount(string?): int
+ AddCheepAsync(string username, string message): Task
+ GetFollowerCheeps(string author, int pagenum): Task<List<CheepDTO>>
+ RemoveCheep(Guid): Task
Azure website hosting
DB

Here we have chosen some relevant classes, to show our onion architecture. The most important part to notice here are that dependencies flow inward, ensuring that the inner layers remain independent of the outer layers. This Makes sure that we try to hold the coupling low between the layers, and makes it more adaptable to change, because we make sure that the outer layers can change without affecting the inner layers. From the outer layer, the 'program.cs' class can create the razor pages, send it to azure, and also have access to the database.

2.3 Architecture of deployed application

The system has a Client Server architecture, where the web server and web database are hosted via Azure.

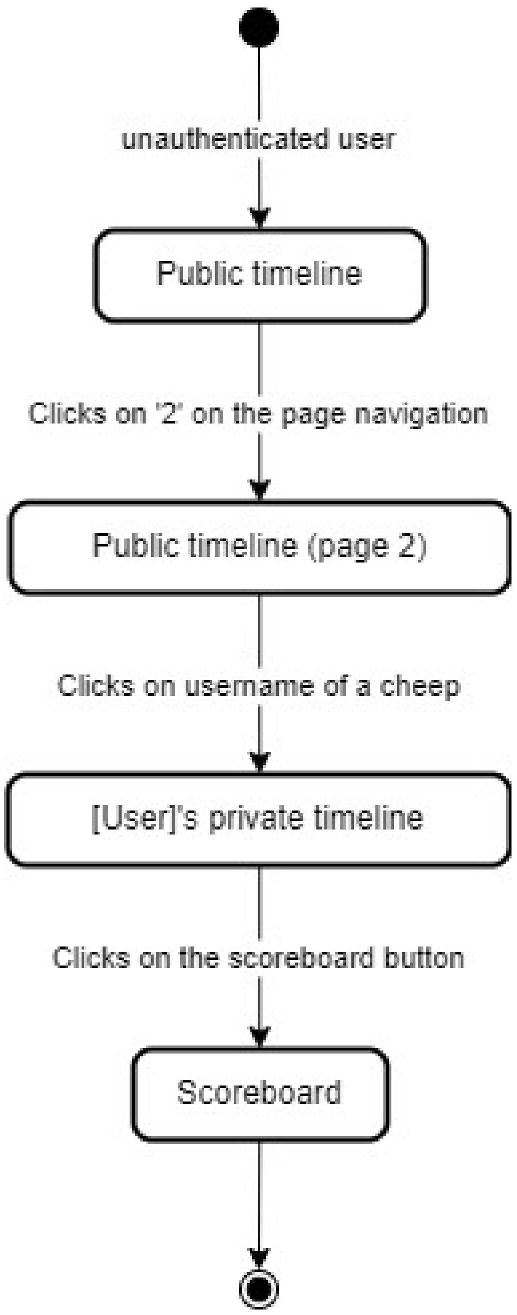


It was chosen to show that multiple clients can access the web application server at the same time.

2.4 User activities

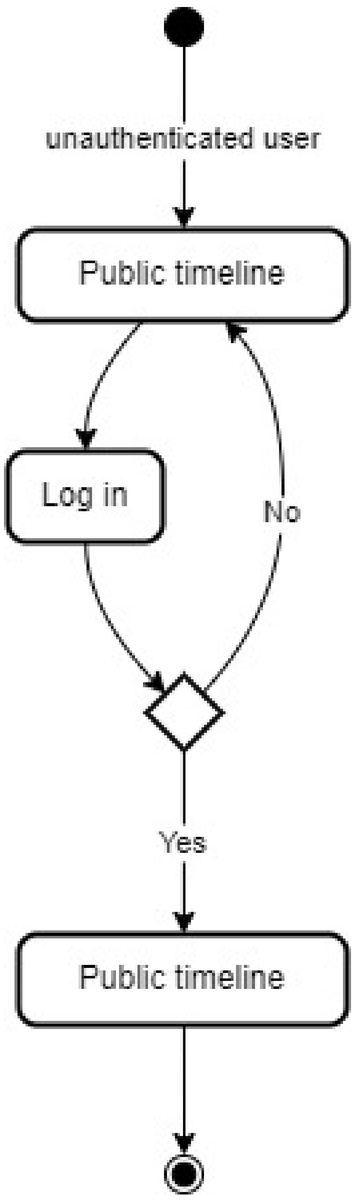
2.4.1 Unauthenticated user

The following activity diagram shows a typical user journey for an unauthenticated user.


unauthenticated user
Public timeline
Clicks on '2' on the page navigation
Public timeline (page 2)
Clicks on username of a cheep
[User]'s private timeline
Clicks on the scoreboard button
Scoreboard

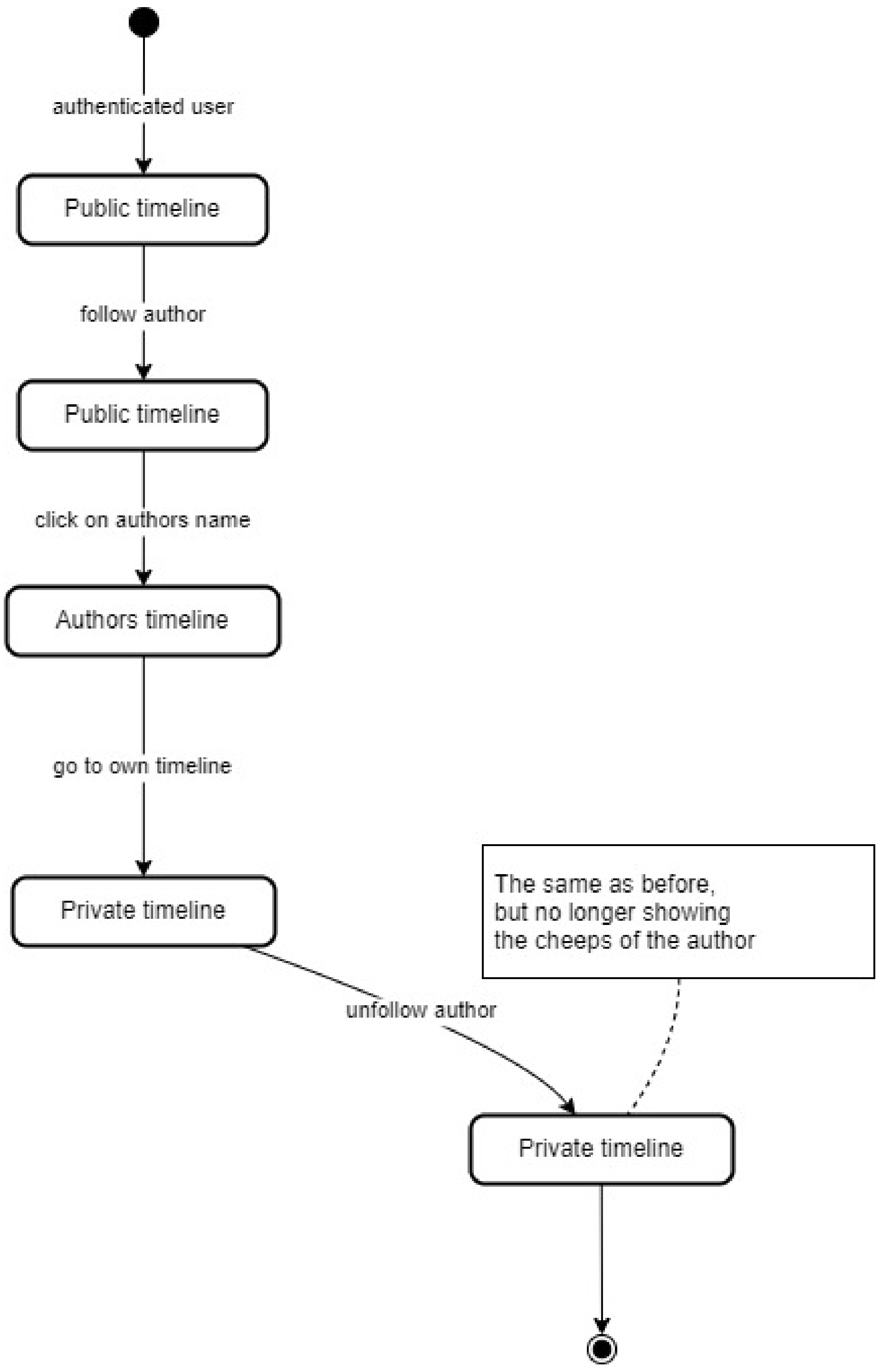
2.4.2 Log in

The following diagram shows the process of signing into Chirp! We decided to use ASP.NET identity for our authentication. We decided to do so, to avoid having to gather the information needed directly from the users. Instead ASP.NET identity allows us to gather the information from the github account of the user that logged in.



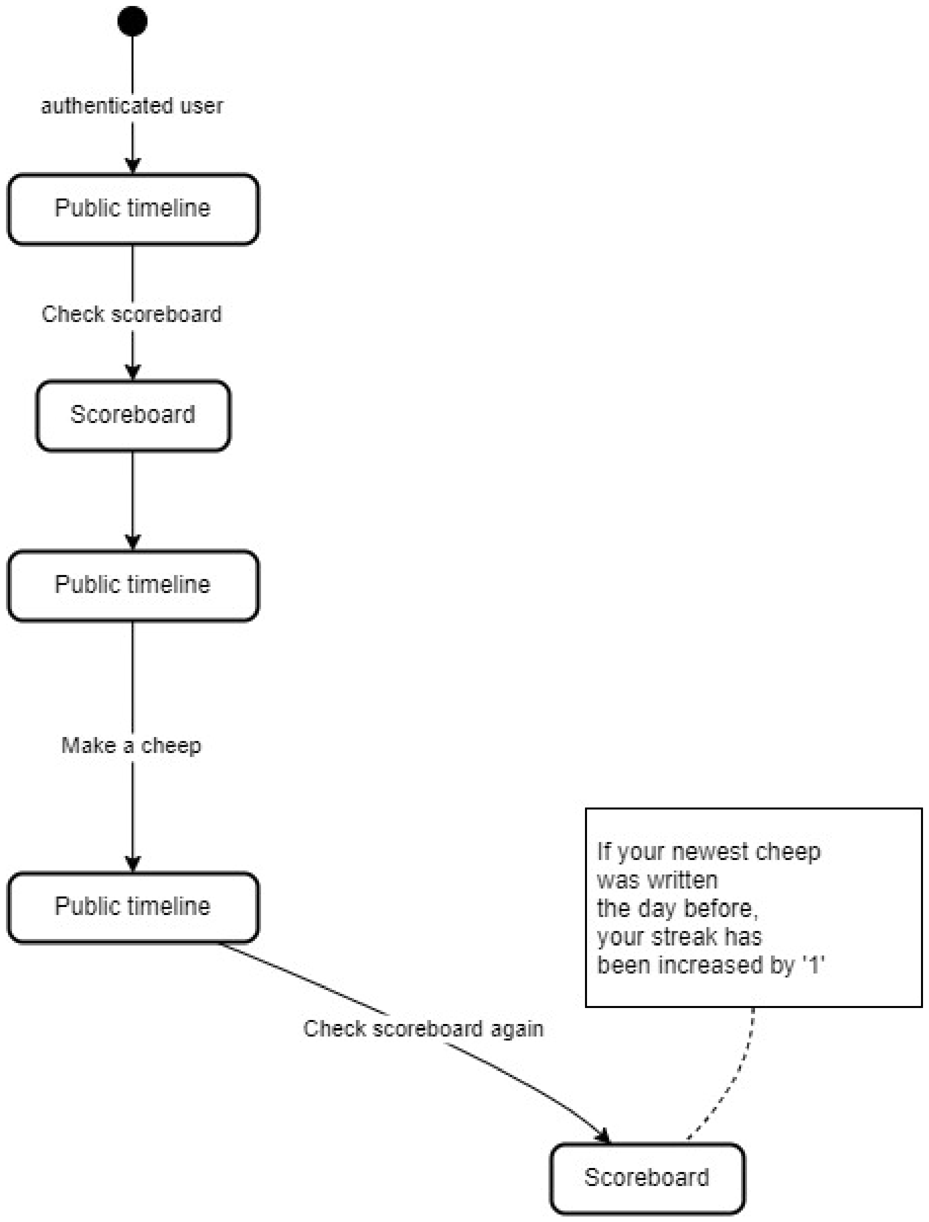
2.4.3 Follow and unfollow

The following diagram shows the process of following an author, viewing their timeline, viewing their cheeps on your own timeline and unfollowing the author.


authenticated user
Public timeline
follow author
Public timeline
click on authors name
Authors timeline
go to own timeline
Private timeline
unfollow author
The same as before, but no longer showing the cheeps of the author
Private timeline

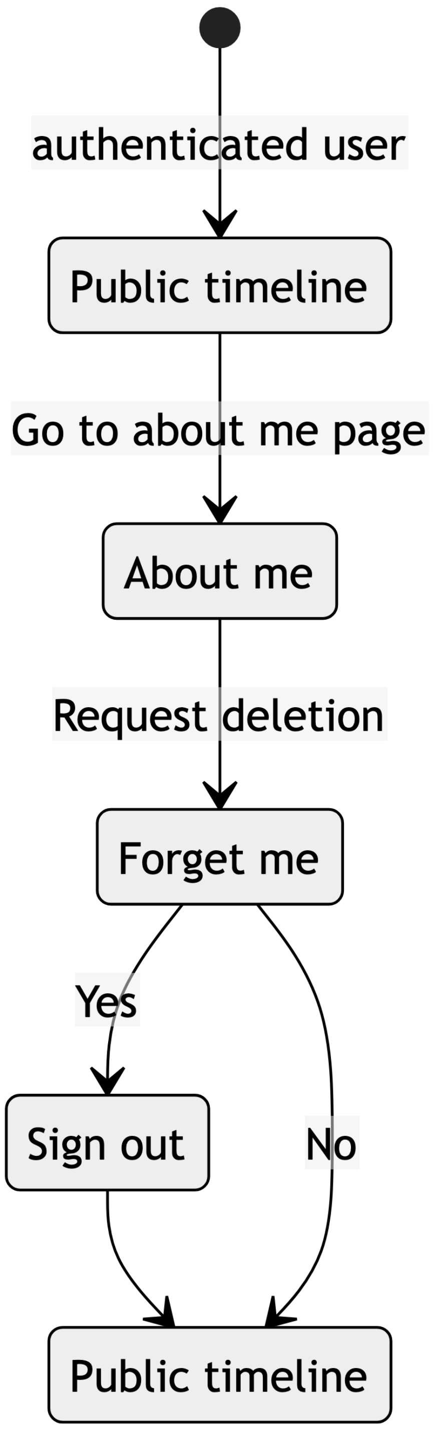
2.4.4 Adding a cheepstreak

The following diagram shows the process of checking the scoreboard, writing a cheep on the public timeline, and checking the scoreboard again to check whether your streak has increased or not.



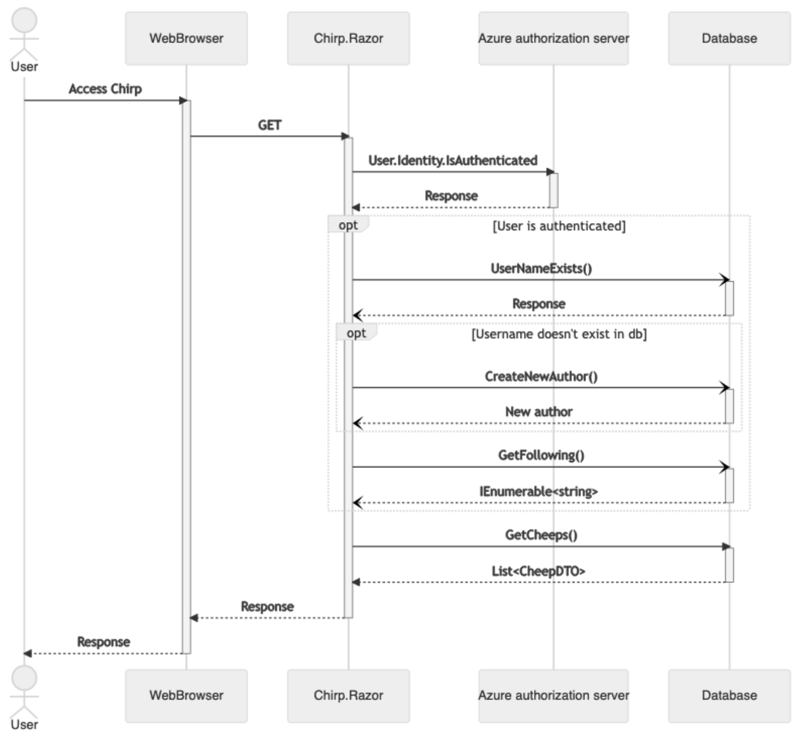
2.4.5 Deleting account

The following diagram shows the process of requesting a deletion of your account, and accepting or rejecting a deletion of the account.


authenticated user
Public timeline
Go to about me page
About me
Request deletion
Forget me
Yes
Sign out
No
Public timeline

2.5 Sequence of functionality/calls trough Chirp!

2.5.0.1 Sequence diagram



This sequence diagram shows what happens when a user accesses the web application.

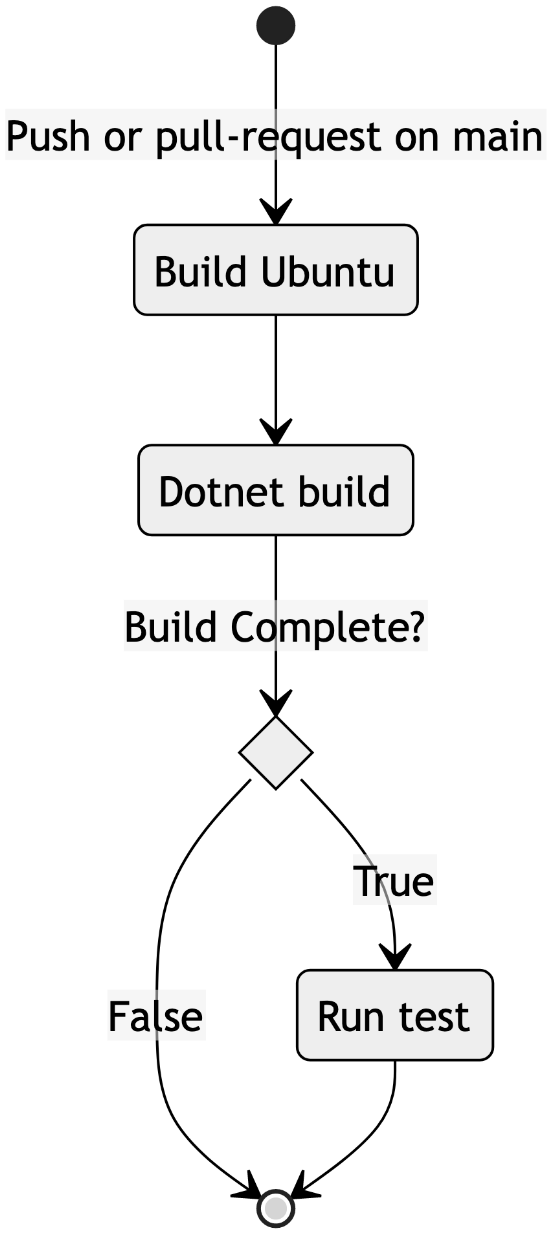
It also shows what happens if the user is authenticated, including what happens if the user hasn't been added to the database yet.

3 Process

3.1 Build, test, release, and deployment

3.1.1 Automatic build and test

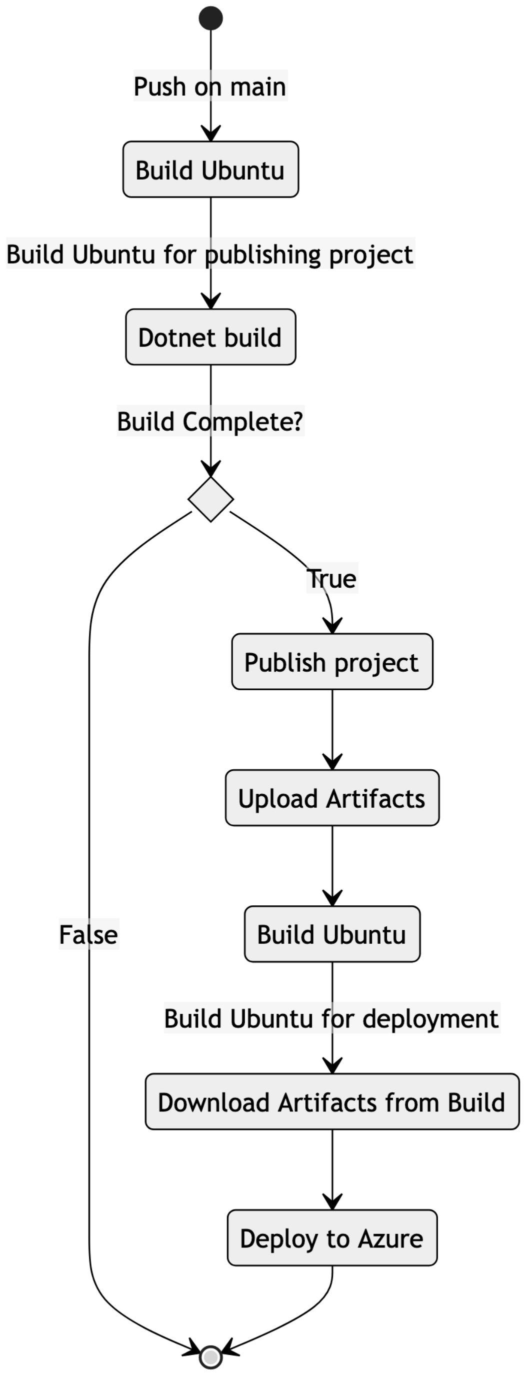
Whenever a push is made to main, or a pull request is made, Github will build and test our program, to make sure that we do not implement a feature that does not pass all our earlier defined tests.


Push or pull-request on main
Build Ubuntu
Dotnet build
Build Complete?
True
False
Run test

Another important note is that all tests are not run by Github, due to the `--filter` added. This ensures that tests that contain the words `Playwright` or `IntegrationTest` in their fully qualified name are not run. This is due to these tests not always compiling correctly on github.

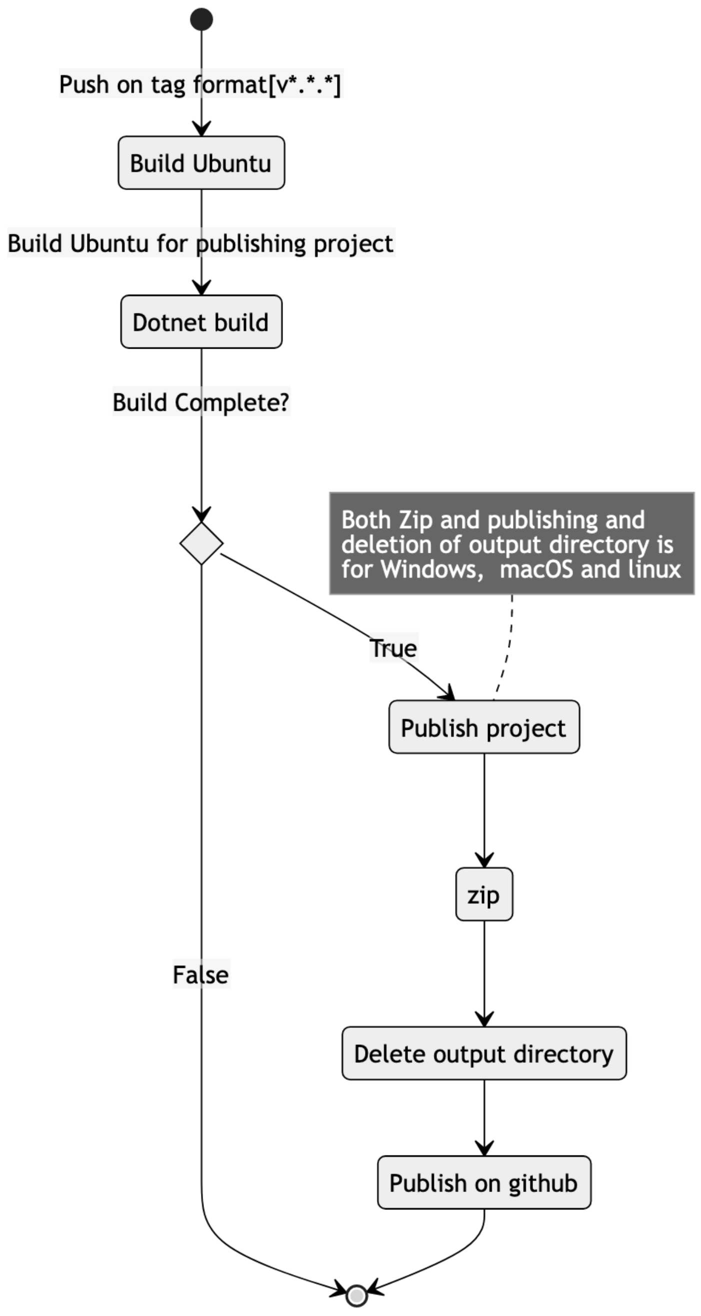
3.1.2 Build and deploy to Azure

When a push is made to main, it is automatically deployed to our Azure website. We discussed having the program tested before deploying it to make sure that it would work, but decided not to do that since some of our tests was testing our website directly, which could cause problems.


Push on main
Build Ubuntu
Build Ubuntu for publishing project
Dotnet build
Build Complete?
True
False
Publish project
Upload Artifacts
Build Ubuntu
Build Ubuntu for deployment
Download Artifacts from Build
Deploy to Azure

3.1.3 Automatic build and release to github

Whenever a tag is pushed with the format `v*.*.*` it is automatically build, published and released to github, with a zip folder for both windows, macos and linux.


Push on tag format[v*.*.*]
Build Ubuntu
Build Ubuntu for publishing project
Dotnet build
Build Complete?
Both Zip and publishing and deletion of output directory is for Windows, macOS and linux
True
Publish project
zip
False
Delete output directory
Publish on github

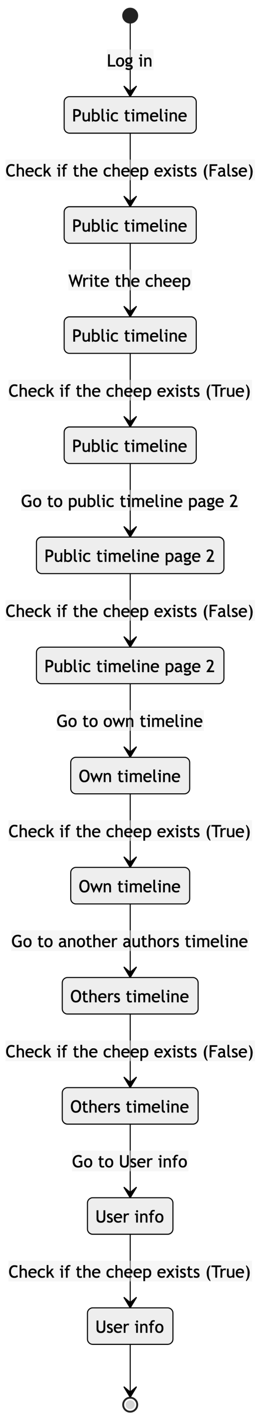
3.2 End-2-End testing

In the current implementation of our End-2-End test, it is running via localhost. This means that the database being used is the online Azure Database(It would in the future be favorable to have a local database running in memory instead, to ensure the data has not been manipulated with, and it would ensure more durability for the tests). Because of this it currently would not be preferable to have tests like for example ones that check for if there exists an unfollow button, due to it not being assured the tests before have not manipulated with the followers.

3.2.1 Issues with the End-2-End testing

Due to the fact that Chirp! was implemented with B2C authentication via Github, Authentication in End-2-End testing was not easy to implement. This was primarily due to the chance of Github's "device verification" feature would occasionally pop up during testing, which the playwright testing had no way of solving. Because of this the tests have been made with one part without log in, that should work without any problems, and one with logging in, that occasionally fails due to what was mentioned before. We did however prepare some test cases for End-2-End testing, for when we got the log in working, here is a diagram showing one of the test cases we prepared.

The following test case checks if a new cheep shows up on the right pages, and nowhere else. Every time 'the cheep' is mentioned, we check for the specific text defined in the test and the author of the test user:

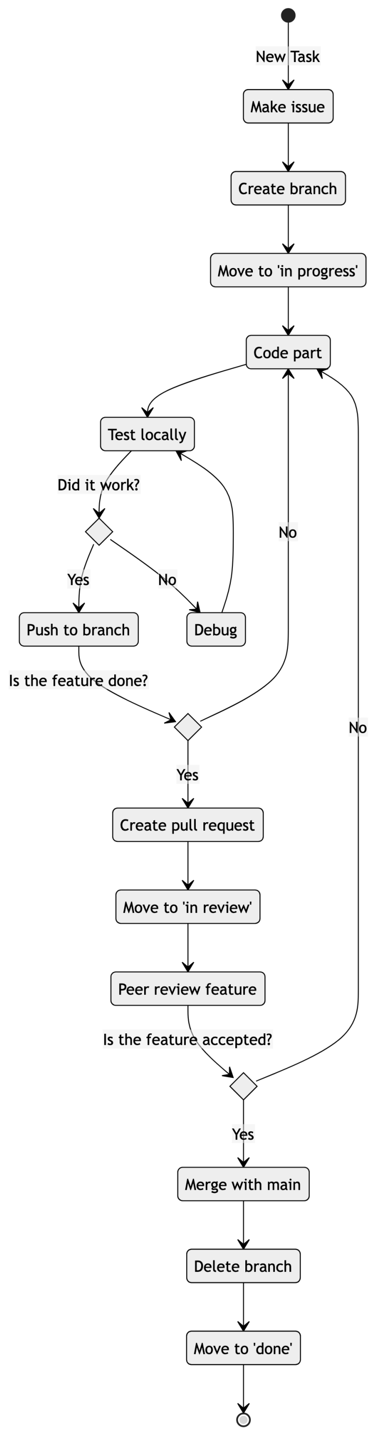


Log in
Public timeline
Check if the cheep exists (False)
Public timeline
Write the cheep
Public timeline
Check if the cheep exists (True)
Public timeline
Go to public timeline page 2
Public timeline page 2
Check if the cheep exists (False)
Public timeline page 2
Go to own timeline
Own timeline
Check if the cheep exists (True)
Own timeline
Go to another authors timeline
Others timeline
Check if the cheep exists (False)
Others timeline
Go to User info
User info
Check if the cheep exists (True)
User info

3.3 Team work

3.3.1 Process of implementing new feature

Here is a diagram of a normal process from having a new feature in mind, to having it made and integrated in our program. It shows, the coding and testing process as well as the issue's state on the project board

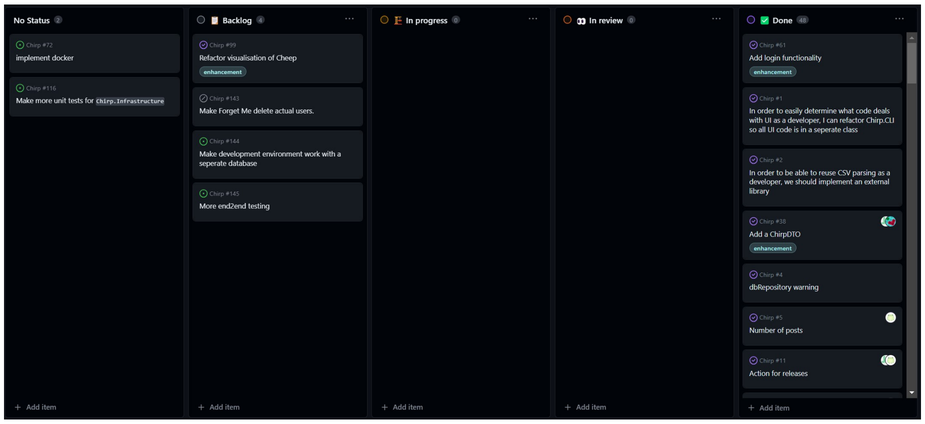

New Task
Make issue
Create branch
Move to 'in progress'
Code part
Test locally
Did it work?
Yes
No
Push to branch
Debug
Is the feature done?
No
Yes
Create pull request
Move to 'in review'
Peer review feature
Is the feature accepted?
No
Yes
Merge with main
Delete branch
Move to 'done'

3.3.2 Missing features/functionality

The ‘Forget me’ feature is used by users to delete their accounts. In it’s current form in the program the feature flips the boolean ‘Hidden’ on an Author to be set to true. This means the feature is not deleting anything from our database, but just hiding it on the web application. With more time we would have liked to change the feature so it deletes users and their cheeps from the database. This way the feature would be more GDPR compliant, so we won’t have to manually delete data from the database if a user requests for their data to be deleted.

Currently when running our program in a development environment the program will still use the same database as on our production environment. With more time we would have liked to make the development environment run on a separate database, to avoid tampering with the live database when testing and making new features.

3.3.3 Project board workflow



No Status 2

Chirp #72
implement docker

Chirp #116
Make more unit tests for `Chirp.Infrastructure`

+ Add item

Backlog 4

Chirp #99
Refactor visualisation of Cheep
enhancement

Chirp #143
Make Forget Me delete actual users.

Chirp #144
Make development environment work with a seperate database

Chirp #145
More end2end testing

+ Add item

In progress 0

+ Add item

In review 0

+ Add item

Done 48

Chirp #61
Add login functionality
enhancement

Chirp #1
In order to easily determine what code deals with UI as a developer, I can refactor Chirp.CLI so all UI code is in a seperate class

Chirp #2
In order to be able to reuse CSV parsing as a developer, we should implement an external library

Chirp #38
Add a ChirpDTO
enhancement

Chirp #4
dbRepository warning

Chirp #5
Number of posts

Chirp #11
Action for releases

+ Add item

In our project board we have 5 columns, ‘No status’ is the first column, where all new issues end up, and this is where issues stay until we start working on them. When a project is being worked on it should be moved to ‘In progress’, and once ready for review and the pull request is up it is intended to move to ‘In review’. Once the pull request is accepted and issue is done, it should be moved to ‘Done’ and the issue closed. We also have the column ‘Backlog’ which is used for issues that we decided are not a priority, and can be looked at if we have time after doing more pressing issues.

At first we had more columns in the board, but we deleted these, as we weren’t using them, and as such they were just cluttering the board, and making it harder to get a grasp of the issues. Much in the same vein we haven’t been great at using the ‘In review’ column, but we decided to keep the column since it’s purpose is still relevant.

3.4 How to make *Chirp!* work locally

To run our Chirp! locally, the first step is to clone our git repository:

```
git clone https://github.com/ITU-BDSA23-GROUP7/Chirp
```

When the repository is clone, make your way into the `Chirp/src/Chirp.Razor` directory:

```
cd Chirp
cd src
cd Chirp.Razor
```

In this directory you need to set up a user secret containing a connection string to an sql-server using the following command:

```
dotnet user-secrets set "AZURE_SQL_CONNECTIONSTRING" "[insert connection string]"
```

Where `[insert connection string]` is replaced with a connection string referring to an sql server.

When the user secret is made, the program can be run from the same directory using one of the two following commands:

```
dotnet run
```

```
dotnet watch
```

When the program is running you can find it in your browser by following the url in your terminal e.g. `localhost:5000`.

3.5 How to run test suite locally

To run our End-2-End test in the root directory `/Chirp` from the command:

```
npx playwright install
```

Then run the program as described above, because the playwright tests use the program running on localhost.

Open another terminal in the root of our repository, `/Chirp`, run the command:

```
dotnet test
```

4 Ethics

4.1 License

We have chosen the MIT License for our application.

4.2 LLMs, ChatGPT, CoPilot, and others

Through the project, we have used LLM's occasionally. Most of all we used ChatGPT to help us understand different new libraries or architectures that we needed to use. ChatGPT was occasionally used to help us debug our code. It rarely had a direct influence on what we wrote, and has been co authored whenever this occurred.

We used CoPilot once to speed up a task, but has never really used it for anything helpful. CoPilot was co authored as well.