

Chirp! Project Report

ITU BDSA 2023 Group 24

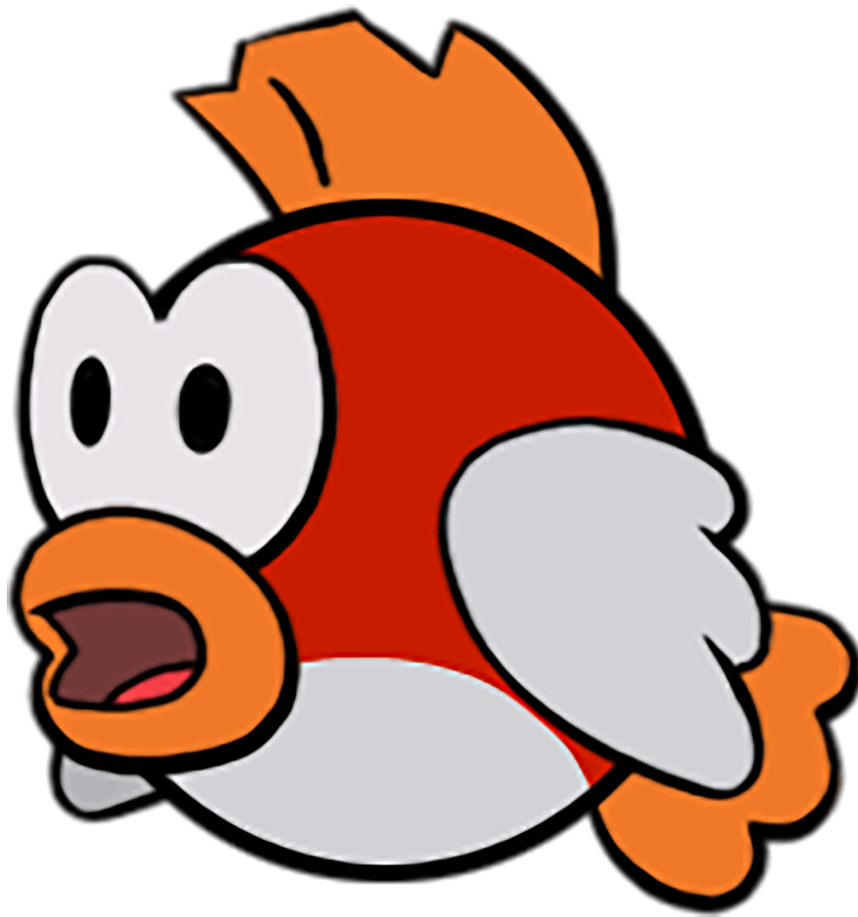
Anton Vadsholt avad@itu.dk

David August Ringenson drin@itu.dk

Emil Fenger emfe@itu.dk

Frederik Bjerre Bertelsen frbb@itu.dk spicyoverlord@gmail.com

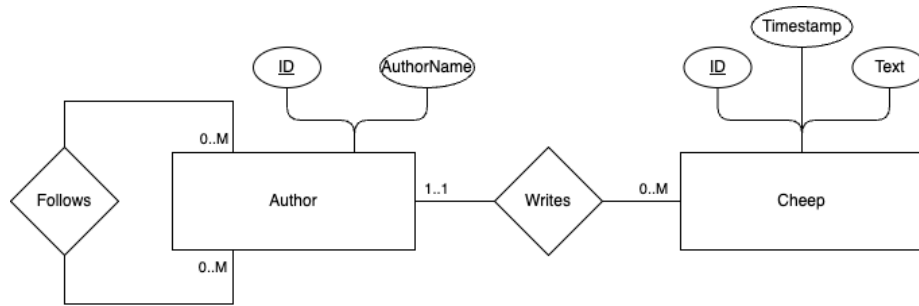
Oskar Wernegreen oskw@itu.dk



1 Design and Architecture of *Chirp!*

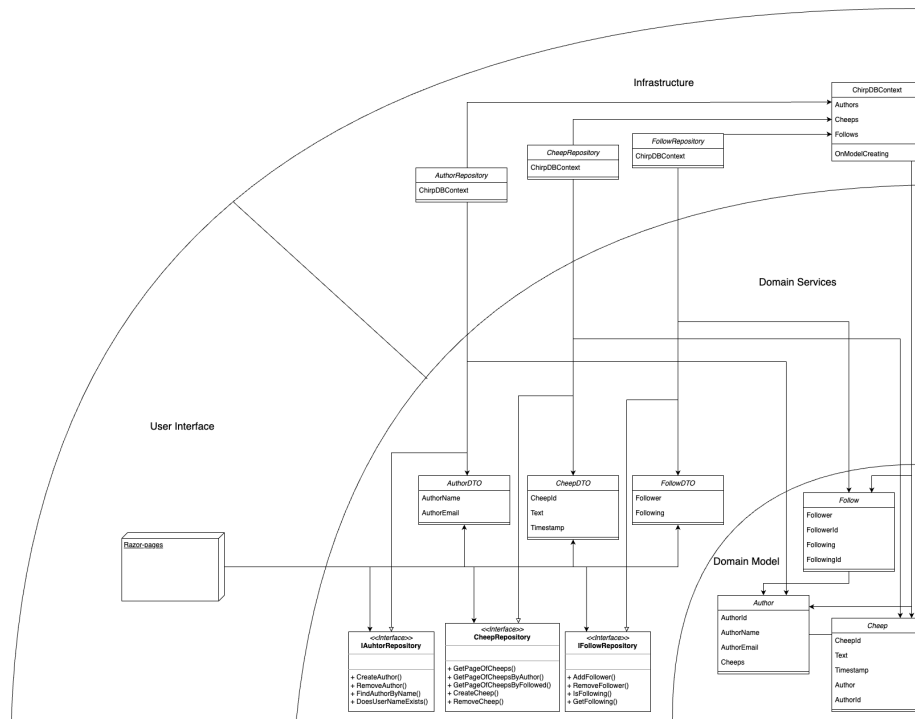
1.1 Domain model

The chirp model contains 3 entities Author, Cheep and Follow - Author contains information about the user and a list of cheeps that the author has authored. - Cheep is the post type of Chirp. A cheep has a single author, a timestamp and the text of the message that was cheeped. - Follow is the relation between two authors where one can follow another. The Follow class contains two authors, the follower and the following.



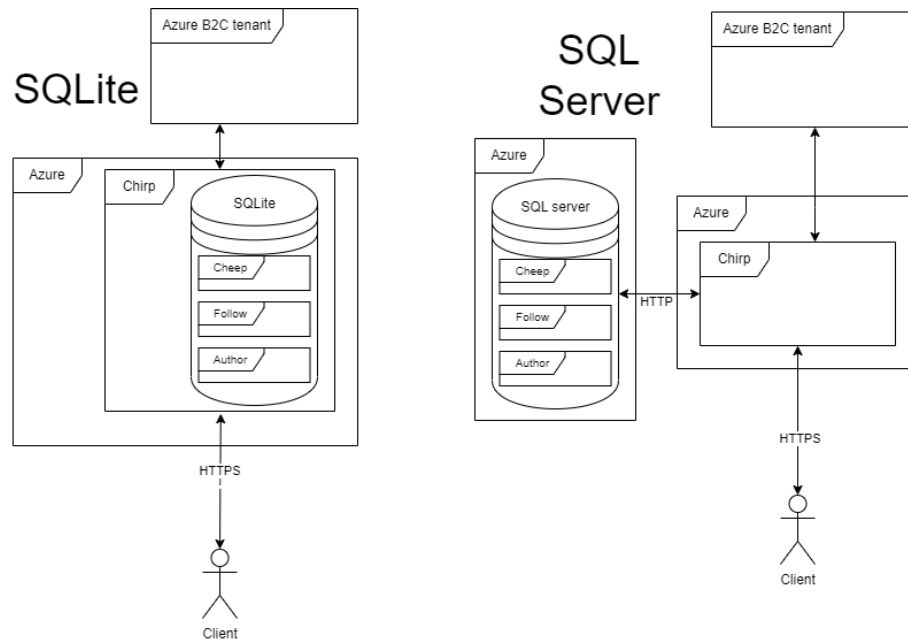
1.2 Architecture — In the small

This application is structured in an onion architecture. The entities in the domain model have DTOs which contain all data necessary for the view. As seen in the illustration below the AuthorDTO does not contain the values 'AuthorID' or 'Cheeps' even though they are contained in the Author entity. This is because these values are not necessary in the view, but are useful in the database. Though the DTOs contain almost the same values as the entities, these ensure forward compatibility by making dependency injection possible. The repositories for the entities in the domain services layer are kept in the infrastructure layer. The repositories implement from an interface in the domain layer which is available to the UI. The repositories translate the data stored in the database to DTOs. The repositories also make the necessary checks on the input values to avoid invalid data being saved to the database. The user interface consists of Razor pages which have references to the repository interfaces. Program.cs exists outside of the onion architecture and runs the setup and organizes the interactions across layers.



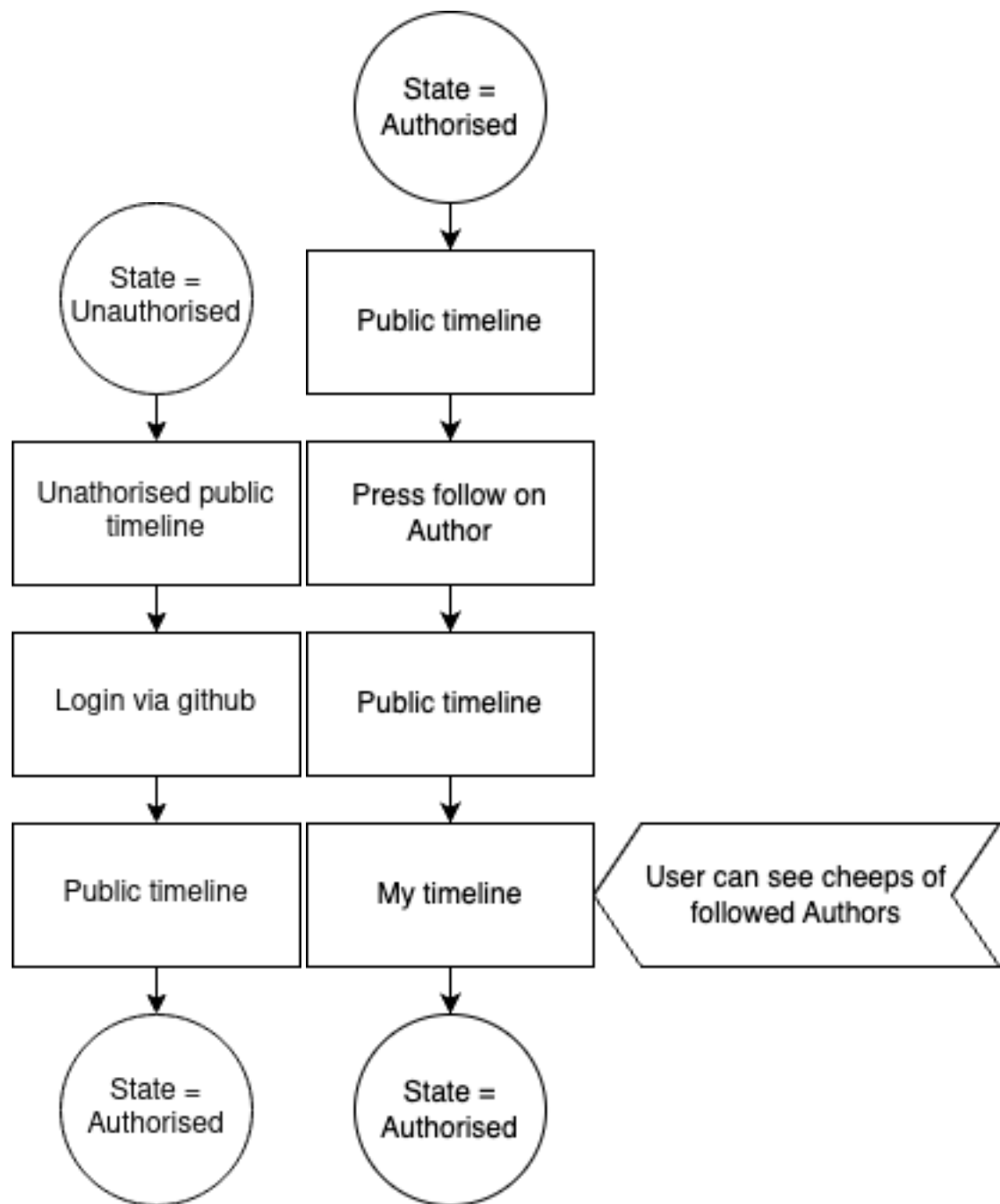
1.3 Architecture of deployed application

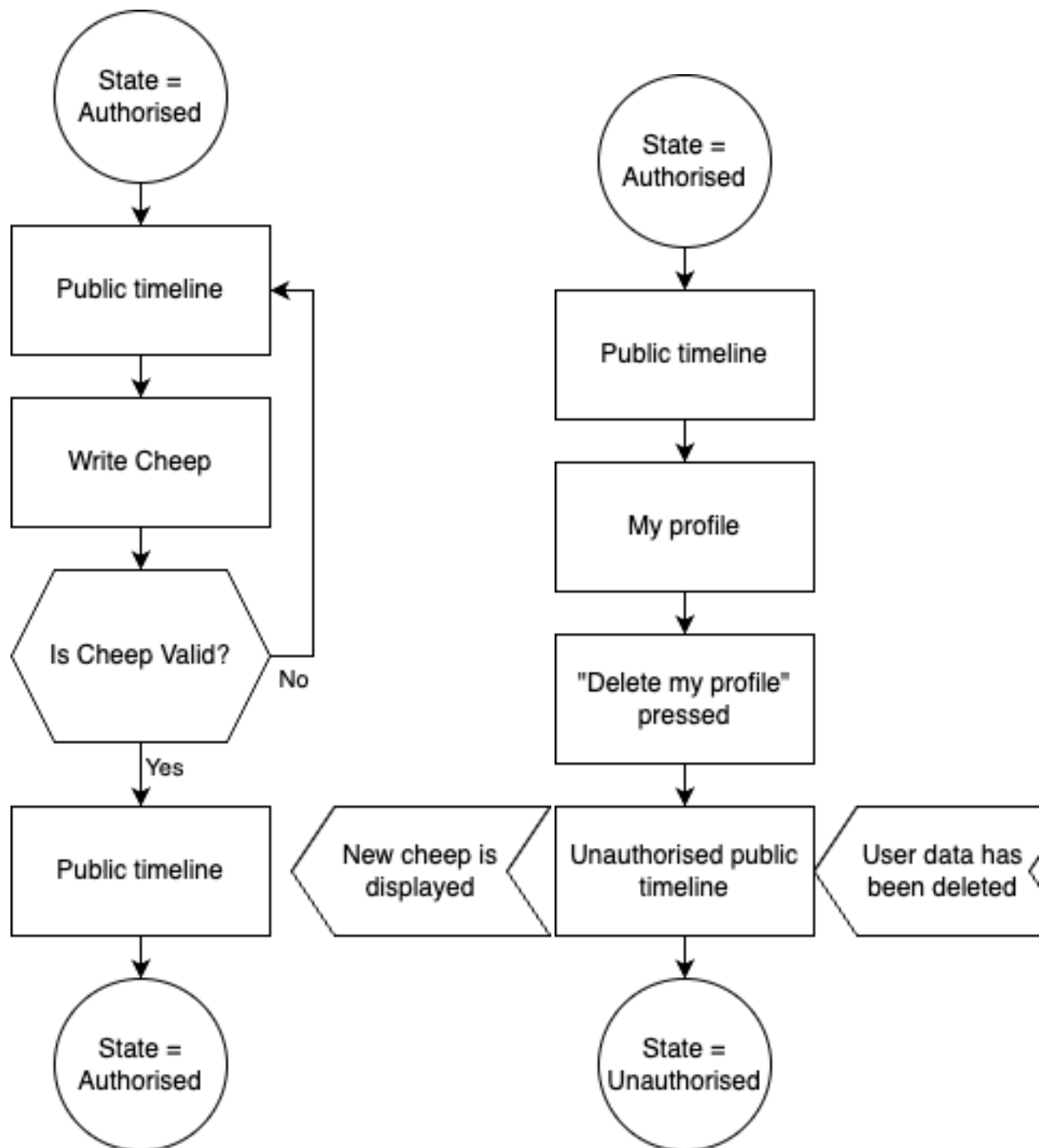
Below is shown two illustrations one depicting the client server relationship while having a sqlite database and the other with an sql server database. After changing Chirp to work with an Azure SQL database, we quickly ran out of credits. We could have solved this by switching to using a different group member's credits. However, had we done this, we would still risk running out of credits before the project was graded. Instead we chose to switch back to using SQLite. We saw no issue in doing this, as we had already demonstrated that we could get Chirp to work with the Azure SQL database. Link to commit hash of last version of Chirp with a hosted database: [Commit link](#)



1.4 User activities

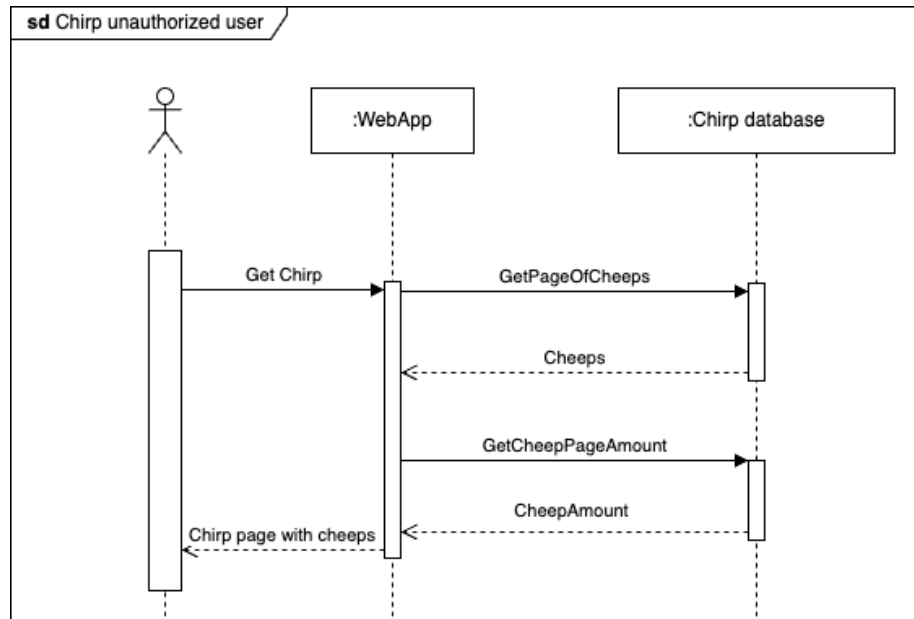
Below are the illustrations of the most important user activities in Chirp. These includes log in, follow another user, write cheep and deleting your profile. For each task or page, if it is not specified, the user is authorised. Cheep and follow can start from all timeline pages.





1.5 Sequence of functionality/calls through *Chirp!*

As seen in the sequence diagram, when connecting to the web application as an unauthorized user two calls to the database are made. First the cheeps that should be displayed are retrieved from the database. Secondly in order to display page count in the user interface, the total amount of cheeps accessible are retrieved. Then the finished page is returned to the user.



1.6 Extra features - wild style

We chose to implement two extra features beyond the required functionality of Chirp. The first was cheep deletion. Implementing it was simple, as a method for removing cheeps from the database already existed in the cheep repository. However, it also required us to include the ID of cheeps in the cheep DTO even though the ID is never shown to users. This is not ideal, though it is a necessary compromise to make. The same would be necessary for most features that add further functionality to cheeps, like “likes”, commenting, etc.

The second feature we added was “tagging” other authors in cheeps. This means a user can create links to other authors when writing cheeps (to tag, use the format ‘@(author)’). A tagged author is not notified of them being tagged. The intention was for tagging to be for readers of the cheep more so than the tagged author. Notifying tagged authors would be a good, and obvious, further enhancement for this feature.

Usage example:

What's on your mind OskarsGit?

hey, everyone should check out @(Jacqueline Gilcoine)'s cheeps, they are really cool

Cheep!

Example result:

[OskarsGit](#) [delete](#) hey, everyone should check out [Jacqueline Gilcoine](#)'s cheeps, they are really cool – 20-12-2023 18:53:54

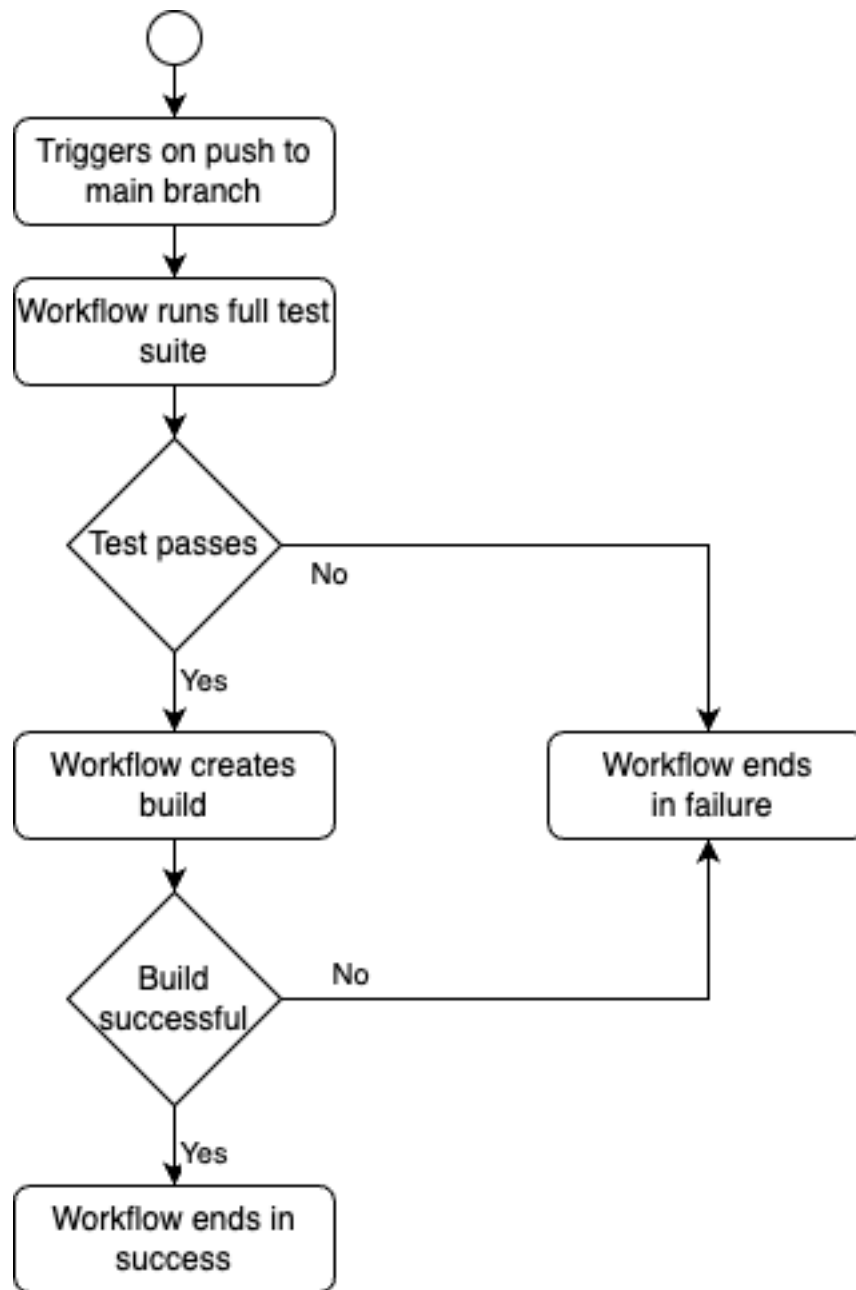
2 Process

2.1 Build, test, release, and deployment

For our builds, tests, releases and deployments we use github workflows exclusively. This has the purpose of automating these tasks to save time. We use three different workflows to achieve these goals that trigger at different times, “build and test”, “Release” and “Deployment”.

2.1.1 Build and test workflow

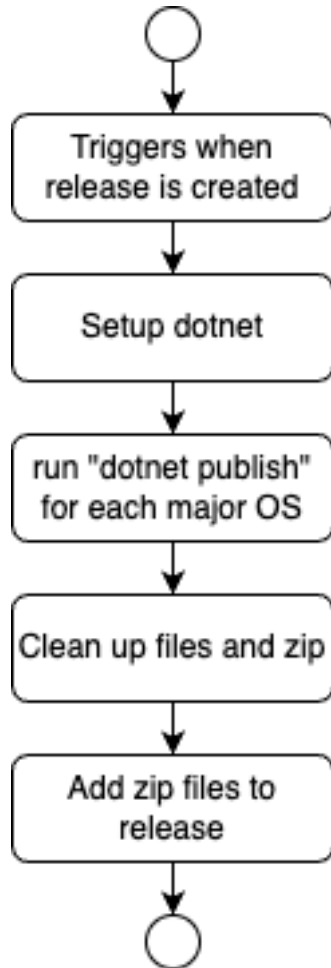
Our build and test workflow triggers on all pushes to branches main and development. This had the purpose of running before we merged into the main branch. However, this functionality was lost, when we were asked to not use a development branch, as we did not update our workflow. Our intention was to have a development branch, that all code would be reviewed on, before it being merged into main. We decided that it was obsolete to have this workflow run on all pushes.



2.1.2 Release workflow

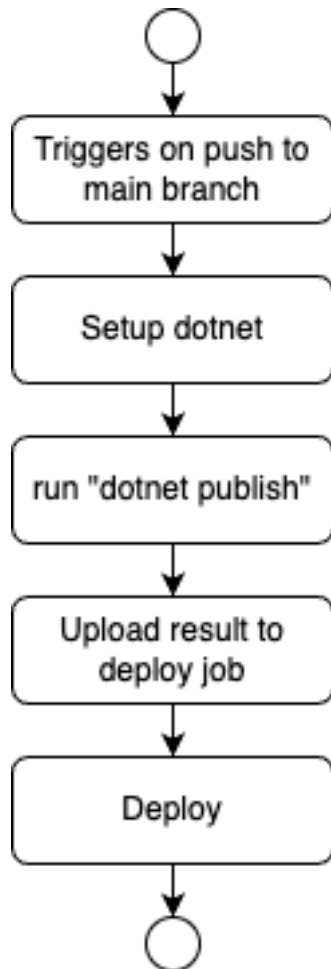
This workflow triggers when we manually create a release. We wanted to use github to automatically create a release when we pushed with a tag, but we could

not get it to work. We prioritized continuing development, as this wouldn't save a lot of time compared to manually creating releases. This workflow creates a build for each OS and cleans up and zips the files, and sends them to the release.



2.1.3 Deployment workflow

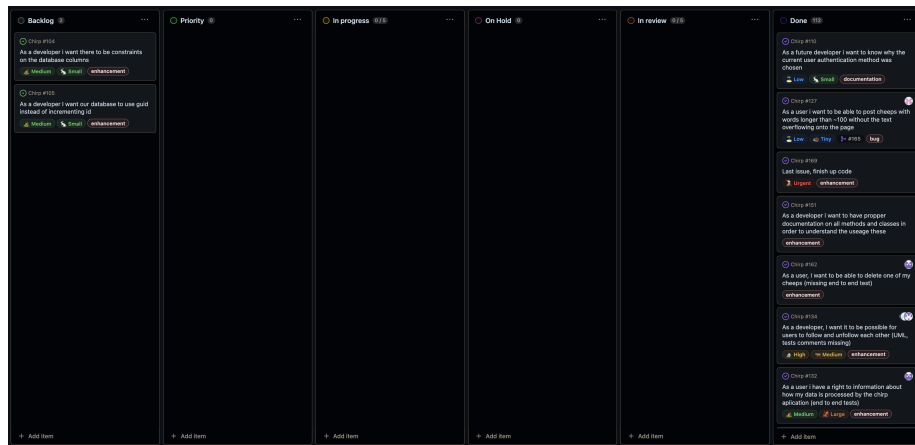
This workflow triggers on all pushes to main. This workflow is only expected to be triggered through a pull request, so all code should have been tested and a build created before the pull request is created, where the results can be seen. This workflow sets up our .NET environment with EF Core, and runs “dotnet publish”. Then the result of “dotnet publish” is copied to another job that deploys it to the server. This other job is generated by Azure and slightly modified to deploy the correct folder.



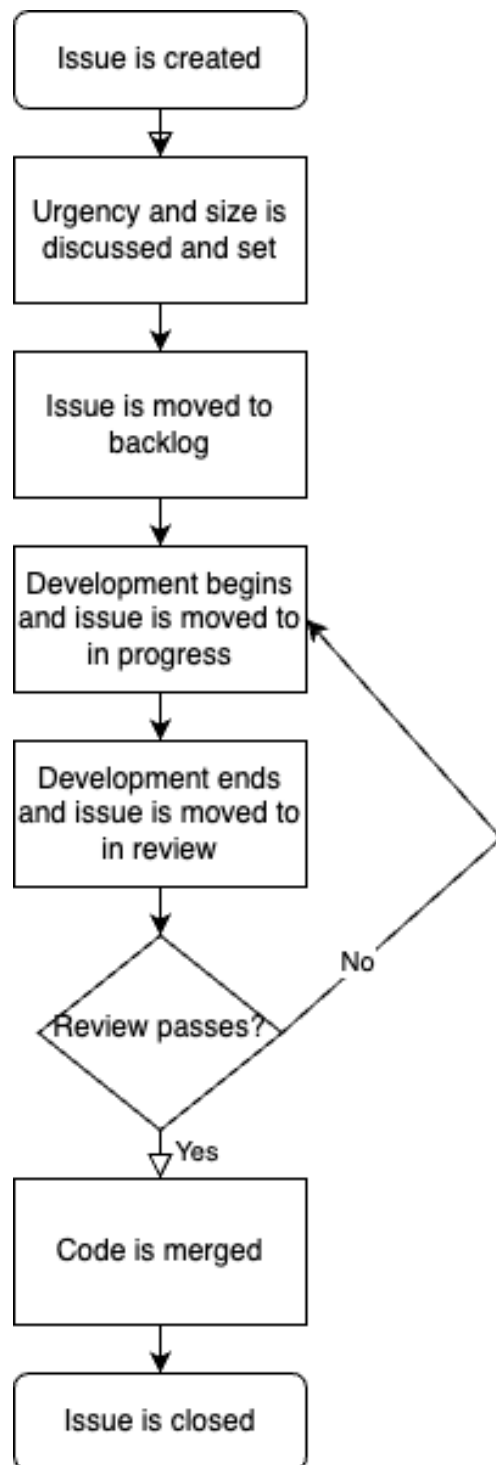
2.2 Team work

At the very beginning of the group work, we wrote a document (“group contract”) detailing the good practices we wanted to follow. This included agreements about coding practices, branch management, etc. We also outlined our expectations of this project, and our own engagement.

Throughout development, we kept our project board updated. In the end, it looked like this:



The two backlog issues are about “constraints on the database model” and “use GUID instead of incrementing ID”. The constraints on the data model are mostly taken care of in other places, for instance when writing a cheep, though constraining the database in addition to this would be preferable. The change from incrementing ID to GUID is mostly relevant in the hypothetical case that the database grows very large, ie. a matter of future-proofing. Both were deprioritized because of time constraints, as neither of these are essential to a program at this scale.




After each lecture, we went through the project readme and wrote each task as an issue. During development, we wrote further issues when necessary. After each issue was created, we decided on the urgency and size (in time). Then the issue was moved to 'backlog' on the project board. Group members were then assigned to work on the issue. When development on an issue started, it was moved to 'in progress'. When the development was done, including writing and running tests, the issue was moved to 'in review'. If the review passed, the code was merged to main, the issue was closed and the related branches were deleted. If the review did not pass, it was put back into 'in progress'.

We had 2 issue templates "Bug report" and "Feature request":

Issue: Bug report










Create a report to help us improve. If this doesn't look right, [choose a different type](#).

**Add a title**

Title

Add a description

WritePreview

H B I   |    |  @   

Bug description
A clear and concise description of what the bug is.



Steps to reproduce the behavior

1. Go to '...'
2. Click on '...'
3. Scroll down to '...'
4. See error

Expected behavior
Write a description of what should happen (without the bug).

Success Criteria

- [] Write the success criteria here.
- [] ...
- [] There are no warnings or errors when running 'dotnet build'


 Markdown is supported  Paste, drop, or click to add files

Submit new issue

14

Issue: Feature request

Suggest an feature for Chirp! If this doesn't look right, [choose a different type](#).



Add a title

Add a description

Write

Preview

H B I | | @

Feature description

Write a description of what the feature is and what it will do.

****[Corresponding lecture notes](https://github.com/itu-bdsa/lecture_notes/blob/main/sessions/session_X/...)****

Success Criteria

- [] Write the success criteria here.
- [] ...
- [] There are comments for each new method created
- [] There are tests that tests the implemented functionality
- [] The tests pass without issue
- [] There are no warnings or errors when running 'dotnet build'

Markdown is supported
 Paste, drop, or click to add files

Submit new issue

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

2.3 How to make *Chirp!* work locally

First, clone the repository from github to local storage. Do so however you prefer. Navigate to `/Chirp/src/Chirp.Web`, and run:

```
dotnet run
```

This starts Chirp. A localhost link will be written in the terminal. Follow it, and you are now on a locally hosted Chirp.

See README.md for how to run with a SQL Server database.

2.4 How to run test suite locally

Run command in repository root:

```
dotnet test
```

We have two types of tests. Firstly we have unit tests in the form of tests on the cheep, author and follow repositories. These tests make sure that our repository tasks work as intended. Secondly we have integration/end-to-end tests using WebApplicationFactory, where we fetch the html code for the application and then use it to confirm that the different pages in the application contain the correct information. We do not have tests for the application while logged in since we didn't have enough time to make playwright work.

3 Ethics

3.1 Data handling

For Chirp, we do not need to process much data about a user. We justify our data processing with point (f) of the first subparagraph of article 6 of the GDPR¹, with the data processing being in the legitimate interest of the users. The data processed is strictly the data necessary for Chirp to function. The legal requirement of data security is mostly ensured by Azure AD B2C. The “my profile”-page of a user is only available to the user itself, to protect data confidentiality, meaning a user is only authorized to view their own profile page. This page exists to fulfill the user’s “right to access”², ie. the right to know what data is being processed about them, and why. Here a user will also find the “Delete my profile”-button, which removes all data about a user from both public view and the database. This button is a requirement, as users have a “right to erasure”³. The current implementation does not erase user Identity data from Azure B2C, due to time constraints.

3.2 License

After some consideration, we decided to use the “GNU General Public License”, Version 3⁴, which is an open source, “copyleft” license. Most importantly, this means that: - The code is open for anyone to use, copy, and modify. - Any derivative work must also be licensed under the GNU General Public License Version 3, and therefore also open for others to use, copy and modify.

We chose this license because of our shared interest in open source development. Another alternative we considered was the MIT license, which would allow anyone to copy or use our code in any way, including for closed source purposes. While it is quite unlikely that anyone will ever use our code for a different project, we still chose this “copyleft” license out of principle. All used libraries use the MIT license which is less restrictive, meaning that there are no licensing conflicts as far as we can tell.

3.3 LLMs, ChatGPT, CoPilot, and others

Throughout the development process, we made limited usage of LLMs, in particular ChatGPT. When setting up Azure B2C and other things relating to the Azure server, we made use of ChatGPT, as we found many of the guides relating to Azure to be confusing or inconsistent. We chose to use ChatGPT to boil these down to a few briefly-described steps. As such, ChatGPT was used more as a summation tool than a “problem solver” in these cases.

¹<https://gdpr-info.eu/art-6-gdpr/>

²<https://gdpr-info.eu/art-15-gdpr/>

³<https://gdpr-info.eu/art-17-gdpr/>

⁴<https://www.gnu.org/licenses/gpl-3.0.en.html>

We did not use ChatGPT much for the coding itself, simply because we did not find it very useful. For instance, we had issues getting “tag helpers” for Razor Pages to work. After failing to find answers on the internet, we asked ChatGPT, which only suggested all the things we had already tried. We recognize that even in these cases, we should have included ChatGPT as co-author on the commits, in spite of its limited usefulness.

To conclude, ChatGPT sped up some parts of the development process, and slowed other processes. Ultimately, we believe it was more helpful than not.