Chirp! Project Report (Techical documentation)

ITU BDSA 2024 Group 14

August Bugge aubu@itu.dk
Daniel Trebbien Haugbølle dtha@itu.dk
Jonas Boll Esser jones@itu.dk
Jonathan Antoine Villeret jonv@itu.dk
Oliver Starup osta@itu.dk

1 Table of Contents

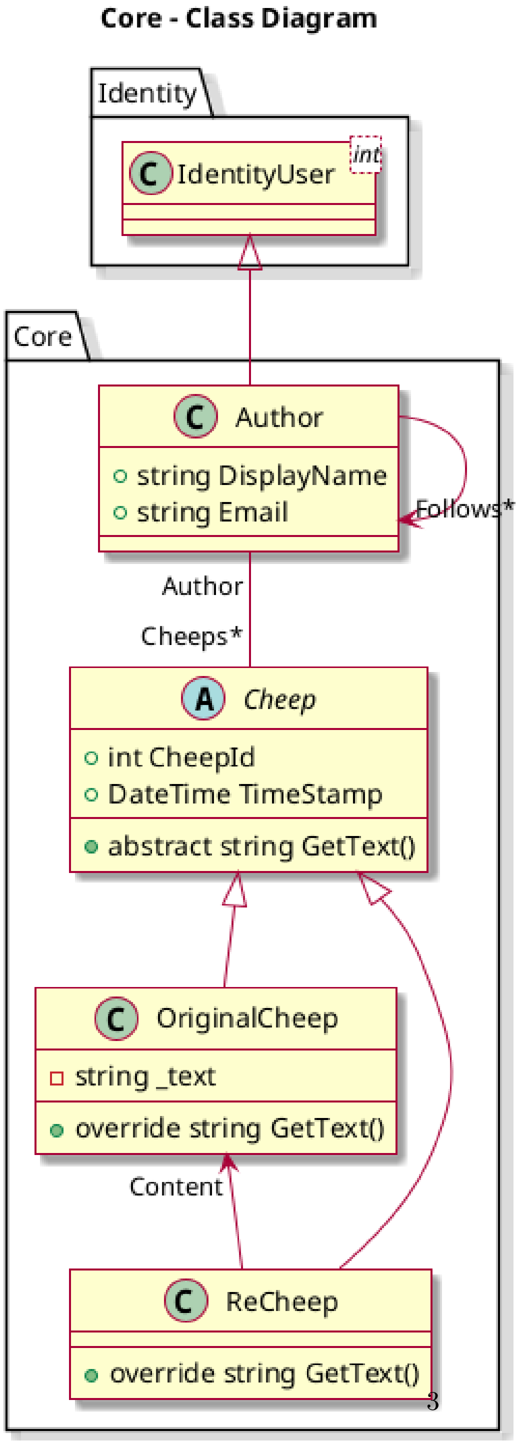
2 Introduction

The following report was created for the course Analysis, Design and Software Architecture. The report will go into the architecture of the project, design decisions, development project, and specific ethics decisions.

This documentation contains inline PlantUML, that automatically compiles using GitHub Actions and Docker. It is therefore recommended to read the PDF document to see the rendered PlantUML diagrams.

3 Design and architecture

3.1 Domain model

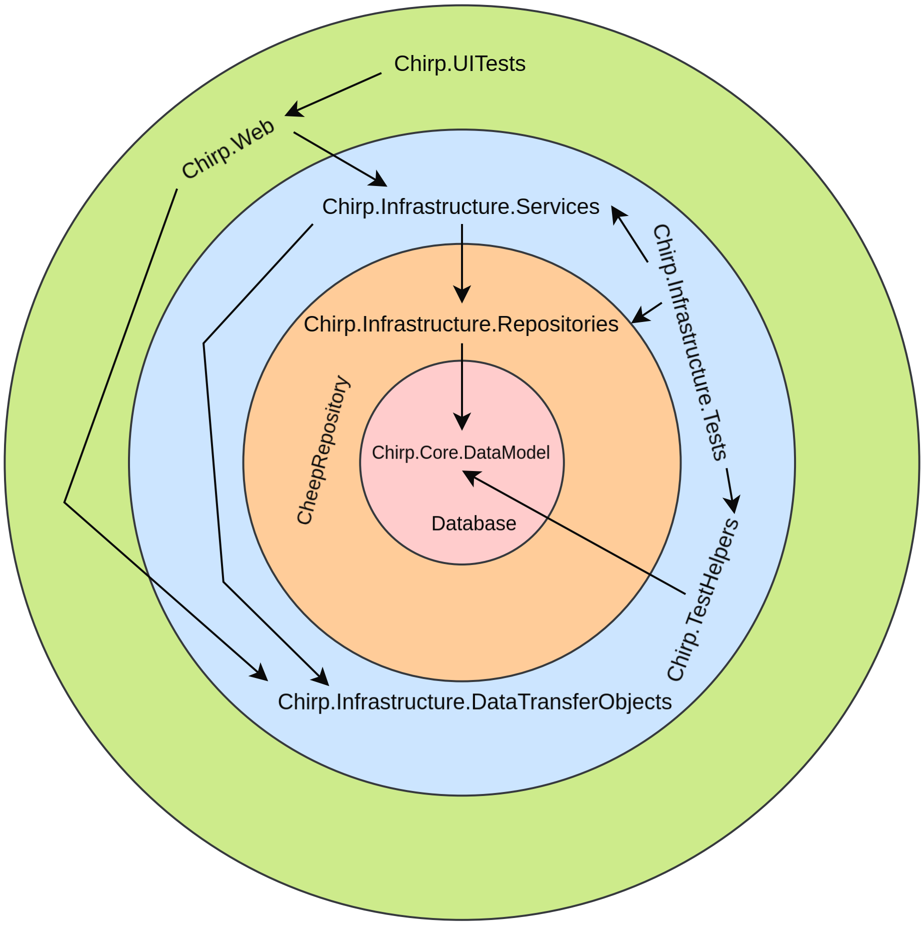


The Domain Model is implemented in the *Core* project. This contains the classes that represent the core of the data structure. The core has, by design, very few dependencies. It depends only on `Microsoft.AspNetCore.Identity.EntityFrameworkCore`. It is the entities defined in the *Core* that are saved in the database.

An `Author` represents a user of the system. It extends the `IdentityUser` class to allow for authentication. It extends from `IdentityUser<int>` to have an integer key, to follow the same structure as Cheeps.

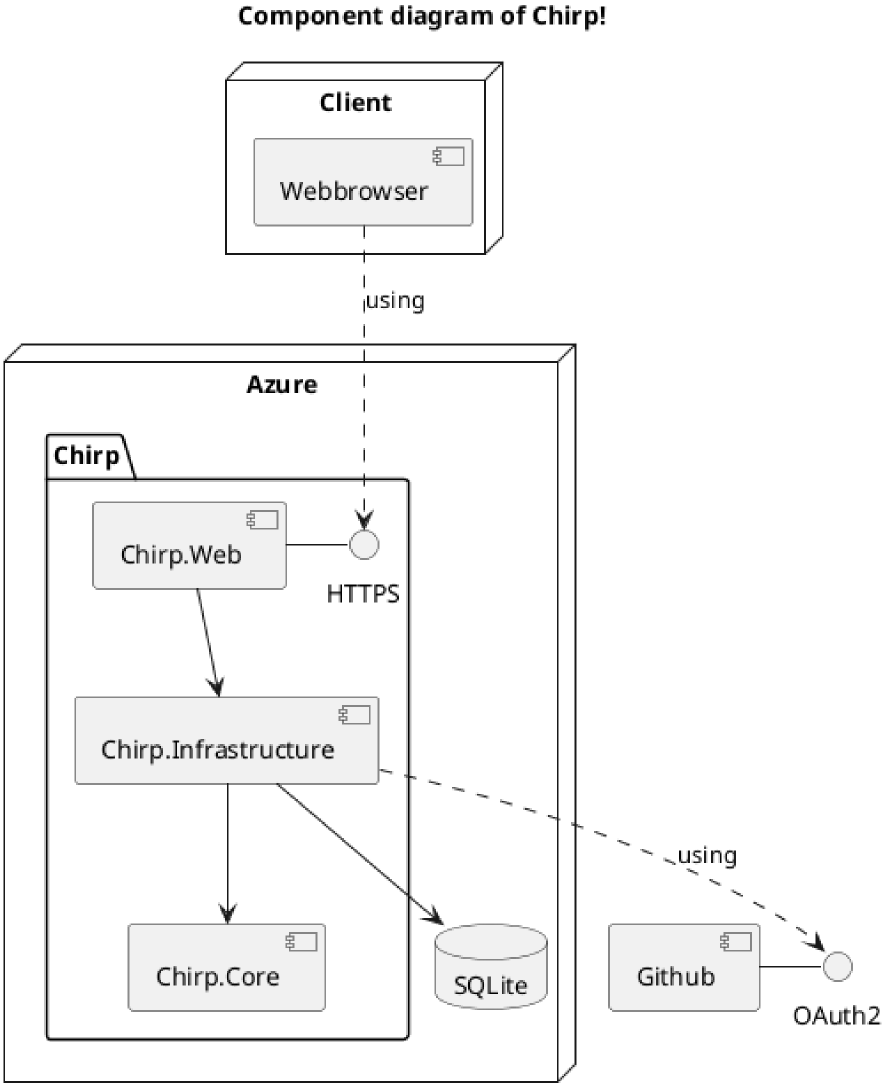
A `Cheep` represents something an `Author` can post. `OriginalCheep` represents a `Cheep` written by the `Author`, while a `ReCheep` represents a repost of an `OriginalCheep` by another `Author`.

3.2 Architecture — In the small


Chirp.UITests
Chirp.Web
Chirp.Infrastructure.Services
Chirp.Infrastructure.Tests
Chirp.Infrastructure.Repositories
CheepRepository
Chirp.Core.DataModel
Database
Chirp.TestHelpers
Chirp.Infrastructure.DataTransferObjects

Above is a diagram of the onion structure of the program. It follows the onion structure, however some onion layers contains more than one .NET project. The *Core* .NET project correspond to the core onion layer while the *Infrastructure* .NET project is split across both the repository layer and service layer. The DTOs exist in the service layer, since these are only used in Chirp.Infrastructure.Services and Chirp.Web. The outermost layer contains the frontend Razor Pages and the UITests.

3.3 Architecture of deployed application



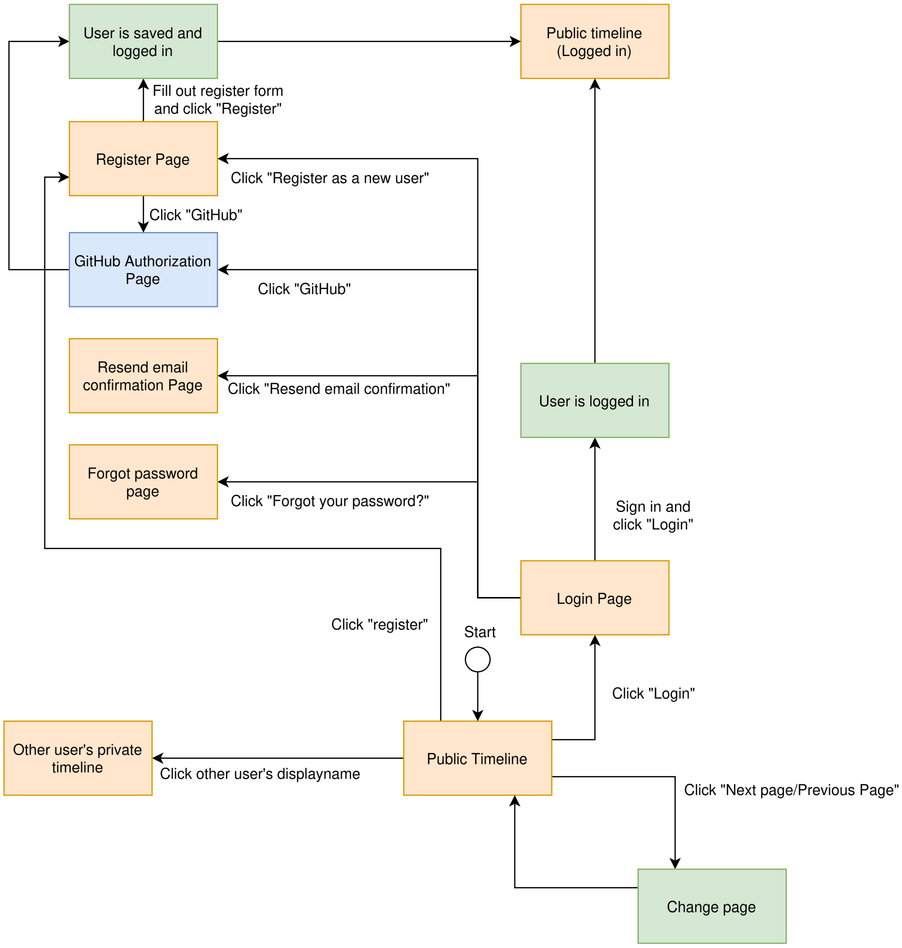
The *Chirp!* application is hosted on Azure as an App Service. The Chirp.Web package exposes access to the application through Razor Pages. Whenever a client wants to access the app, they connect through HTTPS to Chirp.Web. When the client opens the app, the Chirp.Web makes a call to Chirp.Infrastructure which acceses the SQLite database. If the client chooses to, they can register an account with GitHub through OAuth in which case GitHub handles this request.

3.4 User activities

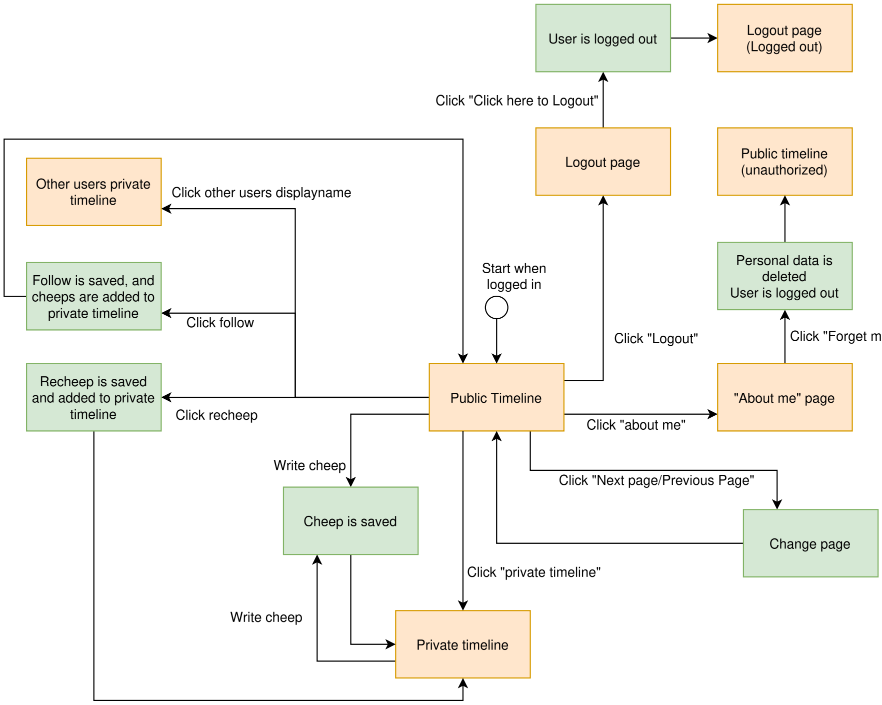
Unregistered users start on the public timeline and can either register or login to become an authorized user. They can also view the cheeps on the public timeline, change page, and view other users' private timeline, by clicking on their names. Once authorized you can do the same as an unauthorized user, but in a addition they can post new cheeps, follow other users, or recheep their cheeps. They can also view their information under "about me", and in there they can also use the "Forget me!" feature to delete all personal information about themselves.

Below are two activity diagrams, about authorized and unauthorized users. The internal pages are orange boxes, actions are green boxes, and external pages are blue boxes.

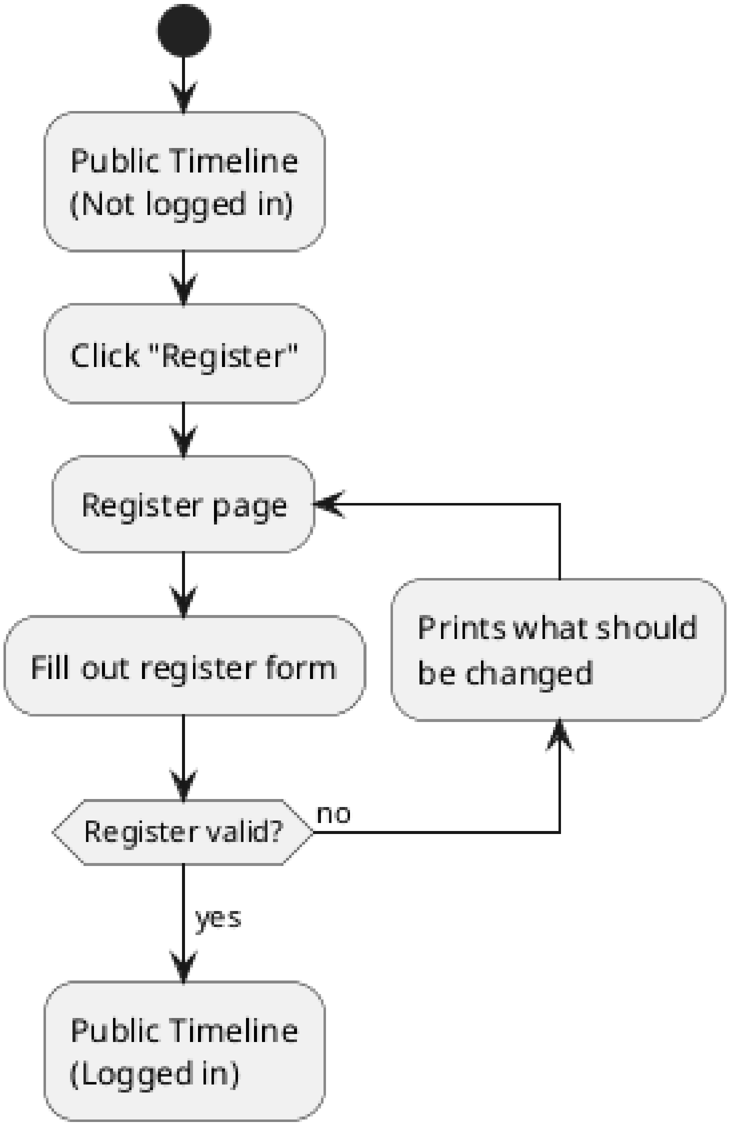
Activity diagram for unauthorized user:


User is saved and logged in
Public timeline (Logged in)
Fill out register form and click "Register"
Register Page
Click "Register as a new user"
Click "GitHub"
GitHub Authorization Page
Click "GitHub"
Resend email confirmation Page
Click "Resend email confirmation"
User is logged in
Forgot password page
Click "Forgot your password?"
Sign in and click "Login"
Login Page
Click "register"
Start
Click "Login"
Other user's private timeline
Click other user's displayname
Public Timeline
Click "Next page/Previous Page"
Change page

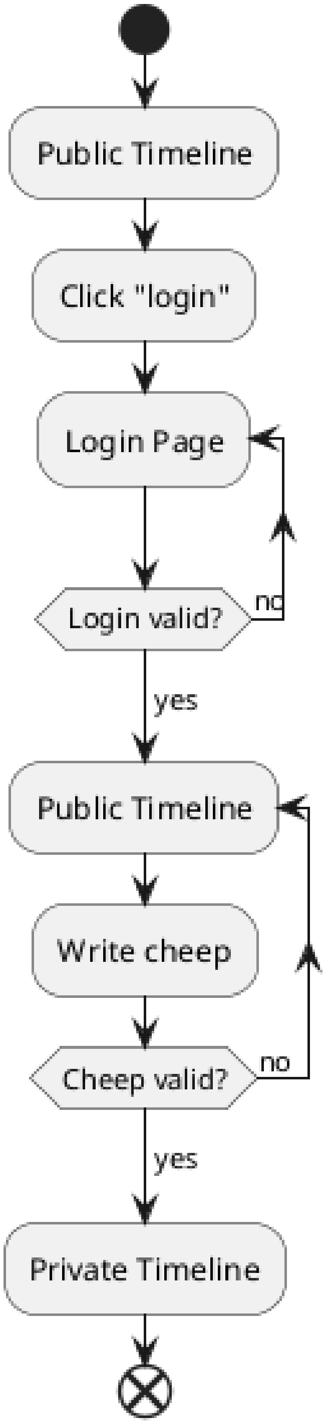
Activity diagram for authorized user:


User is logged out
Logout page (Logged out)
Click "Click here to Logout"
Logout page
Public timeline (unauthorized)
Other users private timeline
Click other users displayname
Personal data is deleted User is logged out
Start when logged in
Follow is saved, and cheeps are added to private timeline
Click follow
Click "Logout"
Click "Forget me!"
Recheep is saved and added to private timeline
Click recheep
Public Timeline
"About me" page
Click "about me"
Write cheep
Click "Next page/Previous Page"
Cheep is saved
Change page
Click "private timeline"
Write cheep
Private timeline

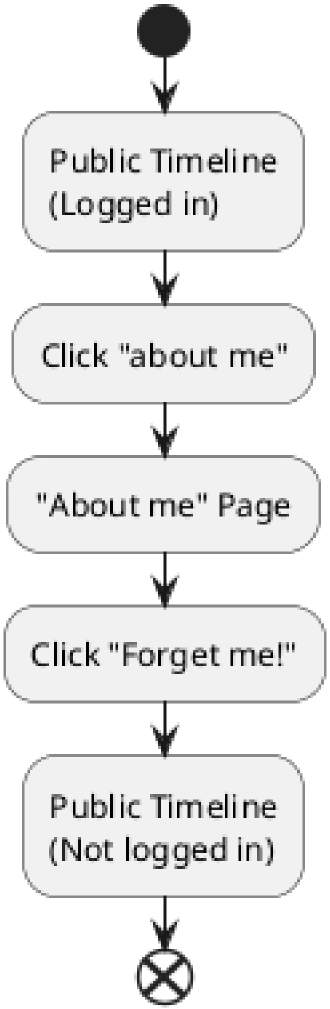
Here are three user journeys representing typical user experiences on our site.
User registering for the site:



User logging in and writing a cheep:

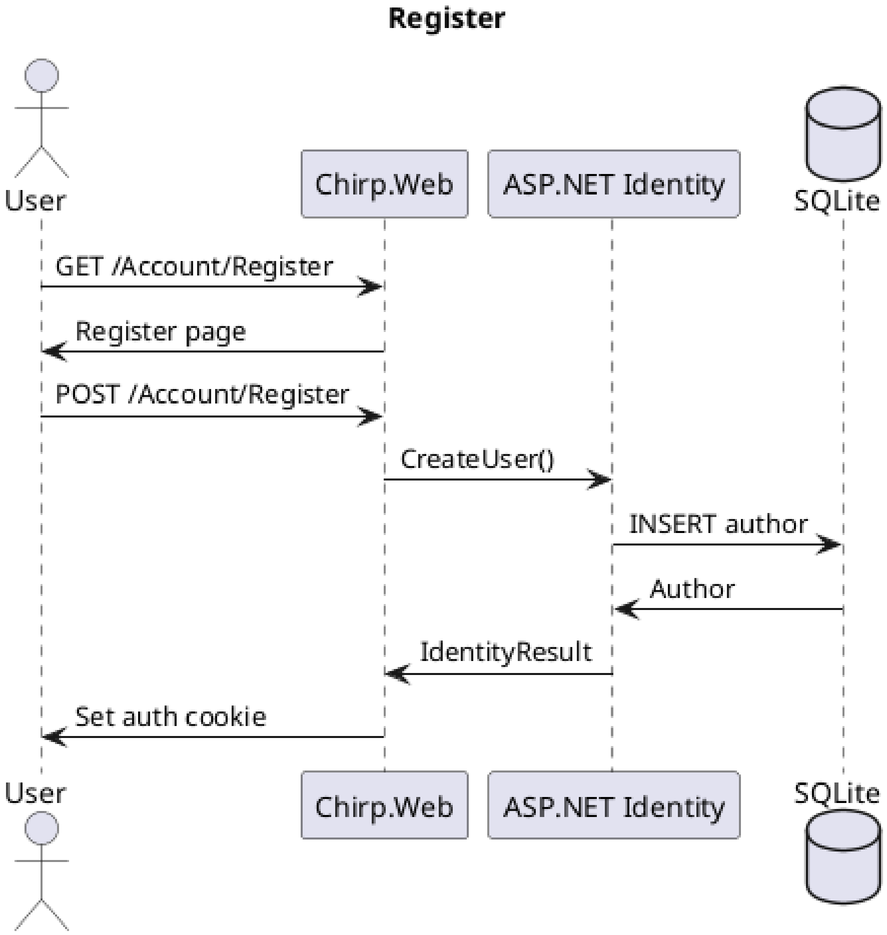

Public Timeline
Click "login"
Login Page
Login valid?
no
yes
Public Timeline
Write cheep
Cheep valid?
no
yes
Private Timeline

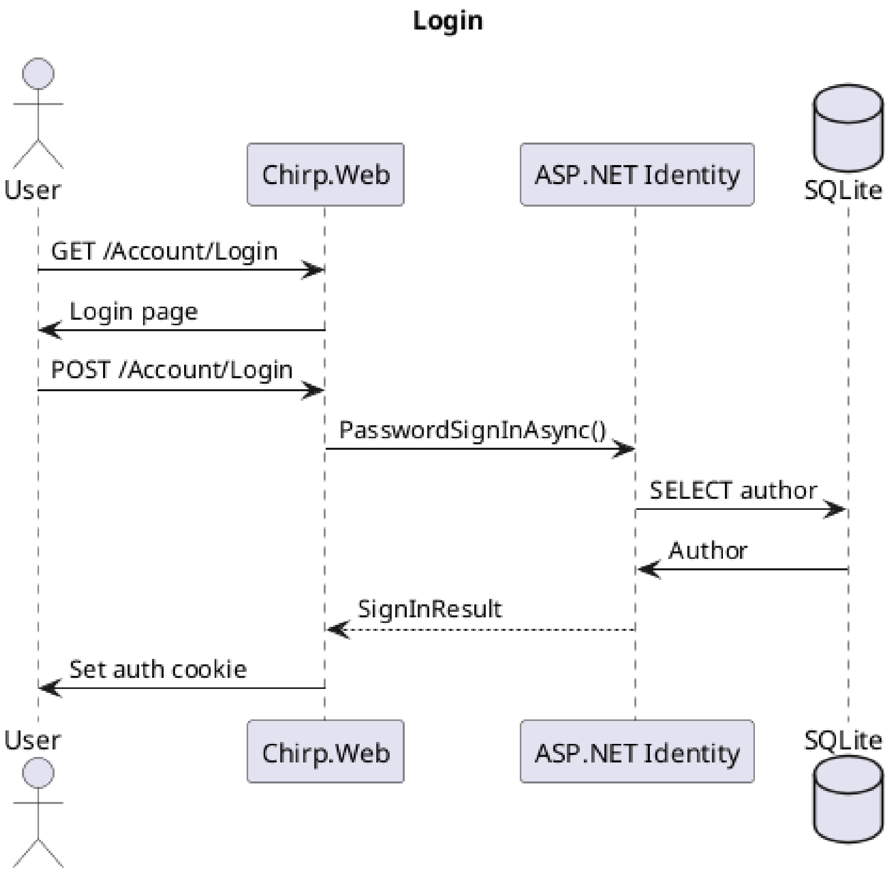
User using the “Forget me!” feature to delete all data about them:

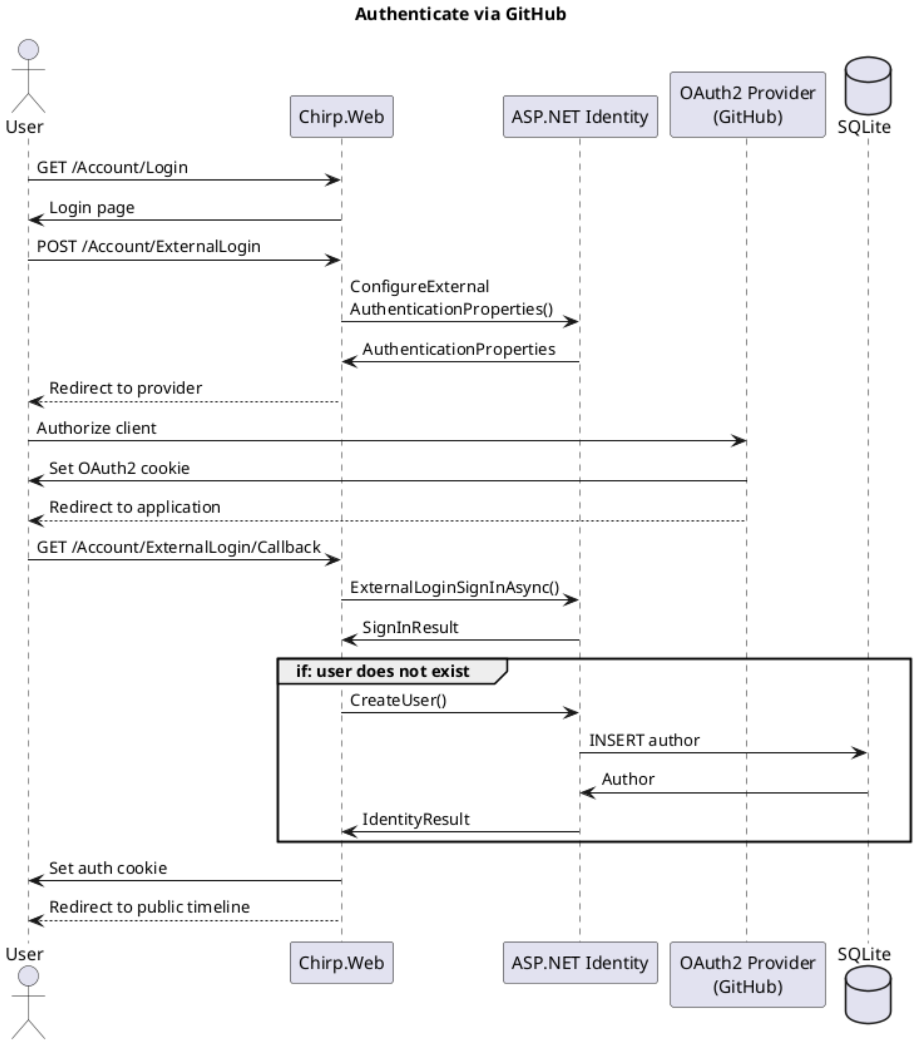


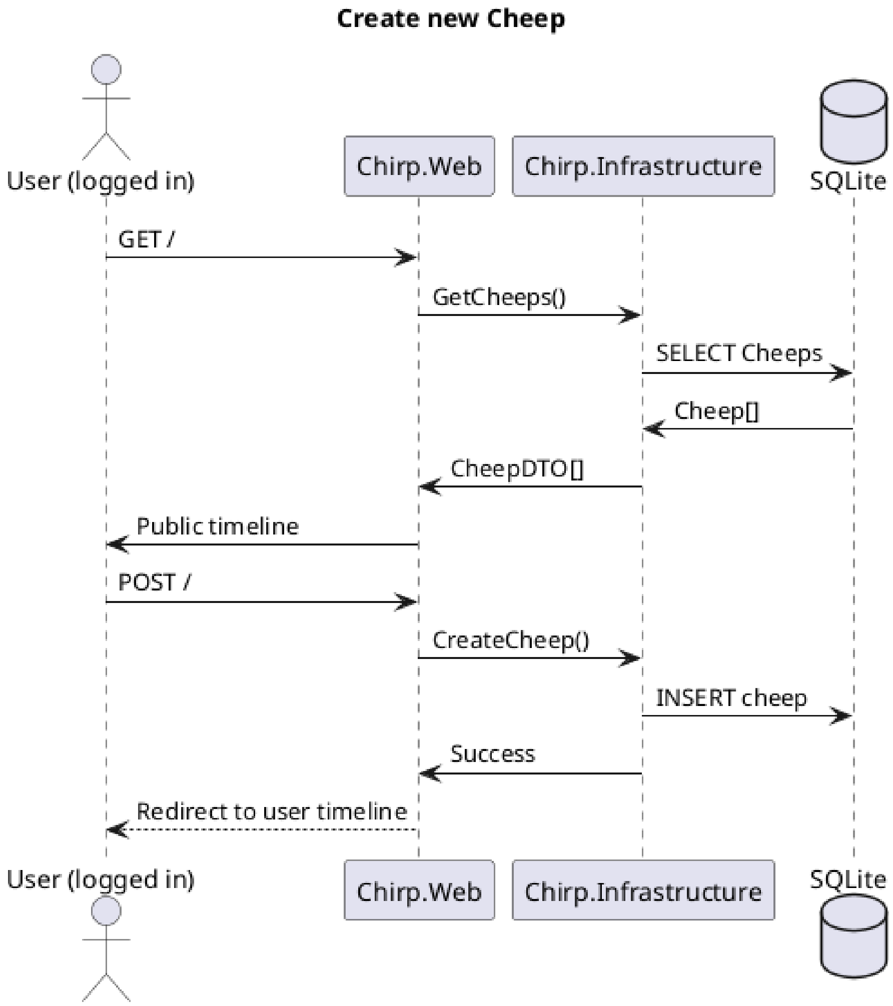
3.5 Sequence of functionality/calls trough *Chirp!*

To illustrate the interworkings of the *Chirp!* application, Sequence Diagrams of common operations are provided here.


Register
User
Chirp.Web
ASP.NET Identity
SQLite
GET /Account/Register
Register page
POST /Account/Register
CreateUser()
INSERT author
Author
IdentityResult
Set auth cookie
User
Chirp.Web
ASP.NET Identity
SQLite


Login
User
Chirp.Web
ASP.NET Identity
SQLite
GET /Account/Login
Login page
POST /Account/Login
PasswordSignInAsync()
SELECT author
Author
SignInResult
Set auth cookie
User
Chirp.Web
ASP.NET Identity
SQLite


Authenticate via GitHub
User
Chirp.Web
ASP.NET Identity
OAuth2 Provider (GitHub)
SQLite
GET /Account/Login
Login page
POST /Account/ExternalLogin
ConfigureExternal AuthenticationProperties()
AuthenticationProperties
Redirect to provider
Authorize client
Set OAuth2 cookie
Redirect to application
GET /Account/ExternalLogin/Callback
ExternalLoginSignInAsync()
SignInResult
if: user does not exist
CreateUser()
INSERT author
Author
IdentityResult
Set auth cookie
Redirect to public timeline
User
Chirp.Web
ASP.NET Identity
OAuth2 Provider (GitHub)
SQLite



3.6 Design decisions

3.6.1 Self-contained releases

During the development of Chirp, a decision was made to make releases self-contained. It was not a requirement to have self-contained releases, because it was assumed that all interested users can use the application with .NET 7.0. This is not the case for this project, since it uses .NET 8.0. Therefore, it is important for the releases to be self-contained.

In general, it has been decided that emails are unique which means two users with different usernames still cannot have the same email. We have made the decision to delete the user with the username “Helge” from the production database so that he can register his github account which uses the same email.

3.6.2 Onion Architecture

In session 7 of the course, requirement 1.f describes an implementation of an Onion Architecture. We have chosen not to follow the described interpretation rigorously, because we believe it violates the principles of the Onion Architecture pattern. Instead, we have chosen to have our *Core* contain the Data Model for our project. This was done, so our *Core* does not depend on any of the other projects. Ideally, the *Core* would have no dependencies, and represent only the idea of our data structure. Our *Author* class needs to depend on *Identity*, to implement authorization, though.

In our *Infrastructure* layer, we have defined DTOs, repositories and repository interfaces. These are defined here, because they are not directly related to the data model, but to how the data is stored in a database. Both of these depend on the *Core*. If the DTOs/repository interfaces had been in *Core*, and the Model been in *Infrastructure*, the *Core* would depend on an outer layer, which would have been unacceptable. We also have Services defined here, that depend on the repositories and DTOs.

Our *Web* project depends on the inner layers.

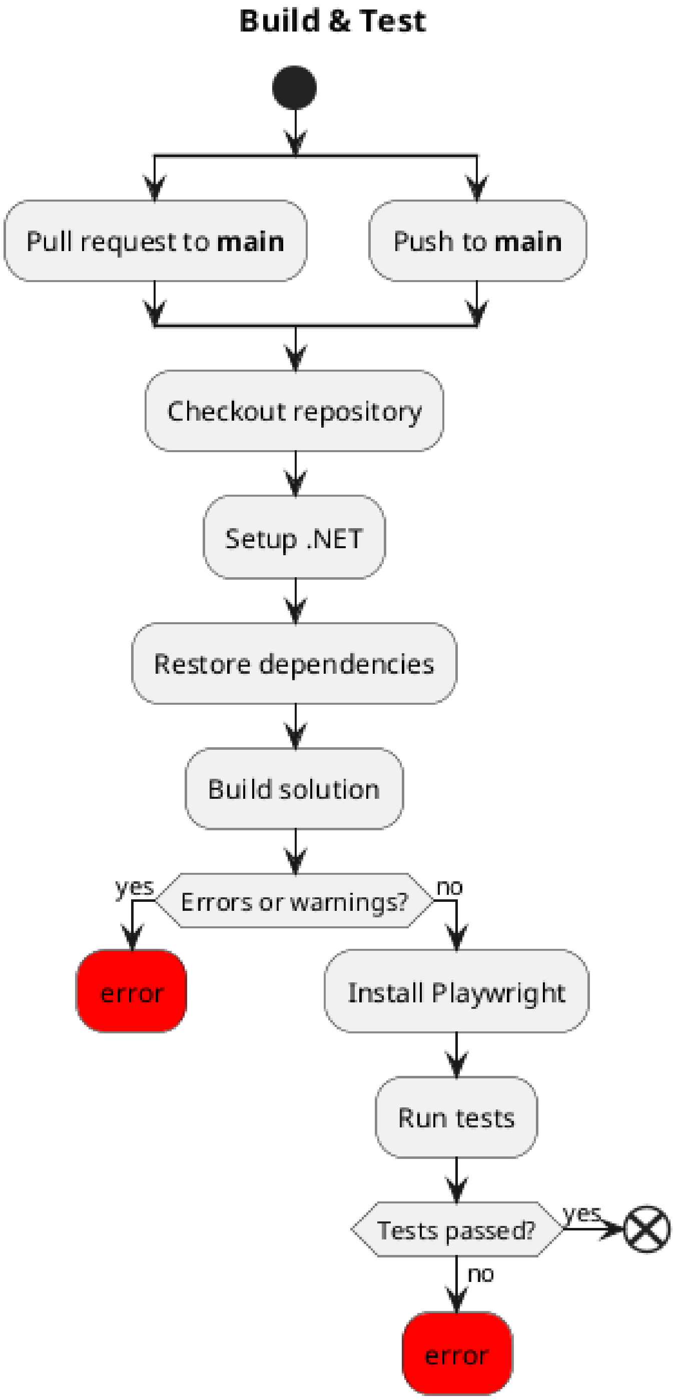
This enforces the idea of antisymmetric dependencies in the Onion Architecture; all layers in the system only depend on the layers that are deeper in the architecture. We could also have chosen to have our repository interfaces in the *Core*, but we did not.

4 Process

4.1 Build, test, release, and deployment

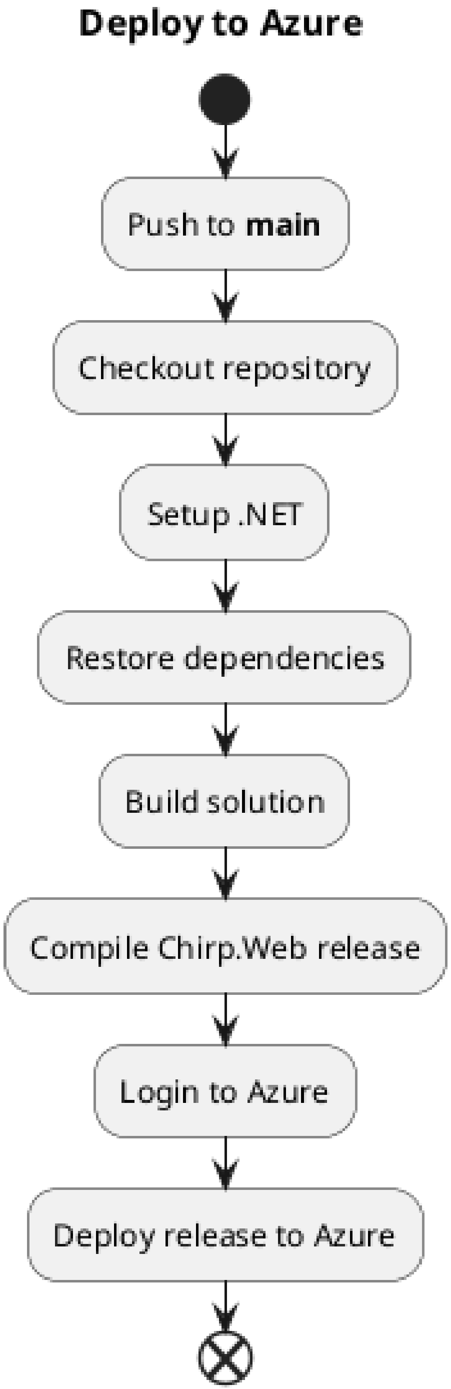
Build & Test

The `build_and_test.yml` workflow triggers on changes to main and ensures that all tests pass and that no warnings can enter the code base.


Build & Test
Pull request to main
Push to main
Checkout repository
Setup .NET
Restore dependencies
Build solution
Errors or warnings?
yes
no
error
Install Playwright
Run tests
Tests passed?
yes
no
error

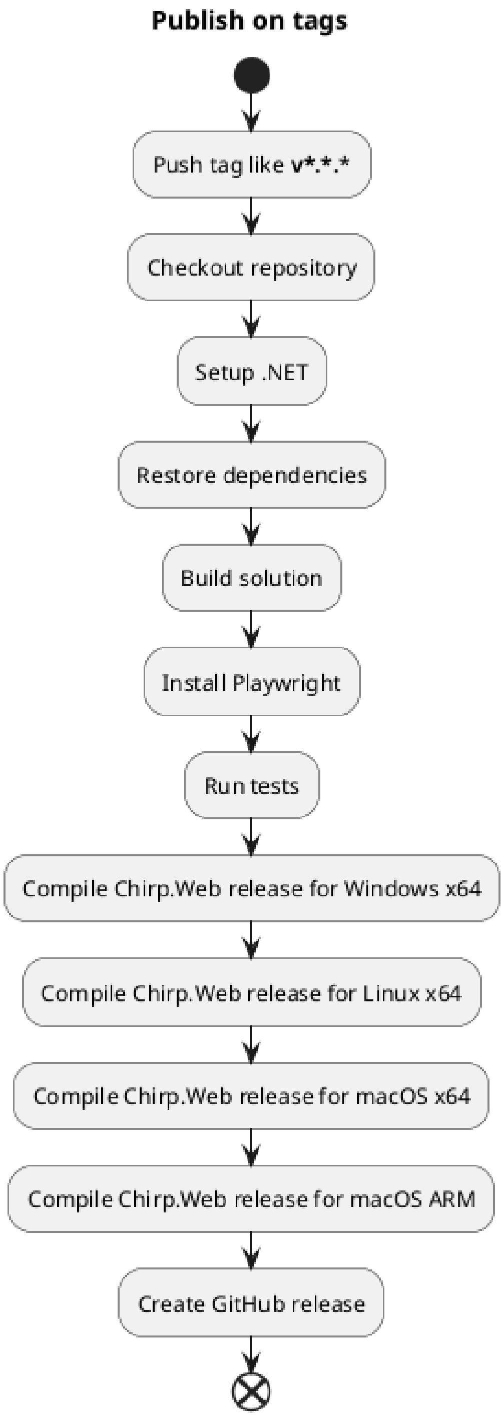
Deploy to Azure

The `deploytoazure.yml` workflow publishes the `Chirp.Web` Razor application to the Azure App Service to ensure that our production environment is always up to date.


Deploy to Azure
Push to **main**
Checkout repository
Setup .NET
Restore dependencies
Build solution
Compile Chirp.Web release
Login to Azure
Deploy release to Azure

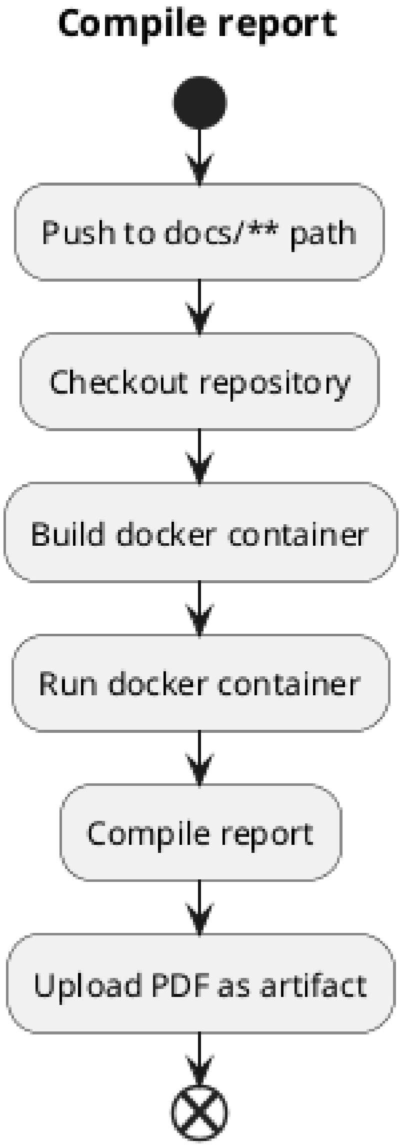
Publish on tags

The `publish_on_tags.yml` workflow creates a GitHub release containing the compiled binaries for various systems.

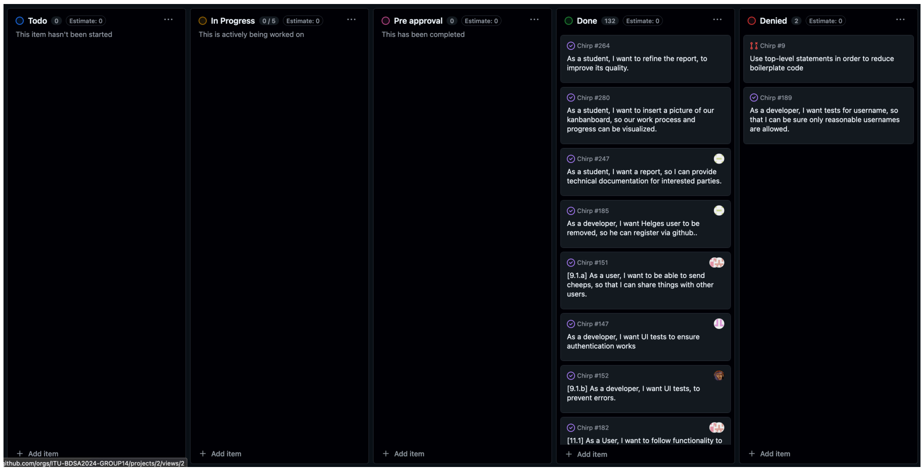

Publish on tags
Push tag like **v*.*.***
Checkout repository
Setup .NET
Restore dependencies
Build solution
Install Playwright
Run tests
Compile Chirp.Web release for Windows x64
Compile Chirp.Web release for Linux x64
Compile Chirp.Web release for macOS x64
Compile Chirp.Web release for macOS ARM
Create GitHub release

Compile report

The `compile_report.yml` workflow compiles the report and included PlantUML diagrams as PDF.



4.2 Team work



Todo 0 Estimate: 0

This item hasn't been started

+ Add item

In Progress 0 / 5 Estimate: 0

This is actively being worked on

+ Add item

Pre approval 0 Estimate: 0

This has been completed

+ Add item

Done 132 Estimate: 0

Chirp #264
As a student, I want to refine the report, to improve its quality.

Chirp #280
As a student, I want to insert a picture of our kanbanboard, so our work process and progress can be visualized.

Chirp #247
As a student, I want a report, so I can provide technical documentation for interested parties.

Chirp #185
As a developer, I want Helges user to be removed, so he can register via github..

Chirp #151
[9.1.a] As a user, I want to be able to send cheeps, so that I can share things with other users.

Chirp #147
As a developer, I want UI tests to ensure authentication works

Chirp #152
[9.1.b] As a developer, I want UI tests, to prevent errors.

Chirp #182
[11.1] As a User, I want to follow functionality to

+ Add item

Denied 2 Estimate: 0

Chirp #9
Use top-level statements in order to reduce boilerplate code

Chirp #189
As a developer, I want tests for username, so that I can be sure only reasonable usernames are allowed.

+ Add item

github.com/orgs/ITU-BDSA2024-GROUP14/projects/2/views/2

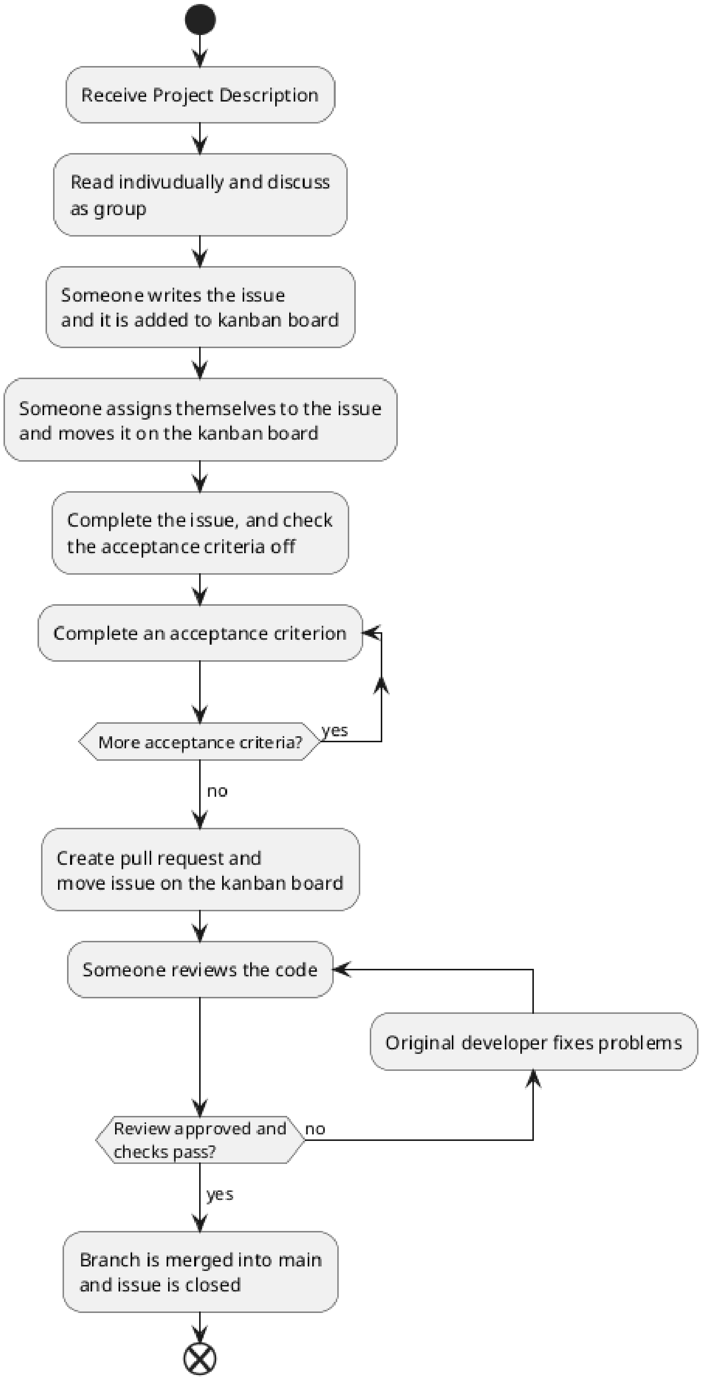
The screenshot above shows our project board at its final state. All required features have been implemented.

Below is a flowchart illustrating our workflow after a new project description is published. When the project description comes out, we read it together individually and discuss it as a group, to get a rough idea of how we want to tackle the problems. Then the description is made into GitHub issues, with user stories and acceptance criteria, and added to the GitHub kanban board.

Later someone will assign themselves to the issue, create a branch and move the issue to "In Progress" on the kanban board. They will complete the issue and check off the acceptance criteria. When the issue is complete, and all tests pass, they will move it to "Pre approval" on the kanban board, and create a pull request.

Another developer will review the pull request, and either suggest changes, in which case the original developer fixes the problems, and requests a re-review.

When it is approved in review and all tests pass, the branch is merged into main and the issue is moved to "Done" on the kanban board.



Development workflow

Receive Project Description
Read indivudually and discuss as group
Someone writes the issue and it is added to kanban board
Someone assigns themselves to the issue and moves it on the kanban board
Complete the issue, and check the acceptance criteria off
Complete an acceptance criterion
More acceptance criteria?
yes
no
Create pull request and move issue on the kanban board
Someone reviews the code
Original developer fixes problems
Review approved and checks pass?
no
yes
Branch is merged into main and issue is closed

4.3 How to make *Chirp!* work locally

To run the project you need the following programs

- .NET 8
  - *How to install .NET*
- Git CLI

Run the following commands

```
git clone https://github.com/ITU-BDSA2024-GROUP14/Chirp.git
```

After cloning the project go into *Chirp.Web* project

```
cd ./Chirp/src/Chirp.Web
```

Run the following command to set up user secrets for running the program locally

```
dotnet user-secrets set "authentication:github:clientId" "Ov23liOEFAiXHOnNGkH3"
dotnet user-secrets set "authentication:github:clientSecret" /
"9cc3aae28d9e5fdfe27f42158842f92687964382"
```

Now run `dotnet run`. The program is now running locally, go to http://localhost:5273 to interact with it[1]

4.4 How to run test suite locally

4.4.1 How to run test suite

To run the test suite locally Playwright and .NET 8 needs to be installed. If using another OS than Windows, make sure PowerShell is installed, so you can install Playwright.

- *How to install PowerShell*
- *How to install .NET*

After installing the required software, run the following command from PowerShell, in the root of the project.

```
dotnet build
./test/UITests/bin/Debug/net8.0/playwright.ps1
```

If running the commands from another terminal than PowerShell, run `pwsh ./test/UITests/bin/Debug/net8.0/playwright.ps1` instead of the second line.

After the above listed programs are installed on your computer, run the following command in the project root to run all tests

```
dotnet test
```

[1] Pushing secrets like these to a public github repo is a bad idea. They are added here to make it easier to run the program, without having to create your own Github OAuth token, and the program *Chirp!* is not important enough for this to be a problem.

4.4.2 Tests in *Chirp!*

To ensure requirements, prevent bugs, and that new code does not break old code, the project contains Unit tests, Integration tests, and End-to-End tests.

These tests can be found in the following folders

```
Project root
|-- test
    |-- Chirp.Infrastructure.Tests
    |-- IntegrationTests
    |-- UITests
```

4.4.2.1 Chirp.Infrastructure.Tests The *Infrastructure test* project contains unit tests for

- CheepRepository
- ChirpService
- AuthorService

These tests cover all methods located in their respective classes.

4.4.2.2 IntegrationTests The *integration test* project contains integration tests, using HTTP requests to test if the website contains implemented features in the HTML code.

4.4.2.3 UITests In the *UITests* project Playwright has been used to create End to End tests for the project. These test the UI, and features that are relient on being logged in.

5 Ethics

5.1 License

When deciding which license to use the most important consideration is whether any GPL libraries are used in the project, since the licence then must be GPL. None of the libraries used has the GPL license, or any other copyleft license. Therefore the choice of license was left open, and the MIT license was chosen. The MIT license is one of the most permissive licenses, allowing the program to be used for almost anything as long as the original copyright notice and license are included. This also means the program can be used as is, and the developers have no responsibility for maintaining the product. As discussed in the open source lecture, there are many advantages with using open source as your license. Additionally, since this is an educational project, it makes sense to both be as open source as possible, and to not take responsibility for maintaining the code longterm.

5.2 LLMs, ChatGPT, CoPilot, and others

In the development process LLM were used sparingly to support the coding process. Riders Line Completion were occasionally used to finish lines of code, when it came with good suggestions. This runs locally and does not communicate over the internet, probably making them use less power compared ChatGPT or similar LLM's. It assisted making templates for documentation, so it was easy to fill out, and wrote some of the setup code for some of the simpler tests. ChatGPT was used occasionally to suggest names, explain error messages, and other similar uses.

Often the answers were wrong or irrelevant, especially regarding ChatGPT, however it rarely took long to figure out whether the answer was useful, so it did not waste much time. It was helpful as support and probably sped up the coding process, but the final product most likely did not change because of it.