Chirp! Project Report

ITU BDSA 2024 Group 22

Alberte Bülow alby@itu.dk Anna Perlman anyp@itu.dk
Karoline Petersson kamp@itu.dk Olivia Brophy brop@itu.dk
Stina Knudsen stik@itu.dk

1 Design and Architecture of *Chirp!*

This is our report for the course Analysis, Design and Software Architecture in the Autumn semester of 2024. Throughout, the report we will go into detail of how the different elements were implemented and the general structure of our Chirp! application.

1.1 Domain model

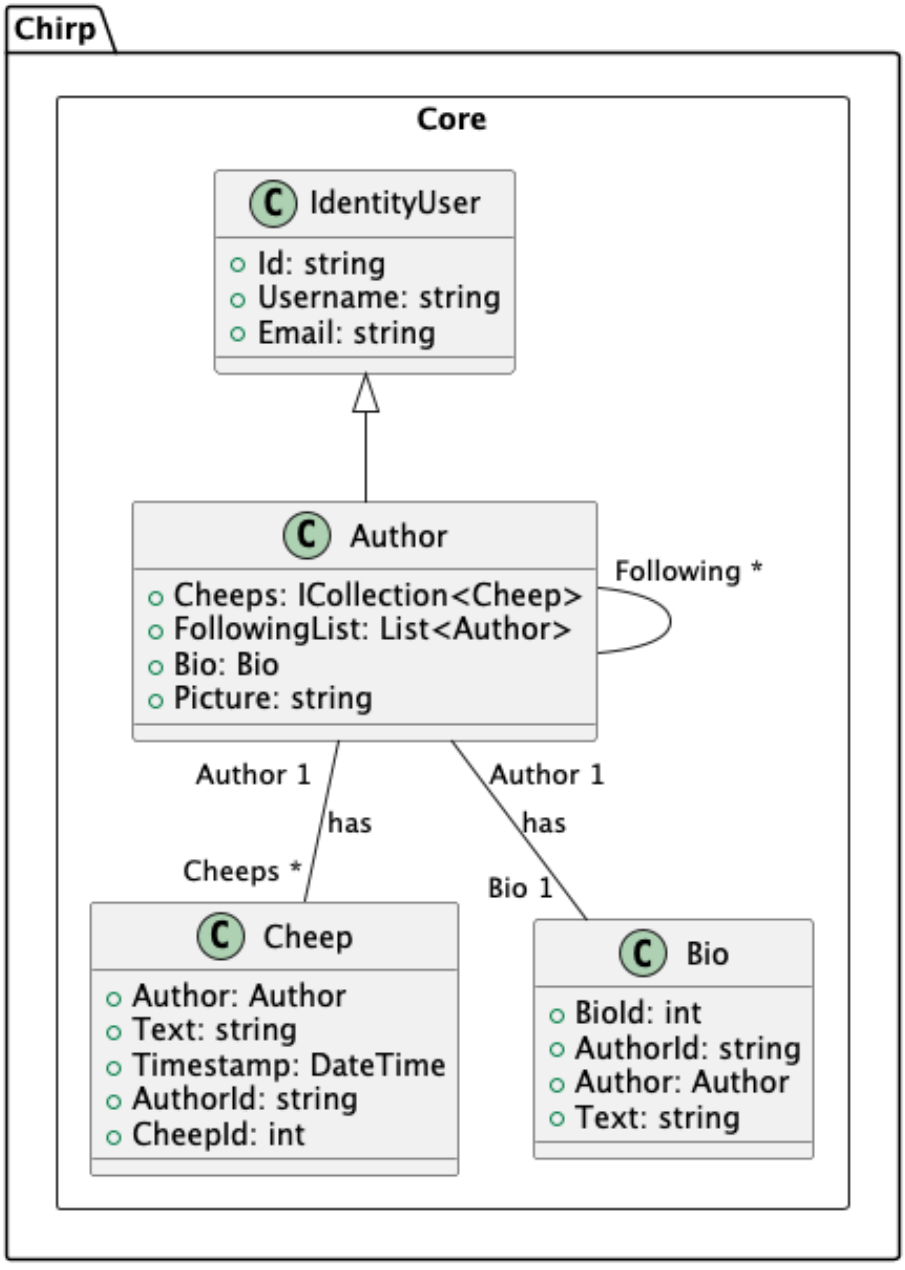


Figure 1: An overview of the domain model

The UML diagram above illustrates the domain model. It depicts the relations and cardinalities between the entities. The Author entity inherits from the

ASP.NET Identity's IdentityUser providing the application with key components regarding the Author-related data. As additional features to the project requirements, both a Bio and the Author's ability to have a (profile) picture were implemented. As shown in the figure, the Bio entity was implemented somewhat like the Cheep entity, in the sense that the entity refers back to an Author. However, unlike Cheep, each Author can only have a single Bio. The profile picture is implemented as a reference to an uploaded image stored elsewhere in the application.

1.2 Architecture — In the small

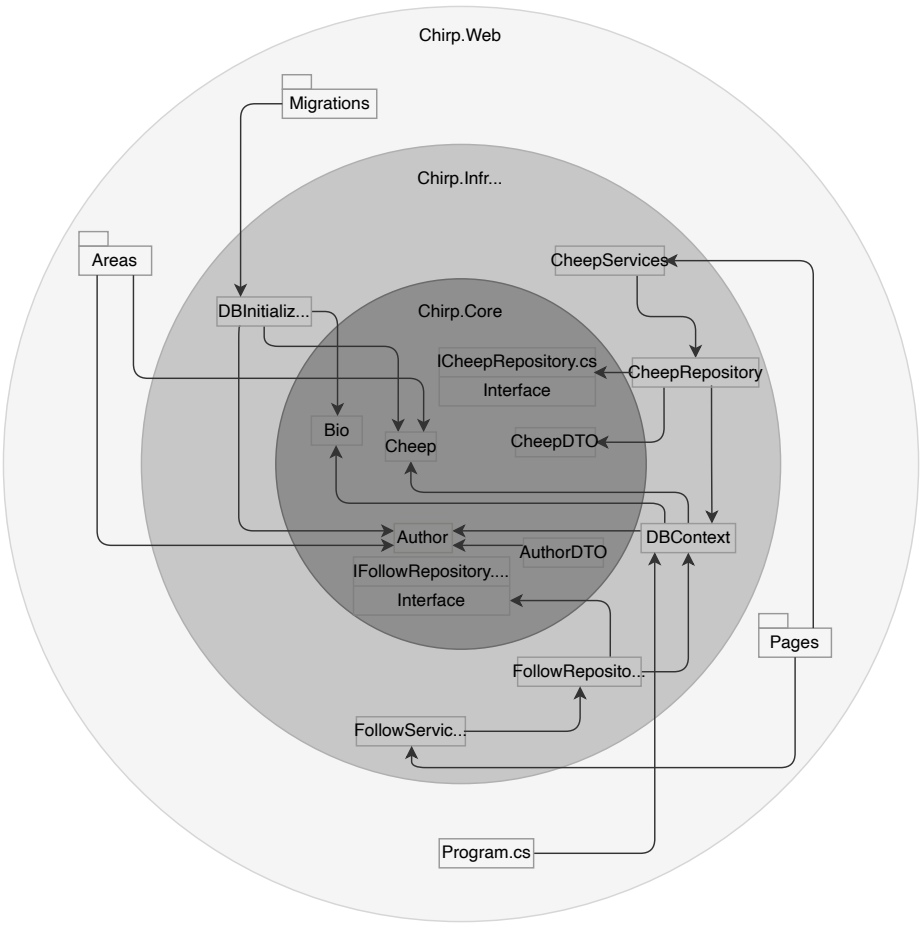


Figure 2: Layers of our architecture

Figure 2 demonstrates our use of the Onion Architecture pattern in our implementation. Each layer shows the different classes, interfaces and packages, each belonging to a layer in our program. Here, the dependencies are shown with arrows, highlighting how the arrows are pointing inwards across the layers,

living up to the criteria of the Onion Architecture pattern. The figure illustrates how the dependencies do not flow outward, and how the inner layers have no knowledge of the outer layers. We are aware that our database does not follow the rules of the Onion Architecture pattern, since it resides in Chirp.Web and not Chirp.Core. But due to continuous issues regarding Azure, we prioritised having a working website, hence breaking the pattern.

1.3 Architecture of deployed application

Figure 3 represents a client-server application deployed on Microsoft Azure. Clients send HTTPS requests to the Azure App Service, where Chirp.Web processes user requests and interacts with the database. Data is securely stored and managed in the Azure SQL Database, with optional authentication via OAuth for Github user sign-up and sign-in.

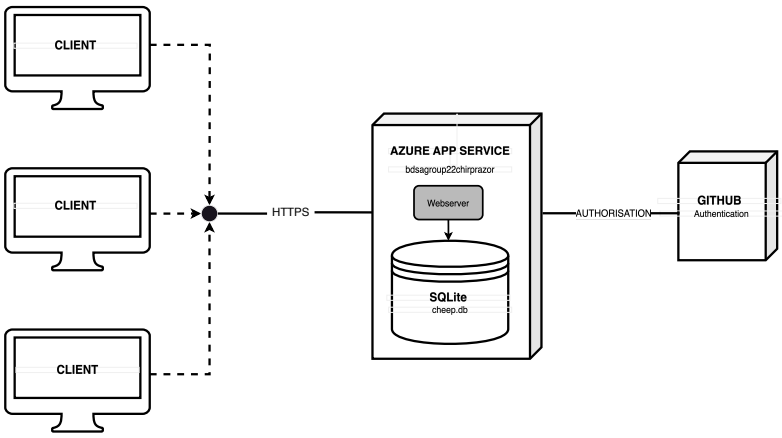


Figure 3: Activity diagram for unauthorised users

1.4 User activities

Chirp is a relatively simple program to navigate through. To illustrate the primary functions of a typical user journey, we have created activity diagrams for both authorised and unauthorised users.

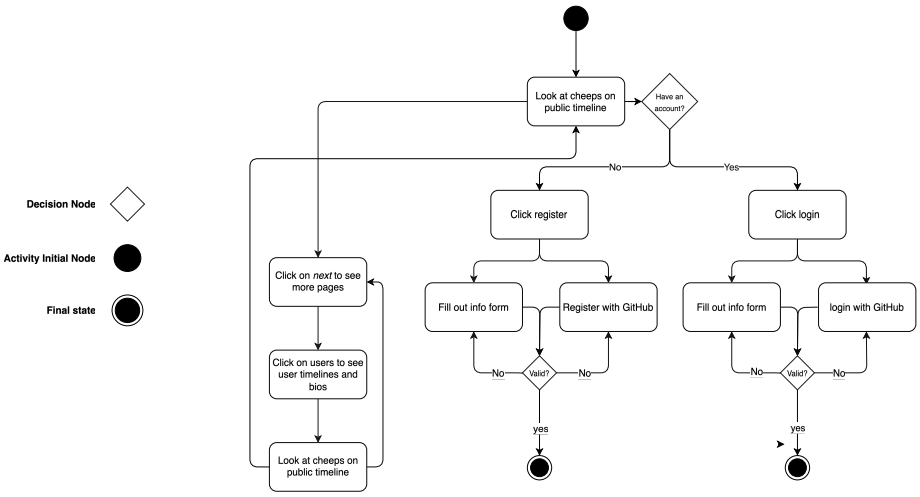


Figure 4: Component diagram illustrating the deployed application

Figure 4 illustrates the flow for unauthorised users interacting with Chirp. Users can browse the public timeline, view more pages, go to other users timelines, or navigate to registration/login options. If a user has no account, they can click register and proceed either by filling out the registration form or using GitHub. Both paths require validation. If the user already has an account, they can click login. Once the user successfully registers or logs in, they are directed to the system's authorised user area.

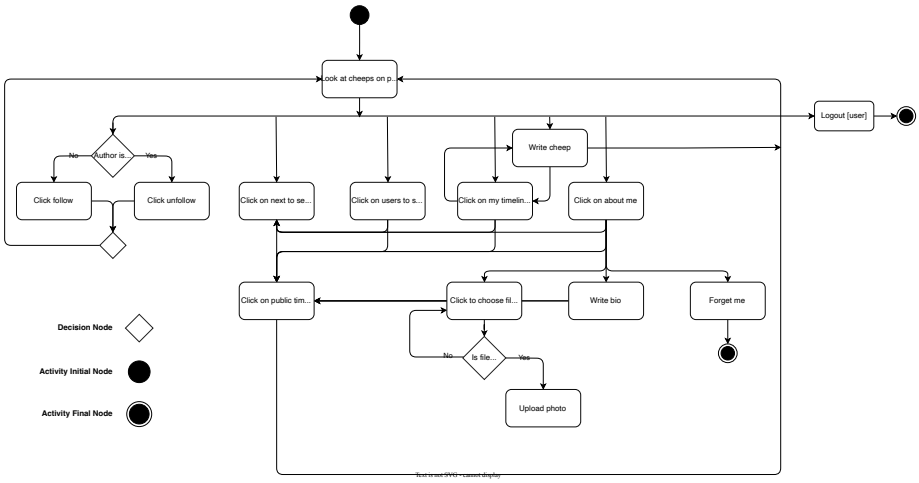


Figure 5: Activity diagram for authorised users

Figure 5 illustrates the flow of actions for an authorised user, starting with viewing cheeps on the public timeline. Users can follow or unfollow other users, navigate to other pages, or view user timelines and bios, depending on their preferences. By accessing "my timeline", users can see their bio, their authored

cheeps, and followed authors' cheeps. Authorised users can update their profiles by choosing a file to upload a profile picture, writing a bio, or clicking "Forget me!" to delete their account and all their data. Users have the option to write new cheeps, interact with personalised or general public content, and log out, completing the interaction cycle.

1.5 Sequence of functionality/calls through *Chirp!*

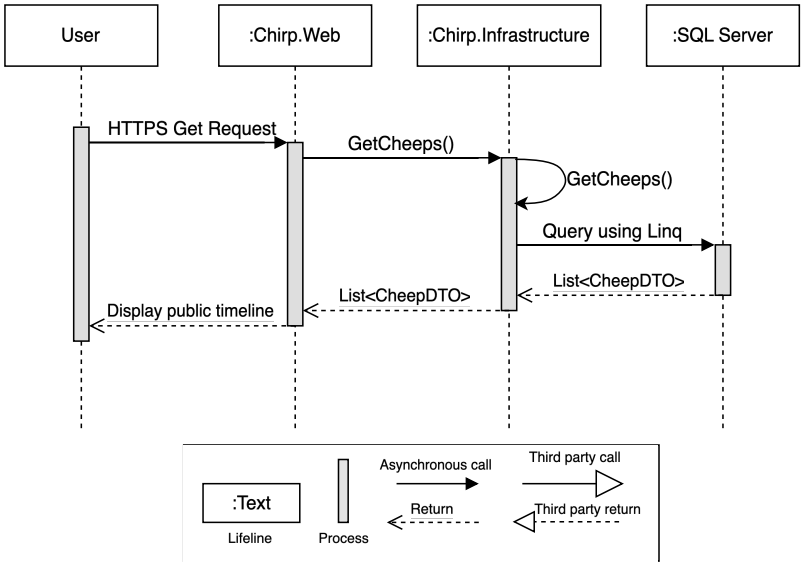
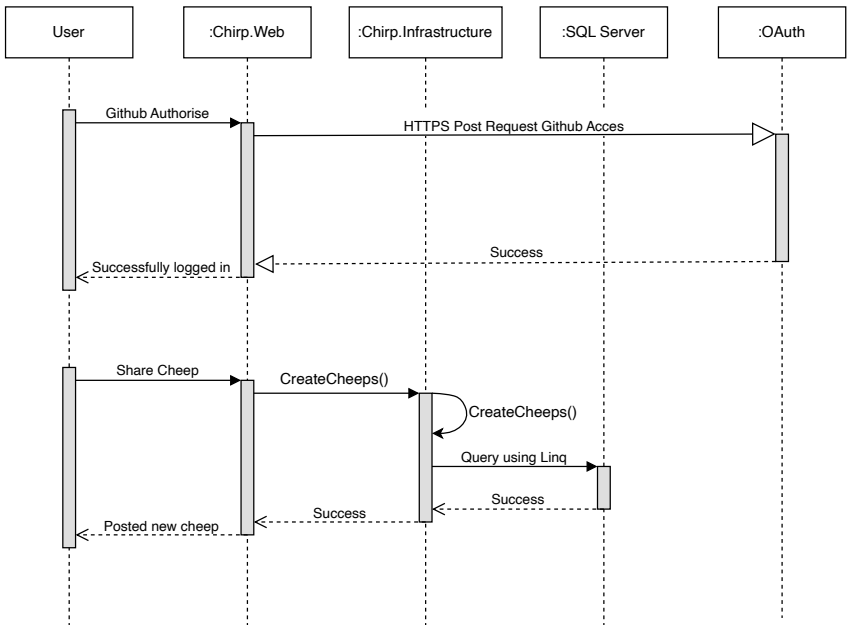


Figure 6: Sequence diagram for an unauthorized user

The sequence diagram above, figure 6, demonstrates the sequence of an unauthorised user accessing our Chirp application. Here the user's request goes through the layers of our implementation, where a list of the 32 newest cheeps are retrieved from the database and returned to the user on the webpage.

Figure 7 below is a sequence diagram of a user logging in on our application and afterwards posting a cheep to the timeline. The request to log in with Github is forwarded to OAuth where the login is handled. The request will return as a success and the user will be redirected to the public timeline. Here, the now authenticated user can post a cheep which will be stored in the database and shown on the public timeline.



Text is not SVG - cannot display

Figure 7: Sequence diagram for authenticated user posting a cheep

2 Process

2.1 Build, test, release, and deployment

2.1.1 Building and testing

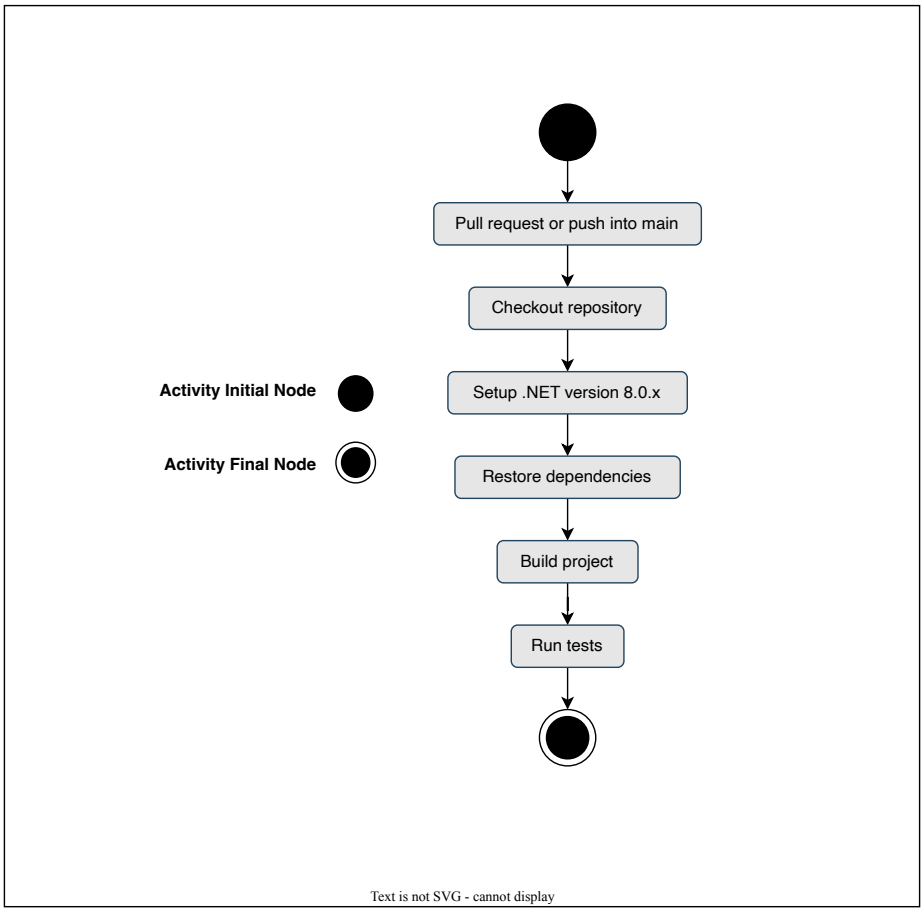


Figure 8: Illustration of the building and testing workflow

When a pull request or a push is made to the main branch in GitHub, a workflow verifying that building and testing the application still runs.The tests running are the tests in Chirp.Tests where both repositories, CheepRepository and FollowRepository, their functionalities are tested, and http responses are tested. It should be noted that the playwright tests are not run in this workflow but on separate testings.

2.1.2 Releasing to GitHub and deploying application

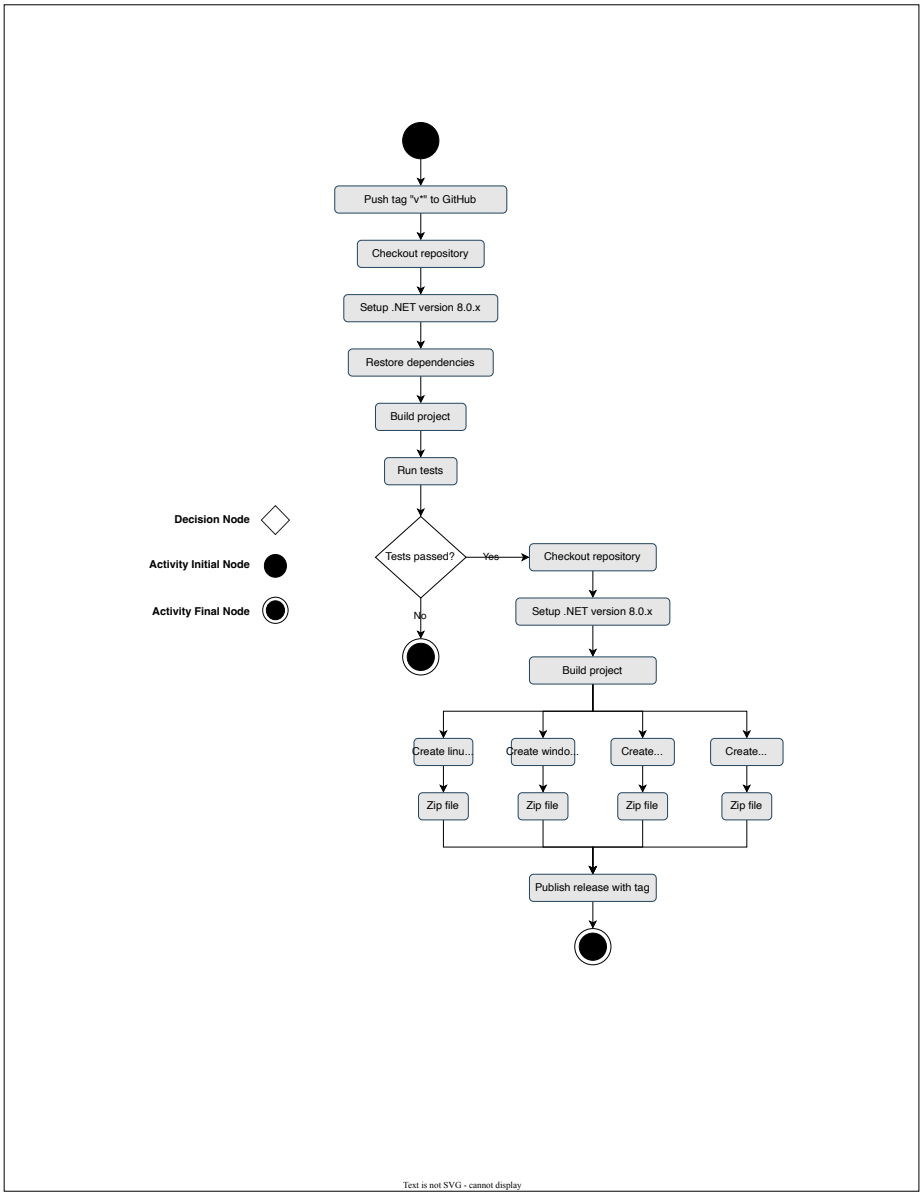


Figure 9: Illustration of the release workflow

When a tag is pushed to GitHub the building and testing workflow will run as described earlier. If the build and test succeeds there will be created a new release via the workflow as seen in figure 9 with the given tag number and the application will be built in files for Windows, Linux, MacOS, and MacOS-arm. After ending this flow the next flow, as seen on figure 10 beneath, starts running

- the deployment flow. Here the program is once again built and all the necessary files for publishing the application are collected in a new directory (an artifact) and uploaded to GitHub to make it available for deployment. The artifact is downloaded and the ready-for-publish files are now available for Azure to deploy the application. This marks the end of the lines of flows.

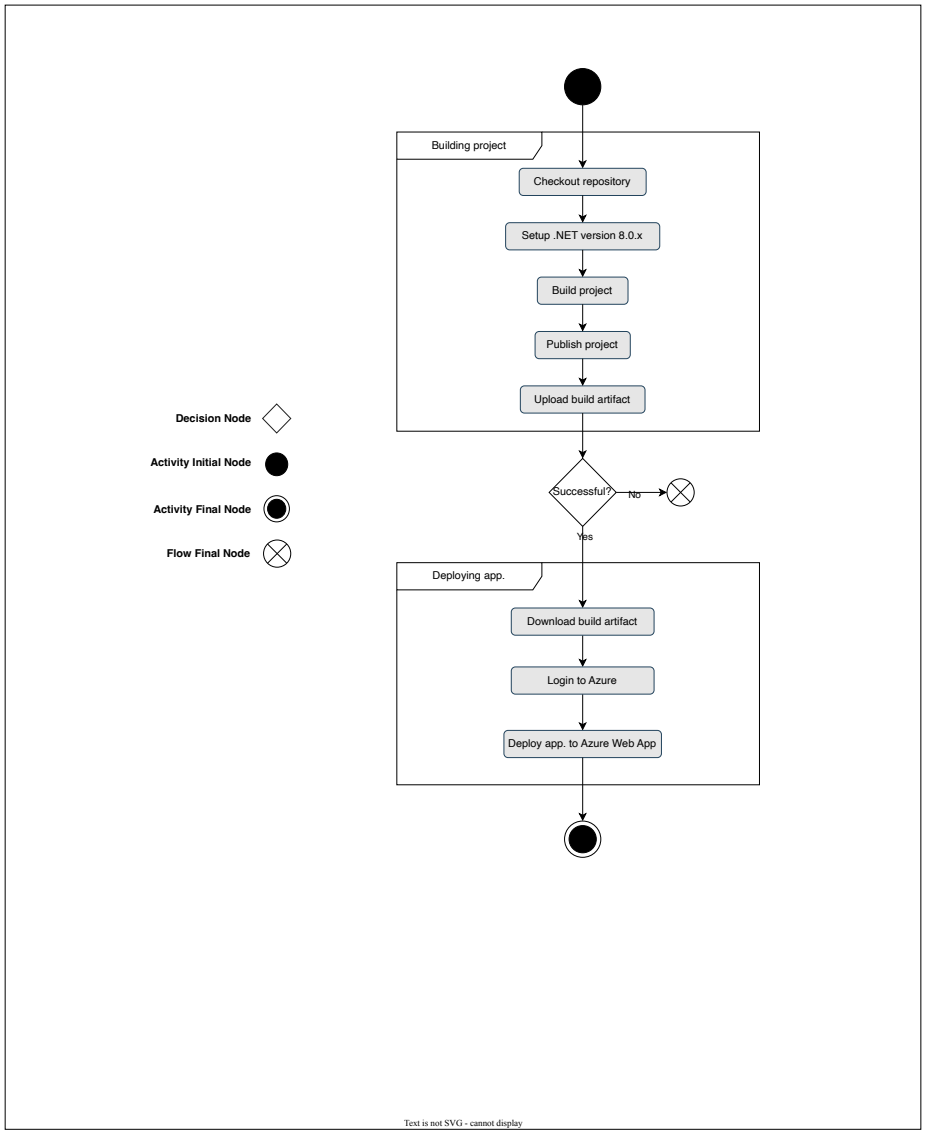


Figure 10: Illustration of the deployment workflow

2.2 Team work

Figure 11 below is an image of our project board right before hand-in. Here, almost all of our tasks are resolved and live up to their respective acceptance criteria. Some tickets regarding Chirp CLI were paused, since we did not have time to resolve them before we switched to Razor Pages. They got placed in "Paused tickets" in case there was time to resolve them later. We managed to fulfill the requirements set throughout the course and the ones that we set ourselves. One part of our program that we would have liked to work more on if we had the time, would be adding more tests, although we believe that our tests still cover most of our program.

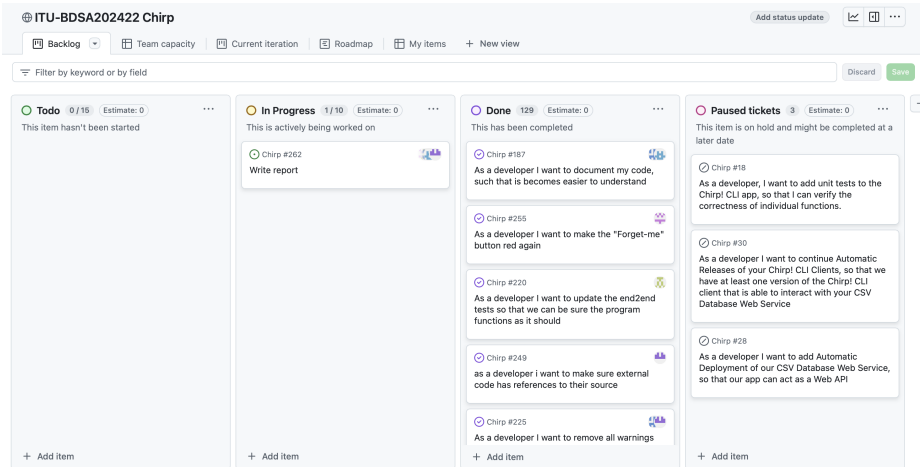


Figure 11: Our project board

During the course we updated our project board constantly and continuously wrote new tickets after each lecture. We then delegated the tickets among the group members and started working on the issue and during the last stretch of the course, we started assigning specific group members to tickets. We worked in many different constellations throughout the project, both doing pair, mob and individual programming depending on the extent of the issue.

Figure 12 shows the workflow for each issue starting with the creation of a new branch and concluding with its deletion, to minimise the number of active branches.

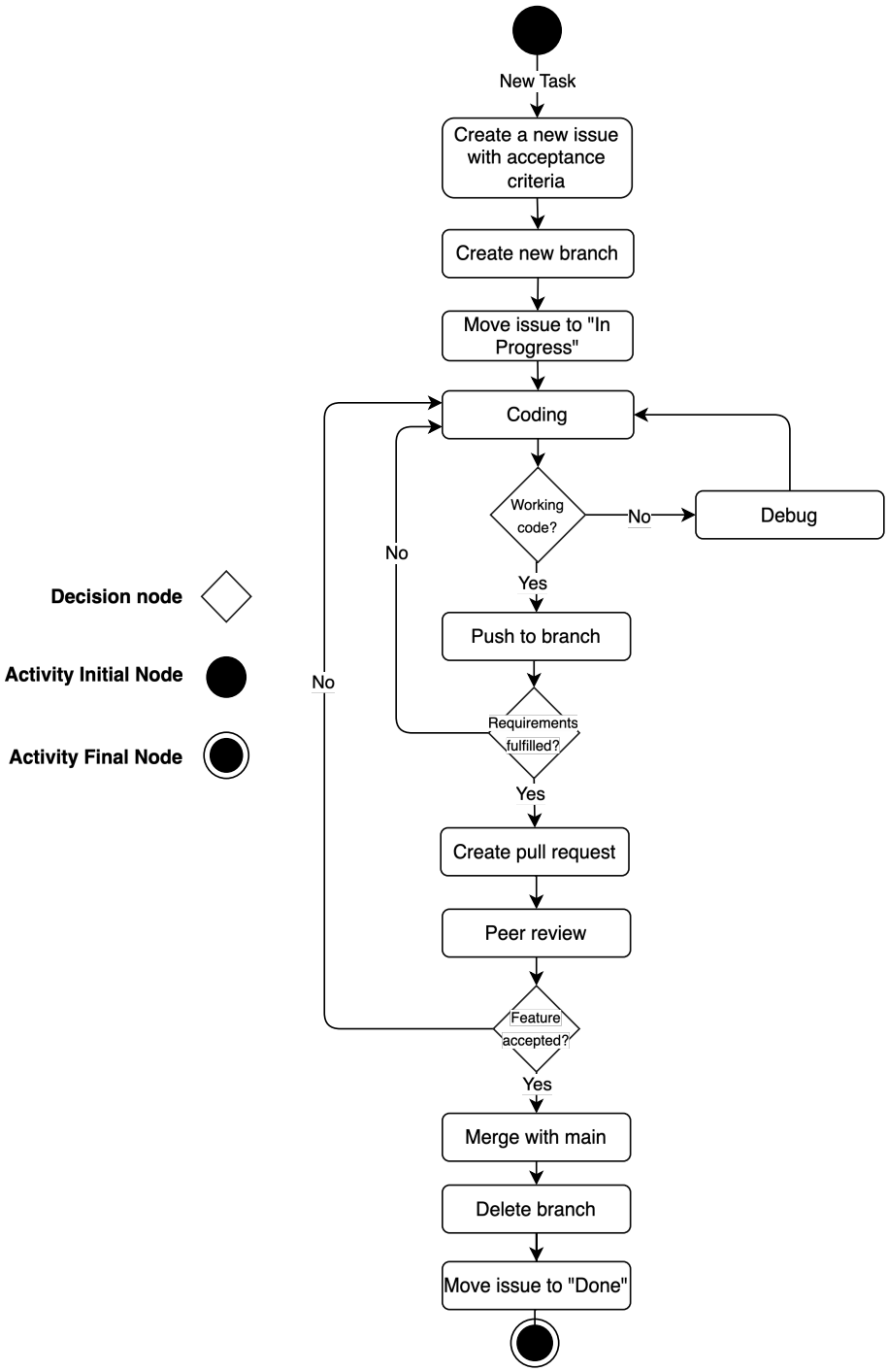


Figure 12: Activity diagram showing work progress for an issue

2.3 How to make *Chirp!* work locally

To run the system, follow these steps.

Step 0: Ensure you have the correct software downloaded

Software needed:

Rider or Visual Studio Code

.NET 8.0

If you don't have .NET 8.0 installed, please use this link to download it:

https://dotnet.microsoft.com/en-us/download/dotnet/8.0

Step 1: Cloning the project

Navigate to the Github repository page for Group 22:

https://github.com/ITU-BDSA2024-GROUP22/Chirp

Press on the green button '<> Code' and copy the HTTPS address

Navigate to the folder where you want to clone the repository:

```
cd path/to/your/folder
```

Use the copied repository URL:

```
git clone <repository-url>
```

Step 2: Open project

Open in Rider or Visual Studio Code. Make sure to be in the root of the directory.

Navigate to Chirp.web

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

Step 3: set user secrets

Run these two commands to set the user secrets locally.

```
dotnet user-secrets set "authentication_github_clientId" "Ov23liZYXvXPxOxqjMap"
dotnet user-secrets set "authentication_github_clientSecret" "ab07248b11a19096e2c822b9660567
```

Step 4: run the program on localhost

```
dotnet run
```

Click on the URL and you will be directed to the web application.

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5273
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /Users/oliviabrophy/Desktop/3. SEMESTER/BDSA/Chirp/src/Chirp.Web
```

Be aware that when logging in, it needs your username and not e-mail.

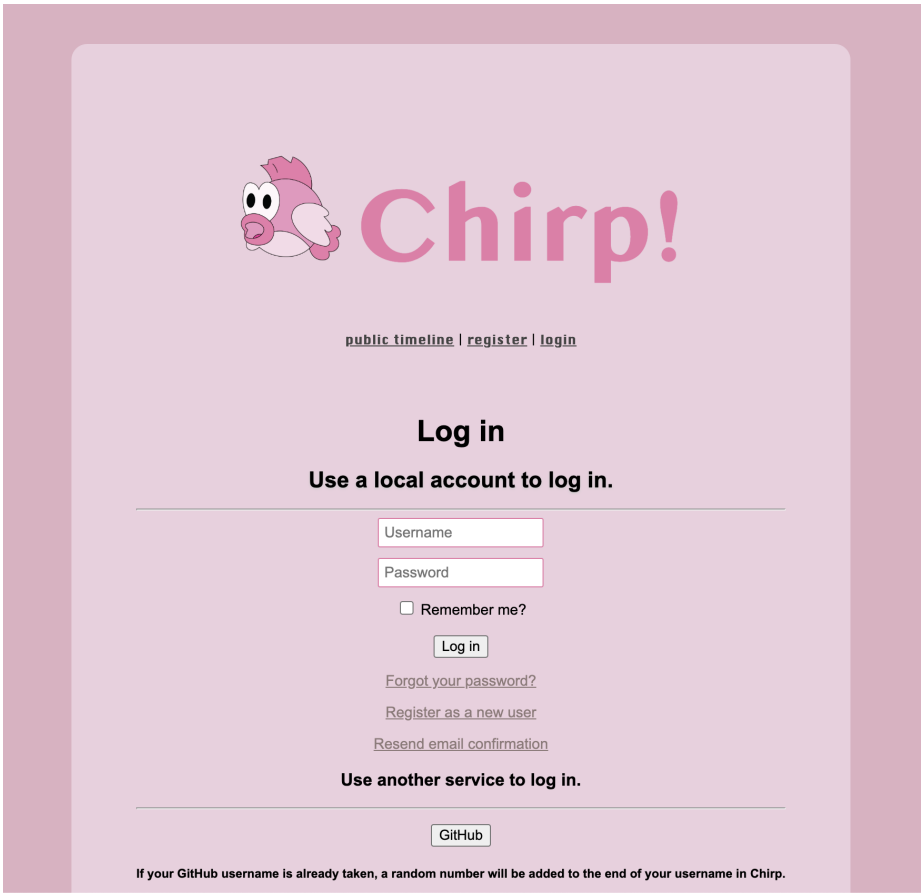


Figure 1: fig##

To shut down the application, press Ctrl+C in the local terminal. ## How to run test suite locally

Step 0: To run test cases from the root of the Chirp Application, run the following commands:

```
cd test/Chirp.Tests
```

Step 1: Test by running the following command:

```
dotnet test
```

Unit tests

The CheepRepositoryTest class tests various methods of the CheepRepository for managing authors and cheeps, including creating authors, retrieving cheeps, updating bios, and deleting authors. It covers scenarios like empty repositories, pagination, non-existent authors, and updating bio details.

The FollowRepositoryTest class tests the functionality of following, unfollowing, and retrieving Cheeps from followed users. It covers scenarios like pagination, handling non-existent users, and preventing duplicate follows.

Integration tests

The tests are set up to verify the application's functionality using an in-memory database, where user profiles, timelines, and Cheeps are created and retrieved. Each test ensures that the relevant data is displayed correctly on the user's profile and timeline. Additionally, tests for bio sections and "About Me" pages are included to ensure correct display of user information.

Playwright tests

Playwright is a framework for testing. We utilised it for end-to-end (E2E) and UI tests. These tests simulate user interactions, such as navigating through pages, inputting data, logging in/out, and clicking buttons, to validate the application's functionality. These tests help identify UI defects and confirm smooth user experiences across all scenarios. To ensure the playwright tests always work, random usernames are generated during each test run, as the usernames are required to be unique.

To run the playwright tests, please follow the instructions below. The instructions assume you are in the root directory of the program.

Step 0: If you don't have playwright installed, make sure to download it by running the following commands. If it is installed you can continue to step 1.

```
npx playwright install
```

Step 1: Start by opening a terminal and run the program. Go to the previous guide on how to run the program locally. The program should keep running during the test run.

Step 2: Navigate to PlaywrightTests from the root of the repository

```
cd PlaywrightTests
```

Build the project

```
dotnet build
```

Step 3: If this is the first time running our playwright tests, you need to run an extra command. This however requires you to have power-shell installed:

```
pwsh bin/Debug/net8.0/playwright.ps1 install --with-deps
```

Now you can run the tests by typing:

```
dotnet test
```

The UI/E2E tests in this test suite validate the overall functionality of the application, ensuring that the user interface elements work as intended and that navigation between URLs behaves correctly.

3 Ethics

3.1 License

This software is licensed under the MIT license. The MIT license is an open source and permissive license that allows other users to modify, use and distribute the software, just needing the original license to be included. The license has been chosen to make the code freely accessible to other users and therefore encourages collaborations across developers and companies while maintaining minimal restrictions.

3.2 LLMs, ChatGPT, CoPilot, and others

During this project, we used ChatGPT to support our development. We were mindful about how we used it and tried to avoid using it for writing concrete code. Since we are not used to a program that does not only run locally, debugging was harder than our previous semesters. So we sometimes used LLM's to help us find bugs, making the process a lot easier.

When the weekly requirements were more open-ended, it was a good tool for interpretation and helping us get started.. Sometimes we also used it for specific code examples, primarily during testing, since we were also introduced to new ways of testing during this course. We found these uses of LLM's very helpful and they definitely sped up our development, especially in instances where we were very stuck.

4 Appendix

- 2021, *Open Source Software Licenses 101: The MIT License*
  Last visited on 15'th of December 2024
  https://fossa.com/blog/open-source-licenses-101-mit-license/
- Fakhroutdinov, Kirill, *Intro to UML class diagrams*
  Last visited on 16'th of December 2024
  https://www.uml-diagrams.org/class-diagrams-overview.html
- Fakhroutdinov, Kirill, *Intro to UML activity diagrams*
  Last visited on 18'th of December 2024
  https://www.uml-diagrams.org/activity-diagrams.html
- Fakhroutdinov, Kirill, *Intro to UML component diagrams*
  Last visited on 17'th of December 2024
  https://www.uml-diagrams.org/component-diagrams.html
- Fakhroutdinov, Kirill, *Intro to UML sequence diagrams*
  Last visited on 16'th of December 2024
  https://www.uml-diagrams.org/sequence-diagrams.html