Chirp! Project Report

ITU BDSA 2025 Group 16

Aksel Emil Dyhr akdy@itu.dk Samuel Madsen slma@itu.dk
Victor Svendsen Visv@itu.dk
Torkil á Torkilsheyggi toat@itu.dk
Sune hesselberg shes@itu.dk

1 Design and Architecture of *Chirp!*

1.1 Domain model

The following is the domain model diagram of the *Chrip!* project. The project centers around four main entities for which repositories and services exist: cheeps, authors, follows, and likes.

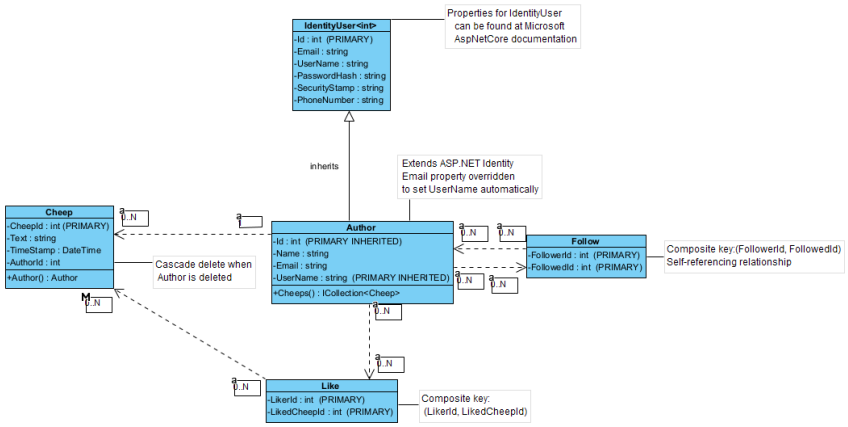

Figure 1: Domain model

1.2 Architecture — In the small

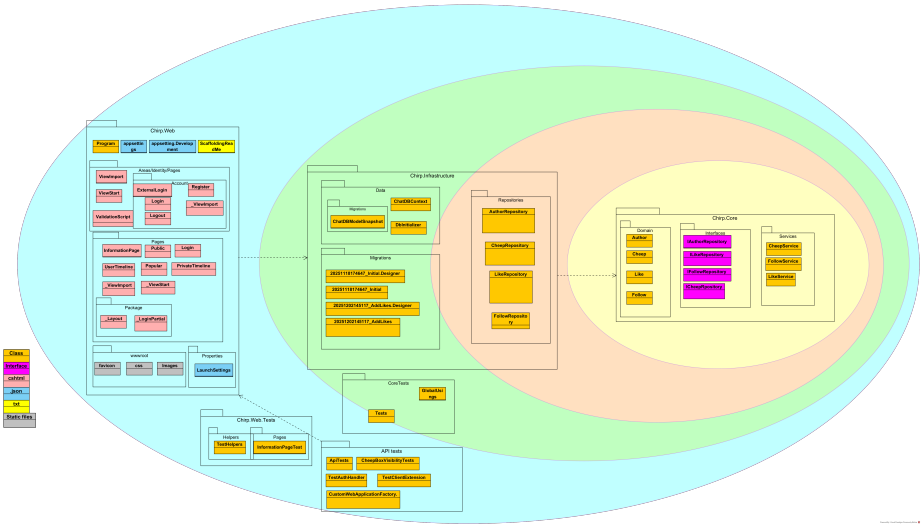

Figure 2: Onion architecture

Each layer of the onion architecture is represented by a .NET project or a .NET test project. Within each project, there may be additional folders or subfolders that can only reference or implicitly use other projects. Each layer in the architecture depends on the inner projects.

1.3 Architecture of deployed application

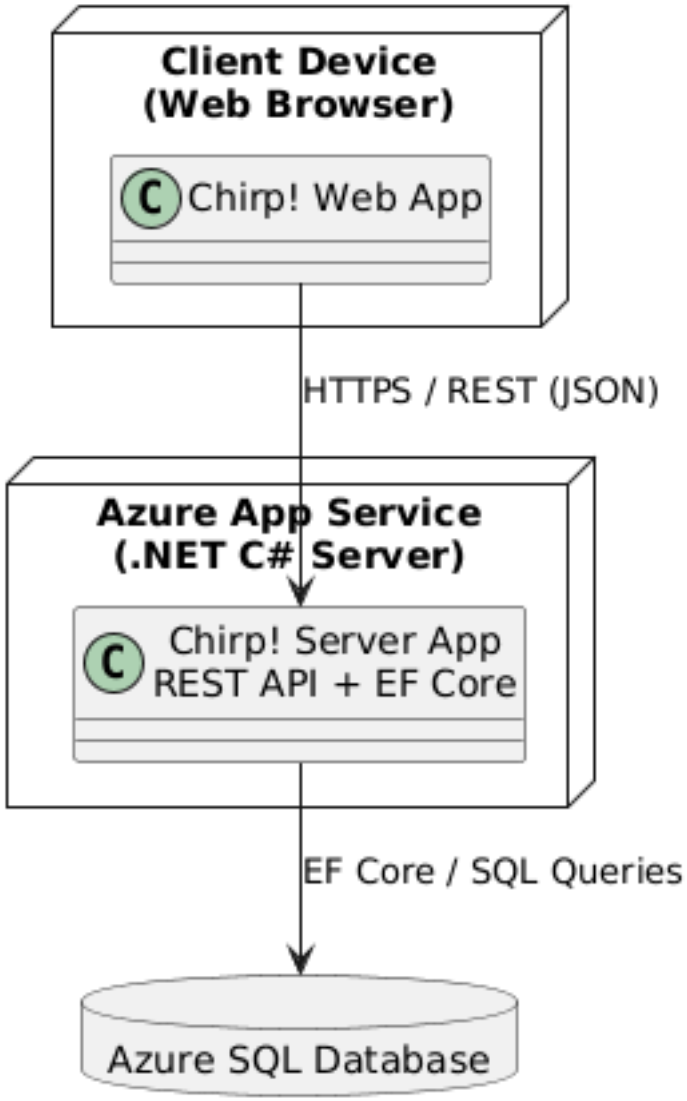

Figure 3: Architecture of deployed application

The Chirp! web app runs in a browser as the client, sending HTTPS requests to the .NET server deployed on Azure App Service. The server handles business logic, processes requests, and interacts with the Azure SQL Database to store and retrieve persistent data such as users, posts, and follows.

1.4 User activities

The first diagram below illustrates the options available to a non-authenticated user, starting from the public page.

The second diagram illustrates the options available to users who have been authenticated.

For clarity, Activities have been colored either blue or green. Blue represents states/displays of the application. Green represents buttons/actions available to the user, such as *"click: public page"* The following and like features are available across multiple pages in the application. However, due to the fact that the flow of those features is the same regardless of where they are performed, they are only represented once in the diagram.

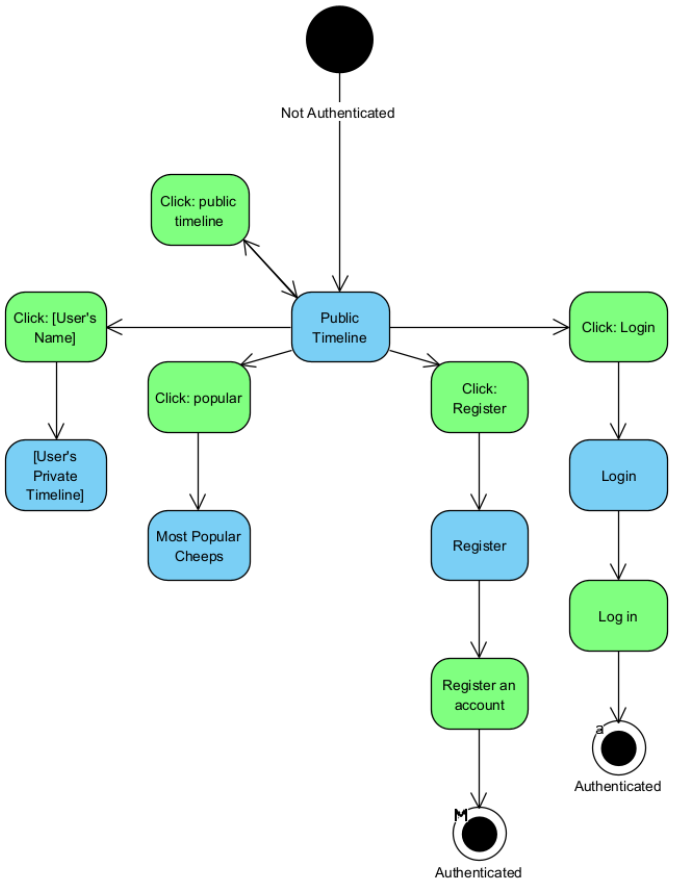

Figure 4: Non-authenticated user activities

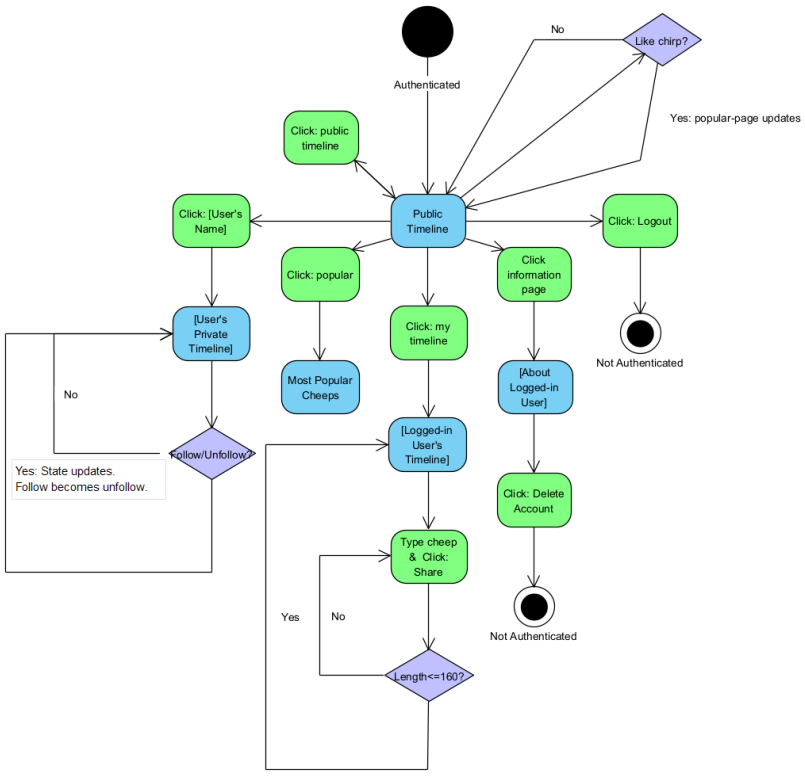

Figure 5: Authenticated user activities

1.5 Sequence of functionality/calls trough *Chirp!*

The following sequence diagram illustrates the request flow when a client accesses the public page of the application. It shows how an HTTP request is processed through the web layer, page model, application services, infrastructure, and database, and how data is returned and rendered as an HTML response.

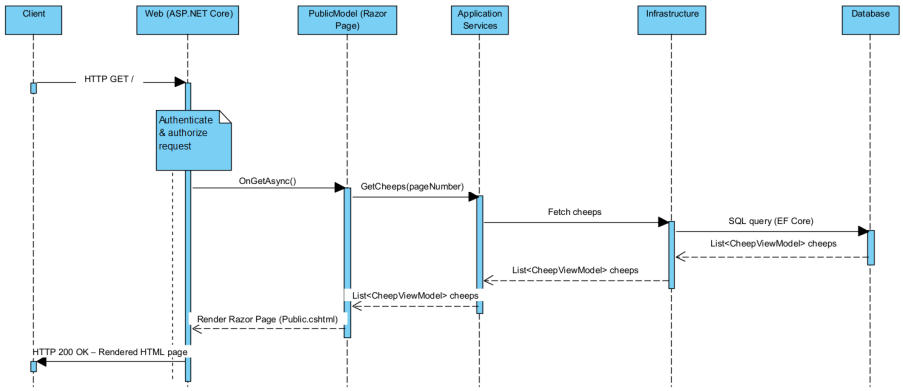

Figure 6: Sequence of functionality

2 Process

2.1 Build, test, release, and deployment

2.1.1 Build & Test Diagram

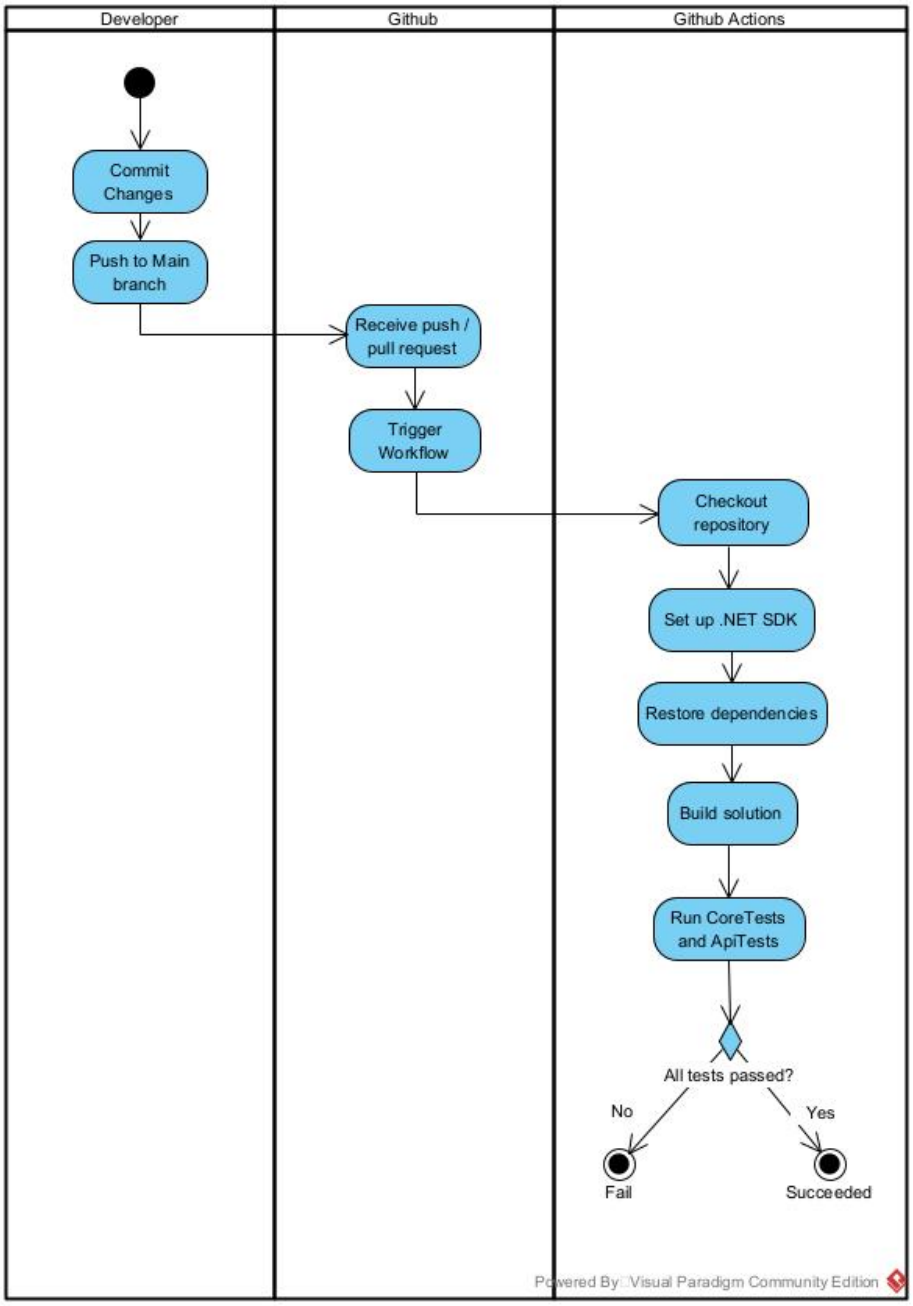

Figure 7: Build & test diagram

This activity diagram illustrates the continuous integration workflow for the Chirp! application. When a developer pushes changes or opens a pull request on the main branch, GitHub triggers a CI workflow. The workflow checks out the repository, sets up the .NET environment, restores dependencies, builds the solution, and runs both core and API tests. The process includes a decision point where the workflow either fails if any test fails or completes successfully if all tests pass.

2.1.2 Build & Deploy to Azure

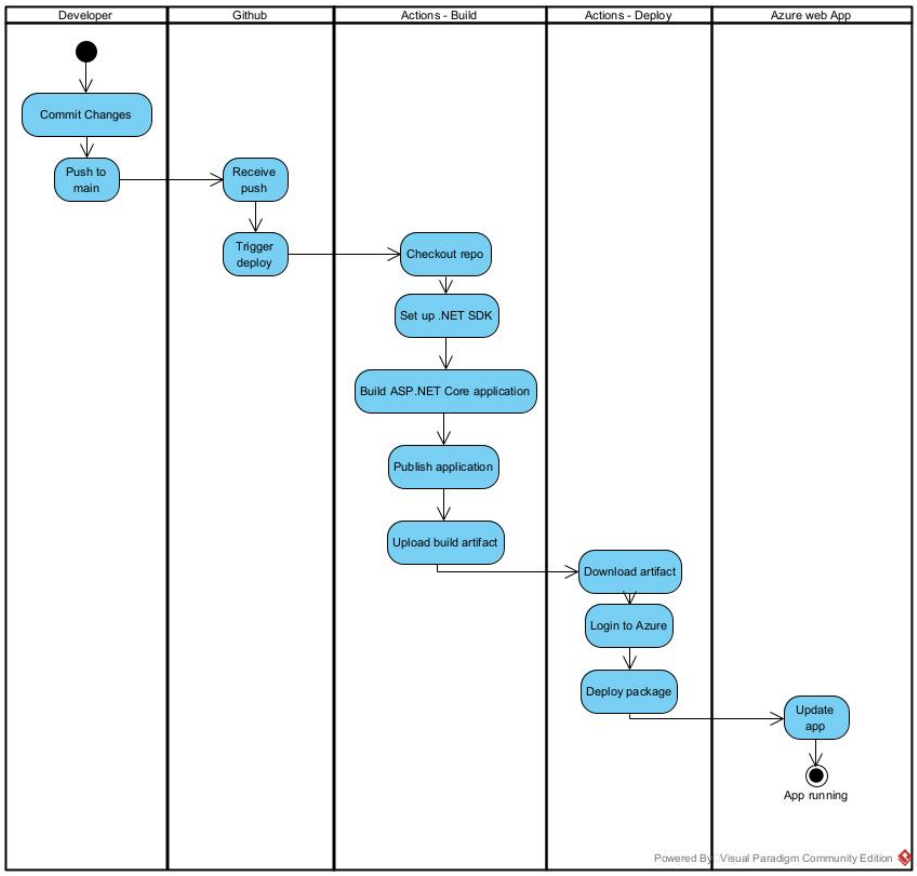

Figure 8: Build & Deploy to Azure

This activity diagram shows the continuous deployment workflow for Chirp!. When a developer pushes changes to the main branch, GitHub triggers a deployment workflow. The application is built and published using GitHub Actions, and the resulting artifact is uploaded and passed to a deployment job. The workflow then authenticates with Azure and deploys the application to the

production slot of an Azure Web App. Once deployed, the updated application is running in the production environment.

2.1.3 Release Workflow

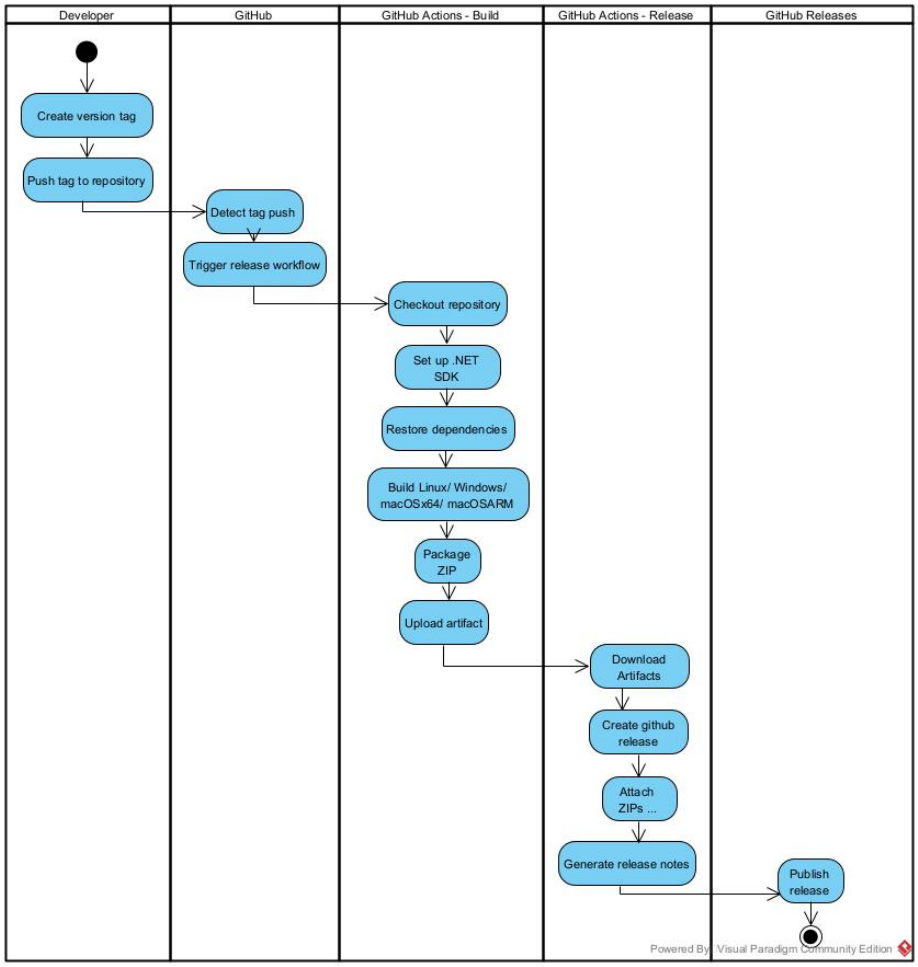

Figure 9: Release Workflow

This activity diagram describes the release workflow for Chirp!, which is triggered when a version tag is pushed to the repository. The workflow builds and publishes the application for multiple operating systems in parallel, including Linux, Windows, and macOS. Each build is packaged as a ZIP artifact and uploaded. After all builds complete, a GitHub Release is created, the artifacts are attached, and release notes are generated automatically. The final result is a published versioned release available for download.

2.2 Team work

Tasks or functionalities that are incomplete: Some of our tests are old, and although they are not redundant, they could be modernized to better reflect the current state of the project. Additionally, some parts of our program lack test coverage.

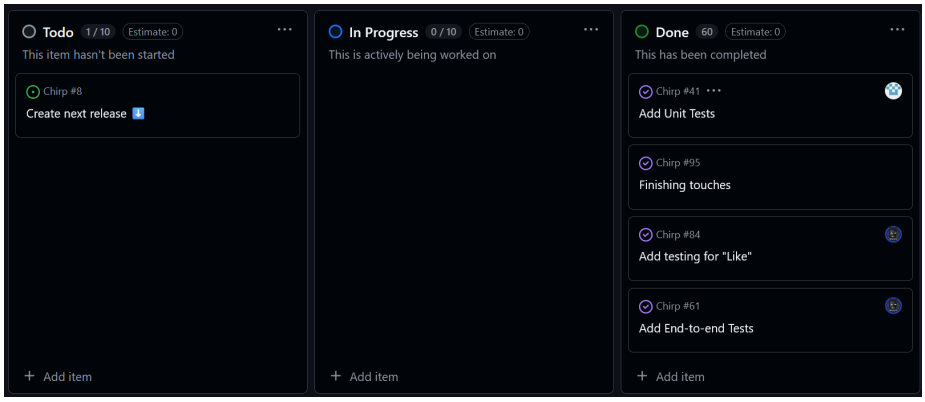

Figure 10: Git projectboard

An issue would be created from the requirements given in the course. It would then be added to our Git to-do list. From there we would assign issues to different group members. Mostly just one person per issue. When a task is completed a pull request is made, and from there another group member reviews the work, and decides if it is acceptable. If it is, the branch is merged into the main branch, and if it is not acceptable, the reviewer tells the group member what is wrong and what has to be changed. From there the group member has to make a pull request again, when the task is complete.

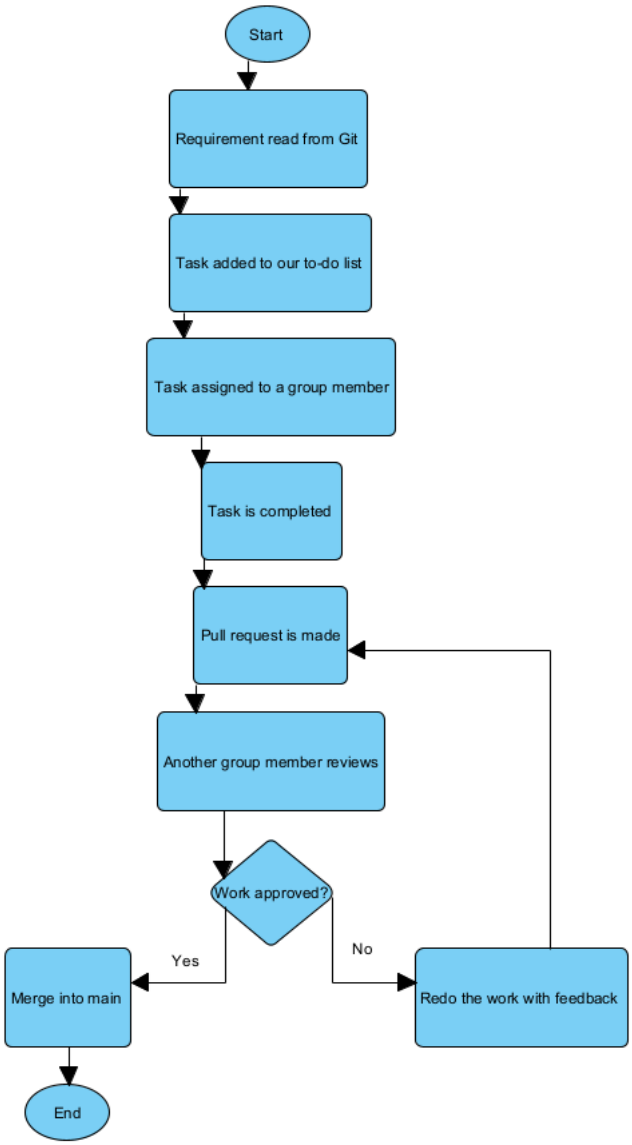

Figure 11: Git teamwork diagram

2.3 How to make *Chirp!* work locally

Make sure you have the following downloaded: .NET 8 SDK

To run *Chirp!*: 1. Clone the repository to your directory of choice

```
git clone https://github.com/ITU-BDSA2025-GROUP16/Chirp.git
```

2. Set up user-secrets *(required for Github OAuth login)* 2.1. Access your github developer settings 2.2. Click **New OAuth App** 2.3. Fill in the following:

Application name: Chirp Local **Homepage URL:** http://localhost:5696 **Authorization callback URL:** http://localhost:5696/signin-github

2.4. Click **Update application** 2.5. Copy your *Client ID*, and generate a *Client secret* 2.6. Finally, set your secrets:

```
cd src/Chirp.Web
dotnet user-secrets set "authentication:github:clientId" "<your-client-id>"
dotnet user-secrets set "authentication:github:clientSecret" "<your-client-secret>"
```

3. While still in the chirp.web directory, run the project

```
dotnet run
```

4. The web application can be accessed on http://localhost:5696 in a browser. You should see the *Chirp!* public page here.

2.4 How to run test suite locally

The project has 3 seperate test suites: * ApiTests * Chirp.Web.Tests * CoreTests

From the root, first

```
dotnet restore
dotnet build
```

Then access your desuired test suite, e.g.

```
cd tests/ApiTests
```

Then run the tests inside it

```
dotnet test
```

The suites contain unit tests, integration tests, and E2E tests. With these, the system's core components (pages, repositories, services, etc.) are tested both individually and when working together. The E2E tests target two specific user flows: liking cheeps and following users.

3 Ethics

3.1 License

Chirp is licensed under MIT.

3.2 LLMs, ChatGPT, CoPilot, and others

As large language models (LLMs) have become increasingly integrated into the creative process, we have naturally turned to them for inspiration and guidance

regarding assignment requirements. Our group has prioritized transparency by consistently co-authoring with LLMs whenever they are utilized. Generally, we have used LLMs not for coding purposes but as sources of inspiration and as a collaborative partner.