Chirp! Project Report

ITU BDSA 2023 Group `10`

Theis Per Holm thph@itu.dk
August Kofoed Brandt aubr@itu.dk
Jonas Skjødt skjo@itu.dk

sdsdyfg

1 Design and Architecture of *Chirp!*

Chirp! is a mini social media platform based around user generated "cheeps", 160 character text-based messages which stores the input of the cheep author's shared message. It is hosted on Azure App Service and utilizes ASP.Net 7.

1.1 Domain model

The Chirp! domain model represents the fundamental structure and behavior of the application. It describes how users, who can be authors once they write one or many cheeps, also can react to cheeps and follow other users. The Follows, Reaction, Author, Cheep, and User classes manage these relationships and interactions on the platform.

1.2 Architecture — In the small

1.3 Architecture of deployed application

1.3.1 Architecture

The project uses the Razor Pages framework to serve Chirp.Web. Chirp.Web effectively sets up the GUI for the website, and is a blend of static html components like images, css, javascript files, viewcomponents, and razor pages, all of them collectively handles the user's experience on the Chirp platform.

The backend of the website is handled by Chirp.Core and Chirp.Infrastructure. As its innermost layer, Chirp.Core handles the Data Transfer Objects (DTOs) and the interfaces. The DTOs are data carriers, exchanging data related to authors, cheeps, users, follows, and reactions between the different parts of the project.

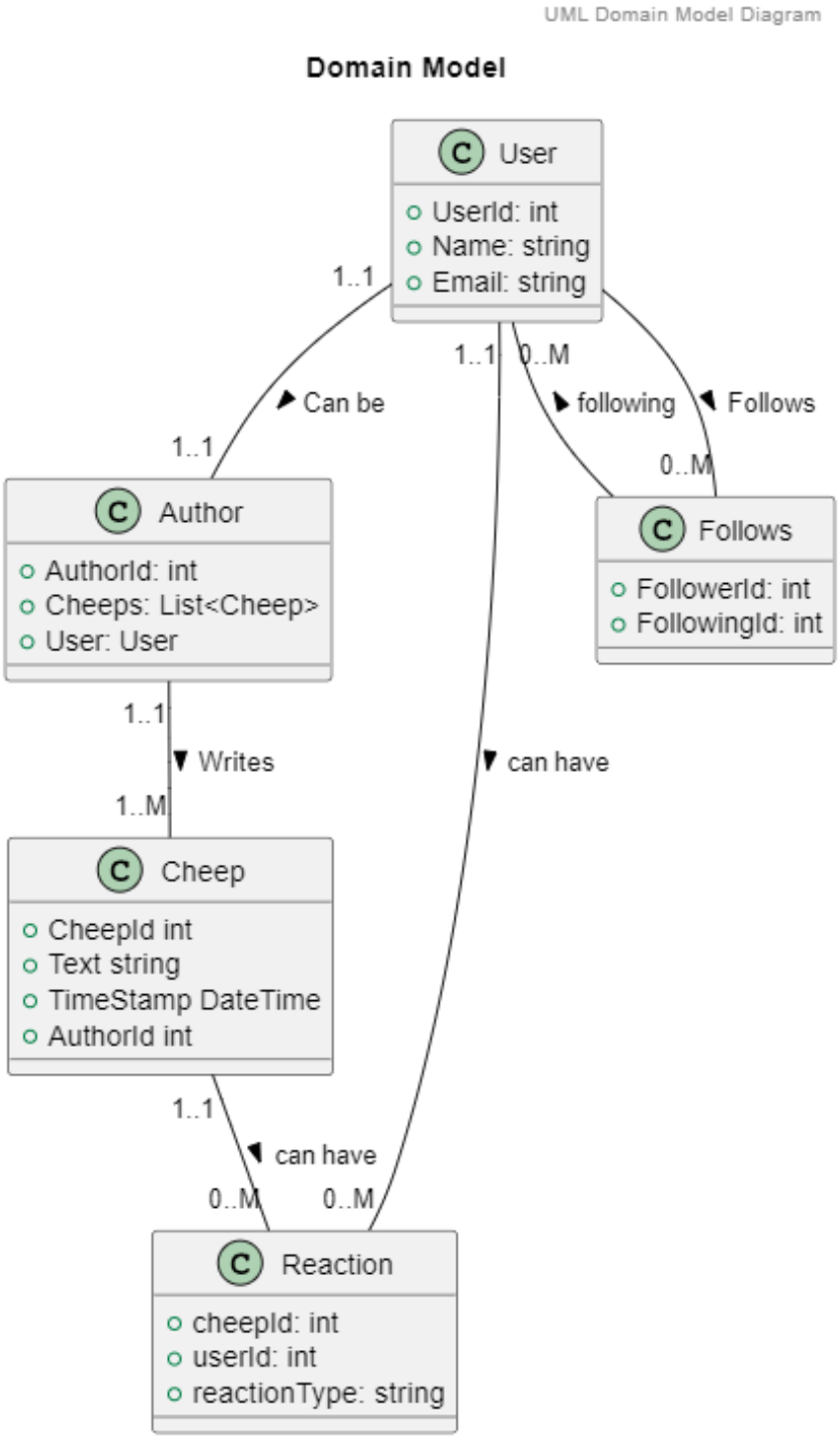

Figure 1: Illustration of the *Chirp!* data model as UML class diagram.

Chirp.Infrastructure is the persistent data layer of the Chirp application. This is where models for Author, Cheep, Follows, Reaction, User, and their corresponding repositories exist, handling the actual data operations with the database. It also includes the Dbinitializer, which helps manage the database state.

1.3.2 Client-server communication

The database server is hosted on SQL Server on Azure App Service. The database server works with the Chirp.Infrastructure layer where it stores and retrieves data as requested by the application. The Chirp.Infrastructure then communicates with Chirp.Web and ensures the data flow.

The communication between the client application and the server is carried out through HTTP and HTTPS protocols to Chirp.Web. These protocols are safely handled by the Azure App Service using their TCP/IP service, giving high certainty of reliable data transmissions.

When a user interacts with the Chirp website, an HTTP request is sent to the Azure server, and any following interactions made are as shown on the diagram below.

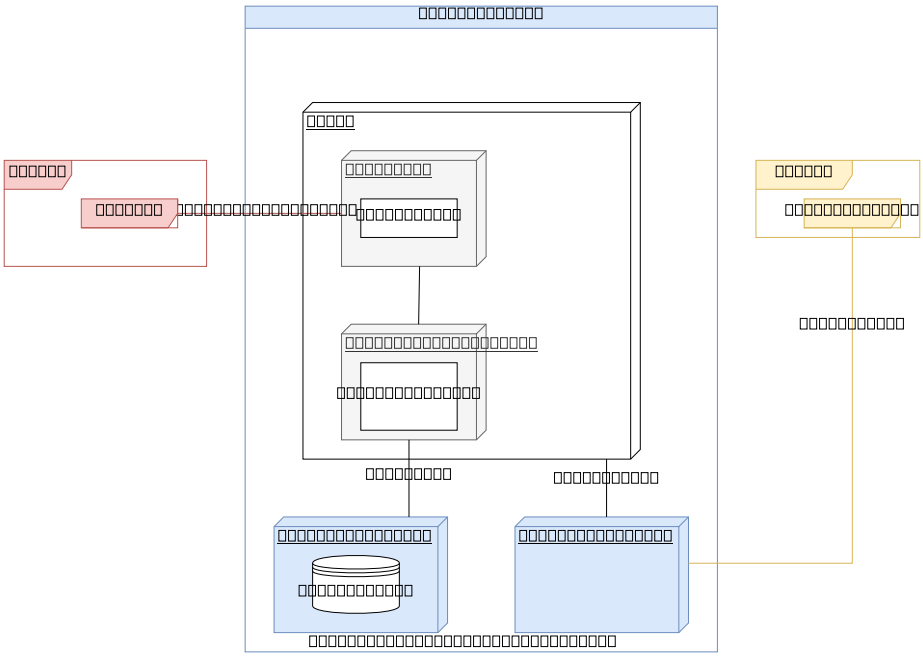

Figure 2: Illustration of the *Chirp!* deployed architecture as UML class diagram.

1.3.3 User Authentication

User authentication in Chirp is managed through Azure's B2C and GitHub's OAuth authentication service. When a user attempts to log in, the application uses Azure's B2C service which connects them to GitHub's OAuth login page. After the user enters their GitHub credentials and grants the necessary permissions, GitHub sends an authorization code back. This authorization code is handled by an ASP.NET handler form in Chirp which then exchanges it for an access token.

The access token is a unique string that represents the user's authorization to access their GitHub data. This token is stored securely in the user's session and is used to authenticate subsequent requests made by the user.

By delegating authentication to GitHub, Chirp can provide a secure and user-friendly login process. It also reduces the risk of storing and managing sensitive user data on the Chirp server, as the user's password is never directly handled by the application.

1.3.4 External services

Besides using Github for Chirp's login authentication, Chirp also uses Github's profile images as a user's image avatar in the application. If the Chirp user has no avatar uploaded to their Github profile, Github will serve the default avatar image.

Chirp also uses the JQuery library as a dependancy to enhance Chirp users with an AJAX script to dynamically load the follow, unfollow, and reaction forms. This ensures better user experience and fewer elements loaded once interacted with, compared to that of the entire webpage being reloaded on each button interaction.

Lastly, when a cheep author includes a YouTube URL within a cheep, the Chirp application utilizes regular expressions (regex) to extract a unique 11-character video ID from the URL. This ID is then used to generate an iframe that embeds the corresponding YouTube video directly into the cheep.

1.4 User activities

The User activities diagram shows what is possible for a user to do on our Chirp application. Both when authenticated and unauthenticated.

Web version of *User activities* diagram

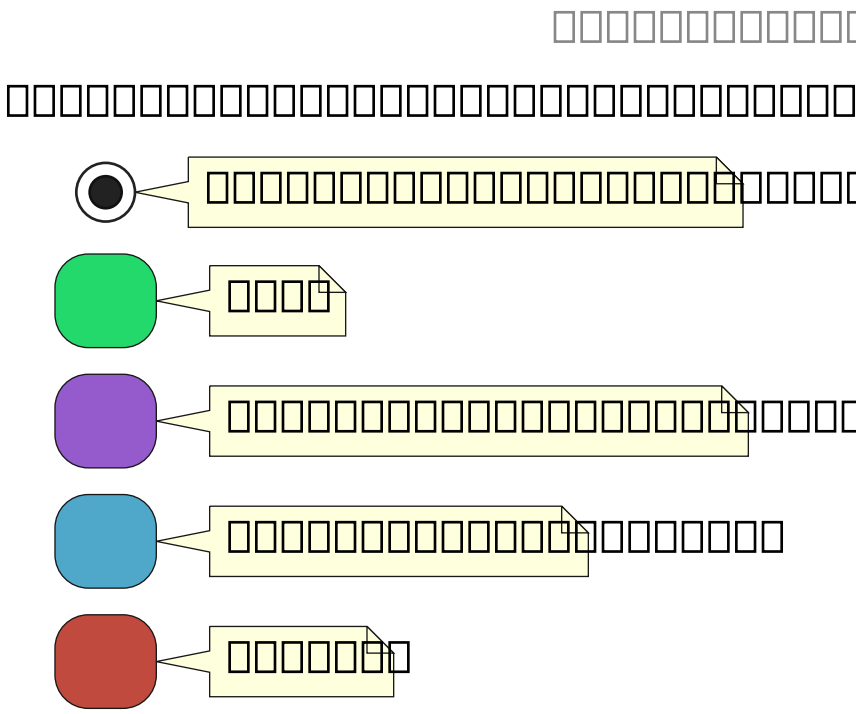

Figure 3: User activities activity diagram legend

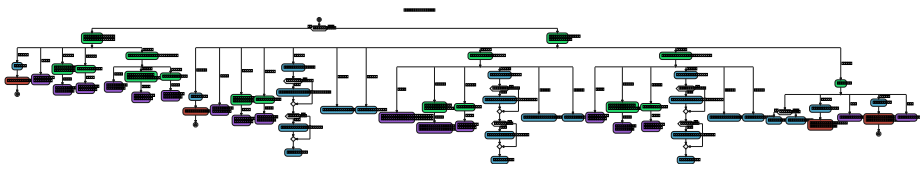

Figure 4: User activities activity diagram

1.5 Sequence of functionality/calls trough *Chirp!*

2 Process

2.1 Build, test, release, and deployment

The group employed the use of Github Workflows/Actions to build, test, release and deploy the app to Azure. The UML activity diagrams below show how each of the Workflows work

2.2 Team work

//TODO: Add picture of project board

Above is an image of the project board for our chirp project. While the vast majority of issues are done and closed, some issues are still open.

Overview of uncompleted issues: | Issue | Description | Not completed because: | | ——— | ——— | ——— | | #175 (Bugs) | Make playwright UI tests able to run with a workflow through GitHub actions. | Problems with getting playwright test to run in "Headless" mode.Because running UI tests on GitHub actions was not an important requirement for the project, the issue was not prioritized. | | #263 (Bugs) | When running integration tests on a unix based system, the tests would give a `System.InvalidOperationException`. | Since the exception didn't stop the tests from running and passing, the issue was not prioritized. | | #225 (Not started) | Be able to run `playwright` tests on a dockerized container of the project. | The scope of this issue was too big for the deadline of the project, and not a necessary addition. | | #279 (Not started) | Change general exception throws to be more specific to the reason it was thrown. | Since most of the program uses general exceptions, solving this issue would take some time. Because this issue didn't impact the performance or the functionality of the program, and due to the deadline approaching, it was decide that other issues would be prioritized. |

2.2.1 Group's work activities

Web version of *Group's work activities* diagram

The group would create issues after a lecture, based on the project work given and tasks that needed to be solved. New issues where added to the "Not started" column on the project board or "Bugs" if the issue pertained to a bug.

How work was conducted on the issues, can be seen in the diagram above [team workflow diagram].

Assigned group members take responsibility for updating the issue and the project board.

The amount of group members on one issue would vary depending on the issue. Smaller or more specialized issues would often only be assigned one group

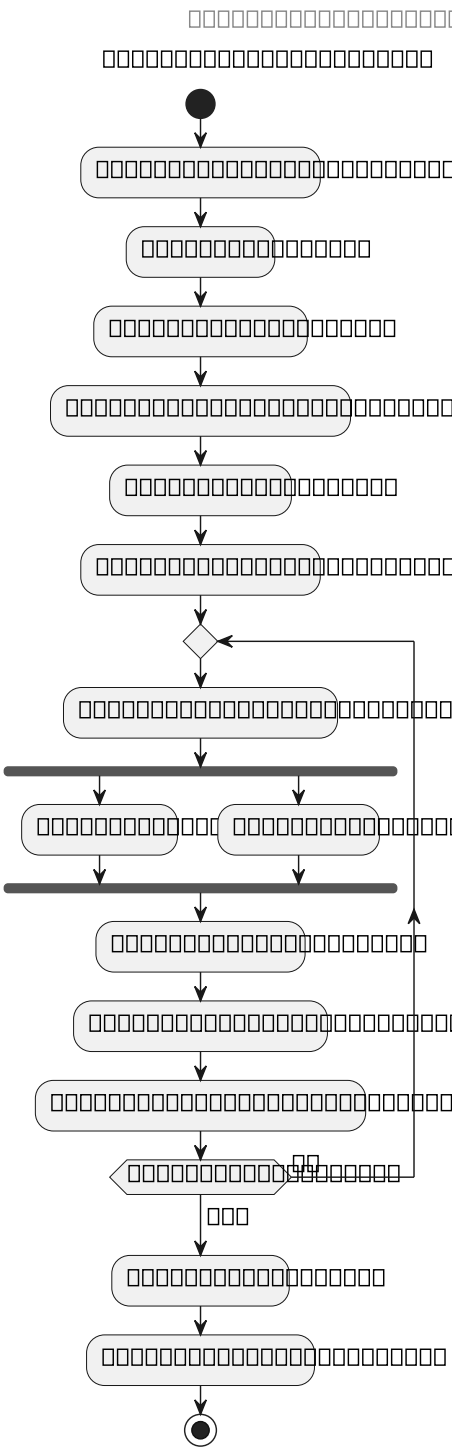

Figure 5: Activity diagram of group workflow

member. A specialized issue could be one that pertained to a subject that one group member was significantly more experience in than other members were.

To automate closing and moving of issues on the project board when a pull-request for an issues was merged, the group used GitHub keywords like “Resolves” and “Closes” with links to the issues.

2.3 How to make *Chirp!* work locally

For a full guide on how to run the project locally see the ReadMe.md on the public repository: Chirp ReadMe.md

2.4 How to run test suite locally

3 Ethics

3.1 License

The group has chosen the MIT open source software license

> MIT License
>
> Copyright (c) [year] [fullname]
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

//TODO: figure out how we want to source from things VISITED 18/12 Source: Github choose a license site

The group chose this license as it was a good fit for the groups requirements of an open source license in that it basically has no restrictions for any end user or somebody who wants to work with the codebase. We also sign off any warranty or liability which is great for a small group project that we more than likely wont want to take further in the future.

3.2 LLMs, ChatGPT, CoPilot, and others

The use of LLMs like ChatGPT and Copilot has been documented on github commits as a co-author when used. You can see the number of these commits on the page linked here: ChatGPT Co-authored commits. Sadly the page that shows the actual commits doesn't have the commits that it contributed on as these were done on separate branches whose commits seem to not carry over to the main branch's working tree.