Chirp! Project Report

ITU BDSA 2023 Group 25

Silas Wolff siwo@itu.dk
Sebastian Blok segb@itu.dk
Karl Gustav Løhr kagl@itu.dk
Adam Nørgård Aabye aaab@itu.dk
Atila Arianpour atia@itu.dk

1 Design and Architecture of *Chirp!*

1.1 Domain model

Our domain model is built around the core concept of an Author, which is central to the *Chirp!* application's functionality. An Author represents a user of the application, encapsulating their identity and interactions within the system. The UML class diagram model the key entities that make up our application.

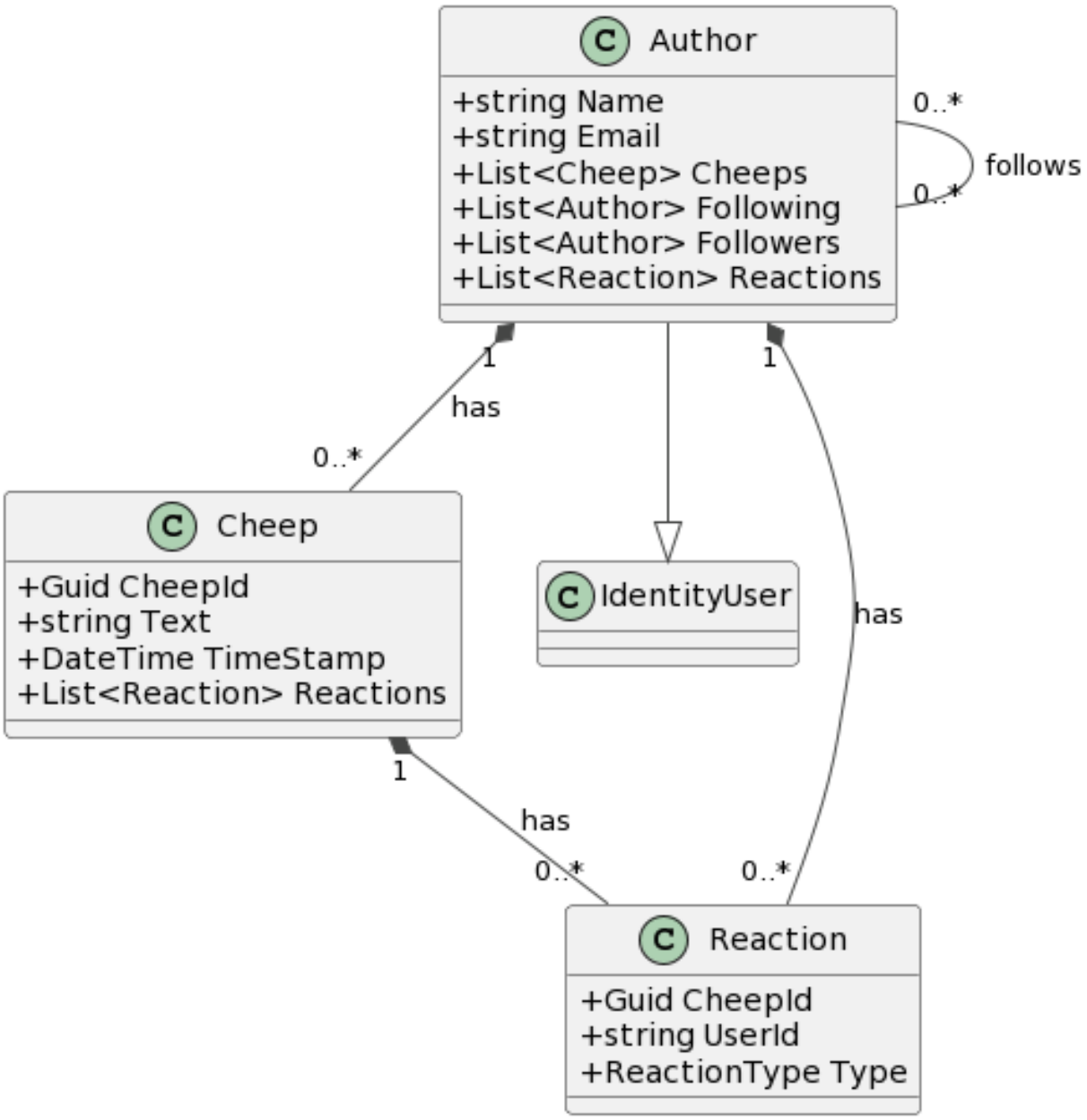

Figure 1: Domain Model

1.1.1 Author

The Author entity is an extension of the ASP.NET Identity's IdentityUser, inheriting features essential for authentication and authorization.

Each Author has a collection of Cheep entities, representing the messages, or posts, that the author creates within the application. This one-to-many relationship is depicted by a composition association, emphasizing that Cheeps are intrinsic to their Author and cannot exist independently.

In addition to creating cheeps, Authors can 'follow' other Authors. This is represented by a many-to-many self-referencing association, indicating that an Author can follow multiple other Authors and also be followed

by multiple others. This relationship captures the essence of the application's social interaction capabilities.

1.1.2 Cheep

A Cheep is essentially a message, or a singular piece of communication, created by an Author. Each Cheep is uniquely identified by a Guid and contains the message text along with a timestamp of its creation. Cheeps is in a one-to-many composition with Authors, meaning that, Authors can have many cheeps, but cheeps must have exactly one author. Additionally, they have a strong life-cycle dependency.

1.1.3 Following

Hmm... lidt tbd, lad os lige snakke om implementationen, evt association class, ellers lister som nu

1.1.4 Reactions

A reaction is entity that refers to the interactive engagement that users can express in response to a cheep. These reactions are represented by a "thumps up" emoji that turns red if pressed. For each cheep a number of reactions will be displayed in the application. Each reaction is uniquely identified by a Guid representing the cheep, a string which represents the user who has reacted and a reactiontype representing which type of reaction it is. Reactions is in one-to-many relationships with both authors and cheeps meaning that both authors and cheeps can have many reactions but each reaction is uniquely related to one author and one cheep.

```
var cheep = _databaseContext.Cheeps
    .Include(c => c.Reactions)
    .FirstOrDefault(c => c.CheepId == cheepId);
var author = _databaseContext.Users.SingleOrDefaultAsync(a => a.Name == authorName);
```

The reactions functionality is controlled by its interface IReactionRepository with it's three mandatory methods, HasUserReacted, ReactionOnCheep and GetReactionAmount. ReactionOnCheep is an asynchronous task that takes a reactiontype, cheepid and username. The task starts by checking for the specific username and cheepId in the database and if any of those two are null the method returns an exception.

```
if (cheep != null || author != null)
{
    var reaction = new Reaction
    {
        CheepId = cheepId,
        AuthorName = authorName,
        ReactionType = reactionType
    };

    var currentReaction = _databaseContext.Reactions.FirstOrDefault(
        r => r.CheepId == reaction.CheepId
            && r.AuthorName == reaction.AuthorName
            && r.ReactionType == reaction.ReactionType
    );

    if (currentReaction is null)
    {
        _databaseContext.Reactions.Add(reaction);
    }
    else
    {
        _databaseContext.Reactions.Remove(reaction);
    }
    await _databaseContext.SaveChangesAsync();
}
else
{
    throw new NullReferenceException("Cheep not found");

}
```

If not an instance of the object reaction will be created. Along side the task instantiate a variable "currentReaction" and check if the cheep already has a reaction from the user in the database. Based on the outcome of the check the system will either add the reaction to the database or remove it from database. This is because the ReactionOnCheep task handles both cases where a user wants to react on the cheep(add a reaction to the database) or remove the reactions from the cheep(remove the reaction in the database).

The asynchronous task "HasUserReacted" is responsible for letting the system know if a user has already reacted on a cheep. It takes a cheepId and username as arguments. Firstly it check whether or not a user is to be found in the database and if that is the case an exception is thrown. In the case where a user is

found the task checks if the database contains a reaction sat on the cheep and if the user is the owner of that reactions. The result will be a boolean which is depended on the user interaction with the cheep.

The last task is responsible for letting the system know how many reactions a cheep has. This is simply done be checking the database how many reactions are related to the cheepId the task takes as argument. These get put in a list where it asynchronously returns the total number of reactions.

1.2 Architecture — In the small

Our application is separated into 3 main layers, that are common for the onion architecture

Layers: * core * infrastructure * web

Dependencies: * identity -> infrastructure * ef core -> infrastructure * core -> infrastructure * OAuth -> web * core -> web * infrastructure -> web *

1.3 Architecture of deployed application

1.4 User activities

1.5 Sequence of functionality/calls trough *Chirp!*

2 Process

2.1 Build, test, release, and deployment

We use Github Actions to automate the build, test, release and deployment process of our executables and website. This pipeline is centered around two workflows, one for the executables, and one for the website.

2.1.1 Publishing workflow

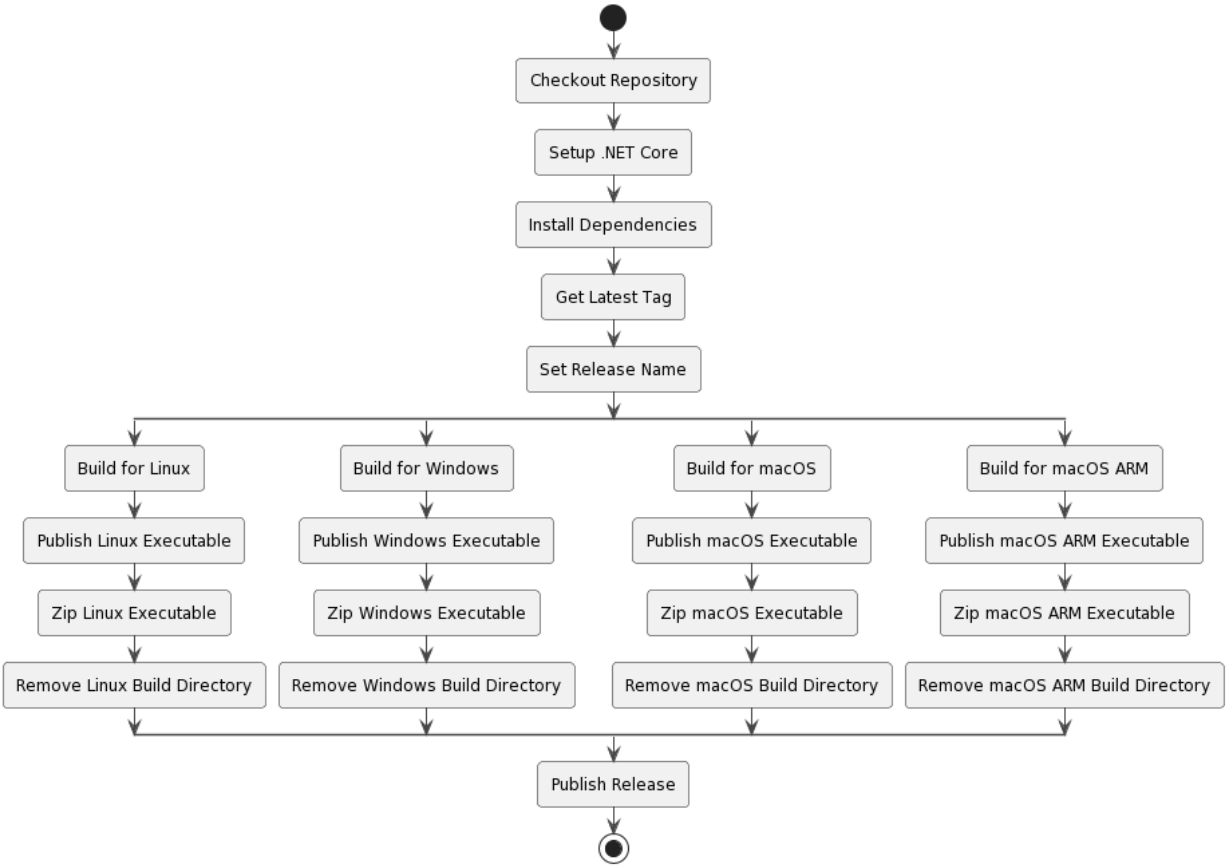

Figure 2: Activity Diagram for Publishing

The publishing workflow focuses on creating executable artifacts for various platforms, including Linux, Windows, macOS, and macOS ARM. This workflow is triggered by pushing specific version tags, reflecting our versioning strategy. We apply the semver strategy.

After the standard initial steps of checking out the repository and setting up the .NET Core environment, the workflow splits into parallel tasks for each platform. Each branch involves building the application, publishing the executable, zipping the file, and removing the build directory. This parallel structure allows for efficient and simultaneous preparation of executables for different platforms.

The final step is publishing the release on GitHub, attaching all the zipped executables. This process not only automates the release creation but also ensures that our application is readily available for a wide range of platforms, enhancing its accessibility to users.

2.1.2 Deployment workflow

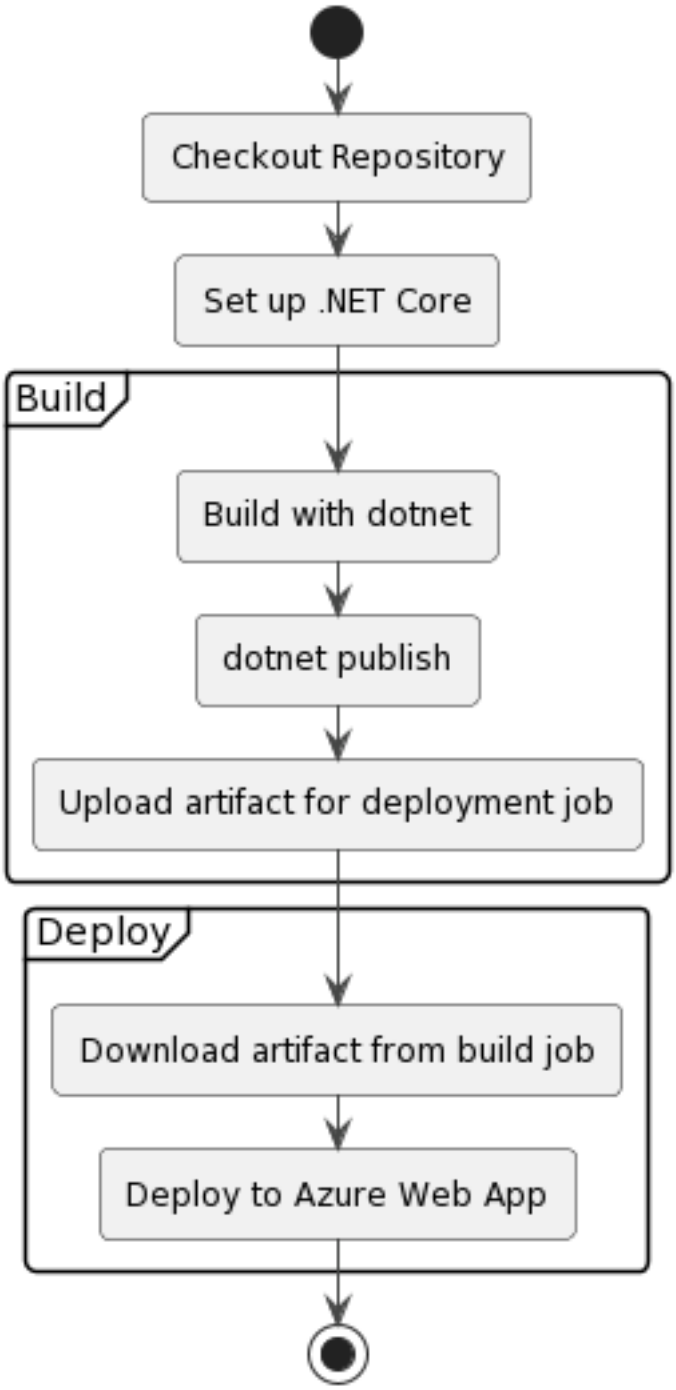

Figure 3: Activity Diagram for Deployment

Note: Figure x, 'Activity Diagram Deployment' should have been here, but latex has its own life, and its not easy to fix when it needs to go through pandoc.

Our deployment workflow, as visualized in the activity diagram, efficiently manages the deployment of the

Chirp! application to Azure Web App. The process is triggered upon pushes to the main branch or via manual dispatch, ensuring that our latest stable build is always deployed.

The workflow begins with checking out the repository and setting up the .NET Core environment. The build phase involves compiling the code and publishing it to a specified directory. The published application is then uploaded as an artifact, ready for deployment.

In the deployment phase, the build artifact is downloaded and deployed to the Azure Web App. This automated process ensures a consistent and reliable deployment strategy, minimizing human error and streamlining our release process.

2.2 Team work

2.3 How to make *Chirp!* work locally

2.4 How to run test suite locally

3 Ethics

We are commited to being inclusive and respectful to anyone, related or unrelated to the work done in this project. We have included a slightly modified version of the Citizen Code of Conduct, that has been included in our repository on Github under CODE_OF_CONDUCT.

3.1 License

We have picked the MIT License for our project. The MIT License is a simple and highly permissive open-source software license. It is one of the least restrictive, of the established licenses available. Under the MIT License, users are granted almost unrestricted freedom to use, modify, distribute, and sublicense the software. Our only requirement is that the copyright notice and license is included with the software when redistributed. As our group name is included in the MIT-license, this ensures that we are attributed. Additionally, the license absolves us of any responsibility or liability of how the software is used by others.

We have included the full text of the license below:

```
MIT License

Copyright (c) 2023 ITU-BDSA23-GROUP25

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.
```

The license can be found on our Github repository, under LICENSE.

3.2 LLMs, ChatGPT, CoPilot, and others

Lad os lige snakke om hvad vi siger her?

Lille test

4 Perspektivering, eller overvejelser, eller noter, eller fri leg?

Den her sektion er ikke en del af templatet, men jeg tænker her kan vi skrive nogle overvejelser som vi har gjort og som vi måske/måske ikke vil have med i rapporten.

Overvejelse: Bør followers i class diagrammet være en association class?