Chirp! Project Report

ITU BDSA 2025 Group <5>

Oscar Dalsgaard Jakobsen osja@itu.dk
Niels Laier Jensen niej@itu.dk
Tobias Oliver Nielsen toon@itu.dk
Martin Antonius Jæger maraj@itu.dk
Carl Frederik Thomsen cfth@itu.dk

1 Design and Architecture of *Chirp!*

1.1 Domain model

Our domain model consists of two main elements, and one for infrastructure purposes: - User

A 'User' is an entity who can write new Cheeps, and follow other Users. The relation from one user to another user describes a one-to-many relation, where one user can have many followers. - Cheep

The 'Cheep' entity is the posts in the Chirp application. An author can write many Cheeps, which explains the one-to-many relation. - Follow

The 'Follow' table keeps track of which users follow who, which can both be linked through UserId's, or an entity of a User class.

Below is a diagram visualizing the relations between our different entities.

1.2 Architecture — In the small

Our project follows the principles in Onion Architecture, where the presentation, the infrastructure and the domain are separated into layers. This minimalizes coupling and supports maintainability as well as testability.

The presentation happens in the Web layer, which handles application startup as well as user interaction.

The Infrastructure layer acts as an intermediary between User Interface and the core logic. Managing data persistence, repositories as well as database migrations among other things.

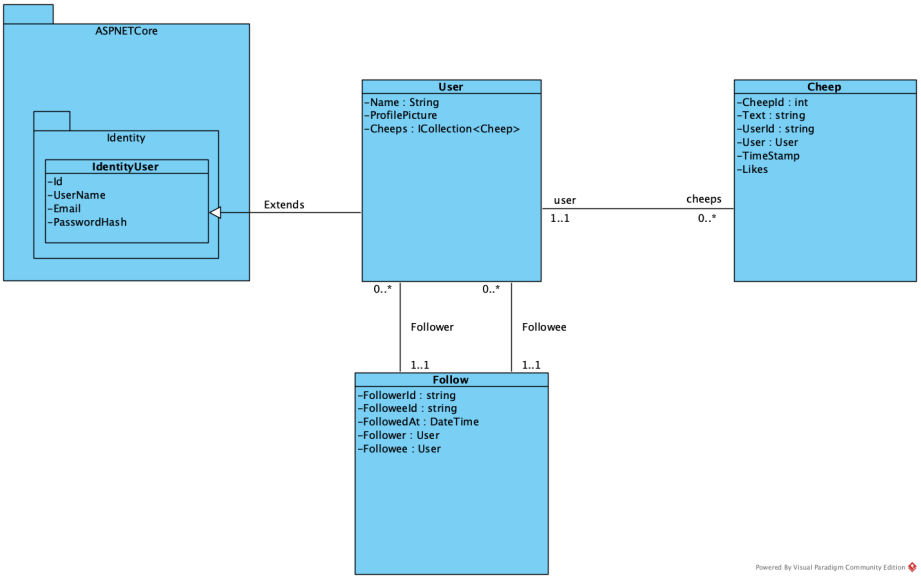
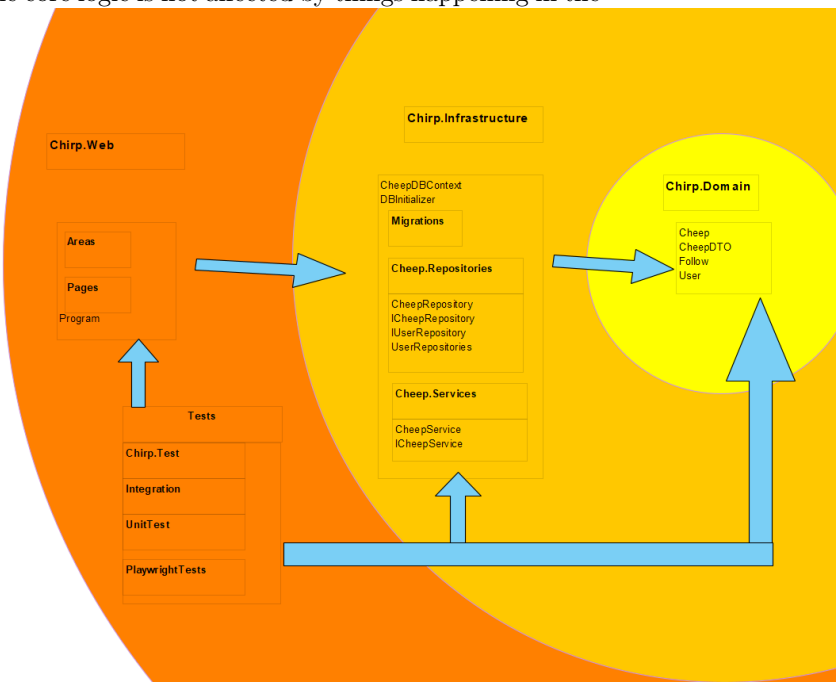


Figure 1: Illustration of the '*Chirp!*' data model as UML class diagram.

The Domain layer is completely independent, since the dependencies are directed inward. This ensures that the core logic is not affected by things happening in the



infrastructure layer.

1.3 Architecture of deployed application

Users send HTTPS requests from their browser (the client) to our application hosted on Azure. Azure runs our ASP.NET Core server, which processes request using Razor Pages. The server accesses data from a SQLite database via Entity Framework Core and handles user authentication with ASP.NET Core Identity.

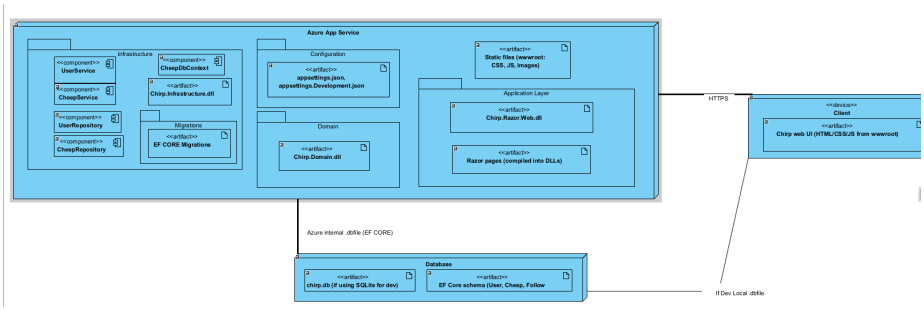


Figure 2: Architecture of Deployed application

1.4 User activities

A user activity diagram showing what an unauthorized user can do and how they become authorized.

A user activity diagram showing the actions available to an authorized user, such as managing their account, posting cheeps or following other users.

A user activity diagram showing the entire process of an unauthorized user becoming authorized and how they can interact with Chirp!

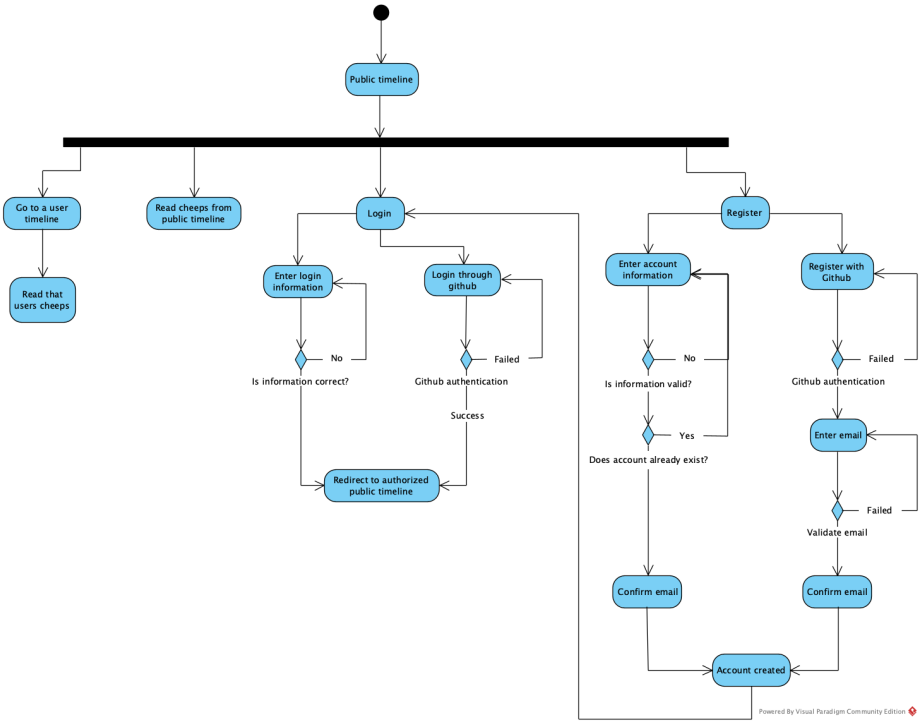


Figure 3: Unauthorized user activity diagram

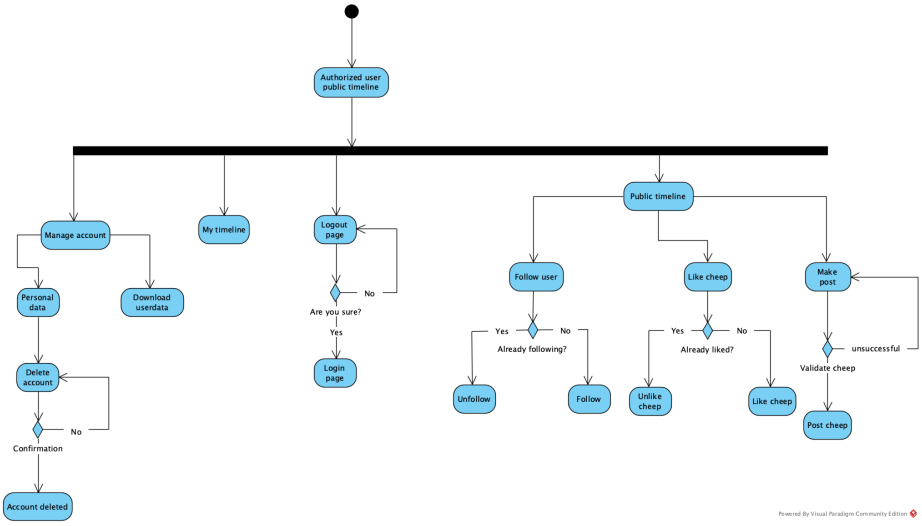
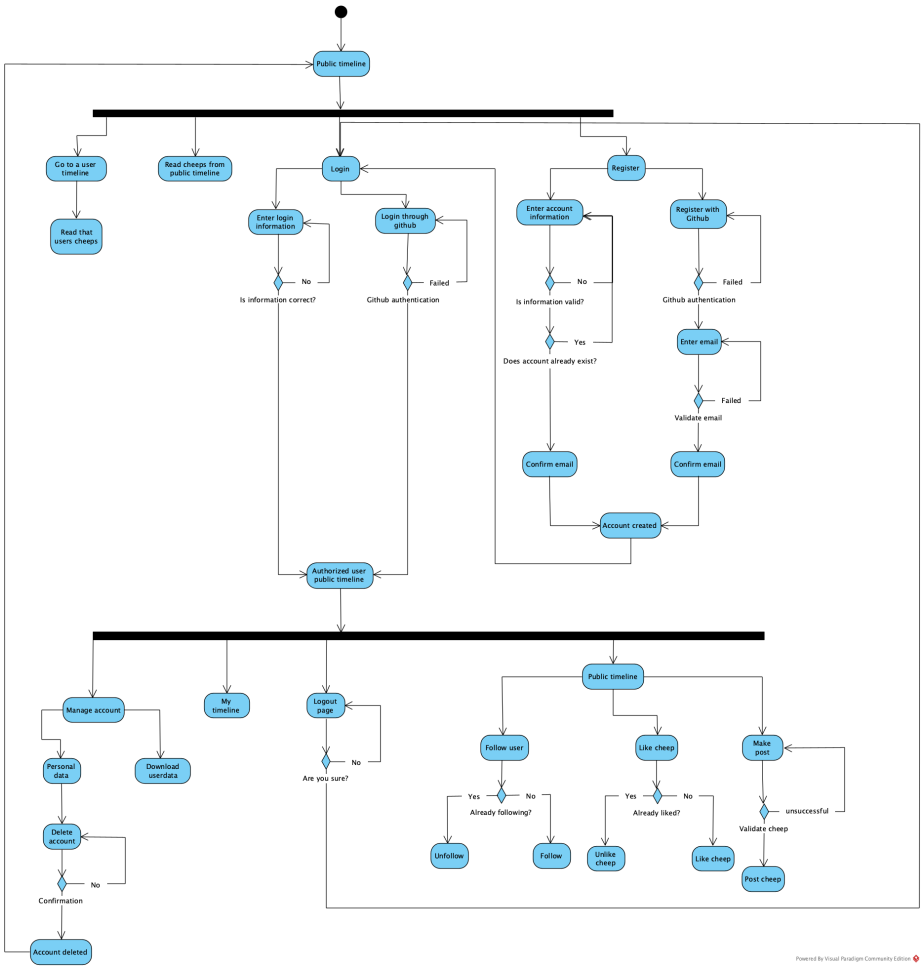
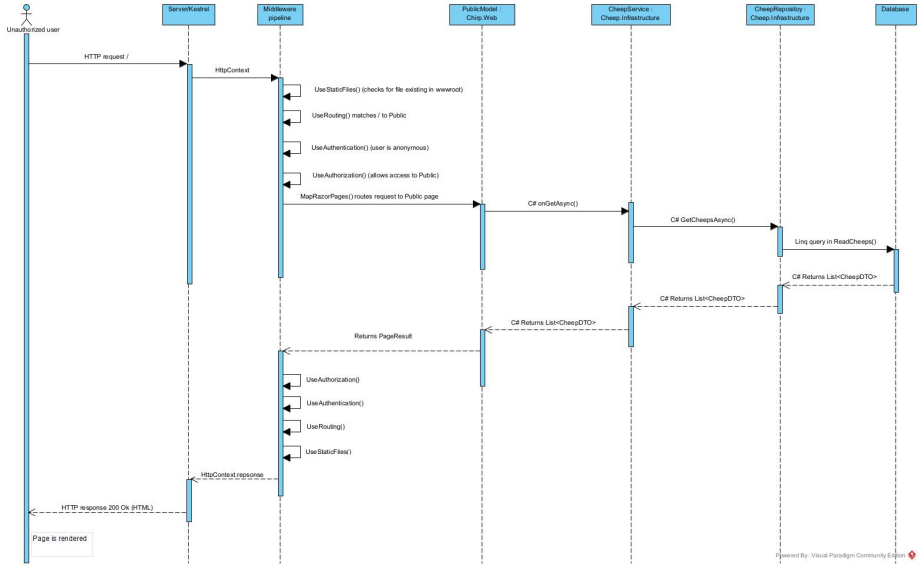


Figure 4: Authorized user activity diagram



1.5 Sequence of functionality/calls through *Chirp!*

This illustrates the flow of messages and data sent through our Chirp application. It is illustrated for an unauthorized user sending an HTTP request to root endpoint, and ending up with a completely rendered web-page returned to the user.



Small comments on the middleware pipeline:
Requests also go through the middlewares UseExceptionHandler(), UseHsts(), UseHttpsRedirection(), When the app is not in development

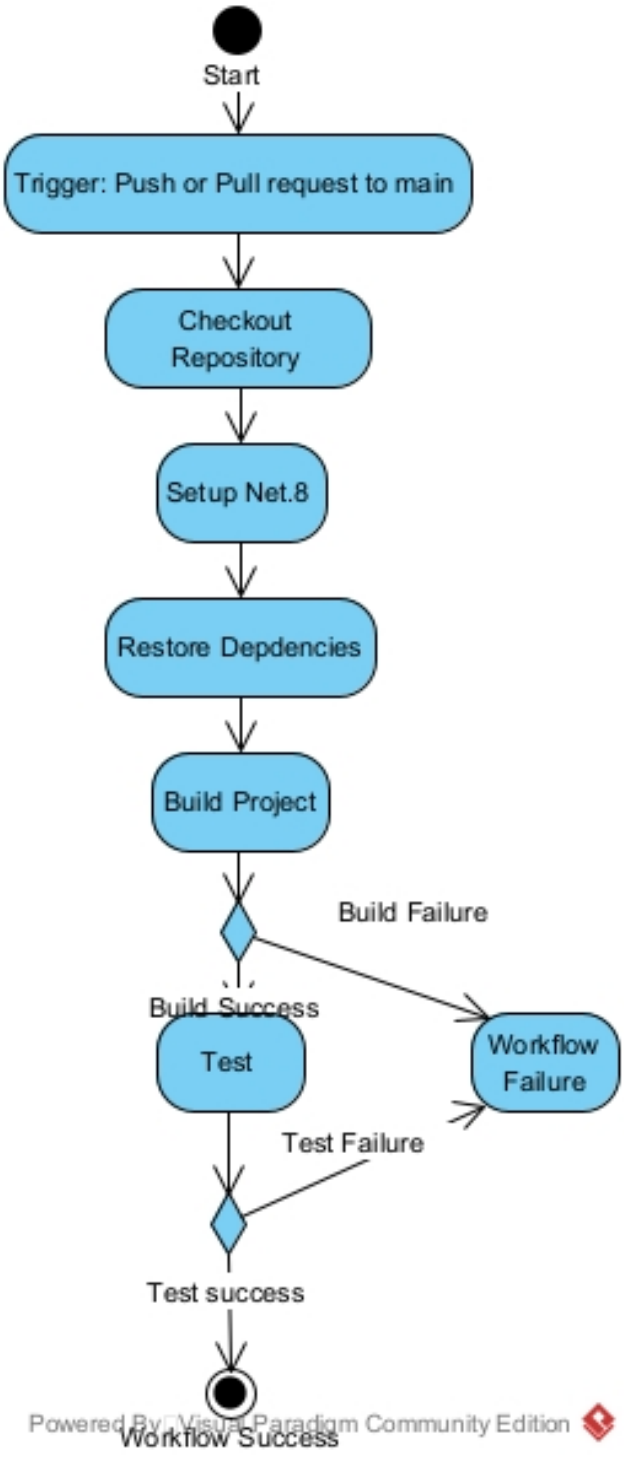
We have chosen to show the process of going through middlewares as self-messages.

The middleware pipeline and Server/Kestrel lifeline is added as lifelines to completely show how the request is handled in ASP.NET (see figure 3.1 p. 32 ASP.NET Core IN ACTION third edition)

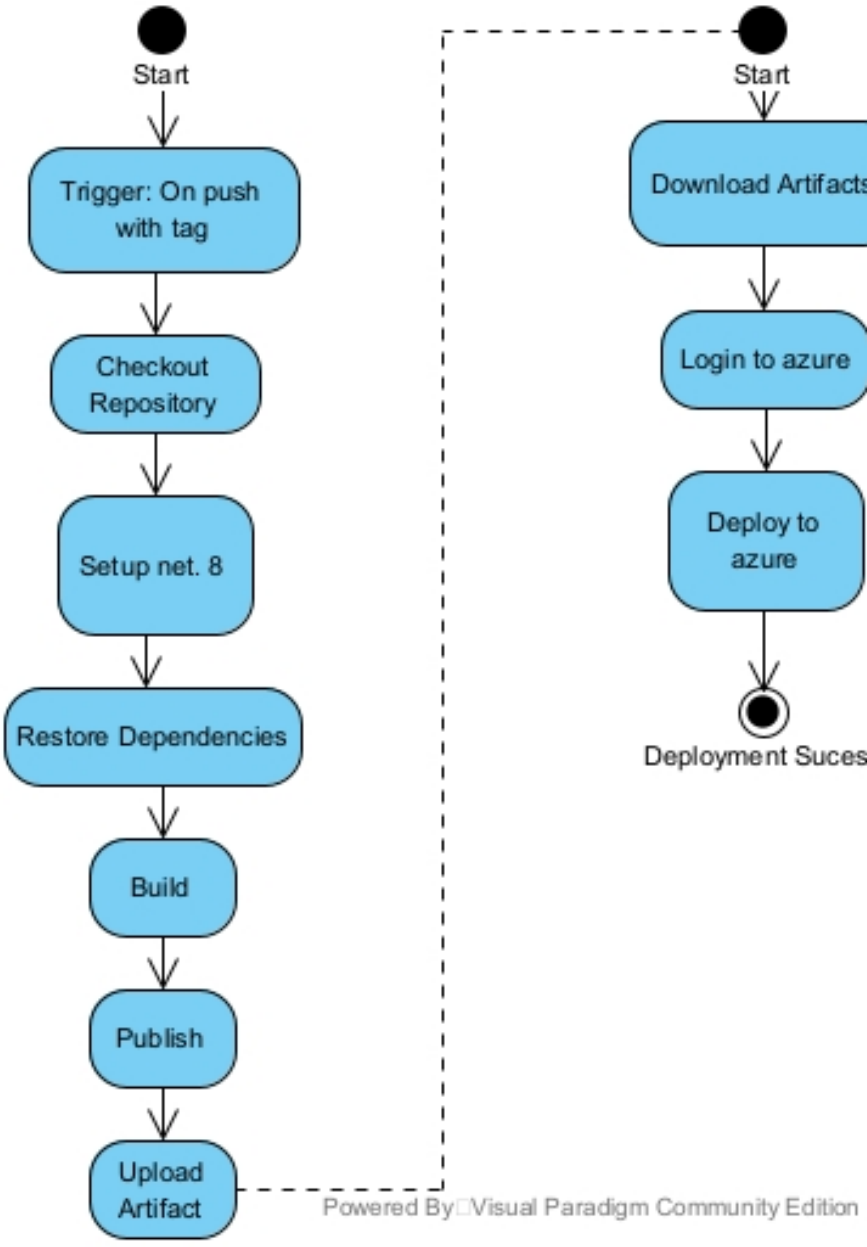
2 Process

2.1 Build, test, release, and deployment

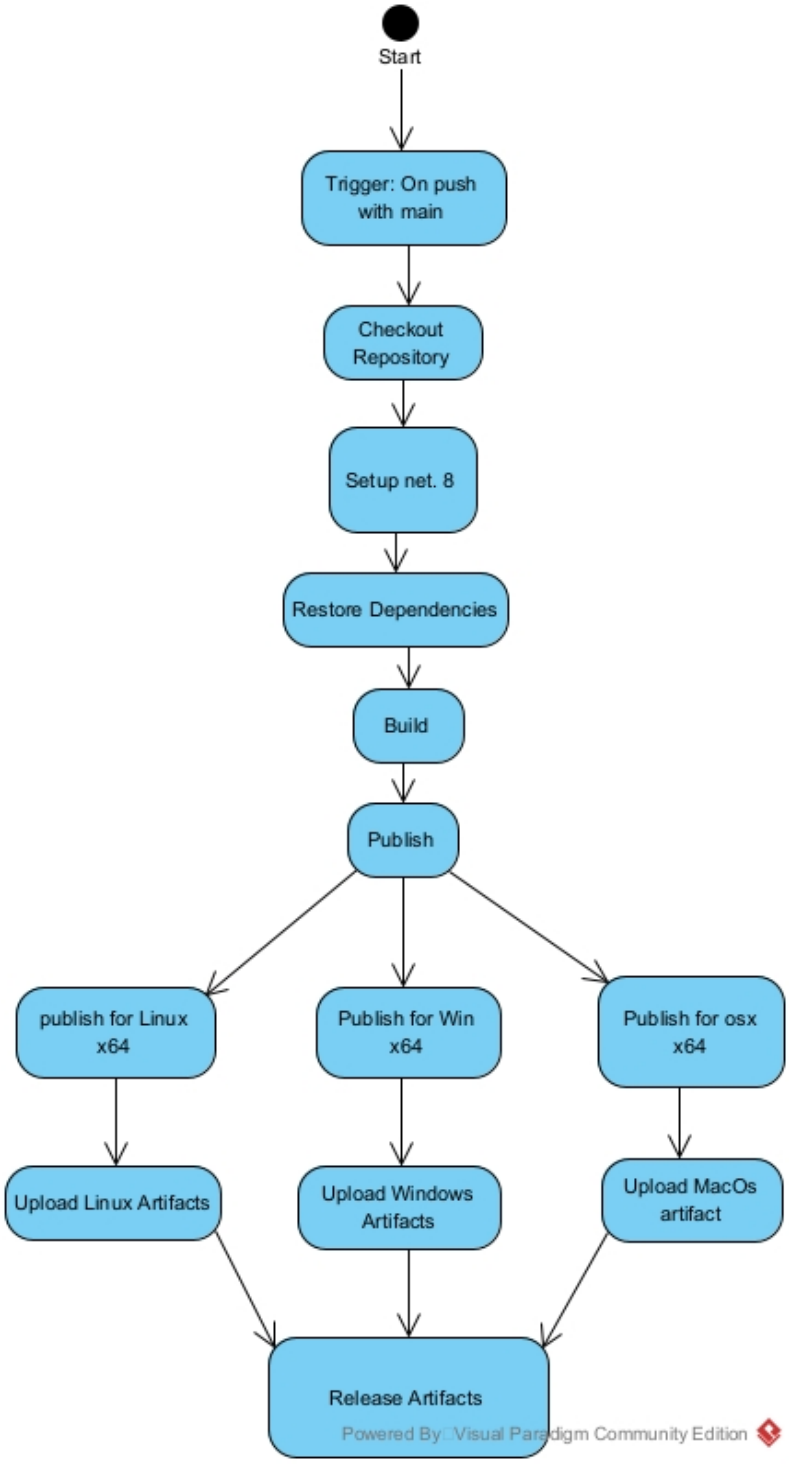
Below are our three activity diagrams showing our different workflows:



Workflow for Building and Testing:

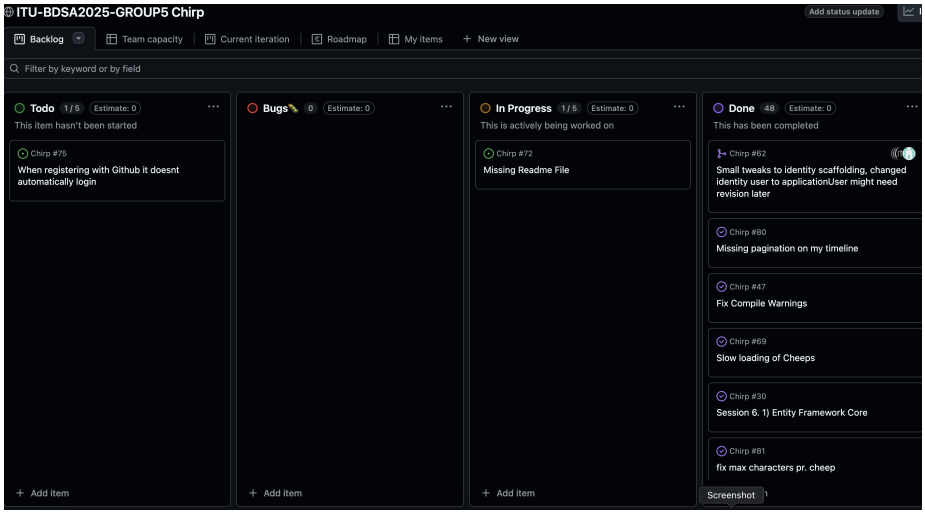


Workflow for deployment for Azure:

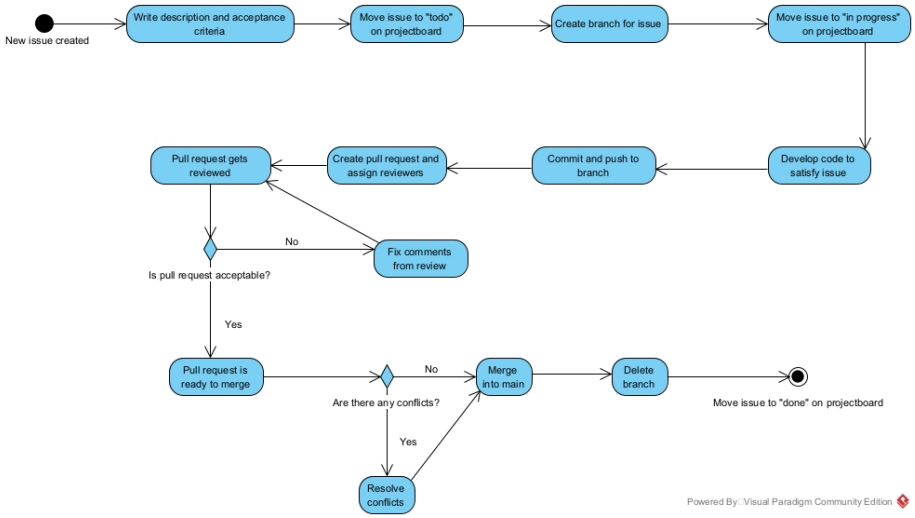


Workflow for making new releases:

2.2 Team work



As of handing in our program, we still have a few unsolved issues. In our case we haven't solved the new feature of automatically logging a user in when they've registered with Github, and are still working on our ReadMe. These are not mandatory features, but shows that you can keep on polishing this project forever, with more and more new features.



This is a illustration which briefly shows the flow of an issue being created to the feature being merged into main.

2.3 How to make *Chirp!* work locally

Install .NET 8.0 SDK (if not already installed)

Clone the repository:
From a terminal run:

To make sure dependencies are installed In Chirp folder run:

Configure user-secrets for OAuth Github:

If your still in the Chirp.Web folder, start the application with:

or you can start the application from root folder Chirp:

Open a browser and go to:

Now you should be at the public timeline for Chirp!

2.4 How to run test suite locally

Install Powershell (if not already installed)

Before we can run all the test we need to install Playwright Go to the Test and then the PlayWrightTests folder:

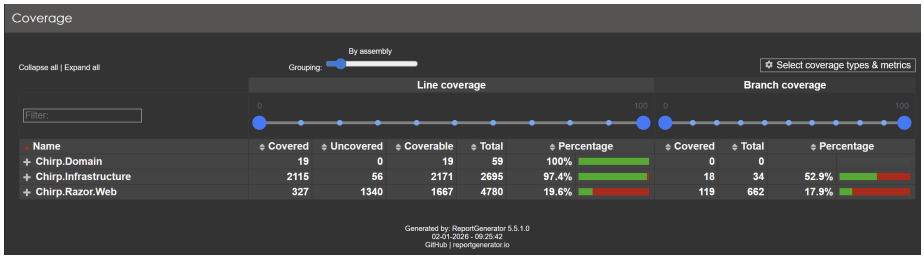
Install PlayWright:

You will need to have 'Powershell' to run this command

Go back into the Test folder

To run all tests:

Test coverage:



In our unit tests and integration tests, we focus on testing core methods, ensuring functionality, and using the Act–Assert pattern for readability.

We used PlayWright for making end-to-end and UI tests. The reason Chirp.Web has low test coverage, it's because of all of the scaffoled files.

3 Ethics

3.1 License

We have chosen the MIT license, which is a commonly used license. The MIT license is a permissive license, which provides more freedom, if there were other

who wanted to reuse the software. This includes: - Commercial use - Modification - Distribution - Private use

We chose a permissive license over a copyleft license, since we are developing our project for educational purposes. Since the project doesn't contain any business-critical logic, we don't need to enforce strict ownership.

3.2 LLMs, ChatGPT, CoPilot, and others

LLMs was used assisting the development of our project. The LLMs that was used was: - ChatGPT - CoPilot - Gemini

LLM's was used for: - Explain and understand ASP.NET core concepts besides the Microsoft documentation. - Some error messages was fed to LLM's to help identify errors. - Write HTML/CSS for the UI. - Assisting in writing of workflows - Assisting in phrasing of documentation - Helping build the WebApplicationFactory for Playwrightests

The responses were genuinely helpful in understanding the ASP.NET core, writing HTML for UI and assisting with workflow. LLM's responses regarding errors, can be very helpful to identify the problem, but you have to be careful when looking at its solutions. In our experience, it's far more reliable to trace through the code yourself and use the LLM's error analysis as a hint rather than a prescription. Blindly following its solutions can lead into a spiral of new errors.

Overall we like to believe that the use of LLM's has helped the development of the project. That said, as mentioned above, we have experienced that you should not blindly follow LLM's solutions, because you can end up in a spiral of new errors and issues. Instead, we experienced, that we benefited more from tracing through the code yourself, or using pen and paper to make it clear for you, how to solve the error.

3.3 Appendix

Name	Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage
− **Chirp.Domain**	**19**	**0**	**19**	**59**	**100%**	**0**	**0**	
Chirp.Domain.Cheep	6	0	6	18	100%	0	0	
Chirp.Domain.CheepDTO	5	0	5	13	100%	0	0	
Chirp.Domain.Follow	5	0	5	16	100%	0	0	
Chirp.Domain.User	3	0	3	12	100%	0	0	
− **Chirp.Infrastructure**	**2115**	**56**	**2171**	**2695**	**97.4%**	**18**	**34**	**52.9%**
Chirp.Infrastructure.CheepDbContext	32	0	32	57	100%	0	0	
Chirp.Infrastructure.CheepRepository	95	12	107	172	88.7%	5	12	41.6%
Chirp.Infrastructure.CheepService	34	5	39	143	87.1%	2	4	50%
Chirp.Infrastructure.DbInitializer	691	0	691	705	100%	4	4	100%
Chirp.Infrastructure.Migrations.CheepDbContextModelSnapshot	322	0	322	358	100%	0	0	
Chirp.Infrastructure.Migrations.InitialMigration	324	2	326	383	99.3%	0	0	
Chirp.Infrastructure.Migrations.MigrationUpdate	551	20	571	653	96.4%	0	0	
Chirp.Infrastructure.UserRepository	44	15	59	128	74.5%	3	6	50%
Chirp.Infrastructure.UserService	22	2	24	96	91.6%	4	8	50%
+ **Chirp.Razor.Web**	**327**	**1340**	**1667**	**4780**	**19.6%**	**119**	**662**	**17.9%**

Figure 5: Test full coverage report