Chirp! Project Report

Table of Contents

1 Introduction

1.1 Design and architecture

1.1.1 Domain model

The following domain model captures the main entities and relationships in Chirp!.

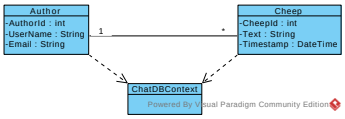

Figure 1: Domain model

The domain model consists of Author and Cheep. Author represents a user in the system and is integrated with ASP.NET Identity. An author can create multiple cheeps, and each cheep belongs to exactly one author. Authors can follow other authors, forming a many-to-many relationship. A cheep contains textual content and a timestamp. All entities are persisted using Entity Framework Core via ChatDBContext.

1.1.2 Architecture — In the small

The following diagram illustrates the organization of the code base and its project dependencies.

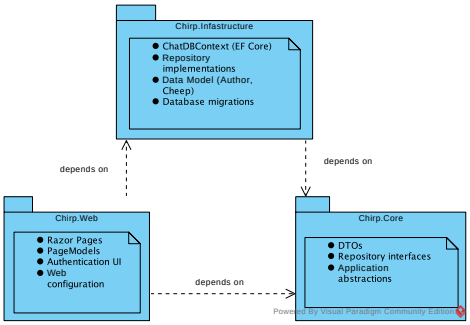

Figure 2: Architecture - in the small

The application follows an onion architecture. The presentation layer is implemented in Chirp.Web and contains Razor Pages, page models, and authentication endpoints. The application layer is represented by service and DTO abstractions defined in Chirp.Core. The domain logic is centered around the Author and Cheep entities. The infrastructure layer is implemented in Chirp.Infrastructure and contains repository implementations, database context, migrations, and ASP.NET Identity integration. Dependencies point inward toward the core and domain layers.

1.1.3 Architecture of deployed application

The following diagram shows the deployed Chirp! system is composed and how components communicate.

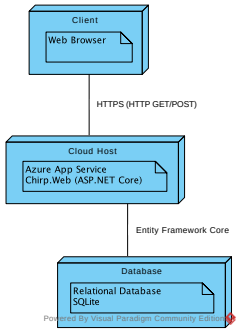

Figure 3: Deployment architecture

Chirp! is implemented as a client-server web application. The client is a web browser that renders Razor Pages generated by the server. The server is an ASP.NET Core application built from Chirp.Web and deployed using GitHub Actions. The server communicates with a relational database through ChatDBContext using Entity Framework Core. All communication between client and server is performed over HTTPS.

1.1.4 User activities

The following activity diagram illustrates a typical user journey through Chirp!.

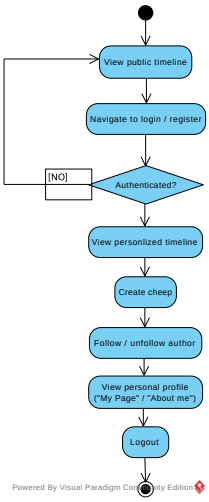

Figure 4: User activities

A non-authenticated user is presented with the public timeline rendered by Public.cshtml. Non-authenticated users can view cheeps but cannot post cheeps or follow other users. The user can authenticate using ASP.NET Identity login functionality. After authentication, the user can post cheeps, follow other authors, and view personalized timelines such as MyPage and FollowingTimeline. Authenticated users can log out at any time.

1.1.5 Sequence of functionality/calls through Chirp!

The following sequence diagram shows the flow of an unauthenticated request to the root endpoint and the data retrieval required to render the page.

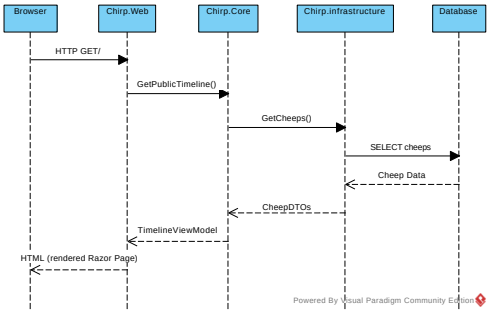

Figure 5: Sequence diagram: GET /

The sequence starts with an HTTP GET request to the root endpoint by a non-authenticated user. The request is handled by a Razor Page in Chirp.Web. The page model invokes repository methods through application-level abstractions. CheepRepository retrieves cheep data from the database via ChatDBContext. The retrieved data is mapped to DTOs and returned to the Razor Page. The Razor Page renders the HTML response, which is sent back to the client browser.

1.2 Process

1.2.1 Build, test, release, and deployment

Build, test, and deployment are automated using the GitHub Actions workflow bdsagroup23chirprazor2025.yml. On each push or pull request, the solution is built using the .NET SDK. The test projects in test/ are executed, including unit and integration tests. If all tests pass, the application is deployed to the hosting environment. A failing build or test step prevents deployment.

1.2.2 Team work

The project board reflects the state of development immediately before hand-in. A small number of tasks remain unresolved, primarily related to optional features and UI polish. All required functionality for the Chirp! application is implemented.

Development starts with the creation of an issue describing a task. A feature branch is created from main. The feature is implemented and tested locally. A pull request is opened and reviewed. After approval, the changes are merged into the main branch.

1.2.3 How to make Chirp! work locally

The repository is cloned from GitHub. Dependencies are restored using dotnet restore. The database is initialized using Entity Framework Core migrations. The application is started using dotnet run from the Chirp.Web project. The application is accessible in a web browser on the configured local port.

1.2.4 How to run test suite locally

To run the test suite locally, execute the following command from the root of the repository:

```
dotnet test
```

The test suite includes: - **Unit tests**: Testing individual components in isolation (e.g., repository methods). - **Integration tests**: Testing the interaction between multiple components and the database. - **UI/End-to-End tests**: Testing the application from a user's perspective using Playwright.

1.3 Ethics

1.3.1 License

The application is released under the MIT License.

1.3.2 LLMs, ChatGPT, CoPilot, and others

ChatGPT was used during development as a support tool. It was primarily used for explanations, code suggestions, and assistance with documentation. The responses were helpful for understanding concepts and accelerating development. All generated content was reviewed and adapted manually. Overall, the use of LLMs reduced development time without replacing developer judgment.