Chirp! Project Report

ITU BDSA 2025 Group 19

Hans Voldby Larsen havl@itu.dk

1 Design and Architecture of *Chirp!*

1.1 Domain model

The domain model is a consists of the classes in the core domain of interest, and the relationships between them. The domain model is visually shown in figure @fig:domain-model as a UML diagram. Only the core entities are shown so other classes of the app concerning eg. infrastructure or web pages are not part of the domain model.

The core domain consists of the entities Author, Cheep and Follow, where an Author can publish multiple Cheeps and each Cheep is authored by exactly one Author.

Authors can follow other authors, which is modeled through the Follow entity representing the follower–followee relationship.

Authentication and authorization are handled via ASP.NET Identity and are therefore not part of the core domain model. The property ApplicationUserId in the class Author (which is used to connect a user indentity with an Author) is therefore not included in the Domain model.

1.2 Architecture — In the small

Figure below illustrates the internal architecture of the Chirp! application. The Solution follows a layered architecture with a clear separation between presentation, application, domain, and infrastructure concerns.

The **domain layer** (project Chirp.Domain) contains the core domain entities and is completely independent of other layers. The class Author in the domain has a property called ApplicationUserId and is used to associate a domain author with an authenticated user. This property is represented as a primitive value and does not introduce a dependency on the authentication framework

The **application layer** (project Chirp.Application) defines service interfaces and operates exclusively on data transfer objects (DTOs), and therefore does

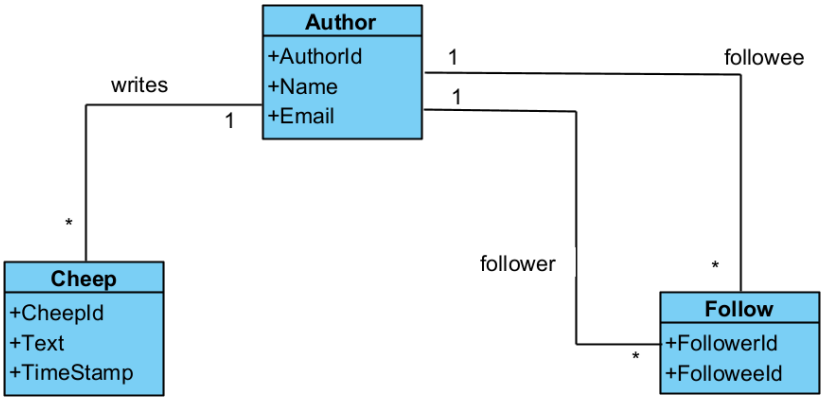

Figure 1: Illustration of the *Chirp!* data model as UML class diagram.

not depend directly on the domain layer. Data Transfer Objects (DTOs) are used to decouple the application and presentation layers from the domain model by exposing only the data required for a specific use case.

The **infrastructure layer** (project Chirp.Infrastructure) provides concrete implementations of application interfaces and provides the persistence of data using entity framework.

The **presentation layer** (project Chirp.Razor) contains the Razor Pages responsible for handling user interaction, as well as the Program.cs entry point where the application is configured, dependencies are registered, and authentication and is set up. Ideally the Razor pages should only depend on the interfaces and DTOs in the application layer. This was not quite achieved so there are some dependence on the Domain classes as well.

Figure Y complements the onion architecture overview by showing a UML package diagram that maps the architectural layers to the concrete project structure and illustrates the dependency relationships between them.

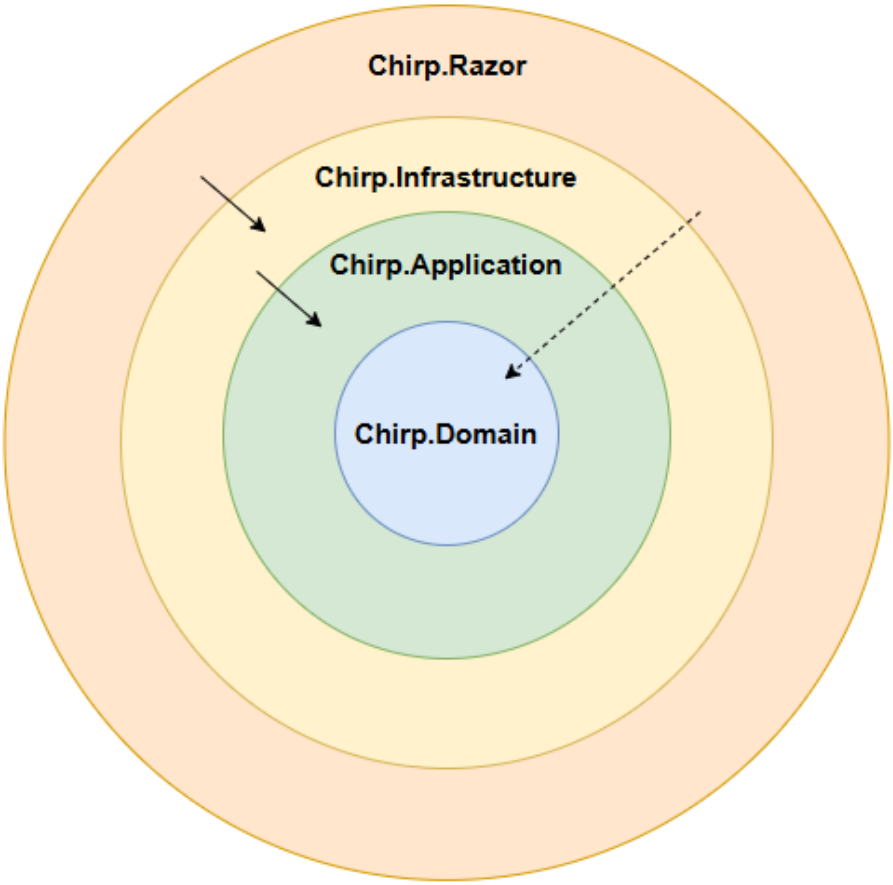

Figure 2: Illustration of the *Chirp!* onion Archhitecture.

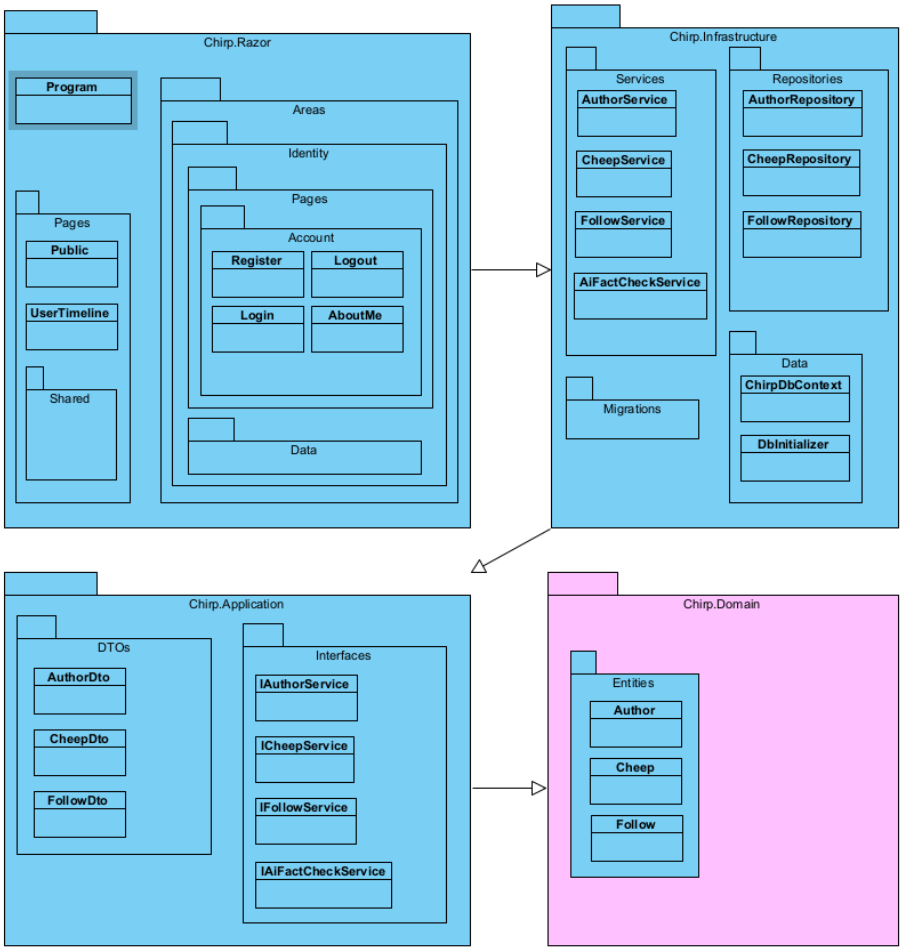

Figure 3: Illustration of the *Chirp!* Archhitecture.