UML Diagrams

Candidate Number: 3

November, 2025

1 Data Model

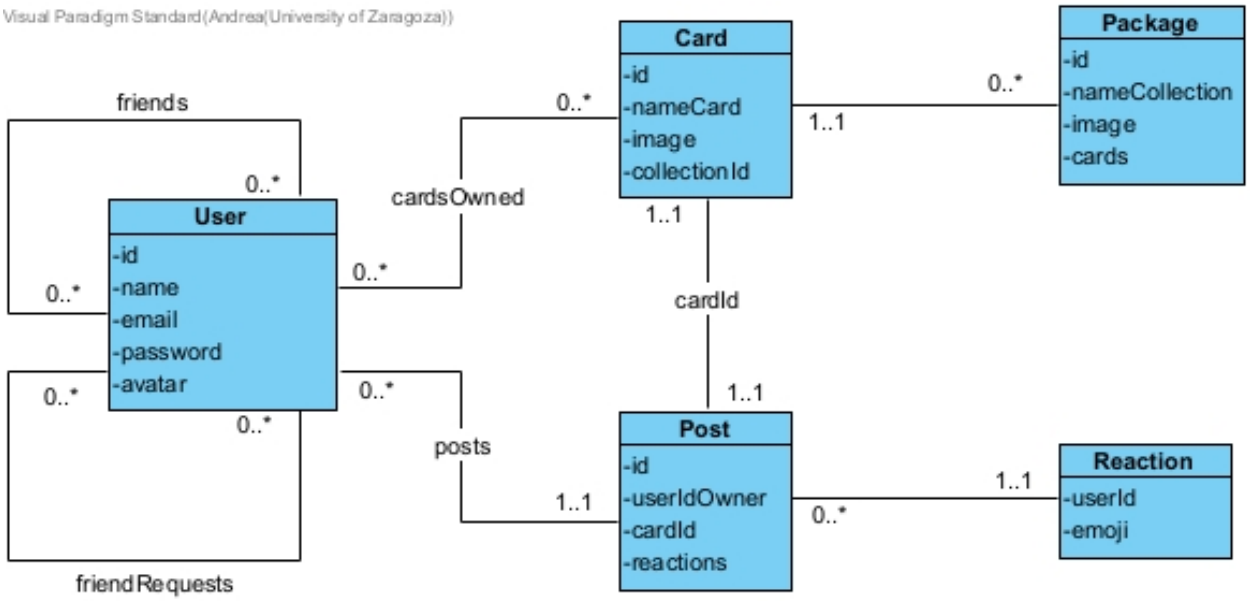

Figure 1: Data Model Class Diagram

The diagram represents the main data structures used in the back-end and how the system manages information between users, cards, packages, posts, and reactions. The User entity is central because the back-end constantly handles user accounts and friendships. Each user can own several Card objects, which come from different Packages. This structure allows the back-end to store collections efficiently and recover them quickly when a user opens a pack or views their collections. The Post entity connects the social and collecting parts of the system. When a user posts a card, the post is linked to both the owner and the specific card. Each post can be reacted by different users, for that, the Reaction saves the user and the emoji reacted. Overall, the diagram shows how the back-end organizes and moves data in a predictable and secure way, supporting the core game and social features.

2 Software Architecture: Layered Architecture

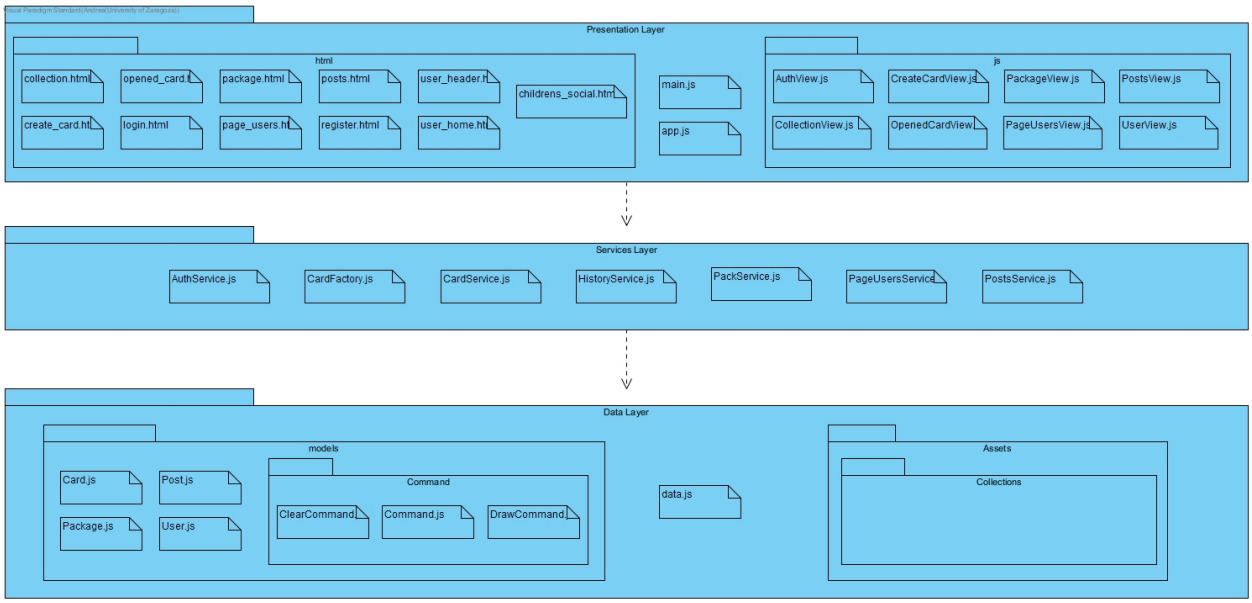

Figure 2: Layered Architecture Diagram

This architecture is organized into three main layers. The Presentation Layer contains the HTML pages and JavaScript view classes responsible for rendering the interface and capturing user interactions. These views communicate with the Service Layer, which implements the application's service logic through different components. This layer also includes a CardFactory class, implementing the Gang of Four Factory Method pattern to create card objects dynamically based on their collection. The Data Layer defines the data models (User, Card, Package, Post), and also includes a Command submodule that encapsulates user actions such as drawing or clearing cards, the dummy data of the UI and several images in assets. The separation of layers supports maintainability, scalability, and a clear flow of data across the system.

3 Gang of Four design pattern: Factory Method

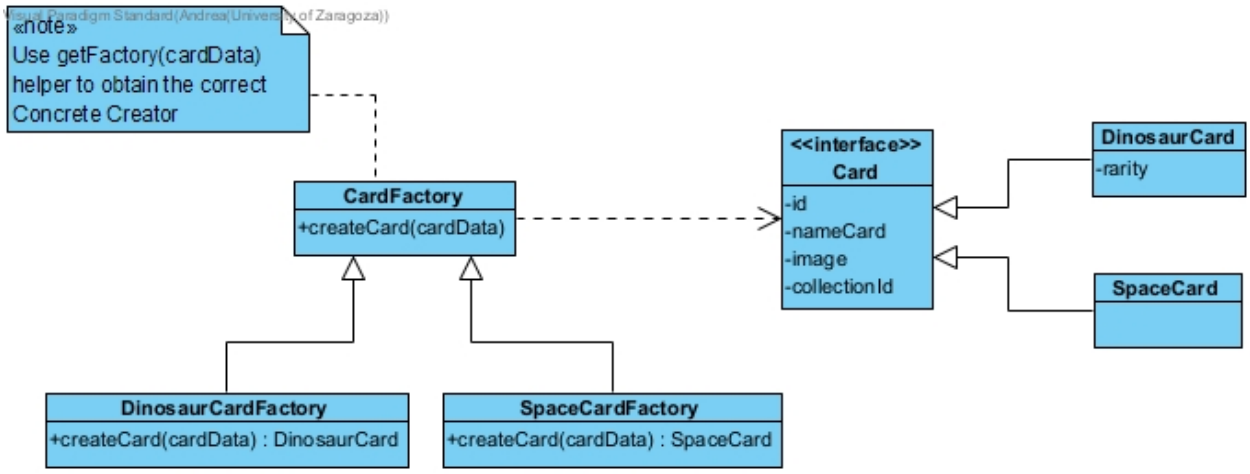

Figure 3: Factory Method Class Diagram

This diagram shows the Factory Method pattern applied in the system to create card objects dynamically. CardFactory is the abstract creator that defines the createCard() method. DinosaurCardFactory and SpaceCardFactory are concrete creators that instantiate DinosaurCard and SpaceCard, respectively. Card is the abstract product, while DinosaurCard and SpaceCard are concrete products. This design decouples card creation from package management, enabling the addition of new collections without changing existing logic, improving maintainability and scalability.