# **Protocol for the Monster Card Game Application**

**1. App Design Overview**

The Monster Card Game project is designed as a server-based application that manages user interactions through controllers and business logic in services. It models core entities such as cards, decks, and battles and uses an HTTP server to process client requests.

**2. Main Design Decisions**

* **Custom HTTP Server**: The application uses its own implementation of an HTTP server, derived from a given Server Base, which is structured into request and response handling, routing, and utilities. This design allows for flexibility and control over the server behavior.
* **MVC Pattern**: The app follows an MVC-like structure:
  + **Controllers**: Handle HTTP requests and map them to the appropriate services.
  + **Services**: Contain the business logic (e.g., handling users, cards, and game-related operations).
  + **Models**: Represent the core entities like User, Card, MonsterCard, and Deck.

**3. Code Structure**

The project is organized into several packages:

* **Controllers**: Responsible for request handling (UserController, BattleController, DeckController).
* **Models**: Define game entities (Card, MonsterCard, SpellCard, User).
* **Services**: Business logic is abstracted into services (UserService, BattleArena).
* **HTTP Server**: The custom HTTP server manages low-level request and response handling, along with routing (Request, Response, Router).

**4. Key Classes**

* **Main.java**: The main entry point of the application initializes game entities like MonsterCard, SpellCard, and creates users and a package of cards for testing purposes.
* **UserController.java**: Implements functionality for retrieving and managing user data. It uses a dummy data access layer (UserDummyDAL), but the comment suggests that the repository pattern should be used for persistent data handling.

**5. Class Diagram (Structure Overview)**

* **Main**: Initializes the game and sets up core objects.
* **Controllers**: Classes such as UserController, BattleController, and DeckController map HTTP requests to game logic.
* **Models**: Game entities are structured into classes like Card, MonsterCard, and Deck.
* **Services**: Handles business operations such as battles (BattleArena), user management (UserService), and session control (SessionService).

**Class Diagram:**

**Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung**

**6. Design Decisions**

* **Singleton Pattern**: The UserDummyDAL uses a singleton pattern to ensure only one instance of the dummy data access layer is used across the application.
* **In-Memory Database Setup**: Dummy data is used in UserController for testing purposes, allowing easy development and testing of functionality without requiring a database connection.