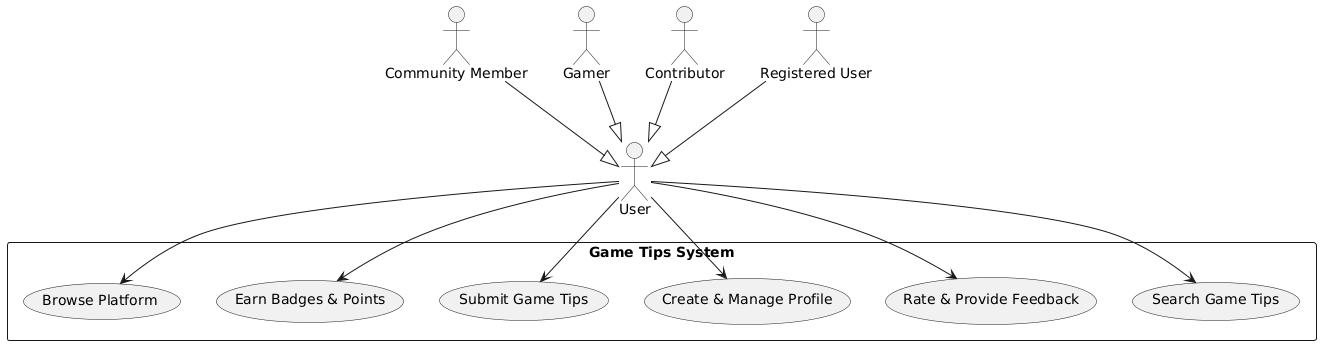
# Based on our group project's user stories, these are the required diagrams below:

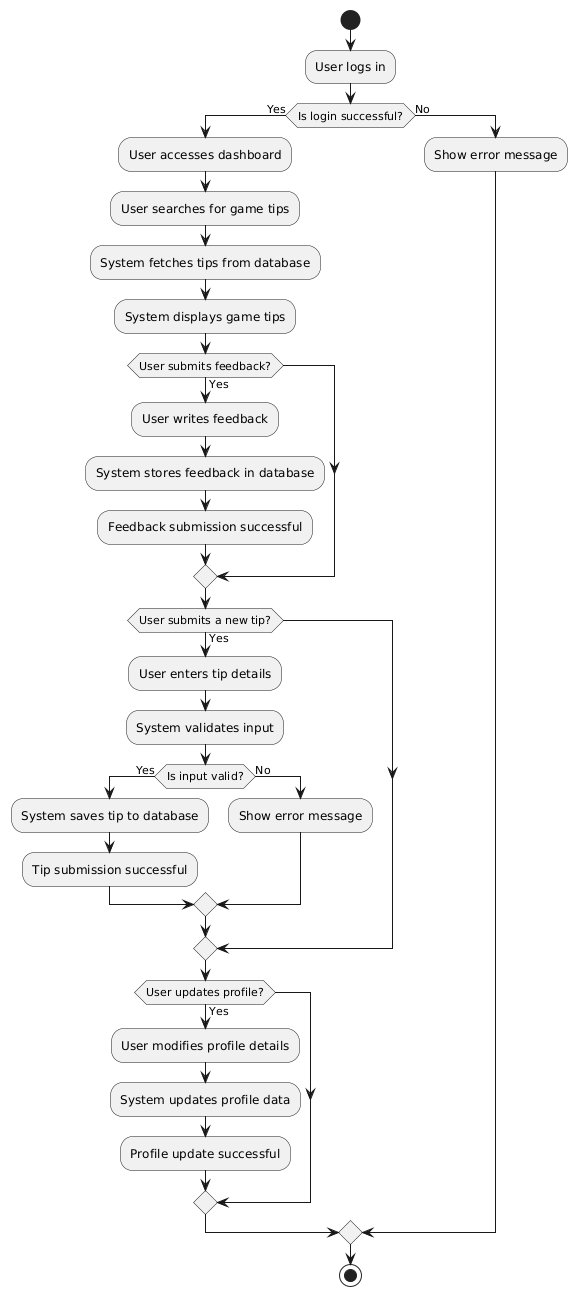
## **1. Use Case Diagram**

All user types (Community Member, Gamer, Contributor, and Registered User) are generalized under a single **User actor** while still interacting with the system in the same way.



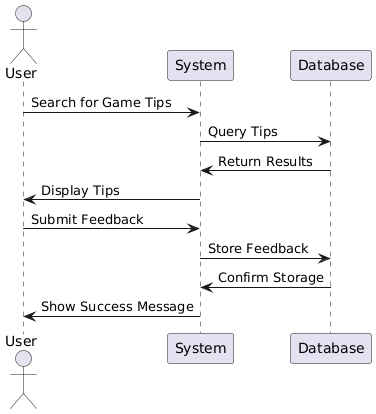
## **2. Activity Diagram**

Activity diagrams illustrate the flow of processes within the system, showcasing decision points, loops, and different possible user actions in a structured format.



## **3. Sequence Diagram**

Sequence diagrams represent the interaction between different components over time, showing how messages and actions flow between users, the system, and the database.

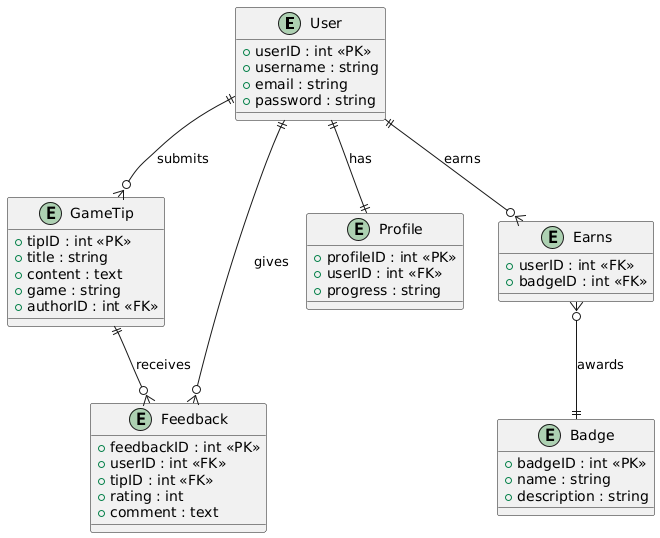


## **4. Entity-Relationship Diagram (ERD) and Class Diagram**

The ERD and Class Diagram are essential in defining the system's structure, but they serve different purposes. Below are the breakdowns of each:

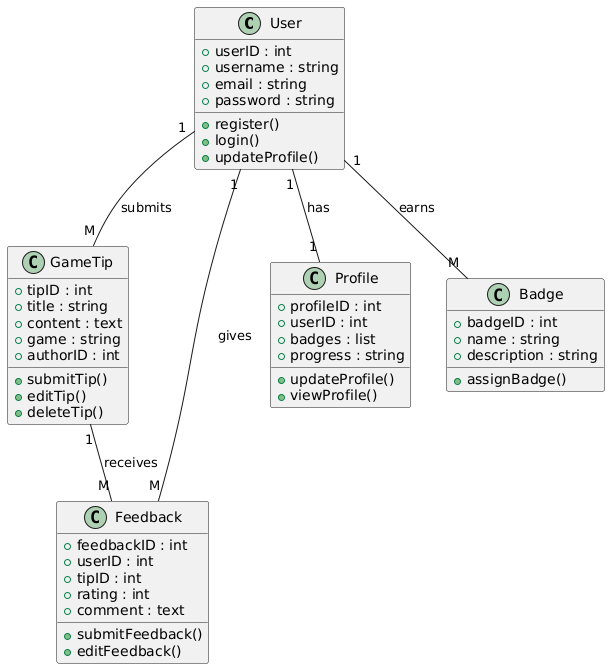
### **Entity-Relationship Diagram (ERD)**

The ERD is used for database modeling, showcasing how different entities (tables) relate to each other using primary and foreign keys. It provides a structured layout of data storage relationships.



### **Class Diagram**

Class diagrams define the system’s structure in an object-oriented way, displaying attributes, methods, and the relationships between different classes.



## **5. Differences Between ERD and Class Diagram**

While both diagrams represent system structure, they serve different purposes. The ERD is primarily focused on database design, whereas the class diagram is essential for object-oriented programming and system architecture.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Class Diagram** | **Entity-Relationship Diagram (ERD)** |
| **Purpose** | Represents the object-oriented structure of the system. | Represents the database structure and relationships. |
| **Focus** | Focuses on **objects, attributes, methods, and associations.** | Focuses on **entities, attributes, and their relationships.** |
| **Contains Methods?** | Yes, includes **methods/functions** (e.g., register (), login ()). | No methods, only **data attributes**. |
| **Relationships** | Uses **associations (solid lines)** and **multiplicities (1:M, 1:1, etc.).** | Uses **crow’s foot notation** to represent **1:1, 1:M, and** **M:N** relationships. |
| **Primary & Foreign Keys** | Does not explicitly define **Primary Keys (PKs) or Foreign Keys (FKs)**. | Clearly marks **Primary Keys (PK)** and **Foreign Keys (FK)** for database design. |
| **Many-to-Many (****M:N) Relationships** | Represented using **association classes or collections**. | Requires an **intermediate (junction) table** (e.g., Earns between User and Badge). |
| **Example Representation** | + register () (Method inside User class) | + userID : int <<PK>> (Primary Key inside User entity) |
| **Use in Development** | Helps in **object-oriented programming (OOP) implementation**. | Helps in **relational database modeling**. |

These diagrams collectively help in designing and understanding the system, ensuring clarity in both database and object-oriented implementation.