GRASP Pattern Application

for

Soccer Live Platform

Version: 1.0

Prepared by: Usman Awan | 22F-3378

Organization: FAST NUCES CFD

Date: 9 Dec, 2024

Contents

What is GRASP?

GRASP stands for **General Responsibility Assignment Software Patterns**. It is a set of nine object-oriented design principles that help in assigning responsibilities to classes and objects in a way that leads to a clean, maintainable, and extensible system. The GRASP principles guide how to allocate responsibilities to different components of a system, ensuring that the design is modular, flexible, and easy to manage. The GRASP patterns are designed to help developers make decisions about **class responsibilities**, **object collaboration**, and **communication** between classes. These principles serve as a set of best practices for object-oriented design, helping to create systems that are easier to understand, modify, and extend.

Why Do We Need to Use GRASP?

1. **Enhances Maintainability**: GRASP ensures classes have clear, focused responsibilities, making the system modular and easier to maintain.
2. **Improves Flexibility**: It encourages designs that can adapt to future requirements, with principles like Polymorphism and Low Coupling enabling easy extensions.
3. **Reduces Complexity**: GRASP simplifies the design by focusing each class on a single responsibility and minimizing dependencies.
4. **Facilitates Reusability**: By applying Polymorphism and Pure Fabrication, classes become more reusable in different contexts.

Why Are We Using GRASP in the Soccer Live Project?

1. **Clear Responsibilities**: GRASP ensures each component (e.g., MatchDataProcessor, NotificationController) handles specific tasks, simplifying maintenance.
2. **Scalability and Flexibility**: GRASP helps the system grow with new features, such as adding match types using Polymorphism.
3. **Improved Communication**: Low Coupling and Indirection ensure components interact efficiently, reducing dependencies.
4. **Future-Proofing**: GRASP principles ensure the system is easy to extend, supporting growth in users and data.
5. **Efficiency and Performance**: GRASP optimizes components, like the StreamingEngine, by focusing on specific tasks, improving performance.

GRASP Pattern Application in Soccer Live Platform

1. Creator:

Explanation:
The Creator pattern assigns the responsibility for creating objects to the class that has the necessary data to do so or is directly involved in the aggregation of objects.

Application in Soccer Live:

- **Class Responsible:** Admin
- **Responsibility: The Admin is responsible for creating UserProfile objects when a new user registers.**
- **Reasoning:** From the **Class Diagram**, the Admin is the class that deals with user registrations. During this process, the system collects user information (e.g., userId, email, password) and creates a UserProfile for the user. This follows the Creator pattern, as the Admin has all the necessary data for user profile creation.
- **Improvement:** By adhering to the Creator principle, the Admin can efficiently manage the creation of user profiles, ensuring proper encapsulation and making the system easier to maintain. This design also enhances flexibility, as the user registration process can be modified without disrupting other parts of the system.

---

2. Information Expert:

Explanation:
The Information Expert pattern assigns responsibility to the class that has the necessary information to perform a task or operation.

Application in Soccer Live:

- **Class Responsible:** MatchDataProcessor
- **Responsibility: The MatchDataProcessor class is responsible for providing match statistics, real-time updates, and detailed match information.**
- **Reasoning:** Referring to the **Class Diagram**, MatchDataProcessor is the class that directly handles and processes data related to the match, such as score updates, player statistics, and

match events. This class is best positioned to process and format match data, making it the expert on the information it manages.

- **Improvement:** By following the Information Expert principle, MatchDataProcessor ensures that match-related operations are handled by the class with the most relevant and up-to-date data. This reduces redundancy and improves maintainability by centralizing responsibility for match data processing.

---

3. Controller:

Explanation:
The Controller pattern assigns the responsibility of controlling user interactions, often receiving inputs and delegating tasks to other classes.

Application in Soccer Live:

- **Class Responsible:** NotificationController
- **Responsibility: The NotificationController handles events related to notifications such as match updates, administrative alerts, or important system messages.**
- **Reasoning:** In the **Class Diagram**, the NotificationController is responsible for managing notification-related tasks. It processes user requests for notifications and manages the flow of messages regarding new updates (e.g., when a match starts or a goal is scored). The NotificationController class coordinates actions triggered by users and ensures proper notification handling.
- **Improvement:** This design maintains a clear separation between the user interface and business logic by centralizing notification management in the NotificationController. This separation makes it easier to modify or extend notification functionalities without impacting other parts of the system.

---

4. Low Coupling:

Explanation:
Low Coupling minimizes dependencies between classes to ensure that changes to one class do not affect others.

Application in Soccer Live:

- **Class Responsible:** StreamingEngine

- **Responsibility: The StreamingEngine minimizes dependencies on third-party APIs (e.g., payment gateways, codec providers).**
- **Reasoning:** According to the **Class Diagram**, StreamingEngine is responsible for streaming matches, but it does not directly interact with external services such as payment systems or codec providers. Instead, it interfaces with abstracted APIs (e.g., through adapters or facades) to maintain loose coupling with these services.
- **Improvement:** By ensuring that the StreamingEngine does not directly rely on external APIs, the system can easily accommodate changes in third-party services (such as new payment gateways or streaming technologies) without significant rework. This promotes flexibility and scalability.

---

5. High Cohesion:

Explanation:
High Cohesion means that a class should only have responsibilities that are closely related to each other. It improves maintainability and reduces complexity by keeping functionality focused within specific classes.

Application in Soccer Live:

- **Class Responsible:** StreamingEngine
- **Responsibility: The StreamingEngine is responsible solely for streaming-related tasks, such as video playback, stream quality adjustments, and data transmission.**
- **Reasoning:** From the **Class Diagram**, StreamingEngine is only responsible for managing the streaming functionality and nothing else. It does not handle tasks related to user profiles or notifications, ensuring high cohesion within the class.
- **Improvement:** By following the High Cohesion principle, the StreamingEngine class remains simple, easier to maintain, and less prone to errors. This enhances the system's extensibility, as changes to the streaming functionality can be made independently without affecting other features like user management or notifications.

---

6. Polymorphism:

Explanation:
Polymorphism allows objects of different types to be treated as objects of a common superclass, enhancing flexibility and extensibility by enabling different behaviors under a unified interface.

Application in Soccer Live:

- **Class Responsible:** Match
- **Responsibility: The Match class can represent different types of matches (e.g., LiveMatch, ReplayMatch, HighlightMatch).**
- **Reasoning:** Based on the **Class Diagram**, the Match class can have different subclasses that represent different match types. For instance, LiveMatch might include real-time updates and streaming features, while ReplayMatch might involve video playback and highlights.
- **Improvement:** Using polymorphism, the system can easily introduce new match types (e.g., FantasyMatch or ArchiveMatch) without altering the core match logic. This ensures the system remains flexible and extensible, enabling new features without disrupting existing functionality.

---

7. Pure Fabrication:

Explanation:
Pure Fabrication involves creating a class that does not represent a real-world entity but is needed to support the design and functionality of the system.

Application in Soccer Live:

- **Class Responsible:** ReportGenerator
- **Responsibility: The ReportGenerator class is responsible for generating reports about user activity, system performance, and other analytics.**
- **Reasoning:** ReportGenerator does not map to a physical entity or real-world object but is essential for the system's operation. It can collect and format data about user interactions, such as popular matches, user preferences, or system load.
- **Improvement:** Creating a ReportGenerator as a pure fabrication improves system monitoring and analytics without overloading core classes with unrelated responsibilities. This enables better decision-making based on system metrics and user behavior.

---

8. Indirection:

Explanation:
Indirection introduces an intermediary class to mediate communication between components, simplifying complex interactions and reducing direct dependencies.

Application in Soccer Live:

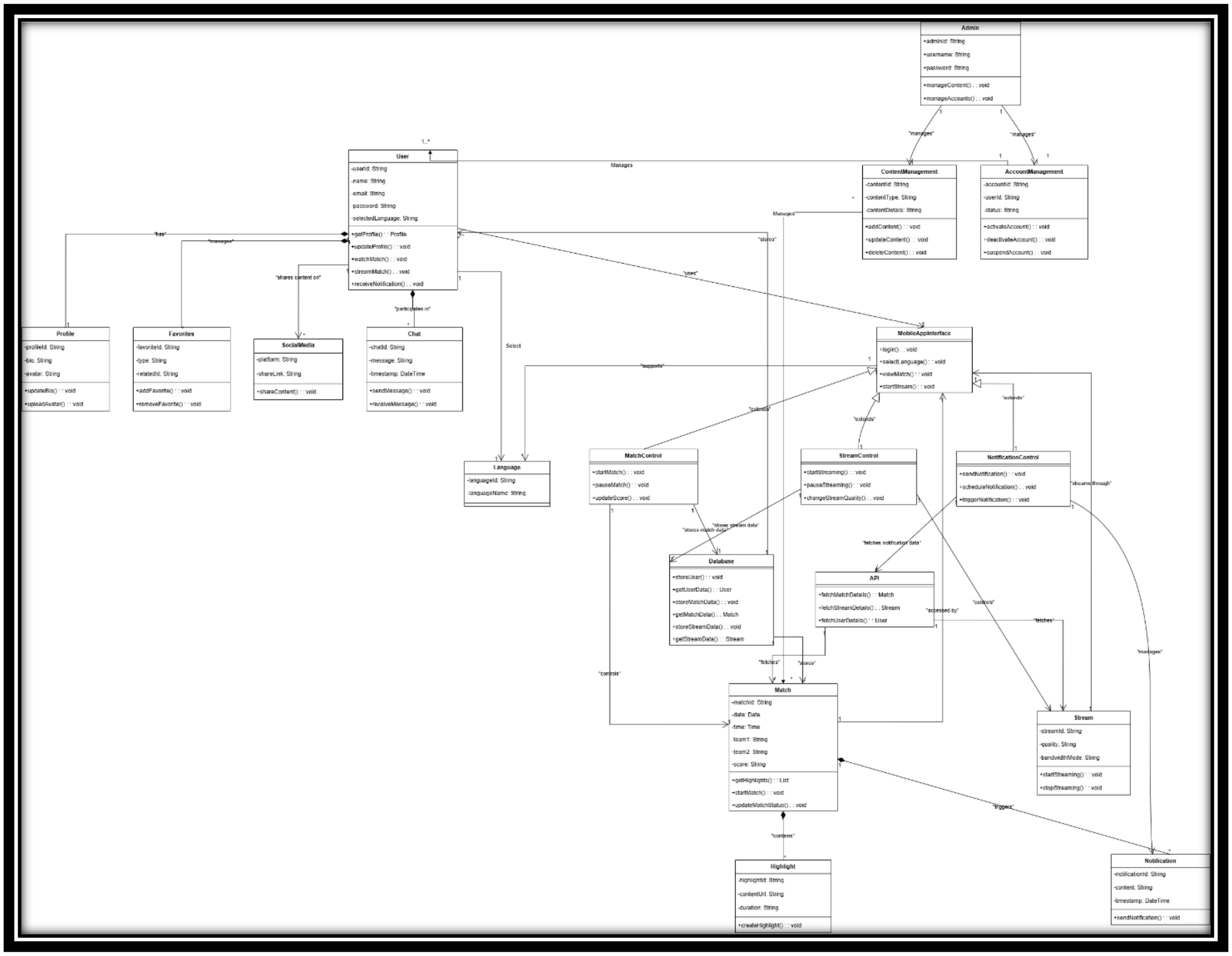
- **Class Responsible:** APIGateway
- **Responsibility: The APIGateway acts as an intermediary between the Soccer Live platform and external services (e.g., third-party match statistics API, payment systems).**
- **Reasoning:** From the **Class Diagram**, the APIGateway sits between the platform and external services, ensuring secure and efficient communication with external resources. For example, it handles requests for live match statistics or user payments.
- **Improvement:** Using Indirection with an APIGateway reduces the dependency between the core platform and third-party services. This simplifies security management and allows the platform to integrate new services without direct modification of core components.

Conclusion:

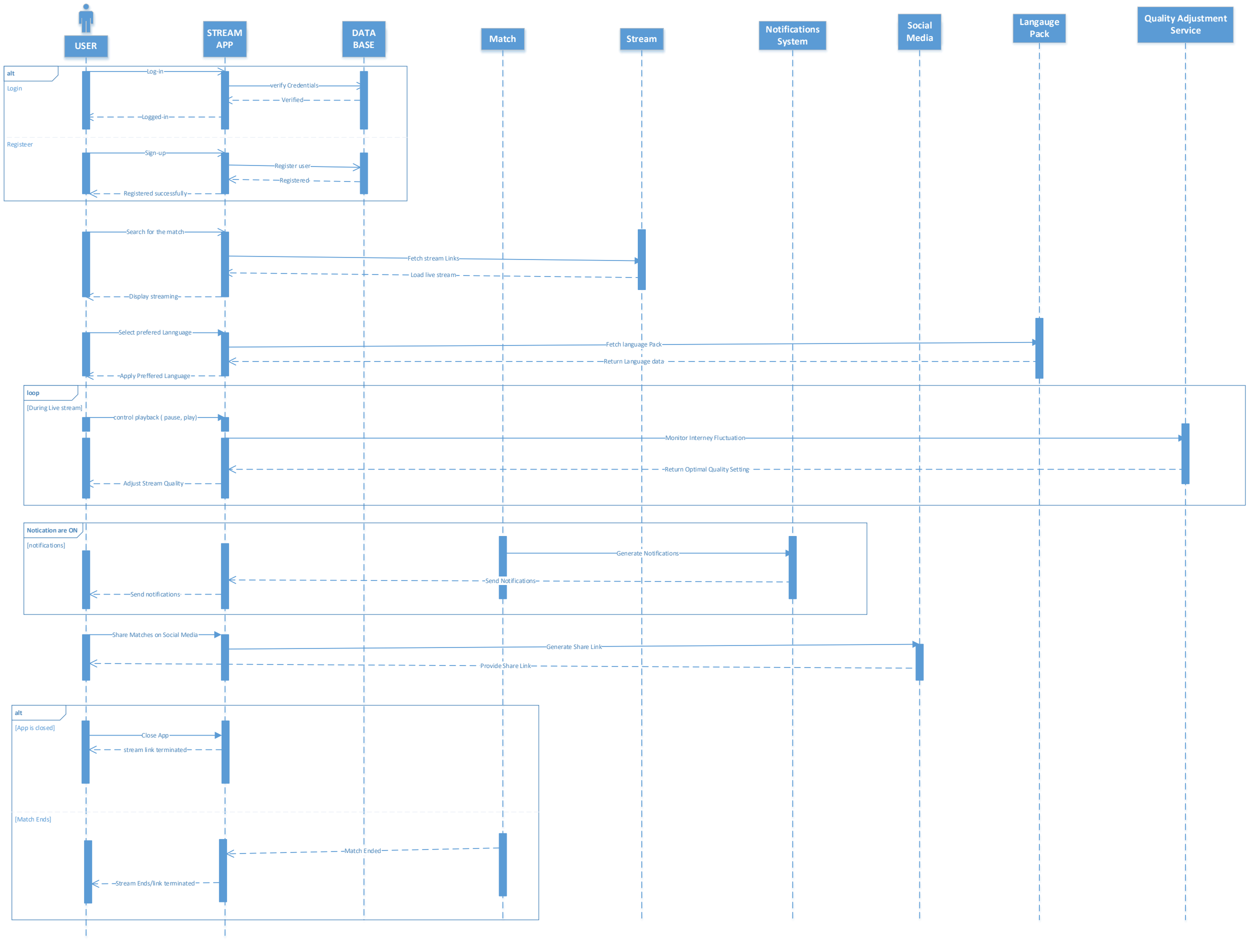
This GRASP analysis demonstrates the application of design principles that optimize the **Soccer Live** platform's architecture. By applying the **Creator**, **Information Expert**, **Controller**, **Low Coupling**, **High Cohesion**, **Polymorphism**, **Pure Fabrication**, and **Indirection** patterns, we ensure the system is scalable, maintainable, and flexible. These principles are reflected in the **class diagram**, which illustrates how these patterns are implemented within the system components, and the **sequence diagram**, which shows how they interact during key system processes (e.g., match streaming, notifications, and user management).

By adhering to these GRASP principles, the **Soccer Live** platform is well-prepared for future growth, including new match types, third-party integrations, or advanced features, without compromising the system's core functionality. This approach enhances the system's extensibility, reduces complexity, and ensures that the platform can adapt to new requirements seamlessly.

Class Diagram



Sequence Diagram


USER
STREAM APP
DATA BASE
Match
Stream
Notifications System
Social Media
Langauge Pack
Quality Adjustment Service
alt
Login
Log-in
verify Credentials
Verified
Logged-in
Registeer
Sign-up
Register user
Registered
Registered successfully
Search for the match
Fetch stream Links
Load live stream
Display streaming
Select prefered Lannguage
Fetch language Pack
Return Language data
Apply Preffered Language
loop
[During Live stream]
control playback (pause, play)
Monitor Interney Fluctuation
Return Optimal Quality Setting
Adjust Stream Quality
Notication are ON
[notifications]
Generate Notifications
Send Notifications
Send notifications
Share Matches on Social Media
Generate Share Link
Provide Share Link
alt
[App is closed]
Close App
stream link terminated
[Match Ends]
Match Ended
Stream Ends/link terminated