Application Structure Software Architecture Patterns

This document will explore the most common Application Structure Software Architecture patterns: Layered, Microkernel, CQRS, Event Sourcing, and CQRS combined with Event Sourcing. Each pattern will be described using diagrams, advantages, and disadvantages.

by Michael Hanna

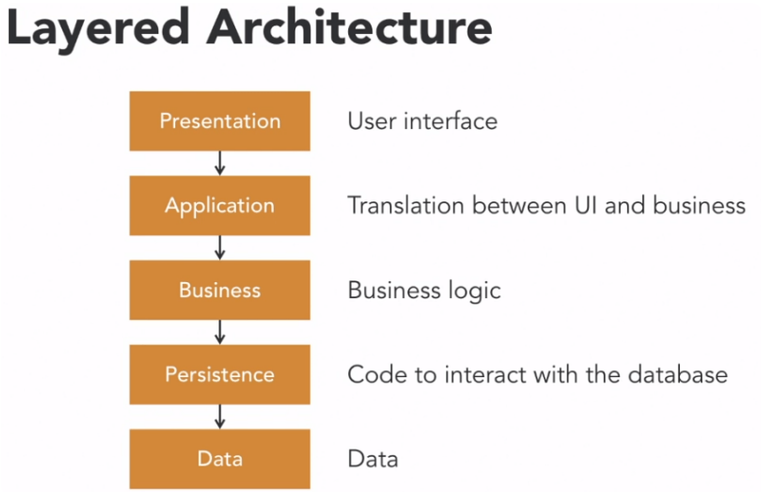
What are the Application Structure Software Architecture Patterns?

Application Structure Software Architecture Patterns are general, reusable solutions to commonly occurring problems in software architecture. They provide a structure for solving problems and help to ensure that software systems are designed to satisfy the desired properties such as maintainability, scalability, portability, and others. Common examples include Layered, Microkernel, CQRS, Event Sourcing, and CQRS combined with Event Sourcing.

Layered pattern

> The Layered Pattern, also known as tiered architecture, is characterized by the separation of concerns into distinct layers, each with a well-defined role and set of responsibilities. Each layer provides services to the layer above it and uses services provided by the layer below it.

Diagram



Advantages

- Clear separation of concerns, making it easier to maintain and update code.
- Enforcement of modular design and code reuse.
- Provides flexibility to add, remove or replace layers as needed.

Disadvantages

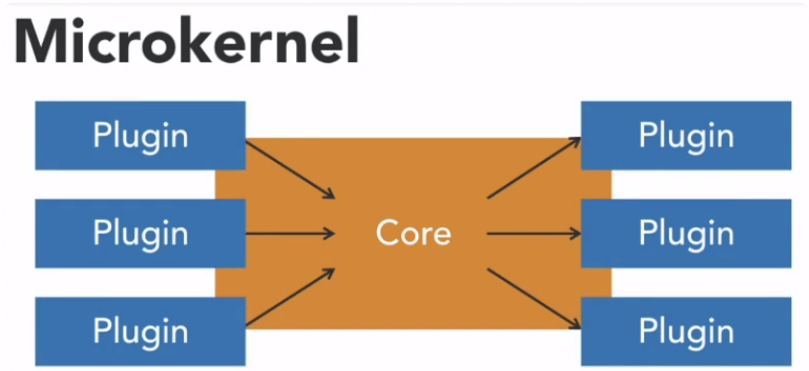
- Can lead to overhead due to the need to mediate communications between the layers.
- Can lead to reduced performance due to the need to pass information between the layers.

Microkernel pattern

> The Microkernel Pattern, also known as the **plugin** architecture, separates the application into two parts: the core system and the plugin modules. The core system is basic and consists of only the most essential components. All other components are implemented in plugin modules that are loaded dynamically into the application.

Great for applications such as: task scheduler, workflow, data processing, graphic designer, web browser.

Diagram



Advantages

- Enables customization of the system without modifying the core code (clean separation).
- Easy to extend functionality by adding or removing plugin modules (very flexible).
- Increases system performance and stability, as only the core code is loaded when the application starts.

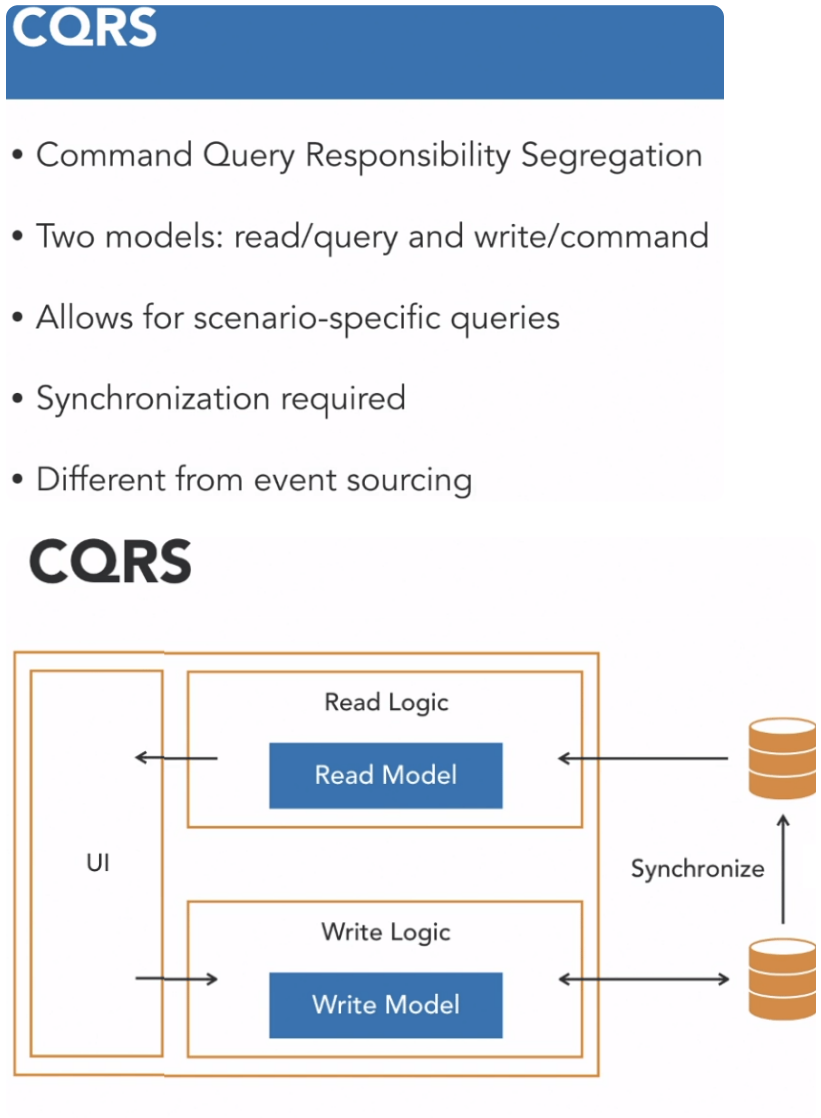
Disadvantages

- Can increase system complexity and reduce maintainability due to the number of plugins and their interactions.
- Can create coupling between plugins, making it hard to understand and debug.
- Cannot really trust the plugins.
- The core API might not fit future plug-ins.

CQRS pattern

> The Command Query Responsibility Segregation (CQRS) Pattern separates the system into two distinct parts: the Command model, responsible for commands that change the state, and the Query Model, responsible for queries that read the state. The two models can be completely independent, with different data stores, APIs, and communication models.

Diagram



Advantages

- Enables scalability by allowing different parts of the system to be scaled independently.
- Improves performance by optimizing databases and queries for reads or writes.
- Improves system agility, as changes to the Command model do not affect the Query model.
- Easier to communicate with the stockholders.

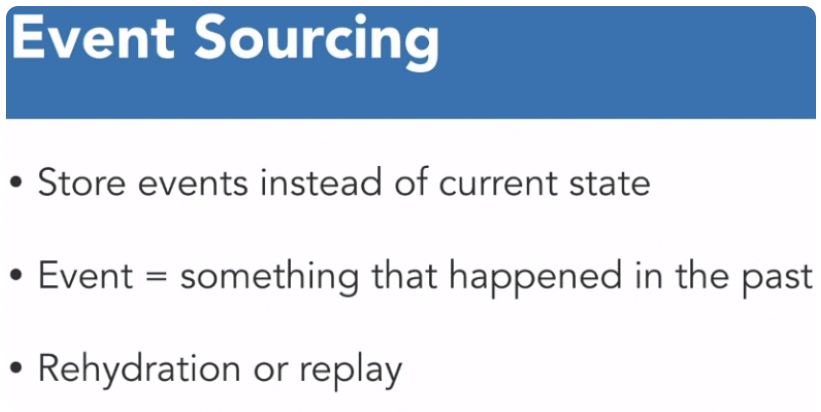
Disadvantages

- Can lead to increased complexity, as there are two different models to maintain.
- Can lead to increased effort in maintaining consistency between the two data models.
- High learning curve for developers.
- The read logic and write logic are separated, sometimes leading to asynchronous behavior.

Event Sourcing pattern

> The Event Sourcing Pattern stores changes to the system state as a sequence of events, instead of just storing the current state. Each event represents a change to the state of the system and includes metadata such as timestamps and user identity.

Diagram



Advantages

- Has traces of events.
- Enables auditability and traceability of all changes to the system state.
- Provides a complete history of the system, which can be used for debugging, analysis, and recovery.
- Improves system scalability, as the state can be distributed among different nodes.

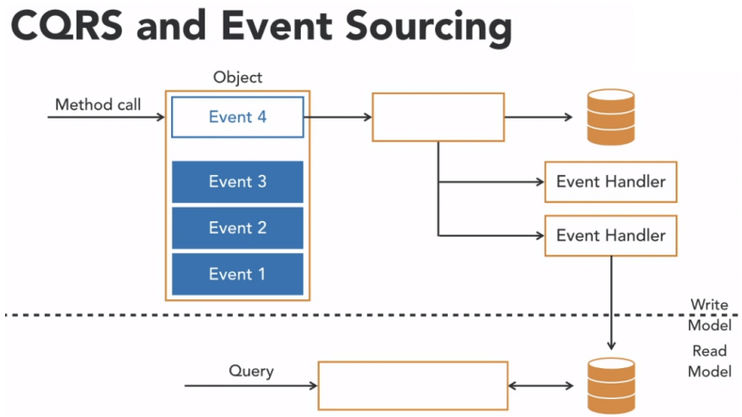
Disadvantages

- Can lead to increased storage requirements, as all changes are stored.
- Can lead to increased complexity, as there are more moving parts to the system.
- High learning curve for developers.

CQRS with Event Sourcing pattern

> CQRS with Event Sourcing Pattern combines the CQRS and Event Sourcing patterns to provide better scalability and performance. Writes are stored in the Event Store, and events are projected to the Read Model. Querying the Read Model provides fast and efficient results, while events are used to reconstruct the state of the system.

Diagram



Advantages

- Provides high scalability and performance by separating the read and write stores.
- Enables fault tolerance and disaster recovery through the event log.
- Provides flexibility and agility through the separation of the data models.

Disadvantages

- Can lead to increased complexity due to the number of moving parts and interactions between them.
- Can lead to increased storage requirements, as all changes are stored in the Event Store.

Conclusion

The Application Structure Software Architecture Patterns provide a collection of tools for software architects to use when designing software systems. Each pattern comes with different advantages and disadvantages, and it is up to the designer to choose the pattern that best fits the needs of the system. By understanding these patterns, a software architect can make informed decisions about the best way to tackle a software design problem in a particular context.