Software Architecture Patterns

**Event Driven Architecture**

* *A distributed, asynchronous architecture pattern used to produce scalable applications*. It can be used for small, simple apps as well as complex ones.
* Though it is scalable for small and big projects, its architecture pattern is complex primarily due to its asynchronous, distributed nature
* It is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events.
* It usually consists of two main “topologies”: the mediator and the broker
* **Mediator Topology**
  + Used for events that have multiple steps and require some level of orchestration to process the event according to business logic
  + Main types of components within this topology:
    1. Event Queue
       - Transports the event to the mediator
    2. Event Mediator
       - Receives initial event and orchestrates that event by sending additional asynchronous events to event channels
    3. Event Channels
       - Asynchronously pass specific processing events to the event processor
    4. Event Processors
       - Listen on the event channels, receive the event from the mediator and execute business logic to process the event
       - Should perform a single business task and not rely on other event processors

Diagram

Description automatically generated

Diagram

Description automatically generated

* **Broker Topology**
  + No central mediator; message flow is distributed across the event processor components in a chain-like fashion.
  + Useful when you have a relatively simple event flow and do not want or need central event orchestration
  + Main components within broker topology:
    1. Broker
       - Centralized of federated and contains all event channels used within flow. The channels can be queues, message topics, or both
    2. Event processor

Diagram

Description automatically generated  
Diagram

Description automatically generated

* Pros of EDA:
  + Ease of deployment due to decoupled nature of event processor components. Broker topology is even easier than mediator.
  + High performance through asynchronous nature
  + High scalability through decoupled event processors
* Cons:
  + Hard to test due to reliance on generation of events and the asynchronous nature of application.
  + Difficult to develop