CS5002NI

Software Engineering

Agendas

- Software Architecture
- Importance of Software Architecture
- Different types of Software Architecture

Design Models

- Data/Class design
- Architecture design
- Interface design
- Component-Level design

WHAT IS SOFTWARE ARCHITECTURE?


Perry and Wolf

SWA = {Elements, Form, Rationale}

what how why


Shaw and Garland

SWA [is a level of design that] involves

- description of elements from which systems are built
- interactions among those elements
- patterns that guide their composition
- constraints on these patterns

Software Architecture

- An outline that allows you to express and define a structural schema for all kinds of software system
- Reusable solution that provides predefined set of subsystems, roles and responsibilities, including the rules and roadmap for defining relationships

Software Architecture

- Addresses concerns like performance limitations, high availability, and minimizing business risk
- Countless piece of architecture that implies the same pattern

Importance of Software Architecture

- Defining basic characteristics of an Application
  - Knowing each architecture’s characteristics, strengths, and weaknesses becomes important for choosing the right one to meet your business objectives
  - For instance, some architecture patterns can be naturally used for highly scalable applications, whereas others can be used for agile applications

Importance of Software Architecture

- Maintaining Quality and Efficiency
  - High possibility that any application you build might face quality issues
  - Selecting an architecture pattern can help minimize the quality issues while simultaneously maintaining efficiency

Importance of Software Architecture

- Providing Agility
  - Natural for software applications to undergo numerous modifications and iterations during software development and even after production
  - Planning a core software architecture beforehand provides agility to the application and makes future moderations effortless

Importance of Software Architecture

- Problem Solving
  - Prior planning and knowledge of a software architecture give a clear idea of how the application and its components will function
  - With an architecture in place, the developing team can adopt the best practices to resolve complex processes and solve any errors in the future

Importance of Software Architecture

- Enhance Productivity
  - Irrespective of the skills and knowledge one has about a programming language, framework, or application, there has to be certain standardized principles
  - With an appropriate application pattern in place, the company can quickly grasp the project's status
  - Productivity rates improve the moment an architecture pattern is in place to make the project scope clear

Different Types of Software Architecture Pattern

- Layered Architecture Pattern
- Event-Driven Architecture Pattern
- Microkernel Architecture Pattern
- Microservices Architecture Pattern

Layered Architecture Pattern

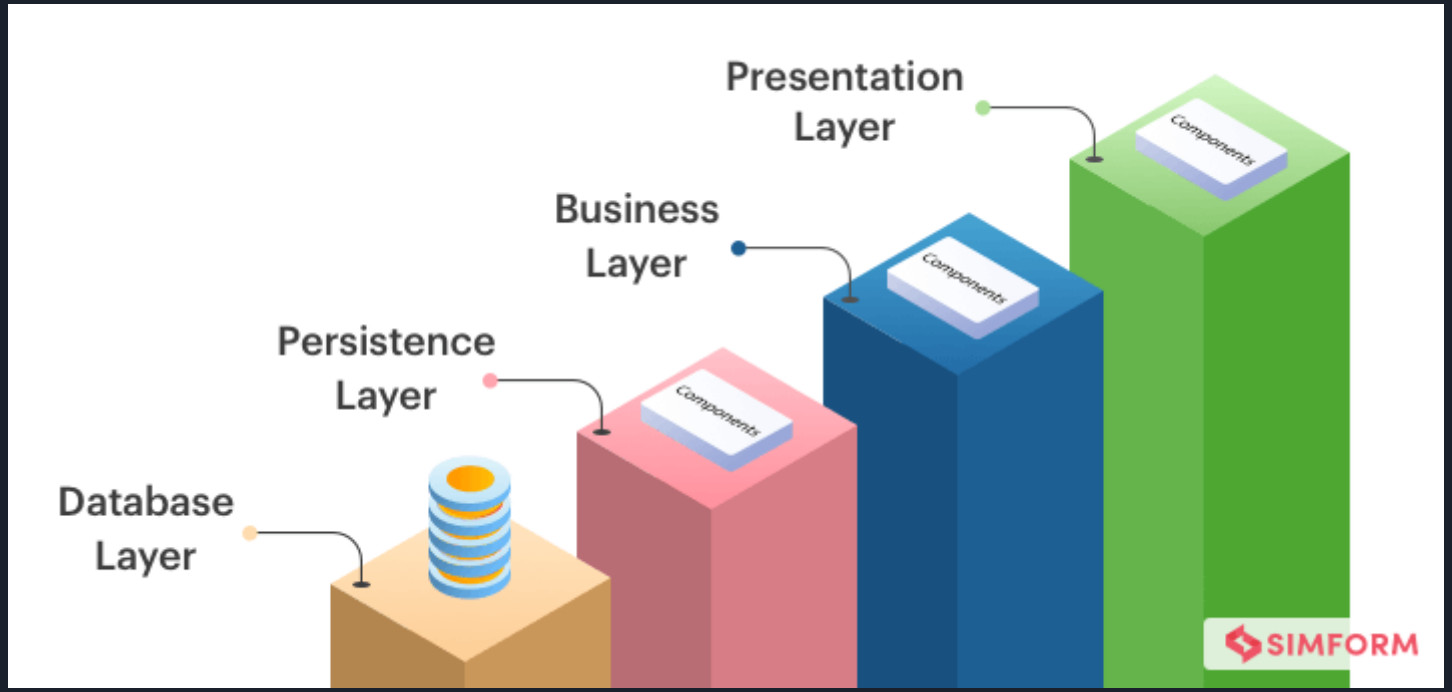
- Tiered Architecture or N-Tier Architecture
- Most common architecture - usually build around a database
- Code is arranged so the data enters the top layer and works its way down each layer until it reaches the bottom, which is usually a database

Layered Architecture Pattern

- 4 Distinct Layer
  - Presentation
  - Business
  - Persistence
  - Database

Layered Architecture Pattern

- Stands out - each layer plays a distinct role with the application and is marked as closed
- Concepts – layers of isolation – enables you to modify components within one layer without affecting the other layers


Presentation Layer
Business Layer
Persistence Layer
Database Layer
Components
Components
Components
SIMFORM

Layered Architecture Pattern - Usage

- Application that are needed to be build quickly
- Enterprise applications that require traditional IT departments and processes
- Appropriate for teams with inexperienced developers and limited knowledge of architecture patterns
- Application that require strict standards of maintainability and testability

Layered Architecture Pattern - Downside

- Performance wise, it can be inefficient sometimes.
- Skipping previous layers to create tight coupling can lead to a logical mess full of complex interdependencies
- Basic modification can require a complete redeployment of the application

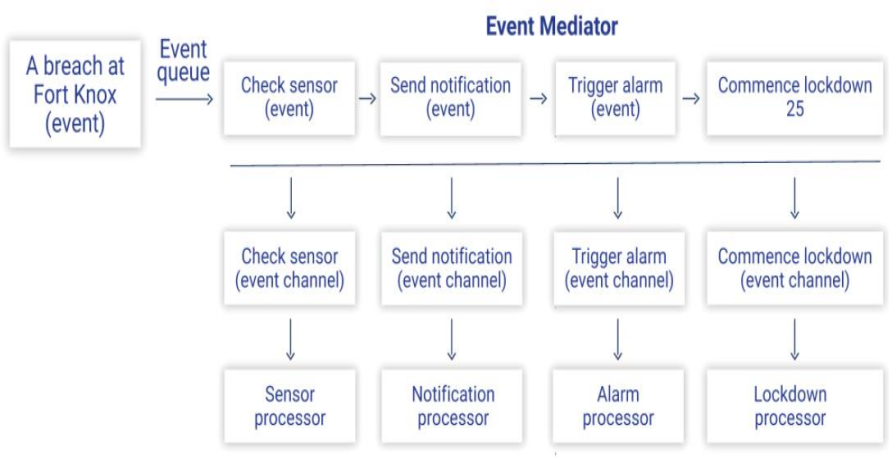
Event Driven Architecture Pattern

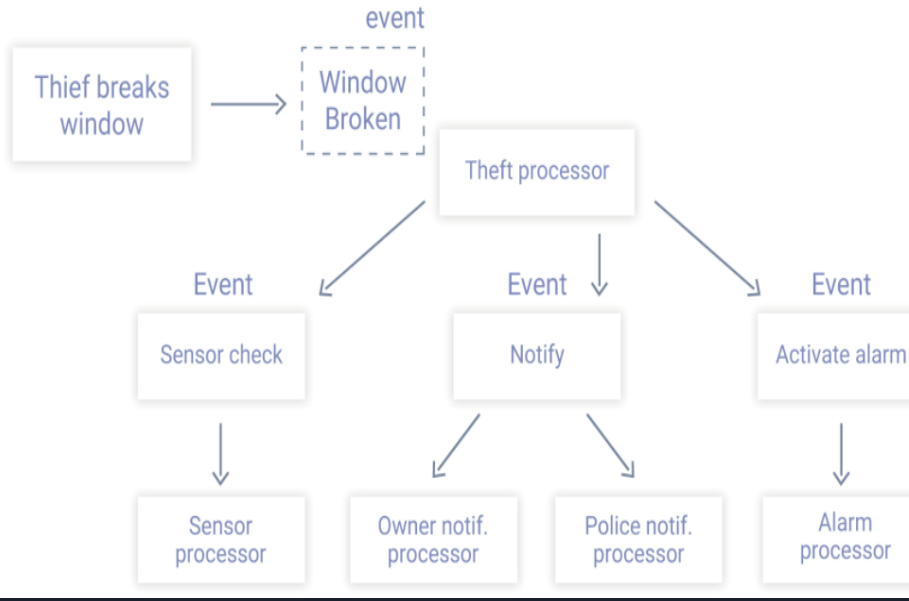
- Building a central unit that accepts all data and then delegates it to the separate modules that handle the particular type

- Made up of decoupled, single-purpose event processing components that asynchronously receive and process events

Event Driven Architecture Pattern

- Consist of two topologies
  - **Mediator** - when multiple steps are needed to be orchestrated within an event bus through a central mediator
  - **Broker** - used to chain events together without using a central mediator


Event Mediator
A breach at Fort Knox (event)
Event queue
Check sensor (event)
Send notification (event)
Trigger alarm (event)
Commence lockdown 25
Check sensor (event channel)
Send notification (event channel)
Trigger alarm (event channel)
Commence lockdown (event channel)
Sensor processor
Notification processor
Alarm processor
Lockdown processor


event
Thief breaks window
Window Broken
Theft processor
Event
Event
Event
Sensor check
Notify
Activate alarm
Sensor processor
Owner notif. processor
Police notif. processor
Alarm processor

Event Driven Architecture Pattern - Usage

- Application where individual data blocks interact with only a few modules
- For real time systems and data tracking.

Event Driven Architecture Pattern - Downside

- Testing individual modules can only be done if they are independent, otherwise, they need to be tested in a fully functional system

- When several modules are handling the same event, error handling becomes challenging to structure.


kubernetes
Containers Orchestration
docker
Containerization
DATADOG
Logging and Audit
MICROSERVICES
Pipeline Automation
GitLab
Asynchronous Messaging
kafka
Performance Monitoring
Prometheus

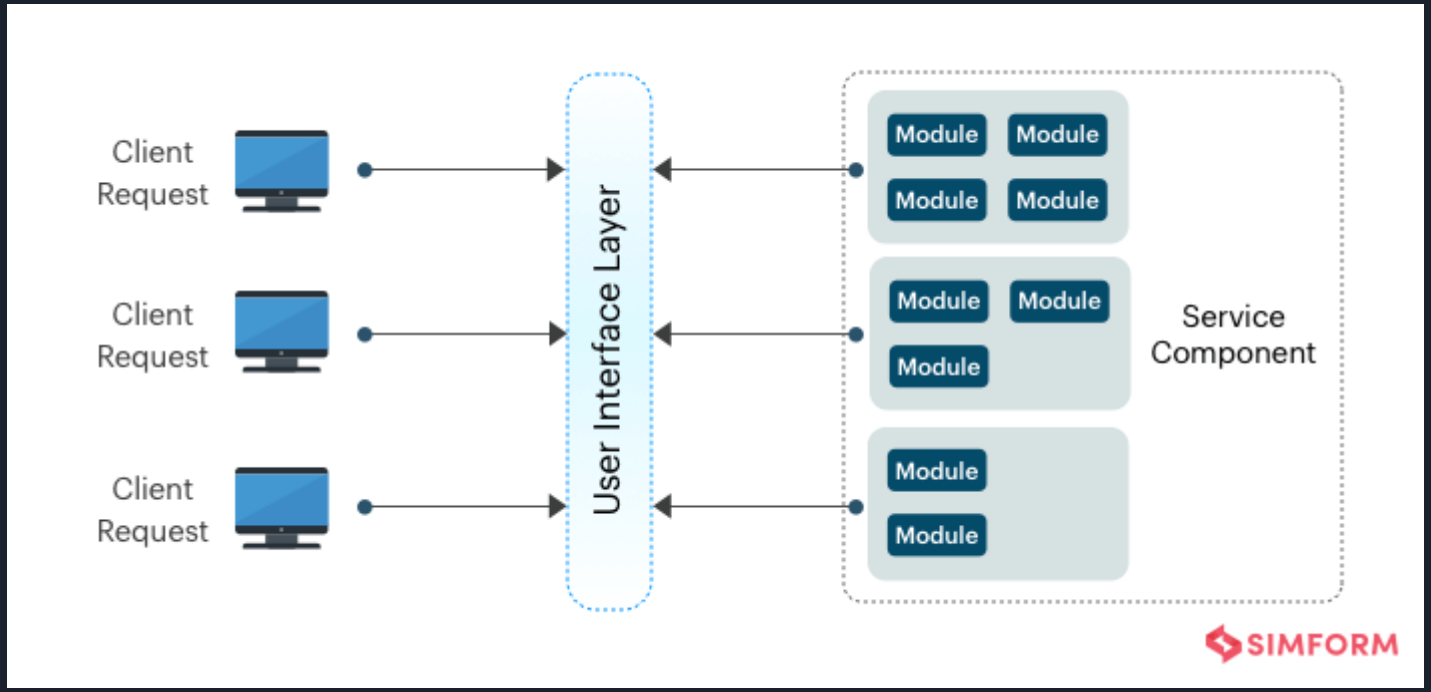
Microservices Architecture Pattern

- Instead of building one big program, the goal is to create a number of different tiny programs and then create a new little program every time someone wants to add a new feature
- The components are deployed as separate units through an effective, streamlined delivery pipeline
- The pattern’s benefits are enhanced scalability and a high degree of decoupling within the application

Microservices Architecture Pattern

- Owing to its decoupled and independent characteristics, the components are accessed through a remote access protocol

- Same components can be separately developed, deployed, and tested without interdependency on any other service component.


Client Request
Client Request
Client Request
User Interface Layer
Module
Module
Module
Module
Module
Module
Module
Module
Module
Service Component
SIMFORM

Microservices Architecture Pattern - Usage

- For applications where scalability matters.
- Business and web applications that require rapid development and remote has teams globally.

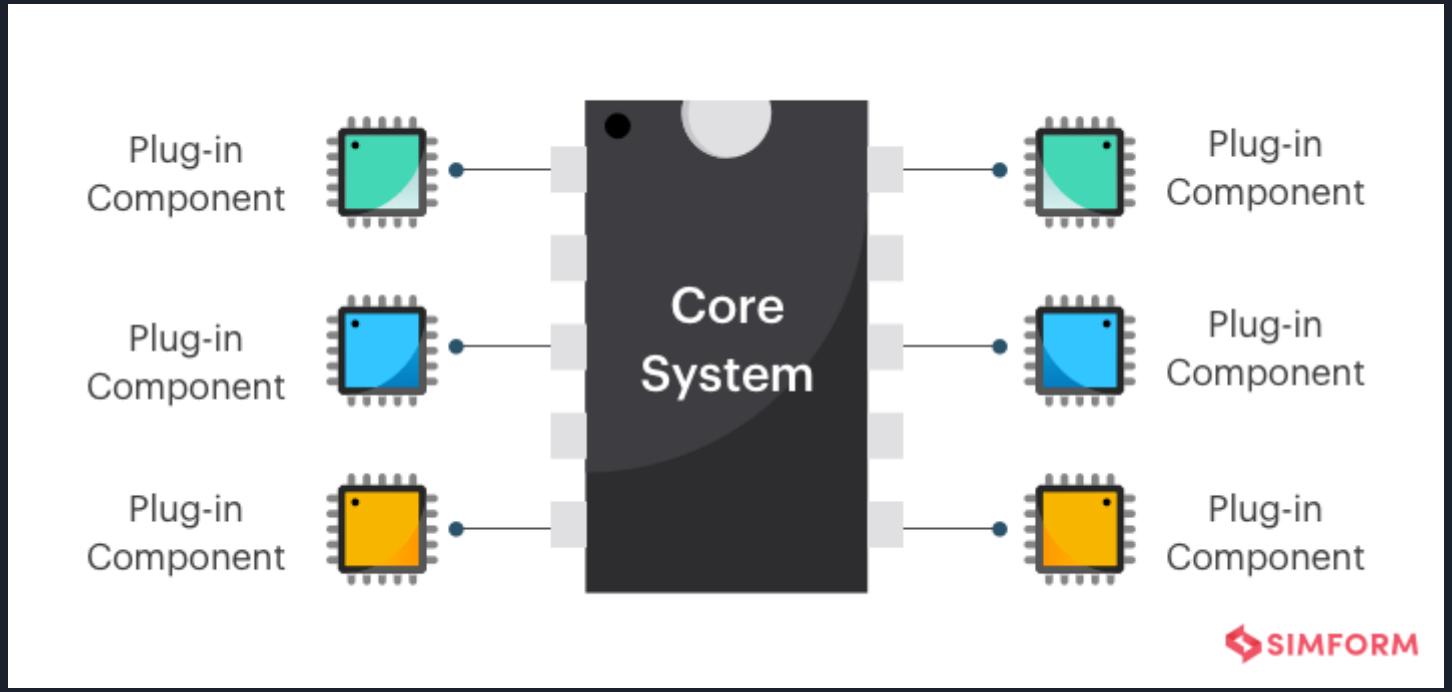
Microservices Architecture Pattern - Downfall

- Designing the right level of granularity for a service component is always a challenge.
- All applications do not include tasks that can be split into independent units.
- Performance can be affected when tasks are spread across different microservices.

Microkernel Architecture Pattern

- Basically a two component architecture pattern
  - a core system
  - several plugin modules

- The core system works on minimal functionality to keep the system operational.
- The plug-in modules are independent components with specialized processing.


Plug-in Component
Plug-in Component
Plug-in Component
Core System
Plug-in Component
Plug-in Component
Plug-in Component
SIMFORM

Microkernel Architecture Pattern - Usage

- Applications that suits evolutionary design and incremental development.
- Applications that have a fixed set of core routines and dynamic set of rules that needs frequent updates.

Microkernel Architecture Pattern - Downfalls

- The plugin must have good handshaking code so that the microkernel is aware of the plugin installation and is ready to work.
- Changing a microkernel is almost impossible if there are multiple plugins depending on it.
- It is difficult to choose the right granularity for a kernel function in advance and more complex at a later stage.

THANK YOU!