Software Architecture

Eduard-Gabriel Poesina

/thoughtworks

Definition of Software Architecture

Software Architecture

Definition

In “Who needs an Architect?”, Martin Fowler adamantly refuses to provide a definition for the software architect.

“Architecture is about the important stuff.. Whatever that is” - Ralph Johnson

Software Architecture

Definition

Historically, the role of the software architect was only in relation to purely technical aspects of architecture (modularity, components and patterns).

As the aspects of software development became more complex, so did the necessity of domains that software architects needed to cover.

Software Architecture

Definition

The common definitions of software architecture.

1. **Blueprint of the system**

2. **Roadmap of the system**

3. **It's the most important part**

Blueprint of the system

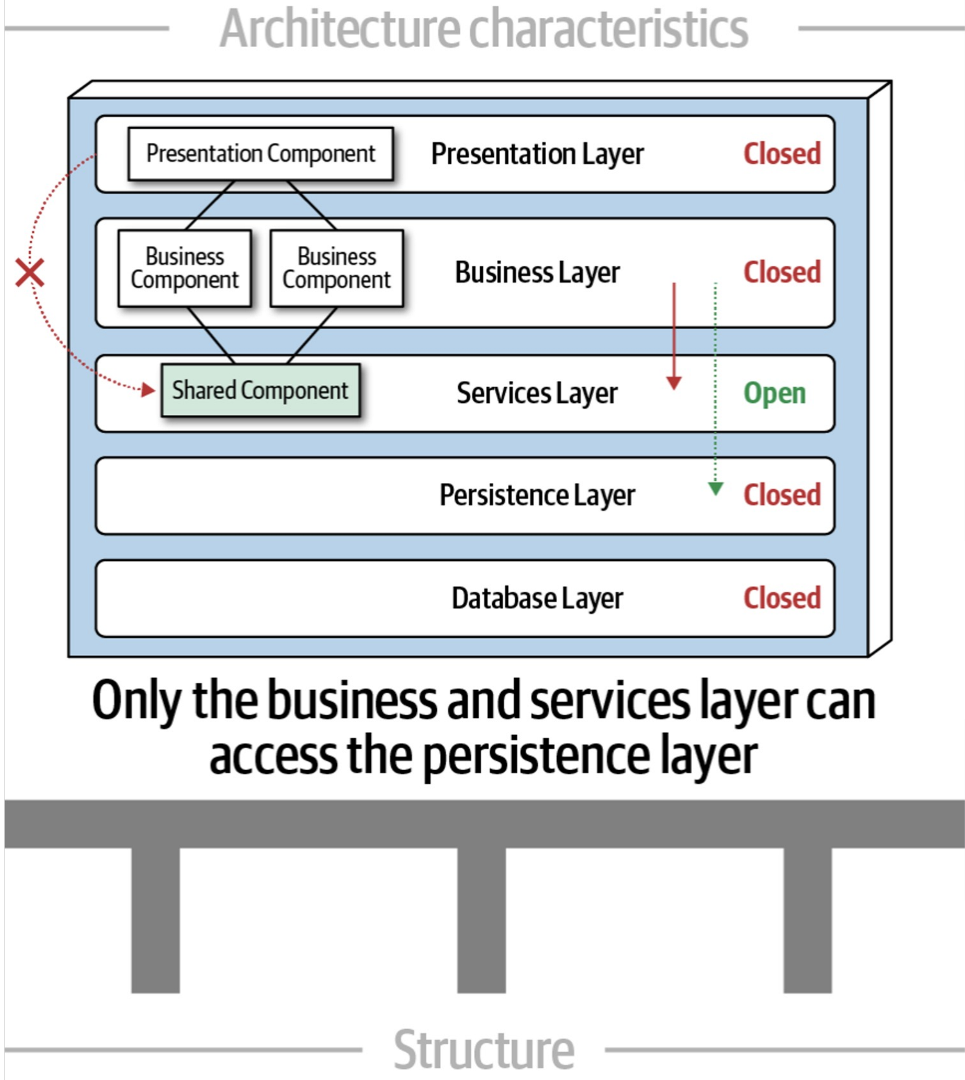
Explains the structure of the system. However lacks information about characteristics, decisions and design principles chosen in the system.

Characteristics are performance measures of the system.

- Availability
- Reliability
- Fault Tolerance
- Security
- Deployability
- Performance

Blueprint of the system

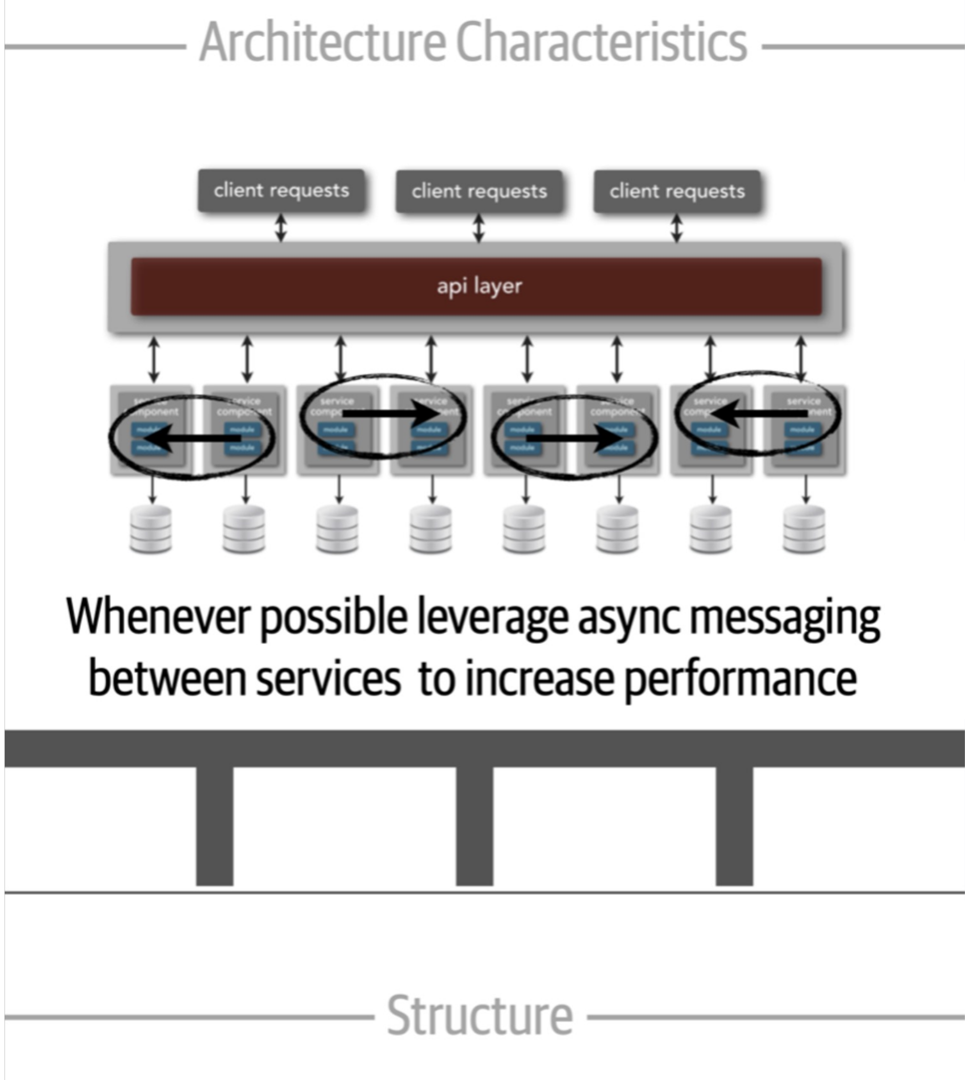
Architecture decisions are the second factor of a software architecture and reflects the rules of how a system should be constructed.



Blueprint of the system

Design principles are a different to architecture decisions. Design decisions are a guideline rather than a hard-rule.

An architecture decision can not cover every condition and option which might appear but a design principle may be used a sa guideline.


Architecture Characteristics
client requests
client requests
client requests
api layer
Whenever possible leverage async messaging between services to increase performance
Structure

Expectations

The definition of the software architect is as difficult as the definition for software architecture.

It is a lot easier to just list the expectations for it.

- Make architecture decisions
- Continuously analyze the architecture
- Keep current with the latest trends
- Ensure compliance with decisions
- Diverse exposure and experience along with domain knowledge
- Interpersonal skills and ability to understand and navigate politics

With great power....

The responsibility of the software architect has grown larger and larger the last decade as it encompassed more and more responsibilities.

1. Engineering Practices

2. DevOps

3. Process

4. Data

Laws of Software Architecture

thoughtworks

Software Architecture

Laws of software architecture

First law of software architecture

Everything in software architecture is a trade-off.

Corollary 1

If an architect thinks they have discovered something that isn't a trade-off, more likely they just haven't identified the trade-off yet.

Software Architecture

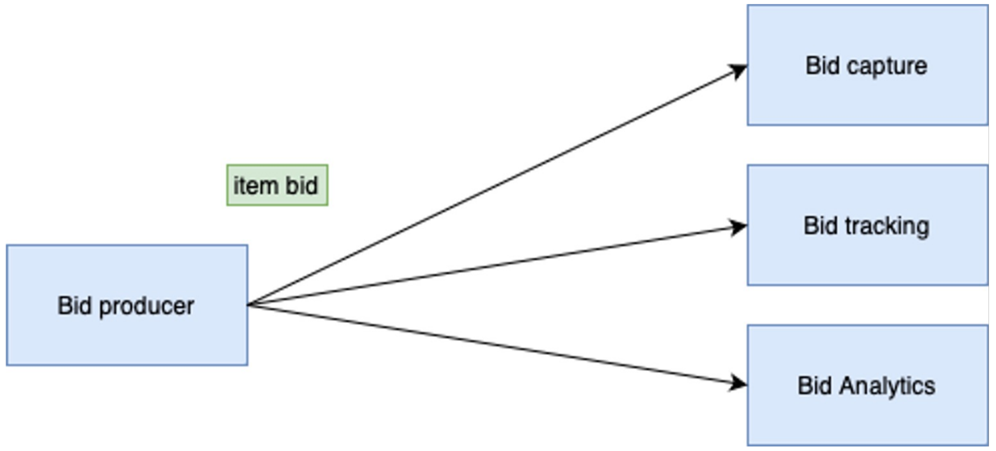
Laws of software architecture

Second law of software architecture

Why is more important than how

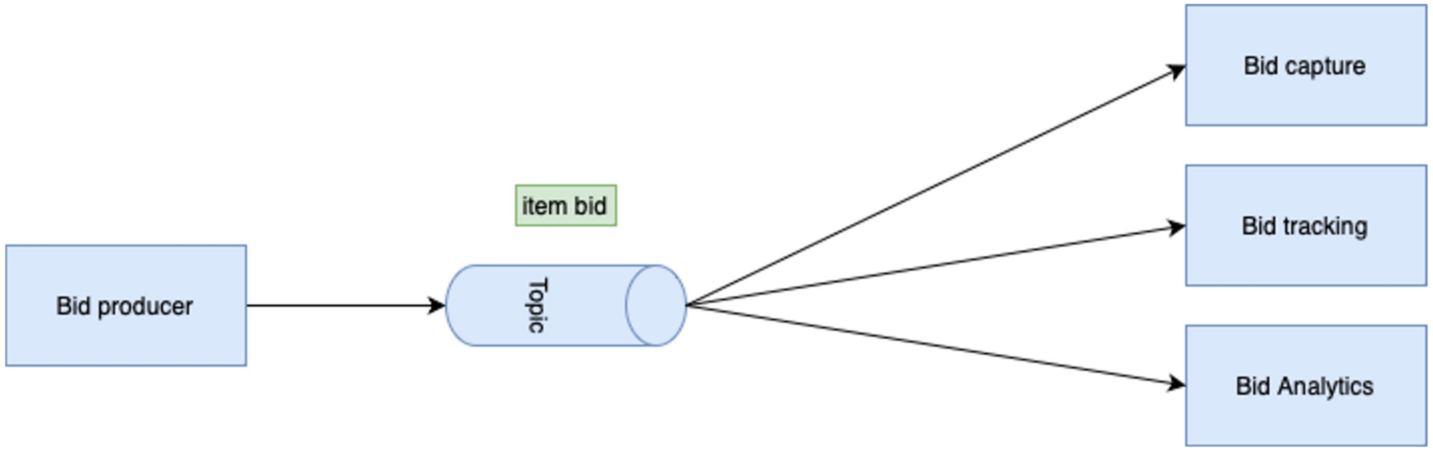
Software Architecture

Tradeoffs



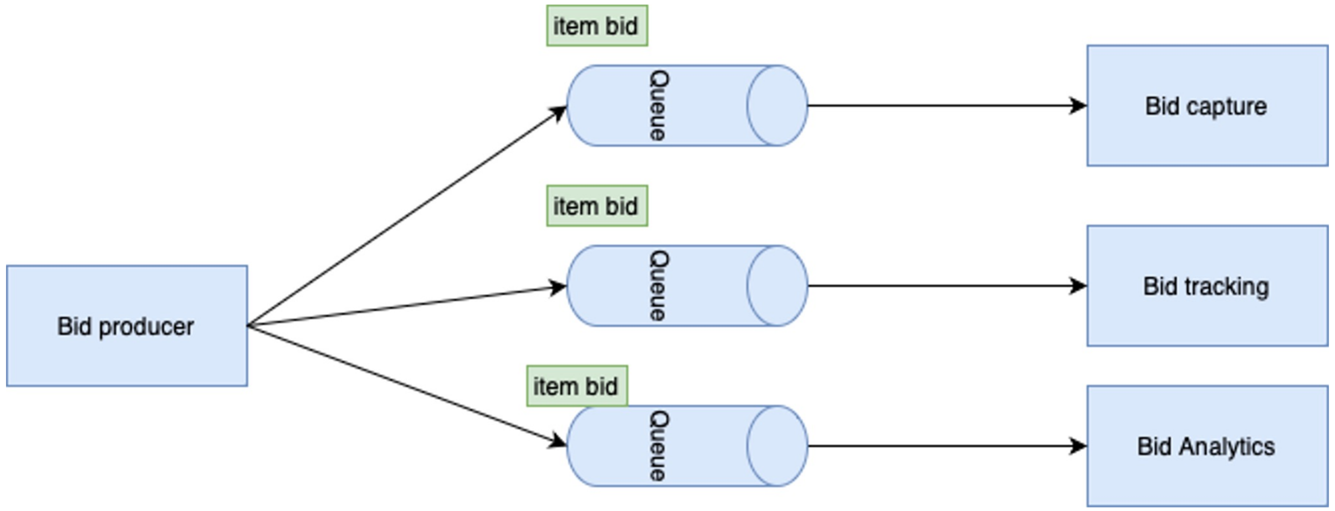
Software Architecture

Tradeoffs



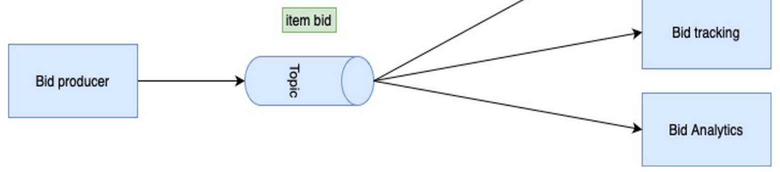
Software Architecture

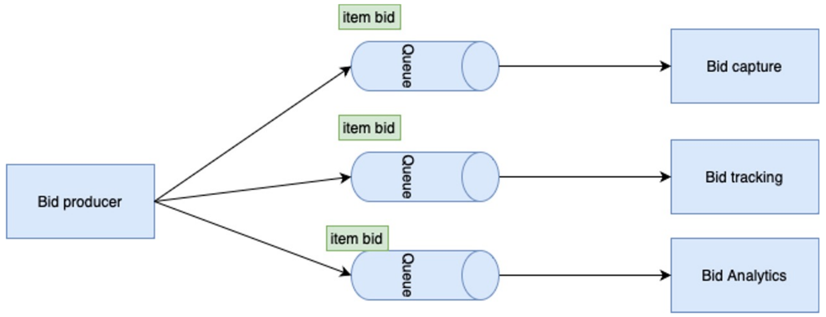
Tradeoffs



Software Architecture

Tradeoffs





Software Architecture Styles

/thoughtworks

Software Architecture Styles

Fundamental Patterns

There are several fundamental patterns that appear throughout the history of software architecture because they provide a useful perspective on organizing code, deployments or other aspects of architectures.

Big ball of Mud. The anti-pattern.	Unitary Architecture	Client/Server
Haphazardly structured, sprawling, sloppy, duct-tape and-baling-wire, spaghetty-code jungle.	Everything is build around a single system. After some time evolved to Client/Server architecture.	Separates the technical functionality between frontend and backend. It has many flavous: desktop + database / browser + webserver / three tier

Software Architecture Styles

Fundamental Patterns

Architecture styles can be classified into two main types: monolithic (single deployment unit of all code) and distributed (multiple deployment units connected through remote access protocols).

Monolithic:

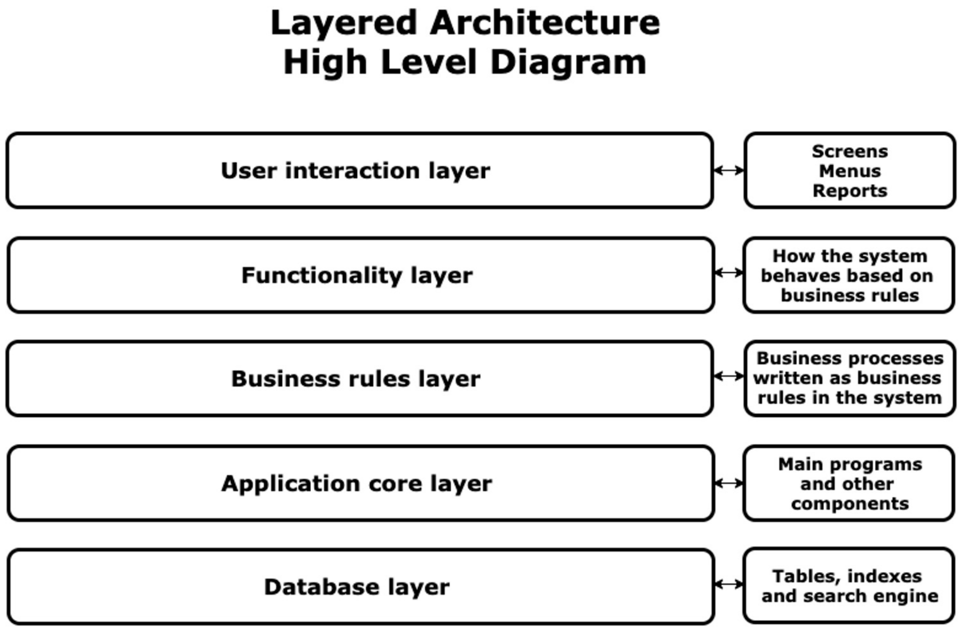
1. **Layered Architecture**
2. **Pipeline Architecture**
3. **Microkernel architecture**

Distributed:

1. **Service-based architecture**
2. **Event-driven architecture**
3. **Space-based architecture**
4. **Microservices architecture**

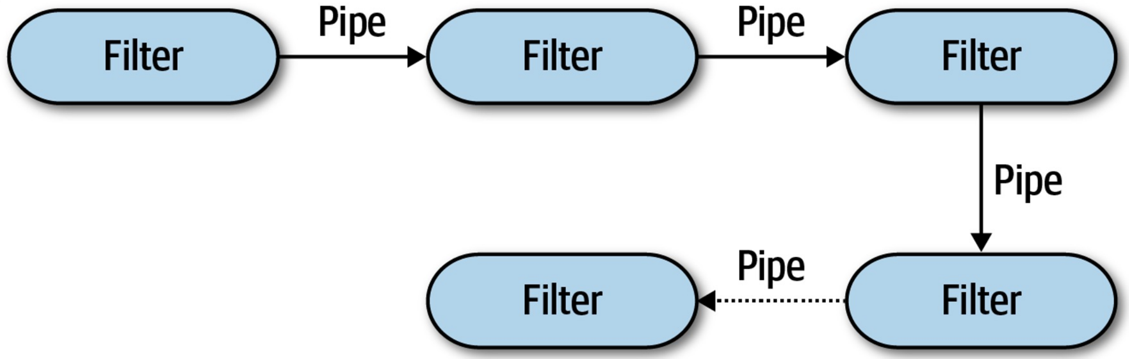
Software Architecture

Architectural Styles - Layered Architecture


Layered Architecture
High Level Diagram
User interaction layer
Screens
Menus
Reports
Functionality layer
How the system behaves based on business rules
Business rules layer
Business processes written as business rules in the system
Application core layer
Main programs and other components
Database layer
Tables, indexes and search engine

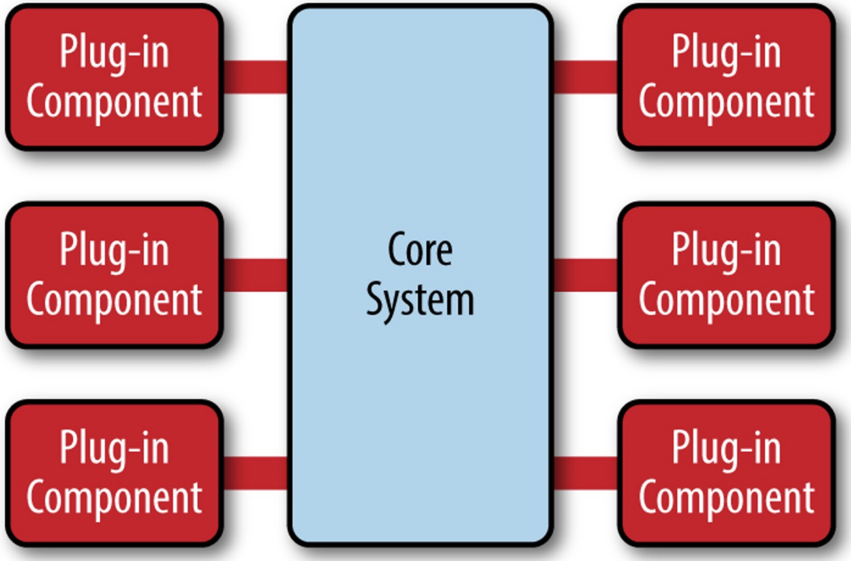
Software Architecture

Architectural Styles - Pipeline architecture



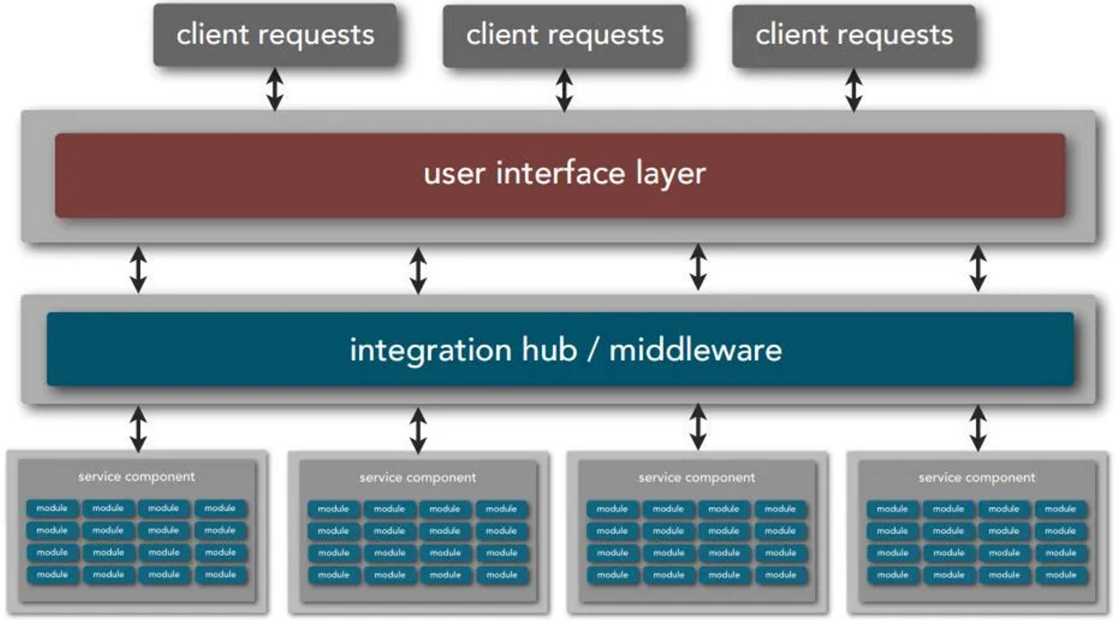
Software Architecture

Architectural Styles - Microkernel architecture



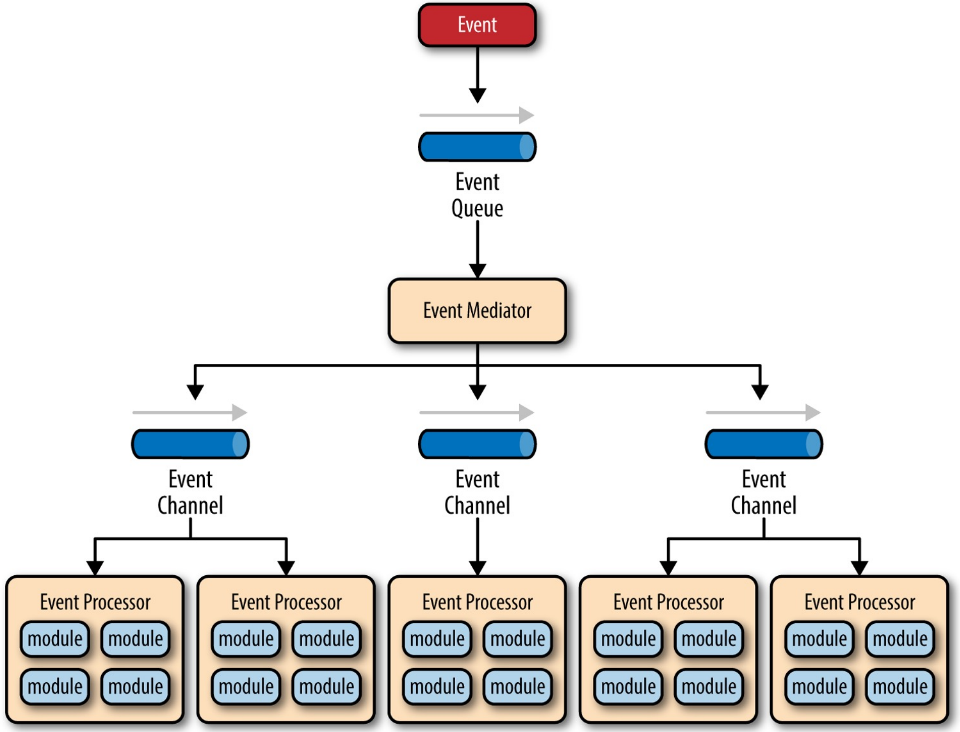
Software Architecture

Architectural Styles - Service-based architecture



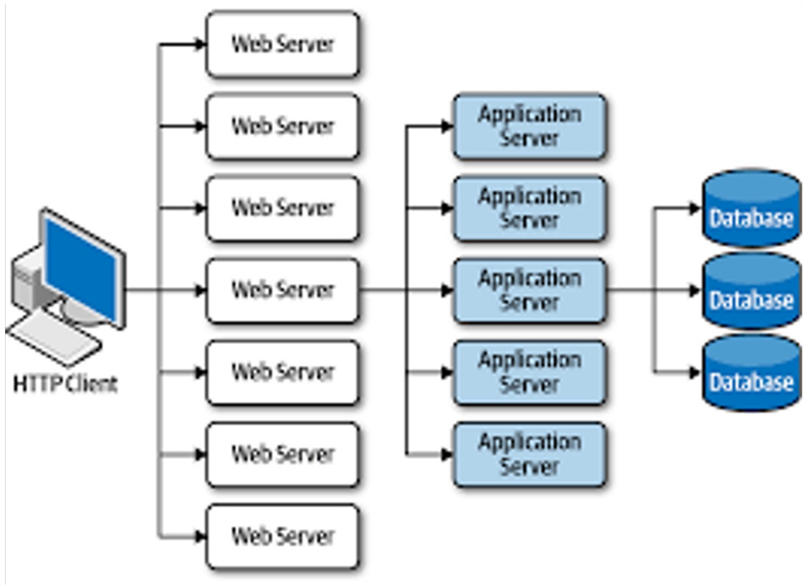
Software Architecture

Architectural Styles - Event-driven architecture



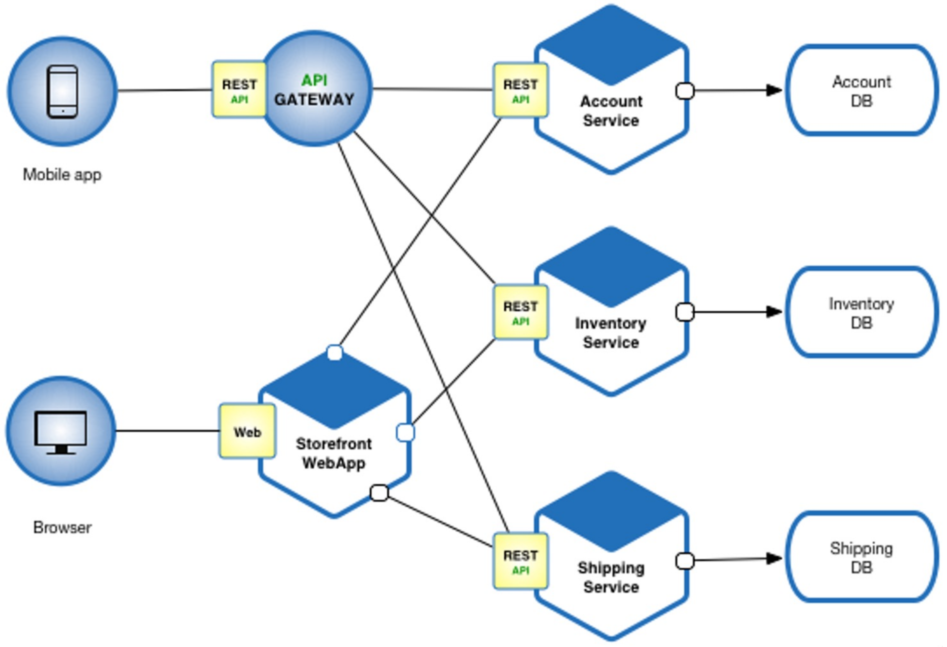
Software Architecture

Architectural Styles - Space-based architecture



Software Architecture

Architectural Styles - Microservice architecture



Software Architecture Patterns

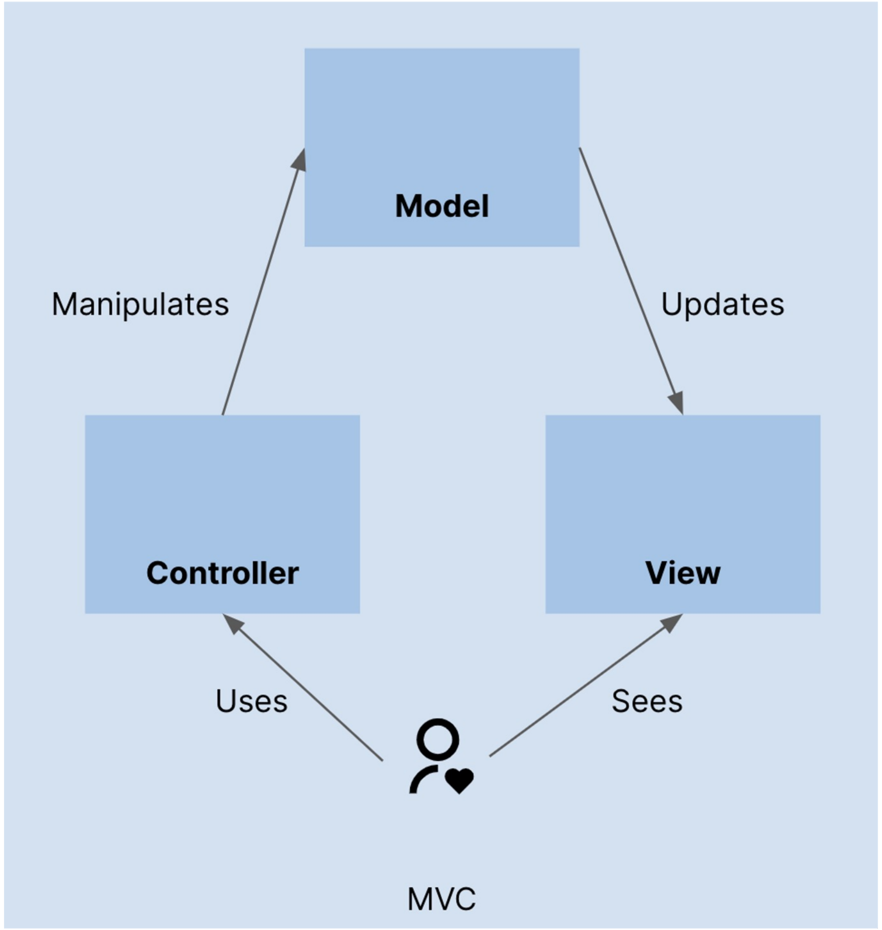
/thoughtworks

Software Architecture

Architectural Patterns - MVC Pattern

Model View Controller

- One of the earliest patterns;
- Composed of three parts
  - Model – describes the buisness logic and the used models;
  - *View* – the UI components;
  - *Controller* – processer inputs and makes the connection between the View and Model;
- Advantages:
  - Decouples development and ensures easiness of parallel development, testing and maintenance
  - Multiple views can be connect to same controllers and models;

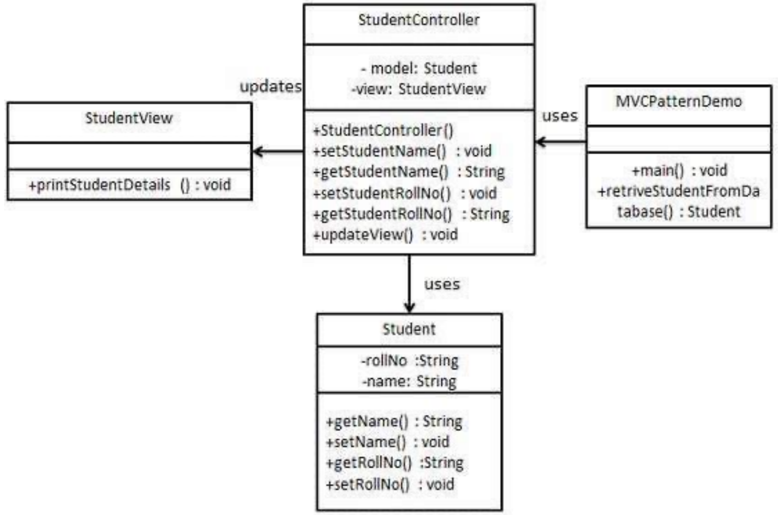


Software Architecture

Architectural Patterns - MVC Pattern

Model View Controller

- One of the earliest patterns;
- Composed of three parts
    - Model – describes the buisness logic and the used models;
    - *View* – the UI components;
    - *Controller* – processer inputs and makes the connection between the View and Model;
- Advantages:
    - Decouples development and ensures easiness of parallel development, testing and maintenance
    - Multiple views can be connect to same controllers and models;

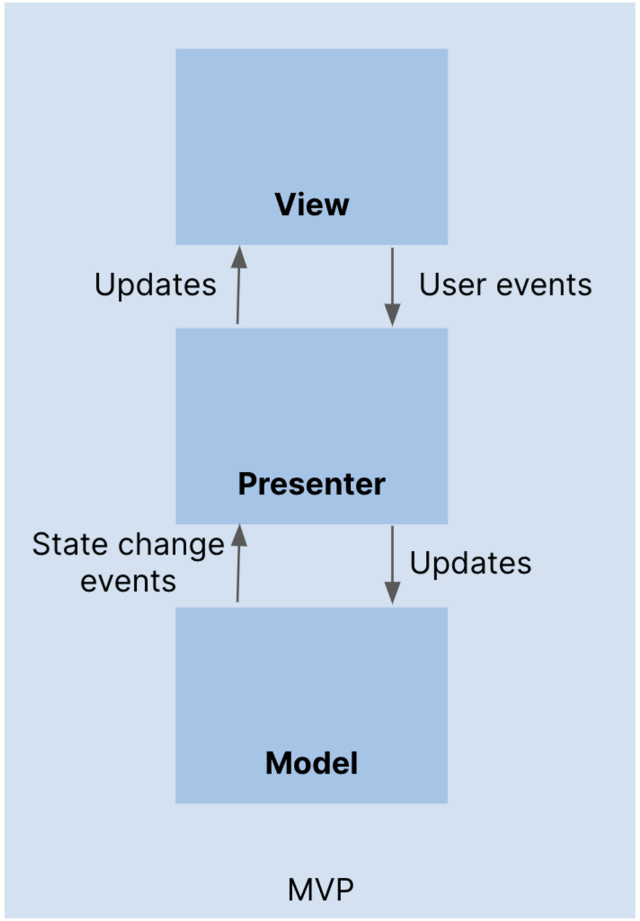


Software Architecture

Architectural Patterns - MVP Pattern

Model View Presenter

- Composed of three parts
    - Model – describes the buisness logic and the used models;
    - *View* – the UI components;
    - *Presenter* – processes inputs and makes the coordination between model and view possible;
- Similar to MVC but it is view-centric;
- Decouples the view even more;

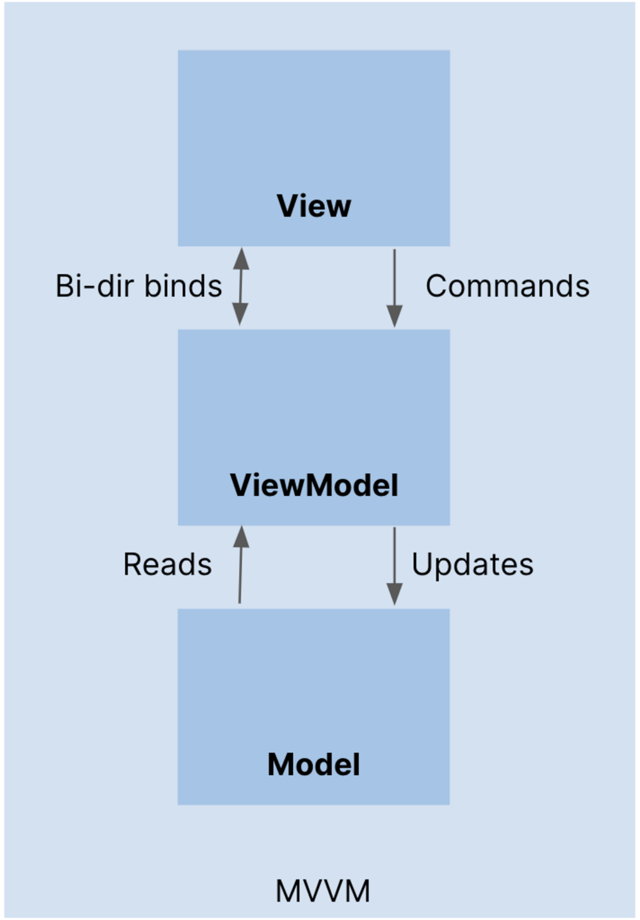


Software Architecture

Architectural Patterns - MVVM Pattern

Model View Presenter

- Composed of three parts
  - Model – describes the buisness logic and the used models;
  - *View* – the UI components;
  - *ViewModel – abstraction of View;*
- It has a bidirectional binding between the View and ViewModel;
- Decouples responsabilities



Thank you!

Eduard-Gabriel Poesina
Senior Data Scientist
eduard.poesina@thoughtworks.com

/thoughtworks