1 Software Architecture

Software Architecture is concerned with how the software is designed and implemented. It is often a combination of design and implementation and is often defined in terms of dimensions.

Software Architecture is a fast evolving and ever changing field. Some practices of the past are nowadays discouraged, while some others, which were once discouraged, are now encouraged.

1.1 Architects and Developers Interoperability

Architects and Developers work together closely. They give each other feedback and adapt to each others works. This is done iteratively, which promotes the evolution of the software.

Architects

- Extract the architecture characteristcs from requirements analysis
- Choose set of styles for software system
- Create the component structure

Developers

- Design class structure for each component
- Design user interface
- Write and test source code

1.2 Dimensions

Architectural Characteristics

Concerned with how different characteristcs the software should fulfill.

Operational	Structural	Cross-Cutting
Availability	Extensibility	Accessibility
Scalability	Maintainability	Privacy
Performance	Leveragibility	Security
...	...	...

Architectural Style (Software System Structure)

Concerned with different layers of a software system are connected.

Presentation Layer
Business Layer
Persistence Layer
Database Layer

Decisions Concerning Architecture

Clarifies some questions before the architecture is implemented. For example:
- Which web framework?
- What are each layers responsibilities?
- How do layers communicate with each other?
- Which data formats should be used?
- ...

Design Principles

Concerned with how the architecture should be implemented on a technical level. For example:
- NoSQL Databases are preferred
- Immutable data structures are preferred
- Asynchronous messaging between services when possible
- Avoid usage of caching in clients
- ...

1.3 Architecture Characteristics

- Specifies non-domain design consideration: How to implement a given requirement
- Influences the structural design aspects: Requirements of specific architectural elements
- Critical that application performs as intended: Meets functional and non-functional requirements

Operational Characteristics

- **Availability:** When the system must be operational (specific times, continous, etc.)
- **Performance:** Response time, peak analysis, stress test, etc.
- **Scalability:** Functions with increased numbers of requests, users, etc.

Structural Characteristics

- Extensibility: Ability to add new features
- Maintainability: Ability to modify existing features
- Leveragibility: Ability to reuse existing features
- Localization: Support for different languages, currencies, units, etc.
- Configuration: Ability for user to configure the system to their needs via configuration interface

Cross-Cutting Characteristics

- Accessibility: Usability for a lot of people, especially people with disabilities
- Privacy: User data inaccessible to unauthorized parties
- Security: Encryption of database, network traffic, authentication, authorization, resilience to attacks, etc.

1.4 Architectural Styles

- Help specify fundamental structure of a software system
- Impacts appearance of concrete software architectures
- Defines global properties:
  - Interoperability of components

  - Boundaries of subsystems
  - etc.

A software system can have multiple architectural styles.

An architectural style does almost always bring trade-offs. Being aware of them is important to choose the right one for your needs.

1.5 Monolithic Architectural Styles

1.5.1 Layered Architectural Styles

Operational	Structural	Cross-Cutting
Availability Scalability Performance ...	Extensibility Maintainability Leveragibility ...	Accessibility Privacy Security ...

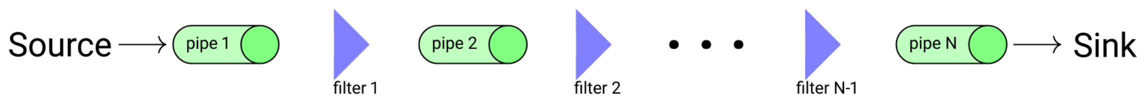
Trade-Offs

+ Simplicity	- Elasticity and scalability
+ Cost	- Performance (No parallelization)
+ Reliability	- Availability (Long startup time)

Layered Architectures in general:

- Technologically partitioned, not domain partitioned
- Works well for small to medium sized systems
- Serves as a starting point for larger systems, can be changed later
- Problem: Often created unconciously as it reflects the organisational structure of the company

1.5.2 Pipes and Filters

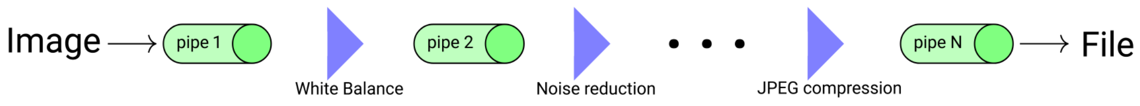


Pipes and Filters

- Pipes:
  - Unidirectional, point-to-point channels from data source to target
  - Allow any data format, although smaller data formats are preferred for better performance
- Filters:
  - Self contained and independent from other filters
  - Stateless, does not depend on past data and realizes exactly one task

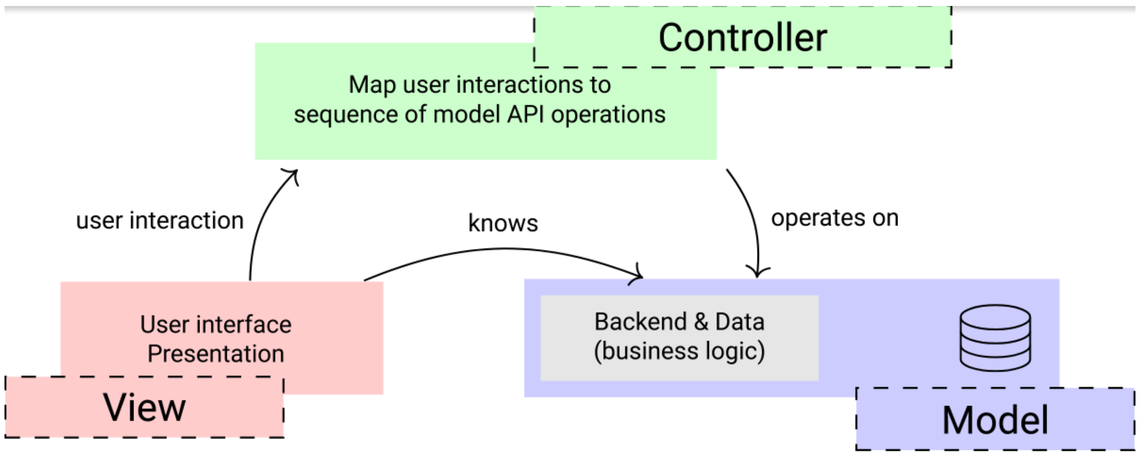
Different Types of Filters

- Producer:
  - Producer: Source of data
  - Transformer:
    1. Receives data from input channel
    2. Performs operations on the data
    3. Forwards result via output channel
  - Consumer: Sink of data, Display output, written file, database, etc.
  - Tester:
    1. Receives data from input channel
    2. Tests whether data satisfies certain conditions
    3. Redirects data accordingly to different output channels



Example of Pipes and Filters: Image Processing

1.5.3 Model-View-Controller (MVC)


Controller
Map user interactions to sequence of model API operations
user interaction
knows
operates on
User interface Presentation
View
Backend & Data (business logic)
Model

- Seperates system into three parts:
- Model:
  - Business logic and data storage
  - Independent of input behaviour and output representation
- View:
  - Presentation of the model data to the user
    - ∗ Data obtained from model
    - ∗ Often more than one view
- Controller:
  - Translates user interactions to operations on the model
  - Each view has its own controller
  - All interactions with the model are done via the controller
- Controller and View are directly coupled with the model
- The model is independent of the controller and view

Change Propagation Mechanism

- Ensures consistency between the UI and the model
- Views register themselves at a model (Controllers too if behaviour depends on model state)
- Model notifies registered objects of changes

Trade-Offs

- Updates all registered objects, even if they are not affected
- Increase in complexity due to seperate view and controller components without gaining much flexibility
- High dependency between view and controller

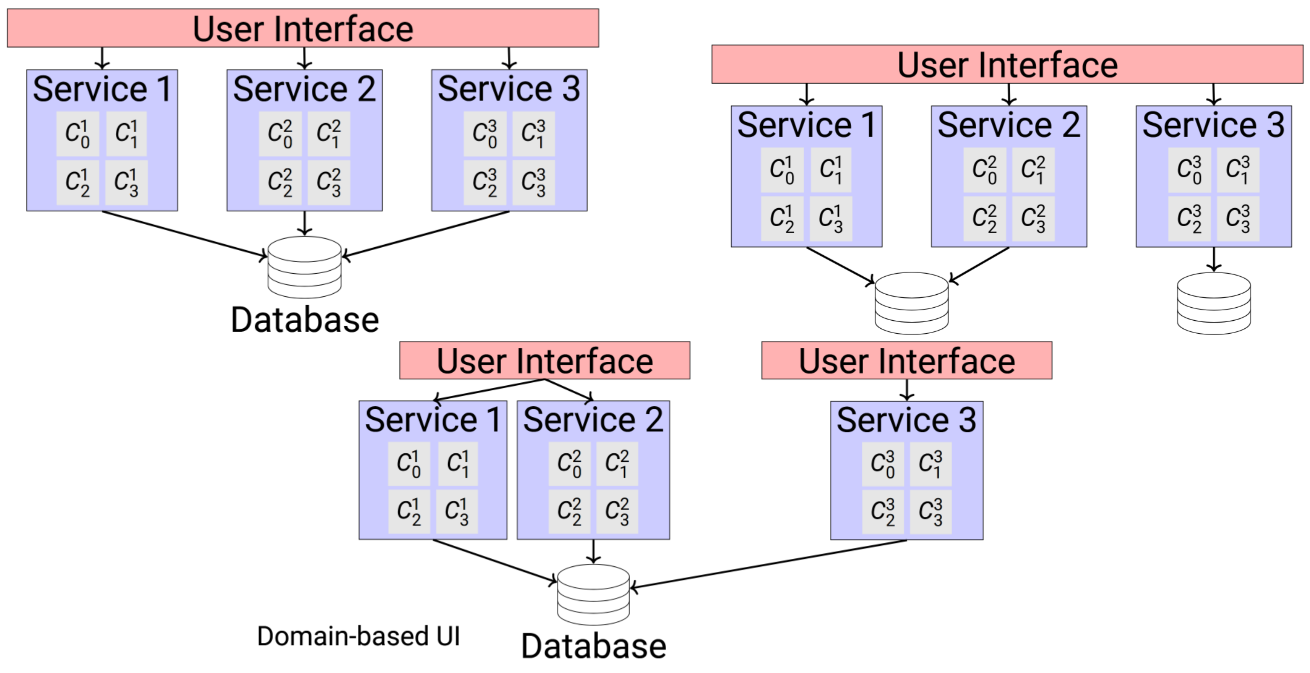
MVC should be used when:

- Application is interactive and:
- Number and kind of views are not fixed or unknown
- Display and application behaviour must reflect changed immediately
- Changing and porting the ui should not affect the applications core

1.6 Distributed Architectural Styles

1.6.1 Service-Based

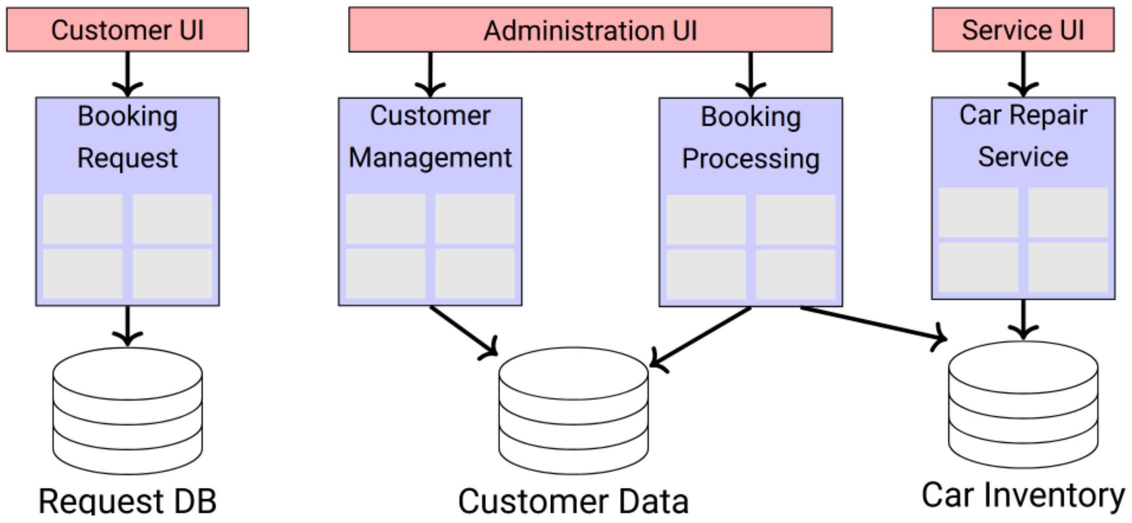
Service-Based Architecture is a style of software architecture where the application is decomposed into service components that are independent of each other and are able to be provided separately. They can take different forms:



In these examples access permissions must be defined. For example: One service might have read-only access to a database while another has write access.

More concrete this can look like this:

Choosing the right architecture style is tricky as it depends on many factors, such as the domain, the characteristcs, data architecture, organizational factors and the devolpment factors as a whole.



Therefore communication and documentation is very important.