**1. Introduction to Software Architecture**

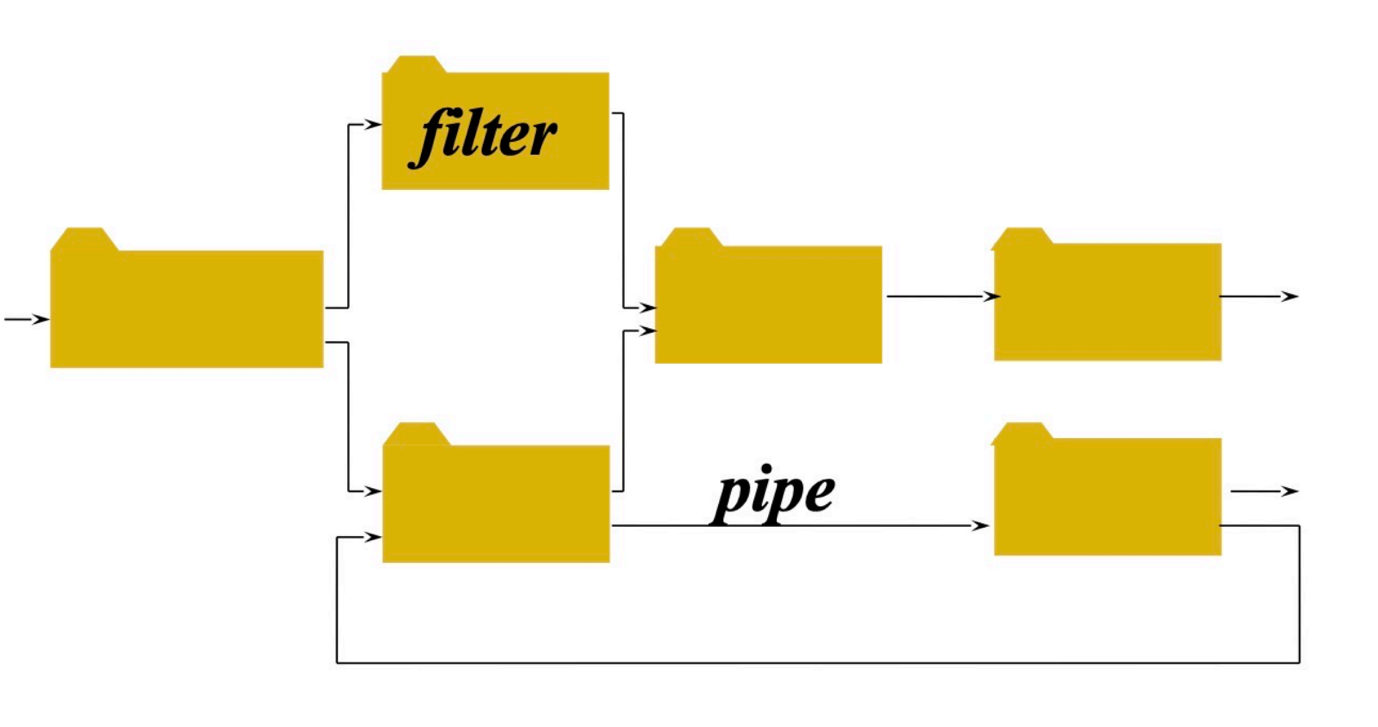
**Goal**:

* Understand what software architecture is and explore commonly used architectures.
* Learn to use design rationale to document the system.

**2. Common Frameworks in Software Architecture**

**Components**:

* **Components**: Individual parts of the system that perform specific functions.
* **Connectors**: The ways in which these components interact, such as through function calls or data exchanges. E.g: database queries
* **Constraints**: Rules or restrictions that apply to how components and connectors can be organized, such as ensuring there are no circular dependencies.

**3. Pipe and Filter Architecture**

**Structure**:

* Think of this as an assembly line where each station (filter) does a specific job, like transforming raw materials into a finished product.
* **Example**: Imagine a water filtration system where water passes through several filters, each removing different impurities. The water flows from one filter to the next through pipes, and each filter operates independently of the others.

**Key Points**:

* **Filters**: Each filter processes data independently. For example, one filter might clean the water, while another adds minerals.
* **Pipes**: The pipes transfer data from one filter to the next, similar to how water moves from one stage of filtration to another.
* **Independence**: Filters don't know or care about each other. This allows you to change one filter without affecting the others, making the system flexible.
* **筛选器**：每个筛选器独立处理数据。例如，在文本处理系统中，一个过滤器可能会标记文本，另一个过滤器可能会删除停用词，第三个过滤器可能会执行词干提取。
* **管道**：管道将数据从一个过滤器传输到下一个过滤器，确保数据在系统中无缝流动。

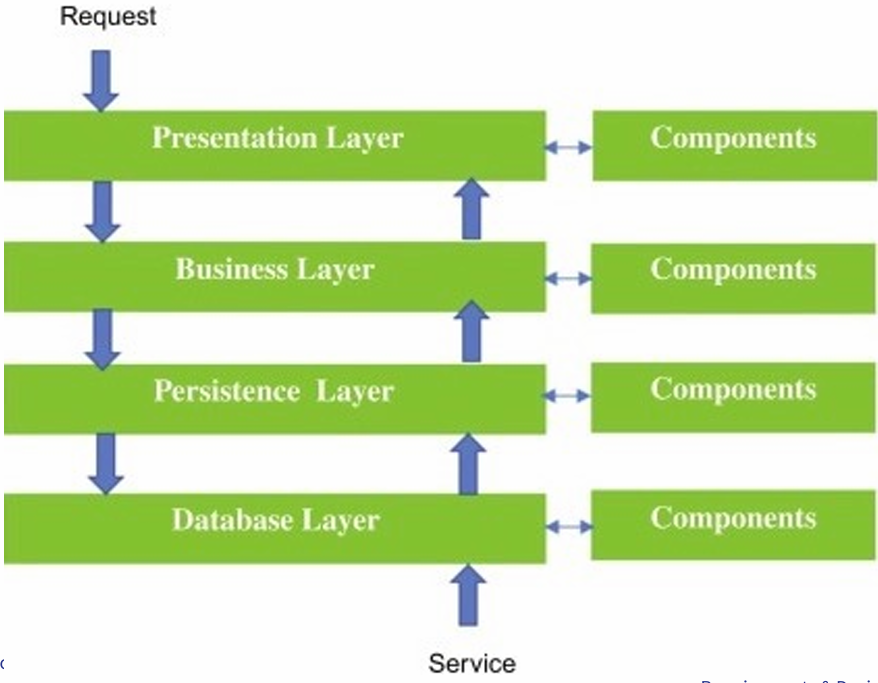
**Advantages of Pipe and Filter Architecture:**

1. **Modularity**:
   * Filters can be developed and tested independently.
   * Easy to modify or replace filters without impacting the entire system.
2. **Reusability**:
   * Filters can be reused across different systems.
   * Common processing tasks can be encapsulated in reusable filters.
3. **Scalability**:
   * The system can be easily scaled by adding more filters or pipes.
   * Can handle varying data loads effectively by parallelizing filters.
4. **Maintainability**:
   * Clear separation of concerns makes the system easier to understand and maintain.
   * Bugs can be isolated within individual filters, simplifying debugging and testing.

**Disadvantages of Pipe and Filter Architecture:**

1. **Performance Overhead**:
   * Data needs to be transferred between filters, which can introduce latency.
   * The overhead of context switching between filters can impact performance.
2. **Data Transformation Cost**:
   * Each filter may need to transform data into a specific format before passing it to the next filter, adding to processing time.
3. **Complexity in Data Dependencies**:
   * If filters need to share state or context, managing these dependencies can become complex.
   * Ensuring data consistency between filters can be challenging.
4. **Increased Resource Usage**:
   * Each filter operates as an independent unit, potentially leading to higher resource consumption (CPU, memory).

**4. Layered Architecture**

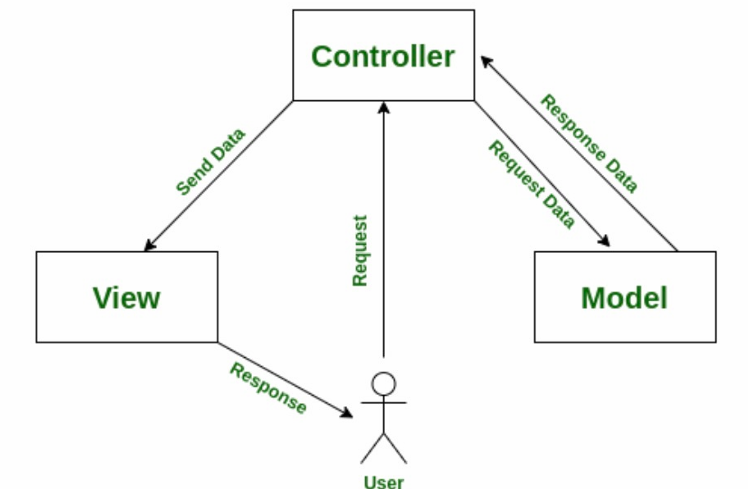
****

**Structure**:

* Think of this as a multi-layer cake, where each layer has a specific role. The top layers rely on the layers below them to function properly.
* **Example**: Consider a banking system with layers such as:
  1. **Presentation Layer**: The user interface where customers interact with the bank
  2. **Business Logic Layer**: The rules and processes that manage bank operations like transactions and account management
  3. **Data Layer**: Where all the customer and transaction data are stored

**Key Points**:

* **Separation of Concerns**: Each layer has a distinct responsibility, making the system easier to understand and modify.
* **Interaction**: Components within a layer interact with each other and with the adjacent layers. For example, the user interface interacts with the business logic, which in turn interacts with the data layer.
* **Advantages**: Easier to maintain and enhance. Changes in one layer usually only affect that layer and the ones directly connected to it.
* **Disadvantages**: Not all systems fit neatly into layers. Performance can suffer if there's too much separation between high-level functions and their implementations.

**5. Model-View-Controller (MVC) Architecture** 100%****

**Structure**:

* Imagine a theater production where:
  + **Model**: The script that contains all the data and logic of the play.
  + **View**: The stage where the play is performed and visible to the audience.
  + **Controller**: The director who ensures the script is followed and manages how the play is performed.

**Key Points**:

* **Model**: Manages the data and business logic. In our theater example, this is the script.
* **View**: Displays the data to the user. This is the stage and everything the audience sees.
* **Controller**: Handles input from the user and updates the model or view accordingly. This is the director managing the play's execution.

**Example**:

* In a web application, the **Model** might be the database, the **View** is the web page, and the **Controller** handles user inputs like clicking a button or submitting a form.

**Benefits of MVC**

* **Separation of Concerns**: Each component has a specific role, making the system easier to manage and update.
* **Reusability**: Components can be reused in other systems. For example, the View can be updated without changing the Model or Controller.
* **Maintainability**: Easier to manage and debug because changes in one component do not heavily affect the others.

**场景**：用户想要借出一本书。

1. **用户操作（输入）：**
   * 用户在在线图书馆系统中搜索“了不起的盖茨比”并决定查看。
2. **控制器**：
   * 接收输入（搜索查询和结帐请求）。
   * 告诉模型找到图书并将其状态更新为“已签出”。
   * 更新视图以反映更改（显示“了不起的盖茨比”不再可用）。
3. **型号**：
   * 在数据库中搜索“了不起的盖茨比”。
   * 将“了不起的盖茨比”的状态更改为“已签出”。
   * 更新用户的记录以显示他们已签出“了不起的盖茨比”。
4. **视图**：
   * 更新网页以从可用书籍列表中删除“了不起的盖茨比”。
   * 向用户显示一条确认消息，表明他们已成功签出图书。

**6. Other Architectures**

**Examples**:

* **Client-Server Architecture**: Like a restaurant where the client (customer) orders food from the server (waiter), who then retrieves it from the kitchen (server).
* **Process Control Architecture**: Think of an industrial plant where various sensors and controllers manage and automate processes.
* **Microservices Architecture**: Breaks down an application into small, loosely connected services, each handling a specific function.
* **Event-Driven Architecture**: Systems where events trigger specific actions or workflows.
* **Service-Oriented Architecture**: Uses services as building blocks to create business applications.

**7. Design Rationale**

**Definition**:

* The reasons behind design decisions. It’s like explaining why you chose a particular route to reach your destination.

**Importance**:

* Helps in understanding the reasoning behind decisions, which is crucial when updating the system or introducing new team members.
* Supports decision-making by documenting the pros and cons of various alternatives.

**Components**:

* **Issues**: The problems or requirements that needed addressing.
* **Proposals**: The different solutions considered.
* **Criteria**: The standards used to evaluate the proposals.
* **Arguments**: The reasoning for and against each proposal.
* **Decision**: The final choice and why it was made

**8. Design Rationale Document Example**

**Issue**: How to implement the database engine?

**Proposals**:

1. **Object-Oriented Database**:
   * **Pros**: Handles complex data relationships well, aligns with modern practices.
   * **Cons**: May be slow for large datasets, poor integration with some existing technologies.
2. **Relational Database**:
   * **Pros**: High performance, well-supported, integrates well with many systems.
   * **Cons**: Not ideal for complex data relationships.
3. **File System**:
   * **Pros**: Simple for data that's infrequently accessed, good for long-term storage.
   * **Cons**: Requires custom code, not suitable for complex queries.

**Criteria**: Need to integrate with existing technologies.

**Resolution**: Choose the relational database due to its robust performance and compatibility with other systems.