**Software Architectural Styles Overview**

Below is a table summarizing the advantages and disadvantages of different software architectural styles.

|  |  |  |
| --- | --- | --- |
| **Architectural Style** | **Advantages** | **Disadvantages** |
| **Layered Architecture** | Simple and easy to understand - Allows separation of concerns - Promotes reusability | Can become rigid - Difficult to change layers independently - Can lead to performance bottlenecks |
| **Client-Server** | - Clear separation of responsibilities - Easy to scale (e.g., adding more servers) - Simple to implement | Single point of failure (the server) - Performance can be limited by network speed - Scaling the server can be complex |
| **Event-Driven** | Promotes loose coupling - Highly scalable and flexible - Can handle asynchronous processing effectively | Complex to manage and debug - Hard to guarantee order of events - Potential for high latency in processing |
| **Microservices** | - Each service is independent and scalable - Technology agnostic (services can use different languages/technologies) - Easier to deploy and update services individually | Complex to manage multiple services - Increased overhead in communication - Requires strong DevOps and monitoring practices |
| **Repository Style** | - Centralizes data management - Promotes consistency - Easy to implement changes across multiple components | - Can become a bottleneck - Tight coupling between components - Poor scalability and performance |

**Social Media Application Architecture**

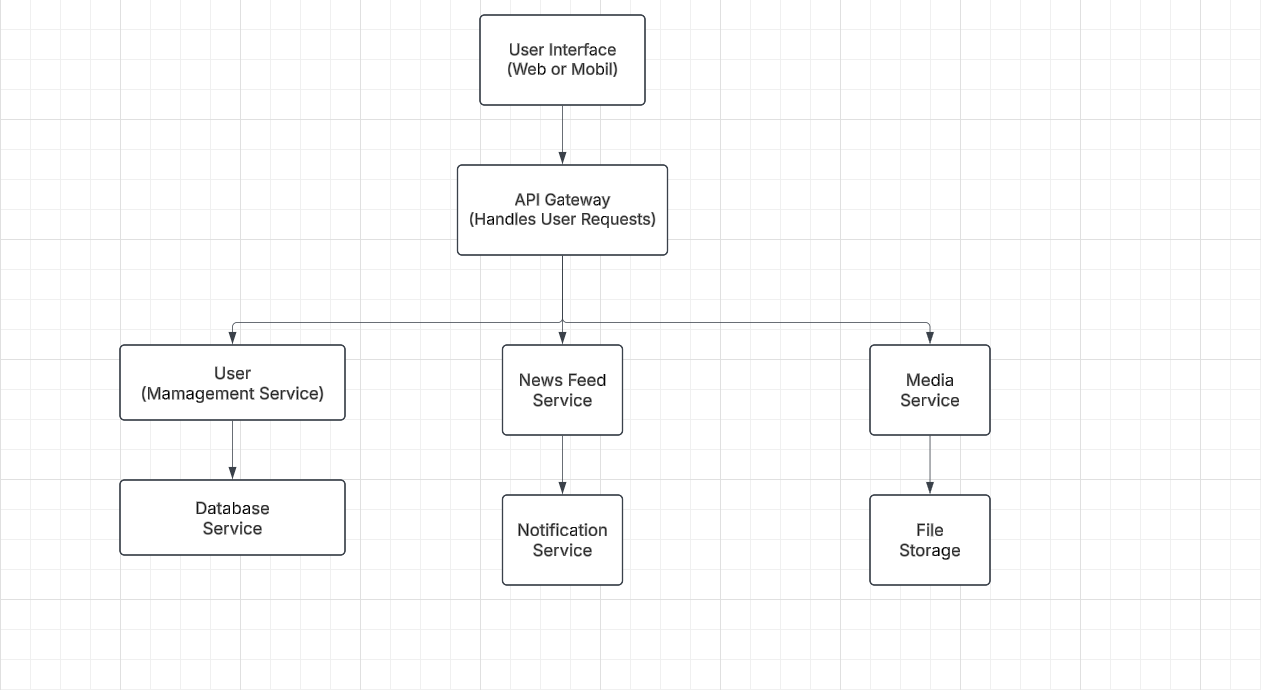
Let’s select a social media application (e.g., Facebook, Twitter, or Instagram) and analyze its software architecture.

**Architecture Style: Microservices**

A social media application typically follows the **Microservices Architecture** style. This is because social media platforms consist of several independent services like user management, messaging, notifications, news feed, and media management, which can be developed, deployed, and scaled independently.

**Architecture Block Diagram**

A simple block diagram illustrating the components and interactions of a social media platform might look like this:



**Daily Software Application (Social Media)**

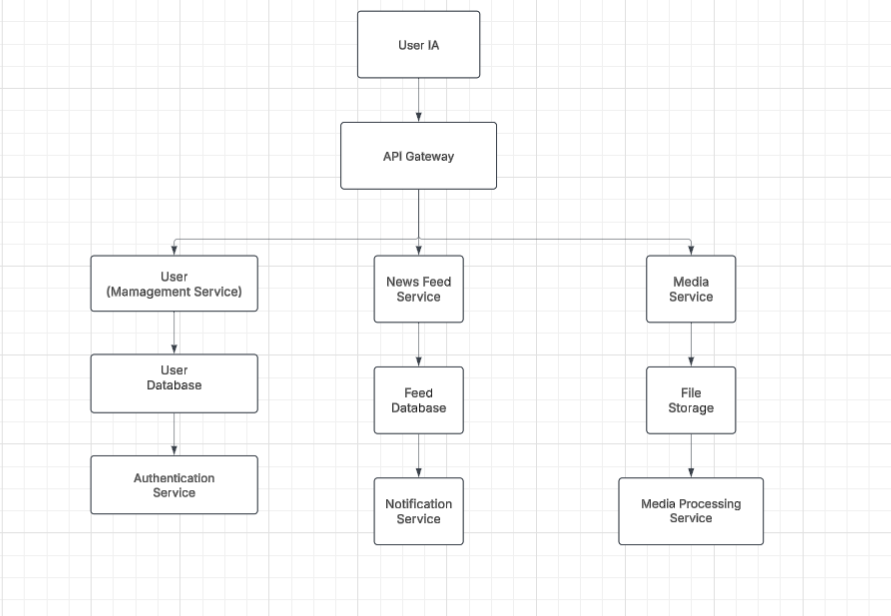
Let’s consider Instagram as a daily software application to analyze.

**Key Components of Instagram and their Relationships**

1. **User Interface (UI):** The app or web interface that allows users to interact with the platform.
2. **API Gateway:** Handles requests from the client and forwards them to the appropriate services.
3. **User Management Service:** Manages user authentication, profile details, and account settings.
4. **News Feed Service:** Generates and serves the content for a user’s feed, typically a personalized stream of posts from followed users.
5. **Media Service:** Handles image and video uploads, storage, and processing.
6. **Notification Service:** Sends notifications to users about activities like likes, comments, follows, and messages.
7. **Database Services:** Stores user data, media, messages, etc.
8. **File Storage:** Manages the physical storage of media files.

**UML Component Diagram**

Here's a basic UML component diagram of Instagram's architecture:



**Explanation of Components:**

* **User UI:** The front-end, which can be a web or mobile application where users interact.
* **API Gateway:** Serves as a central point to handle all requests to various services.
* **Service Components:**
  + **User Management Service**: Handles account creation, login, and user details.
  + **News Feed Service**: Personalizes and serves the user’s feed.
  + **Media Management Service**: Deals with uploading, storing, and retrieving media.
* **Database and Storage:**
  + **User Database**: Stores user profiles and data.
  + **Feed Database**: Stores posts and their metadata.
  + **File Storage**: Manages physical files, such as images or videos.