Architectural style

"An Architectural style is a transformation that is imposed on the design of an entire system. The intent is to establish a structure for all components of the system."

What do SE architectural styles describe?

(1) **A set of components** (e.g., a database, computational modules) that perform a function required by a system.

(2) **A set of connectors** that enable "communication, coordination and cooperation" among components.

(3) **Constraints** that define how components can be integrated to form the system.

(4) **Semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

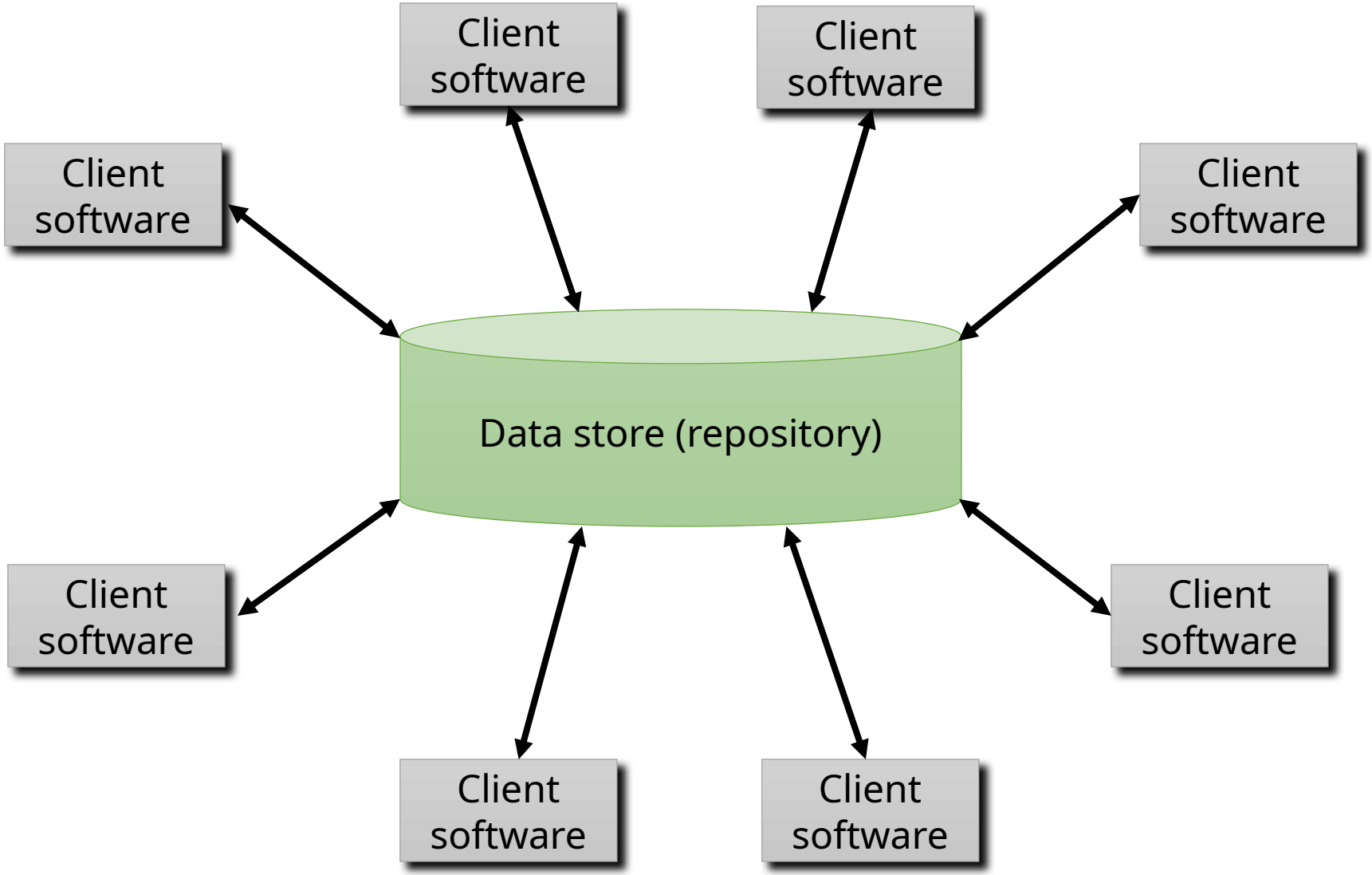
Common SE architectures

- Data-centered (repository)
- Client-server
- Model-view-controller
- Layered architectures
- Microservice architecture

Studying SE architectural styles

Description	A brief overview of the architecture: • what does it do? • What is the rational? • Which problem does it address?
When used	Applicability: • What are the situations in which the style can be applied? • What are examples of poor designs that the style can address?
Advantages	What are the benefits of using the style?
Disadvantages	What are the probable negative consequences?

Data-centered (repository) architectures



Data-centered (repository) architectures

Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
When used	You should use this pattern when you have a system in which **large volumes of information** are generated that has to be stored for a long time. You may also use it in **data-driven systems** where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be **independent**—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed **consistently** (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a **single point of failure** so problems in the repository affect the whole system. May be **inefficiencies** in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

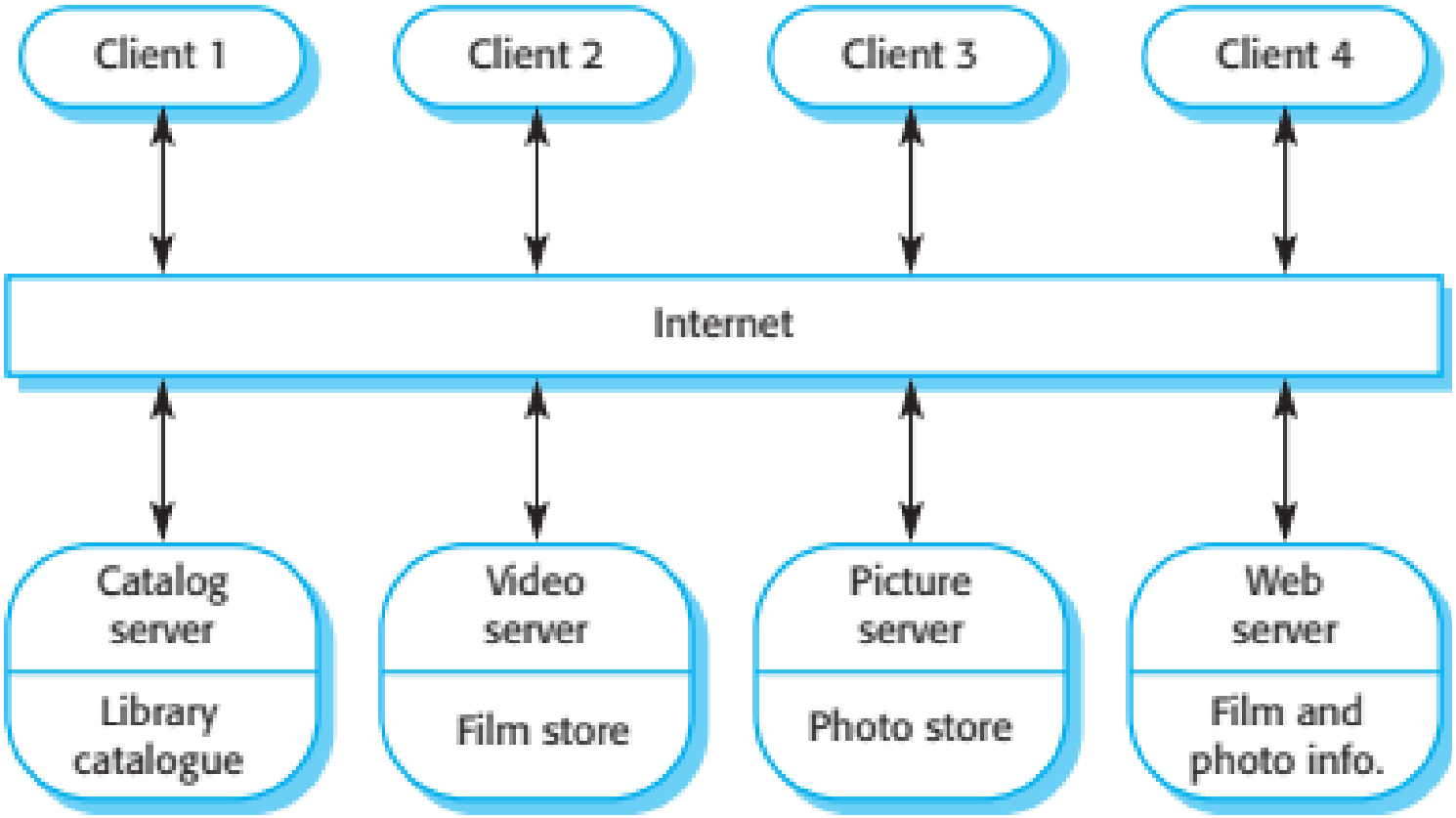
Client-server architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

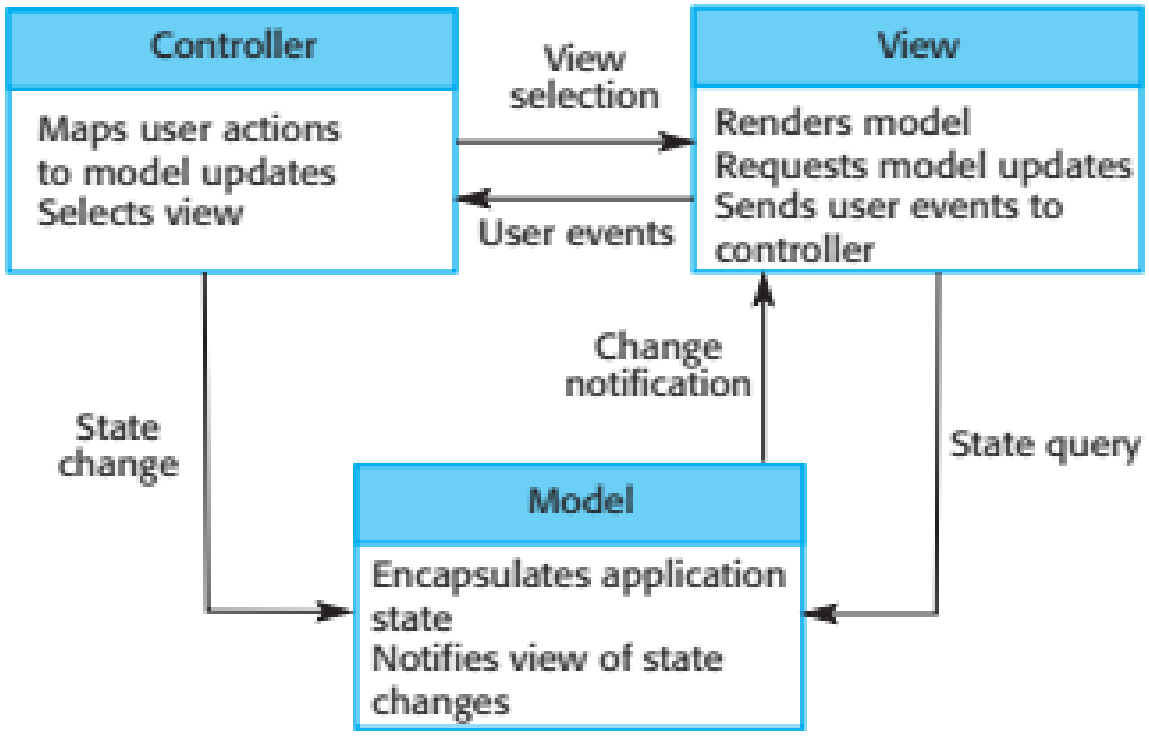
Client-server architecture

Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

A client–server architecture for a film library



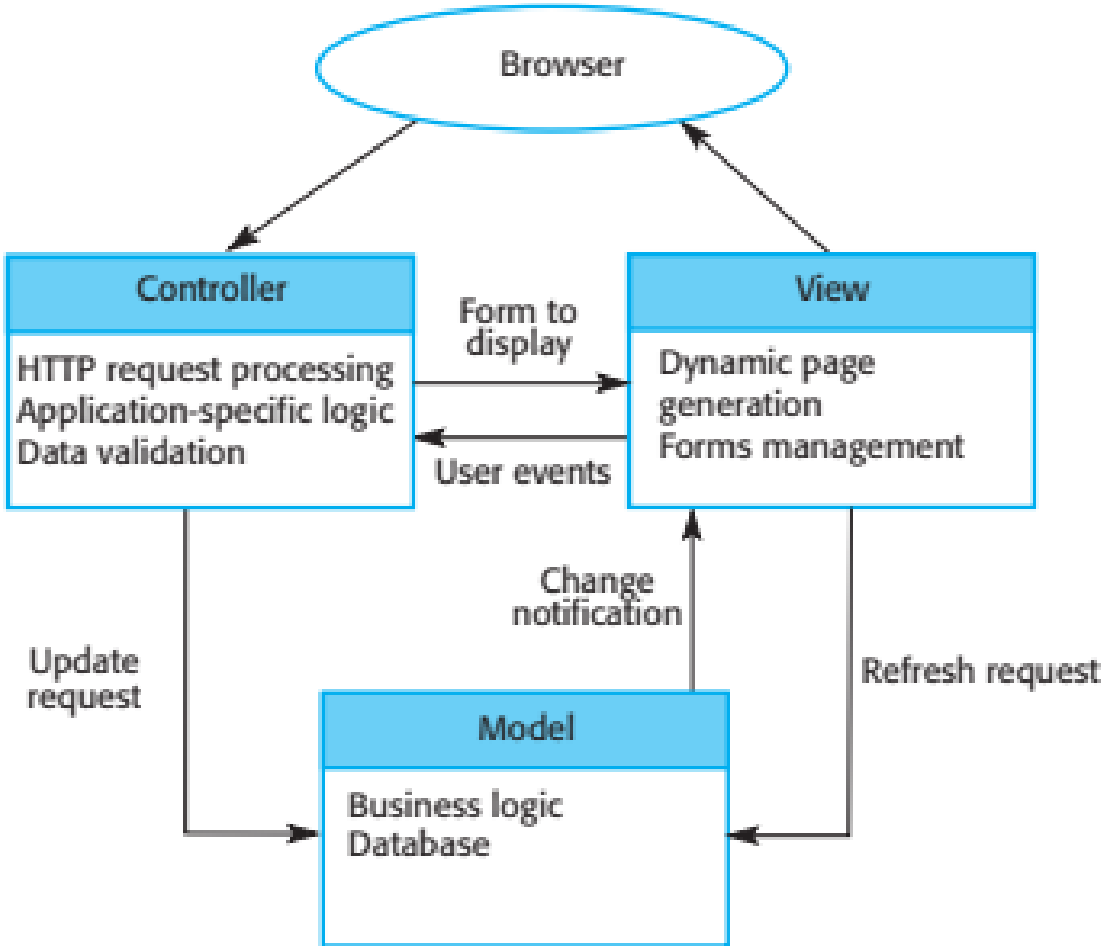
Model-View-Controller architectures



Model-View-Controller (MVC) architectures

Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

Web application architecture using the MVC pattern (run-time architecture)



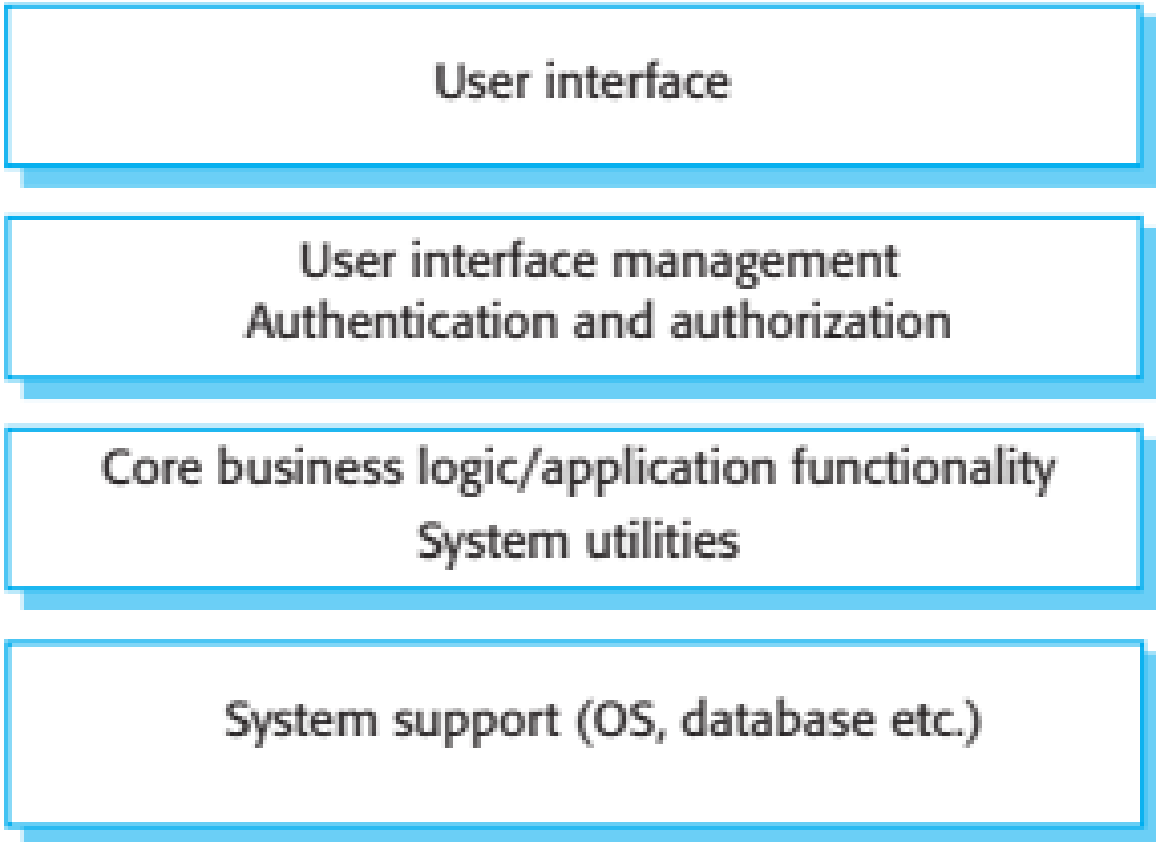
Layered architecture

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

Layered architectures

Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

A generic layered architecture

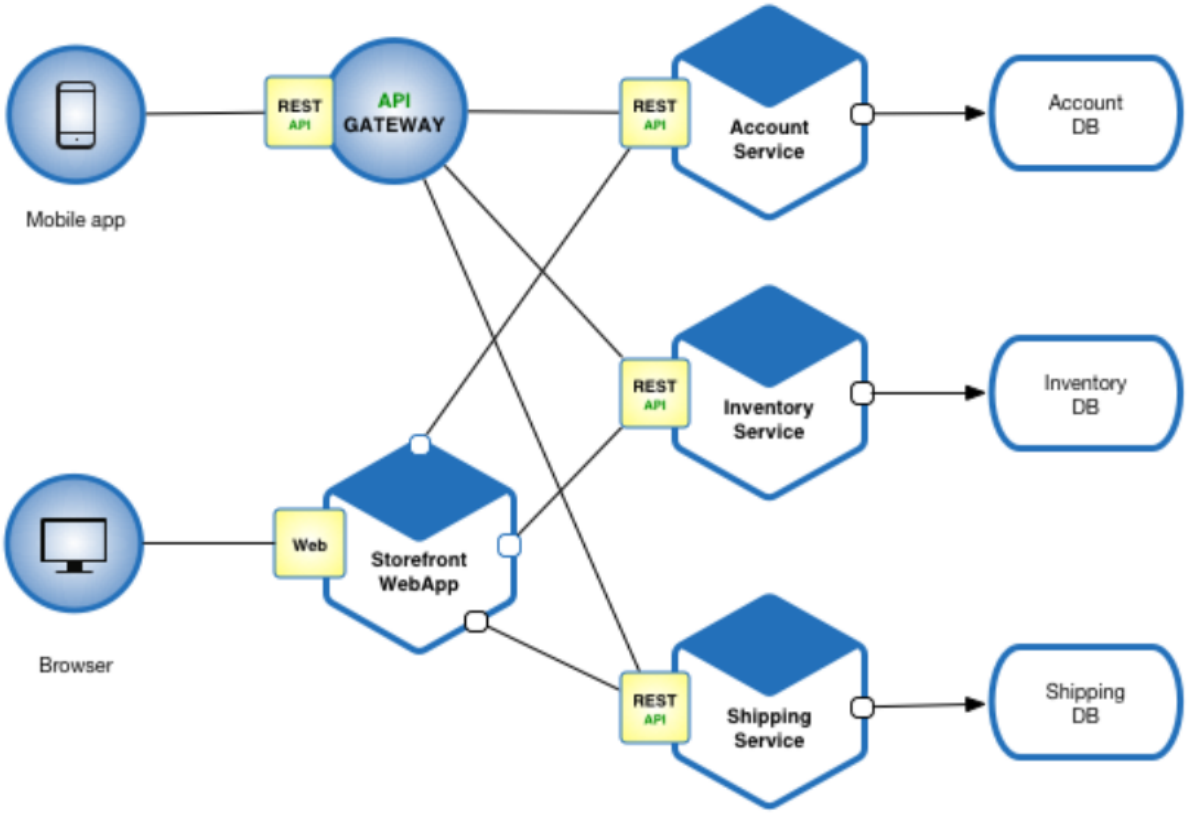


Localize machine dependencies in inner layers, this makes it easier to provide multi-platform implementations of application. Only the inner layer need be re-implemented to take care of different OS, hardware and database

15

Microservice

An architectural style that structures an application as a collection of services.



Advantages

- Enables the continuous delivery and deployment of large, complex applications.
  - Improved maintainability - each service is relatively small and so is easier to understand and change
  - Better testability - services are smaller and faster to test
  - Better deployability - services can be deployed independently
  - It enables you to organize the development effort around multiple, autonomous teams. Each (so called two pizza) team owns and is responsible for one or more services. Each team can develop, test, deploy and scale their services independently of all of the other teams.

Advantages

- Each microservice is relatively small:
  - Easier for a developer to understand
  - New developer can quickly become productive
  - The application starts faster, which makes developers more productive, and speeds up deployments

Advantages

- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests.
- In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.
- Eliminates any long-term commitment to a technology stack.
- When developing a new service you can pick a new technology stack.
- Similarly, when making major changes to an existing service you can rewrite it using a new technology stack.

Drawbacks

- Developers must deal with the additional complexity of creating a distributed system:
  - Developers must implement the inter-service communication mechanism and deal with partial failure
  - Implementing requests that span multiple services is more difficult
  - Testing the interactions between services is more difficult
  - Implementing requests that span multiple services requires careful coordination between the teams
  - Developer tools/IDEs are oriented on building monolithic applications and don’t provide explicit support for developing distributed applications.
- Deployment complexity. In production, there is also

Drawbacks

- Increased memory consumption. The microservice architecture replaces N monolithic application instances with NxM services instances.
- If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes.
- Moreover, if each service runs on its own VM (e.g. EC2 instance), as is the case at Netflix, the overhead is even higher.