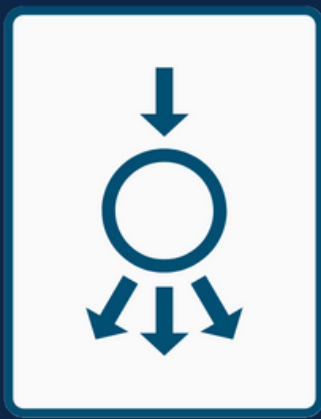


# Design Patterns

## Used Across Different Software Layers



**Load Balancer  
Layer**



**Application  
Server Layer**



**API Gateway  
Layer**



DesignGurus.io

# Client Layer



**Client-side Load Balancing:** Distributes requests from the client directly to the available service instances.

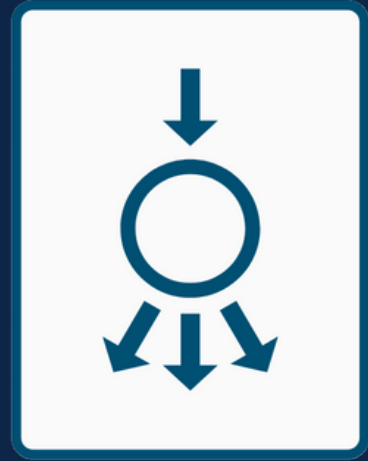
**State Pattern:** Allows an object to alter its behavior when its internal state changes.

**Composite UI Pattern:** Composes responses from multiple microservices to render the complete UI.



DesignGurus.io

# Load Balancer Layer



**Geographical Distribution:** Routes traffic based on the geographical location of the client.

**Health Checks:** Periodically checks the health of the servers and routes traffic only to healthy ones.

**Affinity Based Routing:** Routes the user's request to the same server for maintaining session persistence.

**Least Connections:** Routes traffic to the server with the fewest active connections.

# API Gateway Layer



**Backend for Frontend (BFF):** Tailors API responses to the needs of individual client types.

**Circuit Breaker:** Detects failures and prevents applications from trying to perform actions that are doomed to fail.

**Retry Pattern:** Helps to handle transient failures when it tries to connect to a remote service or network resource.

**Request Collapsing:** Collapses multiple requests for the same operation into a single request.

# Web Server Layer



**Page Cache Pattern:** Stores the output of expensive operations and reuse it to avoid duplicated work.

**Compression Pattern:** Reduces the size of the response to improve load times.

**Lazy Loading:** Defers initialization of an object until the point at which it is needed.

**Content Negotiation Pattern:** The server generates different versions of a resource and serves the one matching the client's criteria.

# Application Server Layer



**Saga Pattern:** Manages long-running transactions and deals with failures and compensating transactions.

**CQRS (Command Query Responsibility Segregation):** Separates read and write operations to improve performance and scalability.

**Proxy Pattern:** Provides a surrogate or placeholder for another object to control access to it.

**Chain of Responsibility:** Passes the request along a chain of handlers.

# Caching Layer



**Sidecar Caching:** Deploy a dedicated cache alongside each microservice to provide isolated and scalable caching functionality.

**Cache Chaining:** Arrange multiple cache layers hierarchically to handle different granularity or lifetime, querying each layer sequentially on a cache miss.

**Time-to-Live (TTL) Caching:** Assigns a predefined lifespan to each cache entry, removing or refreshing the entry once its lifespan expires.

# CDN Layer



**Prefetching:** Anticipates user actions and loads resources ahead of time.

**Parallel Requesting:** Makes multiple requests in parallel to improve load times.

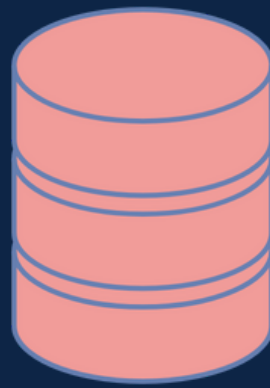
**Edge Computing:** Processes data closer to the location where it is needed.

**Domain Sharding:** Splits resources across multiple domains to increase parallel downloads.

**Adaptive Image Delivery:** Delivers images tailored to the device and user context.



# Database Layer



**Sharding Pattern:** Distributes data across multiple databases to improve scalability.

**Replication Pattern:** Keeps copies of data in multiple locations for availability and reliability.

**Read-Replica Pattern:** Uses read replicas to offload read operations from the primary database instance.

**Query Object Pattern:** An object that represents a database query.



DesignGurus.io

Learn about the  
Microservices Design Patterns  
in Grokking **Microservices**  
**Design Patterns** from  
**DesignGurus.io**



DesignGurus.io