System Design Fundamentals

Course code:CB2001101

What is a system?

- Computing System
- Real World System

What is System Design?

- System design is the process of defining the **architecture, interfaces, and data** for a system that satisfies specific requirements. System design meets the needs of your business or organization through coherent and efficient systems. It requires a systematic approach to building and engineering systems. A good system design requires us to think about everything, from infrastructure all the way down to the data and how it's stored.

Why is System Design so important?

System design helps us define a solution that meets the business requirements. It is one of the earliest decisions we can make when building a system. Often it is essential to think from a high level as these decisions are very difficult to correct later. It also makes it easier to reason about and manage architectural changes as the system evolves.

Steps for approaching this system design

1. **Understand the requirements:** Before starting the design process, it is important to understand the requirements and constraints of the system. This includes gathering information about the problem space, performance requirements, scalability needs, and security concerns.
2. **Identify the major components:** Identify the major components of the system and how they interact with each other. This includes determining the relationships between different components and how they contribute to the overall functionality of the system.
3. **Choose appropriate technology:** Based on the requirements and components, choose the appropriate technology to implement the system. This may involve choosing hardware and software platforms, databases, programming languages, and tools.
4. **Define the interface:** Define the interface between different components of the system, including APIs, protocols, and data formats.
5. **Design the data model:** Design the data model for the system, including the schema for the database, the structure of data files, and the data flow between components.
6. **Consider scalability and performance:** Consider scalability and performance implications of the design, including factors such as load balancing, caching, and database optimization.
7. **Test and validate the design:** Validate the design by testing the system with realistic data and use cases, and make changes as needed to address any issues that arise.
8. **Deploy and maintain the system:** Finally, deploy the system and maintain it over time, including fixing bugs, updating components, and adding new features as needed.

Functional Requirements

- These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.
- These are represented or stated in the form of input to be given to the system, the operation performed, and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Non-Functional Requirements

- These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:
- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Difference

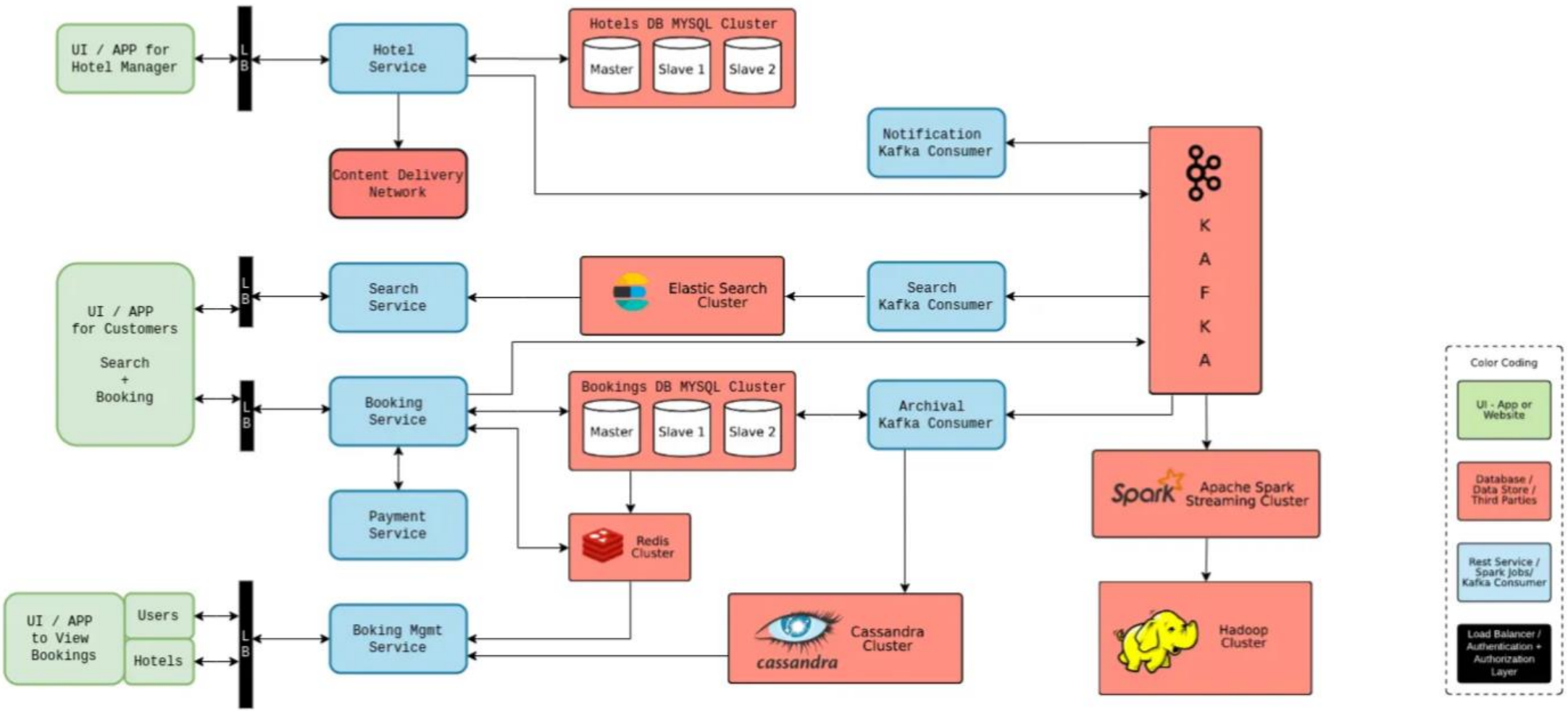
Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.

Contd..

Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example 1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.	Example 1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

Example: Airbnb System Architecture

Airbnb/Booking.com System Design



https://interviewnoodle.com/hotel-booking-system-design-airbnb-1283bbef71ac

Functional Requirements

1. **Home owner**
    - Able to register the hotel on the platform
    - Add/update/delete room types in the Hotel
    - Add/update/delete rooms of given room types
    - Define the price and inventory of room types on a daily basis
2. **User/ Customer**
    - Able to search available hotels by city, check-in, and check-out date
    - able to select a hotel and see all the available hotel types and their prices
    - Able to select the desired room type and proceed with the booking
    - Receive the notification about the booking details once the booking is completed

Non-Functional Requirement

- System handling operations related to hotel managers/Homeowners and booking flows must be highly consistent
- A Discovery platform showing hotels to the customers should be highly available
- The system should have low latency
- The system should be highly scalable to handle the increasing number of hotels and the number of new incoming customers
- The system should be able to handle concurrent requests such that no two customers should be able to book the same room on a particular day

Rest APIs

1.Register hotel
Post/ hotel/ Register
2.Add room type in a hotel
POST /hotel/{hotel_id}/room-type
3.Add Room In Hotel
POST /hotel/{hotel_id}/room-type/{room_type_id}/room
4.Return the list of nearby hotels
GET /hotels/location/{location_id}
5.Given a hotel return its detail
GET /hotel/{hotel_id}
6.Book a hotel room
POST /booking
7.Return the bookings for a user
GET /user/{user_id}/bookings
8.Returns the bookings for a hotel
GET /hotel/{hotel_id}/bookings
9.Check-In to hotel
PUT /booking/{booking_id}/check-in
10.Check-Out from hotel
PUT /booking/{booking_id}/check-out

APIs 1,2,3 will be part of the **Hotel Management Service.**
APIs 4,5 will be part of the **Discover Platform.**
APIs 6,9,10 will be part of the **Booking Service.**
APIs 7,8 will be part of the **Booking History Service.**

API

- An API is a set of definitions and protocols for building and integrating application software. It's sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response).
- In other words, if you want to interact with a computer or system to retrieve information or perform a function, an API helps you communicate what you want to that system so it can understand and fulfill the request.
- You can think of an API as a mediator between the users or clients and the resources or web services they want to get. **It's also a way for an organization to share resources and information while maintaining security, control, and authentication—determining who gets access to what.**
- Another advantage of an API is that you don't have to know the specifics of caching—**how your resource is retrieved** or **where it comes from**.

Restful APIs

What is REST?

- Representational State Transfer (REST) is a software architecture that imposes conditions on how an API should work. REST was initially created as a guideline to manage communication on a complex network like the internet. You can use REST-based architecture to support high-performing and reliable communication at scale. You can easily implement and modify it, bringing visibility and cross-platform portability to any API system.
- API developers can design APIs using several different architectures. APIs that follow the REST architectural style are called REST APIs. Web services that implement REST architecture are called RESTful web services. The term RESTful API generally refers to RESTful web APIs. However, you can use the terms REST API and RESTful API interchangeably.

Contd.

The following are some of the principles of the REST architectural style:

Uniform interface

- The uniform interface is fundamental to the design of any RESTful webservice. It indicates that the server transfers information in a standard format. The formatted resource is called a representation in REST. This format can be different from the internal representation of the resource on the server application. For example, the server can store data as text but send it in an HTML representation format.

Statelessness

- In REST architecture, statelessness refers to a communication method in which the server completes every client request independently of all previous requests. Clients can request resources in any order, and every request is stateless or isolated from other requests. This REST API design constraint implies that the server can completely understand and fulfill the request every time.

Layered system

- In a layered system architecture, the client can connect to other authorized intermediaries between the client and server, and it will still receive responses from the server. Servers can also pass on requests to other servers. You can design your RESTful web service to run on several servers with multiple layers such as security, application, and business logic, working together to fulfill client requests. These layers remain invisible to the client.

Cacheability

- RESTful web services support caching, which is the process of storing some responses on the client or on an intermediary to improve server response time. For example, suppose that you visit a website that has common header and footer images on every page. Every time you visit a new website page, the server must resend the same images. To avoid this, the client caches or stores these images after the first response and then uses the images directly from the cache. RESTful web services control caching by using API responses that define themselves as cacheable or noncacheable.

Code on demand

- In REST architectural style, servers can temporarily extend or customize client functionality by transferring software programming code to the client. For example, when you fill a registration form on any website, your browser immediately highlights any mistakes you make, such as incorrect phone numbers. It can do this because of the code sent by the server.

Contd..

- What are the benefits of RESTful APIs?

RESTful APIs include the following benefits:

Scalability

- Systems that implement REST APIs can scale efficiently because REST optimizes client-server interactions. Statelessness removes server load because the server does not have to retain past client request information. Well-managed caching partially or completely eliminates some client-server interactions. All these features support scalability without causing communication bottlenecks that reduce performance.

Flexibility

- RESTful web services support total client-server separation. They simplify and decouple various server components so that each part can evolve independently. Platform or technology changes at the server application do not affect the client application. The ability to layer application functions increases flexibility even further. For example, developers can make changes to the database layer without rewriting the application logic.

Independence

- REST APIs are independent of the technology used. You can write both client and server applications in various programming languages without affecting the API design. You can also change the underlying technology on either side without affecting the communication.

What does the RESTful API client request contain?

- *GET*
- Clients use GET to access resources that are located at the specified URL on the server. They can cache GET requests and send parameters in the RESTful API request to instruct the server to filter data before sending.
- *POST*
- Clients use POST to send data to the server. They include the data representation with the request. Sending the same POST request multiple times has the side effect of creating the same resource multiple times.
- *PUT*
- Clients use PUT to update existing resources on the server. Unlike POST, sending the same PUT request multiple times in a RESTful web service gives the same result.
- *DELETE*
- Clients use the DELETE request to remove the resource. A DELETE request can change the server state. However, if the user does not have appropriate authentication, the request fails.

Class Assignment