(Part 1)

API Design

Software System Design

Spring 2024 @ Ali Madooei

Learning Outcomes

By the end of this lecture, you should be able to:

- Understand the significance of Web APIs in software development.
- Learn the principles of designing RESTful APIs.
- Explore other API design methodologies like GraphQL and RPC.
- Understand how the web works and the role of DNS in web communication.

Use Case: University Student Hub App

An app designed for university students serves as a centralized hub for accessing a variety of resources both on and off campus. The app includes a news feed that provides information about the latest events, courses, clubs, and other resources. It utilizes dynamic data retrieval to fetch personalized content, supports real-time updates for the most up-to-date information, and enables complex queries to aggregate data from multiple sources. The primary goals of the app are to enhance student engagement, facilitate access to educational and extracurricular activities, and improve communication within the university community.

Use Case: Boom Virtual Meeting Service

The service offered by Boom, a virtual meeting app, specializes in recording online video meetings, processing the videos to improve compression and quality, and providing auto-captioning for transcription purposes. To achieve this, the service relies on efficient video processing algorithms, low-latency communication for real-time video processing, bidirectional streaming for simultaneous upload and download of video data, and efficient serialization to minimize bandwidth usage. The main goal of the service is to enhance the virtual meeting experience by providing high-quality recordings and accessible transcriptions. To facilitate this, a robust and high-performance API is required to manage the complex interactions between the video processing services and other components.

Use Case: To-Do App

A to-do app enables users to manage tasks and organize their daily activities. The app offers features for creating, viewing, editing, and deleting tasks, each with attributes like title, description, due date, and priority. It should have a user-friendly interface, ensure data persistence, and support notifications for upcoming deadlines. The app needs to handle various user interactions efficiently and provide a seamless experience across web and mobile platforms. The backend should offer a consistent and straightforward interface across web and mobile platforms, as well as support caching for quick access to frequently viewed tasks.

Definition & Purpose of APIs

API is a set of rules that defines interactions between software applications, allowing them to communicate and share functionalities.

- **Integration:** Facilitates communication between various software entities, enabling seamless interaction and data sharing.
- **Automation:** Supports task automation, allowing systems to interact programmatically without human intervention.
- **Efficiency:** Optimizes data transfer by enabling extraction of specific data between applications.

Impact & Importance of APIs

APIs are vital, acting as the binding agent for diverse software components, and enabling the creation of sophisticated, feature-rich applications.

- **Modularity:** Promotes development of clear, independent, and manageable software components, enhancing maintainability.
- **Innovation:** Fosters creation of new business models and extension of services, by allowing integration and building upon existing services.
- **Scalability & Performance:** Facilitates efficient scalability of individual components, ensuring high performance and availability.

Types of APIs

APIs serve diverse purposes and use cases, enabling interaction with various software components, services, and systems.

- **Web APIs:** Enable communication over the web, typically using the HTTP/HTTPS protocol.
- **Library-based APIs:** Packaged as libraries or SDKs, providing routines and tools for software development.
- **Operating System APIs:** Allow interaction with underlying OS services, enabling tasks like file handling and memory allocation.
- **Database APIs:** Enable communication between applications and database management systems, allowing CRUD operations on database records.

Introduction to Web APIs

Web APIs allow different software systems to communicate with each other over the web using IPS protocols.

- **Open APIs:** Accessible to developers and the public (e.g., Twitter API).
- **Internal APIs:** Used within a company for internal services.
- **Composite APIs:** Access multiple endpoints in one call; useful for microservices.
- **Partner APIs:** Exposed to specific partners or collaborators.

Detour Begins!

How does the web work?

The Web vs The Internet

- The Internet (or internet) is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to communicate (exchange data).
- The World Wide Web (WWW), or the Web, is an information system consisting of interlinked webpages that enable information sharing over the Internet.
- People often use "the Web" and "the Internet" to mean the same thing.

How the internet works!

- Understanding how the internet works will come in handy when we design software systems.
- For example, when designing Application Programming Interfaces (APIs), you need to understand how applications "communicate" (exchange data) over the Internet.
- As another example, when designing a content-heavy platform, understanding CDNs and web caching mechanisms ensures efficient content delivery, reducing latency, and providing a better user experience.

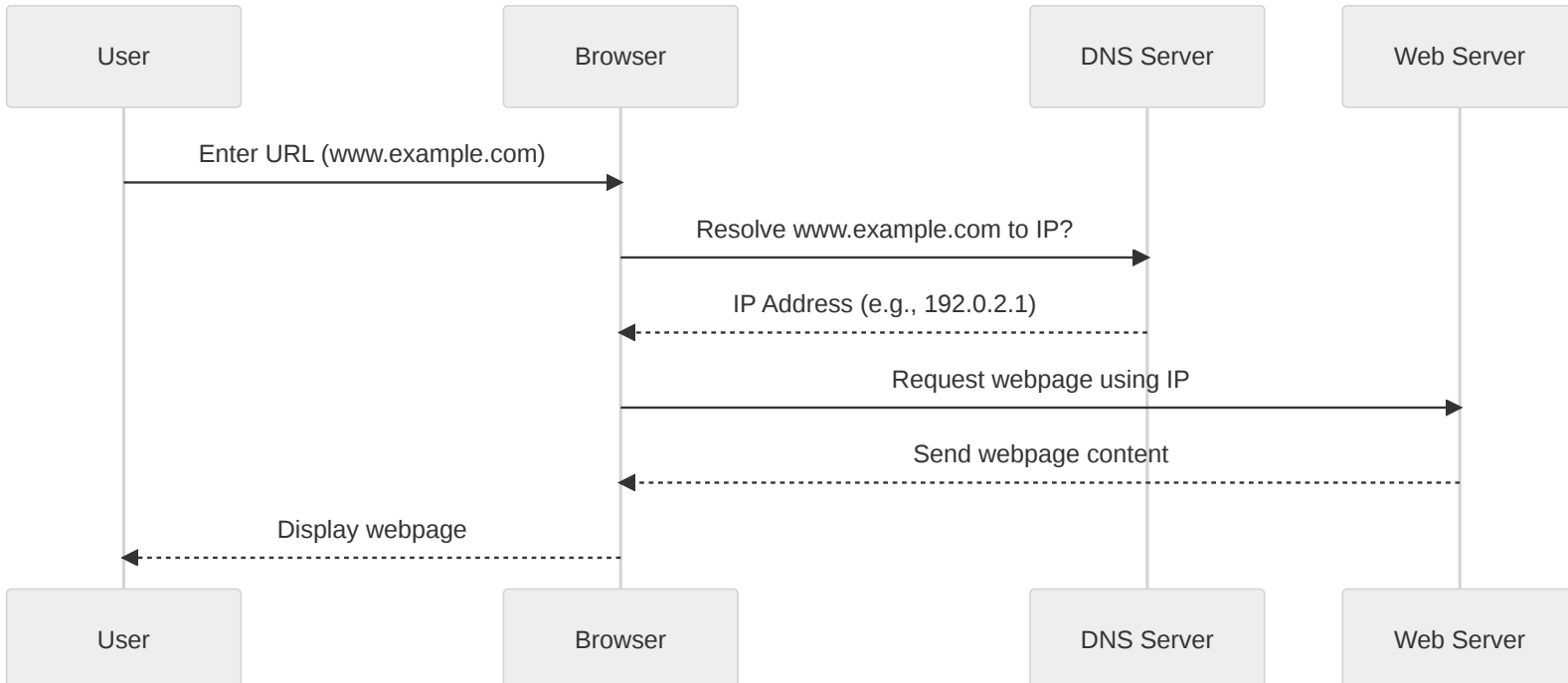
Overview of the Internet’s Architecture

The Internet is a vast network of interconnected devices and systems. Its architecture can be visualized as:

- **End Devices**: Computers, smartphones, IoT devices.
- **Intermediate Devices**: Routers, switches, modems.
- **Networks**: Local Area Networks (LAN), Wide Area Networks (WAN).
- **Connection Types**: Wired (Ethernet), Wireless (Wi-Fi, Cellular).
- **Services**: DNS, CDN, Proxies, Web servers.

This architecture facilitates data exchange and communication globally.

What happens when you type a URL into your browser?



What is a URL?

We use Universal Resource Locator (URL) instead of IP addresses to access resources on the web.

- A URL is a string that provides the location of a resource on the internet and a mechanism for retrieving it.
- Example: `https://www.cs.jhu.edu/faculty/ali-madooei`
  - `https` → Protocol
  - `www.cs.jhu.edu` → Domain
  - `faculty` → Path
  - `ali-madooei` → Resource

What is a Domain Name?

A hierarchical (and easy-to-remember) naming system in place of IPs on the web.

- Each domain name must be unique and have to be registered with private domain registrars (like NameCheap or GoDaddy).
- Example: `www.cs.jhu.edu`
  - `edu` → Top-level domain (TLD) indicates that the website is an educational institution.
  - `jhu` → Domain name (Johns Hopkins University).
  - `cs` → Subdomain (Computer Science, a division of JHU).
  - `www` → Stands for World Wide Web. Can be eliminated!

Decipher a URL: Activity

Look at the URL provided and identify and write down its various components:

`https://shop.example.com/products/item?id=123&color=blue#reviews`

Decipher a URL: Activity Solution

`https://shop.example.com/products/item?id=123&color=blue#reviews`

- **Protocol:** `https`
- **Subdomain:** `shop`
- **Domain:** `example`
- **TLD:** `.com`
- **Path:** `/products`
- **Resource:** `/item`
- **Query Parameters:** `id=123&color=blue`
- **Fragment Identifier:** `#reviews`

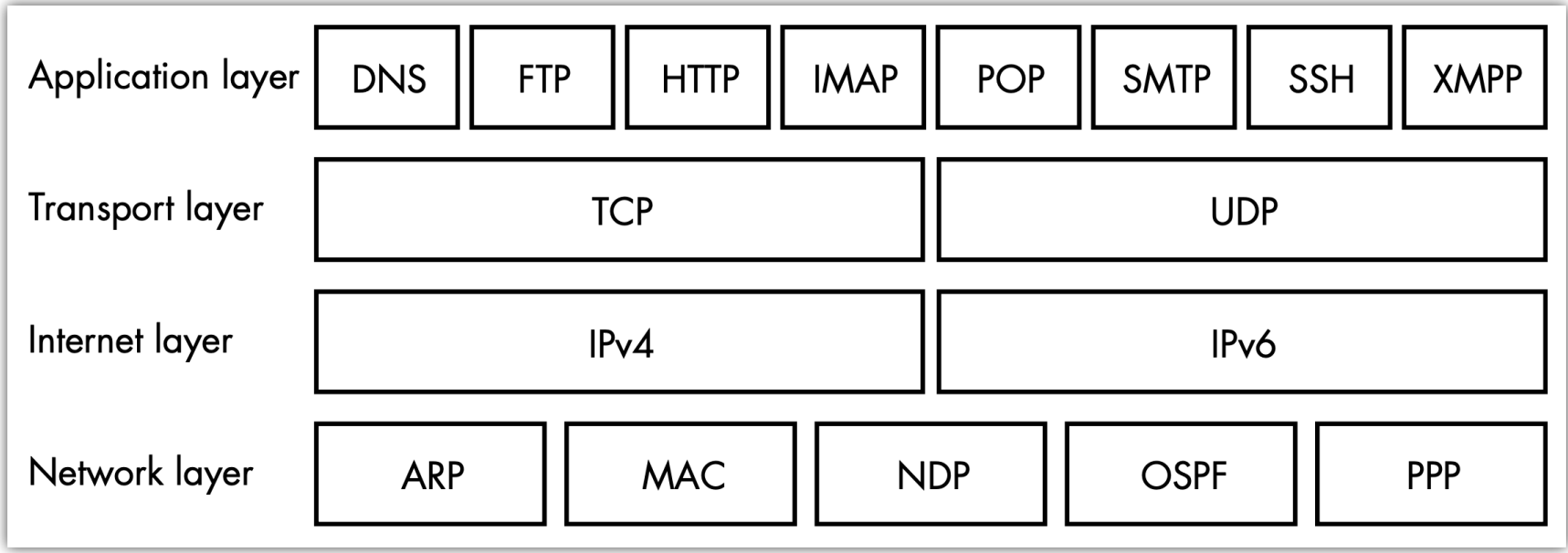
What is a Protocol?

A set of rules that define how data is transmitted over the internet.

- **Example:** HTTP/HTTPS is a protocol used for transmitting web pages over the internet.

Internet Protocol Suite

A framework of several network protocols used for "communication" (data exchange) over the Internet.



Internet Protocol (IP) Addresses

The Internet Protocol (IP) is the network layer communications protocol in the Internet protocol suite.

- Each computer (machine) on the internet has a unique IP that is assigned to it in a systematic and structured manner.
- IPs are issued by the Internet Corporation for Assigned Names and Numbers (ICANN) through internet service providers (ISPs) and other intermediaries.
- IP addresses are binary numbers written in IPv4 (older version) or IPv6 (newer version) syntax.

Domain Name System (DNS)

The internet’s "phone book" (global directory) that maps domain names to IP addresses.

- **Example**: `https://www.cs.jhu.edu/` ➔ `128.220.13.64`
- **Lookup Process**: a series of queries begin, moving from root servers to domain-specific ones, finding the exact address.
  - **Root Server**: Directs query to the relevant Top-Level Domain (TLD) server.
  - **TLD Server**: Points to the correct authoritative nameserver.
  - **Authoritative Nameserver**: Provides the IP address or the next chain link.

Domain Name Lookup

A simplified view of what happens when you view a webpage in a web browser!

- **URL Entry**: User inputs a URL.
- **Local Caches**: Your computer first checks if it already knows the IP from previous visits (Browser Cache, OS Cache, Router Cache).
- **ISP Cache**: If not locally, your Internet Service Provider might have the IP cached.
- **DNS Lookup**: If the IP isn’t cached, a series of queries begin, moving from root servers to domain-specific ones, finding the exact address.
- **IP Retrieved**: DNS server returns the corresponding IP address.
- **Connection**: Browser connects to the server at the IP.

From Connection to Communication

- Once you found the IP address and connection is established, transport layer protocols such as TCP allow communication between the client and server.
- However, protocols such TCP do not indicate how the exchanged data should be interpreted.
- The application layer protocols will help to decide how the exchanged data should be interpreted.
- The most important of these is the HTTP!

HyperText Transfer Protocol (HTTP)

Web servers use HTTP to send web pages and their resources to web browsers.

- HTTP is also commonly used to build Application Programming Interfaces (**APIs**).
- It works as a **request-response** protocol in the client-server computing model.
- The software that sends a request is called the **client**, and the **server** responds to the request.
- The exchange is done in **plain text**, which is often *compressed* and *encrypted*.

HTTP Request

An HTTP request consists of the following elements:

- Method (type of request)
- URL (address of the resource)
- Headers (metadata about the request)
- Body (optional content being sent with the request)

```
curl -X POST https://www.example.com/api/data \
     -H "Content-Type: application/json" \
     -d '{"key":"value", "username":"jdoe"}'
```

HTTP Verbs

HTTP methods (or verbs) are used to indicate the action to be performed on a resource.

- **GET**: Retrieves data from a server.
- **POST**: Sends data to a server to create a resource.
- **PUT**: Sends data to a server to update or replace a resource.
- **DELETE**: Sends a request to delete a resource.
- **HEAD**: Retrieves metadata (headers) about a resource.
- **OPTIONS**: Retrieves the available methods for a resource.
- **PATCH**: Sends data to update or modify a resource partially.

HTTP Methods Match-up: Activity

Match the correct HTTP method with the appropriate action:

- Search for a book on Software System Design on Amazon
- Like a YouTube video
- Place an order on DoorDash
- Change your display name on Slack
- Remove an item from your shopping cart
- Sign into your Gradescope account

HTTP Methods Match-up: Activity Solution

- Search for a book on Software System Design on Amazon: **GET**
- Like a YouTube video: **POST**
- Place an order on DoorDash: **POST**
- Change your display name on Slack: **PATCH**
- Remove an item from your shopping cart: **DELETE** (or **PATCH** depending on the API design)
- Sign into your Gradescope account: **POST**

HTTP Response

- **Status Code**: A three-digit code that indicates the status of the response (e.g., 2xx for OK, 3xx for redirection, 4xx for invalid requests, 5xx for server errors).
- **Status Message**: An optional short description of the status code.
- **Content Type**: The type of content contained in the response (e.g., text/html, application/json).
- **Cache-Control**: Instructions for caching the response at intermediate points.
- **Response Body**: The actual content of the response.

```
HTTP/1.1 201 Created
Date: Mon, 11 Sep 2023 18:56:34 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: application/json
Connection: keep-alive
Content-Length: 75
Cache-Control: max-age=3600, must-revalidate

{
    "status": "success",
    "message": "Data successfully received."
    "userId": "12345"
}
```

HTTP Response Codes

HTTP status codes to convey the outcome of API requests accurately.

- **2xx: Success** (e.g., 200 OK, 201 Created) - Indicates successful requests.
- **3xx: Redirection** (e.g., 301 Moved Permanently, 304 Not Modified) - Requires the client to take additional action to complete the request.
- **4xx: Client Errors** (e.g., 400 Bad Request, 404 Not Found) - Signifies errors from the client side.
- **5xx: Server Errors** (e.g., 500 Internal Server Error) - Represents errors from the server side.

HTTP Cache-Control

The `Cache-Control` header provides directives for caching responses at intermediate points.

- **max-age**: Specifies the maximum time a response can be cached.
- **no-store**: Instructs caches not to store the response under any circumstances.
- **must-revalidate**: Response can be stored in caches and can be reused while fresh.
- **public**: Allows any cache to store the response.
- **private**: Indicates that the response is intended for a single user and should not be stored by shared caches.

And many more...

HTTP Content Types

The `Content-Type` header specifies the type of content contained in the response. (Also known as MIME type)

- **text/html**: HTML content.
- **application/json**: JSON data.
- **application/xml**: XML data.

And many more...

HTTP’s Stateless Nature

HTTP was designed to be stateless.

- When a client sends a request to a server, the server processes the request, sends a response, and forgets about the interaction.
- This design was sufficient when web pages were static and read-only.

Stateful vs Stateless

- In a **stateless** connection, each request is treated as an isolated transaction. The server does not remember past interactions, processing every request without knowledge of previous ones.
- In a **stateful** connection, the server remembers the client’s previous interactions. This allows for continuity across multiple interactions.

The Stateless Design of HTTP

HTTP was designed to be stateless to:

- **Simplicity**: Easier for servers to handle requests without maintaining user context.
- **Scalability**: Stateless nature means no need for servers to retain session data, allowing them to handle more requests.
- **Speed**: Each request is independent, reducing server processing time.
- **Flexibility**: Stateless design ensures that each request contains all the necessary information, making it adaptable.

However, modern web applications require some state management for personalized user experiences.

Modern Needs and Stateful HTTP

Today, many websites allow users to interact with them: logging in, adding items to a cart, personalizing content, etc.

- For these functionalities, the server needs to remember the user across multiple requests. This is where the concept of an **"HTTP session"** comes in.
- While stateful connections improve user experience, they introduce *security concerns* that developers must address.

HTTP Session

A conversation (data exchange) between a user’s browser and a web server.

- The server remembers the user across multiple interactions.
- It allows for continuity across multiple interactions.
- HTTP sessions are necessary for functionalities that require the server to remember the user, such as logging in or personalizing content.

Internet **cookies** and **tokens** are two common ways to establish HTTP Session.

Detour Ends!

We will talk about cookies and tokens in future lessons. For now, let’s end this detour and go back to our main topic: API Design.