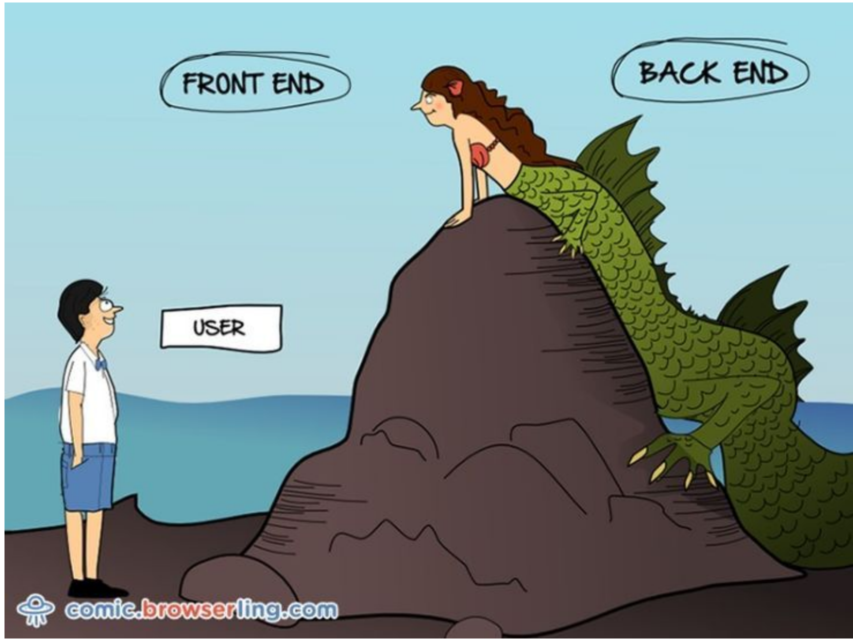
Server Internals

Kameswari Chebrolu

Department of CSE, IIT Bombay



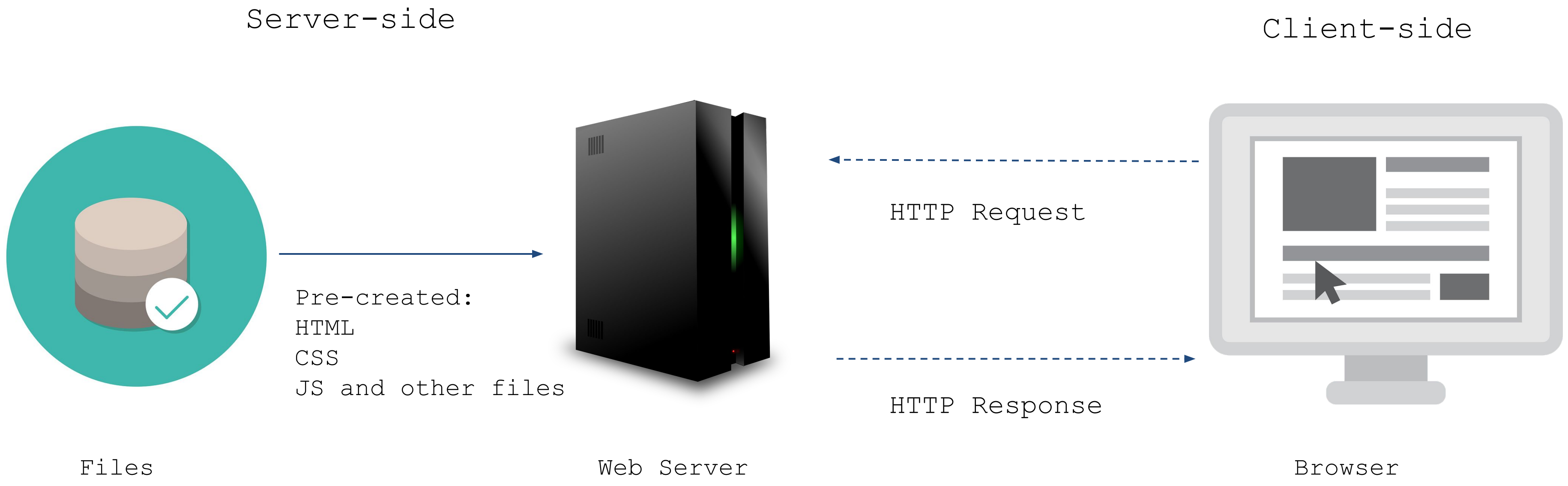
https://i.pinimg.com/736x/d5/12/48/d5124806ed216c46103e65c475d3bb81.jpg

Background: Overall Outline

- ~~What constitutes a webpage?~~
- ~~What goes on inside a Browser?~~
  - ~~What standard security mechanisms implemented?~~
- ~~How do client and server communicate?~~
  - ~~HTTP/HTTPs protocol~~
  - ~~Session Management via cookies and tokens~~
- **How does a web server process requests and generate responses?**
  - **Static vs Dynamic content**

Server Content

- Servers serve two types of content: Static and Dynamic
- Focus: Static (for now)
  - HTML files, image files, javascript files
  - Locate the requested file on file system and return unaltered in HTTP response


Server-side
Client-side
HTTP Request
Pre-created:
HTML
CSS
JS and other files
HTTP Response
Files
Web Server
Browser

- Modern web servers may do some things extra!
  - Dynamically compress large resource files using gzip
    - Reduces bandwidth used in the response
  - Add caching headers in HTTP responses to instruct browser to cache
    - Decreases page load times in future if same file needed!

Web Server Software

- Plays a key role in managing connections and serving static (as well as dynamic content)
- Apache HTTP Server (Apache): Oldest and most widely used open-source web server
  - Process driven architecture (not very memory efficient)
  - Configurability via files
    - main configuration file (apache2.conf)
    - .htaccess files in specific directories (override global settings)

- Rich ecosystem of modules that extend functionality
  - Modules handle various tasks such as authentication, URL rewriting, caching, load balancing, logging etc
    - mod_ssl: Adds support for SSL/TLS encryption.
    - mod_rewrite: Enables powerful URL rewriting and redirection
      - http://example.com/product/123 rewritten to http://example.com/product.php?id=123
    - mod_proxy: Provides support for proxying requests to other servers.
    - mod_auth: Handles authentication and access control

- Nginx: (pronounced "engine-x")
  - Configuration typically done via nginx.conf
  - Lightweight, high-performance web server
    - Event-driven architecture → handles concurrency very well
  - Also supports reverse proxying
  - Nginx often preferred for static content and as reverse proxy
  - Focuses on a small core feature set
    - Number of modules not as extensive as Apache's

- Microsoft Internet Information Services (IIS): Supports various web technologies and integrates well with other Microsoft products

IIS

Directory Structure (Linux)

- Specifics vary based on distribution and server software
- Web Server Root: Base directory from which web server serves files (user requested)
  - HTML, CSS , Javascript files are placed here
  - For Apache, the default root directory is often /var/www/html
  - For Nginx, the default root directory is /usr/share/nginx/html
- Server Configuration Files: Configuration files for a web server are usually found in /etc
  - Apache configuration files might be in /etc/apache2
  - Nginx configuration files might be in /etc/nginx

- Logs: access logs and error logs, are typically found in /var/log
  - For Apache, logs might be in /var/log/apache2
  - For Nginx, logs might be in /var/log/nginx
- Virtual Hosts (if applicable): One web server can host multiple websites
  - Configurations for these virtual hosts may be stored in a subdirectory of the server's configuration directory
    - E.g., /etc/apache2/sites-available/ for Apache

Sample Apache Configuration

```
# Global configuration
ServerRoot "/etc/apache2"

# Listen on port 80
Listen 80

# Server-wide defaults
<Directory />
            Options FollowSymLinks
            AllowOverride None
            Require all denied
</Directory>

<Directory /var/www/>
            Options Indexes FollowSymLinks
            AllowOverride None
            Require all granted
</Directory>

# Logging
ErrorLog ${APACHE_LOG_DIR}/error.log
LogLevel warn
CustomLog ${APACHE_LOG_DIR}/access.log combined

# Include module configurations
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf

# Include additional directory configurations
IncludeOptional conf-enabled/*.conf

# Virtual hosts
IncludeOptional sites-enabled/*.conf
```

Explanation

- ServerRoot Directive: top-level directory where server's configuration files and assets are located
  - Set to "/etc/apache2"
- Listen Directive: port on which Apache listens for incoming connections
  - Set to default port 80

- <Directory> Directives: Define settings for specific directories
  - First <Directory> block sets options for the root directory ("/")
  - Second block for the default document root ("/var/www/")
  - Options: Specifies the default behavior for the specified directory
    - "FollowSymLinks" option allows symbolic links (symlinks) to be followed
    - Indexes" option allows server to generate and display directory listings for directories that don't have an index file (e.g. index.html)

- AllowOverride: Which directives can be overridden using .htaccess files
  - All: Allows all directives to be overridden by .htaccess files
  - None: Don’t allow directives to be overridden by .htaccess files
- Require: Access control rules
  - all: It refers to all users or clients, effectively meaning "everyone."
    - Require ip 192.168.1.0/24: Restricts access to only clients within the specified IP range
  - access is denied by default for / and explicitly allowed for document root

- Logging Section: Configures error and access logs
  - ${APACHE_LOG_DIR} is a variable often set to var/log/apache2
    - Defined in /etc/apache2/envvars
  - ErrorLog: Specifies the path to the error log file
  - LogLevel: Sets the verbosity level
  - CustomLog: Defines the format and location of the access log
    - “Combined" is one of a predefined log format

- Include Directives: Includes additional files
  - The .load files typically contain LoadModule directives, which load Apache modules.
  - The .conf files typically contain configuration settings
    - Related to the loaded modules (under mods-enabled)
    - Or other configurations (conf-enabled or sites-enabled for virtual hosting etc)

Motivation: Dynamic Content

- Add new item to inventory → painful to manually create a new product page
  - Repeat lot of code across each page
  - Any change to page structure → changes to many pages!
- Good to have an automated way to read from database and create product page!
- Other examples:
  - Display score/marks specific to logged in student
  - Process form submissions at server and provide feedback to user

Dynamic Content

- Content dynamically generated on the fly in response to user request
- Example:
  - Receive a HTTP GET Request for a product
  - Server determines the product ID
  - Fetches data from database
  - Constructs the HTML page for the response by inserting the data into a HTML template
    - Any changes need to be done in one place, in a single template, and not across many static pages!

Technology

- Server-Side Scripting Languages: Executed on server to create dynamic content
  - **PHP**, Python (Django), Ruby (Ruby on Rails), Node.js etc
- **Templates**: Help separate logic from presentation
  - Servers embed dynamic content within HTML templates
- Databases: Dynamic content generated via data stored and retrieved from databases
  - MySQL, PostgreSQL, MongoDB etc
  - Server-side scripts use queries (e.g. SQL) to interact with databases

- Web Frameworks: Frameworks like **Flask**, Django, Ruby on Rails, and Express.js provide structured web application development
  - Often based on Model-View-Controller (MVC)
  - Separate application into
    - models (data and business logic)
    - views (presentation)
    - controllers (handling user input and interaction)

- **Web APIs** (Application Programming Interfaces): Dynamic content can be sourced from external APIs
  - Websites can pull data from third-party services or other platforms
  - RESTful APIs are commonly used
- Real-Time Technologies:
  - **WebSockets**: provide a full-duplex communication channel between server and client
  - Server-Sent Events (SSE): a mechanism for sending updates from the server to client

Server-side Scripting: PHP

- PHP stands for Hypertext Preprocessor
- Embedded within HTML code itself
  - Allows developers to seamlessly mix server-side logic with HTML markup
  - Executed on the server, and the resulting HTML is sent to client

Example

```php
<body>
        <h1>PHP Example</h1>

        <?php
        // Check if the 'name' parameter is present in the URL
        if (isset($_GET['name'])) {
        // Get the user's name from the 'name' parameter
        $userName = htmlspecialchars($_GET['name']);

        // Display a personalized greeting
        echo "<p>Welcome, {$userName}!</p>";
        } else {
        // Display a message if 'name' parameter is not provided
        echo "<p>Please provide your name in the URL.</p>";
        }
        ?>
</body>
```

http://www.example.com/welcome.php?name=Chotu

(question mark (?) indicates the beginning of a query string. Query string is a set of key-value pairs separated by ampersands (&))

- isset($_GET['name']): Checks if 'name' parameter is present in URL
- $_GET['name']: retrieve user's name
- htmlspecialchars() sanitizes the input
- Displays a personalized greeting using the user's name
- If 'name' parameter not provided, another message is shown

Workflow with PHP

1. **Client Request:** A user enters URL or clicks on a link
2. **Web Server Handling:** Web server (e.g., Apache or Nginx) receives the request
3. **PHP Processing:**
   - Requested resource is a PHP file → web server hands over request to PHP interpreter
   - PHP scripts often interact with databases (e.g., MySQL, PostgreSQL) to retrieve or update data.
   - PHP generates dynamic content, typically HTML, based on the logic in PHP script and any data retrieved from database
4. **Response to Client:**
   - Generated content is sent back as an HTTP response to user's web browser

In file /etc/apache2/apache2.conf include php related configuration

```
LoadModule php_module modules/libphp.so

AddHandler application/x-httpd-php .php
```

(LoadModule loads php module from given path and names it php_module

AddHandler associates given MIME type to files ending in .php

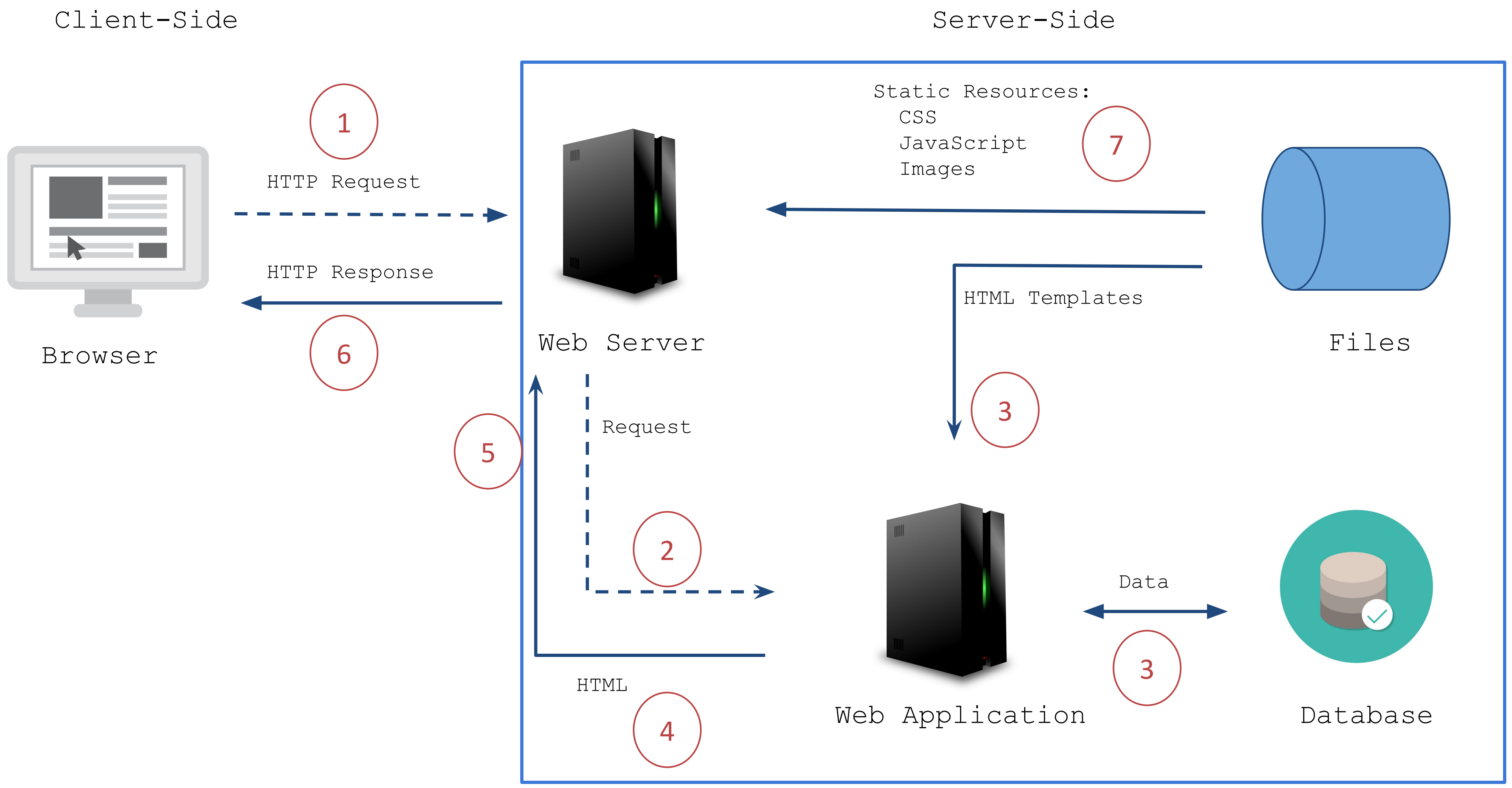
php_ module communicates with the PHP interpreter to execute the PHP script

PHP interpreter processes the script, executes the PHP code and generates dynamic content (HTML) and sends it back to php_module

module sends the HTTP response to client

Web Application Frameworks

- Software frameworks make it easier to write, maintain and scale web applications
  - Provide tons of tools and libraries that simplify common tasks
    - Routing URLs to appropriate handlers, template engines, interacting with databases, supporting sessions and user authorization, formatting output (e.g. HTML, JSON, XML), and improving security etc
- Examples of popular web frameworks
  - Django (Python), Ruby on Rails (Ruby), Flask (Python), Spring Boot (Java)


Client-Side
Server-Side
1
HTTP Request
HTTP Response
6
Browser
Static Resources:
CSS
JavaScript
Images
7
Web Server
HTML Templates
Files
3
Request
5
2
Data
3
HTML
4
Web Application
Database

Workflow

- Web browser sends a HTTP GET request
  - E.g. https://www.example.com/product/123 or https://www.example.com/product?id=123
  - Note: GET request is used because request is only fetching data
- Web Server detects request is "dynamic" and forwards it to Web Application for processing
  - How does it know where to forward?
    - Based on pattern matching rules defined in its configuration

- Web Application identifies intention of request is to get product information of product with id=123
  - Gets required information from the database
  - Dynamically creates a HTML page by putting retrieved data into placeholders inside a **HTML template**
- Returns generated HTML to web browser (via the Web Server), along with HTTP status code (200 for success)
  - Web Application may return another code is something fails
  - E.g. "404" to indicate product does not exist

- Web Browser processes returned HTML
  - Can send separate requests for other CSS or JavaScript files
  - Web Server will return such static files directly
    - Correct file handling based on configuration rules and URL pattern matching!
- Note: Server-side code does not have to return HTML files only, it can dynamically create and return other types of files
  - E.g. text, PDF, CSV, JSON, XML, etc

Example: Flask

```python
from flask import Flask, render_template, request

app = Flask(__name__)

# Route to the home page
@app.route('/')
def home():
    # Get the 'username' query parameter from the URL
    username = request.args.get('username', 'Guest')

    # Data to be passed to the template
    user = {'username': username}

    # Render the template 'index.html' with the provided data
    return render_template('index.html', user=user)

if __name__ == '__main__':
    app.run(debug=True)
```

- Flask application defines a route for the home page (`'/'`)
  - When a user accesses this page, the `home` function is called
- request object from Flask is used to access the query parameters in the URL
  - retrieves 'username' parameter, and if it's not present, it defaults to 'Guest'
- A dictionary (`user`) is created, and the `render_template` function is called to render 'index.html'
  - `user` data is passed to it
- In the template (`index.html`), Jinja2 syntax (`{{ user.username }}`) is used to insert dynamic data into the HTML content

(Jinja2 is a template engine for Python programming language)

Template (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>Flask Template Example</title>
</head>
<body>
        <h1>Hello, {{ user.username }}!</h1>
        <p>This is a simple example of using templates with Flask.</p>
</body>
</html>
```

Web APIs

- API: Application Programming Interface
  - Set of rules and protocols that allows one application to interact with another
- Web API?
  - Enables communication and data exchange between web servers and also client-server
  - Typically use HTTP (Hypertext Transfer Protocol) as the communication protocol

- Many web APIs follow REST principles
  - RESTful (Representational State Transfer) APIs
  - **Stateless**, scalable, often idempotent and adhere to standard HTTP methods (GET, POST, PUT, DELETE) for performing operations on resources
    - Stateless: each request from a client contains all info needed for server to respond
      - No client state stored between requests.

- Web APIs expose specific URLs (endpoints) to which clients send HTTP requests
  - Each endpoint represents a specific resource or functionality provided by the API
- Web APIs exchange data in standardized formats, such as JSON (JavaScript Object Notation) or XML (eXtensible Markup Language)

Example

Endpoint: http://api.example.com/users (This URL serves as the base for all user-related operations in the API)

```
GET /users HTTP/1.1
Host: api.example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
{
"id": 1,
"username": "john_doe",
"email": "john.doe@example.com"
},
{
"id": 2,
"username": "jane_smith",
"email": "jane.smith@example.com"
}
]
```

```
GET /users/1 HTTP/1.1
Host: api.example.com

Response:

    HTTP/1.1 200 OK
    Content-Type: application/json

    {
    "id": 1,
    "username": "john_doe",
    "email": "john.doe@example.com"
    }
```

```
POST /users HTTP/1.1
Host: api.example.com
Content-Type: application/json

{
  "username": "new_user",
  "email": "new.user@example.com"
}
```

Response

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": 3,
  "username": "new_user",
  "email": "new.user@example.com"
}
```

- For security,
  - Web APIs often require authentication
    - Clients must provide valid credentials (e.g., API keys, tokens) in the requests
  - Authorization also important
    - What actions or resources a particular user or application can access?
    - Ouath2.0 tokens and JWTs contain relevant info (user roles/permissions)

  - Cross-Origin Resource Sharing (CORS) help web pages hosted on different domains to access web APIs
    - Relevant CORS headers specify which domains are allowed to make requests!
    - More in depth coverage later under attacks!
- Popular third-party web APIs: Twitter API, Google Maps API, GitHub API
  - Developers use these APIs to integrate third-party services into their applications

- As an aside: there are Browser APIs also
  - Extend functionality of the browser
    - E.g. Geolocation API returns coordinates of where browser is located
    - Web form API for input data validation
    - Web storage api for storing and retrieving data within browser
    - Web fetch api (saw as part of AJAX)s

Data

- Exchange of data between a client and a server can occur in various formats
  - Already saw: html, css, js files; images in form of png, jpg; pdfs, docs etc
- APIs often involve transfer of structured data
  - JSON and XML are commonly used
  - Earlier in web's history, XML was in vogue, now JSON is more used!

JavaScript Object Notation (JSON)

- A lightweight data interchange format
  - Easy for humans to read/write
  - Also for machines to parse and generate
- Data represented as key-value pairs
  - Supports data types such as strings, numbers, booleans, arrays, and objects
  - Data is enclosed in curly braces {}
  - Key-value pairs are separated by colons
  - Arrays are represented using square brackets []
  - Case-sensitive and whitespace-insensitive

```
{
  "name": "Ravi",
  "age": 22,
  "isStudent": true,
  "grades": [90, 85, 92],
  "address": {
       "city": "Mumbai",
       "zipcode": "400076"
  }
}
```

- MIME type: application/json
- Limitations:
  - no support for comments
  - no native support for binary data
  - Inability to represent circular references in objects
- More advanced features needed: use XML, Protocol Buffers, or GraphQL

Web Sockets

- WebSockets provide a full-duplex communication channel over a single, long-lived connection
  - Full-Duplex Communication: data can be sent in both directions simultaneously
    - Traditional HTTP follows a half-duplex model; client sends request, server responds!
  - Persistent Connection: connection remains open for as long as both client and server want

- WebSocket Protocol operates over a single, dedicated TCP connection.
  - Need to use ws:// and wss://
- Suitable for real-Time Applications
  - E.g. chat, online gaming, financial trading, live updates etc
- More details later under attacks!

Summary as History

- Static Web Pages (1990s): No interactivity and dynamic content
- JavaScript (1995): interactivity within browser
- AJAX (early 2000s): use JavaScript to make asynchronous requests
  - For more responsive and interactive web applications
- Server-Side Technologies (PHP, ASP, JSP, early 2000s): server-side scripting languages
  - Embed server-side logic into web pages to generate dynamic content

- Web Frameworks and MVC Architecture (mid-2000s): Model-View-Controller (MVC) style architectural patterns
  - Organize code, separating concerns and promoting development of dynamic and maintainable web applications
- Web 2.0 and Rich Internet Applications (RIAs, mid-2000s): shift towards more user-centric and interactive web experiences
  - Based on Adobe Flash, advanced JavaScript libraries (e.g., jQuery)

- Single Page Applications (SPAs, 2010s): load a single HTML page and dynamically update content
  - Based on Angular, React, and Vue.js etc
- WebAssembly (Wasm, 2015s): a binary instruction format that enables high-performance execution of code (C, C++, Rust) on web browsers
- Progressive Web Apps (PWAs, current): Use modern web capabilities to deliver an app-like experience across different devices
  - Offline support, push notifications, and improved interactivity and performance

References

- https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction
- https://www.w3schools.com/php/
- https://www.geeksforgeeks.org/flask-tutorial/ and https://www.geeksforgeeks.org/getting-started-with-jinja-template/
- https://www.tutorialspoint.com/restful/index.htm (REST api)
- https://www.tutorialspoint.com/websockets/index.htm
- https://www.w3schools.com/js/js_json_intro.asp (json)