MAD 1 WEEK 8

Lecture 8.1: Application Frontend

What is the Frontend?

The frontend is the part of the application that users interact with directly. Its key responsibilities include:

- **Rendering the User Interface (UI)**: Displaying visuals and layouts that users see.
- **Handling User Input**: Managing interactions like clicks, typing, and navigation.
- **Communicating with the Backend**: Sending and receiving data from the server to ensure functionality.

Types of User-Facing Interfaces

1. **General GUI Applications**: Traditional desktop-based graphical interfaces.
2. **Browser-Based Clients**: Interfaces accessed through web browsers.
3. **Custom Embedded Interfaces**: Unique systems designed for specific hardware or devices.

Web Applications

Components of Web Frontend Development

1. **HTML (HyperText Markup Language)**:
   - Defines the structure of web pages.
   - Example: Headings, paragraphs, tables.
2. **CSS (Cascading Style Sheets)**:
   - Styles the webpage (colors, layouts, fonts).
3. **JavaScript**:
   - Adds interactivity and dynamic behavior to web pages.

Types of Web Pages

1. Fully Static Pages

- Pages are generated beforehand and do not change dynamically.
- In contrast, static web pages are usually simple HTML files that do not change or are not generated dynamically by the server. They are served as-is without any server-side processing.

- **Advantages**:
  - High performance due to precompiled content.
  - Suitable for low-complexity websites.
- **Disadvantages**:
  - Limited flexibility for personalized or user-specific content.
- **Tools for Static Site Generation**:
  - Jekyll
  - Hugo
  - Gatsby

2. Run-Time HTML Generation

- **Description**: HTML is generated dynamically when a user accesses the page.
- **Technologies**:
  - Python (e.g., Flask, Django)
  - Ruby (Rails)
  - PHP (WordPress, Drupal)

Advantages:

- Extremely flexible, suitable for applications requiring personalized content.

Disadvantages:

- Higher server load due to dynamic generation.
- Potential database hits per request.
- Performance optimization needed (e.g., caching).

How Web Browsers Handle the Load

Workflow of a Typical Browser:

1. **Issuing Requests**:
   - Sends HTTP requests to servers.
2. **Receiving Responses**:
   - Retrieves data and files (HTML, CSS, JavaScript).

```
@app.route(‘/home’)
Def home:
student=Stdent.query.get
print(student)
```

Return home.html

3. **Rendering Content**:
   - Converts data into the visual UI.
4. **Waiting for User Input**:
   - Responds to interactions like clicks or form submissions.

Optimization Techniques

- Caching: Storing previously loaded data for faster access.
- Content Delivery Networks (CDNs): Distributing content geographically to reduce latency.

Lecture 8.2: Asynchronous Updates

What are Asynchronous Updates?

Asynchronous updates allow parts of a webpage to be updated dynamically without reloading the entire page. This improves user experience (UX) by:

- **Updating Only Specific Sections**: Refreshes parts of the page while leaving the rest unchanged.
- **Loading Extra Data in the Background**: Retrieves and processes data after the main page has already loaded and rendered.
- **Providing a Quick and Responsive Interface**: Enables seamless interactions and real-time updates.

Examples of Asynchronous Updates

1. **Facebook**:
   - Notifications, messages, and friend requests are updated without refreshing the page.
2. **Twitter**:
   - Displays new tweets and notifications in real time.
3. **GitHub**:

    - Commits, pull requests, and issues are updated dynamically without a full page reload.

Technologies for Asynchronous Updates

1. AJAX (Asynchronous JavaScript and XML)

- A technique for making asynchronous web requests.
- Allows the browser to fetch data from the server and update the page without a full reload.

2. WebSockets

- Provides full-duplex communication channels over a TCP connection.
- Ideal for real-time applications (e.g., chat apps, live notifications).

Document Object Model (DOM)

What is the DOM?

- A tree structure that represents the logical layout of a document (e.g., HTML or XML).
- Allows direct manipulation of page content and structure.

Features of DOM Manipulation:

1. **API Interaction**:
    - DOM provides methods like querySelectorAll for easy access to document elements.
2. **Styling with CSS**:
    - Use CSS for the visual appearance of elements.
3. **JavaScript Integration**:
    - JavaScript is the primary tool for interacting with and manipulating the DOM.

Example of DOM Manipulation

javascript

```
const paragraphs = document.querySelectorAll("p");
// paragraphs[0] is the first <p> element
// paragraphs[1] is the second <p> element, etc.
alert(paragraphs[0].nodeName);
```

In this example:

- The querySelectorAll method selects all <p> elements in the document.
- The alert method displays the name of the first <p> element.

Additional Concepts

Canvas:

- A powerful tool for drawing graphics or creating animations directly on a webpage.

Offline Web Storage:

- Technologies like localStorage and sessionStorage allow storing data locally on the client’s browser.

Drag and Drop:

- Enables users to interact with elements by dragging and dropping them within the browser.

L8.3: Browser/Client Operations

Minimal Requirements

1. **Basic Hardware and OS**:
   - A functional device with an operating system.
2. **Network Connectivity**:
   - Internet access to load web content.
3. **Compatible Browser**:
   - A modern browser or client application.

Text-mode and Accessibility

- **Text-mode Displays**: Render web content primarily as text.
- **Accessibility**: Enhances usability for screen readers and users with disabilities.
- **Compatibility**: Works seamlessly across various devices.

Page Styling

- Achieved using **CSS (Cascading Style Sheets)**:
  - Customizes fonts, colors, layout, and design elements.
  - Improves the visual appeal and user experience of web pages.

Interactivity

- Adds user engagement through scripting languages like **JavaScript**.
- Enables dynamic responses to user actions and inputs.

JavaScript Engines

- **Function**: Interpret and execute JavaScript code.
- **Purpose**:
  - Converts JavaScript into machine code for performance.
  - Powers interactivity and dynamic features in web pages.

Client Load

- Relates to the computational capabilities of the client device.
- Influences the performance and speed of web pages.
- Varies based on the hardware and processing power of the device.

Machine Clients

- Include **personal computers** and **laptops**:
  - Provide higher computational capacity and memory.
  - Deliver better performance compared to mobile devices.

Alternative Scripting Languages

1. **TypeScript**, **CoffeeScript**, **Dart**: Provide features beyond JavaScript.
2. **Brython** and **PyScript**: Enable writing Python code for the browser.
3. **Challenges**:
   - Limited cross-browser compatibility.
   - Smaller community support compared to JavaScript.
   - Require extra compilation steps.

WASM (WebAssembly)

- **Purpose**: High-performance execution of non-JavaScript code in browsers.
- **Applications**:
  - Handles computationally intensive tasks.
  - Executes near-native speed applications.

Enscripten

- Converts **C/C++** code into WebAssembly.
- Enables compatibility with web browsers.
- Delivers high-performance applications with cross-platform support.

Native Mode

- Directly accesses hardware-specific functionalities.
- Leverages device capabilities for better performance.
- Seamlessly integrates with operating systems and hardware.

L8.4: Client-side Computations and Security Implications

Validation

Frontend Validation

- **Immediate Feedback**: Validates user input with JavaScript before submission.
- **Real-Time Correction**: Reduces invalid submissions, saving server resources.
- **User-Friendly**: Displays contextual error messages near form fields.

Backend Validation

- **Data Integrity**: Ensures accuracy of submitted data.
- **Security**: Protects against malicious inputs.
- **Business Logic**: Enforces application rules and consistency.

Example: Frontend Validation

HTML

html

```html
<form>
  <label for="mail">Email</label>
  <input type="email" id="mail" name="mail" required />
</form>
```

JavaScript

Javascript
function(var i=0;i++;i<10)

```javascript
const email = document.getElementById("mail");
email.addEventListener("input", function (event) {
  if (email.validity.typeMismatch) {
    email.setCustomValidity("I expect an e-mail, buddy!");
  } else {
    email.setCustomValidity("");
  }
});
```

More Examples:

1. Change Text Content

Updates the text content of an element.

HTML:

```
<div id="demo">Original Text</div>
<button onclick="changeText()">Change Text</button>
```

JavaScript:

```
function changeText() {
  const element = document.getElementById("demo");
  element.textContent = "Text has been changed!";
}
```

Example in vs code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="demo">Original Text</div>
<button onclick="changeText()">Change Text</button>
<script>
    function changeText() {
  const element = document.getElementById("demo");
  element.textContent = "Text has been changed!";
}
</script>
</body>
```

```
</html>
```

---

2. Change Background Color

Changes the background color of a page or element.

HTML:

```
<button onclick="changeColor()">Change Background Color</button>
```

JavaScript:

```
function changeColor() {
  document.body.style.backgroundColor = "lightblue";
}
```

---

3. Show/Hide an Element

Toggles the visibility of an element.

HTML:

```
<div id="message">Hello, I am visible!</div>
<button onclick="toggleVisibility()">Show/Hide</button>
```

JavaScript:

```
function toggleVisibility() {
  const element = document.getElementById("message");
  element.style.display = element.style.display === "none" ?
"block" : "none";
}
```

Captcha

- Verifies users against bots.
- Prevents automated attacks but may raise privacy concerns due to third-party dependencies.

Crypto-mining

- Client-side computation for cryptocurrency mining.
- Sends results back to the server via asynchronous calls.

Sandboxing

1. **Isolated Environment**: Restricts web applications from accessing sensitive resources.
2. **Limited Privileges**: Reduces risks of malicious activity.
3. **Malware Protection**: Prevents unauthorized code execution.
4. **Enhanced Safety**: Runs untrusted code securely.

Overload and Denial of Service (DoS)

- **Overload**: Excessive requests slow or crash the server.
- **DoS/DDoS**: Malicious flooding of requests disrupts services.
- **Mitigation**: Use rate limiting, traffic filtering, CDNs, and load balancers.

Access Native Resources

- **Risks**: Unauthorized access to the file system, camera, or microphone.
- **Solution**: Browsers sandbox applications to isolate them, minimizing risks.

A server has a 16-core CPU, 64 GB RAM and 1 Gbps network connection. It can run a Python Flask application that can generate 500 HTML pages per second. Each page also has a 1 MB image that needs to be downloaded by the client. What will be the maximum number of requests per second that the server can handle?
500/s
1gbps

1Gb=1000/8=125MB/s
125/1=125

1Gb/S=1000Mb=1000/8=125MB/s

1MB

500 pages —>500MB
1 page —> 1MB
?pages —>125MB?

125 pages

b=bit
B=Byte

1byte=8bits

A server has a 16-core CPU, 64 GB RAM and 2 Gbps network connection. It can run a Python Flask application that can generate 250 HTML pages per second. Each page also has a 500 KB image that needs to be downloaded by the client. What will be the maximum number of requests per second that the server can handle?

2Gb/s =2000/8=250MB
250X500KB=125000=125MB
1page—-->500KB

?pages—>250MB

250MBX1000KB=250000KB/500KB=500 Pages

[Here we are taking the total mb and converting to kb then dividing by the amount of kb per page]

A server has a 16-core CPU, 64 GB RAM and 1 Gbps network connection. It can run a Python Flask application that can generate 125 HTML pages per second. Each page also has a 500 KB image that needs to be downloaded by the client. What will be the maximum number of requests per second that the server can handle?

1 gbps=1000/8=125MB
125 X 500KB=62500KB

1 Page—>500KB
?pages—>125MB
125MBX1000KB=125000KB/500KB=250Pages