# Week 8: Application Frontend

## L8.1: *Application Frontends*

### Application Frontend

- The frontend is the part of the application that the user interacts with.
- The frontend is responsible for:
  - Rendering the UI
  - Handling user input
  - Communicating with the backend
- User-facing Interface can be:
  - General GUI application on Desktop
  - Browser based client
  - Custom embedded interface

### Web Applications

- Browser based: HTML, CSS, JavaScript
  - **HTML** is used to define the structure of the page
  - **CSS** is used to define the style of the page
  - **JavaScript** is used to define the behavior of the page

#### Fully static pages

- A lot of pages on the site are statically generated
  - Compiled ahead of time
  - Not generated at run-time
- Excellent for high performance
- Not very flexible
  - JavaScript can be used to add some dynamism
- Some static site generators are:
  - Jekyll
  - Hugo
  - Gatsby

#### Run-time HTML generation

- Traditional web applications based on CGI/WSGI.
  - Python (Flask, Django), Ruby (Rails), PHP, etc.
  - PHPs core concept is to generate HTML at run-time

    - Wordpress, Drupal, etc. are traditional CMS applications
- The server generates the HTML at run-time
    - The server is responsible for generating & serving the HTML.
- Very flexible
- Server needs to be able to handle the load
    - Every page has to be generated dynamically
    - May involve database hits
    - Cost, Speed
- Caching and other techniques can be used to improve performance

## Client Load

- A typical web browser does:
    - issue requests, wait for response
    - render HTML, CSS, JavaScript
    - wait for user input

Why not let client do more work?

- Client-side scripting
    - JavaScript can be used here
    - Component frameworks like *React, Vue, Angular* allow reuse, complex Interactions.

| Aspect | Server-Side Rendering (SSR) | Static Webpages | Client-Side Rendering (CSR) |
|---|---|---|---|
| ***Initial Load Time*** | Faster initial load time due to pre-rendered content. | Fastest initial load time as pages are pre-built. | Slower initial load time as content is generated on the client. |
| ***SEO Friendliness*** | Highly SEO-friendly as search engines can crawl pre-rendered content. | SEO-friendly as content is available directly in HTML. | Less SEO-friendly as search engines may have difficulty indexing dynamic content. |
| ***User Interaction*** | Limited interactivity until full page is loaded. | Limited interactivity until full page is loaded. | Interactive user experience as content is rendered on the fly. |
| ***Server Load*** | Moderate server load due to pre-rendering for each request. | Minimal server load as pages are served as static files. | Reduced server load, but more client-side processing required. |
| ***Complexity*** | More complex server-side code and server configuration. | Less complex server-side code and configuration. | More complex client-side code and handling of state management. |

| Aspect | Server-Side Rendering (SSR) | Static Webpages | Client-Side Rendering (CSR) |
|---|---|---|---|
| ***Offline Accessibility*** | Partial offline accessibility with pre-rendered content. | Full offline accessibility as pages are static. | Limited offline accessibility as dynamic content requires server access. |
| ***Real-time Updates*** | Limited real-time updates without additional logic. | No real-time updates without regenerating pages. | Real-time updates can be achieved through AJAX or WebSockets. |
| ***Development Flexibility*** | Requires back-end development for rendering and API integration. | Limited flexibility, especially for frequently changing content. | Offers more flexibility in terms of dynamic content and rich interactions. |
| ***Mobile Performance*** | Depends on the amount of client-side rendering required. | Excellent mobile performance due to static pages. | Depends on the complexity of client-side code and data handling. |
| ***Caching*** | Limited caching potential due to dynamic content generation. | Highly cacheable as static pages can be served from CDNs. | Caching requires careful management to ensure up-to-date content. |
| ***Security*** | Server-side rendering provides better security control as data processing is done on the server. | Static pages offer improved security as there's no dynamic server-side interaction. | Client-side rendering may expose certain logic to the client, requiring additional security measures. |
| ***Accessibility*** | SSR allows for better accessibility compliance through server-generated content. | Static pages can be made accessible and comply with accessibility guidelines. | CSR requires careful consideration to ensure accessibility for dynamic content. |
| ***Development Complexity*** | More complex development process due to combining client and server logic. | Simpler development process, suitable for static content and simpler projects. | Complex development process due to handling both client-side rendering and data management. |
| ***Scalability*** | SSR may require more server resources for handling rendering requests. | Static pages can be easily distributed and scaled through CDNs. | CSR requires careful optimization to maintain performance at scale. |
| ***Maintenance*** | More maintenance needed for server-side rendering logic and infrastructure. | Low maintenance as static pages need infrequent updates. | Maintenance required for client-side logic, especially when new features are added. |

| Aspect | Server-Side Rendering (SSR) | Static Webpages | Client-Side Rendering (CSR) |
|---|---|---|---|
| ***Best Use Cases*** | Suitable for content-rich websites, e-commerce platforms with dynamic content, and complex user interfaces. | Ideal for blogs, informational websites, and websites with less dynamic content. | Great for web applications and interactive interfaces that require real-time updates and user interactions. |

# L8.2: *Asynchronous Updates*

## Asynchronous Updates

- Updating only the part of the page
  - Load extra data in the background after the main page has been loaded and rendered.
- Quick response on main page, for better UX.
- Example:
  - **Facebook**: Updates notifications, messages and friend requests without refreshing the page.
  - **Twitter**: Dsipays new tweets and notifications in real time
  - **Github**: Updates commits, pull requests and issues without refreshing the page.
- Can be achieved using:
  - **AJAX**: Asynchronous JavaScript and XML
  - **WebSockets**: Full-duplex communication channels over TCP connection.

### DOM

- Tree structure representing logical layout of document
- *Direct manipulation of tree possible*
- **Application Programming Interface** (API)
  - *Canvas*
  - *Offline*
  - *Web Storage*
  - *Drag and Drop* & ...
- JavaScript primary means of manipulating
- CSS used for styling
- Example: *This DOM specifies that the* `querySelectorAll` *method in this code snippet must return a list of all the* `<p>` *elements in the document*

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

- We can do a lot of DOM manipulation using JavaScript.
- Check here

---

# L8.3: *Browser/Client operations*

## Minimal Requirements

- Basic hardware components and an operating system.
- Network connectivity to access the internet.
- A compatible web browser or client application.

## Text-mode and Accessibility

- Text-mode displays web content primarily in text form.
- Enhances accessibility for screen readers and users with disabilities.
- Ensures wider compatibility and ease of use for different devices.

## Page Styling

- Page styling is achieved using CSS (Cascading Style Sheets).
- Customizes fonts, colors, layout, and design elements.
- Enhances the visual appeal and user experience of web pages.

## Interactivity

- Interactivity allows user engagement with web pages.
- Achieved through scripting languages like JavaScript.
- Provides dynamic responses to user actions and input.

## JavaScript Engines

- JavaScript engines interpret and execute JavaScript code.
- Convert JavaScript instructions to machine code for performance.
- Power interactivity and dynamic functionalities in web pages.

## Client Loads

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

## Machine Clients

- Machine clients include personal computers and laptops.
- Possess higher computational capacity and memory.
- Offer better performance compared to mobile devices.

## Alternative Scripting Languages

- TypeScript, CoffeeScript, and Dart are alternatives to JavaScript.
- Brython allows writing Python code that runs in the browser.
- Pyscript enables using Python as a scripting language for web development.
- They offer unique syntax and features for creating interactive web applications.

### Problems with Alternatives

- Alternative scripting languages may lack cross-browser compatibility.
- Limited community support and resources compared to JavaScript.
- Some alternatives require additional compilation steps, hindering development speed.

## WASM (WebAssembly)

- WebAssembly is a binary instruction format for web browsers.
- Enables high-performance execution of code written in other languages.
- Provides an alternative to JavaScript for certain computationally intensive tasks.

### Enscripten

- Translates C/C++ code to WebAssembly for browser compatibility.
- Enables high-performance web applications with near-native speed.
- Provides portability and cross-platform support for C/C++ applications on web browsers.

## Native Mode

- Native mode allows web browsers to access hardware-specific functionalities.
- Enhances performance by leveraging device capabilities directly.
- Enables seamless integration with the operating system and hardware resources.

---

# L8.4: *Client side computations and security implications*

## Validation

### Frontend Validation:

- **Immediate Feedback**: Uses JavaScript to validate user input before submission.
- **Real-Time Correction**: Prevents invalid data submission, reducing server requests.

- **User-Friendly**: Displays error messages near relevant form fields.

### Backend Validation:

- **Data Integrity**: Validates data on the server-side to maintain accuracy.
- **Security**: Guards against malicious input and prevents unauthorized actions.
- **Business Logic**: Enforces application rules and consistency.

## Front-end form validation 🔗

- `required` : Specifies whether a form field needs to be filled in before the form can be submitted.
- `minlength` and `maxlength` : Specifies the minimum and maximum length of textual data (strings).
- `min` and `max` : Specifies the minimum and maximum values of numerical input types.
- `type` : Specifies whether the data needs to be a number, an email address, or some other specific preset type.

## JavaScript Validation 🔗

- We can write script in javascript to do validation.
- Example - **Checking for a valid email address**
  - HTML

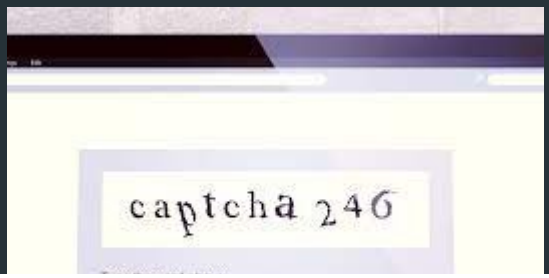
```
<form>
    <label for="mail">Email</label>
    <input type="email" id="mail" name="mail" required />
</form>
```

  - JavaScript

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

# Captcha

- Captcha involves client-side computation for user verification.
- It enhances security by preventing automated bot attacks.
- However, Captcha raises privacy concerns as it requires interaction with third-party services.




## Crypto-mining

- Crypto-mining involves client-side computation for cryptocurrency mining.
- Can implement crypto-mining in web applications using JavaScript.
- Script will send results back to server through async calls

## Sandboxing

1. **Isolated Environment:** Sandboxing creates a restricted environment for web applications, isolating them from the underlying operating system and other applications.
2. **Limited Privileges:** It restricts the application's access to sensitive resources and functionalities, reducing the potential impact of security breaches or malicious activities.
3. **Preventing Unauthorized Access:** Sandboxing helps prevent unauthorized access to user data and system resources, minimizing the risk of data breaches.
4. **Malware Protection:** It acts as a shield against malware and malicious code by preventing unauthorized code execution outside the sandboxed environment.
5. **Enhanced User Safety:** By confining web applications to a sandbox, users can safely run untrusted code without compromising the overall system's security and stability.

## Overload and DoS (Denial of Service)

- **Overload:** Occurs when a web server is unable to handle an excessive number of requests, leading to slow response times or system crashes.
- **DoS (Denial of Service):** Involves deliberate flooding of requests to overwhelm the server and disrupt its normal operation, denying service to legitimate users.
- **Distributed DoS (DDoS):** Multiple sources are used to generate the flood of requests, making it harder to mitigate and identify the attackers.
- **Attack Vectors:** Overload and DoS attacks exploit vulnerabilities in server configurations, resource limitations, or software weaknesses.
- **Mitigation:** Implementing rate limiting, request validation, and traffic filtering can help prevent and mitigate the impact of overload and DoS attacks. Additionally, Content Delivery Networks (CDNs) and load balancers can distribute incoming requests to handle traffic spikes. Regular security testing and monitoring

are essential to detect and respond to such attacks promptly.

## Access Native Resources

- **Security Risks:** Granting web applications access to native resources like the file system, camera, or microphone can lead to unauthorized data access or device control.
- **Sandbox Environment:** Modern browsers use sandboxing techniques to restrict web applications' access to native resources. This containment minimizes potential security risks by isolating the application from the underlying system.

---