CoGrammar

Welcome to this session:
Fetching and Displaying From APIs

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators answering questions.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Skills Bootcamp Cloud Web Development

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Skills Bootcamp Cloud Web Development

- For all **non-academic questions**, please submit a query: ***www.hyperiondev.com/support***
- **Report a safeguarding incident: *www.hyperiondev.com/safeguardreporting***
- We would love your feedback on lectures: *Feedback on Lectures*
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

What is a client side application?

A. A subscription based application
B. An application that runs code directly on a users system
C. Any application that is built for a customer
D. None of the above

What can we build using the DOM that we can't building using HTML?

A. Websites with fixed content
B. Websites with multiple pages
C. Responsive websites
D. Websites with dynamic features

Learning Outcomes

- Identify the role of static and dynamic websites
- Distinguish between server-side rendering and client-side rendering
- Discuss the issues faced when implementing client side rendering
- Identify the relationship between client side applications and APIs and the importance of the DOM in this relationship.

Lecture Overview

- ➔ Static vs Dynamic Website
- ➔ Client-Side Rendering (CSR) vs Server-Side Rendering (SSR)
- ➔ Client-Server Architecture
- ➔ Designing an API Based Application
- ➔ Optimising API and Client Communication

Static and Dynamic Website

- **Static** and **dynamic** pages refer to how the HTML content on a webpage can change during a users session.
- A **static** page will display the same content every time the page is loaded.
- The content on a **dynamic** page on the other hand will change based on certain circumstances
  - A user action
  - A specific user accessing the page
  - Changes in the database/API
  - etc

Why do we need Dynamic Sites

- Dynamic websites are important in modern web development as they allow us for:
  - Customized user interactions
  - Remote changes of HTML content
  - Reducing repetition when building HTML
- Dynamic websites allow us to build **web application** that can:
  - Perform complex business operations without the need to install native applications.
  - Connect to external data sources, and display changing data in near real-time

Rendering Dynamic Content

- To make a website dynamic, we need to be able to take programming logic and convert it into HTML allowing us to:
  - Adding or removing HTML elements to a page
  - Changing the content in an HTML element
  - Performing operations based on HTML events.
  - Etc.
- When using programming logic to update the HTML, we're able to make use of programming concepts like conditions, loops and much more to:
  - Conditionally render content
  - Perform near real-time content updates
  - Respond to user actions
  - etc

Client Side and Server Side Rendering

- Dynamic websites usually consist of complex logic that might require them to :
  - Run complex operations (algorithms)
  - Connect to external services and consolidate results.
  - Etc
- The results of these operations will need to be translated into something that can be displayed on the browser.
- There are two main approaches to how we can handle these operations and display content in the most effective manner based on a specific use-case
  - **Server Side Rendering (SSR)**
  - **Client Side Rendering (CSR)**

Server-Side Rendering

- JavaScript is the only programming language that can run on a browser, but it's not always the best language for specific use cases
- **Server-Side Rendering (SSR)** allows us to write code in another language (or with JavaScript through Node.js) and generate the HTML that will be sent to the user for display
- SSR is useful in situations where you want to minimise the operations that would be performed on a client due to:
  - Performance - SSR reduces the load on a users browser
  - Security - The client only gets the HTML content and has no access to backend logic
  - Etc

Client-Side Rendering

- The major drawback of **SSR** is the dependence on a backend server to perform all of the operations.
- When traffic is high, the service might get slower as many people would need their content rendered by the same server at once
- **Client Side Rendering (CSR)** will perform any changes in the UI on the clients browser
- To perform client side rendering, JavaScript needs to be enabled on the browser in order for changes to be made
- By using the **DOM**, we can make changes to the HTML content without having to convert to generate the HTML like SSR.

Client-Side Rendering

- The major drawback of **SSR** is the dependence on a backend server to perform all of the operations.
- When traffic is high, the service might get slower as many people would need their content rendered by the same server at once
- **Client Side Rendering (CSR)** will perform any changes in the UI on the clients browser
- To perform client side rendering, JavaScript needs to be enabled on the browser in order for changes to be made
- By using the **DOM**, we can make changes to the HTML content without having to convert to generate the HTML like SSR.

Client-Side Rendering: Issues

- CSR is the easiest approach to implement, but comes with it's own challenges.
  - The more code and content, the slower the initial load time
  - All code and credentials required for performing operations needs to be sent to the client making it viewable by the user
  - Application performance is dependent on an individual users hardware resources

Client-Server Architecture

- To get the best parts of SSR and CSR, we can make use of the **Client-Server** architecture
- The Client-Server architecture allows us to perform all of our **backend** operations on a dedicated server where we can:
  - Write code in any programming language
  - Perform complex operations
  - Connect to confidential resources without the users knowledge
- All of the **frontend** operations will be performed on the client, allowing us to:
  - Display output from the server
  - Respond to user actions

Client-Server Architecture: Implementation

- **APIs** can be used to facilitate the communication between the client and the server.
- The client application will be able to:
  - Send data from client side operations to the API
  - Make requests for data from the API
  - Display the output of responses in the browser

Let's take a break

CoGrammar

Designing and Application Around an API

- In situations where a frontend application is being developed around an existing API, it's important to plan implementation.
- There are a few things that you can consider when looking at the API
  - Which API endpoints are important for the application?
  - For each specific operation, understand the format that data will be sent
  - For each specific operation, understand how the output is formatted
  - For each specific operations, identify the important attributes that need to be extracted
- It's always a good idea to use a tool like **Postman** to gain an understanding of the API before jumping into the actual code.

Designing and Application Around an API

- Once you know how data is returned and how it's passed to the API, you can consider how the user interface will work Eg.
  - For **Get** requests, if the output is a list of values, are there certain elements that can be used to display the content
  - For **Post** requests, should there be a text based input, or would it make more sense to have multiple choice, or a combination of both
- Having a reference to the desired final outcome will make it easier to know how your JavaScript code will operate, Eg.
  - Having a list of content based on API responses would require you to create entire elements in the DOM
  - Having custom messages would most likely require only a change in the content of an existing element.

Optimising API Queries

- When working with public APIs, there are certain limitations that we might come across, including:
  - Rate limits
  - Pay per request
- Custom APIs also have their own drawbacks, like:
  - Excessive volume
- It's important to reduce the amount of API calls that we make to:
  - Reduce the strain on a single server during high usage
  - Reduce the cost of making API calls
  - Stay within rate limits.

Optimising API Queries

- **Caching** responses is one of the best ways to reduce the calls that are made to the API
- To cache responses, we need to store their response locally, and everytime we get to an operation that needs specific data, we can reference the cached response
- The simplest approaches for caching responses are:
  - **LocalStorage**
  - **SessionStorage**

Optimising API Queries

- **Preloading** is another approach that can be used to reduce the number of calls that are made to the API
- With this approach, you can request more information from the API than you currently need, and store the excess data locally and reference it when and if it's needed.
- After the initial load, this approach will also make the application seem more responsive on the users side as they won't have to wait for API responses for more information to show.

Deploying a Client-Side Application

- When deploying a CSR application, it's important to remember that all of the HTML, CSS and JavaScript code needs to be sent to the client in order to load the site.
- To minimise the initial load time, it's a good idea to always bundle your site to make the content that is sent much smaller.
- Tools like **Vite**, which can be used to create React applications, will have their own bundlers that can handle this.
- When working with vanilla HTML and JS, we can make use of packages like:
  - **Webpack**
  - **ESBuild**

Deploying a Client-Side Application

- Bundling the website won't only reduce the size of the package, but it can also act as a layer of protection to hide certain details
- Since most client side applications connect to APIs that require API keys, the bundled code will make secrets less accessible

Which of the following is **NOT** a benefit of Server-side rendering?

A. Faster loading times on the users browser
B. Ability to use more programming languages
C. Easier to secure secrets
D. Can make direct changes to the DOM

CoGrammar

What is the benefit of using Client-Server?

A. All of the logic will be stored on the client application, making the application faster
B. The server application handles all of the complex operations and the client application focuses on the user interaction.
C. The server renders all of the HTML reducing the load on the client.
D. None of the above

Questions and Answers

CoGrammar

Thank you for attending

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education