TASK

Introduction to Full Stack Web Development

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO FULL STACK WEB DEVELOPMENT TASK!

Welcome to your first task at this level of the Bootcamp. Congratulations on making it to this point! In the previous level of the Bootcamp, you got to grips with the core languages used for front-end web development. In this level, you will start working with some of the most exciting and popular libraries and frameworks for full-stack JavaScript web development. By the end of this level, you will be able to code front-end and back-end applications using some of the most popular tools in the industry in a professional manner. Before you begin full-stack web development, there are a few crucial concepts that you need to understand that will be addressed in this task. This task will also give you a brief overview of what to expect from this level.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A note from the

HyperionDev Team

Congratulations on making it to level 2 of this Bootcamp! In this level you will build on the basics you learned in level 1 and develop some of the core skills needed to become a full-stack web developer.

To become an effective web developer you need to master working with some of the most important tools for any software developer including version control systems like git and the command line interface. You will learn to use these tools early on in this level.

To be a full stack web developer you also need to be able to write code for the front-end and back-end of your web application. In level 1 you learnt to use JavaScript. In level 2 you will use JavaScript for front-end and back-end development. To be able to do this, you will also learn more about how JavaScript works and some of its more advanced features, including how to work with some of the most popular JavaScript libraries, frameworks and runtimes. You will use REACT for front-end development and Node.js and Express for JavaScript development on the server.

Besides your Capstone Projects, you will also create REACT and full-stack applications that interact with third-party APIs and that interface with your own custom API that you will build using Express. You will learn to deploy all these apps with Heroku.

FULL STACK WEB DEVELOPMENT

A web developer should know how to build a website from the ground up, meaning that they should be able to create custom code to accommodate a client's unique needs and develop everything on the webpage, from the site layout to features and functions.

Web Development can be divided into three parts:

- Client-side scripting: This is code that executes in a web browser and determines what your customers or clients will see when they land on your website.
- Server-side scripting: This is code that executes on a web server and powers the behind-the-scenes mechanics of how a website works.

- Database technology: This stores and manages all the data that is needed for your web application. You will learn about this in the final level of this Bootcamp.

There are many frameworks and languages that can be used for full-stack web development. You will be learning to use the MERN (Mongo DB, Express, REACT, node.js) stack. The principles that you learn using this stack will be transferable to any full-stack development project that you develop with any framework or language.

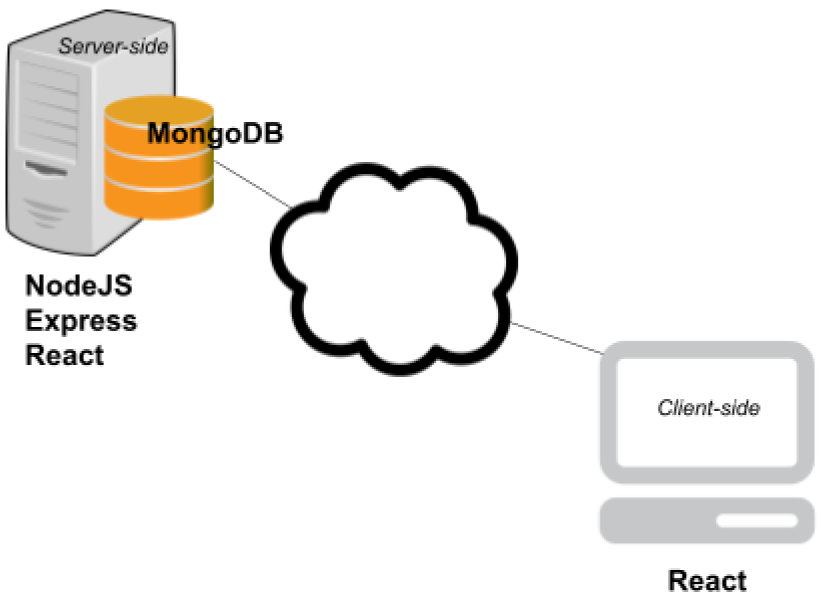
THE MERN STACK

A ***stack*** *is a term used to describe a collection of technologies that are used together to create a web application*. There are a number of web development stacks including:

- The LAMP stack: ie. a stack of technologies (Linux, Apache, MySQL, PHP) that is used to create web applications.
- The MEAN stack: technologies for web development in a MEAN stack are MongoDB, Express, AngularJS, Node.js.
- The MERN stack: as REACT has become more popular, it has often been used in place of AngularJS in the MEAN stack. This has resulted in the MEAN stack being replaced with the MERN stack.

You will be focussing on the MERN stack for the remainder of this bootcamp.

JavaScript was historically only used as a client-side scripting language. It was used in the same way that you have used it in the previous level of this Bootcamp. This has changed though and, as mentioned previously, JavaScript can now be used for full-stack web development. This can be done in various ways using a combination of many different frameworks and libraries. One of the most popular approaches to full-stack web development using JavaScript is using the MERN stack. Each of the components of the MERN stack is briefly introduced here. You will use, and learn much more about, each of these components over the rest of the tasks in this level.



- **REACT**: REACT is a JavaScript library created by Facebook that can be used to create views (the components that the user sees on the browser) that are rendered in HTML. Although the main purpose of REACT is to create front-end applications that display information to users, you will notice in the diagram above that REACT is isomorphic (i.e. it can run on the client-side and on the server).

- **Node.js**: JavaScript was initially designed to run in a browser. Node.js was designed so that JavaScript can be used, not just in a browser on the client-side, but on web servers as well. JavaScript is run in a browser using a JavaScript Engine (such as Chrome's V8 engine). Node.js designers made a JavaScript runtime environment that runs without a browser. In other words, Node.js is a runtime environment that can run JavaScript files without a browser on a web server.

- **Express**: Express is an unopinionated web framework for development using Node.js. In the same way that server-side web development can be supported by a web framework like Django, server-side web development using Node.js can be made easier using Express.

- **MongoDB**: MongoDB is a non-relational database that stores data as documents or objects. Whereas relational databases are queried using SQL, MongoDB uses a query language based on JSON.

Extra resource

The MERN stack is becoming increasingly popular and is a powerful stack to work in. If you're able to build and deploy good MERN applications, it will greatly help your career prospects as a developer. Read more about the benefits of using the MERN stack **here**.

SPOT CHECK 1

Let's see what you can remember from this section.

1. What is client-side scripting?
2. What is server-side scripting?
3. What is database technology?
4. Is the REACT framework client-side, server-side, or a database technology?
5. Is the Express framework client-side, server-side, or a database technology?
6. Is the Node framework client-side, server-side, or a database technology?
7. Is the Mongo framework client-side, server-side, or a database technology?

WEB DEVELOPER RESPONSIBILITIES

Large-scale web projects often divide these tasks among multiple web developers: one developer may focus on setting up the back-end of a site (back-end developer) while another focuses on the client-side to add style and functionality to the website itself (front-end developer).

There are three main types of web developers: front-end developers, back-end developers and full-stack developers.

Front-End Developer

The part of a website that your user sees and interacts within the browser is known as the front-end. Everything a user sees, clicks or uses to input or retrieve information is the work of a front-end developer. A front-end developer is responsible for creating the user-facing functionality of a website.

A front-end developer creates interaction and user experience using scripts embedded in a website's HTML. The front-end developer creates client-side software that brings the web designer's final design of a site to life. They create code that breaks the design down into separate components, then delivers information and functionality made possible by the back-end developer.

On the front-end, everything is built with a combination of HTML, CSS, and client-side scripts like JavaScript. These are the core elements of front-end development. In addition to being adept at these three main languages, front-end developers also need to be familiar with libraries like AngularJS or REACTJS, which ensure the content looks good no matter the device, and which package code into a more useful, time-saving form.

A front-end developer's responsibilities typically include:

- Prioritising user experience
- Bringing a designer's concept to life using HTML, CSS, and JavaScript
- Production and maintenance of websites and web application user interfaces
- Creating tools that enhance user interaction with the site in any browser
- Implementing responsive design for mobile sites
- Using a project management tool like GitHub to maintain software workflow management
- Looking at search engine optimisation (SEO) best practices
- Testing the site for usability and fixing any bugs during development

Back-End Developer

The back-end is basically the technology and programming that "power" a website. It consists of a server, an application and a database. A back-end developer builds and maintains the technology that powers those components which enable the user-facing side of the website to exist.

Whenever you navigate to a website such as **https://www.hyperiondev.com/**, for example, the site's servers send information to your computer or mobile device, which becomes the page you see. This is the result of the work of a back-end developer. In addition, when you enrolled in your course at Hyperion, the storage of your personal information was also made possible by the work of a back-end developer.

Back-end developers use server-side languages, which run on web servers, cloud-based servers, or a hybrid combination of both, to build an application. Examples of server-side languages are Node.js, PHP, Ruby, Python, Java, and .Net. Back-end developers also use tools like MongoDB, MySQL, Oracle, and SQL Server to find, save, or change data. The data are then served back to the user in front-end code which a user interacts with. Anything you see on a site is made possible by back-end code, which exists on, and is powered by, a server. Back-end developers use the tools mentioned above to create or contribute to web applications with clean, portable, well-documented code.

A back-end developer takes finished front-end code and gives it working functionality. For example, a back-end developer can make values in a drop-down menu possible by building the infrastructure that pulls values from the database.

A back-end developer's responsibilities typically include:

- Database creation, integration, and management
- Using back-end frameworks like Express.js to build server-side software
- Web server technologies
- Cloud computing integration
- Server-side programming languages
- Content management system development, deployment, and maintenance
- API integration
- Security settings and hack prevention
- Reporting, generating analytics and statistics
- Backup and restore technologies for a website's files and database

Full-Stack Developer

A full-stack developer can be thought of as a "jack-of-all-trades" since they can work on the full "stack" i.e. both the front-end and back-end. Being a Full Stack Developer doesn't necessarily mean that you have mastered everything required to work with the front-end or back-end, it rather means that you are able to work on both sides and understand what is going on when building an application.

There is often not a clear distinction between front-end and back-end development. Front-end developers often need to learn additional back-end skills and vice versa. A full-stack developer can work on the server-side of web programming like back-end developers, as well as speak the front-end languages that control how content looks on a site's user-facing side, fluently.

A full-stack developer has knowledge in all stages of software development and should be proficient in:

- Server, Network, and Hosting Environments: i.e. understanding what can break and why; appropriate use of the file system, cloud storage and network resources; knowing application scale, given the hardware constraints; and working side by side with DevOps
- Data Modeling: i.e. knowing how to create a reasonably normalised relational model, complete with foreign keys, indexes, views, lookup tables, etc; being familiar with the concept of non-relational data stores and understanding where they shine compared to relational data stores
- Business Logic: i.e. having solid object-oriented skills
- API layer/Action Layer/MVC: i.e. knowing how the outside world operates against the business logic and data model; using frameworks; being able to write clear, consistent, and simple-to-use interfaces
- User Interface: i.e. understanding how to create a readable layout; being able to acknowledge the need for help from artists and graphic designers to implement a good visual design; using HTML5 and CSS; using JavaScript (node, backbone, knockout etc.)
- User Experience: i.e. being able to build web applications that offer users a pleasant experience when interacting with them. For example, being able to step back and look at a process that currently needs seven clicks and four steps and get it down to just one click; write useful error messages
- Customer and Business Needs: i.e. having a grasp of what is going on when the customer uses the software; having a grasp of the customer's business

Often the distinction between a front-end developer and a backend developer becomes blurred. This is especially true since many front-end frameworks allow for server-side rendering.

A note from our coding lecturer

Pieter

There's a reason that true full-stack web developers are sometimes referred to as 'unicorns'. It's the rarity factor. A genuine full-stack web developer has such a diverse range of skills that they are hard to find. ***Read this post*** *to find out more about some of the sought-after skills you as a full-stack web developer will cultivate.*

SPOT CHECK 2

Let’s see what you can remember from this section.

1. What are 3 main responsibilities of a front-end developer?
2. What are 3 main responsibilities of a back-end developer?
3. What are 3 main responsibilities of a full-stack developer?

CRUCIAL KNOWLEDGE FOR BACK-END DEVELOPMENT

To be a full stack web developer one needs to know what is happening at the back-end of a web application on the server.

Web applications are usually backed by two types of servers:

1. Web servers
   A web server is basically a computer that has an Internet connection and the necessary software stored on it to be able to make web pages available to clients. A web server will store resources that are associated with web applications, for example, any images, HTML, CSS, and JavaScript files that are needed to make a web application function. In addition to this, a web server must have the software installed on it that allows it to act as an HTTP server (e.g. **IIS**). An HTTP server is basically a component that receives HTTP requests and sends the appropriate HTTP responses.

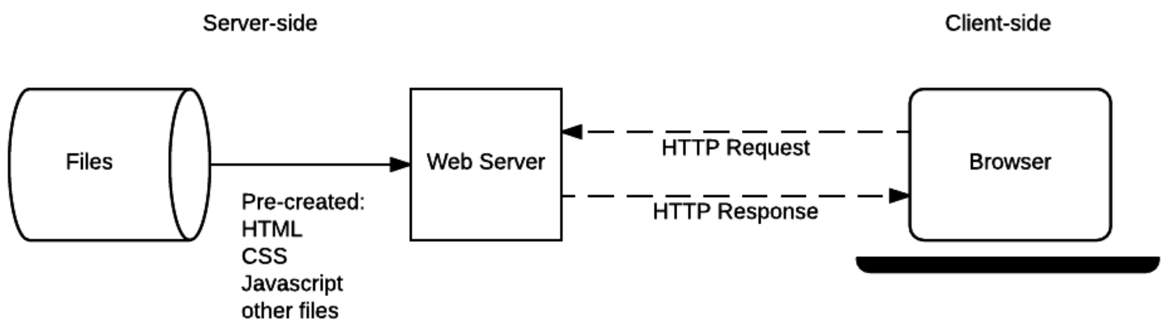
   The HyperText Transfer Protocol (HTTP) is used by web browsers to communicate with web servers. An HTTP request is sent from your browser to the target web server whenever you click a link on a web page, submit a form, or run a search. Web servers wait for these request messages and then process them when they arrive. The web servers then reply to the web browser with an HTTP response message.

   A static website returns the same hard-coded content from the web server whenever a particular resource is requested.

2. Application servers
   Application servers contain the business logic (code) needed to dynamically build resources to be passed back to the browser. An application server will usually have a web framework (e.g. a Node application server will usually use

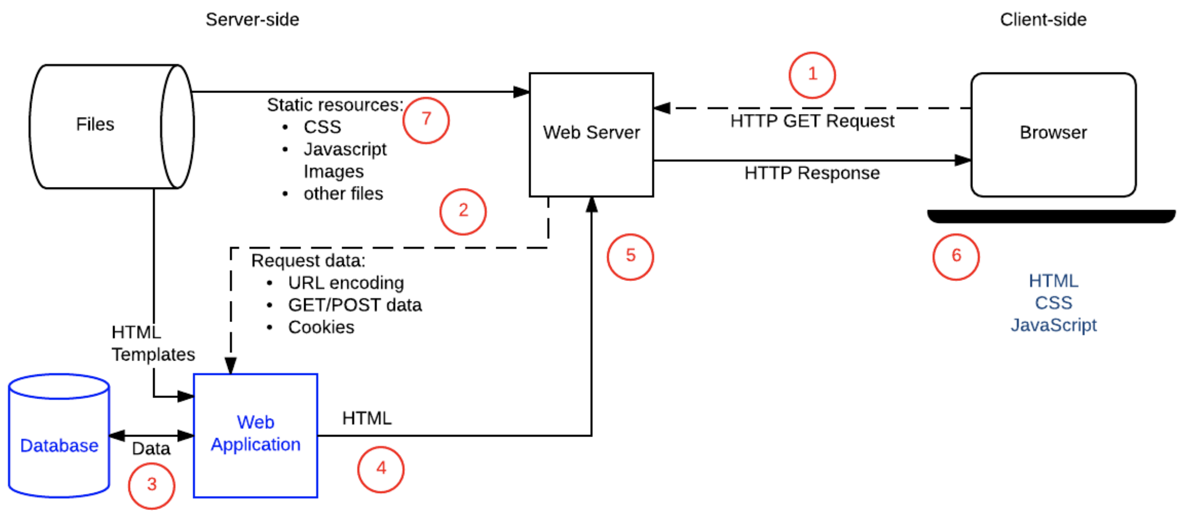
Express) running on it. The framework is able to match requests and dynamically generate responses.

If you only want your website to serve static pages, then you only need a web server.



*Web Server Architecture for a Static Site (**developer.mozilla.org**)*

However, if you want to build a web application that responds to requests with dynamic resources, then you need a web server and an application server.



*Web Server Architecture for a Dynamic Site (**developer.mozilla.org**)*

Typically a reverse proxy sits in front of your web servers and web application servers and routes traffic to the appropriate server. An application server is most likely run as a proxy of a web server. The web server usually just passes non-static requests to the application server.

In later tasks, you are going to learn to build servers using Node.js and Express.

Take note:

Here are a few **helpful definitions** from **NGINX**:

"A **proxy server** is a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet."

"A **reverse proxy server** is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers."

CRUCIAL KNOWLEDGE FOR FRONT-END DEVELOPMENT

How does a browser actually work?

As noted previously, a front-end developer is concerned about generating the resources (HTML, CSS, JavaScript, images, etc.) that will be displayed in the browser. To understand this role more fully, it is important to have at least a basic overview of how a web browser actually works. As we know, a web browser is used to request resources from a web server and then present these resources to the user. To be able to do this, a web browser contains the components depicted in the image below:

- The user interface: the interface that you interact with when you open a browser. It includes the address bar, back, forward, and refresh buttons etc.
- The rendering engine: displays the requested content. By default, the rendering engine can parse and display HTML and XML documents and images.
- The browser engine: takes inputs from the user interface (e.g. the URL input by the user) and sends this to the rendering engine and vice versa.
- Networking: used for making network calls such as HTTP requests.
- UI backend: allows the browser to interface with your device's operating system.
- JavaScript interpreter: used to parse and execute JavaScript code.
- Data persistence: used to store data such as cached files, cookies, sessionStorage, and localStorage.

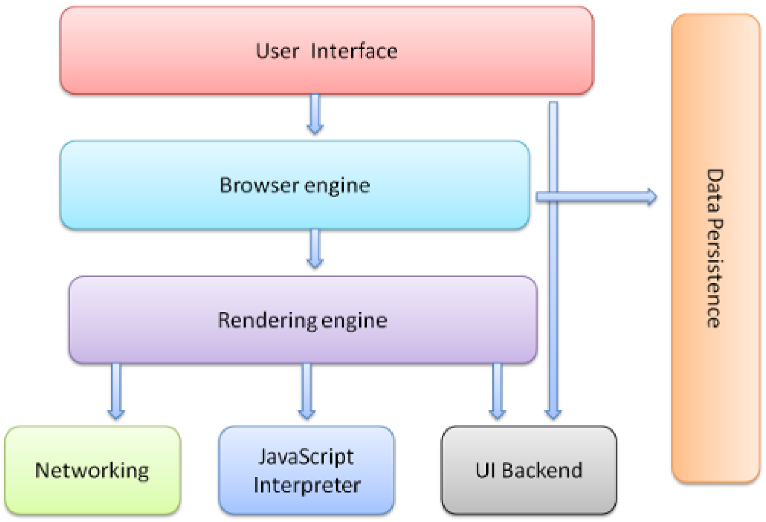


Image source:
https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/#The_browser_high_level_structure

The rendering engine does the following to render the page:

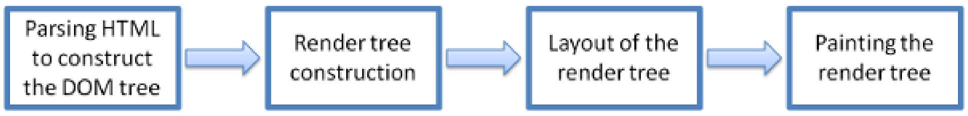
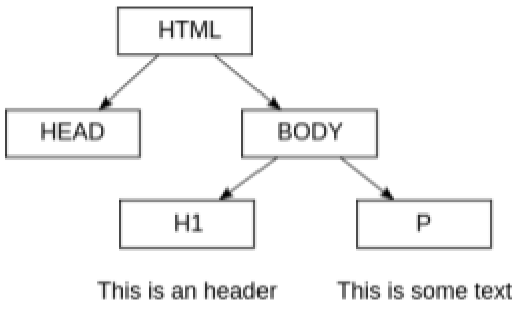

Parsing HTML to construct the DOM tree
Render tree construction
Layout of the render tree
Painting the render tree

Image source:
https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/#The_main_flow

Firstly, the browser parses the HTML to create a DOM tree. The CSS is then parsed and combined with the DOM tree to create a render tree which describes not only what elements should be rendered but how these elements should be styled. The render tree doesn't describe where on the page the various elements in the DOM tree should be positioned - this happens during the layout process. Finally, all the elements are 'painted' to display them in the browser UI.



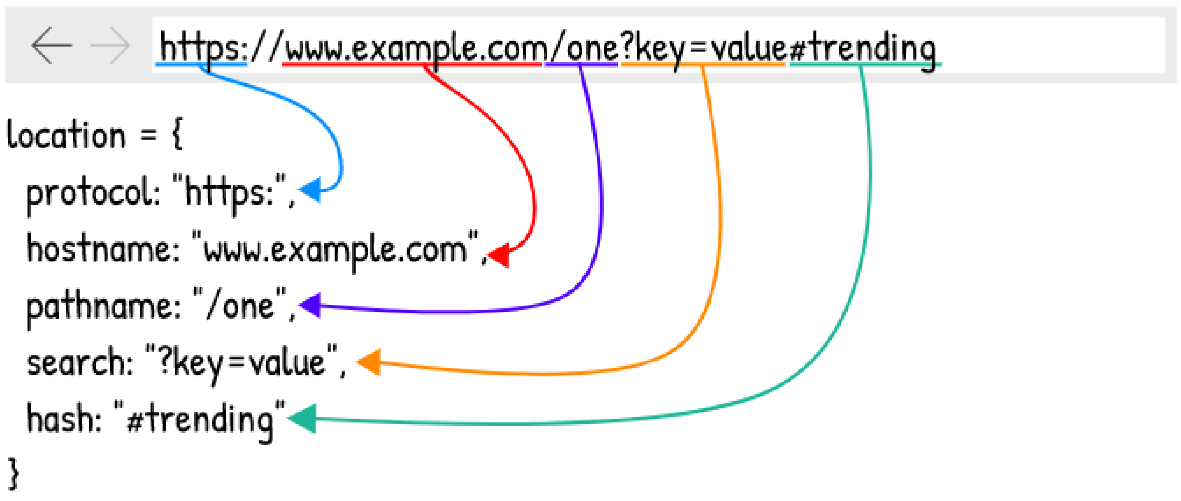
Example of a DOM tree.

There are a number of different types of front-end applications. Some of these are discussed next.

Single Page Applications (SPA)

REACT defines a single-page application (SPA) as, “an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.”

SPAs use the **location interface** to get information about the URL for a given resource. **Window.location** or **document.location** return a location object. The properties of a location object map directly with the parts of the URL as shown in the image below:



Location properties map from the URL. Image source:
https://blog.pshrmn.com/entry/how-single-page-applications-work/

Usually, when the location object changes, the browser makes a new request to the server and creates a new **Document** (that describes and provides methods for interacting with the DOM) based on the response it receives. However, with an SPA a new Document is NOT created every time the location object changes. Rather, an SPA will reuse and update the active Document when the location changes instead of making a new request to the server. It uses the **History API** to do this. When we created REACT applications using Create REACT App, we create single-page applications. Some SPAs use server-side rendering.

Server-side rendering (SSR)

As the name suggests, server-side rendering (SSR) is the process of taking a JavaScript framework (like REACT) and rendering it to static HTML and CSS on the server. As we saw previously, a browser's rendering engine is good at rendering HTML and CSS quickly and efficiently. It takes longer to parse JavaScript on a Browser though. JavaScript frameworks like REACT use JavaScript to render HTML. With SSR, this JavaScript is executed on the server and the resulting HTML that is rendered is then sent to the browser.

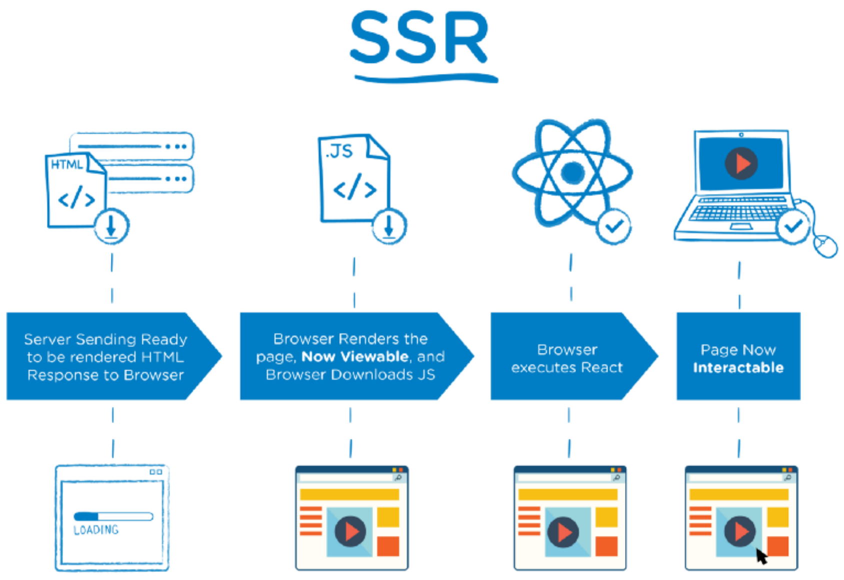

SSR
HTML
.JS
Server Sending Ready to be rendered HTML Response to Browser
Browser Renders the page, **Now Viewable**, and Browser Downloads JS
Browser executes React
Page Now **Interactable**
LOADING

Image source:
https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

Client-Side Rendering (CSR)

Whereas with SSR your server's response to the browser is the HTML of your page that is ready to be rendered, with CSR the browser gets a pretty empty document with links to your javascript. That means your browser will start rendering the HTML from your server without having to wait for all the JavaScript to be downloaded and executed.

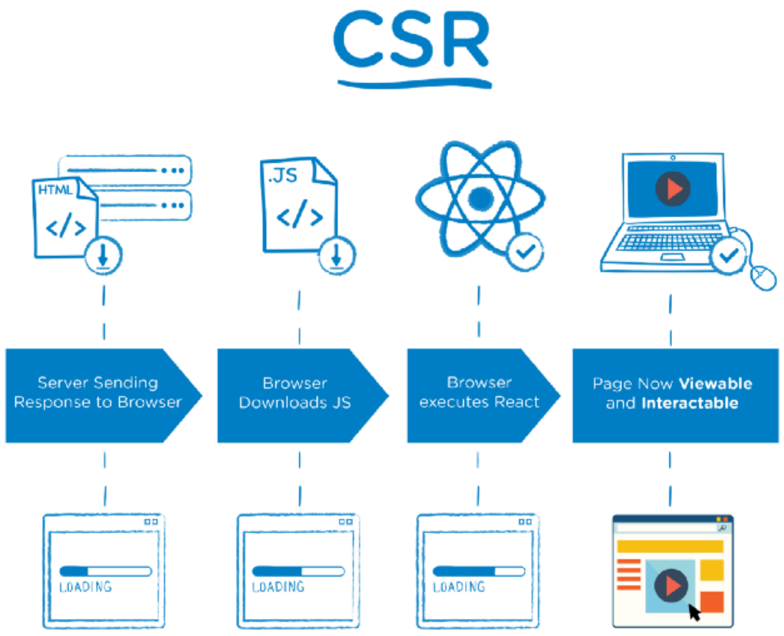
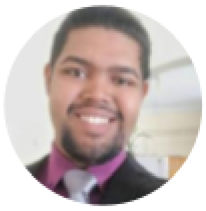

CSR
HTML
.JS
Server Sending Response to Browser
Browser Downloads JS
Browser executes React
Page Now **Viewable** and **Interactable**
LOADING
LOADING
LOADING

Image source:
https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8



A note from our coding mentor
Seraaj

*Getting a bit impatient about making money with your new web development skills? The great thing about a part-time online programming course is that you can start earning money from web development almost from the word go. You are about to learn to code using a very popular front-end library, REACT. Check out **these tips** to help you earn while you learn.*

SPOT CHECK 3

Let's see what you can remember from this section.

1. What is the user interface?
2. What is server-side rendering?
3. What is client-side rendering?

Compulsory Task 1

Follow these steps:

- Create a file called **full-stack answers.txt** in which you ***discuss*** the following questions in your own words:
    - What is full-stack web development? In your discussion include a description of what a stack is.
    - What is the MERN stack?
    - What types of server functions are typically implemented for the back-end of a web application? In your discussion, explain the role of each of the following: web server, application server, and proxy server.
    - What happens when a browser is used to send a request to a server? How does this differ with SSR and CSR?

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Click here to share your thoughts anonymously.

SPOT CHECK 1 ANSWERS

1. Client-side scripting: This is code that executes in a web browser and determines what your customers or clients will see when they land on your website.
2. Server-side scripting: This is code that executes on a web server and powers the behind-the-scenes mechanics of how a website works.
3. Database technology: This stores and manages all the data that is needed for your web application.
4. Client-side and server-side.
5. Server-side
6. Server-side
7. Database technology

SPOT CHECK 2 ANSWERS

1. A front-end developer's responsibilities typically include (any 3):
    a. Prioritising user experience
    b. Bringing a designer's concept to life using HTML, CSS and JavaScript
    c. Production and maintenance of websites and web application user interfaces
    d. Creating tools that enhance user interaction with the site in any browser
    e. Implementing responsive design for mobile sites
    f. Using a project management tool like GitHub to maintain software workflow management
    g. Looking at search engine optimisation (SEO) best practices
    h. Testing the site for usability and fixing any bugs during development
2. A back-end developer's responsibilities typically include (any 3):
    a. Database creation, integration and management
    b. Using back-end frameworks like Express.js to build server-side software
    c. Web server technologies
    d. Cloud computing integration
    e. Server-side programming languages
    f. Content management system development, deployment and maintenance
    g. API integration
    h. Security settings and hack prevents
    i. Reporting, generating analytics and statistics
    j. Backup and restore technologies for a website's files and database
3. A full-stack developer should be proficient in (any 3):
    a. Server, Network and Hosting Environments: i.e. understanding what can break and why; appropriate use of the file system, cloud storage and network resources; knowing application scale, given the hardware constraints; and working side by side with DevOps

b. Data Modeling: i.e. knowing how to create a reasonably normalised relational model, complete with foreign keys, indexes, views, lookup tables, etc; being familiar with the concept of non-relational data stores and understanding where they shine over relational data stores
c. Business Logic: i.e. having solid object-oriented skills
d. API layer/Action Layer/MVC: i.e. knowing how the outside world operates against the business logic and data model; using frameworks; being able to write clear, consistent and simple-to-use interfaces
e. User Interface: i.e. understanding how to create a readable layout; being able to acknowledge the need for help from artists and graphic designers to implement a good visual design; using HTML5 and CSS; using JavaScript (node, backbone, knockout etc.)
f. User Experience: i.e. being able to build web applications that offer users a pleasant experience when interacting with them. For example, being able to step back and look at a process that currently needs seven clicks and four steps and get it down to just one click; write useful error messages
g. Customer and Business Needs: i.e. having a grasp of what is going on when the customer uses the software; having a grasp of the customer's business

SPOT CHECK 3 ANSWERS

1. The interface that you interact with when you open a browser. It includes the address bar, back, forward, and refresh buttons
2. The process of taking a JavaScript framework (like REACT) and rendering it to static HTML and CSS on the server
3. Whereas with SSR your server's response to the browser is the HTML of your page that is ready to be rendered, with CSR your browser will start rendering the HTML from your server without having to wait for all the JavaScript to be downloaded and executed.