

2222124-ACCA7004- Assignment1-Final

by Callan Birch-Williams

Submission date: 31-Mar-2023 09:42PM (UTC+0100)

Submission ID: 201237634

File name: 42481_Callan_Birch-Williams_2222124-ACCA7004-Assignment1-Final_488464_1963971661.pdf
(481.86K)

Word count: 4243

Character count: 28329

UWTSD

DVLA: MSc Software Engineering (Web Development)

Web Development (ACCA7004), Assignment 1

Callan Birch-Williams, 2222124

Word Count: 3282

Table of Contents

Introduction.....	3
Server-side scripting	3
<i>Overview.....</i>	<i>3</i>
<i>Languages</i>	<i>4</i>
Client-side scripting	5
Server-side technology: Node.js	6
<i>JavaScript</i>	<i>6</i>
<i>Architecture.....</i>	<i>6</i>
<i>Evaluation.....</i>	<i>8</i>
Client-side technology: React.....	9
<i>AJAX.....</i>	<i>12</i>
<i>JWT.....</i>	<i>13</i>
Virtualisation technology: Docker.....	14
<i>Containers</i>	<i>15</i>
<i>Evaluation.....</i>	<i>15</i>
Bibliography	17
 Figure 1: Static website architecture [1].....	3
Figure 2: Dynamic website architecture [1].....	3
Figure 3: Node.js event loop [21].	7
Figure 4: Node.js example HTTP server.	7
Figure 5: Virtual DOM rendering visualisation [30].	9
Figure 6: Example TODO application.....	10
Figure 7: Example TODO application main component JSX.....	10
Figure 8: Example TODO application list component JSX.	11
Figure 9: Ajax within React example.....	12
Figure 10: VM and Docker structure visualisation [33].	14

Introduction

The purpose of this document is to outline the fundamental differences between server-side and client-side web technology, and how they interact with each other, providing a critical evaluation of modern web development frameworks.

Server-side scripting

Overview

Server-side scripting, within the context of web development, refers to the programming of software that runs on a web server to generate dynamic responses to a client's browser. Typically, a client's request is processed, and a response is generated, commonly after performing some I/O operation to retrieve data.

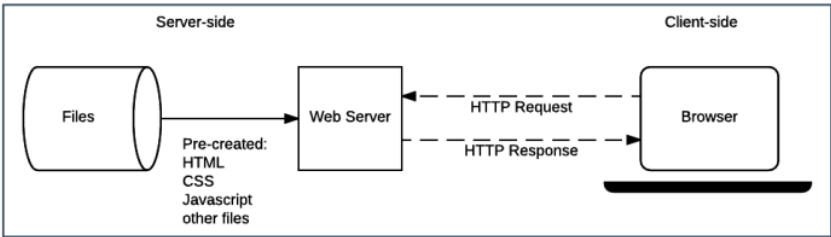


FIGURE 1: STATIC WEBSITE ARCHITECTURE [1].

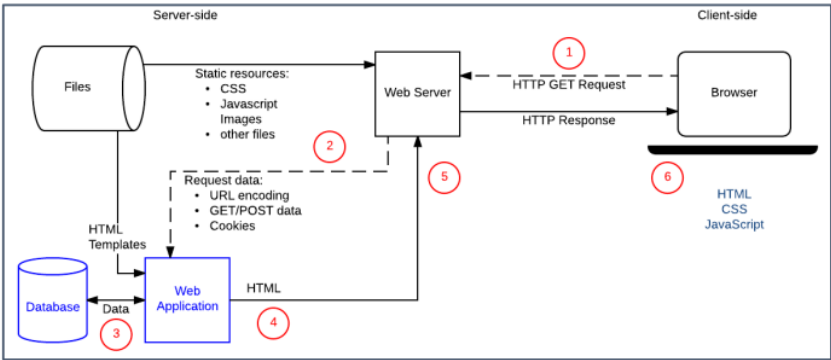


FIGURE 2: DYNAMIC WEBSITE ARCHITECTURE [1].

Figure 1 demonstrates the simplified architecture of a static website. A static website simply returns pre-created resources when they are requested. The client sends a HTTP request specifying the resource required.

Figure 2 on the other hand, demonstrates the architecture of a dynamic website. Like a static website, the overarching content layout and other relevant resources are provided normally (HTML templates, styling, etc.). However, some of the response content is generated dynamically by a server-side application, typically after retrieving data from a database or API. This is what is referred to as server-side scripting.

Languages

There are various languages used in modern web development when it comes to creating a server-side application. This varies from pure scripting languages to sophisticated frameworks and database query languages. Some common examples are as follows:

- PHP is a general-purpose scripting language designed primarily for web development [2]. It remains one of the most widely used server-side scripting languages used for web development. As of 2023 PHP is used in 77.4% of all websites whose server-side language is known [3].
- Django is a high-level Python web framework, designed to encourage rapid development while still providing scalability and security [4].
- Node.js is a runtime environment built on the V8 JavaScript engine that allows JavaScript to run on the server-side [5]. It is a low-level implementation, and it is therefore common for developers to harness the power of a framework such as Express [6].
- Java is a commonly used server-side language when it comes to enterprise level applications. Coupled with the Spring Framework [7], it is known for its security and scalability, lending itself to large-scale projects.
- Ruby on Rails, unlike the other languages on this list, is a full-stack web development framework. This equips developers with all the tools needed to build web applications on both the front and back end [8]. It places an emphasis on design patterns and paradigms, with the most obvious example being its design around the model-view-controller pattern.

Databases are a fundamental component of server-side development, as such there are languages to interface with them. More specifically, a query language is a domain-specific programming language for searching and changing the content of a database [9]. Most relational databases are equipped with some dialect of SQL (Structured Query Language), the standard language for relational database management systems [10]. NoSQL databases on the other hand do not follow a language standard and their query language typically depends on the individual database program (An example of this will be explored in the Node.js section of this document). To avoid learning multiple query languages most server-side languages have ORM libraries to abstract this layer away from the developer [11].

Client-side scripting

Client-side scripting refers to the programming of software that runs on the client-side. In the context of web page applications this is the browser. As discussed previously, these are files that are sent to the client when a web page is loaded. Client-side scripting primarily deals with the user interface with which the user interacts. Client-side scripting is essential to modern web development where web pages are expected to be interactive and dynamic.

At a base level, by far the three major components of modern client-side scripting are HTML, CSS, and JavaScript:

- HTML defines the layout of the content and provides structure for design. It is the fundamental component of any web page.
- CSS is used to apply styling to the different elements of a web page. It can be considered responsible for the look and feel of a web page.
- JavaScript is used to add interactivity to a web page by handling complex functionality (Drop-down menus, pop-up windows, web queries, etc.) and is the programmatic language out of the three.

Other client-side scripting languages include Adobe Flash Player [12] and VBScript, though these languages do not see modern use in web development as a result of discontinuation and incompatibility with most modern browsers respectively (Old web pages using VBScript can only function on Edge when in IE compatibility mode [13]).

While strictly a library for JavaScript rather than a scripting language of its own, jQuery changed the way modern JavaScript is written by providing easier access to HTML document traversal, manipulation, event handling and AJAX [14]. jQuery is starting to see less use in modern web development due to the prevalence of frontend frameworks and modern browsers introducing many of the features of jQuery into plain JavaScript.

While not necessarily scripting languages on their own, frameworks such as Angular, React and Vue change the experience of the developer significantly by changing the approach to creating web pages with HTML, CSS, and JavaScript. The importance of modern frontend development frameworks will be further explored in the React section of this document.

Server-side technology: Node.js

Node.js is an open-source, cross-platform runtime environment that enables JavaScript to be ran outside of a browser, primarily designed for building network applications [5]. It is built upon Google's open-source, high-performance JavaScript and WebAssembly engine, V8 [15]. There have been versions of Node.js coupled with various underlying engines such as ChakraCore or SpiderMonkey, though these repositories are no longer maintained [16] [17]. The creation of Node.js is accredited to Ryan Dahl in 2009, as a solution to the lack of JavaScript web server technology at the time [18].

JavaScript

By making use of JavaScript on the server, Node.js has various unique benefits when it comes to web development. Developers can write code in one language, reducing the disconnect between frontend and backend development, increasing understanding and allowing for code to be written that can be reused trivially across the entire stack.

In the context of I/O, JavaScript is an appealing option. JSON is one of the most popular data interchange formats used in modern web development [19] and is native to JavaScript, making it easy to handle. Additionally, various NoSQL databases such as MongoDB make use of JavaScript for their shell and query language [20].

Modern JavaScript is a compilation target for many languages. As a result of this, traditional backend language features missing from JavaScript (For example, static type definitions) can be added by other languages such as TypeScript.

Architecture

Node.js follows an asynchronous, event-driven architecture using callbacks. Unlike most other modern environments, Node.js does not rely on multithreading to support concurrent execution of business logic [21]. Node.js will never perform direct I/O (Excluding synchronous methods within its standard library) and instead follows an event loop (This is abstracted away from the developer, like browser JavaScript) that assigns operations to the underlying operating system where possible.

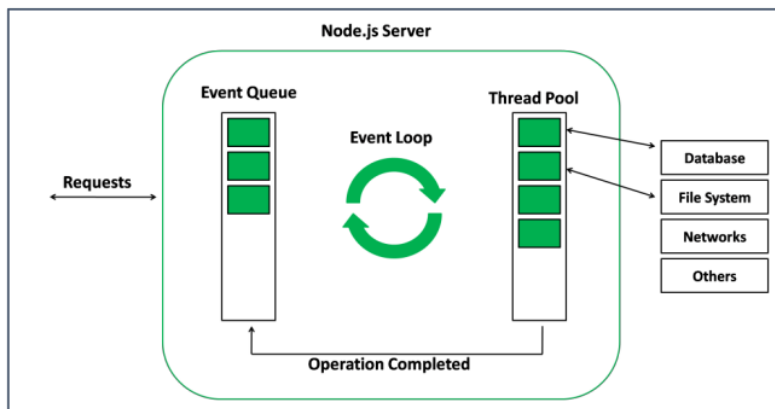


FIGURE 3: NODE.JS EVENT LOOP [22].

It is common in conventional server-sided programming for an operation to block a process from continuing. This is usually overcome using a multithreaded approach. Figure 3 instead provides an example of a Node.js HTTP server that responds to every request with a HTML file. Whenever a request is received, a callback is fired, and the response is constructed. The event loop is able to handle other events while the process is busy reading the file. A comparison can be drawn between this event-driven model with events in browser JavaScript such as onclick. While the callback pattern is intuitive and easy to work with, an over-reliance on callbacks can often result in many nested callbacks causing difficulty with understanding code.

```

const http = require('http');
const fs = require('fs');

const server = http.createServer();

server.on('request', (request, response) => {
  fs.readFile('file.html', function(error, data) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.write(data);
    return response.end();
  });
});

server.listen(3000);
console.log('Server running at http://localhost:3000/');

```

FIGURE 4: NODE.JS EXAMPLE HTTP SERVER.

Evaluation

When evaluating Node.js it is important to consider both its benefits and limitations in modern web development. Some of the additional benefits and disadvantages of Node.js not previously mentioned will be discussed in this section.

Node.js offers easy scalability (Horizontally as opposed to vertically [23]), accredited to its architecture which takes fewer resources for an application to accept concurrent connections in comparison to other approaches. On the other hand, one of the major drawbacks of Node.js is its inability to process heavy CPU-bound tasks. However, Node.js did introduce the worker threads module in 2018 [24] to allow developers to execute multiple threads at the same time.

A major advantage of using Node.js is its ability to target all major operating systems and platforms. As a result of this, it is also common to run Node.js inside of a Docker container (Which will be covered in more detail in the Docker section of this document), which mitigates some of the weaknesses of Node.js such as dependency management and easy of deployment. Furthermore, by taking advantage of frameworks such as Electron [25], applications can be built to run on the desktop without major code rewriting.

A potential benefit of Node.js is its relationship with npm [26], which offers developers a vast library of open-source packages. However, npm lacks a library system that can be found in other languages, resulting in no guarantee that a registry has adequate code and documentation quality [27].

Node.js is committed to a long-term support policy. That being said, its frequent rate of backward-incompatible changes results in development time being allocated to rewriting solutions for newer APIs.

Overall, Node.js has a plethora of benefits and most drawbacks can be mitigated by developer experience and are expected to be overcome as Node.js matures as a technology.

Client-side technology: React

React is an open-source, front-end JavaScript-based library for developing user interfaces based on components [28]. React is mostly concerned with the user interface and rendering individual components to the DOM, therefore it is usually used alongside other technologies to create a web application.

As mentioned previously, React uses a component-based approach as opposed to the traditional DOM-based approach. React developers can build up user interfaces by making use of individual components. Composition is a key feature of React and facilitates the development of more complex functionality by combining smaller, focused functions [29].

A functional approach is achieved in React through the use of JSX, a JavaScript syntax extension that allows HTML code to be embedded within JavaScript documents, keeping markup close to related rendering logic. This differs from the approach to web development prior to the adoption of React, where components are disconnected from their behaviour.

A significant feature of React when compared with traditional JavaScript is the use of a Virtual DOM. In traditional JavaScript data is stored in the DOM, which is created and maintained by the browser and is a representation of all HTML elements on the page. Therefore, any user input is stored by the browser [30]. To access this data in JavaScript it must be found in the DOM. On the other hand, React does not rely on the DOM to store application state, making it easier to handle user data.

Another benefit of using a Virtual DOM is rendering performance. The Virtual DOM evaluates component state and only updates components that have changed in the real DOM, as opposed to updating every element again. Furthermore, the handling of re-rendering is abstracted away from the developer in React, whereas traditional JavaScript web applications required the developer to carefully handle rendering of the DOM.

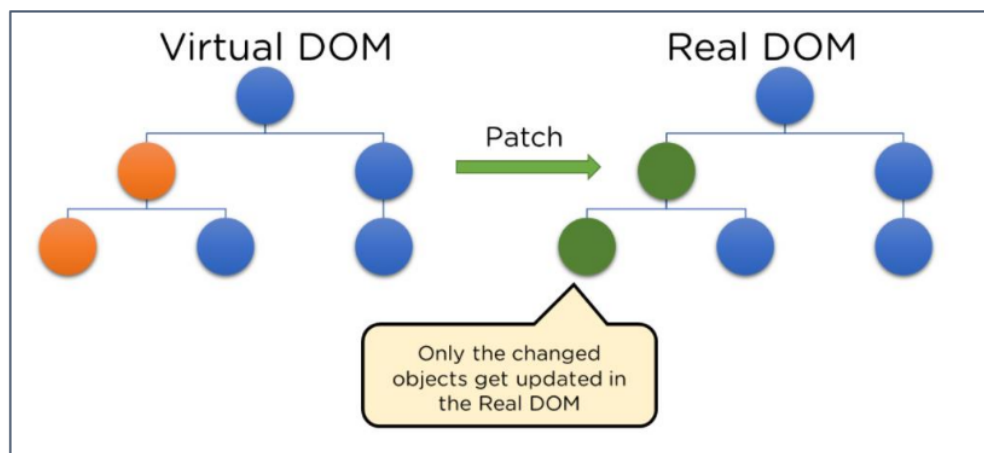


FIGURE 5: VIRTUAL DOM RENDERING VISUALISATION [31].

To appreciate the component-based approach of React, consider the following example of a simple TODO application. The main application components markup has been simplified into logical elements of the user interface by splitting them into their own React components. Additionally, a parent hierarchy is used to isolate components away from other components that do not need to know about their existence. Functionality and layout of the list component is maintained closely within the same function.

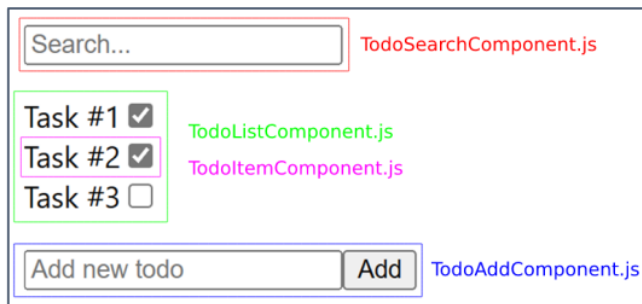


FIGURE 6: EXAMPLE TODO APPLICATION.

```
//...
return (
  <div>
    <TodoSearchComponent searchTerm={searchTerm} onSearch={handleSearch}
  />
    <TodoListComponent todos={todos} searchTerm={searchTerm}
      toggleComplete={handleToggleComplete} />
    <TodoAddComponent onAdd={handleAddTodo} />
  </div>
);
//...
```

FIGURE 7: EXAMPLE TODO APPLICATION MAIN COMPONENT JSX.

```
const TodoListComponent = ({todos, searchTerm, toggleComplete}) => {  
  const filteredTodos = //...  
  
  return (  
    <ul style={{listStyle: 'none', padding: 0}}>  
      {filteredTodos.map(todo => (  
        <TodoItemComponent key={todo.id} todo={todo}  
          toggleComplete={toggleComplete} />  
      ))}  
    </ul>  
  );  
};  
  
export default TodoListComponent;
```

FIGURE 8: EXAMPLE **TODO** APPLICATION LIST COMPONENT JSX.

AJAX

AJAX (Asynchronous JavaScript and XML) is a set of web development techniques that facilitate asynchronous updating of web pages without the entire page being reloaded. Despite its original name, modern implementations of the AJAX techniques typically make use of JSON rather than XML.

AJAX and related technologies were intrinsic to the advancement of web applications and have become essential in modern web development. As a result, the underlying principles are typically abstracted away from the developer despite being closely adhered to. A modern example of AJAX within React, using the JavaScript Fetch API, is shown in Figure 9. The `useEffect()` React hook will be called after the DOM is updated (After the component is initially rendered). Once data is available from the Fetch API callback, the `useState()` hook will cause the React component to be re-rendered with the new information.

```
import React, { useEffect, useState } from 'react';

function CatFactComponent() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://catfact.ninja/fact')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error(error));
  }, []);

  return (
    <div>
      <p>{data ? data.fact : 'Loading Fact...'}</p>
    </div>
  );
}

export default CatFactComponent;
```

FIGURE 9: AJAX WITHIN REACT EXAMPLE.

JWT

JWT (JSON Web Token) is a standard that defines a method for securely transmitting information as a JSON object between two parties. A JWT is comprised of three components:

- Header
- Payload
- Signature

The header contains the type of token being used, as well as the algorithm used to sign it. The payload contains the claims. Claims are defined as the statements about an entity (Most commonly the user) and additional information. For example, this could be the user's ID, name, role, etc. in JSON form. Finally, the signature is a hash of the header, payload and a secret key known only to the server that issued the token. The signature can be used to verify the content hasn't been changed as well as verify the sender of the JWT [32].

The most common use case for JWT within web development is user authentication and secure data transmission. Once a user has logged in, the server can generate a JWT and send it to the client. The client can then use the JWT in subsequent requests to access protected resources.

While React does not use JWTs directly, React applications are frequently built to integrate with back-end APIs that make use of JWT for authentication. When a user logs into a React application, the back-end server can generate a JWT and return it to the client as part of the authentication response. To make use of the JWT within React, third-party libraries such as `jsonwebtoken` [33] to encode and decode JWTs. An implementation of AJAX such as the native Fetch API can then be used to make requests, including the JWT in the request headers.

Virtualisation technology: Docker

Virtualisation is a technology that is used to create virtual representations of servers, storage, networks, and other physical machines. A traditional virtual machine is a system which behaves exactly like a computer. That is, it makes it possible to run what appears to be multiple computers on the same hardware. Each virtual machine requires its own operating system and then the hardware is virtualised using a piece of software referred to as a Hypervisor which manages the resource allocation of the physical hardware.

Docker is a modern, open-source platform that replaces the traditional virtual machine approach to virtualisation with a concept known as “containers” at the operating system level to make the creation, deployment and running of applications easier [34].

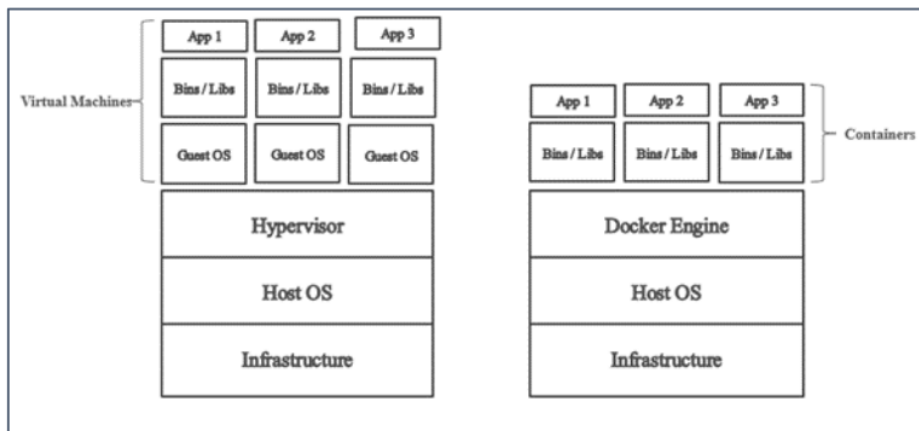


FIGURE 10: VM AND DOCKER STRUCTURE VISUALISATION [35].

Figure 10 clearly outlines the fundamental difference between Docker and virtual machines. Each virtual machine has the guest operating system above the host operating system. Each Docker container, on the other hand, shares the host operating system resulting in much lower overhead when it comes to managing the container system. Docker containers are managed by the Docker engine. Resources such as network, memory, CPU, and I/O are allocated by the Linux kernel without having to start a virtual machine [36]. Docker therefore lends itself to situations where multiple applications can be run on the same operating system kernel. Virtual machines are still appropriate when this isn't true [35].

Containers

Docker containers are a lighter-weight approach to virtualisation as they do not use a hypervisor, resulting in faster resource provisioning and availability of new applications [37]. A Docker image is a standalone, executable package of software bundled with all its dependencies (Code, runtime environment, libraries, configuration files, etc.). An image becomes a container at runtime, in the case of Docker, this is when they are run on the Docker engine (Shown above the host operating system in Figure 10 previously).

Evaluation

The adoption of Docker can be accredited to the solutions it provides to various elements of modern web development:

- **Portability:** Docker's container-based platform makes it easy to deploy applications to different environments. That being said, as Docker containers use the host operating system kernel, there may be instances where an application requires a special system call that is not available.
- **Consistency:** The Docker engine ensures that containerised applications will run the same way across multiple environments, mitigating the risk of platform-specific bugs and divergent behaviour. Additionally, configuration is done at the container level rather than specific configuration for each platform.
- **Efficiency:** Instances of containerised applications use much less memory than traditional virtual machines. Additionally, they start up and stop quicker and fit more densely on their hosting hardware [38]. Configuration of containers and various environments (Development, production, etc.) takes less time and effort, enabling developer time to be allocated elsewhere. All these benefits often result in reduced costs when compared with virtual machines.

One of the major benefits of Docker in modern web development is how it integrates with other leading technologies and methodologies. Cloud computing providers such as AWS (Amazon Web Services) and GCP (Google Compute Platform) have added support for Docker containers. Docker's consistency between environments, the ability to isolate dependencies, scale quickly horizontally using containers and speed of deployment make it an ideal candidate for CI/CD in modern web development [39]. These benefits additionally align themselves with the modern agile approach to development where it is common for multiple developers to be working on the same area of code.

While Docker has quickly become a major component of modern web development, there are undoubtedly some cases in which traditional virtualisation is still preferred.

Docker's shared kernel architecture and flexible configuration makes it more susceptible to security vulnerabilities when compared with a virtual machine in general [40]. Attackers can access both the host operating system as well as other containers by exploiting kernel vulnerabilities or misconfiguration.

Another consideration to make is that Docker can be expensive for existing teams to adopt as developers have to learn how to use various Docker technologies (Especially in cases where there is no existing Docker base image for your use case), though this is becoming less of a concern as the concept of containerisation matures and becomes a more common developer skill.

Bibliography

- [1] Mozilla Corporation, "Introduction to the server side - Learn web development | MDN," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction. [Accessed 03 2023].
- [2] The PHP Group, "PHP: Hypertext Preprocessor," [Online]. Available: <https://www.php.net/>. [Accessed 03 2023].
- [3] W3Techs, "Usage Statistics and Market Share of PHP for Websites, March 2023," [Online]. Available: <https://w3techs.com/technologies/details/pl-php>. [Accessed 03 2023].
- [4] Django Software Foundation, "The web framework for perfectionists with deadlines | Django," [Online]. Available: <https://www.djangoproject.com/>. [Accessed 03 2023].
- [5] T. O. Foundation, "Node.js," [Online]. Available: <https://nodejs.org/en>. [Accessed 03 2023].
- [6] OpenJs Foundation, "Express - Node.js web application framework," [Online]. Available: <https://expressjs.com/>. [Accessed 03 2023].
- [7] VMware Tanzu, "Spring | Home," [Online]. Available: <https://spring.io/>. [Accessed 03 2023].
- [8] Ruby on Rails, "Getting Started with Rails - Ruby on Rails Guides," [Online]. Available: https://guides.rubyonrails.org/getting_started.html. [Accessed 03 2023].
- [9] Engati, "Query language | Engati," [Online]. Available: <https://www.engati.com/glossary/query-language#:~:text=REQUEST%20A%20DEMO-,What%20is%20query%20language%3F,or%20data%20from%20a%20database..> [Accessed 03 2023].
- [10] International Organization for Standardization, "ISO/IEC 9075-1:2016," 2016. [Online].
] Available: <https://www.iso.org/standard/63555.html>.
- [11] I. V. Abba, "What is an ORM - The Meaning of Object Relational Mapping Database Tools," [Online]. Available: <https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/>. [Accessed 03 2023].
- [12] Adobe, "Adobe Flash Player End of Life," [Online]. Available:
] <https://www.adobe.com/products/flashplayer/end-of-life-alternative.html>. [Accessed 03 2023].
- [13] Microsoft, "Internet Explorer mode in Microsoft Edge," [Online]. Available:
] <https://support.microsoft.com/en-us/microsoft-edge/internet-explorer-mode-in-microsoft-edge-6604162f-e38a-48b2-acd2-682dbac6f0de>. [Accessed 03 2023].

- [14 OpenJS Foundation, "jQuery," [Online]. Available: <https://jquery.com/>. [Accessed 03 2023].
]
- [15 V8, "V8 JavaScript engine," [Online]. Available: <https://v8.dev/>. [Accessed 03 2023].
]
- [16 Nodejs, "nodejs/node-chakracore: Node.js on ChakraCore," [Online]. Available:
] <https://github.com/nodejs/node-chakracore>. [Accessed 03 2023].
- [17 Mozilla, "mozilla/spidernode: Node.js on top of SpiderMonkey," [Online]. Available:
] <https://github.com/mozilla/spidernode>. [Accessed 03 2023].
- [18 R. Dahl, "Node.js," in *Presentation at the European JavaScript Conference*, Berlin, 2009.
]
- [19 JSON, "Introducing JSON," [Online]. Available: <https://www.json.org/json-en.html>. [Accessed 03
] 2023].
- [20 MongoDB, Inc., "Write Scripts for the mongo Shell - MongoDB Manual," [Online]. Available:
] <https://www.mongodb.com/docs/v5.0/tutorial/write-scripts-for-the-mongo-shell/>. [Accessed
03 2023].
- [21 S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network
] Programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80-83, 2010.
- [22 GeeksforGeeks, "Node.js Event Loop," [Online]. Available:
] <https://www.geeksforgeeks.org/node-js-event-loop/>. [Accessed 03 2023].
- [23 H. Martland, "Horizontal vs Vertical Scaling in Node.js | by Harry Martland | Better
] Programming," [Online]. Available: <https://betterprogramming.pub/horizontal-vs-vertical-scaling-in-node-js-1b4f3ec8282>. [Accessed 03 2023].
- [24 "Worker threads | Node.js v19.8.1 Documentation," [Online]. Available:
] https://nodejs.org/api/worker_threads.html. [Accessed 03 2023].
- [25 OpenJS Foundation, "Build cross-platform desktop applications with JavaScript, HTML, and CSS
] | Electron," [Online]. Available: <https://www.electronjs.org/>. [Accessed 03 2023].
- [26 Github, "About npm | npm Docs," [Online]. Available: <https://docs.npmjs.com/about-npm>.
] [Accessed 03 2023].
- [27 EPAM Anywhere, "5 Main Node.js Advantages and Disadvantages | EPAM Anywhere Business,"
] [Online]. Available: <https://anywhere.epam.com/business/node-js-pros-and-cons>. [Accessed 03
2023].
- [28 Meta Open Source, "React," [Online]. Available: <https://react.dev/>. [Accessed 03 2023].
]

- [29 F. Copes, "React Concept: Composition," 29 09 2019. [Online]. Available:
] <https://flaviocopes.com/react-composition/>. [Accessed 03 2023].
- [30 N. Ruzicka, "React vs. Vanilla Javascript. Because web apps are becoming... | by Nick Ruzicka | Medium," [Online]. Available: https://medium.com/@Nick_C_Rose/react-vs-vanilla-javascript-f5af0a9305a2. [Accessed 03 2023].
- [31 C. Deshpande, "The Best Guide to Know What Is React [Updated]," [Online]. Available:
] <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs#:~:text=React%20is%20a%20JavaScript%2Dbased,frontend%20libraries%20for%20web%20development..> [Accessed 03 2023].
- [32 JWT, "JSON Web Token Introduction - jwt.io," [Online]. Available: <https://jwt.io/introduction>.
] [Accessed 03 2023].
- [33 NPM, "jsonwebtoken - npm," [Online]. Available:
] <https://www.npmjs.com/package/jsonwebtoken>. [Accessed 03 2023].
- [34 Docker Inc., "Docker: Accelerated, Containerized Application Development," [Online]. Available:
] <https://www.docker.com/#>. [Accessed 03 2023].
- [35 A. Abdullahi, "Docker vs Virtual Machine (VM) - Understanding the Differences," [Online].
] Available: <https://geekflare.com/docker-vs-virtual-machine/>. [Accessed 03 2023].
- [36 K.-T. Seo, H.-S. Hwang, I.-Y. Moon, I.-Y. Moon and B.-J. Kim, Performance Comparison Analysis
] of Linux Container and Virtual Machine for Building Cloud, 2014.
- [37 IBM Cloud Team, "Containers vs. Virtual Machines (VMs): What's the Difference? | IBM,"
] [Online]. Available: <https://www.ibm.com/cloud/blog/containers-vs-vms>. [Accessed 03 2023].
- [38 S. Yegulalp, "Why you should use Docker and Containers," [Online]. Available:
] <https://www.infoworld.com/article/3310941/why-you-should-use-docker-and-containers.html>.
[Accessed 03 2023].
- [39 E. Novoseltseva, "Using docker in software development by ING, Spotify & ADP," [Online].
] Available: <https://apiumhub.com/tech-blog-barcelona/top-benefits-using-docker/>. [Accessed 03 2023].
- [40 T. Bui, "Analysis of Docker Security," Espoo, 2015.
]

FINAL GRADE

81 / 100

GENERAL COMMENTS

Instructor

CSvsSS (25):17

node (25):22

AJAX/React/JWT (25):23

Docker (25): 19

Feedback:

=====

You could have provided a little more discussion of offloading processing to the client and security issues. Quite like the server-side diagrams, but you could have mentioned that the static website architecture is related to cloud computing blob storage. Not sure the examples sufficiently demonstrated the difference between CS and SS scripting.

Node.js described well and a number of benefits outlined, although some could have been expanded upon.

AJAX/React discussion is very good, JWT could have been expanded upon.

Docker discussion could have given more consideration to the flexibility of containers and VMs e.g. speed or startup/shutdown and when to use a VM.

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19
