

1300155 ELE web dev 1

by Elanwy Evans

Submission date: 31-Mar-2023 07:37PM (UTC+0100)

Submission ID: 201229171

File name: 42119_Elanwy_Evans_1300155_ELE_web_dev_1_488464_1721995794.docx (1.84M)

Word count: 4356

Character count: 25581



Prifysgol Cymru

Y Drindod Dewi Sant

University of Wales

Trinity Saint David

University of Wales Trinity Saint David

School of Computing

31/03/2023/ACCA7004

Assignment 1

Web Development

By Elanwy Evans (Student ID 1300155)

Table of Contents

<i>Client-side & Server-side Scripting.....</i>	<i>3</i>
Languages	5
Required Run-time Environment.....	5
Database Connectivity.....	7
Browser	7
Customization	7
<i>Node JS.....</i>	<i>7</i>
<i>AJAX.....</i>	<i>10</i>
<i>React.....</i>	<i>13</i>
<i>JSON Web Tokens.....</i>	<i>19</i>
<i>Virtual Machines & Containers</i>	<i>20</i>
<i>Bibliography.....</i>	<i>23</i>

Client-side & Server-side Scripting

When creating a dynamic HTML page, the source code responsible for changing the web page must either be executed on the client side or the server side [1]. The client side usually refers to the web browser and the server side refers to the web server. Figure 1 below highlights the key differences between both types of scripting.

CLIENT SIDE SCRIPTING VERSUS SERVER SIDE SCRIPTING	
CLIENT SIDE SCRIPTING	SERVER SIDE SCRIPTING
A technique used in web development that involves using scripts that runs on the client machine's browser	A technique used in web development that involves using scripts on the webserver to produce a response that is customized for each client's request to the website
Executed in the client side or the web browser	Executed in the back end or the web server
HTML, CSS, and JavaScript are used	PHP, Python, Java, Ruby, and ASP.NET are used
Does not provide much security for the data	Provides more security for the data
	Visit www.PEDIAA.com

Figure 1: Differences Between Client and Server-side Scripting [2]

The process of client side scripting

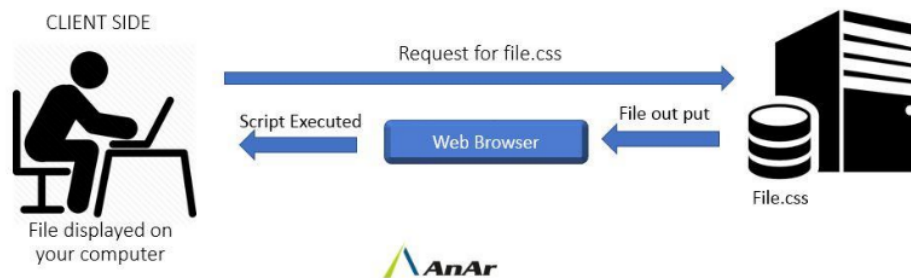


Figure 2: The Process of Client-side Scripting [3]

As shown in figure 2 above, the client-side scripting process begins with a file request (usually HTTP) from the client to the web server (e.g. triggered by clicking a button on the web page) [4]. A client-side script can be executed without communicating with a server, however a server must be involved if the client requires external information e.g. from a database [5]. In this case, the web server would be linked to a database server. The server retrieves the file from the database API, then returns it to the web browser. The browser runs the script before displaying the updated web page on the client's computer. The source code is accessible by the client [1], making it a less favourable approach for applications dealing with sensitive and/or private data.

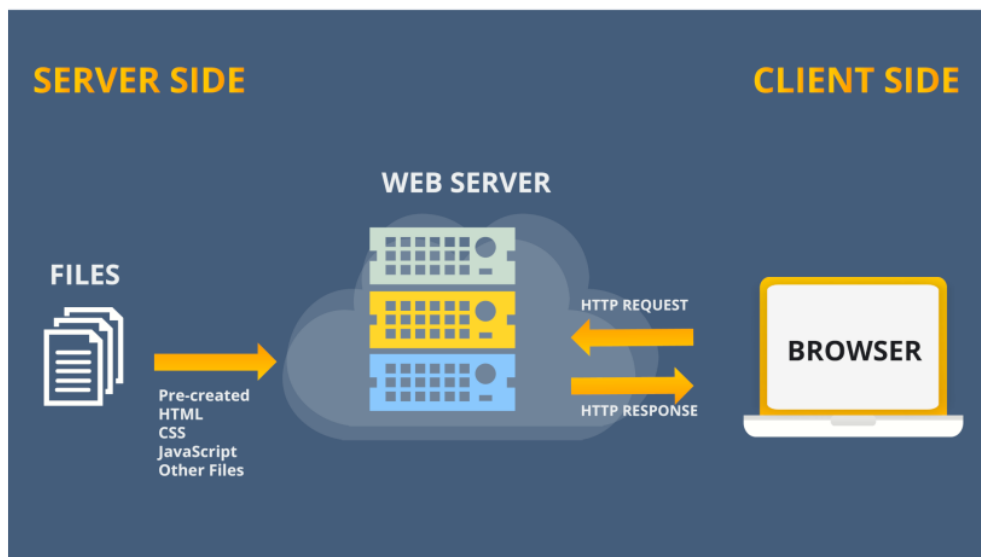


Figure 3: Diagram Demonstrating Server-side Scripting [6]

The server-side scripting process begins in the same way, where a HTTP request is sent from the client to the web server (shown in figure 3). The web server retrieves the requested file from the database then inserts the information into a script. The

server executes the script then returns the results back to the client. This keeps the source code hidden from the client [1], making it more suitable for applications dealing with sensitive data.

Languages

Client-side and server-side scripts are usually written in different programming languages (with the exception of JavaScript, which can be used on both sides). Client-side scripts are written in HTML, CSS and JavaScript, whereas server-side scripts may be written in a number of languages, the most common ones being PHP, Python, Ruby, Java, GoLang, C#, and JavaScript (NodeJS) [7].

Required Run-time Environment

When using JavaScript end-to-end, the run-time environment differs depending on whether you're executing server-side or client-side scripts. To be able to display HTML elements on a user's computer, the JavaScript code must be executed within the browser [1] (see figure 4 as an example). This script would also include HTML & CSS, which are responsible for the structure and styling of the web page [1] [8]. The JavaScript code would be responsible for handling DOM (Document Object Model) events such as button clicks etc. It also manages asynchronous HTTP requests and responses to and from the server.

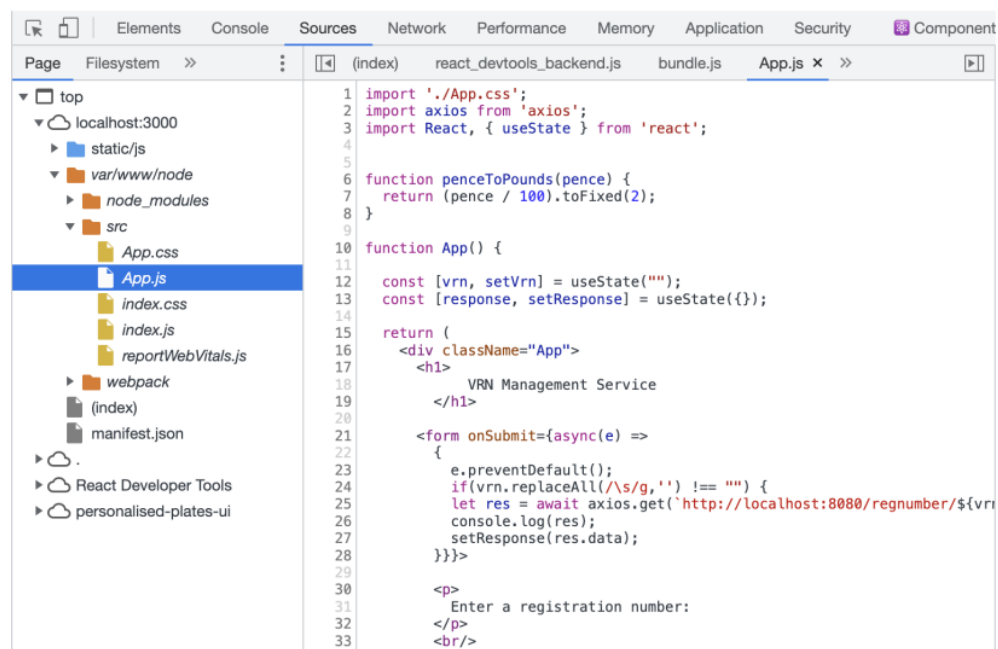


Figure 4: Example of a Client-Side JavaScript React Application being Served within a Chrome Browser (Shows Sources tab on Chrome Developer Tools Interpreting the Example Vehicle Management Service Code)

On the server-side, the JavaScript code is executed using the NodeJS run-time environment. NodeJS was created with the purpose of providing an alternative runtime environment to a browser [9]. Server-side code can vary depending on the application. These scripts can range from managing the routing of URL paths on a webpage to defining RESTful API endpoints and data transfer objects communicated over HTTP. It is also at this server-side layer that database connectivity is managed (see figure 5 below).

```
//vehicle-service.js
service.get('/vehicles', async(req, res) => {
  if(typeof req.query.vrn === "string"){
    await db.SelectByVrn(req.query.vrn, res);
  } else {
    await db.Select(res);
  }
})

//pgdbvehicle.js
SelectByVrn = async (vrn, res) =>
{
  let client = GetClient();
  console.log('Looking for all the vehicles with ${vrn}');
  try
  {
    await client.connect();

    let query = format('SELECT * FROM "Vehicle" WHERE "RegistrationNumber"=%L', vrn);
    let results = await client.query(query);

    if(results.rows.length > 0)
      console.log('Found ${results.rows.length} records');
    else
      console.log('Records not found for ${vrn}');

    res.send(results.rows);
  }
  catch(error)
  {
    console.log(error);
    res.status(403).send(error);
  }
  finally
  {
    await client.end();
  }
};
```

Figure 5: Example of a Server-Side Script Defining a Vehicles API which has a GET vehicles Endpoint that Queries a Postgres Database

Database Connectivity

Database interactions are defined within the server-side script to avoid SQL injection attacks [10]. It does this by performing input validation on the data submitted from the client-side before executing a query in the database. Client-side scripts do not usually include any references to the database. To render information from the database onto the web page, the client-side script must establish a connection with the web server, to allow the exchange of HTTP requests/responses. The web server would then exchange requests/responses with the database server and return the data to the client [4].

Browser

Server-side scripting is browser-independent and returns simple HTML, which can be executed by all browsers [1]. Similarly, the HTTP responses are sent in JSON, which all browsers are capable of interpreting. The disadvantage of server-side scripting is that the web server must consume invaluable resources to parse every page before returning the results to the browser, which can slow down the web page.

Client-side scripting is dependent on the client's browser, meaning that some browsers and/or different versions may be unable to run certain scripts. Despite this, the benefit is that each browser uses its built-in resources to run compatible scripts, thus easing the burden on the server [1].

Customization

Another feature of server-side scripting is that the content of the web page can be customized for individual users, arguably creating better user experience [4]. This is done using 'sessions', where the server stores individual user information and sends specific responses based on that data. Server-side programming can be used to store account information such as username and password, so the user doesn't have to enter their credentials every time they want to log into a website. It can also be used to manage user access to specific content e.g. logging into a specific social media account and being shown content relevant to that profile. Servers are also capable of sending notifications to the user via the browser, email, text etc.

Applications that utilise client-side scripting cannot store individual user information, therefore the web page displayed will contain the same information regardless of the user.

Lastly, client-side scripts cannot access local files, directories and/or databases therefore if an application requires access to local data, server-side scripting must be used [1].

Node JS

NodeJS is a popular open-source, cross-platform run-time environment and library used to run JavaScript applications on the server-side, outside of a web browser [4]. It uses Google's V8 high-performance JavaScript engine [11] which is also used by Google Chrome and other browsers.

As opposed to traditional server-side applications, NodeJS uses asynchronous, non-blocking, event-driven I/O operations (such as reading and writing data, making HTTP requests and communicating with databases) [12]. It uses a single thread with an event loop to serve multiple requests simultaneously. This means that the program does not have to wait for the completion of an operation before continuing onto the next one. It is used to generate dynamic web pages and perform CRUD operations for the server.

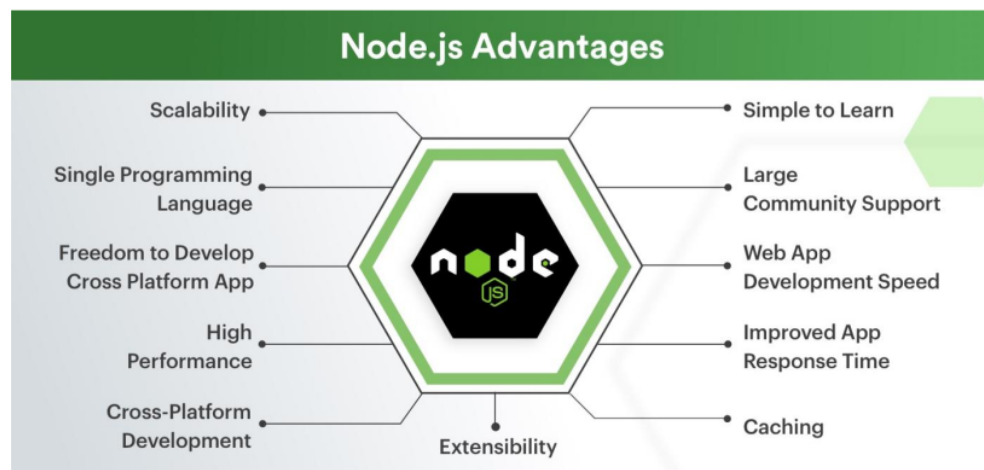


Figure 6: Advantages of NodeJS

Some of the advantages of using NodeJS demonstrated in figure 6 include:

1. Scalability

NodeJS' use of non-blocking I/O operations makes the process more fast, efficient, highly performant and scalable when compared with other scripting languages [12]. Because it does not demand much memory and uses a single thread to handle multiple requests, it is vertically scalable and can be used to create dynamic, high-traffic applications [13].

2. High Performance

In comparison with other server-side scripting languages, NodeJS is usually faster at handling requests and more performant overall. It uses a Node wrapper around the V8 engine, both of which are written in C++, making it faster than Python, Ruby and Perl [14].

3. Single Programming Language

Because NodeJS uses JavaScript end-to-end, it means that the entire application can be deployed quickly and easily as developers don't have to spend time learning another language. Using a single language across the application means it's easier for a developer to wire-up the communication between the front and back end. It also means that the entire program could potentially be written by a single JavaScript developer, eliminating the need for a company to hire both front-end and back-end engineers.

4. Simple to Learn

NodeJS is written in JavaScript which is simple to understand, and is one of the most popular programming languages [15].

5. Cross Platform Development

Because JavaScript is an interpreted language, it does not need to be compiled before it is executed. This means the application can be fragmented into small packages of code (known as modules) to be executed using most operating systems e.g. Windows, iOS, Android etc.

6. NPM (Node Package Manager)

NodeJS comes with its own package manager known as NPM, making it fast and simple to install packages from the internet [13]. Because the developer does not need to rely on third party tools to install additional libraries/frameworks, it saves them time in the long run and the application can be deployed more quickly.

7. Large Community Support

Developers can create and share modules with each other, which encourages community growth and well-developed support networks [15]. Its growing popularity has led leading tech giants such as Microsoft and Google to develop and share solutions to common problems within the community [16].

8. Caching

NodeJS allows caching of data to a user's computer, which means a repeated request for that resource does not require the server to communicate with the database again, thus decreasing the response time [15].

Despite the many advantages of using NodeJS, it also has its limitations. Some of which include:

1. Unstable API

The NodeJS API changes regularly, making it unstable [16]. This often results in applications which are not backwards-compatible [14]. For this reason, developers must be particularly vigilant when updating applications to use newer versions of NodeJS.

2. Limited Tooling

Developers using NodeJS often depend on third-party libraries/modules to be able to perform actions that are not included in the core NodeJS library [16]. These modules are often useful; however many are not reviewed adequately, are not very well documented, lose active support over time and/or are inferior in quality.

3. Asynchronous

As NodeJS uses the asynchronous programming model, it utilises callback functions to maintain concurrency [15]. This can trigger a race condition, which is an undesirable situation that occurs when two or more processes try to access the same resource simultaneously e.g. a record in a database, and at least one of them is attempting to modify the resource [16]. Race conditions are notoriously difficult to manage and can take developers a significant amount of time to identify and fix.

Overall, NodeJS is a useful tool used to deliver a dynamic, real-time, end-to-end application quickly and easily, with very few limitations.

AJAX

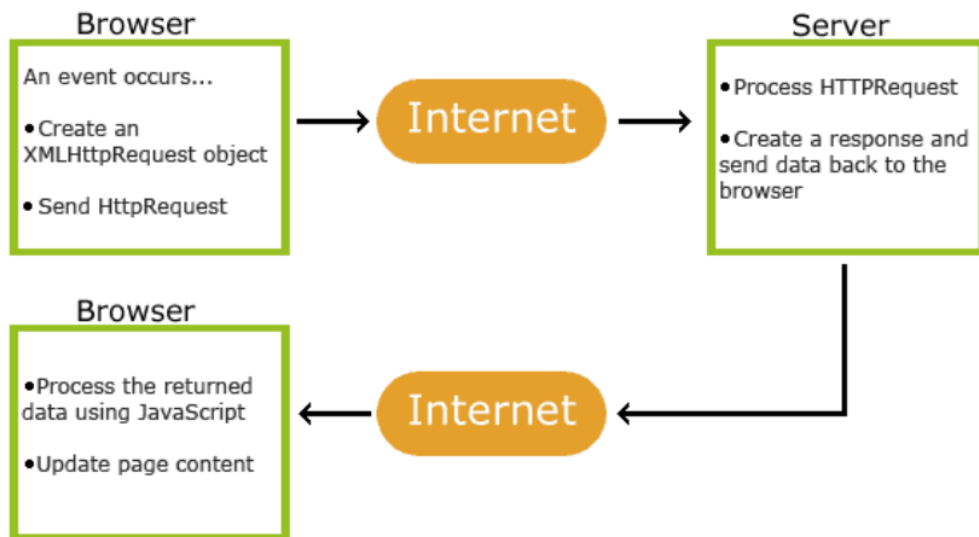


Figure 7: How AJAX Works [17]

In the early 90s, if any updates were made to a web page, the entire page would need to be refreshed before those changes became visible to the user. This meant that even a minor change would require the entire content of the page to be reloaded from the server, placing additional load on the server and making the process inefficient due to high bandwidth consumption.

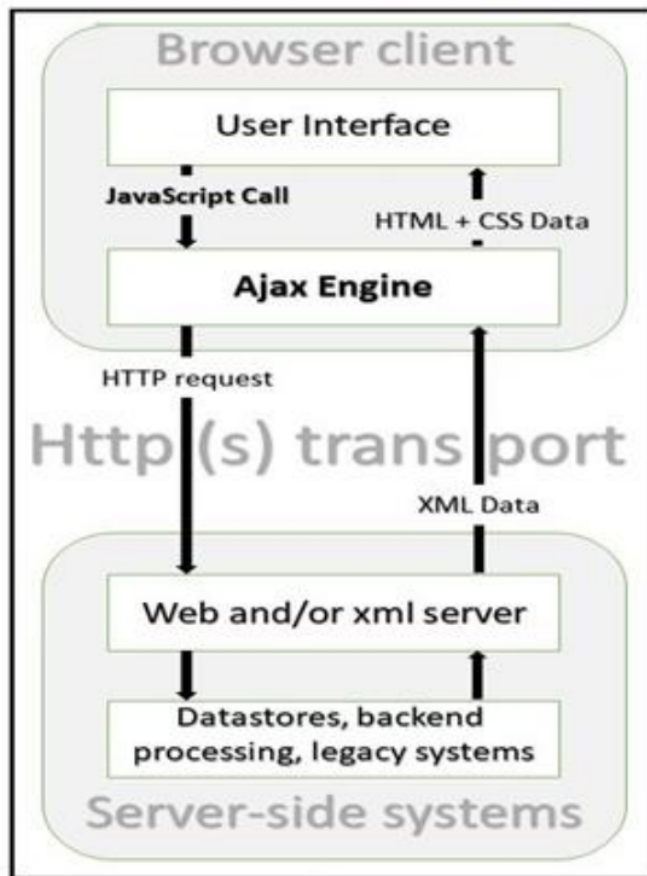


fig : AJAX Web Application Model

Figure 8: AJAX Web Application Model

In 2005, James Garrett developed AJAX (Asynchronous JavaScript and XML) [18], which is a client-side SDK (software development kit) that allows web pages to be rendered asynchronously [18]. This allows a user to continue interacting with the page whilst requests and responses are handled by the web server in the background (see figure 8 for a breakdown of the structure). It uses HTML, CSS, XML/JSON and JavaScript to create fast, efficient applications [19] and makes it possible to modify sections of a web page without having to reload the entire page. Its use of JavaScript means that it is easy to understand and debug and is compatible with most browsers [20]. As demonstrated in figure 7, AJAX uses XMLHttpRequest objects to communicate with a server [21].

Using AJAX gives developers the ability to create a web page that auto-reloads new data, without needing to submit the web page to the server to be processed first [18]. It utilises a polling mechanism to retrieve regular updates from a data source that is in a constant state of flux, which makes the process of communication between the client and server more efficient. Together, these processes save bandwidth and improve performance, responsiveness and user satisfaction [19]. AJAX also aids the

secure transmission of data by using a one-one interface between the client and server, making it not only more efficient, but more secure than traditional web applications.

AJAX is open-source and allows developers to build additional functionality into their applications which can be shared with the wider community. This gives developers the flexibility to incorporate the latest functionality/features into their applications and keep them up-to-date. It has encouraged the creation of ground-breaking technologies such as Web Sockets (a persistent connection between a client and a server), which has led to the creation of modules such as push notifications, peer-to-peer communication and real-time chat applications [22] [23].

React

React (or ReactJS) is a JavaScript library that utilises AJAX to build a single page application (SPA) [24]. It relies on the segregation of different components, which are eventually combined to form the application. For example, a component file called 'heading' is created, which will include all the logic specific to the functionality of that component. The logic is abstracted away from all other components, which will exist in their own separate files. Once all components have been created, their tag names can be referenced as part of the central application file, making the structure of the web page much easier to read.

React uses conditional rendering, which is a technique used to conditionally draw HTML markup within the DOM [24]. It achieves this using JSX (JavaScript HTML) which allows you to write typical JavaScript if statements to decide which components appear on the page [25].

React uses a useState hook, which enables the state of each component to be retained in memory even if that component is redrawn [24]. By default, the state is not retained if the web page is reloaded. The useState hook also provides a setter function to update the state variable and triggers re-rendering of the component.

React also provides the ability to pass a function as an event handler to perform updates based on DOM events. For example, you can call a useState hook based on an onClick event to update the state variable. This variable can then be used for conditional rendering; for instance, changing the button's colour after being clicked [26].

Figures 9 to 15 demonstrate how AJAX and React are used to build webpages which can send HTTP requests to APIs and render based on the response.

VRN Management Service

Enter a registration number:

Search

No results for VRN CV68MOO

Figure 9: Example VRN Management Service Webpage using React

App

props

new entry: ""

hooks

1 State: "CV68MOO"

2 State: {}

rendered by

createRoot()

react-dom@18.2.0

source

src/index.js:10

Figure 10: Chrome React Plugin Illustrating Registration Number Field Stored as a State Variable through a useState Hook

```

import './App.css';
import axios from 'axios';
import React, { useState } from 'react';

function penceToPounds(pence) {
  return (pence / 100).toFixed(2);
}

function App() {

  const [vrn, setVrn] = useState("");
  const [response, setResponse] = useState({});

  return (
    <div className="App">
      <h1>
        VRN Management Service
      </h1>

      <form onSubmit={async(e) =>
        {
          e.preventDefault();
          if(vrn.replaceAll(/s/g, "") !== "") {
            let res = await axios.get('http://localhost:8080/regnumber/${vrn}');
            console.log(res);
            setResponse(res.data);
          }}>

        <p>
          Enter a registration number:
        </p>
        <br/>
        <input id="vrn-textbox" name="vrn-textbox" type="text" value={vrn} onChange={(e) =>
          {
            setVrn(e.target.value)
          }}/>
        <br/><br/>

        <body>
          <button id="search-button" name="Search button" value="Search">Search</button>

          <br/><br/>
          { /* if 'response.pricInPence is true (i.e. not null), then execute everything after '&&' */ }
          {response.pricInPence &&
            <table id="available-plates" name="Available plates"> Available plates
              <tr>
                <th>
                  VRN: {response.vehicleRegistrationNumber}
                </th>

```

```

    <th>
    Price: £{penceToPounds(response.pricelnPence)}
    </th>

    <th>
    { /* need to create an onsubmit for the buy now button */ }
    <button id="buy-now-button" name="Buy now button">Buy Now</button>

    </th>
  </tr>
</table>
{!response.pricelnPence && <p>No results for VRN {vm}</p>}

</body>
</form>

</div>
);
}

export default App;

```

Figure 11: Example React Application Implementing an AJAX Request using the Axios Wrapper Library

VRN Management Service

Enter a registration number:

Available plates

VRN: CV68MOO Price: £3000.00

Figure 12: Example Showing the Result of a User Clicking the Search Button (Triggers an onSubmit Event which Sends an AJAX request via HTTP to the Web Server. The Page then Updates to Reflect the Response)

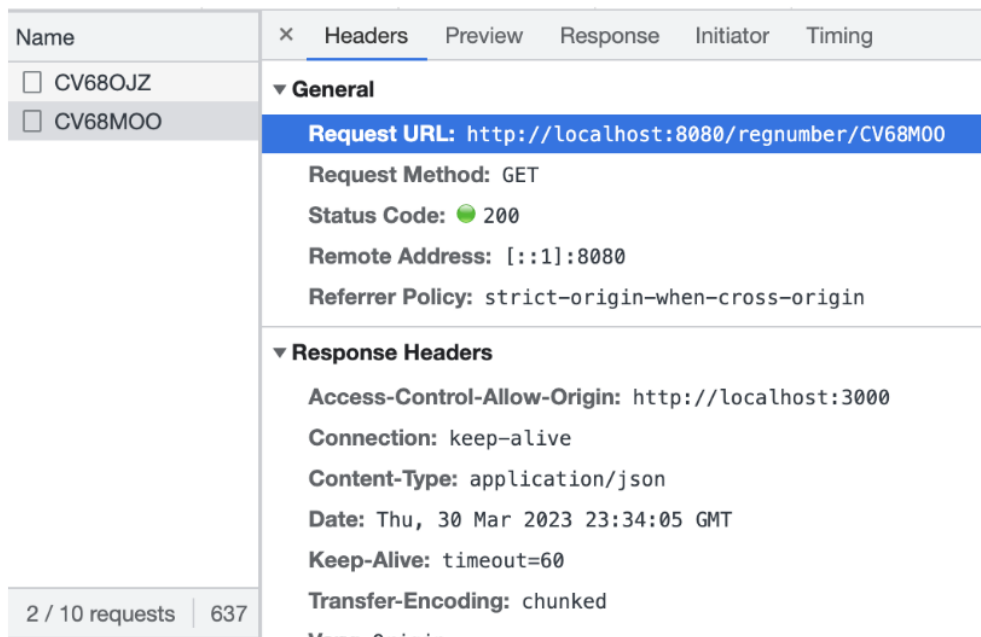


Figure 13: Screenshot of the Chrome Developer Network Tool (Shows the HTTP GET Request which was Invoked after the onSubmit Function was Called)

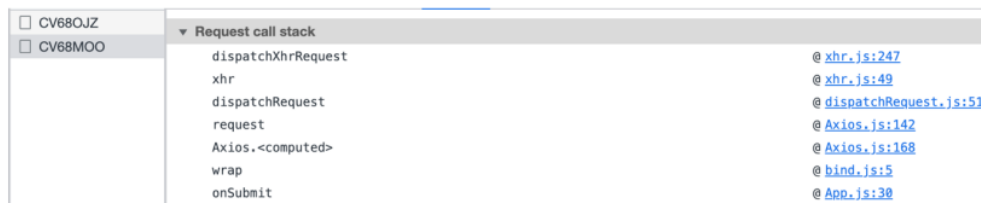
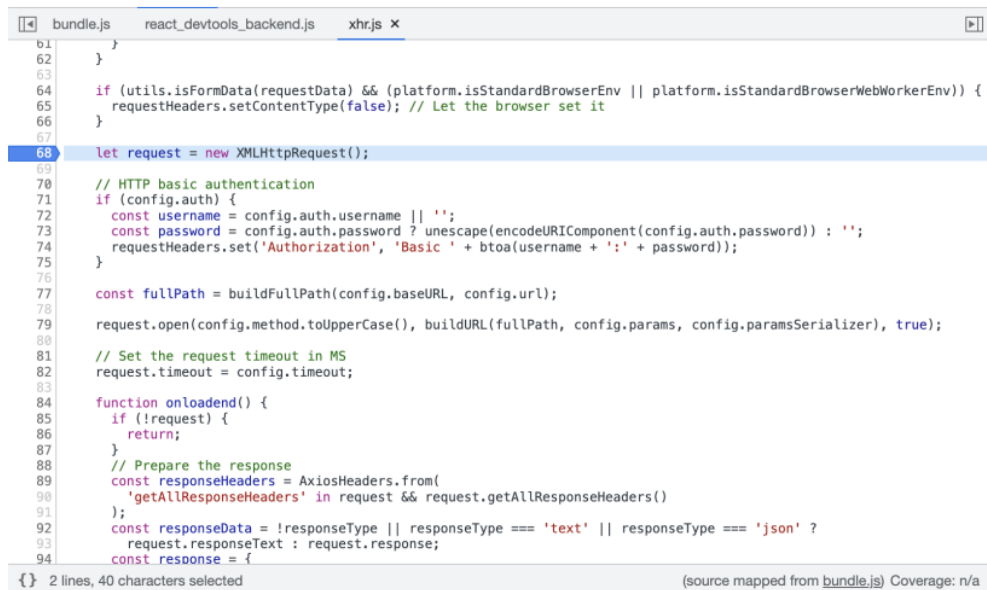


Figure 14: Screenshot of the JavaScript Call Stack within the Initiator Tab which Shows Axios Using the XHR Library



The screenshot shows a code editor with three tabs: `bundle.js`, `react_devtools_backend.js`, and `xhr.js`. The `xhr.js` tab is active, displaying JavaScript code. Line 68 is highlighted with a blue background, showing the initialization of an `XMLHttpRequest` object: `let request = new XMLHttpRequest();`. The code includes various configuration steps such as setting content type, handling authentication, and setting timeouts. At the bottom, a status bar indicates "2 lines, 40 characters selected" and "(source mapped from bundle.js) Coverage: n/a".

```
61 }
62 }
63
64 if (utils.isFormData(requestData) && (platform.isStandardBrowserEnv || platform.isStandardBrowserWebWorkerEnv)) {
65   requestHeaders.setContentType(false); // Let the browser set it
66 }
67
68 let request = new XMLHttpRequest();
69
70 // HTTP basic authentication
71 if (config.auth) {
72   const username = config.auth.username || '';
73   const password = config.auth.password ? unescape(encodeURIComponent(config.auth.password)) : '';
74   requestHeaders.set('Authorization', 'Basic ' + btoa(username + ':' + password));
75 }
76
77 const fullPath = buildFullPath(config.baseURL, config.url);
78
79 request.open(config.method.toUpperCase(), buildURL(fullPath, config.params, config.paramsSerializer), true);
80
81 // Set the request timeout in MS
82 request.timeout = config.timeout;
83
84 function onloadend() {
85   if (!request) {
86     return;
87   }
88   // Prepare the response
89   const responseHeaders = AxiosHeaders.from(
90     'getAllResponseHeaders' in request && request.getAllResponseHeaders()
91   );
92   const responseData = !responseType || responseType === 'text' || responseType === 'json' ?
93     request.responseText : request.response;
94   const response = {
```

Figure 15: Screenshot Showing the Interpreted Bundled JS XHR Library used by Axios (Here is Where the XMLHttpRequest Object is Initialized)

JSON Web Tokens

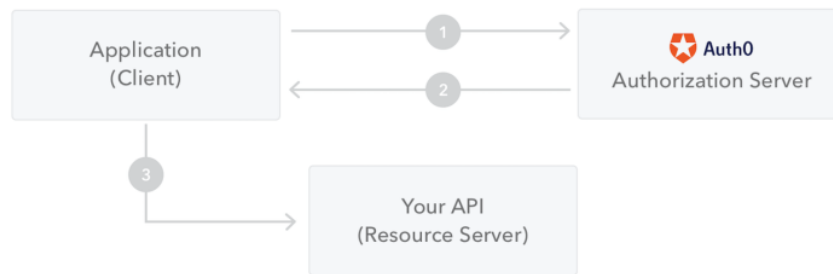


Figure 16: Diagram Demonstrating how a JWT is Obtained and Used [27]

JSON web tokens (JWTs) are used to securely transmit information between two parties in HTML and HTTP environments [28]. To obtain a web token, the client must first be successfully authorized by an authorization server (e.g. by submitting a valid username & password). The authorization server then returns a web token that can be used by the client as a credential to gain access to the resource server (see figure 16).

Because JWTs can be signed, it provides certainty that the clients are who they say they are and assures that the contents have not been modified [29]. As a result, they prevent CSRF (Cross-site Request Forgery) attacks (unauthorised requests sent from the user, but are trusted by the application). However, it's important to note that unless a JWT is encrypted, its contents are visible to other parties (although they cannot change it).

Virtual Machines & Containers

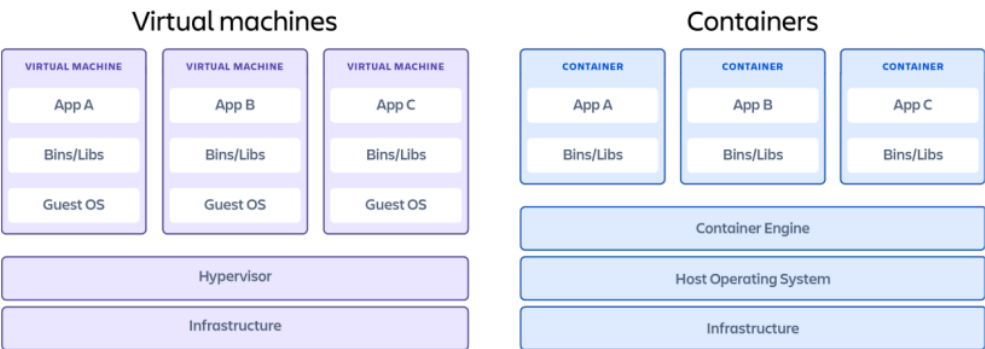


Figure 17: Structure of Virtual Machines and Containers [30]

A virtual machine (VM) is essentially a digital version of a physical computer [31] that runs independently within the host system, and therefore requires its own operating system and software licences. A VM utilises a hypervisor, which is a computer hardware or software that enables a system to host multiple VMs.

A container is a software package that contains the code, libraries and dependencies of an application, which is virtualized above the operating system (OS) level. In contrast to virtual machines, containers are restricted to run using the same operating system as the host [32] (see figure 17 for structure of VMs & containers). Table 1 below highlights the fundamental differences between both.

Table 1: Differences between VMs and Containers

Virtual Machine	Container
Virtualises physical hardware using a hypervisor (a lightweight software layer) [33]	Virtualises the operating system [33]
Uses its own OS (independent of the host's OS) which uses more resources, resulting in poorer performance	Runs using the host's OS without requirement for additional software, resulting in better performance [34]
Cannot share applications with neighbours	Can share applications with neighbouring containers
Starts up slowly because it must first boot up the OS [35]	Starts up quickly as the OS is already running [35]
Heavyweight as it demands a lot of memory space for containment of the OS, CPU, disk, networking devices & additional software packages	Lightweight as it contains only high-level software above the OS level [34]
Runs independently from the host's system and neighbouring VMs therefore if it gets hacked, the problem is isolated to that VM	Runs from the host's system so if hacked, containers are also at risk of exploitation [34]
The number of VMs that can run on a system is limited by the amount of storage space available on the host system. Purchasing additional storage/using external Cloud services can be costly [36] [32]	Hundreds (if not thousands) of containers can be run on a single host system as they do not demand much storage space [34] which makes modelling networks cheap & easy [32]
Only 1 VM can be started at a time 	Several containers can be started simultaneously
Cannot be version-controlled	Uses images which can be version-controlled
Not very portable as the VM becomes bound to a specific piece of hardware which can be difficult to set up. Some systems don't support virtualisation	Portable as the image can be shared with the wider community (via a Docker registry e.g. DockerHub) and deployed from different host environments, maintaining behavioural consistency [33] [34]

The expansion of the internet in the last 25 years has led to the creation of thousands of programming languages and frameworks, and an increasing demand for deployment to a large number of servers and environments [34].

In the same way that real-life shipping containers are used to distribute huge amounts of goods across the globe, Solomon Hykes wanted to be able to transfer software between different machines over the internet. In 2013, he created an open-source service called Docker, which allowed software packages to be housed and shipped inside their own portable containers [37].

As a result, developers started using Docker as it was a faster and easier way to deploy software across a growing number of environments, without having to worry about interaction between containers [32].

Docker has become a lucrative tool for tech companies as it cuts down on the amount of time developers spend manually provisioning different environments and configuring different host systems. In comparison with VMs, Docker containers can be run reliably and consistently in multiple different environments (as long as the system uses the same OS) [32]. Docker also provides the ability for developers to segregate and test different parts of a project in isolation [35]. These features therefore allow companies to deliver software quickly and efficiently.

Containers offer more and/or better features than VMs, they are far less complex to set up and offer much better performance (as demonstrated in table 1), which has driven an increasing number of developers to adopt Docker. Docker is a much cheaper alternative to VMs as it is open-source and consumes less memory, meaning less demand to purchase additional storage or utilise expensive cloud environments [38]. Docker is also more cost effective as it doesn't require additional software subscriptions if the host system is already subscribed.

Word count: 3,098

Bibliography

- [1] Anonymous, "Scripting: Server-side vs. client-side - ProQuest," *Active Server Developer's Journal*, vol. 2, no. 9, pp. 15–16, 1998, Accessed: Mar. 31, 2023. [Online]. Available: <https://www.proquest.com/docview/206870773/abstract/935860513CDC44CAPQ/1?accountid=130472>
- [2] Lithmee, *Difference Between Client Side Scripting and Server Side Scripting*, Oct. 06, 2018. <https://pediaa.com/difference-between-client-side-scripting-and-server-side-scripting/>
- [3] AnAr, *How Does Client-Side Scripting Improve Web Application Performance?*, Mar. 02, 2017. <https://anarsolutions.com/client-side-scripting/>
- [4] MDN contributors, *Introduction to the server side*, Mar. 18, 2019. https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction
- [5] H. Yu and Z. Lu, "How to Choose Server-Side Scripting Languages & Database Servers in Web Mining," *2008 International Symposium on Knowledge Acquisition and Modeling*, Dec. 2008, doi: <https://doi.org/10.1109/kam.2008.83>.
- [6] EBSIntegrator, *Basics - a user perspective on server-side scripting*, Jul. 12, 2020. <https://ebs-integrator.com/blog/basics-server-side-scripting/>
- [7] T. Tran, *Top 7 Server-side Scripting Languages - Orient Software*, Dec. 01, 2021. <https://www.orientsoftware.com/blog/server-side-scripting-languages/>
- [8] R. Johansson, *Developing With Web Standards – Recommendations and best practices*, Nov. 01, 2008. https://www.456bereastreet.com/lab/developing_with_web_standards/
- [9] D. Herron, *Node.js Web Development*, 5th ed. Packt Publishing Ltd, 2020.
- [10] K. Kamtuo and C. Soomlek, "Machine Learning for SQL injection prevention on server-side scripting," *2016 International Computer Science and Engineering Conference (ICSEC)*, pp. 1–6, Dec. 2016, doi: <https://doi.org/10.1109/ICSEC.2016.7859950>.
- [11] V8.dev, *Documentation*. <https://v8.dev/docs>
- [12] NodeJS, *About Node.js®*. <https://nodejs.org/en/about>
- [13] M. Satheesh, B. J. D'mello, and J. Krol, *Web Development with MongoDB and NodeJS - Second Edition*, 2nd ed. Packt Publishing, 2015.
- [14] M. Kumar, *Using Node.js; What Are The Pros And Cons*, Apr. 07, 2021. <https://www.linkedin.com/pulse/using-nodejs-what-pros-cons-mukesh-sharan>

- [15] B. Syed, *Beginning Node.js*. United States: Apress, 2014.
- [16] S. Tondon, *The Pros and Cons of Node.js Web App Development: A Detailed Look*, Mar. 01, 2022. <https://javascript.plainenglish.io/the-pros-and-cons-of-node-js-web-app-development-a-detailed-look-c91a22f013c>
- [17] W3Schools, *AJAX Introduction*. https://www.w3schools.com/js/js_ajax_intro.asp
- [18] A. Zaidman, N. Matthijssen, M.-A. Storey, and A. van Deursen, "Understanding Ajax applications by connecting client and server-side execution traces," *Empirical Software Engineering*, vol. 18, no. 2, pp. 181–218, Mar. 2012, doi: <https://doi.org/10.1007/s10664-012-9200-5>.
- [19] A. Mesbah and A. van Deursen, "Migrating Multi-page Web Applications to Single-page AJAX Interfaces," *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, 2007, doi: <https://doi.org/10.1109/csmr.2007.33>.
- [20] E. J. Bruno, "Ajax: Asynchronous JavaScript and XML," *Dr.Dobb's Journal*, vol. 31, no. 2, pp. 32–35, Feb. 2006, Available: <https://www.proquest.com/docview/202704452/abstract/1386D089F6764548PQ/1?accountid=130472>
- [21] A. White, "AJAX accelerates Web applications," *Network World*, vol. 23, no. 2, p. 33, Jan. 2006, Available: <https://www.proquest.com/docview/215973671/abstract/730682B84AAB49BFPQ/1?accountid=130472>
- [22] A. Lombardi, *WebSocket*. CA: O'Reilly Media, Inc., 2015.
- [23] S. Bhattacharya, *What is AJAX and how it is an important tool for software professionals*, Mar. 16, 2022. <https://content.techgig.com/upskilling-at-techgig/what-is-ajax-and-how-it-is-an-important-tool-for-software-professionals/articleshow/90264533.cms>
- [24] A. Fedosejev, *React.js Essentials*. Birmingham: Packt Publishing Ltd, 2015.
- [25] React.dev, *Quick Start*. <https://react.dev/learn>
- [26] React.dev, *Quick Start*. <https://react.dev/learn/responding-to-events>
- [27] JWT, *Introduction to JSON Web Tokens*. <https://jwt.io/introduction>
- [28] JWT, *JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. JWT.IO allows you to decode, verify and generate JWT*. <https://jwt.io/>
- [29] Auth0, *JSON Web Tokens*. <https://auth0.com/docs/secure/tokens/json-web-tokens>

- [30] I. Buchanan, *Containers vs Virtual Machines*.
<https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms>
- [31] AWS, *What's The Difference Between Containers And Virtual Machines?*
<https://aws.amazon.com/compare/the-difference-between-containers-and-virtual-machines/#:~:text=A%20container%20is%20a%20software>
- [32] I. Miell and A. Sayers, *Docker in Practice*, 2nd ed. New York, United States: Manning Publications, 2019.
- [33] IBM Cloud Team, *Containers vs. VMs: What's the Difference?*, Apr. 09, 2021.
<https://www.ibm.com/cloud/blog/containers-vs-vms>
- [34] T. Combe, A. Martin, and R. Di Pietro, "To Docker or Not to Docker: A Security Perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, Sep. 2016, doi: <https://doi.org/10.1109/mcc.2016.100>.
- [35] B. I. Ismail *et al.*, "Evaluation of Docker as Edge computing platform," *2015 IEEE Conference on Open Systems (ICOS)*, Aug. 2015, doi: <https://doi.org/10.1109/icos.2015.7377291>.
- [36] P. Tyagi, D. Kapil, R. C. Joshi, and E. S. Pilli, "Virtual Machine Portability - A Novel Approach," *2013 International Conference on Cloud Computing and Big Data*, Dec. 2013, doi: <https://doi.org/10.1109/cloudcom-asia.2013.108>.
- [37] S. Agarwal, S. Jain, and A. Kumar, "GUI Docker Implementation: Run Common Graphics User Applications Inside Docker Container," *2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Dec. 2021, doi: <https://doi.org/10.1109/smart52563.2021.9676316>.
- [38] Aeris, *DOCKER THE COMPACT CONTAINER PLATFORM CLOUD AND CONTAINERIZATION TECHNOLOGY*. <https://www.aeris-consulting.com/wp-content/uploads/2022/04/Docker.pdf>

FINAL GRADE

91 / 100

GENERAL COMMENTS

Instructor

CSvsSS (25):23

node (25):24

AJAX/React/JWT (25):23

Docker (25): 21

Feedback:

=====

You could have talked a little more about the benefits of offloading processing from the server to the client and security issues.

Very good analysis of node.js benefits and drawbacks.

Nice discussion of AJAX/React with examples.

The docker/VM discussion was good but the tables should have been incorporated into the discussion.

PAGE 1

PAGE 2

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

PAGE 20

PAGE 21



Comment 1

Not sure what this is trying to say? We can setup scaling groups and autoscale VMs as required.

PAGE 22

PAGE 23

PAGE 24

PAGE 25
