Research Assignment

Assignment 1

Class

Client-Server Web Development

Tutor

Doug MacKenzie

Date Last Revised

Tuesday, 11 April 2023

Table of Contents

A. Philosophy of Client-Server Computing

1. Client-Server Computing

The client-server architecture, otherwise known as client-server computing, began to surface in the early 1980s within the United States as computing environments evolved from immense mainframes to distributed processing utilizing numerous PCs or workstations (Information Processing Society of Japan, n.d.).

Organizations swiftly embraced the client-server architecture, which became utilized heavily in their business communication and automation infrastructure (Information Processing Society of Japan, n.d.). Succeeding this, the client-server architecture has had to evolve immensely to keep up with requirements for client-server computing.

Client-server computing is a shared application framework, in which the application is divided into tasks between servers and clients. A client-server application may occupy the same computer system or transmit data through the internet or a computer network (HEAVY.AI, 2022).

The client depends on dispatching a request to an additional program in an effort to acquire a service created by a server. The server creates a single or multiple programs that allocate resources with and disperse work among clients (HEAVY.AI, 2022).

The client-server architecture transmits data in a request and response communication (messaging) pattern. As this is the case, the client-server model must conform to a ubiquitous communications protocol, which formally dictates the language, dialog pattern, and rules to be utilized. Dependent on the development team, client-server communication frequently conforms to the TCP/IP protocol (HEAVY.AI, 2022).

The client-server model is comprised of four distinct architectures for various situations. These architectures are one-tier, two-tier, three-tier, and n-tier (HEAVY.AI, 2022).

The one-tier architecture is composed of an elementary program operating on a single computer system without the necessity to access a computer network. User requests do not handle any network protocols, consequently the computer network is disburdened of the extra internet traffic, and the code is simple (HEAVY.AI, 2022).

The architecture for tier-two consists of the server, and client, alongside the protocol that connects the two tiers together. The GUI (graphical user interface) code inhabits the computer system in which the site is being accessed (the client host), while the business logic occupies the server host. The client-server graphical user interface can be written in such high-level languages as C++, Java, or PHP (HEAVY.AI, 2022).

The three-tier architecture is comprised of a data tier, which is composed of a data server that stores the user's information, an application tier, which is the business logic layer that carries out comprehensive computing, and the presentation tier, which is the graphical user interface layer (HEAVY.AI, 2022).

The architecture for the n-tier dissects an application into rational layers, which handle dependencies, physical tiers, divide responsibilities, refines scalability, and worsens latency from the extra network communication. The n-tier architecture can operate with the closed-layered or open layered principles. The closed-layered principles state that a layer may only converse with the preceding layer, whereas the open-layered principles state that a layer may communicate with any layers below it (HEAVY.AI, 2022).

2. Client-Server Architectures

The client-server architecture provides two client architectures to choose from, fat client, and thin client.

I. Fat Client-Server Architecture

A traditional client otherwise known as the fat client, is a configuration of client-server architecture where a networked computer system stores the lion share of resources for a client-server application locally, instead of the client-server application distributing the resources over a network to client devices (Gillis, 2020).

A client device could be, for example, cell phones, as they have their own display, storage mechanisms, application software, and more essential local resources, which are essential elements for a fat client-based application to operate successfully (Gillis, 2020).

Fat clients are linked to a server over a network connection. However, fat-clients do not require maintained internet connections to function. As a result, fat clients do not necessitate the need to occupy any server computing resources. The majority of resources will be accessible on the client, so it may operate independently. However, a temporary internet connection is required in order to download client-server applications to a computer system, alongside data and updates (Gillis, 2020).

Fat clients could shine within environments where the predominant server has restricted computing and storage potential, or experiences excessive network traffic, as well as in work-from-home environments (Gillis, 2020).

i. Advantages of Fat Client Architecture

The fat client architecture provides numerous benefits, including but not limited to, working offline, less server requirements, improved flexibility, and increased server capacity.

Fat clients can work offline because they do not require a constant internet connection. Fat clients normally have the software and hardware requirements required to operate provided to them by the client host, primarily without the prerequisite to be connected to a central server, an example of this is Microsoft Office (Gillis, 2020).

The fat client architecture does not require tremendous server requirements. The reason for this, is because the fat client performs most of the application processing on the client device, this allows for simple economical servers (Gillis, 2020).

Fat clients improve flexibility by having client devices operate off their own local resources, such as the user interface, storage mechanism, processing power, and operating system. Client-server applications should be able to operate from any physical location on a client device, given the devices are able to maintain a momentary connection to a central server to download any required data (Gillis, 2020).

The utilization of fat clients usually signifies increased server capacity is available. As a result of less requirements that a server must expend to each independent client, the server may be accessible to more clients, thus increasing server capacity (Gillis, 2020).

ii. Disadvantages of Fat Client Architecture

The fat client architecture, unfortunately, has many drawbacks, among them being, security, network traffic, data storage, investment into each client, and maintenance (Gillis, 2020).

The fat client architecture stores its resources on the client machine, this means the user will now have to be progressively more responsible for the protection and security of their client device, in order to keep the stored data safe (Gillis, 2020).

Fat clients may have loads of network traffic because each client device requires the application to bring data through a network to download and work on locally (Gillis, 2020).

The fat client architecture stores its data on the client device, this decreases the requirements for the central server. However, this can be a doubtful advantage as data must now be backed up in order to make sure the data isn't lost forever when or if something goes wrong on the client machine (Gillis, 2020).

Fat clients require a higher up-front cost for hardware and software. Proceeding this, a continual cost will exist for maintenance and updates (Gillis, 2020).

The fat client architecture may include updates for any software or hardware fixes, alongside any security improvements across the connected client machines (Gillis, 2020).

II. Thin Client Architecture

Thin clients otherwise known as lean clients, utilize a virtual desktop computing model which function on the resources stored on a central server, rather than on a client device's resources. Often, thin clients take the shape of low-budget computing devices that immeasurable rely on a server for computing. Thin clients may also be classified as application software that utilize the client-server model to perform the majority of processing on a central server (Gillis, 2021).

IT staff, company employees, and public environments such as government offices, schools, or libraries may use thin clients because of the level of manageability, scalability, and security that is provided to them by the thin client architecture (Gillis, 2021).

The thin client architecture functions through connecting to a server-side computing environment. The server will usually hold data such as programs and memory. In basic terms, the computer environment is stored on a server. Lean clients are controlled server side via a VDI (virtual desktop infrastructure). Lean clients are dependent on a sustained internet connection to a primary server for full computation functionality and do not perform a lot of processing on the client device's hardware itself (Gillis, 2021).

The term thin client or lean client is derived from the fact that mini computers within networks are frequently clients and not servers. The goal of the thin client architecture is to limit the capacity for clients to only mandatory applications, so as to keep the purchased client "thin" in terms of the client software they include. Clients may be for example laptops, mobile devices, or desktop computers (Gillis, 2021).

Lean clients may be designated to employees across the working industry for countless reasons. For example, thin clients may be utilized to replace computers and to aid in accessing virtualized applications or desktops. It is generally accepted that it is more cost efficient to utilize lean clients in comparison to a desktop machine where all the processing is performed locally. This is the case since each individual lean client does not need to be powerful or new, because most of the processing is performed server-side (Gillis, 2021).

Lean clients may also be utilized within remote environments, so users do not have to fret about frequently repairing their PCs. If the client device is receiving most of its data from a server, there will be less functional parts client-side to fret about. In addition to this, businesses that may need client devices to have increased security may choose thin clients over other architectures such as the fat client architecture (Gillis, 2021).

Lean clients sound great however, they may not be the best choice for every situation. For example, users would require an environment where they have a stable and strong internet connection. Ferocious applications may run slow, as many people could be accessing the network simultaneously. In these cases, lean clients are more recommended for businesses that utilize less ferocious applications and have back-end infrastructure to embrace the requirements of each thin client such as a stable and strong internet connection (Gillis, 2021).

i. Advantages of Thin Client Architecture

The lean client architecture provides countless benefits, including but not limited to, increased protection, centralized manageability, and less budgetary needs.

Lean clients provide increased protection over other such architectures like the fat client architecture. This is because, lean clients do not store their resources on the client device, instead, lean clients store their resources on a central server. This in turn, provides increased protection against malware and other such attacks as the intruder must now attack the central server (Gillis, 2021).

The lean client architecture utilizes a virtual desktop computing model which functions on the resources stored on a central server. This adds centralized manageability to the client-server architecture as the developers no longer need to send updates out to each individual client device, instead, developers only need to update the application located on the central server (Gillis, 2021).

Lean clients utilize a virtual computing environment located server-side. This reduces the budgetary needs for client devices as they no longer need to be powerful or new in order to operate as clients successfully (Gillis, 2021).

ii. Disadvantages of Thin Client Architecture

The thin client architecture, unfortunately, has a handful of drawbacks, among them being, a mandatory internet connection, and decreased operation speeds.

The thin client architecture operates by connecting to a virtualized computing environment on a server. Unfortunately, this means the client device must have a continuous internet connection in order to connect to the server and run the application (Gillis, 2021).

Lean clients utilize a virtual desktop computing model which function on the resources stored on a central server. This may sound like an upside, however, networks are predominantly sluggish compared to relying on internal computer components (Gillis, 2021).

B. Single Page Applications (SPAs)

Single-page applications (SPAs) are a great tool for creating astonishingly interactive and unique experiences for web users. A single-page application is a web application or website that transforms the present web page dynamically utilizing new data from a web server such as new HTML content (Lawson, 2022).

In former times, a primary model arose in the website and web application development industry known as a multi-page application (MPA). In this solution, websites and web applications are based upon multiple pages being created in separate HTML files. Dynamic HTML content is produced by the backend, and swapping between separate pages involves dispatching a request to the server to obtain an additional page, subsequently downloading the additional page to a client device (ĆWIK, 2021).

This causes mass delays in loading and presenting the content, predominantly as data transfer does not always yield smooth outcomes. This causes a result, where the waiting time for the system to react is occasionally too long for the modern user's expectations (ĆWIK, 2021).

In comparison to multi-page applications, single-page applications are more swift, straightforward, and user-friendly. A single-page application exclusively utilizes one HTML file and will not load new pages while the application is being utilized. All HTML content can be loaded at the same time after communication with the server has been established. Succeeding this, content and data may be generated on a client without additional transportation from the server. This method ensures, after clicking on a subpage, the content is presented instantly, without the hassle of waiting for a server response (ĆWIK, 2021).

This system constructs single-page applications to provide a strongly preferable user experience than a multi-page application. Single-page applications react instantaneously and without "stitches" in the shape of noticeable transitions between one page to another, or application status changes. In this regard, SPAs bear a striking resemblance to computer applications (installed on a device and capable of operating offline) will, with the suitable operating power of a device, not perform sluggish and rapidly respond to the user's commands with extraordinary ease (ĆWIK, 2021).

The conceptualization of single-page applications is not recent. Technology not dissimilar from the modern implementation of SPA was expressed in a patent from 2002. However, its implementation was based upon the AJAX mechanism which has been functioning as a network standard since 2006. The AJAX mechanism is a set of techniques utilized for the construction of web systems utilizing various client-side technologies in order to create asynchronous applications. AJAX makes it possible for a program to send and receive data from the server in the background, without modifying the current state of the active page and the content display mode, which in turn, stops the program from having to disturb the user experience (ĆWIK, 2021).

However, the arrival of AJAX was not a revolutionary breakthrough moment in the expansion of single-page applications. Circulation of single-page applications is a result of a broader occurrence related to the swift development of the frontend and the expanded significance of UX in recent times. The first series of single-page applications were created as Flash applications or Java applets in addition with the "pure" jQuery or Java library. Although, the creation of these applications were tremendously facilitated by systems such as React.js, Angular, or Vue. The systems were relatively new JavaScript frameworks at the time which were utilized to create the frontend (ĆWIK, 2021).

Single-page applications have numerous advantages which have made them the preferred solution in many scenarios today. Single-page applications function well especially in projects of various web tools where beautiful UX and speed are crucial with content being of little importance. There are countless examples of single-page applications in use today. The most welcomed of which are Google Maps, Netflix, Facebook, PayPal, or Gmail (ĆWIK, 2021).

The single-page application model provides a range of benefits including, increased speed, improved user experience, and easier debugging.

Single-page applications may have longer start times than multi-page applications. This is the case because, single-page applications download all their files to the client at startup. However, after initial load, SPAs become astonishingly faster than MPAs as they no longer need to download files to the client from the server when a new page is loaded (ĆWIK, 2021).

The single-page application model provides more speed and smoothness over the multi-page application model making for a more attractive user experience. This is a powerful argument in advocacy of single-page applications as UX is of great importance in today's society (ĆWIK, 2021).

Single-page applications are easier to debug than multi-page applications. This is because, instead of reading through thousand of lines of code on the server side, developers can debug single-page applications by simply examining the JavaScript code rendered within the browser. Additionally, numerous JavaScript frameworks utilized to create SPAs have built-in debugging tools (ĆWIK, 2021).

The single-page application model, unfortunately, has numerous drawbacks including, positioning difficulty, slower start times, and incompatibility with older browsers.

The single-page application model may be a hassle when it comes to positioning on search engines such as Google. This is the case since, search engine crawlers aren't very efficient with JavaScript-based websites or web applications (Ascendle, 2021). For this reason, it is highly recommended to utilize a hybrid or MPA solution if you are developing for a system where content plays an important role (ĆWIK, 2021).

Single-page applications may have longer start times than multi-page applications. This is the case because, single-page applications download all their files to the client at startup. This can be a disheartening nuisance if the internet connection is sluggish or weak as it may take minutes to access the website or web application (ĆWIK, 2021).

Single-page applications may be incompatible with older browsers. Unfortunately, this is a result of the widespread utilization of JavaScript frameworks and fresh APIs when developing them (ĆWIK, 2021).

The single-page application model is highly recommended for company or consumer applications that require numerous interactions with the user where the smoothness and dynamics of their performance is crucial. The multi-page model will function effectively for news services, whilst hybrid models are more suited for e-commerce and other such internet tools which include countless sites requiring indexing in search engines (ĆWIK, 2021).

C. REST APIs

A RESTful API otherwise known as a REST API, is an application programming interface (cyberspace API or API) that adheres to the constraints of REST architectural style and permits the interaction with RESTful web services. REST stands for representational state transfer and was developed by computer scientist Roy Fielding (Red Hat, 2020).

The recurring grouping of single-page applications and RESTful APIs exceeds the fact that both have become popular buzzwords in recent times. The architectural styles of SPAs (single-page applications) and REST are not dependent on each other in principle. However, it would be extremely strenuous to visualize either triumphing without the other in regard to the development of websites or web applications (Downey, 2013).

The dependency of single-page applications on REST is conceivably the most apparent. In order to function, single-page applications must retrieve and manipulate data utilizing countless calls to a server using a method other than the traditional pattern of page refreshes (Downey, 2013).

SOAP (a messaging protocol for exchanging structured information) web services have extensively been and stand to be an option (SmartBear, 2020). In conjunction with this, JavaScript XML-HTTP-Request objects are able to return data in a wide variety of compositions. However, creating enormous and intricate SOAP requests and parsing their XML-based results within JavaScript remains error-prone and highly complex. Producing a practical JavaScript library for SOAP more often than not means creating it specifically for a single SOAP API. As a result, some developers choose to avoid creating SOAP requests from JavaScript, instead making such calls inside server-side code and dispatching the results to the client-side utilizing an alternative mechanism (Downey, 2013).

In comparison, countless web developers find making calls to RESTful APIs and parsing the JSON-formatted results simple. In conjunction with utilizing jQuery (a JavaScript library designed to simplify HTML DOM traversal and manipulation, event handling, animation, and Ajax), making the whole process insignificant (OpenJS Foundation, 2023). In addition to this, while each SOAP API must be learned once more, RESTful APIs are normally predictable and intuitive. RESTful APIs have made the notion of a single-page application attainable for countless development teams (Downey, 2013).

Intriguingly, single-page applications make it possible to create rich web-based applications within a RESTful architecture. Developers usually think of REST as a form of API, that is an alternative to SOAP, however, REST is more than that. REST is an architectural style established by a set of six constraints which aim to encourage maintainability and scalability. If the architecture of a specific website or web application conforms to these constraints, we may call it RESTful (Downey, 2013).

The constraints that establish a RESTful website or web application are:

1. Client-Server Architecture - A REST architecture demands separation between the frontend (graphical user interface) and backend (business logic). The implementation of this, hopes to improve scalability by simplifying the server’s role and enabling elements to evolve solitarily (Downey, 2013).

2. No States – The server must not hold any session data on behalf of the client, instead the client should store all session data. This is the case because, each request from client to server needs to contain all data necessary to process the request (Downey, 2013).

3. Cache – Server responses need to be tagged as non-cacheable or cacheable (Downey, 2013).

4. United Interface – Components need to converse through united interfaces to separate implementations from the service they may provide (Downey, 2013).

5. Layered systems – Individual layers within the system must not perceive layers outside the layer to which it is promptly connected. This provides the possibility of numerous proxy servers without a dramatic escalation in complexity (Downey, 2013).

6. Code on Demand – The server may provide the functionality to clients to download and execute scripts on request (Downey, 2013).

1. Evaluation and Recommendation

I. Table of Comparison

Parameter	React	Angular	Vue JS	Svelte	Preact	Solid JS
Details						
Initial Release	May, 2013	September, 2009	February, 2014	November, 2016	November, 2015	2018
Type	Library	Framework	Framework	Framework	Library	Framework
Support	Facebook	Google	//Team	Community	Individual	Team
Strengths and Weaknesses						
Ease of Use and Learning Curve						
Documentation	Sub-Standard	Sub-Standard	Comprehensive	Satisfactory	Comprehensive	//Sub-Standard
Language Support	JSX and TypeScript	TypeScript	HTML, JavaScript, and JSX	Hybrid Language Comprised of HTML, JavaScript, and TypeScript	JSX	//TypeScript
Learning Curve	Low – Medium	Low - Medium	Low - Medium	Low	Low – Medium	//Low – Medium
Performance						
Size	Medium	Large	Small	Small	Small	//Small
DOM	Virtual	Incremental	Virtual	Real	Virtual	//Real
Component Based Architecture (CBA) Support (Reusable Components)	Yes	Yes + Services and Dependency Injection (Angular Injector)	Yes	Yes	Yes	Yes
Community Support						
Community Experience	Comprehensive	Satisfactory	Satisfactory	Sub-Standard	Satisfactory	Sub-Standard
Training Resources	Comprehensive	Comprehensive	Comprehensive	Satisfactory	Satisfactory	Sub-Standard

Community Support	Comprehensive	Comprehensive	Comprehensive	Satisfactory	Satisfactory	Sub-Standard
Popularity						
NPM JS (Weekly Downloads)	16,837,641	2,759,410	3,187,972	434,621	2,055,359	82,278
Stack Overflow (Weekly Posts)	1,425	36	198	23	2	0
Popularity	High	Medium	Medium	Low	Medium	Low
Other						
IDE Support	Good	Good	Good	Good	Good	Poor
Routing	Provided with React-Router	Provided with Angular/Router	Provided with Vue-Router	Provided with Svelte-SPA-Router	Provided with React-Router	Provided with Solid-Start
SEO Friendly	Yes	No	Yes	Yes	Yes	Yes with the Aid of the Babel Plugin
Native Support For An Architectural Pattern	N.A.	MVVM or MVC	MVVM	MVVM	N.A.	Specialized
Data Binding Support	Unidirectional	Bidirectional	Bidirectional	Bidirectional	Unidirectional	Unidirectional
Project Size Support	Flexible	Flexible	Small - Medium	Flexible	Small	Flexible

II. React Library

i. React Overview

React otherwise known as ReactJS, is a UI development tool first released by the tech giant Meta (originally Facebook) in May 2013, and whilst the corporation still maintains the library, it has now become open source. React is frequently referred to as a framework for the purposes of simplicity. However, React is in fact a JavaScript library. This is the result of the library targeting a single layer of an architectural pattern, more specifically the presentation or UI layer (Adam, 2023).

React is utilized by many corporate giants today, amongst them being, PayPal, Twitter, Airbnb, Walmart, and Netflix. This is the result of Reacts technical qualities being a good fit for both small and large web development projects (Adam, 2023).

A component-based library with unidirectional data binding, React has had a large influence on modern day front-end web development, and remains the dominant technology used for JavaScript applications (Adam, 2023).

React may be accredited with the establishment of such concepts as functional, immutable states, and declarative programming, which were not precedingly uncommon in front-end development. React's other grand achievement was the establishment of the Virtual DOM, which has aided in greatly improving app performance and user experience especially regarding a dynamic UX (Adam, 2023).

ii. React Strengths

Code Simplistically and Flexibility – React utilizes JSX (JavaScript XML) which is an extension to JavaScript based in ES6 (Egghead, n.d.). This gives React a more forgiving learning curve for JavaScript developers in comparison to learning TypeScript for Angular (Adam, 2023). However, React does provide it's own version of TypeScript which developers may utilize if they wish.

Component Based Architecture – Any React website or web application is constructed out of numerous components, and each individual component posses its own controls and logic. The components are in charge of outputting a mini, reusable segment of HTML content which may be reutilized where the developer believes is best. The reutilized code aids in helping the developer make the application simpler to develop and maintain. The components may be imbedded within other components to allow advanced applications to be built of simplistic building blocks. React utilizes a virtual DOM mechanism to load these components within the web application. The virtual DOM operates swiftly as it only modifies individual DOM elements rather than reloading the complete DOM each time a new component must be loaded (JavaTpoint, 2021).

Virtual DOM – React utilizes the virtual DOM to greatly increase performance. The DOM (Document Object Model) is a multi-platform programming API which handles Extensible Markup Language (XML), Hypertext Markup Language (HTML), or Extensible Hypertext Markup Language (XHTML). When React first released in 2013, numerous developers were confronted with the problem of the DOM significantly hurting their applications performance. This was the result of the DOM having to update each time a user interacted with the web application. To solve this issue, React introduced the virtual DOM. React's virtual DOM exists wholly in memory and is a rendition of the browsers DOM. Due to this, when a developer utilizes a React component, it does not write directly to the browser DOM. Rather, the web application is writing the virtual components that react will soon turn into the DOM, extending to faster and smoother performance results (JavaTpoint, 2021; KnowledgeHut, 2022).

Simple to Learn and Utilize – React is simple to learn and utilize. The library has over sixteen million downloads a week via NPM with an active Stack Overflow page with approximately one thousand and four hundred posts a week, alongside this, the library has a very prominent tutorial prominence on YouTube and other such platforms. As a result, countless tutorials and training resources are available for newcomers. Any web developer should be able to simply understand and begin creating web applications utilizing React in a few days. The library is the view segment of Model-View-Controller (MVC). This is the result of the library not having the proper model and controller segments of the MVC architectural pattern.

iii. React Weaknesses

Architectural Pattern – React, unfortunately, only provides the view segment of the Model-View-Controller (MVC) architectural pattern. This means that developers will need to search for fitting libraries or frameworks to fill the gaps that React has left behind. Furthermore, this opens a can of warms for developers to contend over which is the better tool set for the web application that must be created, which could cost precious development time.

Poor Documentation – A criticism levelled against React libraries under unfortunate circumstances. The React library has a very vigorous community. Alongside this, Facebook frequently supports the library with regular updates and new development tools. Unfortunately, when new updates and tools are released, documentation is often lacking, which can make it difficult for developers to get up to speed with new features. As a result, React library integration can be very poor (JavaTpoint, 2021).

iv. React Conclusion

Unfortunately, although the React library was the primary choice due to previous experience in a former class. The choice has transformed into a misguided selection. React has numerous benefits including, a virtual DOM, healthy community, and is simplistic and easy to learn. However, the library has two severe disadvantages which make it undesirable, needless flexibility in the architectural pattern, and poor documentation.

React, unfortunately, has needless flexibility in the architectural pattern for future projects as the library only provides the view segment of the Model-View-Controller (MVC) architectural pattern. This leaves developers to search for fitting libraries and frameworks to fill the model and controller segments of the MVC architectural pattern, this behavior is highly undesired in future development projects. In conjunction with this, React has very poor documentation, creating a development experience which is much more difficult than it needs to be.

III. Angular Framework

i. Angular Overview

Angular sometimes referred to as the aged AngularJS, is a JavaScript framework based upon TypeScript. When AngularJS was first released by Google in September 2009, it created a stir as one of the first JavaScript-based front-end frameworks (Adam, 2023).

Unfortunately, the release of Meta's more versatile React JavaScript library in May 2013, drew great attention to Angular's downfalls and developers began to move away from the Framework (Adam, 2023).

In a significant example of why healthy development ecosystems are driven by rivalry, and competing standards and tools, React caused Google to take it to the next level. The result of this was a new framework which replaced the previously released AngularJS in 2016. The new framework simply went by the name 'Angular' (Adam, 2023).

All Angular releases preceding version 2.0 are AngularJS, and all subsequent releases are Angular. In comparison to JavaScript libraries such as React, Angular is an end-to-end framework that provides all of the main components required to build an enterprise-grade web application using a architectural design pattern. Developers may utilize Angular to create Model-View-ViewModel (MVVM) or Model-View-Controller (MVC) web applications (Adam, 2023).

The limitations inherently present in Angular's stricter structure, which is arguable its strength, have seen React succeed it in popularity in the last few years. In addition to this, numerous software developers argue that Vue.js, designed by a development team lead by ex-Googler Evan You to improve upon what he considered to be Angular's weaknesses, is the superior framework (Adam, 2023).

Angular continues to be popular for web development projects where a more assertive structure is considered an advantage, like massive applications where enforced consistency is advantageous. Backed by Google, Angular is also considered to be future-proofed. Alongside this, the framework has a deep pool of experienced developers on the market (Adam, 2023).

ii. Angular Strengths

TypeScript – Angular was created and utilizes TypeScript which considerable simplifies coding for newcomers to JavaScript and TypeScript, this may be accredited as an advantage over other frameworks. The benefit of TypeScript is that it provides advanced error-detection capabilities. This allows programmers to identify and remove common mistakes while they are actively typing the code. Such an enrichment is predominantly useful for advanced prolonged projects as it may help a team extraordinarily diminish the time required for quality assurance and debugging (Shikht, 2023).

Vigorous Ecosystem – Developers do not use Angular just to utilize the platform itself. The framework is complemented with an extended list of resources for Angular developers, among them being, add-ons, tools, and plugins etc. The resources aid developers to build advanced web solutions in a swift period, procedurally generate documentation, resolve issues more rapidly, and execute many more tasks. Angular Material also exists, which aims to streamline the design process. Angular Material is a UI library that supplies a variety of preconstructed elements that can be simply implemented into a web application. As a result, web development teams may utilize less time on developing a product which may save a lot of money for a client (Shikht, 2023).

Incremental DOM – Angular, dissimilar to React, utilizes an Incremental DOM. The primary rule behind incremental DOM utilizing ivy's rendering engine, is that each component, after being processed with the incremental DOM, takes the shape of a sequence of ivy rendering engine instructions. The incremental DOM now acting as a form of Virtual DOM, creates a structure out of the instructions creating a DOM tree. Succeeding this, the incremental DOM will update the instructions, when a component is modified. The fascinating thing about the ivy instructions is that Angular does not interpret them as a rendering engine, instead, they are a rendering engine. As a result, Angular does not expend time interpreting the component, instead it utilizes references, provided by the ivy rendering engine with its instructions. In turn, this means that components that do not have specific instructions may be left behind. This process is referred to as "tree shaking". One more advantage Angular's incremental DOM has over React DOMs and other such DOMs, is memory usage. The Virtual DOM generates an entirely new tree for a component each time it is modified. Wherefore, it requires a heap of memory to "keep" these trees before transporting them to Real DOM. Incremental DOM requires much less memory because its memory only covers modifications in the structure (Slemzina, 2023).

iii. Angular Weaknesses

TypeScript – Angular was developed and uses TypeScript which could be considered as a severe disadvantage. This is the result of TypeScript being rather difficult to learn for JSX developers.

Dependency Injection – Angular comes with an integrated mechanism to inject rules or dependencies determining how individual pieces of code converse with each other and act under specific states, instead of creating the dependencies inside the components. This allows programmers to configure or modify dependencies without changing an app module in addition to reusing the dependencies across numerous modules. Unfortunately, this can also be extremely time-consuming, and it may be tricky to create dependencies for components (Adam, 2023; AltexSoft, 2022).

Stagnating Community – Unfortunately, Angular seems to have a stagnating community, as while React has over sixteen million downloads a week via NPM with an active Stack Overflow page with over one thousand posts a week. Angular only has three million downloads a week via NPM, with a Stack Overflow page with approximately forty posts a week. Whilst the NPM download count is nothing to scoff at, it does ring true that it's only one eighth the number of downloads for React. However, Angular does have a decent tutorial prominence on YouTube and other such platforms. As a result, while the frameworks community may be stagnating, there are numerous tutorials and training resources available online for newcomers.

iv. Angular Conclusion

Angular is a framework which provides countless benefits including, a robust ecosystem, an incremental DOM, and TypeScript which is simple to learn for newcomers. Unfortunately, however, the framework has one key disadvantage which makes it unappealing for future development projects, a stagnating community.

The Angular framework has a stagnating community, which may create a worrisome development environment where developers are unable to obtain help from the framework's respective community in regard to a specific problem or issue, which in turn, may halt development.

IV. Vue JS Framework

i. Vue JS Overview

Vue.js commonly known as Vue (pronounced "View"), was first released in February 2014, and subsequently relaunched in 2016. Vue describes itself as an "intuitive, fast and integrable MVVM for creating interactive interfaces." (Adam, 2023).

Vue inhabits a fascinating position in the triumvirate of frontend JavaScript frameworks. Vue is not hyped to the degree of React and is not as established as Angular, it has spent the last few years silently minding its own business and building an engaged and eager community of developers (Adam, 2023).

The primary goal of Vue.js is to make concepts in web UI development (declarative UI, components, time travel debugging, and hot-reloading etc.) more digestible. Vue.js is less opinionated, it is viewed as more straightforward to learn for youthful developers compared to other frameworks, including but not limited to Angular (Adam, 2023).

Vue.js was created by a development team lead by former Google Engineer Evan You. The development team hoped in combining the strengths of both React and Angular, that this would go some ways in mitigating their respective weaknesses (Adam, 2023).

Although Vue.js arrived on the development scene many years after both Angular and React, the framework has rapidly gained traction. Vue.js is utilized by numerous large corporations today including, Alibaba, Expedia, and Nintendo (Adam, 2023).

The fact that Vue.js does not have the benefit of being backed by a tech giant makes its success all the more noteworthy. However, Vue.js's unconstrained nature is seen by many in the web development community as a positive. The independent nature of this framework has resulted in a particularly diligent open source community supporting the framework. Questions and problems are usually responded to much more swiftly than is the case with the big, corporation backed frameworks like React and Angular (Adam, 2023).

One more productive side effect of Vue not being produced by a tech giant with huge resources is clear API/code which has not been over-engineered (Adam, 2023).

ii. Vue JS Strengths

Documentation – Vue's documentation is simple to comprehend and efficiently structured. In addition to this, the open-source community is noteworthily active in responding to questions or issues. The benefits mentioned previously result in obtaining experience with the framework rather painless for more seasoned developers (Adam, 2023).

Basic Learning Curve – Vue.js has a basic learning curve. All developers require is acquaintance with HTML, and ES5 JavaScript (plain JavaScript). Acquiring these basic skills, developers may start developing significant applications in less than a day of reading the guide (Vue.js, n.d.).

Component Based Architecture – All Vue web applications are built out of numerous components, and each component possess its own controls and logic. The components are responsible for outputting a diminutive, reusable segment of HTML content which may be reused where the developer believes is the most optimal. The reused code aids in helping the developer make the web application easier to develop and maintain. The components may be imbedded within other components to allow modern applications to be built out of simplistic building blocks (JavaTpoint, 2021).

Virtual DOM – Vue employes virtual DOM, a duplicate of the web application's original DOM is created so that a sufficient number of modifications may be made without re-rendering the whole DOM. As a result, this drastically minimizes the number of DOM manipulations required to be managed by Vue. The virtual DOM makes mobile application development swifter, alongside enhancing overall performance of existing web applications. All these factors result in superior performance, increased speed, and swifter DOM operations (Sharma, 2021).

Virtual DOM Event Sourcing and Reactive Two-Way Data-Binding – Vue JS attempts to combine the best of both worlds between React and Angular. The most noteworthy example of this is virtual DOM event sourcing (like React) and reactive two-way data binding (like Angular).

iii. Vue JS Weaknesses

Financial Backing – Unfortunately, Vue is comparatively new to React and Angular, and does not have the financial support and backing of large tech corporations such as Facebook or Google. Vue still lacks a few features which large corporations would tend to search for. For this reason, Vue is most commonly utilized in smaller projects (Adam, 2023).

Language Barrier – Vue is distinctly popular in the Chinese market. Historically, this caused a consequential segment of the content, i.e., plug-in descriptions, instructions, and community discussions on Vue to be written in Chinese. Luckily, there has been predominant English-language activity within the Vue community in recent years which has mitigated these concerns (Adam, 2023).

Less Demand – Unfortunately, there has been a lack of Vue developers. The reason for this is a dilemma, scarcely any Vue developers can put companies off the framework, whilst lower client demand demoralizes software developers from learning the framework. Nevertheless, this is a weakness that is becoming less noticeable over time (Adam, 2023).

iv. Vue JS Conclusion

Vue JS seems like an ideal choice for any future development projects. The reason for this is the framework has many advantages such as, comprehensive documentation, a basic learning curve, a virtual DOM, a component-based architecture, virtual DOM event sourcing and reactive two-way data binding. Unfortunately, the framework does have a few disadvantages such as problematic financial backing, language barriers, and less demand.

Vue's documentation is easy to comprehend and efficiently structured. In addition to this, the open-source community is active in responding to questions or issues. In conjunction with these factors, Vue has a basic learning curve as all developers require is acquaintance with HTML, and ES5 JavaScript. The aforementioned benefits result in obtaining workable experience with the framework a breeze for more seasoned developers.

Vue's performance and accessibility is also adequate. The framework utilizes a component-based architecture which uses individual components with their own logic and controls to aid developers in making a web application out of building blocks which is simple to develop and maintain. In addition to this, the virtual DOM makes mobile application development instantaneous, alongside strengthening overall performance of existing web applications. In conjunction with this, the framework utilizes virtual DOM event sourcing and reactive two-way data binding to make maintaining the relationship between the view and the model simple. All these factors result in superior performance, swifter DOM operations, and increased speed.

V. Svelte Framework

i. Svelte Overview

The initial idea for Svelte was born from a eureka moment of software engineer Rich Harris in 2016. Rich Harris hit upon the idea of building a JavaScript framework subtracting any framework-specific runtime (Adam, 2023).

The idea was accomplished by Svelte compiling framework-specific code into clean HTML, CSS, and JavaScript, with the compiled code rendered to the browser. The approach was not completely new in software development. However, it was revolutionary in the context of front-end development. Svelte's approach signifies that the browser does not have to parse and compile the code at runtime, resulting in swifter load times (Adam, 2023).

Svelte's subsequent unique quality is the inclusion of first-class support of reactivity. The support of reactivity allows for improved performance without the need for a Virtual DOM, which makes Svelte's rendering the swiftest of any JavaScript framework (Adam, 2023).

Svelte was written in TypeScript. However, making full use of the framework does not require knowledge of TypeScript. This is the result of the framework utilizing a component-based architecture which relies on JavaScript, HTML, and CSS (Adam, 2023).

Svelte has impressed a lot of JavaScript developers, and while the framework is still in its very early infancy, it is attracting a lot of interest and swiftly gaining traction in front-end web development communities. A few brave souls have gone as far as to suggest Svelte could, in time, replace React as the dominant JavaScript resource. However, like Vue, a lack of giant tech backing can be anticipated to result in a traction bottleneck at a certain point in time, in spite of its technical qualities (Adam, 2023).

The jump in the number of JavaScript developers utilizing Svelte, from just 8% in 2019 to 21% in 2022, is noteworthy and if it continues in this trajectory, it may perhaps become seen as a mainstream, and commercially viable alternative to Angular, React, and Vue (Adam, 2023).

ii. Svelte Strengths

Low Learning Curve – Dissimilar to Angular and React, the learning curve for Svelte is extremely low. This is the result of the framework utilizing no special syntax such as JSX to learn or advanced APIs such as Angular to recollect. Everything is created using standard-compliant TypeScript, JavaScript, and HTML. Additionally, the framework does incorporate some new syntax for template logic and directives. The component API is simplistic and effortless. In addition to these factors, the documentation is also quite simplistic to follow (Sasidharan, 2021).

Svelte Compiler – Svelte is first and foremost a compiler. This is the result of the .js file being loaded directly on to a website's page to be rendered. In conjunction with this, Svelte's code does not require loading the whole library on the browser. In addition, the overhead produced by a virtual DOM is reduced in Svelte as all objects are only updated once at compile time (JavaTpoint, 2021).

Component Based Architecture – Dissimilar to React and other such libraries, Svelte utilizes the component-based architecture in a tremendously different fashion. React utilizes a component-based architecture which is comprised of JSX files for the functional and presentational logic for the graphical user interface. However, Svelte differs in this regard, as it is a language in itself, comprised of the three standard languages, JavaScript for the functional logic, HTML for the presentational logic, and CSS for the styling logic. This maintains a clear distinction between different logic groups. Alongside this, cutting down the development time which would usually be expended on searching for a snippet of code within a JSX file (Bhushan, 2023).

iii. Svelte Weaknesses

Limited Range of Libraries and Plugins – Unfortunately, Svelte has a rather limited collection of available readymade plugins and libraries which may make the process of finding solutions for specific use cases rather demanding. As a result, experienced developers may need to create more custom code to implement some operations than they would if utilizing a more mature alternative such as React, Angular, or Vue (Adam, 2023).

Underdeveloped Community – Unfortunately, Svelte is still a very young framework which does not yet have a multitude of significant use cases or third-party tools. Sadly, lack of tooling and tooling support is presently the main criticism of Svelte (Adam, 2023). In addition to this, the community has not had time to mature which can be seen by taking a glancing at the statistics for NPM downloads and Stack Overflow Posts for the framework. Svelte barely has four hundred and thirty-five thousand downloads a week, alongside approximately twenty posts on Stack Overflow a week. Fortunately, it seems there is a growing community on YouTube and other such platforms with numerous tutorials and training resources available online for newcomers.

iv. Svelte Conclusion

Svelte is a framework which delivers a great number of benefits including, a low learning curve, specialized component-based architecture, and the svelte compiler which reduces the overhead produced by a virtual DOM. However, Svelte does have two severe disadvantages which cannot be ignored, an underdeveloped community, and a limited range of plugins and libraries.

Unfortunately, Svelte has an underdeveloped community. This is the consequence of the framework still being immensely youthful. As a result, Svelte does not yet have a vast amount of noteworthy use cases. In conjunction with this, the framework is still lacking third party tools such as plugins and libraries. All the aforementioned factors make the framework a poor choice for selection in regard to future development projects.

VI. Preact Library

i. Preact Overview

Preact is a JavaScript library with the sole purpose of providing the strengths of React in a smaller package. The JavaScript library has a size of 3.5kb as opposed to Reacts size of 30kb. This makes Preact an attractive alternative to React, at least from a technical standpoint, for single-page applications (SPAs) (Adam, 2023).

Preact was developed by Googler Jason Miller with the assistance of a group of contributors. The library is utilized by well known businesses such as, Lyft, Uber, Groupon, Tencent, and Domino’s (Adam, 2023).

The launch of Preact 10 cleaned up the library's code, included new features, and compatibility enhancements to support additional third-party libraries (Adam, 2023).

Preact's speed and size make it ideal for light mobile web apps or web widgets that React may be too heavily featured to cover. Preact's understated nature signifies that it will likely be considered for small projects where performance is prioritized or embeddable components which prioritize lightness (Adam, 2023).

ii. Preact Strengths

Size – Preact's mini size and efficient rendering algorithm result in a swifter rendering performance in comparison to other front-end libraries such as React. This sets up Preact to be a superb choice for developing high performance orientated websites, particularly on low-end desktop, or mobile devices (Adam, 2023).

React Qualities – Preact utilizes a similar API and programming model to React. As a result, this makes it relatively simple for React developers to learn and utilize Preact. In conjunction with this, React code may easily be migrated to Preact, without the hassle to recreate everything from scratch (Adam, 2023).

High Performance DOM – Preact happens to be amidst the swiftest virtual DOM libraries out there today. This was made possible due to the libraries condensed size, alongside it's predictable and elementary diff implementation (Panchal, 2021).

Sizeable Ecosystem – Preact provides a good set of documentation, which makes the learning curve low. In conjunction with this, the library has a sizeable community with numerous contributors ready to perform bug fixes and provide solutions to problems that may arise (Dhaduk, 2022). The library has over 2 million downloads a week via NPM.

iii. Preact Weaknesses

React Qualities – Preact was developed as a weightless alternative to React. Consequently, most of the development via Preact imitates React as opposed to innovating web application development (Dhaduk, 2022). Unfortunately, this also means that Preact inherits numerous flaws from React. An example of this is the limitation with the Architectural Pattern. Preact like its counterpart, is a UI library, which only provides the view segment of the Model-View-Controller (MVC) architectural pattern. This means that developers must search for fitting libraries or frameworks to fill the gaps that Preact has left behind. Furthermore, this opens a can of warms for developers to contend over which is the better tool set for the web application that must be created, which could cost precious development time.

Performance Impact from Additional Libraries – Unfortunately, with Preact, it is a requisite to utilize additional libraries such as preact/compat and preact/test-utils etc., to bring connectivity between React and Preact-based NuGet Package Manager (NPM) packages. Sadly, as a result, this causes the project to become large and slow down (Dhaduk, 2022).

Ecosystem – Preact may provide a good set of documentation, have a sizeable community with numerous contributors ready to perform bug fixes and solve problems, and have a substantial weekly download count for an individual driven project. Unfortunately, however, the Stack Overflow page for Preact only has an approximate weekly post amount of two. Furthermore, the frameworks YouTube prominence is sub-par at best.

iv. Preact Conclusion

Preact is a supplementary library to React which provides a few benefits in comparison to the other five selected frameworks and libraries including, a small size, high performance DOM, sizeable ecosystem, and compatibility with React code which may be easily migrated to Preact. Unfortunately, the library does have two key disadvantages which make it undesirable for future projects, a small active community, and an unnecessary flexibility in the architectural pattern.

Preact, unfortunately, follows in the boots of its bulky predecessor React. As a result, the library has unnecessary flexibility in the architectural pattern for futures projects as the library only provides the presentational layer of the Model-View-Controller (MVC) architectural pattern. This leaves developers to search for the correct frameworks and libraries to fill the model and controller components of the MVC architectural pattern, this behavior is tremendously undesired in future development projects. In conjunction with this, the library has a very small active community with an estimated Stack Overflow presence of two posts a week. As a result, this creates a development experience where developers may not be able to obtain help with specific issues or problems.

VII. Solid JS Framework

i. Solid JS Overview

Solid JS is an open-source declarative JavaScript framework, which was developed by Ryan Carniato, and subsequently released in 2018. The framework was designed for developing large user interfaces with maximum control over reactivity. Solid JS has rapidly impressed JavaScript developers who view it as offering a great balance between pragmaticism and performance. The framework was built on and supports TypeScript (Adam, 2023).

Solid JS shares numerous similarities with the React library, among them being, components as JavaScript functions that return JSX for the user interface and unidirectional data binding. Solid's most significant difference from React is utilizing a compiler similar to Svelte's that converts code to vanilla JavaScript, becoming as close to the DOM as possible rather than a Virtual DOM (Adam, 2023).

Solid JS is regarded as the first truly reactive JavaScript framework with function components only being called once. The reactive nature of Solid allows developers to do things unfeasible in other frameworks and libraries, amongst them being, setting intervals predictably at the top level (Adam, 2023).

Data that changes is observed by the framework and surgically update this data in the exact location in the DOM when it changes instead of re-rendering the entire component, making it fairly reactive. Nested reactivity is accomplished through the use of stores whose return values are a proxy object whose properties can be tracked (Adam, 2023).

ii. Solid JS Strengths

Size – Solid JS's small size and speed make it ideal for developing JavaScript widgets that will minimally impact host websites (Adam, 2023).

Reusability – All segments of Solid JS are independently reusable, which makes it feasible to develop a wide variety of different project types with the framework. In addition to this, the simplicity of Solid's reactive system means it is very simple to hook any other state system into Solid JS components (Adam, 2023).

iii. Solid JS Weaknesses

Limited Documentation – Unfortunately, developing a web application using a relatively new framework with a smaller active community than other frameworks and libraries can come with its own set of challenges. Limited documentation may cause developers to struggle to locate solutions when problems arise (Graffersid, 2023).

TypeScript – Solid JS utilizes TypeScript which could be considered as a severe disadvantage. This is the result of TypeScript being rather difficult to learn for JSX developers.

DOM Interaction – Unfortunately, in Solid JS, all rendering and DOM interaction is an aftereffect of the reactive system. This means that whilst it is feasible to manage things like exit animations in a similar fashion to React or Vue utilizing the reactive lifecycle. The framework does not keep track of the DOM in a way that is accessible from outside of a user's own references (Adam, 2023).

Solid JS, being declarative, makes debugging more of a pronounced challenge. Templates are simple to debug as Solid places all the DOM operations out in the wild so any issues may be spotted. However, bugs that occur elsewhere, which is most bugs, can be difficult to find (Adam, 2023).

Community – Unfortunately, as Solid JS is only five years old, and is not backed by a huge tech corporation, the framework has not had time to mature. This can be seen in the statistics for NPM downloads and Stack Overflow posts for the framework. Solid only has an estimated weekly NPM download count of eighty-two thousand, with no active Stack Overflow page. Fortunately, it appears there is a growing community for the framework on YouTube and other such platforms.

iv. Solid JS Conclusion

Solid JS is a framework which delivers a few benefits including, a small size, and reusability as all segments of Solid JS are independently reusable. Unfortunately, the framework has many more disadvantages which make it an unideal choice for future development projects, limited DOM interaction, an undeveloped community, and limited documentation.

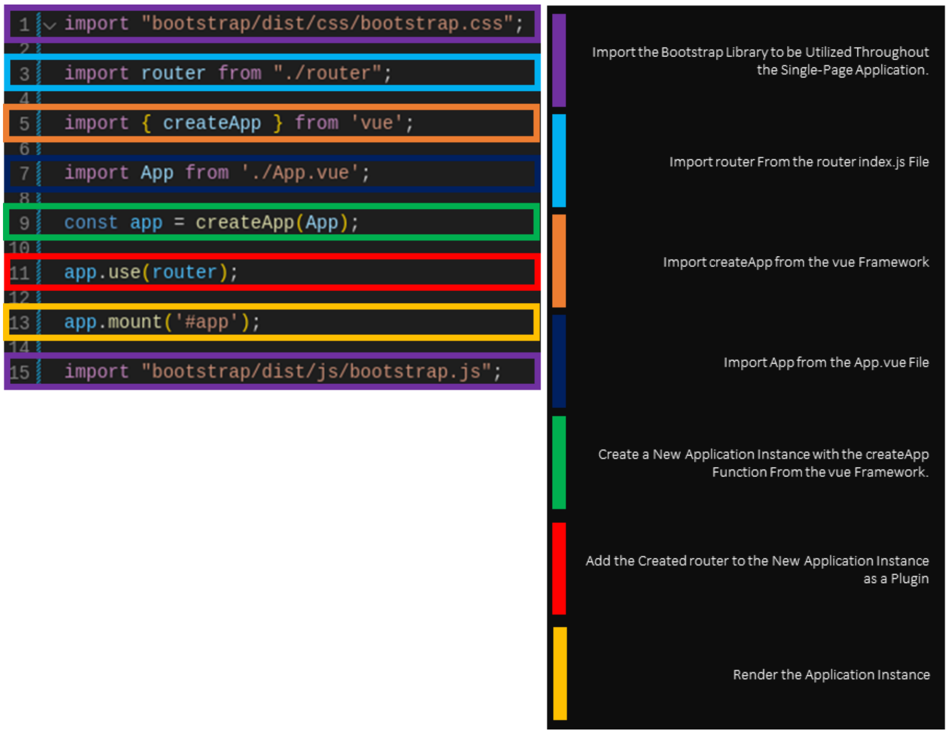
Unfortunately, Solid has limited DOM interaction. This is the result of any rendering and DOM interaction being an aftereffect of the reactive system, while it is feasible to manage things like exit animations. The framework does not record the DOM in a way that is accessible from outside of a user's own references.

The framework also has an undeveloped community. This is the consequence of the framework still being exceedingly youthful and not being backed by a big tech corporation. Due to these factors, the frameworks documentation is severely limited.

2. Vue JS Code Example

Please note many graphical user interface elements have been styled utilizing Bootstrap.

main.js



```
import "bootstrap/dist/css/bootstrap.css";

import router from "./router";

import { createApp } from 'vue';

import App from './App.vue';

const app = createApp(App);

app.use(router);

app.mount('#app');

import "bootstrap/dist/js/bootstrap.js";
```

Import the Bootstrap Library to be Utilized Throughout the Single-Page Application.

Import router From the router index.js File

Import createApp from the vue Framework

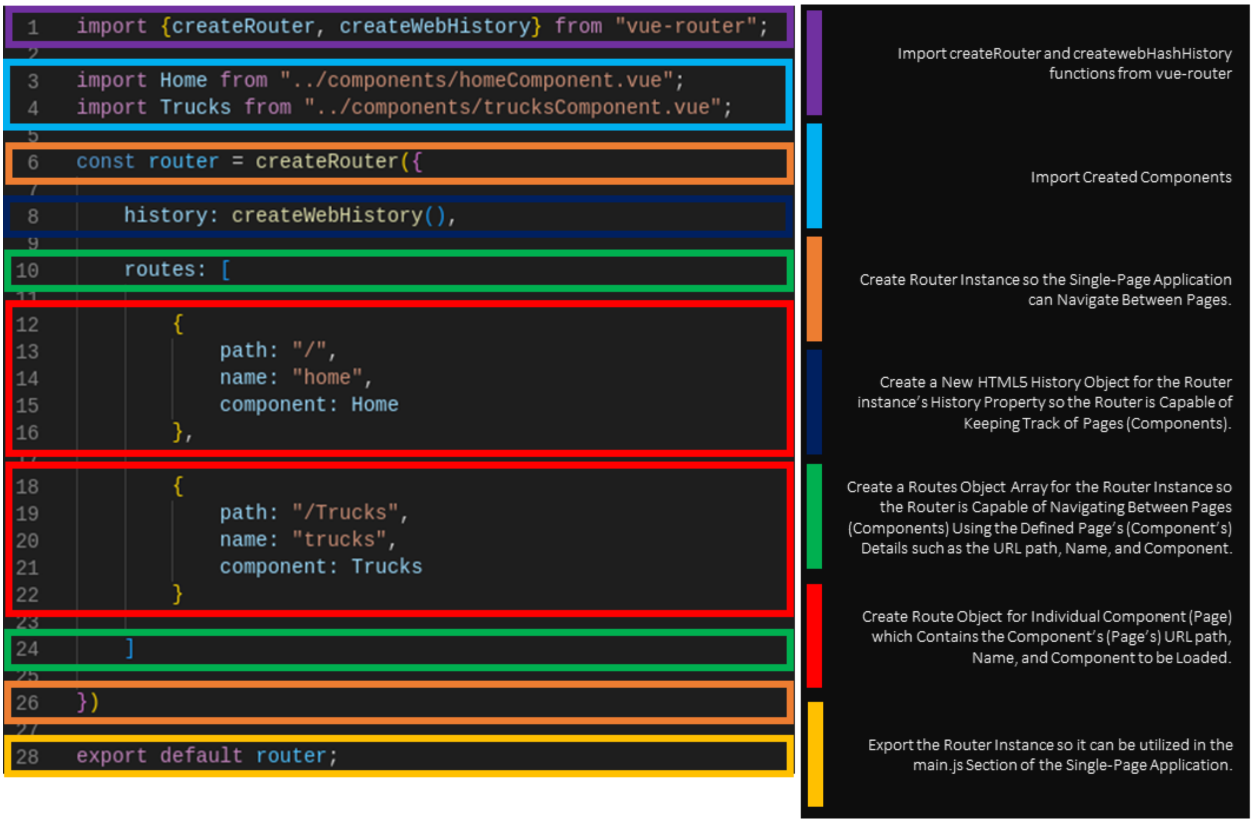
Import App from the App.vue File

Create a New Application Instance with the createApp Function From the vue Framework.

Add the Created router to the New Application Instance as a Plugin

Render the Application Instance

Router index.js



```
import {createRouter, createWebHistory} from "vue-router";

import Home from "../components/homeComponent.vue";
import Trucks from "../components/trucksComponent.vue";

const router = createRouter({

    history: createWebHistory(),

    routes: [

        {
            path: "/",
            name: "home",
            component: Home
        },

        {
            path: "/Trucks",
            name: "trucks",
            component: Trucks
        }

    ]

})

export default router;
```

Import createRouter and createwebHashHistory functions from vue-router

Import Created Components

Create Router Instance so the Single-Page Application can Navigate Between Pages.

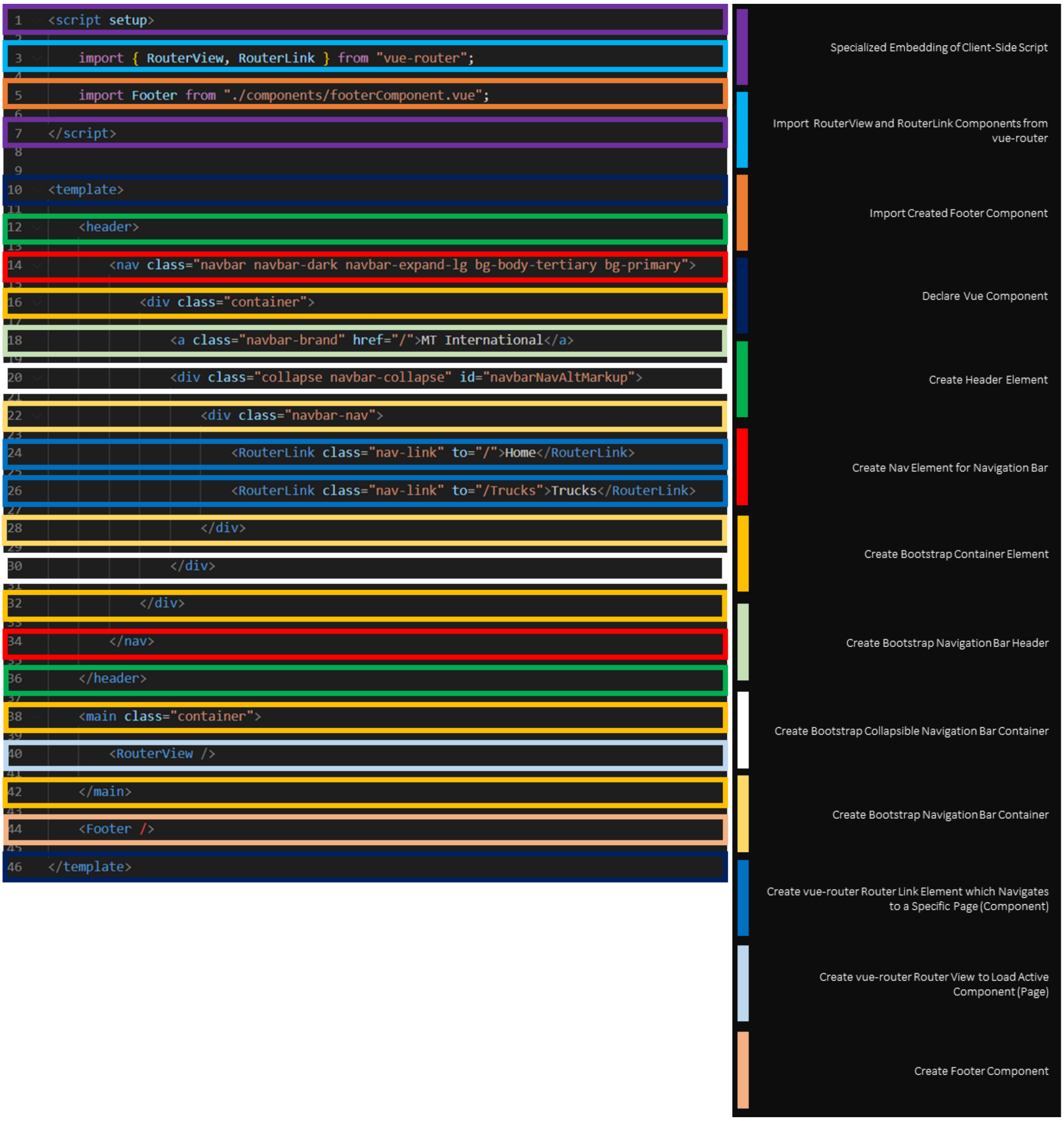
Create a New HTML5 History Object for the Router instance's History Property so the Router is Capable of Keeping Track of Pages (Components).

Create a Routes Object Array for the Router Instance so the Router is Capable of Navigating Between Pages (Components) Using the Defined Page's (Component's) Details such as the URL path, Name, and Component.

Create Route Object for Individual Component (Page) which Contains the Component's (Page's) URL path, Name, and Component to be Loaded.

Export the Router Instance so it can be utilized in the main.js Section of the Single-Page Application.

App.js



```
<script setup>

    import { RouterView, RouterLink } from "vue-router";

    import Footer from "./components/footerComponent.vue";

</script>


<template>

    <header>

        <nav class="navbar navbar-dark navbar-expand-lg bg-body-tertiary bg-primary">

            <div class="container">

                <a class="navbar-brand" href="/">MT International</a>

                <div class="collapse navbar-collapse" id="navbarNavAltMarkup">

                    <div class="navbar-nav">

                        <RouterLink class="nav-link" to="/">Home</RouterLink>

                        <RouterLink class="nav-link" to="/Trucks">Trucks</RouterLink>

                    </div>

                </div>

            </div>

        </nav>

    </header>

    <main class="container">

        <RouterView />

    </main>

    <Footer />

</template>
```

Specialized Embedding of Client-Side Script

Import RouterView and RouterLink Components from vue-router

Import Created Footer Component

Declare Vue Component

Create Header Element

Create Nav Element for Navigation Bar

Create Bootstrap Container Element

Create Bootstrap Navigation Bar Header

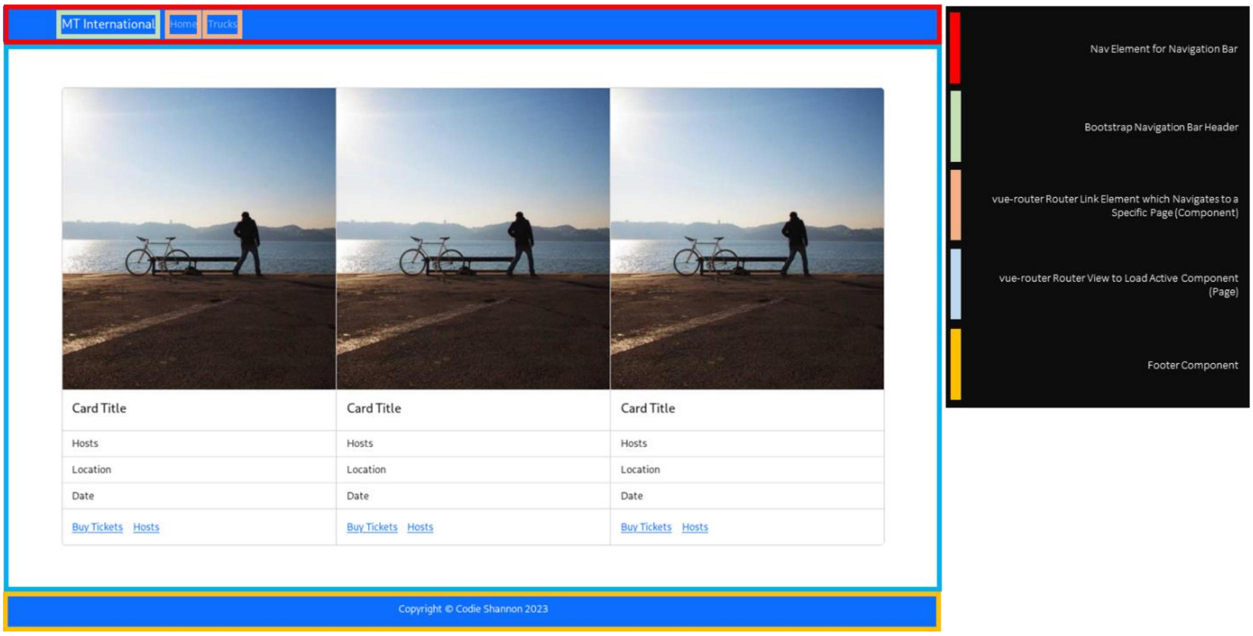
Create Bootstrap Collapsible Navigation Bar Container

Create Bootstrap Navigation Bar Container

Create vue-router Router Link Element which Navigates to a Specific Page (Component)

Create vue-router Router View to Load Active Component (Page)

Create Footer Component



MT International

Home

Trucks

Card Title

Hosts

Location

Date

Buy Tickets Hosts

Card Title

Hosts

Location

Date

Buy Tickets Hosts

Card Title

Hosts

Location

Date

Buy Tickets Hosts

Copyright © Codie Shannon 2023

🟥 Nav Element for Navigation Bar

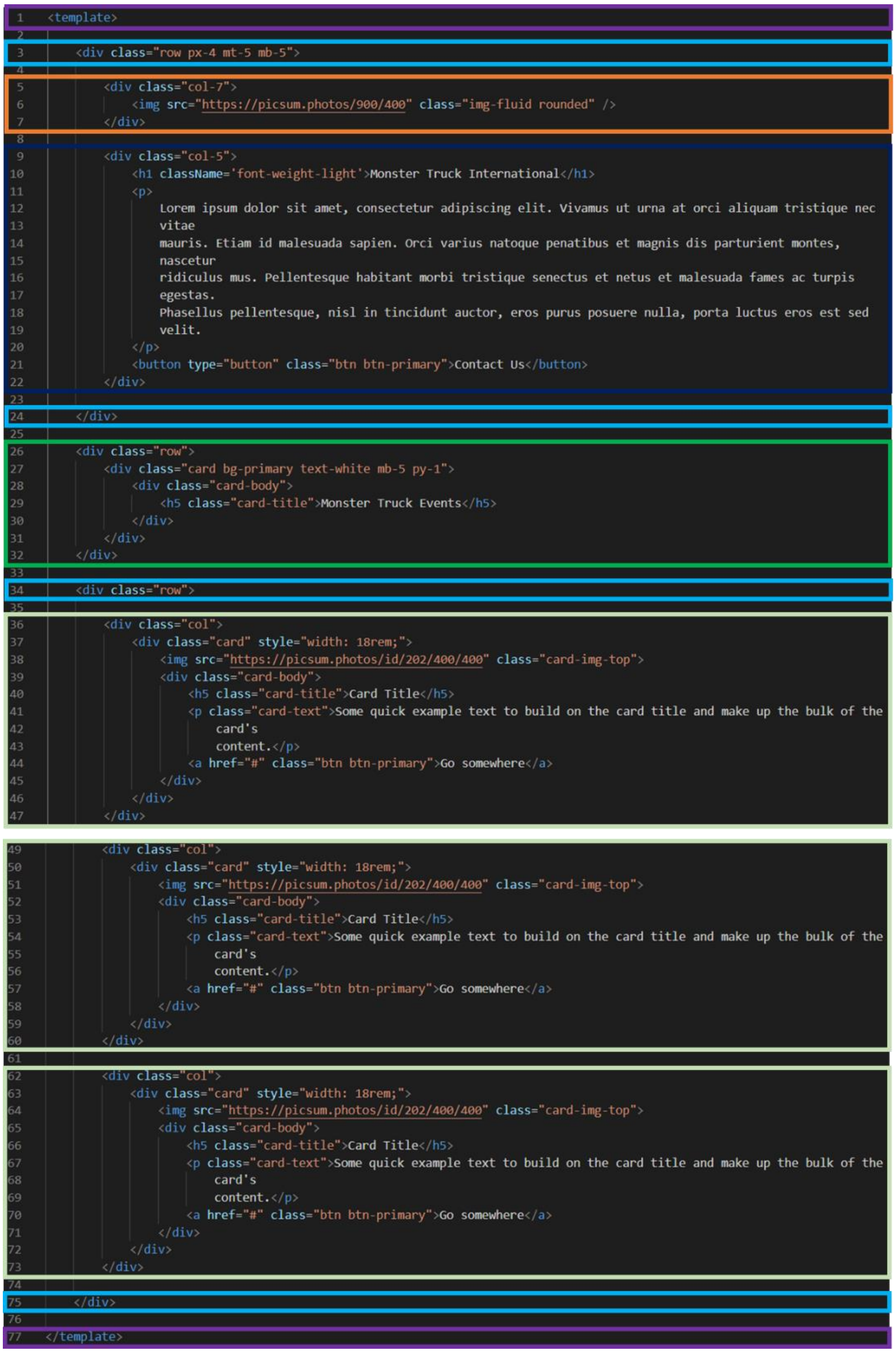
🟩 Bootstrap Navigation Bar Header

🟧 vue-router Router Link Element which Navigates to a Specific Page (Component)

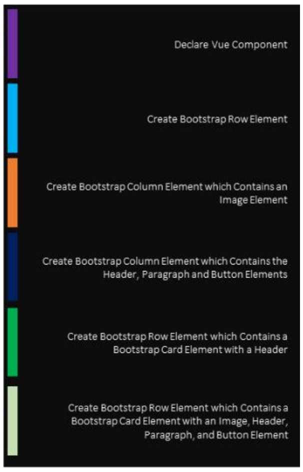
🟦 vue-router Router View to Load Active Component (Page)

🟨 Footer Component

VI. Home Component



```
<template>

    <div class="row px-4 mt-5 mb-5">

        <div class="col-7">
            <img src="https://picsum.photos/900/400" class="img-fluid rounded" />
        </div>

        <div class="col-5">
            <h1 className='font-weight-light'>Monster Truck International</h1>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus ut urna at orci aliquam tristique nec
                vitae
                mauris. Etiam id malesuada sapien. Orci varius natoque penatibus et magnis dis parturient montes,
                nascetur
                ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis
                egestas.
                Phasellus pellentesque, nisl in tincidunt auctor, eros purus posuere nulla, porta luctus eros est sed
                velit.
            </p>
            <button type="button" class="btn btn-primary">Contact Us</button>
        </div>

    </div>

    <div class="row">
        <div class="card bg-primary text-white mb-5 py-1">
            <div class="card-body">
                <h5 class="card-title">Monster Truck Events</h5>
            </div>
        </div>
    </div>

    <div class="row">

        <div class="col">
            <div class="card" style="width: 18rem;">
                <img src="https://picsum.photos/id/202/400/400" class="card-img-top">
                <div class="card-body">
                    <h5 class="card-title">Card Title</h5>
                    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the
                        card's
                        content.</p>
                    <a href="#" class="btn btn-primary">Go somewhere</a>
                </div>
            </div>
        </div>

        <div class="col">
            <div class="card" style="width: 18rem;">
                <img src="https://picsum.photos/id/202/400/400" class="card-img-top">
                <div class="card-body">
                    <h5 class="card-title">Card Title</h5>
                    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the
                        card's
                        content.</p>
                    <a href="#" class="btn btn-primary">Go somewhere</a>
                </div>
            </div>
        </div>

        <div class="col">
            <div class="card" style="width: 18rem;">
                <img src="https://picsum.photos/id/202/400/400" class="card-img-top">
                <div class="card-body">
                    <h5 class="card-title">Card Title</h5>
                    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the
                        card's
                        content.</p>
                    <a href="#" class="btn btn-primary">Go somewhere</a>
                </div>
            </div>
        </div>

    </div>

</template>
```



🟪 Declare Vue Component

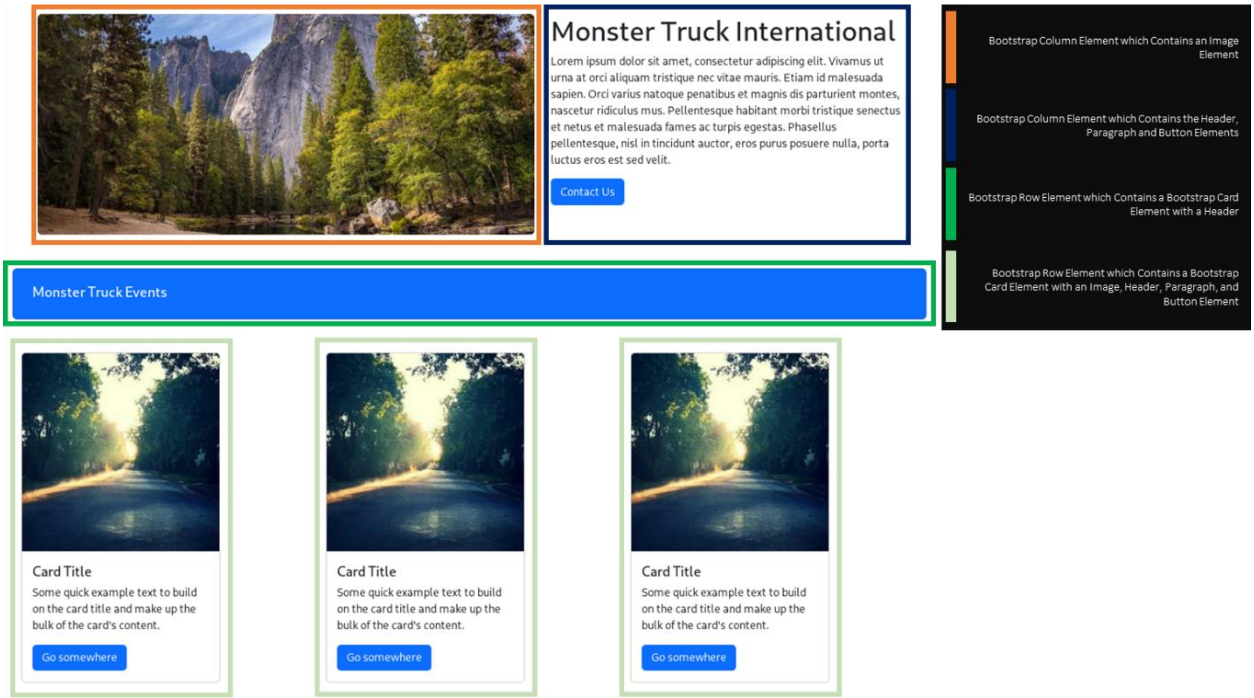
🟦 Create Bootstrap Row Element

🟧 Create Bootstrap Column Element which Contains an Image Element

■ Create Bootstrap Column Element which Contains the Header, Paragraph and Button Elements

🟩 Create Bootstrap Row Element which Contains a Bootstrap Card Element with a Header

▨ Create Bootstrap Row Element which Contains a Bootstrap Card Element with an Image, Header, Paragraph, and Button Element



Monster Truck International

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus ut urna at orci aliquam tristique nec vitae mauris. Etiam id malesuada sapien. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Phasellus pellentesque, nisl in tincidunt auctor, eros purus posuere nulla, porta luctus eros est sed velit.

Card Title

Some quick example text to build on the card title and make up the bulk of the card's content.

Card Title

Some quick example text to build on the card title and make up the bulk of the card's content.

Card Title

Some quick example text to build on the card title and make up the bulk of the card's content.

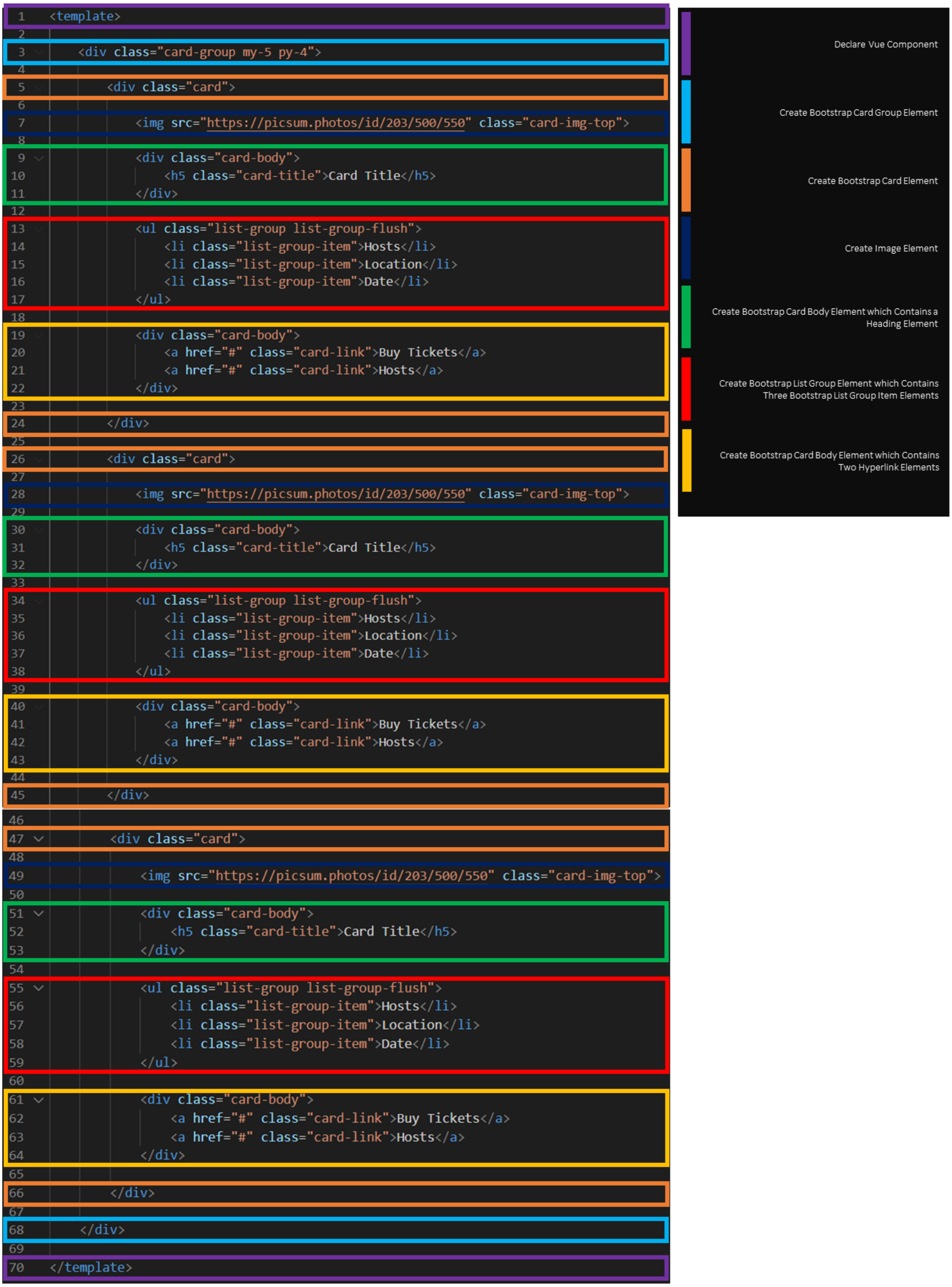
🟧 Bootstrap Column Element which Contains an Image Element

🟦 Bootstrap Column Element which Contains the Header, Paragraph and Button Elements

🟩 Bootstrap Row Element which Contains a Bootstrap Card Element with a Header

🟩 Bootstrap Row Element which Contains a Bootstrap Card Element with an Image, Header, Paragraph, and Button Element

VII. Trucks Component



```
<template>

    <div class="card-group my-5 py-4">

        <div class="card">

            <img src="https://picsum.photos/id/203/500/550" class="card-img-top">

            <div class="card-body">
                <h5 class="card-title">Card Title</h5>
            </div>

            <ul class="list-group list-group-flush">
                <li class="list-group-item">Hosts</li>
                <li class="list-group-item">Location</li>
                <li class="list-group-item">Date</li>
            </ul>

            <div class="card-body">
                <a href="#" class="card-link">Buy Tickets</a>
                <a href="#" class="card-link">Hosts</a>
            </div>

        </div>

        <div class="card">

            <img src="https://picsum.photos/id/203/500/550" class="card-img-top">

            <div class="card-body">
                <h5 class="card-title">Card Title</h5>
            </div>

            <ul class="list-group list-group-flush">
                <li class="list-group-item">Hosts</li>
                <li class="list-group-item">Location</li>
                <li class="list-group-item">Date</li>
            </ul>

            <div class="card-body">
                <a href="#" class="card-link">Buy Tickets</a>
                <a href="#" class="card-link">Hosts</a>
            </div>

        </div>

        <div class="card">

            <img src="https://picsum.photos/id/203/500/550" class="card-img-top">

            <div class="card-body">
                <h5 class="card-title">Card Title</h5>
            </div>

            <ul class="list-group list-group-flush">
                <li class="list-group-item">Hosts</li>
                <li class="list-group-item">Location</li>
                <li class="list-group-item">Date</li>
            </ul>

            <div class="card-body">
                <a href="#" class="card-link">Buy Tickets</a>
                <a href="#" class="card-link">Hosts</a>
            </div>

        </div>

    </div>

</template>
```

🟪 Declare Vue Component

🟦 Create Bootstrap Card Group Element

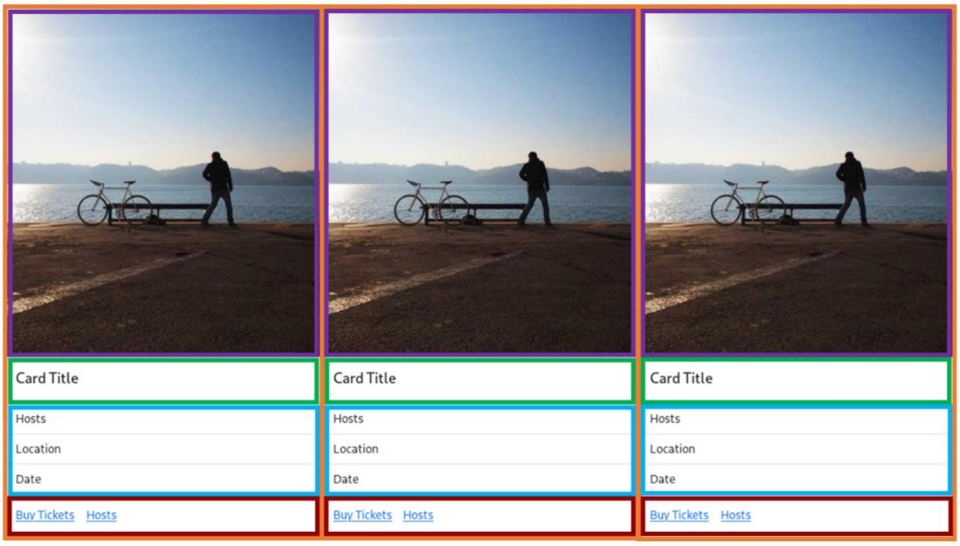
🟧 Create Bootstrap Card Element

■ Create Image Element

🟩 Create Bootstrap Card Body Element which Contains a Heading Element

🟥 Create Bootstrap List Group Element which Contains Three Bootstrap List Group Item Elements

🟨 Create Bootstrap Card Body Element which Contains Two Hyperlink Elements



Card Title

Hosts

Location

Date

Buy Tickets Hosts

Card Title

Hosts

Location

Date

Buy Tickets Hosts

Card Title

Hosts

Location

Date

Buy Tickets Hosts

🟧 Bootstrap Card Element

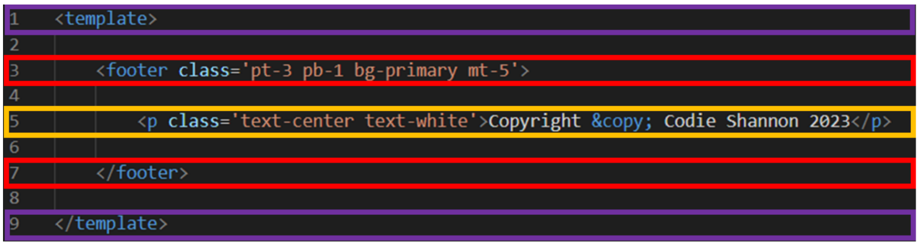
🟪 Image Element

🟩 Bootstrap Card Body Element which Contains a Heading Element

🟦 Bootstrap List Group Element which Contains Three Bootstrap List Group Item Elements

🟥 Bootstrap Card Body Element which Contains Two Hyperlink Elements

VIII. Footer Component

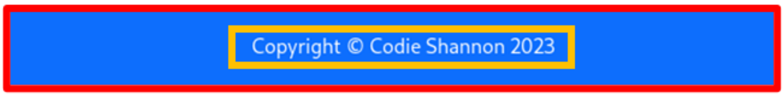


```
<template>

    <footer class='pt-3 pb-1 bg-primary mt-5'>

        <p class='text-center text-white'>Copyright &copy; Codie Shannon 2023</p>

    </footer>

</template>
```

🟪 Declare Vue Component

🟥 Create Footer HTML Element with Bootstrap Styling

🟨 Create Paragraph HTML Element with Bootstrap Styling



Copyright © Codie Shannon 2023

🟥 Footer HTML Element with Bootstrap Styling

🟨 Paragraph HTML Element with Bootstrap Styling

E. References

Adam, J. (2023, March 27). Comparing the technical strengths, weaknesses and use cases of the most popular JS frameworks and libraries. *KruscheCompany*. https://kruschecompany.com/ember-jquery-angular-react-vue-what-to-choose/#React_strengths

AltexSoft. (2022, September 6). *The good and the bad of Angular development*. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

Ascendle. (2021, February 22). *Single-Page Applications: pros and cons.* https://ascendle.com/ideas/single-page-applications-pros-and-cons/#:~:text=Search%20engines%20find%20it%20harder,files%20are%20search%20engine%2Dfriendly.

Bhushan, A. (2023, March 14). Svelte vs React: features, performance, and more. *Kinsta*. https://kinsta.com/blog/svelte-vs-react/

ĆWIK, P. (2021, January 15). Single-Page Application: how SPA works and how it differs from MPA. *Kissdigital*. https://kissdigital.com/blog/single-page-application-how-spa-works-and-how-it-differs-from-mpa#:~:text=The%20first%20single%2Dpage%20applications,used%20to%20create%20the%20frontend.

Dhaduk, H. (2022, February 6). Preact vs. React: a quick comparison. *Hackernoon*. https://hackernoon.com/major-differences-between-preact-and-react-which-one-should-you-use

Downey, J. (2013, December 4). *The RESTFulness of Single Page Applications.* https://jimdowney.net/2013/12/04/the-restfulness-of-single-page-applications/

Egghead. (n.d.). *WTF is JSX?* https://egghead.io/learn/react/beginners/wtf-is-jsx

Gillis, A. (2020). *Thick Client (Fat Client).* What Is. https://www.techtarget.com/whatis/definition/fat-client-thick-client

Gillis, A. (2021). *Thin Client (Lean Client)*. What Is. https://www.techtarget.com/searchnetworking/definition/thin-client

GraffersID. (2023, February 6). *SolidJS vs React: which is better in 2023?* https://graffersid.com/solidjs-vs-react-which-is-better/

HEAVY AI. (2022). *Client-Server*. https://www.heavy.ai/technical-glossary/client-server

Information Processing Society of Japan. (n.d.). *Unix Servers – Brief History*. https://museum.ipsj.or.jp/en/computer/unix/history.html#:~:text=Client%2Dserver%20systems%20began%20to,multiple%20workstations%20or%20personal%20computers.

JavaTpoint. (2021). *Pros and Cons of ReactJS*. https://www.javatpoint.com/pros-and-cons-of-react

JavaTpoint. (2021). *React vs. Svelte*. https://www.javatpoint.com/react-vs-svelte

KnowledgeHut. (2022, November 25). *What are the pros and cons of React*. https://www.knowledgehut.com/blog/web-development/pros-and-cons-of-react

Lawson, K. (2022, September 17). What are Single Page Applications and why do people like them so much? *Bloomreach*. https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application?spz=article_var+navigation_orig

OpenJS Foundation. (2023). *What is jQuery?* https://jquery.com/

Panchal, R. (2021, January 13). Pros and cons of Preact JS. *Rlogical*. https://www.rlogical.com/blog/pros-and-cons-of-preact-js/

Red Hat. (2020, May 8). *What is a REST API?* https://www.redhat.com/en/topics/api/what-is-a-rest-api#rest

Sasidharan, D. (2021, March 18). *What about Svelte? Should you care!* Deepu Tech. https://deepu.tech/what-about-svelte/

Sharma, I. (2021). *What is Vue JS? The pros and cons of Vue.js*. Tatvasoft. https://www.tatvasoft.com/outsourcing/2021/10/what-is-vue-js-and-its-benefits.html

Shikht, I. (2023). *Angular development: pros and cons*. Exoft. https://exoft.net/angular-pros-and-cons/

Slemzina, Y. (2023). Pros and cons of Angular 2+ framework. *Bekey*. https://bekey.io/blog/pros-and-cons-of-angular-two-plus-framework

SmartBear. (2020, January 2). SOAP vs REST. *What's the difference?*
https://smartbear.com/blog/soap-vs-rest-whats-the-difference/

Vue.js. (n.d.). *Comparison with other frameworks*.
https://v2.vuejs.org/v2/guide/comparison.html?redirect=true