**Background research**

Potential Frameworks

* Angular (Google)
* React (Facebook)
* VueJs
* Plain HTML, CSS, and JavaScript (do not use a framework)
* Django

Potential Languages:

* JavaScript
* TypeScript
* Python

Potential Styling libraries:

* Bootstrap
* Foundation
* Bulma

**What is Angular?**

It is a Typescript based framework for building “single-page client applications using HTML and TypeScript” [4]. It is an open-source project started by a google team.

**What is React?**

It is a Component based JavaScript framework for creating complex UI’s. “It is maintained by Facebook and a community of individual developers and companies” [5]. And is one of the more popular frameworks. And supports many 3rd party libraries and modules.

**What is VueJs?**

It is a non-monolithic framework, normally used in conjunction with other libraries to build a fully-fledged Modern website it “is a progressive framework for building user interfaces” [6]. And the “The core library is focused on the view layer only” [6].

**What is Django?**

“Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design” [7]. It used python as its language of choice, making it easy to implement.

**What is Bootstrap, Foundation and Bulma?**

**Bootstrap** is a “a collection of HTML, CSS, and JavaScript tools for creating and building web pages and web applications” [8]. It makes it quick and easy to produce a good-looking website and allows the developer to spend more time working on functionality, than worrying about the style of components.

**Foundation is a** “Foundation is a family of responsive front-end frameworks that make it easy to design beautiful responsive websites, apps and emails that look amazing on any device [9]”. And claims to be the most advanced responsive front-end framework. It is also an open-source project hosted on Github. It is composed of HTML CSS and JavaScript.

**Bulma** is a “free, open-source framework that provides ready-to-use frontend components” [10]. It is a relatively new library. However, this technology is easy to use and its implementation and is similar to a vanilla bootstrap implementation.

**What is JavaScript?**

Mozilla defines JavaScript as **“a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.”** [12]. It is also the language of choice for the module then general. It is largely used in web development.

**What is Python?**

Python.org describes its language as **“an interpreted, object-oriented, high-level programming language with dynamic semantics”** [13]. It is easy to use with simple syntax and has a large range of 3rd party modules and libraries. Python is generally only used for font-end web development injunction with Django, however, is extensively used for backend processes.

**What is TypeScript?**

In short TypeScript is JavaScript but enforced static typing. The TypeScript website describes its language as **“an open-source language which builds on JavaScript, one of the world’s most used tools, by adding static type definitions.”** [14]. It is rising in popularity within the web-development industry.

**Justification of the selected technology.**

Selected technologies:

* **React** (Framework)
* **Bootstrap** 4 (Styling CSS classes/Components)
* ‘**react-bootstrap**’ (A Combination of the two libraries above)
* **JavaScript**

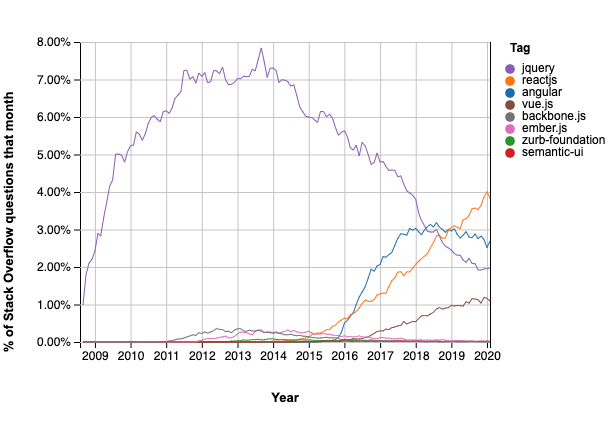
**Reasonings per technology.**

**Why React?**

Firstly, React claims:

**“React makes it painless to create interactive UIs” [1]**

As it did not make sense to create a website using plain html/CSS/JavaScript and we were permitted to use frameworks. As there has been no reason to do so for many years when robust frameworks have been developed to improve quality and security of front-end technologies. React was chosen as it would allow the team to broaden their skills through learning a new front-end framework. We choose ReactJS specifically as it is now one for the most popular frameworks for frontend development as we can see from this graph from stack overflow highlighting the increase of questions related to ReactJS. It also seemed that it would be good opportunity for the team to increase their employability by learning a popular web framework.

**Why React over Angular, Vuejs, Django?**

We decided not to use Angular as it is typescript based, as the module we are doing teaches JavaScript, we decided it would be more familiar to the team to use a framework that uses the JavaScript Language.

We decided not to use Vuejs either despite it also using JavaScript, because of the vast amount of help and support on the internet targeting ReactJS. Due to its popularity and the number of 3rd party libraries it supports.

Django was immediately disregarded as it lies so far out of the scope of the module that we decided not to use it.

**Why Bootstrap 4?**

Bootstrap 4 was chosen as it enables an inexperienced web-development team to produce a visually appealing websites, in the case where the team processes little CSS experience or knowledge, as only 1 member of the team had ever done any frontend web development before. It allowed us to spend more time working on the dynamic elements of a Dynamically Authored Website, than spending 90% of our time trying to fix CSS issues. Importantly Bootstrap is well developed and learning resources are in high supply. And it is implemented in react-bootstrap which is discussed below.

**Why ‘react-bootstrap’ a special combination of ReactJS and Bootstrap 4?**

Firstly ‘react-bootstrap’ claims to be:

**“The most popular front-end framework” (3)**

This again will allow us to expand our employability, as many companies use this framework that we now have experience with.

Main reason to use this library was, that it simplified the relationship between React and Bootstrap, but also still left open the option to use Bootstrap as it was originally designed or too even write totally custom CSS. This is an extremely strong tool for the in-experienced web developer. And again, provided us more time to work on key dynamic elements rather than getting bogged down on CSS issues. The wide range of documentation and guides make this a good choice for a University Project, as we have do not have direct experience with these frameworks so strong resources and guides are key.

**Why JavaScript?**

JavaScript was chosen as the module content was taught in JavaScript. And most web development is using JavaScript. Despite the rise in popularity of Typescript for example. We decided to use JavaScript as React uses it by default and the module was taught in JavaScript, meaning team members we equipped to use the language within the coursework.

**Implementation**

**Usage of 3rd Part libraries and Resources**

We implement the following libraries and do not take credit for the code within them:

**Awesome Fonts**

    "@fortawesome/fontawesome-svg-core": "^1.2.34",

    "@fortawesome/free-brands-svg-icons": "^5.15.2",

    "@fortawesome/free-regular-svg-icons": "^5.15.2",

    "@fortawesome/free-solid-svg-icons": "^5.15.2",

    "@fortawesome/react-fontawesome": "^0.1.14",

Awesome fonts were used to implement some icons and fonts.

**React Libraries**

    "react": "^17.0.1",

    "react-bootstrap": "^1.5.0",

    "react-cookie": "^4.0.3",

    "react-dom": "^17.0.1",

    "react-google-maps": "^9.4.5",

    "react-media": "^1.10.0",

    "react-router-dom": "^5.2.0",

    "react-scripts": "^4.0.3",

    "react-social-embed": "0.0.3",

    "react-twitter-embed": "^3.0.3",

    "@react-google-maps/api": "^2.1.1",

The most import libraries in the react section are react 17.0.1 and react-bootstrap 1.5.0

**React**: was the main framework used to build the website.

**React-Bootstrap** was a library that combined react and bootstrap that help us build a visually appealing website, with limited experience in web development.

**React-router-dom + react-dom:** was used to implement dynamic page generation/navigation.

**React-twitter-embed**: was used to embed a twitter post within the website.

**React-social-embed**: was used to embed a Facebook post into the website.

**React-google-maps** was used to implement google API integrations.

**React-scripts**: provided local development helper scripts.

**React-cookie**: provided functionality to create and read cookies from the user’s browser.

Note other libraries are present in package.json but where either created by the default react app or are not being used or where for local development.

**Changes to original design.**

**Header bar**

The original header bars shape was slightly changed, this was done as the new design with much neater and pleasing to the eye once it was developed than that design presented in the wireframe.

**Banner**

We added a web banner as it was mentioned in the specification, this was not in the original design, however.

**Addition Pages/Functionality**

We add several additional pages/functionalities.

* Addition of a Registration and Login Page.
* Addition of a Profile Page.
* Addition of a Find us Page.
* Addition of a Contact Page.
* Registration Success Page.

We added these pages to fill the website more and provided a better or more informative experience for the user.

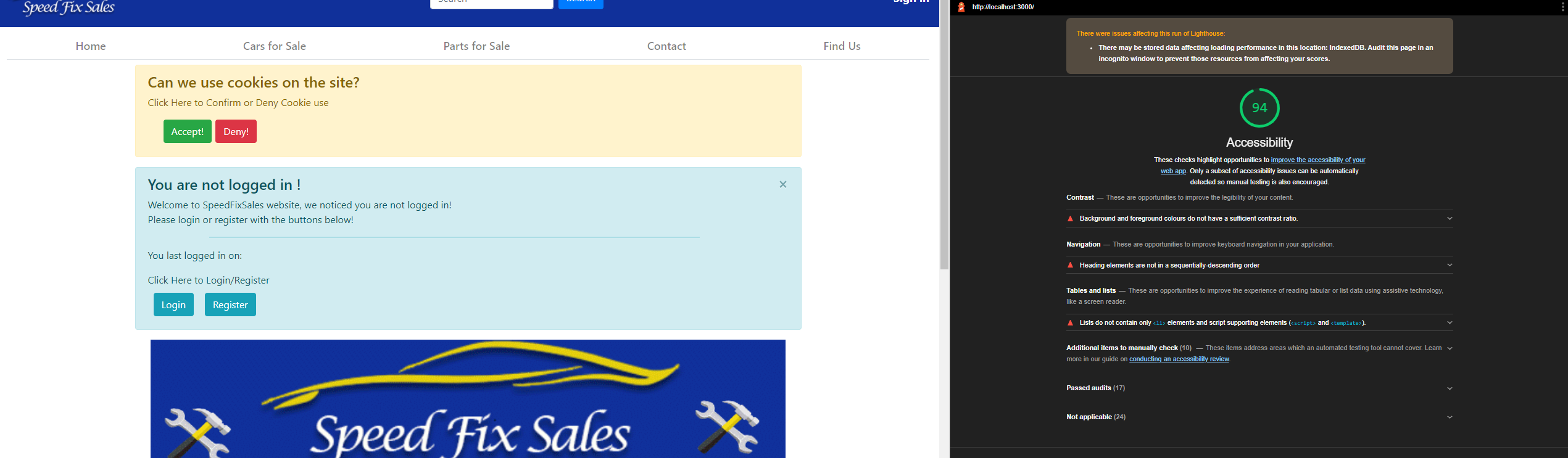
**Standards Compliance**

To adhere to standards compliance, we primarily used a tool called lighthouse, which allowed us to run reports and these reports would highlight compliance issues, we broke these down into compliance with accessibility and compliance with performance. However, **Accessibility** was our main concern to fix first and that is why our **Accessibility** score are much higher as we had to meet **W3C** **Accessibility** guidelines and compliance.

We managed to meet around 95% with W3C compliance on every page (see Appendix 1), however it did not like our design colors choices saying they were too similar and did not have enough contrast with the background. So, we were not able to fix every compliance issue. While maintaining the original design, but 95% is a strong rating for our site to have when it comes to **W3C** **Accessibility** guidelines.

**Accessibility Compliance**

To monitor accessibility, we used a tool within Google Chrome Called Lighthouse here is a typical example of its use:



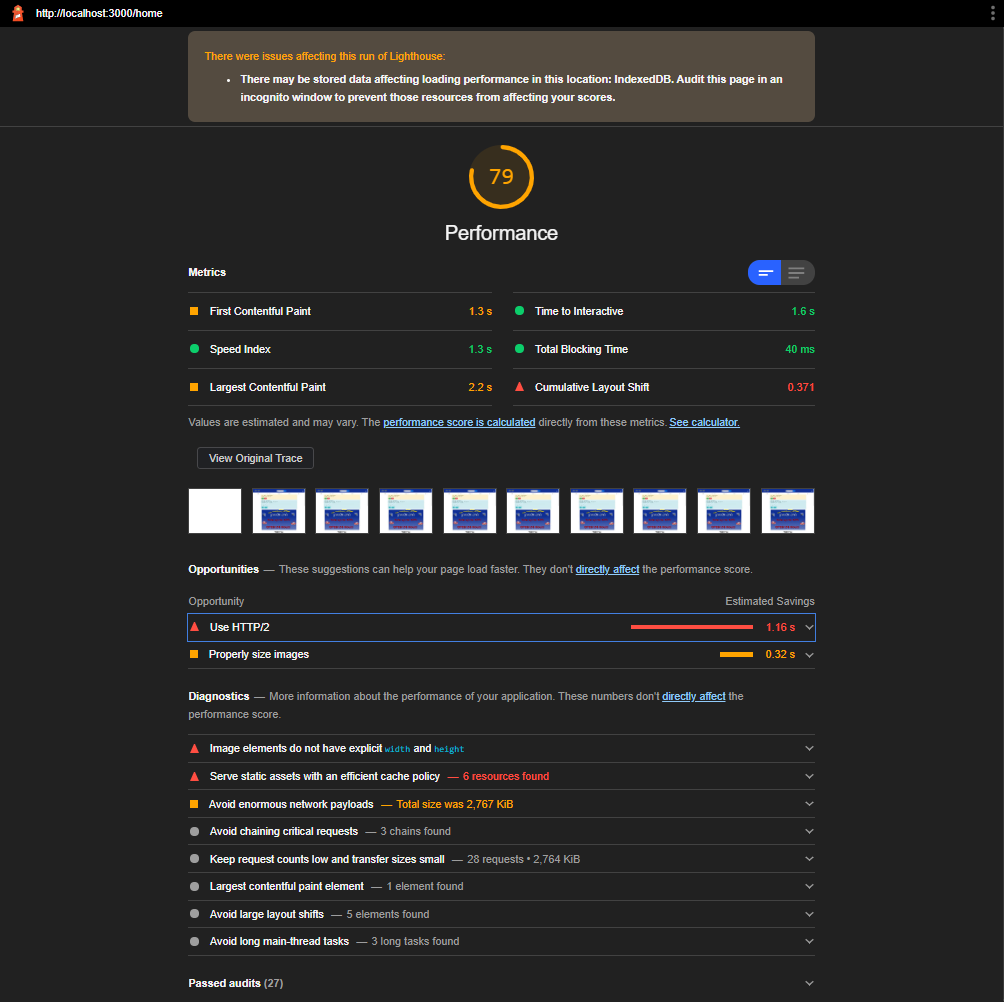
As you can see there, we have an accessible score of 94 on the home page, below it highlights all our issues that we should resolve. We used this tool to **highlight** as many accessibility issues as we could and correct them.

However, we were unfortunately unable to fix all of them, but a lot of the improvements recommended are more of a recommendation rather and a strict compliance error.

To see the full **Accessibility** Report see Appendix 1.

**Performance Compliance**

Again, we used the google tool Lighthouse, here is an example of a performance report. As you can see below it high lights some speed savings that could be made, however we did run into trouble trying to fix all of the issued highlighted but managed to correct many of them, we focused less onperformance, as **Accessibility/WC3 complience** had high priority.

See the full **Performance** Report in the Appendix 2

**Gathering and Conditioning data**

**Cookies/Local Storage**

Our website implements cookies. The first implementation was that the website will ask the user to accept/or deny the use of cookies on the site. If the cookie does not exist a prompt, we be displayed to the user. Once the user makes their choice, subsequent returns the website will not display the prompt.

Second for those users who have logged in before the site uses cookies to remember the username, so that a welcome message can be displayed to user along with the date that they last logged in.

**Filters.**

Our website performs both the gathering and conditioning of data on many of the Pages. Both the Car and Parts Listing Page have several input filters that the user can select, to narrow down the displayed items on the page. Ranging from price filters to milage filters and many more. These filters will dynamically update the page as the user interactions with each filter component.

**User Account and login access.**

There is a registration page that allows the user to create an account and a follow up login page allowing the user to login with the created account, each account will also have access to the profile page where the user’s key information will be displayed. These combinations of pages provide significant gathering and conditioning of data, as data must be input, stored, and retrieved on other pages of the site.

**Dynamic Content**

All the cars and part within the website are loaded from a variable, this makes it easy to add cars or parts to the garage website. No further work is required, than adding the item data. This make is easy to add new content to the website and that content will then be immediately render in the appropriate places within the website.

As filters previously mentioned are interacted with the content will dynamically change to reflect the user selected filters.

An improvement to our website would be to add an Admin section with and add the ability to add or remove cars or parts, however as databases were not allowed within this module coursework, we decided not to Mock this feature, as the work would be significantly more to mock this feature than using a database itself.

**Banner**

The home page contains the websites banner, it is a simple design stating the websites name, highlighting that a sale is currently on and informs the user than the garage us open 24/7.

**Image effects** / **Animation**

The website contains many carousels, that will dynamically cycle through each item in the carousel, image effect buttons overlap the image to allow the user to cycle manually also if desired. Using animations from react-boostrap.

**Dynamic Validation**

We have implemented dynamic validation in the registration page, it will list all the invalid inputs and ask the user too to correct the validation errors. The error prompt alert will dynamically change in real time to reflect the remaining errors that user must correct.

**Dynamic interactive design and event handling techniques**

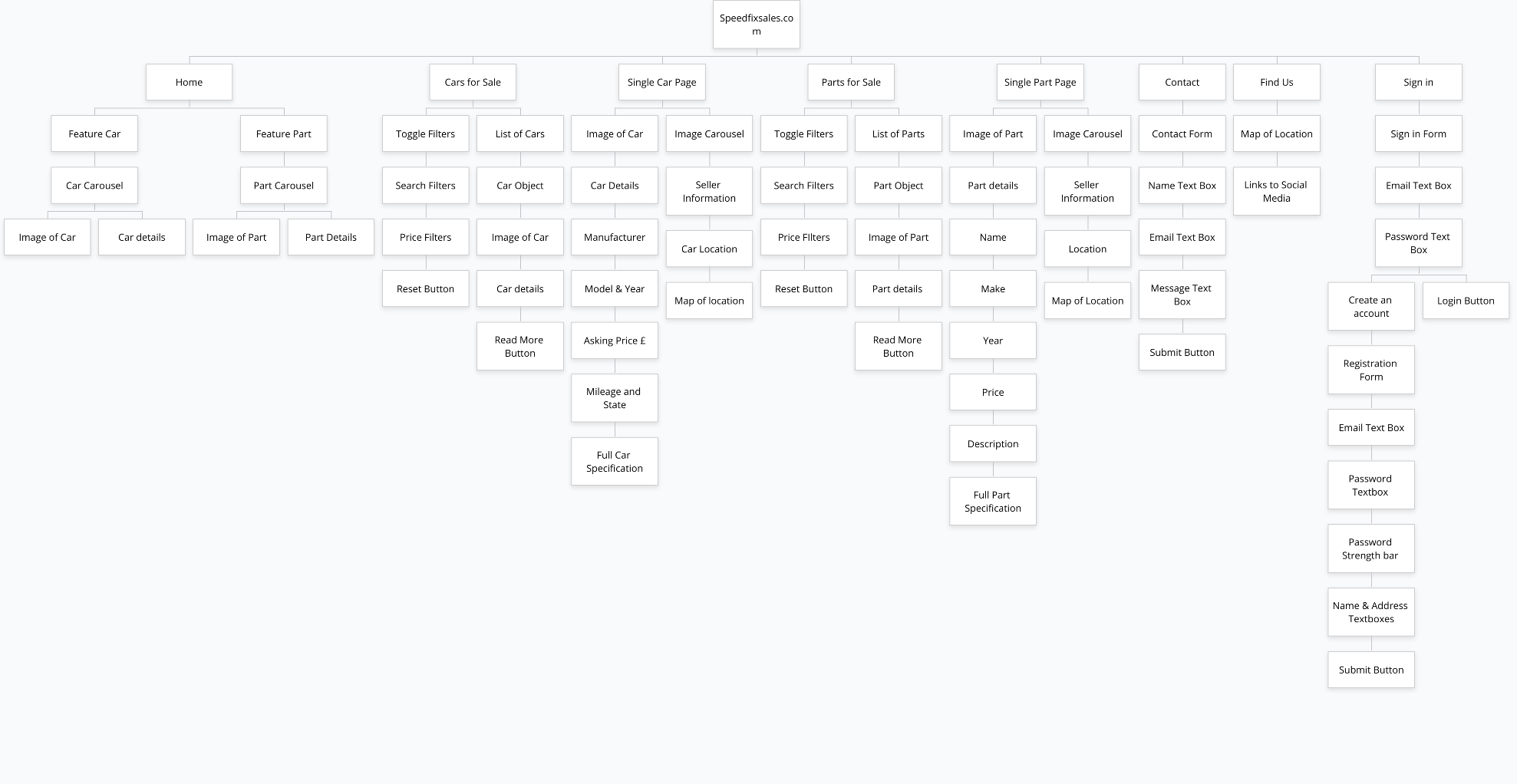
The Website is fully responsive on every page and implements many even handling techniques.

These include the previously mentioned Login Registration data collection and the Listing pages filters. Each “button”, “slider” or other component utilizes React Hooks to update state and re-render components based on the state of the altered variables, without the page refreshing. For Example, this means that when a user types a password into the Registration Page, there is a progress bar that dynamically updates the strength of the password as the user types without refreshing or navigating to other pages.

This is also the case when a user re-sizes the webpage the content will be dynamically formatted in a responsive manner.

**Site Map**

Above is our site map, consisting of all the pages we have created and their flow. As you can see below, we have the Home, Cars for Sale, Single Car Page, Parts for Sale, Single Part Page, Contact, Find Us and a Sign page which holds access to the Register Page.

****

**Consideration of performance and optimization.**

To compare our websites performance, we ran it thought the following tool:

<https://www.websitecarbon.com/>

Which allows us to measure the efficiency of our website as compared to all websites globally. Basically, calculating is carbon output.

“An average website produces 4.61 grams of CO2 for every page view. For websites that have an average of 10,000 page views per month, that makes 553 kilograms of CO2 per year.” [11].

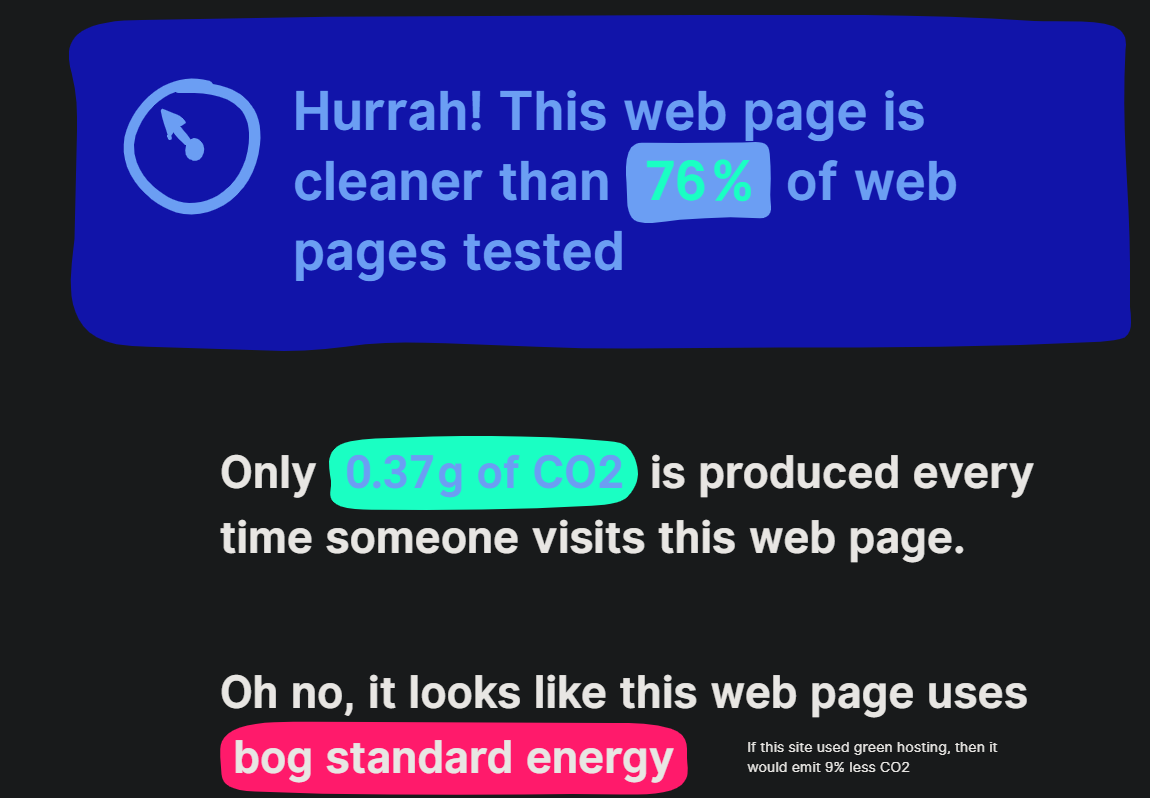
**React Production Build**

React itself has a large amount of built-in optimization features that a user itself does not have be aware of or implement.

React provides the functionality to compile the website into a production build. Production build will significantly speed up site and its efficiency.

**Our energy output/efficiency**

The **production build** of our website was cleaner/more efficiency than 76% of the web and only produced 0.37g of COD per visit.



As we mentioned in our video the Home pages had some hard coded values, in the components they imported, fixing this issue to use a loop instead would free up some memory usage.

Also, the images across the whole site could have been resolution processed to multiple sizes to reduce image load times, however this is rather complex for such a minimal speed increase.

Again, another improvement for efficiency would be to fix more of the performance issues highlighted by the lighthouse tool, however some of these fixes where out of the team’s level of experience and we had trouble understanding some of them.

Many of the issues we tried to fix either broke the layout of the website or cause a bigger performance issue. So, we had to reach a compromise and focused our time on WC3 compliance instead.

**Direct DOM Manipulation and the Virtual DOM (as alternative to DOM structure diagram as discussed with lecturer).**

**As we choose to use react** as our framework of choice, we would have had to additional work to see the DOM structure of a website that is using a Virtual DOM, so instead it was suggested that we discuss what a virtual DOM is and the benefits or negatives of using this system for website development.

**Firstly, what is a regular DOM?**

A **DOM** or **Document Object Model** in short is a “tree-like structure that organizes the elements on a web page and allows scripting languages to access them” [15]. This all happens within the browser’s memory. Basically, when there is a state change in the user interface/application the “Real DOM” gets updated. As the “Real DOM” is structured in a tree it will find be able to find the component/element it needs to change in a relatively short time, however, Mosh Hamedani explains that when this happens the “updated element and it’s children have to be re-rendered to update the application UI” [16]. As you could see if the state changes a lot, there is potentially a lot of re-rendering required within the regular DOM model, as all the children of the changed element require re-rendering. This is where the Virtual DOM comes in and improves this process and why most modern development move towards a virtual DOM, as direct DOM manipulation can slow things down as code academy explains “This slowness is made worse by the fact that most JavaScript frameworks update the DOM much more than they have to.” [15], wasting resources.

**How would you define a Virtual DOM and how does it work?**

The “virtual DOM“ Is becoming the standard among new web applications. The Virtual DOM is a in memory copy of the structure of the real DOM, it is not an exact copy however but more of a light-weight representation of the structure and properties of each DOM element/component. So instead of updating the “Real DOM” directly when state changes we manipulate the “Virtual DOM” instead.

**But how is this faster, sounds like a lot of overhead?**

Yes, there is a memory overhead but ...

The virtual DOM computes changes to the real DOM by comparing the difference in a process called “diffing” [15] or “Compute diff” [16]. This process makes better sense if you imagine it like a **batch** process, where multiple state updates in the virtual DOM can be rendered at once into the “Real DOM”, saving on multiple re-renders. Once the diff has been established, which could contain multiple state updates to the “Virtual DOM”, React or other virtual DOM frameworks, will update **ONLY** those elements that have changed.

For Example, consider a list:

<ul class=”example”>  
<li> Volkswagan </li>  
<li> Ford </li>  
<li> Skoda </li>  
</ul>

If you did not use a virtual DOM and directly manipulated the DOM by removing and item from the list such as “volkswagan”, the whole <ul class=”example”> would need to render itself and all the children <li></li> in the list, then deleting “ford” the same would occur.

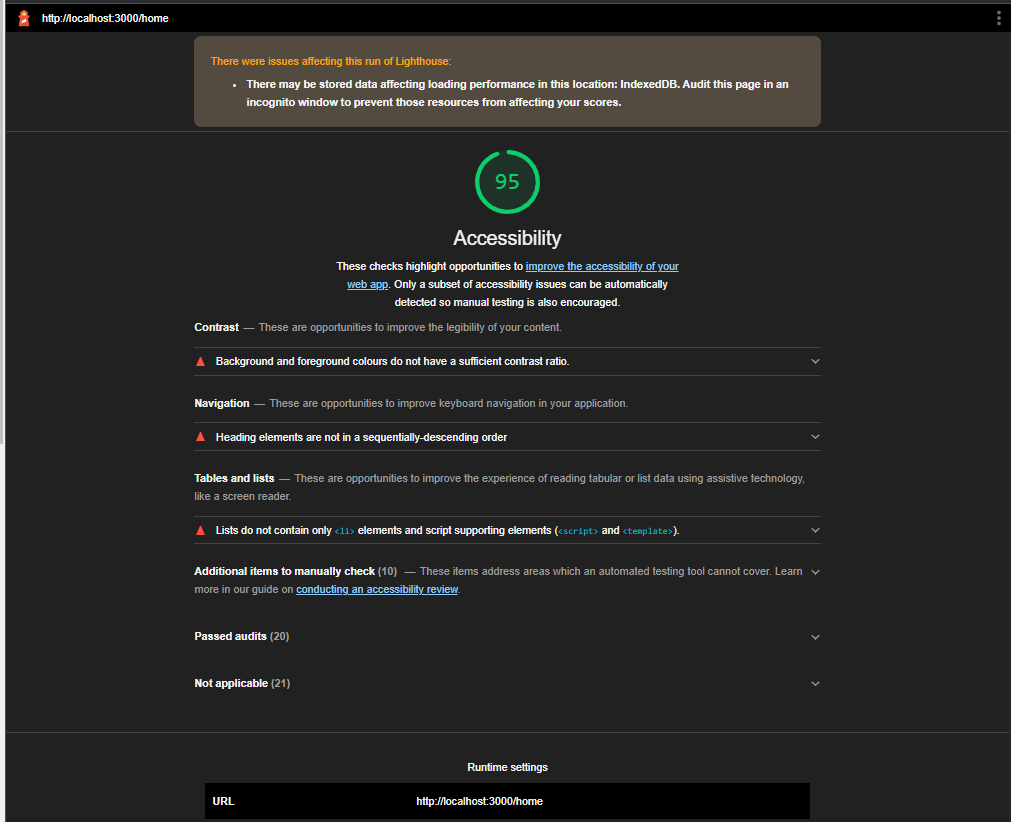
However, if you did this in the **Virtual DOM,** within the **batch** time span this list would only need to be re-rendered once, as both state changes would fall within a single batch DOM diff update. You can see by this simple example you have **a 100% efficiency savings**; however, this is not including the memory overhead and slight CPU usage required by the Virtual DOM, but this overhead is negligible to the energy savings a virtual DOM can result in, particularly when most modern device have plenty of RAM.

Hopefully this explanation is clear and not to technical and could be understood by the majority of readers with some software-based experience.

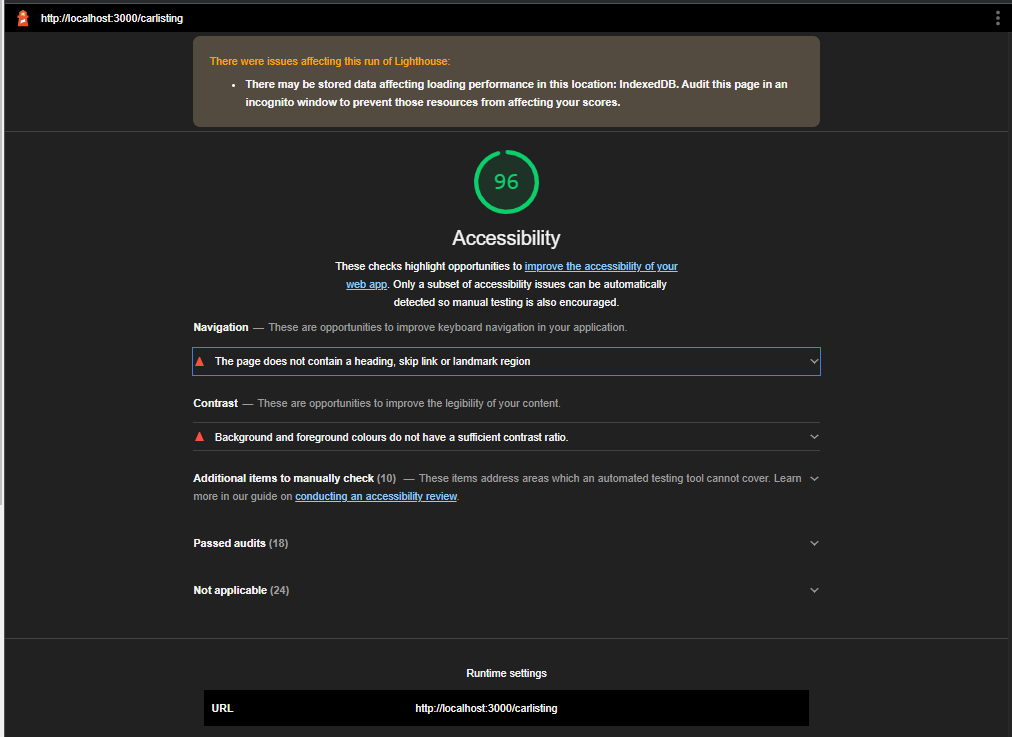
**Appendix 1 (Accessibility Reports from Lighthouse)**

We used lighthouse a google chrome tool and do not take credit for the report’s findings.

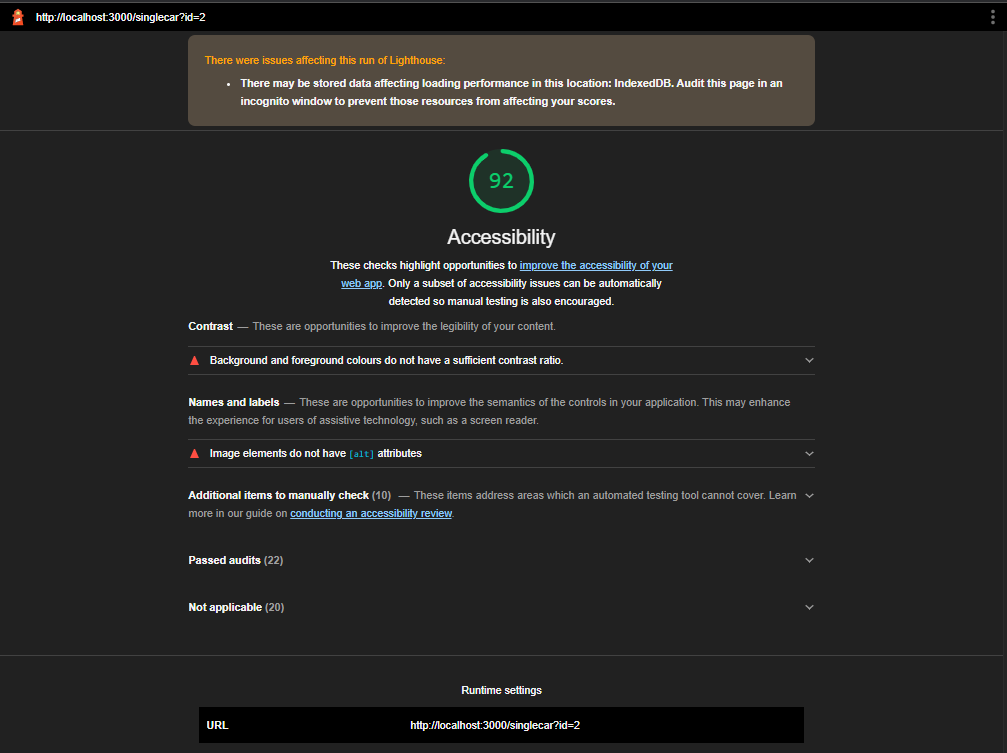
**Homepage Accessibility Report**



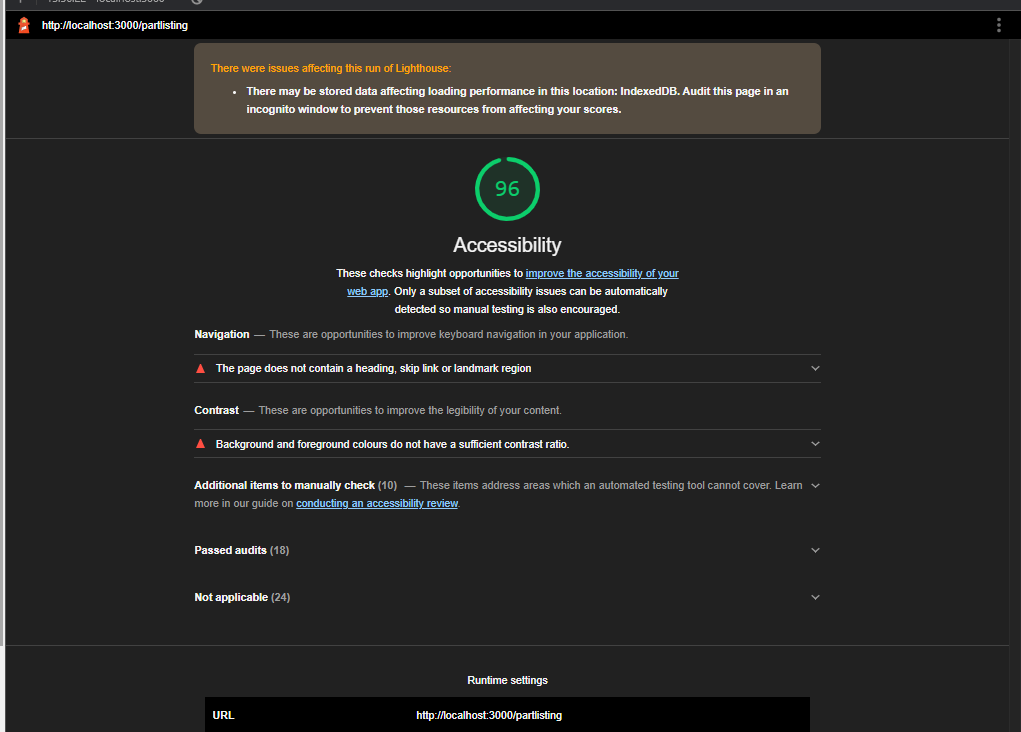
**Car Listing Accessibility Report**



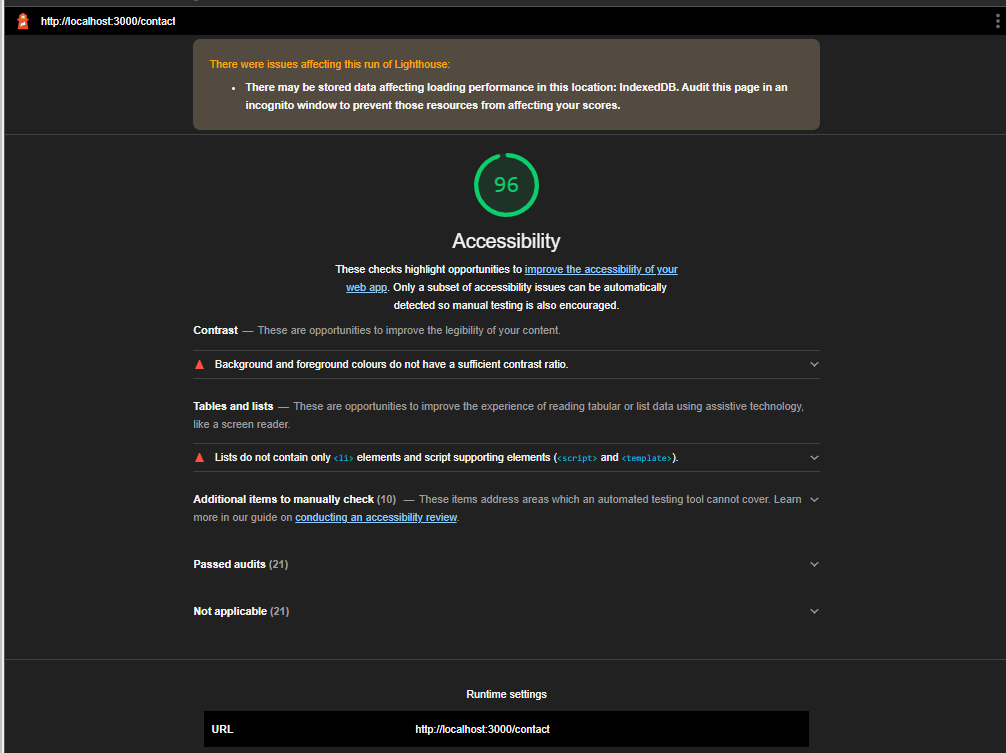
**Single Car Page Accessibility Report**



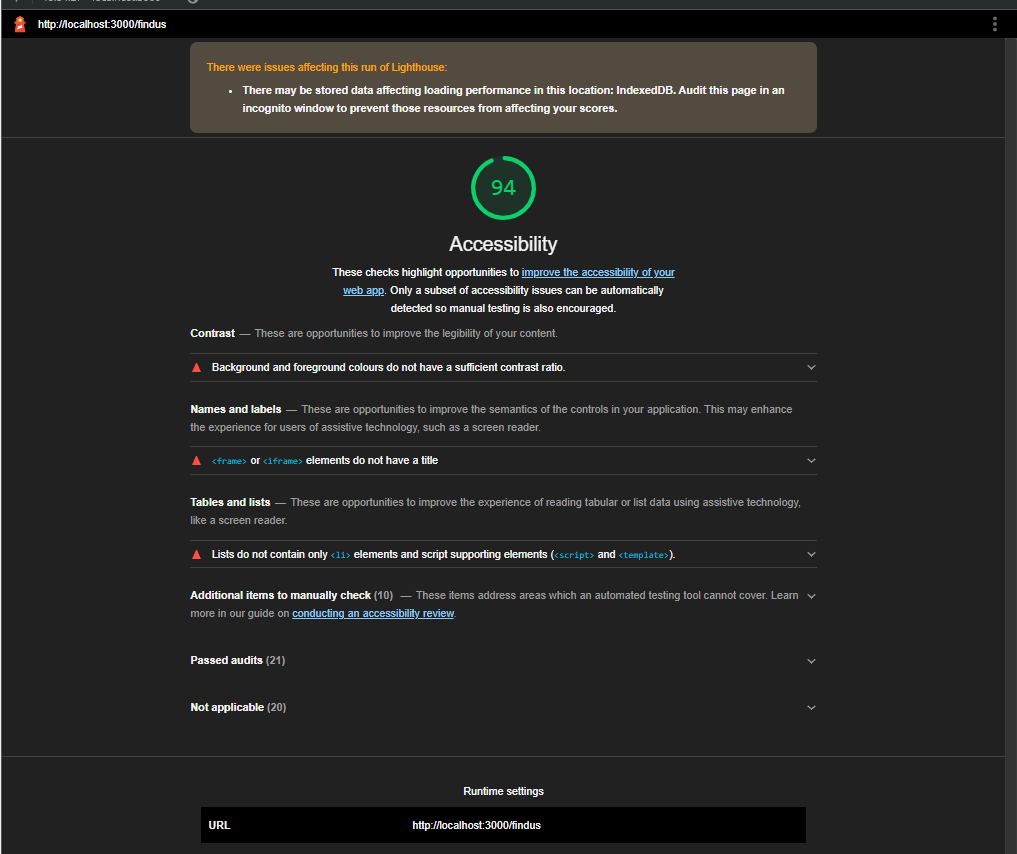
**Part Listing Accessibility Report**



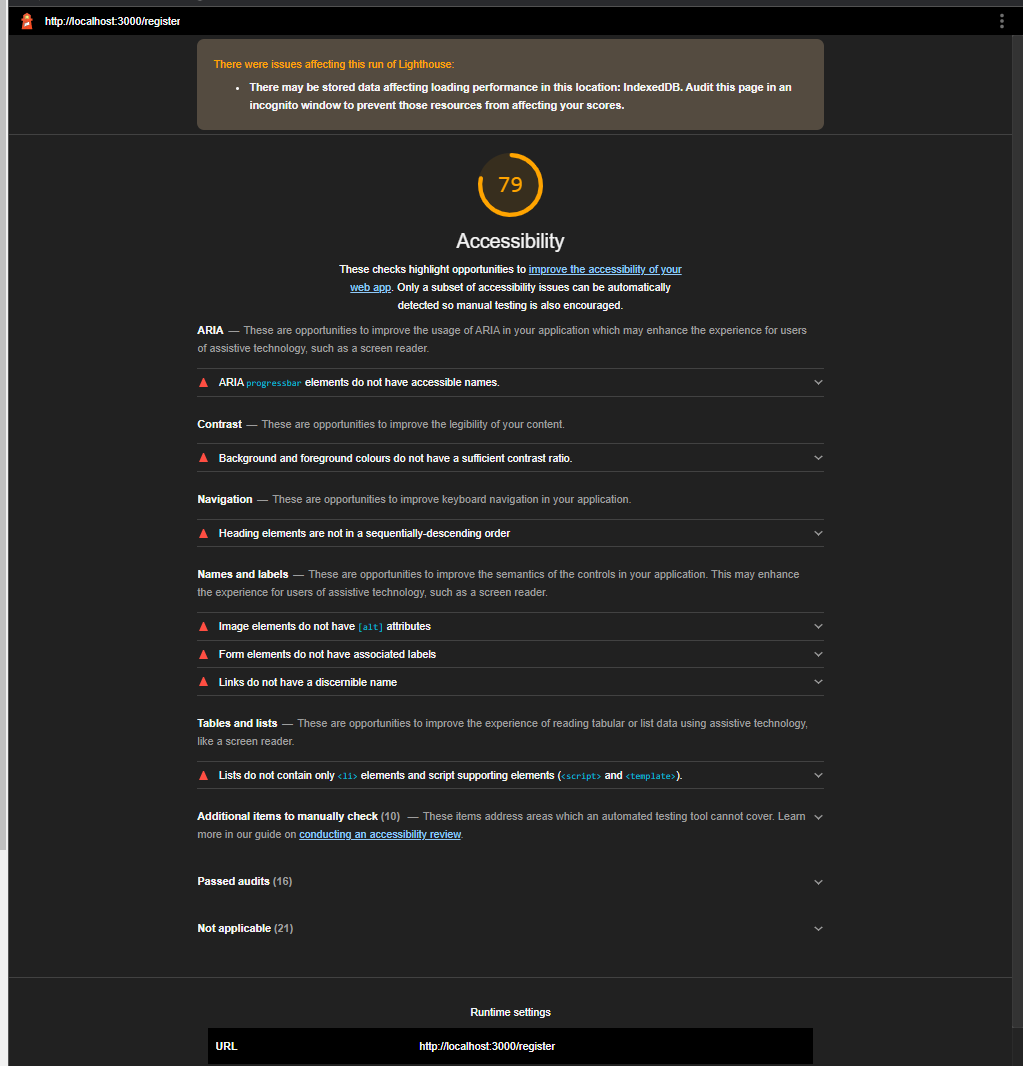
**Contact Accessibility Report**



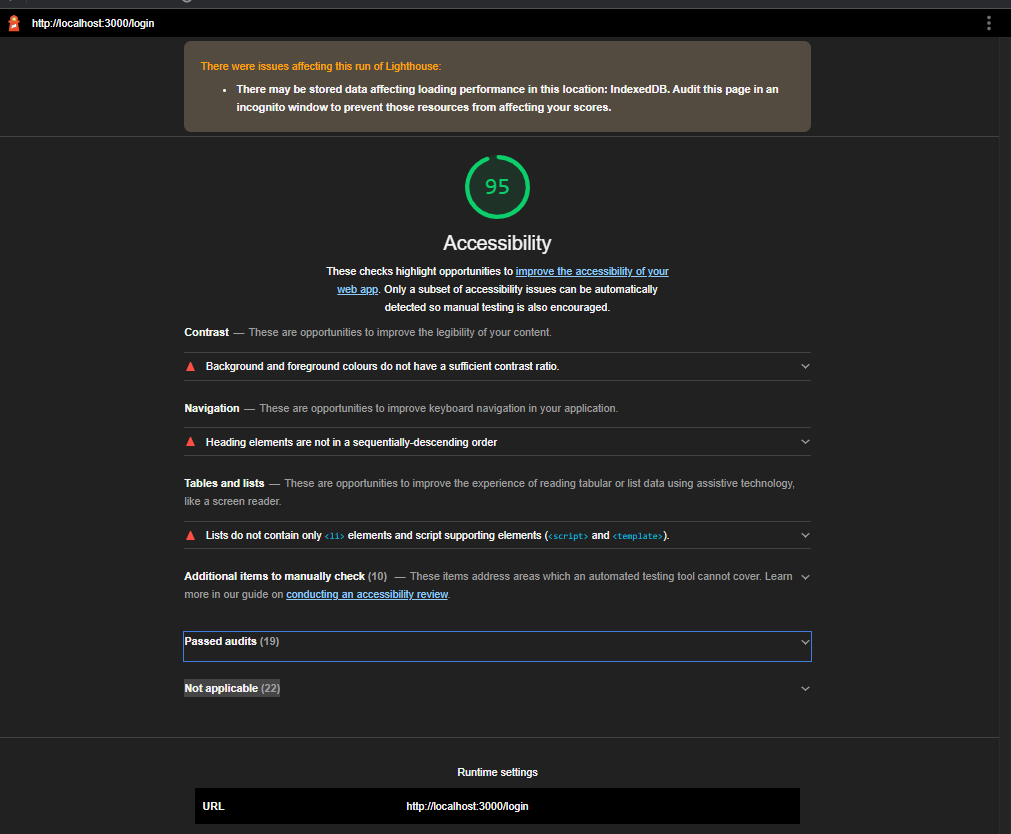
**Findus Accessibility Report**



**Register Accessibility Report**



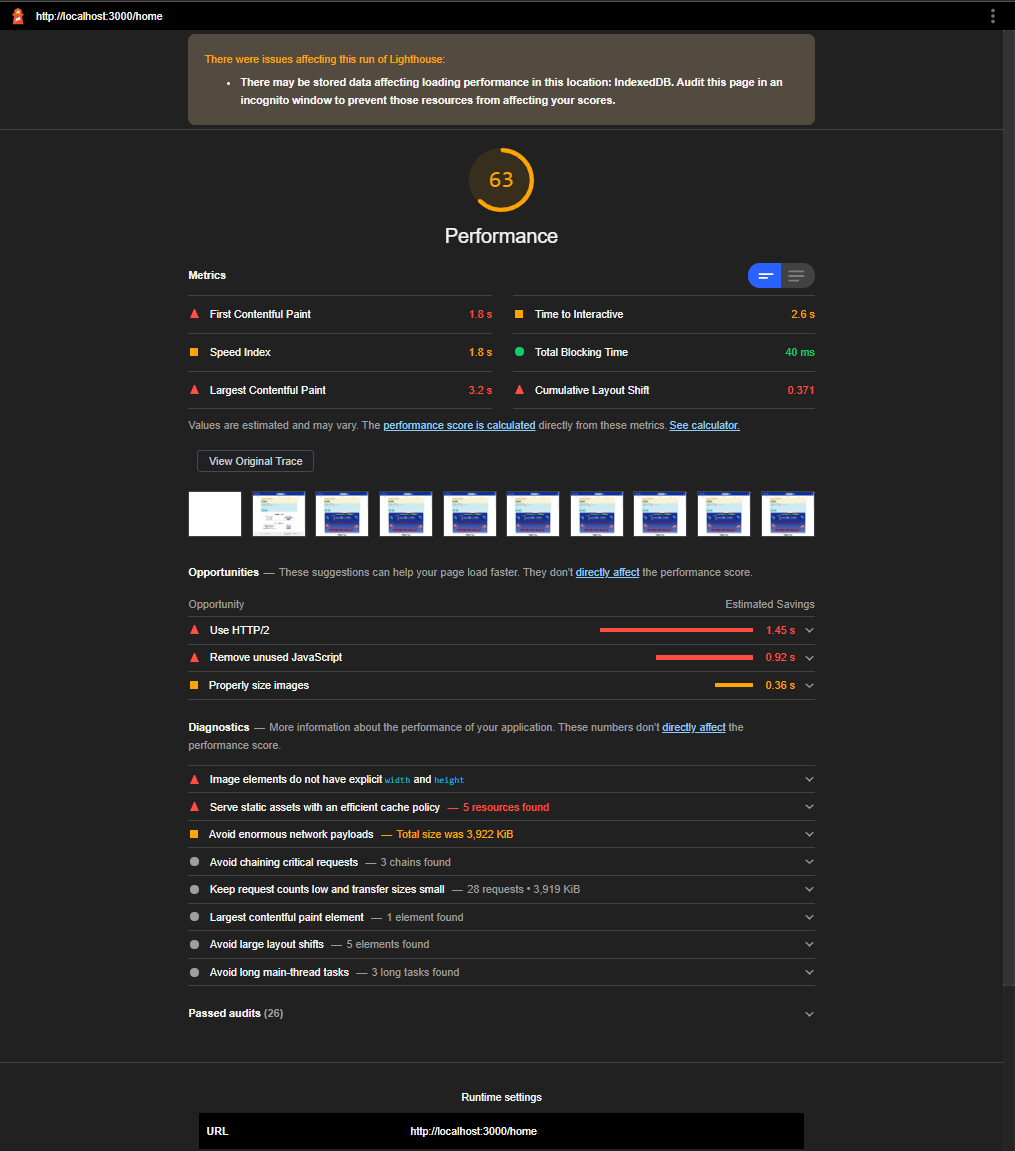
**Login Accessibility Report**



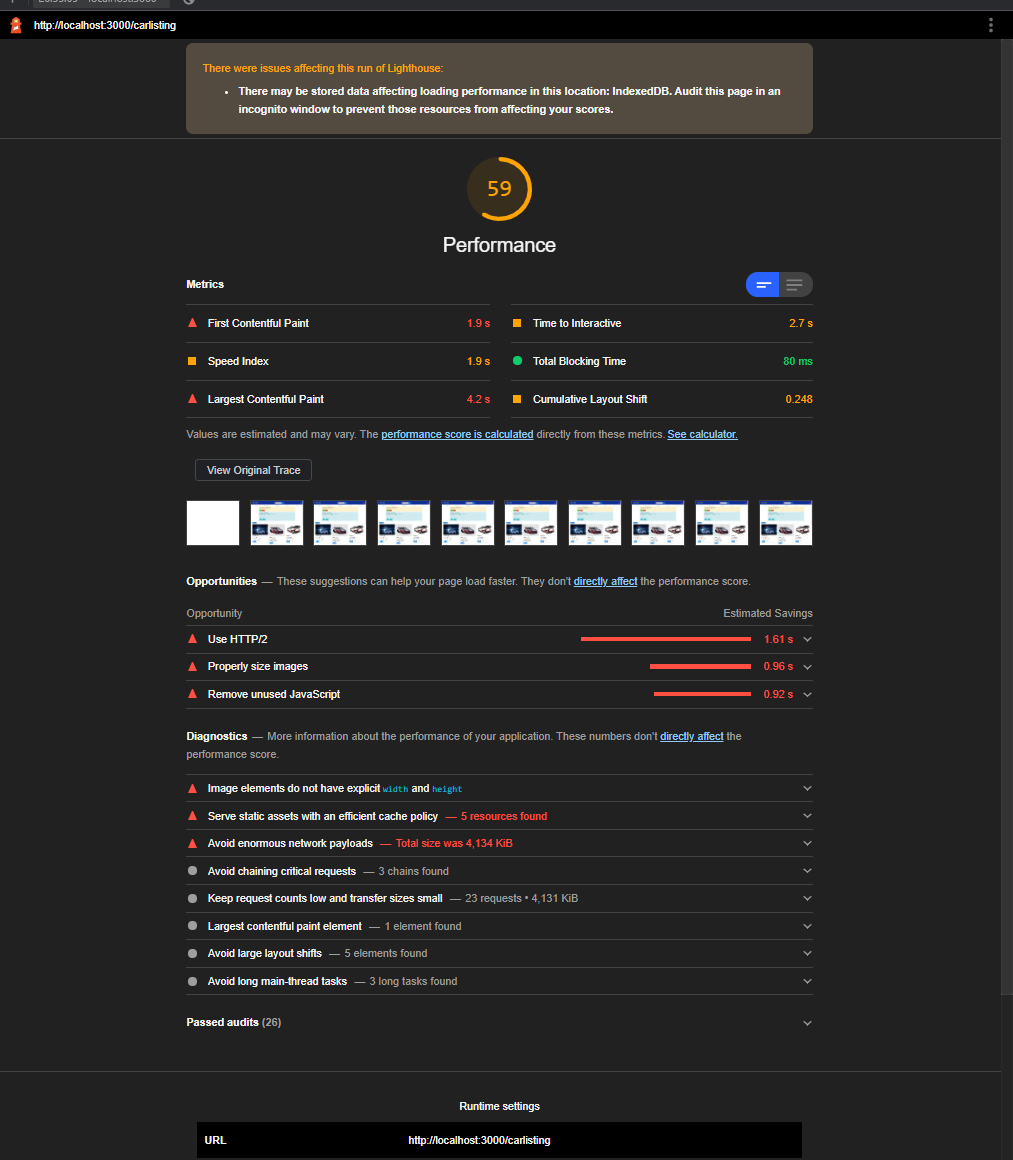
**Appendix 2 (Performance Reports from Lighthouse)**

We used lighthouse a google chrome tool and do not take credit for the reports findings.

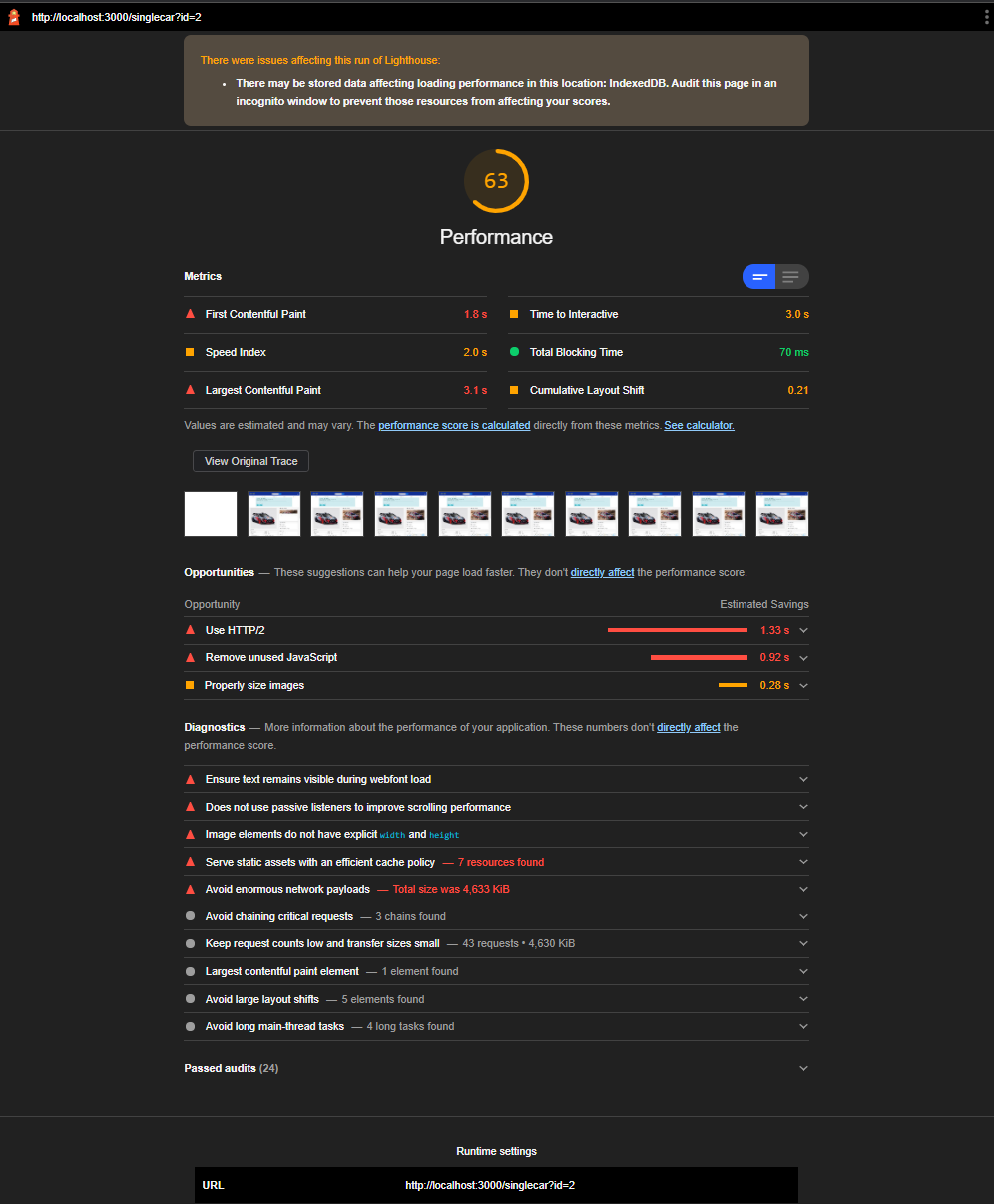
**Homepage Performance Report**



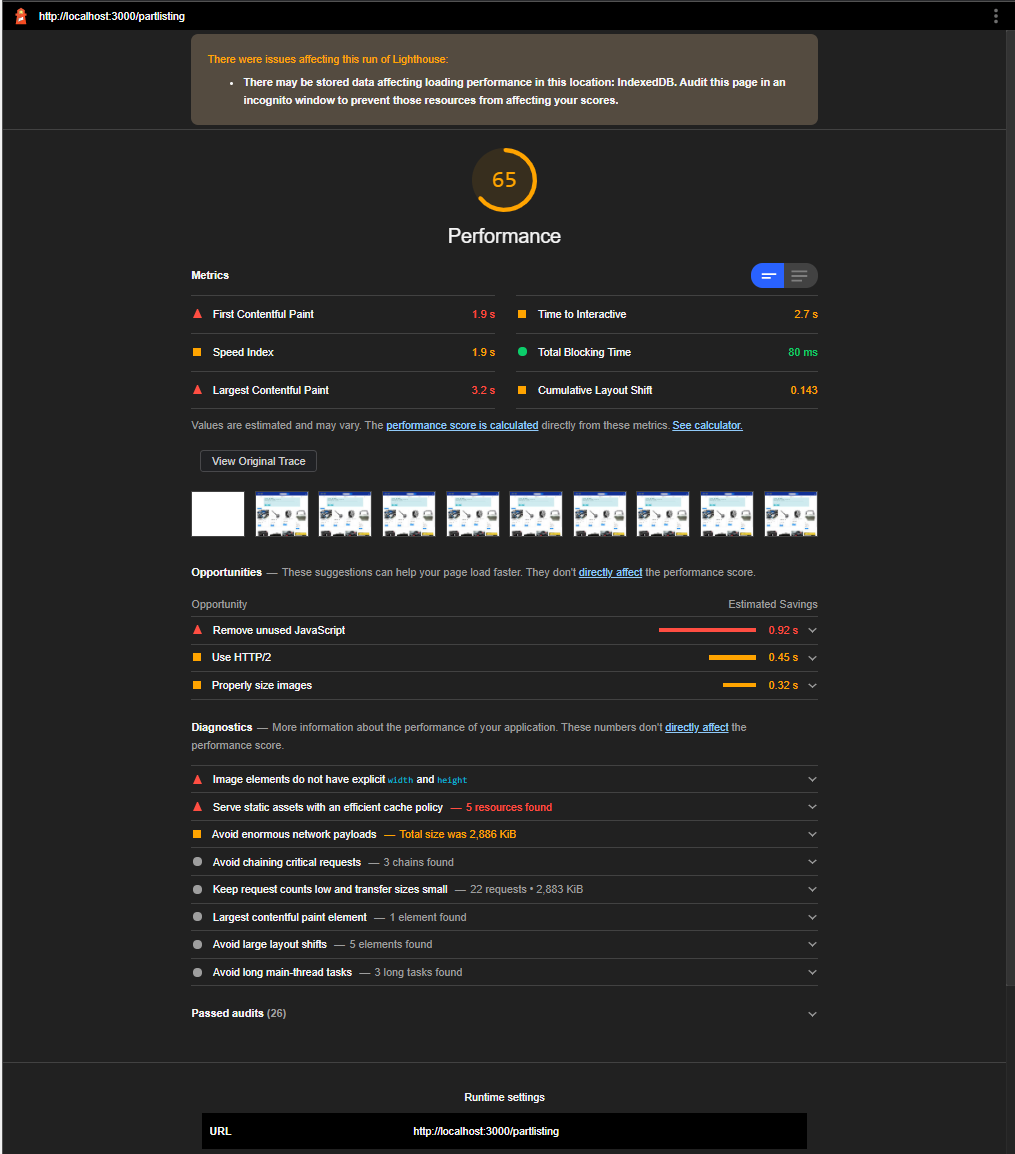
**Car Listing Performance Report**



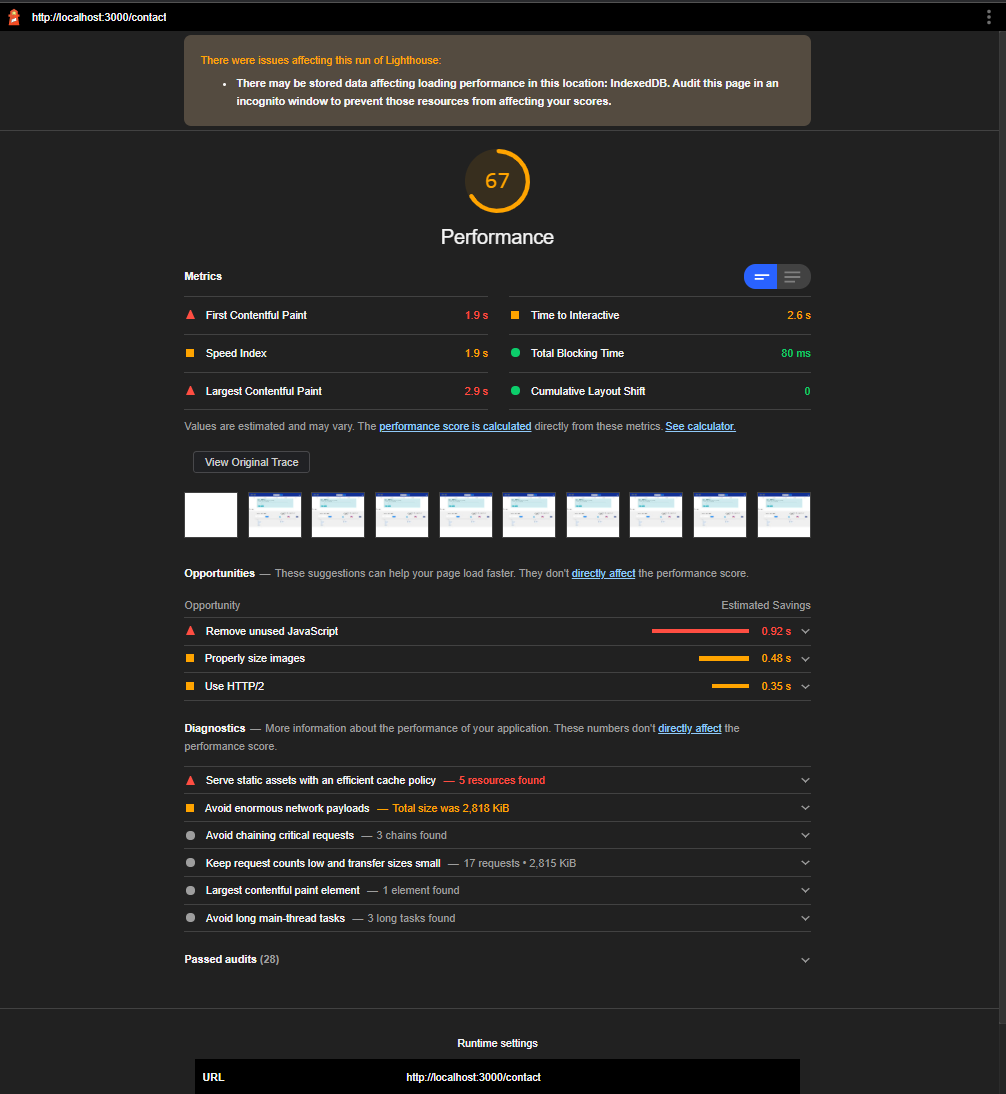
**Single Car Page Performance Report**



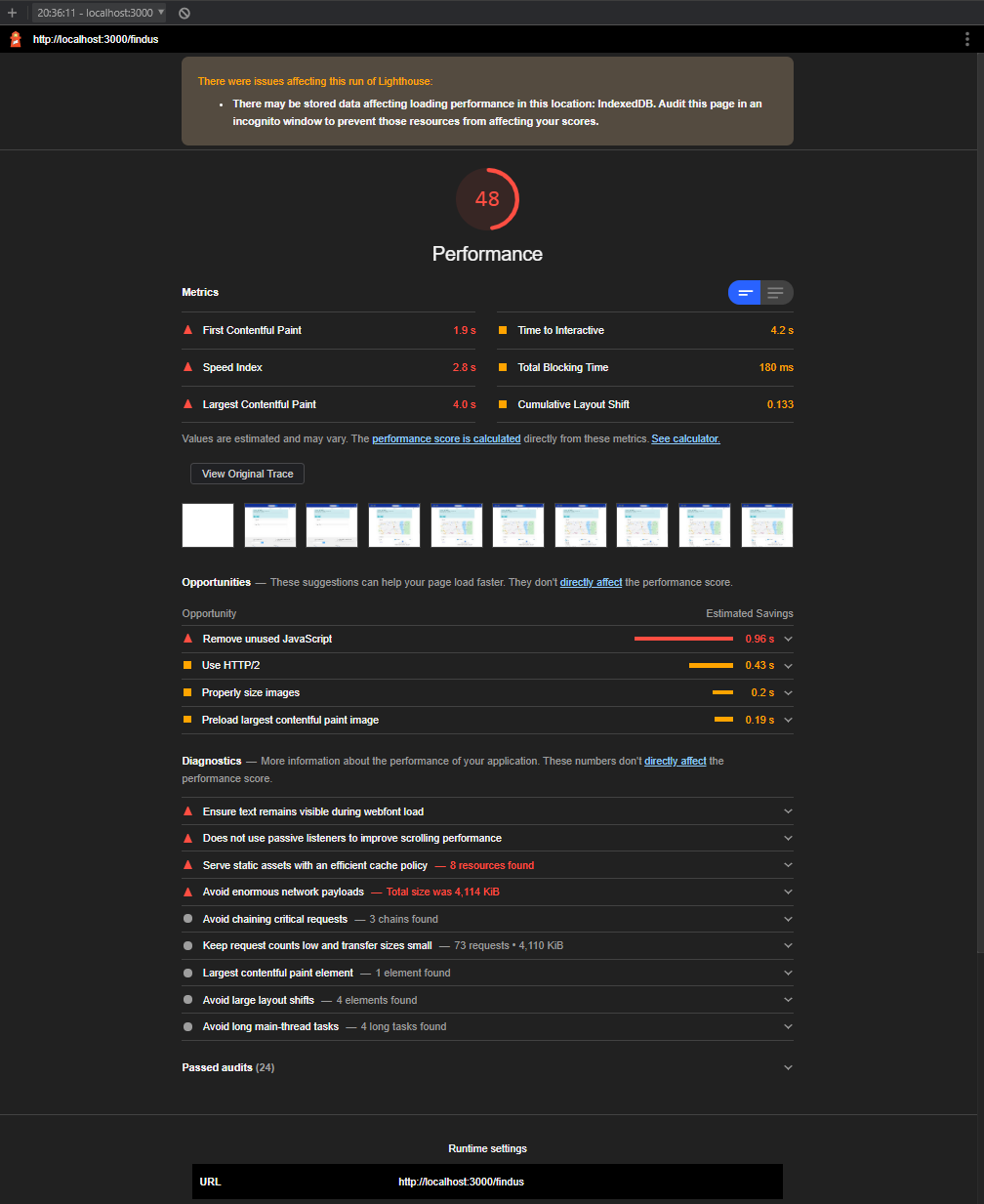
**Part Listing Performance Report**



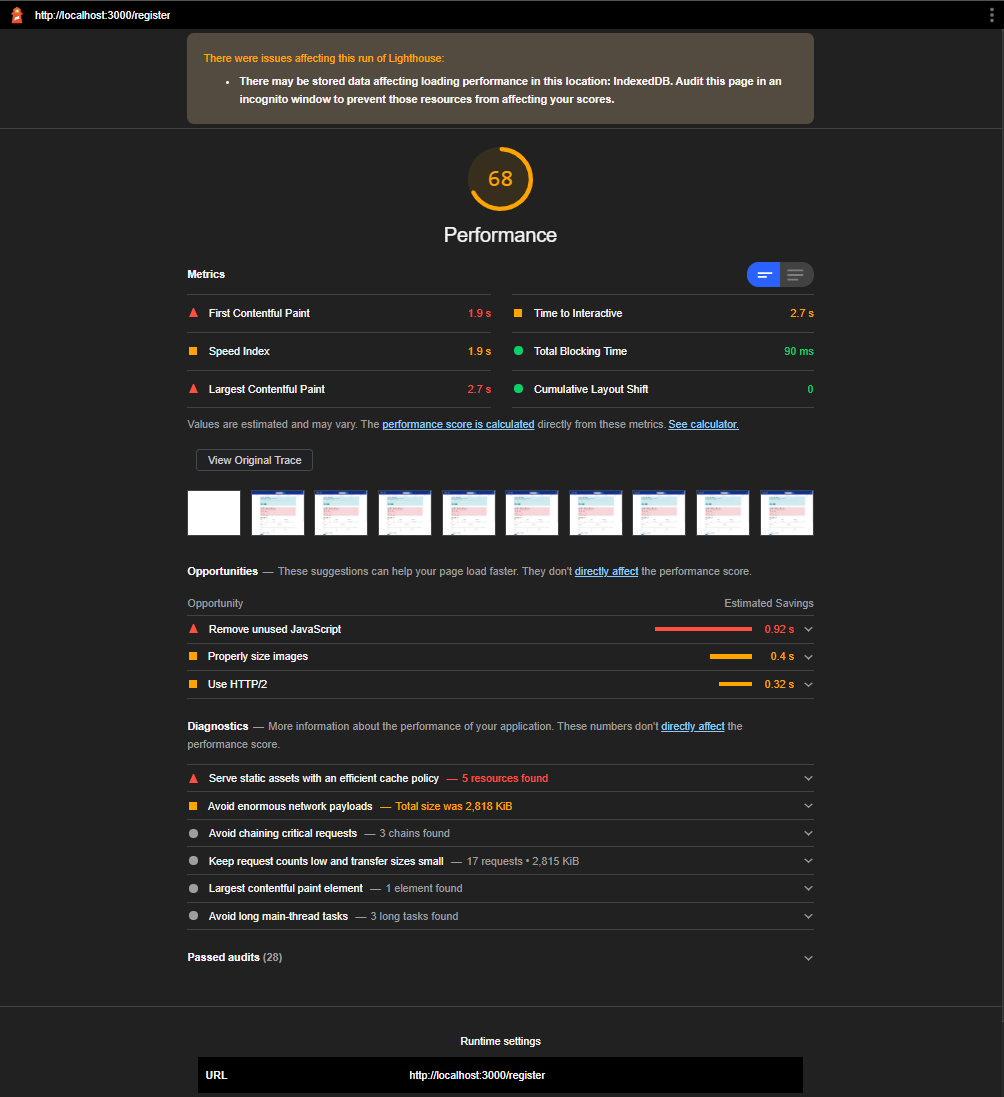
**Contact Performance Report**



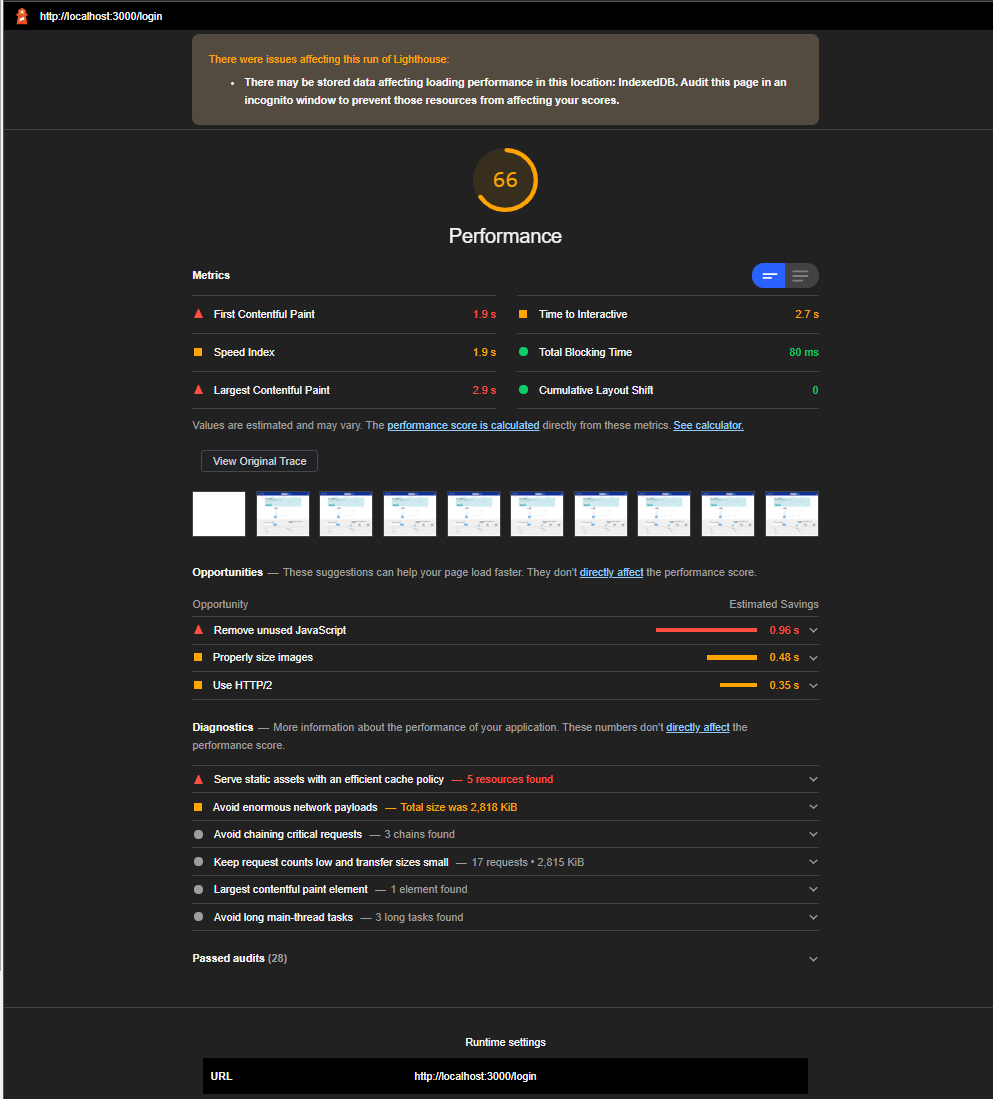
**Findus Performance Report**



**Register Performance Report**



**Login Performance Report**



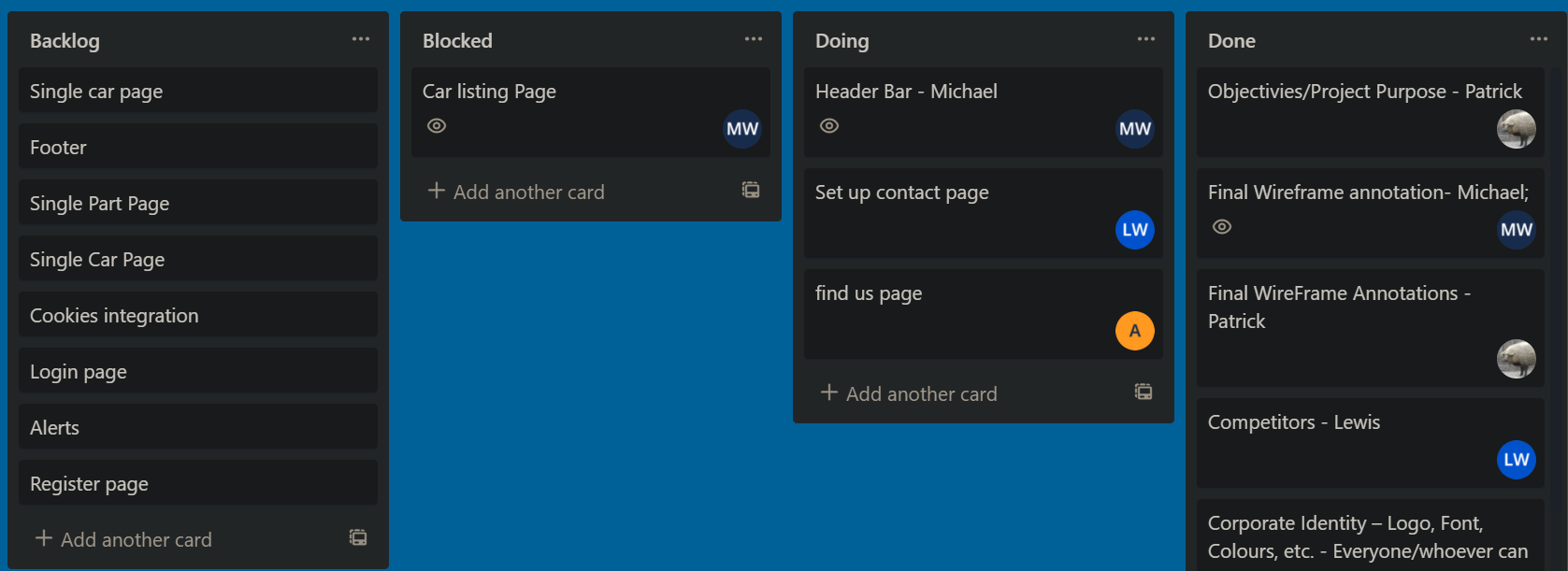
**Testing**

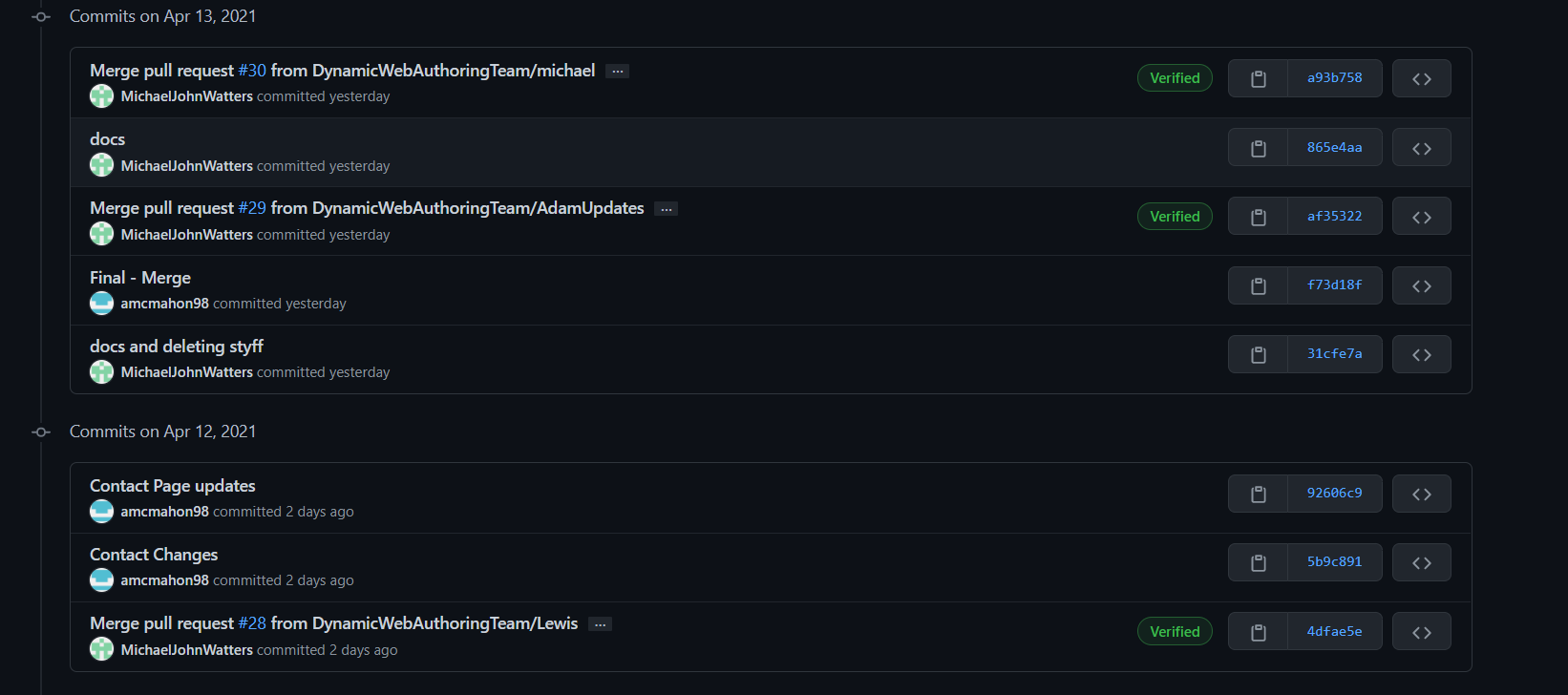
Since the testing document was so large, we decided to move it to a separate word doc, see Testing.docx for Testing component of the project.

**Evaluation**

We delivered on all the pages we original put into original design. As the requirement arose for additional pages, we fitted them into the website following the same style and theme.

I believe we improved on the original design by taking a more Agile Approach and the team used the Kanban system using Trello to monitor tasks and update status. This allowed us to add new stories or improvements as we went to improve the overall finish of the website, this workflow along with git where code would be worked on, in multiple branches and pull requests would be merged after review increased our productivity massively.





We managed to produce both a desktop website and a mobile compliant website that was fully responsive using a combination of customized bootstrap and react-bootstrap components throughout our website and had a very strong accessibility rating.

Our choice of technologies helped a lot, bootstrap was a great tool to work with and we do not take credit for any of the engineering within this library. Again, react was a great choice, the volume of resources the team had access to which allowed us to produce quality functional code with limited experience. And again, we do not take credit for any of the react libraries used or for react itself.

The virtual DOM as used in react allowed us to produce a lag free and very useable website will almost no stutters or lag while navigating.

And the use of the google development tool ‘lighthouse’ helped massively with adhering to standards and fixing other issues with accessibility and performance.

Throughout the process we discovered the value of web-development frameworks libraries and tools and as I result I would lean towards these tools or frameworks when looking towards the final year project.

References

1. <https://reactjs.org/>
2. <https://www.simform.com/wp-content/uploads/2020/02/stacktrend.png>
3. <https://react-bootstrap.netlify.app/>
4. <https://angular.io/guide/architecture>
5. <https://en.wikipedia.org/wiki/React_(JavaScript_library)>
6. <https://vuejs.org/v2/guide/>
7. <https://www.djangoproject.com/>
8. <https://www.toptal.com/front-end/what-is-bootstrap-a-short-tutorial-on-the-what-why-and-how>
9. <https://get.foundation/>
10. <https://bulma.io/>
11. <https://en.reset.org/blog/whats-carbon-footprint-your-website-01162020#:~:text=An%20average%20website%20produces%204.61,carbon%20footprint%20of%20their%20websites>.
12. <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript>
13. <https://www.python.org/doc/essays/blurb/>
14. <https://www.typescriptlang.org/>
15. <https://www.codecademy.com/courses/build-interactive-websites/lessons/javascript-dom/exercises/document>
16. https://programmingwithmosh.com/react/react-virtual-dom-explained/