A CARBON FOOTPRINT CALCULATOR TOOL FOR WEB PAGES

Burak Tokak
Software Engineering MSc Program, Department of Computer Engineering,
Middle East Technical University, Ankara, Turkey

Abstract: As tech makers, we have a responsibility to create products that are optimized for the resources they use in terms of their impact on the environment. Carbon emission from the electricity our servers and devices use during data transfer and processing of the data to browse the web pages must be considered and optimized. The calculator tool that I designed and implemented calculates the carbon footprint of a web page based on server location, and the number of new and returning page views, and offers a list of recommendations to optimize a specific web page using various metrics. I also explain the solutions to issues deploying applications with computation intensive backend, such as this calculator tool.

Keywords: software architecture, web performance, carbon footprint

Introduction

The internet has become a staple in our everyday lives. We use it for work, for play, and to connect with others. It has become an important part of how we live and operate in the world.

However, as our dependence on the internet has grown, so has our carbon footprint caused by it. The internet is powered by electricity, and that electricity often comes from sources that generate carbon emissions, such as coal-fired power plants.

In addition, the data that we transfer and process when we browse the web requires energy to do so. This data transfer and processing result in carbon emissions as well. This carbon footprint from our use of the internet browsing can be minimized with small changes.

The first step in reducing our digital carbon footprint in terms of internet browsing is to understand it. And that's where this carbon footprint calculator comes in. It helps tech makers assess the carbon footprint impact of a web page, taking into account the server location, the number and ratio of new and returning pageviews, and the type of content on the page such as rendered plaintext (files that need additional rendering; HTML, stylesheet, JavaScript, streamed content and various XML formats), images, video formats.

Based on this information, the calculator provides a list of recommendations to optimize the web page for energy use to take actions in order to decrease the carbon footprint impact of a web page.

Similar online tools that calculate the footprint of a web page exist [1], but most fail to offer actionable recommendations to optimize the carbon emission of a particular web page.

The source code for the complete project is open source and browseable via GitHub [2].

Calculation Methodology

I formulate the calculation in two accounts: First, the emission created by the amount of data transferred to the client device from servers (with penalty factors like server location and edge server usage). Second, the emission created by the client device web browsing.

Calculator additionally calculates the amount of trees needed to offset the calculated carbon amount. Depending on [3], the annual CO2 offsetting rate of a tree varies from 21.77 kg CO2/tree to 31.5 kg CO2/tree, I will assume one tree annual CO2 offsetting rate as 24 kg.

Data Transfer and Server Energy Consumption

When a user visits a web page, the server hosting the page uses energy to process the data associated with the request. This includes fetching the requested data from storage, processing it, and sending it back to the user's browser.

In order to calculate carbon emission per KB, I use the average amount of electricity it takes to transfer 1 KB data from server to client and US average electricity source emissions [4] per kWh. Carbon intensity, a measure of how clean the electricity is in terms of how many grams of CO2 are released per kWh produced, will be factored in later in the calculation. These figures keep changing, that's why I built a data crawler that syncs the latest data for the calculator to use; most recent data can be seen in [2].

Additionally, the amount of energy used will depend on the distance the data has to travel. For example, if the web server is located in the United States and the user is visiting the web page from Europe, the data will have to travel a greater distance than if the web server and the user were both located in the United States. Even though this calculator does not take into account the location of the user's device, it will penalize the web pages that are not served via a Content delivery network service by a factor of 0.2. CDNs are commonly used services that distribute data with networks of proxy edge servers spread around the globe. Most web hosting providers offer this service out of the box, standalone services exist, like Cloudflare.

Processor web server location country is another item that I take into account. Different countries generate electricity with varying degrees of carbon intensity. For example, coal power plants tend to have a higher carbon footprint than nuclear power plants. Depending on the [4] 2021 data on carbon intensity of electricity by countries, the calculator will penalize or reward the web page, based on United States carbon intensity, since I take the data transfer emission based on the United States.

The amount of energy used by the web server to process and transfer the data will depend on the number of page views. The type of a page view also matters. While a first time page view downloads all the necessary resources to render the webpage, returning page views would download lesser number of files since static files are cached in the case of using a modern browser (eg. Google Chrome, Firefox) considering the necessary caching policies are in place by the web server software. As an example, NGINX web server software's content caching policy can decide how long an image file should be cached by the browser [5].

Client Device Energy Consumption

In addition to the server-side energy use, the client-side energy use must also be considered. This is the energy used by the user's device to process and display the data that is downloaded from the server.

This calculator will assume all page views are done on a smartphone. On average, half of the devices use WiFi and the other half use cellular data. Depending on the energy consumption data from the research [6], a smartphone consumes 98.1 mW of energy per minute on average during web browsing. Since most of the energy consumption happens during the page load and idle browsing of a website does not consume as much energy, I assume the average time on page metric as one minute. Similarly I derive the amount of CO2 emission based on changing US average electricity source emissions per kWh [4].

The client-side energy use will depend on the type of content on the web page. For example, pages with more images will require more energy to process than pages with less images. In order to factor this in, I am using the Largest Contentful Paint (LCP) metric [7] introduced by Google Lighthouse, LCP the time passed until the web page is completely rendered, including image loading and on load JavaScript operations. As suggested by Lighthouse [7] I take the average LCP metric as 2.5 seconds and penalize or reward web pages over or less than average amount of time.

Recommendation Methodology

In order to optimize the carbon emission of the web page, the calculator offers actionable recommendations for tech makers. As a basis the recommendations made follow PageSpeed Insights [8] performance auditing guidelines and tools.

I picked performance audits that are relevant to the impact on carbon emission from their guidelines.

Table 1. Audits and metrics used for recommendations [8]

Audit / Metric	Carbon Impact
Total Blocking Time (TBT)	High Total Blocking Time means the client device CPU is extensively used during page load. Optimizing this value lowers the energy usage / CO2 emission of the client device.
Main-thread work	Delivering smaller JavaScript payloads at the appropriate time will decrease the useless JavaScript parsing and decrease energy consumption / CO2 emission.
First CPU Idle Time	Lowering the time and workload of the CPU will decrease energy consumption / CO2 emission for client devices.
Efficient static asset cache policy	Long caching policies for static files [5] decreases returning pageview transfer sizes and lowers carbon emission of servers transferring files to client devices.
JavaScript execution time	Delivering smaller JavaScript payloads at the appropriate time will decrease the useless JavaScript parsing and decrease energy consumption / CO2 emission.
Largest Contentful Paint (LCP)	High LCP might mean your web page is intensive in terms of images or styling. Decreasing this time metric lowers the energy usage / CO2 emission of client devices.
Speed Index (SI)	Decreasing SI lowers the energy usage / CO2 emission of client devices.
Cumulative Layout Shift (CLS)	When the CLS metric is high, the graphics processor of the client device uses additional energy, optimizing this index can lower energy usage / CO2 emission.
Initial server response time	High response time might mean either the server is far from the client and without a CDN service, or there are intensive backend tasks, optimizing this value will decrease carbon emission of servers transferring files to client devices.

Audit / Metric	Carbon Impact
Lazy image loading	Lazy loading images helps lower data transfer and lowers carbon emission of servers transferring files to client devices. Additional not rendering offscreen images helps decrease energy consumption by client devices.
Unminified CSS and JavaScript	Minifying CSS and JavaScript lowers data transfer and lowers carbon emission of servers transferring files to client devices.
Unused CSS and JavaScript	Tree-shaking unused CSS and JavaScript rules before deployment lowers data transfer and lowers carbon emission of servers transferring files to client devices.
Optimized images	Optimizing images lowers data transfer and lowers carbon emission of servers transferring files to client devices.
Text compression	Text compression lowers data transfer and lowers carbon emission of servers transferring files to client devices.
Responsive images	Using responsive images can lower data transfer for different device screen resolutions.
Legacy code serving	Not delivering legacy code for modern browsers will cause carbon emission of servers transferring files to client devices.
DOM size	Lower DOM size means lower data transfer and lower energy used / CO2 emitted during page rendering on client devices.
No document.write	Page load time is affected by this audit, optimizing this index can lower energy usage / CO2 emission.

I calculate and test the given web page against the metrics and audits in Table 1, and generate a report and show it on the results screen. In this screen, the metrics are ranked as “Poor”, “Needs work” or “Perfect” in terms of the guideline scoring [8]. Metrics and audits ranked as “Poor” and “Need work” offer links and guides in order to improve them. Additionally all metrics and their carbon impact explanations are displayed on the results page of the calculator.

Software Architecture

This section will show; use case view, process view and physical view of the Carbon footprint calculator project's software architecture.

Use Case View

Here is a use case diagram that shows the scenario of a user's interactions with the calculator tool application.

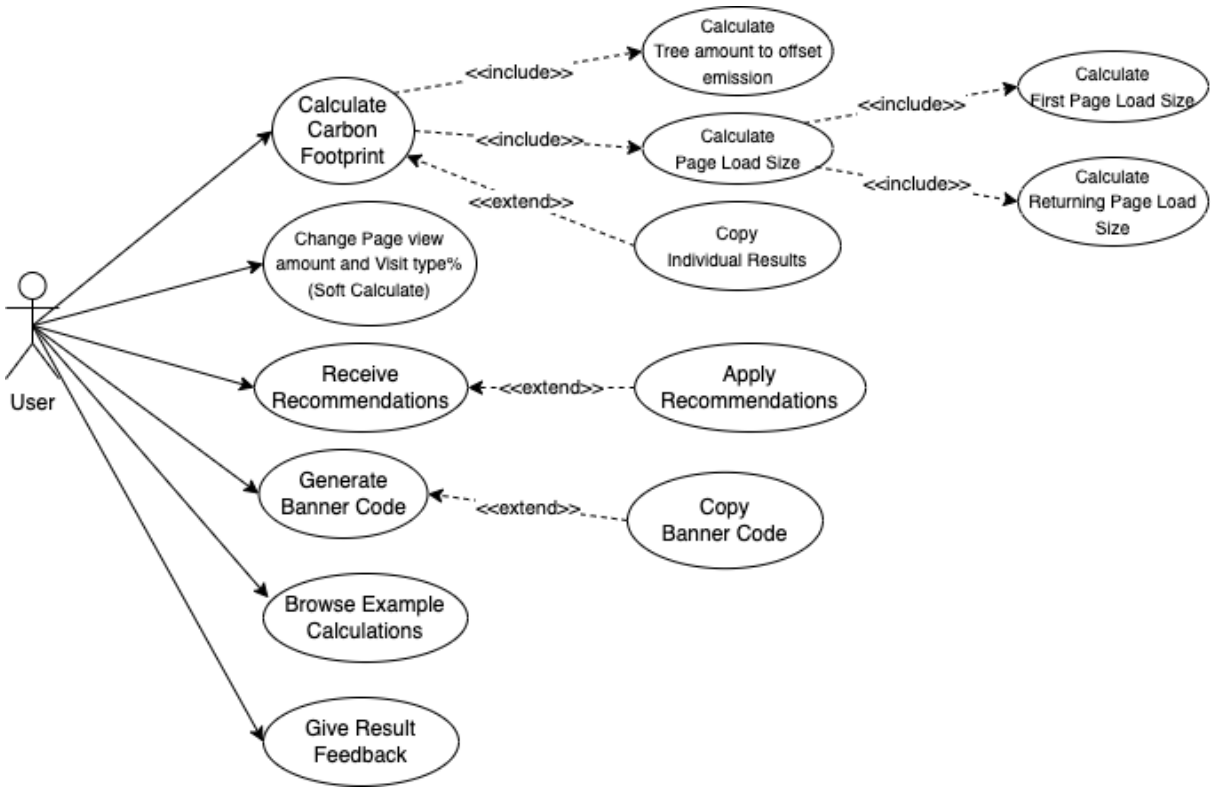

Figure 1. Use Case Diagram

As seen in Figure 1, I intended the calculator tool to be a part of the user's constant workflow. The user is able to calculate their intended web page, receive recommendations, apply them to their staging web page then retest it in a circular way to see the changes in their web page's carbon footprint. This is intended to be an ongoing process, to help the user gradually improve their web page's carbon footprint.

Again as the Use Case Diagram suggests, users are able to both calculate the secondary metrics like page size, LCP, amount of trees to offset at the same time while they calculate the carbon emission amount of the particular web page.

Process View

Here is a sequence diagram of how the calculation process happens via Function Instances, and how these instances interact with and depend on each other.

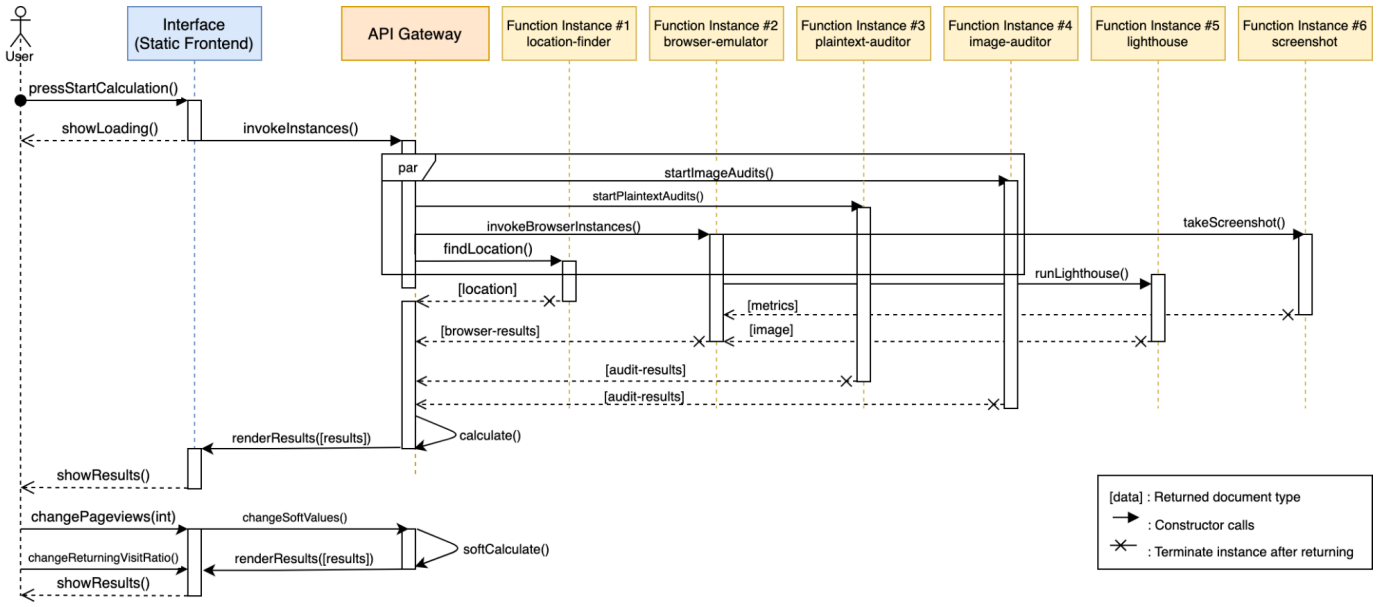

Figure 2. Sequence Diagram

As seen in Figure 2, containerized processes of calculation happen in parallel to each other. The main reason for this is to increase scalability and to enable the system to handle a large amount of requests in an acceptable time frame (on average 15 seconds).

Additionally the function instances shown in the Figure 2 gets constructed on invocation from API Gateway, then gets destroyed after the instance returns data. The API gateway provides the necessary data (inputs) to the function instance, which waits for the returned data from asynchronous instances then processes it and responds with the calculation result.

Users also can perform a "soft calculation" which refers to being able to change the monthly pageview amount and visit type ratio. Changes on these values won't trigger a new calculation process, instead API Gateway uses the existing returned data from the instances to perform a fast calculation. This allows the user to see the effects of the changes on the pageview and visit type ratio to the calculation results in real-time.

Physical View and Deployment

The calculator downloads a web page and performs resource intensive audits on the data with a headless browser emulator in order to render the outputs of a web page on the backend. Performing all the tasks in a user acceptable time frame was an issue during implementation of this project. Containerizing the calculation and auditing tasks parallel was the solution to this issue. On average; the calculation process and processing recommendations, takes 20 seconds. Although a large web page can take up to a minute.

Since the calculator tool is doing a lot of computation at a short period of time in parallel, a potential issue in terms of deployment is the fact that the calculations use considerable server CPU and RAM at the same time. Using a physical server with limited resources would result in a parallel user limit. This means, for example, while 20 different users are using this tool to calculate at a time, the 21st user needs to wait on hold for available server estate. To solve this issue, I choose to deploy self-contained parts of the calculation as Function Instances (high-availability compute infrastructure that usually handles a specific backend task and can scale horizontally) on AWS Lambda [9].

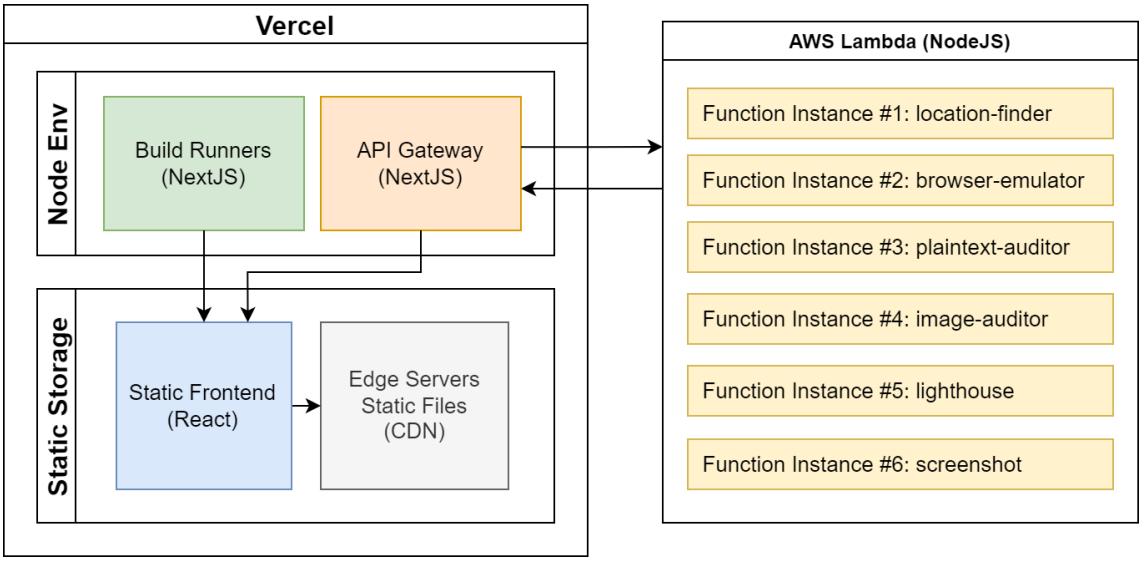

Figure 3. Deployment diagram for the calculator (arrows indicate information flow)

As seen in Figure 3, 6 different functions of the calculation are deployed as Function Instances and able to run parallel to each other upon invocation. During a calculation; API gateway layer, orchestrated by NextJS, invokes and waits for outputs from all the function instances in a parallel way, then parses and sends data to the frontend in JSON format. Specific tasks of these function instances are detailed in Table 2.

Table 2. Function Instances and their tasks

Function Instance	Function definition and task
location-finder	Determines the server country, determines CDN usage and calculates carbon intensity multiplier.
browser-emulator	Runs a headless browser emulator in order to calculate the time related metrics such as LCP, JavaScript execution time and SI (see Table 1).
plaintext-auditor	Audits and generates plaintext related recommendations such as Text Compression and DOM size.
image-auditor	Audits and generates recommendations for images, such as “Optimized Images” and “Responsive Images” (see Table 1).
lighthouse	Runs additional “PageSpeed Insights” audits listed in Table 1.
screenshot	Takes a screenshot of the Largest Contentful Paint (LCP) with a headless browser emulator.

Technologies and Tools

Backend part of this application is heavily network and browser operations intensive. Taking this into account, I used NodeJS, which is a backend environment with proven networking capabilities. Additionally NodeJS runs on V8 JavaScript Engine [10], which comes handy with running headless browsers, especially Chromium [11].

For the frontend and backend API gateway I used NextJS, which combines ReactJS and NodeJS. As suggested by Figure 1, NextJS offers a great development and DevOps experience especially in terms of automating the deployment using Vercel's NextJS specific platform.

For the visual component part of the application I used NextUI, which is a modern ReactJS-based UI library. This library allows to quickly and easily build/modify the interface of the application, and also provides a lot of customizability like colors of the components and their functions.

For managing the project, I used Trello, a popular project management tool. Trello allows me to easily keep track of tasks, assign dependencies of different tasks and create a roadmap and timeline, eventually track the progress of the project.

Finally, I used Postman for API testing, which helps to quickly test API endpoints and their responses.

Future Work

Since the main goal of this calculator is to calculate the amount of carbon emission and amount of trees to offset the yearly emission, a natural feature would be to allow users to plant trees. In the future I plan to add an option for users to plant trees with a monetary donation. This could be done with an external link to an organization like TEMA [12].

Conclusion

With actionable recommendations and detailed calculation stages, the tool I built fills the feature gaps for currently existing solutions. Contrary to current solutions, the calculator offers updated calculations and guidance along the way for web page maintainers during the process of lowering the carbon emission of the webpage.

Web application for the calculator is currently live and operational for open beta testing [13]. During this time I ask users in the results screen for their evaluation of the results with a link to the GitHub issues page of the project for creating an issue about their evaluations and concerns about the specific results.

References

[1] Website Carbon Calculator, https://www.websitecarbon.com/, Last accessed November 2022

[2] Carbon Neutral Website Source Code, https://github.com/btk/carbonneutralwebsite, Last accessed November 2022

[3] Calculation of CO2 offsetting by trees, http://encon.eu/en/calculation-co2-offsetting-trees, Last accessed November 2022

[4] BP Statistical Review of World Energy 2022, 71st edition, pp. 12-14

[5] NGINX Content Caching, https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/, Last accessed November 2022

[6] Carroll, Aaron & Heiser, Gernot. (2010). An Analysis of Power Consumption in a Smartphone. Proc. 2010 USENIX conf., USENIX Assoc.. , p. 9

[7] Largest Contentful Paint, 2020, https://web.dev/lighthouse-largest-contentful-paint/, Last accessed November 2022

[8] About PageSpeed Insights, Google Developer Documents, https://developers.google.com/speed/docs/insights/v5/about?hl=en, Last accessed November 2022

[9] What is AWS Lambda, AWS, https://docs.aws.amazon.com/lambda/latest/dg/welcome.html, Last accessed November 2022

[10] What is V8, https://v8.dev/, Last accessed November 2022

[11] Headless Chromium, https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md, Last accessed November 2022

[12] Türkiye Erozyonla Mücadele, Ağaçlandırma ve Doğal Varlıkları Koruma Vakfı, https://www.tema.org.tr/, Last accessed Nov 2022

[13] Carbon Neutral Website, https://carbonneutralwebsite.org , Last accessed Nov 2022,

A previous version of this paper was submitted to UYMS'22 but rejected