KWAME NKRUMAH UNIVERSITY OF SCIENCE & TECHNOLOGY

COLLEGE OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

MINI PROJECT

PROJECT DOCUMENTATION

ARTICLE SUMMARIZATION GPT-4 MODEL

ADU STEPHEN

4187320

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I extend my heartfelt appreciation to everyone who played a role in the realization of this university level 300 mini-project. My sincere gratitude goes to our esteemed professor for their unwavering guidance, expertise, and continuous encouragement. I am also thankful to my fellow students for their collaboration and shared dedication to excellence. Moreover, I want to acknowledge the university's resources and facilities that enabled us to conduct research and execute this project effectively. This endeavor has enriched my academic experience and provided a platform to cultivate critical skills. Thank you all for contributing to this significant achievement.

PROJECT DOCUMENTATION.
Chapter One -Introduction

Project Topic

AI-Powered Article Summarization for Efficient Information Absorption (GPT-4 MODEL).

In an age shaped by the proliferation of AI, this project seeks to harness its capabilities to simplify the complexities of modern life. The ubiquitous challenge of navigating lengthy articles under time constraints prompts the need for streamlined content absorption. By harnessing the power of AI, specifically the GPT-4 model, this project introduces an innovative solution: a web application that swiftly generates concise article summaries from provided URLs. Beyond mitigating information overload, this novel approach democratizes access to knowledge. By delivering tailored, accessible summaries, the project exemplifies AI's potential to convert intricate content into valuable insights. This endeavour caters to a diverse audience, aligning seamlessly with the demands of today's digital interactions.

Problem Statement

In today's information-rich landscape, lengthy articles and content-heavy web pages pose a significant challenge for individuals seeking efficient knowledge absorption. People often forgo valuable insights due to time constraints and the overwhelming nature of extensive content. This project addresses this issue by leveraging the capabilities of AI, specifically the GPT-4 model, to provide accessible and succinct article summaries. The lack of tools that cater to this need leads to information overload, hampering effective learning, research, and decision-making. The absence of an intuitive summarization solution restricts users from extracting essential information promptly. This project aims to bridge the gap by developing a user-friendly web application that transforms lengthy articles into concise summaries, enhancing information consumption and enabling users to quickly access key insights.

Project Justification

In an era dominated by the surge of AI, the project's rationale lies in leveraging this technology to simplify modern life. Lengthy articles often discourage readers due to time constraints, prompting the need for efficient content absorption. By employing AI, specifically GPT-4, this project proposes a novel approach – a web application that promptly generates article summaries using provided URLs. This innovative solution not only addresses the challenge of information overload but also democratizes access to knowledge. By offering tailored and accessible summaries, the project underscores the potential of AI to transform intricate content into valuable insights, catering to a diverse audience and aligning with the demands of contemporary digital interactions.

Aim of the project

The primary aim of this project is to develop a user-friendly web application that harnesses the power of AI, specifically the GPT-4 model, to provide succinct article summaries. By inputting a Uniform Resource Locator (URL), users can access comprehensive article synopses. The project strives to streamline information consumption, addressing the common challenge of time-consuming content engagement. It seeks to bridge the gap between advanced AI capabilities and everyday needs, offering a tool that transforms intricate articles into easily digestible summaries. Through this endeavour, the project aims to empower users across diverse fields, promoting efficient learning and information retrieval.

Motivation for the Project

The project's motivation stems from harnessing the potential of dynamic AI applications to address real-world challenges. Fuelled by a desire to simplify information consumption, the idea was born from personal experience grappling with lengthy articles. Recognizing the growing influence of AI, the project aims to make this technology accessible to all, enhancing comprehension and saving time. By offering a tool that transforms complex articles into concise summaries, the motivation lies in bridging the gap between advanced AI capabilities and everyday needs. This endeavour not only promotes efficient learning but also cultivates essential skills in AI implementation and web development, aligning with the demands of our ever-evolving digital landscape.

Beneficiaries of the project

The beneficiaries of this project encompass a broad spectrum, ranging from individuals to students and researchers. Everyday readers stand to gain by swiftly grasping key insights from articles without sifting through extensive content. Students benefit from expedited research, bolstering their learning endeavours. Furthermore, professionals seeking quick information access can optimize their efficiency. Aspiring developers are provided an educational gateway, honing web development skills, API integration expertise, and AI understanding. The project's user-friendly interface and time-saving capabilities democratize knowledge, making it an indispensable resource for anyone navigating the information-rich landscape.

Project Limitations

This project's limitations encompass its exclusive focus on article summarization using provided URLs. It lacks the capability to extract content from non-article websites and is designed solely for summarization rather than content extraction. The reliance on URLs restricts its functionality to web-based sources. Additionally, the project is not tailored for platforms like YouTube, which have different content formats. While it offers valuable benefits for articles, it might not be suitable for diverse content types. These limitations shape the project's scope and emphasize its purpose as an article summarization tool.

Academic and Relevance of the project

If a student wants to improve his web development skills, learn how popular frameworks like React works, fetching APIs from different endpoint to include them into the project the individual is building, gaining a bit of knowledge in how AI works as well and gaining the necessary skills to become a good developer, this project can be the gateway. While building, you learn a lot academically and prepares you for the industry the individual will find him or herself.

SCOPE OF THE PROJECT

The scope of this project involves creating a web application powered by the GPT-4 model to provide concise summaries of articles using provided URLs. The application will utilize technologies such as Node.js, React, and Rapid API to fetch and process article content. Users will input URLs into the system, triggering the summarization process However, the project's limitations include focusing solely on articles, not extracting content, and relying on URLs. The tool's potential impact is substantial, saving users time while navigating through long texts and fostering efficient information consumption.

Project Tools

-**Visual studio Code** is a popular coding space provided with necessary tools and extensions to make programming possible. This is a tool that is better for the environment to code out my project.

-**Node.js** is a huge, required dependency that serve as a gateway for coding the popular programming language JavaScript on the Visual Studio Code and code out the logic aspect at the backend in connection to the frontend. This tool is really needed since JavaScript was mainly used to code on the browsers.

-**React** for web frameworks to avoid repetition of codes and build good functional components
which can be implemented anywhere.

-Rapid API for accessing the API of GPT-4 model and incorporating it into the project to summarize articles.

-**Vite** is a popular framework or library under React that helps set the structure and the needed file files to quick start the project at hand.

-**Node Package Manager (NPM)** is a package that contain snippet of codes designed by other developers to deal with a specific function. By using the command line "npm install 'package-name', a code snippet can be downloaded through the terminal provided on the visual studio code."

-React-Redux is a toolkit included in the backend of this project. A function of it is to create a store that will manage the state of the project. This tool is helpful in the project to manage the state of the of the Application Programming Interface when we call for the Get Summary Request provided by Rapid API.

-React-Icons is a library full of icons that can be incorporated into react projects such as buttons. This helps to make good designs and introduce a user-friendly site as well.

Project Deliverables

The project aims to deliver a fully functional web application that offers AI-powered article summarization using the GPT-4 model. The deliverables include an intuitive interface designed with Tailwind CSS for responsive and efficient user experience. The backend will be built using Node.js and React, ensuring smooth communication between the frontend and the Rapid API. The final product will demonstrate the successful integration of AI technology, enabling users to obtain quick and accurate article synopses. Additionally, the project will provide documentation detailing installation, usage instructions, and an overview of the technologies used, facilitating seamless adoption by users and developers interested in similar applications.

Project Design

The project design revolves around creating a seamless user experience through a well-structured web application, fuelled by AI-powered article summarization. The application's design considers both frontend and backend aspects, ensuring efficient data flow and optimal functionality.

UI Design: The chosen design framework, Tailwind CSS, facilitates rapid and responsive UI development. Tailwind's concise CSS properties enable swift and effective design, enhancing user interaction. This design choice aligns with the project's goal of simplicity and user-friendliness.

tailwindcss

Frontend Development : The React framework is adopted to build functional components, fostering code reusability and streamlining development. This choice allows for efficient handling of user interactions and the integration of React-Icons, enhancing the interface's visual appeal.

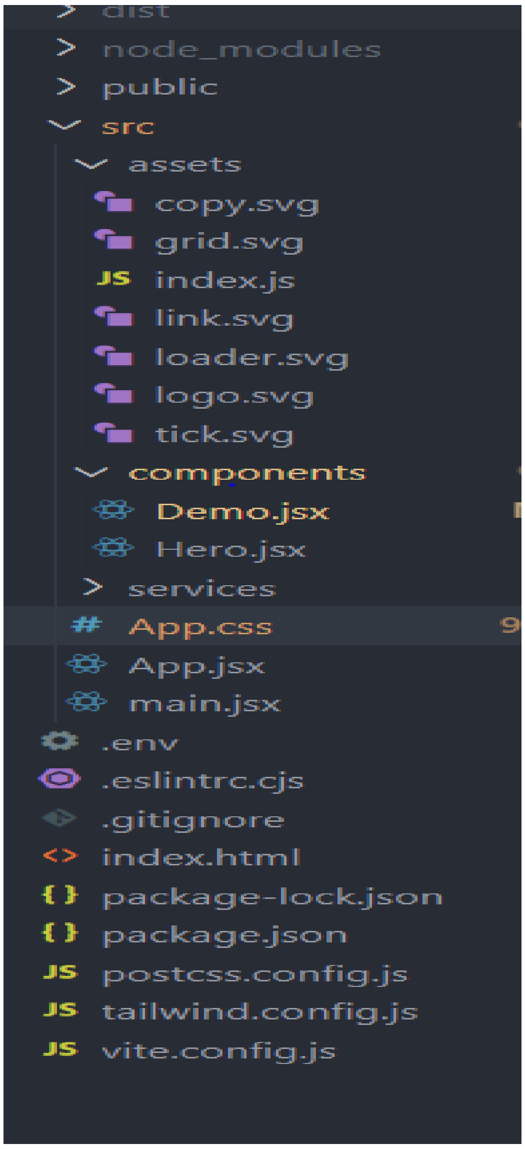
Backend Development: The project leverages Node.js to establish a robust backend infrastructure. This facilitates seamless communication between the frontend and Rapid API endpoints, ensuring timely article retrieval and summary generation.

API Integration: Rapid API's integration into the backend enables the application to access the GPT-4 model for article summarization. React-Redux is employed to manage API states and facilitate smooth data exchange between components.

The project's design orchestrates an efficient, user-friendly web application that harnesses AI to deliver succinct article summaries. The frontend's intuitive interface, backbend's effective data management, and API integration showcase a holistic approach to solving the problem of information overload, catering to diverse user needs in a dynamic digital landscape.

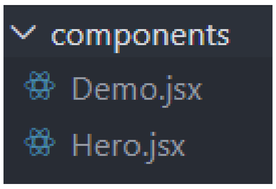
<u>Chapter Two – The Proposed System</u>

<u>Architecture of the Proposed System</u>



```
> dist
> node_modules
> public
v src
  v assets
    copy.svg
    grid.svg
    index.js
    link.svg
    loader.svg
    logo.svg
    tick.svg
  v components
    Demo.jsx
    Hero.jsx
  > services
  App.css          9
  App.jsx
  main.jsx
.env
.eslintrc.cjs
.gitignore
index.html
package-lock.json
package.json
postcss.config.js
tailwind.config.js
vite.config.js
```

Components Folder



```
v components
  Demo.jsx
  Hero.jsx
```

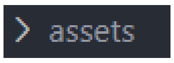
The "components" folder plays a crucial role in maintaining modular and reusable UI elements. It houses files like Demo.jsx and Hero.jsx, defining fundamental components for the project's frontend framework. These components contribute to the web application's structure and functionality, enhancing user interaction and navigation.

Public Folder

```
> public
```

The "public" folder contains resources accessible to all visitors of the web page, such as images, icons, and fonts. These assets are readily available to enhance the visual appeal and user experience of the application. By housing these resources in the public folder, the project ensures easy accessibility and efficient loading for users.

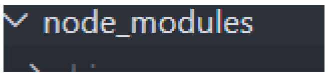
Assets Folder



```
> assets
```

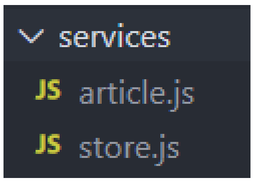
The "assets" folder stores image files, including SVGs, that are integrated into the project. These images contribute to the visual representation of the application and are utilized within components. In the Demo.js file, these assets are utilized to create engaging user interfaces, enriching the overall appearance and interaction of the application.

Node Modules

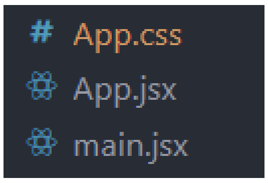

node_modules

The "node modules" folder comprises files that are installed via Node.js and npm. Node modules include dependencies required for the project's functionality, containing libraries and packages that aid in tasks like API integration, UI design, and data management. These modules streamline development, enabling developers to leverage existing tools and functionalities to achieve project goals efficiently.

Service folder


services
article.js
store.js

The "services" folder within the project's directory encapsulates critical components that contribute to the overall functionality and user experience of the web application. This folder follows a structured approach, ensuring a well-organized and modular development process.


App.css
App.jsx
main.jsx

App.css:

The "app.css" file within the "services" folder is dedicated to cascading style sheets, playing a pivotal role in defining the visual appearance and layout of the entire application. CSS is the cornerstone of web design, influencing everything from typography and colour schemes to spacing and responsiveness. Through meticulous styling defined in "app.css," developers can achieve a consistent and visually pleasing interface. This file facilitates the separation of design concerns from other code components, enhancing maintainability and promoting a clean, structured codebase.

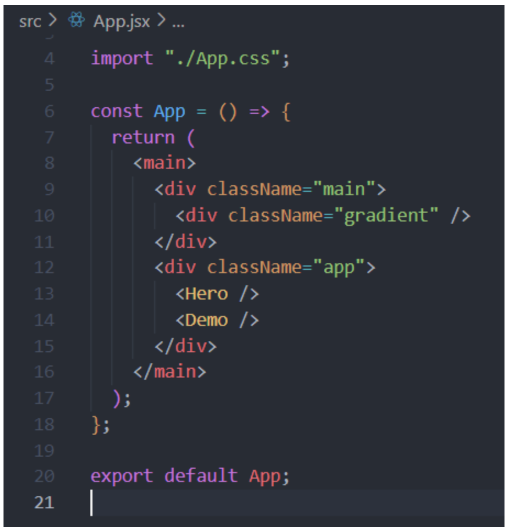
App.jsx:

The "app.jsx" file within the "services" folder represents a central piece of the frontend architecture. This JavaScript file orchestrates the integration and rendering of various UI components, effectively acting as the application's entry point. Through "app.jsx," the project establishes the core structure of the user interface, managing routing, navigation, and the composition of components. By encapsulating these responsibilities within a single file, the application benefits from enhanced organization and readability, allowing developers to grasp the application's flow and structure intuitively.

The provided code snippet represents the main entry point of a React application, showcasing the structure and organization of its components. Let's delve into an explanation of its functionality.

The `App` component is the central starting point of the application. It imports the "App.css" file, which is likely used for styling purposes throughout the application.

Here’s the code snippet of the App.jsx:



src > App.jsx > ...

```
import "./App.css";

const App = () => {
  return (
    <main>
      <div className="main">
        <div className="gradient" />
      </div>
      <div className="app">
        <Hero />
        <Demo />
      </div>
    </main>
  );
};

export default App;

```

Within the `return` statement, the code defines the layout of the main content of the application using JSX (JavaScript XML) syntax. JSX allows developers to write HTML-like code within JavaScript, making it seamless to create user interfaces.

1. `<main>` Tag: The entire content is wrapped within the `<main>` tag, which is a semantic HTML5 element representing the main content of a webpage.

2. `main` and `app` Divs: Inside the `<main>` tag, two `<div>` elements are used to structure the layout. The first `<div>` with the class name "main" is set to contain the background gradient of the application. This might serve as a decorative element to enhance the visual appeal of the page.

3. `gradient` Div: Inside the "main" `<div>`, there's another `<div>` with the class name "gradient." This likely represents a decorative element that utilizes CSS to create a gradient effect. Gradients are often used for backgrounds, borders, or other visual elements.

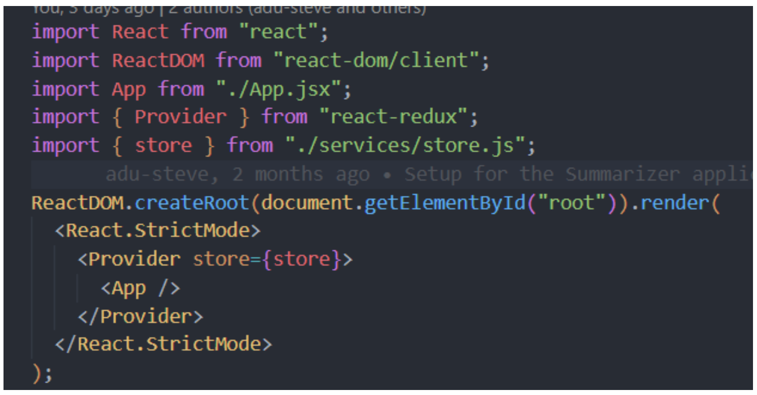
4. `app` Div: The second `<div>` with the class name "app" contains the core components of the application, namely `Hero` and `Demo`. These components appear to be imported from other files.

In summary, the `App` component serves as the container that orchestrates the layout of the application. It establishes a coherent structure by embedding different components within designated sections. This modular approach enhances code readability, maintainability, and scalability, which are fundamental principles in modern web development.

main.jsx:

The "main.jsx" file further enhances the application's functionality by serving as a bridge between the frontend and backend. This JavaScript file enables the incorporation of backend functionalities, API integrations, and data handling. Through "main.jsx," the project establishes a vital connection between user interactions and data manipulation, creating a seamless and responsive user experience. This file's role extends beyond rendering UI elements to ensuring that the application communicates effectively with external resources, enabling the extraction and presentation of article summaries through AI-powered processes.

Here's the code snippet of the main.jsx for this system:



```
You, 3 days ago | 2 authors (adu-steve and others)
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import { Provider } from "react-redux";
import { store } from "./services/store.js";
      adu-steve, 2 months ago • Setup for the Summarizer appli
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

The provided code snippet is a fundamental part of a React application's entry point. It plays a crucial role in rendering the user interface and establishing the application's architecture. Let's break down the code step by step to understand its functionality.

1. Imports: `React` and `ReactDOM`: These are essential libraries for building React applications. `React` is the core library that enables developers to create UI components, while `ReactDOM` focuses on rendering those components in the browser.

   - `App`: This import refers to the main application component, likely containing the entire structure of the app.

   - `Provider`: This import comes from the `react-redux` library, which facilitates the integration of React with Redux, a state management tool.

   - `store`: This import likely references the Redux store, which holds the global state of the application.

2. Rendering the Application: The `ReactDOM.createRoot` method is used to initiate the rendering of the application. It takes the `root` element, typically an HTML element with the ID "root," as its parameter. The `createRoot` method is part of an experimental feature in React that aims to improve performance and concurrent rendering.

3. JSX Elements: Within the `createRoot` method, a JSX element hierarchy is defined. This hierarchy encompasses the following components:

   - `<React.StrictMode>`: This is a wrapper component provided by React that helps detect and highlight potential issues in the application during development. It doesn't affect the production build.

   - `<Provider store={store}>`: This component wraps the entire application and provides the Redux store to all components within its scope. This allows components to access the application's global state.

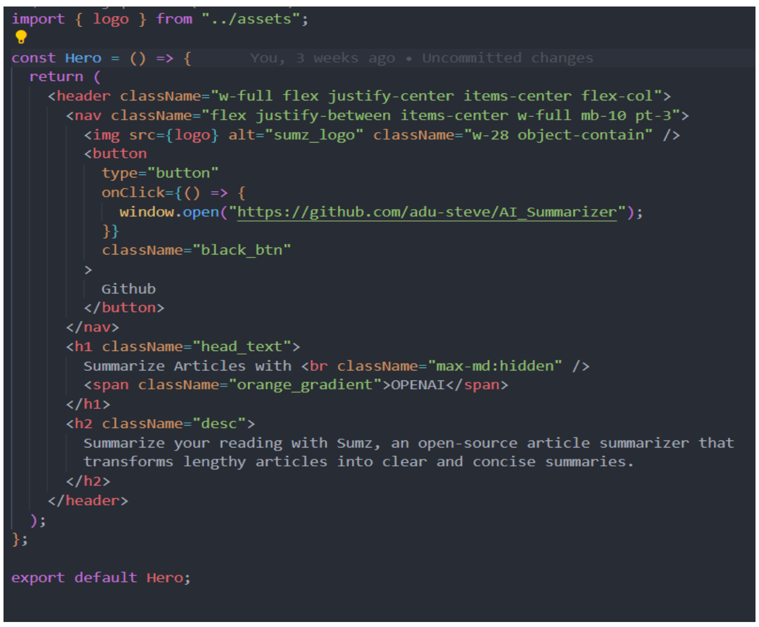
   - `<App />`: The main `App` component is placed within the `Provider` component. This structure ensures that the entire application has access to the Redux store and can utilize its state management capabilities.

4. Rendering and Display: After the JSX elements are constructed, the `.render` method is called on the root element. This triggers the rendering process of the application. The JSX hierarchy is evaluated, and the resulting UI is displayed in the browser.

In summary, the provided code establishes the foundation for a React application's rendering process and state management integration with Redux. It begins by importing essential libraries and components, then defines the JSX structure that constitutes the application's user interface. The `createRoot` method initiates the rendering process, and the `<Provider>` component ensures that the Redux store is available to all components. This code adheres to best practices by utilizing `React.StrictMode` for development checks and incorporating the Redux store for efficient state management. It illustrates the pivotal role of the entry point in structuring and rendering a React application with the necessary global state management capabilities.

System Function

The Hero Component.



```
import { logo } from "../assets";

const Hero = () => {        You, 3 weeks ago • Uncommitted changes
  return (
    <header className="w-full flex justify-center items-center flex-col">
      <nav className="flex justify-between items-center w-full mb-10 pt-3">
        <img src={logo} alt="sumz_logo" className="w-28 object-contain" />
        <button
          type="button"
          onClick={() => {
            window.open("https://github.com/adu-steve/AI_Summarizer");
          }}
          className="black_btn"
        >
          Github
        </button>
      </nav>
      <h1 className="head_text">
        Summarize Articles with <br className="max-md:hidden" />
        <span className="orange_gradient">OPENAI</span>
      </h1>
      <h2 className="desc">
        Summarize your reading with Sumz, an open-source article summarizer that
        transforms lengthy articles into clear and concise summaries.
      </h2>
    </header>
  );
};

export default Hero;
```

The provided code snippet represents a React functional component named "Hero." This component is responsible for rendering a header section on a web page, typically for an application designed to summarize articles using AI technology, particularly the OPENAI platform. The code utilizes JSX (JavaScript XML) syntax to define the structure and content of the header section. Let's break down the functionality of this component step by step.

1. Import:
The code begins by importing the "logo" image from the "../assets" directory. This image is likely a branding logo for the application and will be displayed in the header.

2. Component Structure:
The main function of the "Hero" component is defined using an arrow function. This functional component returns JSX elements that structure the header section of the web page.

3. Header Layout:
The outermost element of the header section is a "header" tag. It has several CSS classes that control its width, flex layout, and alignment of its children in a column formation (flex-col).

4. Navigation Section:
Within the header, there is a "nav" tag containing navigation elements. It uses the CSS classes to align its children horizontally (flex) with space between them (justify-between) and aligns them vertically (items-center). It also includes CSS classes for margin-bottom (mb-10) and padding-top (pt-3).

5. Logo Display:
Inside the "nav" tag, an "img" tag displays the logo image using the "src" attribute. The "alt" attribute provides alternative text for accessibility. The "w-28" CSS class sets the width of the image, and "object-contain" ensures the image fits within the designated space while maintaining its aspect ratio.

6. GitHub Button:
A "button" element is placed next to the logo. It is styled using the "black_btn" CSS class, suggesting it has a black background and white text. It has an "onClick" event listener that, when triggered, opens a new window or tab with the GitHub repository URL. The text within the button is "Github."

7. Title and Subtitle:
Following the navigation section, there are two "h1" and "h2" tags that display the application's title and a descriptive subtitle. The "head text" CSS class styles the title with specific formatting. The "br" tag with the "max-md: hidden" class ensures line break only on smaller screens. The "orange gradient" CSS class applies an orange gradient to the text "OPENAI." The "desc" CSS class styles the subtitle text.

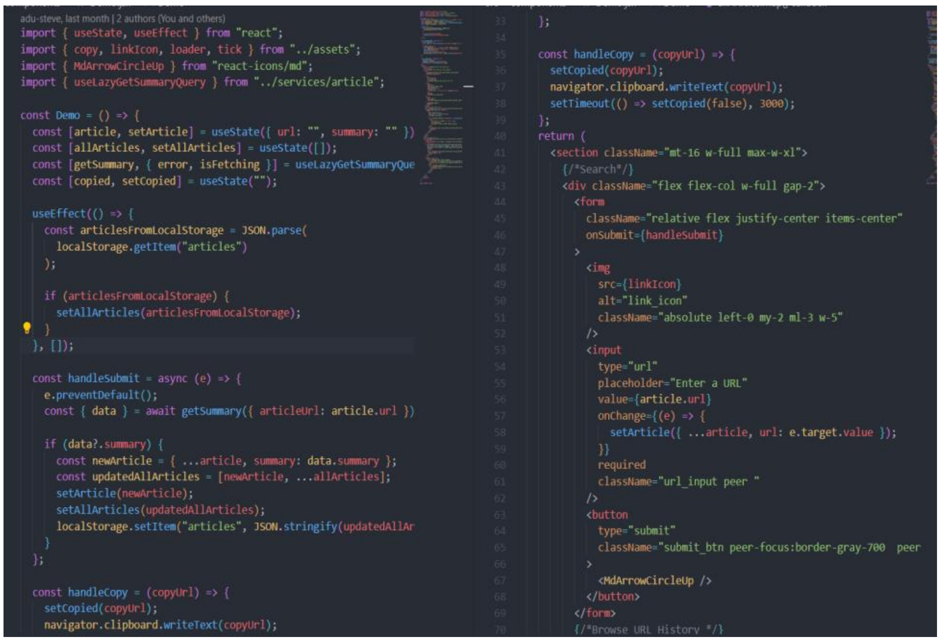
8. Component Export:
The "Hero" component is exported as the default export, indicating it can be imported and used in other parts of the application.

In summary, the "Hero" component creates a visually appealing and informative header section for an application called "Sumz," which aims to summarize articles using the OPENAI platform. The component includes a logo, a GitHub button, a title, and a descriptive subtitle. It effectively combines JSX syntax and Tailwind CSS classes to achieve its visual layout and interactivity, enhancing user engagement and communication of the application's purpose.

Here is the result of the codes above:

Sumz
Github
Summarize Articles with
OPENAI
Summarize your reading with Sumz, an open-source article summarizer that
transforms lengthy articles into clear and concise summaries.

The Demo Component.



```
adu-steve, last month | 2 authors (You and others)
import { useState, useEffect } from "react";
import { copy, linkIcon, loader, tick } from "../assets";
import { MdArrowCircleUp } from "react-icons/md";
import { useLazyGetSummaryQuery } from "../services/article";

const Demo = () => {
  const [article, setArticle] = useState({ url: "", summary: "" })
  const [allArticles, setAllArticles] = useState([]);
  const [getSummary, { error, isFetching }] = useLazyGetSummaryQue
  const [copied, setCopied] = useState("");

  useEffect(() => {
    const articlesFromLocalStorage = JSON.parse(
      localStorage.getItem("articles")
    );

    if (articlesFromLocalStorage) {
      setAllArticles(articlesFromLocalStorage);
    }
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();
    const { data } = await getSummary({ articleUrl: article.url })

    if (data?.summary) {
      const newArticle = { ...article, summary: data.summary };
      const updatedAllArticles = [newArticle, ...allArticles];
      setArticle(newArticle);
      setAllArticles(updatedAllArticles);
      localStorage.setItem("articles", JSON.stringify(updatedAllAr
    }
  };

  const handleCopy = (copyUrl) => {
    setCopied(copyUrl);
    navigator.clipboard.writeText(copyUrl);
```

```
  };

  const handleCopy = (copyUrl) => {
    setCopied(copyUrl);
    navigator.clipboard.writeText(copyUrl);
    setTimeout(() => setCopied(false), 3000);
  };
  return (
    <section className="mt-16 w-full max-w-xl">
      {/*Search*/}
      <div className="flex flex-col w-full gap-2">
        <form
          className="relative flex justify-center items-center"
          onSubmit={handleSubmit}
        >
          <img
            src={linkIcon}
            alt="link_icon"
            className="absolute left-0 my-2 ml-3 w-5"
          />
          <input
            type="url"
            placeholder="Enter a URL"
            value={article.url}
            onChange={(e) => {
              setArticle({ ...article, url: e.target.value });
            }}
            required
            className="url_input peer "
          />
          <button
            type="submit"
            className="submit_btn peer-focus:border-gray-700  peer
          >
            <MdArrowCircleUp />
          </button>
        </form>
        {/*Browse URL History */}
```

Certainly, let's break down the functionality of the "Demo" component and the code functions within it in detail.

The "Demo" component is a React functional component responsible for rendering a user interface that allows users to input article URLs, fetch article summaries from an API, manage a history of summarized articles, and copy URLs to the clipboard. The component employs various hooks, state variables, and interactions to achieve this functionality.

1. State Management:
   - `article`: A state object that holds the current article URL and its corresponding summary.
   - `allArticles`: A state array that stores the history of all summarized articles.
   - `getSummary`: A function returned from the `useLazyGetSummaryQuery` hook for fetching article summaries.
   - `error`: A variable that stores any error returned from the API.
   - `isFetching`: A boolean variable indicating whether data is being fetched.
   - `copied`: A state variable to track the URL that has been copied to the clipboard.

2. Local Storage and Effect:
   - The `useEffect` hook is employed to retrieve article history from local storage when the component mounts. If there are articles in local storage, they are loaded into the `allArticles` state.

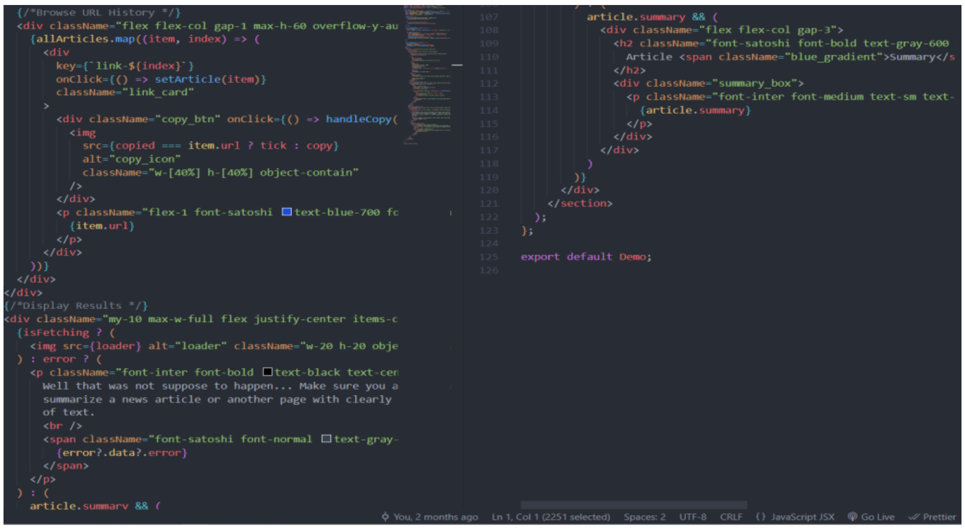
3. Handle Submit:
- The `handleSubmit` function is an asynchronous function triggered when the form is submitted. It prevents the default form behavior.
- It calls the `getSummary` function using the `useLazyGetSummaryQuery` hook, passing the article URL.
- If a valid summary is returned from the API response, a new article object is created with the URL and summary. The `allArticles` state is updated with the new article added at the beginning.
- The updated `allArticles` state is also stored in local storage.

4. Handle Copy:
- The `handleCopy` function is called when the copy button is clicked. It takes the URL of the article to copy.
- It sets the `copied` state to the URL being copied, writes the URL to the clipboard using the `navigator.clipboard.writeText` method, and then resets the `copied` state after a delay.

5. User Interface Sections:
- Search Section: A form for inputting article URLs. The form consists of an input field, a search icon, and a submit button. The `onSubmit` event of the form triggers the `handleSubmit` function.
- Browse URL History Section: Displays a list of summarized articles. Each article is shown in a "link_card" with a copy button. Clicking on an article sets it as the current article being displayed, and the URL is copied when the copy button is clicked.
- Display Results Section: Displays the article summary, loading spinner, or error message based on the API fetch status.

The code snippet demonstrates how the React component effectively integrates various functionalities to provide users with an interactive platform to input URLs, fetch article summaries, manage history, and copy URLs. The use of hooks like `useState`, `useEffect`, and `useLazyGetSummaryQuery` illustrates how React simplifies state management and API interactions. The HTML structure combined with CSS classes from the Tailwind CSS framework ensures an appealing and responsive user interface.

In essence, the "Demo" component encapsulates the core features of an article summarization application, showcasing how React can be used to build dynamic and user-friendly web applications.



```
{/*Browse URL History */}
<div className="flex flex-col gap-1 max-h-60 overflow-y-au
  {allArticles.map((item, index) => (
    <div
      key={`link-${index}`}
      onClick={() => setArticle(item)}
      className="link_card"
    >
      <div className="copy_btn" onClick={() => handleCopy(
        <img
          src={copied === item.url ? tick : copy}
          alt="copy_icon"
          className="w-[40%] h-[40%] object-contain"
        />
      </div>
      <p className="flex-1 font-satoshi text-blue-700 fc
        {item.url}
      </p>
    </div>
  ))}
</div>
</div>
{/*Display Results */}
<div className="my-10 max-w-full flex justify-center items-c
  {isFetching ? (
    <img src={loader} alt="loader" className="w-20 h-20 obje
  ) : error ? (
    <p className="font-inter font-bold text-black text-cen
      Well that was not suppose to happen... Make sure you a
      summarize a news article or another page with clearly
      of text.
      <br />
      <span className="font-satoshi font-normal text-gray-
        {error?.data?.error}
      </span>
    </p>
  ) : (
    article.summary && (
```

```
          article.summary && (
            <div className="flex flex-col gap-3">
              <h2 className="font-satoshi font-bold text-gray-600
                Article <span className="blue_gradient">Summary</s
              </h2>
              <div className="summary_box">
                <p className="font-inter font-medium text-sm text-
                  {article.summary}
                </p>
              </div>
            </div>
          )
        )}
      </div>
    </section>
  );
};

export default Demo;
```

You, 2 months ago   Ln 1, Col 1 (2251 selected)   Spaces: 2   UTF-8   CRLF   {} JavaScript JSX   Go Live   Prettier

6. Browse URL History:
  This section renders a list of previously summarized articles. For each item in the `allArticles` array, a "link_card" element is generated. Clicking on a card sets that article as the current one to display. Here's what each part does:
  - The `.map()` function iterates through each item in the `allArticles` array and generates a card for it.
  - Each card has a `key` attribute set to a unique identifier based on the index.
  - The `onClick` handler attached to the card sets the clicked article as the current article being displayed when clicked.
  - The "link_card" class styles the appearance of each card.

7. Copy Button and Icon:
  Within each card, there's a "copy_btn" element that, when clicked, triggers the `handleCopy` function to copy the article's URL to the clipboard. The appearance of the button changes based on whether the URL has been copied or not:
  - The `onClick` handler on the button invokes the `handleCopy` function, passing the item's URL.
  - The `img` element inside the button displays either the "tick" or "copy" icon based on whether the URL has been copied (`copied === item.url`).

8. Article URL Display:
  Each card also displays the URL of the summarized article. The `item.url` is shown within a `p` element with styling applied:
  - The `font-satoshi` and `font-medium` classes define the font style.
  - The `text-blue-700` class sets the text color.
  - The `truncate` class truncates long URLs if they exceed the available space.

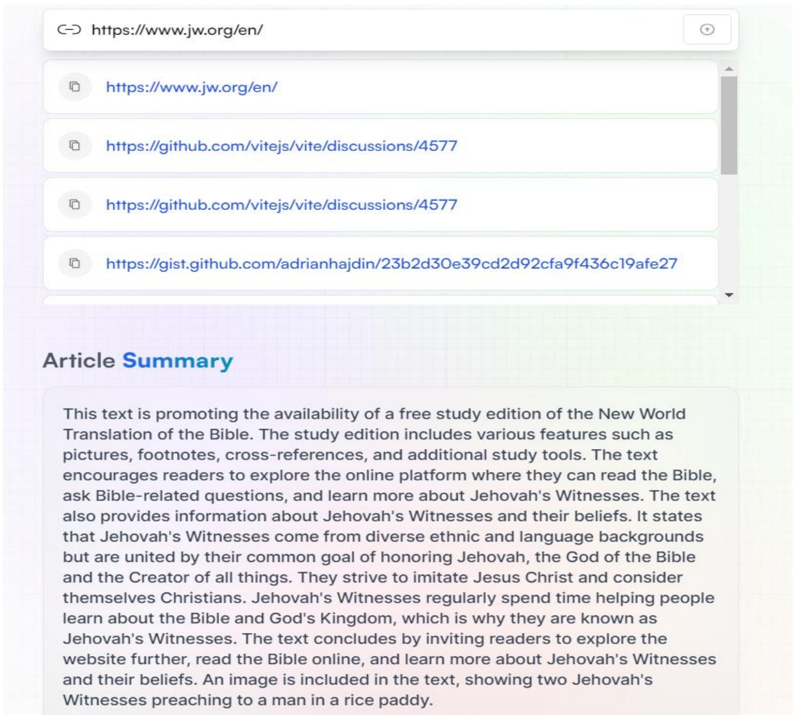
9. Display Results:
  This section is responsible for displaying the fetched article summary or error message, or a loading spinner if data is being fetched.
  - If `isFetching` is true, a loading spinner (loader) is displayed.

- If `error` is truthy, an error message is displayed, indicating that the summarization attempt encountered a problem. The error message and its details are fetched from the API response.
- If the `article.summary` exists, it is displayed as the summarized content within the "summary_box" element.

The entire "Demo" component integrates these sections to create a user-friendly interface for summarizing articles. Users can input URLs, view their history, and see summarized content. The conditional rendering based on `isFetching`, `error`, and `article.summary` ensures that the UI responds appropriately to the API interaction. The Tailwind CSS classes applied to different elements ensure consistent and visually appealing styling.

In conclusion, this part of the code enhances the user experience by displaying a history of summarized articles, allowing for easy copying of URLs, and providing clear feedback on the summarization process, whether it's successful, in progress, or encountering errors. The combination of React components, hooks, and dynamic styling creates a cohesive and functional UI for the article summarization application.

Here is the result of the codes written in the Demo Component with an example:



https://www.jw.org/en/

https://www.jw.org/en/

https://github.com/vitejs/vite/discussions/4577

https://github.com/vitejs/vite/discussions/4577

https://gist.github.com/adrianhajdin/23b2d30e39cd2d92cfa9f436c19afe27

Article **Summary**

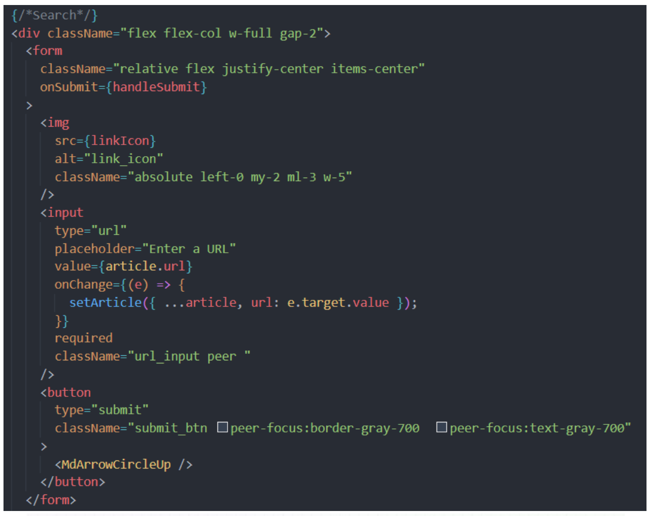
This text is promoting the availability of a free study edition of the New World Translation of the Bible. The study edition includes various features such as pictures, footnotes, cross-references, and additional study tools. The text encourages readers to explore the online platform where they can read the Bible, ask Bible-related questions, and learn more about Jehovah's Witnesses. The text also provides information about Jehovah's Witnesses and their beliefs. It states that Jehovah's Witnesses come from diverse ethnic and language backgrounds but are united by their common goal of honoring Jehovah, the God of the Bible and the Creator of all things. They strive to imitate Jesus Christ and consider themselves Christians. Jehovah's Witnesses regularly spend time helping people learn about the Bible and God's Kingdom, which is why they are known as Jehovah's Witnesses. The text concludes by inviting readers to explore the website further, read the Bible online, and learn more about Jehovah's Witnesses and their beliefs. An image is included in the text, showing two Jehovah's Witnesses preaching to a man in a rice paddy.

Chapter Three-Implementation and Results.

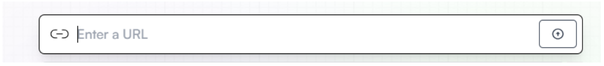
Testing

The testing of this system has been made easy to use and implement. It does not require long method or process to execute it functionality. This helps save time and display the results faster. Here is an example of how the system functionality is implemented.

A search engine is created to take input of the article's URL and process it summary at the backend. Here's the code snippet for creating the input space provided together with the image of the input bar with a placeholder of "Enter a URL".

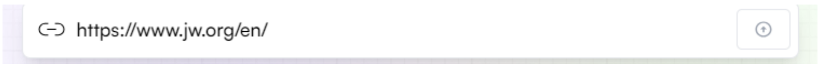


```
{/*Search*/}
<div className="flex flex-col w-full gap-2">
  <form
    className="relative flex justify-center items-center"
    onSubmit={handleSubmit}
  >
    <img
      src={linkIcon}
      alt="link_icon"
      className="absolute left-0 my-2 ml-3 w-5"
    />
    <input
      type="url"
      placeholder="Enter a URL"
      value={article.url}
      onChange={(e) => {
        setArticle({ ...article, url: e.target.value });
      }}
      required
      className="url_input peer "
    />
    <button
      type="submit"
      className="submit_btn ☐peer-focus:border-gray-700  ☐peer-focus:text-gray-700"
    >
      <MdArrowCircleUp />
    </button>
  </form>
```



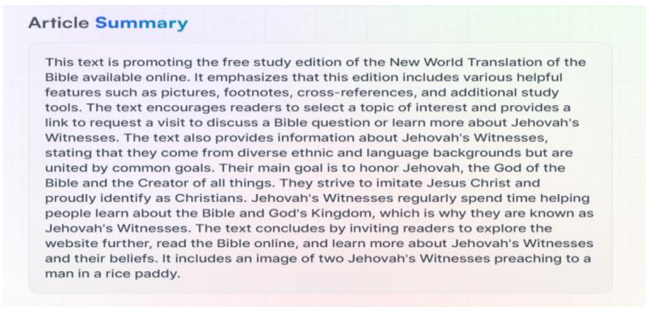
Let say a person wants to insert a URL of https://www.jw.org/en/.

They can type out the URL or copy and paste it in the input space provided. They can click the Enter key or hit the button beside the input space on the far right.



After typing and hitting the Enter key or the button, a loader will appear indicating to the user that the system is trying to summarize the articles on the page provided by the URL.

Here is the result given by the system in summarizing the URL of https://www.jw.org/en/.



Article Summary

This text is promoting the free study edition of the New World Translation of the Bible available online. It emphasizes that this edition includes various helpful features such as pictures, footnotes, cross-references, and additional study tools. The text encourages readers to select a topic of interest and provides a link to request a visit to discuss a Bible question or learn more about Jehovah's Witnesses. The text also provides information about Jehovah's Witnesses, stating that they come from diverse ethnic and language backgrounds but are united by common goals. Their main goal is to honor Jehovah, the God of the Bible and the Creator of all things. They strive to imitate Jesus Christ and proudly identify as Christians. Jehovah's Witnesses regularly spend time helping people learn about the Bible and God's Kingdom, which is why they are known as Jehovah's Witnesses. The text concludes by inviting readers to explore the website further, read the Bible online, and learn more about Jehovah's Witnesses and their beliefs. It includes an image of two Jehovah's Witnesses preaching to a man in a rice paddy.

This is much better compared to reading a whole lot of texts on a website. Also, the process is simple and easy to implement.

Chapter Four - Findings and Conclusions

Challenges Faced

1. AI Integration Complexity: Integrating AI models like GPT-4 requires a deep understanding of machine learning concepts, which can pose a challenge for developers with limited AI experience.

2. Data Privacy and Security: Handling user data, especially URLs, necessitates robust security measures to ensure user privacy and prevent potential data breaches.

3. Algorithmic Refinement: Developing algorithms to accurately and concisely summarize articles while maintaining coherence and relevance can be complex.

4. Adoption Barrier: Convincing users to adopt a new tool and change their content consumption habits may require persuasive marketing strategies.

5. Dynamic Content Handling: Ensuring the tool functions effectively with varying content structures and formats can be challenging

Findings and Conclusions

1. Relevance of AI in Information Absorption: The project highlights the transformative potential of AI in simplifying complex tasks. The integration of the GPT-4 model into the article summarization process significantly enhances the efficiency and effectiveness of information absorption.

2. Enhanced User Experience: The user-friendly web application offers a streamlined solution for users seeking to extract key insights from lengthy articles. The project's intuitive interface, powered by technologies like React and Tailwind CSS, ensures a seamless and efficient user experience.

3. Educational Value: The project serves as an educational platform for developers seeking to enhance their skills. It offers practical experience in web development, API integration, and AI implementation, making it an effective gateway for students and aspiring developers.

The project successfully demonstrates the potential of AI in addressing real-world challenges. Through the creation of an AI-powered summarization tool, it offers a tangible solution to the problem of information overload. The user-friendly interface, coupled with the power of the GPT-4 model, showcases the capabilities of AI in transforming complex tasks into seamless experiences.

Furthermore, the project aligns with the demands of contemporary digital interactions, where efficient content consumption is paramount. By democratizing access to knowledge and simplifying learning, it contributes to the growth and advancement of individuals across various fields.

Recommendations for Future Works and Commercialization

1. Content Extraction Enhancement: While the current project focuses on article summarization, future iterations could explore the possibility of content extraction from various sources beyond articles. This would expand the tool's functionality and utility.

2. Multi-Format Support: Incorporating support for different content formats, such as videos or podcasts, would broaden the tool's scope and cater to a wider range of digital media.

3. Customization Options: Introducing customization options for users to adjust the level of summarization or select specific content segments to include in the summary would enhance the tool's flexibility.

4. Collaborative Summarization: Enabling users to collaborate on summarizing articles could promote knowledge-sharing and collaborative research.

The project holds significant commercial potential, particularly in the education, research, and content consumption sectors. To commercialize the project effectively, the following steps are recommended:

1. Market Research: Conduct thorough market research to identify target audiences, assess competition, and determine potential pricing strategies.

2. User Engagement: Develop engaging marketing strategies, including social media campaigns, tutorials, and demonstrations, to showcase the tool's benefits and features.

3. Monetization Model: Consider offering freemium and premium versions of the tool, where premium users may have access to advanced features or more extensive usage limits.

4. Partnerships: Collaborate with educational institutions, online platforms, and content providers to promote the tool and explore potential integration opportunities

In conclusion, the "AI-Powered Article Summarization for Efficient Information Absorption (GPT-4 MODEL)" project showcases the potential of AI to address real-world challenges. By creating a user-friendly application that offers article summaries, the project demonstrates the transformative impact of AI in enhancing information consumption. With careful consideration of future enhancements, effective commercialization strategies, and the willingness to address challenges, the project can contribute significantly to improving how people interact with information in the digital era.

References

1. Visual Studio Code. (n.d.). Retrieved from https://code.visualstudio.com/

2. Node.js. (n.d.). Retrieved from https://nodejs.org/

3. React. (n.d.). Retrieved from https://reactjs.org/

4. Rapid API. (n.d.). Retrieved from https://rapidapi.com/

5. Vite. (n.d.). Retrieved from https://vitejs.dev/

6. Node Package Manager (NPM). (n.d.). Retrieved from https://www.npmjs.com/

7. React-Redux. (n.d.). Retrieved from https://react-redux.js.org/

8. React-Icons. (n.d.). Retrieved from https://react-icons.github.io/react-icons/

9. Tailwind CSS. (n.d.). Retrieved from https://tailwindcss.com/

10. GPT-4 Model. (n.d.). Retrieved from [Provide relevant source or documentation from OpenAI regarding GPT-4]

11. Tailwind CSS Documentation. (n.d.). Retrieved from https://tailwindcss.com/docs

12. React Documentation. (n.d.). Retrieved from https://reactjs.org/docs/getting-started.html

13. Rapid API Documentation. (n.d.). Retrieved from https://docs.rapidapi.com/docs