Jaypee University of Information Technology
Department of Computer Science and Engineering

Project Report

"GRE Vocab Dictionary"

Course Code : 18B17CI474
Course Name : WEB TECH LAB

Submitted by:

Name : Parth Sharma
Roll no : 211106
Batch : CS41

Submitted to:

Name : Dr. Sunil Datt Sharma

1 | Abstract

This report, entitled "GRE Vocab Dictionary", contains a brief introduction to the different techniques used for making this project their implementation. This report is put together by me as the fulfilment to the lab project assigned by **Dr.Sunil Datt Sharma.**

2 | Acknowledgment

I hereby acknowledge that the material used/written in this report is not, wholly or partially, plagiarized; proper references have been mentioned where we used external printed/written/published material, and the instructor has the right to reject a portion of the report or the report altogether if he finds the fact to be something else than what has been mentioned above.

I have tried my level best to keep the references correct and the facts & figures up-to-date, but being human, any errors or lapses are expected and are requested to be tolerated.

Parth Sharma

[CSE/211106]

Table of Contents

TOPIC

3 | Introduction - GRE Vocab Dictionary

The Graduate Record Examination, commonly known as GRE, is a standardized test required for admission to graduate and business schools in various countries, including the United States, Canada, and Australia. One of the most challenging parts of the GRE is the vocabulary section, which tests a candidate's knowledge of English words and phrases.

The GRE vocabulary section aims to evaluate a candidate's understanding of the English language by measuring their ability to read and comprehend complex texts. The vocabulary section consists of two parts: Text Completion and Sentence Equivalence. In the Text Completion section, a sentence is presented with one, two, or three blanks, and the candidate must select the best answer from a list of possible words or phrases to fill in the blanks. In the Sentence Equivalence section, a sentence is presented with one blank, and the candidate must select two answers from a list of possible words or phrases that would create two sentences with equivalent meanings.

The GRE vocabulary section is considered challenging because it requires a broad understanding of complex vocabulary and nuanced meanings. The exam draws from a wide range of subjects, including literature, history, and science, and it is not uncommon for candidates to encounter unfamiliar words. Therefore, it is essential for candidates to develop a strong vocabulary base to perform well on the GRE.

To prepare for the GRE vocabulary section, candidates should consider learning new words and their meanings regularly. There are various online resources available, such as flashcards, vocabulary lists, and quizzes, which can help candidates improve their vocabulary. In addition, reading extensively and regularly can also help candidates develop their vocabulary, comprehension, and critical reading skills.

In summary, the GRE vocabulary section is an essential part of the exam, and it measures a candidate's ability to understand complex English words and phrases. To perform well on the exam, candidates should develop a strong vocabulary base by using various resources and reading extensively.

4 | Technologies Used

The code includes several technologies and libraries :

1. **HTML**: The code is written using HTML (Hypertext Markup Language), which is the standard markup language for creating web pages.

2. **CSS**: The code includes CSS (Cascading Style Sheets) for styling and formatting the HTML elements.

3. **JavaScript**: Although not explicitly visible in the provided code, there might be JavaScript used to handle dynamic functionality or interactivity on the web page. JavaScript is a programming language commonly used for client-side scripting in web development.

4. **Google Fonts**: The code imports and uses fonts from Google Fonts. Google Fonts is a library of freely available fonts that can be easily embedded into web pages.

5. **Font Awesome:** The code includes a library of icons used to display search, close, and volume icons.

6. **CDN (Content Delivery Network):** CDN is used to serve the Font Awesome icons and Google Fonts CSS file from a remote server.

7. **Meta tags**: The code includes several meta tags for defining the character set, viewport settings, format detection, and compatibility with Internet Explorer.

8. **Anchor tags**: The code includes anchor tags (`<a>`) for creating hyperlinks to different sections or external websites.

9. **Favicon**: The code includes a favicon link tag to specify the website's icon that appears in the browser's tab or bookmark bar.

5 | REQUIREMENTS ANALYSIS

Here's a high-level requirement analysis:

1. **Structuring :** The HTML code creates a structure for the page, including a header, a search bar, and three lists for displaying the word, meaning, example, and synonyms of a searched word.
2. **Styling and Formatting**: The presence of CSS indicates the requirement for customized styling and formatting of HTML elements. The code likely aims to provide an aesthetically pleasing design, with attention to colors, fonts, spacing, and visual hierarchy.
3. **Navigation**: The usage of unordered lists and anchor tags suggests the need for a navigation menu or links to different sections within the webpage or external websites. The requirement could involve creating a user-friendly navigation system to enable easy access to various parts of the website.
4. **Accessibility:** While not explicitly mentioned in the code, accessibility is an important consideration in web development. The requirement may include ensuring that the webpage is accessible to users with disabilities, following web accessibility guidelines such as providing alternative text for images and semantic markup for improved screen reader compatibility.
5. **Performance Optimization**: While not directly visible in the code snippet, performance optimization is a crucial requirement for web development. The final website should load quickly and efficiently to provide a smooth user experience. This requirement may involve optimizing image sizes, minifying code, leveraging caching mechanisms, and following best practices for web performance.

6 | System Design: Hierarchy of react components

Here is a suggested hierarchy of React components for the provided HTML and CSS code:

1. **App:** top-level component that renders the whole application.
2. **SearchBar:** component that contains the search input and search button/icon.
3. **WordCard:** component that displays the word and its basic information (e.g. pronunciation).
4. **AudioButton:** component that plays the pronunciation audio when clicked.
5. **MeaningCard:** component that displays the meaning and example sentence(s) of the word.
6. **SynonymsCard:** component that displays the synonyms of the word.

Note that some components may have child components (e.g. SearchBar has both input and button/icon), and some components may not be displayed initially (e.g. MeaningCard and SynonymsCard are initially hidden until a search is made). Additionally, there may be additional components that are not explicitly defined in the HTML and CSS, such as a component for making API calls to retrieve the word data.

DIAGRAMM

HTML Document

```
<App />
  <SearchBar />
    <input />
    <i className="fas fa-search" />
    <span className="material-icons">close</span>
  <p className="info-text" />
  <Word />
    <div className="details">
      <p>__</p>
      <span>_ _</span>
    </div>
    <i className="fas fa-volume-up" />
    <Content />
      <Meaning />
        <div className="details">
          <p>Meaning</p>
          <span>___</span>
        </div>
      </Meaning>
      <Example />
        <div className="details">
          <p>Example</p>
          <span>___</span>
        </div>
      </Example>
      <Synonyms />
        <div className="details">
          <p>Synonyms</p>
          <div className="list"></div>
        </div>
      </Synonyms>
```

7 | System Architecture

The system architecture for the provided code can be described as a client-server architecture, where the client is a web browser and the server hosts the website.

1. **Client-Side:**
    - The client-side architecture consists of a web browser, such as Google Chrome or Mozilla Firefox, which renders and displays the website to the user.
    - The browser interprets the HTML, CSS, and JavaScript code received from the server and renders the webpage accordingly.
    - The client-side code includes HTML markup for the structure of the webpage, CSS for styling, and JavaScript for client-side interactivity.

2. **Server-Side:**
    - The server-side architecture handles the requests from the client and serves the appropriate responses.
    - The server hosts the website and responds to HTTP requests from the client.
    - It runs server-side scripts or applications that generate the dynamic content of the webpages and retrieve data from databases or other sources.
    - The server-side code includes server-side scripting languages such as PHP, Python, or Node.js, which generate HTML dynamically and interact with databases if required.

3. **Communication:**
    - The client communicates with the server using the HTTP protocol.
    - When the client requests a webpage, it sends an HTTP GET request to the server, specifying the URL of the webpage.
    - The server receives the request, processes it, and generates an appropriate response. - The response typically includes an HTML document along with any additional resources like CSS files, JavaScript files, or images referenced by the HTML.
    - The client receives the response and renders the webpage based on the received HTML, CSS, and JavaScript.

4. **File Structure**:
    - The code snippet provided includes HTML markup, CSS styles, and references to external resources such as images and fonts.
    - The server is responsible for serving these files to the client when requested.
    - The server may have a specific file structure where the HTML, CSS, JavaScript, and other assets are organized in directories for easy access and maintenance.

    - Overall, the architecture follows the standard client-server model where the client (web browser) makes requests to the server, and the server responds with the necessary resources and data to render the webpage on the client-side.

8 | Break Down of the project

The above code is an HTML, CSS, and JavaScript code that creates a simple vocabulary dictionary web page.

The HTML code includes the basic structure of the page, including the page title, external style sheets, a viewport meta tag, and links to icon libraries. It contains a search box to input words, a block of content to display the details of the word, and a section for definitions, examples, and synonyms.

The CSS code includes style rules that define the layout and appearance of the page. It includes font styling, background color, and margin, padding, and border settings for different elements on the page. The CSS also includes rules to create a search bar with a magnifying glass icon and a close icon. When the user starts typing, the search bar enlarges, and the close icon appears. Additionally, the CSS defines the properties of the list items in the content section, including their size, weight, color, and border properties.

The JavaScript code is used to add interactivity to the web page. It includes an event listener that listens for when the user types in the search bar and retrieves the information for the input word from a remote server. The JavaScript also includes code to display and hide the content section of the page when the user clicks on the word item. The code also handles the display of synonyms, meaning, and examples of the word.

Overall, the HTML, CSS, and JavaScript code works together to create a simple and functional vocabulary dictionary web page.

 # HTML Code - GRE Vocab Dictionary

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>GRE Vocab Dictionary</title>
    <link rel="stylesheet" href=".\project1.css">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- CDN Link for Icons -->
    <link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.3/css/all.min.css"/>
    <link rel="icon" href="favicon.ico">
  </head>
  <body>
    <div class="wrapper">
      <header>GRE Vocab Dictionary</header>
      <div class="search">
        <input type="text" placeholder="Search a word" required
spellcheck="false">
        <i class="fas fa-search"></i>
        <span class="material-icons">close</span>
      </div>
      <p class="info-text">Type any existing word and press enter to get
meaning, example, synonyms, etc.</p>
      <ul>
        <li class="word">
          <div class="details">
            <p>__</p>
            <span>_ _</span>
          </div>
          <i class="fas fa-volume-up"></i>
        </li>
        <div class="content">
          <li class="meaning">
            <div class="details">
              <p>Meaning</p>
              <span>___</span>
            </div>
          </li>
          <li class="example">
            <div class="details">
```

```
            <p>Example</p>
            <span>___</span>
          </div>
        </li>
        <li class="synonyms">
          <div class="details">
            <p>Synonyms</p>
            <div class="list"></div>
          </div>
        </li>
      </div>
    </ul>
  </div>
</body>
<script src=".\project1.js"></script>
</html>
```

 # CSS Code

```
/* Import Google Font - Poppins */
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700
&display=swap');
*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Poppins', sans-serif;
}
body{
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  background: #4D59FB;
}
::selection{
  color: #fff;
  background: #4D59FB;
}
.wrapper{
  width: 420px;
  border-radius: 7px;
  background: #fff;
  padding: 25px 28px 45px;
  box-shadow: 7px 7px 20px rgba(0, 0, 0, 0.05);
}
.wrapper header{
  font-size: 28px;
  font-weight: 500;
  text-align: center;
}
.wrapper .search{
  position: relative;
  margin: 35px 0 18px;
}
.search input{
  height: 53px;
  width: 100%;
  outline: none;
  font-size: 16px;
  border-radius: 5px;
```

```
  padding: 0 42px;
  border: 1px solid #999;
}
.search input:focus{
  padding: 0 41px;
  border: 2px solid #4D59FB;
}
.search input::placeholder{
  color: #B8B8B8;
}
.search :where(i, span){
  position: absolute;
  top: 50%;
  color: #999;
  transform: translateY(-50%);
}
.search i{
  left: 18px;
  pointer-events: none;
  font-size: 16px;
}
.search input:focus ~ i{
  color: #4D59FB;
}
.search span{
  right: 15px;
  cursor: pointer;
  font-size: 18px;
  display: none;
}
.search input:valid ~ span{
  display: block;
}
.wrapper .info-text{
  font-size: 13px;
  color: #9A9A9A;
  margin: -3px 0 -10px;
}
.wrapper.active .info-text{
  display: none;
}
.info-text span{
  font-weight: 500;
}
.wrapper ul{
```

```
  height: 0;
  opacity: 0;
  padding-right: 1px;
  overflow-y: hidden;
  transition: all 0.2s ease;
}
.wrapper.active ul{
  opacity: 1;
  height: 303px;
}
.wrapper ul li{
  display: flex;
  list-style: none;
  margin-bottom: 14px;
  align-items: center;
  padding-bottom: 17px;
  border-bottom: 1px solid #D9D9D9;
  justify-content: space-between;
}
ul li:last-child{
  margin-bottom: 0;
  border-bottom: 0;
  padding-bottom: 0;
}
ul .word p{
  font-size: 22px;
  font-weight: 500;
}
ul .word span{
  font-size: 12px;
  color: #989898;
}
ul .word i{
  color: #999;
  font-size: 15px;
  cursor: pointer;
}
ul .content{
  max-height: 215px;
  overflow-y: auto;
}
ul .content::-webkit-scrollbar{
  width: 0px;
}
.content li .details{
```

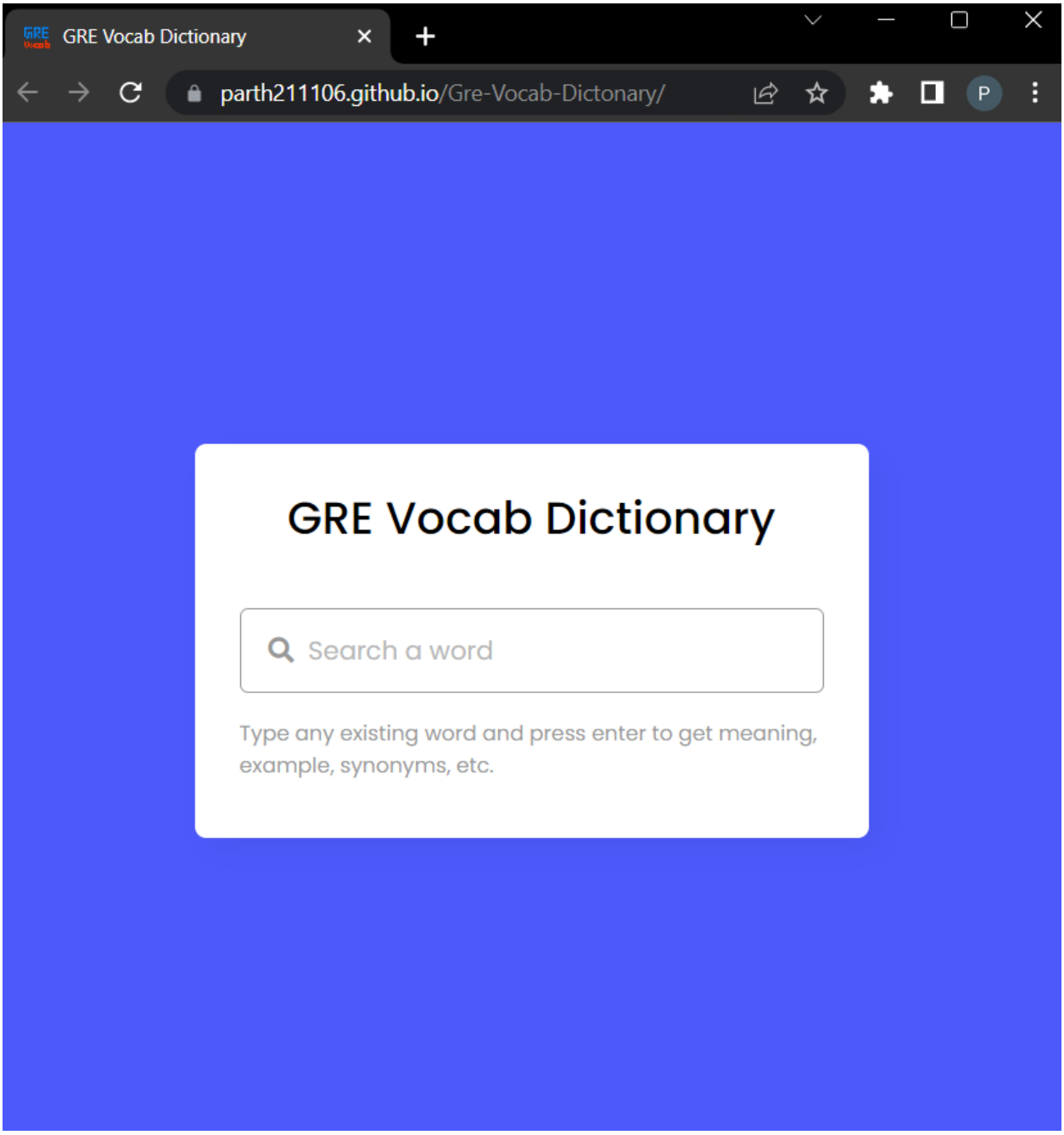
```
  padding-left: 10px;
  border-radius: 4px 0 0 4px;
  border-left: 3px solid #4D59FB;
}
.content li p{
  font-size: 17px;
  font-weight: 500;
}
.content li span{
  font-size: 15px;
  color: #7E7E7E;
}
.content .synonyms .list{
  display: flex;
  flex-wrap: wrap;
}
.content .synonyms span{
  cursor: pointer;
  margin-right: 5px;
  text-decoration: underline;
}
```

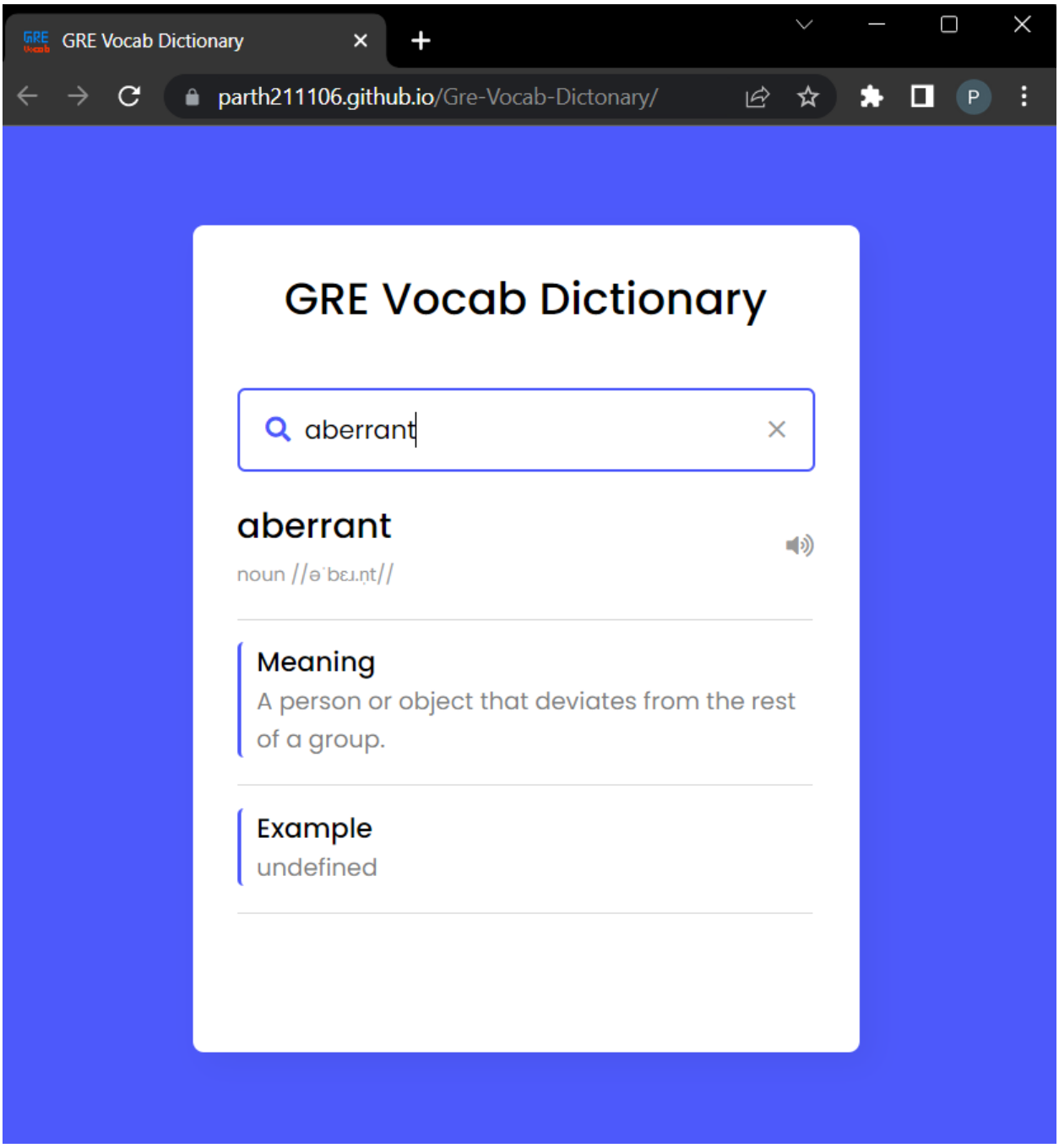
 # Javascript Code

```
const wrapper = document.querySelector(".wrapper"),
searchInput = wrapper.querySelector("input"),
volume = wrapper.querySelector(".word i"),
infoText = wrapper.querySelector(".info-text"),
synonyms = wrapper.querySelector(".synonyms .list"),
removeIcon = wrapper.querySelector(".search span");
let audio;

function data(result, word){
    if(result.title){
        infoText.innerHTML = `Can't find the meaning of
<span>"${word}"</span>. Please, try to search for another word.`;
    }else{
        wrapper.classList.add("active");
        let definitions = result[0].meanings[0].definitions[0],
        phontetics =
`${result[0].meanings[0].partOfSpeech}  /${result[0].phonetics[0].text}/`;
        document.querySelector(".word p").innerText = result[0].word;
        document.querySelector(".word span").innerText = phontetics;
        document.querySelector(".meaning span").innerText =
definitions.definition;
        document.querySelector(".example span").innerText =
definitions.example;
        audio = new Audio(result[0].phonetics[0].audio);

        if(definitions.synonyms[0] == undefined){
            synonyms.parentElement.style.display = "none";
        }else{
            synonyms.parentElement.style.display = "block";
            synonyms.innerHTML = "";
            for (let i = 0; i < 5; i++) {
                let tag = `<span
onclick="search('${definitions.synonyms[i]}')">${definitions.synonyms[i]},
</span>`;
                tag = i == 4 ? tag = `<span
onclick="search('${definitions.synonyms[i]}')">${definitions.synonyms[4]}<
/span>` : tag;
                synonyms.insertAdjacentHTML("beforeend", tag);
            }
        }
    }
}
```

```
function search(word){
    fetchApi(word);
    searchInput.value = word;
}

function fetchApi(word){
    wrapper.classList.remove("active");
    infoText.style.color = "#000";
    infoText.innerHTML = `Searching the meaning of
<span>"${word}"</span>`;
    let url = `https://api.dictionaryapi.dev/api/v2/entries/en/${word}`;
    fetch(url).then(response => response.json()).then(result =>
data(result, word)).catch(() =>{
        infoText.innerHTML = `Can't find the meaning of
<span>"${word}"</span>. Please, try to search for another word.`;
    });
}

searchInput.addEventListener("keyup", e =>{
    let word = e.target.value.replace(/\s+/g, ' ');
    if(e.key == "Enter" && word){
        fetchApi(word);
    }
});

volume.addEventListener("click", ()=>{
    volume.style.color = "#4D59FB";
    audio.play();
    setTimeout(() =>{
        volume.style.color = "#999";
    }, 800);
});

removeIcon.addEventListener("click", ()=>{
    searchInput.value = "";
    searchInput.focus();
    wrapper.classList.remove("active");
    infoText.style.color = "#9A9A9A";
    infoText.innerHTML = "Type any existing word and press enter to get
meaning, example, synonyms, etc.";
});
```

 # Output


GRE Vocab Dictionary
parth211106.github.io/Gre-Vocab-Dictonary/
GRE Vocab Dictionary
Search a word
Type any existing word and press enter to get meaning, example, synonyms, etc.


GRE Vocab Dictionary
parth211106.github.io/Gre-Vocab-Dictonary/
GRE Vocab Dictionary
aberrant
aberrant
noun //əˈbɛɹ.nt//
Meaning
A person or object that deviates from the rest of a group.
Example
undefined

9 | Conclusion and Future work:

This is an HTML and CSS code for a GRE Vocab Dictionary webpage. The HTML code creates the structure and content of the webpage, while the CSS code defines the styling.

The HTML code includes the doctype declaration, html and head tags that contain meta data such as character set, viewport, title, links to external stylesheets and icons, and the JavaScript file. In the body, there is a wrapper div that contains the header, search bar, info text and a list that has a word, meaning, example and synonyms.

The CSS code starts with importing the Google font Poppins. The * selector applies margin, padding, and box-sizing to all elements, and sets the font-family to Poppins. The body is set to display as a flex container with its contents centered both vertically and horizontally, with a minimum height of 100vh and a background color of #4D59FB. The ::selection pseudo-element is styled to change the text color and background color of the selected text. The wrapper div has a fixed width of 420px, a border-radius of 7px, a white background, some padding, and a box-shadow. The header has a font size of 28px, font weight of 500, and text-align of center. The search div is positioned relatively and has some margin. The search input has a height of 53px, width of 100%, a font-size of 16px, border-radius of 5px, some padding, and a border. When the input is focused, the padding and border increase in width. The search i and span are positioned absolutely and have a color of #999. The i is positioned to the left of the input, and the span is positioned to the right. The i has pointer-events disabled. When the input is focused, the color of the i changes to #4D59FB. The span is hidden by default, but appears when the input has valid content. The info-text has a font-size of 13px and a color of #9A9A9A. When the wrapper has a class of active, the info-text is hidden. The ul has a height of 0 and opacity of 0, with overflow-y hidden and a transition. When the wrapper has a class of active, the ul's opacity is set to 1 and height to 303px. The li elements are styled to display as flex, have no list-style, some margin, and align-items and justify-content set to center and space-between, respectively. The border-bottom and padding-bottom are set to differentiate the li elements. The word element has a font size of 22px and font-weight of 500, while the span has a font size of 12px and color of #989898. The i has a color of #999.

Here are some possible future work to improve the GRE Vocab Dictionary:

1. Implement a server-side backend to handle the search and retrieval of word definitions, synonyms, and examples. This can provide better performance and scalability for handling a large number of requests.
2. Add more features such as the ability to save and bookmark words, keep a record of recently searched words, and support multiple languages.
3. Improve the user interface and design to make it more appealing and user-friendly. This can include using better color schemes, typography, and layout, as well as adding animations and visual cues to make the interaction more intuitive.
4. Implement auto-suggestions while the user is typing to reduce the number of keystrokes required to search for a word.
5. Add a feature to display the pronunciation of the word and enable the user to listen to the pronunciation.
6. Implement an option to switch between the British and American English spelling and pronunciation.
7. Implement a feature to display the origin of the word and its etymology.
8. Implement a feature to translate the word and its definition into different languages.
9. Implement a feature to suggest similar or related words that might be of interest to the user.
10. Improve the accuracy and completeness of the word definitions, synonyms, and examples by using a more comprehensive and reliable dictionary source.
11. Implement a feedback system to enable users to report errors or suggest improvements to the dictionary.
12. Improve the accessibility of the application by implementing features such as keyboard navigation and support for screen readers.
13. Optimize the application for different screen sizes and devices by using responsive design techniques.
14. Improve the performance and loading time of the application by optimizing the code and reducing the number of requests to external resources.
15. Implement a feature to share the word and its definition on social media platforms.

10 | REFERENCES

Web Reference:

1. https://developers.google.com/youtube/v3
2. https://scrimba.com/g/glearnreact
3. https://www.techomoro.com/how-to-install-and-setup-a-react-app -on-windows-10/
4. https://scrimba.com/g/gintrotojavascript
5. https://reactjs.org/docs/faq-internals.html
6. https://developers.google.com/apis-explorer