```
// src/components/GoogleTranslate.jsx
import React, { useEffect } from 'react';

const GoogleTranslate = () => {
  useEffect(() => {
    // Add the script only if it's not already loaded
    if (!document.getElementById('google-translate-script')) {
      const script = document.createElement('script');
      script.id = 'google-translate-script';
      script.src =
'//translate.google.com/translate_a/element.js?cb=googleTranslateElementInit';
      document.body.appendChild(script);
    }

    // Define init function
    window.googleTranslateElementInit = () => {
      new window.google.translate.TranslateElement(
        { pageLanguage: 'en' },
        'google_translate_element'
      );
    };
  }, []);

  return <div id="google_translate_element"></div>;
};

export default GoogleTranslate;
```

Here's a **clear, technical + interview-ready explanation** of how multi-language support is implemented using Google Translate in your React app:

Feature Overview

"Added multi-language support to automatically translate the entire website for users across different regional languages."

This implementation uses **Google Translate's Website Translator widget** to dynamically translate all visible content on the page without rewriting or duplicating UI text.

How It Works (High-Level)

1. The website's **default language is English**
2. A Google Translate script is **loaded dynamically**

3. Google scans the DOM and **translates all visible content**
4. Users can select **any supported regional language**
5. Translation happens **client-side**, instantly

Component: GoogleTranslate.jsx

1. Dynamic Script Injection (Performance-Friendly)

```
useEffect(() => {
  if (!document.getElementById('google-translate-script')) {
    const script = document.createElement('script');
    script.id = 'google-translate-script';
    script.src = '//translate.google.com/translate_a/element.js?cb=googleTranslateElementInit';
    document.body.appendChild(script);
  }
}, []);
```

Why this is important:

- Prevents loading the script multiple times
- Avoids performance issues during re-renders
- Ensures translation loads only once globally

---

2. Global Initialization Callback

```
window.googleTranslateElementInit = () => {
  new window.google.translate.TranslateElement(
    { pageLanguage: 'en' },
    'google_translate_element'
  );
};
```

What happens here:

- Google calls this function once the script loads
- Initializes the translation engine
- Tells Google:
    - Default language = English

    - Render language selector inside google_translate_element

---

3. Rendering the Translator UI

```
return <div id="google_translate_element"></div>;
```

- This <div> becomes the **language selector dropdown**
- Users choose their preferred language
- Entire website content updates automatically

Why This Translates the *Entire* Website

Google Translate:

- Walks through the **DOM**
- Translates:
    - Headings
    - Buttons
    - Forms
    - Paragraphs
    - Labels
- Works across:
    - React components
    - Dynamic content
    - Route changes

No manual i18n mapping is required.

Advantages of This Approach

✅ **Fast Implementation**

- No need to rewrite content using i18n libraries
- Works instantly across all pages

✅ **Wide Language Coverage**

- Supports 100+ languages
- Ideal for **regional Indian languages** (Hindi, Marathi, Tamil, etc.)

✅ **Client-Side Translation**

- No backend changes required
- No translation API costs

✅ **Low Maintenance**

- New content is translated automatically
- No need to manage language files

Limitations (Good to Mention in Interviews)

- Machine translation may not be **legally precise**
- URLs and metadata are not translated
- Not suitable for **SEO-based multilingual sites**
- Styling of the Google dropdown is limited

Best Placement in App

Typically placed in:

- Navbar
- Footer
- Settings / Language switcher

Example:

<Navbar />

<GoogleTranslate />

<Routes />

Resume / Interview One-Liner

Implemented multi-language support using Google Translate's Website Translator to dynamically localize the entire React application for users across multiple regional languages.

If You Want to Improve It Further 🚀

- Detect browser language automatically
- Store language preference in localStorage
- Replace with react-i18next for legal-grade translations

If you want, I can also help you **defend this design choice in interviews** or **compare it with i18n-based solutions**.