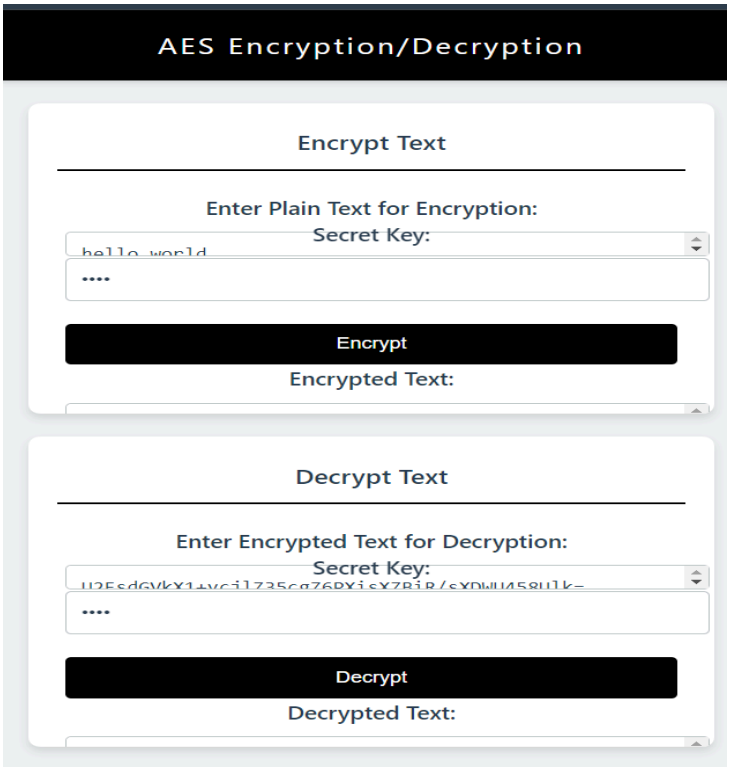
Name: Kaushik Kotian Roll no.:29 Div:D20B Batch:B

ROSP REPORT - A

1. **Repository-1:**https://github.com/hitakritgoplani/security-mini-projects/commit/42be1778d80de691313b69ad66ed63a468064ba9
   Merged Pull Request:https://github.com/hitakritgoplani/security-mini-projects/pull/2

   Issue: Solve the Responsiveness Issue in the Encryption/Decryption Project
   https://github.com/hitakritgoplani/security-mini-projects/issues/1

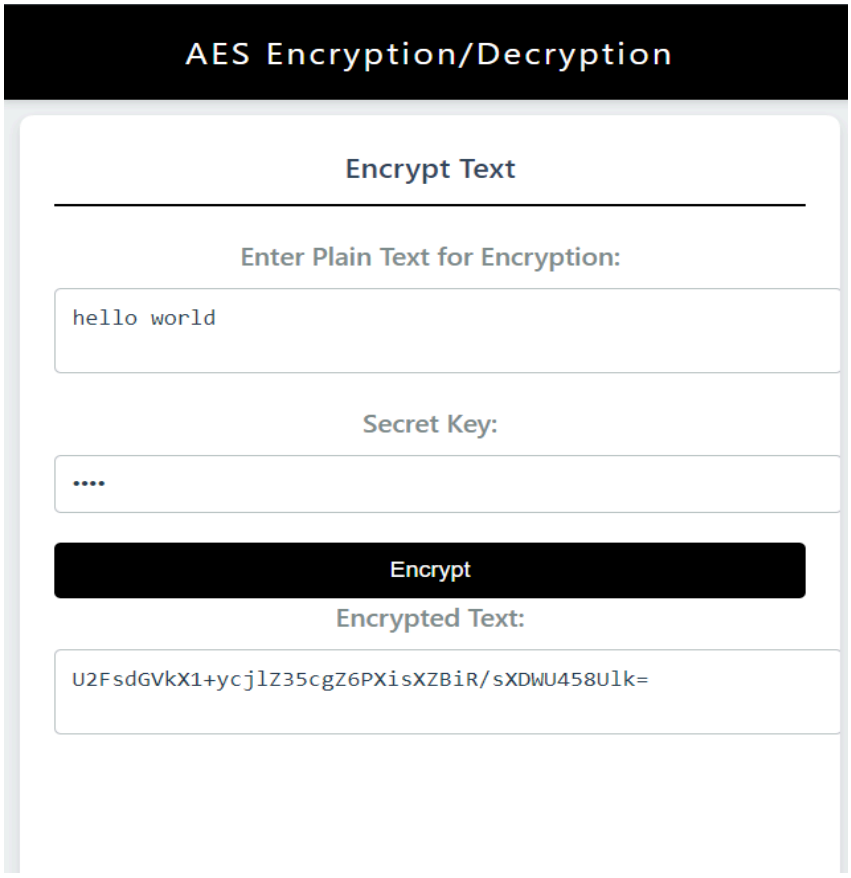
   In the original encryption/decryption project, the layout faced responsiveness issues, particularly on devices with smaller screen sizes like mobile phones and tablets. The content sections (both `.left` and `.right` panels) were designed to sit side by side, which led to horizontal overflow and made it difficult for users to scroll or view the full content on smaller screens.Another problem was the fixed height of the `.home-root` container, which did not adapt well to dynamic screen heights on mobile devices, leading to content cutting off without the ability to scroll through the page.



Solution:

```
@media (max-width: 768px) {
```

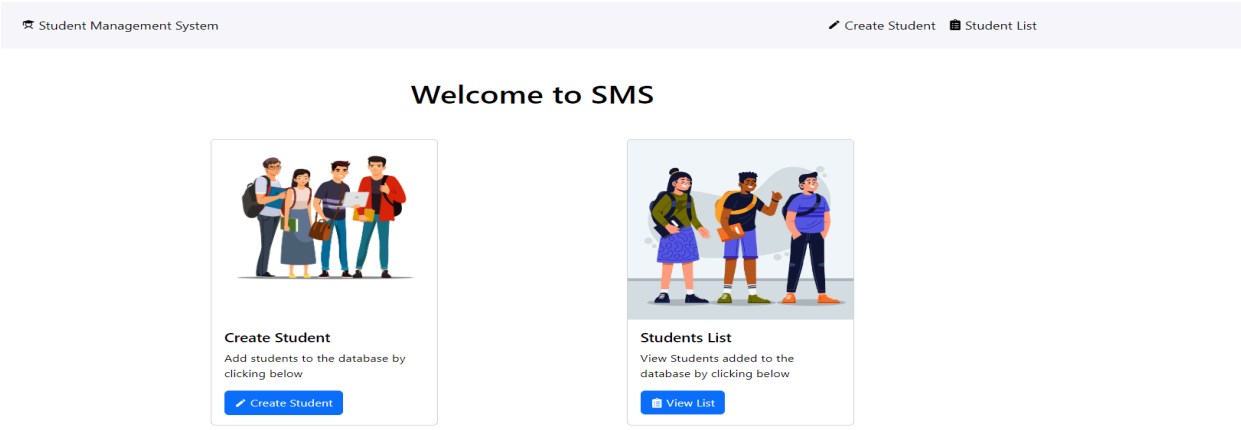
```
  .content-root {
    flex-direction: column;
    overflow-y: scroll;
  }
  header {
    font-size: 20px;
    padding: 15px;
  }

  h3 {
    font-size: 18px;
  }
  .left, .right {
    padding: 20px;
  }

  .home-root {
      height: 200dvh;
  }
}
```



2. **Repository-2:**https://github.com/Sahil-Madhyan/student-management-system/commit/ba9fa7563c2d97a825e1e59c4935c9f5badb5a31
   Merged Pull Request: https://github.com/Sahil-Madhyan/student-management-system/pull/2

   Issue: Light and Dark Theme UI
   https://github.com/Sahil-Madhyan/student-management-system/issues/1

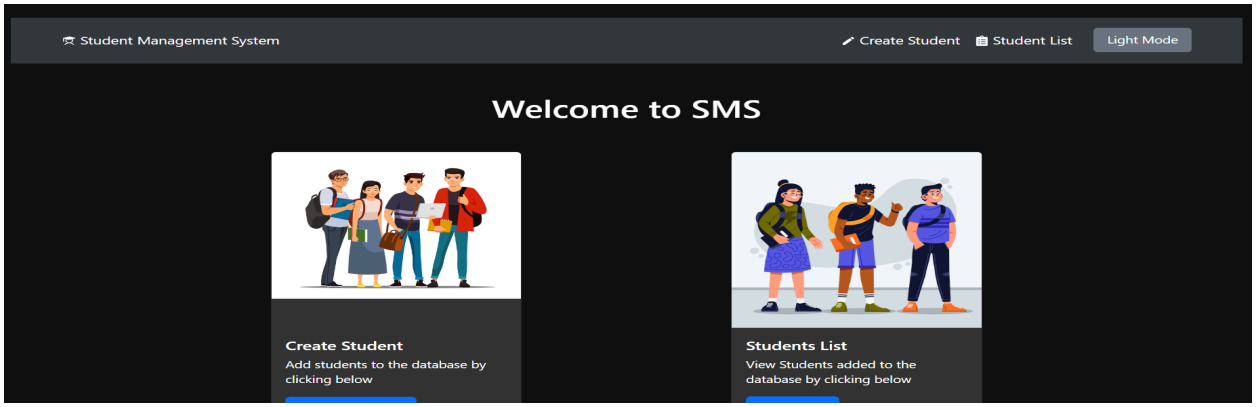
   In the development of a Student Management System (SMS), the need arose to implement a dark mode and light mode toggle feature across multiple components, including the `Navbar` and `Home` pages. The primary challenge was ensuring a seamless transition between the two modes while maintaining a consistent user experience. The system required both visual and functional elements to adapt dynamically when toggling between the themes. Additionally, the user preference for dark or light mode had to persist across page reloads, ensuring that the system remembers the chosen theme. This problem extended across different elements like the background colors, text colors, buttons, and cards, where each component required specific styling adjustments based on the current mode. Managing this at scale across components was complex, especially when dealing with inline styling for dark and light modes, and ensuring that the styling logic didn't interfere with the responsiveness or functionality of the page.



   Solution:

   The solution involved creating a centralized state to manage the dark and light mode functionality, leveraging React's `useState` and `useEffect` hooks. A `darkMode` state was introduced in the `Home` component, which controlled the current theme. A `toggleDarkMode` function allowed users to switch between the two themes dynamically. To ensure that the system remembered the user's choice, the `localStorage` API was used. The selected mode was stored in `localStorage` and was retrieved during the initial load, ensuring the theme

persisted even after a page refresh. Inline CSS styles were used to dynamically adjust the background colors, text colors, and other UI elements based on the `darkMode` state. This method allowed the system to apply different styles when the mode was toggled. The `Navbar` component received the `darkMode` state and the `toggleDarkMode` function as props, ensuring both components stayed in sync with the mode selection.

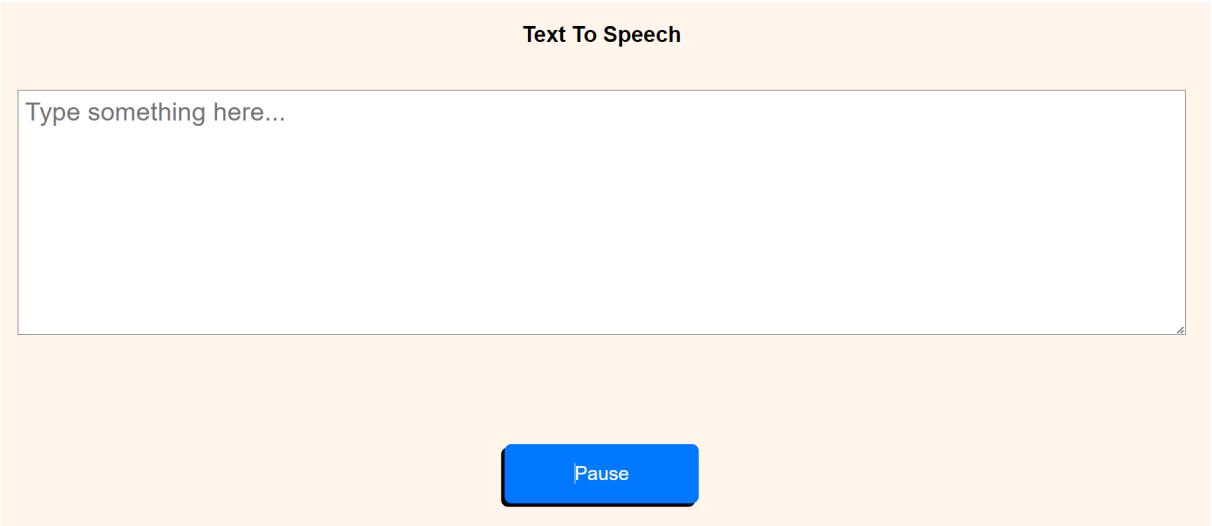


3. **Repository-3:**https://github.com/hitakritgoplani/text-to-speech-js

Merged Pull Request: https://github.com/hitakritgoplani/text-to-speech-js/pull/4

Issue: State Management Button
https://github.com/hitakritgoplani/text-to-speech-js/issues/1

The issue encountered to the functionality of a text-to-speech feature. The user interface allows the user to enter text in a text field and click a "Play" button to initiate speech synthesis. The core problem arises when the "Play" button is clicked without any text entered into the input field. When this happens, the button incorrectly changes its label to "Pause," even though no speech is playing due to the absence of text. This leads to confusion for users, as the button state is misleading. Additionally, there is no indication provided to users that they need to enter text before the speech can be initiated.Another aspect of the issue is the lack of feedback to the user. If a user attempts to play the text while the input field is empty, the interface does not inform the user about the problem. This can degrade user experience, as they may not understand why the application is not responding as expected.



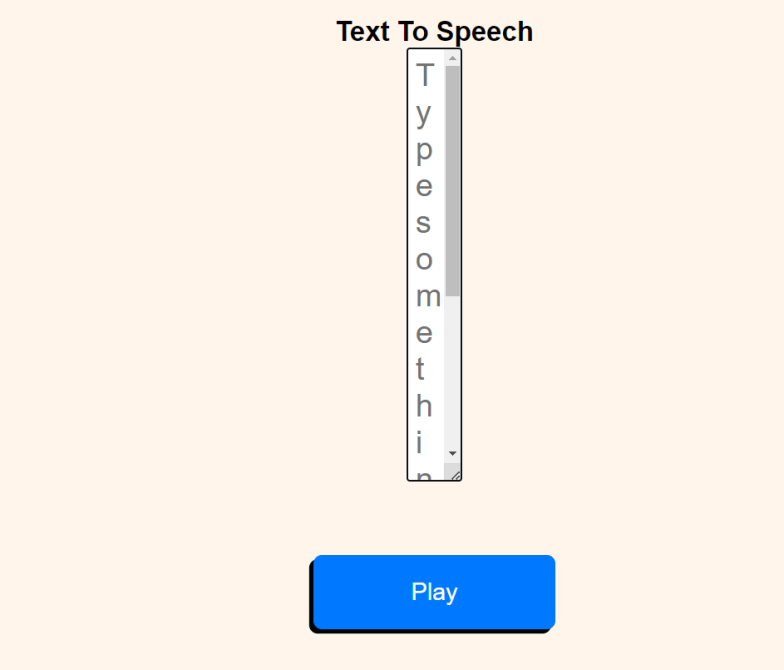
Solution:

Check for Empty Text Field: The solution ensures that before any action is taken, the application checks if the text input field contains any valid text. This is achieved by implementing a condition in the `toggleSpeaking` function to verify whether the `text` variable is empty or consists only of whitespace characters. If the text field is empty, the play button does not proceed to change its label or activate the speech synthesis. This ensures the button's state reflects its actual functionality, preventing it from changing to "Pause" unless text is provided.

User Feedback with `react-toastify`: To improve user experience, the `react-toastify` library was integrated into the application to provide feedback. If the user tries to click the "Play" button without entering any text, a toast notification is triggered, displaying a message like "No text available! Please enter text to play." This real-time feedback helps users understand that they need to enter some text in the input field before using the text-to-speech functionality. The toast notification is non-intrusive and quickly informs the user of the issue.

4. **Repository-4:**https://github.com/hitakritgoplani/text-to-speech-js
   Merged Pull Request: https://github.com/hitakritgoplani/text-to-speech-js/pull/3

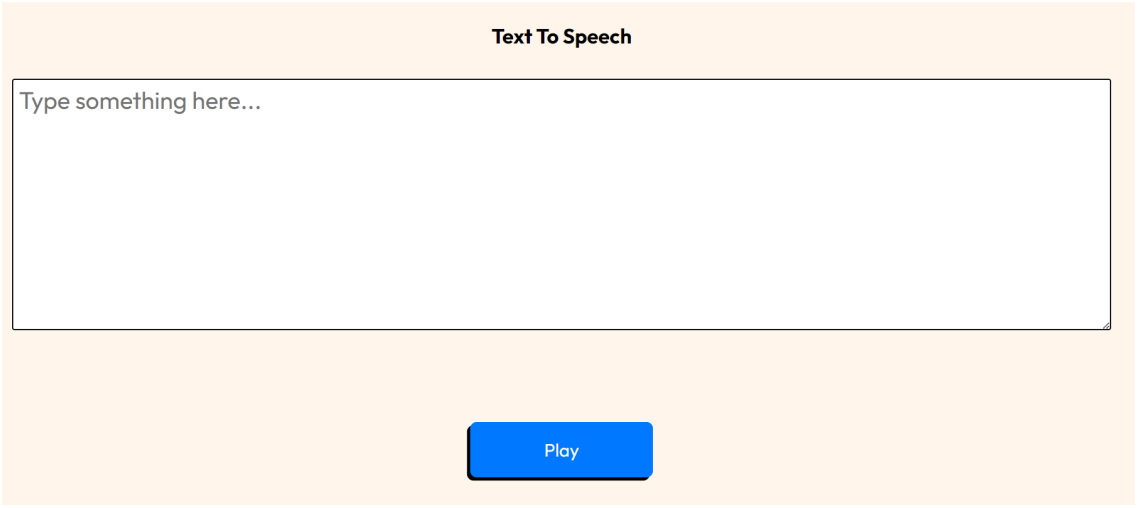
Issue: Resize textfield
https://github.com/hitakritgoplani/text-to-speech-js/issues/2
The issue you're encountering involves the behavior of the text box in your React application. Specifically, when users input text, the text box is resizing horizontally from right to left, which disrupts the UI. Horizontal resizing causes misalignment and makes the interface less intuitive. This occurs because, by default, the browser allows both vertical and horizontal resizing for text areas, meaning users can stretch or shrink the text box in both dimensions. In this case, the unwanted horizontal resizing is affecting the visual structure of the UI, causing elements to shift or overlap.



Solution:
`resize:vertical;`, which restricts the text box to only vertical resizing. This ensures that users can adjust the height of the text box, accommodating larger inputs, but prevents any changes to its width, preserving the overall layout and UI structure. This solution balances usability with visual consistency, maintaining a clean and stable interface while still allowing user control over text input length.

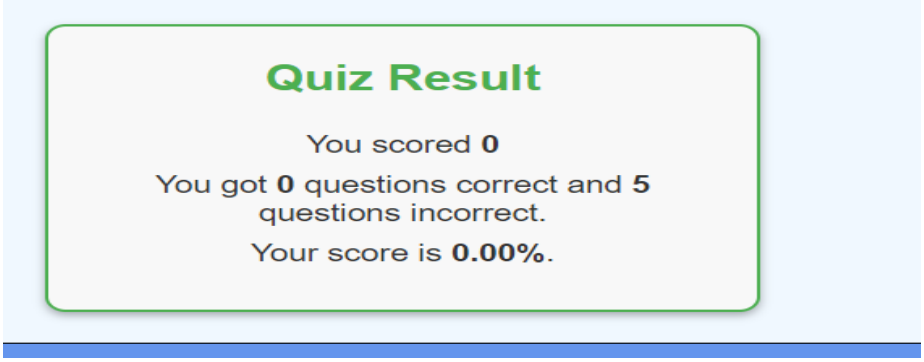
```
.text-field{
    background-color: white;
```

```
width: 90%;
height: 70%;
font-size: 2vw;
font-weight: 400;
padding: 0.5vw;
resize:vertical;  }
```



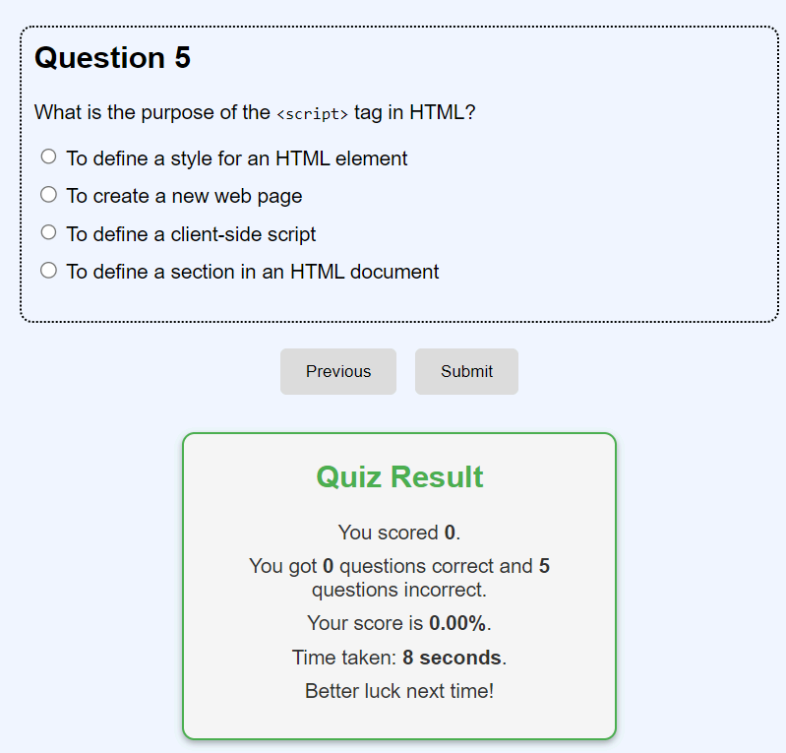
5. **Repository-4:**https://github.com/Sahil-Madhyan/quiz-web-app/commit/a08cd8ea4fdadbb97f03cf3397d570ff7b91b2b4
   Merged Pull Request:https://github.com/Sahil-Madhyan/quiz-web-app/pull/2

   Issue: Display time taken by user to complete test
   https://github.com/Sahil-Madhyan/quiz-web-app/issues/1
   In the current context, the quiz lacks functionality to measure the time taken by the user from the beginning of the quiz (when they start answering the first question) to the submission (when they complete the final question). This issue not only affects the completeness of the quiz application but can also skew the analysis of user performance, as timing plays an important role in determining efficiency.



Solution:

To resolve this issue, a straightforward solution involves implementing a timer in JavaScript. The concept is to record the exact time when the quiz starts and then calculate the elapsed time when the user submits the quiz. JavaScript's `Date.now()` method can be employed to capture the start time of the quiz. When the user submits the quiz, the difference between the current time and the start time can be calculated to determine the total time taken, typically in seconds or minutes. This calculated time is then displayed alongside the quiz results to provide clear feedback.



6. Repository-6:

https://github.com/hitakritgoplani/random-quote-generator/commit/9faf5b03b3b3abe2f2c5d44a4593a2e3ba0a30a8

Merge Pull Request: https://github.com/hitakritgoplani/random-quote-generator/pull/2

Issue:Change the author name while loading a new quote

https://github.com/hitakritgoplani/random-quote-generator/issues/1

When a new quote is being fetched, change the name of an author. Currently the previous author's name is displayed

Solution:

Before adding `setAuthor("")`, when a new quote was being retrieved, the previous author's name would remain visible until the new data was loaded. This could confuse

users by displaying the old author's name with a placeholder like "Getting a quote" for the new quote. By explicitly clearing the author's name with `setAuthor("");` at the start of the `getQuote()` function, you ensure that when a new quote is being fetched, the author's name is temporarily blank. This way, it becomes clear to users that both the quote and the author's name are being updated, preventing any confusion caused by mismatched data on the screen. Once the new data is successfully fetched.