# GAMIFIED TEXT ENCRYPTION USING AES ALGORITHM

Rohan Ghoshal 21BCE5775

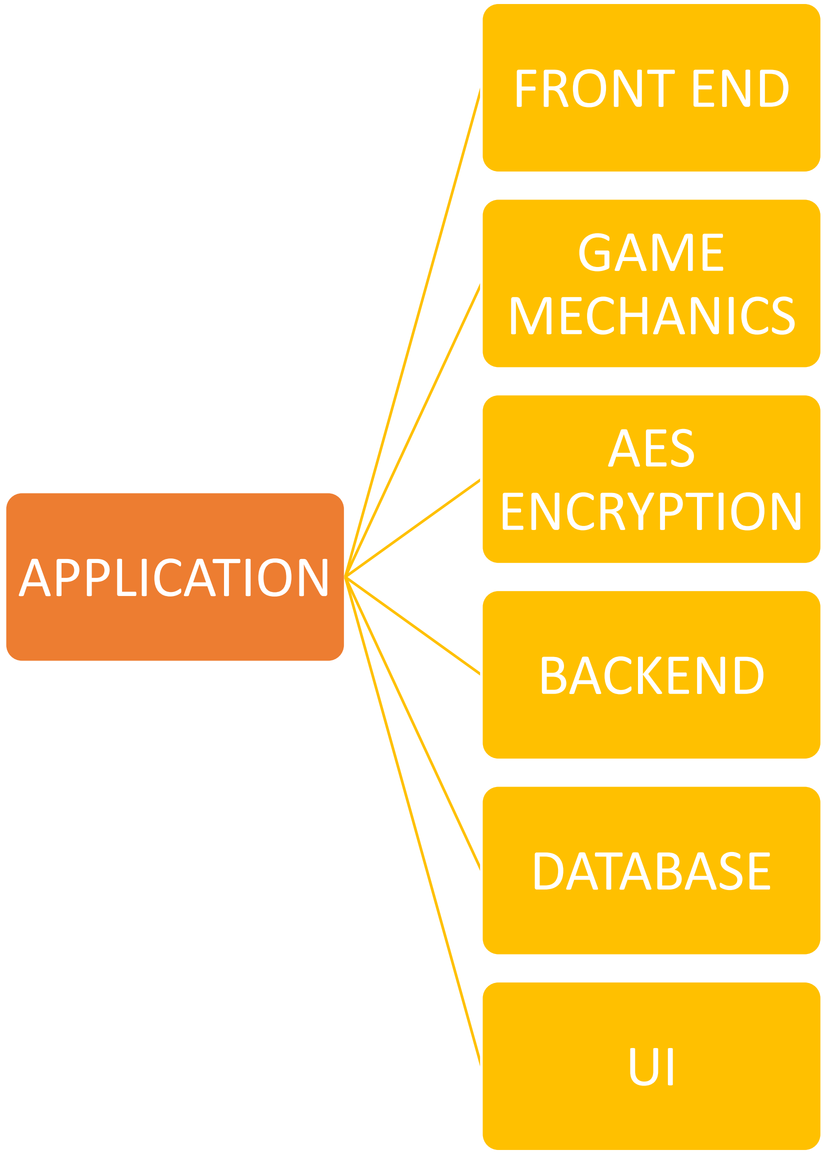
Harsh Patil 21BCE1030

The solution is a Gamified Text Encryption application that aims to revolutionize the traditional text encryption process by incorporating gamification elements and utilizing the AES (Advanced Encryption Standard) algorithm. The system offers an innovative and engaging approach to encrypting text messages and files, making it more appealing and user-friendly.

In the proposed system, users are presented with interactive word games or challenges that they need to solve in order to encrypt their desired message. The encryption key is derived from the solution to the game, adding an extra layer of security to the encryption process. By integrating gamification elements, the system motivates users to actively participate in the encryption process and promotes a deeper understanding of encryption techniques.

Furthermore, the application extends its functionality to file encryption and decryption. Users can leverage the generated encryption key to encrypt their files securely, ensuring their confidentiality and protection. Similarly, the application allows users to decrypt password-protected files using the corresponding encryption key, providing authorized access to the original file content.

The integration of the AES algorithm guarantees strong encryption for both text messages and files, reinforcing the security of the entire system. By combining gamification, text encryption, and file encryption, the proposed system offers an exciting and comprehensive solution to protect sensitive information and promote secure encryption practices.



The frontend of the application is built using JavaScript and React, a popular frontend framework.

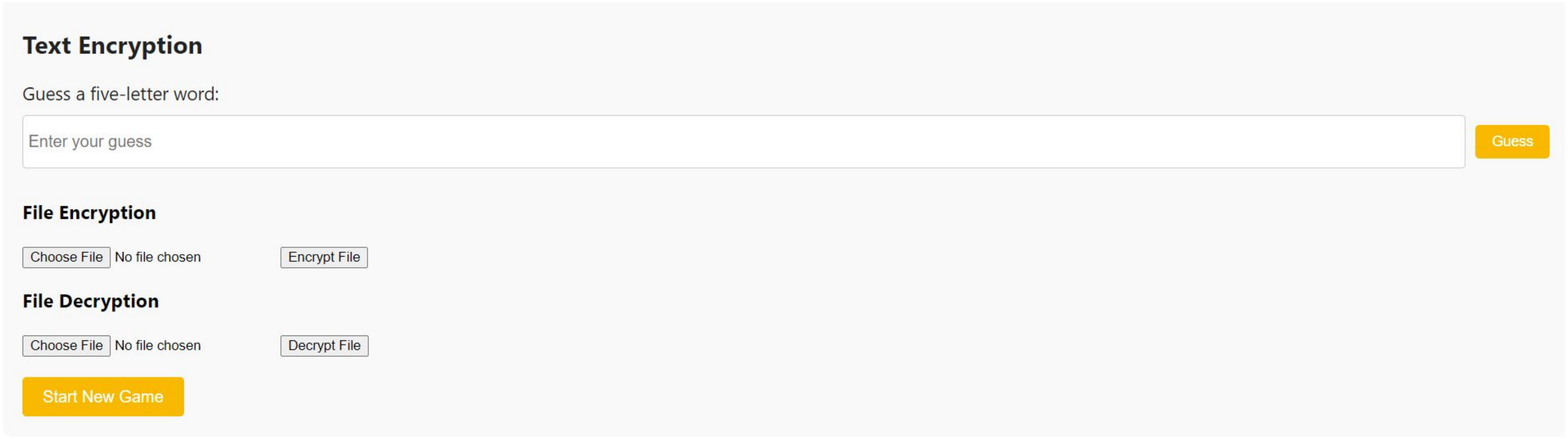
The word guessing game is implemented, allowing users to guess a five-letter word.

The AES encryption module is integrated into the architecture to ensure the secure encryption of user messages. The CryptoJS library is utilized to incorporate AES encryption and decryption functions.

The backend receives user input from the frontend, performs encryption or decryption operations, and communicates the results back to the frontend.

The database component stores user-related data, including encrypted messages, encryption keys, and user profiles.

User interface design is a critical aspect of the architecture, focusing on creating an intuitive and visually appealing interface. CSS styling was applied to create an appealing and professional user interface.



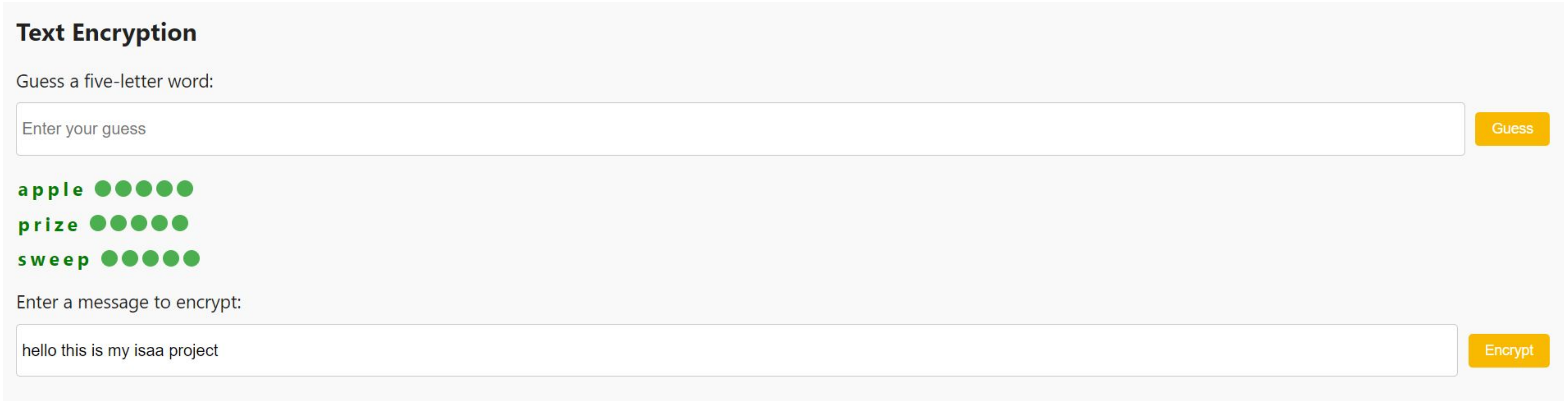
Text Encryption

Guess a five-letter word:

Enter your guess

File Encryption

File Decryption



Text Encryption

Guess a five-letter word:

Enter your guess

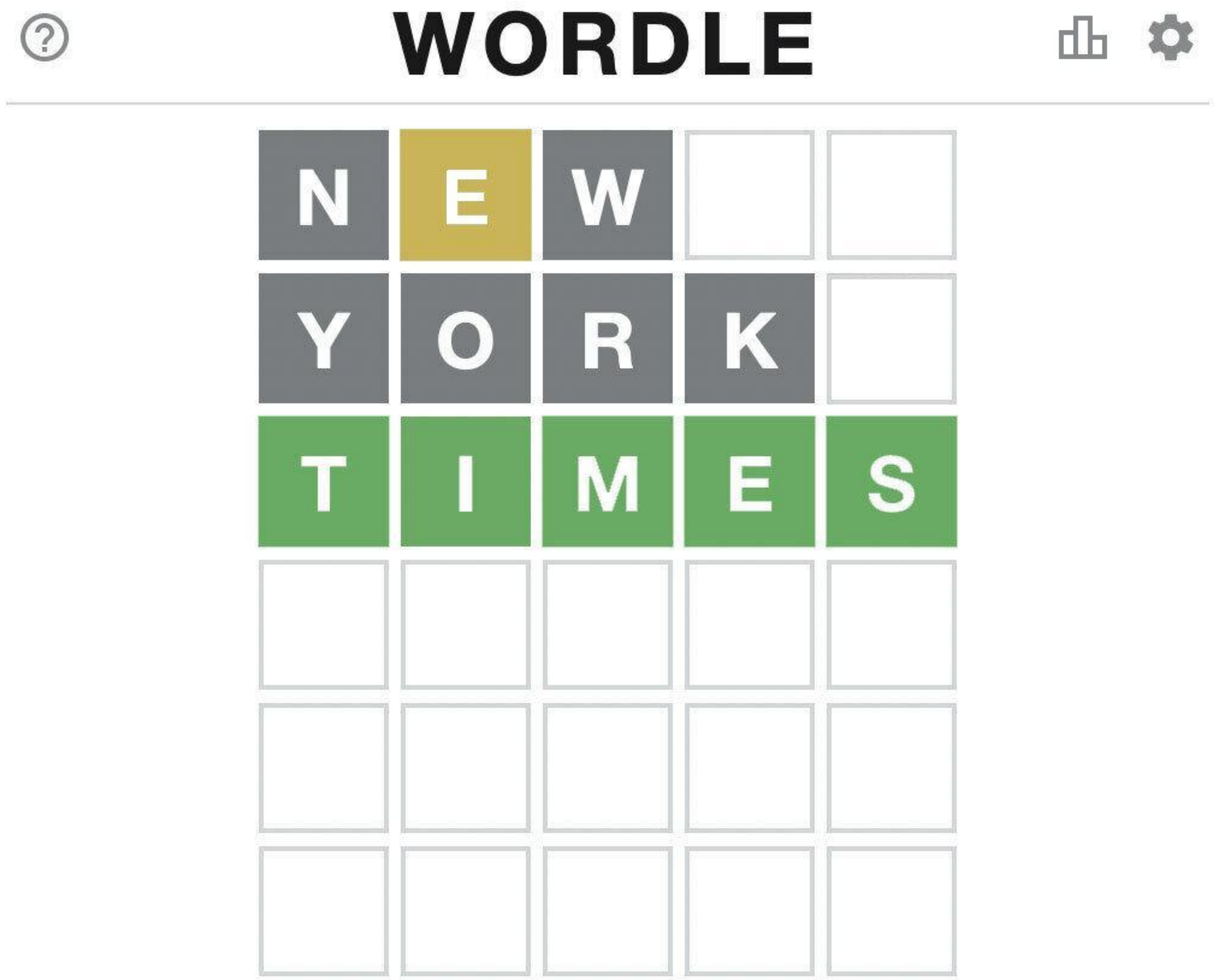
apple ●●●●●

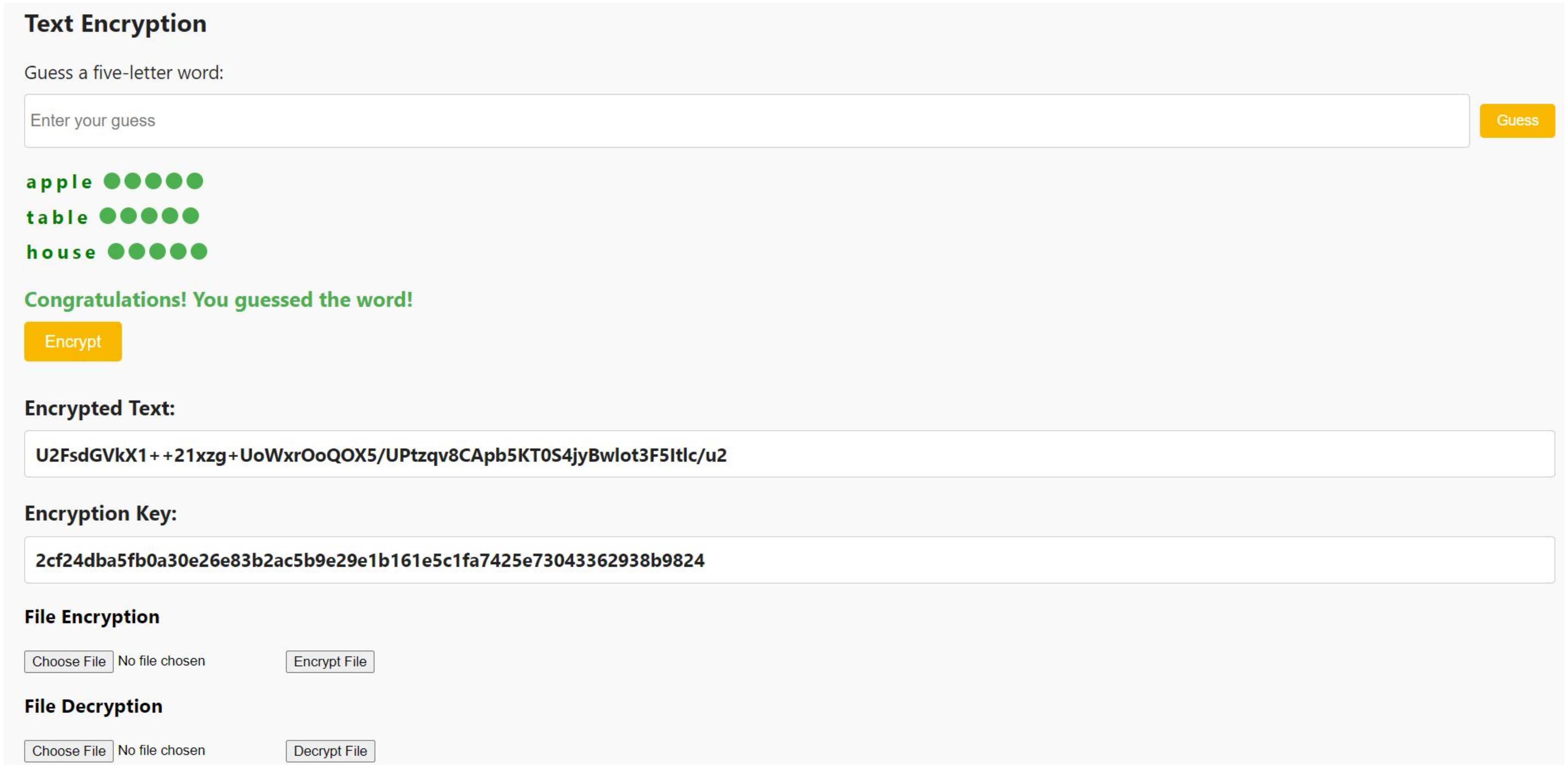
prize ●●●●●

sweep ●●●●●

Enter a message to encrypt:

hello this is my isaa project


WORDLE
N E W
Y O R K
T I M E S



## Text Encryption

Guess a five-letter word:

Enter your guess

apple ●●●●●

table ●●●●●

house ●●●●●

Congratulations! You guessed the word!

**Encrypted Text:**

U2FsdGVkX1++21xzg+UoWxrOoQOX5/UPtzqv8CApb5KT0S4jyBwlot3F5Itlc/u2

**Encryption Key:**

2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

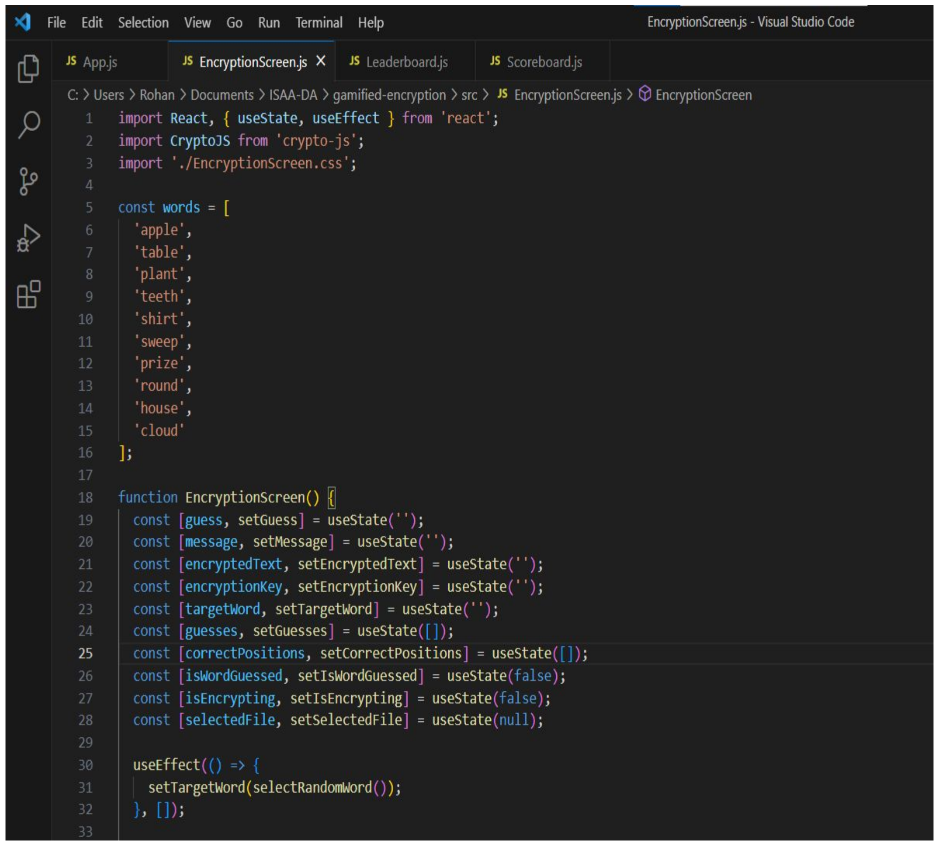
**File Encryption**

**File Decryption**

1. Import Statements:
   - The code imports the necessary modules, including React, `useState`, `useEffect` hooks for managing component state, and `CryptoJS` for text encryption using AES.

2. Initial State:
   - Several state variables are declared using the `useState` hook to manage different aspects of the game, such as the user's guess, the target word, the user's message, encrypted text, encryption key, and various game-related flags.

3. useEffect Hook:
   - The `useEffect` hook is used to run code when the component mounts (initializes). In this case, it sets the initial target word by calling the `selectRandomWord` function.


File Edit Selection View Go Run Terminal Help
EncryptionScreen.js - Visual Studio Code
App.js
EncryptionScreen.js
Leaderboard.js
Scoreboard.js
C: > Users > Rohan > Documents > ISAA-DA > gamified-encryption > src > EncryptionScreen.js > EncryptionScreen

```javascript
import React, { useState, useEffect } from 'react';
import CryptoJS from 'crypto-js';
import './EncryptionScreen.css';

const words = [
  'apple',
  'table',
  'plant',
  'teeth',
  'shirt',
  'sweep',
  'prize',
  'round',
  'house',
  'cloud'
];

function EncryptionScreen() {
  const [guess, setGuess] = useState('');
  const [message, setMessage] = useState('');
  const [encryptedText, setEncryptedText] = useState('');
  const [encryptionKey, setEncryptionKey] = useState('');
  const [targetWord, setTargetWord] = useState('');
  const [guesses, setGuesses] = useState([]);
  const [correctPositions, setCorrectPositions] = useState([]);
  const [isWordGuessed, setIsWordGuessed] = useState(false);
  const [isEncrypting, setIsEncrypting] = useState(false);
  const [selectedFile, setSelectedFile] = useState(null);

  useEffect(() => {
    setTargetWord(selectRandomWord());
  }, []);

```

4. Helper Functions:
   - `selectRandomWord`: Picks a random word from the `words` array.
   - `handleGuess`: Handles user guesses, checks if the guess matches the target word, and sets the state accordingly.
   - `handleTextChange`: Updates the guess or message state based on user input and handles encryption related to the guess.
   - `combineText`: Combines all the user's guesses and the message into a single string for encryption.
   - `encryptText`: Encrypts a given text using AES and returns the encrypted text and encryption key.
   - `generateEncryptionKey`: Generates an encryption key based on the message for text encryption.
   - `handleNewGame`: Resets the game state for starting a new game.
   - `handleFileSelection`: Handles user selection of a file for encryption/decryption.
   - `handleFileEncryption`: Encrypts the selected file using AES and allows the user to download the encrypted file.
   - `handleFileDecryption`: Decrypts the selected file using AES and allows the user to download the decrypted file.
   - `decryptText`: Decrypts the given encrypted data using the encryption key.
   - `getLetterColor`: Determines the color of each letter in the guessed word based on correct and incorrect positions.

5. Render Method:

- The `return` statement includes the JSX code that represents the user interface for the game.
- It displays an input field for the user's guess, a history of previous guesses with feedback on correct positions, and buttons to start a new game and encrypt/decrypt files.
- Depending on the game state, it shows different sections for word guessing and text encryption.
- After encryption, it displays the encrypted text and the encryption key.