**CORE JAVASCRIPT PROJECT REPORT ON**

**RANDOM PASSWORD GENERATOR**

NAME:SATYAM KUMAR SINGH DATE:12-03-2023

COLLEGE:NIT PATNA SUPERVISOR:MISS AISHWARYA

**INTRODUCTION**

A random password generator is a program or script that generates a password consisting of a random combination of characters. This is commonly used for security purposes to create strong passwords that are difficult to guess or crack.

In JavaScript, a random password generator can be created by using built-in functions that generate random numbers and letters.

According to a report by TechBeacon, the use of a strong password generator is a crucial step towards enhancing security in online environments. The report emphasizes the need for a strong password generator that can generate passwords with a mix of upper and lower case letters, numbers, and special characters. It also notes that a password generator must be capable of generating a unique password for each user.

In another report by SitePoint, the importance of using a random password generator to protect against brute force attacks is highlighted. The report notes that a password generator can generate long and complex passwords that are difficult for hackers to guess. It also recommends using a password manager to securely store and manage passwords.

**OBJECTIVE**

The main objectives of a random password generator are to:

1. Enhance security: A random password generator helps to enhance security by generating strong, complex passwords that are difficult to guess or crack. This makes it more difficult for hackers to gain unauthorized access to user accounts.
2. Ensure uniqueness: A random password generator ensures that each user is assigned a unique password. This is important to prevent attackers from using the same password to gain access to multiple user accounts.
3. Promote password hygiene: A random password generator promotes good password hygiene by encouraging users to use strong and unique passwords. This reduces the risk of password-related attacks such as brute force attacks and password guessing.
4. Save time and effort: A random password generator saves time and effort by automatically generating passwords for users. This reduces the need for users to create passwords themselves, which can be time-consuming and lead to the use of weak or easily guessable passwords.
5. Support compliance: Many regulations and standards such as HIPAA and PCI DSS require the use of strong passwords. A random password generator can help organizations comply with these requirements by ensuring that all passwords meet the necessary standards.

In summary, the main objectives of a random password generator are to enhance security, ensure uniqueness, promote good password hygiene, save time and effort, and support compliance with regulations and standards.

**BACKGROUND**

A random password generator is a tool that creates unique, complex passwords that are difficult for hackers to crack. It uses a set of characters and rules to generate passwords randomly. In recent years, with the rise of cybercrime, random password generators have become increasingly popular as a means of enhancing cybersecurity.

JavaScript is a programming language used for creating interactive web pages. It is widely used for building front-end web applications, and it can also be used to create a random password generator. JavaScript can generate random numbers and select characters from a given set to create a password that meets specific criteria.

When creating a random password generator using JavaScript, several factors should be considered. These include the length of the password, the complexity of the characters used, and the number of passwords generated. The generator should also be user-friendly and easy to use.

To build a random password generator using JavaScript, one can use various techniques, including using built-in JavaScript functions such as Math.random(), concatenation, loops, and conditionals. The generated passwords can be displayed on the screen, copied to the clipboard, or saved to a file.

In conclusion, a random password generator is an essential tool for enhancing cybersecurity, and building one using JavaScript can be an exciting and educational project for students and developers alike.

**HARDWARE AND SOFTWARE REQUIRMENTS**

The hardware and software requirements for a random password generator using JavaScript are relatively minimal. Here are some basic requirements:

Hardware Requirements:

* A computer or laptop with a web browser installed.
* A stable internet connection to access the web application.

Software Requirements:

* A code editor or Integrated Development Environment (IDE) to write the JavaScript code. Examples include Visual Studio Code, Sublime Text, or Atom.
* A web browser to test and run the web application. Examples include Google Chrome, Mozilla Firefox, or Microsoft Edge.

To develop a random password generator using JavaScript, one must have a basic understanding of HTML, CSS, and JavaScript. Familiarity with JavaScript libraries and frameworks such as jQuery or React can also be helpful.

It is essential to ensure that the code is compatible with different web browsers, and it is recommended to test the application on various devices and operating systems to ensure cross-platform compatibility.

In conclusion, the hardware and software requirements for developing a random password generator using JavaScript are minimal. With a computer or laptop, web browser, and code editor, one can easily develop a simple and effective password generator that enhances cybersecurity.

**CODING**

**Using <> index.html file:**

<!DOCTYPE html>

<html lang="en" dir="ltr">

  <head>

    <meta charset="utf-8">

    <title>Password Generator JavaScript | Satyam singh</title>

    <link rel="stylesheet" href="style.css">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- Google Icon Link for Icons -->

    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Symbols+Rounded:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200">

    <script src="script.js" defer></script>

  </head>

  <body>

    <div class="container">

      <h2>Password Generator</h2>

      <div class="wrapper">

        <div class="input-box">

          <input type="text" disabled>

          <span class="material-symbols-rounded">copy\_all</span>

        </div>

        <div class="pass-indicator"></div>

        <div class="pass-length">

          <div class="details">

            <label class="title">Password Length</label>

            <span></span>

          </div>

          <input type="range" min="1" max="30" value="15" step="1">

        </div>

        <div class="pass-settings">

          <label class="title">Password Settings</label>

          <ul class="options">

            <li class="option">

              <input type="checkbox" id="lowercase">

              <label for="lowercase">Lowercase (a-z)</label>

            </li>

            <li class="option">

              <input type="checkbox" id="uppercase">

              <label for="uppercase">Uppercase (A-Z)</label>

            </li>

            <li class="option">

              <input type="checkbox" id="numbers">

              <label for="numbers">Numbers (0-9)</label>

            </li>

            <li class="option">

              <input type="checkbox" id="symbols">

              <label for="symbols">Symbols (!-$^+)</label>

            </li>

            <li class="option">

              <input type="checkbox" id="exc-duplicate">

              <label for="exc-duplicate">Exclude Duplicate</label>

            </li>

            <li class="option">

              <input type="checkbox" id="spaces">

              <label for="spaces">Include Spaces</label>

            </li>

          </ul>

        </div>

        <button class="generate-btn">Generate Password</button>

      </div>

    </div>

  </body>

</html>

**Using JS script.js file:**

const lengthSlider = document.querySelector(".pass-length input"),

options = document.querySelectorAll(".option input"),

copyIcon = document.querySelector(".input-box span"),

passwordInput = document.querySelector(".input-box input"),

passIndicator = document.querySelector(".pass-indicator"),

generateBtn = document.querySelector(".generate-btn");

const characters = { // object of letters, numbers & symbols

    lowercase: "abcdefghijklmnopqrstuvwxyz",

    uppercase: "ABCDEFGHIJKLMNOPQRSTUVWXYZ",

    numbers: "0123456789",

    symbols: "^!$%&|[](){}:;.,\*+-#@<>~"

}

const generatePassword = () => {

    let staticPassword = "",

    randomPassword = "",

    excludeDuplicate = false,

    passLength = lengthSlider.value;

    options.forEach(option => { // looping through each option's checkbox

        if(option.checked) { // if checkbox is checked

            // if checkbox id isn't exc-duplicate && spaces

            if(option.id !== "exc-duplicate" && option.id !== "spaces") {

                // adding particular key value from character object to staticPassword

                staticPassword += characters[option.id];

            } else if(option.id === "spaces") { // if checkbox id is spaces

                staticPassword += `  ${staticPassword}  `; // adding space at the beginning & end of staticPassword

            } else { // else pass true value to excludeDuplicate

                excludeDuplicate = true;

            }

        }

    });

    for (let i = 0; i < passLength; i++) {

        // getting random character from the static password

        let randomChar = staticPassword[Math.floor(Math.random() \* staticPassword.length)];

        if(excludeDuplicate) { // if excludeDuplicate is true

            // if randomPassword doesn't contains the current random character or randomChar is equal

            // to space " " then add random character to randomPassword else decrement i by -1

            !randomPassword.includes(randomChar) || randomChar == " " ? randomPassword += randomChar : i--;

        } else { // else add random character to randomPassword

            randomPassword += randomChar;

        }

    }

    passwordInput.value = randomPassword; // passing randomPassword to passwordInput value

}

const upadatePassIndicator = () => {

    // if lengthSlider value is less than 8 then pass "weak" as passIndicator id else if lengthSlider

    // value is less than 16 then pass "medium" as id else pass "strong" as id

    passIndicator.id = lengthSlider.value <= 8 ? "weak" : lengthSlider.value <= 16 ? "medium" : "strong";

}

const updateSlider = () => {

    // passing slider value as counter text

    document.querySelector(".pass-length span").innerText = lengthSlider.value;

    generatePassword();

    upadatePassIndicator();

}

updateSlider();

const copyPassword = () => {

    navigator.clipboard.writeText(passwordInput.value); // copying random password

    copyIcon.innerText = "check"; // changing copy icon to tick

    copyIcon.style.color = "#4285F4";

    setTimeout(() => { // after 1500 ms, changing tick icon back to copy

        copyIcon.innerText = "copy\_all";

        copyIcon.style.color = "#707070";

    }, 1500);

}

copyIcon.addEventListener("click", copyPassword);

lengthSlider.addEventListener("input", updateSlider);

generateBtn.addEventListener("click", generatePassword);

**Using # style.css:**

/\* Import Google font - Poppins \*/

@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600&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: #4285F4;

}

.container{

  width: 450px;

  background: #fff;

  border-radius: 8px;

  box-shadow: 0 10px 20px rgba(0,0,0,0.05);

}

.container h2{

  font-weight: 600;

  font-size: 1.31rem;

  padding: 1rem 1.75rem;

  border-bottom: 1px solid #d4dbe5;

}

.wrapper{

  margin: 1.25rem 1.75rem;

}

.wrapper .input-box{

  position: relative;

}

.input-box input{

  width: 100%;

  height: 53px;

  color: #000;

  background: none;

  font-size: 1.06rem;

  font-weight: 500;

  border-radius: 4px;

  letter-spacing: 1.4px;

  border: 1px solid #aaa;

  padding: 0 2.85rem 0 1rem;

}

.input-box span{

  position: absolute;

  right: 13px;

  cursor: pointer;

  line-height: 53px;

  color: #707070;

}

.input-box span:hover{

  color: #4285F4!important;

}

.wrapper .pass-indicator{

  width: 100%;

  height: 4px;

  position: relative;

  background: #dfdfdf;

  margin-top: 0.75rem;

  border-radius: 25px;

}

.pass-indicator::before{

  position: absolute;

  content: "";

  height: 100%;

  width: 50%;

  border-radius: inherit;

  transition: width 0.3s ease;

}

.pass-indicator#weak::before{

  width: 20%;

  background: #E64A4A;

}

.pass-indicator#medium::before{

  width: 50%;

  background: #f1c80b;

}

.pass-indicator#strong::before{

  width: 100%;

  background: #4285F4;

}

.wrapper .pass-length{

  margin: 1.56rem 0 1.25rem;

}

.pass-length .details{

  display: flex;

  justify-content: space-between;

}

.pass-length input{

  width: 100%;

  height: 5px;

}

.pass-settings .options{

  display: flex;

  list-style: none;

  flex-wrap: wrap;

  margin-top: 1rem;

}

.pass-settings .options .option{

  display: flex;

  align-items: center;

  margin-bottom: 1rem;

  width: calc(100% / 2);

}

/\* .options .option:first-child{

  pointer-events: none;

} \*/

/\* .options .option:first-child input{

  opacity: 0.7;

} \*/

.options .option input{

  height: 16px;

  width: 16px;

  cursor: pointer;

}

.options .option label{

  cursor: pointer;

  color: #4f4f4f;

  padding-left: 0.63rem;

}

.wrapper .generate-btn{

  width: 100%;

  color: #fff;

  border: none;

  outline: none;

  cursor: pointer;

  background: #4285F4;

  font-size: 1.06rem;

  padding: 0.94rem 0;

  border-radius: 5px;

  text-transform: uppercase;

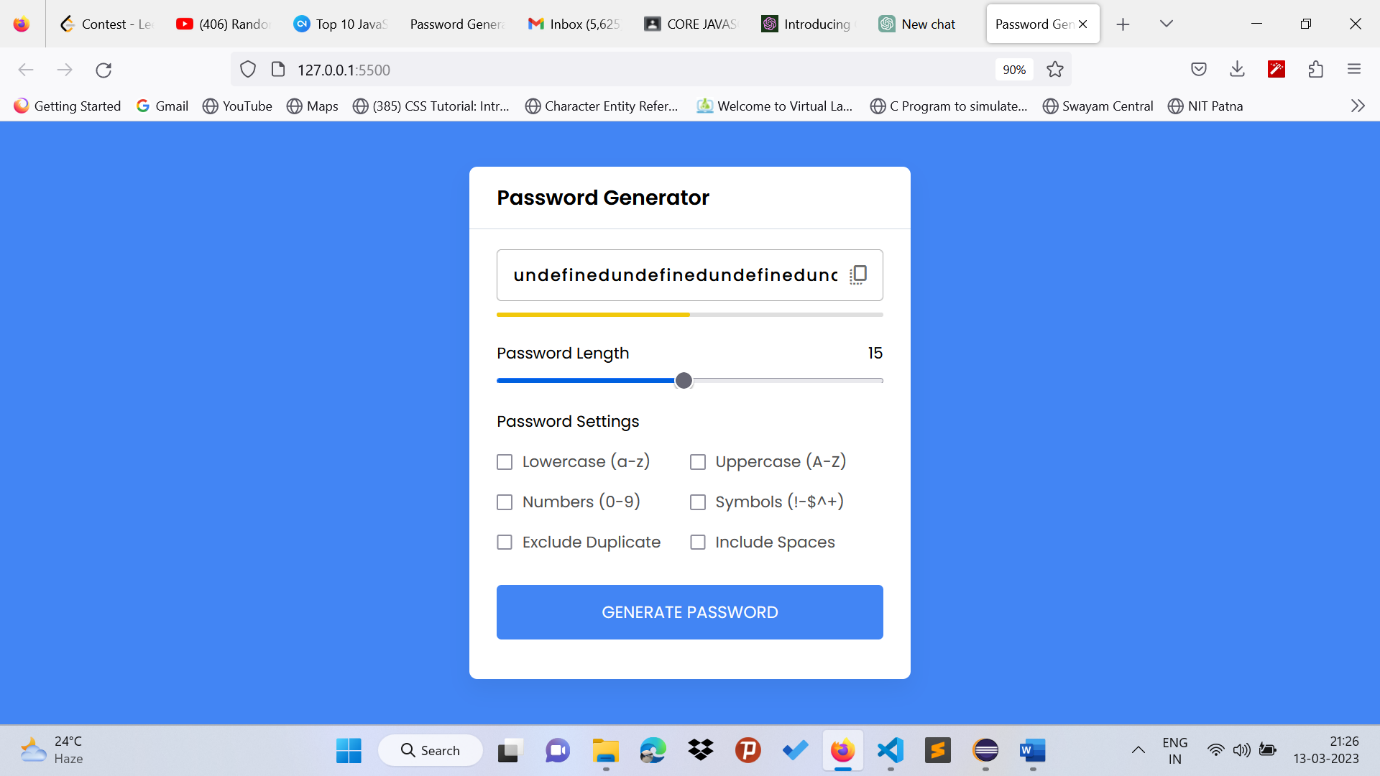
  margin: 0.94rem 0 1.3rem;

}

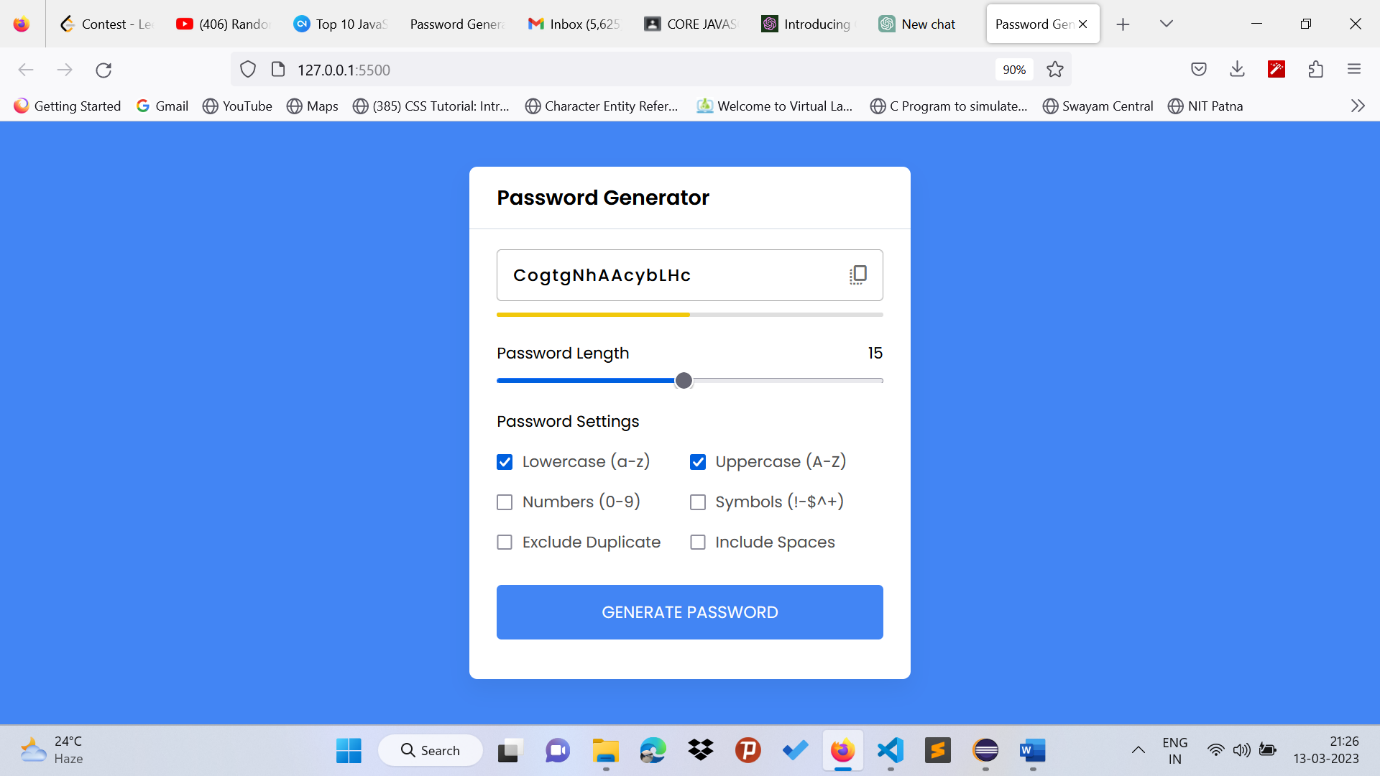
**OUTPUT SCREENSHOT**

**Project-link:** [Tap here](https://imsks135.github.io/project-test/) (https://imsks135.github.io/project-test/)

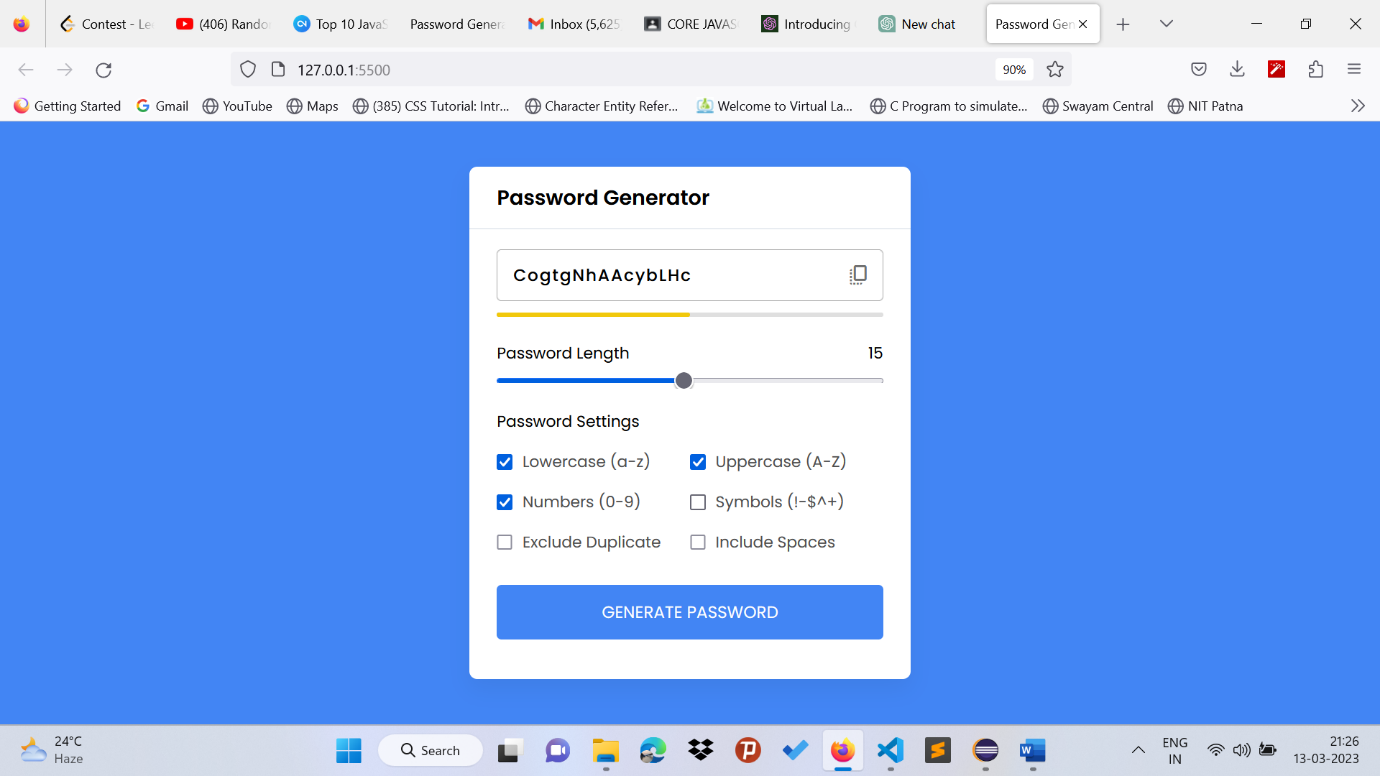
**Undefined output:**

****

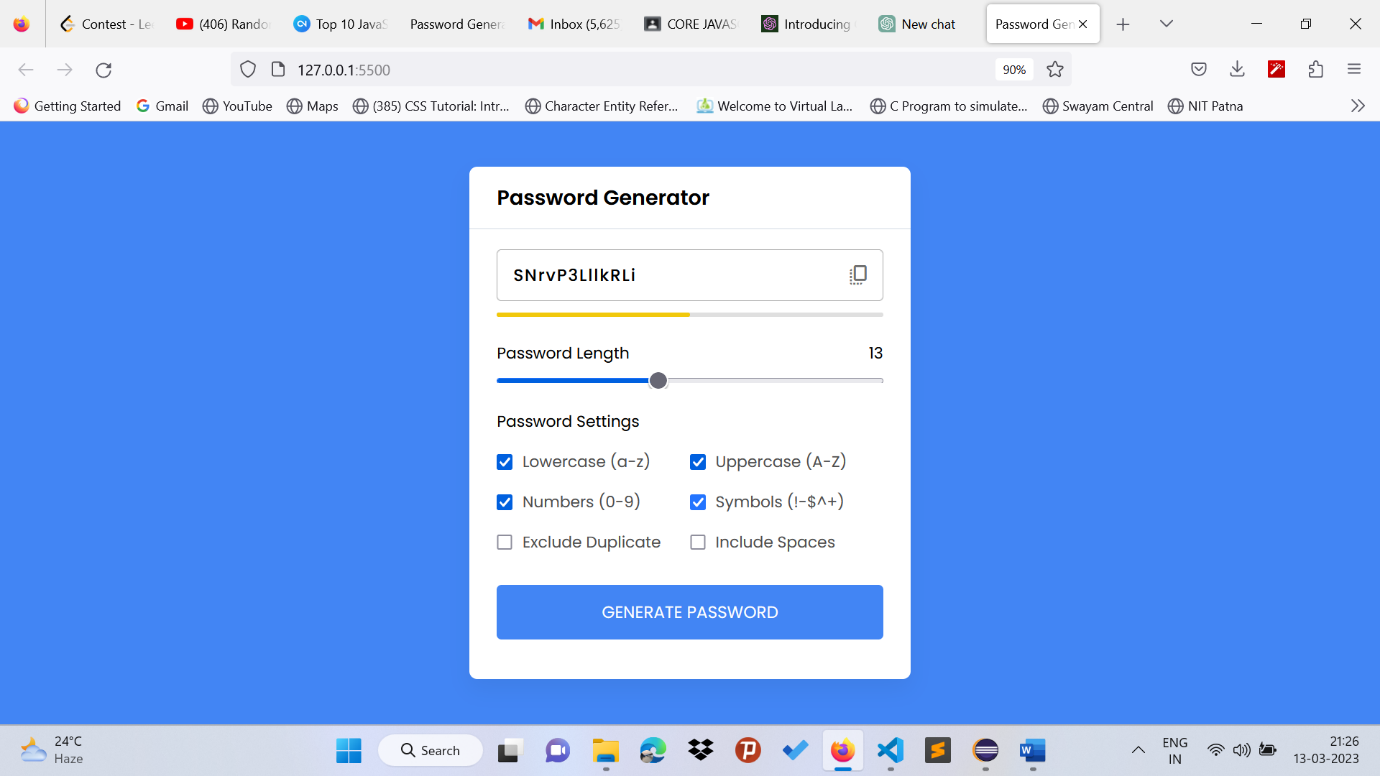
**Using only lowercase(a-z):**

****

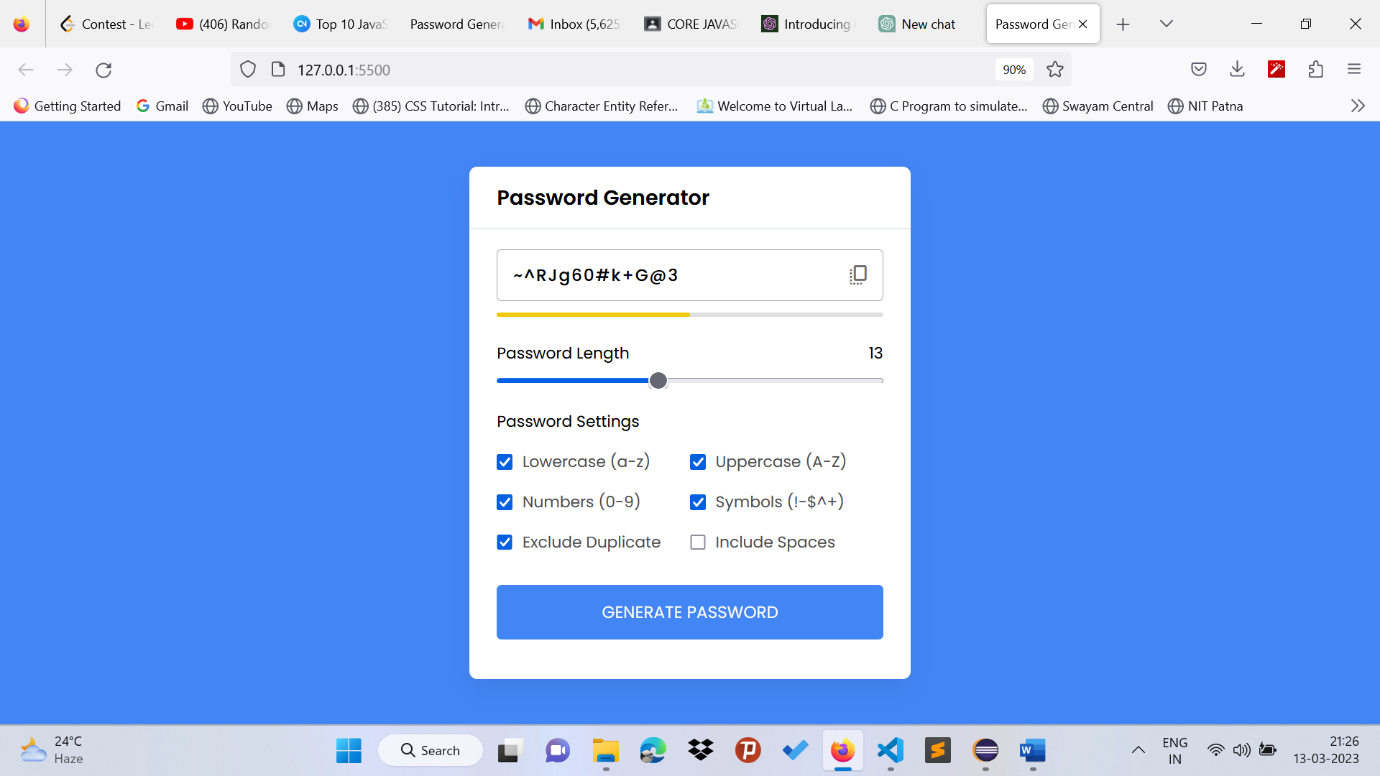
**Using Lowercase,Uppercase,Numbers:**

****

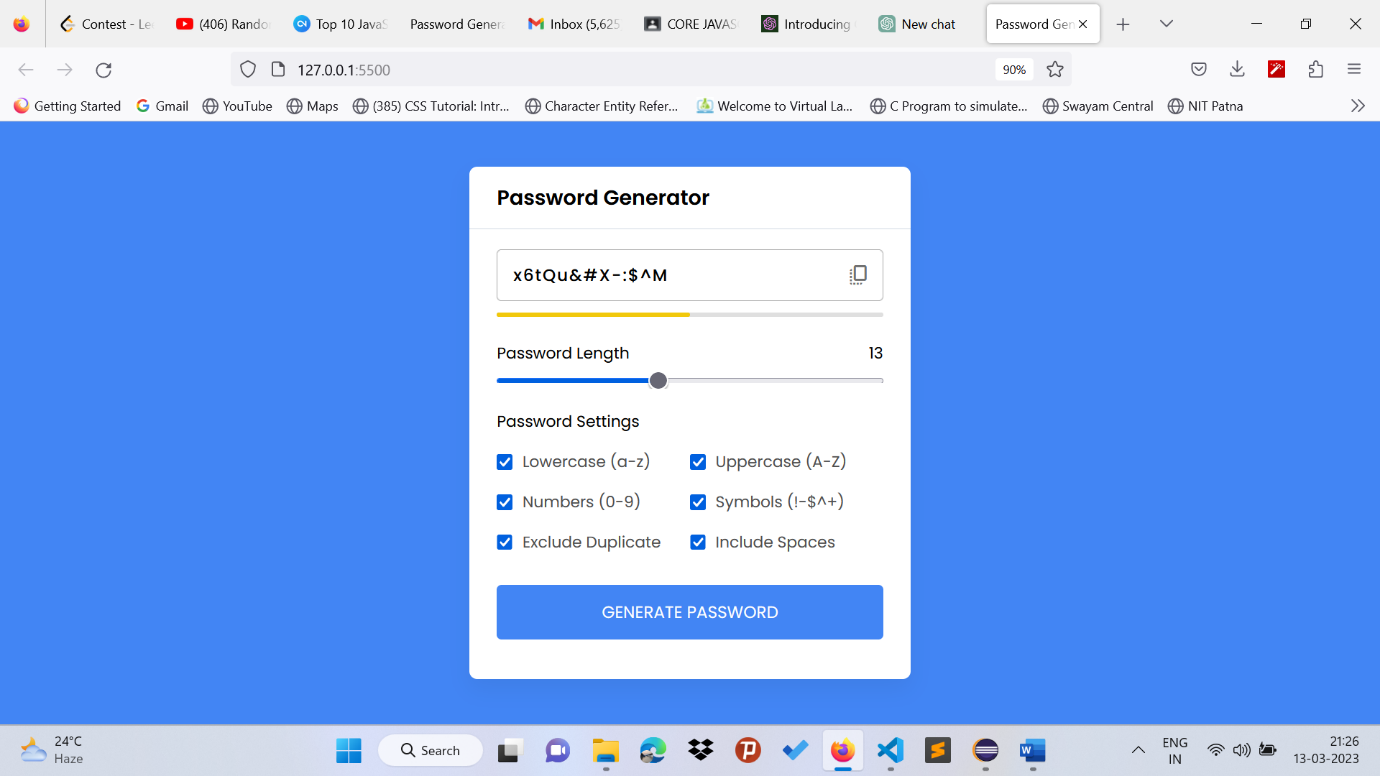
**Using Lowercase,Uppercase,Numbers,Symbols:**

****

**Using Lowercase,Uppercase,Numbers,Symbols,Excluding Duplicates:**

****

**Using all symbols,letters,Numbers,excluding Duplicates:**

****

**FUTURE SCOPE**

The future scope of a random password generator using JavaScript is significant due to the increasing importance of cybersecurity. Here are some potential areas where a random password generator using JavaScript can be applied:

1. Web and mobile applications: With the growing number of web and mobile applications, users are required to create multiple accounts, and using a weak password can lead to data breaches. Integrating a random password generator in such applications can enhance cybersecurity and protect user data.
2. Cloud-based services: Cloud-based services such as Google Drive, Dropbox, and Microsoft OneDrive store large amounts of sensitive data. Integrating a random password generator in such services can ensure that user data is protected from unauthorized access.
3. Internet of Things (IoT): As the use of IoT devices increases, securing these devices becomes crucial. Using a random password generator in IoT devices can ensure that they are protected from hacking attempts.
4. Government and financial institutions: Governments and financial institutions store sensitive data that can be vulnerable to cyber attacks. Integrating a random password generator can enhance security and protect against data breaches.
5. Education and training: As cybersecurity becomes increasingly important, educating and training individuals on the importance of creating strong passwords can be beneficial. A random password generator can be used as a tool to teach users how to create secure passwords.

In conclusion, the future scope of a random password generator using JavaScript is significant due to the increasing importance of cybersecurity in various fields. Integrating a random password generator in web and mobile applications, cloud-based services, IoT devices, government and financial institutions, and education and training can enhance cybersecurity and protect against data breaches.

**CONCLUSION**

In conclusion, a random password generator using JavaScript is an essential tool for enhancing cybersecurity. It generates complex, unique passwords that are difficult for hackers to crack, thus protecting user data from unauthorized access. Developing a random password generator using JavaScript is a relatively simple task that requires minimal hardware and software requirements. The generator can be integrated into various web and mobile applications, cloud-based services, IoT devices, government and financial institutions, and educational settings. The future scope of a random password generator using JavaScript is significant due to the growing need for cybersecurity in various fields. By implementing a random password generator using JavaScript, users can protect themselves against data breaches and ensure that their personal information remains secure.

**REFERENCES AND BIBLIOGRAPHY**

Here are some references and bibliography sources for a random password generator using JavaScript:

1. "JavaScript Random Password Generator" by David Walsh. <https://davidwalsh.name/javascript-password-generator>
2. "How to Create a Random Password Generator with JavaScript" by Freecodecamp. <https://www.freecodecamp.org/news/how-to-create-a-random-password-generator-with-javascript/>
3. "Random Password Generator Using JavaScript" by GeeksforGeeks. <https://www.geeksforgeeks.org/random-password-generator-using-javascript/>
4. "Create a Random Password Generator with JavaScript" by W3Schools. <https://www.w3schools.com/howto/howto_js_password_generator.asp>
5. "JavaScript Password Generator" by Passwords Generator. <https://passwordsgenerator.net/javascript-password-generator/>
6. "Password Generator" by LastPass. <https://www.lastpass.com/password-generator>
7. "Password Generator Tool" by Norton. <https://my.norton.com/extspa/passwordmanager?path=pwd-gen>
8. "Password Generator" by Dashlane. <https://www.dashlane.com/features/password-generator>
9. "The Best Password Generators in 2022" by TechRadar. <https://www.techradar.com/best/best-password-generator>

These sources provide valuable information and insights on how to create a random password generator using JavaScript, as well as recommendations for other password generator tools. They can be used as references and bibliography sources for a project report on random password generators using JavaScript.

THANK YOU