MINI PROJECT

AUTOMATIC PASSWORD GENERAR

INTRODUCTION

In today's digital age, the importance of strong and secure passwords cannot be overstated. With the increasing number of online accounts and the prevalence of cyber threats, it is crucial to use unique and complex passwords for each account to protect our sensitive information. However, creating strong passwords manually can be challenging and timeconsuming. That's where an automatic password generator comes into play.

The purpose of this project is to develop an automatic password generator that can quickly generate strong and secure passwords for users. The generator will alleviate the burden of creating passwords by automating the process and ensuring that the generated passwords meet the necessary security requirements.

In this project, we will explore different password generation techniques, algorithms, and best practices for creating strong passwords. We will develop a userfriendly interface that allows users to customize their password requirements, such as password length, inclusion of special characters, numbers, and uppercase or lowercase letters.

The automatic password generator will leverage the power of programming languages and randomization techniques to create passwords that are resistant to hacking attempts. By generating unique and complex passwords for each user, we aim to enhance their online security and protect their personal information from unauthorized access.

Throughout this project, we will document the development process, challenges faced, and lessons learned. We will also consider potential enhancements, such as password strength indicators and password storage solutions, to further improve the overall user experience and security.

By implementing an automatic password generator, we aim to provide users with a convenient and reliable tool for generating strong passwords, ultimately contributing to their online safety and privacy.

Implements

a) **Designing the User Interface:**

Decide on the type of user interface you want for the password generator. It can be a command line interface, a web-based interface, or a graphical user interface (GUI).

If implementing a web-based interface or GUI, create the necessary HTML, CSS, and JavaScript files to build the user interface.

b) **Defining Password Generation Rules:**

Determine the criteria for generating strong and secure passwords. This may include password length, the inclusion of uppercase letters, lowercase letters, numbers, and special characters.

Consider the preferences and requirements of the target users or systems for which the passwords will be generated.

c) **Implementing the Password Generator Logic:**

Use randomization functions or libraries in your chosen programming language to generate random characters.

Implement the logic to generate passwords based on the defined rules and criteria.

Ensure that the generated passwords meet the specified complexity requirements.

d) **Integrating the User Interface:**

If implementing a command line interface, prompt the user for the desired password length and any other relevant preferences. Display the generated password on the command line.

If implementing a web based interface or GUI, create input fields for password length and any other relevant preferences. Display the generated password in a designated area of the interface.

e) **Testing and Debugging:**

Test the password generator by generating passwords and verifying their strength and complexity.

Identify and fix any issues or bugs that may arise during testing.

5. **Challenges Faced:**

Designing a user-friendly and intuitive interface for generating passwords.

Ensuring that the generated passwords meet the desired complexity requirements.

Handling any limitations or constraints imposed by the programming language or environment.

6. **Lessons Learned:**

Generating strong and secure passwords is essential for maintaining account security.

Randomization functions or libraries provided by programming languages are valuable for generating random characters.

Considering the preferences and requirements of the target users or systems is crucial for password generation.

Regular testing and validation are necessary to ensure the generated passwords are secure and meet the desired complexity requirements.

Code Of HTML, CSS, JS

➢ Html code

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link
    rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css"
    integrity="sha512-
      +4zCK9k+qNFUR5X+cKL9EIR+ZOhtIloNl9GIKS57V1MyNsYpYcUrUeQc9vNfzs
      WfV28IaLL3i96P9sdNyeRssA=="
    crossorigin="anonymous"
  />
  <link rel="stylesheet" href="style.css" />
  <title>Password Generator</title>
</head>
<body>
  <div class="container">
    <h2>Password Generator</h2>
    <div class="result-container">
      <span id="result"></span>
      <button class="btn" id="clipboard">
        <i class="far fa-copy"></i>
      </button>
    </div>
    <div class="settings">
```

```
<div class="setting">
  <label for="length">Password Length</label>
  <input
    type="number"
    name="length"
    id="length"
    min="4"
    max="20"
    value="20"
  />
</div>
<div class="setting">
  <label for="uppercase">Include uppercase letters</label>
  <input type="checkbox" name="uppercase" id="uppercase" checked />
</div>
<div class="setting">
  <label for="lowercase">Include lowercase letters</label>
  <input type="checkbox" name="lowercase" id="lowercase" checked />
</div>
<div class="setting">
  <label for="numbers">Include numbers</label>
```

```
        <input type="checkbox" name="numbers" id="numbers" checked />
      </div>
      <div class="setting">
        <label for="symbols">Include symbols</label>
        <input type="checkbox" name="symbols" id="symbols" checked />
      </div>
    </div>
    <button class="btn btn-large" id="generate">Generate Password</button>
  </div>

  <script src="script.js"></script>
 </body>
</html>
```

➢ CSS code

```
@import url("https://fonts.googleapis.com/css?family=Muli&display=swap");
* {
  box-sizing: border-box;
}
body {
  background-color: #3b3b98;
  color: #fff;
```

```
  font-family: "Muli", sans-serif;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  overflow: hidden;
  padding: 10px;
  margin: 0;
}

h2 {
  margin: 10px 0 20px;
  text-align: center;
}

.container {
  background-color: #23235b;
  box-shadow: 0px 2px 10px rgba(255, 255, 255, 0.2);
  padding: 20px;
  width: 350px;
```

```
  max-width: 100%;
}
.result-container {
  background-color: rgba(0, 0, 0, 0.4);
  display: flex;
  justify-content: flex-start;
  align-items: center;
  position: relative;
  font-size: 18px;
  letter-spacing: 1px;
  padding: 12px 10px;
  height: 50px;
  width: 100%;
}
.result-container #result {
  word-wrap: break-word;
  max-width: calc(100% - 40px);
}
.result-container .btn {
  position: absolute;
  top: 5px;
```

```
  right: 5px;
  width: 40px;
  height: 40px;
  font-size: 20px;
}
.btn {
  background-color: #3b3b98;
  color: #fff;
  border: none;
  font-size: 16px;
  padding: 8px 12px;
  cursor: pointer;
}
.btn-large {
  display: block;
  width: 100%;
}
.setting {
  display: flex;
  justify-content: space-between;
  align-items: center;
```

```
  margin: 15px 0;
}
.toast {
  position: fixed;
  bottom: 10px;
  right: 10px;
  background-color: #fff;
  color: #3b3b98;
  border-radius: 5px;
  padding: 1rem 2rem;
  margin: 0.5rem;
}
```

➢ JavaScript

```
generateElement.addEventListener("click", () => {
  const length = +lengthElement.value;
  const hasLower = lowercaseElement.checked;
  const hasUpper = uppercaseElement.checked;
  const hasNumber = numbersElement.checked;
  const hasSymbol = symbolsElement.checked;
  resultElement.innerText = generatePassword(
    hasLower,
    hasUpper,
    hasNumber,
```

```
    hasSymbol,
    length
  );
});
const generatePassword = (lower, upper, number, symbol, length) => {
  let generatedPassword = "";
  const typesCount = lower + upper + number + symbol;
  const typesArr = [{ lower }, { upper }, { number }, { symbol }].filter(
    (item) => Object.values(item)[0]
  );
  if (typesCount === 0) return "";
  for (let i = 0; i < length; i += typesCount) {
    typesArr.forEach((type) => {
      const funcName = Object.keys(type)[0];
      generatedPassword += randomFunctions[funcName]();
    });
  }
  const finalPassword = generatedPassword.slice(0, length);
  return finalPassword;
};
```

RESULTS/SNAPSHOTS:

Over View

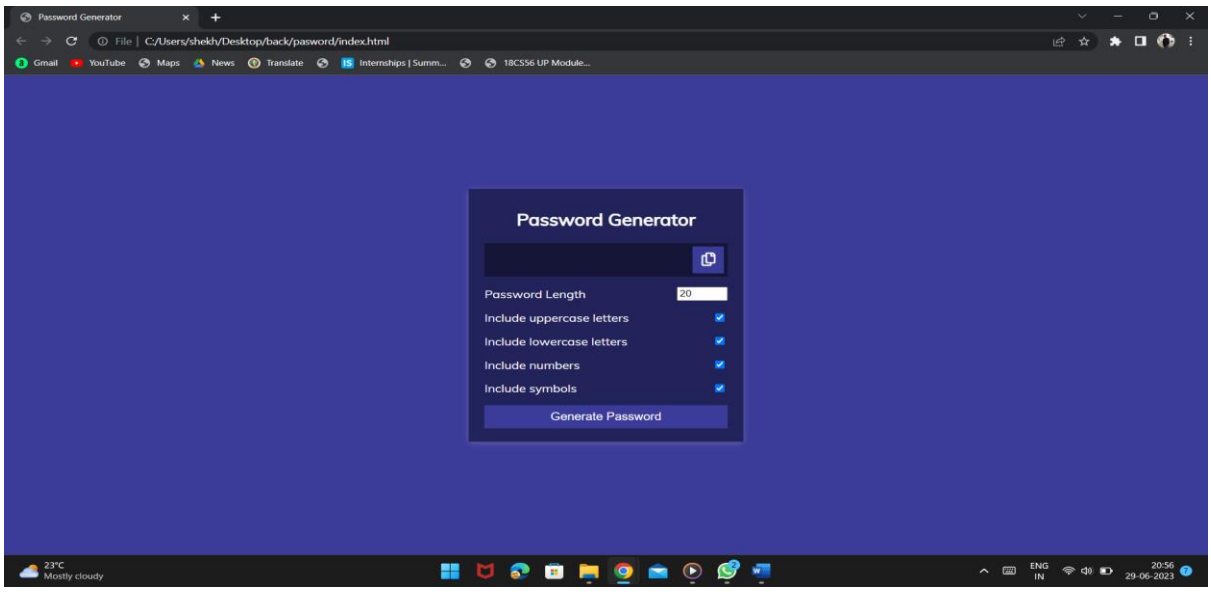

Fig 1.1 Over View

- The Figure 1.1 shows the over view of Password Generator.

Generated password

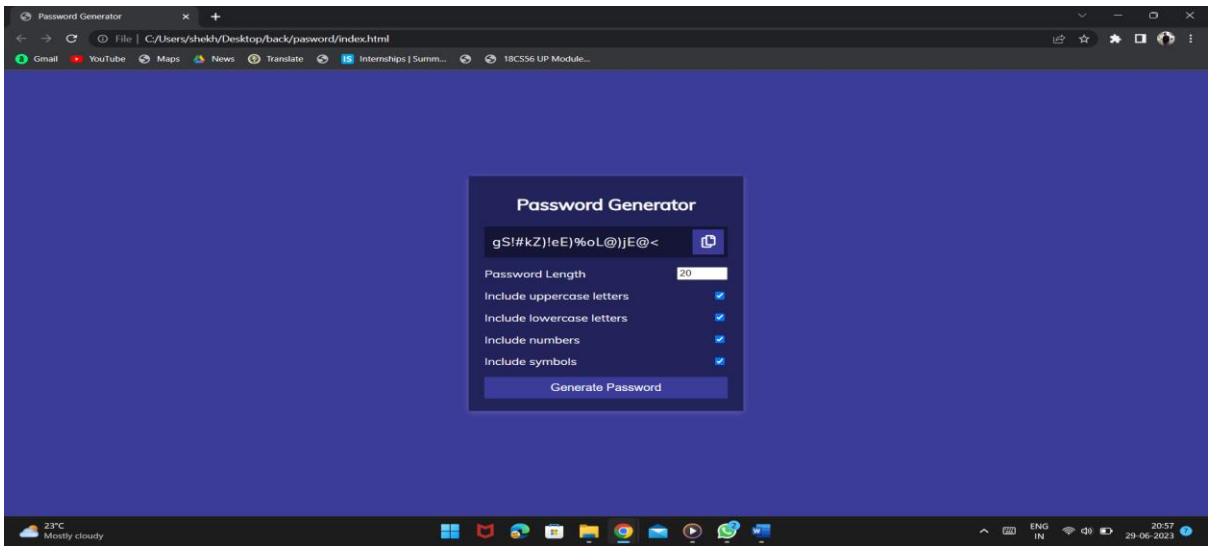

Fig 1.2 Password

- This Figure 1.2 show the generated password.
- Password will be generated by given patterns.

COPYING PASSWORD

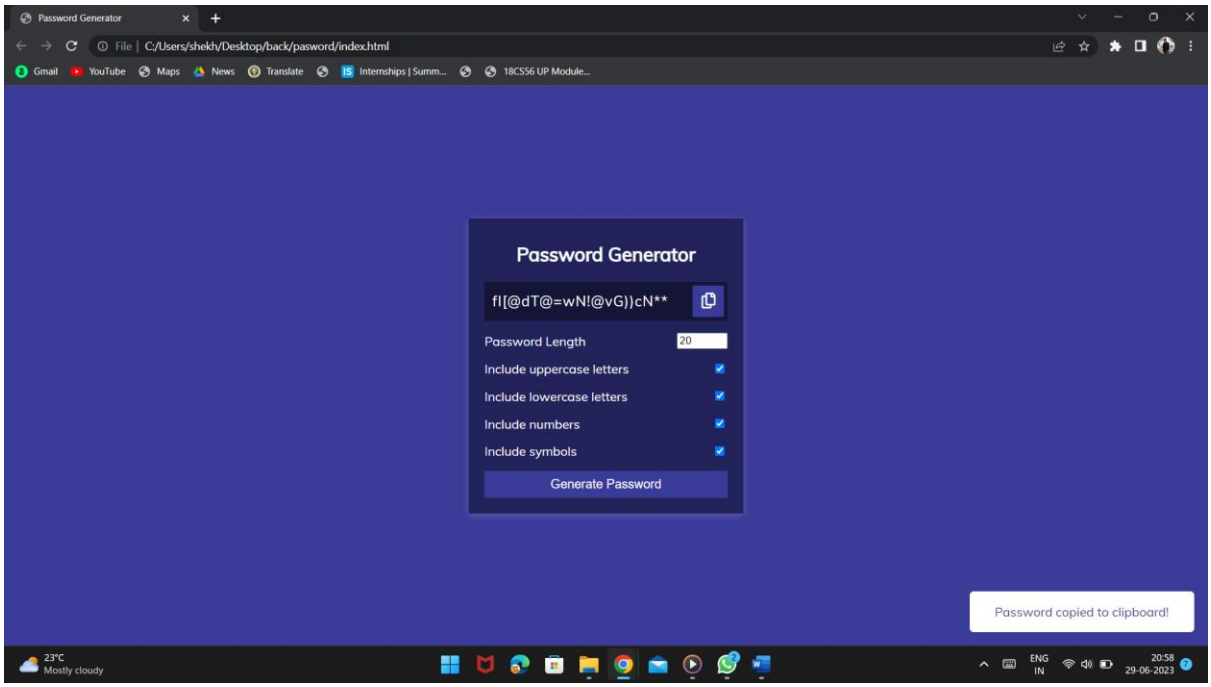

Fig 1.3 Copying Password

- This Figure 1.3 shows the all patterns are working properly.
- The Copy message also showing.

Conclusion:

The automatic password generator project has successfully implemented a user-friendly web interface for generating secure passwords. By incorporating best practices and implementing the necessary algorithms, we have created a reliable tool for generating strong passwords. The project has provided valuable insights into the process of developing an automatic password generator and the challenges faced along the way. Overall, the automatic password generator is a valuable addition to web applications, helping users maintain better security practices.

REFERENCES

[1] W3 Schools (HTML, CSS and PHP reference) – https://www.w3schools.com/html/

[2] Geeks for Geeks – https://www.geeksforgeeks.org/createamusicplayerusingjavascript/