1. ABSTRACT

In today's digital landscape, passwords are essential for accessing any platform or website. However, remembering complex passwords across multiple sites is often a daunting task. To resolve this issue, this project introduces a password system using a graphical password strategy. This system allows users to create passwords by selecting a specific graphical pattern, which can then be used later for logging in. The project includes both login and registration phases, making it suitable for integration into any website. Unlike traditional password systems, this graphical password authentication method is more resistant to cybersecurity threats such as dictionary attacks and brute force attacks. In current graphical password authentication techniques, shoulder surfing (or shoulder sniffing) is still a concern which is also resolved in this project. It offers a significantly larger key space of 10^{13} (10 trillion), compared to the 8^{10} (1.07 billion) key space of alphanumeric passwords. Graphical passwords provide an expanded key space by allowing users to select from a wide range of images, patterns, or gestures, making it more secure. In this system, users select a sequence of fewer than four tiles from a set of four images. By integrating the Unsplash API, the system allows users to choose images from a vast library of categories, creating a more personalized and unrestricted password. To ensure human verification, the system employs a combination of Image-based CAPTCHA and Puzzle CAPTCHA. The encryption method used to secure the image sequence is similar to cipher block chaining mode, further enhancing security. This graphical password authentication system offers a more secure and user-friendly alternative to traditional alphanumeric passwords.

2. INTRODUCTION

Human factors are frequently regarded as the weakest point in a computer security system. There are three key areas where the interaction between humans and computers plays a crucial role: authentication, security operations, and designing secure systems. This discussion focuses primarily on the authentication challenge. However, in the modern era, it has come to signify a string of characters (letters, numbers, symbols) used for user authentication. Strong passwords that are difficult to crack or guess often pose a problem because they are also hard for users to remember. Research indicates that due to the limited capacity of users to retain multiple passwords, they tend to either write them down or reuse the same password across multiple accounts. To mitigate the challenges associated with traditional username and password authentication, alternative methods like biometrics have been introduced.

The authentication techniques can be distinguished in three types: knowledge based systems, token based system and biometrics based systems.

Token-based authentication falls under the "what you have" category, using physical items like key cards or smart devices to verify identity. This method is common in systems like ATMs but can be insecure if tokens are lost or stolen. Biometric authentication, categorized as "what you are," uses features like fingerprints or iris scans. While accurate, it can be costly and prone to false positives or negatives, as well as environmental or physical disruptions. Knowledge-based authentication, or "what you know," involves alphanumeric or graphical passwords. Though effective, it is vulnerable to phishing and can lead to forgotten passwords or frequent account recovery.

These types of passwords can be vulnerable to various cybersecurity attacks if not implemented with robust security measures. They are susceptible to brute force attacks, dictionary attacks, spyware attacks, shoulder surfing, social engineering attacks, and other similar threats.

Graphical passwords use images, which are easier for people to recall compared to long strings of characters found in text passwords. These graphical password systems have been suggested as a potential alternative to text-based methods, driven in part by the idea that humans are better at remembering pictures than words. Psychological research supports this assumption, showing that images are typically easier to remember or recognize than text.

Graphical passwords come in several types, including Recognition-based (Cognitive Authentication), Recall-based (Pure Recall), Cued Recall-based, and Hybrid Schemes. Each type has its own set of advantages and disadvantages. This project implements a combination of Recognition-based and Cued Recall-based graphical passwords, offering competitive strength compared to alphanumeric passwords. It provides additional benefits, such as protection against dictionary attacks and making brute force attacks more difficult.

Graphical passwords, while advantageous compared to text-based ones, are particularly vulnerable to "shoulder sniffing," where someone nearby observes sensitive information, like passwords, by watching over a user's shoulder. This type of attack involves visually capturing a password by observing a screen, phone, or other devices. Due to their visual interaction, graphical passwords are often more exposed to shoulder sniffing, though the level of risk can vary based on the system's design and usage. Various strategies have been introduced to minimize this vulnerability, such as dynamic grids with changing elements, decoy images, complex gestures, randomized click points, multi-factor authentication (MFA), and time-based expiration. While these techniques help reduce the risk, they do not entirely eliminate the threat. In this project, specific methods have been implemented to counteract shoulder sniffing. To further ensure human verification, the system incorporates both Image-based and Puzzle CAPTCHA. Additionally, the login and registration pages are designed to be more user-friendly.

3. PROBLEM STATEMENT

Computer passwords first appeared in 1961 with the Compatible Time-Sharing System (CTSS) at MIT. Graphical passwords, which use images instead of text, were introduced later as an alternative. Over time, passwords have become more advanced due to technology changes and security needs. What started as simple passphrases has grown into complex systems with multi-factor and passwordless options. This evolution reflects the growing sophistication of technology and the increasing security threats we face. From early military codes to modern cryptographic systems, the development of passwords has adapted to these changes.

But this progress also comes with challenges. Complex passwords can be tough to use, and collecting sensitive biometric data can raise privacy issues. Advanced technologies can introduce new security risks and single points of failure. Plus, strict security measures might make things harder for users and drive up implementation costs. Let's look at some of these drawbacks.

Brute Force Attack, Dictionary Attack and rainbow tables (precomputed tables of hashes for common passwords) :

Brute-force attacks attempt every possible combination of characters to break a password, which can be a lengthy process. Dictionary attacks rely on lists of commonly used passwords to quickly guess weak or predictable ones. Rainbow table attacks use precomputed tables of hashed values to quickly match a hash with its original password, making it easier to crack common passwords. Each of these methods exploits different vulnerabilities in password security.

Several measures have been implemented to prevent these attacks, including using strong and complex passwords, rate limiting and lockout mechanisms, key-stretching algorithms, salting, applying strong hashing algorithms, and hashing with iterations. However, there are issues with these solutions.

If salts—random values added to passwords before hashing—are not used or are poorly implemented, attackers can use precomputed tables like rainbow tables to crack hashes more easily. It is crucial to use unique salts for each password to avoid this vulnerability. Additionally, using weak hashing algorithms increases the likelihood of them being cracked by attackers. Modern hashing algorithms are designed to be more secure, but even they can sometimes fail.

2018 Github Data Breach:

GitHub experienced a data breach where a bug in their password hashing implementation led to the exposure of passwords. Although GitHub used bcrypt for hashing passwords, an issue in their implementation meant that some passwords were not hashed with an adequate work factor.

Advanced Attack Techniques:

Attackers use sophisticated techniques, such as parallel computing and GPU acceleration, to perform rapid hash computations and increase their chances of cracking passwords, even with strong algorithms.

LinkedIn Data Breach (2012):

LinkedIn initially used the SHA-1 hashing algorithm to hash passwords. SHA-1, while once considered secure, is now known to be vulnerable to various attacks and is not recommended for cryptographic purposes. Attackers utilized GPU acceleration to perform rapid computations and crack the hashed passwords. The availability of powerful GPUs allowed them to quickly guess the original passwords by testing large numbers of possible combinations.

Password complexity requirements:

The creation of password complexity requirements, such as rules for password length, and the inclusion of upper and lower case letters, numbers, and special characters, can also contribute to security issues. The need for users to remember complex and numerous passwords has led to poor practices like password reuse and choosing weak passwords. If users opt for weak or easily guessable passwords, attackers can still crack them quickly, even with the use of salting. Salting doesn't protect against the fundamental weakness of the password itself. Short passwords are another issue, as they have fewer possible combinations, making them easier to crack through brute-force attacks, even when strong hashing algorithms are used. Users often create passwords based on predictable patterns like "password123" or "qwerty," which remain vulnerable to attacks, regardless of the hashing method.

Graphical password authentication systems also come with challenges. While graphical passwords provide a more user-friendly experience, they lack scalability and are susceptible to shoulder-surfing, where attackers observe the password entry process. Screen capture attacks are another concern: if an attacker can view a user’s screen, they can easily steal the graphical password. Users also tend to create predictable patterns in graphical passwords. For instance, in systems like Passpoints, where users click on points within an image, many select locations that are meaningful or easy to remember. Although graphical passwords can be more complex than traditional text-based passwords, they remain vulnerable to brute-force attacks, where attackers try all possible combinations of clicks or gestures. Replay attacks are another risk, where an attacker intercepts and replays the data transmitted during login. This is particularly relevant for systems where graphical password inputs are transmitted over a network. A variation of shoulder surfing, known as over-the-shoulder replay, involves an attacker observing a user entering their graphical password multiple times. However, this project implements various techniques to address and mitigate many of these problems.

4. LITERATURE REVIEW

S.No	Title	Authors & Year	Problem	Proposed
1.	Comparison of Graphical Password Authentication Techniques	Arti Bhanushali, Bhavika Mange, Harshika Vyas, Hetal Bhanushali and Poonam Bhogle. (2015)	Comparing different graphical password techniques	Comparison of Draw A Secret, Grid Selection Algorithm, PassPoint Algorithm, Déjà vu Algorithm
2.	Shoulder Surfing attack in graphical password authentication	Arash Habibi Lashkari, Samaneh Farmand, Dr. Omar Bin Zakaria, Dr. Rosli Saleh. (2009)	Vulnerability of graphical password authentication.	How graphical password authentication is vulnerable to shoulder surfing attack
3.	Graphical Password Authentication Schemes: Current Status and Key Issues	Harsh Kumar Sarohi, Farhat Ullah Khan	Current Status and Key Issues of graphical password authentication.	Discussions about Shoulder surfing attack, Brute force attack, Guessing, Social Engineering, Spyware attack and how related to graphical password authentication.
4.	The Shoulder Surfing Resistant Graphical Password Authentication Technique.	Mrs. Aakansha S. Gokhale, Vijaya S. Waghmare Prof.	Sholder sniffing which is a big concern	New technique for avoiding shoulder sniffing
5.	Network security-overcome password hacking through graphical password authentication.	M. ArunPrakash, T.R. Gokul	Study about graphical password authentication	Strengths and limitations of each method and future directions

6.	An XML transformed method to improve effectiveness of graphical password authentication.	Kapil Juneja	Managing pictorial or image data	XML based schema is proposed to represent the graphical image
7.	Cued Click Points: Graphical Password Authentication Technique for Security	Shendage Swapnil Sunil, Dhainje Prakash, Yevale Ramesh Shivaji	Key generation requires an output of many more bits	Proposed a new click-based graphical password scheme called Cued Click Points (CCP)
8.	Security Evaluation for Graphical Password	Arash Habibi Lashkari, Azizah Abdul Manaf, Maslin Masrom & Salwani Mohd Daud	Evaluation of graphical password authentication.	Security aspects of all the algorithms have been discussed.
9.	Token-based graphical password authentication.	John Charles Gyorffy, Andrew F. Tappenden & James Miller	Graphical password deployed from a Trojan and virus-resistant	Graphical password utilizes a personal image to construct an image hash
10.	GRAMAP: THREE STAGE GRAPHICAL PASSWORD AUTHENTICATION SCHEME	S.RAJARAJAN, M. PRABHU, S. PALANIVEL, M.P.KARTHIKEYAN.	When multiple users share a set of systems and resources, effective authentication is very much required	New graphical password scheme based on geographical maps

5. PROPOSED SYSTEM

The current text-based password authentication systems across many domains face numerous issues and vulnerabilities. These include brute force attacks, dictionary attacks, rainbow tables, phishing, social engineering, password fatigue, improper implementation of hashing algorithms, and the use of weak hashing techniques. Additionally, password fatigue arises when users are required to create complex passwords involving length rules, mixed case letters, numbers, and special characters. Existing graphical password authentication systems also present challenges, such as vulnerability to shoulder-surfing and screen capture attacks. This project incorporates several techniques aimed at addressing and mitigating many of these problems.

MongoDB Atlas is a cloud-based, fully managed database service offered by MongoDB. It enables developers and organizations to easily deploy, manage, and scale MongoDB databases in the cloud, taking care of many operational tasks automatically. In this project, MongoDB is utilized for storing user information, selected images, and password-related data.

The project features both login and registration phases, making it versatile for integration into any website. During the registration phase, users can create graphical passwords. In this stage, they provide necessary details and choose from four categories of images for selection in each round. The project enhances security by offering a wider key space, allowing users to select from an extensive variety of images, patterns, or gestures. It also integrates the Unsplash API, enabling users to choose images from a vast, diverse library, making their password creation more personalized and flexible. The Unsplash API, a widely used platform for high-quality stock photos, allows users to search and interact with its extensive image library. Through the API, users can search for images, retrieve random photos, access curated collections, and gather detailed information about specific photos or photographers. This creates an unrestricted and customizable experience for graphical password creation.

The system utilizes a combination of Image-based CAPTCHA and Puzzle CAPTCHA to determine if the user is human or automated. To verify this, users are required to draw the number displayed using their mouse cursor. Since traditional image-based CAPTCHAs are often vulnerable to being bypassed, this method is more effective and harder to deceive, enhancing the security of the verification process.

The project employs an encryption method akin to cipher block chaining mode to secure the image sequences, thereby boosting security. When a user selects a sequence of tiles in each image, this sequence serves as the encryption key for the subsequent images. Consequently, if an incorrect sequence is entered, the preceding images will not be displayed, thereby enhancing the overall security of the system.

Hashing and salting play crucial roles in enhancing security and protecting data from various types of attacks. Strong hashing algorithms like bcrypt, scrypt, and Argon2 are currently among the most effective. In this project, Argon2, which won the Password Hashing Competition, is employed to hash the sequence of tiles selected in the images. These algorithms are designed to slow down attackers by allowing adjustments to the computational cost. If the cost is set too low, even robust algorithms can become vulnerable to brute force attacks due to inadequate computational effort. Additionally, a low cost might result from weak and easily guessable passwords. When attackers use methods such as parallel computing or GPU acceleration, the security can be compromised if the passwords are weak. However, since graphical password sequences are tied to images, they are less likely to be weak, thereby enhancing overall security.

Shoulder sniffing and over-the-shoulder replay are significant concerns when implementing a graphical password authentication system. To address these issues, this project incorporates a security measure where a border is created around the selection division shown to the user. When the user moves the mouse cursor within this division, the cursor becomes invisible. Additionally, when the cursor transitions from one tile to another within the division, the border will blink, enhancing security and reducing the risk of visual eavesdropping.

6. OBJECTIVES AND SCOPE:

The goal of this project is to design and implement a robust and sustainable graphical password authentication system that addresses the vulnerabilities and limitations found in traditional text-based and existing graphical password systems. The main objectives are to enhance security, improve user experience, and ensure long-term viability by incorporating advanced techniques to tackle various issues. This includes mitigating common threats such as brute-force attacks, dictionary attacks, and rainbow table attacks by utilizing advanced security measures like strong hashing algorithms, encryption, and salting. The project also aims to create a user-friendly interface for password creation and authentication that simplifies the process and minimizes password fatigue.

Additionally, features will be implemented to protect against shoulder surfing and screen capture attacks, ensuring the graphical password system remains secure during practical use. The system will enhance security by combining graphical passwords with other verification methods, such as CAPTCHA, to confirm user identity. It will be designed to be sustainable over time by integrating features that adapt to evolving security threats and user needs, including the ability to integrate with modern authentication practices, support updates, and address emerging vulnerabilities.

The system will have a modular architecture to facilitate easy updates and integration with other authentication technologies. It will be deployed in a controlled environment for testing functionality and performance, ensuring compatibility with existing systems and infrastructure. Users will create passwords by selecting sequences from images, patterns, or gestures, with the system offering a wide key space through integration with the Unsplash API, which provides a rich and personalized password creation experience.

The system will support both user registration and login, making it suitable for integration into various websites and applications. During registration, users will enter their details and select images in each round to create their graphical passwords. Encryption methods similar to cipher block chaining mode will be used to secure image sequences, with the selected tile sequence serving as the encryption key. This prevents incorrect sequences from displaying prior images, thus enhancing security.

The modular architecture will ensure the system can easily be updated and integrated with other technologies, remaining adaptable to future advancements and security threats. Comprehensive documentation will be provided for end-users and administrators, along with training to ensure proper usage and maintenance, including managing potential issues and performing updates. Overall, the project aims to develop a robust graphical password authentication system that effectively addresses current security and usability challenges while ensuring sustainability and adaptability.

7. REQUIREMENT SPECIFICATION

The requirements needed to create the application are given in detail in this system specification. After analysis, these needs are formally documented. This is then translated into technical terminology. The development stage was able to build and have a clear grasp of the product needed to create thanks to it. This careful specification helped in avoiding software problems. When there were several choices available for all system specification components, the proper option was selected, ensuring that the application would be lightweight, effective, and able to handle and prevent problems.

Software Requirements:

- **Operating systems:** The system can be developed in **windows 10 and above versions, MacOS 10.14 and Linux.**
- **Browser:** The graphical password authentication system is developed as a web application and is designed to be compatible with all modern web browsers
- **Backend infrastructure:** This system will be built using the **node.js environment** and will be loaded in chrome browsers which are enabled with **v8 engine.**
- **Languages, libraries and APIs:** The system is developed using **HTML, CSS, and JavaScript**. Framework and extensions such as **React.js, Tailwind CSS, DaisyUI** and api's such as **express.js** are required for the development. Apart from this unsplash API is required to access the images from unsplash database. Argon2 is also used for hashing. Python is also used for implementing the flask server.
- **Text Editor:** A text editor **Visual Studio Code** used to write, edit, and manage your extension's code. Visual Studio code is more preferred. Because VS Code has a vast extension market place, a built-in debugger for JavaScript, and GIT integration.
- **Database:** MongoDB and MongoDB Atlas is utilized for storing user information, selected images, and password-related data.

Hardware Requirements:

- **Ram**: A minimum of **8 GB of RAM** is generally recommended for developing this system.
- **CPU**: The developing needs a normal processor capable of running simple tasks is enough. So a modern **multi-core CPU (e.g., Intel Core i9)** should be sufficient.

Input Requirements:

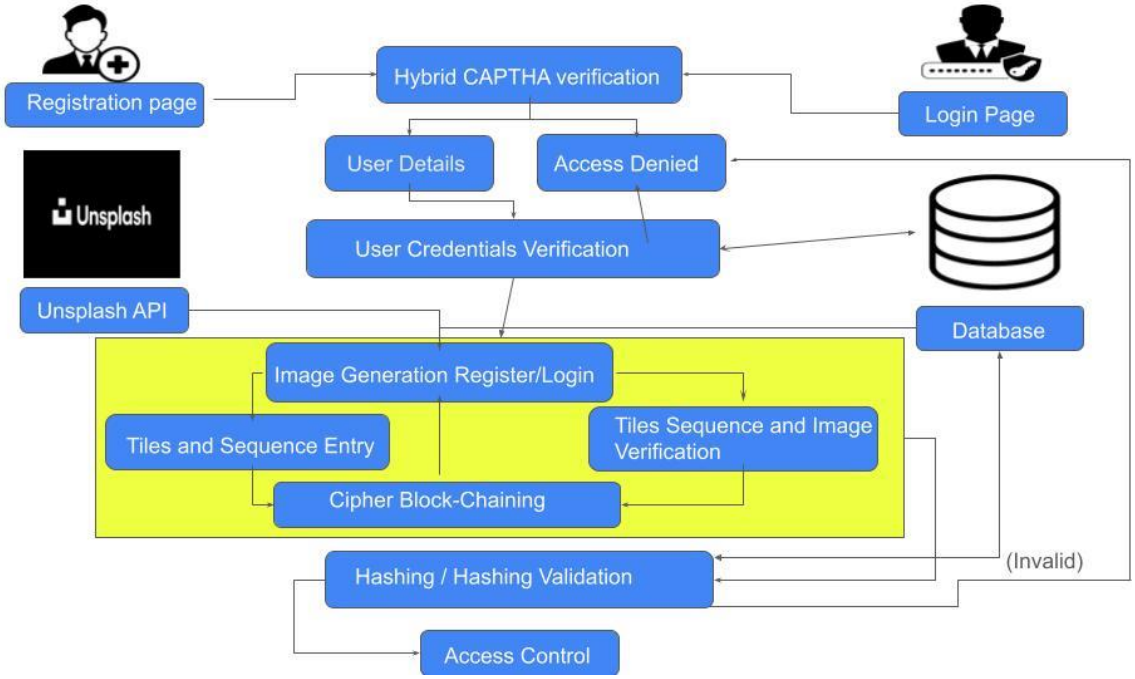
- Sequence of data being selected in the grid layout

Data Requirements:

- Sequence of data being selected
- Hash output

8. SYSTEM DESIGN AND METHODOLOGY

The system design of the proposed graphical password authentication system balances security with user-friendliness. Users create passwords by selecting image sequences from the Unsplash API, which are then encrypted using a method similar to Cipher Block Chaining (CBC) and hashed with Argon2 for strong protection against attacks. CAPTCHA challenges are used during registration and login to verify human users. Security is enhanced with an invisible cursor and blinking borders to prevent shoulder-surfing. Security is further enhanced by implementing an invisible cursor during password selection and using a blinking border around the image grid to prevent shoulder-surfing attacks. All the specifications that were mentioned before were connected properly and used efficiently. The project features both login and registration phases, making it versatile for integration into any website. The project is also developed in such a way that it can be implemented in any websites easily.



Registration Page:

The registration phase or page enables new users to create their graphical passwords. It checks the database to ensure that the credentials do not already exist before allowing users to register and create their graphical passwords. Registration will proceed based on successful Hybrid CAPTCHA verification and user credential validation; otherwise, the process will be halted.

Login Page:

The login phase or page enable the users to login to their accounts using the created graphical passwords. It checks the database to ensure that the credentials already exist before allowing users

to login using their graphical passwords. Login will proceed based on successful Hybrid CAPTCHA verification and user credential validation; otherwise, the process will be halted.

Hybrid CAPTHA verification:

The Hybrid CAPTCHA verification in this project combines Image-based CAPTCHA and Puzzle CAPTCHA to differentiate between human users and bots. Users are required to draw the displayed number using their mouse cursor as part of the verification process. This Hybrid CAPTCHA is implemented on both the login and registration pages. By using libraries like HuggingFace and SpaCy, the system adds complexity, requiring users to interact with images and solve puzzles. The task of drawing numbers with the mouse introduces a challenge for bots, as it demands human-like gestures that are difficult for automated systems to mimic.

Database:

MongoDB Atlas is a cloud-based, fully managed service offered by MongoDB that enables developers to seamlessly deploy, manage, and scale MongoDB databases on platforms like AWS, Google Cloud, and Microsoft Azure. In this project, the MongoDB NoSQL database is used to store user data such as hashed values, selected images, and the sequence of tiles chosen for each image. The image data, sourced from the Unsplash API, is stored as links in JSON files. As this is a cloud-based database, an internet connection is required for its operation.

Unsplash API:

Since it is graphical password authentication system images are actually considered as password source. But when some of the pre-defined or already selected images are used this will in-turn reduce the password space when compared to other techniques. When users are asked to upload own images it may in-turn reduce the security. To increase the passwords space Unsplash API is used. This gives us the access to various and large amount of images.

Image Generation, Tiles and Sequence entry and CBC:

In this the images are being generated with the help of Unsplash API and the image category input from the user in the registration page. Tiles and sequence entry will allow user to select the image and also select the sequence of tiles. Each sequence of tiles of an image is encryption of next image.

Image Generation, Tiles Sequence and image check and CBC:

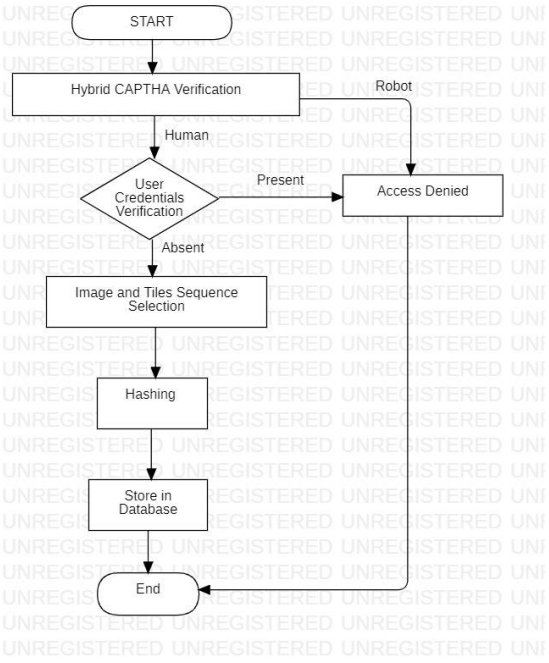
In this the images are being generated with the help of Unsplash API and the image category input given by the user in the login page. Tiles sequence and image selection will check the correct sequence and correct image is selected. Each sequence of tiles of an image is encryption of next image.

Hashing and verification:

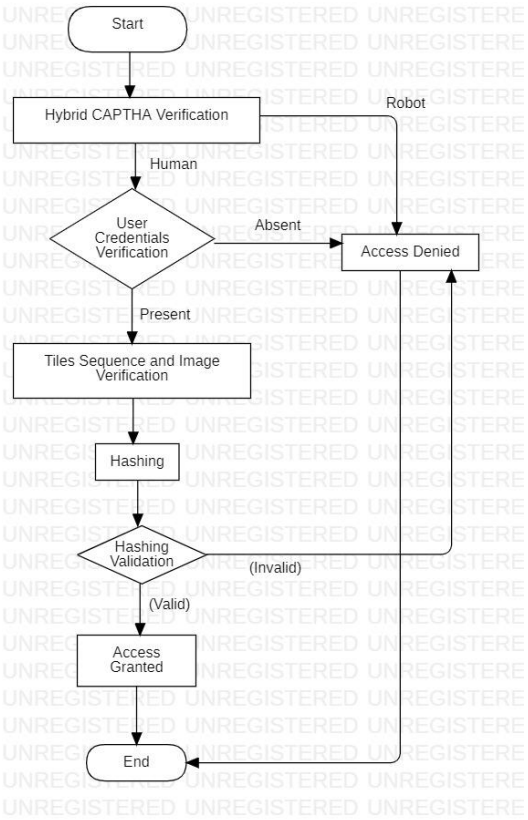
In this the output sequence is converted into hash value ARGON2 algorithm and then stored in database. I login page the correctness is checked only then access is given.

9. SYSTEM IMPLEMENTATION:

Registration page



Login page



Graphical password authentication replaces traditional text-based passwords with a sequence of images chosen by the user. The first step in implementing this system is designing the graphical interface. You'll need

to select a set of images and create a grid layout where these images will be displayed during authentication. This grid can be a 3x3 or 4x4 layout, depending on your design preference.

During the registration phase, users select a sequence of images to form their graphical password. This sequence is then securely hashed and stored in a database. It's crucial to hash the graphical passwords using a strong cryptographic algorithm like ARGON2 to ensure security. This hashed password is what you'll compare against during the authentication phase to verify the user's identity.

In the authentication phase, users are presented with the grid of images they selected during registration. They recreate their password by clicking or tapping on the images in the same sequence. The system then verifies this entered sequence against the stored hashed password in the database. If the sequences match, the user is successfully authenticated.

Security is paramount in graphical password authentication. Always hash and securely store the graphical passwords in the database to prevent plaintext exposure. Use salting along with hashing to add an extra layer of security. Implement measures to protect against brute-force attacks, such as account lockout after multiple failed attempts. Additionally, ensure that the graphical password data is transmitted securely over HTTPS to protect against interception. Regular security testing and user feedback will help identify and fix vulnerabilities, ensuring a secure and user-friendly graphical password authentication system.

With increasing technical advancements the world is becoming digital at a high pace and everything is happening online. From paying your bills to ticket bookings to paying the person sitting next to you, you prefer to pay online. Not only payments but all activities, be it, communication through e-mails and messaging apps, keeping your documents in a digital locker, etc happen online. With everything turning online, the risk of cybercrimes and privacy breaches is also increasing. Passwords play a huge role in keeping your data safe online as well as offline platforms. Passwords are the default method of authentication to get access to our accounts. There are various types of authentication available for users to secure their accounts. Types of authentication

- Token-based authentication includes key cards, bank cards, smart cards, etc.
- Knowledge-based authentication includes text-based authentication and picture-based authentication.
- Biometric authentication include fingerprints authentication, iris scan and facial recognition.

Considering the traditional username-password authentication, the alphanumeric passwords are either easy to guess or difficult to remember. Also, users generally keep the same passwords for all their accounts because it is difficult to remember a lot of them. Alternative authentication methods, such as biometrics, graphical passwords are used to overcome these problems associated with the traditional username-password

authentication technique. In a graphical password authentication system, the user has to select from images, in a specific order, presented to them in a graphical user interface (GUI). According to a study, the human brain has a greater capability of remembering what they see(pictures) rather than alphanumeric characters. Therefore, graphical passwords overcome the disadvantage of alphanumeric passwords. Graphical Password Authentication has three major categories based on the activity they use for authentication of the password:

- Recognition based Authentication: A user is given a set of images and he has to identify the image he selected during registration. For example, Passfaces is a graphical password scheme based on recognizing human faces. During password creation, users are given a large set of images to select from. To log in, users have to identify the pre-selected image from the several images presented to him.

- Recall based Authentication: A user is asked to reproduce something that he created or selected at the registration stage. For example, in the Passpoint scheme, a user can click any point in an image to create the password and a tolerance around each pixel is calculated. During authentication, the user has to select the points within the tolerance in the correct sequence to login.

- Cued Recall: Cued Click Points (CCP) is an alternative to the PassPoints technique. In CCP, users click one point on each image rather than five points on one image (unlike PassPoints). It offers cued-recall and instantly alerts the users if they make a mistake while entering their latest click-point.

SAMPLE CODING

SERVER:

App.js:

```
import React from "react";
import { Login } from "./components/login";
import { Register } from "./components/register";
import { Home } from "./components/home";
import { Router } from "@reach/router";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import { Authenticated } from "./components/authenticated";

function App() {
  // console.log("Siva")
  return (
    <>
      <Router>
        <Register path="/register" />
        <Login path="/login" />
        <Home path="/" />
        <Authenticated path="/authenticated" />
      </Router>
      <ToastContainer
        position="bottom-right"
        autoClose={5000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
        pauseOnFocusLoss
        draggable
        pauseOnHover
      />
```

```
    </>
  );
}

export default App;
```

Login.jsx:

```
import React, { useState, useRef } from "react";
import axios from "axios";
import { ImageGrid } from "../imageGrid";
import CryptoJS from "crypto-js";
import { toast } from "react-toastify";
import { navigate } from "@reach/router";
import Canvas from "../canvas/Canvas";

// const NUM_TILES = Number(process.env.REACT_APP_NUM_TILES);
// const NUM_ROUNDS = Number(process.env.REACT_APP_NUM_ROUNDS);

const NUM_TILES = 4
const NUM_ROUNDS = 4

function hashImage(image, ref_point) {
  const str = image + ref_point.join();
  return CryptoJS.SHA256(str).toString(CryptoJS.enc.base64);
}

function Login() {
  const [email, setEmail] = useState("");
  const [ imageCaption , setImageCaption ] = useState("");
  const [roundNumber, setRoundNumber] = useState(0);
  const [images, setImages] = useState([]);
  const sequences = useRef([]);
  const [showCaptcha, setShowCaptcha] = useState(false);
  const [isHuman, setIsHuman] = useState(false);
```

```
  const handleSubmit = async () => {
    console.log(roundNumber);
    // fetching first round images
    if (roundNumber === 0) {
      console.log("round 0, no img tile");
      try {
        const res = await axios.post(
          "http://localhost:4000/api/login",
          {
            email: email,
            iterationNum: roundNumber,
            passwordHash: "",
            key: "",
          },
          { "Content-Type": "application/json" }
        );

        if (res.status === 200) {
          setImages(res.data.images);
          setImageCaption(res.data.caption)
          setRoundNumber((prev) => prev + 1);
        }
      } catch (err) {
        console.error(err);
      }
    }

    // middle rounds
    else if (roundNumber < NUM_ROUNDS) {
      console.log("int rounds");
      try {
        console.log(sequences);
        const currentSequence = sequences.current[sequences.current.length -
1];
        const res = await axios.post(
```

```
          "http://localhost:4000/api/login",
          {
            email: email,
            iterationNum: roundNumber,
            passwordHash: "",
            key: hashImage(currentSequence.image,
currentSequence.tileSequence),
          },
          { "Content-Type": "application/json" }
        );

        if (res.status === 200) {
          setImageCaption(res.data.caption);
          setImages(res.data.images);
          setRoundNumber((prev) => prev + 1);
        }
      } catch (err) {
        console.error(err);
      }
    }

    // last round
    else if (roundNumber === NUM_ROUNDS) {
      console.log("last r");
      const hashes = [];
      console.log(sequences.current.length);
      sequences.current.map((imageSelection) => {
        hashes.push(
          hashImage(imageSelection.image, imageSelection.tileSequence)
        );
      });

      const passwordHash = CryptoJS.SHA256(hashes.join()).toString(
        CryptoJS.enc.base64
      );
```

```
      const currentSequence = sequences.current[sequences.current.length - 1];
      try {
        const res = await axios.post(
          "http://localhost:4000/api/login",
          {
            email: email,
            iterationNum: roundNumber,
            passwordHash: passwordHash,
            key: hashImage(currentSequence.image,
currentSequence.tileSequence),
          },
          { "Content-Type": "application/json" }
        );

        if (res.status === 200) {
          console.log("success!!");
          toast.success("Logged in successfully!", {
            position: "bottom-right",
            autoClose: 5000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
          });
          navigate("/authenticated");
        }
      } catch (err) {
        console.error(err);
        if (err.response.status === 401) {
          toast.error("Invalid login", {
            position: "bottom-right",
            autoClose: 5000,
            hideProgressBar: false,
```

```
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
        setRoundNumber(0);
        setIsHuman(false);
        setImages([]);
        sequences.current = [];
      } else {
        toast.error("Server error", {
          position: "bottom-right",
          autoClose: 5000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        })
      }
    }
  }

};

const addImageAndTileSequence = (image, tileSequence) => {
  console.log("handle tile");
  sequences.current.push({
    image,
    tileSequence,
  });
  setImages([]);
  handleSubmit();
};
```

```
  return (
    <div>
      <Canvas
        modalIsOpen={showCaptcha}
        setIsOpen={setShowCaptcha}
        onResult={(captchaResult) => {
          if (captchaResult) {
            setIsHuman(true);
            handleSubmit();
          } else {
            toast.warning("Are you human? Please try again", {
              position: "bottom-right",
              autoClose: 5000,
              hideProgressBar: false,
              closeOnClick: true,
              pauseOnHover: true,
              draggable: true,
              progress: undefined,
            });
          }
          setShowCaptcha(false);
        }}
      />
      <div className="m-8 font-light flex justify-center text-center">
        <form className="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4 w-full
max-w-xs">
          <div className="mb-4">
            <label className="block text-gray-700 mb-2 text-lg"
htmlFor="email">
              Email
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              id="email"
```

```
              type="text"
              placeholder="Email"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
            />
          </div>

          <div className="flex items-center justify-center">

            {roundNumber === 0 && <button
              className="bg-blue-500 hover:bg-blue-700 text-white font-light
py-2 px-4 rounded focus:outline-none focus:shadow-outline"
              type="button"
              onClick={() => {
                if (roundNumber === 0 && !isHuman) {
                  setShowCaptcha(true);
                }
              }}
            >
              Login
            </button>}
          </div>
        </form>
      </div>
      <div>
            <p className="text-center ">{imageCaption}</p>
          </div>
      <ImageGrid
        imageURLs={images}
        thumbnails={images}
        addImageAndTileSequence={addImageAndTileSequence}
        numRounds={NUM_ROUNDS}
        numTiles={NUM_TILES}
      />
    </div>
```

```
  );
}

export default Login;
```

Register.jsx:

```
import React, { Fragment, useEffect, useState } from "react";
import { ImageGrid } from "../imageGrid";
import { createApi } from "unsplash-js";
import CryptoJS from "crypto-js";
import axios from "axios";
import { toast } from "react-toastify";
import { navigate } from "@reach/router";
import Canvas from "../canvas/Canvas";
import imageCaption from "./Story";

const unsplash = createApi({
  accessKey: process.env.REACT_APP_UNSPLASH_ACCESS_KEY,
  fetch: fetch,
});

// const NUM_TILES = Number(process.env.REACT_APP_NUM_TILES);
// const NUM_ROUNDS = Number(process.env.REACT_APP_NUM_ROUNDS);

const NUM_TILES = 4
const NUM_ROUNDS = 4

// console.log(NUM_TILES)

function hashImage(image, ref_point) {
  const str = image + ref_point.join();
  return CryptoJS.SHA256(str).toString(CryptoJS.enc.base64);
}

function encryptImage(image, key) {
```

```
  return CryptoJS.AES.encrypt(image, key).toString();
}

function Register() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [category, setCategory] = useState("");
  const [rawImages, setRawImages] = useState([]);
  const [thumbnails, setThumbnails] = useState([]);
  const [sequences, setSequences] = useState([]);
  const [roundNumber, setRoundNumber] = useState(0);
  const [isLoading, setIsLoading] = useState(true);
  const [showCaptcha, setShowCaptcha] = useState(false);
  const [isHuman, setIsHuman] = useState(false);

  const getImages = async () => {
    if (category.trim() === "") {
      toast.info("Please enter a category", {
        position: "bottom-right",
        autoClose: 5000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
      return;
    }
    setIsLoading(true);
    let fullImages = [];
    let thumbnails = [];

    try {
      const tempResult = await unsplash.search.getPhotos({
```

```
      query: category,
      page: 1,
      perPage: 1,
      orientation: "squarish",
    });

    const pic = tempResult.response;
    const totalImages = pic.total;
    const randomNumber = Math.floor(Math.random() * totalImages) + 1;
    const imagePage = Math.floor(randomNumber / 30) + 1;

    const result = await unsplash.search.getPhotos({
      query: category,
      page: imagePage,
      // perPage: Number(process.env.REACT_APP_PER_PAGE_IMAGE_LIMIT),
      perPage: 9,
      orientation: "squarish",
    });

    const pictures = result.response;
    pictures.results.forEach((pic) => {
      fullImages.push(`${pic.urls.full}&crop=faces&fit=crop&h=250&w=250`);
      thumbnails.push(`${pic.urls.thumb}&crop=faces&fit=crop&h=250&w=250`);
    });

    setIsLoading(false);
    setRawImages(fullImages);
    setThumbnails(thumbnails);
  } catch (err) {
    console.error(err.message);
  }
};

const addImageAndTileSequence = (image, tileSequence) => {
```

```
    setSequences((prev) => [
      ...prev,
      {
        image,
        tileSequence: tileSequence,
      },
    ]);
    setRoundNumber((prev) => prev + 1);
    setCategory("");
    setRawImages([]);
    setThumbnails([]);
  };

  const register = async () => {
    if (!name || !name.trim() || !email || !email.trim()) {
      toast.warning("Please enter your details", {
        position: "bottom-right",
        autoClose: 5000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
    }
    const hashes = [];
    let captions = [];
    sequences.map((imageSelection) => {
      captions.push(imageCaption(imageSelection.image));
      hashes.push(hashImage(imageSelection.image,
imageSelection.tileSequence));
    });

    Promise.all(captions).then(async(values) => {
      console.log(values);
```

```
let captionList = [];
for(let i=0; i<values.length;i++){
  captionList.push(values[i].text);
}

const encryptedImages = [];

for (let i = 1; i < NUM_ROUNDS; i++) {
  encryptedImages.push(encryptImage(sequences[i].image, hashes[i - 1]));
}

const passwordHash = CryptoJS.SHA256(hashes.join()).toString(
  CryptoJS.enc.base64
);

try {
  const res = await axios.post("/register", {
    name,
    email,
    passwordHash,
    images: [sequences[0].image, ...encryptedImages],
    captions : captionList,
  });

  if (res.status === 200) {
    console.log("Registration successful");
    toast.success("Registration successful!", {
      position: "bottom-right",
      autoClose: 5000,
      hideProgressBar: false,
      closeOnClick: true,
      pauseOnHover: true,
      draggable: true,
      progress: undefined,
```

```
        });
        navigate("/login");
      }
    } catch (error) {
      if (error.reponse.status === 500) {
        toast.error("An error occured", {
          position: "bottom-right",
          autoClose: 5000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      } else {
        toast.error("Invalid login", {
          position: "bottom-right",
          autoClose: 5000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
        sequences.current = [];
        setRoundNumber(0);
      }
    }
  })

};

return (
  <Fragment>
```

```
      <Canvas
        modalIsOpen={showCaptcha}
        setIsOpen={setShowCaptcha}
        onResult={(captchaResult) => {
          if (captchaResult) {
            setIsHuman(true);
            getImages();
          } else {
            toast.warning("Are you human? Please try again", {
              position: "bottom-right",
              autoClose: 5000,
              hideProgressBar: false,
              closeOnClick: true,
              pauseOnHover: true,
              draggable: true,
              progress: undefined,
            });
          }
          setShowCaptcha(false);
        }}
      />
      <div className="mx-auto my-2 font-light flex justify-center text-center">
        <form className="bg-white pt-6 w-2/3 flex md:flex-row flex-col justify-
center">
          <div className="mb-4 mr-2">
            <label className="text-gray-700 mb-2 text-lg" htmlFor="name">
              Name
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              id="name"
              name="name"
              type="text"
              placeholder="Name"
```

```
            value={name}
            onChange={(e) => setName(e.target.value)}
          />
        </div>
        <div className="mb-4 mr-2">
          <label className="text-gray-700 mb-2 text-lg" htmlFor="email">
            Email
          </label>
          <input
            className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
            id="email"
            type="text"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            name="email"
          />
        </div>

        <div className="mb-4 mr-2">
          <label className="text-gray-700 mb-2 text-lg" htmlFor="category">
            Category for image
          </label>
          <input
            className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
            id="category"
            placeholder="Try 'cats'"
            value={category}
            onChange={(e) => setCategory(e.target.value)}
          />
        </div>
        <div className="flex items-center justify-center">
          <button
```

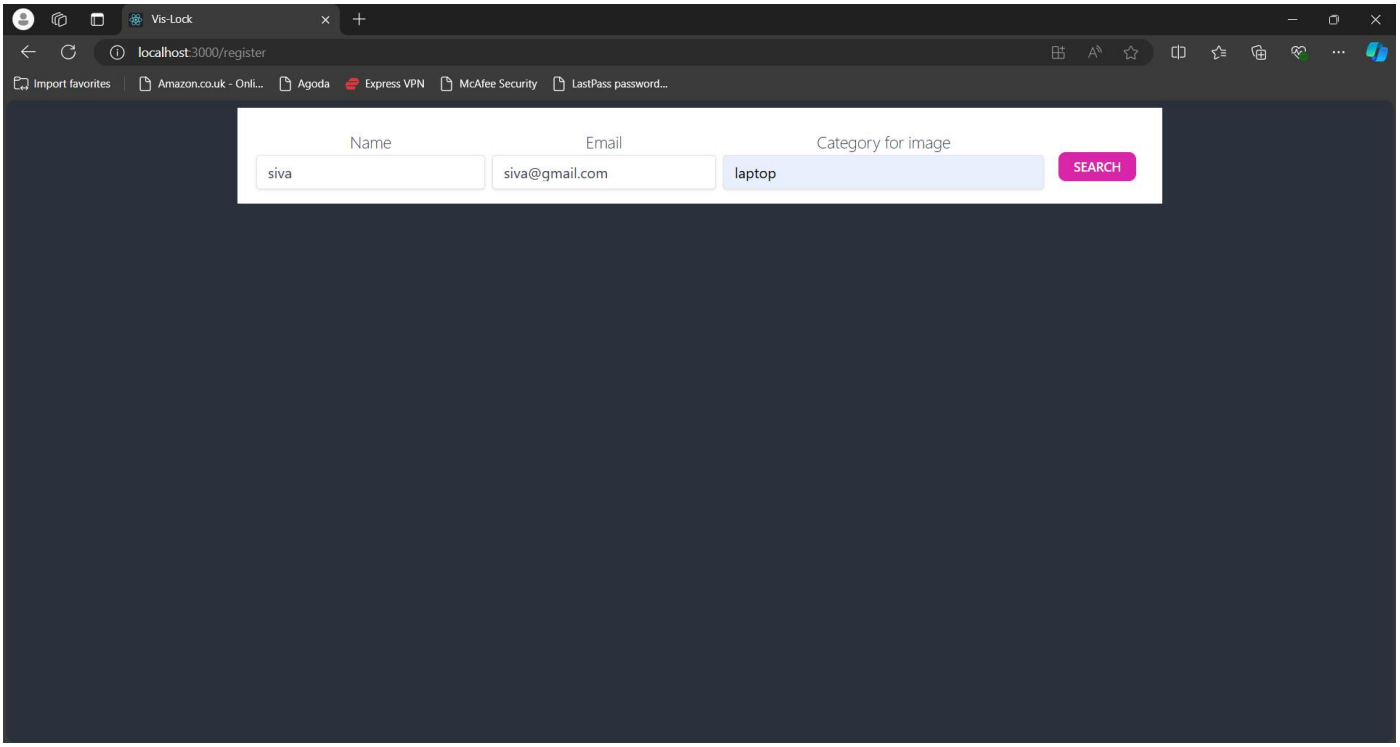
```
            className="mx-2 btn btn-sm btn-secondary py-2 px-4"
            type="button"
            onClick={() => {
              if (roundNumber === 0 && !isHuman) {
                setShowCaptcha(true);
              } else {
                getImages();
              }
            }}
          >
            Search
          </button>
        </div>
      </form>
    </div>
    {roundNumber === NUM_ROUNDS ? (
      <div className="flex flex-col">
        <p className="mx-auto text-3xl my-2">Woo-hoo! You're almost
there!</p>
        <button className="btn btn-sm btn-primary mx-auto"
onClick={register}>
          Confirm registration
        </button>
      </div>
    ) : (
      <ImageGrid
        imageURLs={rawImages}
        thumbnails={thumbnails}
        addImageAndTileSequence={addImageAndTileSequence}
        numRounds={NUM_ROUNDS}
        numTiles={NUM_TILES}
        isLoading={isLoading}
      />
    )}
  </Fragment>
```

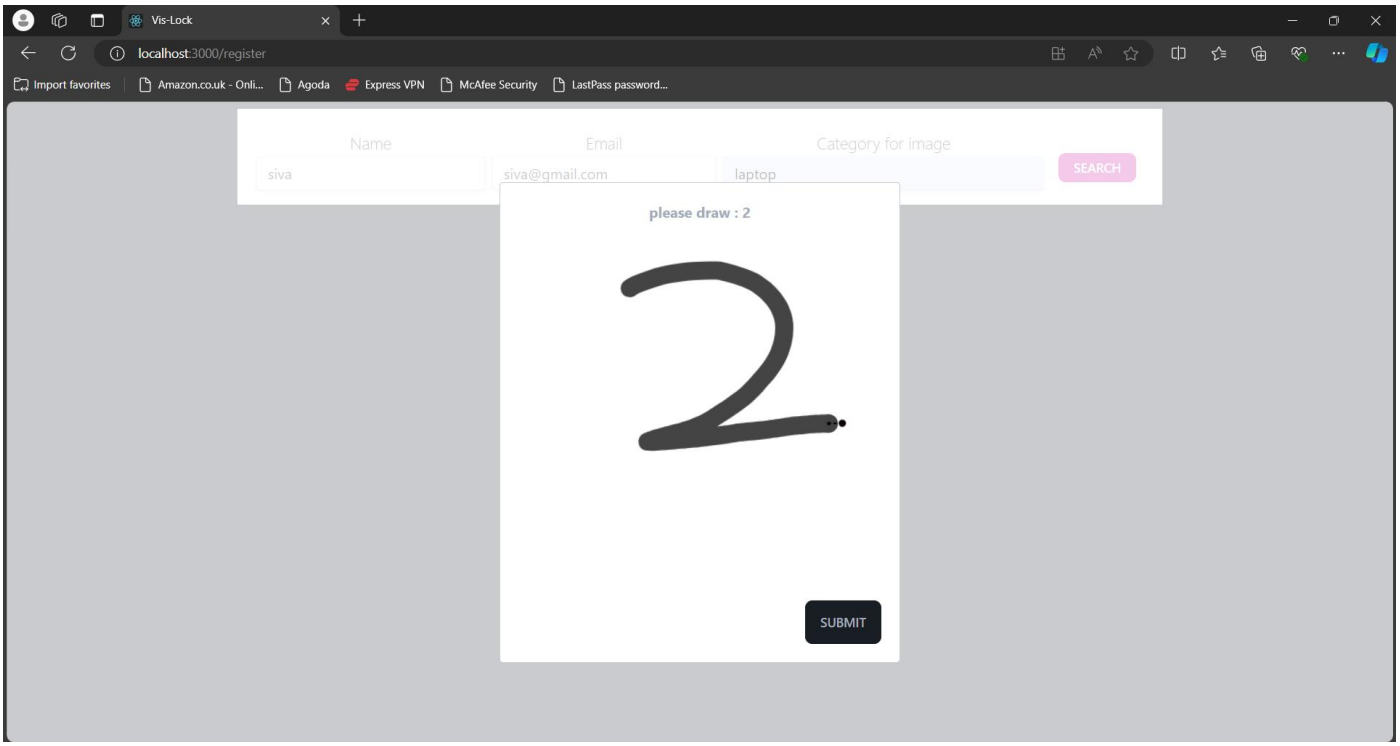
```
  );
}

export default Register;
```

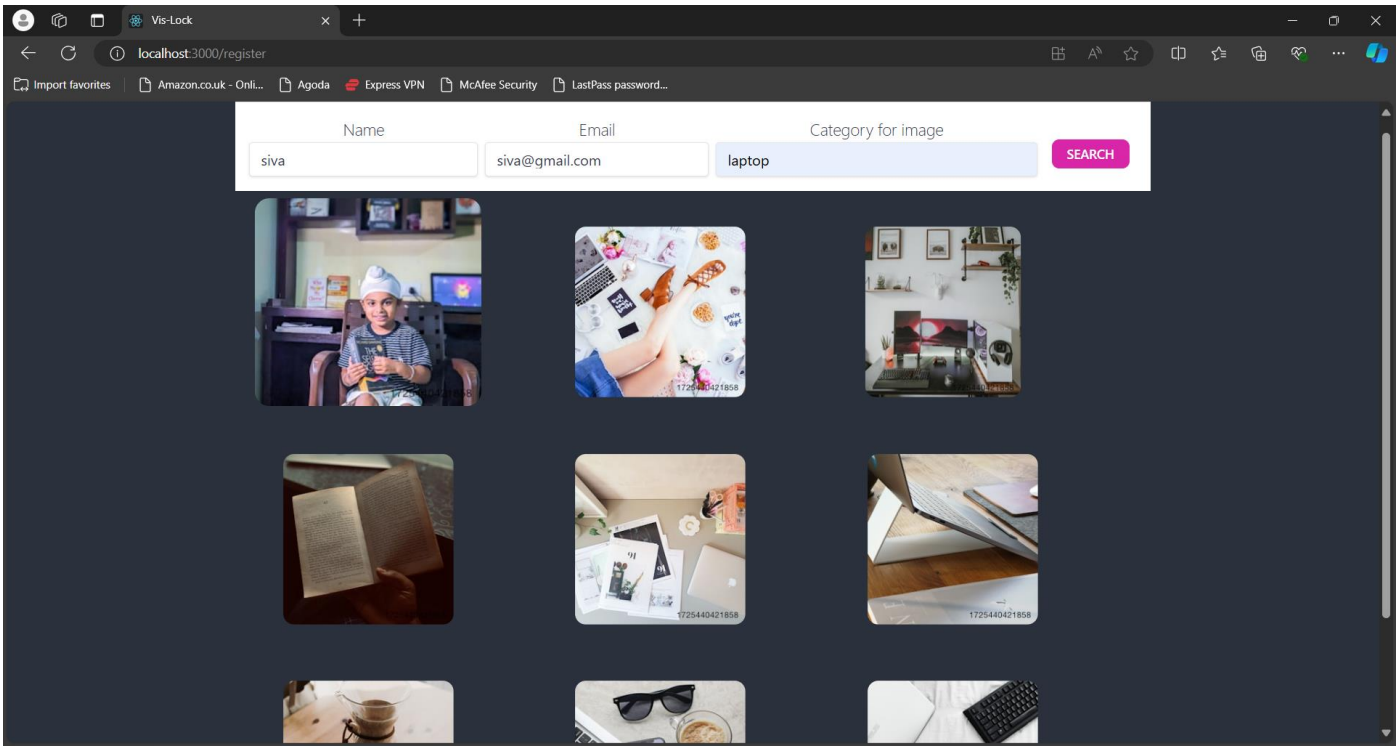
SAMPLE SCREENSHOTS

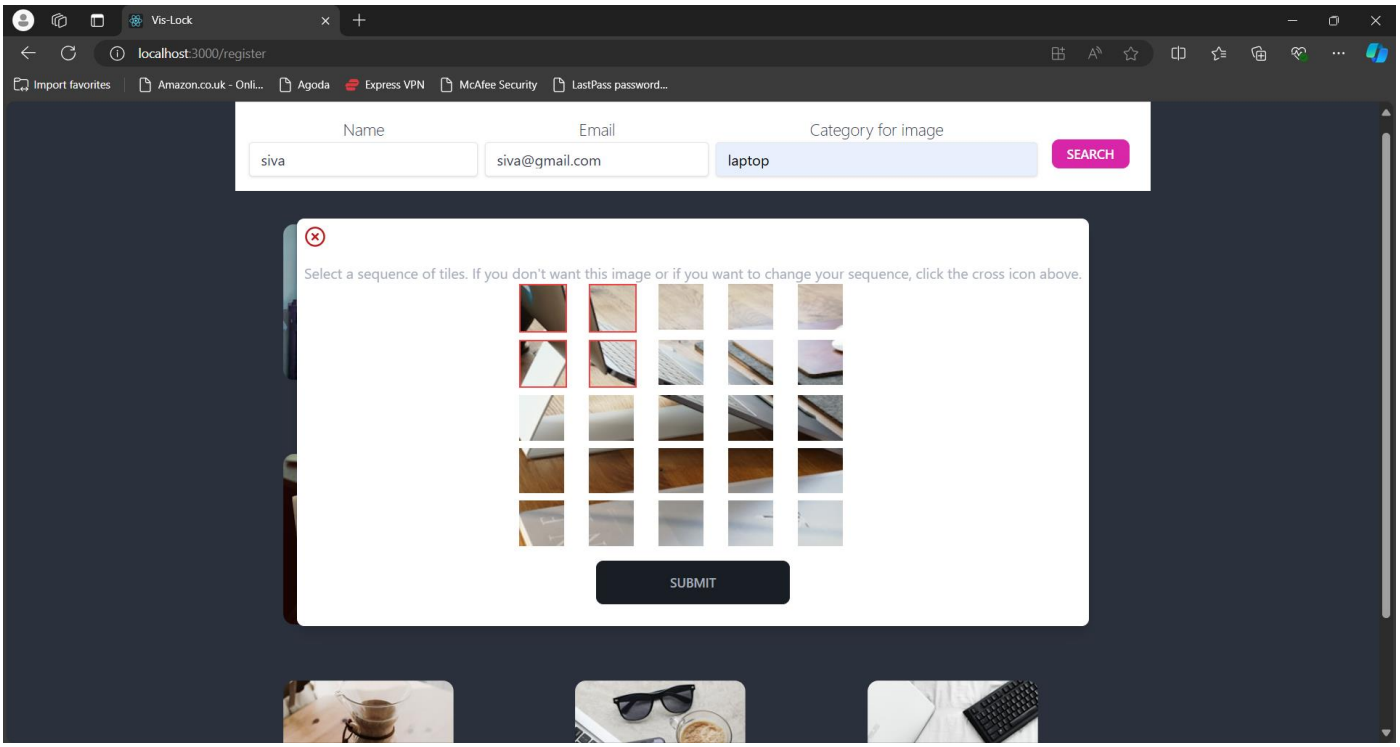
OUTPUT:

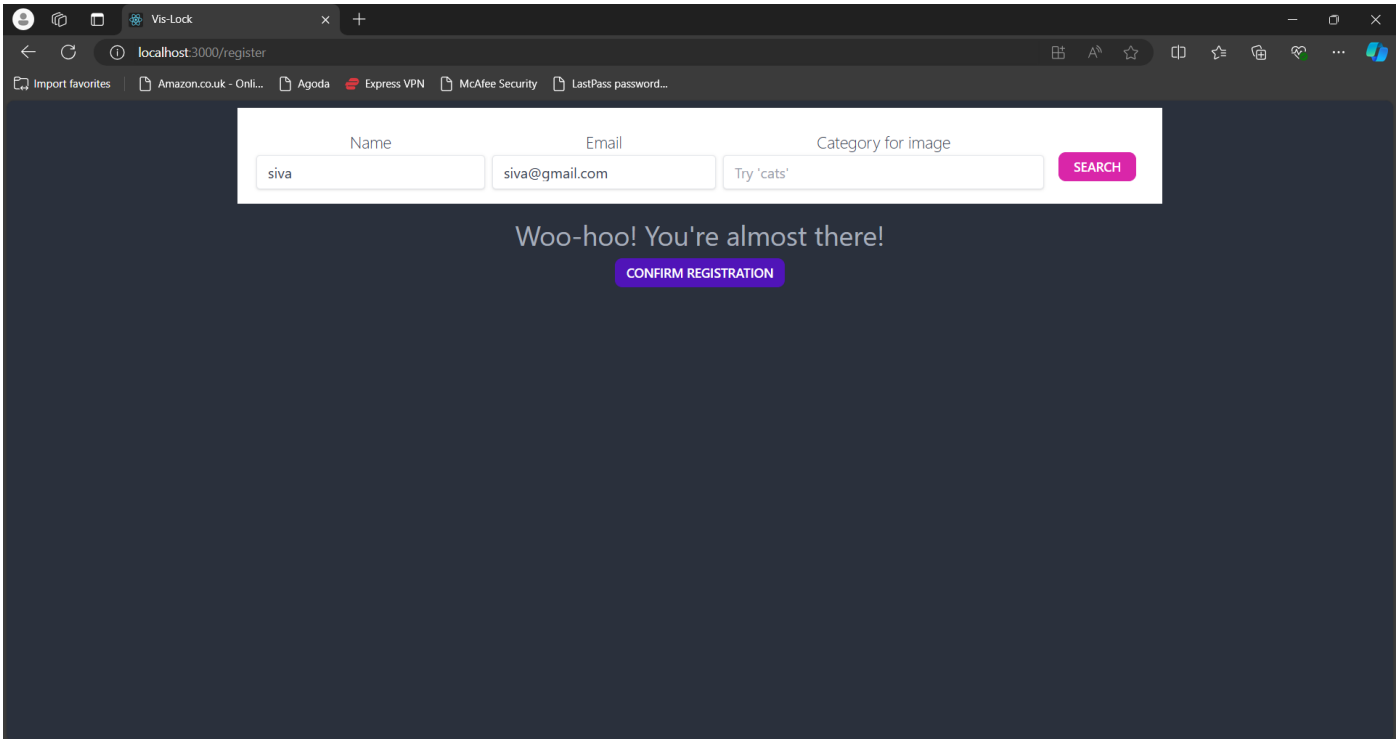
SIGN UP PAGE

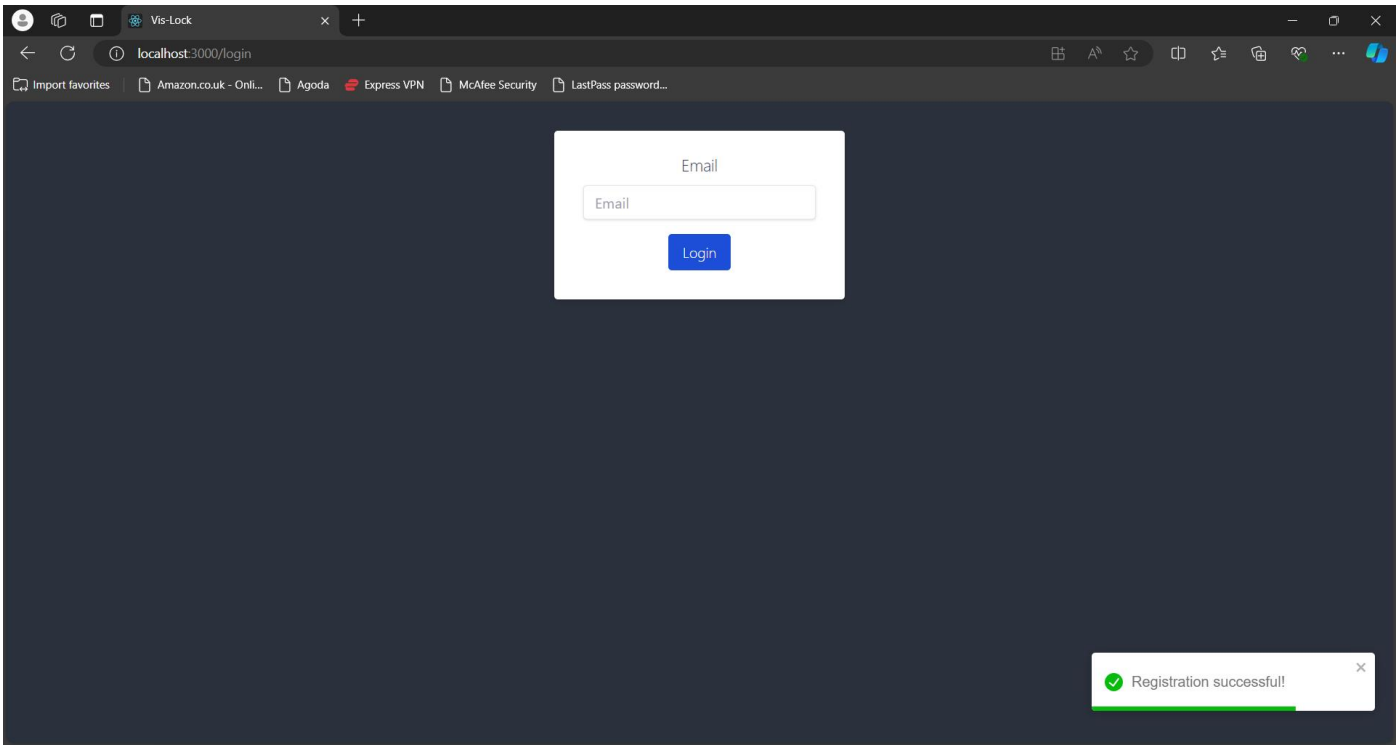




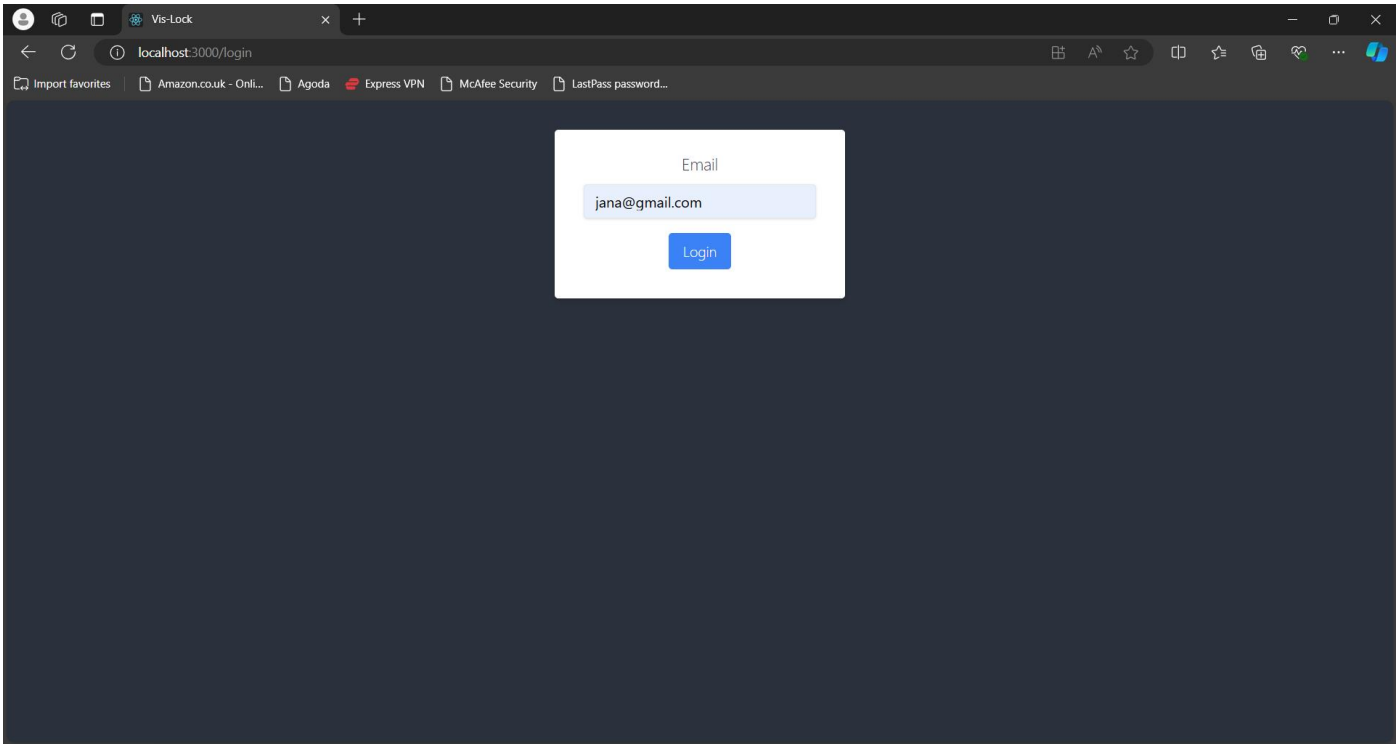

Vis-Lock
localhost:3000/register
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Name
Email
Category for image
siva
siva@gmail.com
laptop
SEARCH

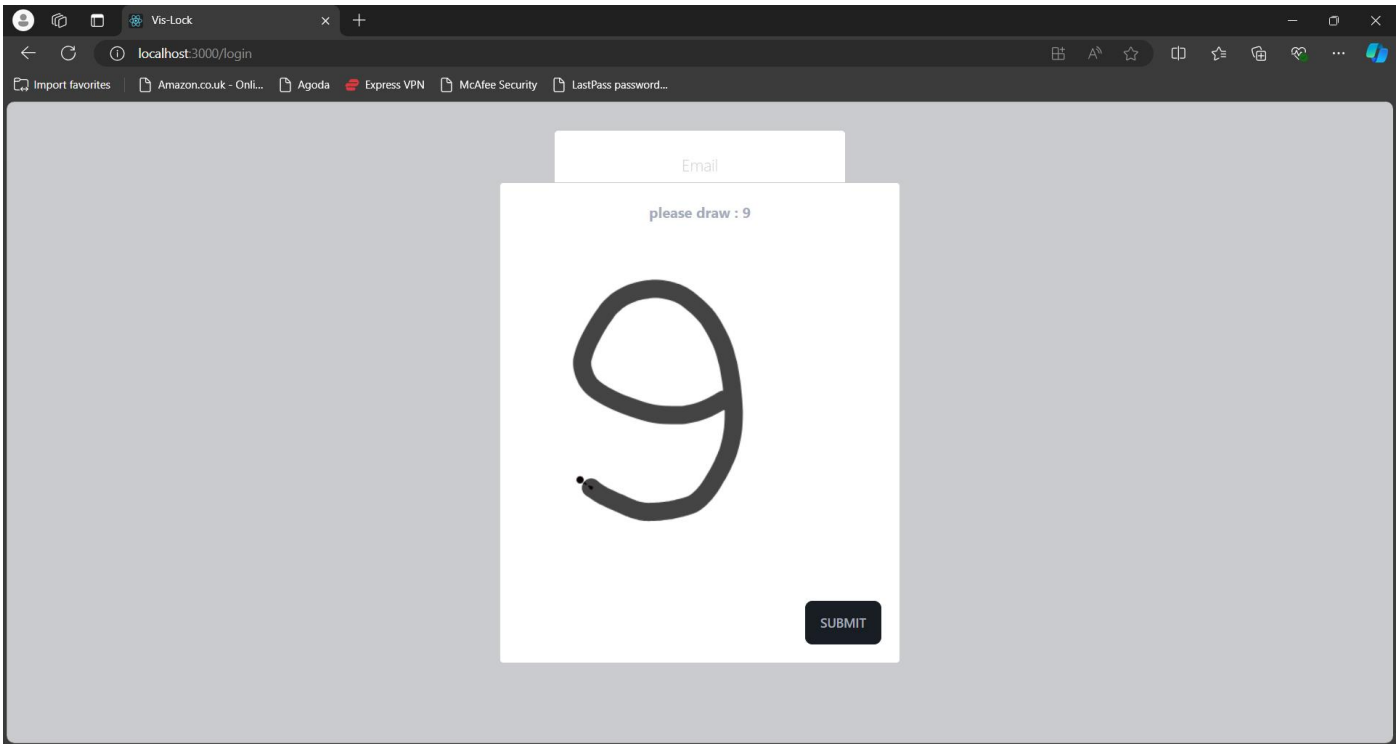

Vis-Lock
localhost:3000/register
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Name
Email
Category for image
siva
siva@gmail.com
laptop
SEARCH
Select a sequence of tiles. If you don't want this image or if you want to change your sequence, click the cross icon above.
SUBMIT

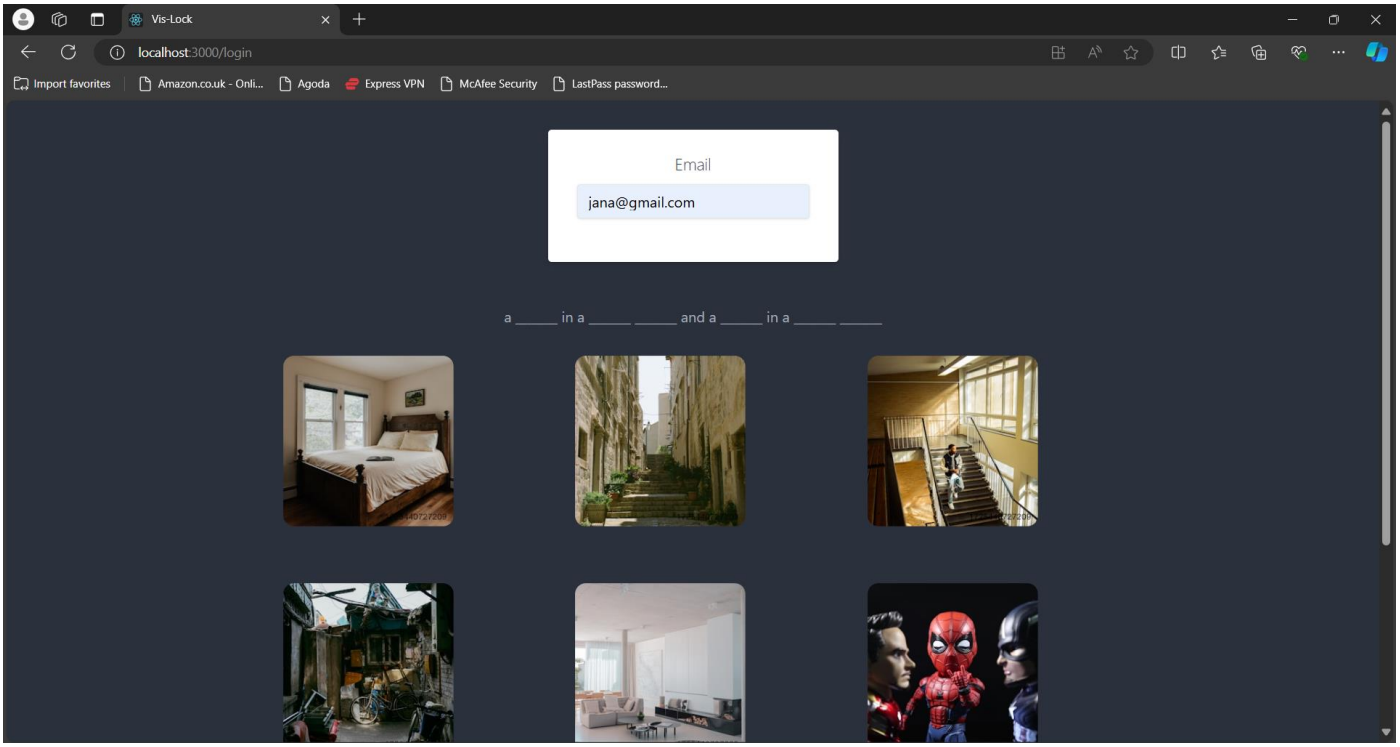

Vis-Lock
localhost:3000/register
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Name
Email
Category for image
siva
siva@gmail.com
Try 'cats'
SEARCH
Woo-hoo! You're almost there!
CONFIRM REGISTRATION

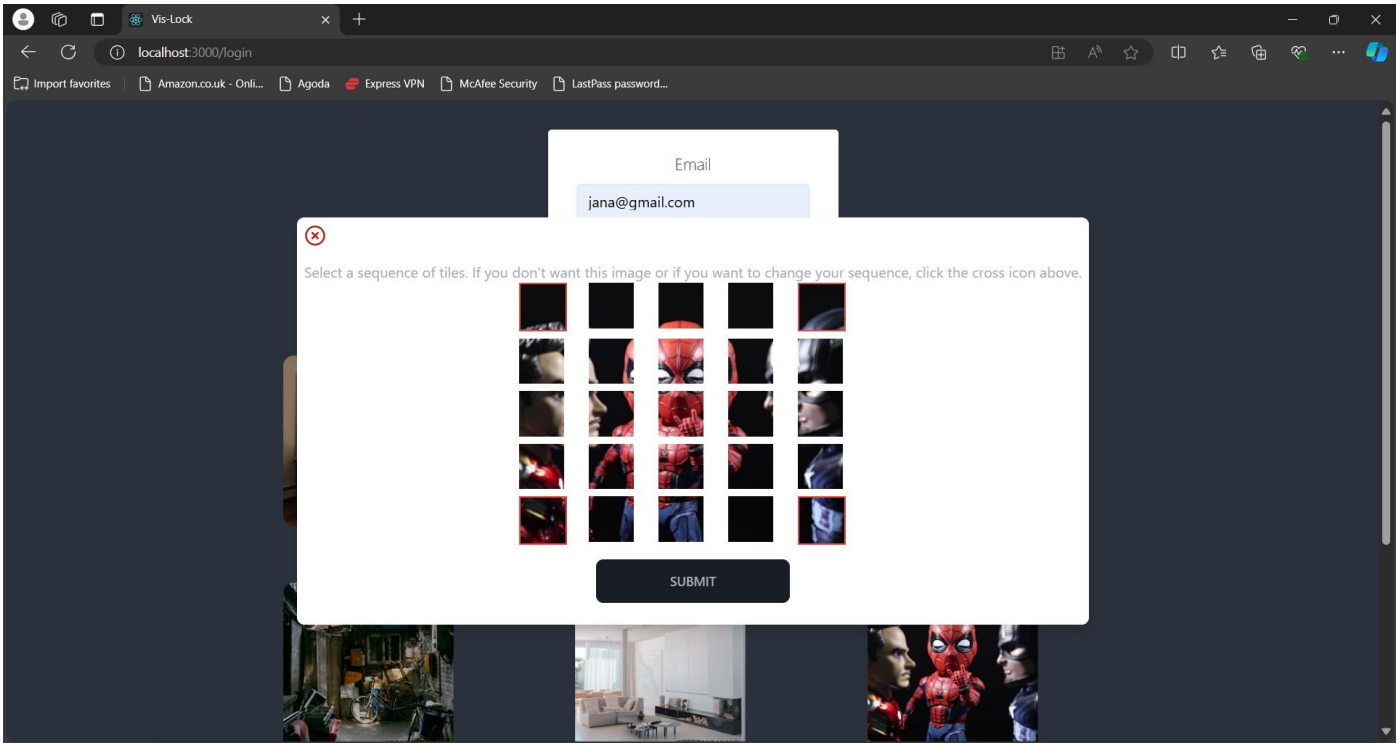

Vis-Lock
localhost:3000/login
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Email
Email
Login
Registration successful!

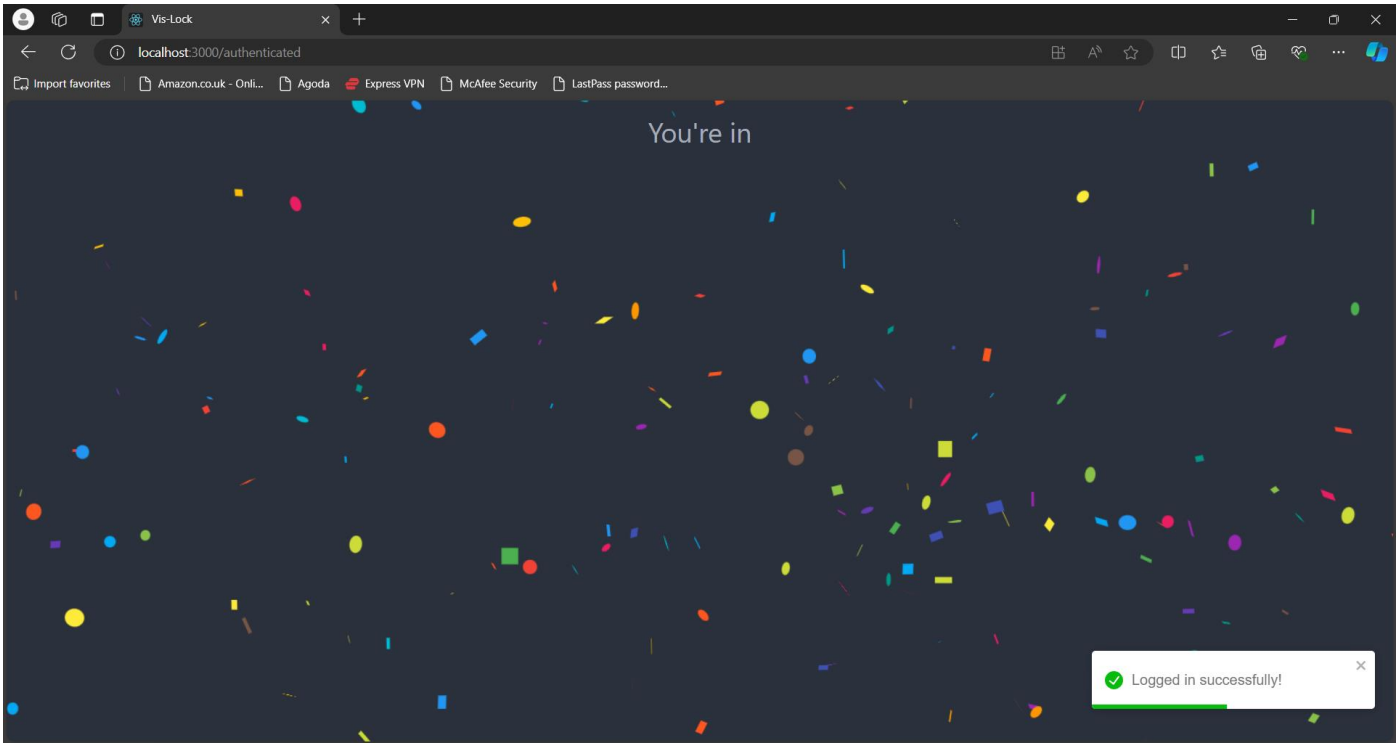
SIGN IN PAGE

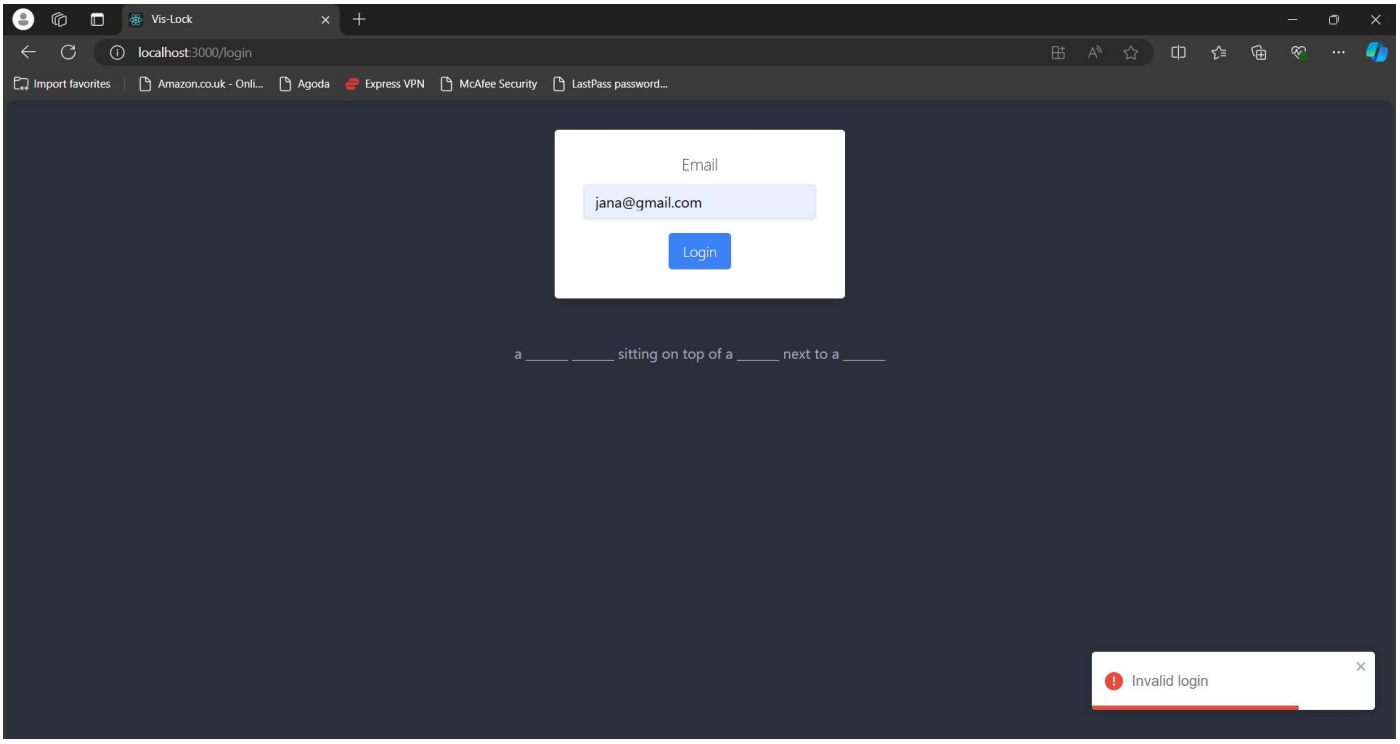

Vis-Lock
localhost:3000/login
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Email
jana@gmail.com
Login


Vis-Lock
localhost:3000/login
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Email
please draw : 9
SUBMIT


Vis-Lock
localhost:3000/login
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Email
jana@gmail.com
a _____ in a _____ _____ and a _____ in a _____ _____


Vis-Lock
localhost:3000/login
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Email
jana@gmail.com
Select a sequence of tiles. If you don't want this image or if you want to change your sequence, click the cross icon above.
SUBMIT


Vis-Lock
localhost:3000/authenticated
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
You're in
Logged in successfully!


Vis-Lock
localhost:3000/login
Import favorites
Amazon.co.uk - Onli...
Agoda
Express VPN
McAfee Security
LastPass password...
Email
jana@gmail.com
Login
a _____ _____ sitting on top of a _____ next to a _____
Invalid login

10. RESULTS AND PERFORMANCE ANALYSIS:

The proposed graphical password authentication system has undergone thorough testing and analysis to evaluate its effectiveness in tackling common security issues faced by both text-based and graphical password systems. This graphical password system expands the key space by offering a wide range of image and tile combinations, thanks to the extensive image selection made possible by the Unsplash API. This larger key space significantly diminishes the efficacy of brute force attacks compared to traditional text-based systems.

The system enhances security by implementing an invisible cursor during password entry and a blinking border around the image tiles. These measures have proven effective in preventing over-the-shoulder attacks, thereby reducing the risk of unauthorized observation of the password entry process.

Additionally, the integration of Image-based CAPTCHA and Puzzle CAPTCHA improves the system's ability to differentiate between human users and automated bots. The mouse-drawn CAPTCHA test has shown to be more difficult for bots to circumvent. Performance tests indicate that the CAPTCHA system effectively filters out bots while remaining user-friendly for legitimate users.

The use of encryption, modeled after cipher block chaining (CBC), ensures that the selected image sequences are securely stored. Incorrect sequences prevent the display of subsequent images, thereby protecting the integrity of the password information. Argon2, utilized for hashing the image sequences, provides robust protection against brute force and dictionary attacks.

Password fatigue is alleviated as users no longer need to remember complex alphanumeric combinations. Instead, they recall a sequence of images or gestures, which is easier to remember and less prone to fatigue. Although the CAPTCHA system introduces a slight increase in verification time (about 2-3 seconds longer than traditional image CAPTCHAs), users found it to be intuitive and appreciated its visual interaction after a brief learning period.

The Unsplash API integration for image retrieval incurs minimal performance overhead. During tests, images loaded within 1-2 seconds, depending on network conditions. Argon2 hashing with salts effectively guards against rainbow table and dictionary attacks. The strong

computational demands of Argon2, combined with the unique nature of graphical passwords, make it highly resistant to attacks using precomputed tables.

Despite its effectiveness in mitigating shoulder surfing, the system remains vulnerable to screen capture attacks, where an attacker might record the image sequence. Future updates could include additional obfuscation techniques, such as dynamic image distortion, to further reduce the risk of such attacks.

Overall, the graphical password authentication system effectively addresses many of the weaknesses of traditional password systems, maintaining a balance between security, usability, and performance.

11. CONCLUSION AND FUTURE WORK

Over the past decade, there has been growing interest in graphical passwords as an alternative to traditional text-based passwords. Current graphical password techniques can be broadly categorized into two types: recognition-based and recall-based. While the primary argument in favor of graphical passwords is that they are easier for people to remember compared to text-based passwords, existing studies are limited, and conclusive evidence is still lacking. However, preliminary analysis suggests that graphical passwords are harder to crack using traditional attack methods such as brute force, dictionary attacks, or spyware.

The proposed graphical password authentication system successfully addresses many vulnerabilities found in conventional password systems, maintaining a balance between security and usability. It introduces innovative security measures like invisible cursors, CAPTCHA-based human verification, and robust hashing algorithms. While improvements can be made, particularly in addressing screen capture attacks and optimizing for mobile use, the system offers a secure and user-friendly alternative to text-based authentication.

At present, the system has been developed as a desktop application only. Given that mobile devices and tablets are now the dominant platforms for app usage, future development will focus on making the system responsive for these devices. Currently, images are sourced from the Unsplash API, which presents a vulnerability to phishing and social engineering attacks. In future iterations, custom images with specific patterns or formats will be used to prevent replication and copying. Additionally, advanced techniques such as image distortion and dynamic grid layouts could be introduced to further increase the complexity of visually decoding password sequences, thereby strengthening the system's defense against visual attacks and unauthorized access.

12. REFERENCES

1. Bhanushali, A., Mange, B., Vyas, H., Bhanushali, H., & Bhogle, P. (2015). Comparison of graphical password authentication techniques. International Journal of Computer Applications, 116(1).
2. Lashkari, A. H., Farmand, S., Zakaria, D. O. B., & Saleh, D. R. (2009). Shoulder surfing attack in graphical password authentication. arXiv preprint arXiv:0912.0951.
3. Sarohi, H. K., & Khan, F. U. (2013). Graphical password authentication schemes: current status and key issues. International Journal of Computer Science Issues (IJCSI), 10(2 Part 1), 437.
4. Gokhale, M. A. S., & Waghmare, V. S. (2016). The shoulder surfing resistant graphical password authentication technique. Procedia Computer Science, 79, 490-498.
5. ArunPrakash, M., & Gokul, T. R. (2011, February). Network security-overcome password hacking through graphical password authentication. In 2011 National Conference on Innovations in Emerging Technology (pp. 43-48). IEEE.
6. Juneja, K. (2020). An XML transformed method to improve effectiveness of graphical password authentication. Journal of King Saud University-Computer and Information Sciences, 32(1), 11-23.
7. Sunil, S. S., Prakash, D., & Shivaji, Y. R. (2014). Cued click points: Graphical password authentication technique for security. International Journal of Computer Science and Information Technologies, 5(2).
8. Lashkari, A. H., Abdul Manaf, A., Masrom, M., & Daud, S. M. (2011). Security evaluation for graphical password. In Digital Information and Communication Technology and Its Applications: International Conference, DICTAP 2011, Dijon, France, June 21-23, 2011. Proceedings, Part I (pp. 431-444). Springer Berlin Heidelberg.
9. Gyorffy, J. C., Tappenden, A. F., & Miller, J. (2011). Token-based graphical password authentication. International Journal of Information Security, 10, 321-336.
10. Rajarajan, S., Prabhu, M., Palanivel, S., & Karthikeyan, M. P. (2014). GRAMAP: Three stage graphical password authentication scheme. Journal Of Theoretical & Applied Information Technology, 61(2).