**CA2/CAP776/DE328**

**Note: Project allocation is given in the excel sheet shared with you.**

**Date of allocation:25th sept.**

**Date of submission:3rd oct.(no further extension will be given)**

**Problem statements**

**Project Title1: News Headlines Console Screen with Login and API Integration**

**Problem Statement:**

Create a Python-based console application where users need to log in to access the top 5 news headlines based on a keyword of their choice. The user credentials should be stored in a CSV file, and a password reset option should be available. The application should limit tailed login attempts to prevent unauthorized access. Upon successful login, the app will fetch and display news headlines from **NewsAPI**.

**Requirements:**

1. **Login System:**
   * User credentials are stored in a CSV file (regno.csv). The file contains fields like email, password (hashed), and security\_question (for password recovery).
   * During login, the system will prompt the user for their email and password.
2. **Input Validation:**
   * The email must be in a valid format (e.g., example@domain.com).
   * The password must meet a minimum length of 8 characters, contain at least one special character, and be validated against the stored password (use hashing for comparison).
3. **Forgot Password:**
   * If the user chooses to reset their password, they are prompted to enter their registered email.
   * If the email exists in the CSV file, the user must answer a security question correctly. Upon correct answer, they can set a new password.
4. **Login Attempts:**
   * The user is allowed up to 5 login attempts. After 5 failed attempts, the user is logged out, and further attempts are denied until the application restarts.
5. **API Integration (NewsAPI):**
   * After successful login, prompt the user to enter a keyword or topic of interest.
   * Use the **NewsAPI** to fetch the top 5 latest news headlines related to the keyword.
   * Display the headlines, along with the source, in the console.
6. **Error Handling:**
   * Handle cases where the API key is invalid or expired.
   * Handle network errors (e.g., no internet connection).
   * Handle cases where no news results are found for the given keyword.

**Project Title2: NASA Data Console Screen with Login and API Integration**

**Problem Statement:**

Create a Python-based console application where users need to log in to access space-related data using the **NASA API**. The user credentials should be stored in a CSV file, and a password reset option should be available. The application should limit tailed login attempts to prevent unauthorized access. Upon successful login, the app will fetch and display data from the NASA API.

**Requirements:**

1. **Login System:**
   * User credentials are stored in a CSV file (regno.csv). The file contains fields like email, password (hashed), and security\_question (for password recovery).
   * During login, the system will prompt the user for their email and password.
2. **Input Validation:**
   * The email must be in a valid format (e.g., example@domain.com).
   * The password must meet a minimum length of 8 characters, contain at least one special character, and be validated against the stored password (use hashing for comparison).
3. **Forgot Password:**
   * If the user chooses to reset their password, they are prompted to enter their registered email.
   * If the email exists in the CSV file, the user must answer a security question correctly. Upon correct answer, they can set a new password.
4. **Login Attempts:**
   * The user is allowed up to 5 login attempts. After 5 failed attempts, the user is logged out, and further attempts are denied until the application restarts.
5. **API Integration (NASA API):**

**Near Earth Object (NEO) Feed**

**Description:** Provides data about asteroids and other near-Earth objects (NEOs) tracked by NASA.

**Data Available:**

* + Name of the NEO
  + Close approach date
  + Estimated diameter (in meters)
  + Velocity (in km/h)
  + Miss distance (in kilometers)
  + Hazardous status (true/false)

**API** <https://api.nasa.gov/neo/rest/v1/feed?api_key=YOUR_API_KEY>

**NASA Solar System Dynamics (SSD) API**

* **Description:** Provides data on planets, moons, comets, asteroids, and other objects in the Solar System.
* **Data Available:**
  + Orbital elements (e.g., semi-major axis, eccentricity)
  + Physical parameters (e.g., diameter, mass)
  + Object classification (e.g., planet, dwarf planet)
  + Discovery date
  + Rotation period

**API** https://ssd-api.jpl.nasa.gov/

1. **Error Handling:**
   * Handle cases where the API key is invalid or expired.
   * Handle network errors (e.g., no internet connection).
   * Handle cases where no results are found (e.g., no Mars Rover photos available for the selected date).

**Project Title3: Air Quality Monitoring with Secure Login**

**Problem Statement**: Create a Python-based console application that allows users to log in and monitor the air quality index (AQI) for a location of their choice. The application should securely handle user credentials, provide password recovery options, and restrict login attempts to prevent unauthorized access. Upon successful login, the system should fetch and display real-time air quality data from a public API based on a user-specified city or location.

**Requirements:**

**1. Login System:**

**User Credentials:**

Store user credentials in a CSV file (regno.csv), containing fields such as email, hashed password, and security\_question. Use hashing (e.g., bcrypt) to securely store and compare passwords.

**Login Process:**

The system should prompt users to enter their registered email and password.

Authenticate users against the stored credentials in the CSV file.

**2. Input Validation:**

Email Validation:

Ensure the entered email is in a valid format (e.g., user@example.com).

Password Validation:

The password must meet the following criteria:

Minimum length of 8 characters.

At least one uppercase letter, one lowercase letter, one digit, and one special character.

Hash the password for secure comparison.

**3. Forgot Password:**

Provide a “forgot password” option on the login screen.

Password Reset Process:

The system prompts the user to enter their registered email.

If the email exists in the CSV file, the user must correctly answer the stored security question.

Upon answering the question correctly, allow the user to set a new password (validated against the same password criteria).

Update the CSV file with the new hashed password.

**4. Login Attempts:**

Attempt Limitation:

Limit login attempts to 5. After 5 failed attempts, deny further login attempts until the application is restarted.

Security Measures:

After each failed attempt, inform the user of how many attempts remain.

Upon exceeding the limit, lock out the user and exit the application.

**5. API Integration (Air Quality Data):**

API Usage:

After a successful login, prompt the user to enter a city name or location of interest.

Use the OpenWeather Air Pollution API (or any free air quality API) to fetch real-time AQI data.

Display the following air quality information for the specified location:

AQI (Air Quality Index): Numerical value representing the air quality.

Main Pollutants: Information about key pollutants (PM2.5, PM10, Ozone, etc.).

Health Recommendations: Based on AQI value, provide health advice (e.g., safe outdoor activities, mask-wearing, etc.).

**Project Title4: Sunset and Sunrise Times Console Application with Secure Login**

**Problem Statement:** Create a Python-based console application that allows users to log in and retrieve sunset and sunrise times for any city they specify. The application should include secure login functionality, password recovery options, and enforce a limit on login attempts. Upon successful login, the system will fetch and display sunset and sunrise times using a public API for a user-specified location.

**Requirements:**

**1. Login System:**

* **User Credentials:**
  + Store user credentials in a CSV file (regno.csv), containing fields such as email, hashed password, and security\_question.
  + Hash the password (e.g., using bcrypt) for secure storage and comparison.
* **Login Process:**
  + The user is prompted to enter their email and password.
  + Authenticate the credentials by comparing the input with the stored information in the CSV file.

**2. Input Validation:**

* **Email Validation:**
  + Ensure that the email is in a valid format (e.g., user@example.com).
* **Password Validation:**
  + The password must meet the following requirements:
    - Minimum of 8 characters.
    - At least one uppercase letter, one lowercase letter, one digit, and one special character.
  + Hash the input password for comparison with the stored hashed password.

**3. Forgot Password:**

* Provide a "forgot password" option on the login screen.
* **Password Reset Process:**
  + Prompt the user to enter their registered email.
  + If the email exists in the CSV file, ask them to correctly answer the security question stored with their profile.
  + Upon correctly answering the security question, allow them to reset the password, ensuring it follows the same password validation rules.
  + Update the CSV file with the newly hashed password.

**4. Login Attempts:**

* **Attempt Limitation:**
  + Limit the user to 5 login attempts. After 5 failed attempts, deny further attempts until the application is restarted.
* **Security Measures:**
  + Notify the user of how many attempts remain after each failed login.
  + Lock the user out and terminate the application upon exceeding the attempt limit.

**5. API Integration (Sunset and Sunrise Data):**

* **API Usage:**
  + After successful login, prompt the user to enter the city or location for which they want to retrieve sunset and sunrise times.
  + Use the **Sunrise-Sunset API** (or similar free API) to fetch sunset and sunrise data.
  + Display the following information:
    - **Sunrise Time:** The time the sun rises at the specified location.
    - **Sunset Time:** The time the sun sets at the specified location.
    - **Day Length:** The total duration of daylight.
    - **Solar Noon:** The time at which the sun is at its highest point in the sky.

**6. Error Handling:**

* **Invalid API Key:** Handle cases where the API key is invalid or expired, and show an appropriate error message.
* **Network Errors:** If there is no internet connection or the API cannot be reached, display a meaningful error message and suggest checking the connection.
* **Invalid Location:** If no data is available for the specified city, handle it gracefully by informing the user and suggesting they verify the location name or try another location.
* **Time Zone Considerations:** Ensure the displayed times are adjusted to the user’s local time zone (if required).

**Project Title5: IP Geolocation Console Application with Secure Login**

**Problem Statement:** Create a Python-based console application that allows users to log in and retrieve geolocation data based on their IP address or a specified IP. The application should include a secure login system, password recovery options, and a limit on login attempts to prevent unauthorized access. Upon successful login, the system will fetch and display geolocation information using an IP Geolocation API for the user’s IP address or any IP they choose to query.

**Requirements:**

**1. Login System:**

* **User Credentials:**
  + Store user credentials in a CSV file (regno.csv) with fields like email, hashed password, and security\_question.
  + Use password hashing (e.g., bcrypt) for secure storage and comparison of passwords.
* **Login Process:**
  + The user is prompted to enter their email and password.
  + Authenticate the entered credentials by comparing them with the stored data in the CSV file.

**2. Input Validation:**

* **Email Validation:**
  + Ensure that the entered email follows a valid format (e.g., user@example.com).
* **Password Validation:**
  + The password must adhere to the following rules:
    - At least 8 characters in length.
    - Contains one uppercase letter, one lowercase letter, one number, and one special character.
  + The password input is hashed for secure comparison with the stored hashed password.

**3. Forgot Password:**

* Provide a "forgot password" option on the login screen.
* **Password Reset Process:**
  + Prompt the user to enter their registered email.
  + If the email exists in the CSV file, ask them to correctly answer the security question stored with their profile.
  + Upon answering the question correctly, allow the user to reset the password, ensuring it meets the password validation criteria.
  + Update the CSV file with the new hashed password.

**4. Login Attempts:**

* **Attempt Limitation:**
  + Allow the user up to 5 login attempts. After 5 failed attempts, prevent further login attempts until the application is restarted.
* **Security Measures:**
  + After each failed attempt, notify the user of how many attempts remain.
  + Upon exceeding the limit, lock the user out and exit the application.

**5. API Integration (IP Geolocation):**

* **API Usage:**
  + After a successful login, prompt the user to enter an IP address or choose to use their own IP for geolocation data.
  + Use a free IP Geolocation API (such as **IP-API** or **ipstack**) to fetch geolocation data.
  + Display the following geolocation information:
    - **Country:** The country where the IP is located.
    - **City:** The city of the IP.
    - **Region:** The region or state.
    - **Latitude and Longitude:** The geographical coordinates.
    - **Timezone:** The time zone of the location.
    - **ISP (Internet Service Provider):** The ISP associated with the IP.
* **Sample Response Data:**
  + Country: United States
  + City: New York
  + Region: New York
  + Latitude: 40.7128
  + Longitude: -74.0060
  + Timezone: America/New\_York
  + ISP: Comcast Cable

**6. Error Handling:**

* **Invalid IP Address:** Handle cases where the entered IP address is invalid and notify the user.
* **Network Errors:** If there's no internet connection or the API is unreachable, display a meaningful error message and suggest checking the connection.
* **Invalid API Key (if required):** Handle scenarios where the API key is missing or expired, and display an appropriate error message.
* **No Data Found:** Handle cases where no geolocation data is available for the entered IP, and suggest checking the input IP address or trying another one.

**Project Title6: Stock Market Data Console Application with Secure Login**

**Problem Statement:** Create a Python-based console application that allows users to log in and retrieve stock market data for a company of their choice. The application should include a secure login system, password recovery options, and a limit on login attempts. Upon successful login, the system will fetch and display real-time stock data using a public Stock Market Data API for the user’s specified company or ticker symbol.

Requirements:

**1. Login System:**

**User Credentials:**

Store user credentials in a CSV file (regno.csv) containing fields like email, hashed password, and security\_question.

Hash the password (e.g., using bcrypt) for secure storage and comparison.

**Login Process:**

The user is prompted to enter their registered email and password.

Authenticate the entered credentials by comparing them with the stored data in the CSV file.

**2. Input Validation:**

**Email Validation:**

Ensure the entered email follows a valid format (e.g., user@example.com).

**Password Validation:**

The password must meet the following criteria:

At least 8 characters in length.

Contains one uppercase letter, one lowercase letter, one number, and one special character.

The password input should be hashed for secure comparison with the stored hashed password.

**3. Forgot Password:**

Provide a "forgot password" option on the login screen.

Password Reset Process:

Prompt the user to enter their registered email.

If the email exists in the CSV file, ask them to correctly answer the security question stored with their profile.

Upon answering the security question correctly, allow the user to reset the password, ensuring it meets the same password validation criteria.

Update the CSV file with the new hashed password.

**4. Login Attempts:**

**Attempt Limitation:**

Limit the user to 5 login attempts. After 5 failed attempts, deny further login attempts until the application is restarted.

**Security Measures:**

Notify the user of how many attempts remain after each failed login.

Upon exceeding the limit, lock the user out and terminate the application.

**5. API Integration (Stock Market Data):**

**API Usage:**

After a successful login, prompt the user to enter the ticker symbol or company name for which they want to retrieve stock data.

Use a free Stock Market Data API (such as Alpha Vantage, IEX Cloud, or Yahoo Finance API) to fetch real-time stock data.

Display the following stock market information:

Current Price: The latest price of the stock.

Open Price: The stock price at the market open.

High Price: The highest price of the stock for the day.

Low Price: The lowest price of the stock for the day.

Previous Close: The stock's closing price from the previous day.

Volume: The total number of shares traded during the day.

**6. Error Handling:**

Invalid Ticker Symbol: Handle cases where the entered ticker symbol is invalid or doesn't exist, and notify the user.

Network Errors: If there's no internet connection or the API cannot be reached, display a meaningful error message and suggest checking the connection.

Invalid API Key (if required): Handle scenarios where the API key is missing, invalid, or expired, and display an appropriate error message.

No Data Found: Handle cases where no stock data is available for the entered ticker symbol or company, and suggest verifying the input or trying another stock.

**Project Title7: Digital PC Game Price Comparison Console Application with Secure Login (Using CheapShark API)**

Problem Statement: Create a Python-based console application where users can log in and search for the latest prices of digital PC games from multiple stores (such as Steam, GreenManGaming, and Fanatical) using the CheapShark API. The application should include a secure login system, a password recovery option, and a limit on login attempts. After successful login, users can search for the latest deals on a specific game and view pricing data from various stores.

Requirements:

1. Login System:

* User Credentials:
  + Store user credentials in a CSV file (users.csv) with fields like email, hashed password, and security\_question.
  + Use password hashing (e.g., bcrypt) for secure storage and comparison of passwords.
* Login Process:
  + The user is prompted to enter their registered email and password.
  + The system validates the credentials by comparing them with the data stored in the CSV file.

2. Input Validation:

* Email Validation:
  + Ensure that the entered email follows a valid format (e.g., user@example.com).
* Password Validation:
  + The password must meet the following criteria:
    - At least 8 characters in length.
    - Contains one uppercase letter, one lowercase letter, one number, and one special character.
  + The password input should be hashed for secure comparison with the stored hashed password.

3. Forgot Password:

* Provide a "forgot password" option on the login screen.
* Password Reset Process:
  + Prompt the user to enter their registered email.
  + If the email exists in the CSV file, ask the user to answer their security question correctly.
  + Upon answering the security question correctly, allow the user to reset the password (ensuring it meets the validation criteria).
  + Update the CSV file with the new hashed password.

4. Login Attempts:

* Attempt Limitation:
  + Limit the user to 5 login attempts. After 5 failed attempts, prevent further login attempts until the application is restarted.
* Security Measures:
  + Notify the user of how many login attempts remain after each failed login.
  + If the limit is exceeded, lock the user out and terminate the application.

5. API Integration (CheapShark API):

* API Usage:
  + After a successful login, prompt the user to enter the name of a game for which they want to search for price comparisons.
  + Use the CheapShark API to fetch the latest prices and deals from various digital game stores like Steam, GreenManGaming, and Fanatical.
  + Display the following data for the game:
    - Game Title: The name of the game.
    - Store Name: The store offering the game (e.g., Steam, Fanatical).
    - Normal Price: The regular price of the game.
    - Sale Price: The discounted price of the game (if available).
    - Savings: The percentage of savings compared to the normal price.
    - Deal Rating: A score representing how good the deal is.
    - Link to Store: URL to the store where the game is available.