Seth Sutton

Team Member Information

* + suttons7@mymail.nku.edu

Stage B

Login Security

Feature 1: reCaptcha

Rationale

Goal:  
- Enhance login security by implementing reCAPTCHA to prevent automated bots from accessing the Wiki System.

Technology Required:  
- pip install Flask-WTF Flask

-[https://www.google.com/recaptcha/admin/Links to an external site.](https://www.google.com/recaptcha/admin/)

[Links to an external site.](https://www.google.com/recaptcha/admin/)- Flask framework for rendering templates, validators and handling user requests.

Certainly, here are the implementation steps for your registration form with reCAPTCHA:

Implementation Steps:

1. Import Librarys  
   -  pip install Flask-WTF Flask

2. Form Creation:  
   - Create a registration form using Flask-WTF, with fields for name, password, email, and a reCAPTCHA field.  
   - Apply validation rules to form fields, such as data requirements and length restrictions. (This step will be done once our sign up page is created)

3. reCAPTCHA Configuration:  
   - Configure reCAPTCHA settings for your Flask application, including the public and private keys obtained from the reCAPTCHA dashboard found in google.

   - Acquire site key and secret key

4. Render the Registration Page:  
   - Render the signup page, passing the form as a context variable for rendering.

5. User Registration Form Submission:  
   - Handle HTTP POST requests to the registration route. example:

@app.route("/", methods=['GET','POST'])

   - Check if the form data is valid by using the `validate\_on\_submit()` method.

6. reCAPTCHA Verification:  
   - Ensure that reCAPTCHA verification is integrated with the form submission.  
   - If the form is successfully validated and reCAPTCHA is passed, proceed with user registration.

7. Success Message:  
   - If both form validation and reCAPTCHA verification are successful, display a success message to the user, indicating that their registration was successful.

8. Unsuccessful:  
   - If the form is not yet submitted or contains errors, render the registration form again, allowing the user to correct any errors and attempt registration again.

9. Run the Flask Application:  
   - Start the Flask application with debugging enabled to test the registration form and reCAPTCHA integration.

Requirements

* As a user, I want to encounter a reCAPTCHA challenge during the sign in process to ensure that I am not a bot.
* As a user, I want the reCAPTCHA challenge to be user-friendly and not excessively complicated.
* As a user, I want to be alerted if I entered the incorrect information and be able to keep trying until I enter the correct information.

Deliverables

* Design:
  + A screenshot of a computer

    Description automatically generated
* code:
  + https://github.com/lschwartz15/440project
* test:
  + https://github.com/lschwartz15/440project
* documents:
  + https://github.com/lschwartz15/440project

Test Plan

Unit tests:

* Test checkbox is checked or unchecked
* Test random generation of matching game every time a user registers.

Feature 2: MFA (Multi-Factor Authentication)

Rationale

Goal:  
- Enhance security for user logins by implementing Multi-Factor Authentication (MFA).

Technology Required:  
- Authenticator app on user's phone.  
- Phone with a functioning camera to scan QR codes.  
- Python library pyotp (Python One-Time Password Library) for generating 6-digit tokens.  
- qrcode library for generating QR code images.

-flask tools for validators and text fields

Implementation Steps:  
1. Develop the User Registration Process:  
   - During user registration, generate a random secret key for each user.  
   - Create a QR code provisioning URL using `pyotp` for the user.  
   - Generate a QR code image with the URL using the `qrcode` library.  
   - Save the QR code image and the user's secret key.

2. User Setup:  
   - After registration, instruct the user to download an authenticator app on their phone (e.g., Google Authenticator, Authy).  
   - Prompt the user to scan the QR code image with the authenticator app using their phone's camera.  
   - The authenticator app saves the secret key and generates time-based 6-digit tokens.

3. Token Generation and Verification:  
   - Use `pyotp` to generate a 6-digit token based on the user's secret key.  
   - The token changes every 30 seconds as per the time-based algorithm.  
   - When the user logs in, they enter the current token generated by their authenticator app.

4. User Access:  
   - Verify the entered token against the expected token using `pyotp`.  
   - Grant access if the tokens match.

5. Token Rotation:  
   - Every 30 seconds, a new token is generated by the authenticator app.  
   - The user is required to enter the current token to log in.

Requirements

* As a user, I want to scan a QR to add my token to my authenticator.
* As a user, I want the token to be on a timer to generate a new code every 30 seconds.
* As admin, user is required to have an authentication app and a working camera on their device.

Deliverables

* Design:
  + A screen shot of a computer program

    Description automatically generated
* code:
  + https://github.com/lschwartz15/440project
* test:
  + https://github.com/lschwartz15/440project
* documents:
  + https://github.com/lschwartz15/440project

Test Plan

Unit tests:

* Test token expiration timer
* Test the token generation
* Test user input valid/invalid
* Test the generation of the QR code image

Milestones and Deadline

* + **Deadline**: 11/6
  + Milestone 1
    - Date: 11/4
    - Goal: Draw diagrams or UML and create code for reCAPTCHA and MFA
  + **Deadline**: 11/13
  + Milestone 2
    - Date:
    - Goal: Create separate branch and implement code into our Riki system
  + **Deadline**: 11/20
  + Milestone 3
    - Date:
    - Goal: Testing MFA and reCaptcha security and outputs
  + **Deadline**: 11/27
  + Milestone 4
    - Date:
    - Goal: Merge into main branch and testing the functionality with other features.
  + **Deadline**: 12/4
  + Milestone 5
    - Date:
    - Goal: Add extra any needed features or provisions and practice for presentation

Risk Analysis

1. Added Complexity of Creating a user, since user requires a second device with an authenticator.
2. Extra Login Steps
3. Lockout Risk
4. Implementation these features in the system could potentially be difficult

Stage C: Progress

Feature Implementation

Week 1?

Created reCaptcha and MFA Prototypes

Summary:

I have made two applications:

1. reCaptcha:

* This application consist of an app.py and templates folder with a register.html
* In my app, I create a Simple RegisterForm that has a name, password, email, recaptcha, and submit button, which is all used for my register() method.
* With a Flask app that holds the configs of a secret key, recaptcha\_use\_SSL, recaptcha\_public\_key, recaptcha\_private\_key, and options for the theme
* The public keys and private keys come from [https://www.google.com/recaptcha/admin/Links to an external site.](https://www.google.com/recaptcha/admin/)
  + This provides us our checkbox with matching image challenge
* Uses decorator to route to our register.html
* The register.html has outputs the form of our registration.

2. MFA:

* This application consist of an app.py and templates folder that holds our login.html, mfa.html, and register.html
* In my app, I created another Registerform with Register, MFA, and Login methods
* Each method Renders our their pairing html
* In our mfa method once the user is registered, we have a secret key to create our tokens. Along with the name of user and issuer name.
* Also In our mfa we create our qr\_code image and path to that holds all our images for each different user made
* Once user scans qrcode they click a login button to get to our login page
* In our login page, I only put the Token Text Box because I only care if the token works not the password or username

Also created the UML diagrams to show how the data flow will work with the applications

Milestones or risks in this week

Achieved my Milestone 1.

Week 2

Summary

Modified and attempted to implement the MFA feature to our Riki system. The modifications I added to the prototype was to provided links where users could find authenticators ether in google store or apple store on there devices. Also fixed the prototype to generate different QR code images for each user created and stored them in all the images in a folder. In the our Riki system I added a Text Field for the 6 digit code that each user will have to enter on our login page. I have yet to figure out where I can add these modifications from my prototype without breaking the Riki System, so I will build my focus and learn where to add the routings for the MFA in my milestone 3.

Week 3

Summary

Added a MFA Token text box for the login and created an mfa.html. Next  week focus is to implement reCAPTCHA and MFA for users.

Week 4

Summary

Achieved my milestone 2, implemented reCAPTCHA and handled routing from a sample signup -> MFA -> login. Created a separate branch in GitHub. I am having issues with generating the QR code image.

A screenshot of a login screen

Description automatically generated

Code: https://github.com/lschwartz15/440project/tree/seth/

Next goal fixed issues and styling.

Week 5

Summary

I have cleaned up all errors with the MFA, fixed login errors, fix sign up errors, merged into main, created unit test, and styled html pages. Achieved storing the users when sign up also using reCAPTCHA, able to generate the correct QR image for each different user, successfully login with correct credentials and successful testing.

reCAPTCHA Cycle:

A screenshot of a computer screen

Description automatically generated

SignUp and Login Cycle:

A screenshot of a login screen

Description automatically generated

Code: [https://github.com/lschwartz15/440project/tree/seth/Links to an external site.](https://github.com/lschwartz15/440project/tree/seth/)