Coding Challenge: Secure Authentication and Authorization in ASP.NET Core MVC

Problem Statement:
You are tasked with building a secure ASP.NET Core MVC web application for a task management platform. This platform allows users to register, log in, manage tasks, and access different features based on their roles. The application needs to strictly follow secure coding practices to protect against common web vulnerabilities like Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), and ensure that user sessions are managed securely. Additionally, it must implement secure forms-based authentication and authorization, using both role-based and claims-based access control.

Your goal is to implement secure forms authentication, authorization, and secure session management. You will also need to ensure that the application is protected against CSRF and XSS vulnerabilities. Follow the user stories and ensure the application meets the security requirements.

User Stories

User Story 1: Secure Forms Authentication
As a developer, I want to configure secure forms authentication so that user login and registration processes are protected from unauthorized access.

Acceptance Criteria:

- Implement forms-based authentication using ASP.NET Core Identity or custom cookie-based authentication.
- Ensure user credentials (password) are hashed and salted before being stored in the database.
- Implement secure login and registration forms, validating and sanitizing user inputs to prevent SQL Injection and XSS.
- Redirect users to a "Dashboard" page upon successful login.
- Ensure the login page is protected, so users must be authenticated before accessing any restricted pages.

User Story 2: Secure Authorization with Roles and Claims
As a developer, I want to implement role-based and claims-based authorization to control access to specific areas of the application based on user roles.

Acceptance Criteria:

- Implement two roles: “Admin” and “User.”
- Admin users should be able to access the /Admin/ManageTasks page, while regular users can only access the /User/TaskList page.

- Use claims-based authorization to manage permissions beyond basic roles (e.g., allow “User” to have a “CanEditTask” claim, enabling them to edit their own tasks).
- Configure authorization policies to control access to sensitive routes based on roles and claims.

User Story 3: Protecting Against CSRF and XSS
As a developer, I want to protect the application against CSRF and XSS attacks so that user data and actions are secure.

Acceptance Criteria:

- Ensure that all forms (e.g., login, registration, task management) are protected by anti-forgery tokens to prevent CSRF attacks.
- Use ASP.NET Core's built-in anti-forgery token mechanisms for all form submissions.
- Ensure that all user-generated content (e.g., task descriptions, comments) is encoded to prevent XSS attacks.
- Use proper input validation and output encoding on all text input fields, ensuring user data is safe from malicious scripts.

User Story 4: Secure Session Management
As a developer, I want to ensure secure session management so that user sessions are protected from hijacking or unauthorized access.

Acceptance Criteria:

- Implement session management with secure cookies (using the HttpOnly and Secure flags).
- Set an expiration time for user sessions, and automatically log users out after a period of inactivity (e.g., 15 minutes).
- Ensure that all cookies are transmitted securely over HTTPS by enforcing the `Secure` cookie attribute.
- Use a secure mechanism for handling session tokens, such as JWTs or ASP.NET Core cookies.
- Ensure that session information is protected from being accessed by unauthorized parties.

Bonus: Secure Logout Implementation
As a developer, I want to implement a secure logout feature to properly invalidate user sessions.

Acceptance Criteria:

- When users log out, their session or authentication token should be invalidated.
- Clear authentication cookies upon successful logout.
- Redirect the user to the login page after logging out.
- Ensure that the user cannot access restricted pages after logging out.

Additional Requirements:

- The application should be an ASP.NET Core MVC application with a clear MVC structure.
- Use dependency injection for any services, such as user authentication and authorization logic.
- Store user data in a simple in-memory database or SQL database for the purposes of this challenge.
- Follow best practices for implementing secure input validation, output encoding, and session management.