Here’s a problem statement specifically focused on the topics you mentioned, without introducing any advanced concepts:

Coding Challenge: ASP.NET Core Secure Application

Problem Statement:
You are tasked with building a basic ASP.NET Core MVC web application for a small online shopping platform. The platform allows users to register, log in, view products, and make purchases. Due to security concerns, the application must follow best practices for input validation, output encoding, and secure coding techniques to prevent common vulnerabilities such as SQL Injection and Cross-Site Scripting (XSS). Additionally, secure authentication and authorization must be implemented to ensure that only authorized users can access certain parts of the platform.

Your goal is to implement secure input validation, output encoding, and role-based authentication in the application. Follow the user stories and ensure the security requirements are met.

User Stories

User Story 1: Secure Input Validation and Output Encoding

As a developer, I want to implement secure input validation and output encoding so that the application is protected against SQL Injection and XSS attacks.

Acceptance Criteria:

- Implement secure input validation for user-provided data (e.g., username, email, and password) on both the client and server sides.
- Ensure that user inputs are validated to prevent SQL Injection. Use parameterized queries or ORM (like Entity Framework) for database operations.
- Ensure that all user-generated content (e.g., reviews, comments, and product descriptions) is properly encoded when displayed to prevent XSS attacks.
- Password field validation: Enforce strong password rules (e.g., minimum length of 8 characters, at least one uppercase letter, one number, and one special character).
- Ensure that any form submissions (such as registration or product feedback) validate the input data and display appropriate error messages for invalid inputs.

User Story 2: Preventing SQL Injection

As a developer, I want to implement best practices to prevent SQL Injection attacks in the application.

Acceptance Criteria:

- Ensure that all database queries are parameterized to prevent direct inclusion of user input in SQL queries.
- Avoid dynamic SQL queries or string concatenation involving user data.
- If raw SQL queries are used, ensure they are properly sanitized and parameterized.
- Use an ORM (e.g., Entity Framework) for all data access to ensure automatic protection against SQL Injection.

User Story 3: Authentication and Authorization
As an admin, I want to restrict access to certain areas of the application (e.g., admin dashboard) to only authorized users.

Acceptance Criteria:

- Implement user authentication using either cookie-based or JWT-based authentication.
- Create two roles: "Admin" and "Customer."
- Admins should have access to the /Admin/Dashboard route, while customers should only be able to view product listings and place orders.
- Implement basic role-based authorization using ASP.NET Core's built-in authorization features to ensure that users can only access pages or routes based on their roles.
- Unauthorized users trying to access restricted areas should be redirected to the login page or shown an "Access Denied" message.

User Story 4: Secure User Registration and Login
As a developer, I want to implement secure user registration and login functionality to ensure the application is safe from common web vulnerabilities.

Acceptance Criteria:

- Ensure that user registration and login forms validate and sanitize inputs to prevent malicious data from being processed.
- Passwords should be stored securely (hashed and salted) in the database.
- Implement email validation to ensure that users provide a valid email address.
- On login, ensure that the authentication process is secure and prevents brute-force attacks (e.g., using CAPTCHA or rate-limiting).

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

Acceptance Criteria:

- When users log out, the session or authentication token should be invalidated.
- Clear any authentication cookies or tokens upon successful logout.
- Redirect the user to the login page after logging out.

Additional Requirements:

- The application should be an ASP.NET Core MVC application with a basic structure.
- Use dependency injection for services (e.g., for user authentication).
- Store user data in an in-memory database or simple SQL database for the purpose of this challenge.
- Focus on implementing secure input validation, output encoding, and basic authentication/authorization features.