Title: Angular Route Guards- Real World Scenario Lab

Objective: The objective of this lab exercise is to understand and practice Angular Route Guards by implementing them in a real-world scenario. Route guards are used to control access to certain routes in an Angular application based on specific conditions.

Scenario: Imagine you are developing a content management system where users can create, edit, and view articles. Some articles are public and can be accessed by anyone, while others are private and require authentication. Additionally, certain routes are restricted to administrators only. In this lab, you will implement route guards to control access to these routes based on user authentication and authorization.

Lab Steps:

1. Setting Up the Angular Project:
    - Create a new Angular project using Angular CLI.
    - Set up necessary components and modules for the content management system, such as ArticleListComponent, ArticleDetailComponent, LoginComponent, AdminComponent, and AuthService.
2. Implementing Authentication Service:
    - Create an AuthService responsible for handling user authentication and storing user credentials.
    - Implement methods like login(), logout(), and isAuthenticated() in the AuthService.
3. Implementing Route Guards:
    - Create an AuthGuard to protect routes that require authentication.
    - Implement the canActivate() method in the AuthGuard to check if the user is authenticated using the AuthService.
    - Apply the AuthGuard to routes that require authentication, such as the ArticleDetailComponent route.
4. Implementing Authorization:
    - Create an AdminGuard to protect routes that require administrator privileges.
    - Implement the canActivate() method in the AdminGuard to check if the user is an administrator.
    - Apply the AdminGuard to routes that require administrator privileges, such as the AdminComponent route.

5. Testing Route Guards:
    - Test the functionality of route guards by attempting to access protected routes without authentication or proper authorization.
    - Verify that users are redirected to the login page or denied access based on their authentication status and privileges.
6. Advanced Route Guard Scenarios:
    - Implement a scenario where certain routes are accessible only to authenticated users but not to administrators.
    - Explore the implementation of canActivateChild() method to guard child routes within a parent route.
    - Implement a scenario where route guards depend on asynchronous operations, such as fetching user roles from a backend API.
7. Additional Challenges (Optional):
    - Implement role-based access control (RBAC) using route guards to restrict access to specific routes based on user roles.
    - Experiment with canActivateChild() to protect nested child routes within feature modules.
    - Implement a scenario where route guards are used in conjunction with lazy-loaded modules.

Conclusion: Reflect on the importance of route guards in Angular applications for controlling access to routes based on authentication and authorization requirements. Discuss how route guards contribute to enhancing application security and providing a seamless user experience. Encourage students to explore more advanced topics related to Angular route guards, such as implementing custom guard interfaces and handling asynchronous operations within guards.