C# BASICS

Training Assignments

Document Code	**25e-BM/HR/HDCV/FSOFT**
Version	**1.1**
Effective Date	**20/11/2012**

Hanoi, 06/2019

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1.	01/Oct/2018	Create new	Draft		
2.	01/Jun/2019	Update template	Fsoft template	DieuNT1	

Contents

Angular with ASP.NET Core Web API: Quiz Application 4
Objectives: 4
Prerequisites: 4
Problem Requirements: 4
Task 1: Angular Fundamentals 4
Task 2: User Interaction and Navigation 5
Task 3: Data Management and Integration 6
Task 4: Security and Advanced Concepts 7
Task 5: Deployment and Integration 8
Mark Scale: 9

FRESHER ACADEMY		
	CODE:	ANG.P.L001
	TYPE:	MEDIUM
	LOC:	190
	DURATION:	900 MINUTES

Angular with ASP.NET Core Web API: Quiz Application

Objectives:

» Master Angular Fundamentals: Gain a solid understanding of core Angular concepts like components, templates, data binding (one-way and two-way), and built-in pipes (e.g., DatePipe) for data formatting.

» Component Development Expertise: Develop reusable and maintainable components encompassing UI logic and layout elements (common header, footer, subject list, individual quiz display, etc.).

» User Interaction and Navigation: Implement user interaction using directives (ngIf, ngFor) and navigate between different application sections (quizzes list, quiz detail) using Angular Router.

» Form Creation and Management: Create forms for login, register, subject/quiz creation/editing, and question creation (including answer options) utilizing FormsModule or ReactiveFormsModule based on complexity.

» Data Management and API Integration: Develop services using HttpClient to communicate with the ASP.NET Core Web API backend. These services will handle fetching data (quizzes, subjects), submitting answers, and potentially caching frequently accessed data.

» Dependency Injection: Understand and leverage dependency injection to promote loose coupling between components and services, improving testability and maintainability.

» Security Awareness: Implement security best practices like input validation and sanitization to mitigate potential vulnerabilities and protect user data.

» Advanced Concepts Exploration: Explore different rendering strategies like Client-side Rendering (CSR), Server-side Rendering (SSR), and Static Site Generation (SSG) to choose the appropriate one based on performance, SEO, and user experience considerations.

» Unit Testing Expertise: Write unit tests for components, services, and other application logic using a testing framework like Jasmine and Karma to ensure core functionalities and data behavior work as expected.

» Deployment Readiness: Learn how to build the Angular application for production using the Angular CLI and understand different deployment options (static hosting, cloud platforms) based on project requirements.

Prerequisites:

» Working environment: Visual Studio Code/Visual Studio 2013 or higher.

» Delivery: Source code packaged in a compress archive.

Problem Requirements:

This assignment outlines the development of an Angular frontend application to complement the ASP.NET Core Web API Quiz Application you've built..

Task 1: Angular Fundamentals

Description:

This task focuses on building a foundational understanding of Angular concepts, including components, templates, data binding, and pipes.

Function Requirements:

- **Project Setup:**
    - Create a new Angular project using the Angular CLI **(ng new quiz-app).**
    - Install Bootstrap/Tailwind, Font-Awesome packages
- **Components:**
    - Create common component to reuse in other components
        - Layout
        - Header
        - Footer
    - Create reusable components to complete UI for Subjects
        - Example:
          Subject UI
            - subject-list: Show list of Subject in table
    - Utilize the **@Component** decorator to define component metadata (selector, template, styles).
- **Templates:**
    - Write HTML templates for components using Angular directives and expressions.
    - Leverage two-way data binding **([(ngModel)])** to bind component properties to template elements.
- **Data Binding and Pipes:**
    - Implement data binding (one-way, two-way) to display application data in templates.
    - Utilize built-in pipes (e.g., DatePipe, CurrencyPipe) to format data for display.

      Date format to display:

        - dd MM YYYY – 16 May 2024

Hints:

- Use Angular CLI commands for project creation and component generation (**ng generate component**).
- Refer to Angular documentation for detailed information on components, templates, and data binding.

Business Rules:

- Components should be self-contained units responsible for UI and logic.
- Templates should be clear, concise, and utilize Angular directives for interactivity.
- Data binding should provide a seamless connection between component data and the view.

Evaluation Criteria:

- Functional components displaying quiz data and user interfaces.
- Correct utilization of templates with data binding and pipes for data presentation.
- Adherence to Angular best practices for component structure and template design.

Submission file:

- Zip solution folder to a zip file
- File: FullName_ANG_QuizApp_Task_01_v1.0.zip

Estimated Time: 180 minutes.

<u>Task 2: User Interaction and Navigation</u>

Description:

This task focuses on implementing user interactions and routing within the Angular application.

Function Requirements:

- **Directives:**
  - Implement built-in directives like ngIf, ngFor to conditionally render content and iterate over data.
  - Consider using custom directives for complex UI behaviors.
- **Routing:**
  - Configure routing using RouterModule to define routes for different views (quizzes list, individual quiz detail).
  - Utilize navigation components (router-link) to link between routes and enable user navigation.
  - Complete UI for List of Subjects, Quiz, Question (Include Answer).
- **Forms:**
  - Create Login and Register Form UI for Authentication module
  - Create Create and Edit Form UI for Subject, Quiz, Question (Include Answer).
  - Leverage FormsModule for basic forms or explore ReactiveFormsModule for more complex scenarios..

Hints:

- Use Angular Router documentation to understand route configuration and navigation.
- Explore the available directives and built-in form modules for functionalities..

Business Rules:

- Directives should enhance the interactivity and behavior of components.
- Routing should enable seamless navigation between different application sections.
- Forms should provide a user-friendly interface for interaction and data submission.

Evaluation Criteria:

- Implementation of directives to control content display and user interaction.
- Defined routes and navigation components for user flow within the application.
- Functional forms for user input and data manipulation.

Submission file:

- Zip solution folder to a zip file
- File: FullName_ANG_QuizApp_Task_02_v1.0.zip

Estimated Time: 180 minutes.

Task 3: Data Management and Integration

Description:

This task focuses on managing application data and integrating with the ASP.NET Core Web API backend.

Function Requirements:

- **Services:**
  - Create services to communicate with the ASP.NET Core Web API. Utilize HttpClient to make HTTP requests to API endpoints (e.g., fetch quizzes, submit answers).

    - Consider using an in-memory data service for caching frequently accessed data.
- **Dependency Injection:**
    - Inject services into components using the @Inject decorator.
    - Understand the concept of dependency injection for managing dependencies within the application.
- **API Integration:**
    - Implement services to interact with the Quiz API endpoints (GET, POST, PUT, DELETE).
    - Handle API responses, parse data, and update the application state accordingly.

Hints:

- Refer to Angular documentation on HTTP services and dependency injection.
- Use libraries like rxjs for asynchronous data handling and observable patterns.

Business Rules:

- Services should encapsulate data access logic and communication with the API.
- Dependency injection should promote loose coupling and testability of components.
- API calls should be handled appropriately, including error handling and data processing.

Evaluation Criteria:

- Development of services to manage data communication and interaction with the API.
- Correct implementation of dependency injection for service consumption within components.
- Successful integration with the Quiz API for data retrieval and manipulation.

Submission file:

- Zip solution folder to a zip file
- File: FullName_ANG_QuizApp_Task_03_v1.0.zip

Estimated Time: 180 minutes.

Task 4: Security and Advanced Concepts

Description:

This task delves into security considerations and explores advanced Angular features.

Function Requirements:

- **Authentication/Authorization:**
    - Integrate with the ASP.NET Core Web API's authentication mechanisms (JWT).
    - Store tokens securely and use them for authorized API calls in Angular services.
- **Security:**
    - Implement best practices for secure application development (e.g., input validation, sanitization).
    - Consider security vulnerabilities and implement strategies to mitigate risks.
- **CSR, SSR, SSG:**
    - Explore concepts of Client-side Rendering (CSR), Server-side Rendering (SSR), and Static Site Generation (SSG).
    - Choose an appropriate rendering strategy based on application requirements (performance, SEO).
- **Unit Testing:**

  - Write unit tests for components, services, and other application logic using a testing framework (e.g., Jasmine, Karma).
  - Ensure core functionalities and data behavior are tested effectively.

Hints:

- Leverage libraries like angular2-jwt for JWT token management in Angular.
- Research best practices for secure coding and data handling in Angular applications.
- Explore Angular Universal for SSR and SSG capabilities.
- Utilize testing frameworks and tools for writing unit tests.

Business Rules:

- Authentication and authorization should restrict access to protected resources and functionalities.
- Security measures should safeguard user data and prevent vulnerabilities.
- Rendering strategy should balance performance, SEO, and user experience.
- Unit tests should provide a safety net for application logic and maintain code quality.

Evaluation Criteria:

- Implementation of authentication and authorization using JWT tokens.
- Adherence to security best practices to minimize application risks.
- Understanding and consideration of different rendering strategies for Angular applications.
- Writing unit tests to ensure component and service functionality.

Submission file:

- Zip solution folder to a zip file
- File: FullName_ANG_QuizApp_Task_04_v1.0.zip

Estimated Time: 180 minutes.

Task 5: Integration and Deployment

Description:

This task focuses on deploying the Angular application and integrating it with the ASP.NET Core Web API backend.

Function Requirements:

- **Integration with ASP.NET Core Web API:**
  - Configure base URL for API calls in Angular services to point to the deployed backend API.
  - Ensure seamless communication and data exchange between frontend and backend.
- **Deployment:**
  - Utilize tools like Angular CLI (ng build) to build the Angular application for production.
  - Consider deployment options (static hosting, cloud platforms) based on project requirements.

Hints:

- Configure environment variables for different deployment environments (development, production).
- Research deployment strategies and tools relevant to your chosen hosting platform.

Business Rules:

- The frontend and backend should communicate effectively for successful application operation.
- Deployment should be efficient and result in a functional, accessible application.

Evaluation Criteria:

- Successful integration with the deployed ASP.NET Core Web API backend.
- Building and deployment of the Angular application for production environment.
- Understanding of different deployment strategies and considerations.

Submission file:

- Zip solution folder to a zip file
- File: FullName_ANG_QuizApp_Task_05_v1.0.zip

Estimated Time: 180 minutes.

Mark Scale:

OOP design	10%	Function requirements	60%
Business rules	15%	Main function	15%