**Dot Net Full Stack Developer**

Capstone Project Problem Statement



**Blog Tracker Application**

**Capstone Project**

**Prerequisites:** Basics of C#, MS-SQL Server, ASP.Net MVC, Web API, and NUnit

**Case Study:**

Simplona Tech Solutions needs a full stack **Blog Tracker Application (BTA)** to maintain blog details written by their employees as part of the continuous learning process. The employee must write one blog per week and add blog-related information like the blog title, subject, date of creation, and blog URL in the Blog Tracker Application. You need to create a Full Stack Application that allows employees to add their blog-related details and show all blogs written by employees on the home page.

**D. Unit Testing**

1. **System Overview:**

**E. Docker**

**Containerization**

**Docker Compose**

**NUnit**

**Azure**

**SQL Server**

**Database**

**Web API Application**

**Data Access Logic**

**MVC**

**A. Data Access Layer**

**B. App Service Layer**

**C. App UI Layer**

**A. Data Access Layer (DAL):** This layer will be a type of Class Library. This application will create a database using Entity Framework - Code First Approach on **Azure** and implement Database functionalities such as saving blog details, to get all blogs details, get the blog by ID, update blog details, to delete blog details.

Consider the following Entity Model classes to create a database for the same:

1. Add entity class **AdminInfo** and public properties:

|  |  |
| --- | --- |
| **Property Name** | **Type** |
| EmailId | String |
| Password | String |

1. Add entity class **EmpInfo** and public properties as below:

|  |  |
| --- | --- |
| **Property Name** | **Type** |
| EmailId | String |
| Name | String |
| DateOfJoining | Datetime |
| PassCode | Int |

1. Add entity class **BlogInfo** and public properties as below:

|  |  |
| --- | --- |
| **Property Name** | **Type** |
| BlogId | Int |
| Title | String |
| Subject | String |
| DateOfCreation | Datetime |
| BlogUrl | String |
| EmpEmailId | String |

4. After creating the above entity classes, create the context class by inheriting **DbContext** class

5. Add one more class by inheriting **DropCreateDatabaseIfModelChanges<contextclass>** and default data into AdminInfo table by overriding the SEED method

6. Write database functionalities for EmpInfo entity using Data Repository Pattern: saving employee details, to get all employee details, and to validate employees while logging in

7. Write database functionalities for BlogInfo entity using Data Repository Pattern: saving blog details, to update blog, to delete a blog, to get all blog details, and to get the blog by ID

**B. App Service Layer:** This layer will be a type of ASP.Net Web Application (Web API).  
This application contains RESTful services to consume functionalities from the DAL class.

1. Add a Web API controller and write action methods to issue a GET and POST and PUT and DELETE request to perform all the functionalities written for EmpInfo and BlogInfo entities inside DAL layer

2. Use attribute-based routing while implementing these functionalities

3. Enable Swagger support for documentation to test this layer

4. Enable CORS (Cross Origin Resource Sharing) policies for all controllers

**C. App UI Layer:** This will be an ASP.Net MVC as a front-end application to implement the following functionalities:

1. Display all blog details on the home page written by all employees

2. Authenticate admin using admin Email ID and password

3. Show page for adding employee details for the admin

4. Authenticate employee using employee EmaiI ID and passcode

5. Show a page for adding blog details to employees

**Note:** While writing views, use bootstrap to design responsive web UI for all above functionalities

**D. DALTest Layer:** This layer is a type of Class Library to test DAL layer functionality using the NUnit and Moq framework.

**E. Docker Containerization:** Create Docker Compose using the Docker container orchestration feature through visual studio for AppLayer and Service Layer projects

**Sample input/add output:**

**1. Home page (All blogs added by all employees):**Graphical user interface, text, application, chat or text message

Description automatically generated

**2. After redirecting to blog page:**

Graphical user interface, application

Description automatically generated

**3. Login as admin:**

Graphical user interface, text, application, chat or text message

Description automatically generated

**4. Admin login page:**

Graphical user interface, text, application, email

Description automatically generated

**5. Employee list page:**

Table

Description automatically generated

**6. Add a new employee page:**

Graphical user interface, text, application, email

Description automatically generated

**7. View employee list:**

Table

Description automatically generated

**8. Logging out from admin:**

Table

Description automatically generated

**9. Log in as an employee:**

Graphical user interface, text, application

Description automatically generated

**10. Employee login page:**

Graphical user interface, text, application, email

Description automatically generated

**11. Redirecting to add a new blog:**

Graphical user interface, text, application, email

Description automatically generated

**12. Add a new blog page:**

Graphical user interface, text, application, chat or text message, email

Description automatically generated

**13. Logging out after adding a new blog:**

Graphical user interface, text, application, chat or text message

Description automatically generated

**14. All added blogs by all employees:**

Graphical user interface, text, application, chat or text message

Description automatically generated

**15. After creating Docker Compose in Docker Desktop:**

Graphical user interface, text, application, email

Description automatically generated