Blog Tracker Application

Introduction

The primary aim of this project is to develop a comprehensive Blog Tracker application using cutting-edge web development and database technologies. The project encompasses several pivotal goals, including the creation of a web application, a web API, and the utilization of Docker containers for deployment. This application aims to provide bloggers with a robust platform for creating, managing, and sharing their content. It places a strong emphasis on secure authentication and authorization mechanisms while adhering to best practices in web application development.

Step 1: Create an Azure SQL Database

1. Sign in to the [Azure portal](https://portal.azure.com/).

2. Click on "Create a resource" and search for "SQL Database."

3. Click on "SQL Database" and then click "Create."

4. Fill in the necessary information like the database name, resource group, server, and configure any additional settings required.

5. Review and create the database. Once created, copy the connection string, which you'll use in your web app.

Step 2: Create a Web App MVC Project

1. Open Visual Studio.

2. Create a new ASP.NET Core MVC project named "BlogTracker."

Step 3: Create Model Classes

1. Create three model classes: `EmpInfo.cs`, `BlogInfo.cs`, and `AdminInfo.cs` to represent your database entities.

2. Define the properties for each model class

Step 4: Install Required Package

1. Install the necessary NuGet packages:

- `*Microsoft.EntityFrameworkCore.SqlServer*`
- `*Microsoft.EntityFrameworkCore.Tools*`

2. Create a `DbContext` class, `BlogTrackerDbContext.cs`, that inherits from `DbContext`, and define `DbSet` properties for your model classes.

Step 5: Configure Connection Strin

1. In the `appsettings.json` file, create a connection string using the copied connection string from Azure SQL Database.

```
"ConnectionStrings": {
    "BlogTrackerDbContext": "your_connection_string_here"
}
```

2. In `Program.cs`, use the connection string when configuring the `DbContext` with the `builder`:

```
builder.Services.AddDbContext<BlogTrackerDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("BlogTrackerDbContext")));
```

Step 6: Entity Framework Migrations

1. Open the Package Manager Console in Visual Studio.

2. Run the following commands to create and apply migrations:

```
Add-Migration InitialCreate
Update-Database
```

3. This will generate the database schema based on your model classes.

Step 7: Database Seeding

1. In `Program.cs`, create a database seeding method. This is where you can add default values, including admin accounts for authentication.

2. Use the `EnsureCreated` and `Migrate` methods of the `DbContext` to apply migrations and create the database if it doesn't exist.

Step 8: Authentication Setup

1. Configure authentication in your project. You can use ASP.NET Core Identity for user authentication.

2. Modify the `EmpInfo` controller to include authentication attributes to restrict access to authenticated users.

Step 9: Routing Configuration

1. Adjust routing in `Program.cs` to achieve the desired URL routing and redirects.

Step 10: Test Class Library

1. Create a class library project for your tests using NUnit and Moq frameworks.

2. Install the necessary NuGet packages for NUnit and Moq.

3. Create test fixtures using `[TestFixture]` and write test methods using `[Test]`.

4. Use the NUnit test explorer in Visual Studio to run tests and check if they pass or fail.

Step 11: Create a Web API for Backend

1. Create a Web API project in your solution to handle backend functionality for `BlogInfo` and `EmpInfo`.

2. Implement controllers and actions to expose the necessary API endpoints.

3. Configure routing and authentication for your API.

Step 12: Dockerize the Web App and Web API

1. Create Dockerfiles for both the web app and the web API. These Dockerfiles should specify the necessary runtime and build environments.

2. Build Docker images for both projects using the following commands:

docker build -t blog-tracker-web-app -f BlogTracker/Dockerfile .

docker build -t blog-tracker-web-api -f WebApi/Dockerfile .

3. Create a `docker-compose.yml` file in your solution's root directory to define how to run both containers together:

4. Use Docker Compose to run both containers at the same time:

docker-compose up

Conclusion

The successful completion of the Blog Tracker project represents a significant milestone in modern web application development and deployment practices. The project has effectively realized its objectives, including the creation of a functional web application and web API, both of which have been Dockerized for seamless deployment. Incorporating user authentication and authorization mechanisms has added an essential layer of security and access control. The project's emphasis on unit testing using NUnit and Moq frameworks underscores its commitment to code quality and maintainability, making it a noteworthy contribution to contemporary web application development practices.

GitHubLink:

https://github.com/Mogal74/Capstone-Project.git