# Blog Management Application

## Table of Contents

* [Introduction](" \l "introduction)
* [Setup Instructions](" \l "setup-instructions)
* [Running the Application](" \l "running-the-application)
* [Design Decisions](" \l "design-decisions)
* [Application Structure](" \l "application-structure)

## Introduction

This is a full-stack Blog Management application developed using .NET Core for the backend and Angular for the frontend. The application allows users to create, read, update, and delete blog posts. The backend follows the CQRS (Command Query Responsibility Segregation) pattern and the frontend is developed using Angular.

## Setup Instructions

### Prerequisites

* .NET Core SDK 6.0 or later
* Node.js 14 or later
* Angular CLI 12 or later
* Visual Studio 2022 (recommended for .NET development)
* Git

**Backend Setup**  
  
1. git clone <https://github.com/sudiptamax/BlogApplication.git>

1. dotnet restore
2. dotnet ef database update
3. dotnet run

**Frontend Setup**

1. Navigate to the frontend directory:  
    cd BlogApplication/BlogClientApp
2. Install the dependencies:  
    npm install
3. Run the application

ng serve

1. Open your browser and navigate to <http://localhost:4200.>

**Running the Application**

Backend

To run the backend API server:

cd BlogApplication/BlogApi

dotnet run

Frontend

To run the frontend Angular application:

cd BlogApplication/BlogClientApp

ng serve

Url: Navigate to <http://localhost:4200> in your web browser.  
  
  
Design Decisions

### CQRS Pattern

The backend API is designed using the CQRS pattern to separate the concerns of reading and writing data. This allows for more scalable and maintainable code. Commands handle operations that change the state (create, update, delete), while queries handle read-only operations.

### Angular Frontend

The frontend is developed using Angular for its powerful features and modular structure, which helps in creating a maintainable and scalable front-end application.

### Encrypted IDs

For security reasons, blog post IDs are encrypted using SHA-256. This ensures that sensitive data is not exposed directly through URLs.

## Application Structure

### Backend (BlogApi)

* **Controllers**: Handle HTTP requests and delegate to appropriate handlers.
* **Handlers**: Contain the business logic for handling commands and queries.
* **Commands**: Represent operations that change the state.
* **Queries**: Represent read-only operations.
* **Repositories**: Handle data access logic.
* **Models**: Represent the data structures used in the application.

**BlogApi**

│ BlogApi.sln

│ Program.cs

│ Startup.cs

│

├───**Controllers**

│ BlogController.cs

│

├───**Handlers**

│ ├───**Commands**

│ │ CreateBlogPostHandler.cs

│ │ UpdateBlogPostHandler.cs

│ │ DeleteBlogPostHandler.cs

│ └───**Queries**

│ GetBlogPostByIdHandler.cs

│ GetAllBlogPostsHandler.cs

│

├───**Commands**

│ CreateBlogPostCommand.cs

│ UpdateBlogPostCommand.cs

│ DeleteBlogPostCommand.cs

│

├───**Queries**

│ GetBlogPostByIdQuery.cs

│ GetAllBlogPostsQuery.cs

│

├───**Repositories**

│ IBlogRepository.cs

│ BlogRepository.cs

│

└───**Models**

BlogPost.cs

### Frontend (BlogClientApp)

* **Components**: Angular components for different views.
* **Services**: Handle API calls and data manipulation.
* **Models**: Define the data structures used in the frontend.
* **Routes**: Define the navigation and routing for the application.

**BlogClientApp**

│ angular.json

│ package.json

│ tsconfig.json

│

└───**src**

├───**app**

│ ├───**components**

│ │ ├───home

│ │ │ home.component.html

│ │ │ home.component.ts

│ │ │

│ │ ├───create-update-blog

│ │ │ create-update-blog.component.html

│ │ │ create-update-blog.component.ts

│ │ │

│ │ ├───listing

│ │ │ listing.component.html

│ │ │ listing.component.ts

│ │ │

│ │ └───blog-detail

│ │ blog-detail.component.html

│ │ blog-detail.component.ts

│ │

│ ├───models

│ │ blog.model.ts

│ │

│ ├───services

│ │ blog.service.ts

│ │ notification.service.ts

│ │

│ └───app-routing.module.ts

│ app.module.ts

│

├───assets

└───environments

environment.ts

environment.prod.ts

This README provides a guide to setting up, running, and understanding the design decisions behind your Blog Management application.

### **Potential Scalability Enhancements:** There are several ways this application can scale with huge number of blog posts. As this application implementation architecture wise able to scale in following areas.

### **Client-Side (Angular Application) Scalability**

### Lazy Loading Modules:

### Implement lazy loading for feature modules to improve initial load time.

### Pagination and Infinite Scroll:

### Use pagination or infinite scroll to load and display only a subset of blog posts at a time.

### Optimized Data Fetching:

### Implement delay for search inputs and avoid fetching data on every keypress

### Caching and State Management:

### Use caching mechanisms and state management libraries like NgRx or Akita to reduce redundant API calls.

### **Server-Side (API) Scalability**

### Database Optimization:

### Use indexing, sharding, and partitioning to optimize database performance.

### Efficient Querying:

### Implement efficient querying strategies, such as using proper indexes and avoiding N+1 query problems.

### Caching:

### Use caching layers like Redis or Memcached to cache frequent read queries.

### Load Balancing:

### Implement load balancing to distribute incoming requests across multiple servers.

### Horizontal Scaling:

### Scale horizontally by adding more instances of the API server.

### Asynchronous Processing:

### Use message queues like RabbitMQ or Kafka for handling background tasks and asynchronous processing.

### Microservices Architecture:

### Break down the application into microservices to handle different functionalities independently.

### Database Sharding:

### Split the database into smaller, more manageable pieces called shards.

### NoSQL Databases:

### Use NoSQL databases like MongoDB or Cassandra for handling large volumes of unstructured data.

### Cloud Storage:

### Utilize cloud storage solutions like Amazon S3 or Azure Blob Storage for storing large files.

### Monitoring:

### Use monitoring tools like Prometheus, Grafana, or New Relic to monitor the application’s performance and health.

### Regular Maintenance:

### Perform regular maintenance tasks such as database optimization, code refactoring, and dependency updates.

Output of the application :









