DOCUMENTACIÓN

TÉCNICA

WHAT WE DO

Fecha
28/08/2023

TECNOLOGÍAS PROYECTO

1. Tecnologías Utilizadas

En este proyecto, utilizamos una combinación de tecnologías modernas para desarrollar una aplicación web robusta y funcional. A continuación, se presentan las principales tecnologías que forman la base de nuestro proyecto:

1.1 ASP.NET Core MVC

ASP.NET Core MVC es un marco de trabajo de código abierto desarrollado por Microsoft que nos permite construir aplicaciones web escalables y de alto rendimiento. Basado en el patrón de diseño Modelo-Vista-Controlador (MVC),

ASP.NET Core MVC nos permite separar las preocupaciones de la aplicación y facilita el desarrollo y mantenimiento del código.

1.2 ADO.NET (ORM)

Para interactuar con la base de datos, utilizamos ADO.NET como nuestro ORM (Object-Relational Mapping). ADO.NET nos permite mapear objetos de la aplicación a tablas en la base de datos, simplificando la comunicación con el sistema de gestión de bases de datos y proporcionando un acceso eficiente y seguro a los datos.

1.3 Vistas Razor

Las Vistas Razor son una parte esencial de nuestro proyecto, ya que nos permiten definir la interfaz de usuario de nuestra aplicación web. Utilizamos la sintaxis de Razor para combinar C# y HTML, lo que facilita la generación dinámica de contenido en las páginas web y la integración de datos del servidor en las vistas.

1.4 jQuery

Para mejorar la interactividad y la dinámica en el lado del cliente, utilizamos la biblioteca jQuery. jQuery simplifica la manipulación del DOM y la comunicación con el servidor, lo que nos permite crear experiencias de usuario más fluidas y atractivas.

HERRAMIENTAS DE DESARROLLO

2. Control de Versiones y Colaboración

En este proyecto, hemos adoptado una metodología de control de versiones y colaboración que nos permite trabajar de manera eficiente y coordinada. A continuación, se describen los aspectos clave de nuestro flujo de trabajo:

2.1 Repositorio en GitHub

Hemos configurado un repositorio en GitHub como el punto central de nuestro control de versiones. Utilizamos la rama master como rama principal donde se encuentra la versión estable y funcional de la aplicación. Además, hemos creado una rama independiente para cada miembro del equipo.

- **Master**: Contiene la versión estable y funcional de la aplicación. Las actualizaciones en esta rama siguen un proceso de revisión antes de ser fusionadas.
- **Ramas de Equipo**: Cada miembro del equipo tiene su propia rama para trabajar en nuevas características o correcciones. Estas ramas permiten el desarrollo paralelo sin interferencias.

2.2 IDE: Visual Studio 2022

Como entorno de desarrollo principal, utilizamos Visual Studio 2022. Este IDE proporciona herramientas sólidas y una integración perfecta con ASP.NET Core MVC, lo que agiliza la creación y el mantenimiento de nuestro proyecto. Aprovechamos las características del IDE, como la depuración en tiempo real y el resaltado de sintaxis, para asegurarnos de que nuestro código sea confiable y eficiente.

2.3 Flujo de Trabajo

Nuestro flujo de trabajo general sigue estos pasos:

a. **Creación de Ramas**: Cada miembro del equipo crea una rama independiente para trabajar en una tarea específica.
b. **Desarrollo y Pruebas:** Los cambios se realizan en las ramas individuales. Realizamos pruebas unitarias y de integración para asegurarnos de que las nuevas funcionalidades funcionen correctamente.
c. **Solicitud de Extracción (Pull Request)**: Una vez que una tarea se completa, el miembro del equipo crea una solicitud de extracción desde su rama a la rama master. Esto permite una revisión por parte de otros miembros del equipo antes de fusionar los cambios.
d. **Revisión y Comentarios:** Otros miembros del equipo revisan el código y proporcionan comentarios, sugerencias o correcciones si es necesario.
e. **Fusión y Despliegue:** Después de la revisión exitosa, los cambios se fusionan en la rama master. Posteriormente, desplegamos la versión actualizada en el entorno de producción.

ARQUITECTURA

3. Arquitectura: Modelo-Vista-Controlador (MVC)

En este proyecto, hemos adoptado la arquitectura Modelo-Vista-Controlador (MVC) para organizar y estructurar nuestro código de manera eficiente. La arquitectura MVC se utiliza ampliamente en el desarrollo de aplicaciones web y nos ayuda a lograr una separación clara de las responsabilidades, lo que facilita la mantenibilidad y escalabilidad de nuestra aplicación.

3.1 Modelo

El Modelo representa la capa de datos y la lógica empresarial de nuestra aplicación. Aquí es donde gestionamos la interacción con la base de datos utilizando ADO.NET, definimos las clases y entidades que representan nuestros datos y realizamos operaciones como la recuperación, actualización y eliminación de registros.

3.2 Vista

La Vista es la capa encargada de la presentación de la interfaz de usuario. En nuestro caso, utilizamos Vistas Razor para definir cómo se muestra la información al usuario final. Las Vistas combinan HTML y la sintaxis de Razor para generar contenido dinámico y presentar datos provenientes del Modelo. Utilizamos estilos de Bootstrap para lograr un diseño atractivo y responsivo.

3.3 Controlador

El Controlador es el intermediario entre el Modelo y la Vista. Aquí gestionamos las solicitudes del usuario, procesamos la lógica necesaria y actualizamos el Modelo según sea necesario. Los Controladores reciben la entrada del usuario a través de las rutas URL y determinan cómo responder a esas solicitudes, preparando los datos para su presentación en las Vistas.

3.4 Beneficios de MVC

La arquitectura MVC nos ofrece varios beneficios clave en este proyecto:

- **Separación de Responsabilidades:** Cada componente (Modelo, Vista, Controlador) tiene una función específica y no se mezclan las lógicas de negocio con la presentación.
- **Mantenibilidad:** La separación de responsabilidades facilita la identificación y solución de problemas, y permite cambios en una parte del sistema sin afectar a otras.
- **Reutilización de Código:** Gracias a la modularidad de MVC, es posible reutilizar componentes como Vistas y Modelos en diferentes partes de la aplicación.
- **Escalabilidad:** Con MVC, podemos agregar nuevas características o modificar existentes sin reescribir grandes secciones del código.

LIBRERÍAS Y FRAMEWORKS ADICIONALES

4. Librerías y Frameworks Adicionales

En este proyecto, hemos aprovechado diversas librerías y frameworks adicionales para mejorar y ampliar la funcionalidad de nuestra aplicación. A continuación, se detallan algunas de las librerías clave que hemos incorporado:

4.1 FirebaseStorage.Net

Utilizamos la librería FirebaseStorage.Net para la gestión de imágenes en nuestra aplicación. Esta librería nos permite cargar, descargar y administrar imágenes en el servidor de almacenamiento de Firebase. Al crear o modificar un evento, aprovechamos FirebaseStorage.Net para subir fotos a Firebase Storage, lo que enriquece la experiencia visual de los usuarios.

4.2 Microsoft.Extensions.Options

La librería Microsoft.Extensions.Options nos ofrece una manera eficiente de administrar y acceder a las opciones de configuración en nuestra aplicación. Utilizamos esta librería para gestionar la configuración, como las credenciales de Firebase, de manera centralizada y accesible en todo el código.

4.3 Microsoft.VisualStudio.Web.CodeGeneration.Design

Esta librería es fundamental para el proceso de generación de código en ASP.NET Core MVC. La utilizamos para generar automáticamente código de controladores y vistas a partir de nuestros modelos y entidades, lo que agiliza el proceso de desarrollo y mejora la coherencia del código generado.

4.4 System.Data.SqlClient

La librería System.Data.SqlClient nos proporciona la funcionalidad necesaria para conectarnos y comunicarnos con la base de datos SQL Server. Utilizamos esta librería para realizar operaciones de lectura y escritura en la base de datos, interactuando con nuestras entidades y modelos de datos.

4.5 FirebaseAuthentication

Aunque no fue mencionada previamente, es importante mencionar que también usamos FirebaseAuthentication para la autenticación de usuarios. Esta librería facilita la autenticación segura y eficiente de los usuarios a través de las capacidades de autenticación de Firebase.

4.6 Bootstrap

Para estilizar y dar formato a nuestras vistas Razor, hemos integrado el framework CSS Bootstrap. Bootstrap nos proporciona una amplia gama de componentes y estilos predefinidos que nos ayudan a lograr un diseño atractivo y responsivo en nuestra interfaz de usuario.

SEGURIDAD Y AUTENTICACIÓN

5. Seguridad y Autenticación

La seguridad es una preocupación fundamental en nuestra aplicación. Hemos implementado un sistema de autenticación que garantiza que solo los usuarios autorizados puedan acceder a ciertas partes del sistema. A continuación, se describe cómo manejamos la autenticación y el acceso a roles específicos en nuestra aplicación.

5.1 Autenticación y Roles

Para garantizar que los usuarios accedan solo a las partes de la aplicación para las que tienen permiso, utilizamos la funcionalidad de autenticación y roles proporcionada por ASP.NET Core. La autenticación se realiza a través de cookies y las solicitudes deben incluir credenciales válidas para acceder a las rutas protegidas.

5.2 Flujo de Autenticación

Cuando un usuario intenta iniciar sesión en la aplicación, realizamos las siguientes acciones:

i. **Autenticación de Usuario**: Comprobamos si el usuario es un usuario regular o una empresa y verificamos si las credenciales proporcionadas (correo electrónico y contraseña) son válidas utilizando el servicio de autenticación correspondiente (**_ServicioUsuario** o **_ServicioEmpresa**).
ii. **Creación de Claims**: Si las credenciales son válidas, creamos un conjunto de claims (reclamaciones) que contienen información relevante sobre el usuario, como su nombre, correo electrónico, rol, identificador, saldo y puntos de usuario.
iii. **Creación de Identidad**: Utilizamos los claims para crear una identidad (**ClaimsIdentity**) que representa al usuario autenticado.
iv. **Inicio de Sesión**: Iniciamos una sesión de autenticación utilizando el esquema de autenticación de cookies (**CookieAuthenticationDefaults.AuthenticationScheme**) y proporcionamos la identidad creada.

5.3 Autorización de Roles

Una vez que un usuario está autenticado, podemos controlar el acceso a ciertas partes de la aplicación basándonos en sus roles. Utilizamos el atributo **[Authorize(Roles = "Usuario")]** o **[Authorize(Roles = "Empresa")]**en nuestros controladores para asegurarnos de que solo los usuarios con el rol "Usuario" o "Empresa"puedan acceder a esas rutas.

5.4 Seguridad y Protección de Datos

Además de la autenticación y autorización, también estamos comprometidos con la protección de los datos del usuario. Utilizamos técnicas de encriptación para asegurar que las contraseñas y otros datos sensibles se almacenen de manera segura en nuestra base de datos.

CONFIGURACIÓN Y FLUJO DE EJECUCIÓN

6. Configuración y Flujo de Ejecución

En esta sección, exploraremos cómo se configuran los servicios y se establece el flujo de ejecución en nuestra aplicación ASP.NET Core MVC. La configuración de servicios y el manejo de solicitudes se realizan en el archivo de inicio de la aplicación (Program.cs).

6.1 Configuración de Servicios

En el archivo **Program.cs**, configuramos los servicios requeridos para nuestra aplicación. Utilizamos el método **AddAuthentication** para habilitar la autenticación mediante cookies, y especificamos opciones como la ruta de inicio de sesión y el tiempo de expiración. Además, utilizamos **AddHttpContextAccessor** para permitir el acceso al contexto HTTP en nuestros servicios.

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.Authentic
    .AddCookie(option =>
    {
        option.LoginPath = "/Auth/Login";
        option.ExpireTimeSpan = TimeSpan.FromDays(30);
    });

builder.Services.AddHttpContextAccessor();
```

6.2 Flujo de Ejecución

A continuación, establecemos el flujo de ejecución de nuestra aplicación en el mismo archivo Program.cs. Utilizamos los métodos **UseExceptionHandler** para manejar excepciones en caso de que la aplicación no esté en modo de desarrollo, y **UseHttpsRedirection** y **UseStaticFiles** para redirigir a HTTPS y servir archivos estáticos, respectivamente.

```
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
```

Luego, definimos las rutas de los controladores utilizando **MapControllerRoute** y establecemos un enrutamiento predeterminado hacia el controlador **"Home"** y la acción **"Inicio"**.

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Inicio}");
```

Finalmente, utilizamos el método **Run** para iniciar la aplicación y hacer que esté lista para manejar solicitudes entrantes.

6.3 Conclusiones

La configuración y el flujo de ejecución son fundamentales para el funcionamiento adecuado de nuestra aplicación ASP.NET Core MVC. Al definir servicios, rutas y middleware, aseguramos que la aplicación pueda manejar solicitudes de manera eficiente y segura.

ACCESO A DATOS Y SERVICIOS IMPLEMENTADOS

7. Acceso a Datos y Servicios Implementados

En esta sección, exploraremos cómo hemos implementado y utilizado servicios basados en ADO.NET para acceder y manipular los datos en la base de datos. Las interfaces que definimos actúan como contratos para la implementación de los servicios que manejan consultas, inserciones y actualizaciones en la base de datos.

7.1 Acceso a Datos a través de Interfaces

En nuestro proyecto, hemos definido una serie de interfaces que describen los métodos necesarios para interactuar con diferentes tablas de la base de datos.

Las interfaces incluyen:

- IUsuarioService: Define métodos para gestionar usuarios en la base de datos.
- IEmpresaService: Define métodos para gestionar empresas en la base de datos.
- IEventoService: Define métodos para gestionar eventos en la base de datos.
- IUbicacionService: Define métodos para gestionar ubicaciones en la base de datos.
- ICategoriaService: Define métodos para gestionar categorías en la base de datos.
- IDescuentoService: Define métodos para gestionar descuentos en la base de datos.

7.2 Implementación de los Servicios

Cada una de estas interfaces se implementa mediante clases concretas que utilizan ADO.NET para realizar las operaciones de consulta, inserción y actualización en la base de datos. Por ejemplo, UsuarioService, EmpresaService, EventoService y otros implementan sus respectivas interfaces y proporcionan la lógica para trabajar con los datos.

7.3 Uso en el Controlador y Vistas

En nuestros controladores y vistas, utilizamos la inyección de dependencias para acceder a estos servicios implementados. Por ejemplo, en un controlador, podemos tener el siguiente código:

```
public class UsuarioController : Controller
{
    private readonly IUsuarioService _usuarioService;

    public UsuarioController(IUsuarioService usuarioService)
    {
        _usuarioService = usuarioService;
    }
```

7.4 Beneficios del Enfoque

Este enfoque de servicios implementados con ADO.NET y la inyección de dependencias nos brinda varios beneficios, como:

- **Separación de Responsabilidades:** Mantenemos una separación clara entre la lógica de negocio y el acceso a la base de datos.
- **Reutilización de Código**: Al definir interfaces y servicios, podemos reutilizar la lógica de acceso a datos en diferentes partes de la aplicación.
- **Facilita las Pruebas Unitarias**: Podemos realizar pruebas unitarias en los servicios sin depender de la base de datos real.