Ciclo Superior en Desarrollo de Aplicaciones Web

Módulo Proyecto

TaskTrack

Autor/es: Manuel Falagán, Kai Peña

Tutor/es: Roberto Fernández

Madrid, Junio 2024

Resumen

Este proyecto de fin de grado presenta el diseño y desarrollo de una aplicación web centrada en la robustez y calidad del código para la gestión de calendarios y eventos. La aplicación permite a los usuarios registrarse, iniciar sesión, crear y clasificar eventos por prioridad mediante un sistema de colores, y compartir eventos de manera eficiente. La arquitectura del sistema se basa en microservicios conectados mediante API REST, utilizando contenedores Docker desplegados en AWS para asegurar escalabilidad y fiabilidad.

El backend de la aplicación está construido con ASP.NET Core y C#, siguiendo una arquitectura de servicios y el patrón Modelo-Vista-Controlador (MVC), con principios de inyección de dependencias y separación de preocupaciones. La autenticación se maneja mediante JSON Web Tokens (JWT) y la persistencia de datos se realiza a través de Entity Framework Core y una base de datos MySQL. El frontend, desarrollado con Angular, proporciona una interfaz de usuario dinámica y responsiva. Este proyecto demuestra la implementación técnica de una solución robusta y eficiente, asegurando el cumplimiento de normativas legales y la optimización de costos mediante el uso de tecnologías de código abierto y automatización con GitHub Actions.

Índice

Listado de abreviaturas, siglas y acrónimos

AWS	Amazon Web Services.
CDTI	Centro para el Desarrollo Tecnológico Industrial.
CI	Continuous Integration (Integración Continua).
CRUD	Create, Read, Update, Delete (Crear, Leer, Actualizar, Eliminar).
CD	Continuous Deployment (Despliegue Continuo).
GDPR	General Data Protection Regulation (Reglamento General de Protección de Datos).
IDE	Integrated Development Environment (Entorno de Desarrollo Integrado).
IoC	Inversión de Control.
JWT	JSON Web Token.
MVC	Modelo-Vista-Controlador.
ORM	Object-Relational Mapping.
QA	Quality Assurance (Aseguramiento de la Calidad).
SPA	Single Page Application.

Glosario

1. **API REST**: (Representational State Transfer) Arquitectura de software que define un conjunto de restricciones para crear servicios web.
2. **Angular**: Framework de desarrollo de aplicaciones web dinámicas y responsivas, basado en componentes.
3. **ASP.NET Core**: Framework de desarrollo web de código abierto desarrollado por Microsoft para construir aplicaciones modernas y conectadas a Internet.
4. **AWS**: (Amazon Web Services) Plataforma de servicios en la nube que proporciona infraestructura escalable y de alta disponibilidad.
5. **Backend**: Parte de una aplicación que gestiona la lógica de negocio y la interacción con la base de datos.
6. **CD**: (Continuous Deployment) Práctica de ingeniería de software donde el código se despliega automáticamente después de pasar todas las pruebas necesarias.
7. **CI**: (Continuous Integration) Práctica de ingeniería de software que permite integrar cambios de código frecuentemente y ejecutar pruebas automáticas para detectar errores rápidamente.
8. **Docker**: Plataforma que permite desarrollar, enviar y ejecutar aplicaciones dentro de contenedores.
9. **Docker Compose**: Herramienta que permite definir y ejecutar aplicaciones Docker multicontenedor.
10. **Entity Framework**: (EF) Framework de mapeo objeto-relacional (ORM) para .NET que permite interactuar con bases de datos utilizando objetos .NET.
11. **Frontend**: Parte de una aplicación que interactúa directamente con el usuario, presentando la interfaz gráfica.
12. **GitHub Actions**: Plataforma de integración continua y entrega continua (CI/CD) que permite automatizar flujos de trabajo directamente desde GitHub.
13. **Inyección de dependencias**: Patrón de diseño que permite desacoplar la creación de objetos de su uso, facilitando el mantenimiento y la prueba de aplicaciones.
14. **JWT**: (JSON Web Token) Estándar abierto basado en JSON para crear tokens de acceso que permiten la autenticación sin estado.
15. **Microservicios**: Estilo de arquitectura que estructura una aplicación como un conjunto de servicios pequeños y autónomos, cada uno ejecutando un proceso único.
16. **MySQL**: Sistema de gestión de bases de datos relacional de código abierto.
17. **OpenAPI**: Especificación para definir interfaces de programación de aplicaciones (API) que permite a los humanos y a los sistemas descubrir y entender las capacidades de un servicio sin acceso al código fuente.
18. **ORM**: (Object-Relational Mapping) Técnica de programación para convertir datos entre sistemas incompatibles usando lenguajes orientados a objetos.
19. **SPA**: (Single Page Application) Aplicación web que interactúa con el usuario cargando una única página HTML y actualizando dinámicamente el contenido conforme es necesario.

1 Introducción

Este proyecto de fin de grado presenta el diseño y desarrollo de una aplicación web avanzada para la gestión de calendarios y eventos, con el objetivo de proporcionar una herramienta flexible y personalizada. La aplicación permite a los usuarios registrarse, iniciar sesión, crear y clasificar eventos según su prioridad para gestionar su tiempo de manera eficiente.

El documento se estructura en varios apartados que cubren desde el planteamiento del problema hasta la justificación y los objetivos del proyecto. Se describen los requisitos técnicos y legales necesarios para llevar a cabo el desarrollo, incluyendo el uso de tecnologías modernas como .NET Core para el backend, Angular para el frontend, y Docker para la implementación de contenedores. También se discuten las ayudas y subvenciones económicas disponibles, la viabilidad técnica y económica, y los recursos necesarios. La planificación del proyecto se detalla mediante un cronograma y un presupuesto estimado. Finalmente, se presenta la documentación técnica de la aplicación, las conclusiones y las posibles líneas futuras de desarrollo.

2 Planteamiento del problema

La gestión eficaz del tiempo es un desafío constante tanto para individuos como para organizaciones. Sin una guía temporal, un estudiante podría olvidar un examen crucial, un profesional podría perder una reunión importante y una familia podría ver interrumpidos sus planes por compromisos olvidados.

La falta de organización diaria, tanto a nivel personal como profesional, se convierte en un obstáculo que nubla nuestra capacidad para priorizar tareas, delegar responsabilidades y enfrentarnos al día a día. El caos resultante no solo puede tener consecuencias graves en nuestro desempeño académico y profesional, sino que también puede erosionar nuestra salud mental y bienestar general al generar estrés y ansiedad.

3 Justificación.1 Beneficiarios

Nuestra solución busca ofrecer una herramienta que no solo permita organizar y visualizar eventos de manera eficiente, sino que también adapte la experiencia a las necesidades individuales de cada usuario. De esta forma, en este proyecto proponemos el desarrollo de una aplicación web que permite a sus usuarios clasificar eventos según la prioridad que se le quiera asignar para gestionar eficazmente su tiempo de manera flexible y personalizada con un diseño moderno e intuitivo.

Pero el principal punto sobre el que gira este proyecto es asegurar la calidad y robustez del código, utilizando principios y patrones de diseño sólidos y tecnologías modernas que garanticen un desarrollo mantenible y escalable. Además de minimizar errores y optimizar la eficiencia del desarrollo con la automatización de pruebas y despliegues, queremos asegurarnos de ofrecer una implementación de contenedores que asegure consistencia entre los entornos,

proporcionando una infraestructura fiable y escalable. Este enfoque en la calidad del código asegura que la aplicación no solo cumpla con sus objetivos funcionales, sino que también ofrezca una base sólida y extensible para futuras mejoras y funcionalidades adicionales.

Los beneficiarios de este proyecto incluyen:

- Profesionales que requieren un control detallado y compartido de sus agendas.
- Estudiantes que necesitan organizar sus tareas y eventos académicos.
- Empresas que requieren coordinación y planificación compartida de actividades entre equipos y departamentos.
- Usuarios individuales que desean mejorar su organización personal y gestión del tiempo.
- Organizaciones educativas para planificar actividades, clases y eventos académicos.

Esta aplicación ofrece una solución integral que permite una organización más eficiente y colaborativa al brindar las herramientas necesarias para que cada usuario gestione su tiempo a su manera, adaptándose a las diversas necesidades de cada uno.

4 Objetivos

Los objetivos que esperamos conseguir según el planteamiento del proyecto son los siguientes:

- Desarrollar una aplicación web eficiente y robusta.
- Implementar una arquitectura de software óptima.
- Desarrollar código limpio y fácil de interpretar.
- Diseñar una estructura de archivos y carpetas modular que simplifique su mantenibilidad y escalabilidad.
- Permitir el registro e inicio de sesión de usuarios.
- Incluir una agenda personal para agregar, editar y eliminar eventos.
- Implementar una funcionalidad para manejar eventos según su prioridad mediante el uso de colores elegidos por el usuario.
- Desarrollar un dashboard de eventos próximos.
- Diseñar una interfaz de usuario intuitiva y personalizable.
- Implementar un backend sólido y fiable que comunique correctamente con el frontend.
- Automatizar la generación de servicios y despliegue con OpenAPI, Codegen y GitHub Actions.
- Desplegar la aplicación en AWS usando contenedores Docker.
- Evaluar la usabilidad, rendimiento y seguridad de la aplicación.

5 Requisitos técnicos y legales

Por un lado, los requisitos técnicos necesarios para llevar a cabo el proyecto son los siguientes:

Backend: **.NET**

La elección de .NET como tecnología para el backend se basa en su robustez, escalabilidad y soporte a largo plazo. .NET permite el desarrollo de aplicaciones de alto rendimiento, y su compatibilidad con múltiples plataformas lo hace ideal para este proyecto. Además, forma parte de la formación recibida durante el curso.

Frontend: **Angular**

Angular es un framework que nos va a permitir hacer que nuestra aplicación sea dinámica y responsiva. Su arquitectura basada en componentes facilita la modularidad y el mantenimiento del código, características que consideramos prioritarias en el desarrollo de este proyecto. Además, separa el frontend del backend, permitiendo que cada uno sea una aplicación independiente.

Interconexión: **API REST**

La implementación de API REST garantiza una comunicación eficiente entre el frontend y el backend, permitiendo una integración fluida y escalable de los servicios.

Generación de servicios: **OpenAPI** y **Codegen**

El uso de OpenAPI y Codegen para la generación de servicios asegura que las APIs estén bien documentadas y estandarizadas, facilitando su integración y mantenimiento. Cualquier cambio en la API o su uso se sincroniza automáticamente entre el frontend y el backend.

Automatización: **GitHub Actions**

GitHub Actions permite la automatización de tareas de desarrollo, como la generación de servicios y la integración continua, mejorando la eficiencia del desarrollo y reduciendo errores.

ORM: **Entity Framework**

Entity Framework facilita la interacción con la base de datos, permitiendo un desarrollo más rápido y eficiente mediante la manipulación de datos a través de objetos .NET.

Base de datos: **MySQL**

Para implementar una base de datos relacional robusta y garantizar la integridad y disponibilidad de los datos, utilizamos MySQL, una opción confiable ampliamente utilizada para el almacenamiento de información.

Contenedores: **Docker** y **Docker Compose**

El uso de Docker y Docker Compose permite crear entornos de desarrollo y producción consistentes y aislados, mejorando la portabilidad y escalabilidad de la aplicación. Así mismo, se facilita la escalabilidad horizontal y el balanceo de carga mediante Kubernetes.

Despliegue: **AWS**

AWS ofrece una infraestructura flexible y escalable para el despliegue de la aplicación, asegurando alta disponibilidad y rendimiento.

Autenticación y autorización: **JWT**

La autenticación mediante JWT es *stateless* (sin estado), lo que permite que las peticiones se manejen sin problemas por distintos servidores, garantizando así una arquitectura modular y escalable. Esta técnica mejora la seguridad y eficiencia en la gestión de sesiones de usuario.

Por otro lado, los requisitos legales incluyen:

- ➢ Protección de datos: Cumplimiento con GDPR para la gestión de datos de usuarios.
- ➢ Licencias de software: Uso de tecnologías y bibliotecas con licencias adecuadas.
- ➢ Prevención de riesgos laborales: Aplicación de políticas de seguridad y salud laboral durante el desarrollo para garantizar un entorno de trabajo seguro y adecuado.

6 Ayudas y subvenciones económicas

Para la financiación del proyecto, se pueden explorar diversas ayudas y subvenciones disponibles en España. Programas como el Centro para el Desarrollo Tecnológico Industrial, que ofrece apoyo financiero a proyectos de investigación y desarrollo, pueden ser una fuente valiosa. Además, el programa "Horizon Europe" y el Plan de Recuperación, Transformación y Resiliencia de la Unión Europea proporcionan fondos para iniciativas tecnológicas innovadoras. A nivel local, las subvenciones para la digitalización de pymes ofrecidas por la Comunidad de Madrid, y las ayudas del Ministerio de Asuntos Económicos y Transformación Digital, enfocadas en la innovación tecnológica, pueden brindar un soporte financiero crucial. Aprovechar estas oportunidades puede reducir significativamente los costos del proyecto y asegurar los recursos necesarios para su desarrollo y ejecución.

7 Viabilidad

El análisis de viabilidad económica de este proyecto considera varios factores clave que influyen en los costos totales y el tiempo de desarrollo, integrando también su viabilidad técnica y legal.

Viabilidad técnica

La elección de .NET para el backend y Angular para el frontend asegura una base sólida y escalable para el desarrollo de la aplicación, aprovechando tecnologías modernas y ampliamente utilizadas con comunidades de soporte activas. Esto facilita la resolución de problemas y la implementación de nuevas funcionalidades, garantizando un amplio soporte y documentación extensiva. La automatización mediante GitHub Actions, junto con el uso de contenedores Docker, asegura un proceso de desarrollo eficiente y libre de errores. Docker Compose permite crear entornos de desarrollo consistentes, mejorando la calidad del código y facilitando las pruebas. Además, AWS proporciona la escalabilidad y fiabilidad necesarias para un despliegue robusto, asegurando que la aplicación pueda manejar una carga variable de usuarios y crecer según las necesidades.

Viabilidad económica

El uso de tecnologías de código abierto como .NET Core, Angular, MySQL y Docker reduce significativamente los costos de licencias y proporciona una comunidad de soporte activa. La implementación de contenedores Docker permite crear entornos consistentes y optimiza el uso de recursos del servidor, reduciendo los costos de infraestructura en AWS. La automatización mediante GitHub Actions minimiza el tiempo y los recursos necesarios para tareas repetitivas como la integración y el despliegue continuo, permitiendo que el equipo de desarrollo se enfoque en la implementación de nuevas funcionalidades.

Viabilidad legal

El cumplimiento de las normativas de protección de datos, como el GDPR, y las regulaciones de seguridad y salud en el ámbito laboral aseguran que el proyecto se desarrolle dentro de los marcos legales establecidos en España, evitando posibles sanciones. Además, el uso de tecnologías con licencias adecuadas garantiza que no haya infracciones de propiedad intelectual.

Estos factores combinados no solo protegen a los usuarios y desarrolladores, sino que también fortalecen la integridad y la credibilidad del proyecto, asegurando un desarrollo legalmente sólido y responsable que resulte económicamente rentable.

8 Recursos

Recursos Humanos

- Desarrollador de backend: Con conocimientos en .NET y Entity Framework, encargado de implementar la lógica de negocio y la interacción con la base de datos.
- Desarrollador de frontend: Responsable del desarrollo de la interfaz de usuario y la experiencia del usuario, con entendimiento en Angular.
- DevOps: Encargado de la configuración y mantenimiento de contenedores Docker y la infraestructura en AWS.
- Analista de QA: Responsable de realizar pruebas exhaustivas para asegurar la calidad del software.

Recursos Materiales

- Servidores AWS: Para el despliegue y operación de la aplicación.
- Equipos de desarrollo: Computadoras y software necesario para el desarrollo y pruebas.
- Herramientas de desarrollo: IDEs, herramientas de automatización, y sistemas de control de versiones.

Estos factores aseguran una planificación financiera eficiente y un desarrollo sostenible, proporcionando una base sólida para la implementación y el éxito del proyecto.

9 Planificación

La planificación se estructurará en las siguientes fases:

1. Investigación y formación: 1 mes

2. Análisis de requisitos: media semana
3. Diseño del sistema: media semana
4. Desarrollo del backend: 4 semanas
5. Desarrollo del frontend: 4 semanas
6. Integración y Pruebas: 1 semana
7. Despliegue: 1 semana
8. Evaluación y Ajustes: 2 semanas

10 Presupuesto económico

El presupuesto del proyecto se desglosa en varios componentes clave, que incluyen recursos humanos, recursos materiales y otros costes operativos. A continuación, se presenta una estimación detallada de los costos asociados solo al desarrollo y despliegue de la aplicación.

- Recursos humanos:
  - Equipo de desarrollo y diseño: Formado por dos desarrolladores full stack, con 2.000€ por mes, estimando 3 meses de trabajo.
- Recusos materiales:
  - Ordenadores para el desarrollo: 1.500€ por ordenador, total de 3.000€
  - Dominio: 10€ por año
  - IP fija: 10€ por año
  - Base de datos (Amazon RDS MySQL): 5€ por mes
  - Computación (Amazon EC2): 10€ por mes

Total: 9.060€

Este presupuesto proporciona una visión clara y detallada de los costos asociados al desarrollo y despliegue de la aplicación, asegurando una planificación financiera adecuada y eficiente para la ejecución del proyecto.

11 Documentación técnica

11.1 Requisitos

11.1.1 Requisitos funcionales

- Registro e inicio de sesión de usuarios: Los usuarios deben poder registrarse e iniciar sesión en la aplicación de manera segura.
- Calendario personalizado: Los usuarios pueden crear, editar y clasificar eventos del calendario según su prioridad mediante un sistema de colores.
- Dashboard de eventos: Visualización de eventos próximos de manera clara y accesible.
- Perfil de usuario: Para editar los datos de la cuenta y personalizarla.

11.1.2 Requisitos no funcionales

- ➢ Usabilidad: Interfaz amigable y fácil de usar, diseñada para mejorar la experiencia del usuario y fomentar que llegue a un público objetivo amplio.
- ➢ Eficiencia: Respuesta rápida y eficiente a las solicitudes de los usuarios.
- ➢ Rendimiento: Capacidad de manejar múltiples usuarios simultáneamente sin degradación del servicio.
- ➢ Seguridad: Protección de datos personales y cumplimiento de normativas de seguridad y privacidad.
- ➢ Modularidad: Arquitectura del sistema diseñada para ser altamente modular, permitiendo la fácil adición y modificación de componentes.
- ➢ Prioridad al open source: Uso de tecnologías y herramientas open source para fomentar la transparencia, colaboración y reducción de costos.
- ➢ Automatización: Implementación de procesos automatizados para pruebas, integración y despliegue continuo, asegurando una mayor eficiencia y reducción de errores.

11.1.3 Requisitos de bases de datos

La base de datos se compone de las siguientes tablas:

Evento:

Id	1	int	[v]	[v]	PRI
Title	2	longtext	[v]	[ ]	
Description	3	longtext	[ ]	[ ]	
DueDate	4	date	[v]	[ ]	
Priority	5	int	[v]	[ ]	
UserId	6	int	[v]	[ ]	MUL

Token:

Id	1	int	[v]	[v]	PRI
Value	2	longtext	[v]	[ ]	
Expiration	3	datetime(6)	[v]	[ ]	
UserId	4	int	[v]	[ ]	MUL
IsValid	5	tinyint(1)	[v]	[ ]	

Usuario:

Id	1	int	[v]	[v]	PRI
Username	2	longtext	[ ]	[ ]	
Email	3	longtext	[ ]	[ ]	
Password	4	longtext	[v]	[ ]	

11.2 Arquitectura

El sistema se basa en una arquitectura de microservicios conectados mediante API REST y sigue el patrón de diseño MVC para el backend y el patrón de Single Page Application (SPA) para el frontend. Cada microservicio se despliega como un contenedor Docker independiente, lo que facilita la escalabilidad horizontal. Este enfoque permite añadir nuevas funcionalidades y manejar una alta carga de usuarios de manera eficiente.

El backend de la aplicación sigue el patrón de diseño MVC, donde los controladores manejan las solicitudes HTTP entrantes, las vistas presentan los datos al usuario y los modelos representan las entidades de datos y gestionan la lógica de negocio. Este patrón organiza la estructura del código de manera clara y facilita la separación de responsabilidades.

El frontend está desarrollado como una SPA utilizando Angular. Este enfoque permite que la aplicación se cargue una vez y las actualizaciones se manejen dinámicamente en el navegador, proporcionando una experiencia de usuario fluida y rápida. La SPA se comunica con el backend a través de API REST, lo que permite una integración eficiente y modular.

Cada funcionalidad principal de la aplicación se implementa como un microservicio independiente, desplegado en contenedores Docker. Estos microservicios pueden escalarse horizontalmente según la demanda, mejorando la eficiencia y la capacidad de manejo de usuarios concurrentes.

La aplicación sigue los siguientes patrones y principios arquitectónicos:

1. Inversión de control e inyección de dependencias: Los servicios (AuthService, EventService, JwtService) se inyectan en los controladores a través de la inyección de dependencias. Este patrón desacopla los componentes y permite gestionar las dependencias de manera eficiente, facilitando el mantenimiento y las pruebas. IoC asegura que las dependencias se resuelvan automáticamente, promoviendo una arquitectura más flexible y modular.
2. Separación de preocupaciones: La lógica de negocio se separa de los controladores y se encapsula en los servicios correspondientes. Esto mejora la organización del código y facilita la escalabilidad y el mantenimiento.
3. Arquitectura en capas: La aplicación se divide en capas lógicas:
    a. Controladores: Manejan las solicitudes HTTP entrantes y definen los puntos de entrada de la API. Ejemplos incluyen AuthController.cs para la autenticación de usuarios y EventController.cs para operaciones CRUD de eventos.
    b. Servicios: Implementan la lógica de negocio. AuthService.cs maneja la autenticación, EventService.cs gestiona eventos y JwtService.cs se encarga de los tokens JWT.
    c. Modelos: Representan las entidades de datos, como usuarios, tokens y eventos (User.cs, Token.cs, Event.cs).
    d. Contexto de Base de Datos: ApplicationDbContext.cs hereda de DbContext de Entity Framework Core para configurar y acceder a la base de datos MySQL.
    e. Autenticación JWT: Se utiliza JWT para la autenticación de usuarios, permitiendo una autenticación sin estado y segura.

La autenticación se maneja mediante JWT, y la persistencia de datos se realiza a través de Entity Framework Core y una base de datos MySQL. La implementación de contenedores Docker y el despliegue en AWS aseguran escalabilidad, flexibilidad y fiabilidad, proporcionando una solución robusta y eficiente para la gestión de calendarios y eventos.

11.3 Diseño

El diseño de la aplicación se centrará en una interfaz intuitiva y amigable, utilizando Angular para el frontend. Se definirán patrones de diseño para asegurar la consistencia y facilidad de

mantenimiento del código. El diseño también considerará principios de diseño responsive para asegurar una experiencia óptima en diferentes dispositivos. Para ello, se utilizará Bootstrap, un framework CSS ampliamente adoptado, para agilizar el desarrollo de una interfaz estética además de responsiva. Bootstrap ofrece una amplia variedad de componentes predefinidos y un sistema de grid flexible, lo que facilita la creación de un diseño coherente y adaptativo, garantizando una experiencia de usuario uniforme en diferentes resoluciones y dispositivos.

11.4 Implementación

La implementación se realizará siguiendo metodologías ágiles, permitiendo iteraciones rápidas y la integración continua de nuevas funcionalidades mediante GitHub Actions. El uso de contenedores Docker asegurará que el entorno de desarrollo sea consistente con el entorno de producción, minimizando los problemas de compatibilidad. Además, se adoptará un enfoque incremental e iterativo, organizando las tareas y fases del desarrollo en un diagrama de Gantt. Este enfoque permitirá una planificación clara y estructurada, facilitando el seguimiento del progreso y asegurando que cada iteración aporte mejoras significativas y funcionales a la aplicación.

11.5 Pruebas

Se realizarán pruebas en varias fases:

- Pruebas unitarias: Para asegurar que cada componente funciona correctamente de manera aislada. Esto incluye pruebas de componentes Angular para verificar la funcionalidad de la interfaz de usuario, pruebas de endpoints de la API para asegurar que las solicitudes y respuestas son correctas, y pruebas de servicios backend para validar la lógica de negocio.

- Pruebas de integración: Para verificar que los componentes interactúan correctamente entre sí, asegurando que el frontend y el backend se comunican de manera eficiente y que los datos fluyen correctamente a través del sistema.

- Pruebas de rendimiento: Para evaluar el comportamiento del sistema bajo carga y asegurar que puede manejar múltiples usuarios simultáneamente sin degradación del servicio, garantizando una experiencia de usuario consistente y de alta calidad bajo diversas condiciones de uso.

12 Conclusiones

La aplicación desarrollada cumple satisfactoriamente con los objetivos planteados, proporcionando una herramienta robusta y flexible para la gestión del tiempo y la organización de eventos. Se ha logrado desarrollar una aplicación web eficiente implementando una arquitectura de software óptima basada en microservicios, que asegura escalabilidad y facilidad de mantenimiento.

La calidad del código se ha priorizado mediante la adopción de principios de diseño sólidos como la inyección de dependencias y la separación de preocupaciones. Se ha desarrollado código limpio y fácil de interpretar, lo que facilita la colaboración entre desarrolladores y la futura ampliación de funcionalidades. La estructura modular de archivos y carpetas simplifica tanto la mantenibilidad como la escalabilidad de la aplicación.

El registro e inicio de sesión de usuarios se han implementado de manera segura utilizando JWT para la autenticación, permitiendo una gestión eficiente de las sesiones de usuario. La agenda personal permite agregar, editar y eliminar eventos, y la funcionalidad de manejo de eventos según su prioridad mediante colores elegidos por el usuario mejora significativamente la organización y visualización de los eventos.

El backend sólido y fiable, desarrollado en ASP.NET Core, asegura una comunicación correcta con el frontend basado en Angular, garantizando una experiencia de usuario fluida. La automatización de la generación de servicios y despliegue con OpenAPI, Codegen y GitHub Actions ha permitido una integración continua y eficiente, minimizando errores y mejorando la calidad del desarrollo.

Finalmente, el despliegue de la aplicación en AWS utilizando contenedores Docker asegura una infraestructura escalable y flexible, capaz de manejar una alta carga de usuarios. La evaluación de la usabilidad, rendimiento y seguridad ha confirmado que la aplicación no solo cumple con sus objetivos funcionales, sino que también ofrece una solución eficiente, segura y de alta calidad para la gestión del tiempo y la organización de eventos.

13 Líneas futuras

En futuras versiones, se podrían considerar diversas mejoras y funcionalidades adicionales para personalizar aún más la aplicación y adaptarla a las necesidades y gustos individuales de los usuarios. Estas mejoras pueden incluir:

- Integración con otros servicios de calendario: Facilitar la sincronización y gestión centralizada de eventos desde diferentes plataformas, como Google Calendar, Outlook y Apple Calendar, proporcionando una experiencia unificada para los usuarios.
- Aplicaciones móviles: Desarrollar aplicaciones nativas para iOS y Android, permitiendo a los usuarios acceder y gestionar sus calendarios desde cualquier lugar y en cualquier momento. Las aplicaciones móviles ofrecerán notificaciones push, acceso offline y una experiencia de usuario optimizada para dispositivos móviles.
- Personalización avanzada: Permitir a los usuarios personalizar aún más su experiencia con opciones como temas, fuentes, y disposición de la interfaz. Los usuarios podrían configurar la apariencia y el comportamiento de la aplicación para adaptarla mejor a sus preferencias individuales.
- Funciones avanzadas de análisis de datos: Implementar herramientas de análisis que permitan a los usuarios obtener insights sobre el uso de su tiempo y la distribución de sus actividades. Estos análisis podrían incluir gráficos y reportes personalizados que muestren tendencias, patrones y áreas de mejora.
- Recordatorios inteligentes y notificaciones personalizadas: Desarrollar funcionalidades avanzadas de recordatorios que se adapten a las rutinas y preferencias del usuario, enviando notificaciones personalizadas y contextuales para asegurar que los usuarios nunca olviden un evento importante.

Estas líneas futuras tienen el potencial de mejorar significativamente la experiencia del usuario, haciéndola más personalizada, eficiente y adaptativa. Al continuar evolucionando y expandiendo las capacidades de la aplicación, se asegurará que siga siendo una herramienta valiosa y relevante para la gestión del tiempo y la organización de eventos.

Referencias

- Microsoft Docs: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0
- Angular: https://angular.io/docs
- Docker: https://docs.docker.com/
- GitHub: https://docs.github.com/en/actions
- Amazon Web Services: https://docs.aws.amazon.com/
- OpenAPI Initiative: https://www.openapis.org/
- MySQL.: https://dev.mysql.com/doc/
- JWT.io: Introduction to JSON Web Tokens - https://jwt.io/introduction/
- Entity Framework Core. (2023). EF Core Documentation. https://docs.microsoft.com/en-us/ef/core/
- Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.