

INTRODUCCIÓN

Este Trabajo de Fin de Grado presenta el diseño, desarrollo e implementación de la aplicación NoMoreMeetings, que proporciona un análisis detallado del consumo de tiempo de reunión en base a los datos extraídos de los calendarios de Google de los usuarios.

MOTIVACIÓN

En el entorno laboral actual, la gestión eficaz del tiempo es esencial, y ser capaz de comprender cómo empleamos nuestro tiempo puede ser una herramienta de gran valor capaz de mejorar nuestra productividad y el equilibrio entre la vida laboral y personal.

El auge de las reuniones virtuales, especialmente debido al auge del trabajo remoto causado por la pandemia de COVID-19 ha provocado que muchos trabajadores experimenten una sensación de "fatiga de reuniones". Esta fatiga a menudo ocurre cuando las reuniones ocupan demasiado tiempo de la jornada laboral, lo que deja poco espacio para concentrarse y ser productivo. Sin embargo, a menudo nos es difícil cuantificar cuánto tiempo pasamos realmente en reuniones.

Además, las soluciones de calendario actuales, como Google Calendar y Microsoft Outlook, no brindan suficientes funciones ni opciones de personalización para analizar el tiempo dedicado a las reuniones.

Partiendo de esta situación, la motivación de este proyecto es proporcionar una herramienta fácil de usar que permita a los usuarios tener una visión detallada de cuánto tiempo pasan en reuniones, de una forma adaptada a cada situación personal y, en definitiva, proporcionar las estadísticas necesarias para tomar decisiones sobre el uso del tiempo de trabajo.

OBJETIVOS

El objetivo principal de este proyecto es desarrollar una aplicación que permita a los usuarios de Google Calendar analizar y visualizar en detalle su uso del tiempo en las reuniones.

Los objetivos específicos son:

1. Proporcionar una vista detallada del tiempo dedicado a las reuniones aceptadas, marcadas como programadas y aquellas marcadas como "fuera de la oficina".
2. Permitir que los usuarios especifiquen sus horas de trabajo y excluyan ciertos períodos (por ejemplo, pausas para el almuerzo) del análisis.
3. Analizar el calendario para un cierto período determinado en días por parte del usuario.
4. Presentar datos de forma clara e intuitiva con texto y gráficos descriptivos.

ALCANCE

Este proyecto se centra en el diseño e implementación de una aplicación web que interactúe con Google Calendar. La aplicación podrá analizar y presentar datos sobre el tiempo de citas del usuario. Se proporcionará un desglose detallado del tiempo dedicado a los diferentes tipos de eventos del calendario y se ofrecerá la opción de excluir ciertos períodos de tiempo del análisis, como el tiempo dedicado a almorzar.

El proyecto no se integrará con otros sistemas de calendario que no sean Google Calendar. También se limitará a brindar análisis basados en datos históricos y no incluirá predicciones sobre el uso futuro del tiempo. Finalmente, si bien la portabilidad de los datos es una consideración importante, este proyecto no abordará los problemas relacionados con la agregación o exportación de datos para su uso posterior.

ALTERNATIVAS AL PROYECTO

Existen varias herramientas en el mercado que ofrecen características similares a este proyecto, pero ninguna cubre todas las necesidades y tareas, especialmente la edición.

1. **Google Calendar y Microsoft Outlook:** Ambas herramientas ofrecen funciones adicionales para el análisis del calendario. Sin embargo, su potencial es limitado. No permiten una revisión exhaustiva de los horarios de las reuniones, como distinguir entre reuniones aceptadas o de tipo "fuera de la oficina". Además, no permiten excluir del análisis ciertos períodos, como las pausas para el almuerzo.
2. **Herramientas de análisis de tiempo y productividad:** Existen varias herramientas en el mercado como RescueTime, Toggl y Timely que brindan un análisis detallado del uso del tiempo. Sin embargo, estas herramientas se centran en el seguimiento del tiempo dedicado a tareas con finalidades como cuantificación de esfuerzos u optimización del tiempo, en lugar del análisis y extracción de datos de tiempo gastado en reuniones en función de los datos del calendario.
3. **Herramientas de planificación de reuniones:** Clockwise, Clara y X.AI son algunas de las herramientas que ofrecen análisis de reuniones y/o funciones de programación inteligente. Aunque estas herramientas proporcionan algunos análisis útiles, como la identificación de conflictos de programación y sugerencias de horarios óptimos, ninguna de ellas ofrece el nivel de análisis detallado de las reuniones que este proyecto busca proporcionar.

COMPARACIÓN RESPECTO DE ALGUNAS ALTERNATIVAS

Funcionalidades	NoMoreMeetings	Google Calendar	Microsoft Outlook	RescueTime	Clockwise	Clara
Análisis de tiempo en reuniones	✓	✓	✓	✗	✓	✓
Distinguir entre tipos de reuniones	✓	✗	✗	✗	✗	✗
Excluir ciertos períodos de tiempo del análisis	✓	✗	✗	✓	✗	✗
Análisis basado en los datos del calendario	✓	✓	✓	✗	✓	✓

Seguimiento del tiempo dedicado a las tareas	X	X	X	✓	X	X
Análisis detallado de las reuniones	✓	X	X	X	✓	✓
Identificación de conflictos de programación	X	✓	✓	X	✓	✓
Sugerencias de horarios óptimos	X	X	X	X	✓	✓

Tabla 1: Comparativa entre NoMoreMeetings y las alternativas existentes

A pesar de la existencia de varias alternativas, ninguna satisface las necesidades específicas que NoMoreMeetings pretende cubrir.

- Google Calendar y Microsoft Outlook proporcionan análisis básicos de calendario, pero carecen de la capacidad para distinguir entre tipos de reuniones y de excluir ciertos períodos de tiempo del análisis.
- RescueTime, al igual que Toggl, aunque útil para el seguimiento del tiempo, no se integra con el calendario y, por tanto, no puede ofrecer análisis de reuniones.
- Clockwise y Clara, aunque proporcionan algunas funcionalidades que NoMoreMeetings no tiene, como la detección de conflictos de programación y la sugerencia de horarios óptimos, no realizan un análisis detallado del tiempo pasado en reuniones ni diferencian entre diferentes tipos de reuniones.

Estas carencias subrayan la necesidad de una solución más completa, personalizable y centrada en el análisis y la extracción de datos relevantes que puedan facilitar la toma de decisiones, objetivo último de NoMoreMeetings.

ASPECTOS TEÓRICOS

CONCEPTOS IMPORTANTES

AUTENTICACIÓN Y REGISTRO CON GOOGLE

Se contempla la implementación de un proceso de autenticación y registro basado en el uso de la cuenta de Google del usuario. Este proceso tendrá dos ventajas significativas: por un lado, proporcionará un mecanismo de autenticación seguro y de confianza, y por otro, permitirá el acceso a los datos del calendario de Google del usuario, necesarios para el funcionamiento de la aplicación.

OAuth2

El protocolo que se utilizará para el proceso de autenticación y autorización es OAuth2, un estándar ampliamente aceptado y utilizado para permitir la autorización segura en aplicaciones web, móviles y de escritorio. Este protocolo facilitará un “token de acceso” junto con un “token de refresco” que habrán de ser persistidos y que se utilizarán para autenticar las solicitudes a la API de Google Calendar.

ANÁLISIS DE LOS DATOS DEL CALENDARIO

Se llevará a cabo el análisis de los datos extraídos del calendario del usuario. Este análisis implicará el procesamiento de los eventos del calendario para calcular métricas relevantes, como el porcentaje de tiempo total gastado en reuniones, así como una categorización en base al tipo de reuniones, proporcionando así una visión clara de cómo se distribuye el tiempo de trabajo del usuario.

API DE GOOGLE CALENDAR

Para acceder a los datos del calendario, se utilizará la API de Google Calendar. Esta interfaz permitirá a la aplicación extraer los datos del calendario del usuario de una manera estructurada y segura.

PERSONALIZACIONES DE USUARIO

Se ha de permitir a los usuarios personalizar el análisis de su tiempo. Los usuarios podrán definir aspectos como su horario de trabajo, las horas de comida y el tiempo mínimo entre reuniones que se considerará como tiempo de reunión. Estos ajustes habrán de ser persistidos, de modo que las preferencias del usuario se conserven de cara a todos los análisis futuros que se realicen.

TECNOLOGÍAS UTILIZADAS

A lo largo del desarrollo del proyecto, se utilizarán diversas tecnologías para cubrir diferentes aspectos del sistema. A continuación, se proporciona una descripción breve de cada una de ellas:

BACKEND

- **Java 17:** Es el lenguaje de programación que se utilizará para el desarrollo del backend.
- **Micronaut 3.7.0:** Este es el framework que se utilizará para el desarrollo del backend.
- **PostgreSQL 15:** Es el sistema de gestión de bases de datos que se utilizará para el almacenamiento y gestión de datos.
- **Liquibase:** Se utilizará para la gestión de la base de datos, permitiendo un fácil mantenimiento y evolución del esquema de la base de datos.
- **JUnit 5, AssertJ y TestContainers:** Estas son las librerías que se utilizarán para la realización de tests unitarios y de integración, incluyendo aquellos que requieran de una base de datos.

FRONTEND

- **ECMAScript 2021:** Es la versión de JavaScript utilizada para el desarrollo del frontend.
- **React 18:** Es el framework que se utilizará para el desarrollo del frontend.
- **NPM 9.5:** Es la herramienta utilizada para la gestión de dependencias y la construcción del proyecto.
- **Axios 1.3:** Es la librería que se utilizará para la realización de peticiones HTTP desde el frontend.
- **MDB React 6 y MUI 5.12:** Son las librerías que se utilizarán para la creación de componentes de la interfaz de usuario.
- **Dayjs 1.11:** Es la librería que se utilizará para el manejo de fechas en el frontend.

HERRAMIENTAS DE DESARROLLO Y DESPLIEGUE

- **Git:** Es el sistema de control de versiones que se utilizará durante el desarrollo del proyecto.
- **GitHub:** Se utilizará como repositorio remoto durante el desarrollo del proyecto. Además, se utilizará GitHub Actions para la integración continua.
- **Azure:** Es el proveedor de servicios cloud que se utilizará para el despliegue de la aplicación. En él se desplegarán tanto el backend como el frontend, y también se alojará la instancia de PostgreSQL que se utilizará en producción.
- **Docker:** Se utilizará para el despliegue de la aplicación, tanto frontend como backend.
- **Qodana:** Es una herramienta de análisis de calidad del código de JetBrains que permite obtener reportes de calidad para detectar problemas, reportar sugerencias y detectar 'code smells'.
- **Google Calendar API:** Se utilizará para la extracción de datos del calendario de Google del usuario.

- **OAuth2:** Se utilizará como protocolo para la autenticación y autorización de los usuarios.
- **Gradle:** Se utilizará para la gestión de dependencias y la construcción del proyecto.

PLANIFICACIÓN INICIAL DEL PROYECTO

En este capítulo se detalla la planificación inicial del proyecto, la descomposición del trabajo, así como el presupuesto inicial.

PLANIFICACIÓN INICIAL

El proyecto se desarrollará entre el 22 de noviembre de 2022 y el 06 de junio de 2023. El total de horas de trabajo se estima en 281 horas. La jornada estimada por semana de trabajo es de unas 10h laborables, repartidas a razón de 2h viernes, 4h los sábados y 4h los domingos. Los martes se dedicarán 30 minutos a reuniones de seguimiento del proyecto.

La Tabla 1 ilustra la descomposición del trabajo de todo el proyecto, mostrando el código EDT de cada tarea, su nombre, la duración en horas y las fechas de inicio y finalización de cada tarea.

EDT	Descripción	Tiempo	Fecha de inicio	Fecha de fin
0	Desarrollo del sistema NoMoreMeetings	281 horas	22/11/2022	06/06/2023
1	Selección de lenguajes de programación	4 horas	22/11/2022	26/11/2022
2	Selección de framework	4 horas	26/11/2022	27/11/2022
3	Configuración del entorno de trabajo	4 horas	27/11/2022	02/12/2022
4	Desarrollo del backend	136 horas	02/12/2022	05/03/2023
4.1	Desarrollo del módulo de autenticación con los sistemas de Google	45 horas	02/12/2022	01/01/2023
4.2	Desarrollo del módulo de configuración del usuario	28 horas	01/01/2023	21/01/2023
4.3	Desarrollo del módulo de análisis del calendario	35 horas	21/01/2023	12/02/2023
4.4	Integración de módulos de backend	28 horas	14/02/2023	05/03/2023
5	Desarrollo del frontend	86 horas	05/03/2023	06/05/2023
5.1	Desarrollo de componentes y lógica compartida	28 horas	05/03/2023	25/03/2023
5.2	Desarrollo de la vista de configuración de usuario	27 horas	25/03/2023	15/04/2023
5.3	Desarrollo de la vista de Home	9 horas	15/04/2023	21/04/2023
5.4	Desarrollo de la vista del análisis de calendario	22 horas	21/04/2023	06/05/2023
6	Despliegue e integración en la nube	39 horas	06/05/2023	03/06/2023

6.1	Selección de tecnologías	4 horas	06/05/2023	07/05/2023
6.2	Configuración del entorno de trabajo	4 horas	07/05/2023	12/05/2023
6.3	Integración del backend	12 horas	12/05/2023	20/05/2023
6.4	Integración del frontend	12 horas	20/05/2023	28/05/2023
6.5	Despliegue del servidor	6 horas	28/05/2023	03/06/2023
7	Reuniones periódicas (bisemanales)	7 horas	22/11/2022	23/05/2023
7.1	Reunión 1	0.5 horas	22/11/2022	22/11/2022
7.2	Reunión 2	0.5 horas	06/12/2022	06/12/2022
7.3	Reunión 3	0.5 horas	20/12/2022	20/12/2022
7.4	Reunión 4	0.5 horas	03/01/2023	03/01/2023
7.5	Reunión 5	0.5 horas	17/01/2023	17/01/2023
7.6	Reunión 6	0.5 horas	31/01/2023	31/01/2023
7.7	Reunión 7	0.5 horas	14/02/2023	14/02/2023
7.8	Reunión 8	0.5 horas	28/02/2023	28/02/2023
7.9	Reunión 9	0.5 horas	14/03/2023	14/03/2023
7.10	Reunión 10	0.5 horas	28/03/2023	28/03/2023
7.11	Reunión 11	0.5 horas	11/04/2023	11/04/2023
7.12	Reunión 12	0.5 horas	25/04/2023	25/04/2023
7.13	Reunión 13	0.5 horas	09/05/2023	09/05/2023
7.14	Reunión 14	0.5 horas	23/05/2023	23/05/2023
8	Reunión final	1 hora	06/06/2023	06/06/2023

TABLA 2: Planificación temporal del proyecto

En las siguientes ilustraciones se puede ver la planificación temporal del proyecto en formato Gantt. En la ilustración se muestra la duración de las tareas del proyecto, exceptuando las reuniones periódicas. En la ilustración se muestra la planificación temporal de las reuniones periódicas.

TRABAJO FIN DE GRADO - NOMOREMEETINGS

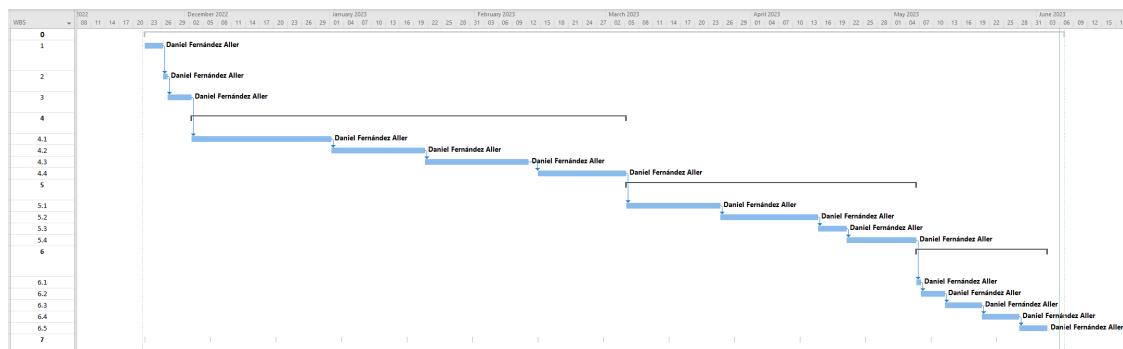


Ilustración: Diagrama de Gantt del proyecto, excluyendo las reuniones periódicas

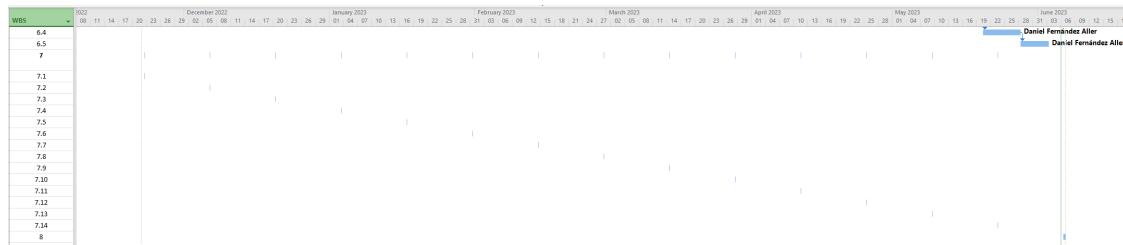


Ilustración: Diagrama de Gantt de las reuniones previstas

PRESUPUESTO INICIAL

El presupuesto inicial del proyecto se ha calculado con base en la planificación temporal del proyecto y a los costes de los recursos humanos y materiales. Se trata de un presupuesto interno, por lo que no se omite ningún coste.

El coste de los recursos humanos se ha calculado con base en el coste de la hora de trabajo del único desarrollador del proyecto, que es de 20€/hora. En la tabla se muestra el coste de este desarrollador.

Personal	Número	Coste salarial anual	Horas de trabajo anuales	Coste por hora
Desarrollador	1	75000€	1880	40,00€
Analista	1	122200€	1880	65,00€

TABLA 3: Coste por hora del personal del proyecto

El desarrollador necesitará una serie de equipos informáticos, licencias y servicios para poder desarrollar el proyecto. En la tabla se muestra el coste de estos recursos.

Recurso informático	Unidades	Coste unitario	Coste mes	Tipo	Plazo de amortización
Ordenador portátil	1	1000€	16,67€	Amortización	5 años
Microsoft Windows	1	120€	10€	Amortización	1 año
IntelliJ IDEA	1	0€	0€	Amortización	-
DataGrip	1	0€	0€	Amortización	-
Azure for Education	1	0€	0€	Amortización	-

GitHub	1	0€	0€	Amortización	-
Visual Studio Code	1	0€	0€	Amortización	-
TOTAL	-	-	26,67€	-	-

TABLA 4: Coste de los equipos informáticos y licencias del proyecto

Servicio	Coste mensual
Consumo de electricidad	35€
Consumo de Internet	25€
Consumo de agua	13€
Materiales de oficina	10€
Mantenimiento y reparaciones	15€
TOTAL	98€

TABLA 5: Coste de los servicios del proyecto

Para calcular el presupuesto inicial de costes se ha consignado una partida equivalente al 5% del coste total del proyecto para imprevistos. En la tabla se muestra el presupuesto inicial del proyecto.

Código	Partida de coste	Total
1	Selección de lenguajes de programación	260,00€
2	Selección de framework	260,00€
3	Configuración del entorno de trabajo	160,00€
4	Desarrollo del backend	5440,00€
5	Desarrollo del frontend	3440,00€
6	Despliegue e integración en la nube	1460,00€
7	Reuniones	280,00€
8	Reunión final	40,00€
9	Otros costes	872,69€
10	Imprevistos	620,63€
TOTAL	-	13592,32€

TABLA 6: Resumen del presupuesto inicial del proyecto

La tabla muestra la partida de coste referente a la selección de lenguajes de programación.

I1	Descripción	Cantidad	Unidades	Precio unitario	Total
01	Selección de lenguajes de programación	4	horas	65,00€	
					260,00€

TABLA 7: Partida de coste referente a la selección de lenguajes de programación

La tabla muestra la partida de coste referente a la selección de framework.

I1	Descripción	Cantidad	Unidades	Precio unitario	Total
01	Selección de framework	4	horas	65,00€	
					260,00€

TABLA 8: Partida de coste referente a la selección de framework

La tabla muestra la partida de coste referente a la configuración del entorno de trabajo.

I1	Descripción	Cantidad	Unidades	Precio unitario	Total
01	Configuración del entorno de trabajo	4	horas	40,00€	
					160,00€

TABLA 9: Partida de coste referente a la configuración del entorno de trabajo

La tabla muestra la partida de coste referente al desarrollo del backend.

I1	I2	Descripción	Cantidad	Unidades	Precio unitario	Total
01	4.1	Desarrollo del módulo de autenticación con los sistemas de Google	45 horas	40,00€	1800,00€	
02	4.2	Desarrollo del módulo de configuración del usuario	28 horas	40,00€	1120,00€	
03	4.3	Desarrollo del módulo de análisis del calendario	35 horas	40,00€	1400,00€	
04	4.4	Integración de módulos de backend	28 horas	40,00€	1120,00€	
						5440,00€

TABLA 10: Partida de coste referente al desarrollo del backend

La tabla muestra la partida de coste referente al desarrollo del frontend.

I1	I2	Descripción	Cantidad	Unidades	Precio unitario	Total
01	5.1	Desarrollo de componentes y lógica compartida	28 horas	40,00€	1120,00€	
02	5.2	Desarrollo de la vista de configuración de usuario	27 horas	40,00€	1080,00€	
03	5.3	Desarrollo de la vista de Home	9 horas	40,00€	360,00€	
04	5.4	Desarrollo de la vista del análisis de calendario	22 horas	40,00€	880,00€	
						3440,00€

TABLA 11: Partida de coste referente al desarrollo del frontend

La tabla muestra la partida de coste referente al despliegue e integración en la nube.

I1	I2	Descripción	Cantidad	Unidades	Precio unitario	Total
01	6.1	Selección de tecnologías	4 horas	65,00€	260,00€	
02	6.2	Configuración del entorno de trabajo	4 horas	40,00€	160,00€	
03	6.3	Integración del backend	12 horas	40,00€	480,00€	
04	6.4	Integración del frontend	12 horas	40,00€	480,00€	
05	6.5	Despliegue del servidor	6 horas	40,00€	240,00€	
						1460,00€

TABLA 12: Partida de coste referente al despliegue e integración en la nube

La tabla muestra la partida de coste referente a las reuniones periódicas.

I1	Descripción	Cantidad	Unidades	Precio unitario	Total
01	Reuniones periódicas	7	horas	40,00€	
					280,00€

TABLA 13: Partida de coste referente a las reuniones periódicas

La tabla muestra la partida de coste referente a la reunión final.

I1	Descripción	Cantidad	Unidades	Precio unitario	Total
01	Reunión final	1	horas	40,00€	
					40,00€

TABLA 14: Partida de coste referente a la reunión final

La tabla muestra la partida de coste referente a otros costes.

I1	Descripción	Cantidad	Unidades	Precio unitario	Total
01	Equipos informáticos y licencias	7	meses	26,67€	
02	Servicios	7	meses	98,00€	
					872,69€

TABLA 15: Partida de coste referente a otros costes

PRESUPUESTO INICIAL DEL CLIENTE

El siguiente presupuesto es el presupuesto inicial del cliente, que se ha calculado en base al presupuesto inicial del proyecto. Se ha aplicado un margen de beneficio del 25% sobre el presupuesto inicial del proyecto.

En este presupuesto se han omitido las partidas referentes a reuniones y otros costes, así como el dinero dedicado a imprevistos, ya que se refieren a procesos internos del proyecto. Estas partidas se incluirán en el presupuesto final del cliente prorrteadas en las partidas de análisis y desarrollo.

Además de dicho prorrteo, se ha aplicado el margen de beneficio del 25% a todas las partidas.

I1	Partida	Total
1	Selección de lenguajes de programación	324,75€
2	Selección de framework	324,75€
3	Configuración del entorno de trabajo	250,00€
4	Desarrollo del backend	7292,50€
5	Desarrollo del frontend	4632,50€
6	Despliegue e integración en la nube	2965,00€
SUBTOTAL	-	14790,00€
IVA 21%	-	3105,90€
TOTAL	-	17895,90€

TABLA 16: Resumen del presupuesto inicial del cliente

ANÁLISIS DEL SISTEMA

DESCRIPCIÓN FUNCIONAL

NoMoreMeetings es una aplicación web diseñada para optimizar la comprensión del tiempo dedicado a las reuniones laborales. Para facilitar el acceso, permite el registro y el inicio de sesión a través de la cuenta de Google del usuario.

Una vez que un usuario accede, la aplicación proporciona una página de configuración de perfil donde se pueden personalizar las horas de trabajo. Este ajuste es crucial para la aplicación, ya que los cálculos que realiza se basan en estas horas.

Además, NoMoreMeetings permite a los usuarios especificar qué calendario se debe tener en cuenta para los cálculos, proporcionando la capacidad de seleccionar aquel calendario que sea relevante para el trabajo. Los usuarios también pueden seleccionar el período temporal que desean utilizar para los cálculos, permitiendo realizar los cálculos solo en las horas de trabajo.

La aplicación se encarga de mostrar los datos de una manera fácil de entender. Ofrece una página de visualización de datos donde los usuarios pueden ver cuánto tiempo han gastado en reuniones. No solo proporciona el tiempo total, sino que también muestra una gráfica para ayudar a visualizar la distribución del tiempo. Así, se puede apreciar claramente el porcentaje de tiempo dedicado a reuniones frente al tiempo libre.

La aplicación ofrece la opción de configurar el tiempo entre reuniones que se cuenta como tiempo de reunión. Esta selección se refleja en los cálculos que realiza la aplicación.

REQUISITOS FUNCIONALES

1. La aplicación debe permitir a los usuarios registrarse utilizando su cuenta de Google.
2. La aplicación debe permitir a los usuarios registrados iniciar sesión utilizando su cuenta de Google.
3. La aplicación debe proporcionar una página de configuración de perfil para usuarios con sesión iniciada.
4. La aplicación debe permitir a los usuarios con sesión iniciada editar sus horas de trabajo en la página de configuración de perfil.
5. La aplicación debe reflejar los cambios en las horas de trabajo de un usuario en sus cálculos.
6. La aplicación debe permitir a los usuarios con sesión iniciada seleccionar qué calendarios utilizar para los cálculos en la página de configuración de perfil.
7. La aplicación debe reflejar los cambios en los calendarios seleccionados por un usuario en sus cálculos.
8. La aplicación debe permitir a los usuarios con sesión iniciada seleccionar el período temporal a utilizar para los cálculos en la página de configuración de perfil.
9. La aplicación debe reflejar los cambios en el período temporal seleccionado por un usuario en sus cálculos.
10. La aplicación debe proporcionar una página de visualización de datos para usuarios con sesión iniciada.
11. La aplicación debe mostrar a los usuarios con sesión iniciada la cantidad de tiempo que han gastado en reuniones en la página de visualización de datos.

12. La aplicación debe mostrar a los usuarios con sesión iniciada una gráfica que refleje el tiempo que han gastado en reuniones en la página de visualización de datos.
13. La aplicación debe mostrar a los usuarios con sesión iniciada el porcentaje de tiempo que han gastado en reuniones y el porcentaje de tiempo libre en la página de visualización de datos.
14. La aplicación debe permitir a los usuarios con sesión iniciada configurar el tiempo entre reuniones que cuenta como tiempo de reunión en la página de configuración de perfil.
15. La aplicación debe tener en cuenta el tiempo seleccionado por un usuario entre reuniones como tiempo de reunión al realizar los cálculos.

REQUISITOS NO FUNCIONALES

1. **Rendimiento:** La aplicación deberá procesar las solicitudes de los usuarios en un tiempo promedio inferior a 5 segundos.
2. **Disponibilidad:** La aplicación deberá tener una disponibilidad del 99%, lo que implica que puede permitirse hasta 87.6 horas de inactividad al año.
3. **Escalabilidad:** La aplicación deberá ser capaz de manejar un incremento del 50% en el número de usuarios concurrentes en 1 minuto sin degradación del rendimiento por encima del 10%.
4. **Accesibilidad:** La aplicación deberá cumplir con las pautas de accesibilidad WCAG 2.1 Nivel A, permitiendo su uso a usuarios con discapacidades.
5. **Seguridad:** La aplicación deberá cumplir con los estándares de seguridad y privacidad, incluyendo el Reglamento General de Protección de Datos (GDPR). Todos los datos deberán ser encriptados en tránsito y en reposo.
6. **Compatibilidad de navegador:** La aplicación deberá ser compatible y funcional en las últimas versiones de los navegadores más utilizados (Google Chrome, Mozilla Firefox, Safari y Microsoft Edge).
7. **Interoperabilidad:** La aplicación deberá interactuar de manera eficiente con las APIs de Google, en particular con Google Calendar y Google Authentication.
8. **Usabilidad:** La aplicación debe ser intuitiva y fácil de usar, con una interfaz de usuario que requiera un mínimo de capacitación para los nuevos usuarios.
9. **Mantenibilidad:** El código de la aplicación deberá seguir las buenas prácticas de programación, con una cobertura de pruebas unitarias superior al 70%.
10. **Resiliencia:** La aplicación deberá ser capaz de recuperarse de fallos y reanudar su funcionamiento normal en un tiempo inferior a 1 hora.

CASOS DE USO EXTRAÍDOS

REGISTRARSE CON LA CUENTA DE GOOGLE

Un usuario anónimo (no registrado) accede a la aplicación. La aplicación le ofrece la opción de iniciar sesión con la cuenta de Google. El usuario anónimo escoge iniciar sesión con la cuenta de Google y entonces la aplicación le redirige al sistema de autenticación de Google, donde le pedirá permisos para ver su calendario. Una vez completado el registro el sistema le redirige a la pantalla principal con la sesión iniciada.

INICIAR SESIÓN CON LA CUENTA DE GOOGLE

Un usuario registrado accede a la aplicación. La aplicación le ofrece las opciones de iniciar sesión con su cuenta de Google o registrarse (para usuarios no registrados). El usuario registrado selecciona iniciar sesión con su

cuenta de Google y entonces la aplicación le redirige al sistema de autenticación de Google. Una vez completado el inicio de sesión el sistema le redirige a la pantalla principal con la sesión iniciada.

CONFIGURAR EL TIEMPO DE TRABAJO

Un usuario con la sesión iniciada accede a la página de configuración de su perfil. La aplicación muestra las opciones para editar las horas de trabajo. El usuario indica que quiere editar sus horas de trabajo, introduce las nuevas horas y confirma los cambios. La aplicación actualiza el perfil del usuario con las nuevas horas de trabajo.

SELECCIONAR CALENDARIOS PARA CÁLCULOS

Un usuario con la sesión iniciada accede a la página de configuración de su perfil. La aplicación muestra las opciones para seleccionar el calendario a utilizar. El usuario indica que quiere seleccionar el calendario, elige el que quiere utilizar y confirma su selección. La aplicación actualiza el perfil del usuario con el calendario seleccionado.

SELECCIONAR EL PERIODO TEMPORAL PARA LOS CÁLCULOS

Un usuario con la sesión iniciada accede a la página de configuración de su perfil. La aplicación muestra las opciones para seleccionar el periodo temporal a utilizar. El usuario indica que quiere seleccionar el periodo temporal, elige el que quiere utilizar y confirma su selección. La aplicación actualiza el perfil del usuario con el periodo temporal seleccionado.

VER EL TIEMPO GASTADO EN REUNIONES

Un usuario con la sesión iniciada accede a la página de visualización de datos. La aplicación muestra la cantidad de tiempo que el usuario ha gastado en reuniones tanto en texto como en una gráfica.

VER EL PORCENTAJE DE TIEMPO GASTADO EN REUNIONES Y EL PORCENTAJE DE TIEMPO LIBRE

Un usuario con la sesión iniciada accede a la página de visualización de datos. La aplicación muestra el porcentaje de tiempo que el usuario ha gastado en reuniones y el porcentaje de tiempo libre, ambos representados tanto en texto como en una gráfica.

CONFIGURAR EL TIEMPO ENTRE REUNIONES QUE CUENTA COMO TIEMPO DE REUNIÓN

Un usuario con la sesión iniciada accede a la página de configuración de su perfil. La aplicación muestra las opciones para configurar el tiempo entre reuniones que cuenta como tiempo de reunión. El usuario indica que quiere configurar este tiempo, selecciona el intervalo que quiere que cuente como tiempo de reunión y confirma su selección. La aplicación actualiza el perfil del usuario con el nuevo intervalo de tiempo entre reuniones.

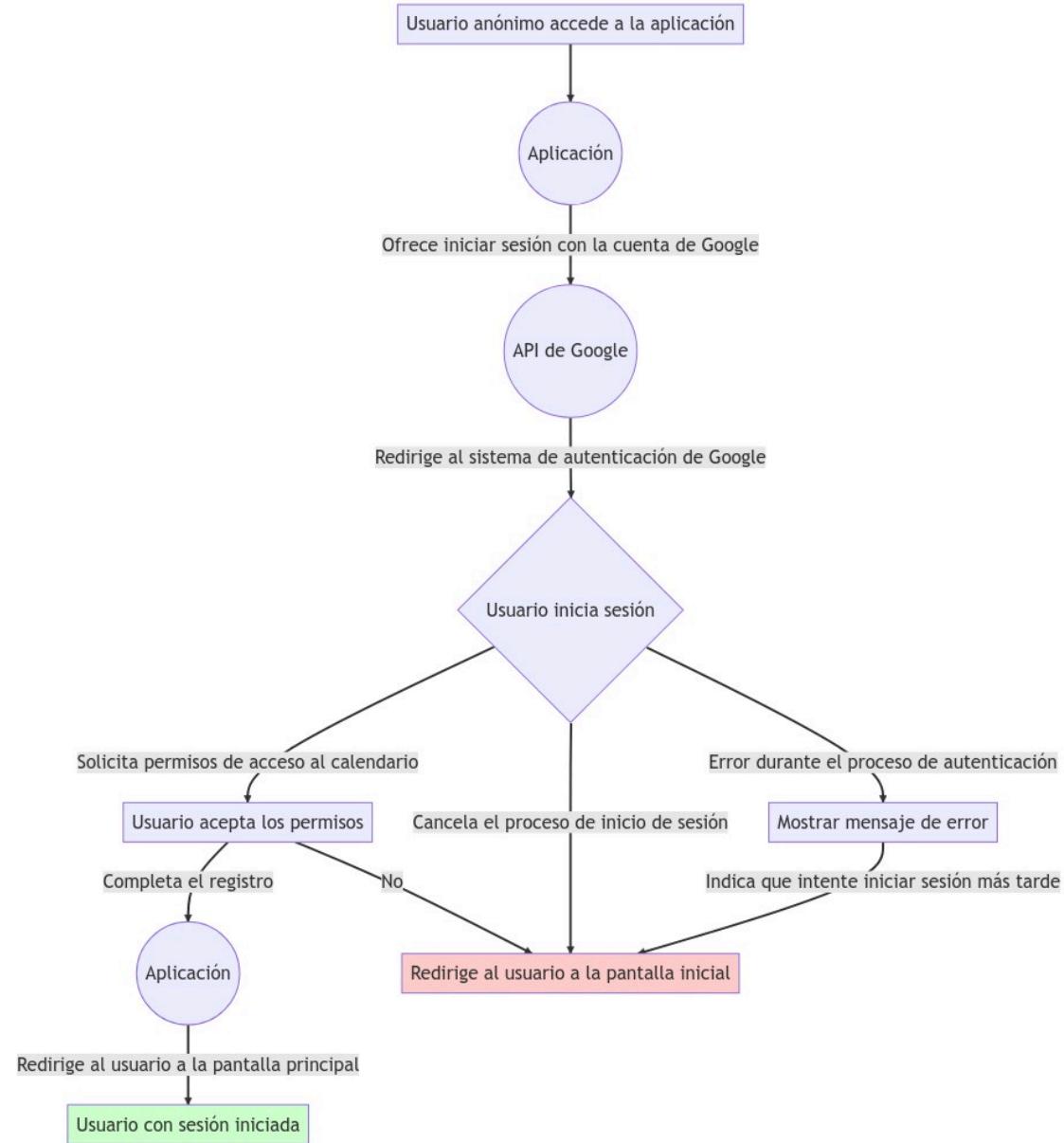
CASOS DE USO DETALLADOS

REGISTRARSE CON LA CUENTA DE GOOGLE

Caso de Uso	Registrarse con la cuenta de Google
Actor Principal	Usuario
Actores Secundarios	API de Google
Stakeholders y sus intereses	Usuario: Quiere registrarse en la aplicación para poder utilizar sus funcionalidades. API de Google: Proporciona la autenticación y autorización para el registro del usuario.
Precondiciones	El usuario debe tener una cuenta de Google válida. El usuario no debe estar registrado previamente en la aplicación.
Postcondiciones	El usuario está registrado en la aplicación y puede iniciar sesión con su cuenta de Google.
Escenario principal	<ol style="list-style-type: none"> 1. Un usuario anónimo accede a la aplicación. 2. La aplicación ofrece la opción de iniciar sesión con la cuenta de Google. 3. El usuario anónimo selecciona iniciar sesión con la cuenta de Google. 4. La aplicación redirige al usuario al sistema de autenticación de Google, que solicita permisos para acceder al calendario del usuario. 5. Tras completar el registro, el sistema redirige al usuario a la pantalla principal con la sesión iniciada.
Escenarios alternativos	<p>3a. El usuario cancela el proceso de inicio de sesión:</p> <ol style="list-style-type: none"> 1. El usuario decide cancelar el proceso de inicio de sesión. 2. El sistema redirige al usuario a la pantalla inicial sin iniciar sesión. <p>4a. El usuario rechaza los permisos para acceder al calendario:</p> <ol style="list-style-type: none"> 1. Google solicita al usuario permisos para acceder al calendario y el usuario rechaza. 2. El sistema informa al usuario que los permisos son necesarios para usar la aplicación. 3. El usuario... <ol style="list-style-type: none"> 1. Decide conceder los permisos. 2. Cancela el proceso de inicio de sesión. <p>5a. Error durante el proceso de autenticación:</p> <ol style="list-style-type: none"> 1. El sistema detecta un error durante el proceso de autenticación. 2. El sistema muestra un mensaje de error al usuario e indica que intente iniciar sesión más tarde.
Requisitos especiales	La aplicación debe cumplir con los términos de servicio de la API de Google. La aplicación debe manejar de manera segura y confidencial los tokens de acceso proporcionados por Google.

Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo del número de nuevos usuarios.
---------------------------------	---

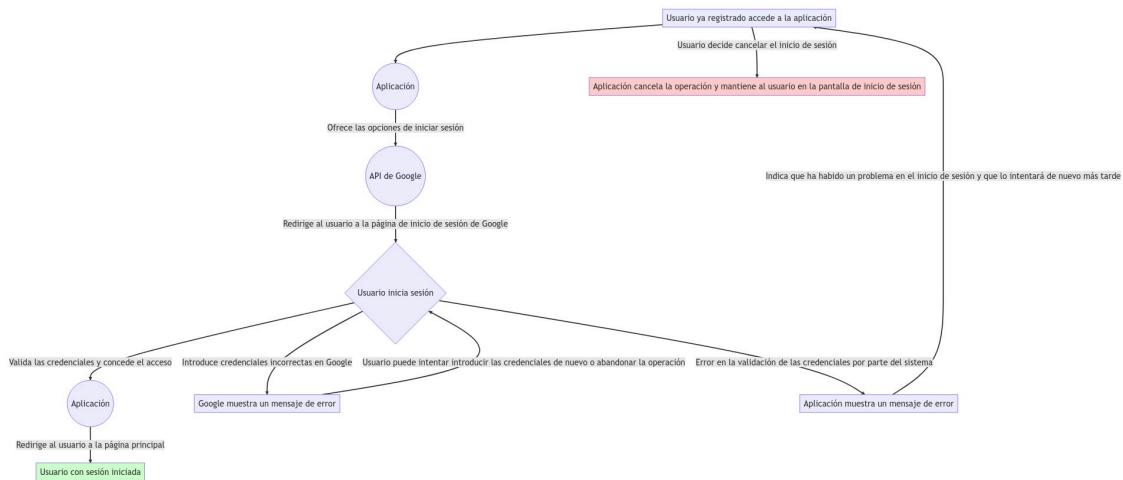
DIAGRAMA DE ACTIVIDAD



INICIAR SESIÓN CON LA CUENTA DE GOOGLE

Caso de Uso	Iniciar sesión con la cuenta de Google
Actor Principal	Usuario
Actores Secundarios	API de Google
Stakeholders y sus intereses	Usuario: Quiere iniciar sesión en la aplicación para acceder a sus funcionalidades. API de Google: Proporciona la autenticación y autorización para el inicio de sesión del usuario.
Precondiciones	El usuario debe tener una cuenta de Google válida. El usuario debe estar registrado previamente en la aplicación.
Postcondiciones	El usuario ha iniciado sesión en la aplicación y puede acceder a sus funcionalidades.
Escenario principal	<ol style="list-style-type: none"> 1. Un usuario ya registrado accede a la aplicación. 2. El sistema ofrece al usuario las opciones de iniciar sesión. 3. El usuario selecciona la opción de iniciar sesión con la cuenta de Google. 4. El sistema redirige al usuario a la página de inicio de sesión de Google. 5. El usuario inicia sesión con su cuenta de Google. 6. El sistema valida las credenciales y concede el acceso, redirigiendo al usuario a la página principal de la aplicación.
Escenarios alternativos	<p>5a. El usuario introduce credenciales incorrectas en Google:</p> <ol style="list-style-type: none"> 1. Google identifica que las credenciales son incorrectas y muestra un mensaje de error. 2. El usuario puede intentar introducir las credenciales de nuevo o abandonar la operación. <p>6a. Error en la validación de las credenciales por parte del sistema:</p> <ol style="list-style-type: none"> 1. El sistema identifica que ha habido un error durante la validación de las credenciales. 2. El sistema muestra un mensaje de error al usuario indicando que ha habido un problema en el inicio de sesión y que lo intentará de nuevo más tarde. <p>1-3a. El usuario decide cancelar el inicio de sesión:</p> <ol style="list-style-type: none"> 1. El usuario solicita al sistema que cancele la operación de inicio de sesión. 2. El sistema cancela la operación y mantiene al usuario en la pantalla de inicio de sesión.
Requisitos especiales	La aplicación debe cumplir con los términos de servicio de la API de Google. La aplicación debe manejar de manera segura y confidencial los tokens de acceso proporcionados por Google.
Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo del número de usuarios activos.

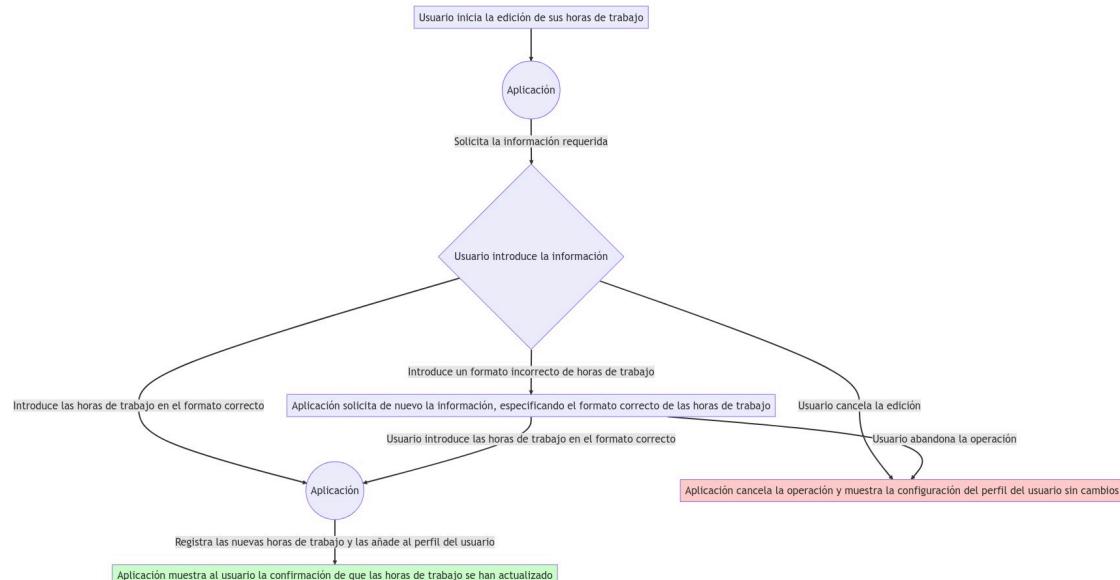
DIAGRAMA DE ACTIVIDAD



CONFIGURAR EL TIEMPO DE TRABAJO

Caso de Uso	Configurar el tiempo de trabajo
Actor Principal	Usuario
Stakeholders y sus intereses	Usuario: Quiere configurar sus horas de trabajo para que la aplicación pueda realizar cálculos precisos.
Precondiciones	El usuario debe estar registrado y haber iniciado sesión en la aplicación.
Postcondiciones	Las horas de trabajo del usuario se han actualizado en la aplicación.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario inicia la edición de sus horas de trabajo. 2. El sistema solicita la información requerida para esta edición: las horas de trabajo del usuario. 3. El usuario introduce la información requerida. 4. El sistema registra las nuevas horas de trabajo y las añade al perfil del usuario. 5. El sistema muestra al usuario la confirmación de que las horas de trabajo se han actualizado.
Escenarios alternativos	<p>1-3a. El usuario cancela la edición:</p> <ol style="list-style-type: none"> 1. El usuario solicita al sistema que cancele la edición de las horas de trabajo. 2. El sistema pide confirmación. 3. El usuario confirma. 4. El sistema muestra la configuración del perfil del usuario sin cambios. <p>4a. El usuario introduce un formato incorrecto de horas de trabajo:</p> <ol style="list-style-type: none"> 1. El sistema solicita de nuevo la información faltante, especificando el formato correcto de las horas de trabajo. 2. El usuario... <ol style="list-style-type: none"> 1. Introduce las horas de trabajo en el formato correcto. 2. Abandona la operación.
Requisitos especiales	La aplicación debe validar que las horas de trabajo introducidas por el usuario son válidas y están en el formato correcto.
Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo de cuántos usuarios deseen actualizar sus horas de trabajo.

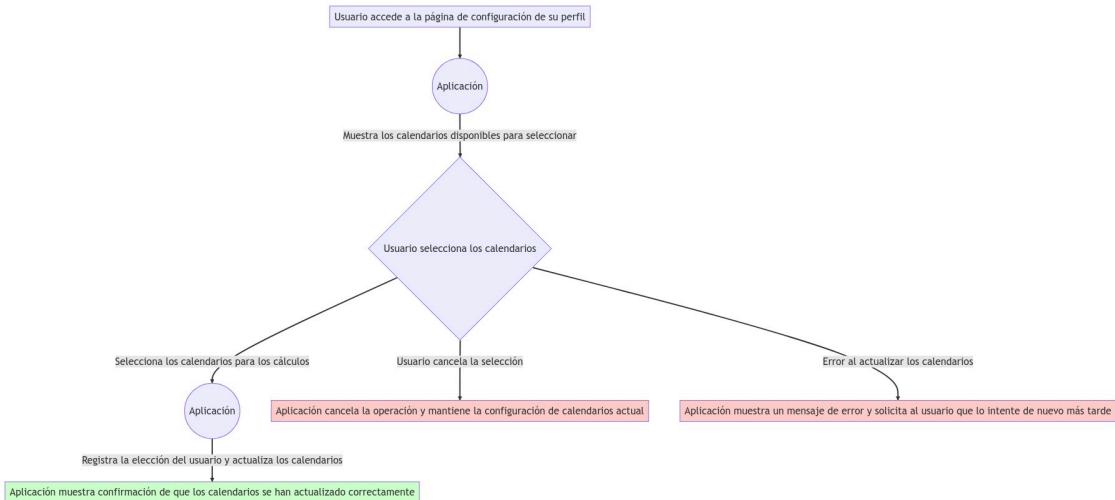
DIAGRAMA DE ACTIVIDAD



SELECCIONAR CALENDARIOS PARA LOS CÁLCULOS

Caso de Uso	Seleccionar calendarios para los cálculos
Actor Principal	Usuario
Stakeholders y sus intereses	Usuario: Quiere seleccionar qué calendarios se deben utilizar para los cálculos de la aplicación.
Precondiciones	El usuario debe estar registrado y haber iniciado sesión en la aplicación.
Postcondiciones	Los calendarios seleccionados por el usuario se utilizan para los cálculos de la aplicación.
Escenario principal	<ol style="list-style-type: none"> 1. Un usuario con sesión iniciada accede a la página de configuración de su perfil. 2. El sistema muestra al usuario los calendarios disponibles para seleccionar. 3. El usuario selecciona qué calendarios quiere que se utilicen para los cálculos. 4. El sistema registra la elección del usuario y actualiza los calendarios que se usarán en los cálculos. 5. El sistema muestra una confirmación al usuario de que los calendarios para los cálculos se han actualizado correctamente.
Escenarios alternativos	<p>3a. El usuario decide cancelar la selección:</p> <ol style="list-style-type: none"> 1. El usuario solicita al sistema que cancele la operación de selección de calendarios. 2. El sistema cancela la operación y mantiene la configuración de calendarios actual del usuario. <p>4a. Error al actualizar los calendarios para los cálculos:</p> <ol style="list-style-type: none"> 1. El sistema identifica que ha habido un error al actualizar los calendarios para los cálculos. 2. El sistema muestra un mensaje de error al usuario indicando que ha habido un problema al actualizar los calendarios y solicita al usuario que lo intentará de nuevo más tarde.
Requisitos especiales	La aplicación debe poder acceder a los calendarios seleccionados por el usuario.
Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo de cuántos usuarios deseen actualizar los calendarios que se utilizan para los cálculos.

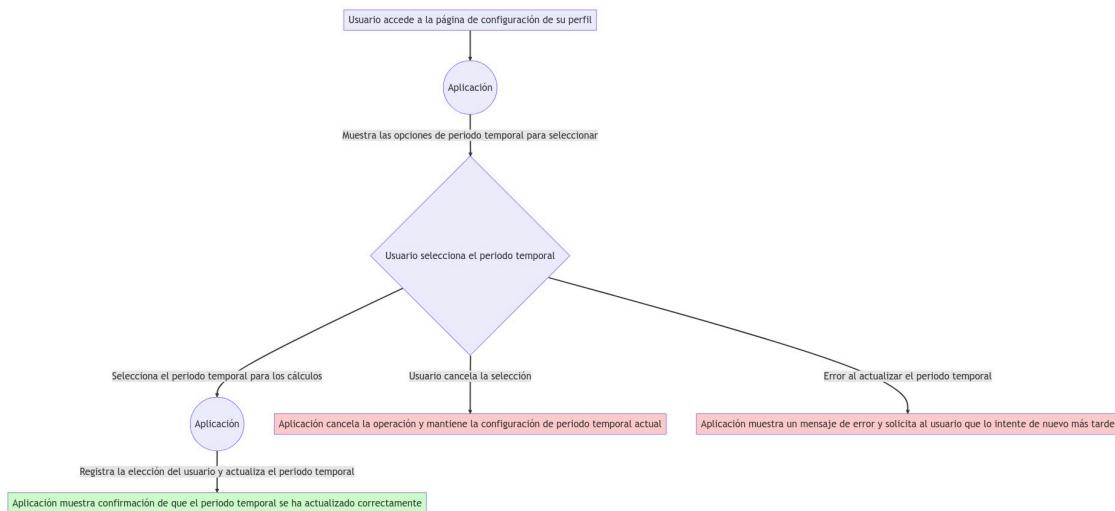
DIAGRAMA DE ACTIVIDAD



SELECCIONAR PERIODO TEMPORAL PARA LOS CÁLCULOS

Caso de Uso	Seleccionar periodo temporal para los cálculos
Actor Principal	Usuario
Stakeholders y sus intereses	Usuario: Quiere seleccionar el periodo temporal que la aplicación utilizará para los cálculos.
Precondiciones	El usuario debe estar registrado y haber iniciado sesión en la aplicación.
Postcondiciones	El periodo temporal seleccionado por el usuario se utiliza para los cálculos de la aplicación.
Escenario principal	<ol style="list-style-type: none"> 1. Un usuario con sesión iniciada accede a la página de configuración de su perfil. 2. El sistema muestra al usuario las opciones de periodo temporal para seleccionar. 3. El usuario selecciona qué periodo temporal quiere que se utilice para los cálculos. 4. El sistema registra la elección del usuario y actualiza el periodo temporal que se usará en los cálculos. 5. El sistema muestra una confirmación al usuario de que el periodo temporal para los cálculos se ha actualizado correctamente.
Escenarios alternativos	<p>3a. El usuario decide cancelar la selección:</p> <ol style="list-style-type: none"> 1. El usuario solicita al sistema que cancele la operación de selección del periodo temporal. 2. El sistema cancela la operación y mantiene la configuración de periodo temporal actual del usuario. <p>4a. Error al actualizar el periodo temporal para los cálculos:</p> <ol style="list-style-type: none"> 1. El sistema identifica que ha habido un error al actualizar el periodo temporal para los cálculos. 2. El sistema muestra un mensaje de error al usuario indicando que ha habido un problema al actualizar el periodo temporal y que lo intentará de nuevo más tarde.
Requisitos especiales	La aplicación debe validar que el periodo temporal seleccionado por el usuario es válido.
Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo de cuántos usuarios deseen actualizar el periodo temporal para los cálculos.

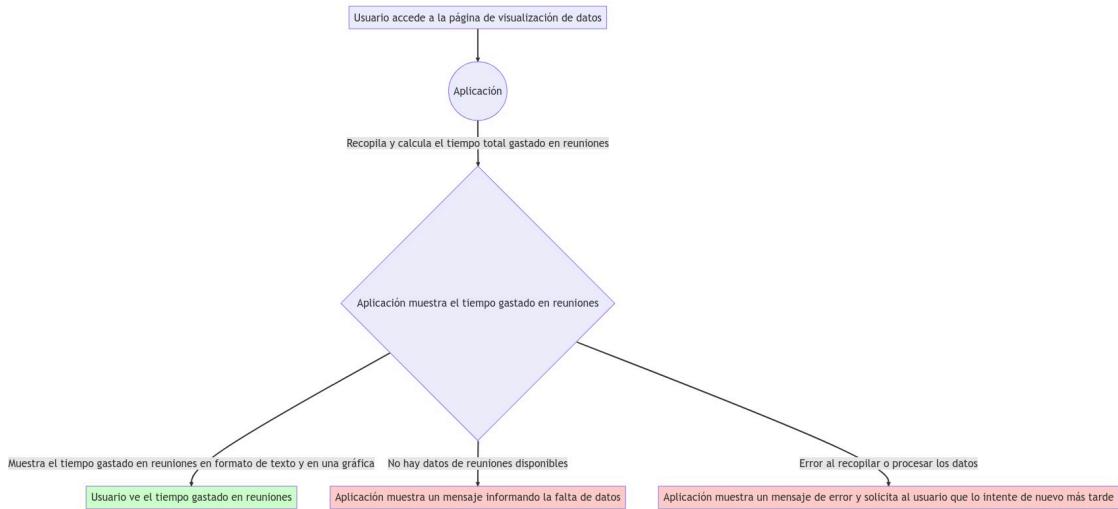
DIAGRAMA DE ACTIVIDAD



VER EL TIEMPO GASTADO EN REUNIONES

Caso de uso	Ver el tiempo gastado en reuniones
Actor Principal	Usuario
Stakeholders y sus intereses	Usuario: Quiere ver cuánto tiempo ha gastado en reuniones.
Precondiciones	El usuario debe estar registrado y haber iniciado sesión en la aplicación.
Postcondiciones	El usuario ha visto cuánto tiempo ha gastado en reuniones.
Escenario principal	<p>1. El usuario con la sesión iniciada accede a la página de visualización de datos.</p> <p>2. El sistema recopila y calcula el tiempo total gastado en reuniones.</p> <p>3. El sistema muestra al usuario la cantidad de tiempo que ha gastado en reuniones en formato de texto y en una gráfica.</p>
Escenarios alternativos	<p>2a. No hay datos de reuniones disponibles:</p> <ol style="list-style-type: none"> 1. El sistema identifica que no hay datos de reuniones disponibles para el usuario. 2. El sistema muestra un mensaje informando al usuario de la falta de datos y sugiere posibles soluciones (por ejemplo, esperar a tener reuniones, etc.). <p>2b. Error al recopilar o procesar los datos:</p> <ol style="list-style-type: none"> 1. El sistema identifica que ha habido un error al recopilar o procesar los datos de las reuniones. 2. El sistema muestra un mensaje de error al usuario indicando que ha habido un problema al procesar los datos y que lo intentará de nuevo más tarde.
Requisitos especiales	La aplicación debe ser capaz de calcular el tiempo gastado en reuniones basándose en los datos del calendario del usuario.
Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo de cuántos usuarios deseen ver el tiempo gastado en reuniones.

DIAGRAMA DE ACTIVIDAD

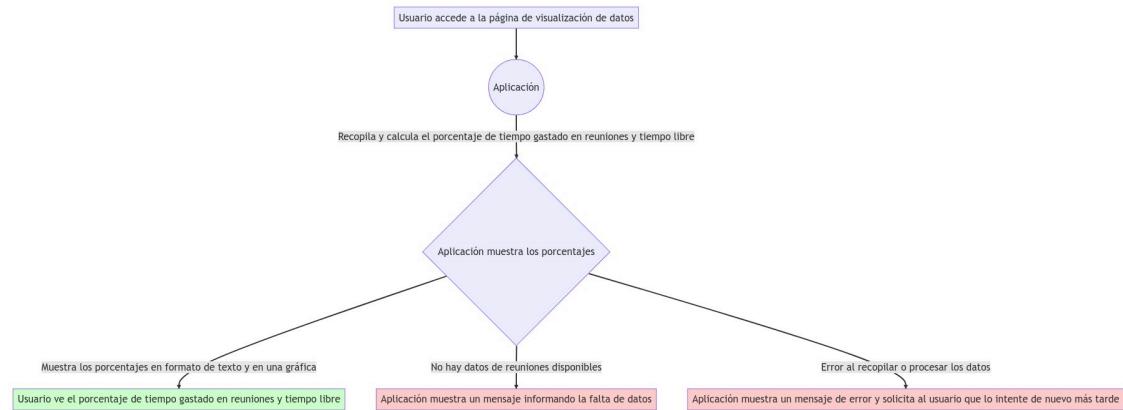


VER EL PORCENTAJE DE TIEMPO GASTADO EN REUNIONES Y TIEMPO LIBRE

Caso de Uso	Ver el porcentaje de tiempo gastado en reuniones y tiempo libre
Actor Principal	Usuario
Stakeholders y sus intereses	Usuario: Quiere ver el porcentaje de tiempo que ha gastado en reuniones y el tiempo libre.
Precondiciones	El usuario debe estar registrado y haber iniciado sesión en la aplicación.
Postcondiciones	El usuario ha visto el porcentaje de tiempo que ha gastado en reuniones y el tiempo libre.
Escenario principal	<ol style="list-style-type: none"> Un usuario con sesión iniciada accede a la página de visualización de datos. El sistema recopila y calcula el porcentaje de tiempo que el usuario ha gastado en reuniones y el tiempo libre. El sistema muestra al usuario estos porcentajes tanto en formato de texto como reflejados en una gráfica.
Escenarios alternativos	<p>2a. No hay datos de reuniones disponibles:</p> <ol style="list-style-type: none"> El sistema identifica que no hay datos de reuniones disponibles para el usuario. El sistema muestra un mensaje informando al usuario de la falta de datos y sugiere posibles soluciones (por ejemplo, esperar a tener reuniones, etc.). <p>2b. Error al recopilar o procesar los datos:</p> <ol style="list-style-type: none"> El sistema identifica que ha habido un error al recopilar o procesar los datos. El sistema muestra un mensaje de error al usuario indicando que ha habido un problema al recopilar los datos y que lo intentará de nuevo más tarde.
Requisitos especiales	La aplicación debe ser capaz de calcular el porcentaje de tiempo gastado en reuniones y el tiempo libre basándose en los datos del calendario del usuario.

Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo de cuántos usuarios deseen ver el porcentaje de tiempo gastado en reuniones y el tiempo libre.
---------------------------------	---

DIAGRAMA DE ACTIVIDAD:

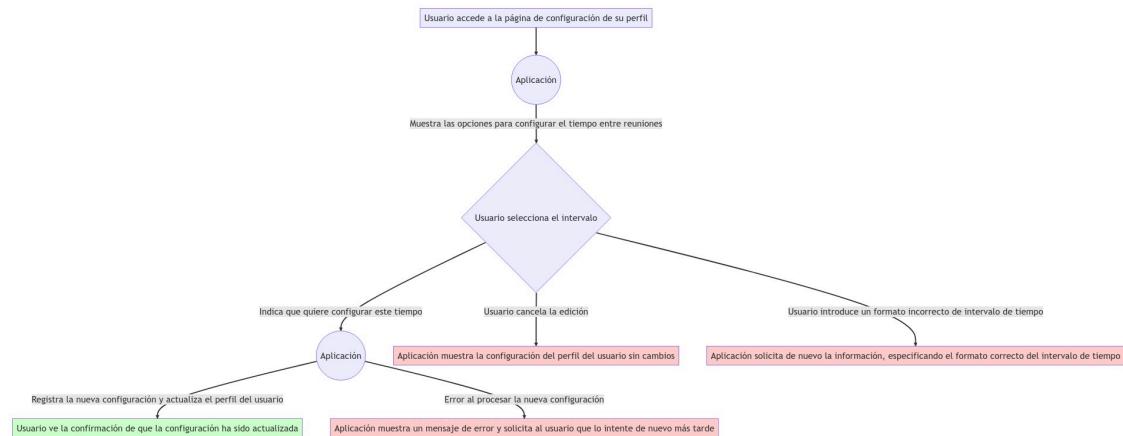


CONFIGURAR EL TIEMPO ENTRE REUNIONES QUE CUENTA COMO TIEMPO DE REUNIÓN

Caso de Uso	Configurar el tiempo entre reuniones que cuenta como tiempo de reunión
Actor Principal	Usuario
Stakeholders y sus intereses	Usuario: Quiere configurar el tiempo entre reuniones que cuenta como tiempo de reunión.
Precondiciones	El usuario debe estar registrado y haber iniciado sesión en la aplicación.
Postcondiciones	El tiempo entre reuniones que cuenta como tiempo de reunión ha sido configurado por el usuario.
Escenario principal	<ol style="list-style-type: none"> Un usuario con sesión iniciada accede a la página de configuración de su perfil. El sistema muestra las opciones para configurar el tiempo entre reuniones que cuenta como tiempo de reunión. El usuario indica que quiere configurar este tiempo y selecciona el intervalo que quiere que cuente como tiempo de reunión. El sistema registra la nueva configuración y actualiza el perfil del usuario. El sistema muestra al usuario la confirmación de que la configuración ha sido actualizada.
Escenarios alternativos	<p>1-3a. El usuario cancela la edición:</p> <ol style="list-style-type: none"> El usuario solicita al sistema que cancele la edición del tiempo entre reuniones que cuenta como tiempo de reunión. El sistema muestra la configuración del perfil del usuario sin cambios. <p>4a. El usuario introduce un formato incorrecto de intervalo de tiempo:</p>

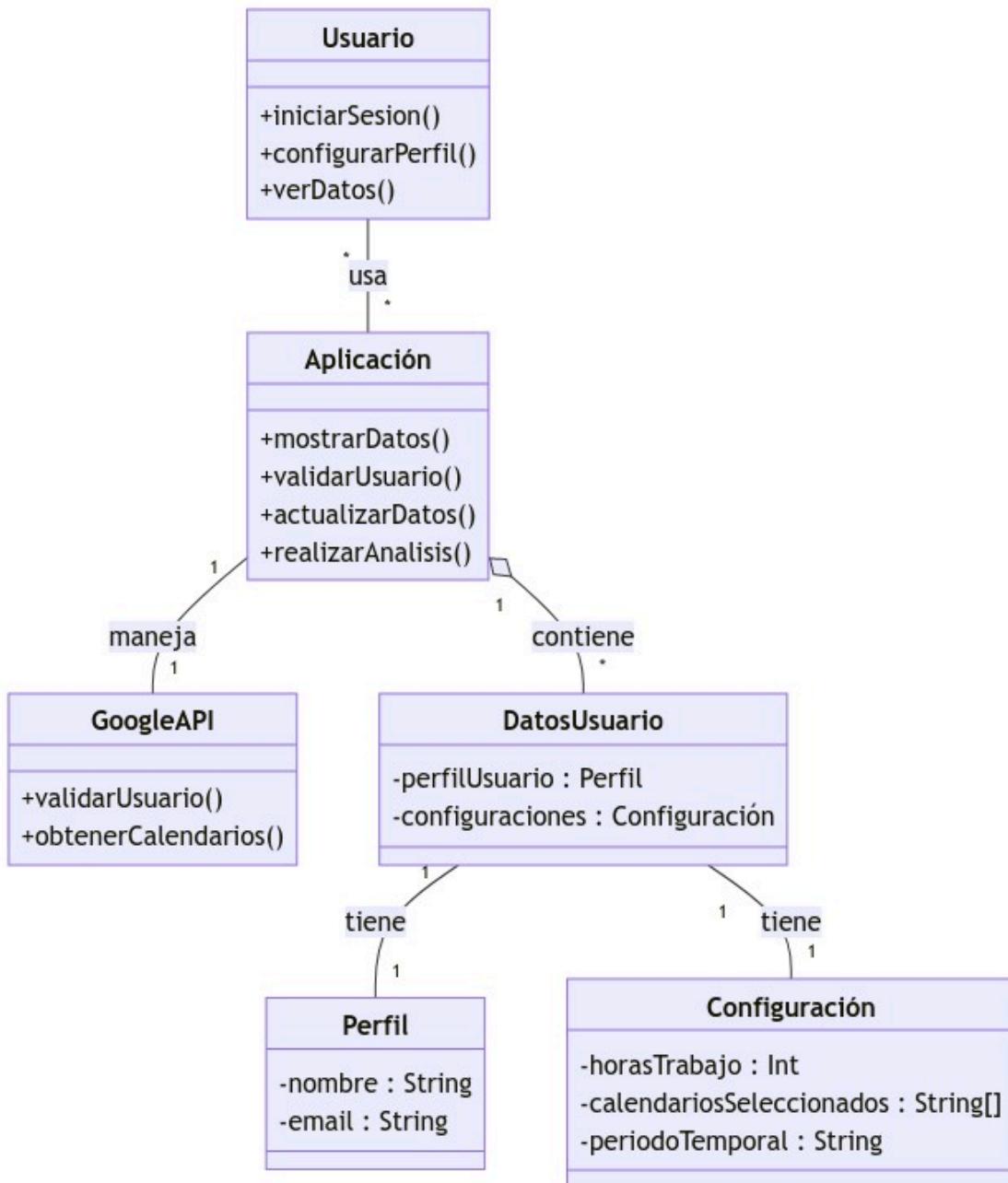
	<p>1. El sistema solicita de nuevo la información, especificando el formato correcto del intervalo de tiempo.</p> <p>2. El usuario...</p> <ol style="list-style-type: none"> 1. Introduce el intervalo de tiempo en el formato correcto. 2. Abandona la operación. <p>4b. Error al procesar la nueva configuración:</p> <ol style="list-style-type: none"> 1. El sistema identifica que ha habido un error al procesar la nueva configuración. 2. El sistema muestra un mensaje de error al usuario indicando que ha habido un problema y que lo intentará de nuevo más tarde.
Requisitos especiales	La aplicación debe validar que el tiempo entre reuniones introducido por el usuario es válido y está en el formato correcto.
Frecuencia de ocurrencia	Puede ocurrir varias veces al día, dependiendo de cuántos usuarios deseen configurar el tiempo entre reuniones que cuenta como tiempo de reunión.

DIAGRAMA DE ACTIVIDAD



ANÁLISIS DEL MODELO

DIAGRAMA DE CLASES



DESCRIPCIÓN DE LAS CLASES

Nombre	Descripción	Responsabilidades	Atributos	Operaciones
Usuario	Representa al usuario de la aplicación.	Interactuar con la aplicación para iniciar sesión, configurar su perfil y ver sus datos.	N/A	iniciarSesion(), configurarPerfil(), verDatos()

Tabla <TBD>: Descripción de la clase Usuario

Nombre	Descripción	Responsabilidades	Atributos	Operaciones
GoogleAPI	Representa la API de Google utilizada para la autenticación y la obtención de calendarios.	Validar al usuario y proporcionar los calendarios del usuario.	N/A	validarUsuario(), obtenerCalendarios()

Tabla <TBD>: Descripción de la clase GoogleAPI

Nombre	Descripción	Responsabilidades	Atributos	Operaciones
Aplicación	Representa la aplicación que el usuario utiliza.	Manejar la interacción del usuario con la aplicación y la API de Google, mostrar los datos del usuario, validar al usuario, actualizar los datos del usuario y realizar análisis.	N/A	mostrarDatos(), validarUsuario(), actualizarDatos(), realizarAnalisis()

Tabla <TBD>: Descripción de la clase Aplicación

Nombre	Descripción	Responsabilidades	Atributos	Operaciones
DatosUsuario	Representa los datos del usuario en la aplicación.	Contener el perfil y las configuraciones del usuario.	perfilUsuario (Perfil), configuraciones (Configuración)	N/A

Tabla <TBD>: Descripción de la clase DatosUsuario

Nombre	Descripción	Responsabilidades	Atributos	Operaciones
Perfil	Representa el perfil del usuario.	Contener el nombre y el correo electrónico del usuario.	nombre (String), email (String)	N/A

Tabla <TBD>: Descripción de la clase Perfil

Nombre	Descripción	Responsabilidades	Atributos	Operaciones
Configuración	Representa las configuraciones del usuario.	Contener las horas de trabajo, los calendarios seleccionados y el periodo temporal del usuario.	horasTrabajo (Int), calendariosSeleccionados (String[]), periodoTemporal (String)	N/A

Tabla <TBD>: Descripción de la clase Configuración

MOCKUPS DE LA INTERFAZ

PÁGINA PRINCIPAL

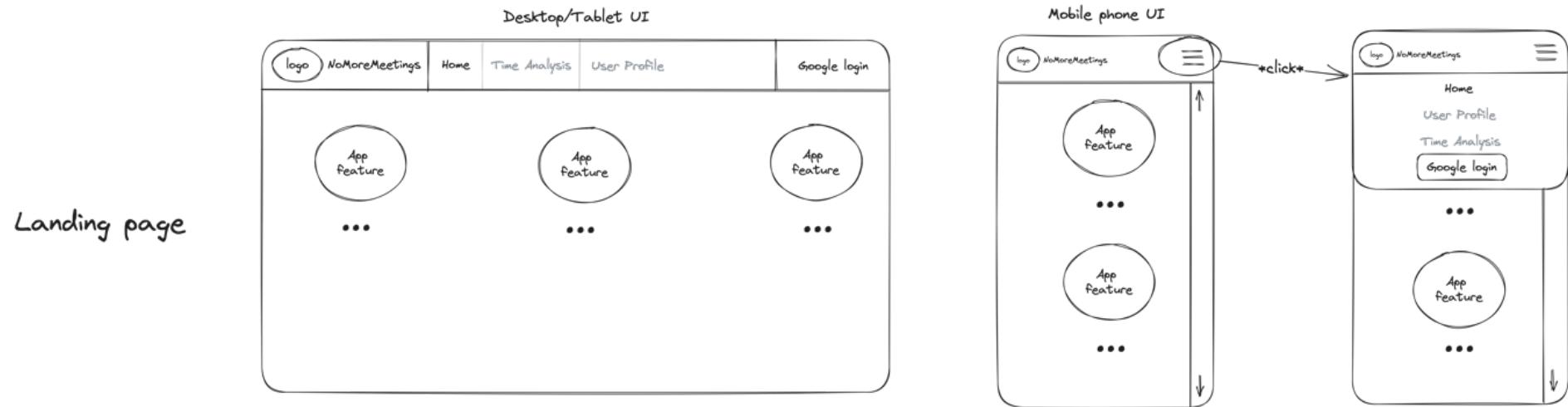


Figura <TBD>: Mockup de la página principal

PERFIL DEL USUARIO

User profile page

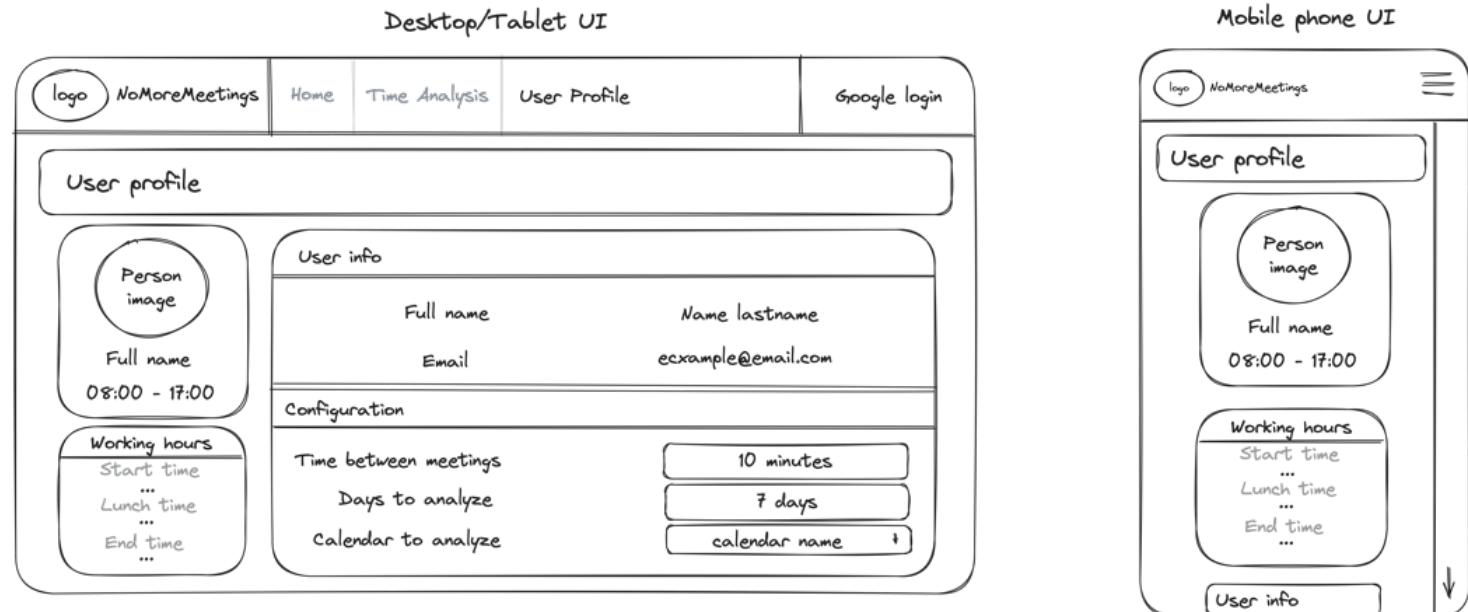
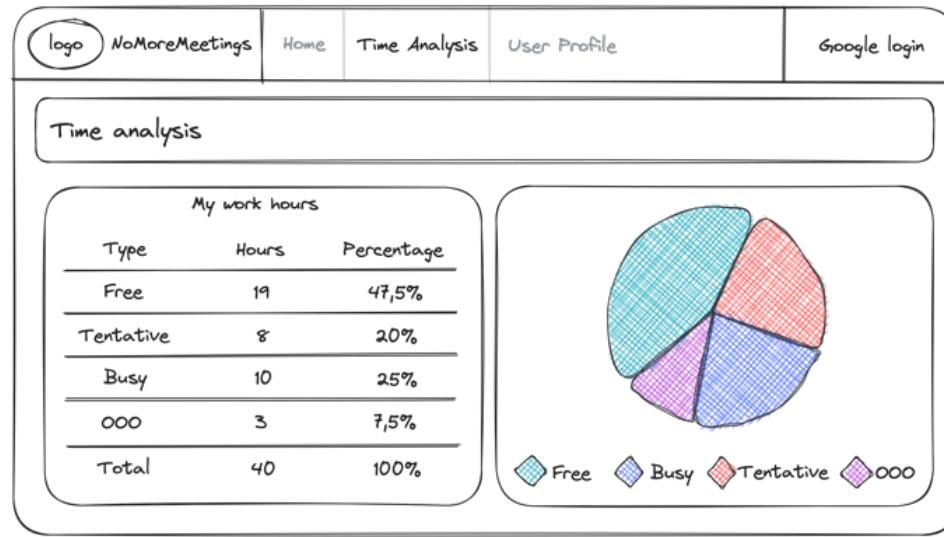


Figura <TBD>: Mockup de la página de configuración del usuario

ANÁLISIS DE DATOS

Time analysis page

Desktop/Tablet UI



Mobile phone UI

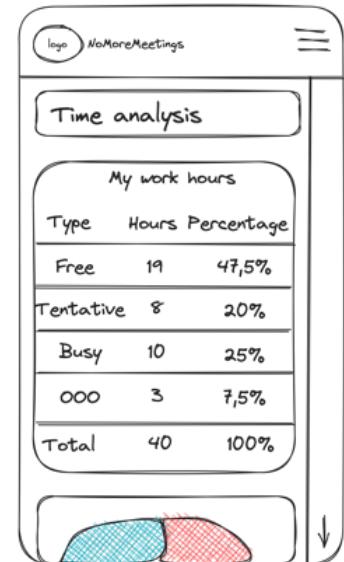


Figura <TBD>: Mockup de la página de visualización de datos

DISEÑO

ARQUITECTURA - DIAGRAMA DE PAQUETES

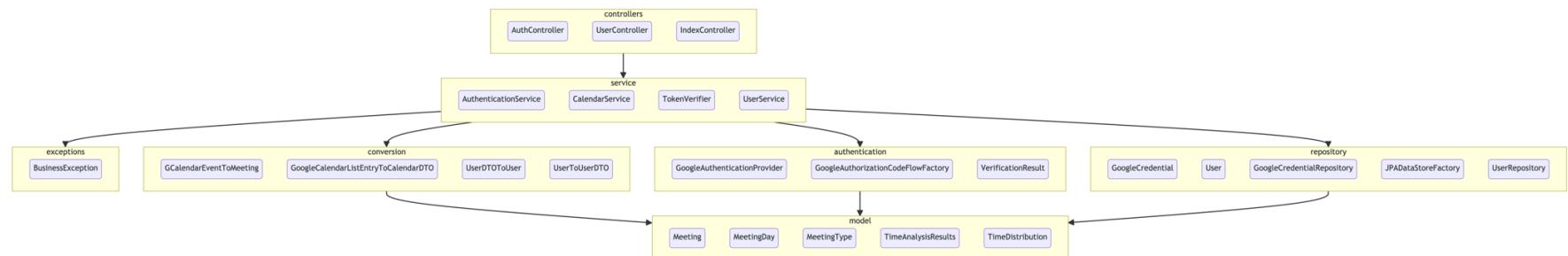


Figura <TBD>: Diagrama de paquetes del backend

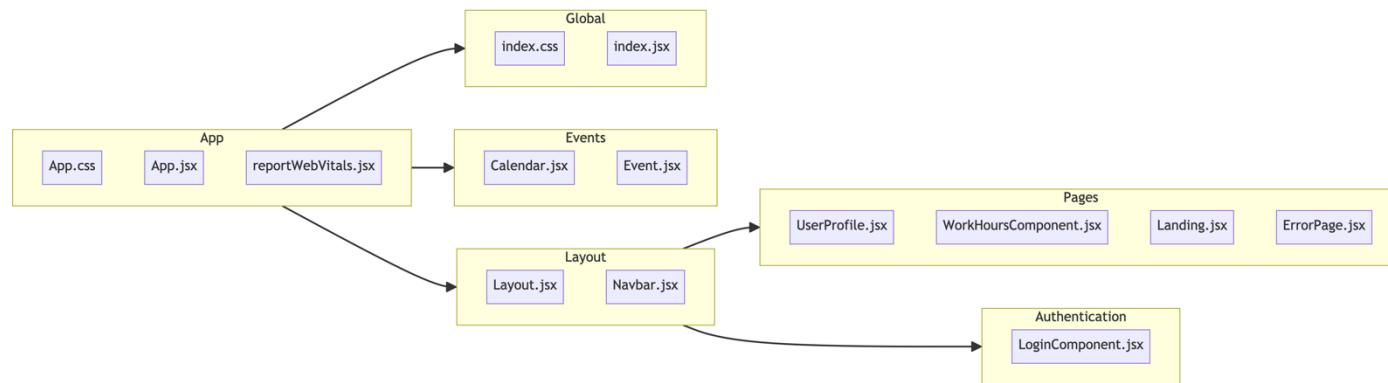


Figura <TBD>: Diagrama de paquetes del frontend

DIAGRAMA DE DESPLIEGUE

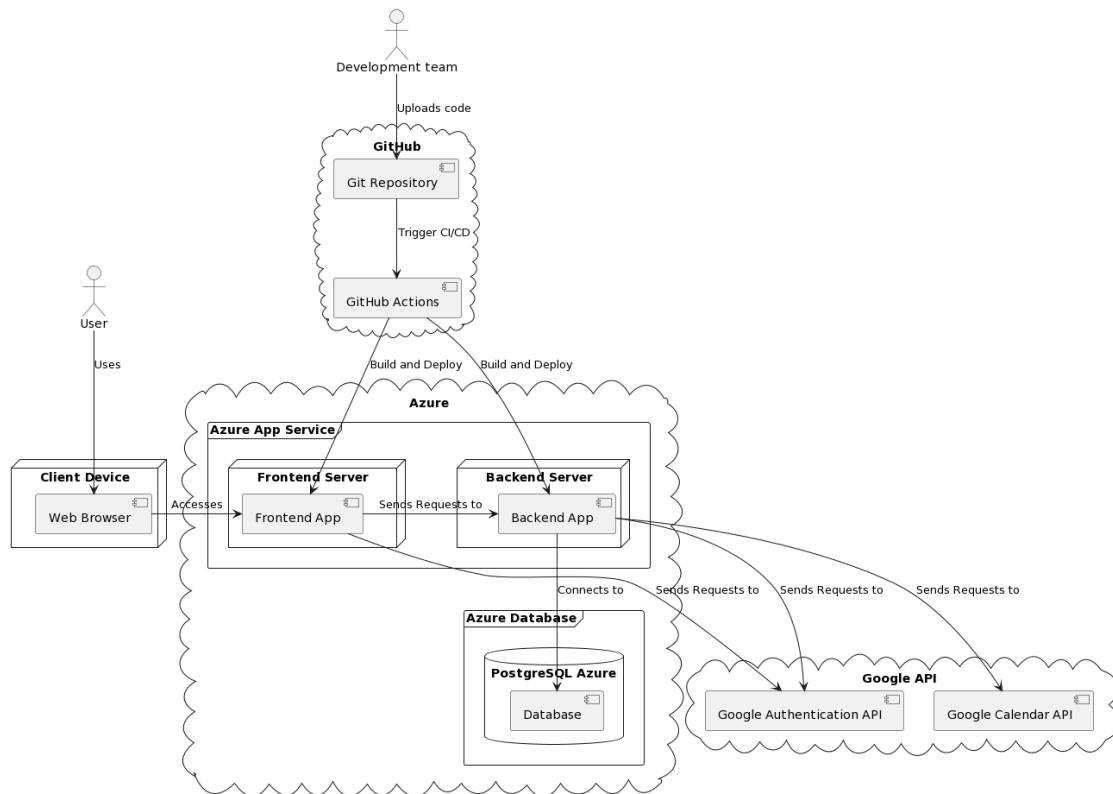


Figura <TBD>: Diagrama de despliegue de la aplicación

PATRONES DE DISEÑO UTILIZADOS

Los patrones de diseño son, de acuerdo con la definición de Erich Gamma en su libro Patrones de Diseño: *"Descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en un determinado contexto"*. En el desarrollo del proyecto se han utilizado diversos patrones, en concreto han sido utilizados: Repository, Factory Method, Singleton, Data Transfer Object (DTO), Builder, Adapter, y Strategy.

Además, se ha utilizado la Inyección de Dependencias, un principio fundamental en el diseño de aplicaciones modernas, y que supone la base de los frameworks de desarrollo moderno como Spring o Micronaut.

A continuación, se proporciona una descripción detallada de cada uno de estos patrones, incluyendo cómo y por qué se han aplicado en el proyecto.

INYECCIÓN DE DEPENDENCIAS

La inyección de dependencias es un principio fundamental en el desarrollo de software moderno y es esencial para los programas basados en el framework de Micronaut. La idea principal de la inyección de dependencias es promover la encapsulación del código eliminando referencias directas a las clases que utilizan, permitiendo por ejemplo definir campos que serán inyectados en base a interfaces, pudiendo decidir que implementación concreta se usará en base a diversas configuraciones. En lugar de que las clases creen las instancias de las clases de las que dependen, esas instancias se "inyectan" en las mediante un "contenedor" o "motor" de inyección de dependencias.

Por ejemplo, las instancias de los servicios (como `UserServicelmp`) y los repositorios (como `UserRepository`) son inyectadas automáticamente por Micronaut. Este enfoque reduce el acoplamiento entre las clases, hace que el código sea más modular y flexible, y facilita las pruebas unitarias al permitir que las dependencias reales se sustituyan fácilmente por objetos simulados (mocks) en las pruebas.

PATRÓN REPOSITORIO

El patrón Repositorio proporciona una abstracción de la persistencia de datos, permitiendo que la aplicación acceda a los datos sin tener que conocer los detalles de cómo se almacenan. En el proyecto se utiliza tanto en `GoogleCredentialRepository` como en `UserRespository`. El primero de ellos es ligeramente más complejo por definir métodos nuevos y ya que una vez explicado uno la implementación del otro se entiende por analogía.

`GoogleCredentialRepository`, al igual que `UserRepository`, es solo una interfaz que extiende de `CrudRepository` sin una clase de implementación declarada ya que Micronaut Data genera automáticamente la implementación de la interfaz en tiempo de compilación. Esta implementación generada se encarga de las operaciones de base de datos, como consultas, inserciones, actualizaciones y eliminaciones.

Micronaut Data utiliza el análisis de nombres de métodos para inferir las operaciones de la base de datos. Por ejemplo, el método `findByPrimaryKey` se traduciría a una consulta SQL que busca una entrada en la base de datos donde la clave coincide con el parámetro proporcionado.

PATRÓN FACTORY METHOD

El patrón Factory Method se utiliza para crear objetos sin exponer la lógica de creación al cliente. En este proyecto, `JPADataStoreFactory` es un ejemplo de este patrón. Esta clase proporciona un método para obtener una instancia de `DataStore`, ocultando los detalles de cómo se crea la instancia. Esto permite la creación de diferentes tipos de `DataStore` si se añaden más implementaciones en el futuro sin cambiar el código que utiliza la factoría.

Se trata de una factory method ya que en el patrón factory method, una clase tiene un método que se utiliza para crear un objeto, y ese método se suele sobrescribir en las subclases para crear diferentes tipos de objetos. En este caso, el método de fábrica es `getDataStore()`, y se puede implementar de diferentes maneras en diferentes clases que implementan la interfaz `DataStoreFactory`.

Hay más ejemplos similares como el `AuthorizationCodeFlowFactory` que devuelve un objeto de tipo `AuthorizationCodeFlow` configurado para el proveedor necesario, en este caso solo existe el método `getGoogleAuthorizationCodeFlow()`.

PATRÓN SINGLETON

El patrón Singleton garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella, compartiendo por tanto estado. En este proyecto, al usar el framework Micronaut basado en inyección de dependencias basado en beans, la mayoría de dependencias inyectadas son en sí mismas un singleton. Tomemos como ejemplo cualquier implementación de un servicio, `UserServicelmp` por ejemplo, en este caso la clase está anotada con `@Singleton`, por lo que cuando esta clase se inyecte mediante el motor de inyección de dependencias de Micronaut, este se encargará de crear una instancia si no existiese ya, y en caso contrario devolver la instancia existente.

Dado que en el proyecto no se instancian clases directamente cuando estas están anotadas con `@Singleton`, `@Repository` o `@Factory` ya que esto sería una mala práctica en el contexto de una aplicación basada en la inyección de dependencias, siempre se usará la misma instancia de todas estas clases.

PATRÓN DE OBJETOS DE TRANSFERENCIA DE DATOS (DTO)

Este patrón se utiliza ampliamente en el proyecto, y su finalidad es abstraer las diferentes capas de la aplicación entre sí, así como ocultar detalles de implementación. De esta manera podremos cambiar la implementación de los servicios que los controladores seguirán manejando DTOs y el usuario de la API no será consciente de estos cambios. Lo mismo sucede si se cambian las entidades, los servicios entienden de objetos de modelo por lo que la implementación específica de las entidades afectara a los repositorios, pero no a los servicios. Nos permite por tanto asegurar la encapsulación y reducir el acoplamiento.

Todos los resultados que la aplicación comparte mediante sus controladores son DTOs generados a partir de las clases del modelo mediante converters. Pero como he explicado también se usa para comunicaciones entre diferentes capas de la aplicación, un ejemplo podría ser la clase VerificationResult.

La clase VerificationResult se utiliza para transferir los resultados de la verificación de autenticación. La clase encapsula un valor booleano para indicar si la verificación fue exitosa y una cadena que proporciona una razón para cualquier fracaso. Esto simplifica la transmisión de datos y permite una mayor flexibilidad si los detalles de la implementación de la verificación cambian en el futuro.

PATRÓN BUILDER

Se usa ampliamente en la construcción de DTOs en el proyecto. Su uso permite mejorar la legibilidad del código, simplificar la construcción de objetos, promover que los objetos sean inmutables y ayuda a cumplir con el principio de responsabilidad única.

La implementación concreta en el proyecto es la proporcionada por Lombok mediante la anotación @Builder con el parámetro de toBuilder puesto a true. Esta implementación nos permite crear objetos a partir de un builder invocado mediante un método estático de la clase anotada. El toBuilder nos permite que de un objeto existente podamos obtener un builder con los fields que tenía el objeto original, esto facilita mucho la modificación de objetos y lo hace de una forma legible, se usa por ejemplo en la actualización de los datos del usuario en UserServiceImp.

PATRÓN ADAPTER

Este patrón se utiliza para convertir la interfaz de una clase en otra interfaz que los clientes esperan. Es decir, ayuda a que las clases trabajen juntas que no podrían hacerlo de otro modo debido a las interfaces incompatibles. Se usa ampliamente para convertir objetos entre diferentes capas de la aplicación, permitiendo mantener un bajo acoplamiento. Un ejemplo podría ser la interfaz Event de la librería de Google Calendar que se convierte a una interfaz Meeting propia de NoMoreMeetings, permitiendo que en el futuro se puedan conectar diferentes proveedores, ya que solo será necesario un nuevo converter (implementación del patrón Adapter). También se ayuda a cumplir con el principio de responsabilidad única.

La clase GCalendarEventToMeeting implementa TypeConverter<Event, Meeting>, que convierte el objeto Event en un objeto Meeting. Esto se realiza en el método convert, que toma un objeto Event, extrae la información necesaria y la utiliza para construir un objeto Meeting mediante el patrón Builder.

PATRÓN STRATEGY

El patrón strategy es un patrón de diseño que permite seleccionar un algoritmo o estrategia en tiempo de ejecución. En lugar de implementar un único algoritmo directamente en la lógica de una clase, el código se refiere a una interfaz abstracta, lo que permite cambiar los algoritmos o estrategias libremente.

Se puede observar su uso en el proyecto en la clase GoogleAuthenticationProvider, que implementa la interfaz AuthenticationProvider. Respecto del patron strategy:

- AuthenticationProvider es la estrategia base o interfaz común que define cómo se deben implementar las estrategias concretas. Define un método authenticate() que todas las estrategias concretas deben implementar.
- GoogleAuthenticationProvider es una estrategia concreta que implementa una versión específica del método authenticate(). En este caso, implementa la autenticación a través de Google.

Se podría tener muchas otras clases, como MicrosoftAuthenticationProvider o YahooAuthenticationProvider que implementen AuthenticationProvider con su propia lógica de autenticación.

El código que llama a authenticate() no necesita saber qué tipo de proveedor de autenticación se está utilizando ya que simplemente llama al método en la interfaz AuthenticationProvider, y el patrón strategy se encarga del resto. Este desacoplamiento hace que el código sea más flexible y fácil de extender con nuevas estrategias de autenticación.

PATRÓN DE COMPOSICIÓN (REACT)

La composición en React se refiere a cómo los componentes de React se pueden combinar y anidar para crear componentes más complejos. En lugar de heredar la funcionalidad de un componente "padre", en React, los componentes "hijos" se pueden pasar a los componentes "padre" como props y pueden ser renderizados por este mismo componente padre.

Este patrón de composición se utiliza en todo React y se usa ampliamente en el frontend del proyecto. Por ejemplo, en el componente App, se utiliza el componente Routes para renderizar diferentes componentes dependiendo de la ruta de la URL. Y este mismo componente contiene a su vez referencias a diferentes componentes como NavBar o UserProfile, permitiendo la navegación y haciendo uso de esta composición característica de React y que, aunque no es considerado un patrón de diseño en los libros escritos por Erich Gamma, Richard Helm, Ralph Johnson, o John Vlissides (también conocidos como el "Gang of Four") sí que puede considerarse comotral, siempre considerando que es característico de las aplicaciones escritas en React.

Además, los hooks de React también facilitan la composición, permitiendo que la lógica del estado y los efectos secundarios se encapsulen en funciones reutilizables, que pueden ser utilizadas en múltiples componentes.

REALIZACIÓN DE CASOS DE USO EN DISEÑO

En esta sección se describe cómo los componentes de software interactúan entre sí y con las interfaces externas para llevar a cabo los casos de uso identificados en el análisis. A continuación, se listan los casos de uso identificados en el análisis y se explica su implementación final.

REGISTRARSE CON LA CUENTA DE GOOGLE

Para el caso de uso "Registrarse usando la cuenta de Google", el diseño se implementa principalmente a través de llamadas entre el frontend y Google por un lado y entre el backend y Google así como entre el backend y el frontend.

FLUJO EN EL FRONTEND

En primer lugar, el usuario accede a la interfaz del frontend y hace clic en el botón para iniciar sesión con Google. Este evento es manejado por la librería de OAuth para React de Google dentro del componente LoginComponent, que procede a realizar una llamada a la API de Google para obtener su idToken, con el que llamará a la ruta '/auth/login' del backend. En este caso el backend se limitará a buscar el usuario en la base de datos, y al no encontrarlo devolverá un 401 Unauthorized con un mensaje de error especificando que el usuario no ha sido encontrado, por lo que el frontend entenderá que el usuario no está registrado.

Si el usuario no está registrado, se realiza una segunda llamada a la API de Google para solicitar los permisos necesarios para acceder al calendario del usuario. Después de obtener el token de identificación que incluye estos permisos, LoginComponent envía una solicitud de registro al backend en la ruta '/auth/signup'. Este proceso se realiza a través de las funciones de callback de la librería de Google antes mencionada y Axios para realizar llamadas al backend.

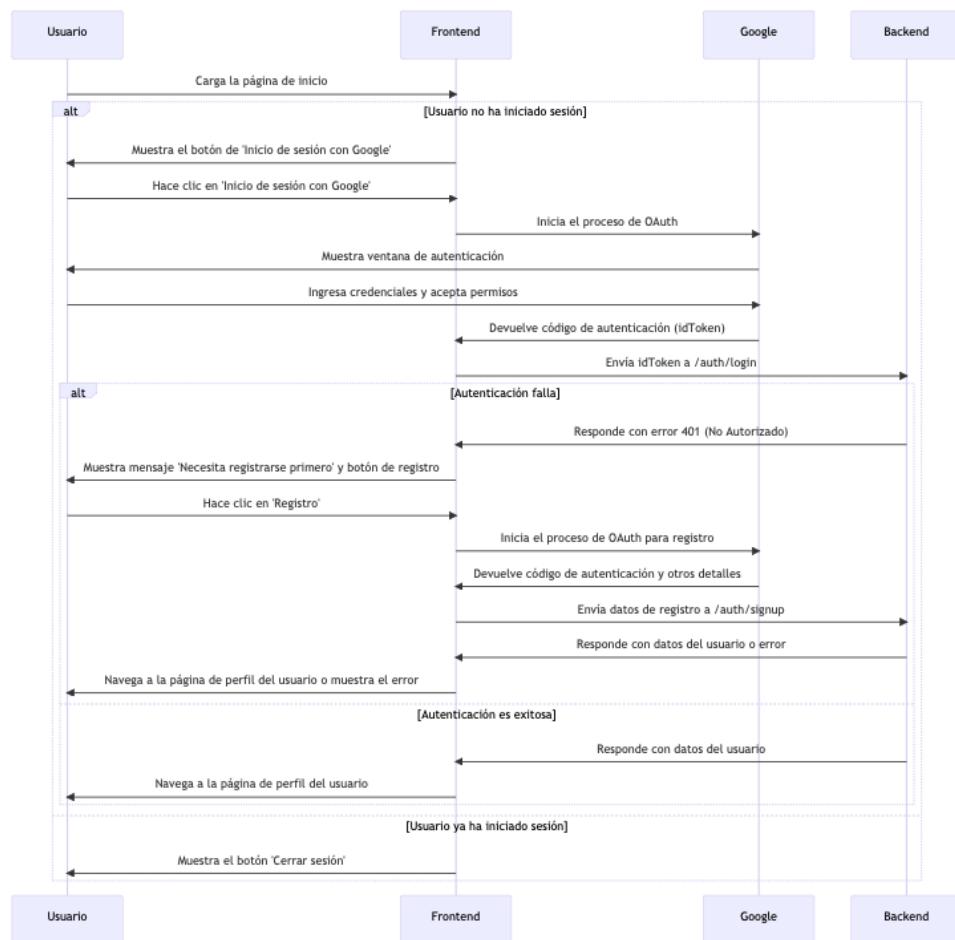


Figura <TBD>: Diagrama de flujo del LoginComponent del frontend

FLUJO EN EL BACKEND

El backend recibe la solicitud a través de la ruta especificada en AuthController, ruta que no necesita de usuario : POST '/auth/signup'.

El controlador entonces llama al método `verify(idToken)` de `GoogleTokenVerifier` para verificar que es un token valido. Si el resultado de la validación es correcto entonces se llama al método `signupUser(payload, codeToken)` de `AuthenticationService`.

Dentro de dicho método se utiliza el método `newTokenRequest` del `AuthorizationCodeFlow` para obtener, llamando a Google, las credenciales necesarias en base al token recibido por parte del frontend. Una vez obtenidas son guardadas en base de datos usando el método `createAndStoreCredential(tokenResponse, email)` del mismo `AuthorizationCodeFlow`.

Una vez guardadas las credenciales de Google se guarda en base de datos el usuario correspondiente con su email como id, usando para ello el método `save` de `UserRepository` y el controlador devuelve un código 201 con el contenido del usuario en el body codificado como json, que el frontend usará para cargar la página de perfil del usuario.

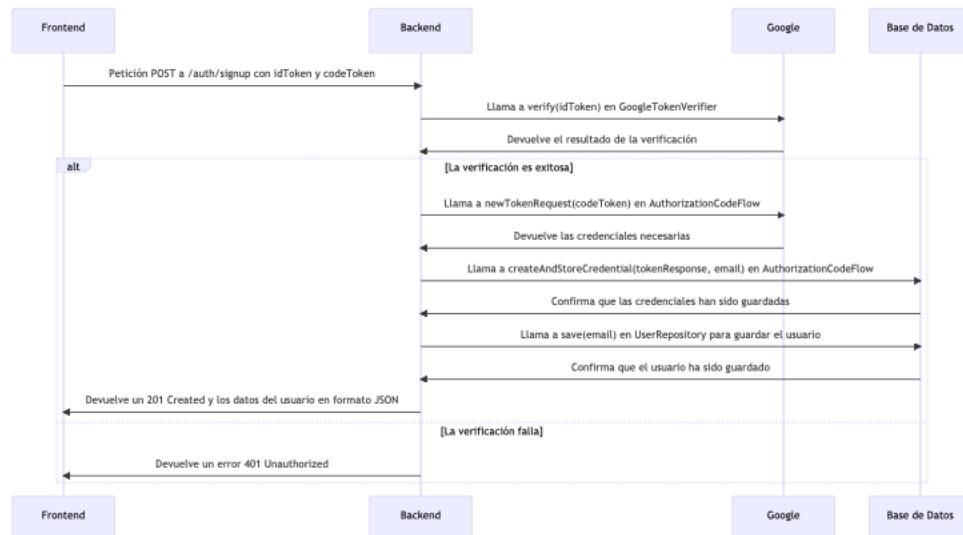


Figura <TBD>: Flujo de registro de usuario en el backend

INICIAR SESIÓN CON LA CUENTA DE GOOGLE

Para este caso el proceso es similar en el frontend para el caso de registro de usuario, ya que se lleva a cabo el mismo flujo descrito en el caso anterior, pero en la primera llamada al backend a la ruta POST '/auth/login' esta sí devolverá un 200 OK con los datos del usuario, al estar ya registrado.

FLUJO EN EL FRONTEND

Se produce el mismo flujo que en el caso anterior, pero al encontrar el backend al usuario tras llamar al endpoint de login, se carga el usuario y se redirige a la página de perfil del usuario directamente.

FLUJO EN EL BACKEND

En este caso en la primera llamada al endpoint de login '/auth/login' se valida el token pasado por el frontend en la clase `GoogleAuthenticationProvider`, al tratarse de un endpoint que requiere de autorización. Esta clase valida que el token sea validado mediante el método `verify(token)` de `GoogleTokenVerifier`, y una vez verificado usa el método `loadCredential(email)` de `AuthorizationCodeFlow`

para cargar el los datos de Google del usuario correspondiente de base de datos. También, acto seguido, se procede a verificar que el token de acceso sigue siendo valido y se solicita uno nuevo si no es así mediante los métodos `getAccessToken()`, que devolverá un nulo si este ha dejado de ser valido, y el método `refreshToken()` de la instancia de credenciales del usuario obtenida anteriormente.

Una vez validado tanto la validez del `idToken` recibido del frontend, la existencia del usuario en base de datos y la no caducidad de su token de acceso, se llama al `AuthController`, donde el método correspondiente a la ruta `'/auth/login'` se encarga de llamar al método `getUser(email)` de `UserService`, el cual a su vez llama a `findById(email)` de `UserRepository`. El controller devuelve un código 200 OK y la información del usuario en formato json como parte del body.

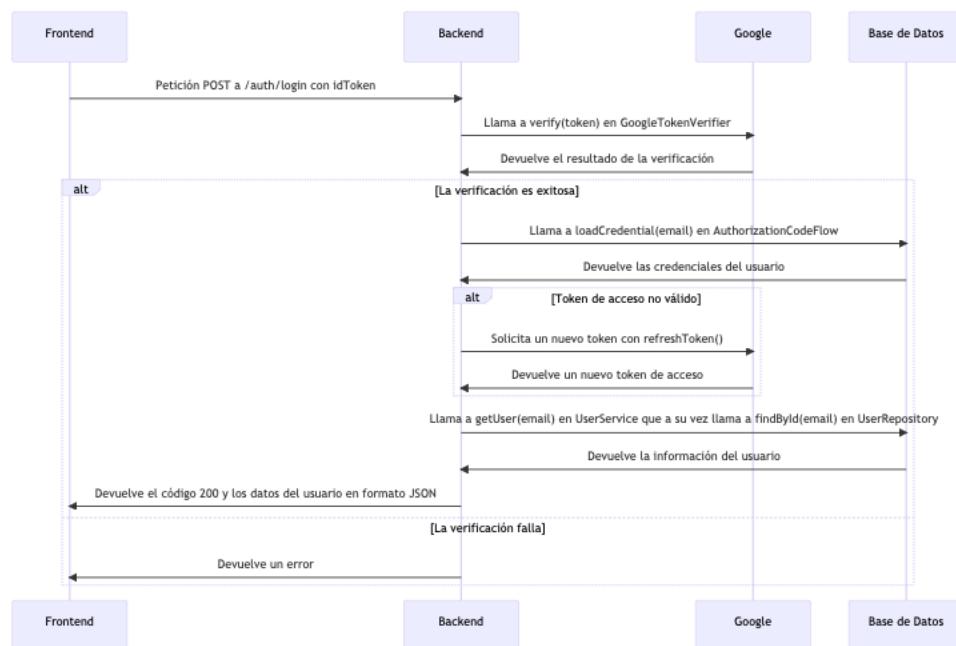


Figura <TBD>: Flujo de login del usuario en el backend

CONFIGURAR EL TIEMPO DE TRABAJO

En este caso en el frontend, una vez situado en la página del perfil del usuario permite al mismo modificar su horario de trabajo mediante un modal. Una vez el usuario guarda los cambios en el modal se llama al backend, concretamente a la ruta PATCH `'/user'`, donde se guardan los cambios en base de datos.

Para ello en el backend se realizan las tareas ya descritas correspondientes a cualquier endpoint securizado, y en el controller se llama al método `updateUser(email, updateRequest)` de `UserService`, el cual a su vez llama a los métodos `findById(email)` y `update(updatedUser)` de `UserRepository`

respectivamente para poder obtener el usuario actual, modificar los datos y guardar los datos de nuevo en base de datos.

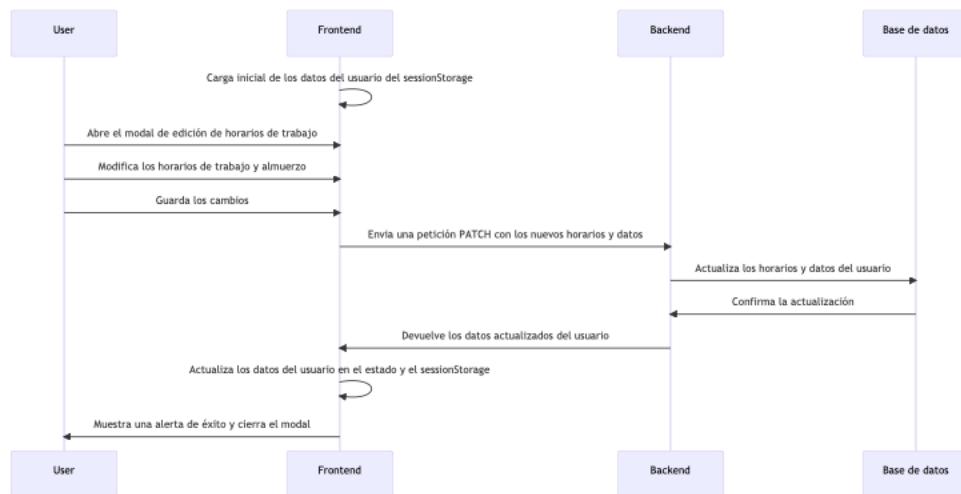


Figura <TBD>: Actualización de los datos de un usuario

SELECCIONAR CALENDARIOS PARA LOS CÁLCULOS

En este caso en el frontend, una vez situado en la página del perfil del usuario permite al mismo modificar su calendario seleccionado de un desplegable. El frontend llama desde UserProfile al backend para obtener la lista de calendarios del usuario, hace una petición GET a /user/calendars que retornara una lista de los calendarios del usuario en formato json.

Para ello en el backend se realizan las tareas ya descritas correspondientes a cualquier endpoint securizado, y en el controller se llama al método getUserCalendars(email) de UserService, el cual:

1. Inicializa el servicio de la librería de Google Calendar que realizara las peticiones necesarias para realizar llamadas a la API de Google Calendar con las credenciales del usuario y de la app.
2. Realiza una llamada a la API de Google Calendar para obtener los calendarios del usuario y los filtra eliminando los ocultos y eliminados.

Una vez el frontend tiene los calendarios se los muestra al usuario en un desplegable, el cual podrá cambiar y cuando en el frontend pulse en el botón de guardado el componente UserProfile del frontend enviara una petición PATCH a '/user' donde indicara el id del calendario seleccionado por el usuario.

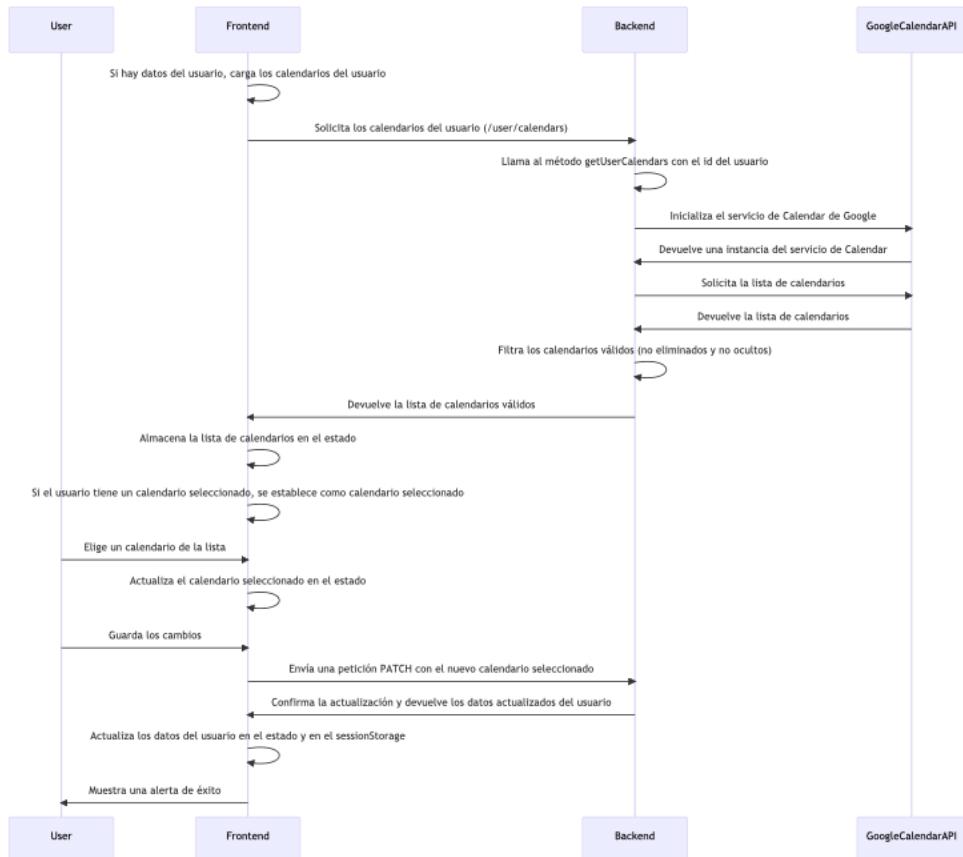


Figura <TBD>: Flujo de modificación de calendario para realizar los cálculos

SELECCIONAR PERIODO TEMPORAL PARA LOS CÁLCULOS

En este caso en el frontend, una vez situado en la página del perfil del usuario, el componente UserProfile muestra un campo de entrada que permite al usuario introducir el número de días que quiere analizar.

Una vez el usuario modifica el número de días a analizar, el estado local del componente (`daysToAnalyze`) se actualiza con el nuevo valor. Cuando el usuario esté satisfecho con los cambios, puede pulsar en el botón de guardado. Al hacer clic en el botón de guardado, el frontend enviará una petición PATCH a '/user' en el backend con la nueva configuración de días a analizar.

En el backend, se recibirá la petición en el endpoint correspondiente y se procederá a actualizar la información del usuario en la base de datos con los nuevos días a analizar tal. Para hacer esto, el backend ejecutará el método `updateUser(email, updateUserDTO)` del `UserService`, que se encargará de realizar la actualización correspondiente en la base de datos. Una vez que la actualización se ha realizado correctamente, el backend devolverá una respuesta con el estado 200 OK y los datos actualizados del usuario en formato JSON.

Finalmente, el frontend recibirá la respuesta del backend, actualizará el estado local del componente y el `sessionStorage` con los nuevos datos del usuario, y mostrará una alerta de éxito para confirmar que los cambios se han guardado correctamente.

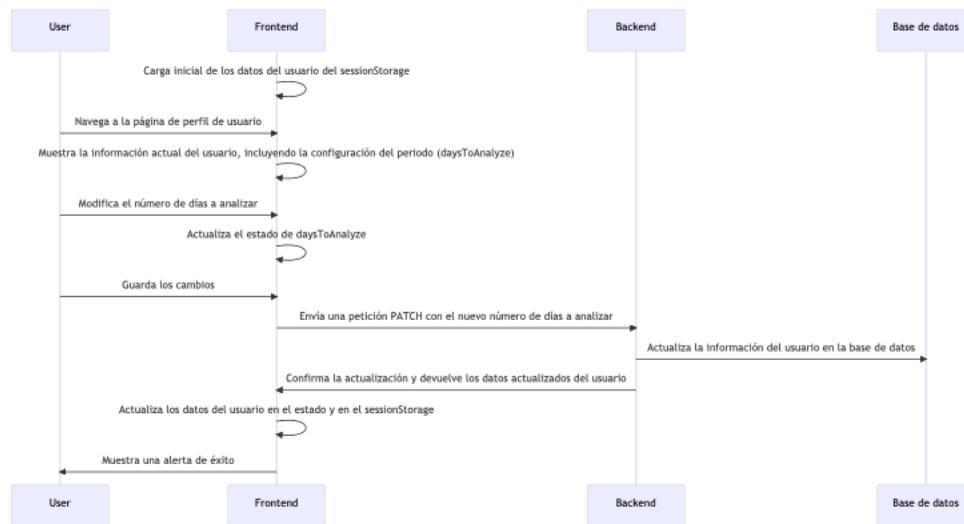


Figura <TBD>: Flujo de actualización del periodo de cálculo

CONFIGURAR EL TIEMPO ENTRE REUNIONES QUE CUENTA COMO TIEMPO DE REUNIÓN

Este caso es muy similar al anterior. En este caso en el frontend, una vez situado en la página del perfil del usuario, en el componente UserProfile existe un campo de entrada que permite al usuario introducir el tiempo deseado (en minutos) entre reuniones.

Cuando el usuario modifica este valor, el estado local del componente (timeBetweenMeetings) se actualiza con el nuevo valor. Una vez que el usuario está satisfecho con los cambios, puede pulsar en el botón de guardado. Al hacer clic en el botón de guardado, el frontend envía una petición PATCH a '/user' en el backend con la nueva configuración de tiempo entre reuniones.

En el backend, esta petición es recibida en el endpoint correspondiente y se procede a actualizar la información del usuario en la base de datos con el nuevo tiempo entre reuniones. Para hacer esto, el backend ejecuta el método updateUser(email, updateUserDTO) del UserService, que se encargará de realizar la actualización correspondiente en la base de datos. Una vez que la actualización se ha realizado correctamente, el backend devuelve una respuesta con el estado 200 OK y los datos actualizados del usuario en formato JSON.

Finalmente, el frontend recibe la respuesta del backend, actualiza el estado local del componente y el sessionStorage con los nuevos datos del usuario, y muestra una alerta para confirmar que los cambios se han guardado correctamente.

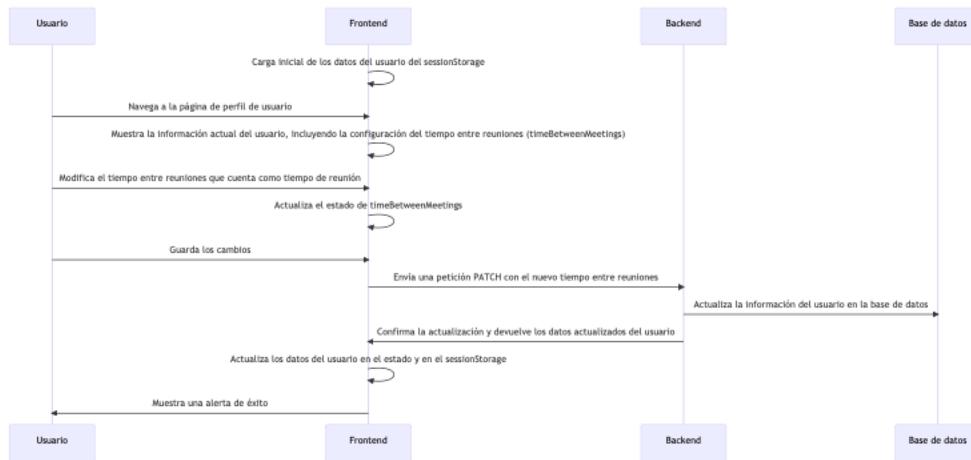


Figura <TBD>: Flujo de actualización del tiempo entre reuniones que se utiliza para hacer los cálculos

VER EL PORCENTAJE DE TIEMPO GASTADO EN REUNIONES Y TIEMPO LIBRE

Este caso de uso es más complejo, debido al procesamiento que se hace en el backend.

FLUJO EN EL FRONTEND

En este caso, en el frontend, una vez situado en la página "Time analysis", el componente WorkHoursComponent extrae la información del usuario de sessionStorage. Inicialmente, intenta cargar los datos de análisis de tiempo del usuario, si estos no están disponibles, realiza una petición GET a '/user/timeAnalysis' en el backend para obtenerlos.

Cuando el frontend recibe los datos, los almacena en el estado local del componente (timeAnalysisData). Luego, se calculan las horas ocupadas, tentativas, fuera de la oficina y libres, y se preparan para mostrarlas en una tabla y un gráfico de tarta. El cálculo de las horas libres se realiza restando las horas ocupadas, tentativas y fuera de la oficina del total de horas trabajadas. Todos estos datos se muestran en una tabla con sus respectivos porcentajes. El gráfico de tarta se genera a partir de estos datos, excluyendo cualquier elemento con valor 0. Finalmente, tanto la tabla como el gráfico de tarta se presentan al usuario en la página de análisis de tiempo.

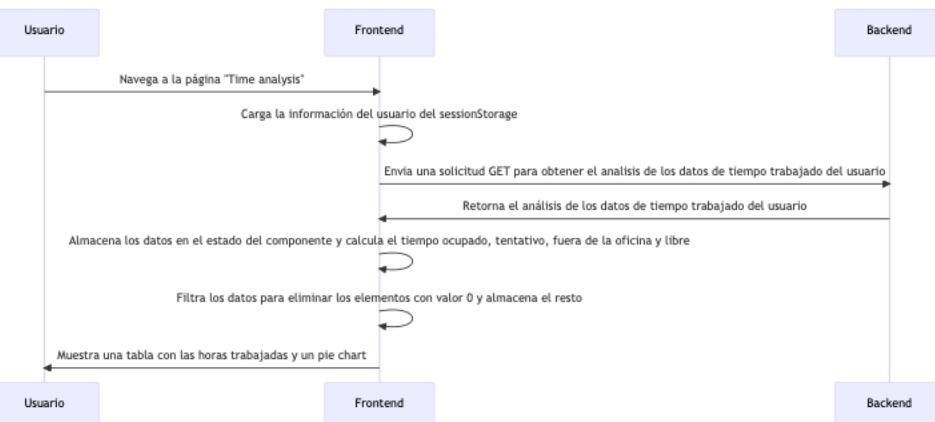


Figura <TBD>: Flujo en el frontend de obtención del análisis de los datos de tiempo de trabajo.

FLUJO EN EL BACKEND

El método principal es `getTimeSpentInMeetings(String userEmail)`. Este método es llamado desde el controller al recibir la petición GET del frontend. Y realiza lo siguiente:

1. Recupera la información del usuario a través del método `getUser(String userEmail)` del servicio `UserService`.
2. Calcula el intervalo de tiempo a analizar utilizando el método `daysToAnalyze()` del objeto `user`, y lo usa para obtener las fechas de inicio y fin.
3. A continuación, llama al método `getMeetings(UserDTO user, LocalDate startDate, LocalDate endDate)`, el cual utiliza `getCalendarMeetings(user, startDate, endDate)` del servicio `CalendarService` para obtener una lista de todas las reuniones del usuario en el intervalo de tiempo especificado.
4. Esta lista de reuniones se pasa al método `getMeetingsMap(List<Meeting> meetings)`, que crea un mapa de objetos `MeetingDay` (uno por cada día que tiene al menos una reunión) indexado por la fecha de las reuniones.
5. Después, se llama a `calculateBusyTentativeOooHours(Map<LocalDate, MeetingDay> meetingsMap, UserDTO user, LocalDate startDate, LocalDate endDate)`, donde está la principal lógica de cálculo y que se encarga de calcular el total de horas ocupadas, tentativas y fuera de la oficina (OOO) para cada día en el intervalo de tiempo.
6. Después, el método `calculateWorkedDays(LocalDate startDate, LocalDate endDate)` calcula la cantidad de días laborales en el intervalo de tiempo, excluyendo sábados y domingos.
7. Finalmente, `getTimeSpentInMeetings(String userEmail)` devuelve una nueva instancia de `TimeAnalysisResults`, que contiene la cantidad total de horas ocupadas, tentativas y OOO, así como las horas trabajadas totales, calculadas multiplicando los días trabajados por las horas de trabajo del usuario obtenidas con `getWorkingHours(UserDTO user)`.

Además, la clase contiene métodos privados que dividen la lógica para hacerla más legible, son los siguientes:

Los métodos `calculateBusyTentativeOooMinutesForDay(Map<LocalDate, MeetingDay> meetings, LocalDate day, int resolutionMinutes, int workingHours, LocalTime startHour, LocalTime endHour)` y `calculateBusyTentativeOooMinutesForInterval(Meeting[] busyMeetings, Meeting[] tentativeMeetings, Meeting[] oooMeetings, LocalTime previousInterval, LocalTime interval, int resolutionMinutes)` trabajan juntos para calcular el total de minutos ocupados, tentativos y OOO para cada día e intervalo de tiempo respectivamente.

El método `adjustDayMinutesToWorkingMinutes(int busyDay, int tentativeDay, int oooDay, int workingMinutes, LocalDate day)` se encarga de asegurar que la cantidad total de minutos (ocupados, tentativos y OOO) no exceda la cantidad de minutos de trabajo en un día. En caso de que sí lo haga, llama a `subtractMinutesFromMeetings(int busyDay, int tentativeDay, int oooDay, int difference)` para ajustar la cantidad de minutos.

Finalmente, `hasMeetingInInterval(Meeting[] meetings, LocalTime start, LocalTime end)` es un método auxiliar que verifica si existe alguna reunión dentro de un intervalo de tiempo específico.

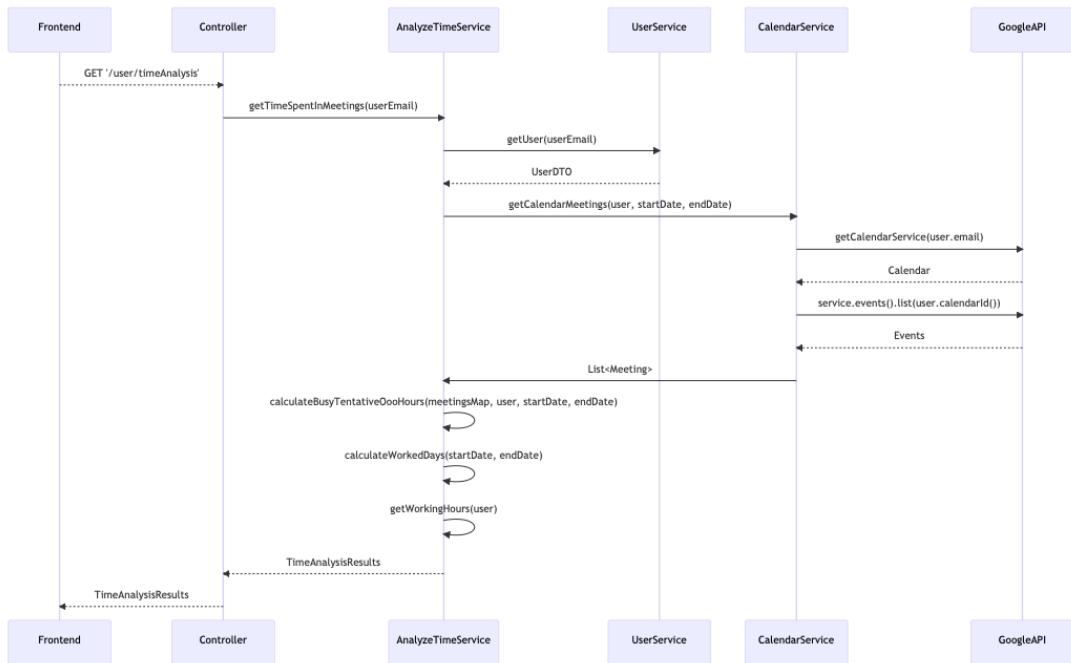


Figura <TBD>: Flujo en backend de cálculo de tiempo gastado en reuniones

DIAGRAMA DE CLASES DE DISEÑO

BACKEND

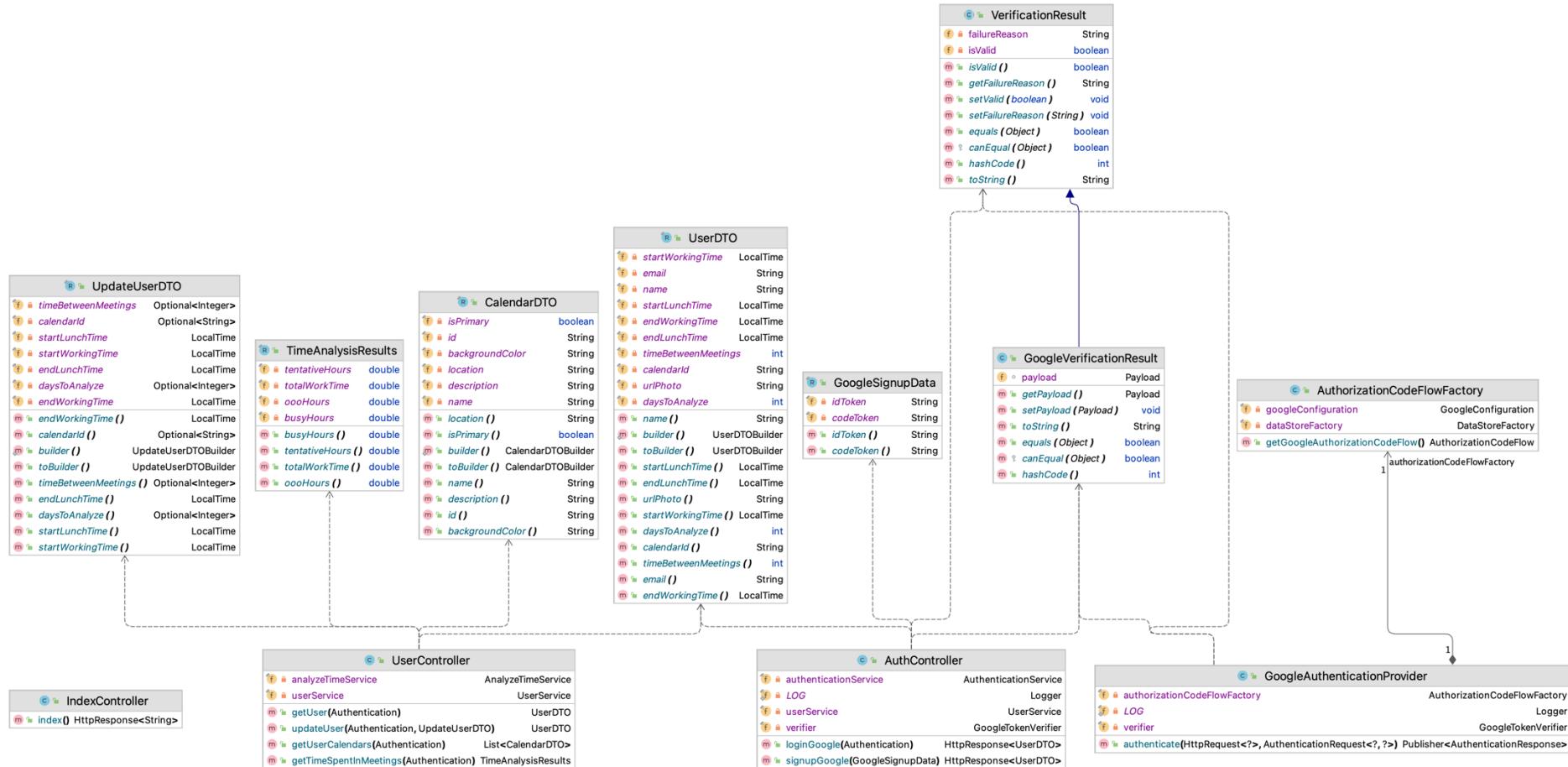


Figura <TBD>: Diagrama de clases de los paquetes controller y authentication, y clases relacionadas del paquete domain

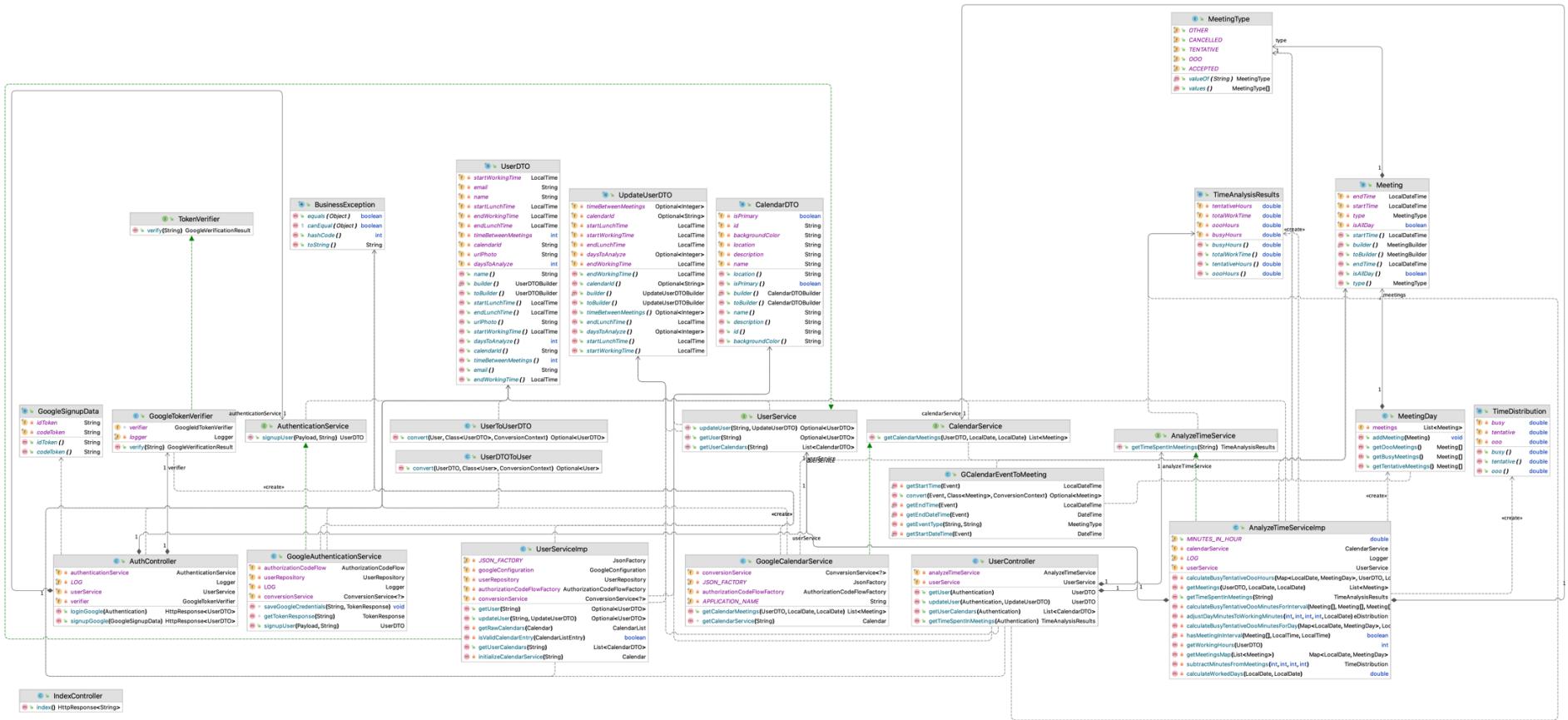


Figura <TBD>: Diagrama de clases de los paquetes service y controller, y clases relacionadas de los paquetes domain, conversion y exceptions



Figura <TBD>: Diagrama de clases del paquete repository y clases relacionadas de service y authentication

FRONTEND

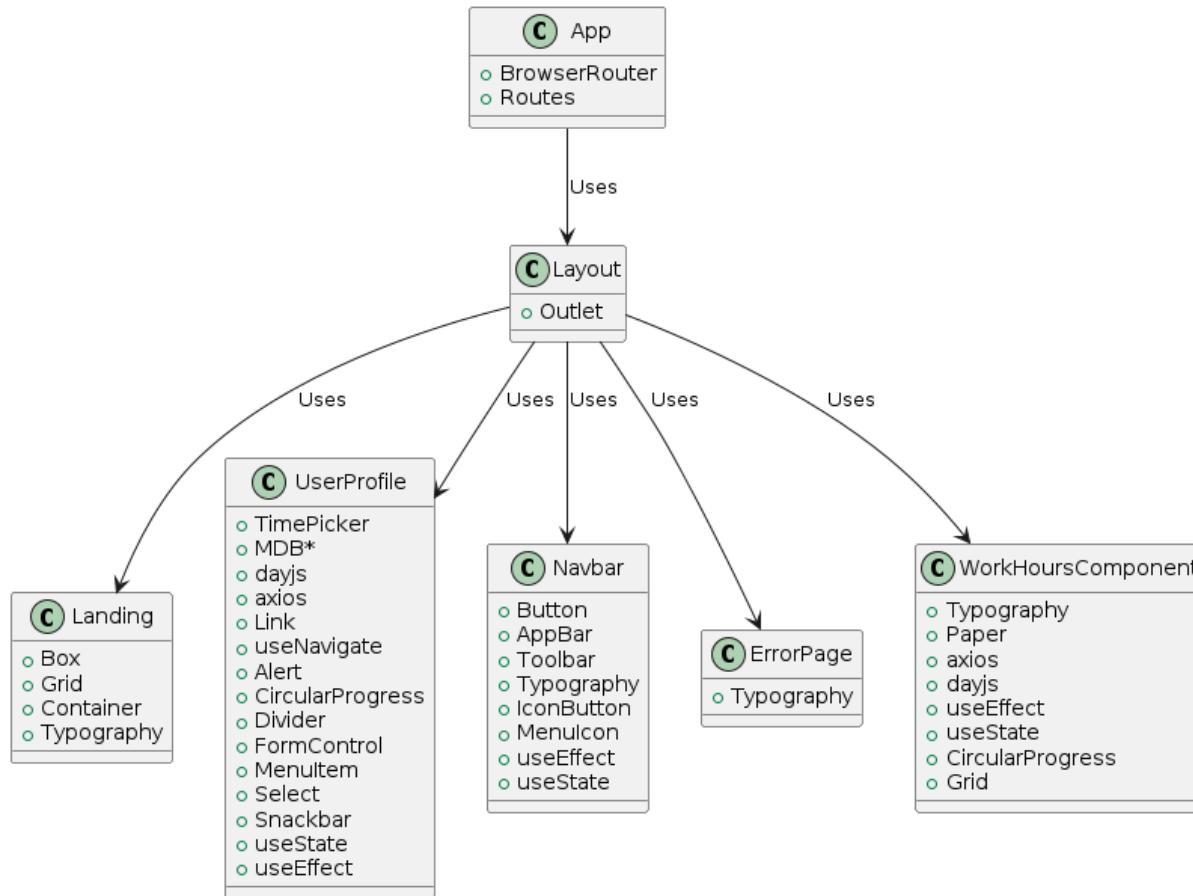


Figura <TBD>: Diagrama de clases del frontend basado en la navegación y el uso de composición de componentes de React

DESCRIPCIÓN DE LAS CLASES DE DISEÑO

CLASES DE DISEÑO DEL BACKEND

El backend se estructura en varios paquetes que encapsulan diferentes áreas funcionales del software. Cada uno de estos paquetes contiene una serie de clases diseñadas para cumplir con los requisitos específicos de su dominio.

En este apartado, se proporciona una descripción detallada de las clases diferenciando por paquetes. Esta descripción abarcará el propósito de cada clase, sus responsabilidades principales y cómo interactúan con otras clases en el sistema.

AUTHENTICATION

COM.DFERNANDEZALLER.AUTHENTICATION

- **VerificationResult:** Esta es una clase abstracta que encapsula el resultado de un proceso de verificación. Contiene dos campos, un booleano `isValid` para indicar si la verificación fue exitosa y una cadena `failureReason` para almacenar el motivo del fallo en caso de que la verificación no haya tenido éxito.

COM.DFERNANDEZALLER.AUTHENTICATION.GOOGLE

- **AuthorizationCodeFlowFactory:** Esta clase es un singleton y una fábrica que se encarga de construir y proporcionar un objeto de tipo `AuthorizationCodeFlow` configurado para Google. Utiliza una `DataStoreFactory` y una `GoogleConfiguration` para configurar el flujo de autorización.
- **GoogleAuthenticationProvider:** Esta clase es un proveedor de autenticación que implementa la interfaz `AuthenticationProvider`. Su principal función es proporcionar un método para autenticar una solicitud HTTP y una solicitud de autenticación. Verifica el token de autenticación y, si es válido, carga las credenciales del usuario y verifica el token de acceso. En caso de que el token de acceso esté desactualizado, intenta actualizarlo. Este proveedor de autenticación utiliza un `GoogleTokenVerifier` y un `AuthorizationCodeFlowFactory`. Todas las peticiones a endpoints que requieran de autenticación son procesadas por esta clase.

COM.DFERNANDEZALLER.AUTHENTICATION.GOOGLE.DTO

- **GoogleVerificationResult:** Esta clase hereda de `VerificationResult` y agrega un campo adicional, `payload`, que es un objeto de tipo `GoogleIdToken.Payload`. Esta clase encapsula el resultado de la verificación de un token de Google. Al igual que su clase padre, puede indicar si la verificación fue exitosa y proporcionar el motivo del fallo en caso contrario. Además, también proporciona la información útil contenida en el payload del token.

CONTROLLER

COM.DFERNANDEZALLER.CONTROLLER

- **AuthController:** Esta clase es un controlador responsable de manejar las solicitudes de autenticación y registro. El controlador `AuthController` utiliza los servicios `AuthenticationService`, `UserService` y `GoogleTokenVerifier` para autenticar a los usuarios y registrar nuevos usuarios en el sistema.
 - *loginGoogle(Authentication authentication):* Este método maneja las solicitudes de autenticación y devuelve una respuesta HTTP que contiene los detalles del usuario si la autenticación es exitosa. Si el usuario no se encuentra, se devuelve un error HTTP 401 - No autorizado.

- *signupGoogle(GoogleSignupData signupData)*: Este método maneja las solicitudes de registro de usuarios. Verifica el token de identificación proporcionado y si es válido, registra al usuario en el sistema. Si el token no es válido, se devuelve un error HTTP 401 - No autorizado.

COM.DFERNANDEZALLER.CONTROLLER.DTO

- **CalendarDTO**: Este objeto de transferencia de datos se utiliza para enviar y recibir detalles del calendario del usuario. Los campos de este DTO incluyen el nombre, el ID, el color de fondo, la descripción, la ubicación y si es el calendario principal del usuario.
- **GoogleSignupData**: Este DTO se utiliza en el proceso de registro de un usuario. Contiene un codeToken y un idToken que se utilizan para autenticar y registrar al usuario en el sistema.
- **UpdateUserDTO**: Este DTO se utiliza para actualizar los detalles de un usuario existente. Los campos de este DTO incluyen las horas de inicio y fin de trabajo y almuerzo, el tiempo entre reuniones, el ID del calendario y los días a analizar.
- **UserDTO**: Este objeto de transferencia de datos se utiliza para enviar y recibir detalles del usuario. Los campos de este DTO incluyen el correo electrónico, el nombre, la URL de la foto del usuario, las horas de inicio y fin de trabajo y almuerzo, el tiempo entre reuniones, el ID del calendario y los días a analizar.

SERVICE

Describiré directamente las implementaciones de las interfaces ya que cada interfaz tiene una implementación.

COM.DFERNANDEZALLER.SERVICE.IMP

- **AnalyzeTimeServiceImp**: es la implementación de la interfaz AnalyzeTimeService. Esta clase se encarga de analizar el tiempo dedicado por el usuario a las reuniones, y proporciona un desglose de dicho tiempo en diferentes categorías.
 - *getTimeSpentInMeetings(String userEmail)*: Este método recibe un correo electrónico de usuario, busca el usuario correspondiente y calcula el tiempo total que el usuario ha pasado en las reuniones. Esto se hace al obtener las reuniones del usuario en un período específico, clasificar estas reuniones por fecha, y luego calcular el tiempo ocupado, tentativo y fuera de la oficina (ooo) para cada día.
 - *getMeetings(UserDTO user, LocalDate startDate, LocalDate endDate)*: Este método recibe un usuario, una fecha de inicio y una fecha de fin, y luego llama al servicio CalendarService para obtener todas las reuniones del usuario dentro de ese rango de fechas.
 - *calculateWorkedDays(LocalDate startDate, LocalDate endDate)*: Este método calcula la cantidad total de días trabajados entre dos fechas dadas, excluyendo los sábados y domingos.
 - *getWorkingHours(UserDTO user)*: Este método devuelve la cantidad de horas de trabajo por día para un usuario específico. Esto se calcula restando el tiempo de almuerzo del tiempo total de trabajo.
 - *getMeetingsMap(List<Meeting> meetings)*: Este método clasifica una lista de reuniones por fecha, guardándolas en un Map donde la clave es la fecha y el valor es un objeto MeetingDay que contiene todas las reuniones de ese día.
 - *calculateBusyTentativeOooHours Map<LocalDate, MeetingDay> meetingsMap, UserDTO user, LocalDate startDate, LocalDate endDate*): Este método calcula el tiempo total ocupado, tentativo y fuera de la oficina para cada día dentro de un rango de fechas dado.

- *calculateBusyTentativeOooMinutesForDay (Meeting[] busyMeetings, Meeting[] tentativeMeetings, Meeting[] oooMeetings, LocalTime previousInterval, LocalTime interval, int resolutionMinutes)*: Este método calcula el tiempo ocupado, tentativo y fuera de la oficina en minutos para un día específico.
- **UserServiceImpl**: Esta es la implementación de la interfaz UserService. Se encarga todas las operaciones relacionadas con los usuarios. Esta anotada tanto con @Singleton como con @Transactional, esta última anotación para garantizar que todas las operaciones de base de datos se manejen como una única transacción.
 - *getUser(String email)*: Se comunica con el repositorio para obtener los datos del usuario y los convierte en un objeto UserDTO.
 - *updateUser((String email, UpdateUserDTO requestDTO))*: Obtiene de base de datos los datos actuales del usuario, actualiza los campos proporcionados en el objeto UpdateUserDTO, guarda el usuario actualizado en la base de datos y devuelve el UserDTO actualizado.
 - *getUserCalendars(String name)*: Inicializa el servicio de calendario de Google, recupera los calendarios del usuario, filtra las entradas no válidas y las convierte en objetos CalendarDTO.
- **GoogleTokenVerifier**: Esta es una implementación de la interfaz TokenVerifier. Se encarga de verificar la validez de los token que se les pasa a su único método mediante el uso de la API de Google.
 - *GoogleTokenVerifier(GoogleConfiguration googleConfiguration)*: Este constructor inicializa el GoogleIdTokenVerifier con un NetHttpTransport y una instancia de GsonFactory. Se configura el verificador para que use el clientId de la configuración de Google.
 - *verify(String token)*: Este método intenta verificar el token proporcionado utilizando el GoogleIdTokenVerifier. Si el token es válido, se crea un nuevo objeto GoogleVerificationResult con campo isValid a true, una cadena vacía como mensaje y la carga útil del token. En caso de que el token no sea válido, se crea un GoogleVerificationResult con campo isValid a false, un mensaje indicando que el token es inválido y una carga útil nula. Durante la verificación, si ocurre una GeneralSecurityException, se lanza una nueva SecurityException con un mensaje explicativo. Del mismo modo, si se produce una IOException, se lanza una BusinessException.
- **GoogleAuthenticationService**: Esta es la implementación de la interfaz AuthenticationService. Se encarga del proceso de signup de los usuarios de Google. Está anotada con @Singleton y @Transactional, esta última implica que todas las operaciones de la base de datos se manejarán como una única transacción.
 - *signupUser(GoogleIdToken.Payload payload, String token)*: Este método es el encargado de registrar a un nuevo usuario. Primero obtiene una respuesta de token a través del método getTokenResponse(token), luego intenta guardar las credenciales de Google con el método privado *saveGoogleCredentials(payload.getEmail(), tokenResponse)*. A continuación, se construye un nuevo objeto User con la información de payload, se guarda en la base de datos y se convierte en un UserDTO para ser retornado.
 - *saveGoogleCredentials(String email, TokenResponse tokenResponse)*: Este método privado verifica si ya existen credenciales para el correo electrónico dado, si es así, se lanza una BusinessException indicando que el usuario ya existe. En caso contrario, crea y guarda las nuevas credenciales y registra un mensaje de seguimiento.

- *getTokenResponse(String token)*: Este método privado intenta crear token que contenga token de acceso, de refresco y la información del usuario a partir del idToken proporcionado. Si se produce una IOException, se lanza una BusinessException con un mensaje explicativo.
- **GoogleCalendarService**: Esta es la implementación de la interfaz CalendarService. Se encarga de aquellas operaciones relacionadas con el calendario de Google como el obtener los calendarios del usuario. Está anotada con @Singleton.
 - *getCalendarMeetings(UserDTO user, LocalDate startDate, LocalDate endDate)*: Este método recupera las reuniones de un calendario dentro de un rango de fechas especificado. Intenta inicializar el servicio de calendario de Google, realiza una petición para obtener los eventos en el rango de fechas establecido y los ordena por la hora de inicio. Luego, convierte cada evento en un objeto Meeting y los devuelve como una lista. En caso de error durante la comunicación con el servicio de Google, se lanza una BusinessException.
 - *getCalendarService(String userEmail)*: Este método privado intenta inicializar el servicio de calendario de Google. Para ello, crea una nueva conexión segura HTTP con Google y utiliza las credenciales del usuario para construir el servicio de calendario. En caso de un error durante la creación de la conexión segura, se lanza una BusinessException.

EXCEPTIONS

COM.DFERNANDEZALLER.EXCEPTIONS

- **BusinessException**: Esta es una clase que extiende RuntimeException, lo que significa que es una excepción no comprobada que puede lanzarse durante la ejecución normal del programa. Está anotada con @Value y @EqualsAndHashCode(callSuper = true), que son anotaciones de Lombok para generar automáticamente métodos *toString()*, *equals()*, *hashCode()* y getters para todos los campos, y un constructor que inicializa todos los campos. También permite que *equals()* y *hashCode()* consideren las propiedades de la clase de la que extiende (RuntimeException).
 - *BusinessException(String message)*: Este constructor toma un mensaje como argumento y lo pasa al constructor de la superclase (RuntimeException), lo que resulta en una excepción con un mensaje determinado.
 - *BusinessException(String message, Throwable cause)*: Este constructor toma un mensaje y una causa (otra excepción) como argumentos y los pasa al constructor de la superclase. Esto resulta en una excepción con un mensaje determinado y una causa subyacente.

CONFIGURATION

COM.DFERNANDEZALLER.CONFIGURATION

- **GoogleConfiguration**: Esta clase es responsable de mapear y alojar las propiedades de configuración específicas de Google en la aplicación. Las propiedades se obtienen del archivo de configuración de la aplicación (por ejemplo, un archivo *application.yaml* o *application.properties*) bajo el prefijo *google*. Esta clase es anotada con varias anotaciones:
 - *@Data*: Una anotación de Lombok que genera automáticamente los métodos getters, setters, *equals()*, *hashCode()* y *toString()* para los campos de la clase.
 - *@Serdeable*: Una anotación de Micronaut que indica que la clase puede ser serializada y deserializada automáticamente por Micronaut.

- `@ConfigurationProperties("google")`: Una anotación de Micronaut que indica que esta clase aloja propiedades de configuración con el prefijo especificado ("google" en este caso).

Los campos de la clase incluyen:

- `clientId`: El ID de cliente del proyecto de Google Cloud asociado.
- `clientSecret`: El secreto de cliente del proyecto de Google Cloud asociado. Este es un valor confidencial que no debe ser expuesto o compartido públicamente.
- `applicationName`: El nombre de la aplicación tal y como se registra en Google Cloud.

MODEL

COM.DFERNANDEZALLER.MODEL

- **Meeting**: Este es un record que representa una reunión. Esta clase tiene cuatro campos:
 - `type`: Un objeto de tipo `MeetingType` que representa el estado de la reunión.
 - `startTime` y `endTime`: Estos campos son objetos `LocalDateTime` que representan el inicio y el fin de la reunión. Están anotados con `@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")` para garantizar que se serializan y deserializan correctamente con el formato de fecha y hora especificado.
 - `isAllDay`: Un valor booleano que indica si la reunión dura todo el día.
- **MeetingDay**: Esta clase representa un día que contiene varias reuniones. Contiene una lista de objetos `Meeting`. Esta clase tiene varios métodos para filtrar y obtener reuniones de diferentes tipos (aceptadas, tentativas, fuera de la oficina).
- **MeetingType**: Este es un tipo de enumeración que representa el estado de una reunión. Los posibles estados son: OOO (fuera de la oficina), TENTATIVE (tentativo), ACCEPTED (aceptado), CANCELLED (cancelado), OTHER (otro).
- **TimeAnalysisResults**: Este es un record que contiene los resultados de un análisis de tiempo. Contiene cuatro campos de tipo `double` que representan el número total de horas ocupadas, horas tentativas, horas fuera de la oficina y el tiempo total de trabajo.
- **TimeDistribution**: Este es un record que representa la distribución del tiempo entre las horas ocupadas, tentativas y fuera de la oficina. Antes de asignar los valores a los campos, estos se redondean a dos decimales.

CONVERSION

- **GCalendarEventToMeeting**: Este es un convertidor de tipo que se utiliza para transformar un objeto `Event` (una clase que representa un evento de Google Calendar) a un objeto `Meeting` (parte del paquete Model). En este convertidor, se extrae y se ajusta la información relevante de `Event` para cumplir con la estructura de `Meeting`.
- **GoogleCalendarListEntryToCalendarDTO**: Este convertidor de tipo transforma un objeto `CalendarListEntry` (una clase de la API de Google Calendar que representa una entrada en la lista de calendarios de un usuario) a un objeto `CalendarDTO` (un objeto de transferencia de datos utilizado y declarado en el paquete Controller).
- **UserDTOToUser** y **UserToUserDTO**: Estos convertidores de tipo se utilizan para transformar entre `UserDTO` (un objeto de transferencia de datos declarado y usado en el paquete controller) y `User` (la entidad que representa al usuario en base de datos).

CLASES DE DISEÑO DEL FRONTEND

El frontend se organiza en torno a varios componentes. Como es habitual en las aplicaciones escritas con React, cada uno de estos componentes tiene un papel específico y contiene la lógica y la presentación

relacionadas con una pieza específica de la interfaz de usuario si bien muchas veces se componen unos de otros. A continuación, se proporciona una descripción detallada de los componentes clave.

Existen algunos componentes más de los aquí descritos cuyo papel es secundario, como ficheros css o clases que describen objetos que ya se han descrito en el backend como Meeting o User. Se omitirán ya que son virtualmente idénticos a los DTOs del backend.

- **App:** Este es el componente principal que encapsula toda la aplicación y define las rutas de esta. Se encarga de definir las diferentes rutas de la aplicación utilizando el componente BrowserRouter de React Router, que permite el enrutamiento en aplicaciones single page. Define rutas para la página de inicio, perfil de usuario, análisis de tiempo y una página de error.
- **Layout:** Este componente actúa como una envoltura para las rutas del frontend. Renderiza la barra de navegación en todas las páginas y utiliza el componente Outlet de React Router para renderizar la ruta actual. El Outlet simplemente renderiza la ruta que se haya seleccionado dentro del componente Routes de la aplicación.
- **Landing:** Este componente define la vista de la página de inicio de la aplicación. Utiliza el sistema de grids de Material UI para organizar el contenido en una disposición de tres columnas. Cada columna tiene un ícono, un título y una descripción. Este componente se centra en mostrar información sobre la aplicación.
- **ErrorPage:** Este es un componente simple que muestra un mensaje de "Página no encontrada". Este componente se muestra cuando el usuario navegue a una ruta que no existe en la aplicación, proporcionando un manejo básico de errores de ruta.
- **WorkHoursComponent:** Este componente proporciona una vista detallada del análisis de tiempo del usuario. Utiliza una llamada al backend para obtener datos de análisis de tiempo y luego presenta la información al usuario de dos formas: una tabla y un pie chart. Además, este componente implementa la lógica necesaria para gestionar la carga de datos y para manejar situaciones en las que los datos no están disponibles o el usuario no está autenticado. Funcionalidades clave del componente:
 - Realizar una llamada a la API del backend para obtener los datos de análisis de tiempo del usuario.
 - Almacenar y gestionar los datos obtenidos del backend en el estado local del componente.
 - Transformar estos datos en un formato adecuado para ser presentados en una tabla y en un gráfico circular.
 - Presentar los datos al usuario tanto en forma de tabla como en un gráfico circular.
 - Manejar casos en los que los datos no están disponibles o el usuario no está autenticado, redirigiendo al usuario a la página de inicio en caso de que no esté autenticado.
 - Implementar un indicador de carga para informar al usuario de que los datos están siendo obtenidos.
- **Navbar:** Este componente es la barra de navegación superior que está presente siempre en la interfaz. En ella se muestra el logo de la aplicación, un enlace a cada vista y un botón para poder iniciar sesión con Google, registrarse con Google o hacer logout en función de la situación actual.
- **LoginComponent:** Este componente forma parte de la NavBar y contiene la lógica necesaria para llevar a cabo el login, el logout, el registro del usuario y los cambios visuales del botón correspondiente de la navbar. Funcionalidades clave del componente:
 - Realizar una llamada a la API de Google para hacer el login del usuario.
 - Realizar una llamada a la API del backend con los datos de la llamada a Google para proceder al login del usuario.

- Realizar una llamada a la API de Google para pedir los permisos necesarios en caso de que el backend responda que no está registrado al intentar hacer el login.
- Realizar una llamada a la API del backend para proceder al registro del usuario con los datos de la llamada a Google para obtener el token con los permisos de lectura de calendario.
- Almacenar y gestionar los datos obtenidos del backend y de Google en el estado local del componente.
- Manejar el estado del botón que ha de mostrarse en cada momento, así como mostrar una alerta si es necesario proceder al registro del usuario.
- **UserProfile:** Este componente es el componente que representa la vista del perfil del usuario. Permite ver la información del usuario como su horario laboral actual, para cuantos días ha de realizarse el análisis de las reuniones del calendario, el calendario a analizar seleccionado, foto y datos personales del usuario y el horario dedicado a almorzar. Además de mostrar esta información contiene la lógica necesaria para modificarla. Funcionalidades clave del componente:
 - Realizar una llamada a la API del backend para poder cargar los calendarios del usuario, y mostrar un spinner de carga mientras se obtienen en el sitio donde se ubica el desplegable para seleccionar calendario.
 - Realizar una llamada a la API del backend para obtener los datos del usuario.
 - Realizar una llamada a la API del backend para modificar los datos del usuario.
 - Mostrar la información relativa al horario laboral y el horario de dedicado a almorzar en un modal y realizar una llamada al backend si los datos son modificados.
 - Mostrar una alerta que informa al usuario si se han guardado los cambios con éxito.

MODELO DE BASE DE DATOS

La base de datos es una base de datos relacional, en concreto se usa una instancia de PostgreSQL v.15 desplegada en la nube de Azure.

Una cosa a tener en cuenta en el diseño es que, aunque es un requisito conocido por el backend el que los IDs del usuario de la aplicación y de las credenciales sea el email y este deba coincidir, no se ha establecido esa relación a la hora de modelar la base de datos debido a que se ha procurado en todo momento hacer un diseño que pueda ser ampliado de forma sencilla a otros sistemas de calendario como Microsoft Outlook, y establecer esa restricción en base de datos puede ser contraproducente para ese objetivo. No obstante, esta restricción para el caso de Google Calendar es controlada en la lógica de la aplicación, las operaciones de guardado de usuarios son transaccionales para evitar tener estados inconsistentes en la base de datos.

A continuación, se puede ver una figura que describe el modelo E-R.

Users		Google_credential	
string	email	string	key
string	name	string	access_token
string	urlPhoto	Long	expiration_time_milliseconds
LocalTime	startWorkingTime	string	refresh_token
LocalTime	endWorkingTime	Instant	created_at
LocalTime	startLunchTime	Instant	updated_at
LocalTime	endLunchTime		
int	timeBetweenMeetings		
string	calendarId		
int	daysToAnalyze		

Figura <TBD>: Modelo E-R de la base de datos de la aplicación

DESCRIPCIÓN DE LAS TABLAS

USERS

Esta tabla almacena la información de los usuarios del sistema. Cada registro en esta tabla representa un usuario individual con sus respectivos detalles y configuraciones.

Atributos:

- email (clave primaria): Representa el correo electrónico único asociado al usuario.
- name: Almacena el nombre del usuario.
- url_photo: Contiene la URL de la foto del usuario.
- startWorkingTime: Indica la hora de inicio de trabajo del usuario.
- endWorkingTime: Indica la hora de finalización de trabajo del usuario.
- startLunchTime: Representa la hora de inicio del tiempo de almuerzo del usuario.
- endLunchTime: Representa la hora de finalización del tiempo de almuerzo del usuario.
- timeBetweenMeetings: Especifica la duración en minutos entre las reuniones del usuario que ha de tomarse como tiempo de reunión durante el análisis.
- calendarId: Almacena el ID del calendario asociado al usuario que se usará para el análisis.
- daysToAnalyze: Indica la cantidad de días a analizar para programar reuniones y eventos en el calendario del usuario.

GOOGLE_CREDENTIALS

Esta tabla almacena las credenciales de Google utilizadas para autenticación y autorización en el sistema. Las credenciales están asociadas a un usuario específico y se utilizan para acceder a los servicios de Google.

Atributos:

- id (clave primaria): Representa la clave única que identifica las credenciales de Google.

- `access_token`: Almacena el token de acceso utilizado para autenticación en los servicios de Google.
- `expiration_time_milliseconds`: Indica el tiempo de expiración en milisegundos para el token de acceso.
- `refresh_token`: Contiene el token de actualización utilizado para obtener un nuevo token de acceso cuando el actual expira.
- `created_at`: Registra la fecha y hora de creación de las credenciales.
- `updated_at`: Registra la fecha y hora de la última actualización de las credenciales.

DISEÑO DE LA INTERFAZ

A continuación, se mostrará el diseño final de las vistas ya planificadas en la sección mockups del capítulo de análisis. La interfaz es responsive, por lo que se adapta al tamaño del dispositivo utilizado.

DISPOSITIVOS DE ESCRITORIO

LANDING PAGE

Esta vista pretende informar al usuario de las funcionalidades de la aplicación y servir como página inicial a la espera de que el usuario inicie sesión o se registre, según el caso.

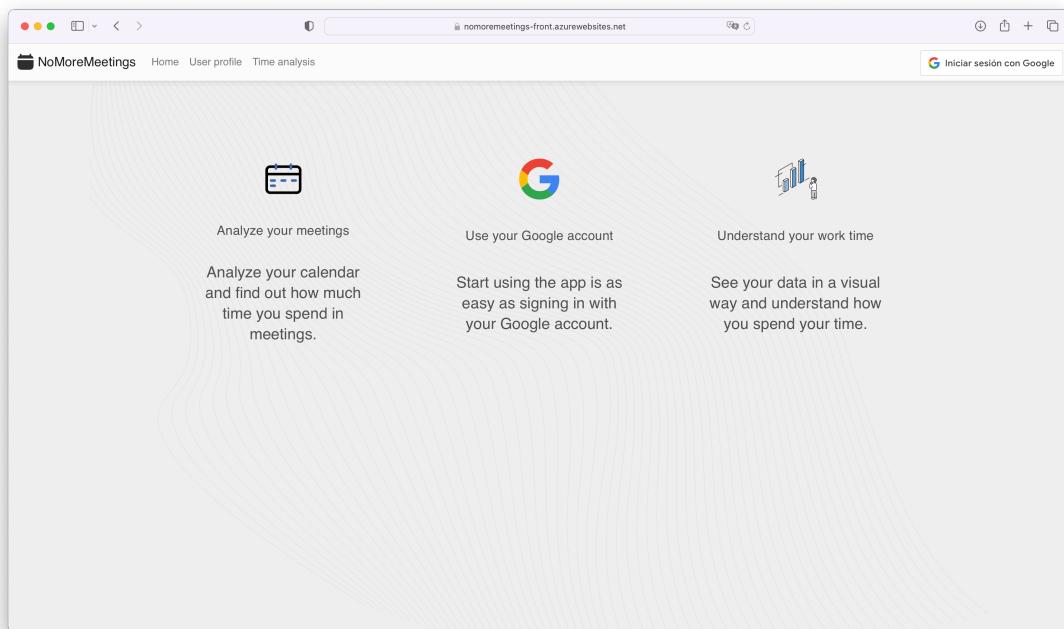


Figura <TBD>: Landing page de NoMoreMeetings vista en un PC

USER PROFILE

Esta vista muestra la información del usuario como nombre, apellidos, horario de trabajo y foto de perfil. Además, muestra configuraciones como el calendario que se usará para realizar los cálculos, el número de días a analizar y el tiempo entre meetings que se tomara como parte de la última meeting al realizar el análisis.

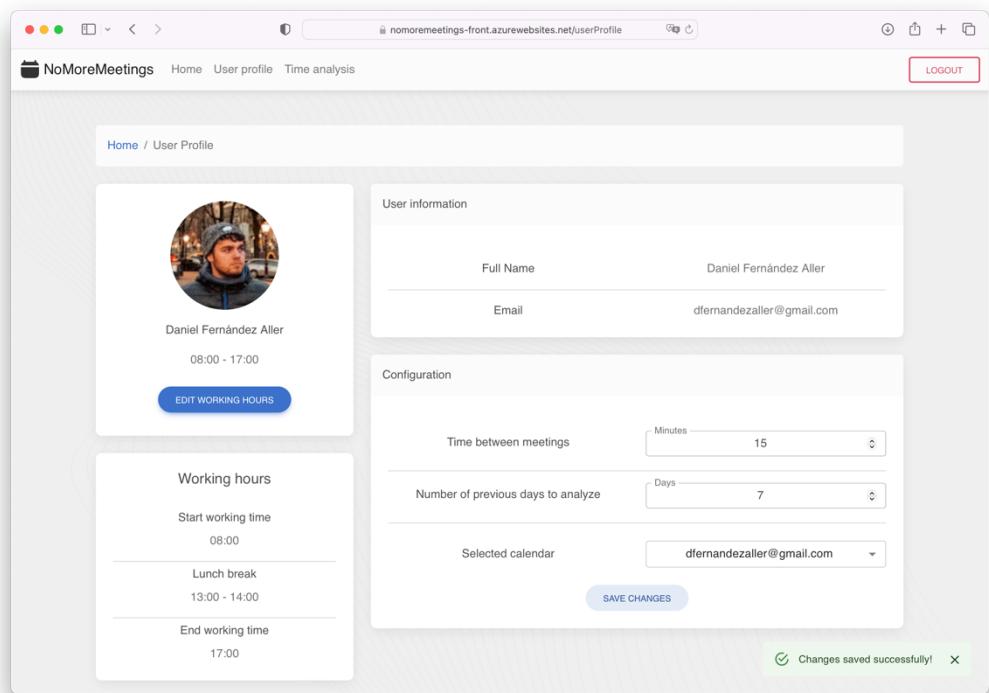


Figura <TBD>: Página del perfil del usuario vista en un PC

TIME ANALYSIS

Esta vista muestra el resultado del análisis de los datos del calendario seleccionado del usuario en base a su configuración.

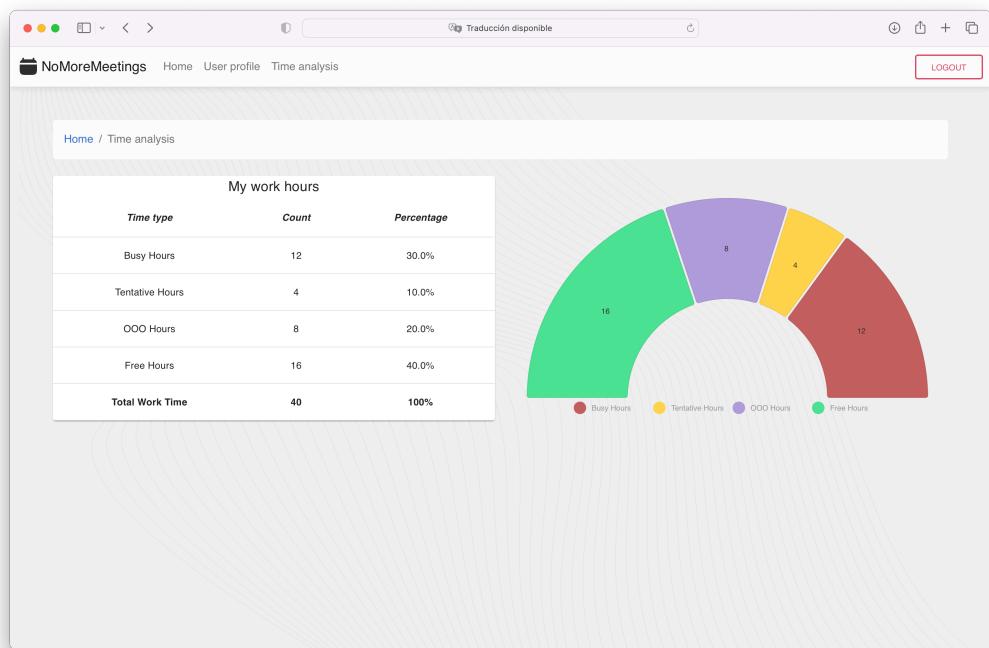


Figura <TBD>: Página de análisis de tiempo del calendario del usuario

DISPOSITIVOS MÓVILES

Veamos ahora la interfaz de usuario en dispositivos móviles.

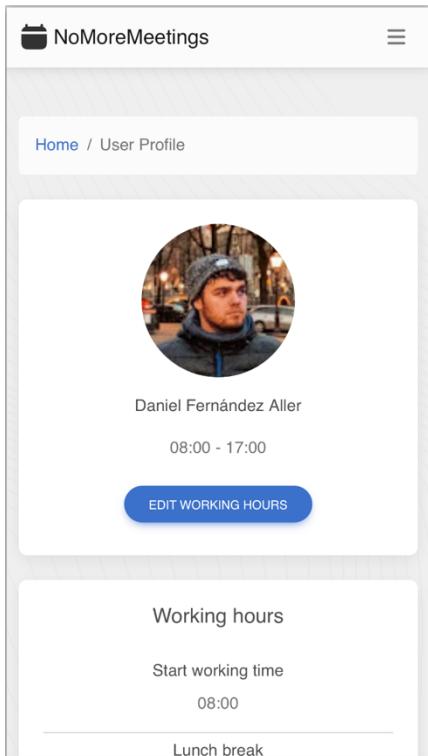


Figura <TBD>: User profile en iPhone

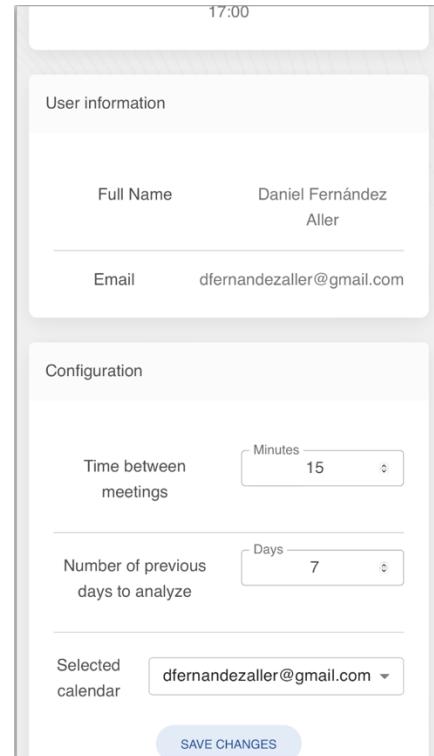


Figura <TBD>: User profile en iPhone

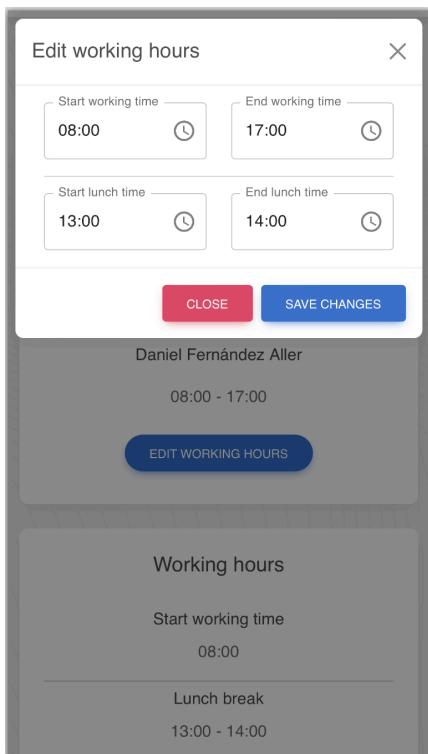


Figura <TBD>: Modal en iPhone

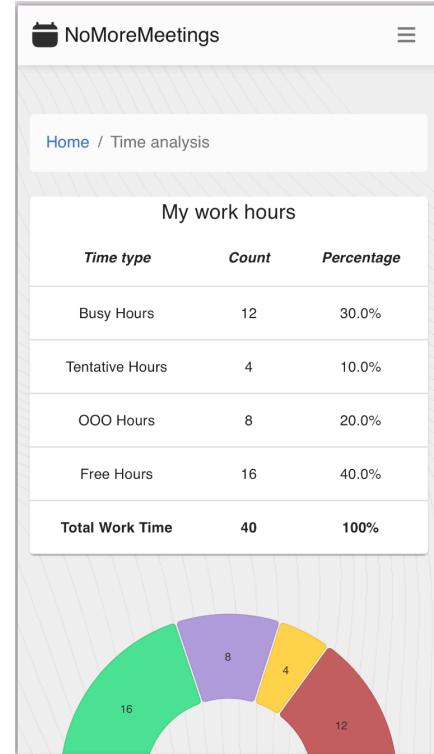


Figura <TBD>: Time analysis en iPhone

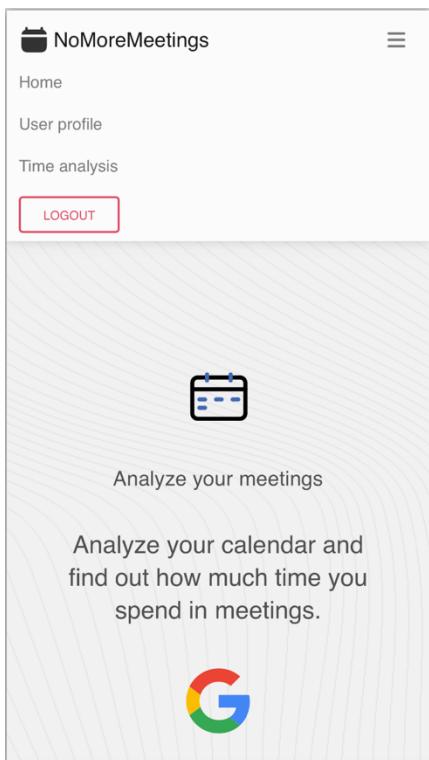


Figura <TBD>: Landing page en iPhone con barra de navegación desplegada

IMPLEMENTACIÓN

MANUAL DEL DESARROLLADOR

PATRONES ARQUITECTÓNICOS APLICADOS

La aplicación sigue una combinación de varias arquitecturas:

ARQUITECTURA DE APLICACIONES DE PÁGINA ÚNICA (SPA)

El frontend de la aplicación se ha desarrollado con React, siguiendo el patrón de Componentes Compuestos. En esta arquitectura, los componentes más grandes se componen de varios componentes más pequeños, creando una estructura de árbol.

ARQUITECTURA ORIENTADA AL DOMINIO (DDD)

Para el backend, hemos seguido la arquitectura DDD. El código se organiza en distintos paquetes para separar la lógica de negocio, los datos y la infraestructura.

INYECCIÓN DE DEPENDENCIAS

Hemos utilizado la Inyección de Dependencias, proporcionada por Micronaut, en todo el backend. Esta técnica permite una mayor flexibilidad, simplifica el código y facilita las pruebas unitarias.

PATRONES DE DISEÑO UTILIZADOS

Hemos utilizado varios patrones de diseño en la aplicación, estos han sido detallados en el capítulo de Diseño, bajo la sección de Patrones de diseño aplicados.

A continuación se listan los patrones de manera resumida:

- **Repository:** Este patrón nos permite encapsular la lógica de acceso a los datos, proporcionando un acceso uniforme a diferentes fuentes de datos.
- **Factory Method:** Este patrón se usa para manejar la creación de objetos, permitiendo que las subclases decidan qué clase instanciar.
- **Singleton:** Usamos este patrón para asegurarnos de que una clase tenga solo una instancia y proporcionar un punto de acceso global a ella.
- **Data Transfer Object (DTO):** Este patrón se usa para transferir datos entre procesos de software, en nuestro caso, entre la interfaz de usuario y los servidores.
- **Builder:** Este patrón nos permite construir objetos complejos paso a paso.
- **Adapter:** Este patrón nos permite convertir la interfaz de una clase en otra interfaz que los clientes esperan.
- **Strategy:** Este patrón se usa para seleccionar un algoritmo en tiempo de ejecución.

LENGUAJES UTILIZADOS

Las tecnologías y lenguajes que se han utilizado para desarrollar esta aplicación son:

BACKEND

- Java 17 LTS
- Micronaut 3.7.0
- PostgreSQL v15
- log4j 2.19
- JUnit 5 con AssertJ y Mockito.
- Liquibase 4

FRONTEND

- JavaScript ES2020
- React 18.2
- Axios 1.3
- dayjs 1.11
- React Oauth Google library 0.7
- Mui/material 5
- Mdb React 6.
- Entorno de ejecución y gestión de paquetes
 - Node 18
 - Npm 9.5

Además, es necesario tener instalado:

- Docker 24.0
- Git 2.39
- Gradle 7.5

PREPARACIÓN DEL ENTORNO DE DESARROLLO

Para preparar tu entorno de desarrollo se necesita:

- **Docker**: Se usa para levantar una instancia de PostgreSQL. Hay un archivo docker-compose en la carpeta resources del backend que facilita este paso.
- **Java 17 LTS**: Para el desarrollo del backend es necesario tener instalado un JDK de Java 17.
- **Node 18 y Npm 9.5**: Para el desarrollo del frontend y la gestión de las dependencias.
- **Gradle**: Para la gestión de las dependencias del backend.
- **IntelliJ IDEA Ultimate**: Como editor de código. Aunque puedes utilizar cualquier editor de tu preferencia, te recomendamos IDEA por su integración con Gradle y su soporte para el desarrollo con React. Como alternativa se puede usar Eclipse para el backend y VS Code para el frontend.
- **IntelliJ DataGrip**: Para la gestión de la base de datos.
- **Git 2.39**: Como herramienta de gestión de versiones.

VARIABLES DE ENTORNO

Es necesario configurar una serie de variables de entorno tanto en el backend como en el frontend. Para ejecutar el backend en local no será necesario nada más que especificar que se ha de cargar el fichero de configuración application-local.yml, que está preparado para comunicarse con la instancia de PostgreSQL inicializada por el fichero docker-compose.

Para producción, se han de especificar las siguientes variables de entorno en el caso del backend:

- **GOOGLE_CLIENT_ID**: Aquí deberá ir el client ID de Google Cloud que se usará en la comunicación con las APIs de Google.
- **GOOGLE_CLIENT_SECRET**: Secreto de Google Cloud que será usado para comunicarse con las APIs de Google.
- **DB_URL**: URL de conexión de tipo JDBC para conectarse a la base de datos.
- **DB_USERNAME**: Nombre de usuario que se usará en la conexión a base de datos.
- **DB_PASSWORD**: Contraseña del usuario que se usará en la conexión a base de datos.

En el caso del frontend, tanto en local como en producción, hay que especificar las siguientes variables de entorno:

- REACT_APP_BACKEND_HOST: URL donde se aloja el backend en el formato {HTTP/HTTPS}://{URL}:{PORT}

MANUAL DE DESPLIEGUE

Este manual proporciona una guía paso a paso para desplegar la aplicación en un entorno de producción.

REQUISITOS

Servidor con Java 17 LTS instalado.

Base de datos PostgreSQL v15.

Docker instalado en el servidor.

Node.js v18 y Npm 9.5 instalados en el servidor.

PASOS PARA EL DESPLIEGUE

1. **Clonar el repositorio del proyecto:** En el servidor de producción, clona el repositorio del proyecto utilizando el comando git clone.
2. **Configurar variables de entorno:** Configura las variables de entorno necesarias para la aplicación, listadas en el manual de desarrollador.
3. **Construir y ejecutar la imagen de Docker para la base de datos PostgreSQL:** Utiliza el archivo docker-compose que se encuentra en la carpeta del backend para levantar una instancia de PostgreSQL. Puedes hacer esto con el comando docker-compose up.
4. **Instalar dependencias del backend:** Navega a la carpeta del backend y ejecuta ./gradlew build para instalar las dependencias necesarias, pasar los tests y construir el proyecto.
5. **Iniciar la aplicación backend:** Una vez construido el proyecto, se puedes iniciar la aplicación backend con el comando:
 - a. java -jar build/classes/java/main/com/dfernandezaller/Application.class.
6. **Instalar dependencias del frontend:** Navega a la carpeta del frontend y ejecuta npm install para instalar las dependencias del proyecto.
7. **Construir el proyecto del frontend:** Ejecuta el comando npm run build. Esto creará una versión de producción de la aplicación frontend en la carpeta build.
8. **Desplegar la aplicación frontend:** Según el servidor donde se desplegará la aplicación, los pasos pueden variar. En general, deberás copiar los contenidos de la carpeta build en la carpeta correspondiente del servidor, generalmente '/public'.

VERIFICACIÓN DEL DESPLIEGUE

Para verificar que la aplicación se ha desplegado correctamente, se ha de:

1. Navegar a la URL de tu aplicación frontend y comprobar que la interfaz de usuario se carga correctamente.
2. Realizar pruebas de conexión a la base de datos y comprobar que las tablas se han creado de manera adecuada.
3. Conectarse a los servicios del backend para asegurarte de que todo funciona como se espera. Si se va a la URL del backend debería aparecer un mensaje notificando que el servidor está listo para recibir peticiones.

MANUAL DE USUARIO

INTRODUCCIÓN

Este manual tiene como objetivo guiar a los usuarios a través de las funcionalidades y características de NoMoreMeetings. Aquí encontrarás instrucciones paso a paso sobre cómo utilizar cada característica, junto con capturas de pantalla para ilustrar cada paso.

REGISTRO Y ACCESO

Cuando accedas a la aplicación, te encontrarás con la página de inicio. Aquí podrás iniciar sesión si ya tienes una cuenta, o registrarte si eres un nuevo usuario.

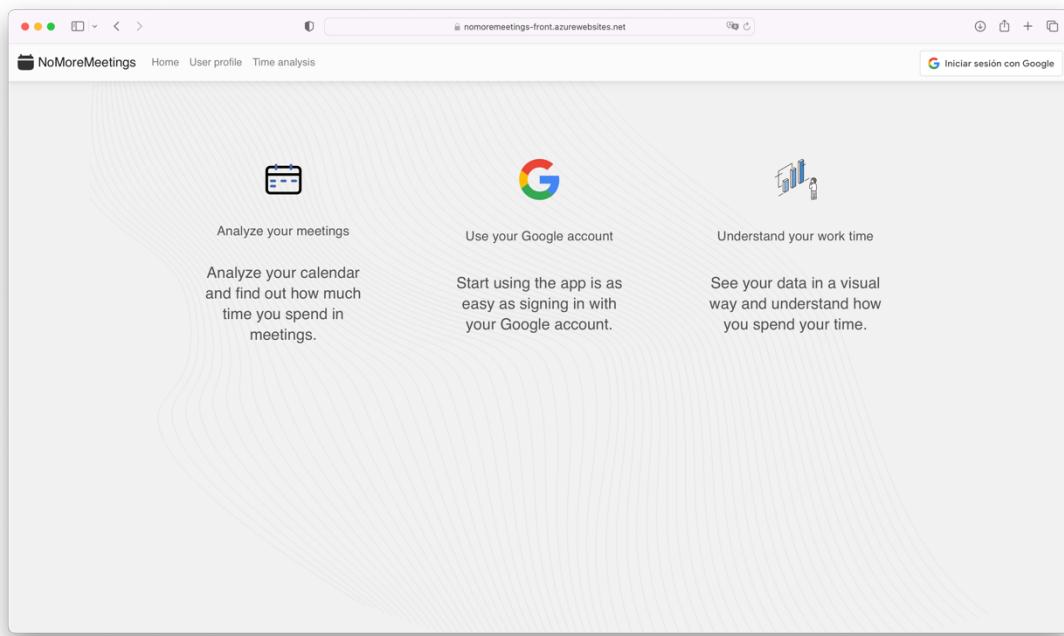


Figura <TBD>: Página de inicio de NoMoreMeetings

INICIO DE SESIÓN

Para proceder al inicio de sesión, se ha de hacer clic en el botón superior izquierdo que dice: Iniciar sesión con Google. Esto te lleva a una página de inicio de sesión de Google donde deberás seleccionar que cuenta quieras usar, y deberás iniciar sesión. Es posible que aparezca un cuadro de selección.

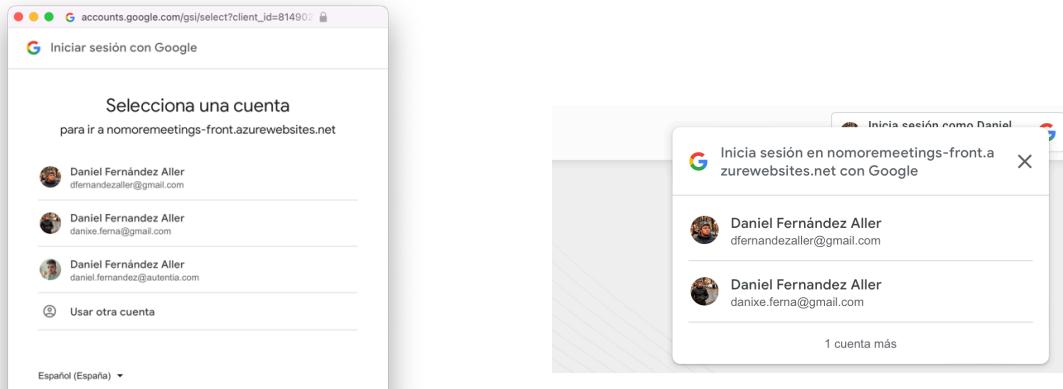


Figura <TBD>: Cuadro de selección de cuenta para iniciar sesión

Figura <TBD>: Página de selección de cuenta

REGISTRO

Para proceder al registro se ha de seguir los mismos pasos que para el inicio de sesión. Una vez que se hayan completado el usuario verá que una alerta informativa al final de la página informa de la necesidad de registro y el botón de inicio de sesión ha cambiado a un botón de registro. Es posible que, en función de la configuración del navegador del usuario, aparezca tras el inicio de sesión directamente una venta de Google para proceder con el registro.

Una vez aparezca la ventana de registro de Google, bien por presionar el botón de registro o bien porque aparezca de forma automática, se ha de dar los permisos necesarios a NoMoreMeetings para acceder al calendario del usuario y ya se habrá completado el registro.

Es posible que aparezca una alerta informando de que la aplicación no está verificada, en ese caso se ha de presionar en ‘Mostrar configuración avanzada’ y acto seguido en ‘Ir a nomoremeetings...’.

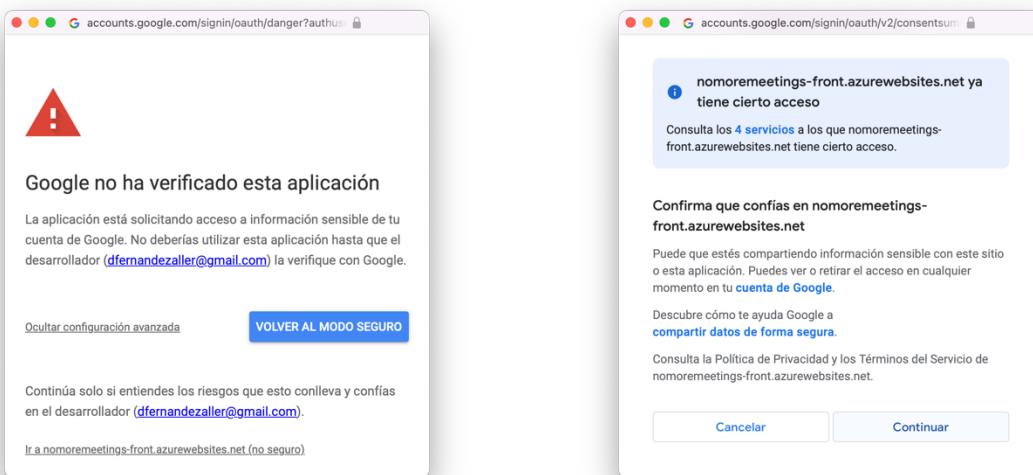


Figura <TBD>: Página de alerta de Google de aplicación no verificada

Figura <TBD>: Página de concesión de permisos de Google

PERFIL DEL USUARIO

En esta página el usuario puede ver toda la información relativa a su perfil de usuario en la aplicación así como modificar la configuración del análisis del calendario. Para modificar la configuración relativa al análisis se dispone de tres cuadros donde se puede:

- Seleccionar el tiempo en minutos entre reuniones que contara como tiempo de reunión.
- Seleccionar el número de días previos al actual a analizar.
- Seleccionar el calendario del usuario que se analizará.

Además de estas opciones hay un botón dedicado a poder cambiar el horario laboral del usuario así como su horario de almuerzo. Es durante el periodo de trabajo que se analizara el tiempo dedicado a reuniones del usuario. Al pulsar el botón azul ‘Edit working hours’ se mostrará una ventana modal que permitirá editar el horario laboral del usuario.

Cualquier cambio en la configuración será guardado cuando se presione ‘Save changes’.

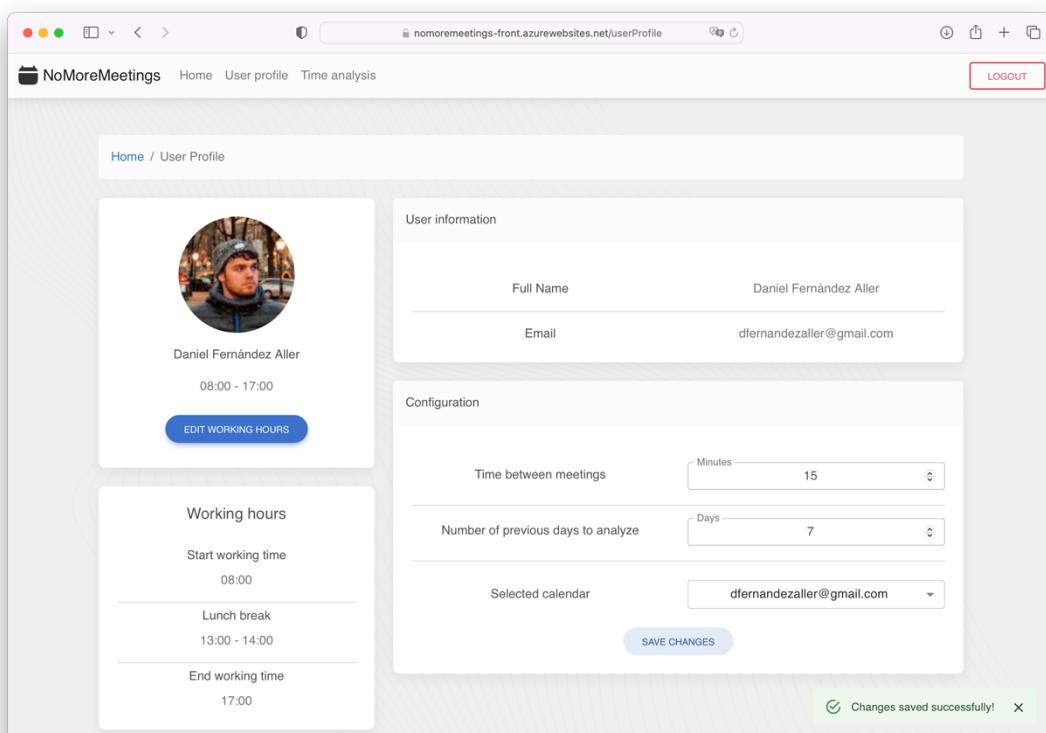


Figura <TBD>: Página de perfil del usuario y configuración

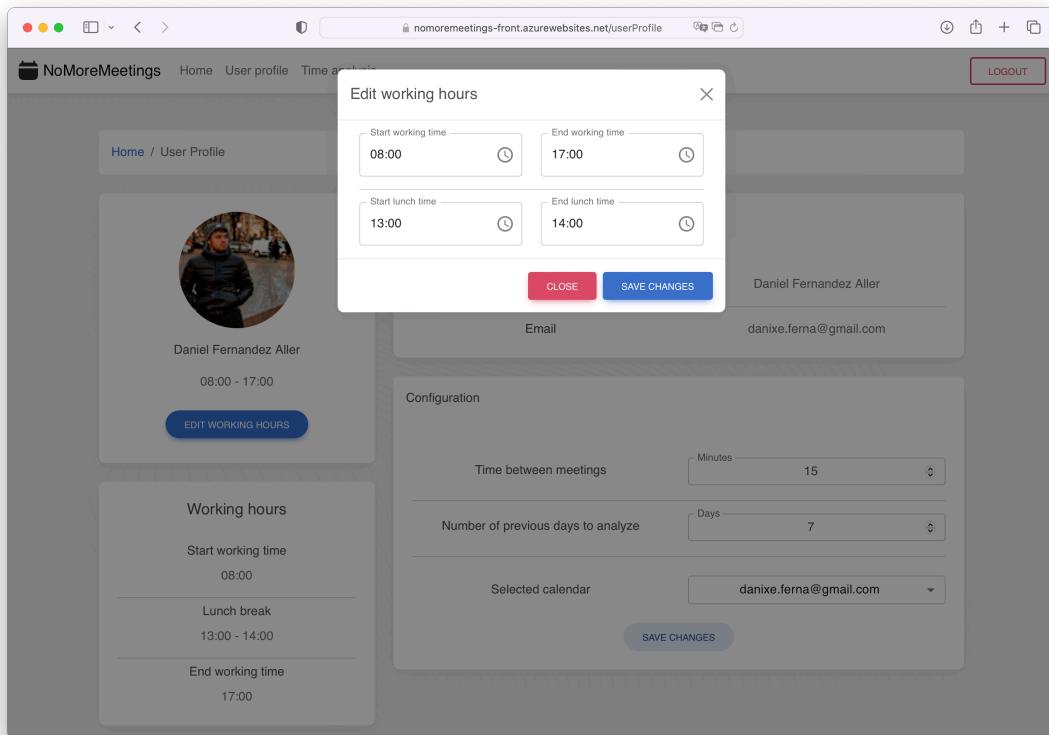


Figura <TBD>: Cuadro modal de modificación del horario laboral

ANÁLISIS DEL TIEMPO

En esta página se pueden visualizar, en un formato de tabla y en una gráfica semi-circular, los datos relativos al tiempo dedicado a las reuniones por el usuario de acuerdo con la configuración establecida en la vista anterior.

El usuario no ha de hacer nada para poder visualizar los datos, es posible que aparezca una pantalla de carga si los datos tomaran mucho tiempo para ser calculados, pero en cualquier caso no se requiere de interacción del usuario.

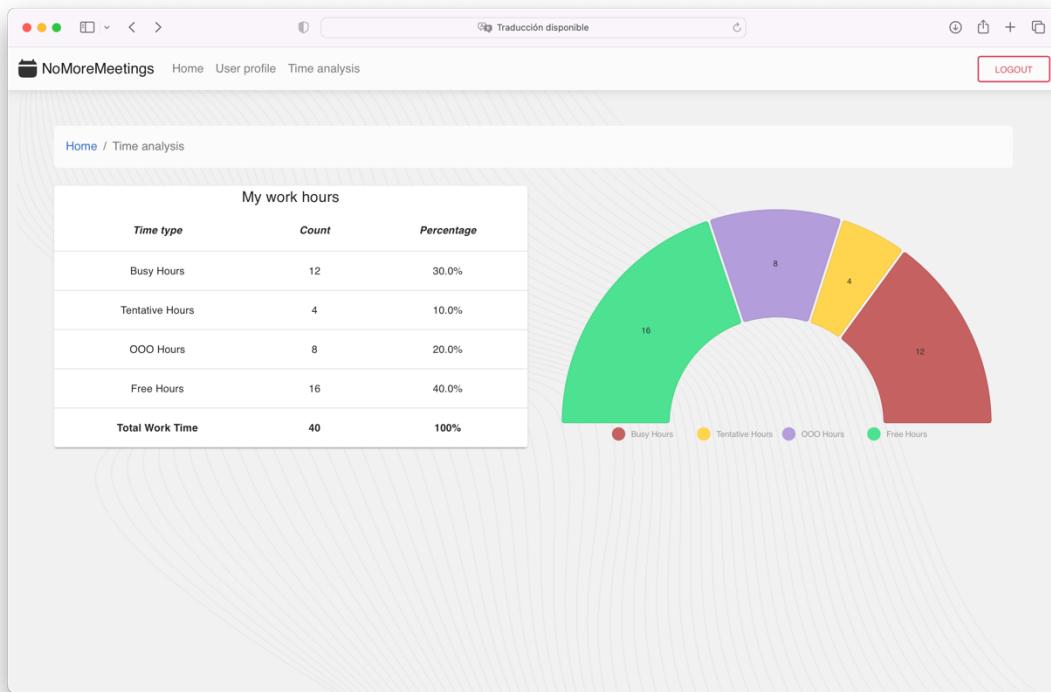


Figura <TBD>: Página de resultados del análisis del calendario del usuario

PROBLEMAS ENCONTRADOS Y SOLUCIONES

Durante el desarrollo de este proyecto, se han encontrado varios problemas. Este capítulo proporciona una visión general de estos problemas y cómo se resolvieron.

PROBLEMAS DE COMPILACIÓN RELACIONADOS CON ONEDRIVE

Para facilitar el trabajo entre varios dispositivos y para mantener una copia de seguridad adicional, al principio la carpeta padre del proyecto se encontraba en OneDrive. Sin embargo, el nombre de algunos ficheros generados durante la compilación que contenían caracteres extraños como ‘\$’ o nombres demasiado largos provocaban que los cambios no se escribieran en el disco porque OneDrive está integrado como parte de Windows, y errores de sincronización de OneDrive pueden provocar problemas a nivel del manejo por parte del sistema operativo de esos ficheros.

No era algo esperado y los problemas eran extraños por lo que a pesar de ser algo aparentemente menor encontrar la causa del problema fue bastante complicado.

Solución: Mover la carpeta del proyecto fuera de OneDrive permitió resolver el problema y la compilación de cambios pudo continuar como se esperaba.

CAMBIOS EN LA API DE GOOGLE

Al comienzo del proyecto se intentó realizar el flujo de autenticación con Google de una forma distinta, consistía en utilizar un flujo propio de Google para el que había muchas librerías ya adaptadas y una gran cantidad de documentación, sin embargo esa forma de autenticarse contra los servicios de Google quedaba obsoleta a partir de marzo de 2023, por lo que cuando fui consciente de esto, a mitad del desarrollo de la autenticación usando el sistema antiguo tuve que cambiar al flujo nuevo basado en OAuth.

Esto además supuso un problema debido a que la propia documentación de Google seguía haciendo referencia a la forma antigua en algunos documentos y a la nueva en otros. Y la documentación de la comunidad y librerías era escasa, debido a que el cambio se había anunciado y la nueva metodología ya estaba disponible pero no era obligatorio el cambio hasta marzo.

Solución: Se realizó una investigación de como funciona OAuth2 y de la nueva forma de trabajar de Google, en algunos casos adaptando ejemplos de la documentación oficial pensados para el flujo antiguo al flujo nuevo.

IMPOSIBILIDAD DE USAR EL FLUJO DE MICRONAUT PARA OAUTH

Relacionado con el problema anterior a la hora de implementar el nuevo flujo había que pedir dos tipos de token, uno genérico que contiene información básica del perfil del usuario y otro que permite obtener a su vez otra serie de tokens que permiten hacer llamadas a la API de Google Calendar y en el que a la hora de pedirlo específicas los permisos que vas a requerir.

Este segundo tipo de token requiere que sepas el código OAuth de los permisos que requieras y ha de ser pedido por el frontend, para obtener un idToken que luego el backend pueda intercambiar por los tokens de acceso y refresco. Este proceso en Micronaut en teoría puede ser automatizado de forma relativamente sencilla, pero la necesidad de usar la librería de Google para Java hacía imposible utilizar el flujo estandarizado de Micronaut ya que es la librería de Google la que gestiona los tokens en base de datos.

Solución: Se decidió hacer una implementación manual del flujo OAuth en el backend usando la interfaz AuthenticationProvider de Micronaut para interceptar las peticiones a endpoints securizados y usar la librería de Google de Java dentro de ese “interceptor” para validar y refrescar los tokens necesarios.

FALTA DE DOCUMENTACIÓN OFICIAL PARA LA OBTENCIÓN DE TOKENS EN EL BACKEND

Si bien ya se ha mencionado que existe una librería de Google para Java que facilita la gestión de estos tokens en el backend lo cierto es que esta librería no dispone de una documentación adecuada. En el momento del desarrollo no existían ejemplos oficiales de ningún proceso. Por ejemplo, para poder obtener tokens de acceso y refresco en OAuth hay que usar un proceso denominado explícito, distinto del implícito que es un procedimiento simplificado para aplicaciones que no persisten datos, este procedimiento explícito requiere que se le pase en la llamada del backend para intercambiar el idToken por los tokens de acceso y refresco una uri a la que la API de Google responderá.

En la documentación explicativa de Google no se menciona la posibilidad de pasar el valor “offline” como accessType que permite obtener los tokens sin dar una uri de redirección como contestación a la propia petición de la librería. Sí está documentado en el código.

Solución: Estudiar el funcionamiento genérico de OAuth 2.0 e inferir el funcionamiento del flujo dentro de la librería de Google de Java y de la de Javascript con la ayuda de los comentarios disponibles en el código para poder avanzar con el desarrollo.

CORS

Tuve problemas relacionados con la configuración de Cross-Origin Resource Sharing (CORS) entre el backend y el frontend.

Solución: Estos problemas se resolvieron configurando correctamente el CORS en el backend del servidor y utilizando la librería Axios en el frontend para realizar las llamadas al backend en lugar de la librería nativa de JavaScript (fetch).

REACTIVE STREAMS Y LA BIBLIOTECA DE GOOGLE

En un primer momento estaba planificado el uso de programación reactiva en el backend, pero la biblioteca de Google no está diseñada de forma reactiva ni lo soporta de forma nativa, por lo que la finalidad de usar programación reactiva que era hacer un código más compacto pero legible no iba a poder conseguirse porque el código se complicaba demasiado al tener que hacer envoltorios reactivos para la librería de Google.

Solución: Se desecharon las ideas porque consumía demasiado tiempo y empeoraba la legibilidad del código.

PROBLEMAS CON WSL

Existen problemas con Windows Subsystem for Linux (WSL) y su comportamiento después de la hibernación. Estos problemas impiden utilizarlo con normalidad cuando el equipo sale de estado de hibernación, impidiendo el uso de Docker basado en WSL.

Solución: A través de la investigación en línea, se descubrió que este es un problema conocido con WSL y se mitigó evitando poner el sistema en estado de hibernación.

PROBLEMAS ENTRE AZURE, GRADLE Y MICRONAUT

La combinación de Azure App Services con Gradle y Micronaut no es nada común. Existe gran cantidad de documentación para el caso de uso de aplicaciones desarrolladas con Spring Boot y Maven que son desplegadas en Azure pero no así para Gradle y Micronaut.

Micronaut utiliza el plugin shadowJar de Gradle para crear el fat jar de la aplicación aunque en un primer momento cuando se observa la configuración parece que se utiliza el plugin de Java para eso. Por defecto este plugin nombra el jar con una combinación de nombre de aplicación y versión, pero Azure solo reconoce los .jar que se nombran como app.jar por lo que el despliegue fallaba porque Azure no era capaz de encontrar el .jar de la aplicación.

Solución: Tras investigar las causas encontré un post de un desarrollador que había tenido este mismo problema en 2019 y después de leerlo configuré adecuadamente el plugin shadowJar para que genere un fichero app.jar allí donde Azure espera encontrarlo.

LEVANTAMIENTO INEFICIENTE DE CONTENEDORES DURANTE LAS PRUEBAS

Una característica importante que me motivo a escoger Micronaut como framework para el backend fue que la rapidez con la que inicia permite hacer tests de integración con la aplicación al completo levantada sin que eso consuma mucho tiempo. Como parte de la batería de pruebas hay tests de integración que acceden a una base de datos que se levanta de manera automática gracias a TestContainers.

El problema que enfrenté es que se levantaba una base de datos por cada clase de prueba, y aunque no era un problema grave ya que se levantan bastante rápido en mi equipo pueden no hacerlo en otro y además no es algo que me gustara porque era poco eficiente y consumía tiempo de CI/CD.

Solución: Decidí usar la posibilidad que ofrece Junit de establecer métodos que se ejecuten antes y después de que comience la batería de tests mediante la interfaz *TestExecutionListener*, además mediante la interfaz *PropertySourceLoader* de Micronaut puedo injectar directamente las propiedades de la base de datos levantada en la configuración de Micronaut.

Para poder ejecutar todo esto antes de la carga de Micronaut y Junit uso el Service Locator de Java que me permite instanciar la clase necesaria para levantar Docker, marcada además con la palabra reservada ‘volatile’ para asegurar que las dos instancias que se crean (una para micronaut y otra para Junit) puedan acceder de forma sincronizada a el estado de la clase.

CONCLUSIONES

A lo largo del proyecto se ha realizado un ejercicio no solo de desarrollo si no de planificación, investigación de tecnologías nuevas o poco exploradas por mí como los framework Micronaut y React, la autenticación con servicios de terceros o el montaje de un proceso de integración y despliegue continuo en la nube. Además, se han mantenido reuniones periódicas y se ha escuchado feedback de manera regular por parte de un stakeholder, el representante de HP que ha propuesto este TFG.

Todo este ejercicio de trabajo e investigación ha derivado en un sistema robusto, modular, preparado para la ampliación con nuevos proveedores de calendario y que ha satisfecho las expectativas de los stakeholders. Un sistema compuesto por un backend escrito en Java con Micronaut, un frontend escrito en JavaScript y React y un sistema de base de datos en la nube con un sistema de integración, testeo y despliegue automatizado.

A modo de reflexión me gustaría destacar que por el camino me ha permitido enfrentarme a un proyecto de manera autónoma y aprender sobre nuevas tecnologías y formas de trabajo que me generaban

curiosidad como es el uso de recursos en la nube, el CI/CD o el uso de Docker para levantar contendores de prueba de manera automática.

TRABAJO FUTURO

A lo largo del desarrollo de NoMoreMeetings se han identificado varias áreas de mejora y características adicionales que podrían implementarse en el futuro para mejorar aún más la eficacia y la utilidad de la aplicación. Algunas de ellas como la conectividad con Exchange ya se han previsto en la fase de diseño y han influido en la toma de decisiones como ya se ha explicado. A continuación, se detallan algunas de las posibles características que podrían desarrollarse en futuras versiones de la aplicación:

REUNIONES REPETITIVAS

Una funcionalidad importante sería que la aplicación pudiera identificar y tener en cuenta las reuniones que se producen con asiduidad. Esta característica aumentaría la precisión del análisis del tiempo de reunión y proporcionaría una representación más realista de la forma en que los usuarios gastan su tiempo ya que permitiría distinguir estas reuniones en la página de resultados de análisis.

CONEXIÓN CON EXCHANGE

Actualmente, NoMoreMeetings es compatible solo con Google Calendar. Sin embargo, Microsoft Exchange también es ampliamente utilizado en el entorno empresarial. Por lo tanto, sería beneficioso para muchos usuarios si la aplicación pudiera conectarse con Exchange.

EXPORTACIÓN DE INFORMES

Por último, aunque la aplicación proporciona un análisis detallado del tiempo de reunión, sería útil poder exportar estos informes en un formato normalizado, como CSV o JSON. Esto permitiría a los usuarios compartir fácilmente sus informes con otros o utilizar los datos en otras aplicaciones o herramientas.

VER INFORMES ANTERIORES

Otra posible ampliación es la capacidad de acceder y visualizar informes de análisis de tiempo de reunión generados previamente. En la versión actual de NoMoreMeetings, los usuarios solo pueden ver el informe más reciente generado. Sin embargo, el acceso a informes anteriores permitiría a los usuarios hacer un seguimiento de su tiempo de reunión a lo largo del tiempo y detectar patrones o tendencias.

Esta característica implicaría almacenar los informes generados de forma segura y proporcionar una interfaz de usuario para que los usuarios puedan acceder a ellos. La implementación de esta funcionalidad permitiría a los usuarios no solo tener una instantánea de su tiempo de reunión actual, sino también una visión más amplia y una perspectiva a lo largo del tiempo.