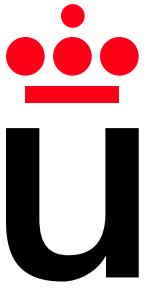


Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

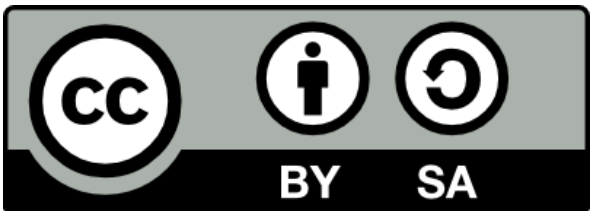
Grado en Ingeniería de Computadores

Curso 2022 2023

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN DE PETICIÓN DE CITAS CON AWS LAMBDA

Autor: Álvaro García Sierra
Tutor: Francisco Gortázar Bellas



Agradecimientos

Gracias a mis padres, por apoyarme en todo momento y creer siempre en mí, incluso cuando yo no lo hacía.

Gracias a mis amigos de Toledo, por poder contar siempre con ellos.

Gracias a mis amigos de la Universidad, sin ellos esta etapa habría sido mucho más difícil.

Resumen

Las aplicaciones de planificación y organización son esenciales en la vida cotidiana de las personas. Sin embargo, a pesar de la amplia oferta de aplicaciones disponibles, aún existe una carencia significativa en el mercado: la falta de aplicaciones que permitan la interacción y colaboración entre diferentes usuarios o servicios.

Muchas personas podrían beneficiarse al compartir y enlazar sus calendarios con otras personas, servicios o locales de interés. Esta funcionalidad permitiría una planificación más efectiva y la creación de calendarios interactivos, donde todos los participantes puedan coordinar sus agendas y establecer citas de manera sencilla.

Esta aplicación trata de cambiar la forma en que las personas organizan sus eventos facilitando la colaboración en la planificación de actividades. Esta aplicación tiene como objetivo servir como una herramienta versátil y útil tanto para usuarios individuales como para empresas, permitiendo una mejor gestión del tiempo y una mayor eficiencia en la coordinación de actividades.

Para el desarrollo de esta aplicación, se ha tomado especialmente en cuenta la arquitectura del sistema. Utilizando tecnologías punteras y destacando la computación en la nube, usando uno de los principales proveedores como es AWS.

Este documento está compuesto por cuatro capítulos. En el capítulo uno, llamado "Introducción", se explicará a modo introductorio el origen de la idea que impulsó la realización de este proyecto, así como el estado del arte de las aplicaciones de calendario. A continuación, en el segundo capítulo titulado "Objetivos", se detallará los objetivos a cumplir para el proyecto, sirviendo como una continuación del capítulo anterior. En el capítulo tres, ".Ejemplos de uso", se muestran capturas de la aplicación. Después, en el capítulo cuatro, "Descripción informática", en primer lugar se nombrarán las tecnologías, herramientas y metodología utilizadas. En este capítulo, también se hablará sobre los requisitos funcionales y no funcionales y se entrará en detalle sobre la arquitectura del sistema y su implementación. Por último, el capítulo cinco, "Conclusiones y trabajo futuro", servirá para dar las conclusiones finales sobre el proyecto y se presentarán posibles mejoras para su futuro.

Índice

Índice de figuras

1. Introducción

1.1. Motivación

Uno de los problemas en la sociedad es el tiempo. Muchas personas están ocupadas todo el día, realizando actividades sin descanso y sin parar. Tal es la magnitud de la situación que a todo el mundo se le ha pasado alguna vez un evento importante, como podría ser recoger a sus hijos del colegio.

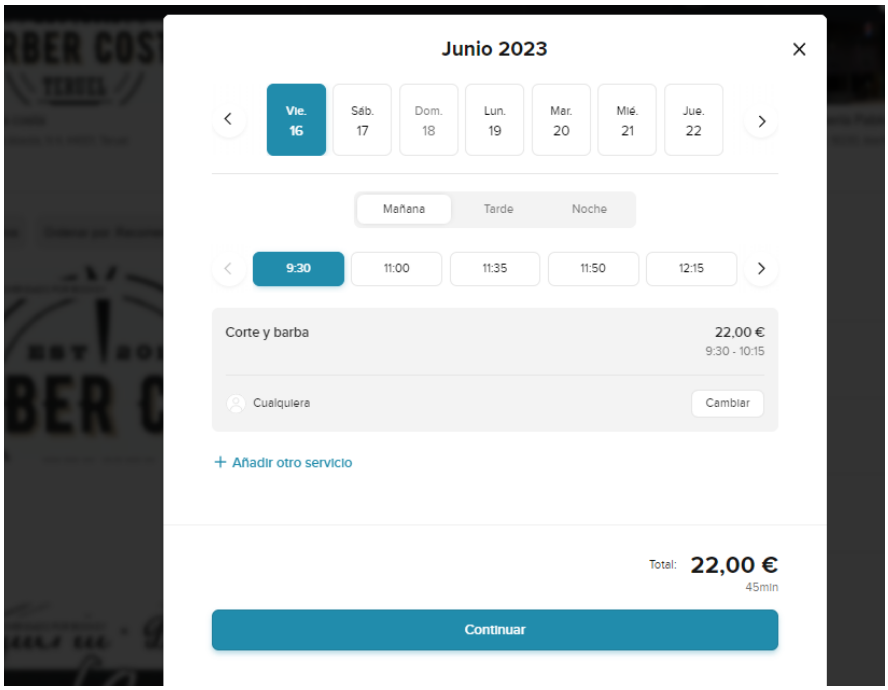
En este aspecto la organización del tiempo es fundamental, permitir y ayudar a las personas y trabajadores a organizarse el tiempo puede resultar en un aumento significativo de su bienestar y eficiencia, permitiendo a su vez tener más tiempo libre.

Hay una gran cantidad de personas que no consiguen administrar su tiempo de la manera más óptima, no porque dependa de ellos mismos, si no porque todos dependemos de los demás. Por ejemplo, una persona puede haberse administrado el día pero si un compañero ajeno a él le cambia una reunión a último momento, puede quedarse un tiempo muerto en el que dicha persona no tenga nada que hacer. Esto puede ser más notable en el ámbito empresarial, concretamente en las empresas que se centren en el sector servicios. Pongamos el caso de un restaurante o una peluquería, si un cliente no se presenta a la cita se genera un espacio de tiempo donde vacío que podría haber sido ocupado por otro cliente. La repetición de clientes que fallen o que anulen su cita con poco margen de tiempo, puede suponer pérdidas importantes de tiempo y de ingresos.

En los últimos años se han desarrollado multitud de aplicaciones para la gestión online, organización y reserva de citas, algunas de estas aplicaciones son Playtomic, Booksy (Ver figura 1), aplicaciones para gimnsasio... Sin embargo estas aplicaciones solo aplican a un campo concreto y no permiten incluir información sobre el ámbito personal del usuario.

Aplicaciones como Google Calendar (Ver figura 2), Calendario Business Agenda o TimeTree sí que ofrecen una manera de organizar el tiempo, sin embargo, estas aplicaciones no permiten la interacción ajena o con servicios de terceros de una forma directa.

Figura 1: Booksy



1.2. Problemática

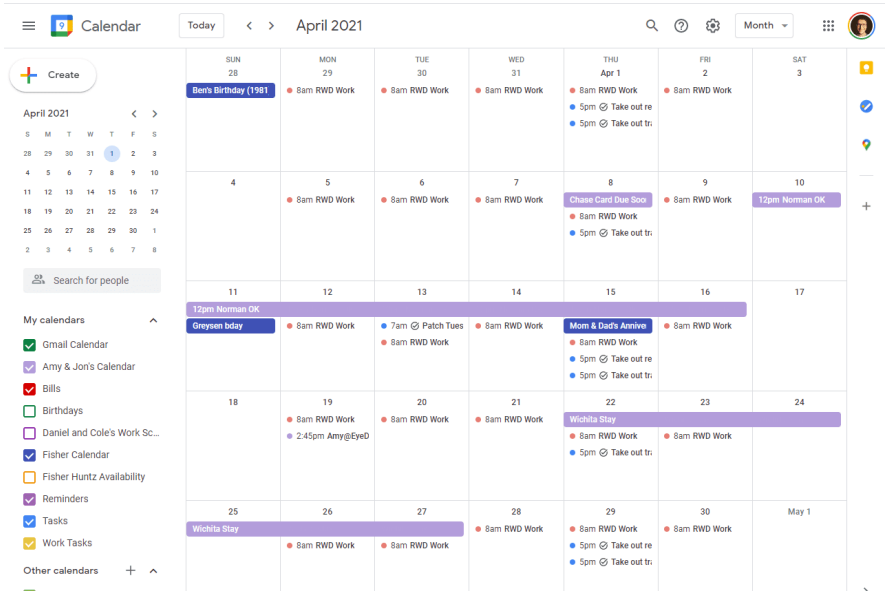
Como ya hemos visto en la sección anterior, existen numerosas aplicaciones para la planificación del tiempo. El uso de este tipo de aplicaciones puede suponer una mejora a corto plazo, pero con solo una de estas no es suficiente para completar tu calendario completamente.

Si bien un usuario podría reservar una cita en un local con Booksy, no podría almacenar sus eventos personales en esta aplicación, y tendría que volver a introducirlo manualmente en otro tipo de calendario que sí permita esta acción, como Google Calendar. Pero, ¿qué pasaría si después de hacer esto una de las dos partes cancelara la cita?

Con el propósito de crear un calendario que pueda abarcar todos los eventos posibles sin tener que usar varias aplicaciones, surge la idea de crear una aplicación que no esté limitada a un campo concreto y que busque la unión entre los calendarios de uso personal, de uso profesional y de uso con terceros.

Es a partir de esta idea, que surge este proyecto, una aplicación que tiene como objetivo abarcar todos estos campos, unificando calendarios, interconectándolos y haciéndolos dinámicos.

Figura 2: Google Calendar



2. Objetivos

Para lograr crear una aplicación competente, se requiere de una gran cantidad de esfuerzo, dedicación y tiempo. Este proyecto tiene como objetivo establecer una base sólida que permita el desarrollo y evolución gradual de la aplicación.

Dado que este proyecto se desarrollará a largo plazo, es esencial definir tanto una arquitectura limpia y escalable, que facilite en gran medida la expansión y el mantenimiento del sistema, como los mecanismos básicos que queremos que nuestro sistema pueda realizar para poder trabajar en ellos en el futuro.

Podemos separar los principales objetivos del proyecto en objetivos técnicos y objetivos utilitarios.

A continuación se destacan los principales objetivos de diseño.

- La aplicación debe estar disponible en la web.
- La aplicación debe permitir la creación de eventos en un calendario.
- Deben existir 2 tipos de usuarios, clientes y locales/servicios.
- La aplicación debe almacenar eventos de los usuarios y enlazarlos entre ellos.
- Los locales deben tener un horario predefinido en base al cual generar posibles citas.
- Los usuarios deben poder solicitar citas a los locales.
- Los locales deben poder aceptar y rechazar dichas citas.
- Usuarios y locales podrán crear sus propios eventos personales.
- Locales podrán crear su propio horario y permitir a varios clientes/usuarios al mismo tiempo.
- Todos los usuarios podrán cambiar su información personal.

Objetivos técnicos.

- Uno de los principales objetivos técnicos es el de explorar las posibilidades y dificultades basándonos en la nube, más concretamente en los servicios de AWS y AWS Lambda y la conexión del frontend con el backend con AWS API Gateway.
- El proyecto ha de usar tecnologías web punteras. El frontend se implementará como una Single Page Application (SPA) en Vue 3. El backend se desarrollará a través de la computación en la nube, usando uno de los principales proveedores como es AWS.
- Debido a que este proyecto necesitará de un gran volumen de usuarios activos para su viabilidad la escalabilidad es un factor a tener en cuenta. Por esto, se tendrá muy en cuenta la arquitectura de la aplicación principalmente en el backend. Debido a esto, se ha decidido hacer uso de una arquitectura *Serverless*, microservicios y bases de datos no relacionales.

- El sistema poseerá una seguridad robusta, implementando mecanismos de autenticación y autorización mediante AWS Cognito y Lambda Authorizers.
- La aplicación tendrá el menor coste posible, utilizando el "Free Tier"[1] de AWS para el desarrollo y procurando el uso más eficiente de recursos.

3. Ejemplos de uso

A continuación se presentan los ejemplos de uso de la aplicación. La aplicación se encuentra operativa, por lo que también se puede probar en la siguiente url: `https://d2rng3ufzekoqa.cloudfront.net/`

Figura 3: Inicio de sesión

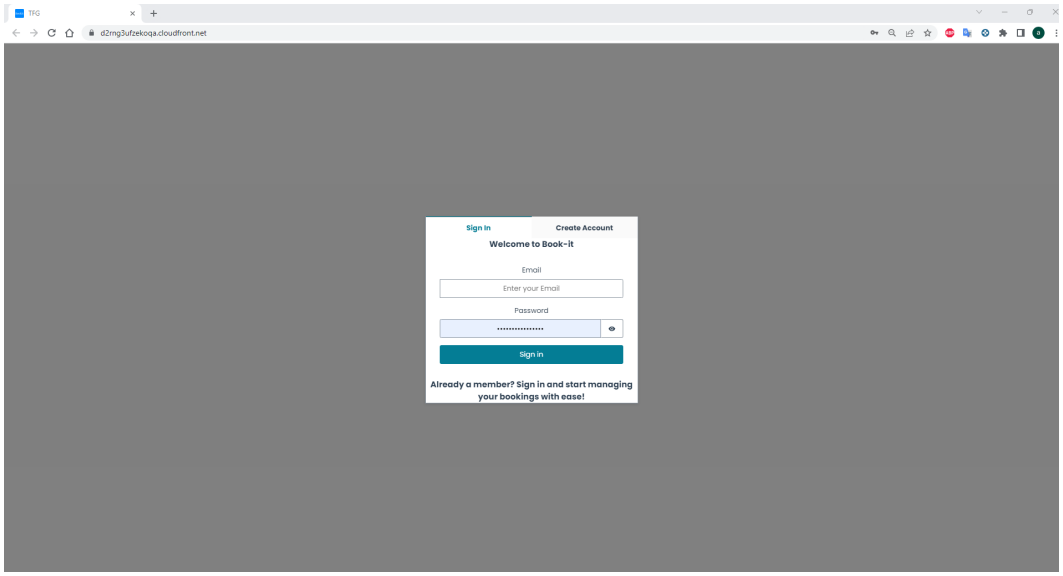


Figura 4: Creación de cuenta

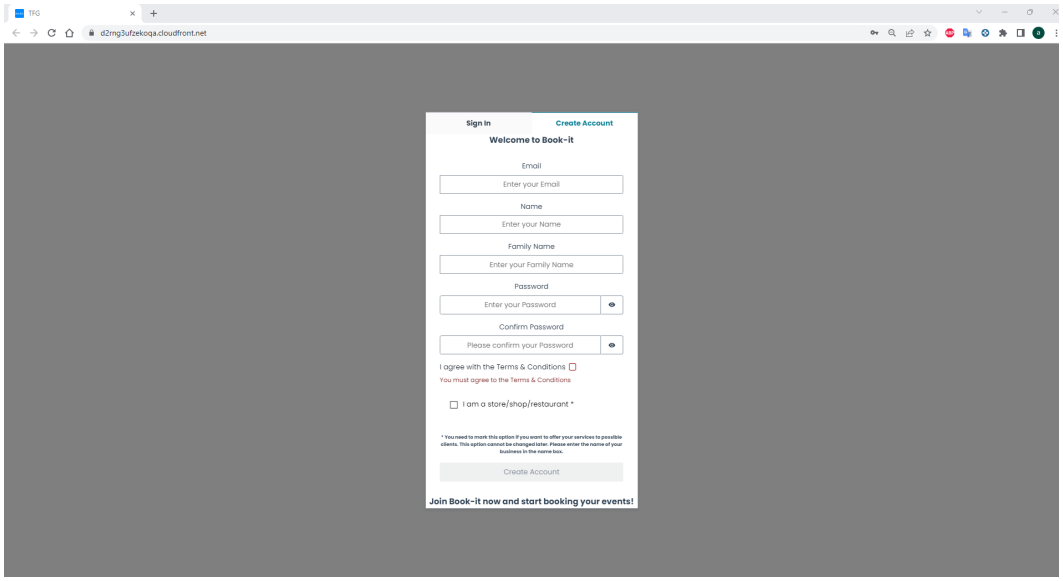


Figura 5: Creación de cuenta usuario local

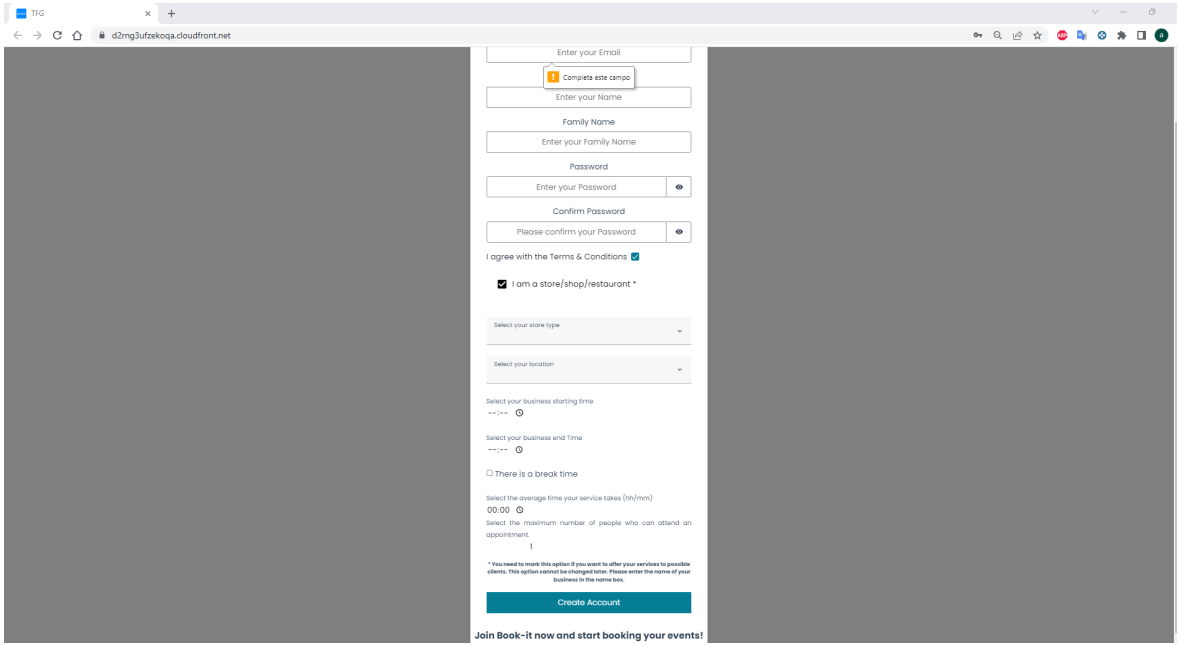


Figura 6: Eventos de un local con barra lateral cerrada y fines de semana desactivados

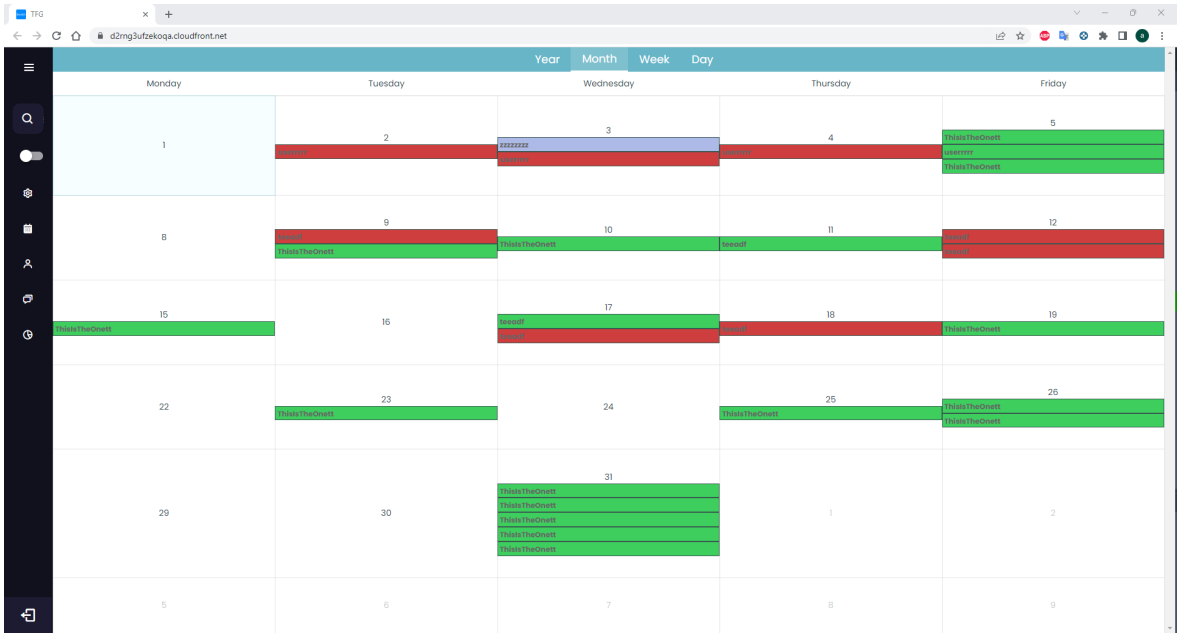


Figura 7: Eventos de un local con barra lateral abierta y fines de semana

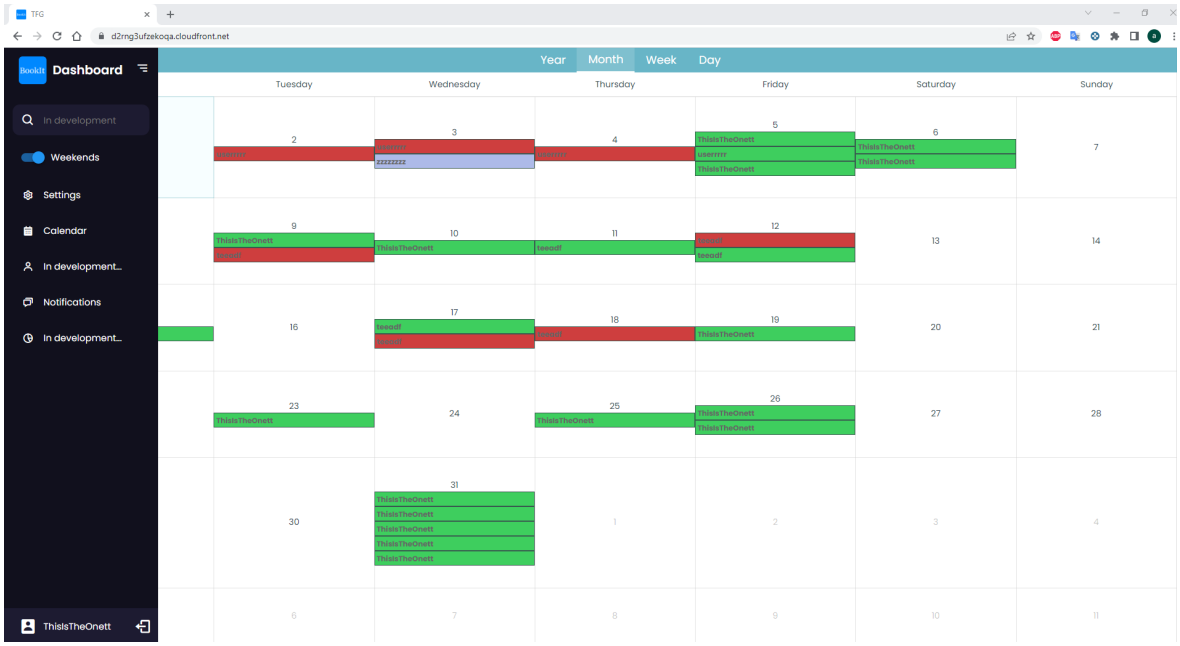


Figura 8: Vista de año

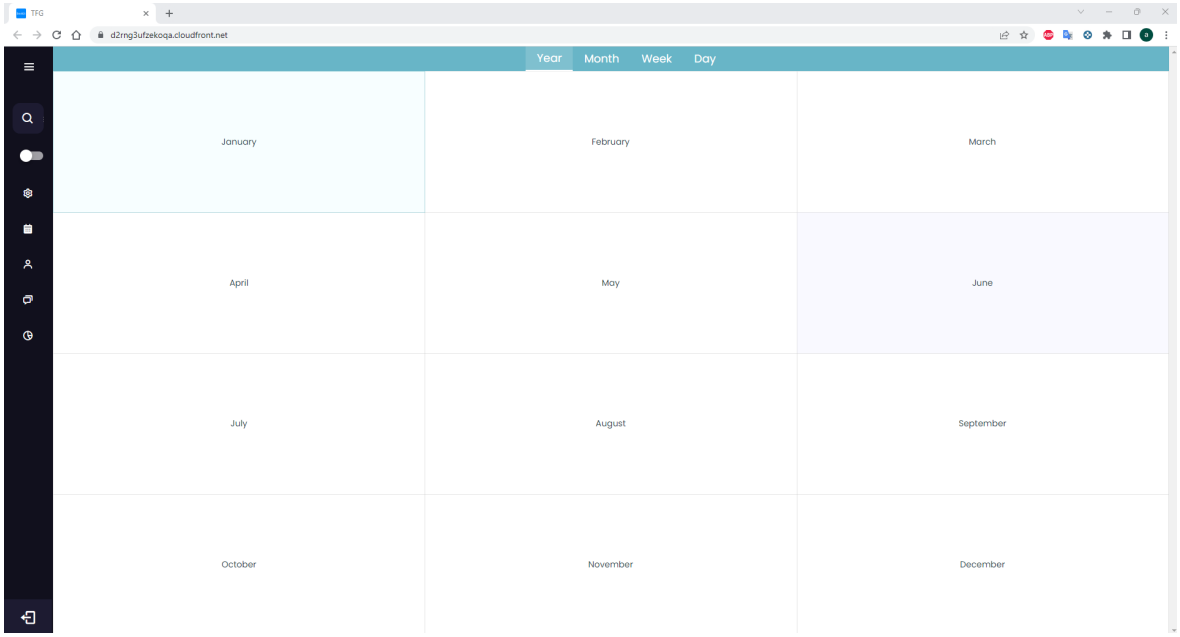


Figura 9: Vista de semana con eventos personalizados

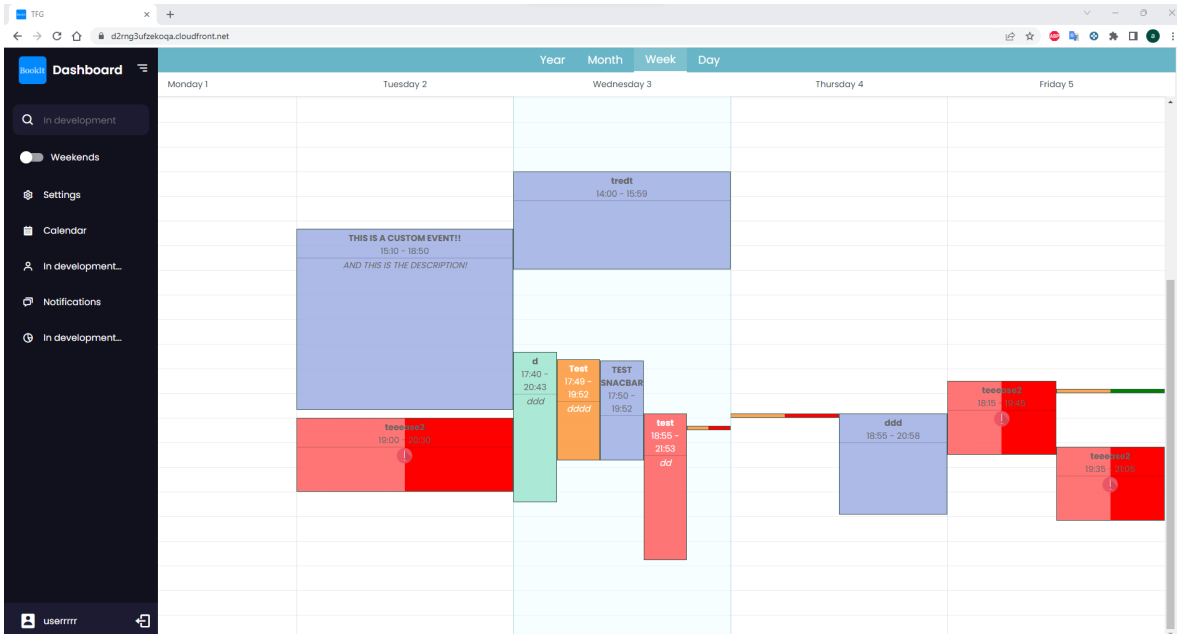


Figura 10: Creación de evento con local

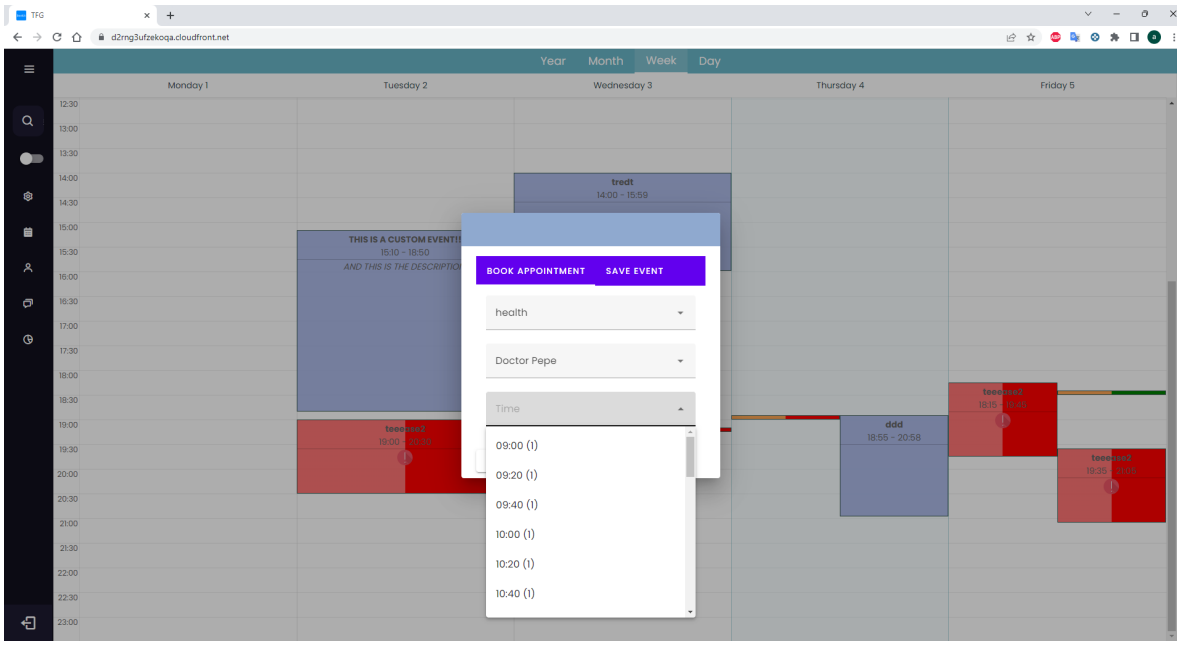


Figura 11: Creación de evento personalizable

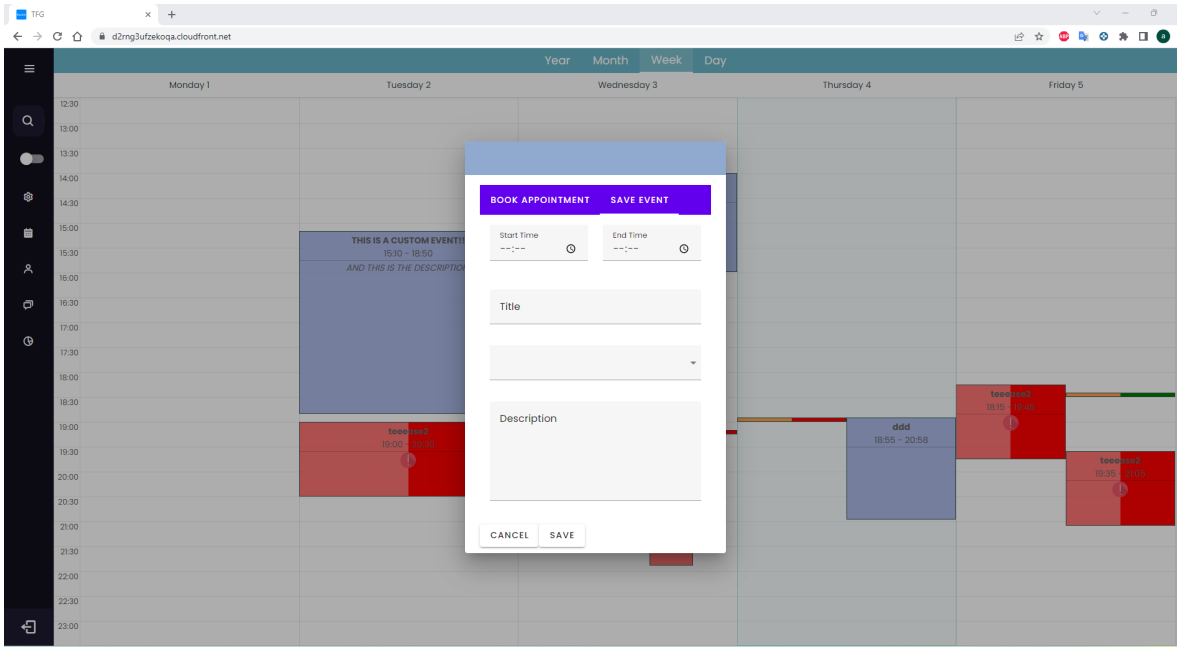


Figura 12: Descripción de eventos personalizados

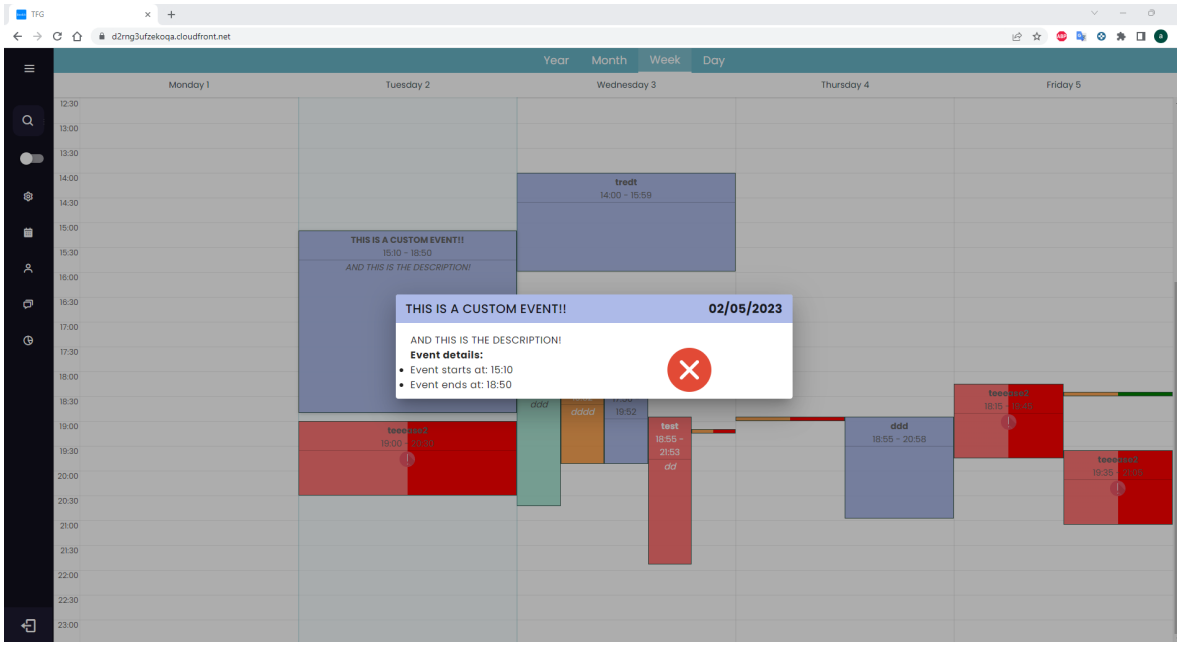


Figura 13: Vista de ajustes de usuarios normales

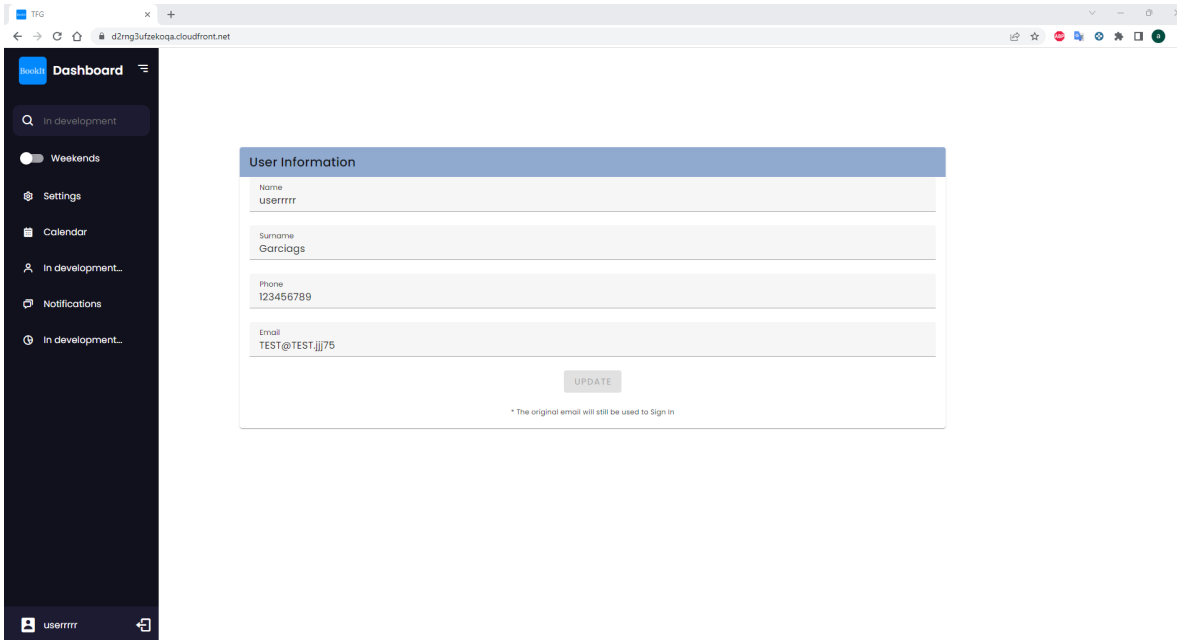


Figura 14: Vista de ajustes de usuarios locales

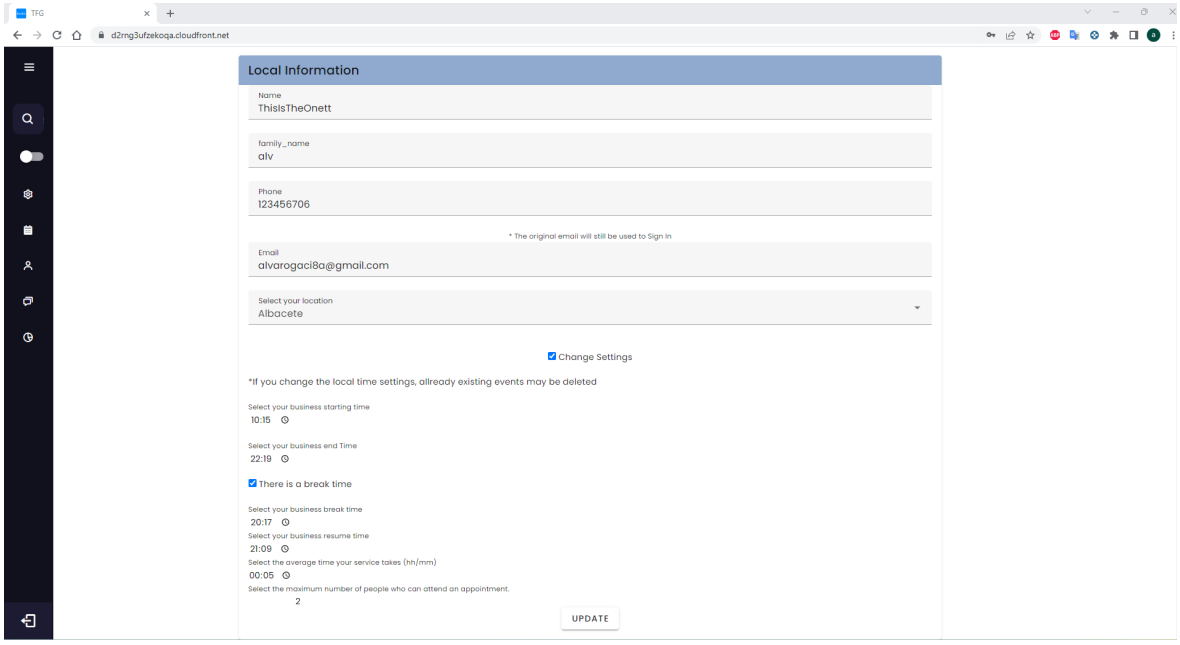


Figura 15: Versión móvil

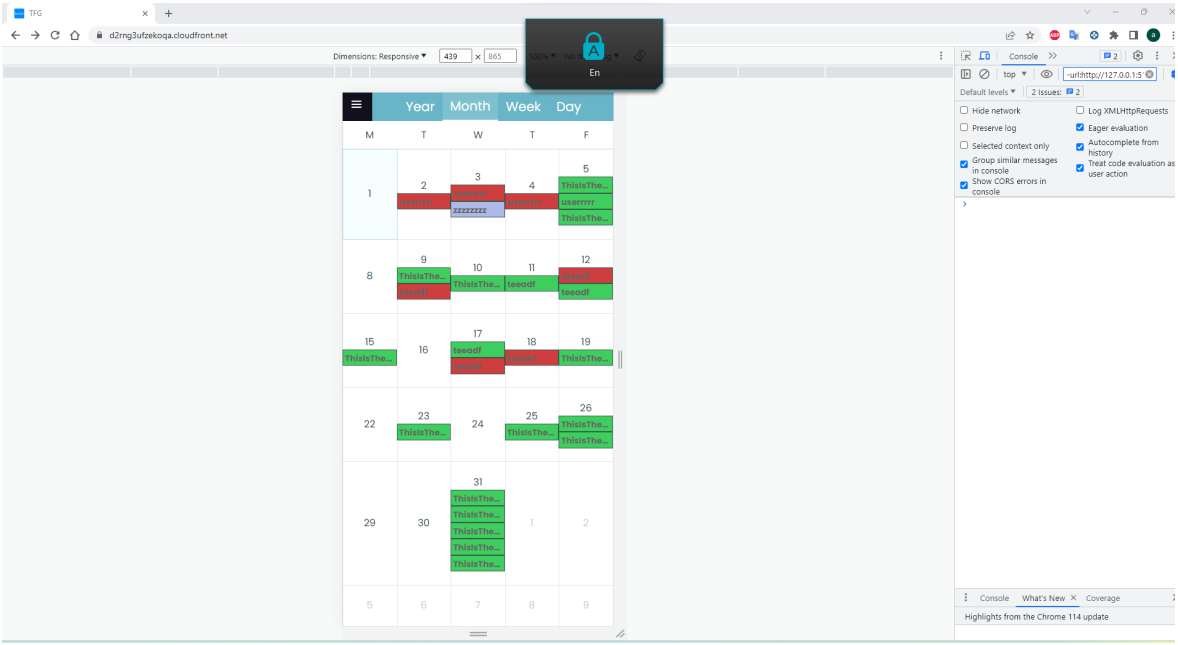


Figura 16: Versión móvil con barra lateral

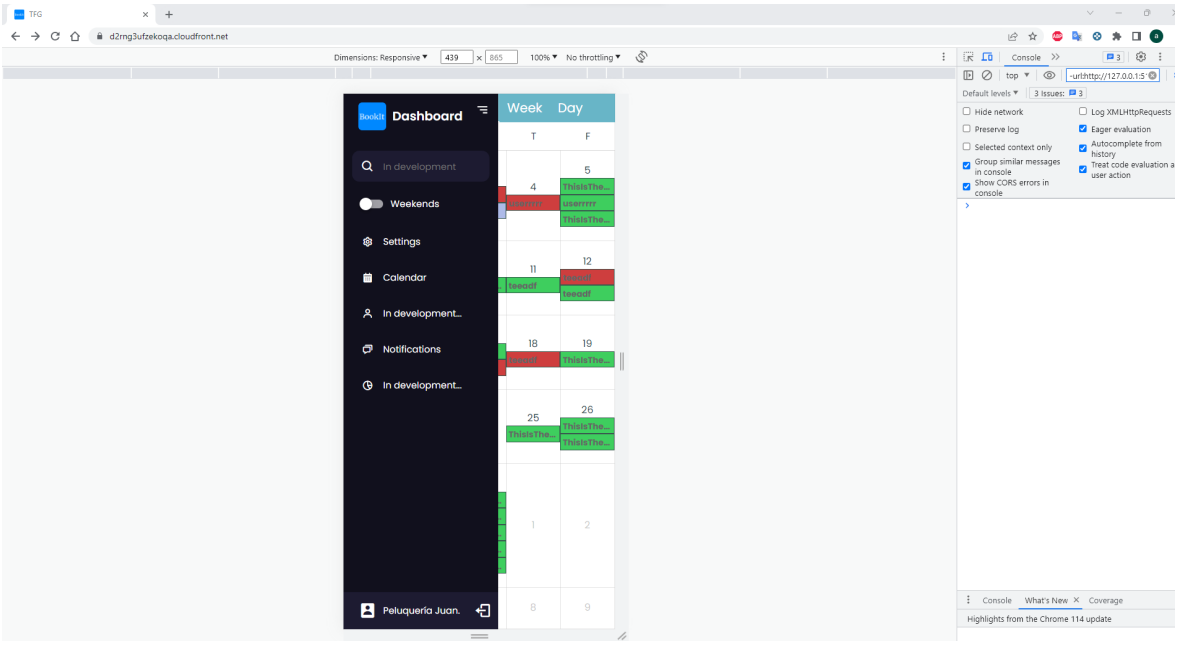
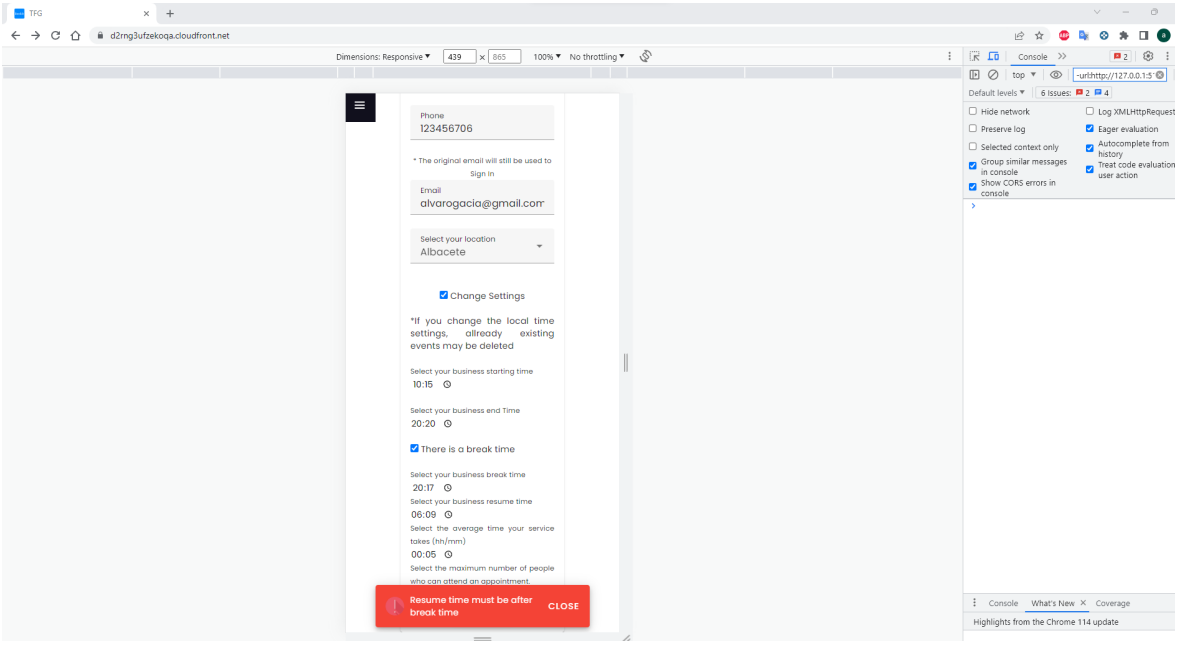


Figura 17: Ajustes con error por campos incorrectos versión móvil



4. Descripción informática

En este capítulo se explica el funcionamiento del proyecto desde un punto de vista técnico. Entrando en detalle sobre la arquitectura del mismo y detallando la implementación de cada uno de estas.

El código utilizado en la aplicación (Frontend y Lambdas) está disponible en el siguiente repositorio.
`https://github.com/AlvaroS11/alvaroGarciaSierraTFG`

4.1. Tecnologías utilizadas

En esta sección se explican las principales tecnologías que han sido seleccionadas para la implementación del proyecto y la razón de su selección. La mayoría de tecnologías utilizadas en la aplicación han sido servicios de AWS.

4.1.1. AWS Lambda

AWS Lambda [2] es un servicio informático de AWS que usa un modelo de ejecución sin servidor o *Serverless* basado en eventos. AWS Lambda es un FaaS (Function as a Service) el cual permite la ejecución de código sin administrar o aprovisionar servidores.

Este servicio responde a eventos y se escala automáticamente según el volumen de solicitudes entrantes. Esto hace de AWS Lambda una manera rentable y eficiente de implementar un código e infraestructura escalable.

La decisión de usar AWS Lambda ha sido debido a las ventajas que ofrece respecto a un sistema monolítico tradicional. Al adoptar un enfoque basado en la tecnología serverless y seguir los principios de microservicios, AWS Lambda facilita la descomposición de aplicaciones en funciones más pequeñas y autónomas. Esto agiliza el desarrollo, permite una mayor escalabilidad y tolerancia a fallos, y optimiza la utilización de recursos al pagar solo por el tiempo de ejecución utilizado. Debido a estos beneficios las grandes empresas están tratando de cambiar a este tipo de servicios, que usan microservicios o funciones serverless, para huir de los monolitos, aunque el desarrollo de estos servicios desarrollo de estos nuevos sistemas conlleva sus propios problemas [3].

4.1.2. AWS API Gateway

Aws API Gateway [4] es un servicio informático de AWS el cual facilita la creación, publicación, monitoreo y protección de las API (Application Programming Interaces) de una aplicación.

API Gateway funciona como una puerta de enlace entre el frontend y el backend. Además, API Gateway otorga una serie de funciones de seguridad, como autenticación y autorización de usuarios y protección contra ataques *DDoS*.

Este servicio ha sido elegido para llevar a cabo las funciones de la API REST de la aplicación debido a que es el principal servicio de AWS que ofrece esta funcionalidad.

Esta API Gateway también usará un *Lambda Authorizer* para autentificar y autorizar a los usuarios en cada una de sus llamadas al API Rest.

4.1.3. AWS Cognito

AWS Cognito[5] es un servicio de AWS el cual facilita la autenticación y acceso de los usuarios. AWS Cognito también facilita la autorización y administración de los usuarios mediante la clasificación de los usuarios en distintos grupos llamados *User Groups*. A estos User Groups se les pueden asignar *Roles* y *Políticas de acceso* para permitir el control de acceso de los usuarios a los recursos.

Este servicio también facilita el registro e inicio de sesión de los usuarios, así como la gestión de contraseñas, autentificación multifactor (*MFA*) y la posiblidad de iniciar sesión con proveedores de identidad externos como Facebook, Google o Apple.

AWS Cognito ha sido elegido para cumplir las funciones de login de los usuarios, así como para la creación de los usuarios y la autentificación y autorización de estos. Esta elección ha sido debida a la rápida integración con el resto de la aplicación, así como a la robusted y seguridad que ofrece.

4.1.4. AWS DynamoDB

AWS DynamoDB [6] es una base de datos no relacional de clave-valor sin servidor y altamente escalable de AWS. Está diseñado para manejar grandes cargas de trabajo y permite un almacenamiento eficiente y recuperación rápida de la información.
AWS DynamoDB ha sido elegido debido a su rapidez y capacidad de escalabilidad horizontal, permitiendo el aumento de capacidad de almacenamiento y rendimiento de forma automática.

4.1.5. AWS S3

AWS S3 [7] (Simple Storage Service) es un servicio de almacenamiento de objetos altamente escalable de AWS. Los datos son almacenados en buckets, los cuales actúan como contenedores lógicos para la administración y organización de los objetos.

AWS S3 proporciona una escalabilidad masiva, lo que significa que puede almacenar una cantidad ilimitada de información, así como una alta durabilidad, seguridad y disponibilidad al replicar los datos en distintas ubicaciones físicas.

La elección de utilizar S3, además de las características anteriores, se debe, a su capacidad para alojar páginas web *SPA*. Con S3 se puede alojar archivos estáticos, por lo que se pueden servir archivos HTML, CSS y JavasScript para ejecutar la página web desde un bucket.

4.1.6. AWS CloudFront

AWS CloudFront [8] es un servicio de AWS que ofrece una red de entrega de contenido (CDN) ofrecido por AWS. Esta CDN permite una entrega rápida y segura de contenido a los usuarios finales de todo el mundo.

Cloudfront ofrece un acceso rápido a los datos, reduciendo la latencia y aumentando la velocidad de carga de las páginas web. Esto es debido a que este servicio almacena en caché el contenido en una red de servidores distribuidos por todo el mundo. Este servicio también ofrece protección mediante opciones de encriptación, autenticación y control de acceso.

La elección de usar AWS Cloudfront se debe al incremento en la velocidad del acceso a la página web, así como a la posibilidad de habilitar HTTPS, añadiendo una capa adicional de seguridad al cifrar la comunicación entre el usuario final y el backend.

4.1.7. AWS CloudWatch

AWS Cloudwatch [9] es un servicio de monitorización de AWS. Este servicio permite almacenar registros, eventos y métricas de otros servicios de AWS.

AWS Cloudwatch se ha usado en este proyecto para ayudar en el desarrollo del mismo mostrando métricas, uso de recursos, latencia y errores.

4.1.8. AWS IAM

Las políticas y permisos IAM [10] definen los permisos para acceder a determinados recursos de AWS. Estas políticas son documentos JSON que son evaluados por AWS para permitir o denegar solicitudes.

Se han utilizado las políticas IAM para otorgar los permisos necesarios a determinados recursos, como las Lambdas y AWS Cognito siguiendo el principio de mínimo privilegio.

4.1.9. AWS Amplify

AWS Amplify [11] es una herramienta de AWS la cual ayuda en la creación de aplicaciones, esta herramienta permite desarrollar de forma sencilla los diferentes proyectos aprovechando otros servicios de AWS.

AWS Amplify se ha usado como soporte para la conexión de AWS Cognito con el frontend.

4.1.10. Vue3

Vue3 [12] es un framework de código abierto para el frontend basado en HTML, CSS y Javascript. Vue3 proporciona un modelo de programación declarativo y basado en componentes.

La decisión de utilizar Vue3 ha sido debida a su rendimiento, modularidad, escalabilidad y familiaridad previa.

4.1.11. HTML

HTML[13] es un lenguaje de marcas de hipertexto para la elaboración de páginas web. Se ha utilizado para crear el cuerpo de los componentes en Vue 3.

4.1.12. CSS

CSS [14], Hojas de estilo en cascada en español, es un lenguaje de diseño gráfico que permite añadir estilo a los documentos HTML. Se ha utilizado en Vue 3 para añadir estilo a los componentes.

4.1.13. Node.js

Node.js [15] es un entorno para la ejecución de aplicaciones Javascript sin necesidad de navegador. Node.js se ha utilizado debido a su familiaridad previa, ecosistema, coherencia en el uso de Javascript y sobre todo por su posibilidad de usar como lenguaje de las Lambdas de AWS.

4.2. Herramientas utilizadas

4.2.1. Postman

Postman [16] es una plataforma que permite la creación y prueba APIs. Ha sido utilizada para realizar comprobaciones sobre los diferentes endpoints de la API.

4.2.2. Git

Git [17] es la herramienta para el control de versiones mas utilizada a nivel mundial. Ha sido utilizado para controlar las versiones de la aplicación.

4.2.3. Github

Github [18] es un servicio basado en la nube el cual permite el alojamiento de repositorios Git de código.

4.2.4. NPM

NPM [19], por sus siglas en inglés Node Packet Manager, es un sistema de gestión de paquetes de código reutilizables de Javascript.

Se ha utilizado para gestionar y añadir las dependencias necesarias tanto para el frontend como para el backend.

4.2.5. Trello

Trello [20] es una aplicación para web y dispositivos móviles orientada a la gestión de proyectos. Esta herramienta facilita el desarrollo de diferentes tipos de productos mediante el uso de tableros organizativos donde se pueden organizar los diferentes eventos de un proyecto. Se ha usado Trello en el proyecto junto a la metodología ágil Scrum para gestionar el progreso del desarrollo del proyecto.

4.2.6. Visual Studio Code

Visual Studio Code [21] es un editor de código de Microsoft. Este editor ha sido utilizado para el desarrollo del frontend debido a su ligereza, rapidez y popularidad para lenguajes interpretados como Javascript.

4.2.7. Vue-cal

Vue-cal [22] es una librería utilizada en el proyecto para facilitar la creación del calendario en el frontend. Se ha decidido utilizar esta librería ya que se adecuaba a las necesidades del proyecto.

4.2.8. Vuex

Vuex [23] es una una librería muy popular en Vue.js que sirve para controlar el estado de la aplicación y almacenar información. Se ha utilizado para almacenar el estado global de la aplicación, permitiendo compartir y sincronizar datos entre las diferentes vistas y componentes de una manera centralizada.

4.3. Metodología

A continuación, se describirá la metodología utilizada en el desarrollo del proyecto.

Se ha optado por la metodología ágil con el fin de aplicar los conocimientos adquiridos durante la carrera. Además el uso de la agilidad ha permitido una gran flexibilidad, al promover la mejora continua del proyecto y poder adaptar el trabajo a los nuevos problemas emergentes o nuevas características.

Dentro de las metodologías ágiles, se ha decidido usar la metodología Scrum permitiendo dividir el trabajo en diferentes Sprints, cada uno de ellos con la intención de resolver una serie de objetivos. En el desarrollo del proyecto, los sprints no han tenido una duración fija de unas semanas, ha diferencia de lo habitual en la metodología Scrum, sino que ha variado según las necesidades del sprint, las nuevas incidencias emergentes y mi situación personal.

A continuación, se procede a explicar los objetivos principales de los cuatro diferentes sprints que se han llevado a cabo.

- Sprint 1. Investigación: El principal objetivo de este sprint ha sido la investigación del desarrollo del proyecto y el estudio de las diferentes opciones de las tecnologías usadas en el proyecto.
- Sprint 2. API REST y Lambdas base: En este sprint se ha creado una API REST 4.5.1 con operaciones CRUD junto a la creación de diferentes Lambas 4.5.2 que operaban estas operaciones básicas. Estas Lambdas hacían dichas modificaciones sobre una base de datos no relacional que también fue creada durante este sprint 4.5.3.
- Sprint 3. Desarrollo del frontend como SPA en Vue 3 4.6, comunicación del frontend con la API REST e implementación de la seguridad en la aplicación 4.5.4. En este sprint también entraban como objetivos la creación de nuevas Lambdas para la gestión de nuevas necesidades.
- Sprint 4. Integración completa entre el frontend y el backend, implementación de nuevas funcionalidades y arreglo de bugs.

Para la gestión de los sprints se ha usado la herramienta Trello 4.2.5, creando un nuevo tablero por cada nuevo sprint y organizando las tareas según su estado (Nuevo, En curso, Cerrado)

4.4. Requisitos

En esta sección se mencionan los requisitos necesarios del sistema para satisfacer las necesidades de los clientes.

Se clasificarán en dos categorías los requisitos, específicamente funcionales y no funcionales. Por un lado, los requisitos funcionales abarcarán todas las capacidades que el sistema debe tener para satisfacer las necesidades del cliente, mientras que los no funcionales se referirán a características generales o restricciones impuestas durante el desarrollo del producto, las cuales no afectan directamente las funciones que debe realizar.

4.4.1. Requisitos funcionales

Aunque la aplicación se comporte de una manera relativamente diferente dependiendo del tipo de usuarios, no se va a hacer una distinción entre ellos ya que la mayoría de sus requisitos son comunes. Sin embargo, para diferenciar los requisitos funcionales podemos dividir estos en dos bloques, eventos y personalización.

4.4.1.1 Eventos

RF1. Los bloques principales del calendario serán los eventos, dichas entidades hacen referencia a una situación para un usuario cualquiera (eventos propios), o para un usuario normal y un local (eventos compartidos o citas), en una franja horaria delimitada.

RF1.1. Los eventos propios pueden tener una descripción y título personalizados y solo están ligados al usuario particular dueño de dicho evento.

RF1.2. Dichos eventos pueden ser rechazados por un local o un usuario en cualquier momento.

RF1.3. Los eventos deben tener un tipo, este tipo servirá a los usuarios normales para filtrar los diferentes locales a la hora de encontrar uno apropiado. Los distintos tipos de eventos existentes en la aplicación no serán limitados, es decir, el sistema deberá de tener la capacidad de aumentar los posibles tipos en el futuro.

RF1.4. La aplicación usará los distintos tipos de eventos y el estado (rechazado o aceptado) para poder identificar más fácilmente los diferentes eventos a simple vista.

RF1.5. Los eventos compartidos pueden ser aceptados por los locales en cualquier momento.

4.4.1.2 Personalización

RF2. Todos los usuarios podrán visualizar su calendario de manera personalizada, así como los eventos programados en él, de acuerdo a sus preferencias. Además del aspecto visual, es importante permitir que la información de todos los usuarios sea correcta, adecuada y ajustable.

- RF2.1. Todos los usuarios podrán elegir entre distintos niveles de detalle como meses, semanas o días, así como mostrar o no los fines de semana.
- RF2.2. La aplicación permitirá el registro tanto de un usuario local, como de un usuario normal. A la hora del registro la aplicación debe almacenar la información básica de cada usuario, como su nombre, correo, número de teléfono etc. Si es un usuario local deberá introducir además información adicional para poder ser filtrado por los demás usuarios y para poder calcular su calendario.
- RF2.3. Los locales podrán modificar sus ajustes y horas de trabajo en cualquier momento. Los usuarios normales también podrán modificar su información personal.

4.4.2. No funcionales

RNF1. El backend se desarrollará usando los servicios en la nube de AWS.

- RNF1.1. El backend de la aplicación será diseñado para ser altamente escalable.
- RNF1.2. Se seguirá una política en el desarrollo de coste cero para los servicios de AWS.
- RNF1.3. La lógica del backend será manejada por los *Microservicios* sin servidor de AWS Lambda.
- RNF1.4. Todos los datos de los usuarios de la aplicación serán almacenados en una base de datos no relacional, DynamoDB.
- RNF1.5. El sistema proveerá una seguridad robusta, con mecanimos de autenticación y autorización, usando AWS Cognito para la administración de usuarios. Todas las comunicaciones del usuario en la aplicación deben usar HTTPS para que estas estén encriptadas y sean seguras.
- RNF1.6. La comunicación con el frontend se realizará mediante una API REST implementada con AWS API Gateway.

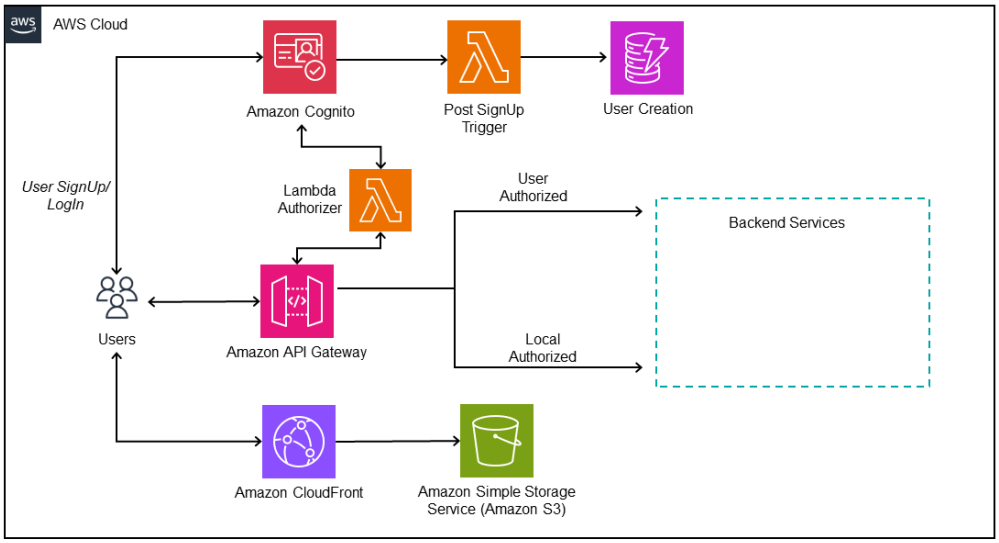
RNF2. El frontend se implementará en Javascript mediante el framework Vue 3.

- RNF2.1. El frontend se implementará como una Single Page Application (SPA).
- RNF2.2. Se adaptará el frontend para mantener una interfaz de usuario amigable y permitir el uso de la aplicación tanto en dispositivos móviles como en ordenador.
- RNF2.3. Se usará Vuex para gestionar el estado del frontend y promover la reusabilidad y la reactividad entre componentes.

4.5. Arquitectura

En esta sección se explica la descripción arquitectónica del sistema.

Figura 18: Arquitectura de la aplicación



El backend de la aplicación podemos dividirlo lógicamente en cuatro componentes fundamentales interconectados entre ellos:

- Conexión con el frontend a través de API Gateway. Esta parte del backend se encarga de exponer las API que permiten la conexión con el frontend. Este servicio sirve como puerta de entrada a las peticiones de los clientes y redirige sus peticiones a sus correspondientes Lambdas. Esta parte también está conectada a AWS Cognito mediante el uso de Lambda Authorizers para la autentificación y autorización de las llamadas.

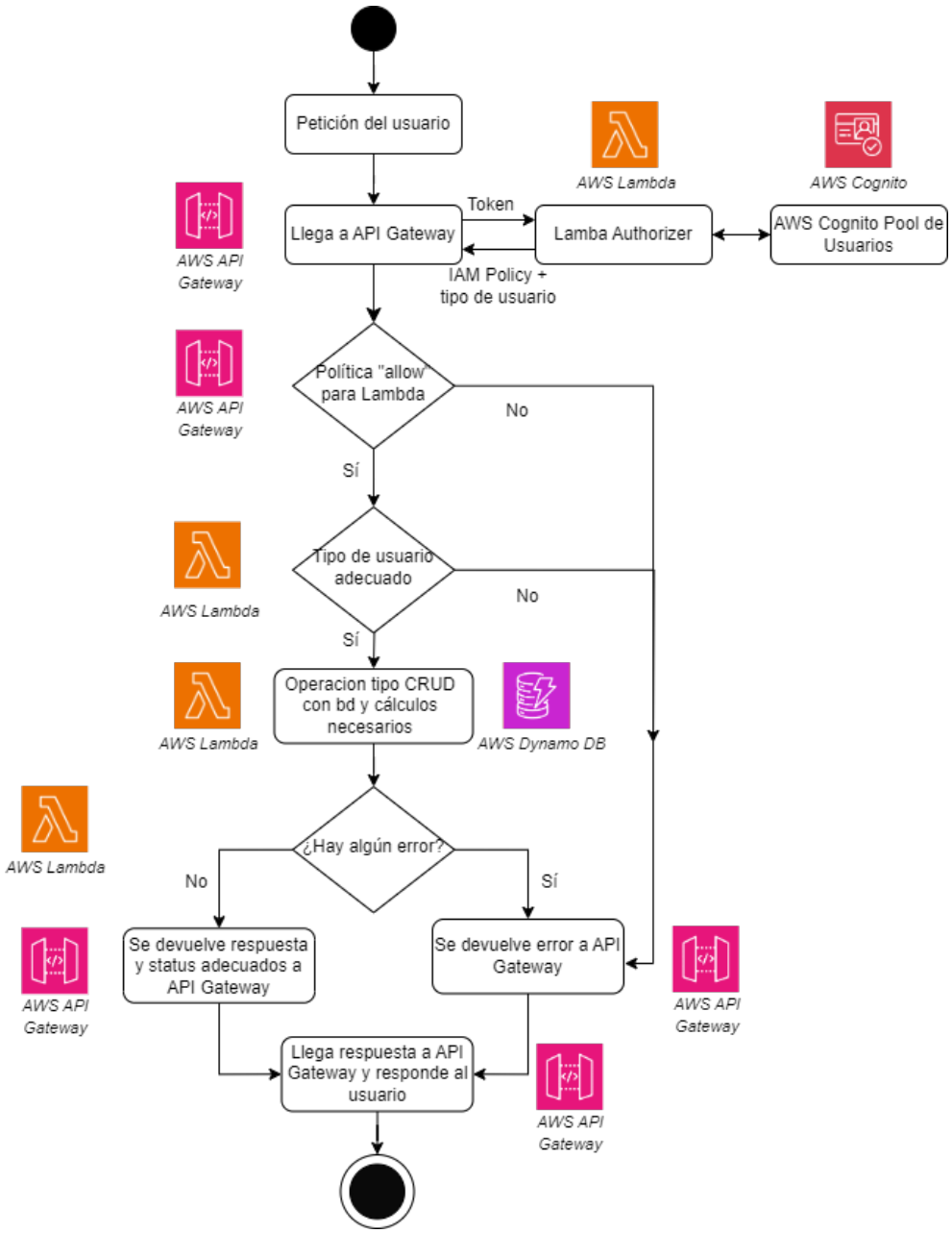
- Funcionalidades de las Lambdas y su conexión con la base de datos DynamoDB. Las Lambdas son invocadas por API Gateway e implementan la lógica de la aplicación. Estas Lambdas se comunican con una base de datos DynamoDB para leer, escribir, actualizar o borrar datos. Las Lambdas son escaladas dinámicamente en función del número de llamadas y demanda de la aplicación, por lo que se garantiza su disponibilidad

- Autenticación y autorización de la aplicación usando AWS Cognito. AWS Cognito es el servicio responsable de permitir el registro e inicio de sesión en la aplicación. Cuando un usuario inicia sesión, AWS Cognito entrega un token el cual es almacenado por el usuario y es enviado en las peticiones a API Gateway. En todas las llamadas se consulta la validez del token (fecha de caducidad...) usando Lambda Authorizer.

- Despliegue de la aplicación como SPA usando AWS Cloudfront y AWS S3. El frontend con el código compilado se ha subido a un bucket de S3 *Bucket* para permitir el acceso a la página web desde la nube. Adicionalmente se ha usado AWS CloudFront el cual actúa como un CDN (Content Delivery Network) para permitir el uso HTTPS, añadiendo una capa adicional de seguridad al cifrar la comunicación entre el usuario final y el backend.

En el siguiente diagrama se puede ver cómo una petición a la API pasa por los diferentes componentes de la arquitectura hasta ser procesada y su respuesta devuelta al usuario.

Figura 19: Diagrama de actividad



4.5.1. Implementación de la API con AWS API Gateway

Para permitir a nuestro frontend comunicarse con los servicios proporcionados por el backend, se ha implementado una API siguiendo los estándares REST RNF1.6..

La comunicación cliente-servidor es entregada por medio de HTTPS usando JSON en los métodos necesarios.

Cuando llega una llamada a la API, API Gateway genera un evento y ejecuta su correspondiente Lambda. Esta Lambda hace los cálculos y conexiones con la base de datos necesarios y devuelve la respuesta adecuada a API Gateway, el cual devolverá a su vez al usuario.

Antes de ejecutar estas Lambdas, todas ellas pasan por un Lambda Authorizer, servicio que se encargan de la autorización y autentificación del usuario para comprobar que la llamada proviene de un usuario existente de la aplicación. Si la petición no está autentificada por un usuario, esta Lambda devolverá un error 401 (Authorization Required)

Los distintos endpoints expuestos en la API del proyecto son descritos a continuación:

- **/lead**: Endpoint destinado a la creación de los "leads" o eventos de un usuario. (LLamada Lambda "GetLead/"updateLead")
- **/lead/id**: Endpoint destinado a la actualización de un evento de un usuario o local. (Llamada Lambda "GetLead") y a la obtención de todos los eventos de un usuario.
- **/leadid/sub**: Endpoint destinado a la obtención de un evento concreto de un usuario, se usan el id de usuario y el id del evento. El usuario debe estar autentificado como el dueño de dicho evento.
- **/local/id**: Endpoint destinado a la obtención de los locales disponibles y a la modificación de ajustes e información de un local. En el caso de la modificación (método PUT) el usuario debe estar autentificado como el local a modificar.
- **/local/id/weekStart/pageSize**: Endpoint destinado a la obtención de los eventos para un local determinado (Llamada Lambda getLeadsOwner). Se usa el id del local, la semana a partir de la que se quiere buscar eventos y el número de páginas a buscar, estos atributos son requeridos ya que la lambda asignada a este endpoint usa paginación. El usuario debe estar autentificado como el dueño de dichos eventos.
- **/localTime/local/day**: Endpoint destinado a la obtención de citas disponibles de un local en un determinado día. No requiere de autentificación adicional.
- **/users**: Endpoint destinado a la actualización de la información de un usuario. El usuario ha de estar autentificado como tal.
- **/users/id**: Endpoint destinado a la obtención de la información de un usuario. El usuario ha de estar autentificado como tal.

4.5.2. Implementación Serverless con AWS Lambda

La implementación del backend de la aplicación involucra el uso de alrededor de veinte Lambdas cada una con una funcionalidad distinta. Todas las Lambdas han sido escritas en Node.js debido a su familiaridad previa, ecosistema y coherencia en el uso de Javascript tanto en el frontend como en el backend.

La mayoría de las Lambdas se encargan de responder a las llamadas de API Gateway. Estas Lambdas están diseñadas para ser reutilizables, por lo que varias llamadas de API Gateway pueden invocar la misma Lambda, pero esta Lambda tendrá diferente comportamiento según el tipo de usuario identificado o los parámetros utilizados.

La ejecución con API Gateway se logra configurando las rutas y métodos de integración adecuados, lo cual permite que las peticiones del cliente sean finalmente enviadas a la Lambda correspondiente para su procesamiento. Estas Lambdas, además de hacer los cálculos necesarios, crean una conexión con la base de datos DynamoDB para realizar las operaciones CRUD (Create, Read, Update, Delete) necesarias en las tablas de la base de datos.

Además de las Lambdas provenientes de API Gateway existe una especial denominada "Lambda Authorizer". Como se ha comentado anteriormente, esta Lambda se sincroniza con la pool de usuarios de AWS Cognito para la verificación de estos. Esta Lambda se encargará de la autentificación y autorización de las peticiones entrantes antes de la ejecución de las Lambdas finales.

Existen siete Lambdas las cuales fueron creadas por AWS Amplify, estas Lambdas son invocadas por AWS Cognito y están involucradas en el proceso de creación, gestión de usuarios, reenvío de contraseñas y envío de correos. La contribución de AWS Amplify será explicada más adelante.

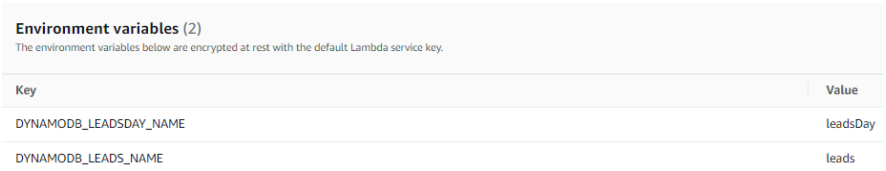
La integración de las Lambdas provenientes de API Gateway con AWS DynamoDB así como con AWS Cognito usan la librería AWS SDK (Software Development Kit) [24] . Esta librería proporciona herramientas para facilitar la comunicación entre las Lambdas y DynamoDB y Cognito.

Todas las Lambdas han sido configuradas añadiendo los parámetros necesarios para su función. Para las Lambdas que interactúan con la base de datos DynamoDB se ha añadido la información sobre la ubicación de esta en un script aparte (Ver código 1) y los nombres de las tablas necesarias como variables de entorno (Ver figura 20).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
const REGION = "us-east-1";
const ddbClient = new DynamoDBClient({region: REGION})
export {ddbClient}
```

Código 1: Declaración de la región de la base de datos

Figura 20: Ejemplo de variables de entorno de una Lambda



Todas las Lambdas tienen asignadas un rol de ejecución, el cual es un rol de AWS Identity and Access Managament (IAM) que concede a la Lambda u otro servicio el o los permisos para acceder a otros recursos de AWS. Todas las Lambdas tienen un permiso de creación de eventos para AWS Cloudwatch (4.1.7). Las Lambdas que operen sobre la base de datos tienen los permisos necesarios para la o las operaciones CRUD que se hagan en esta (Ver figura 21). Otras Lambdas también tienen su permiso específico, como el Lambda Authorizer que requiere acceso de lectura sobre la pool de usuarios de AWS Cognito. Todas las Lambdas siguen el principio de mínimo privilegio con respecto a sus políticas asignadas.

Figura 21: Política de lectura a la tabla de de la base de datos

Para implementar un nuevo servicio en la posible futura expansión de la aplicación, solo haría falta crear una nueva Lambda, enlazarla con los servicios necesarios, como puede ser un endpoint de nuestra API, la pool de usuarios de AWS Cognito o una tabla de la base de datos DynamoDB, y asignar los roles de ejecución y variables de entorno correspondientes.

A continuación se procede a explicar algunas de las Lambdas más significativas:

- **GetLead** Esta Lambda implementa una función muy sencilla de tipo "GET", simplemente devolviendo un evento dado un id, esta función solo es usada como respuesta a una redirección al crear un evento. A esta función puede acceder el usuario dueño de dicho evento.
- **GetLeadsOwner** Esta función es la encargada de devolver los eventos de los locales. Debido a que el volumen de los eventos de un local puede llegar a ser muy elevado, esta Lambda implementa paginación. Esta Lambda recibe como parámetros el id del local, el comienzo del día a buscar y el tamaño de la "página", con el id del local y el día, esta función obtiene de la tabla "LeadsDay"(Eventos

por día) (ver sección 4.5.3) hasta un máximo de 100 eventos. Dado que en dicha tabla, se almacenan todos los id de los eventos existentes y por haber de un día en un local, se obtendrán todos los eventos posibles en días consecutivos hasta llegar a los 100 eventos. Esto se debe a que posteriormente se hará una operación de procesamiento por lotes (Batch Operation) sobre la tabla de "Leads"(Eventos) dado el id de los eventos, que en DynamoDB tiene un máximo de 100 items, para obtener los eventos finales. Después de obtener esta información, la función devolverá los eventos y el último día de eventos enviado. Con esta información, en el frontend se almacenan los eventos recibidos y se analiza si se han obtenido los eventos necesarios para mostrar, si no es así, volverá a llamar a la API para obtener más eventos desde el último día de eventos recibidos, este proceso se repetirá hasta obtener los eventos necesarios. Esta función es accedida por el usuario local acreditado.

Este mecanismo de paginación mejora el rendimiento tanto en el frontend, como en el backend. En el frontend hace que las respuestas sean más fáciles de manejar al tener un número limitado de eventos que procesar y estos estar ya ordenados. En el backend mejora el rendimiento considerablemente al hacer consultas eficientes, sin tener que obtener lo que podría llegar a ser miles de eventos, consumiendo a su vez muchos recursos.

- **UpdateLead** Esta función se encarga de aceptar o denegar eventos. A esta función pueden acceder cualquier tipo de usuario para aceptar un evento personalizado o denegar un evento, pero para aceptar un evento de usuario-local solo podrá acceder a la función el local del evento.

- **GetTimes** Esta función es invocada cuando un usuario quiere conocer los tiempos disponibles de un local en un día por ello, recibe como parámetros el id del local y el día. Esta función también es la encargada de crear los elementos de la tabla "LeadsDay". La creación de los tiempos que ha de tener un Local en un día, se logra obteniendo los ajustes del local (tabla "Locals") e iterando con la duración del servicio desde la hora de apertura hasta la hora de pausa, y desde la hora de reapertura hasta la hora de cierre. A esta función puede acceder cualquier tipo de usuario.

- **CreateUser** Esta operación es invocada por AWS Cognito cuando un usuario completa el proceso de registro en la aplicación (Post Lambda Trigger) (ver figura 25). Como se explica en la sección 4.5.4, después de que un usuario se registre en la aplicación, este ingresará en la pool de usuarios de la aplicación, sin embargo por motivos de eficiencia y seguridad, la información del usuario se almacenará en la tabla "Users"de DynamoDB. Aunque la información del usuario es comprobada en el frontend, esto no es suficiente para garantizar su validez, por lo que esta función comprobará que todos los campos sean correctos y tengan sentido, como por ejemplo, que la hora de apertura de un local no sea inferior a la hora de cierre.

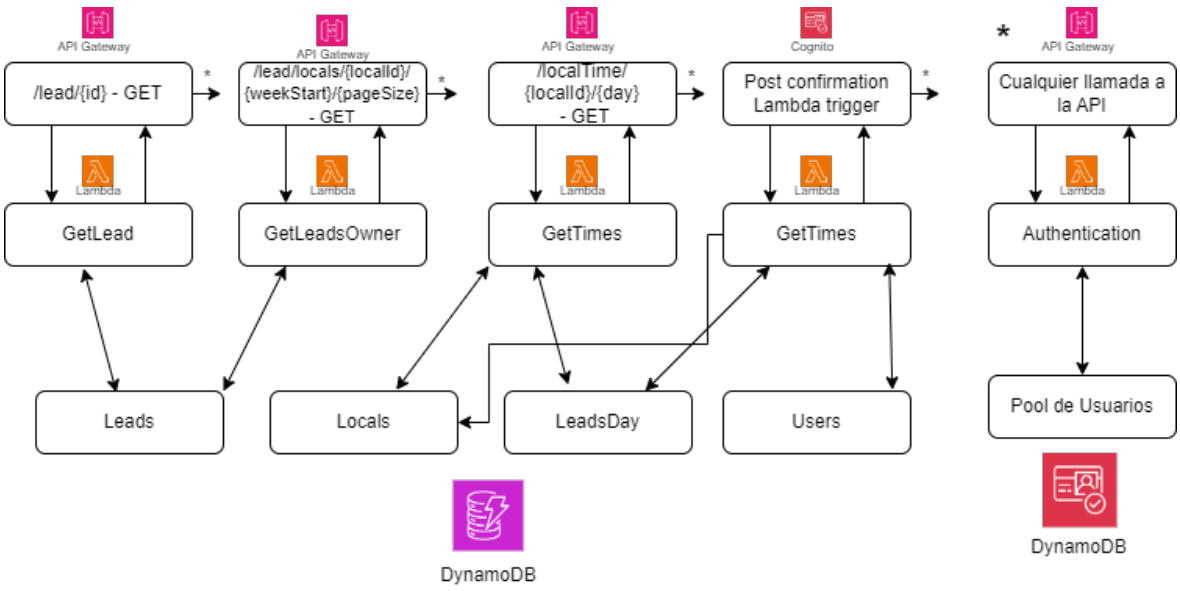
- **Lambda Authorizer**: Esta función es invocada en cada una de las llamadas a la API, su función es autorizar a los usuarios antes de invocar a la Lambda correspondiente. Este Lambda Authorizer comprueba el token JWT previamente

firmado por Cognito para garantizar su autenticidad, esto token ha sido enviado al usuario en el proceso de inicio de sesión. Esta Lambda se comunica con con la pool de usuarios de AWS Cognito y comprueba la autenticidad y validez del token, también comprueba si el usuario es un usuario estándar o un usuario local. Después de esto, si los valores son válidos, la petición es aprobada y se genera una política "Allow"para la Lambda adecuada.

```
if(isLocal === "true" && decodedToken.payload['custom:isLocal'] ===
    "true"){
    callback(null, generatePolicy('local', 'Allow', event.methodArn,
        decoded.sub));
}
// [...]
```

Código 2: Generación de políticas Lambda Authorizer

Figura 22: Diagrama Lambdas



Inicialmente se desarrollaron algunas lambdas para ser utilizadas por otras, esto era óptimo para un desarrollo más rápido, además se aumentaba la modularidad, reusabilidad y simplificación. Pero finalmente, debido a que esto solamente se daba en dos casos y a los inconvenientes de llamar a una Lambda desde otra Lambda, se decidió no invocar estas funciones desde otras Lambdas.

Entre los inconvenientes de llamar a una Lambda desde otra Lambda se encuentran:

- Aumento del tiempo de ejecución debido tanto a la latencia, sobre todo si la función debe ejecutar un "*Arranque en frío*", como al tiempo de inactividad de la primera Lambda que ha de esperar a esta.
- Aumento en el coste. Al aumentar el tiempo de ejecución y la transferencia de datos entre Lambdas el coste final por ejecución es mayor.

Estos aspectos negativos influyen directamente en algunos de los objetivos del proyecto, como la escalabilidad (al aumentar el tiempo de ejecución)RNF1.1. y el coste cero RNF1.2..

```
const lambdaParams = {
                    FunctionName: 'getTimes',
                    InvocationType: 'RequestResponse',
                    LogType: 'None',
                    Payload: payloadStr
                }

                const lambdaResponse = await
                    lambda.invoke(lambdaParams).promise()
                const newStartingDay = JSON.parse(lambdaResponse.Payload)
    // [...]
```

Código 3: Invocación de una Lambda desde otra

4.5.3. Base de datos con AWS DynamoDB

Para el almacenamiento de los datos se ha utilizado una base de datos no relacional orientada a documentos. La decisión de usar esta base de datos ha sido debida a su similitud con la base de datos MongoDB y familiaridad previa y a ser una base de datos NoSql.

El hecho de ser una base de datos no relacional permite a nuestra base de datos una serie de ventajas muy acordes con los objetivos de nuestra aplicación, sobre todo la escalabilidad RNF1.1.. La escalabilidad horizontal de esta base de datos y su carácter descentralizado permite soportar estructuras distribuidas, pudiendo aumentar tanto el número de máquinas (llamadas unidades de escritura y lectura en AWS), conforme al tráfico de la aplicación, como a la capacidad de la base de datos. También nos permite optimizar las consultas para grandes cantidades de datos, integrándose de una forma muy eficiente a las consultas de AWS Lambda que se encargan de recoger los eventos del calendario de los usuarios. Por último, el uso de una base de datos no relacional y su carácter abierto y flexible, ha permito una evolución más rápida del proyecto , permitiendo adaptarse a las necesidades emergentes y nuevos campos necesarios de la base de datos que han ido surgiendo durante el desarrollo de la aplicación. Esta nueva característica de las bases de datos no relacionales también incrementará la rapidez en el futuro desarrollo de la aplicación.

En nuestra base de datos se han creado cuatro tablas distintas para el manejo de la información:

- **leads** En esta tabla se almacenan todos los eventos del calendario. Aprovechando la flexibilidad de la base de datos no relacional la información almacenada a los eventos varía dependiendo de si el evento es propio, o si vincula a un usuario con un local. En el último caso la información se comprende de su "id", id del poseedor del evento, id del local asociado, hora de comienzo, hora de finalización, tipo, si es un evento propio o asociado a un local y si este ha sido aceptado o no (ver figura 23). Mientras que si el evento solamente almacena información del usuario, no almacenará el local pero almacenará una posible descripción.

- **leadsDay** En esta tabla se almacenan las relaciones, ordenadas por día y local, de todos los eventos de un local para un determinado día, así como el tiempo en minutos en el que comienza cada evento y los huecos libres de dicho local en dicho día 24 . Usa el Id del local como clave primaria o "partition Ke".[en] DynamoDB y el día como "sort key", estas dos claves forman la clave primaria compuesta de la tabla. El uso de ambas claves aumenta significativamente la velocidad de lectura tanto cuando un local pide la información de sus eventos por paginación, como cuando un usuario pide los horarios disponibles de un local concreto en un día concreto.

- **locals** En esta tabla se almacena toda la información y ajustes sobre los locales. Usa como "sort key.[el] atributo "type" el cual hace referencia al tipo de local, esto aumenta la velocidad de lectura cuando un cliente solicita la lista de locales existentes. En un desarrollo futuro 5 debería investigarse la posible creación de un nuevo índice "ciudad"para agiliza las consultas por este campo.

Figura 23: Item de la tabla leads

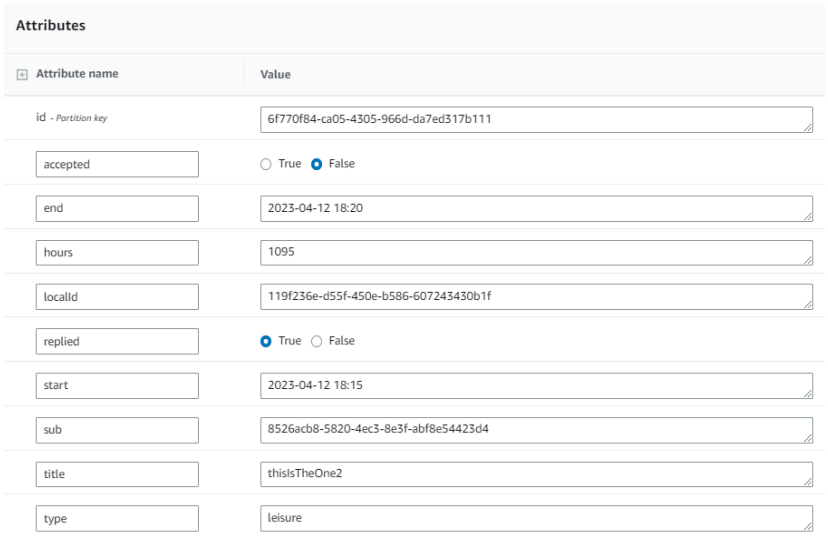
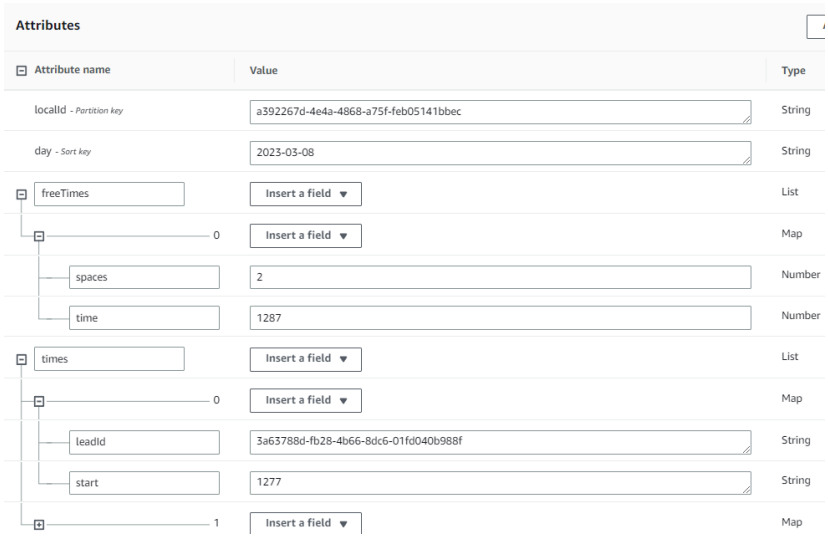


Figura 24: Item de la tabla leadsDay



- **users** En esta tabla se almacena toda la información sobre los usuarios.

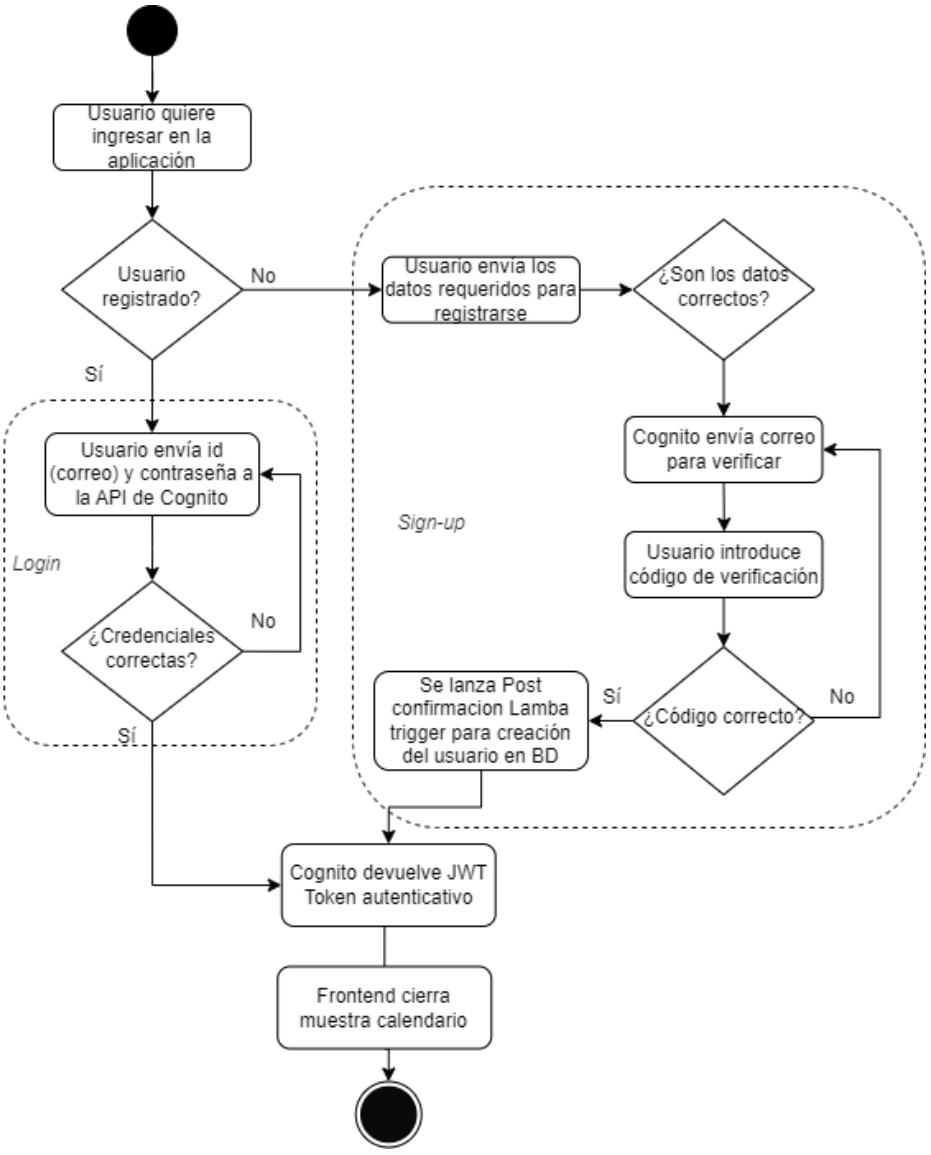
4.5.4. Autenticación y autorización con AWS Cognito

Para aumentar la seguridad de la aplicación, además de la implementación de la comunicación con S3 y la API mediante HTTPS, se han implementado mecanismos de autorización y autenticación. RNF1.5.

AWS Cognito realiza un papel fundamental siendo este el principal encargado de la autenticación, registro y acceso de los usuarios a la aplicación. La implementación con AWS Cognito ha constado en primer lugar de la creación de una pool de usuarios, la cual ha sido configurada para autentificar, registrar y permitir acceso a los usuarios a través de la API de Cognito 29 .

A continuación, se muestra un diagrama de actividad para dar una idea general de los procesos de inicio de sesión e inscripción:

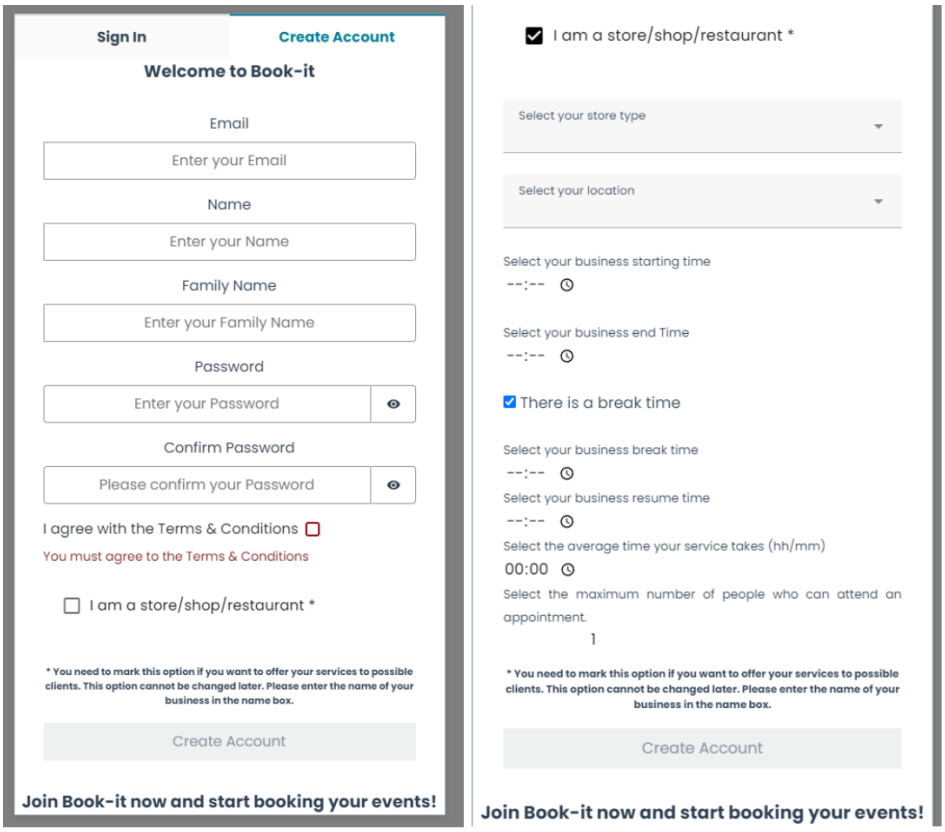
Figura 25: Diagrama de registro e inscripción



El proceso de AWS Cognito en la aplicación implica las siguientes fases:

1. Creación del usuario: Cuando un usuario entra en la aplicación entrará en una vista para autentificarse, si es la primera vez que el usuario usa la aplicación tendrá que registrarse. Durante este proceso se solicitan y validan datos personalizados como nombre, apellido, correo y contraseña, la cual ha sido configurada para requerir con los requisitos de seguridad, como un mínimo de ocho carácteres y el uso obligado de múltiples tipos de carácteres. Si el usuario es un local deberá introducir toda la información necesaria para la creación de los huecos disponibles como el horario de apertura y de cierre, el tiempo por servicio, ya que en este momento la aplicación solo soporta un tipo de servicio por local, el número máximo de usuarios por servicio y un tiempo muerto, por ejemplo una pausa en el horario para comer, si existe (ver figura 26).

Figura 26: Página de registro del usuario



2. Una vez el usuario ha ingresado sus datos, un correo electrónico es enviado por AWS Cognito al correo del usuario para comprobar su veracidad (ver figuras 27 y 28) . Si el usuario es verificado, se ejecuta un disparador para ejecutar una Lambda, la cual se encarga de crear un usuario en la base de datos de la aplicación si los datos proporcionados son correctos (ver sección 4.5.2).

3. Una vez el usuario se ha registrado correctamente, puede iniciar sesión en la aplicación usando sus credenciales. En este punto AWS Cognito permite el inicio de sesión mediante proveedores externos como Google, Facebook o Apple, pero está opción ha sido desactivada ya que se requiere de información adicional que debe ser introducida manualmente por el usuario.

4. Cuando el usuario inicia sesión y es autenticado mediante la API de Cognito

Figura 27: Correo de verificación

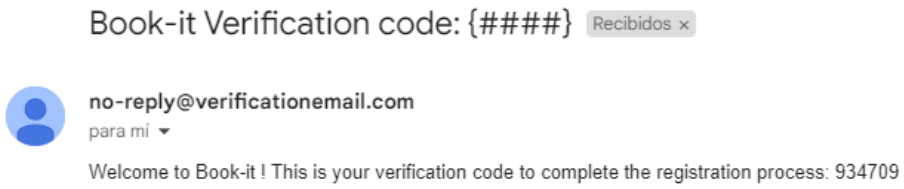
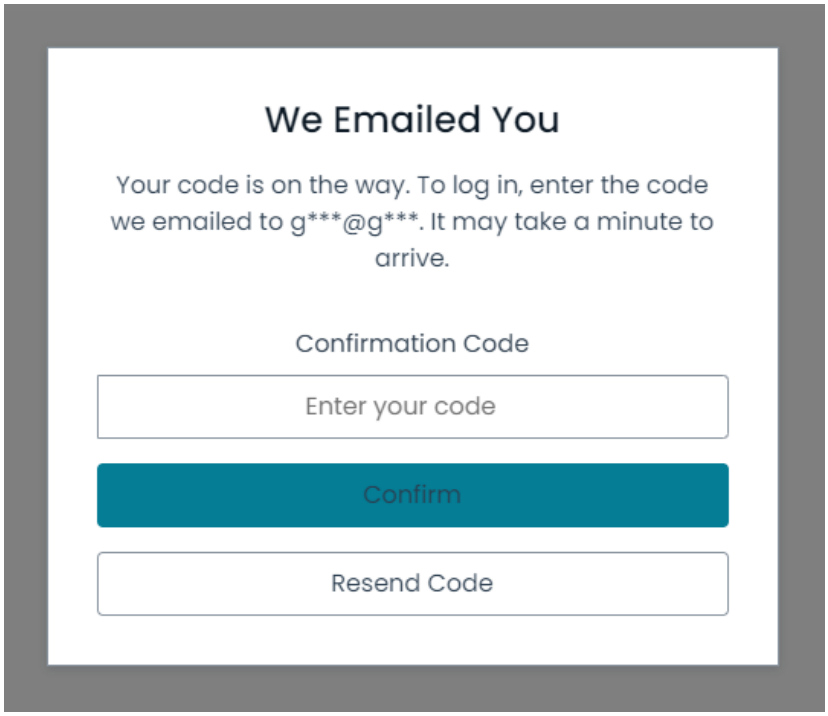


Figura 28: Solicitud de verificación

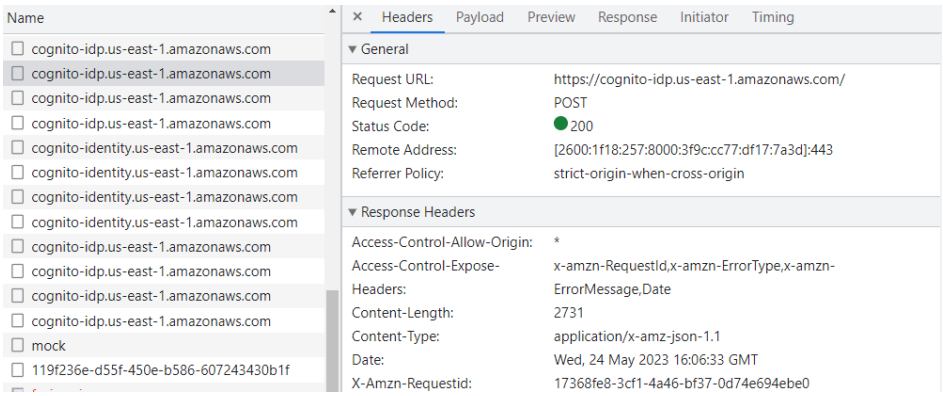


correctamente, esta le envía un token JWT firmado digitalmente por Cognito para garantizar su autenticidad y seguridad.

5. El usuario usará este token en todas sus llamadas a la API de la aplicación para que sean aceptadas por el Lambda Authorizer.

AWS Cognito también permite la autentificación multi factor (*MFA*), envío de SMSs y el rastreo de dispositivos pero estos no están habilitados. La recuperación de contraseñas está habilitada pero no implementada en el frontend.

Figura 29: Conexión a la API de AWS Cognito



4.5.5. Alojamiento de la página con AWS Cloudfront y AWS S3

Se ha usado Cloudfront y S3 para alojar y poder entregar los contenidos de la página web.

Los contenidos estáticos de la página web, como HTML, CSS y Javascript, han sido alojados en un "bucket"de AWS Simple Storage Service (S3). Este "bucket" es un contenedor lógico donde se puede ubicar cualquier tipo de contenido estático. El bucket creado para la aplicación ha sido configurado para alojar una página web estática, conteniendo un endpoint configurado para la entrega de este contenido web estático.

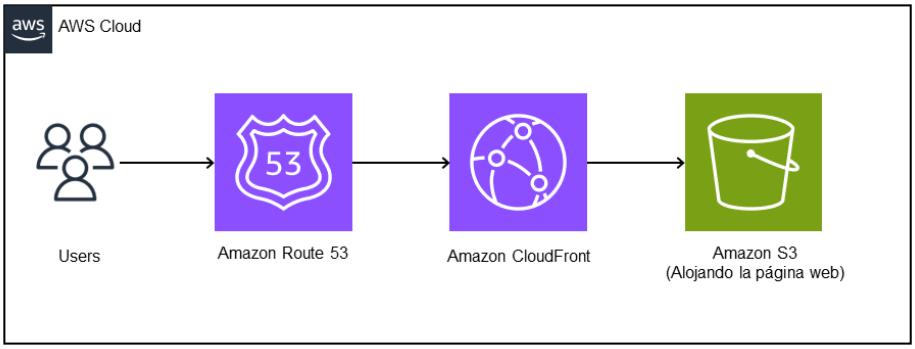
Para asegurar la seguridad, se decidió usar Cloudfront como una capa de distribución de contenido (CDN) el cual apunta al bucket de S3. Cloudfront permitió redirigir todas las solicitudes HTTP a HTTPs para una comunicación segura con los usuarios. Así cuando un usuario accede a la página web mediante el protocolo HTTP, Cloudfront detecta el uso de un protocolo no seguro y automáticamente redirige la solicitud utilizando HTTPs. Este uso de HTTPS para la obtención de la página web junto a su uso en las llamadas a la API garantiza que todas las comunicaciones del usuario en la aplicación estén encriptadas y sean seguras RNF1.5..

Otro de los motivos por los que se decidió el uso de Cloudfront fue por el incremento en la velocidad de carga de la página. Esto es debido a que como Cloudfront, como CDN, distribuye el contenido a los distintos servidores en diferentes ubicaciones distribuidas geográficamente. Cuando un usuario solicita contenido, Cloudfront redirigirá la solicitud al servidor más cercano al usuario, reduciendo la petición en latencia y tiempo de respuesta. Además Cloudfront como otras CDNs utiliza técnicas de almacenamiento en caché, permitiendo entregar el contenido a varios usuarios de manera simultánea desde su caché sin tener que recuperar el contenido estático desde el servidor de origen (S3). En el caso de Cloudfront destaca la llamada "Managed Caching", la cual está optimizada para el almacenamiento en caché del contenido estático, como son los objetos de nuestra página web.

Además, con el uso de Cloudfront seguimos mejorando la escalabilidad de la aplicación, ya que mediante el uso de una CDN aumentamos significativamente la capacidad de manejar grandes volúmenes de tráfico. Esto es debido a que al distribuir la carga entre varios servidores, la CDN puede escalar fácilmente para manejar picos de tráfico sin afectar al rendimiento de la aplicación RNF1.1..

Inicialmente se planteó el uso de AWS Route 53 [25], el servicio de DNS de AWS, usando un dominio propio para la página web que apuntara a Cloudfront 30. Sin embargo, debido a los gastos que supondría la compra de un dominio y el uso de Route 53 (ya que AWS no ofrece un nivel gratuito en este servicio), se decidió omitir esta funcionalidad RNF1.2..

Figura 30: Posible implementación con Route 53



4.6. Desarrollo del frontend basado en Vue 3

Vue 3 [12] es un framework de código abierto de Javascript para construir interfaces de usuario interactivas y reactivas. Vue 3 proporciona un modelo de programacióon declarativo y basado en componentes.

Vue 3 es la última versión de Vue.js, el cual incluye varias mejoras y características respecto a su predecesor, Vue 2, pudiendo destacar el rendimiento y escalabilidad [26] . La decisión de utilizar Vue 3 ha sido debida a su rendimiento, modularidad, escalabilidad y familiaridad previa.

4.6.1. Arquitectura del frontend

4.6.1.1 Reusabilidad

Uno de los elementos principales de Vue 3 es el uso del desarrollo basado en componentes. Estos componentes sirven como los pilares fundamentales de la aplicación, encapsulando las distintas partes de una forma autocontenida y reusable. Vue 3 permite el uso de "Composition API", el cual se ha usado en el proyecto, este conjunto de APIs permiten estructurar y organizar los componentes de una forma sencilla, aumentando así su reusabilidad.

Antes de explicar las principales partes de la estructura de la aplicación, es importante recordar que la funcionalidad y experiencia de usuario es distinta según el tipo de usuario, esto conlleva inevitablemente a la creación de vistas distintas para cada uno de ellos. Sin embargo, se ha evitado la duplicación de código y compartido una cantidad significativa de código entre las vistas similares.

Esta compartición de código se ha logrado mediante la creación de componentes reutilizables, los cuales han sido diseñados con la configuración necesaria para la adaptabilidad a los requisitos de cada tipo de usuario. Se ha creado un script "utils"donde se han agrupado los diferentes métodos reutilizables a lo largo de la aplicación, logrando la duplicación de código compartido.

En el contexto de promover la reutilización de código, también se ha utilizado Vuex 4.2.8, una biblioteca muy popular en Vue.js que sirve para controlar el estado de la aplicación y almacenar información. El uso de la "Store"de Vuex, un contenedor lógico donde se almacena el estado global de la aplicación, ha permitido compartir y sincronizar datos entre las diferentes vistas y componentes de una manera centralizada. También ha permitido la reutilización de código al acceder y modificar esta información de una manera centralizada (Ver figura 33).

4.6.1.2 CSS

Aunque algunos componentes tengan su propio css, existen algunos componentes que comparten el mismo archivo de css, reutilizando estilos y evitando la duplicación, a la vez que ahorrando tiempo y esfuerzo en su desarrollo. El hecho de compartir este archivo css y el uso de la librería "Vuetify"ha garantizado un estilo visual común y uniforme, manteniendo una identidad visual consistente. Este css también es usado de

forma diferente por los componentes según el dispositivo del usuario para garantizar una buena visibilidad de los elementos tanto en ordenadores como en móviles (Ver figura 12) RNF2.2..

4.6.1.3 Gestión de errores

Para el manejo de errores se ha tomado un enfoque de gestión de excepciones descentralizado, teniendo cada tipo de excepción su procedimiento personalizado (Ver código 4).

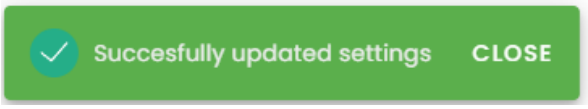
```
} catch (err) {
            this.snackbarText = 'Error creating Event, please try again or
                reload the page';
            this.snackbarColor = 'red';
            this.snackbar = true;
            const data = {
              localId: info.localId,
              day: this.formatDate(info.start)
            }
            store.dispatch('getTimes', data)
          }
          this.showEventCreationDialog = false
        })
    // [...]
```

Código 4: Ejemplo de manejo de una excepción

Además del manejo de datos que trate cada excepción, la mayoría de estas mostrará un snackbar informativo al usuario, al igual que para las comprobaciones previas al introducir información. Estas comprobaciones incluyen que la hora de finalización sea mayor a la de comienzo (para los eventos personalizados y para la modificación y creación de algunos de los ajustes de los locales). Para la comprobación de la modificación de teléfonos y correos electrónicos se han usado una serie de reglas que siguen patrones regex (Ver código 5)

De forma homóloga al uso de un snackbar para informar al usuario de un error, también se ha empleado este snackbar para informar a un usuario que una acción ha sido llevada a cabo con éxito (Ver figura 31) . Esto se puede ver al aceptar o rechazar un evento, al crearlo o al modificar su información.

Figura 31: Snackbar



```
rules: {
       email: value => {
         const pattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/
         return pattern.test(value) || 'Invalid email address.'
       },
       phone: value => {
         const phoneNumberRegex = /^\d{10}$/
         const phoneNumberRegex1 = /^\d{11}$/
         const phoneNumberRegex2 = /^\d{9}$/;
         return (phoneNumberRegex.test(value) ||
             phoneNumberRegex1.test(value) || phoneNumberRegex2.test(value))
             || 'Invalid phone number.'
       }
     }
   // [...]
```

Código 5: Reglas regex

4.6.1.4 Partes de la aplicación

A continuación se procede a explicar las distintas partes de la arquitectura del frontend de la aplicación.

- **components**: En esta carpeta encontramos los componentes que serán usados en la aplicación, como el calendario para los usuarios, el calendario para los locales, la barra lateral y el pop-up para introducir un nuevo evento. Además de la Vuex Store. La Vuex Store es una biblioteca muy popular en Vue .js que sirve para gestionar el estado de la aplicación y almacenar datos, la implementación de esta herramienta será explicada más adelante.

- **views**: En esta carpeta se encuentran las vistas de nuestra aplicación. Las vistas también son componentes, pero estas normalmente son usadas para la navegación usando enrutamiento con "Vue Router". Realmente no se ha usado esta biblioteca, pero se ha usado esta arquitectura para comodidad y estructura de la aplicación, separando los componentes principales de la aplicación, los cuales implementan otros componentes, de los componentes más pequeños y aislados.

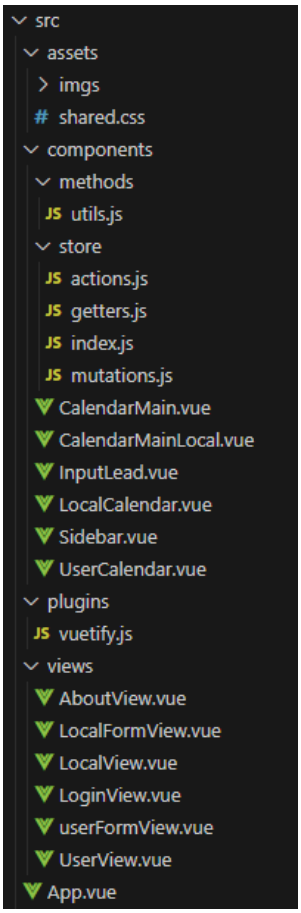
- **utils**: (shared) Esta carpeta solo contiene un script el cual implementa funciones que son reutilizadas a lo largo de la aplicación. Algunas de estas funciones tratan de cálculos matemáticos, manipulación u obtención de fechas o funciones de utilidad.

- **plugins**: Esta carpeta simplemente implementa la inicialización de Vuetify para la autentificación. La implementación de la herramienta de Vuetify será explicado más adelante.

4.6.1.5 Estado de la aplicación

El estado de la aplicación ha sido gestionado haciendo uso de la popular librería de Vue "Vuex". Como se ha explicado anteriormente, Vuex es una librería que sirve

Figura 32: Archivos de la aplicación



para mantener el estado de la aplicación como si se tratara de un almacén (Store) centralizado, de una forma similar a Redux [27] e inspirándose en patrones como Flux [28] o The Elm Architecture [29]. Vuex ofrece una solución para gestionar el estado y datos de la aplicación permitiendo una comunicación fluida entre distintos componentes (Ver figura 33).

EL patrón de Vuex se compone de tres partes principales:

- **State**(Estado): El estado de la aplicación es un objeto plano, el cual contiene los datos de la aplicación. Este estado funciona como una única fuente de verdad y solamente se puede modificar mediante mutaciones y no desde los componentes directamente, esto sirve para garantizar que la información en el estado sea compartida por todos los componentes.

  En la aplicación se ha almacenado toda la información relevante, proveniente del backend. Del estado de la aplicación 6 se deben destacar algunos datos como el array de objetos "leads", el cual almacena todos los eventos del calendario actualmente cargados en la aplicación, estos eventos corresponden con el modelo de la tabla "leads" 23. En el estado también se almacena la información del usuario, (incluyendo su id, nombre, datos personales y token de autenticación), un array de objetos "timeRanges"(cada uno de estos objetos contiene toda la información disponible sobre los tiempos disponibles en un día de un local de manera similar al modelo de la tabla "leadsDay" 24) y otro array de objetos "locals"(que contienen la información sobre los locales cargados para los usuarios

normales).

```
const state = {
  leads: [],
locals : [],
userInfo : {
//[...]
},
timeRanges: [],
    // [...]
```

Código 6: Estado de la aplicación

- **Mutations**(Mutaciones) Las mutaciones son funciones encargadas de cambiar el estado de la aplicación de una manera síncrona. Siguiendo este patrón, el estado sólo debe ser cambiado por medio de estas funciones, cambiando el estado de forma controlada y predecible, para que todos los distintos componentes mantengan el mismo estado. En la aplicación la mayoría de mutaciones son llamadas por las "Acciones", pero las mutaciones también son las encargadas de dar la forma correcta y modificar distintos valores sobre la información que proviene del backend. Por ejemplo, modificando los eventos admitidos y denegados para ser mostrados con un color distinto en el calendario.
- **Actions**(Acciones) Las acciones son las funciones encargadas de realizar funciones asíncronas y realizar llamadas a la API. Estas funciones son invocadas por los diversos componentes y estas, a su vez, llaman a las mutaciones. Además de llamar a la API, algunas de estas funciones tienen una lógica más avanzadas. La función que se encarga de obtener los eventos de los usuarios locales implementa paginación, siendo llamada por el calendario cada vez que el usuario cambia de vista, calculando el primer lunes y el último domingo visibles en la vista, y pidiendo los eventos a la API hasta obtener todos los eventos visibles. Todas las llamadas a la API añaden el token autoritativo, obtenido en el login por AWS Cognito 4.5.4 a la cabecera de la llamada para ser autorizadas en el backend.

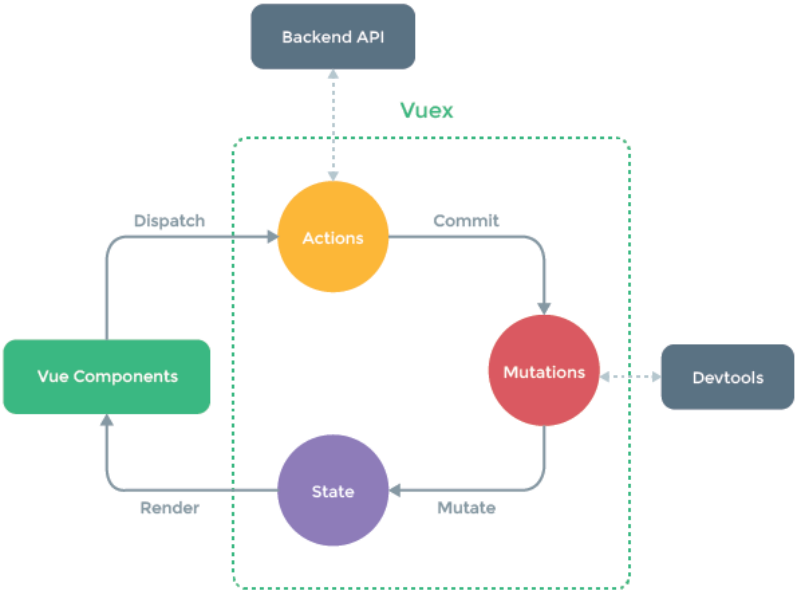
4.6.1.6 Paquetes externos usados en la aplicación

Se ha utilizado la librería de Amplify para Vue para la implementación de la autenticación en el frontend. Esta librería ha aportado una interfaz de usuario estándar para el inicio de sesión y el registro de los usuarios, la cual ha sido modificada para incluir la información necesaria de la aplicación (Ver figura 26 y código 7).

Esta librería también ha facilitado el desarrollo de la comunicación con el servicio de autenticación de AWS Cognito creando los archivos de sincronización necesarios. En el código 8 se puede observar la configuración creada por AWS Amplify para la sincronización y comunicación con AWS Cognito.

Se ha utilizado la librería "vue-cal"4.2.7 como base para el desarrollo de la interfaz del calendario. Esta librería ha aportado una sólida interfaz gráfica para el calendario

Figura 33: Estructura de Vuex



y sus eventos.

Por último, se ha utilizado la librería "Vuetify"[30] para desarrollar un estilo visual agradable y coherente (RNF2.2.) más rápidamente aprovechando sus componentes.

```
      const settings = {
        timePeriod: sTime,
        pauseStart: sBreak,
        close: ending,
        open: starting,
        pauseStop: sResume,
        maxPers: maxPers
      }
      const customSettings = JSON.stringify(settings)
      return Auth.signUp({
        username,
        password,
        attributes: {
          name: attributes.name,
          family_name: attributes.family_name,
          'custom:isLocal': isLocal,
          'custom:city': city,
          'custom:type': type,
          'custom:settings': customSettings
        },
        autoSignIn: {
          enabled: true,
        },
      });
    // [...]
```

Código 7: Datos requeridos para la autenticación

```
// WARNING: DO NOT EDIT. This file is automatically generated by AWS
    Amplify. It will be overwritten.
const awsmobile = {
    "aws_project_region": "us-east-1",
    "aws_cognito_identity_pool_id": "********",
    "aws_cognito_region": "us-east-1",
    "aws_user_pools_id": "********",
    "aws_user_pools_web_client_id": "********",
    "aws_cognito_username_attributes": [
        "EMAIL"
    ],
    "aws_cognito_social_providers": [],
    "aws_cognito_mfa_configuration": "OFF",
    "aws_cognito_password_protection_settings": {
    //[...]
    },
    // [...]
```

Código 8: Datos de configuración de AWS Cognito creados por AWS Amplify

4.7. Retos técnicos

A continuación se describirán las partes de la aplicación que han supuesto mayor complejidad en el desarrollo.

4.7.1. Investigación AWS y arquitectura

El proceso de desarrollo del backend mediante los servicios de AWS fue un desarrollo que involucró un largo periodo de aprendizaje y mejora que requirió mucha dedicación.

En primer lugar el uso de AWS Lambda requirió del aprendizaje de los conceptos y patrones de diseño relacionados con estas funciones sin servidor. También cabe destacar la integración de las Lambdas con los demás servicios de AWS y la necesidad de administración los permisos para esta integración, siguiendo el principio de mínimo privilegio. Fue especialmente tedioso el aprendizaje de las funciones para la modificación de elementos de la base de datos DynamoDB. Algunas de estas funciones también fueron relativamente complejas, en especial la función encargada de la paginación, la cual requirió aprender el funcionamiento del proceso de paginación, su adaptación particular al sistema por días y mucho tiempo en testing y comprobaciones.

La configuración y gestión de API Gateway también presentó dificultades, requiriendo el aprendizaje del funcionamiento de este, su vinculación con las Lambdas y el uso de los métodos de autenticación basada en tokens y Lambda Authorizers.

En el caso de S3 se investigó la mejor manera de configurar el bucket para poder alojar públicamente la página web y su vinculación con CloudFront para una distribución de contenido eficiente de forma segura.

4.7.2. Desarrollo del Frontend

Aunque el desarrollo del frontend no requirió de tanto aprendizaje e investigación como el backend, si que requirió de abundante tiempo y dedicación.

La tardanza en este desarrollo fue debida principalmente a la variabilidad entre los diferentes tipos de usuarios, la implementación del estado de la aplicación, el cual ha de manejar de manera correcta y eficiente grandes cantidades de datos, y la integración y aprendizaje de la librería usada como base en el desarrollo del calendario 4.2.7 .

Entre los apartados que más dificultad presentaron en el frontend destaca la implementación de la autenticación mediante la librería de AWS Amplify para Vue 3 y su conexión con el backend. Esto es debido sobre todo a que la implementación de los campos personalizados que fueron añadidos (Ver códigos 8 y 7), resultó en un proceso delicado y minucioso, presentando varios problemas y la necesidad de numerosos intentos fallidos de implementación.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

Este proyecto me ha resultado tanto a nivel personal, como sobre todo, a nivel profesional, un gran reto, al que he tenido que dedicar mucho tiempo y dedicación. Sin embargo, gracias a este proyecto, he aprendido mucho sobre diferentes tecnologías y metodologías de desarrollo, lo cual ha ampliado mis conocimientos y habilidades en el ámbito profesional. He adquirido experiencia en el uso de tecnologías como AWS (Amazon Web Services), Vue.js y metodologías ágiles como SCRUM.

Además, he tenido la oportunidad de enfrentarme a diversos desafíos técnicos, como la integración de diferentes servicios de AWS, el diseño de una arquitectura escalable y segura, y la implementación de funcionalidades complejas. Estos desafíos me han permitido desarrollar mi capacidad de resolución de problemas y mejorar mis habilidades de programación.

Estoy orgulloso del trabajo realizado y estoy seguro de que los aprendizajes adquiridos me serán de gran utilidad en mi carrera profesional.

5.1.1. Logros alcanzados

Después de varios meses de desarrollo, con gran satisfacción, puedo concluir que el proyecto ha alcanzado el nivel de madurez necesario para afirmar que cumple con los objetivos establecidos.

En este apartado se va a evaluar el grado de cumplimiento de los objetivos propuestos.

Los objetivos de utilidad propuestos han sido en su mayoría superados. Existen dos tipos de usuarios, ambos con funcionalidades específicas. Estos usuarios se pueden relacionar entre sí mediante eventos definidos en el tiempo. Estos eventos vinculantes son creados o propuestos por los usuarios normales, los cuales pueden filtrar por tipo de servicio y elegir una hora disponible para reservar una cita. Los usuarios también pueden tener eventos propios sin relacionarse con otro usuario. Estos eventos propios pueden tener su título, tipo, descripción y horario personalizado.

Los usuarios locales pueden aceptar o rechazar los eventos propuestos por los usuarios normales. Estos eventos propuestos por los usuarios normales a los usuarios locales son creados en base a las "posibles citas", estas posibles citas son, a su vez, creadas a partir del horario definido por los locales y el número de personas que pueden aceptar por cita. Estos datos, al igual que otros como el nombre, correo, teléfono ... Son ajustables tanto para los usuarios locales como para los usuarios normales.

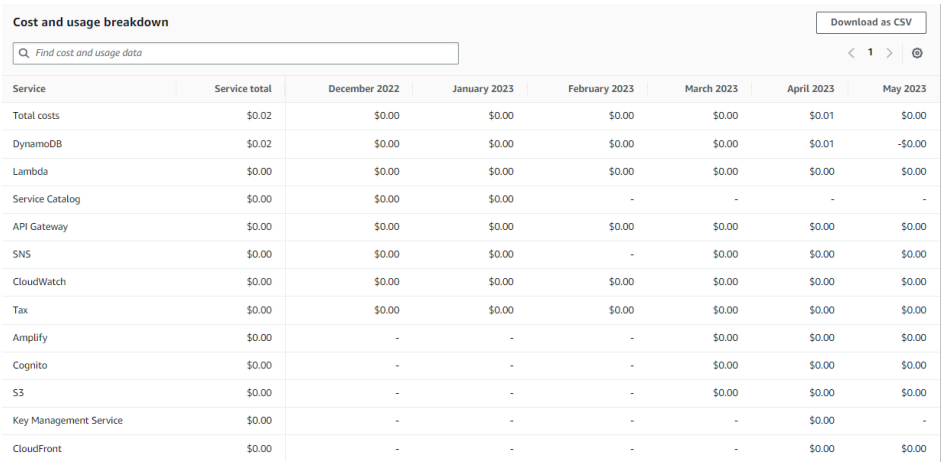
En cuanto a los objetivos técnicos, se investigó en profundidad las posibilidades y servicios útiles que ofrece AWS para la creación de una aplicación de estas características 4.7.1. Gracias a los frutos de esta investigación, se logró desarrollar la aplicación usando tecnologías cloud punteras.

La arquitectura de la aplicación se ha logrado crear para poder soportar teóricamente un gran volumen de usuarios, poniendo especial énfasis en la escalabilidad de los servicios (Ver 4.5.3, 4.5.2, 4.5.5).

El proyecto ha implementado una seguridad robusta mediante mecanismos de autenticación y autorización usando AWS Cognito y Lambda Authorizers.

El coste total del proyecto con AWS ha sido de dos céntimos de euro (Ver figura 34), debido a que este coste ha sido ínfimo, se considera que se ha logrado el coste cero mediante el uso de la "Free Tier"de AWS. Sin embargo, debido a la supuesta evolución en el número de usuarios y su correspondencia directa con el uso de recursos, lo cual afectaría al coste, sería interesante estudiar este aspecto (posiblemente el coste sea debido a las numerosas pruebas hechas en el proyecto) para intentar optimizarlo.

Figura 34: Coste de los servicios de AWS utilizados



Cost and usage breakdown

Download as CSV

Find cost and usage data

1

Service	Service total	December 2022	January 2023	February 2023	March 2023	April 2023	May 2023
Total costs	$0.02	$0.00	$0.00	$0.00	$0.00	$0.01	$0.00
DynamoDB	$0.02	$0.00	$0.00	$0.00	$0.00	$0.01	-$0.00
Lambda	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
Service Catalog	$0.00	$0.00	$0.00	-	-	-	-
API Gateway	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
SNS	$0.00	$0.00	$0.00	-	$0.00	$0.00	$0.00
CloudWatch	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
Tax	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
Amplify	$0.00	-	-	-	$0.00	$0.00	$0.00
Cognito	$0.00	-	-	-	$0.00	$0.00	$0.00
S3	$0.00	-	-	-	$0.00	$0.00	$0.00
Key Management Service	$0.00	-	-	-	-	$0.00	-
CloudFront	$0.00	-	-	-	-	$0.00	$0.00

5.2. Trabajos futuros

La aplicación desarrollada es un proyecto muy ambicioso que necesitaría de abundantes meses o incluso años de desarrollo para ser completada. Este proyecto solamente ha entablado unas bases sólidas desde las cuales se puede seguir desarrollando la aplicación, sirviendo a su vez, como prueba de concepto de todo lo que podría llegar a ser.

Ya que la finalidad de este proyecto es la de abarcar la funcionalidad tanto de calendarios de uso personal, calendarios de uso profesionales y calendarios de uso con terceros, la personalización y posibles características que se pueden incluir son muy numerosas. Algunas de las características más relevantes que podrían mejorar la aplicación son las siguientes:

- **Filtrado por ciudad o cercanía** El filtrado de locales por ciudad o cercanía sería un elemento fundamental para el uso de la aplicación al reservar una cita con un local. Esta característica estaba planteada para llevarse acabo y ya almacena la información de residencia de los usuarios, pero no se ha llegado a finalizar debido a falta de tiempo.

- **Añadir tipos de citas** Cuando un usuario crea una cita con un local, este no puede definir qué tipo de servicio quiere adquirir, aunque esto no sea algo importante para algunos tipos de servicios, como puede ser un restaurante, sí que lo es para otros, como puede ser una peluquería. Esta característica también estaba planteada en un inicio, pero no se llegó a desarrollar por falta de tiempo.
- **Creación de eventos entre usuarios normales** La posibilidad de crear un evento con otro usuario y que este lo acepte o no, al igual que se puede con los usuarios locales. Esto nos llevaría al siguiente punto.
- **Amigos** Para la creación de eventos entre usuarios sería conveniente permitir las relaciones entre usuarios, enviando peticiones de amistad como en aplicaciones de redes sociales.
- **Chat** Creación de un chat, la creación de un chat permitiría a un usuario preguntar dudas o al local comunicar a los usuarios que hay un retraso puntual. Esta parte del chat sería interesante plantearla con tecnología WebSocket, aunque esto podría ser complicado teniendo en cuenta la arquitectura Serverless de la aplicación.
- **Otros** Las posibles características para mejorar la aplicación son tantas, que podrían ir desde una simple mejora estética de esta, hasta el desarrollo nativo para dispositivos móviles, o la integración con otros calendarios como Google Calendar.

Glosario

Arranque en frío Cuando un equipo, en este caso una función Lambda, se levanta, primero prepara un entorno de ejecución. En el caso de AWS Lambda, descargando el código de la función almacenado en S3 y después creando el entorno con la memoria, el tiempo de ejecución y la configuración definida. Este .ªrranque en frío"para una Lambda, no suele repetirse en un largo periodo de tiempo ya que el entorno se mantiene levantado por un tiempo. Si otra petición llega a la misma Lambda y el entorno ya está levantado, el servicio lo reusará.. 27

Bucket Contenedor virtual para objetos almacenados en AWS S3.. 22

DDoS Ataque de denegación de servicio, es un ataque que trata de colapsar un sitio o recurso web con la intención de colapsarlo. . 13

Lambda Authorizer Función de AWS API Gateway que usa una función Lambda para controlar el acceso a la API.. 14

MFA Autenticación multi-factor, es un método de control de acceso en el que para que un usuario pueda iniciar sesión, debe presentar dos o más verificaciones, como por ejemplo usuario y contraseña, y confirmación por correo electrónico.. 14, 33

Microservicios Enfoque arquitectónico en el que el software está compuesto por pequeños servicios independientes, los cuales son ejecutados en su propio proceso.. 20

Políticas de acceso Objeto de AWS que asocia una identidad o recurso definiendo sus permisos, estos permisos determinan si las solicitudes de estas identidades o recursos son permitidas o denegadas.. 14

Roles Identidad de AWS IAM con permisos específicos.. 14

Serverless Modelo de ejecución en la nube, en el que el proveedor de los servicios se encarga de la administración y levanta los servicios bajo demanda.. 3, 13

SPA Single Page Application, es una aplicación web donde sólo exise una página, con el propósito de una experiencia más fluida. Este tipo de páginas reescribe dinámicamente la misma página en vez de cargar nuevas páginas.. 14

User Groups Collección de usuarios IAM de AWS, permite especificar servicios a un grupo de usuarios.. 14

Referencias

[1] AWS, “Aws free tier.” [Online]. Available: https://aws.amazon.com/free/

[2] ——, “Aws lambda.” [Online]. Available: https://aws.amazon.com/lambda/

[3] S. Newman, *Building Microservices, 2nd Edition*, Aug. 2021. [Online]. Available: https://www.amazon.es/Building-Microservices-Sam-Newman/dp/1491950358

[4] AWS, “Aws api gateway.” [Online]. Available: https://aws.amazon.com/apigateway/

[5] ——, “Aws s3.” [Online]. Available: https://aws.amazon.com/cognito/

[6] ——, “Aws dynamodb.” [Online]. Available: https://aws.amazon.com/dynamodb/

[7] ——, “Aws s3.” [Online]. Available: https://aws.amazon.com/s3/

[8] ——, “Aws cloudfront.” [Online]. Available: https://aws.amazon.com/cloudfront/

[9] ——, “Aws cloudwatch.” [Online]. Available: https://aws.amazon.com/aws_cloudwatch/

[10] ——, “Vue 3.” [Online]. Available: https://aws.amazon.com/iam/

[11] ——, “Aws api amplify.” [Online]. Available: https://aws.amazon.com/amplify/

[12] E. You, “Vue 3.” [Online]. Available: https://vuejs.org/

[13] W. W. W. Consortium, “Html.” [Online]. Available: https://html.spec.whatwg.org/multipage/

[14] Mozilla, “Css.” [Online]. Available: https://developer.mozilla.org/es/docs/Web/CSS

[15] N. developers, “Node.js.” [Online]. Available: https://nodejs.org/en

[16] P. developers, “Postman.” [Online]. Available: https://www.postman.com/

[17] Git, “Git.” [Online]. Available: https://git-scm.com/

[18] I. GitHub, “Github.” [Online]. Available: https://github.com/

[19] npmjs, “Npm.” [Online]. Available: https://www.npmjs.com/

[20] Trello, “Trello.” [Online]. Available: https://trello.com/

[21] Microsoft, “Visual studio code.” [Online]. Available: https://code.visualstudio.com/

[22] A. André, “Vue-cal.” [Online]. Available: https://antoniandre.github.io/vue-cal/S

[23] Vuex, “Vuex.” [Online]. Available: https://vuex.vuejs.org/

[24] AWS, “Aws sdk.” [Online]. Available: https://aws.amazon.com/sdk-for-javascript/

[25] ——, “Aws route 53.” [Online]. Available: https://aws.amazon.com/es/route53/

[26] Vue3, “What’s the difference between vue 2 and vue 3?” [Online]. Available: https://vuejs.org/about/faq.html#what-s-the-difference-between-vue-2-and-vue-3

[27] D. A. y Andrew Clark, “Redux.” [Online]. Available: https://es.redux.js.org/

[28] L. Amadi, “The flux architectural pattern of client-side development.” [Online]. Available: https://medium.com/@w3bh4ck/the-flux-architecture-pattern-for-frontend-development-1f2dae32b789

[29] E. Czaplicki, “The elm architecture.” [Online]. Available: https://guide.elm-lang.org/architecture/

[30] Vuetify, “Vuetify.” [Online]. Available: https://vuetifyjs.com/en/