

Sistemas y tecnologías web

Zarahealth

Curso 2019-2020

Pedro Malo - 736201

Javier Martínez - 737910

Daniel Naval - 739274

Víctor Rodríguez - 741414

Índice

Resumen del proyecto	4
Arquitectura de alto nivel (modelo de bloques de alto nivel, cómo se conectan, etc.)	5
Modelo de datos	5
Peticiones a Back-End	7
GraphQL	7
OAuth 2.0	7
Imágenes	7
Implementación	8
OAuth 2.0	8
Passport	9
GraphQL	9
MongoDB	10
React.js	10
Apollo Client	10
React Hooks	10
Material-UI	11
Modelo de navegación y analíticas	11
Dashboard	11
Polen	14
Agua	15
Aire	17
Feed	19
Ajustes usuario	21
Ajustes administrador	24
Despliegue del sistema	27
Validación	28
Front-End	28
Back-End	29
Problemas encontrados durante el desarrollo	30
Análisis de problemas potenciales	31
Distribución de tiempo	32

Gantt	32
Tiempo invertido por alumno	32
Conclusiones	32
Valoración personal de cada miembro	33
Víctor	33
Daniel	33
Javier	34
Pedro	34

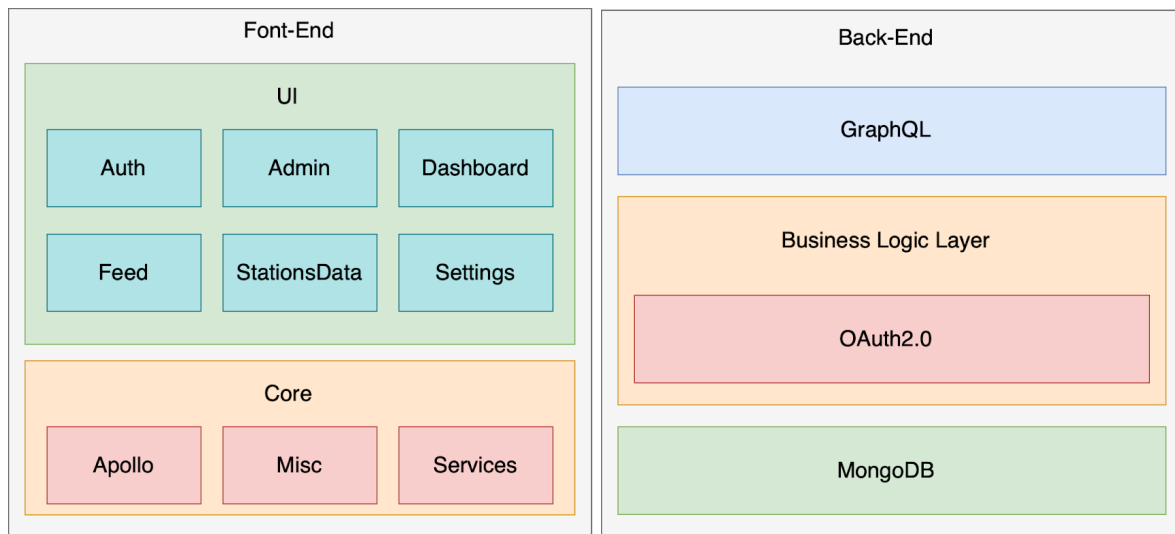
Resumen del proyecto

El proyecto ha consistido en la creación de una aplicación web, en la cual se pueda ver en tiempo real los datos de la calidad del agua y del aire en las diferentes estaciones, además de los niveles de polen de la ciudad de Zaragoza. Para que el usuario disponga de la información sin tener que acceder a los datos de cada estación, se ha diseñado un sistema de avisos el cual si el administrador de la aplicación detecta algún parámetro fuera de lo común avisa a todos los usuarios de la aplicación web. Además, la aplicación permite establecer una serie de umbrales para los niveles de polen y de aire de manera que si se sobrepasa alguno de estos niveles, cuando el usuario inicia sesión le avise de si algún umbral de los que ha establecido se ha superado.

Todos los datos se han extraído de los diferentes servicios de API pública que ofrece el ayuntamiento de Zaragoza para cada uno de los tipos de datos que se proveen.

Para la realización de este proyecto se ha utilizado **React** para diseñar la aplicación web, **Node.js** para el servidor y dentro de este servidor, **GraphQL** como servidor API y **OAuth 2.0** como servidor de autenticación. Para la base de datos se ha utilizado **MongoDB**.

Arquitectura de alto nivel (modelo de bloques de alto nivel, cómo se conectan, etc.)



Modelo de datos

Settings: Se guarda un booleano para air, pollen y water que permite determinar si mostrar datos de cada respectiva categoría.

Broadcasts: Se almacenan todas las notificaciones y avisos que se lanzan desde el panel de administrador para ser posteriormente consultadas y recuperadas.

Activities: Se registra toda actividad que se realiza en la aplicación y se considera relevante, por cada tipo de actividad se almacena dirección IP, agente de usuario y usuario implicado en proceso. Este registro es utilizado para proporcionar métricas del sistema y en un futuro evaluar posibles violaciones de acceso a la cuenta de un usuario.

Clients: Esta colección está diseñada para guardar los clientes web que permiten obtener los tokens para empezar a funcionar con la aplicación. Para esto se necesita guardar el id del cliente, su secreto, un id, las licencias para poder utilizar los diferentes tipos de protocolos en OAuth 2.0 y las uris de redirección, que para este entorno no son requeridas.

Tokens: Esta colección guarda la información necesaria para guardar los diferentes tokens del usuario y que estos puedan acceder siguiendo el estándar OAuth 2.0. Para esto se guarda el accessToken, refreshToken,

cuando expiran ambos tokens y el cliente y el usuario al que pertenece el token.

Feeds: En esta colección se guarda información respectiva a un feed como su fecha de creación, autor, cuerpo, imágenes y título. Una vez se ha creado el feed se necesita guardar información de los comentarios que realizan los diferentes usuarios sobre los feeds así como de su opinión.

Users: Esta colección contiene información respectiva al usuario como la fecha de creación, email, nombre, contraseña cifrada e imagen. También se guarda información de las preferencias del usuario, contiene si el usuario quiere descargar datos en formato csv, los umbrales para el polen, sus estaciones preferidas de aire y agua, y en el caso del aire los umbrales para esta estación. También se contiene información interna útil para el transcurso de la aplicación como el estado de la cuenta (indica si este usuario puede o no acceder a la cuenta), si el usuario es admin y si la cuenta ha sido creada a través de google.

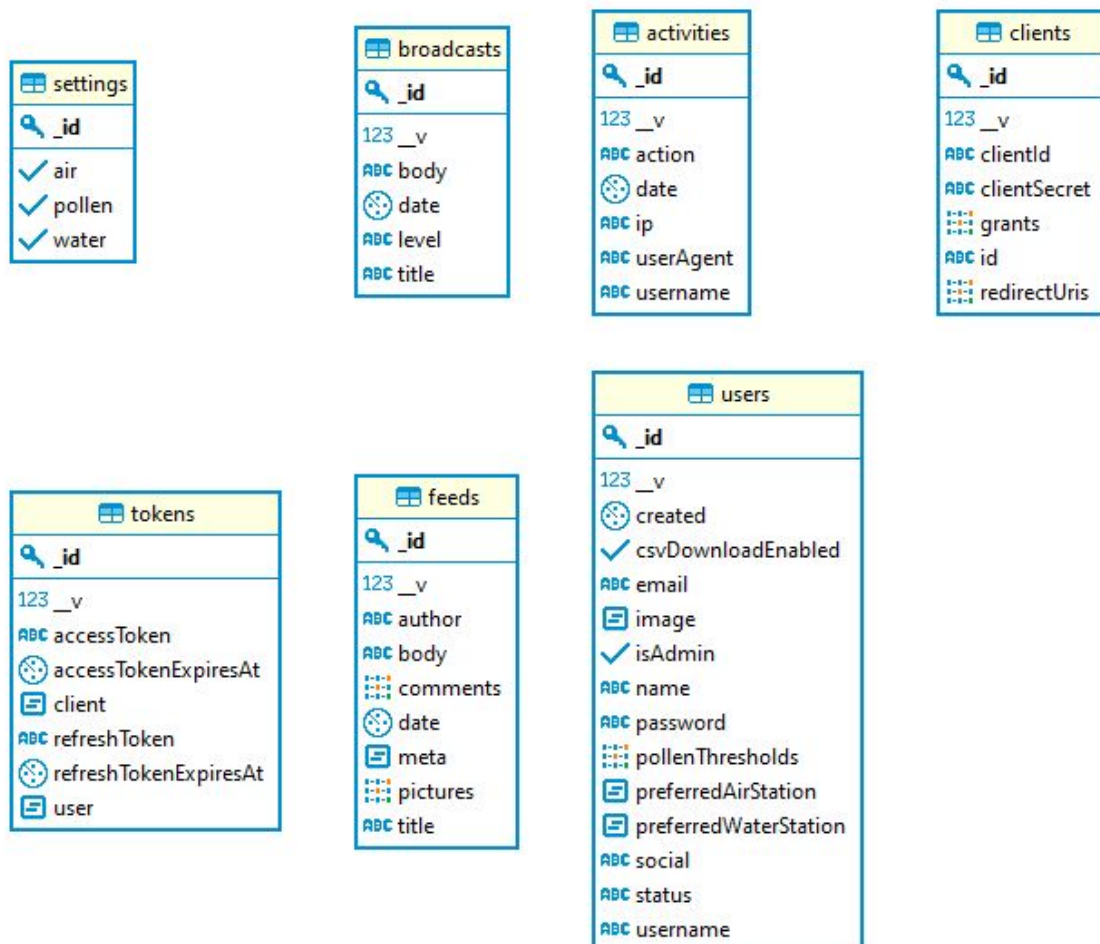


Figura 1: Modelo de datos.

Peticiones a Back-End

GraphQL

GraphQL es autodocumentado, es decir, genera una documentación a través de la definición de las diferentes funciones a las que se puede realizar una petición, todo ello definido en el esquema de GraphQL. Esta documentación contiene las diferentes **queries** y **mutations** que un usuario puede realizar junto con las variables que hay que pasarle y su tipo, también indican el tipo que devuelve. Para tipos diferentes a los básicos, la documentación indica también las diferentes variables que contiene. Para facilitar las peticiones se han añadido comentarios para indicar que hace cada función y que es cada variable, esto también se mostrará en la documentación.

Esta documentación se ha subido a <https://aeri.github.io/NIVERSO/>.

OAuth 2.0

El proceso seguido para documentar ha sido una vez el Back-End ha implementado un servidor OAuth 2.0 estable y funcional. Se ha creado la documentación que explica de forma detallada cómo funciona la autenticación explicando paso por paso el método a seguir.

Esta documentación se encuentra detallada en el siguiente endpoint: <https://zgz.docs.apiary.io>.

Imágenes

Se ha incorporado un mecanismo de recuperación de imágenes almacenadas en la base de datos MongoDB transparente a GraphQL y que proporciona mayor versatilidad al Front End a la hora de trabajar con imágenes. Este sistema ya sea a través de un identificador único de imagen o un nombre de usuario es capaz de recuperar de recuperar el archivo guardado en la base en base64 y transformarlo para ser devuelto como una petición multipart estándar.

Dado que esta funcionalidad se encuentra fuera del esquema GraphQL se ha redactado un guión de funcionamiento para trabajar con este endpoint:

https://github.com/aeri/NIVERSO/blob/master/IMAGE_RETRIEVER_API_DOC.md

Implementación

OAuth 2.0

Se ha utilizado un mecanismo de autenticación mediante OAuth 2.0 tanto para un usuario normal como para uno que inicie sesión a través de google.

En el marco de la OAuth se especifican varios tipos de "grant types" para diferentes casos de uso. En esta implementación se ha optado por la incorporación de los siguientes flujos o "grant types" para la autenticación interna:

- **Client Credentials**: es utilizado por los clientes para obtener un token de acceso fuera del contexto de un usuario. De esta manera los clientes pueden acceder a ciertos recursos sin necesidad de acceder a los recursos de un usuario.
- **Password grant**: es una forma de intercambiar las credenciales de un usuario por un token de acceso, este mecanismo se ha implementado para identificar a los usuarios en el contexto de una petición que necesite autorización y autenticación.

Este marco de OAuth 2.0 explicados son únicamente para la autenticación interna, la autenticación con servicios de terceros implica el uso de Password grant y la explicada a continuación:

Para incorporar el inicio de sesión mediante las credenciales de Google es necesario utilizar de forma obligatoria el flujo o "grant_type" **Authorization Code**, el cual es utilizado por clientes confidenciales y públicos para intercambiar un código de autorización por un token de acceso.

El cliente es llevado a un sitio de Google en el que debe dar permiso para que la aplicación recupere sus datos, después el usuario es redirigido al cliente a través de una URL de redireccionamiento, en ese momento la aplicación obtendrá el código de autorización de la URL y lo utilizará para solicitar un token de acceso. Este proceso se lleva a cabo entre Front End y Back End.

Passport

Para el registro de un usuario se han utilizado los módulos de node ***passport-google-token*** y ***passport***.

Estos módulos se han utilizado únicamente en el marco de trabajo de la autenticación con el servicio OAuth 2.0 de Google ya que permite de una forma muy transparente establecer el intercambio de datos con el servidor a partir de un token de acceso.

Una vez que Front End intercambia el código por un token de acceso, este es enviado a Back End, a continuación se verifica el token usando la función `authenticate` de `passport`, si el token es válido se recibirá el usuario como parte de la respuesta de verificación. A continuación, se guardan los datos del usuario en la base de datos, si es que dicho usuario no existía ya.

Por último, se devuelve un access token del sistema de autenticación OAuth 2.0 interno al front-end para poder actuar en nombre de dicho usuario y seguir el ciclo tradicional de un usuario.

GraphQL

GraphQL es un lenguaje de consulta para las API. GraphQL proporciona una descripción completa y comprensible de los datos de su API, da a los clientes el poder de pedir exactamente lo que necesitan y nada más, facilita la evolución de las API a lo largo del tiempo, y permite potentes herramientas de desarrollo.

Se ha configurado GraphQL de modo que antes de realizar cualquier petición primero se comprueba que la petición está autorizada a utilizar las funciones de la API.

Una vez se tiene una configuración inicial de GraphQL cualquier función creada se tiene que añadir al *schema* la definición de la función y declarar en *rootValue* cuál es la función que implementa el comportamiento de la función definida en el schema.

MongoDB

La base de datos se ha implementado con mongo, en el apartado Despliegue del sistema se detalla cuál ha sido la implementación de la base, y por tanto, como se debería desplegar esta para un correcto funcionamiento.

React.js

La implementación del Front End ha sido realizada mediante el *framework* React.js. Las principales ventajas de React son la simplicidad para el manejo del estado local de los componentes implementados y la facilidad para que reusarlos.

En general, React proporciona pocos mecanismos nativos más allá del renderizado básico, por lo que elementos como la gestión de la persistencia o la navegación han tenido que ser tratados de forma independiente al *framework*. En estos casos concreto, la persistencia ha sido manejada mediante el plugin de *NPM* *apollo-cache-persist*, que vuelva en el almacenamiento local el cache de Apollo (fuente del verdad del estado de la aplicación web) y *react-router*, otro paquete de *NPM* para la gestión de la navegación.

Apollo Client

Para la gestión de la interacción entre el Front End y la API expuesta por el Back End en GraphQL, se ha usado Apollo Client, el plugin estándar en la industria del GraphQL. Apollo Client gestiona errores y cacheado de forma nativa, por lo que libera al desarrollador de tareas tediosas.

Por ejemplo, Apollo Client es capaz de detectar que, al hacer una mutación que actualiza alguna entidad ya cargada en una consulta anterior, debe actualizar de forma automática esa entidad (caso típico: dar *like* a un comentario previamente cargado).

Apollo Client permite además guardar datos en su cache de forma manual, actuando entonces como fuente de verdad del estado de la aplicación, de una forma similar al clásico patrón Redux.

React Hooks

Para la gestión del estado local de los componentes (variables de estado con ciclo de vida breve, fuera del alcance del estado global) se ha priorizado el uso de React Hooks, introducidos recientemente.

Su uso permite manejar el estado de los componentes de una forma más rápida y cómoda, pudiendo usarse con componentes funcionales y por tanto evitando el uso de los clásicos y más verbosos componentes basados en clases.

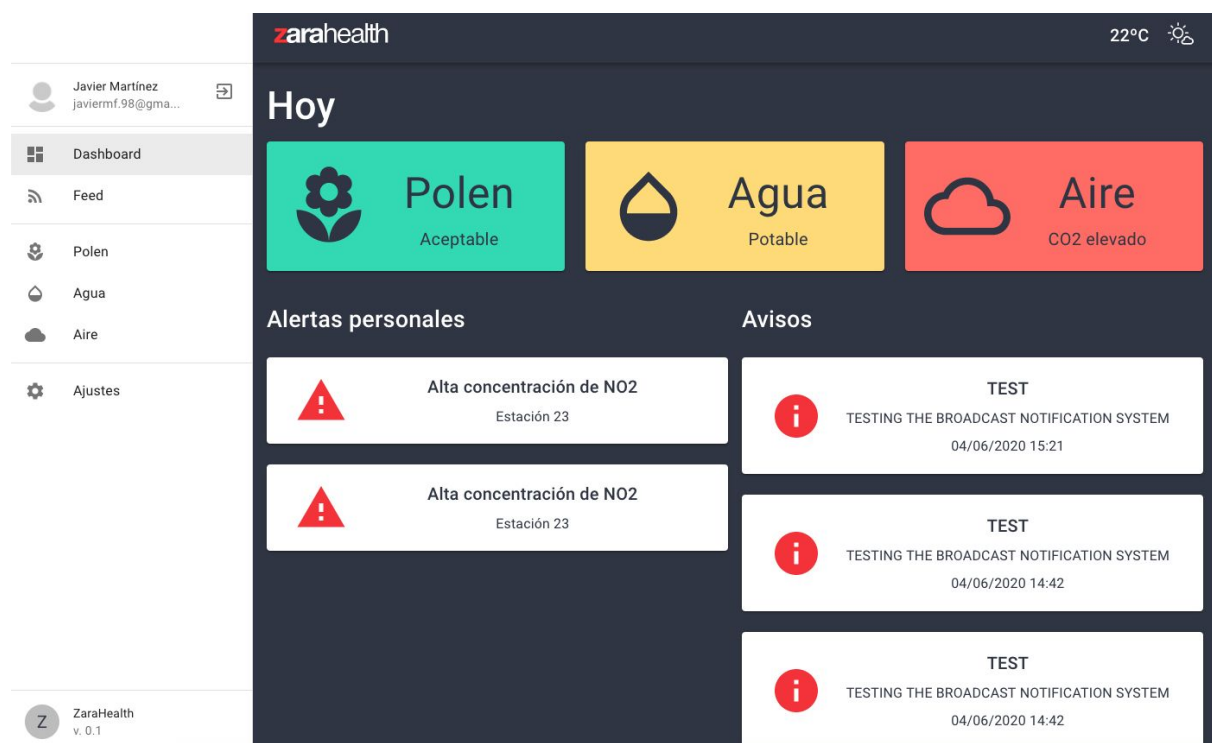
Material-UI

Como librería de estilos se ha empleado Material-UI en lugar de otras opciones clásicas como Bootstrap. Ha sido así por la gran calidad estética de multitud de elementos prediseñados que incluye.

Sin embargo, al margen del aspecto visual, la experiencia de desarrollo no ha sido del todo satisfactoria, ya que es una librería limitada en algunos aspectos, muy verbosa y en general, la documentación es mejorable.

Modelo de navegación y analíticas

Dashboard



Esta vista muestra, en primer lugar, el estado general en que se encuentran los valores de polen, agua y contaminantes del aire. Si el usuario hace clic sobre uno de estos botones podrá ver la información concreta de cada uno de ellos.

Además, en esta vista se muestran las alertas personales de un usuario registrado, que se generan cuando se supera uno de los umbrales establecidos por este. También se muestran los avisos que los administradores del sistema publican, pudiendo ser noticias, advertencias y alertas. El usuario recibirá una notificación si el administrador publica un aviso.

En la parte izquierda se observan un drawer con las diferentes secciones con las que cuenta la aplicación. En el caso de utilizar la aplicación en un dispositivo móvil, el drawer se abrirá al pulsar el icono de la esquina superior izquierda.

En dicho drawer, además de las secciones, el usuario puede iniciar o cerrar sesión, así como registrarse en el sistema si no lo ha hecho. Tras pulsar el botón de iniciar sesión, la aplicación mostrará un diálogo en el formulario de inicio de sesión, permitiendo al usuario iniciar también sesión con Google. En caso de que el usuario desee registrarse en el sistema sin usar los servicios de google, deberá pulsar el botón inferior del diálogo que dice Regístrate.

A

Anónimo

Iniciar sesión

Dashboard

Polen

Agua

Aire

Z

ZaraHealth

v. 0.1

21°C ☀️

Polen

Aceptable

Agua

Potable

Aire

NO2 elevado

Concentración de NO2

Estación 23

☰ zarahealth 21°C ☀️

Hoy

Iniciar sesión

Inicias sesión en ZaraHealth para recibir actualizaciones personalizadas, participar en el feed público y mucho más

Nombre de usuario *

Contraseña *

INICIAR SESIÓN

INICIAR SESIÓN CON GOOGLE

¿Aún no tienes una cuenta? Regístrate

Alertas personales

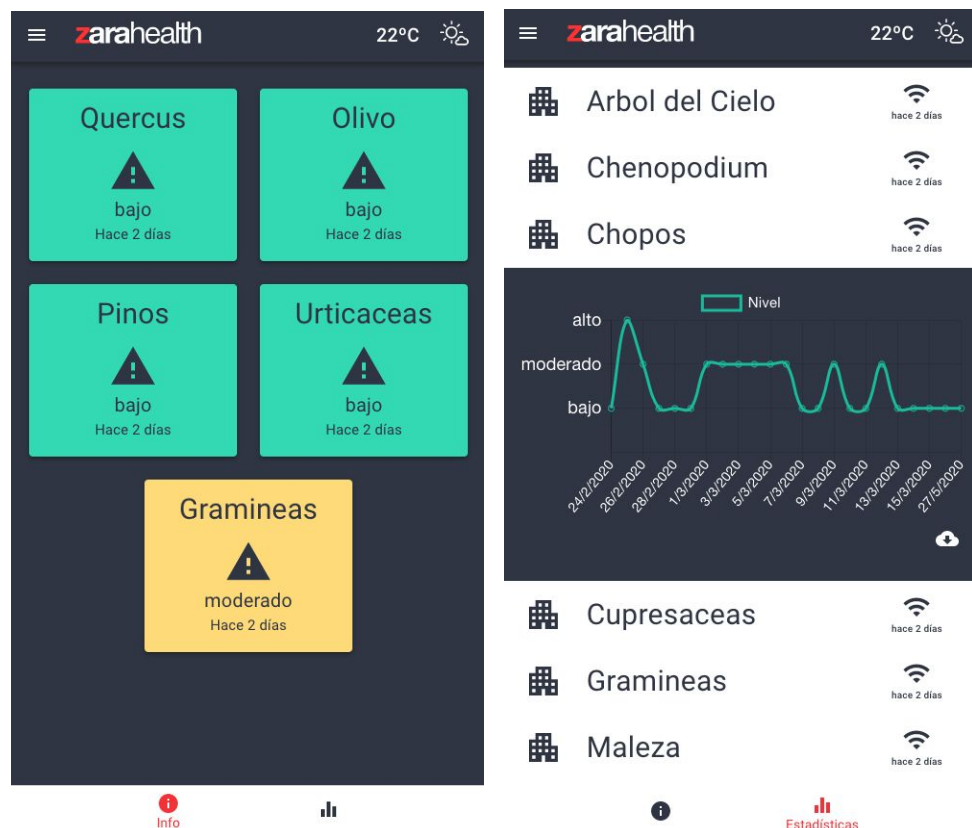
⚠️

Alta concentración de NO2

Estación 23

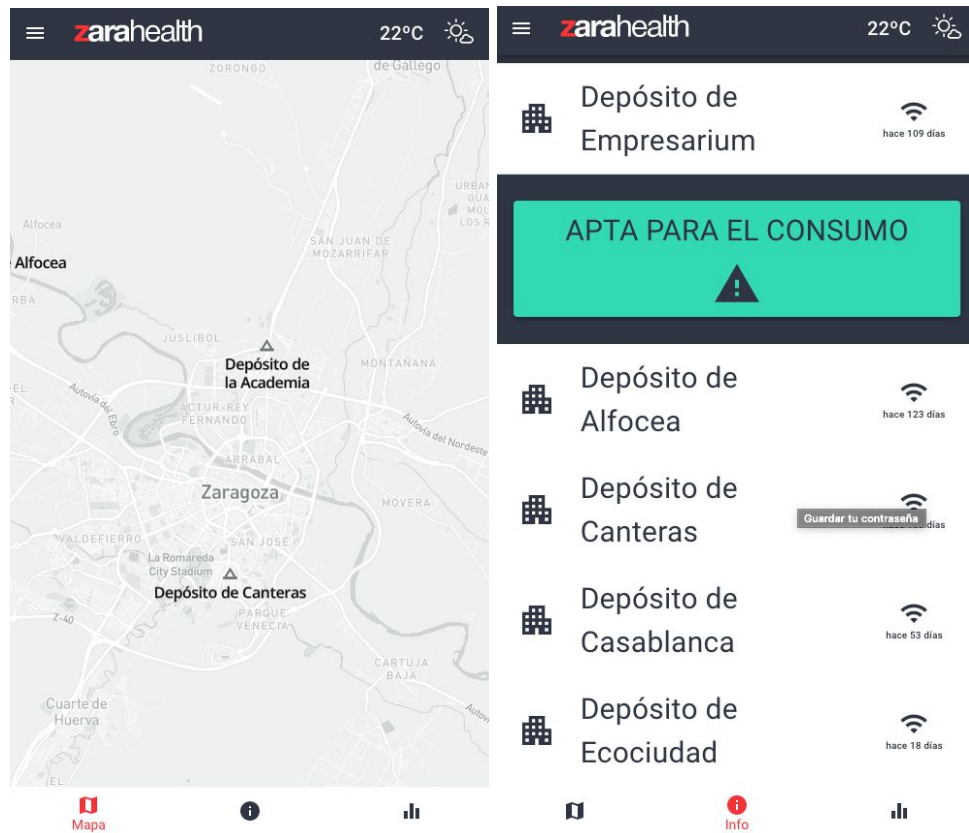
La aplicación ofrece al usuario información sobre los valores del polen, calidad del agua y calidad del aire en Zaragoza. Para acceder a la información de cada uno de ellos, el usuario puede pulsar sobre cada una de las secciones del drawer.

Polen



El usuario, al acceder a la pantalla de información del polen, puede observar 2 secciones. La primera de ellas muestra los datos más actuales de las distintas plantas sobre las que el Ayuntamiento ofrece información. El nivel para cada una de ellas puede ser: bajo, moderado o alto. La segunda sección presenta al usuario gráficas con los datos históricos de cada una de las plantas. Para cada una de las gráficas, si el usuario tiene activada la opción de descarga de datos(pantalla de ajustes), se muestra un botón que permite descargar los datos que muestra la gráfica.

Agua



En la pantalla de información del agua, el usuario puede observar 3 secciones. La primera de ellas muestra un mapa con la situación de cada uno de los depósitos de Zaragoza, pudiendo hacer clic sobre uno de ellos y ver el dato más actual que ofrece el Ayuntamiento sobre si el agua es apta para el consumo en dicho depósito. La siguiente muestra una lista con los depósitos y el dato más actual ofrecido.

La última sección presenta al usuario gráficas con los datos históricos de cada uno de los depósitos. Para cada una de las gráficas, si el usuario tiene activada la opción de descarga de datos, se muestra un botón que permite descargar los datos que muestra la gráfica.



Depósito de
Empresarium

📶
hace 109 días



Depósito de
Alfocea

📶
hace 123 días



Depósito de
Canteras

📶
hace 109 días



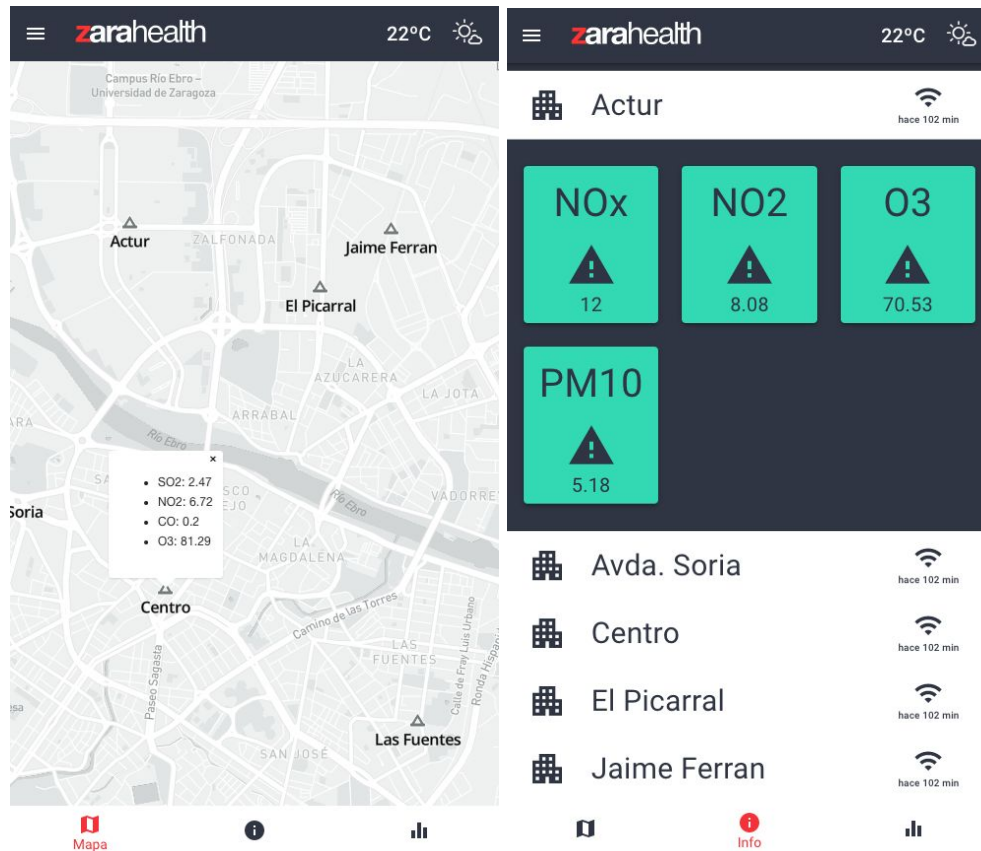
Depósito de
Casablanca

📶
hace 53 días



📊
Estadísticas

Aire



En la pantalla de información del aire, el usuario puede observar también 3 secciones. La primera de ellas muestra un mapa con la situación de cada una de las estaciones de Zaragoza, pudiendo hacer clic sobre una de ellas y ver los dato más actuales que ofrece el Ayuntamiento sobre el nivel de cada contaminante en dicha estación. La siguiente muestra una lista con los estaciones y los dato más actuales ofrecidos.

La última sección presenta al usuario gráficas con los datos históricos de cada una de las estaciones. Para cada una de las gráficas, si el usuario tiene activada la opción de descarga de datos, se muestra un botón que permite descargar los datos que muestra la gráfica. Además, el usuario puede seleccionar el rango de fechas entre las cuales está interesado en ver los datos pulsando sobre el correspondiente botón en cada una de las gráficas.



Avda. Soria

hace 104 min

Centro

hace 104 min

El Picarral

hace 104 min

Jaime Ferran

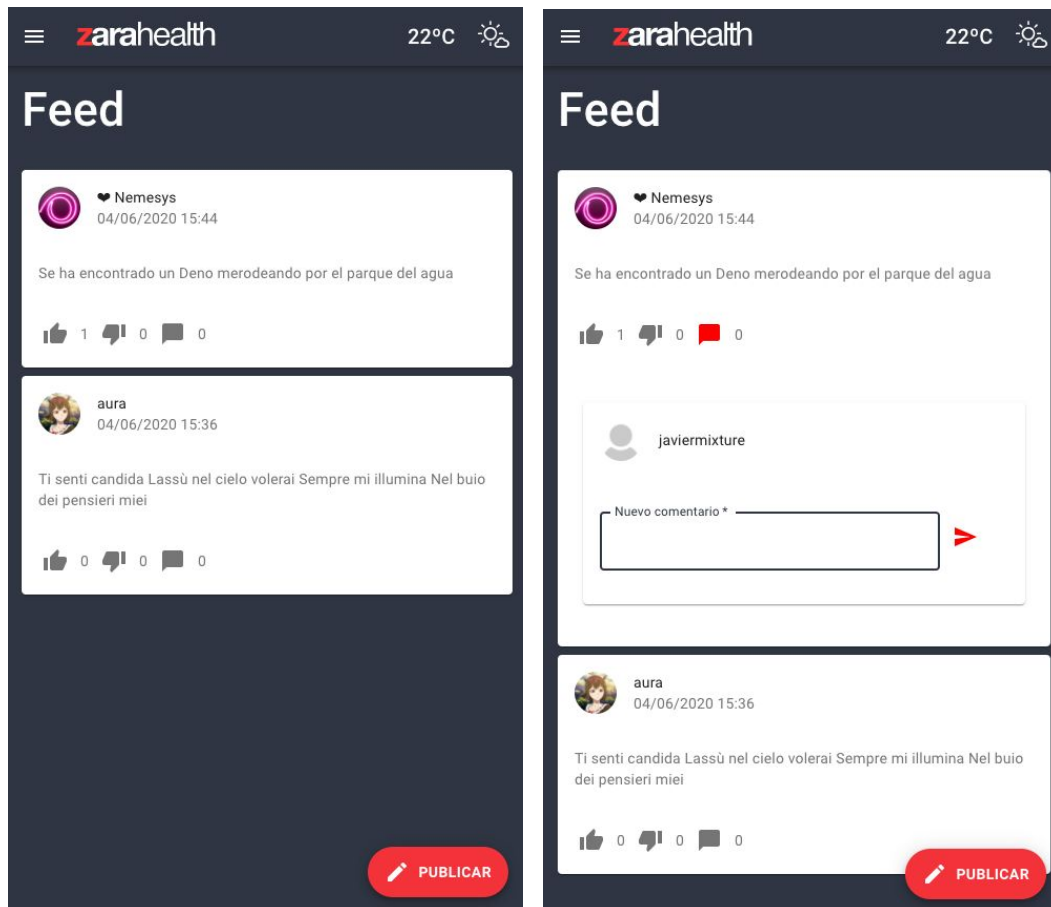
hace 104 min

Las Fuentes

hace 104 min

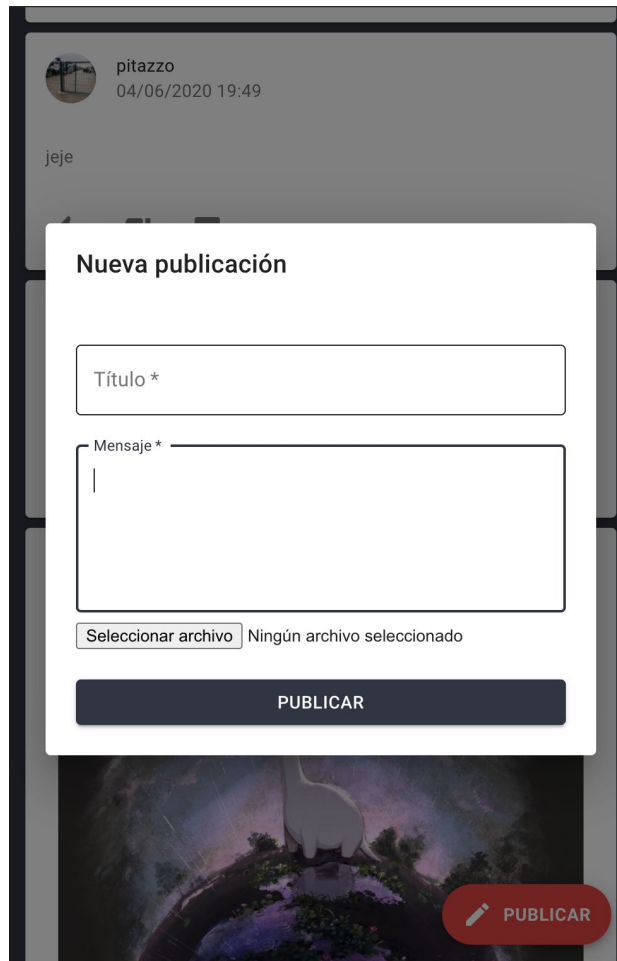
Estadísticas

Feed



Esta vista muestra todos los posts publicados, ordenados de más a menos reciente. Para acceder a esta pantalla el usuario ha tenido antes que iniciar sesión.

Todo usuario registrado en el sistema puede publicar un post. Para ello debe pulsar sobre el botón de publicar situado en la esquina inferior de la pantalla. Tras pulsar el botón, la aplicación le mostrará al usuario un diálogo con el formulario del post. El usuario podrá entonces publicarlo.



El usuario, además, puede indicar si un post le gusta o no, así como publicar un comentario en un post concreto. Para ello debe pulsar sobre los botones situados en la parte inferior de cada uno de los posts.

Ajustes usuario

The screenshot shows the ZaraHealth application interface. At the top, the header includes the 'zarahealth' logo, the temperature '22°C', and a sun icon. Below the header, the main title 'Ajustes' is displayed. A left sidebar contains navigation links: 'Dashboard', 'Feed', 'Polen', 'Agua', 'Aire', and 'Ajustes' (which is highlighted). The 'Polen' section is active, showing a list of plants with dropdown menus for setting alert thresholds. The plants listed are Quercus, Olea_europaea, Pinus, Urticaceae, Poaceae, Platanus, Amaranthaceae, Plantago, Chenopodium, Populus, Cupressaceae, Ailanthus_altissima, Morus_alba, and Bajo. Each plant has a dropdown menu currently set to 'Bajo'. At the bottom left, the ZaraHealth logo and version 'v. 0.1' are visible.

zarahealth 22°C ☀️

Javier Martínez
javiermf.98@gma...

Ajustes

Polen


Umbrales de alerta

Quercus	Olea_europaea
Bajo	
Pinus	Urticaceae
Poaceae	Platanus
Amaranthaceae	Plantago
Chenopodium	Populus
Cupressaceae	Ailanthus_altissima
Morus_alba	
Bajo	

Z ZaraHealth v. 0.1


Esta vista muestra los ajustes de cuenta de un usuario. Para acceder a esta pantalla el usuario ha tenido antes que iniciar sesión y no ser administrador.







En primer lugar, el usuario puede establecer para polen, agua y aire, su estación preferida (en caso de que haya estaciones) y para dicha estación, los umbrales para los distintos elementos, de forma que si dicho umbral se supera en esa estación, el usuario observará una alerta en el dashboard que se lo indique.



Javier Martínez

javiermf.98@gmail...



-  Dashboard
-  Feed
-  Polen
-  Agua
-  Aire
-  Ajustes

Datos personales

Nombre *

Javier Martínez

Email *

javiermf.98@gmail.com

Contraseña nueva

Repetir contraseña

CANCELAR GUARDAR

Imagen de usuario


Seleccionar archivo

nada seleccionado

CANCELAR GUARDAR

Descarga

- ☐

Habilitar descarga CSV
- 

Descargar datos de cuenta

Ajustes administrador

The screenshot displays the ZaraHealth administrator interface. The top header shows the 'zarahealth' logo and the current temperature '22°C' with a sun icon. The left sidebar contains a user profile for 'Javier Martinez' and a list of navigation items: 'Dashboard', 'Feed', 'Polen', 'Agua', 'Aire', and 'Administrador' (highlighted with a gear icon). The main content area is titled 'Estadísticas' and lists the following statistics:

- Usuarios registrados:** 19 usuarios registrados en el sistema
- Usuarios token no expirado:** 19
- Feeds registrados:** 19
- Actividades:** 19

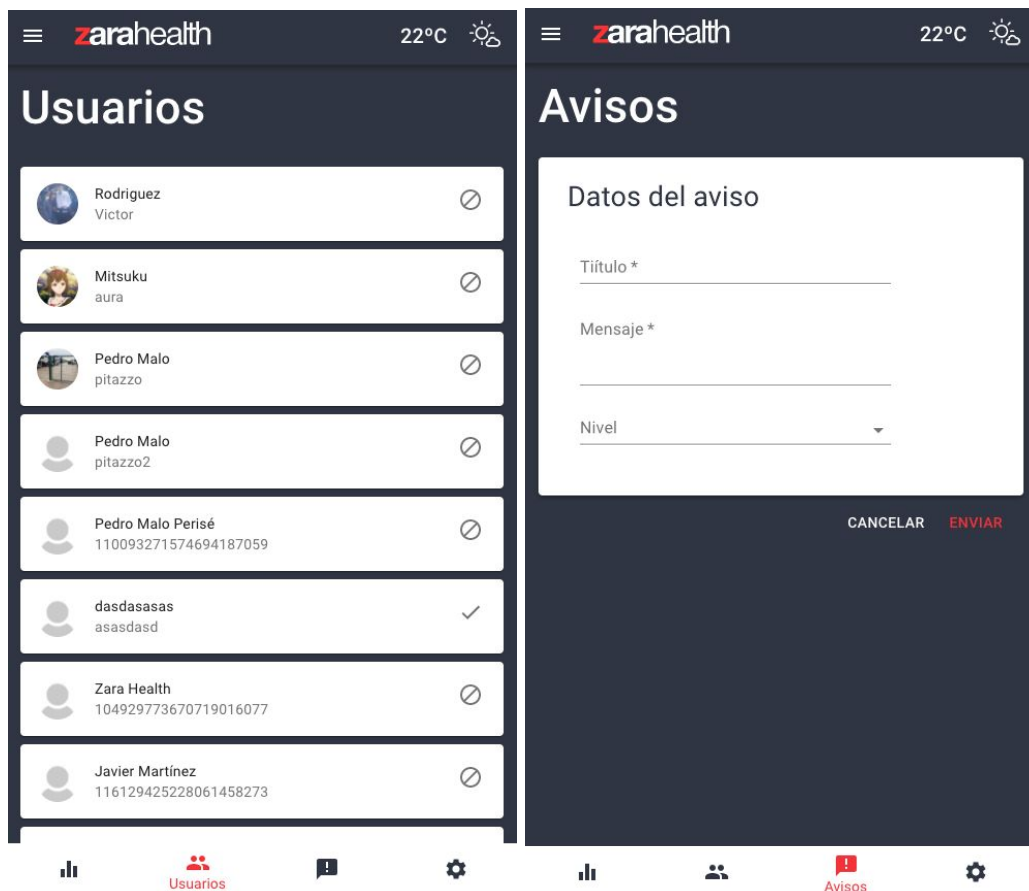
The bottom of the page features a footer with the 'ZaraHealth v. 0.1' logo and a navigation bar with icons for 'Estadísticas', 'Usuarios', 'Feeds', and 'Actividades'.

En caso de que el usuario que ha iniciado sesión sea un administrador, la sección de ajustes cambiará con respecto a los ajustes de un usuario normal.

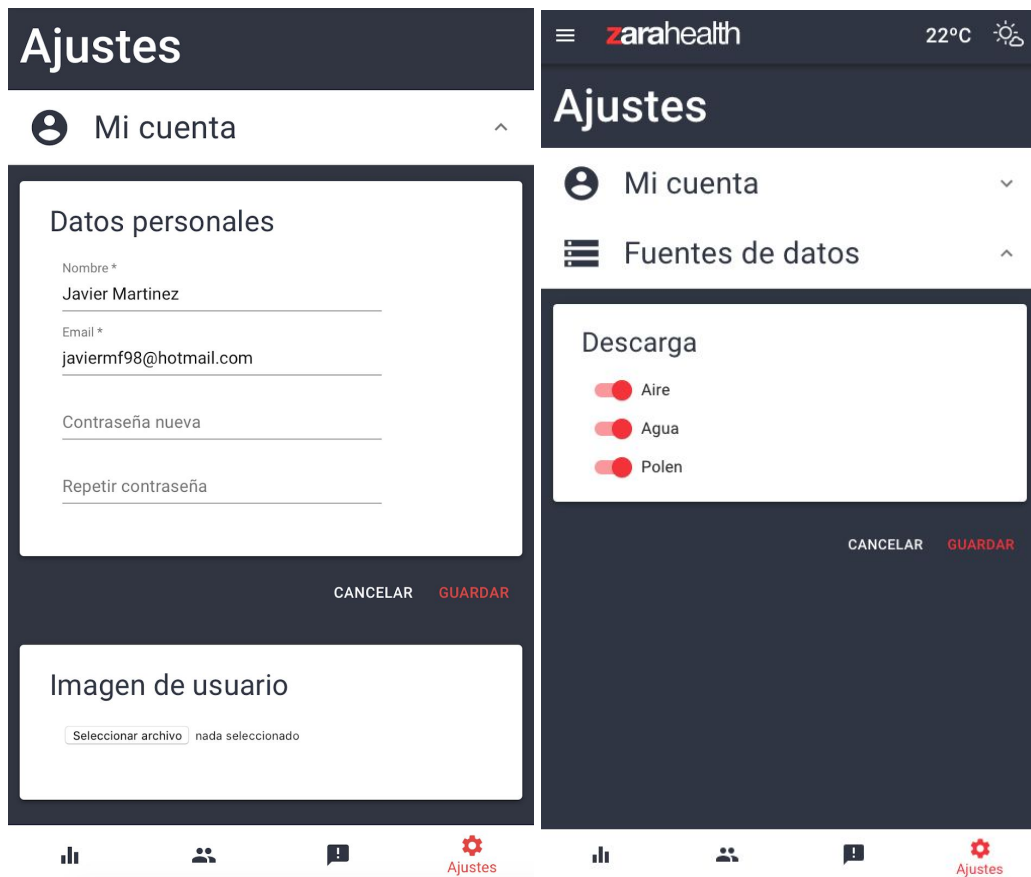
Esta vista muestra, en primer lugar, estadísticas del sistema: número de usuarios registrados, número de usuarios cuyo token no ha expirado, número de feeds registrados y el número de veces que se ha ejecutado cada una de las actividades disponibles.



La siguiente sección a la que el administrador puede acceder es la de Usuarios, que muestra una lista con todos los usuarios registrados en el sistema. El administrador puede bloquear/desbloquear a un usuario, impidiendo/permitiendo el acceso al sistema, pulsando sobre el botón correspondiente en el usuario en cuestión.



La tercera sección a la que el administrador puede acceder es la de Avisos, que permite al administrador publicar una noticia, advertencia o alerta que recibirán como notificación todos los usuarios y podrán observar en el dashboard. Para ello, el administrador cuenta con un formulario para introducir los datos correspondientes.



Por último, el administrador cuenta con una sección de ajustes desde donde puede, por una parte, modificar los datos de su cuenta: nombre, email, contraseña o imagen de perfil. Y por otra parte, seleccionar si los datos de aire, agua o polen son mostrados en la aplicación o no. Si uno de estos no es mostrado, los usuarios verán un aviso en la sección correspondiente que les indicará que los datos no están disponible.

Despliegue del sistema

Para comenzar se debería de desplegar mongo en un servidor web y crear una base de datos donde guardar las colecciones. A continuación se debe crear un usuario que tenga privilegios mínimos en la base de datos creada así como la [inclusión de TLS para cifrar las conexiones con la base](#). En este caso se ha desplegado la base de datos en **Amazon Web Services**. Cabe destacar que se han seguido en todo momento los principios de seguridad y confidencialidad de los datos con mecanismos de autenticación y validación del servidor de la base de datos desde el servidor Node.js.

Posteriormente han de cargarse en la base de datos los clientes y secretos OAuth que serán necesarios para iniciar las comunicaciones mediante este protocolo.

Una vez se tiene la base de datos desplegada y funcionando se debe poner en marcha el servidor web (API) implementado en node.js y desplegarlo en **Heroku**, en este caso se ha utilizado integración continua para el despliegue del servidor web de manera que cuando se hace un nuevo commit a GitHub, **Travis CI** se encarga de realizar el despliegue según el fichero de configuración YAML proporcionado. Un aspecto importante a tener en cuenta es que existe un fichero con datos sensibles (diferentes claves, secretos de APIs de terceros, Firebase...) el cual no se sube a GitHub en claro, en cambio se sube un fichero encriptado con una clave la cual Travis guarda y se encarga de desencriptar este fichero a la hora del despliegue para que no exista el riesgo de que las claves estén visibles en el repositorio de GitHub; para ello se ha seguido la siguiente [documentación](#).

Para el despliegue de los ficheros estáticos del Front-End, se ha hecho uso de la plataforma (PaaS) Firebase Hosting desarrollada por Google. Esta plataforma provee de alojamiento de ficheros estáticos, gestión automática de certificados TLS y provisión de dominios de forma gratuita. La plataforma exige la creación de configuración de un proyecto en la consola web, tras lo que se debe instalar la utilidad de línea de comandos firebase-cli. Tras configurar el proyecto con firebase init, el despliegue se realiza simplemente ejecutando el comando firebase deploy. En el caso de Zarahealth, la aplicación web es accesible desde <https://zarahealth.web.app/>

Validación

Front-End

En cuanto a la validación del Front-End, se ha utilizado Selenium para realizar pruebas automáticas sobre el navegador web. Selenium permite grabar, editar y depurar casos de pruebas que se pueden automatizar. Las pruebas que se han realizado han sido: login de usuario, visualización correcta del dashboard, visualización correcta del feed, creación de un post, creación de un comentario, correcta visualización de estos, visualización de mapas, información y gráficas de aire, agua y polen,

correcta visualización de los ajustes, tanto de administrador como de usuario y comprobación del correcto funcionamiento de estos.

Back-End

A la hora de comprobar que las funciones del Back-End funcionan de forma correcta se ha utilizado Jasmine para realizar dos grupos grandes de pruebas.

Cliente

En esta parte de pruebas en primer lugar se han creado los datos de prueba en la base necesarios para poder obtener un token de cliente y los ajustes para establecer si las funciones relacionadas con los datos de agua, aire y polen deben devolver o no resultados.

A continuación, se crea una base de datos nueva de manera automática en memoria local para no ensuciar la base del servidor con pruebas cada vez que se realizan los tests y se establece conexión con ella.

Por último, para terminar con la configuración de los tests se intercepta la conexión original a la base de datos del servidor AWS haciendo así que se utilice la conexión a la base que se acaba de desplegar.

Las pruebas que se han realizado para el cliente han sido las siguientes:

- Obtener token de cliente.
- Crear un usuario.
- Obtener un token de usuario.
- Obtener tu propio usuario.
- Cambiar el estado de descargar o no un CSV.
- Obtener una medida de polen.
- Obtener todas las medidas de polen.
- Obtener una medida de agua.
- Obtener todas las medidas de agua.
- Obtener una medida de aire.
- Obtener todas las medidas de aire.
- Obtener el tiempo en Zaragoza.
- Actualizar la estación de aire preferida para el usuario.
- Actualizar la estación de agua preferida para el usuario.
- Actualizar el umbral de polen para el usuario.
- Actualizar el umbral de aire para el usuario.
- Crear un feed del usuario.
- Obtener todos los feeds.

- Establecer la opinión del usuario acerca de un feed.
- Comentar un feed.
- Actualizar su información personal.
- Comprobar que no se obtienen métricas.

Administrador

En esta parte de pruebas en primer lugar se han creado los datos de prueba en la base necesarios para poder obtener un token de cliente , los ajustes para establecer si las funciones relacionadas con los datos de agua, aire y polen deben devolver o no resultados y un usuario que sea administrador.

A continuación, se crea una base de datos nueva de manera automática en memoria local para no ensuciar la base del servidor con pruebas cada vez que se realizan los tests y se establece conexión con ella.

Por último, para terminar con la configuración de los tests se intercepta la conexión original a la base de datos del servidor AWS haciendo así que se utilice la conexión a la base que se acaba de desplegar.

Las pruebas que se han realizado para el administrador han sido las siguientes:

- Obtener un token de usuario.
- Obtener métricas.
- Obtener todos los usuarios.
- Actualizar el estado de un usuario (banearlo o desbanearlo).
- Obtener los ajustes de aire, agua y polen.
- Actualizar los ajustes del agua.
- Actualizar los ajustes del aire.
- Actualizar los ajustes del polen.

Problemas encontrados durante el desarrollo

Uno de los principales problemas encontrados fue a la hora de devolver imágenes. En un primer momento Back-End devolvía las imágenes en formato base64 pero esto supuso un gran problema para Front-End ya que se tardaba mucho en procesar y mostrar cualquier imagen. La solución pasó por devolver el id de las fotos en lugar del formato de base64 y con ese id hacer otra petición al Back-End la cual devuelve la foto en formato multipart.

La integración de la autenticación con Google y su servidor OAuth planteó algunos problemas estructurales con el uso de un sistema de autenticación OAuth propio ya que generamos tokens de acceso incompatibles con los de Google y las librerías utilizadas no contemplaban este caso fue necesario tratar la autenticación con Google de manera que fuese equivalente a una autenticación estándar con usuario y contraseña de manera que se vuelve a proporcionar un token de acceso interno.

Otro cosa que nos dio bastantes problemas fue el devolver errores con **GraphQL** ya que algunas llamadas de mongoose funcionan con promesas. Se intentaban lanzar errores dentro de las funciones que tratan las *callback* de las promesas y no se conseguía ya que los resultados se devuelven fuera de estas funciones. La solución consistió en devolver tanto el caso del caso error como el devolver la respuesta de la promesa una vez se llama a la función *callback*.

En cuanto a las dificultades encontradas en el desarrollo del Front-End, una de las principales ha sido lidiar con la aplicación de estilos CSS, e integrarlos con la librería de estilos Material-UI, ya que, en general, la documentación es bastante pobre y el equipo de desarrollo Front-End no tenía demasiada experiencia en este aspecto.

Resulta también destacable como problema la gestión del estado local de la aplicación web y su persistencia en el tiempo. Esto finalmente fue conseguido mediante el cliente GraphQL Apollo Client, que provee de mecanismos de gestión de estado y una extensión para guardar su estado interno en un objeto serializado en el almacenamiento local. A este respecto, las dificultades surgieron principalmente de la ausencia de mecanismos claros para la depuración.

Finalmente, ha resultado también algo complicada la implementación de mecanismos de paginación, ya que el uso de GraphQL y su peculiar sistema de cacheado dinámico invalidaba la mayoría de soluciones clásicas documentadas.

Análisis de problemas potenciales

La mayor limitación del sistema respecto al Back-End sería el almacenamiento en la base de datos, ahora mismo disponemos de 1GB de almacenamiento pero si la aplicación se popularizara y se empezaran a crear usuarios y feeds con sus respectivas imágenes, a parte de todo el flujo de control que se guarda en la base de datos respecto a que es lo

que los usuarios hacen cuando interactúan con la web, se podría superar este tamaño por lo que sería necesario ampliar el almacenamiento si Amazon lo permite o migrar la base de datos a otro servicio (cloud incluso en local) lo que resultaría mucho más costoso.

En cuanto a potenciales limitaciones del Front-End, la principal pasa por la interfaz de usuario de escritorio. Se ha seguido una aproximación *mobile first* que ha priorizado completamente el desarrollo de la interfaz móvil, por lo que en caso de uso generalizado, la interfaz de escritorio bien merecería ser mejorada.

Además, no se han tenido en cuenta el uso de etiquetas de accesibilidad u otros requisitos del estándar WCAG, que, en caso de ser Zarahealth una aplicación de uso generalizado, deberían considerarse.

Distribución de tiempo

Gantt

Se incluye el diagrama de Gantt con la entrega de dicha memoria en el documento GANTT.xlsx.

Tiempo invertido por alumno

	Javier	Pedro	Daniel	Victor	
Tiempo (horas)	104 horas	110 horas	113 horas	107 horas	434 horas

Conclusiones

Zarahealth ha sido una gran oportunidad, tanto para el aprendizaje como para practicar el trabajo en equipo. Estamos muy satisfechos con el resultado final de nuestro trabajo, ya que creemos que hemos creado algo que realmente puede llegar a ser útil para otras personas, y que además gira de forma central en torno a nuestra ciudad, lo cual siempre es un valor añadido.

Si bien el desarrollo ha sido largo y no han faltado los problemas de todos los tipos y clases, concluimos este proyecto satisfechos con lo aprendido y los resultados alcanzados.

Valoración personal de cada miembro

Víctor

En esta asignatura he aprendido el uso de nuevas tecnologías que se utilizan en el día a día de las empresas. Aunque al principio me ha costado un poco adaptarme a como funciona node luego me ha parecido una herramienta muy práctica ya que tiene una gran diversidad de módulos con funciones ya implementadas y una amplia comunidad en la que se puede encontrar información cuando te surge algún problema.

La implementación de un servidor de OAuth 2.0 me ha parecido muy interesante ya que he podido entender cómo funciona este mecanismo internamente.

Por otro lado interactuar con mongo ha sido muy fácil ya que node.js tiene un complemento llamado *mongoose* el cual permite de una forma muy simple guardar, eliminar o modificar la base de datos.

Daniel

La realización de un proyecto tan completo utilizando tecnologías que se encuentran a la vanguardia ha sido una experiencia muy enriquecedora para mi desarrollo personal y con vistas a mi futuro profesional.

Ha resultado muy interesante la integración de Node.js con GraphQL, con MongoDB y con OAuth 2.0 para desarrollar y desplegar en conjunto un sistema seguro, robusto y eficaz. La comunicación y el trabajo en equipo también han sido componentes muy claves en el desarrollo Back End ya que necesitaba de muchísima coordinación entre los miembros a la hora de realizar tareas complejas y críticas, lo que ha permitido avanzar de una forma muy progresiva y veloz las etapas de desarrollo.

En definitiva un proyecto en que se ha favorecido la creatividad, resolutividad y el trabajo en equipo con grandes tecnologías.

Javier

El desarrollo de este proyecto me ha permitido profundizar más en el uso de tecnologías bastante demandadas actualmente como es el caso de React. De todos los frameworks con los que he trabajado, no es el que elegiría para un proyecto futuro, pero sí he podido comprobar las grandes posibilidades que ofrece.

GraphQL, que ha sido novedad para mí, me ha permitido ir un poco más allá de la típica API Rest desarrolla hasta ahora en todos los proyectos de la carrera y comprobar las grandes ventajas de esta tecnología.

También, gracias a la buena documentación de la API desarrollada por el equipo de backend, no he encontrado problemas a la hora de realizar la integración, tal y como me había ocurrido en trabajos anteriores.

En resumen, un conocimiento más exhaustivo de tecnologías punteras.

Pedro

Zarahealth ha sido una buena oportunidad para conocer nuevas tecnologías, herramientas y formas de trabajar. Nunca había desarrollado un sitio web haciendo uso de un *framework* de Javascript, y, aunque no haya sido mi tecnología preferida para hacerlo, me ha permitido conocer de primera mano este entorno hoy en día estándar en la industria del desarrollo web.

Al igual que no creo que React se incorpore a mi *stack* de tecnologías para proyectos futuros, sí lo harán sin duda GraphQL y Node.js, ya que la experiencia de haber trabajado con ellos ha sido realmente positiva.

Finalmente, he podido comprobar como muchos de los problemas experimentados en proyectos anteriores similares (que implicasen integrar interfaces desarrolladas por subequipos separados) no se han dado en este gracias al excelente trabajo de Daniel y Víctor documentando la API, por lo que sin duda copiaré sus pautas de trabajo en futuras ocasiones.

En cualquier caso, valoro la experiencia de la asignatura como muy enriquecedora y positiva.