Memoria Final

Grupo Otters

Pedro Allué Tamargo (758267) Juan José Tambo (755742)
Jesús Villacampa Sagaste (755739) José Félix Yagüe Royo (755416)

24 de mayo de 2021

Índice

Resumen **3**

1. Propuestas similares **5**

2. Arquitectura de alto nivel **5**

3. Modelo de datos **6**

4. API REST **6**

5. Implementación **6**
5.1. Backend . . . 6
5.2. Frontend . . . 6
5.3. Login con proveedores externos . . . 7
5.4. Captcha . . . 7
5.5. Notificación de eventos vía e-mail . . . 7

6. Modelo de navegación **8**
6.1. Login . . . 9
6.2. Registro . . . 9
6.3. Perfil . . . 10
6.4. Estadísticas . . . 10
6.5. Foro principal . . . 11
6.6. Perfil Externo . . . 11
6.7. Foro Peticiones Ayuda . . . 12
6.8. Petición Ayuda . . . 12
6.9. Petición Vacía . . . 13
6.10. Post Individual . . . 13
6.11. Post Vacío . . . 14

7. Analíticas **14**

8. Despliegue del sistema **15**

9. Validación **16**

10. Problemas encontrados durante el desarrollo **16**

11. Análisis de problemas potenciales **17**

12. Distribución de tiempo **17**
12.1. Cronograma del proyecto . . . 17
12.2. Distribución del trabajo . . . 18
12.2.1. Pedro Allué . . . 18
12.2.2. Juan José Tambo . . . 18
12.2.3. Jesús Villacampa . . . 18
12.2.4. José Félix Yagüe . . . 18

13. Conclusiones **19**

14. Valoración personal de cada miembro **19**
14.1. Pedro Allué . . . 19
14.2. Juan José Tambo . . . 19
14.3. Jesús Villacampa . . . 19
14.4. José Félix Yagüe . . . 19

Anexo 1: Modelo de datos **20**

Anexo 2: Diagrama de Gantt **21**

Anexo 3: Diagrama flujo de peticiones **22**

Resumen

En el ámbito de la pandemia del *COVID-19* las tecnologías han ocupado una parte importante de nuestra vida. Nos han servido para acercarnos a aquellas personas de las que la situación nos ha alejado.
Este proyecto nace como una respuesta ante la necesidad humana de relacionarse con otras personas durante el tiempo que puede durar una cuarentena por un contacto estrecho con un positivo de *COVID-19*.
Otra cara de este proyecto es prestar ayuda a aquellas personas que lo necesitan. Por ello se pone a disposición de los usuarios un tablón de peticiones de ayuda.
También se incluye un apartado más numérico para visualizar la evolución de la pandemia en cada zona básica de salud y en la comunidad de Aragón. Los datos se obtienen de la propia plataforma de transparencia del Gobierno de Aragón[1].

[1] `https://transparencia.aragon.es/COVID19`

Datos de la aplicación

- URL de la aplicación: `https://stw-otters-frontend.herokuapp.com/`
- URL de la documentación Swagger: `https://stw-otters-backend.herokuapp.com/api_docs/`
- Usuarios registrados en la aplicación
  - **email – contraseña**
  - jfabra@unizar.es – pass1234 (**es administrador**)
  - 755739@unizar.es – pass1234
  - 755416@unizar.es – pass1234
  - Hay que tener en cuenta que hay acciones que envían correos electrónicos a los usuarios que reciben esas acciones.

1. Propuestas similares

Al dividirse en tres secciones, se van a presentar alternativas a cada una de ellas. Sobre la parte de foro la alternativa más popular sería el modelo de *Reddit*[2], se el sistema de valoración está basado en el de esta página. Los usuario pueden publicar posts de distintos temas y otros usuarios interactuar con los posts a través de comentarios.
Una propuesta si miliar a la sección de estadísticas serían las disponibles en la propia página de la fuente de datos. En esta página se puede acceder a un visualizador de datos en un mapa interactivo[3].
Sobre el apartado de peticiones de ayuda no se han encontrado propuestas similares. No obstante, se puede encontrar cierta similitud entre este apartado y el sistema de *Glovo*[4]. En el sistema *Glovo* los repartidores se asignan a los recados que deben completar. En nuestro sistema son los usuarios los que escogen qué peticiones quieren atender.

2. Arquitectura de alto nivel

El sistema a desarrollar se corresponde con el diagrama de la Figura 1.

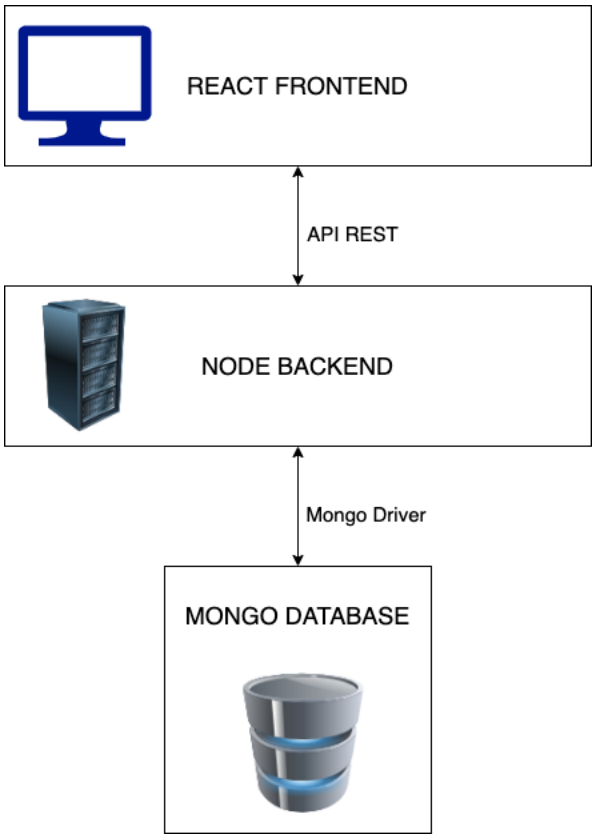

Figura 1: Diagrama de despliegue del sistema desarrollado

Se puede comprobar que el sistema se divide en tres componentes. El primer componente se corresponde con el *frontend*. Está desarrollado en React con *Typescript*. Se conecta al servidor de *backend* mediante la API REST que este expone.
El segundo componente es el *backend*. Está desarrollado con *Node* con el framework *Express* y *Typescript*. Expone una API REST para permitir las conexiones desde el *frontend*. Para permitir la escalabilidad de este componente se extrae su estado a un almacén externo, la base de datos de *Mongo*. Para ello se utiliza el *Driver de Mongo* que provee el *ORM* de *Mongoose*.
El último componente es la base de datos de *Mongo*, permite almacenar la información del *backend*.

[2] `https://www.reddit.com/`
[3] `https://datacovid.salud.aragon.es/covid/`
[4] `https://glovoapp.com/es/es/`

3. Modelo de datos

El modelo de datos utilizado se puede observar en la Figura 14. En la traducción de este modelo conceptual al modelo de *Mongoose* se han sustituido las relaciones entre las colecciones por el identificador del objeto de su tabla correspondiente.
Se ha planteado esta traducción ya que al almacenar el objeto entero no se creaba una colección asociada al esquema. Esta solución no interesa de cara a poder realizar consultas sobre cada colección de forma independiente. Al utilizar distintas colecciones se mejora la concurrencia ya que estos problemas serán menos frecuentes.

4. API REST

La documentación de la API en Swagger se puede encontrar en el enlace: *Documentación Swagger*.
Todos los endpoints (a excepción de los de autenticación) están protegidos mediante el patrón de seguridad de los tokens de tipo *Bearer*. Esto es así ya en el *backend* no se permite la ejecución de los mismos si no está presente el token.

5. Implementación

5.1. Backend

El *Backend* se ha dividido en 4 bloques principales: Servicios, modelos, rutas y controladores.

En el bloque de servicios se introducen distintas funcionalidades que son reutilizadas en varios módulos, como la configuración para el envío de correos, la configuración de las estrategias de *Passport* o la función de parseo de los ficheros *XLSX*.

En el bloque de modelos se han introducido los esquemas de los distintos objetos que se utilizan en nuestro sistema: *Usuario, Zona Sanitaria, Post, Peticiones* y *Comentarios*.

Dentro de las rutas, se indican los distintos endpoints disponibles para cada uno de los modelos, indicando así el método permitido para el mismo. Desde esta manera, se facilita la organización de los endpoints, siguiendo el siguiente esquema: `app.use(‘‘/endpoint’’, <route>)`. Además, facilita también la protección de la API, ya que simplemente con insertar `passport.authenticate('jwt', session: false )` a la sentencia anterior, evita que un usuario que no ha iniciado sesión utilice la API.
Por último, se ha utilizado un controlador para cada uno de los modelos, en los que se desarrolla la funcionalidad de cada uno de los endpoints indicados en las rutas. Cabe destacar que se ha procurado seguir el mismo esquema para las devoluciones de la API, indicando los códigos HTTP correspondientes y controlando los posibles errores que pueden aparecer.

Las imágenes de perfil de un usuario se guardan en formato *base 64* ya que de esta manera se pueden almacenar como un campo del usuario representadas por una cadena, evitando así una consulta extra para obtener la imagen.

Las peticiones creadas por un usuario tienen distintos estados, representados en la figura 16.
Nada mas ser creada, se le asigna el estado *abierta*, a la que pueden asignarse hasta 5 usuarios (el usuario puede cancelar la asignación). Cuando un usuario se asigna, se cambia el estado a *asignada*. Si el/los usuarios asignados eliminan la asignación, el estado de la petición pasa a *abierta*.
Una petición adopta el estado de *cancelada* si se expira estando *abierta* o si es cancelada por el usuario. Si expira estando asignada, por defecto adquiere el estado de *completada*, aunque el usuario creador puede modificar su estado a *cancelada* si el usuario asignado no ha realizado la ayuda. En este caso, se marca un *strike* al usuario asignado. En caso de que reciba 5 *strikes*, el usuario resulta baneado de la aplicación durante una semana.

5.2. Frontend

Para desarrollar el frontend se ha creado un directorio distinto del que contiene el backend, pero dentro del directorio raíz de nuestro repositorio de *GitHub*. La realización de este módulo ha sido en React y Typescript, pero

el fichero raíz, *index.js*, está escrito en Javascript ya que es necesario. Desde él se importa Boostrap y se llama al fichero de Typescript donde está almacenada la información sobre las rutas que tiene el proyecto.
Para desarrollar los distintos apartados que tenemos en el sistema, usuarios, estadísticas, post y peticiones, se han estructurado en carpetas distintas para una mejor organización y claridad. Para realizar las llamadas a la API con el objetivo de obtener datos de la base de datos se ha desarrollado un hook llamado *useGetFetch* la cual unifica todas las llamadas de tipo *GET*, para poder ser usada desde cualquier módulo, consiguiendo así no tener que reescribir código.
También, se ha hecho uso de un token que gestiona las sesiones de los usuarios logueados en el sistema, el valor de este token esta almacenado en el *LocalStorage*, que además escucha los eventos de almacenamiento para realizar una sincronización de las distintas pestañas de la aplicación.

5.3. Login con proveedores externos

Para poder controlar el inicio de sesión de usuarios, se ha utilizado la librería *Passport* ya que permite implementar esta funcionalidad de forma sencilla. En primer lugar se iba a utilizar para controlar el acceso tanto de forma local como mediante proveedores externos (*Google y Facebook*), aunque para estos dos últimos ha sido imposible utilizar esta herramienta, tal y como se explica en la sección 10.
Para el inicio de forma local, se ha usado la estrategia *LocalStrategy* de *passport-local* mediante la cual se comprueba si las credenciales introducidas por el usuario al intentar iniciar sesión son correctas.

Por otro lado, para el inicio de sesión mediante proveedores externos se han usado las siguientes librerías: *react-google-login* y *react-facebook-login* para el *frontend* y *google-auth-library* para el *Backend*.
Para permitir el inicio de sesión mediante estos proveedores, se ha tenido que crear una aplicación en sus entornos de desarrollo [5] y utilizar las claves secretas de las mismas. Mediante el uso de las librerías de *frontend* mencionadas, permitimos que se comunique con el servicio externo correspondiente y seleccionar así una cuenta existente en el mismo. Una vez comunicado con el proveedor externo y obtenido los datos de esa cuenta, se manda la información al *backend*. En el caso de *Google*, se envía un *token* relacionado con el usuario seleccionado mediante el cual, a partir de la librería *google-auth-library* se puede obtener información relacionada con el usuario (nombre, email ...). Para *Facebook*, se envía desde *frontend* un token y un id de usuario relacionado con la cuenta seleccionada, mediante los cuales, realizando una petición a `https://graph.facebook.com/` se obtiene información detallada del usuario.
Finalmente, se crea usuario en caso de que no existiera ese correo con anterioridad y se inicia la sesión del usuario.

5.4. Captcha

Para la implementación del desarrollo opcional del *Captcha* se ha utilizado el ofrecido por *Google* (*ReCaptcha v2*[6]). Este sistema de verificación es muy sencillo de integrar ya que mediante el paquete *react-google-recaptcha*[7] se integra de forma sencilla con el *frontend*. Este componente genera un token que a la hora del registro es enviado al *backend*. En el *backend* se realiza una llamada a la *API* de *Google* para verificar que ese token es válido. En caso de que no sea válido el sistema devuelve una respuesta de error al cliente indicando el motivo del error.

5.5. Notificación de eventos vía e-mail

Se ha implementado un sistema de notificaciones vía e-mail con el módulo de *Node.js*, *Nodemailer*.
Para ello se ha creado una cuenta de correo electrónico de *Gmail*, la cual se utilizará para enviar los correos.
Con el código que se muestra a continuación, desde *backend*, se configura la conexión

```
const transporter = Nodemailer.createTransport({
        port: 465,
        host: 'smtp.gmail.com',
        secure: true,
        auth: {
            user: process.env.EMAIL_USER!,
            pass: process.env.EMAIL_PASS!
```

[5] `https://developers.facebook.com/`, `https://console.cloud.google.com/`
[6] `https://www.google.com/recaptcha`
[7] `https://www.npmjs.com/package/react-google-recaptcha`

```
        }
    })
```

y con el siguiente, se forma el mensaje(asunto y cuerpo) y se escribe el destinatario

```
await transporter.sendMail({
        from: process.env.EMAIL_USER!,
        to: user.email,
        subject: //Aqui se escribe el asunto del mensaje,
        html: //Aqui se escribe el cuerpo del mensaje

    });
```

En función de la notificación se enviará un mensaje u otro.

Este sistema notificará los siguientes eventos:

- El usuario creador de una petición será notificado cuando alguien sea asignado a su petición y se le enviará su correo electrónico para contactar con él.
- El usuario que se ha asignado a una petición será notificado recibiendo información acerca de esa petición y su creador.
- Un usuario que quería asignarse a una petición y había entrado en la cola de espera, consigue asignarse porque el que estaba previamente asignado lo ha cancelado. Este usuario es notificado como el anterior caso.
- El usuario creador de un post recibe un comentario y se le notifica con el nombre del usuario del comentario y el post que ha sido comentado.

6. Modelo de navegación

Para explicar el modelo de navegación se utilizará el prototipo creado con la herramienta *Adobe Xd*, debido a que la creación de las vistas se ha creado siguiéndolo estrictamente. En caso de que se modifique algo en la implementación final se remarcará que ha sido cambiado

6.1. Login

En esta pantalla el usuario podrá iniciar sesión utilizando su nombre de usuario y su contraseña o utilizando su cuenta de *Google* o *Facebook*. Existe una integración con el mecanismo *Oauth2* de Google.

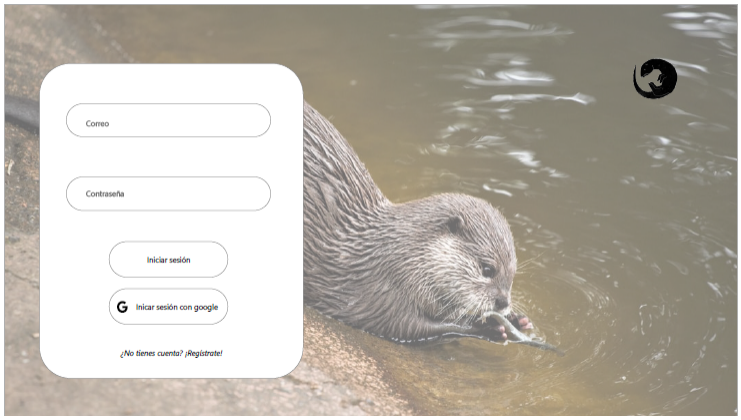

Figura 2: Captura de pantalla de la pantalla de login

6.2. Registro

En esta pantalla se permitirá el registro de nuevos usuarios a partir de sus datos personales (correo electrónico, nombre de usuario, contraseña, zona sanitaria). Para proporcionar más seguridad, se ha añadido un *CAPTCHA* que verifica que la persona que se registra no sea un bot.

Figura 3: Captura de pantalla de la pantalla de registro

6.3. Perfil

En esta pantalla se muestra toda la información relacionada con el usuario, los *posts* que ha realizado y todas las peticiones de ayuda que ha solicitado, indicando si han sido atendidas o no.
También se permite editar cierta información como la contraseña o la zona básica de salud.
Se accede a ella a través de la palabra y el icono en la parte superior derecha, "Cuenta".

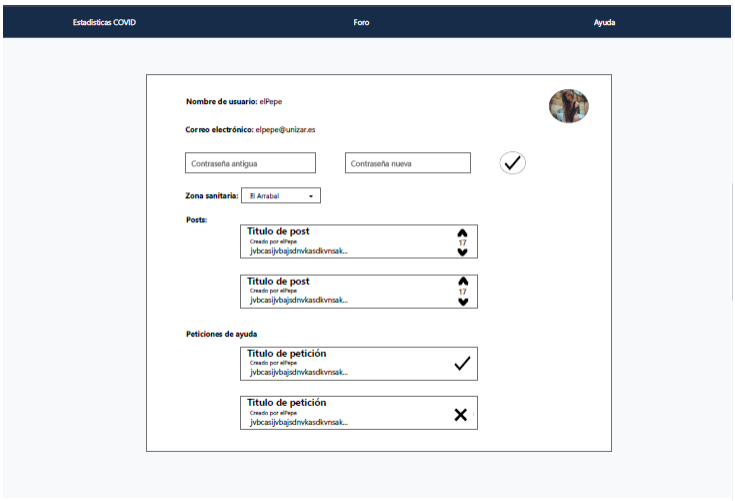

Figura 4: Captura de pantalla de la pantalla de perfil

6.4. Estadísticas

Esta es la pantalla principal del sistema y en ella se muestran gráficas que indican información actual acerca de la situación del *COVID-19*. Por defecto, se muestra una gráfica de los contagios actuales en la zona básica de salud que ha seleccionado el usuario al registrarse en la aplicación. Opcionalmente el usuario puede seleccionar el día y la zona de salud que desee.

También aparece una gráfica que muestra los contagios actuales en la comunidad de Aragón, permitiendo seleccionar opcionalmente otra fecha alternativa, la cual se ha implementado con *ChartJS*

En esta pantalla se utilizarán los datos de la fuente de datos escogida.

Figura 5: Captura de pantalla de la pantalla de estadísticas

6.5. Foro principal

En esta pantalla se pueden observar las publicaciones de otros usuarios. Desde ésta, se visualizará el título, el creador y una parte del cuerpo del post en caso de que sea más largo de 100 caracteres, así como la valoración general de cada post (valoraciones positivas - valoraciones negativas). Se podrán ordenar las publicaciones en función de su valoración, de su número de comentarios y su fecha de publicación, así como añadir una valoración positiva o negativa a un post desde el botón de la esquina superior derecha "Filtrar".

En la parte superior izquierda está el botón "Añadir Post" a través del cual se puede acceder a la pantalla de creación de Post

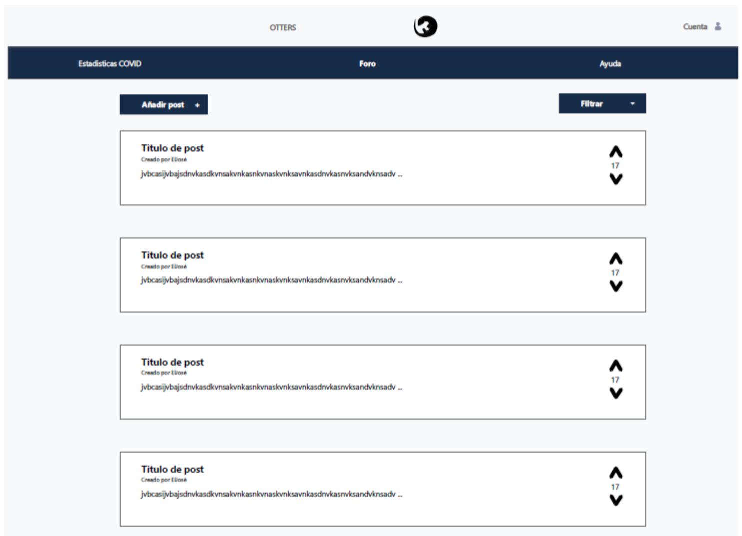

Figura 6: Captura de pantalla de la pantalla de foro principal

6.6. Perfil Externo

En esta pantalla se mostrará la información de un usuario que no seas tú mismo. Además se mostrará una gráfica acerca de los casos *COVID-19* en la zona sanitaria de dicho usuario.
Se podrá acceder a ella clicando en el nombre de un usuario que aparezca como creador de un post o de una petición de ayuda

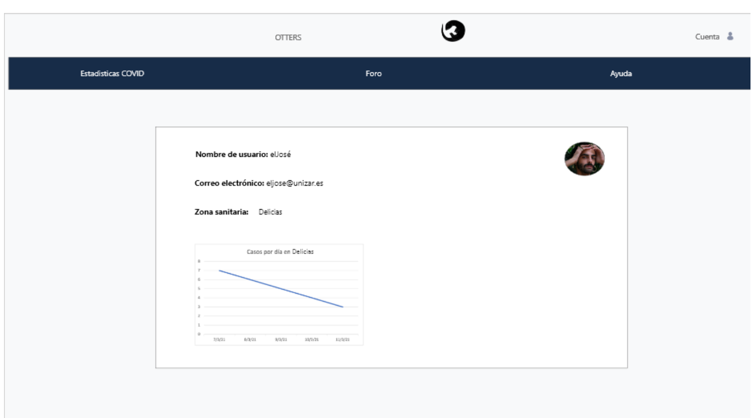

Figura 7: Captura de pantalla de la pantalla de perfil de otro usuario

6.7. Foro Peticiones Ayuda

En esta pantalla se pueden observar las peticiones de ayuda de otros usuarios de la misma zona sanitaria que la seleccionada por el usuario. Aparecerán solamente aquellas peticiones que no hayan sido atendidas y se podrán ordenar en función de su fecha de publicación y su fecha de expiración desde el botón de la esquina superior derecha "Filtrar".
En la parte superior izquierda está el botón "Añadir Petición" a través del cual se puede acceder a la pantalla de creación de Petición de ayuda.

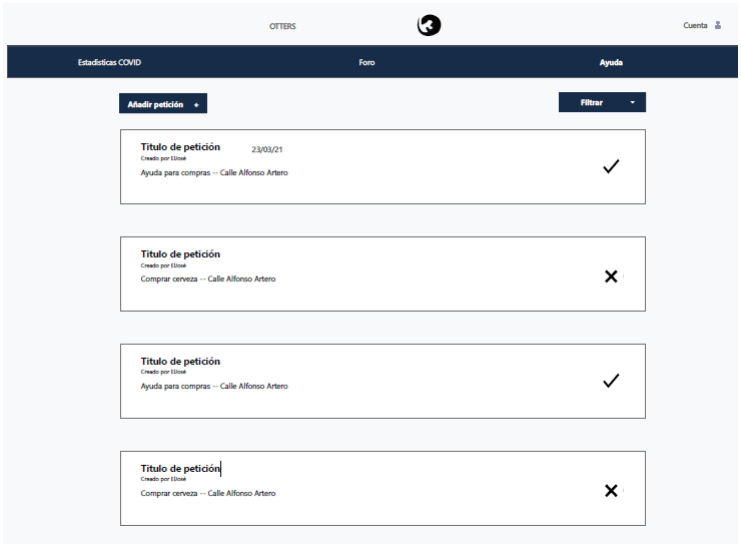

Figura 8: Captura de pantalla de la pantalla de foro de peticiones de ayuda

6.8. Petición Ayuda

En esta pantalla se puede acceder a los detalles de una petición de ayuda concreta publicada en el sistema. Se puede observar que aparece el *email* de contacto para el usuario que crea la publicación. En caso de que el usuario sea el creador de la petición existirá un botón para marcarla como resuelta y así hacerla desaparecer del *feed* de peticiones de ayuda.
También aparece el estado de la petición, es decir, *Abierta*, *Asignada* o *Cerrada* . Además, si no estás asignado, aparecerá un botón para asignarte y una vez clicado ese botón indicará si estás en cola o asignado.

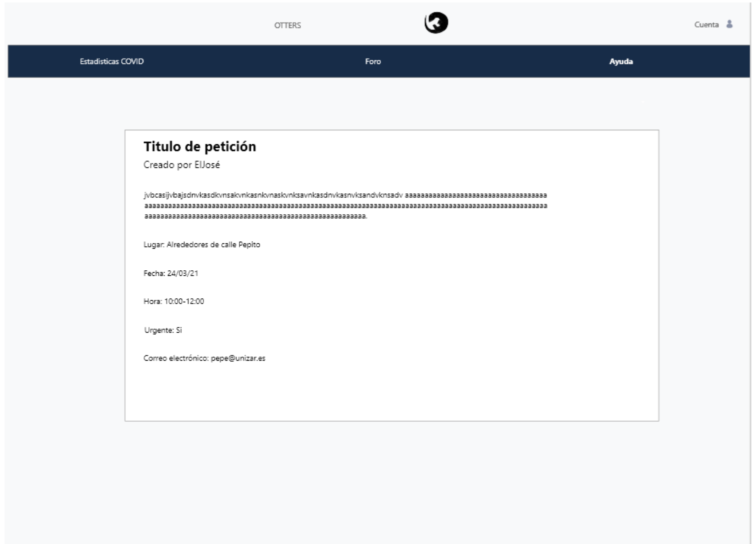

Figura 9: Captura de pantalla de la pantalla que muestra todas las peticiones de ayuda que no han sido procesadas

6.9. Petición Vacía

En esta pantalla el usuario podrá crear peticiones de ayuda que se almacenarán en el sistema en el momento que se presione el botón de la parte inferior. Para ello, el usuario tendrá que rellenar previamente los campos de título, fecha, lugar, hora, correo electrónico, descripción y un marcador para indicar si la tarea es urgente.

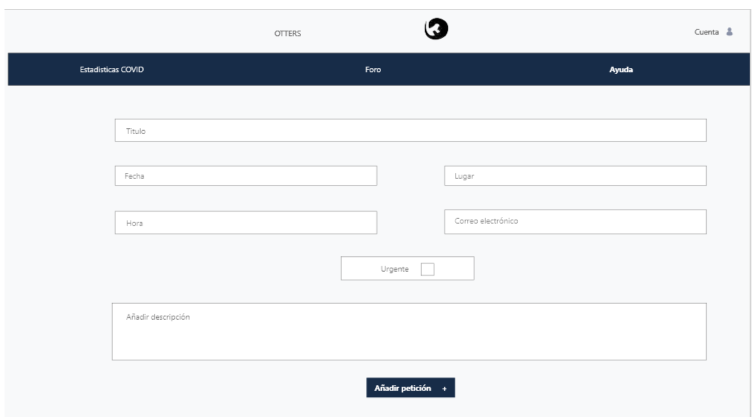

Figura 10: Captura de pantalla de la pantalla para generar una nueva petición de ayuda

6.10. Post Individual

En esta pantalla se muestra el interior de un post creado, el cuál también tiene comentarios asociados. Los comentarios se separan del post mediante cajas distintas para mayor claridad. Además, en la parte derecha se muestran unas flechas que almacenan las opiniones de los usuarios, similar a un sistema de *me gusta* y *no me gusta*.

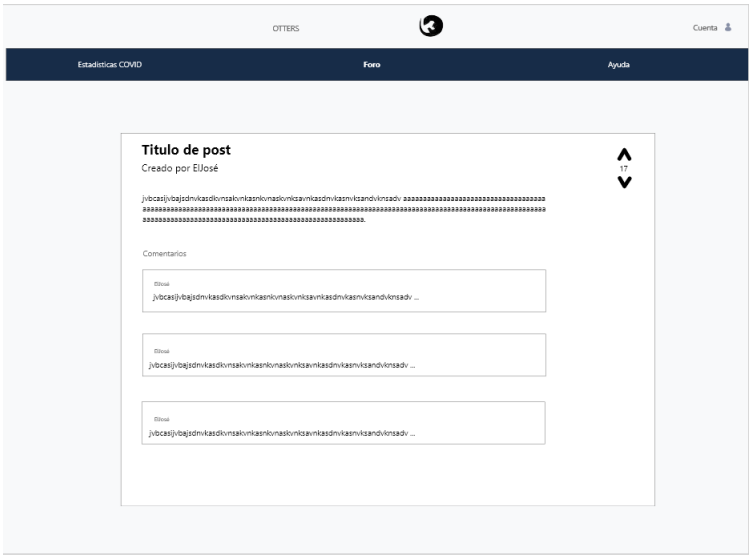

Figura 11: Captura de pantalla de la pantalla de un *post* en concreto junto a comentarios realizados sobre el mismo

6.11. Post Vacío

En esta pantalla el usuario podrá rellenar el campo de título y el contenido para crear un *post* que se publicará en el momento en el que presione el botón de la parte inferior.

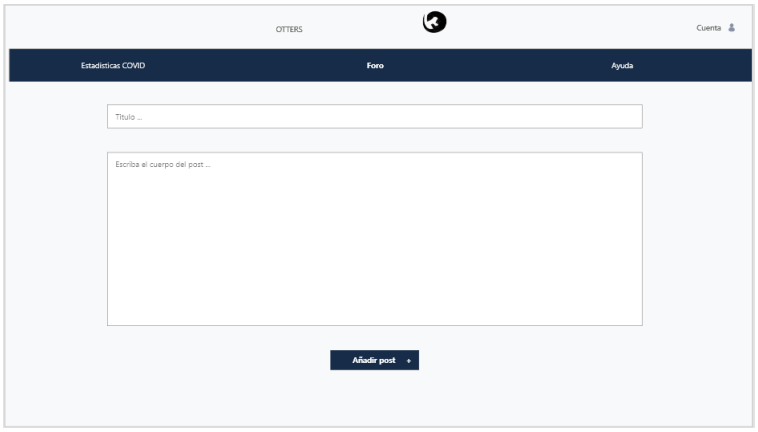

Figura 12: Captura de creación de un post

7. Analíticas

Las estadísticas que se muestran se obtienen (como se ha comentado anteriormente) de los datos del Gobierno de Aragón a través de su plataforma de transparencia[8]. Las gráficas que se utilizan son gráficas de líneas. Con ellas se busca visualizar la evolución de la pandemia a lo largo del tiempo en las distintas zonas de salud y en la comunidad autónoma. Esto permite al usuario hacerse una idea de la evolución de la pandemia.

La Figura 13 muestra como se visualizan las gráficas en la aplicación desarrollada. Se puede observar que las gráficas no muestran datos para todas las fechas. Este problema se comentará posteriormente. No obstante, la gráfica no presenta cortes ni puntos con valor 0, lo cual no sería correcto. En lugar de esto el propio *framework* de las gráficas (*Chartjs*) realiza una interpolación entre los puntos y dibuja una recta entre ellos.
También se puede observar que encima de la gráfica se encuentra un selector de fecha. En este selector se puede

[8] `https://transparencia.aragon.es/COVID19`

obtener de forma textual el número de casos obtenidos para una fecha en una zona sanitaria en concreto.

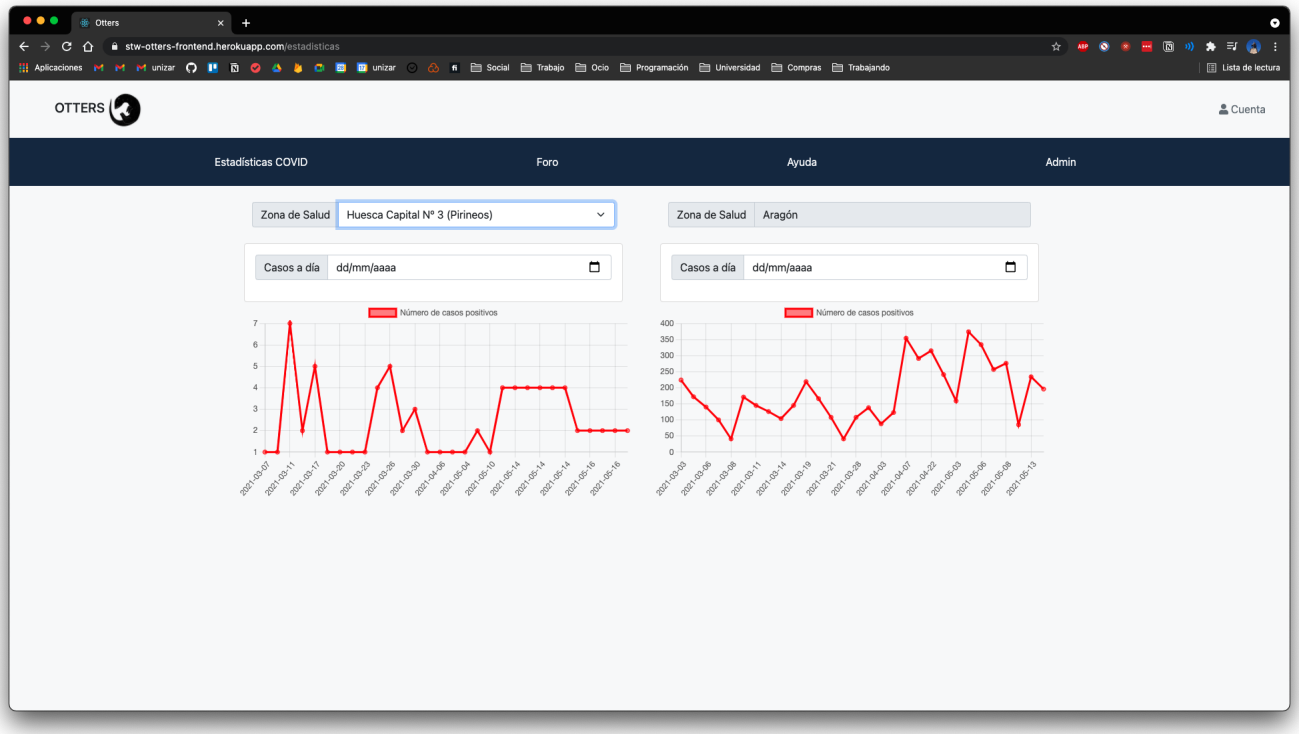

Figura 13: Captura de pantalla de las gráficas

8. Despliegue del sistema

El sistema se despliega sobre *Heroku* utilizando dos nodos. Se utilizará un nodo para el *backend* y se configurarán las variables de *Heroku* de acuerdo a las utilizadas en el fichero `.env` utilizado en desarrollo. Al utilizar la metodología de ficheros de variables de entorno se obtiene una mayor seguridad ya que las contraseñas y variables de producción no se publican en el repositorio. El aspecto del fichero de variables de entorno de este nodo será el siguiente:

```
POPULATEDB=false
PRODUCTION=true
COOKIES_CODE=<cookiescode>
MONGO_URI=mongodb+srv://...
CORS_URI=https://stw-otters-frontend.herokuapp.com
JWT_SECRET=<jwtsecret>
JWT_LIFETIME=86400
JWT_ALGORITHM=HS256
OAUTH_CLIENT_ID=<oauthgoogleclientid>
OAUTH_CLIENT_SECRET=<oauthgoogleclientsecret>
RECAPTCHA_SECRET=<recaptchasecret>
BACK_URI=https://stw-otters-backend.herokuapp.com
EMAIL_USER=ottersemailservice@gmail.com
EMAIL_PASS=<googleaccountapplicationpassword>
```

El nodo del *frontend* se despliega de manera análoga. Se ha utilizado un fichero de desarrollo `.env.local` y por lo tanto se han configurado las variables de entorno en *Heroku*. El fichero de configuración de este nodo sería el

siguiente:

```
REACT_APP_BASEURL=<backendbaseURL>
REACT_APP_RECAPTCHA_CLIENT_KEY=<googlerecaptchakey>
```

Para permitir el uso del *Recaptcha* hay que establecer la dirección del frontend desde la consola de administración del *ReCaptcha*[9].

Un factor importante en el despliegue es mantener las dependencias de desarrollo ya que, al utilizar *Typescript* son necesarias las definiciones de tipos y el compilador `tsc`. Para que *Heroku* no limpie estas librerías es necesario establecer la variable `NPMNPM_CONFIG_PRODUCTION` a `false`.

Para la población de la base de datos se cuenta con una variable de entorno `POPULATEDB` que si toma el valor `true` se ejecuta un script en la iniciación del servidor que añade datos a la base de datos. Esto ha sido muy útil durante el desarrollo ya que permitía a los integrantes del equipo obtener un entorno de ejecución con algunos datos. Esta variable de entorno en producción deberá tener valor `false`.

9. Validación

A la hora de realizar la validación de la API desarrollada en el proyecto se ha utilizado la herramienta *Postman*. A medida que se han realizado las distintas funciones de la API que van a ser llamadas desde el frontend, se han comprobando con llamadas desde *Postman* para garantizar su correcto funcionamiento.
Una vez ya validadas las llamadas por medio de *Postman*, también se han ido comprobando nuevamente a medida que se han integrado con el frontend, para ello el encargado de esa funcionalidad ha realizado las tareas que puede hacer un usuario final asegurando que funcionan de la manera adecuada. Además, al subir los cambios a *GitHub* mediante un *Pull Request* el desarrollador ha asignado a otro miembro para que revise los nuevos cambios para encontrar fallos que el primero puede no haber tenido en cuenta.
Por último, una vez desarrollada la versión final se han realizado pruebas con usuarios externos a proyecto, familiares y amigos, para que prueben todas las funcionalidades y nos den su punto de vista de cara al proyecto.

10. Problemas encontrados durante el desarrollo

Un gran problema que se ha encontrado ha sido a la hora de parsear los ficheros *.xslx* para extraer los datos a mostrar en las gráficas. Con algún fichero en concreto, el dato que se coge está desviado en unas cuantas filas y columnas. Es decir, si se quiere coger los datos a partir de B15, los datos que se cogen son a partir de C23 (es un ejemplo). Este error no tiene sentido aparente alguno y se ha invertido bastante tiempo intentando arreglarlo, intentando parsear todos los archivos *.xslx* a archivos *.csv*. El problemas seguía ocurriendo y ha sido imposible solucionarlo. El método que se ha aplicado para que el error no sea tan grave, ha sido escribir en el día que falla una interpolación de puntos con *ChartJS*. Una posible solución a futuro sería obtener los datos teniendo en cuenta la desviación en los días que ocurran. Pero esta solución no es escalable ya que no se sabe en que ficheros del futuro ocurrirá este error.

Passport se ha utilizado para integrar el inicio de sesión local y la integración de tokens *JWT* correctamente. *Passport* también proporciona una forma simple de integrarse con proveedores externos, pero intentando integrar eso es donde se ha producido el fallo. Al tener el *backend* y *frontend* separado, se produce un error de *Cors* al redirigirse a externos (Comunicación: Frontend -> Backend -> Proveedor externo). Para arreglar este problema se ha cambiado la estructura, haciendo que *frontend* se comunique directamente con el proveedor externo y una vez obtenidos los datos se pasan al *backend* para que los valide y los meta a la BBDD.

La división del trabajo horizontalmente ha sido una decisión que es considerada correcta por parte del grupo, ya que todos integrantes han podido trabajar en todos niveles del sistema, sin embargo, hay aspectos como la sinonimia a nivel intencional que aparecen a la hora de trabajar con esta metodología. En un ejemplo, para lo que el creador del esquema de Posts es "publisher", para el creador de Peticiones es "userId". Ambos representan lo mismo, que es

[9] https://www.google.com/recaptcha/admin/

el id del creador de un post o una petición respectivamente, pero al utilizar nombres distintos, a la hora de realizar pruebas sobre una sección que no es la que se ha trabajado principalmente puede gastarse mucho tiempo en darte cuenta de ese aspecto, ya que no es una cosa que se suela considerar a simple vista.

Al no ser expertos en *React*, incluso algún miembro que lo ha utilizado por primera vez, se han advertido algún comportamiento que ha costado descifrar a la hora de utilizar *useState*. Este ha sido el caso a la hora de realizar filtros a peticiones y posts, ya que si no cambia el conjunto de datos y tan solo cambia el orden de ellos, el componente no se actualizaba. Esto se ha arreglado creando siempre un nuevo conjunto de datos vacío y concatenarle el que deseamos mostrar con la nueva ordenación, ya que así detecta que el conjunto de datos ha cambiado y ya se puede mostrar correctamente.

11. Análisis de problemas potenciales

Uno de los principales problemas a futuro es el cambio en el esquema de los ficheros *XLSX* de los que se obtiene la información de los casos de Covid, ya que cualquier variación, por mínima que sea, haría que la forma actual de obtención de datos no fuera válida y por lo tanto habría que rediseñarla adaptándose a la nueva versión.

Otro problema sería al utilizar proveedores externos para realizar el inicio de sesión. Cualquier cambio en su política de uso puede afectar a esta funcionalidad.

En cuanto al despliegue *Heroku* proporciona un número limitado de horas de uso gratuito. Durante el desarrollo ya ha aparecido el problema de sobrepasar este límite y se ha tenido que introducir determinados datos para conseguir horas extras. Esto supone un problema a futuro ya que cuando se acaben las horas de nuevo, habría que utilizar un plan de pago para poder seguir utilizando esta herramienta.

Dado que la aplicación se centra principalmente en el Coronavirus, si en un futuro se consigue controlar la situación de pandemia, la aplicación perdería su atractivo o finalidad.

Un problema acerca de las herramientas y *frameworks* para desarrollo web. Se actualizan con mucha frecuencia y por lo tanto se debería mantener la aplicación con versión actualizada los mismos. En el caso de esta aplicación se ha usado *React*, que al principio no contaba con las funciones conocidas como *hooks* para evitar repetir código. Ahora con las nuevas versiones se han introducido cambios para mantener un código más limpio y mantenible.

12. Distribución de tiempo

En este apartado se va a explicar la distribución del tiempo. En la primera sección se va a utilizar un diagrama de Gantt para explicar la distribución temporal de las tareas y el orden de desarrollo. En la segunda sección se va a exponer la distribución del trabajo de los miembros del equipo y sus horas invertidas.

12.1. Cronograma del proyecto

En la Figura 15 se puede observar el diagrama de Gantt que marca la evolución del proyecto a lo largo del tiempo. Se pueden distinguir 4 etapas: Inicio del proyecto, creación de la infraestructura del servidor, primera versión y cierre de proyecto. Estas etapas se distinguen mediante los distintos hitos que representan las reuniones de seguimiento con el profesor.
La primera fase del proyecto se enfocó en la confección de la idea del sistema a desarrollar, a quien va dirigido y su diseño inicial. En la segunda fase se creó la infraestructura del servidor: los controladores, las rutas, el modelo de datos. Tras esto empezó la tercera fase. Esta fase está enfocada a la creación del frontal de la página web y su posterior integración con la infraestructura del servidor. En esta fase se integra el componente de autenticación local con *passport* y utilizando los tokens *JWT* se obtiene el mecanismo para autorizar las operaciones en el lado del servidor. La cuarta fase se corresponde con el cierre del proyecto. En esta fase se busca pulir algunos aspectos que no habían quedado claros y que tras la reunión con el profesor se resuelven. También en esta fase se aprovecha para integrarse con servicios de terceros tales como el *ReCaptcha* y la autenticación con *Google* o *Facebook*.

12.2. Distribución del trabajo

En esta sección se va a desglosar el trabajo realizado por cada integrante del equipo así como las horas dedicadas al proyecto.

12.2.1. Pedro Allué

Pedro es el coordinador del equipo. Una parte de su tiempo se ha invertido en la organización del proyecto (reuniones, seguimientos internos, división del trabajo entre los integrantes). Normalmente es el encargado de revisar el trabajo de los otros integrantes mediante los *Pull Requests* y de validarlo para admitirlo en la rama principal del repositorio de desarrollo.
Otra parte de su tiempo ha estado enfocada al desarrollo del componente de las zonas sanitarias, pasando por el proceso de parseo de los ficheros *XLSX* obtenidos del portal de transparencia del Gobierno de Aragón. Además de esto también ha creado la primera versión del componente del perfil de usuario y de administración.

El número de horas invertidas (en código y organización) de este integrante han sido: 80 horas.

12.2.2. Juan José Tambo

Juan José es un miembro especializado en el desarrollo del *Backend*. Se ha encargado de implementar la *API* para las peticiones y su integración con el *frontend*. También, se ha encargado de toda la funcionalidad de *login* y registro de usuarios, integrando así *Passport*. Otra tarea a destacar es la integración con proveedores externos como *Google* o *Facebook* para el inicio de sesión.
Además, se ha encargado de la protección de la *API* para que solo sea accesible a usuarios que han iniciado sesión (mediante tokens *JWT*).

El número de horas invertidas por este integrante ha sido: 75 horas.

12.2.3. Jesús Villacampa

Jesús es un miembro del equipo especializado en el tema de los posts. Se ha encargado de mantener las funcionalidades como la creación y eliminación de posts, creación de valoraciones positivas y negativas, y creación de comentarios, tanto en *backend* como en *frontend*. También junto a José Félix, han sido los encargados de implementar el flujo de proceso de las peticiones en el sistema. Otra de sus tareas durante el desarrollo fue la implementación del sistema de notificaciones por correo electrónico. Ha ayudado a Pedro en la realización de las gráficas con *ChartJS* y ha corregido algún bug de la parte visual y del filtrado de las peticiones en el *frontend*.

El número de horas invertidas por este integrante han sido: 75 horas.

12.2.4. José Félix Yagüe

José Félix es un miembro del equipo especializado en el tema de los usuarios. Se ha encargado de mantener las funcionalidades como el baneo de usuarios, la actualización de los datos de los mismos así como de la gestión de las imágenes de perfil de los usuarios en el formato *base64*. También junto a Jesús Villacampa han sido los encargados de implementar el flujo de proceso de las peticiones en el sistema. Otra de sus tareas durante el desarrollo fue la implementación y el mantenimiento del script de población de la base de datos, el cual es muy útil de cara a permitir a los integrantes levantar un entorno de desarrollo completamente poblado en cuestión de segundos.

El número de horas invertidas por este integrante han sido: 75 horas.

13. Conclusiones

En este proyecto se ha utilizado el stack de tecnologías MERN. Desde el punto de vista del equipo es muy importante conocer todo el stack de tecnologías que están involucradas en un proyecto. Durante el desarrollo al haber escogido Typescript se han encontrado problemas ya que las librerías muchas veces no aceptaban tipos (es el caso de los esquemas de Mongoose). No obstante, ha sido una decisión acertada de cara a crear un código más robusto de cara a los tipos de datos.
En general ha sido un proyecto interesante. De cara al futuro este proyecto podría integrarse con Twitter para mostrar los datos de casos en las zonas de salud al recargar la fuente de datos. También se podría buscar o negociar una API con el Gobierno de Aragón para obtener los datos sin tener que utilizar los ficheros XLSX que, como ya hemos comentado, han presentado varios problemas.

14. Valoración personal de cada miembro

14.1. Pedro Allué

La realización de este proyecto ha sido muy interesante de cara al uso de nuevas tecnologías. En el transcurso de la carrera se enseñan tecnologías asentadas y esta asignatura busca aprender las bases de un desarrollo más actual. En este proyecto se han explorado las posibilidades del stack de tecnologías *MEAN*. En el caso de nuestro proyecto hemos aprovechado las posibilidades de *Typescript* para obtener un código que permite obtener errores de tipos en tiempo de compilación. De esta forma el desarrollo se ha hecho más ágil ya que esta protección ha impedido que ocurrieran errores que serían más complicados de solucionar en tiempo de ejecución.
Nunca habría imaginado que *Javascript* (o *Typescript*) pudieran llegar a gustarme tanto.

14.2. Juan José Tambo

El desarrollo de este proyecto ha resultado interesante y gratificante, ya que se han utilizado tecnologías actuales que no había usado con anterioridad. Asimismo, el tema del coronavirus sobre el que se basa el proyecto da pie a incorporar determinadas funcionalidades como lo son la incorporación de gráficas y datos externos.
Me ha sorprendido el potencial que tiene *React* junto a *Typescript*, ya que permite un desarrollo más sencillo del código al ser tipado e informar de ciertos errores de codificación. Cabe destacar que *Passport* ha sido clave para controlar el inicio de sesión de usuarios tanto de forma local como por medio de proveedores externos.

14.3. Jesús Villacampa

Este proyecto ha sido una gran experiencia. Se han aprendido nuevas tecnologías las cuales no tendría ni nociones de no ser por este proyecto.
Al ser un proyecto muy bien acotado, nos ha permitido a los miembros del grupo repartirnos el trabajo de manera que todos los miembros hemos "tocado" *frontend*, *backend* y *base de datos*, conociendo todos los componentes al sistema y el funcionamiento de los mismos. Esto ha producido que cada uno hayamos tenido la posibilidad de hacer un desarrollo *full stack*.
Por otra parte, estoy contento de que hayamos escogido *React*, ya que considero que es una tecnología que me puede ser de utilidad en el futuro.

14.4. José Félix Yagüe

El desarrollo de este proyecto ha sido entretenido y una experiencia que se diferencia del resto de asignaturas más enfocadas a la realización de un examen.
Además, debido a nuestra organización más horizontal, he tenido la oportunidad de implementar cosas de todos los aspectos de un proyecto de este tipo, y he podido aprender más como funciona el backend de un sistema de este tipo, ya que no había hecho anteriormente algo relacionado con esto. Además, esto nos ha permitido tener una visión más global del proyecto haciendo que podamos entender mejor como se comporta en su conjunto, y validar el trabajo de los demás, cosa que sería más complicada si solo nos enfocásemos en tareas más específicas.

Anexo 1: Modelo de datos

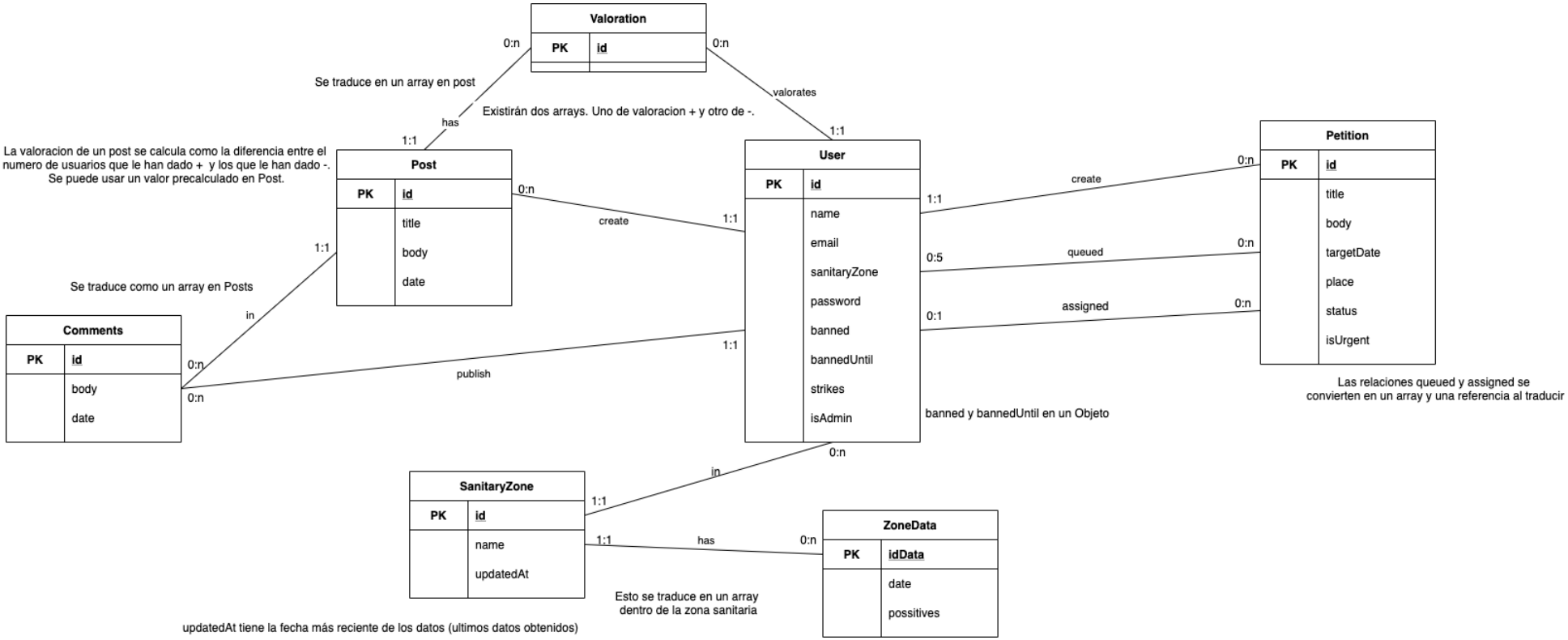

Figura 14: Modelo de datos del sistema

Anexo 2: Diagrama de Gantt

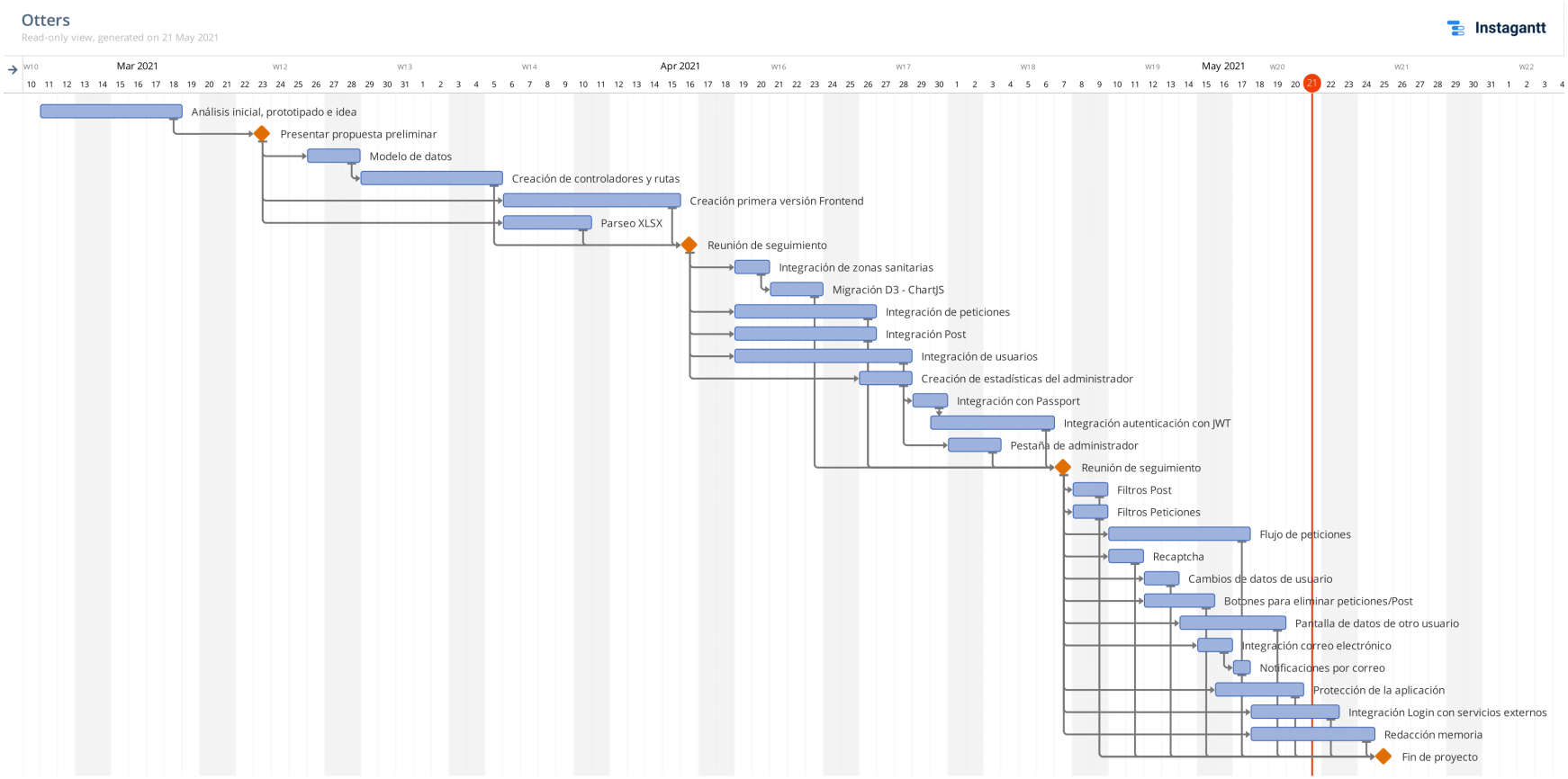

Figura 15: Diagrama de Gantt de la organización del proyecto

Anexo 3: Diagrama flujo de peticiones

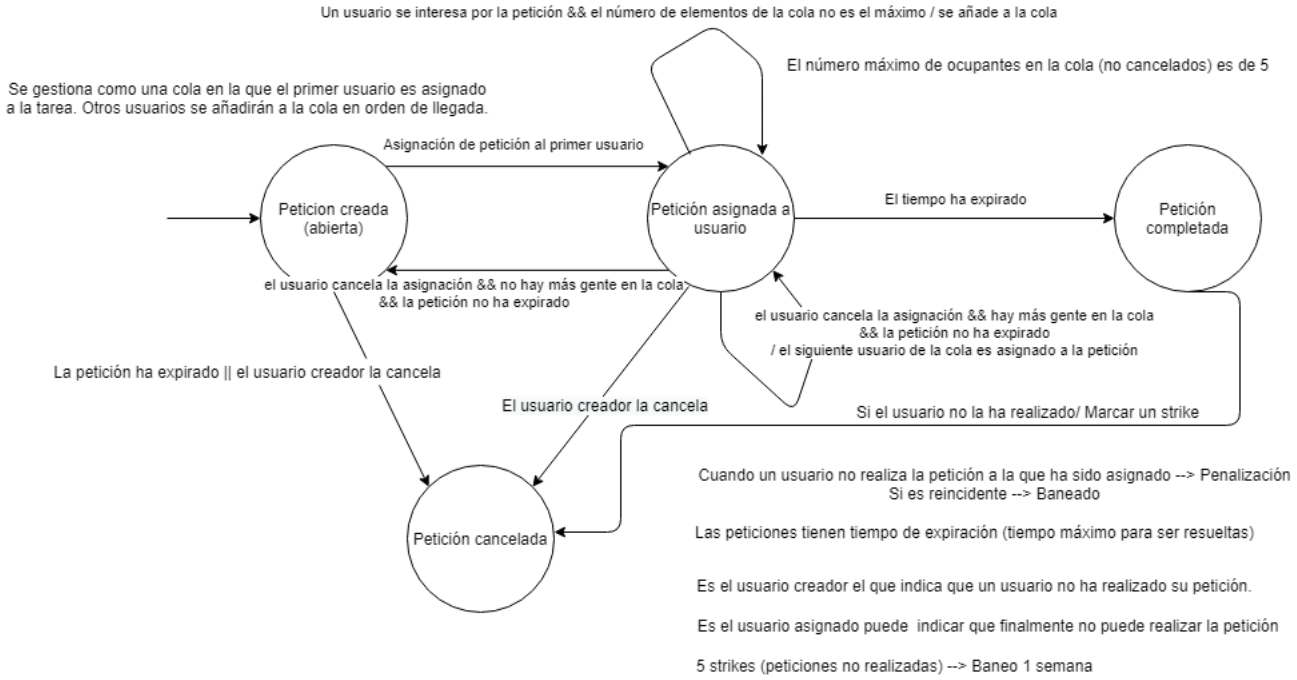

Figura 16: Diagrama de los flujos de las peticiones