# AragoJobLocator

**Sistemas y Tecnologías Web**

**Hector Ninian Herrmann Triviño**
**NIP 646946**

# Índice de contenido

## Credenciales de acceso

**URL de acceso a Heroku**
https://aragojoblocator.herokuapp.com

**usuario y contraseña de administrador de la aplicación**
Usuario: adm
Contraseña: admin

**usuario y contraseña de un usuario demo**
Usuario: demo
Contraseña: demo

**URL de acceso a Servidor Backend**
https://aragojoblocator-back.herokuapp.com

## Resumen

AragoJobLocator en un portal para poder acceder a las ofertas de empleo en Aragón creadas por los boletines oficiales de Zaragoza, Teruel y Huesca, de manera centralizada, a la par que también se pueden visualizar ofertas de empleo creadas por usuarios en la aplicación.

El objetivo es poder dar una visualización sintética de las ofertas y un sistema de filtrado sencillo y ágil para los usuarios, priorizando siempre las ofertas mas recientes.

La aplicación también permite enviar un mensaje a los publicadores de ofertas si son usuarios de AragoJobLocator.

## Propuestas Similares

En la búsqueda de páginas con funcionalidades similares, se encuentran tres tipos de páginas, los agregadores de ofertas, páginas de empleo, y las páginas gubernamentales.

Como páginas de empleo tenemos:

- **Linkedin e Infojobs**: Permiten filtrar ofertas por ciudad y puesto de trabajo, y permiten ver datos de la empresa. Linkedin tambien permite ver datos de trabajadores y gente contratada por la empresa publicante de la oferta.

Como agregador :

- **Google**: Las infoboxes de google dan una visión pequeña y rápida de ofertas de empleo que encajen con la búsqueda, obtenidas de distintas páginas de empleo.

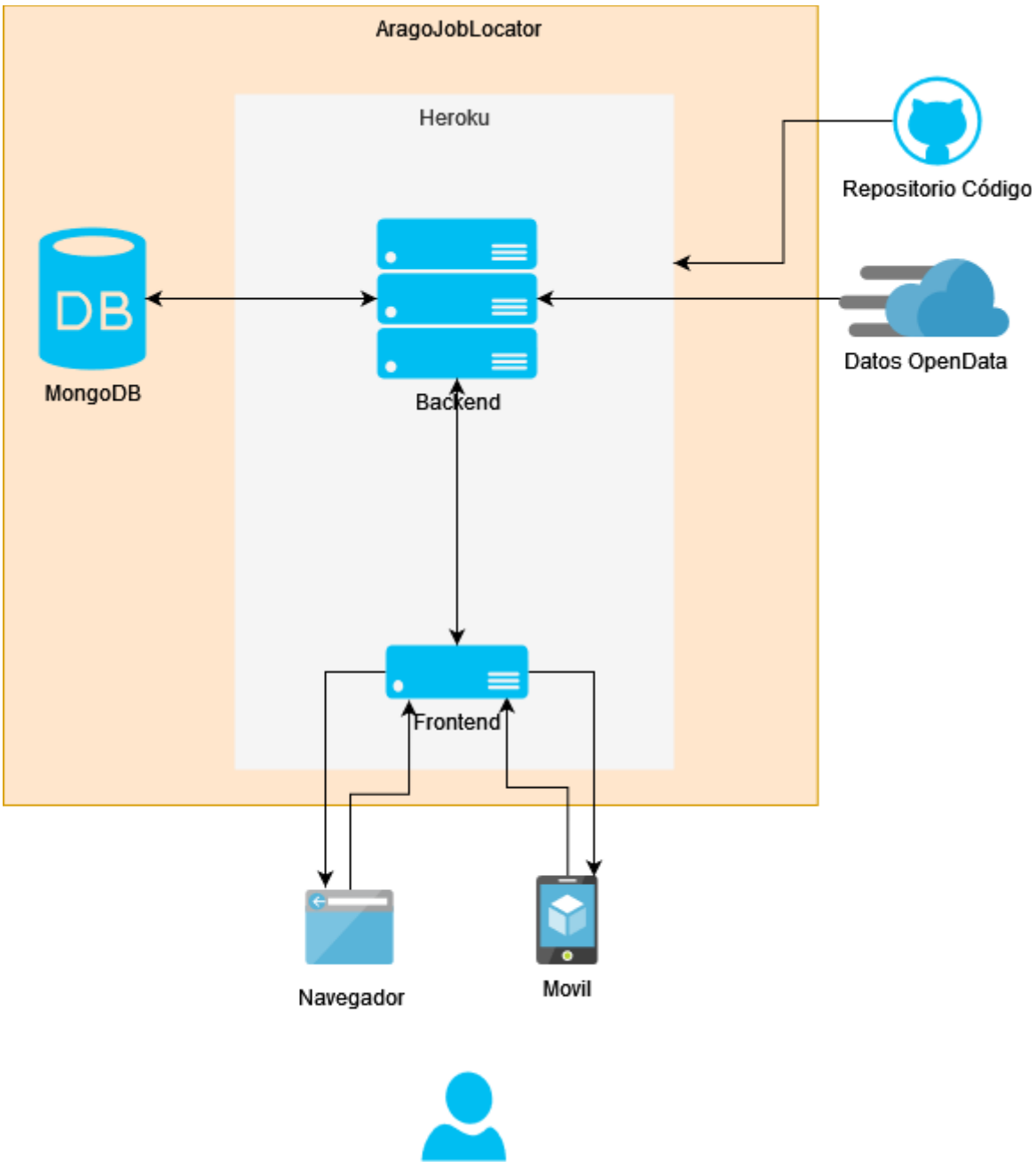
Como páginas gubernamentales:

- **BOA.aragon.es y Administacion.gob.es**: Son páginas que sólo contienen las ofertas publicadas por los boletines oficiales. También permiten filtrar por texto o localidad.

No se ha encontrado ninguna página que mezcle ofertas de usuarios/empresas directamente con las publicadas en los boletines. Se pueden seguir las cuentas en likedin del gobierno de aragón por ejemplo, pero sus ofertas vienen en forma de noticia. Ademas esta manera depende de la intervención del organismo para publicar las ofertas en linkedn, en vez de obtenerlas de una fuente de datos abierta de la comunidad de aragón.

## Arquitectura de Alto Nivel

La arquitectura de la aplicación es la siguiente:




La aplicación consta de dos capas principales, una capa de frontend con un servidor de frontend, que es el punto de acceso del usuario a la web, ya sea mediante navegador en el ordenador o desde el móvil.

El servidor de frontend realiza las peticiones necesarias al servidor de backend para obtener los datos que debe mostrarle al usuario y para realizar los cambios que el usuario haya hecho durante su uso de la página.

El servidor de backend atiende la peticiones del servidor de frontend, y es el encargado de la comunicación con la Base de datos.
Este servidor también es el encargado de tomar las ofertas de las fuentes de datos externas al sistema y persistirlas en la base de datos.

Ambos servidores están publicados en Heroku, y el código utilizado para el despliegue se encuentra en el repositorio de Github de la asignatura.

Por último, la base de datos es una base de datos MongoDB distribuida en cloud mediante AWS.

## Modelo de Datos

Para la base de datos, se ha empleado una base de datos NoSQL. Los esquemas empleados han sido los siguientes:

**Oferta**: Son los datos de las ofertas importadas y generadas por los usuarios. Sus campos vienen principalmente determinados por los los campos en la fuente de datos.

La fuente de datos es la siguiente:
https://opendata.aragon.es/GA_OD_Core/preview?view_id=305&_page=1

La aplicación añade el campo filters y createdByUser si es necesario.

{
**_id**: ID de Mongo
**fuente**: String
**f_publicacion**: Date
**denominacion**: String
**convocante**: String
**url**: String
**f_inicioPresentacion**: Date
**f_finPresentacion**: Date
**contacto**: String
**titulo**: String
**plazas**: Number
**filters**: String Array
**createdByUser**: Boolean
}

**User**: Contiene los datos del usuario. La password está encriptada. El campo perfil podría contener cualquier contenido, aunque se emplean los campos nom,apellodo y adicional, si existen.

{
**username**: String,
**email**: String,
**password**: String
**admin**: Boolean
**selfilters**: String Array
**last_check**: Date
**perfil**: JSON{
**nom**: String

**apellido**: String
**adicional**: String
}

}

**Mensaje**: Almacena los mensajes enviados de un usuario a otro. El modelo está pensado para poder tener una conversación bidireccional. Como al final no se ha implementado un sistema de mensajería tan complejo (se ha implementado un tablón de mensajes), so se le saca el máximo provecho a la estructura.

{
**titulo**: String,
**participante_1**: String,
**participante_2**: String,
**last_date**: Date,
**mensajes**: [{
**emisor**:String,
**contenido**:String,
**fecha_msj**:Date
}]
}

**Metadata**: Este es un esquema sencillo, usado para procesos internos y no para representar objetos. En la aplicación solo se usa para controlar la fecha que hay que utilizar para cargar los datos de la fuente externa.

{
**type**: String,
**value**: String
}

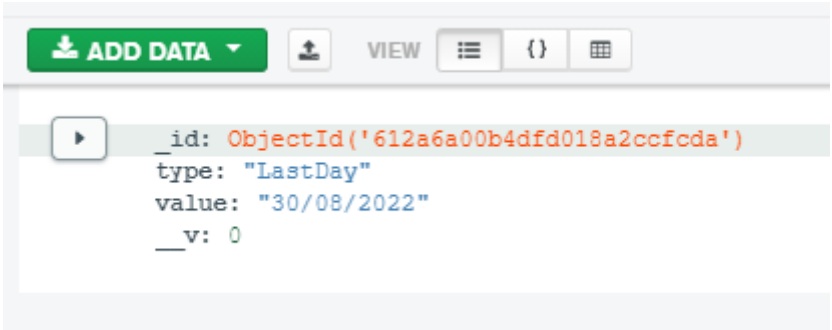



Figura 1: Toda la tabla de metadata

## API REST

La aplicación usa varios servicios en backend. Como no se ha llegado a implementar Swagger, no se tiene la API documentada, se resumen aquí las llamadas que acepta el backend.

En el código las funciones se pueden encontrar en back/controllers/XXXController.js y su ruta en back/routes/XXX.js

**Index**

**Ruta:** /
**Método:** GET
**Uso:** Simplemente devuelve “Piano”, para ver que funciona.

**Listado**

**Ruta: /**listado/
**Método:** GET
**Uso**: Devuelve el listado de Ofertas sin aplicar ningún filtro.

**Ruta: /**listado/
**Método:** POST
**Uso**: Devuelve el listado de Ofertas aplicando los filtros enviados en el cuerpo de la petición.

**Ruta: /**listado/crearOferta
**Método:** POST
**Uso**: Crea una oferta con los datos enviados en el cuerpo de la petición.

**Login**

**Ruta:** /login/
**Método:** GET
**Uso**: Devuelve si el usuario está logado o no, y si es administrador en caso de estar logado

**Ruta:** /login/
**Método:** POST
**Uso**: Realiza el login y devuelve el resultado de la operación.

**Ruta:** /login/logout
**Método:** POST
**Uso**: Hace logout y devuelve el resultado.

**Mensaje**

**Ruta:** /msj/userId
**Método:** GET
**Uso**: Devuelve el listado de los mensajes en los que el usuario actual es partícipe. Emplea los datos de sesión del usuario.

**Ruta:** /msj/create
**Método:** POST
**Uso**: Crea un mensaje con los datos enviados en el cuerpo de la petición.

**Perfil**

**Ruta:** /perfil/user
**Método:** POST
**Uso**: Devuelve los datos de perfil y filtros del usuario enviado en el cuerpo de la petición.

**PerfilConf**

**Ruta:** /perfilConf/profile
**Método:** GET
**Uso**: Devuelve los datos de perfil del usuario activo. Usa los datos de sesión del usuario

**Ruta:** /perfilConf/filters
**Método:** POST
**Uso**: Guarda los filtros seleccionados por el usuario.

**Ruta:** /perfilConf/date
**Método:** POST
**Uso**: Actualiza la última fecha en la que el usuario ha accedido a sus mensajes.

**Ruta:** /perfilConf/profile
**Método:** POST
**Uso**: Actualiza los datos del perfil del usuario.

**Register**

**Ruta:** /registro/
**Método:** POST
**Uso**: Realiza el registro del usuario y devuelve el resultado.

**Stats**

**Ruta:** /stats/offersCreated
**Método:** GET
**Uso**: Obtiene los datos estadísticos de Ofertas creadas por Día

**Ruta:** /stats/offersFilters
**Método:** GET
**Uso**: Obtiene los datos estadísticos de Ofertas por Filtro.

**Users**

**Ruta:** /users/getUsers
**Método:** GET
**Uso**: Devuelve la lista de usuarios sin utilizar parámetros en la búsqueda.

**Ruta:** /users/date
**Método:** GET
**Uso**: Devuelve la última fecha en la que el usuario ha accedido a sus mensajes.

**Ruta:** /users/filters
**Método:** GET
**Uso**: Devuelve el listado de filtros del usuario. Emplea los datos de sesión del usuario.

**Ruta:** /users/name
**Método:** POST
**Uso**: Devuelve el ID del usuario con el nombre especificado en el cuerpo de la petición

**Ruta:** /users/id

**Método:** POST
**Uso**: Devuelve el username del usuario con el ID especificado en el cuerpo de la petición

**Ruta:** /users/delete
**Método:** POST
**Uso**: Borra al usuario especificado en el cuerpo de la petición

**Ruta:** /users/getUsers
**Método:** POST
**Uso**: Devuelve la lista de usuarios utilizando los parámetros especificados en el cuerpo de la petición

## Implementación

### Backend

Para la implementación del backend se ha usado Node y Express. Y para la conexión con MongoDB se ha empleado Moongose como middleware.

El backend utiliza los siguientes ficheros:

*config.js* : Tiene la configuración del puerto y la dirección del cluster de MongoDB
*app.js* : Tiene la configuración del token de Session, para el cual se ha empleado Express-Session, donde se enlace la ruta con los routers y donde se realiza la especificación de CORS, y donde se programa el CRON para poblar la base de datos de manera diaria a las 2 am con los datos del día anterior. Adicionalmente, también lanza el cargado de la base de datos al arracar. Esto es para poder tener la base siempre actualizada, cuando el equipo no haya estado operativo a las 2 am, ya sea porque se este trabajando localmente, el dyno de Heroku se hubiera apagado al no haberlo usado.

*Carpeta Routes* : Los ficheros que enlazan las rutas de la API con los métodos de los controladores que ejecutan la tarea.
*Carpeta Controllers* : Los controladores con los métodos de la API
*Carpeta Models* : Los modelos de datos de la Base de Datos

*filterService.js* : Exporta un método para deducir los filtros que debería tener una nueva oferta.
*db.js*: Es el fichero que se encarga del cargado de datos en la base de datos. Toma los datos de la fuente de datos a partir de una fecha especificada previamente, que se guarda en BD.

### FrontEnd

Para el frontend se ha empleado Angular y Bootstrap. El enrutamiento se realiza mediante Routes de Angular.

Los elementos de frontend son los siguientes:

*gestion-users* : Componente para la gestión de usuarios. Sólo utilizable por un usuario administrador.
Contiene un buscador de usuarios, que busca usuarios con username que contenga el string buscado y devuelve una lista de enlaces a los perfiles públicos de los usuarios. Indica si no se ha encontrado ningún resultado.

*header* : Componente que es una barra de navegación añadida a todas las páginas. Dependiendo de si el usuario está logado o no, muestra unas opciones u otras.

*inicio* : Componente que es la pantalla de inicio. Muestra botones para acceder a otros componentes de la aplicación en función de si el usuario esta logado y si es administrador

*listado* : Componente que muestra los filtros aplicables y un botón de búsqueda. Al realizar la búsqueda, se visualiza la lista de las ofertas.

La búsqueda se puede hacer con o sin parámetros, y los filtros son filtros AND, es decir, al aplicar Zaragoza y Universidad, solo se buscaran las ofertas que cumplan ambos filtros, no cualquiera de ellos.

Si el usuario está logado, tendrá visible otro botón para buscar directamente usando los filtros que haya guardado en su perfil. Si no tiene filtros guardados, se hace una búsqueda sin parámetros.

Por último, desde los datos de la oferta, si la oferta ha sido creada por un usuario de la aplicación, se muestra un enlace a su perfil público. Si el usuario no existe, se reemplaza con una cadena de texto “Usuario eliminado”

Como nota, el filtro de ciudad debería mejorarse ya que al seleccionar un elemento no se puede volver a tener ninguno seleccionado a no ser que se recargue.

*login* : Componente de login del usuario. Indica si el usuario ya existe o si la contraseña es errónea, y cuando se hace login con éxito, redirige a inicio.

*oferta-form* : Componente que es un formulario para crear una oferta nueva. Es necesario estar logado para utilizarlo.
Muestra si falta algún campo obligatorio al intentar crear la oferta. Si se crea con éxito, redirige a inicio.

*perfil*: Componente que es el perfil público de un usuario. Si el usuario no existe, muestra una cadena de texto indicándolo. Si sí que existe, muestra los datos del perfil del usuario y sus filtros guardados.

Si se está logado, muestra un botón para enviar un mensaje al usuario.
Si además se es administrador, se mostrará un botón para borrar al usuario, y al borrarlo se redirigirá a la gestión de usuarios.

*perfil-conf* : Componente que es donde el usuario puede modificar sus datos de perfil, los filtros que quiere guardar y los mensajes que tiene.

Los filtros es mejorable, ya que no se visualiza qué filtros tiene guardados en ese momento.

En el campo de mis datos, los inputs sí que vienen precargados con los datos que tenga el usuario, si es que tiene datos ya guardados en esos campos.

En el tab de mensajes, puede ver los mensajes que otros usuarios (o él mismo) ha recibido. Si la fecha de alguno de los mensajes es posterior a la última vez que se visualizaron los mensajes, aparecerá marcado con un texto “Nuevo” en rojo a continuación del título.

*registro* : Componente que es la pantalla de registro. Indica si el usuario ya existe, si las contraseñas no coinciden o si falta algún campo. En caso de éxito, redirige al login.

*stats* : Componente con las estadísticas de la aplicación. Solo usable por un administrador.

En un inicio, se quiso usar un offcanvas para implementarlo, por la versión de bootstrap utilizada es demasiado antigua, y al subir de versión, daba incompatibilidades con otros elementos, por lo que se ha mantenido como un tab. Otra opción, era hacer un dropdown para seleccionar qué estadística visualizar.

## Modelo de Navegación

El modelo de navegación depende de si el usuario está logado o no o si es administrador. Resultando en 3 opciones diferentes.

Hay que añadir que se ha detectado el error de que un usuario no logado o no admin puede acceder a las zonas restringidas, aunque no pueda usarlas. Esto es porque en frontend no se valida al cargar las páginas que el usuario no sea de la categoría erronea. Solo se comprueba en los componentes cuya vista depende del tipo de usuario al que está sirviendo.

Los métodos no funcionan sin los permisos adecuados, por lo que no hay riesgo de que usuarios sin credenciales realicen actividades prohibidas para ellos, pero considero que no deberían poder cargar la página desde un inicio. (Por ejemplo, a la página de estadísticas puede acceder cualquiera que introduzca la ruta, pero solo un admin verá los datos)

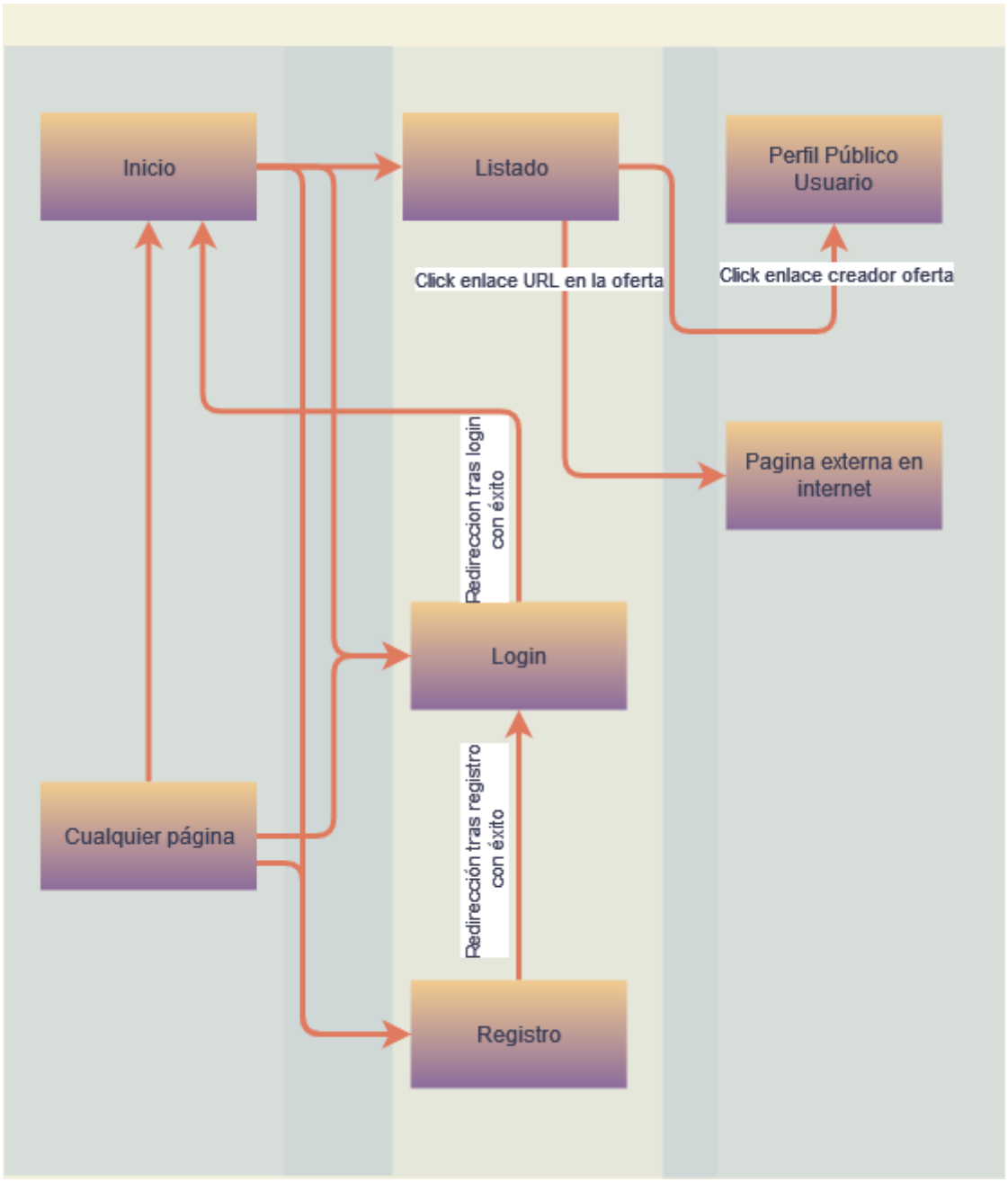



Figura 2: Navegación usuario no logado

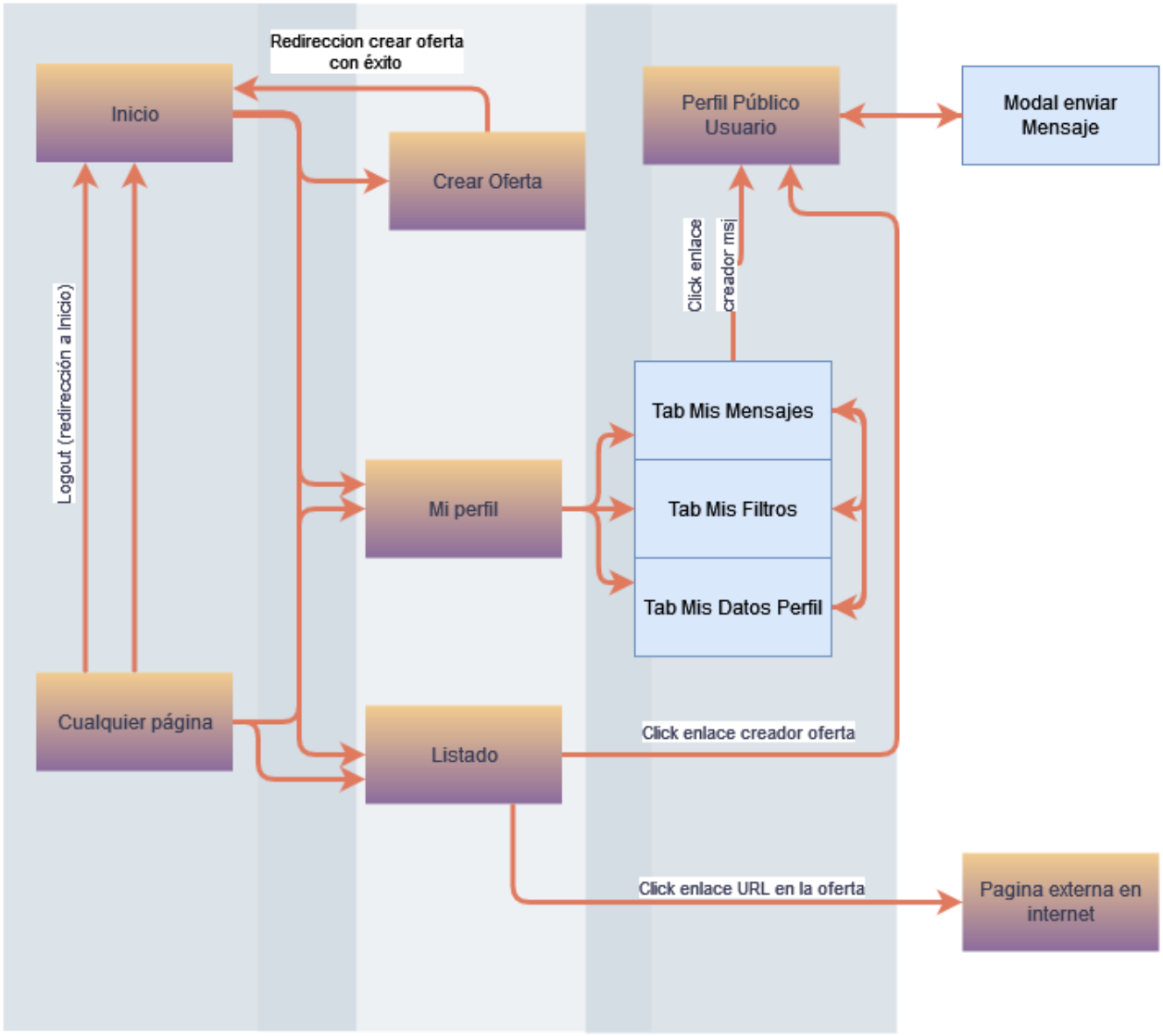



Figura 3: Navegación usuario logado

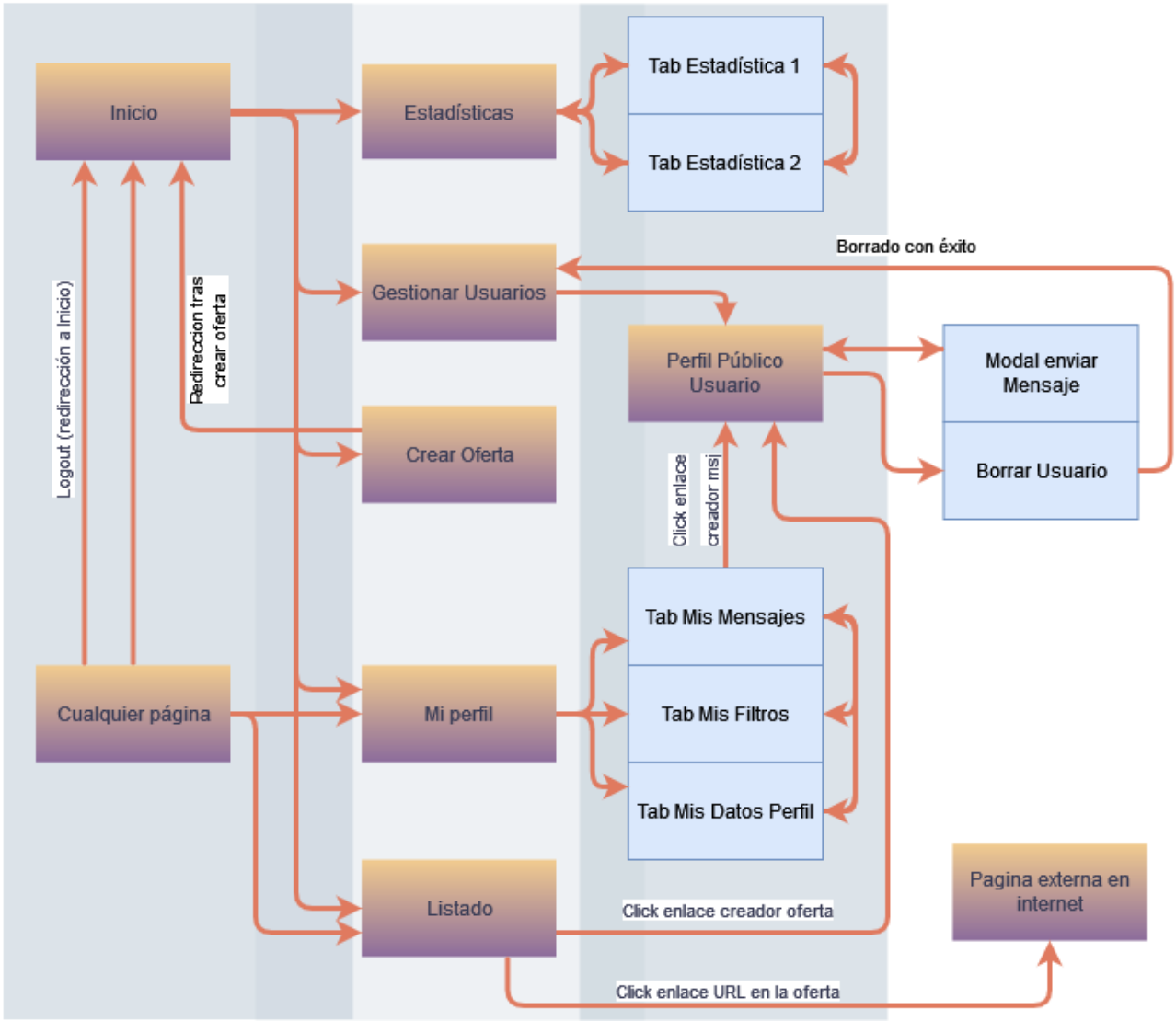



Figura 4: Navegación usuario administrador

## Analíticas

Para la generación de las analíticas se ha empleado ng2-charts, que emplea ChartJS, los datos se obtienes mediante agregación en las consultas de Mongo y en el caso de ofertas por día, se ha tenido que hacer un proceso de map-reduce para pasar las fechas de la Base de Datos, que están en formato ISO, a un formato visible y sin hora para poder agruparlas por día.

Estas son las analíticas:

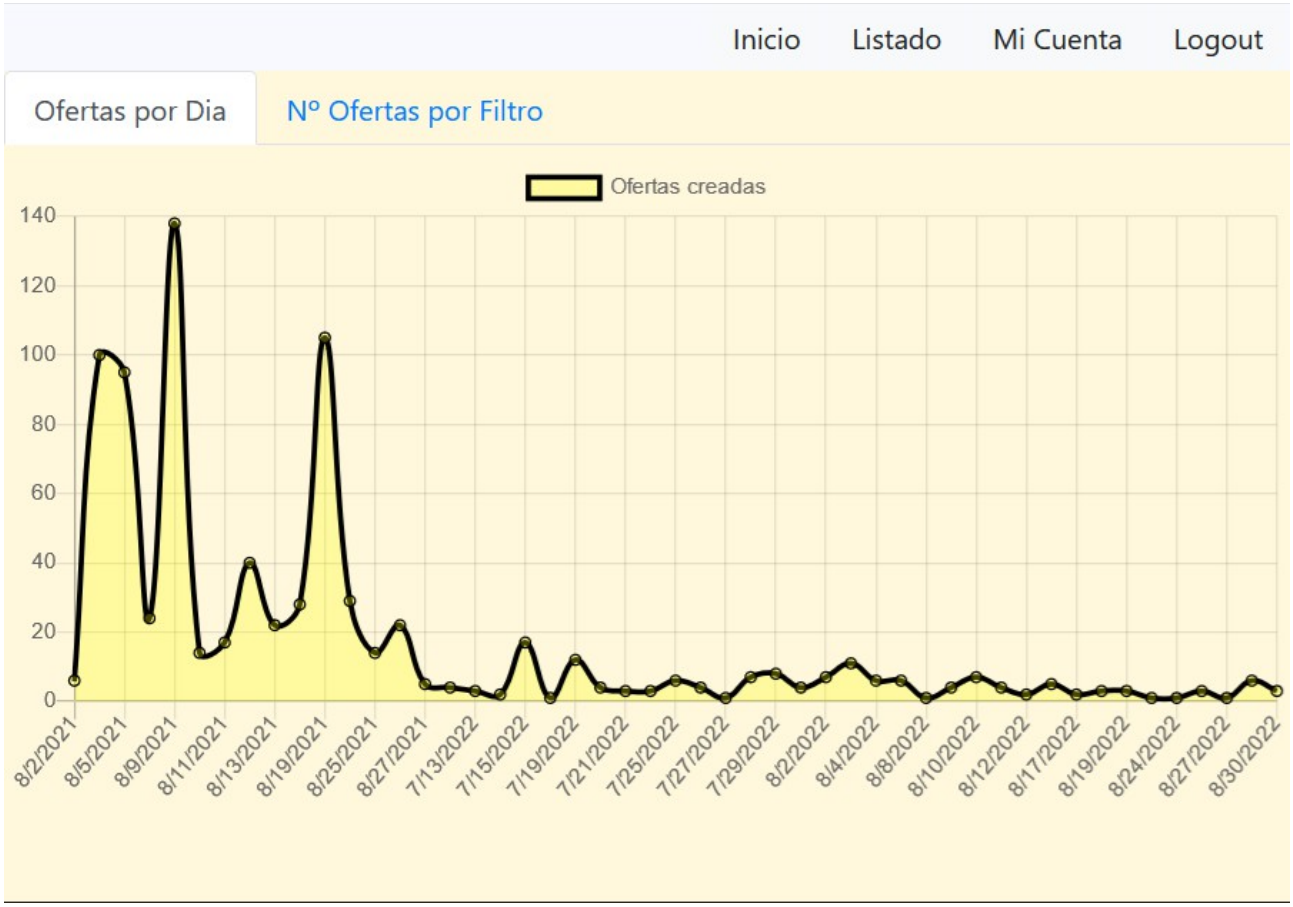



Figura 5: Ofertas por día

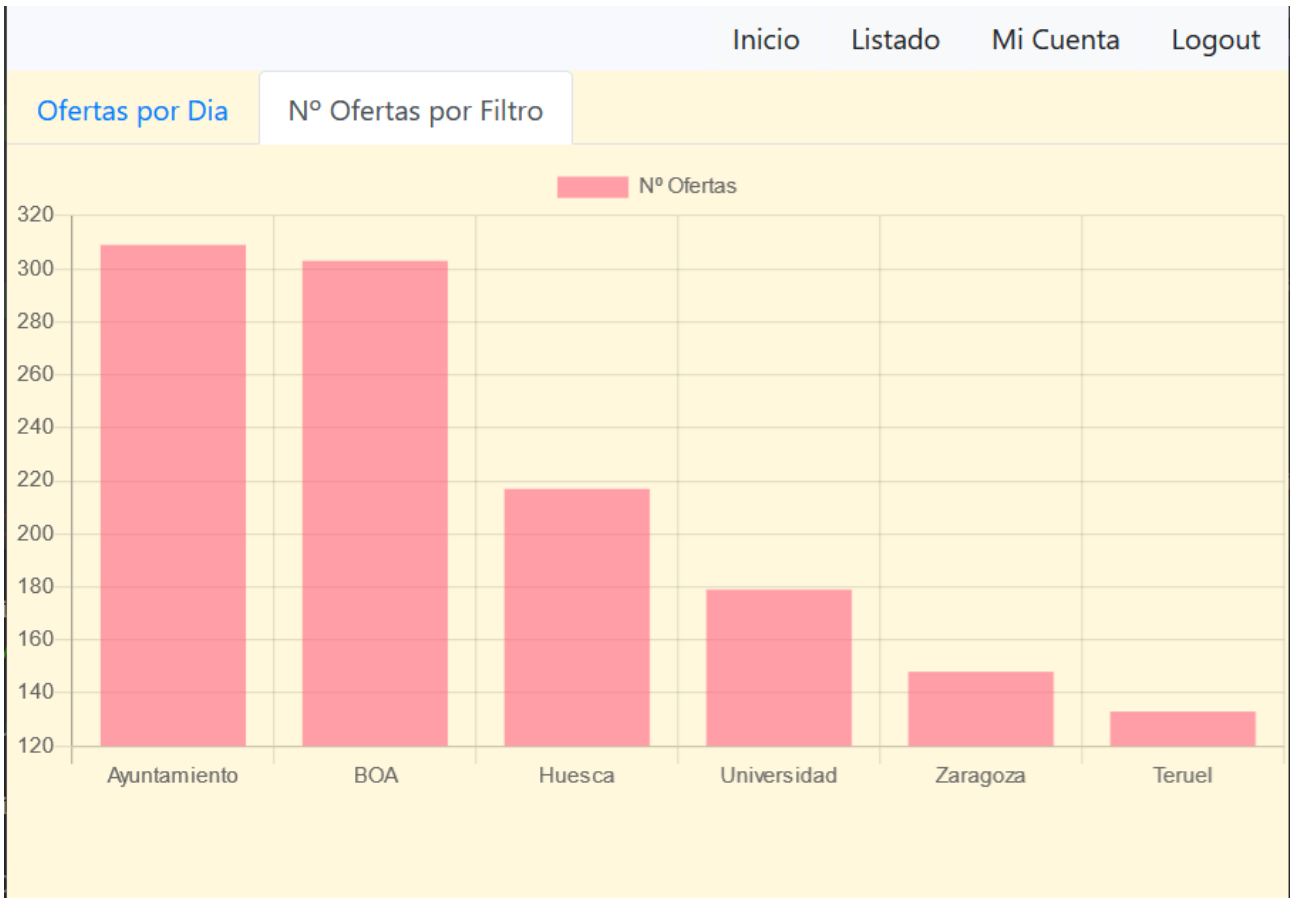



Figura 6: Ofertas por Filtro

## Despliegue del Sistema

Para el despliegue del sistema se ha utilizado Heroku. Como hay que desplegar dos aplicaciones, no se puede trabajar solo con una rama.

El procedimiento de trabajo ha sido crear dos nuevas ramas, frontdeploy y backdeploy, que contendrán el código que se desplegará en Heroku, y se deja la rama Main para desplegar localmente.

Al añadir nuevas funcionalidades y correcciones, se realizan en la rama main, y tras probar su correcto funcionamiento, se hace push, y se mergean los cambios en frontdeploy y backdeploy, y finalmente se despliegan esas ramas en heroku.

Para ello es necesario crear un remote por aplicación, en mi caso, han sido heroku para el frontend y heroku-back para el backend.

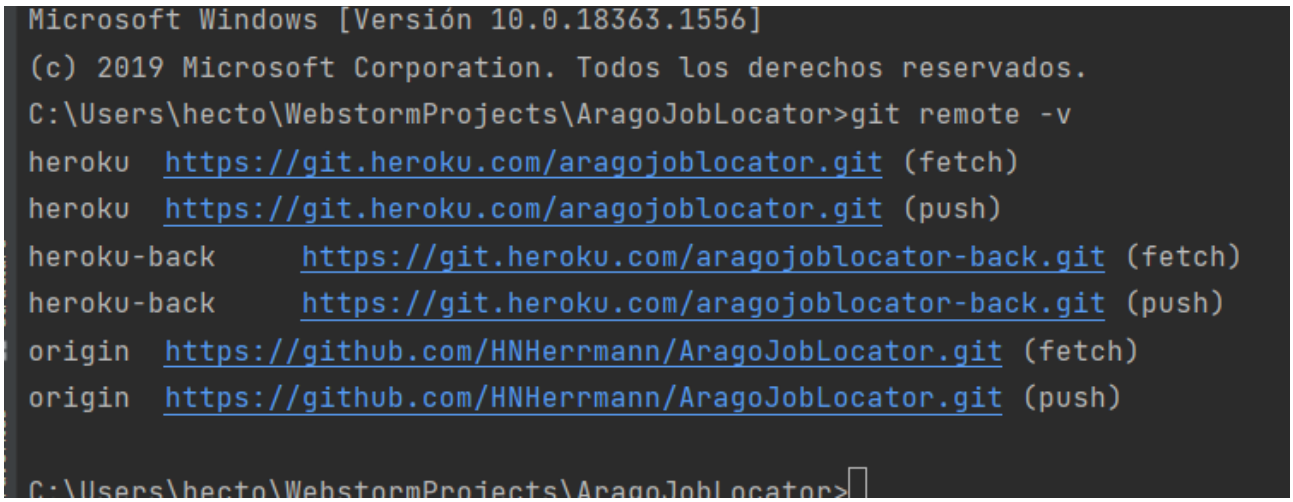


Figura 7: Remotes

De esta manera se puede desplegar mediante “git push heroku frontdeploy:main” y “ git push heroku-back backdeploy:main”.

Sin embargo, para que funcione hay que hacer algunos cambios específicos en cada rama.

**Backend**

Primero es necesario anadir un fichero Procfile indicando el comando para iniciar. En este caso es “node ./back/bin/www”

Luego es necesario modificar la dirección en la configuración de CORS sustituyendo localhost:4200 por la ruta del frontend en heroku.

Como añadido, aunque no se ha hecho, sería recomendable eliminar los ficheros y dependencias de frontend de la rama a pushear a heroku, ya que el servidor de backend no empleará esas bibliotecas y ficheros y solo sobrecarga el deployment a heroku.

**Frontend**

En este caso, se ha añadido un fichero server.js para indicar dónde está el contenido estático de la aplicación, la pantalla de inicio y el puerto.

Y también se crea Procfile con “node server.js”.

Luego se sustituye la dirección del backend de localhost:3000 a la dirección del backend en heroku.

Finalmente, se añade el parámetro - -prod a build en el package.json, para que el contenido estático generado esté generado pensado para ser usado en un entrono real.

## Problemas encontrados

A lo largo del desarrollo de la aplicación, han ocurrido algunos problemas.

- **Alto tiempo entre desarrollos** : Debido a que se trata de una aplicación desarrollada de manera individual, y en periodos “libres” de trabajo y asignaturas, parte del código se implementó en verano de 2021, parte en febrero/marzo de 2022 y parte en verano de 2022. Esto ha traído la complicación de que, algún elemento externo, como la base de datos de OpenData de donde se extraen los datos, había sido modificada ligeramente.

  Adicionalmente, a la hora de evaluar el trabajo completado y reempezar el proyecto de nuevo, se ha encontrado que cosas que se consideraban acabadas, no lo estaban completamente y faltaban algunos detalles por completar.

  Todo esto ha implicado un mayor tiempo de desarrollo del inicialmente estimado.

- **Pocos datos para algunos filtros** : Debido a que algunos días no se publica ninguna oferta en el BOA, u solo 1 o 2, no siempre se tienen datos para todos los filtros. Por ello he dejado en la base de datos los datos repetidos de ofertas de 2021 que se obtuvieron cuando se estaba depurando el proceso de carga.

- **CORS con métodos que no devolvían nada** : Este era un error curioso detectado tras desplegar en Heroku, y es que al devolver 200 sin body, la aplicación daba un error, aparentemente de CORS, entiendo que al hacer timeout esperando algo en el cuerpo de la respuesta. Es por ello, que varias peticiones devuelven un json con algún mensaje cualquiera.

- **Problemas con las fechas** : Debido a que en la fuente de datos, la fecha es una cadena de texto, ha habido que realizar transformaciones con la fecha para guardarlo en la Base de Datos como fecha, y luego volverlo a transformar a cadena para la visualización de datos. Por lo que ha habido que realizar varias transformaciones del dato. En particular, en una de ellas, al crear un Date nuevo, estaba (asumo que porque tomaría el formato americano) cruzando el mes y el día si el día era menor a 13. Es decir, un 12/8/2022, lo interpretaba como el 8 de Diciembre de 2022.

- **"function String() { [native code] }"** : Por algún motivo, al aplicar los filtros a las nuevas ofertas, siempre se genera este valor como primera entrada del array. Por mas que se ha investigado el asunto, no se ha encontrado solución, por lo que se ignora cuando se obtienen los filtros de la oferta.

- **Versiones viejas de código** : Quizás en parte por ser un proyecto iniciado en 2021, en algunos casos la documentación revisada contenía objetos o dependencias no disponibles en la versión utilizada, a veces no pudiendo subir de versión debido a que eso generaba otras incompatibilidades en otra área del código. Esto principalmente ha afectado al offcanvas de bootstrap y la documentación de chartjs, que me ha forzado a usar ng2-charts.

- **Poca habilidad personal con css** : Personalmente no tengo mucha experiencia modificando elementos mediante css y eligiendo colores agradables

a la vista, por lo que dejar un frontend con buenos colores / márgenes /letra, creo que me ha llevado mas de lo que debería. Recuerdo perder mucho tiempo en 2021 haciendo las primeras pantallas de front y la cabezera.

## Problemas potenciales

Ya se ha comentado el problema de la falta de un control total de la navegación. Se debería comprobar las credenciales en todas las pantallas de frontend, y no solo en backend.

Debido a que no quería depender sólo de los datos de ofertas abiertas, la aplicación no presenta ningún tipo de política de borrado de la base de datos. Es evidente que esto es un problema que eventualmente en un entorno real, dispararía el uso de la base de datos hasta quedarnos sin cuota o aumentaría el precio de la aplicación. La política lógica, sería borrar las fechas cuyo plazo de presentación ya haya expirado.

En el caso de la generación de la url del perfil público de un usuario desde mensajes, oferta y el buscador de usuarios, se realiza un número potencialmente elevado de peticiones al servidor de backend. Para cada entrada, se llama a la API para obtener el id y generar la url en frontend.

Por ejemplo, si se obtienen 100 ofertas creadas por usuarios en el listado, se hacen 100 peticiones al backend. Si tuviéramos varios usuarios concurrentes usando la app, básicamente nos podríamos estas haciendo un DDOS a nosotros mismos.

La aplicación permite introducir al usuario urls al crear una oferta. Esto es un problema ya que no se valida que sea segura. Habría que emplear alguna API de terceros como google safe browsing o similares.

De cara a la corrección, el código aún está sucio con clases que no hacen nada, obtenidas de utilizar otros proyectos y código de internet como referencia/base para la aplicación y aún tiene logs. La aplicación funciona igual, pero de cara a corregirla puede ser peor.

## Distribución del tiempo

Debido a la naturaleza del desarrollo que he llevado, que era intentar sacar la asignatura sin meterle mucho tiempo y haciéndolo en dos veranos, es muy complicado medir cuánto tiempo se ha invertido en el proyecto, ni tampoco he llevado ningun diagrama de gannt, kanban o scrum para medirlo.

Adicionalmente también está el hecho de que sí que invertí tiempo en el proyecto que abandoné el año pasado, que sí que ha servido para “ahorrar” tiempo al saber ya cómo hacer según que cosas, lo que dificulta aún mas saber cuanto tiempo he invertido en AragoJobLocator en concreto.

De todas maneras, puedo determinar que mi distribución del tiempo ha sido mala,

principalmente debido a dejarme un par de cosas “sencillas” para la última semana, que luego han resultado llevarme mas tiempo, y también medir mal el tiempo necesario para escribir la memoria y considerar que me daba tiempo a escribirla el último día tras salir del trabajo, como ya puedo confirmar que no es el caso al ser la 1 am del 31.

## Conclusiones y Valoración

Aún y con todas las dificultades que he tenido con la asignatura debido a romper con el grupo anterior y hacer la asignatura a destiempo, y a organizarme mal yo mismo, considero que la experiencia a sido positiva, y me parece una asignatura importante para el desarrollo como ingeniero informático.

Nunca había hecho una web desde 0, y me parece muy potente tener la capacidad de crear webs/apis sencillas en un relativamente corto periodo de tiempo. Tampoco había trabajado con promesas, y ha sido interesante también, aunque considero que podría haber cacharreado más con observables y demás elementos y con elementos de frontend mas avanzados como animaciones. En comparación con otras aplicaciones con páginas de inicio chulas, y con animaciones y florituras, la mía es un pequeño frankenfrontend.

Aunque no vaya a ver todo al hacer el proyecto individual (Protactor, y E2E), (Swagger ya lo conozco de java, asumo que su uso no varía mucho de node a java spring), también he tenido que tocar la aplicación a todos sus niveles, por lo que sí que me quedo contento al desarrollar la aplicación entera, en vez de solo una parte, que es lo que a veces pasa en los grupos al dividir un proyecto.

En cuanto a la asignatura y el profesorado, no tengo ninguna queja. Considero que el material es adecuado para el desarrollo y aprendizaje del proyecto, y las sesiones siempre han estado disponibles incluso durante la época de Covid. Además, se me ha adaptado a mi situación particular y siempre ha habido seguimiento por el profesorado.

En resumen, ha sido buena experiencia y se la recomendaría a cualquier alumno que no haya creado una aplicación desde 0.