AragoJobLocator

Sistemas y Tecnologías Web

Hector Ninian Herrmann Triviño
NIP 646946

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.

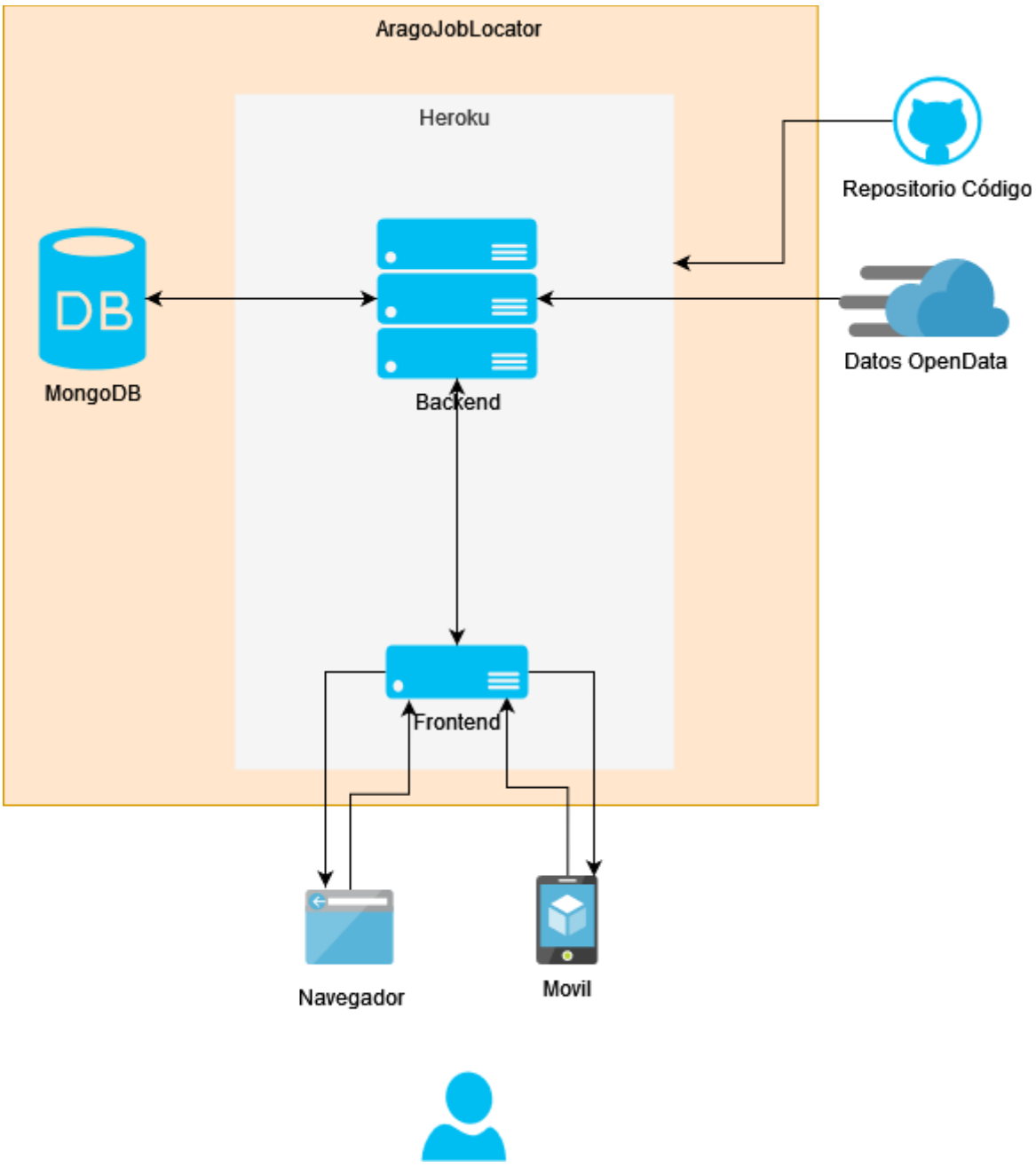
Como páginas gubernamentales:

- **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
}

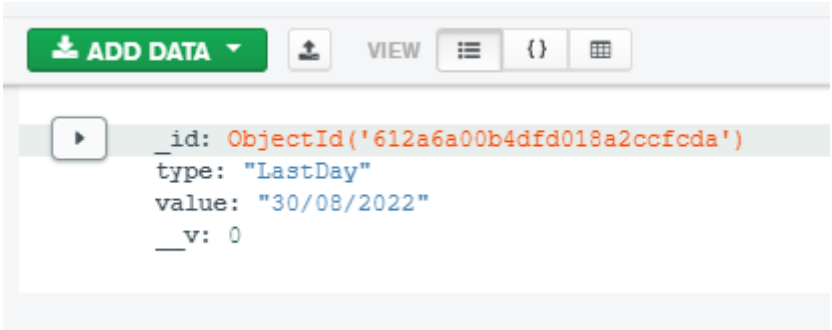


Figura 1: Toda la tabla de metadata

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.