PROYECTO: SISTEMA WEB ADOPTAMASCOTA

**DESCRIPCIÓN:**

Portal para gestión de procesos de adopción de mascotas, permite comunicar personas dispuestas a poner en adopción a sus mascotas, con personas que buscan adoptar una mascota.

**MIEMBROS DEL EQUIPO DE DESARROLLO:**

Benjamín Naranjo y Jorge Roncel

Darío Álvarez y Carlos Espinoza

Vicente Ríos y Oussama Boukich

**OPERACIÓN DEL PORTAL:**

La operación normal del portal se descompone en varios pasos, unos generales de acceso y unos específicos dependiendo de la tarea:

En breves rasgos, la operación del portal, se componen de:

1. Registro de Usuario.
2. Login al Portal, e ingreso a pantalla de bienvenida del perfil del usuario.
3. Crear la mascota y gestionar los datos de esta.
4. Poner una mascota en adopción o buscar alguna de las existentes para solicitar la adopción.

**MICROSERVICIOS QUE COMPONEN EL PORTAL**

El portal ha sido diseñado con una arquitectura de microservicios de Backend y Frontend, diseminada de la siguiente forma:

* Frontend con el patrón de uso de múltiples API’s para una única interfaz común
* Microservicio API REST de Gestión de Usuarios
* Microservicio API REST de Gestión de Mascotas
* Microservicio API REST de Gestión de las Adopciones

**RESPONSABILIDADES DEL PROYECTO:**

|  |  |
| --- | --- |
| **Componente** | **Asignado a:** |
| FRONTEND | Todos los miembros del equipo, divida según los componentes relacionados a cada proceso. |
| API REST GESTIÓN DE USUARIOS | Darío Álvarez y Carlos Espinoza |
| API REST GESTIÓN DE MASCOTAS | Vicente Ríos y Oussama Boukich |
| API REST GESTIÓN DE ADOPCIONES | Benjamín Naranjo y Jorge Roncel |

|  |
| --- |
| MICROSERVICIO ASIGNADO: GESTIÓN DE ADOPCIONES  Por:  Benjamín Naranjo y Jorge Roncel |

1. **URLS DE REPOSITORIOS**

**BACKEND - URL GITHUB:** <https://github.com/jorgeron/FISGrupo2-adoption-backend>

Branch Final: master

**BACKEND – URL INTEGRACIÓN CONTINUA:** <https://travis-ci.org/jorgeron/FISGrupo2-adoption-backend>

**BACKEND – URL DESPLIEGUE:** <https://adoptions-apibackend.herokuapp.com/>

**FRONTEND - URL GITHUB:** <https://github.com/vicnete/FISGrupo2Front>

Branch correspondiente a nuestro desarrollo: adoption-master

**FRONTEND – URL INTEGRACIÓN CONTINUA:**

<https://travis-ci.com/vicnete/FISGrupo2Front>

**FRONTEND– URL COMÚN DE DESPLIEGUE:** [**http://front-apibackend.herokuapp.com/**](http://front-apibackend.herokuapp.com/)

1. **NIVELES DE PRESENTACIÓN**

**A) NIVEL DE MICROSERVICIO INDIVIDUAL: AVANZADA, con cuatro características de las propuestas como adicionales.**

Nuestro microservicio añade a las características del microservicio básico lo siguiente:

**- Implementación de pruebas unitarias utilizando mocks y/o stubs**

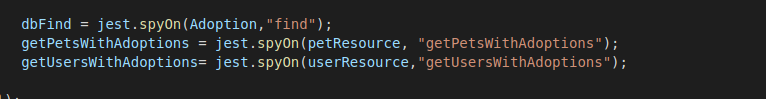
MockImplementation de Autenticación:



MockImplementation de DBSAVE

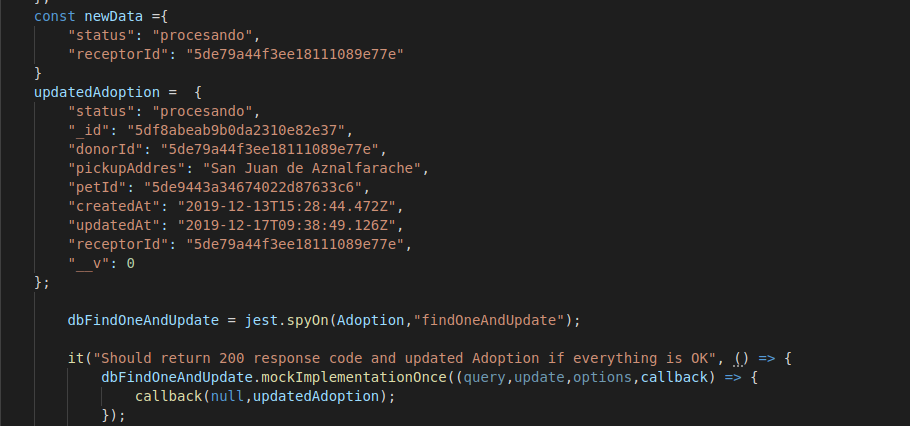


MockImplementation de procesos de Integración con otras API:



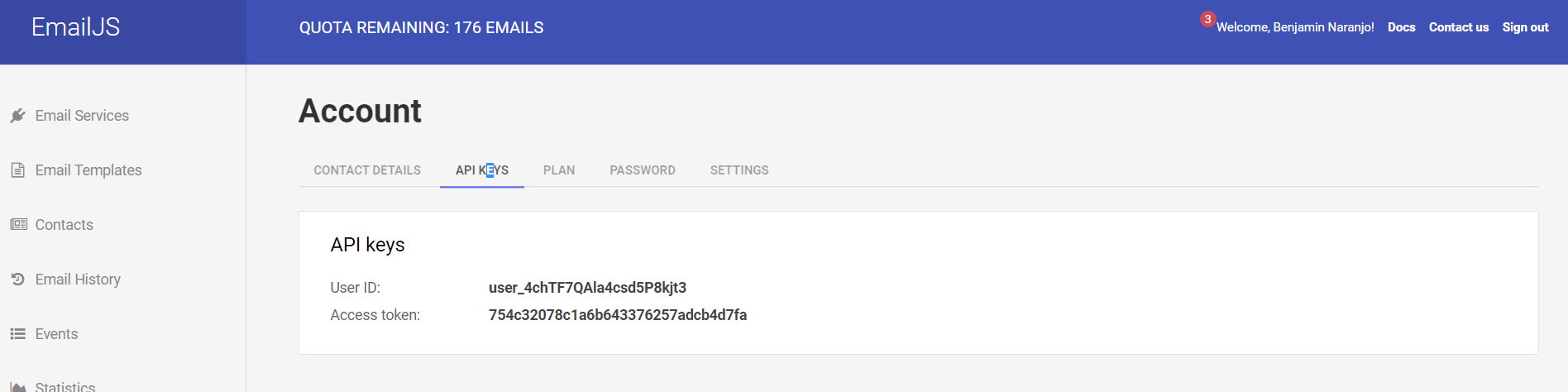


MockImplementation de DBFINDANDUPDATE

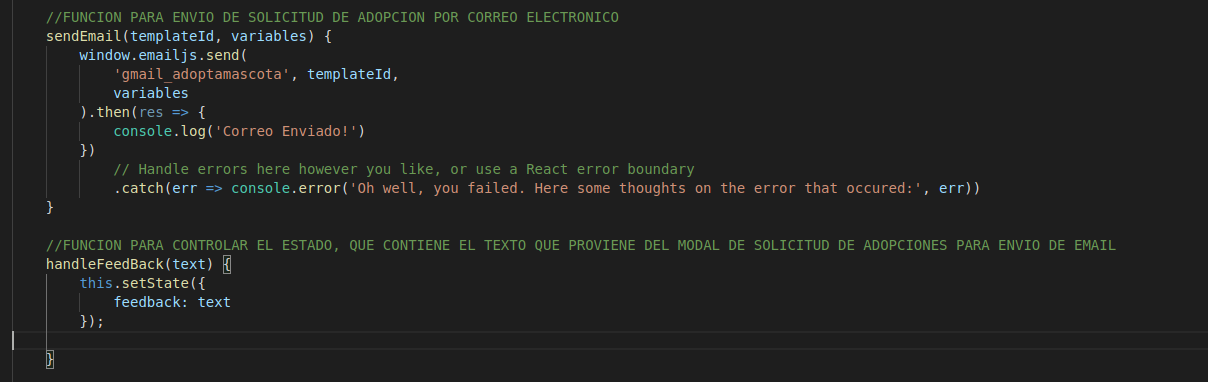


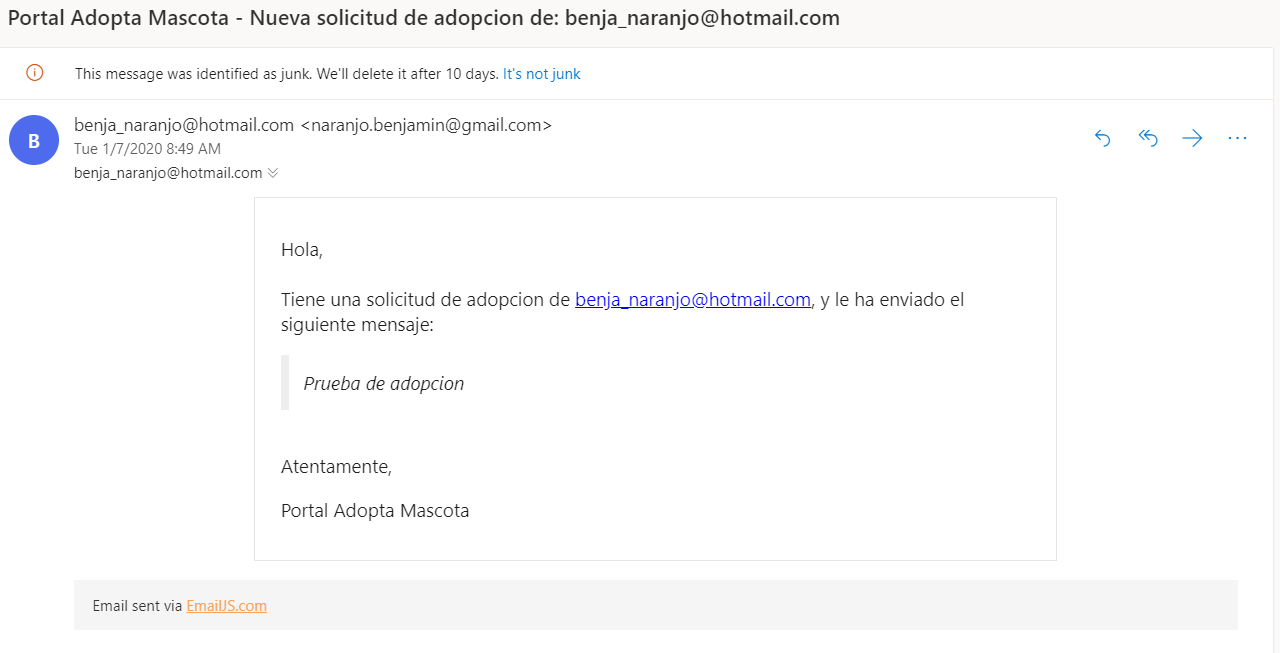
**- Consumo de algún API externa (distinta de las de los grupos de práctica).**

Adición de API de EmailJS en FRONTEND, para envío de solicitud de adopción por correo electrónico.



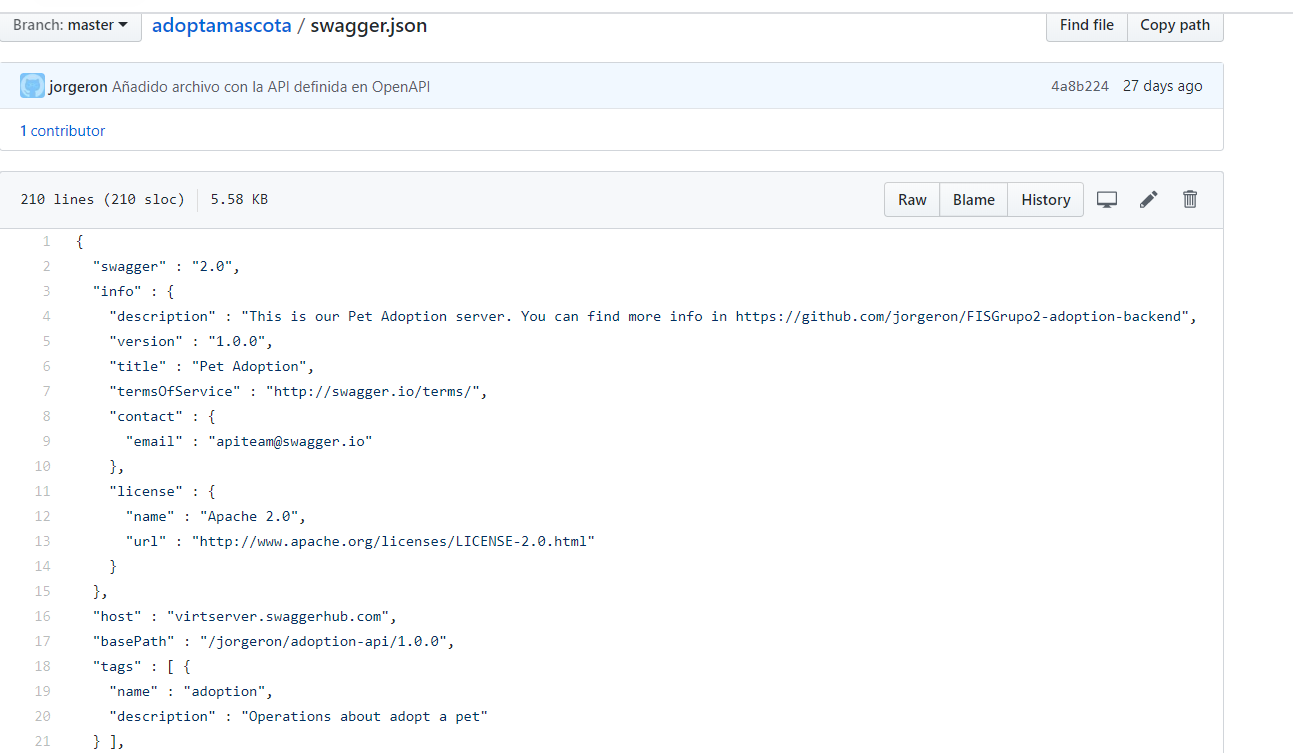






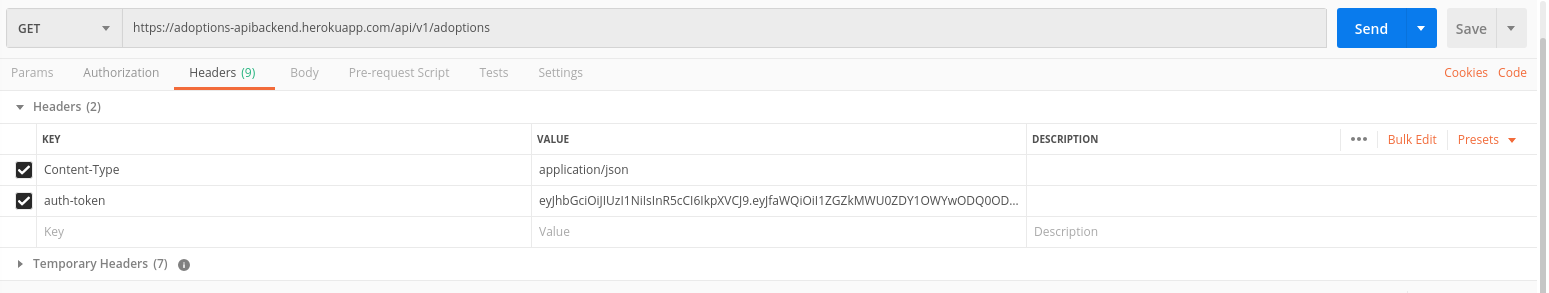
**- Tener el API REST documentado con swagger.**

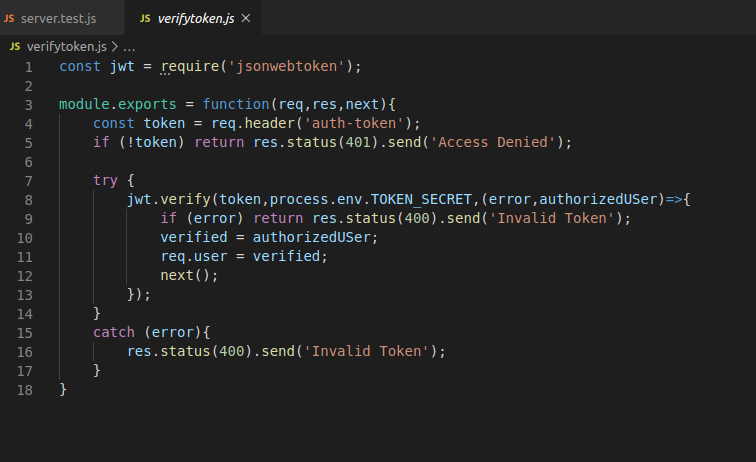
**URL:** <https://app.swaggerhub.com/apis/jorgeron/adoption-api/1.0.0>

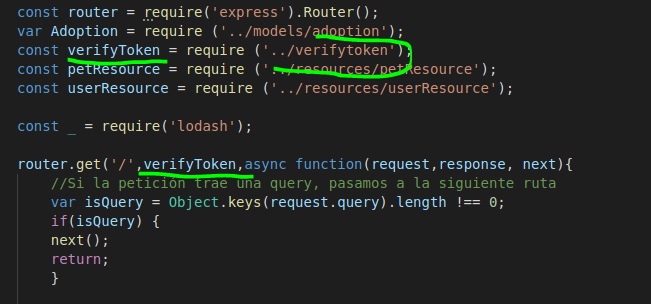


**- Implementación de un mecanismo de autenticación más completo**

Procedimiento de verificación de token a cada request de la API Backend





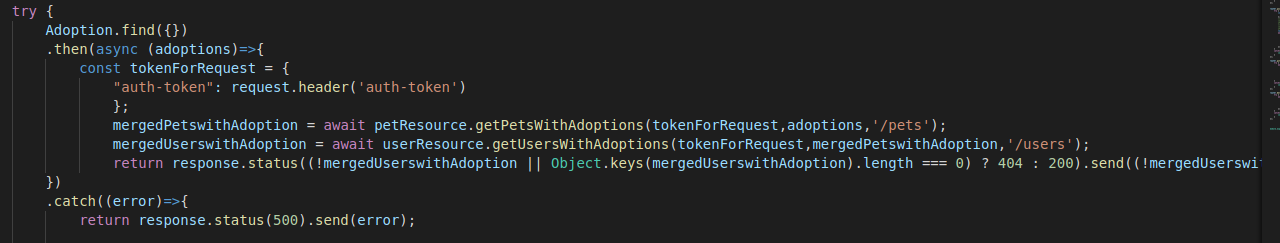


**B) NIVEL DE APLICACIÓN COMPLETA: BÁSICO MÁS DOS CARACTERÍSTICAS AVANZADAS**

**A nivel de aplicación completa presentamos, el cumplimiento del nivel básico, con una respuesta unificada de las adopciones con todos los datos de la mascota y del usuario respectivo, así como dos características del modo avanzado**

**Característica nivel básico**

- Interacción completa entre todos los microservicios de la aplicación integrando información.

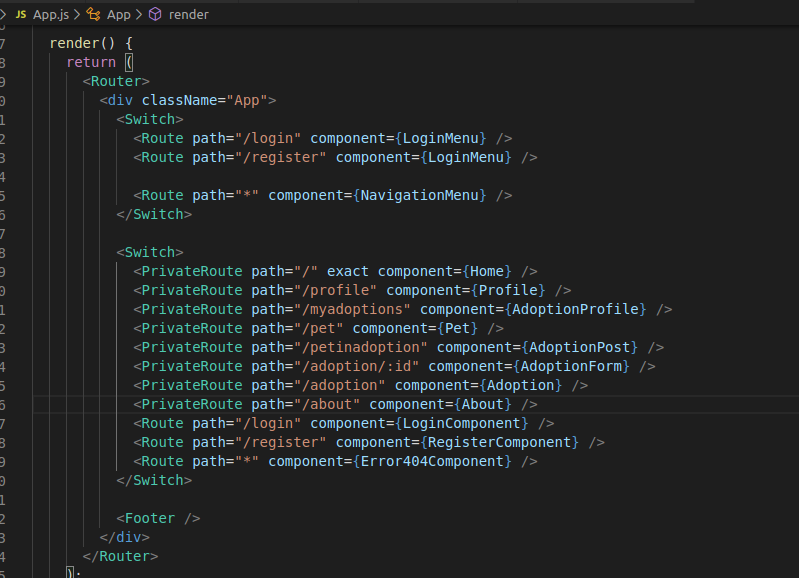




**Características nivel avanzado**

**- Tener un front end común común que integre los front ends de cada uno de los microservicios.**

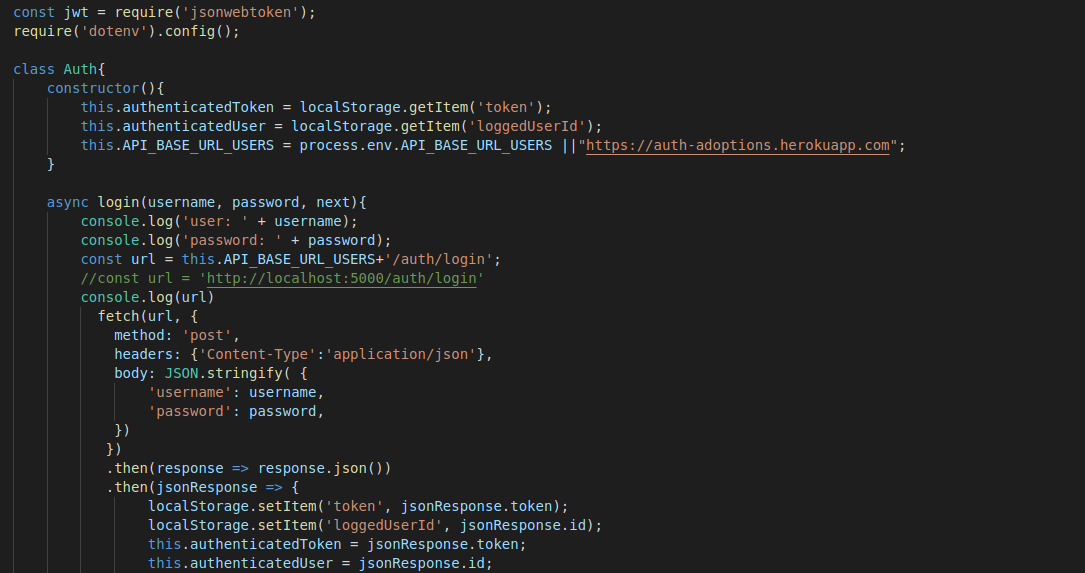
Interfaz única con las rutas a cada componente que gestiona un recurso API de la aplicación

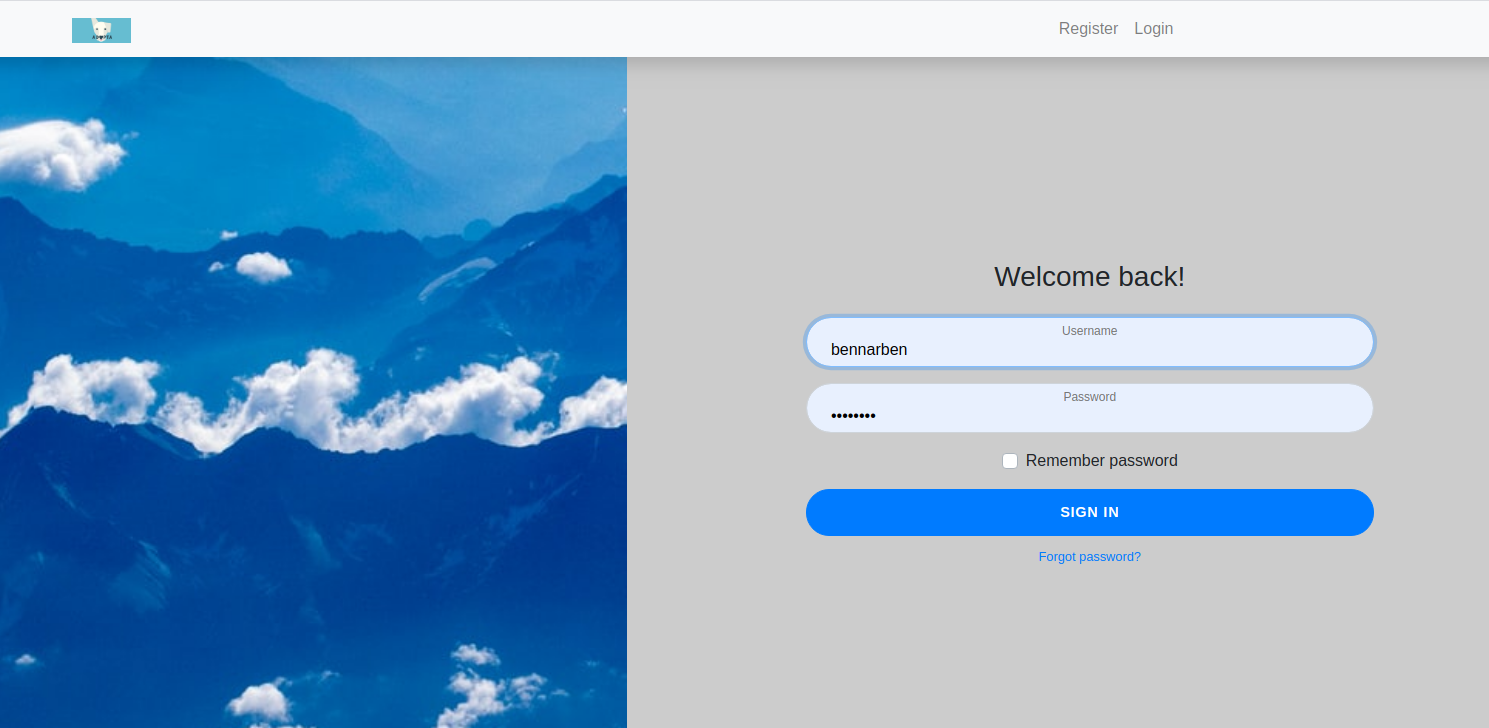




**- Implementación de un mecanismo de autenticación homogéneo para todos los microservicios.**

Todo el Frontend gestiona las peticiones utilizando la misma autenticación JWT, con la base de datos de usuarios, para la gestión de todos los microservicios a través de un único login unificado.



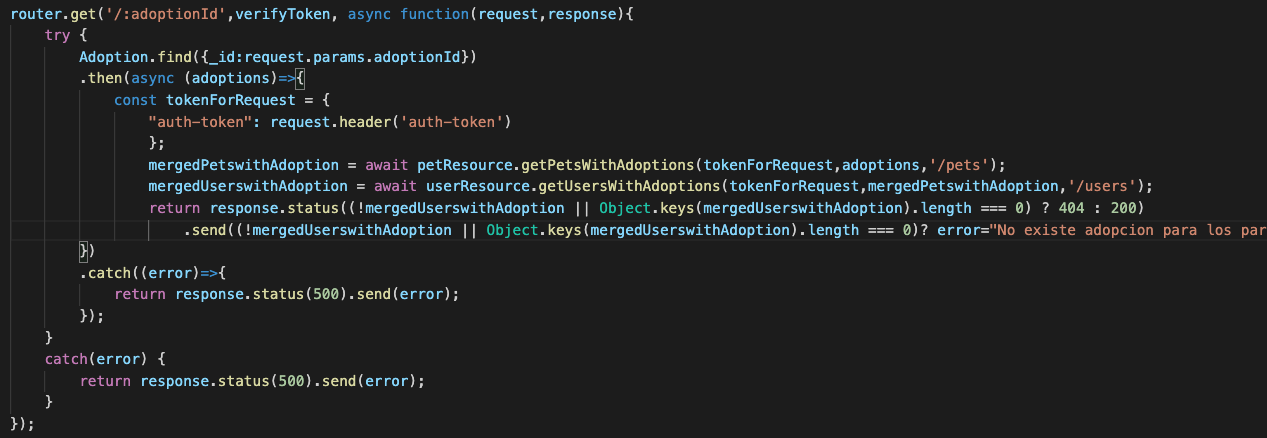


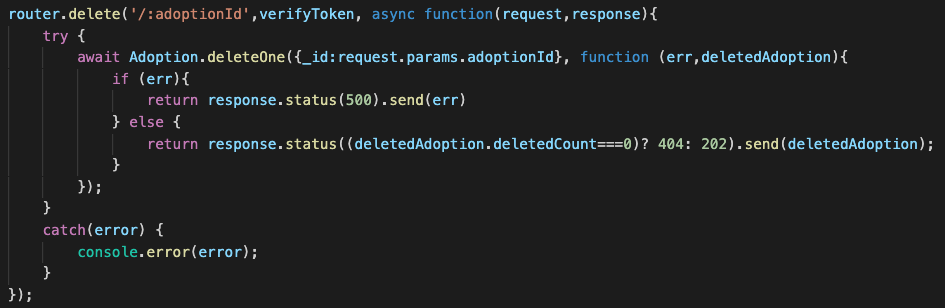
1. **JUSTIFICACIÓN DE CÓMO HAN IDO CONSIGUIENDO CADA UNO DE LOS REQUISITOS DEL MICROSERVICIO**

En este apartado se explicará cómo hemos conseguido cada requisito. En cada explicación se insertarán algunas imágenes de ejemplo donde se visualizará parte del código implementado.

* **El backend debe ser una API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado.**

La implementación de la API REST se ha llevado a cabo siguiendo los vídeos de la asignatura, usando los frameworks NodeJS y expressJS. Se han implementado los métodos CRUD básicos y, adicionalmente, un método GET filtrando parámetros mediante query y otro método GET que recibe un ID por parámetros y devuelve una adopción. En las imágenes se muestran algunos métodos.

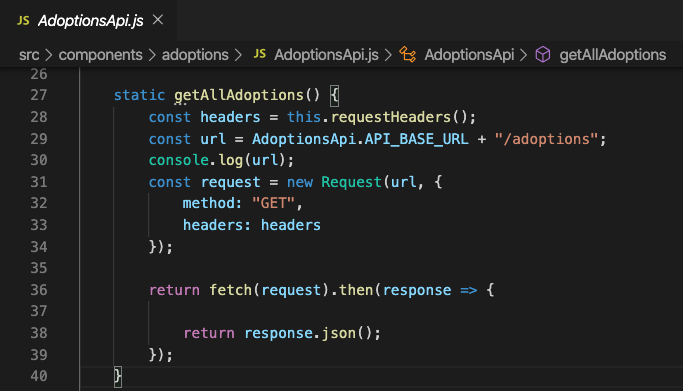


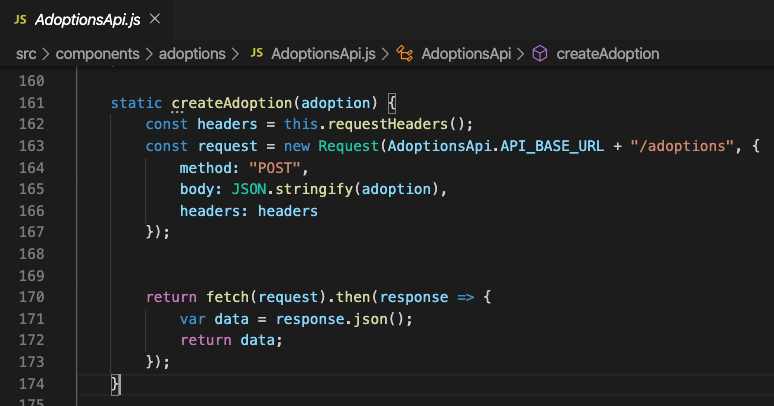


* **Debe tener un frontend que permita hacer todas las operaciones de la API (este frontend puede ser individual o estar integrado con el resto de frontends).**

En este caso se ha optado por la opción de implementar un frontend común que se integre con todas las APIs. Dicho frontend ha sido desarrollado con el framework React usando los vídeos de la asignatura como base.

Para la comunicación con el backend de nuestro microservicio se ha creado un fichero “AdoptionsApi.js” dedicado exclusivamente a esta tarea.



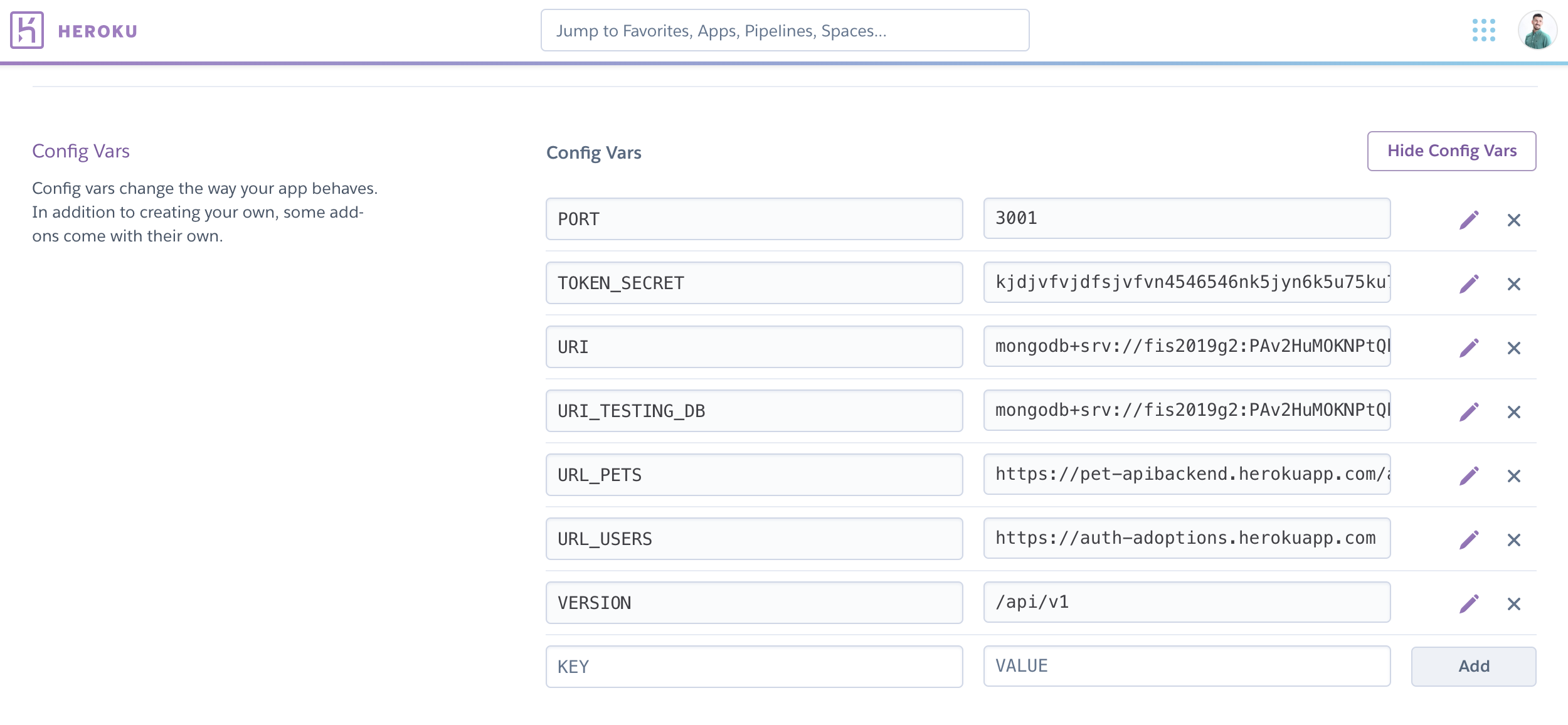


* **Debe estar desplegado en la nube y ser accesible en una url.**

Para que nuestro microservicio esté desplegado en la nube y sea accesible desde una URL, se ha usado Heroku, siguiendo los vídeos de la asignatura. En concreto, la URL donde se encuentra accesible nuestro microservicio es:

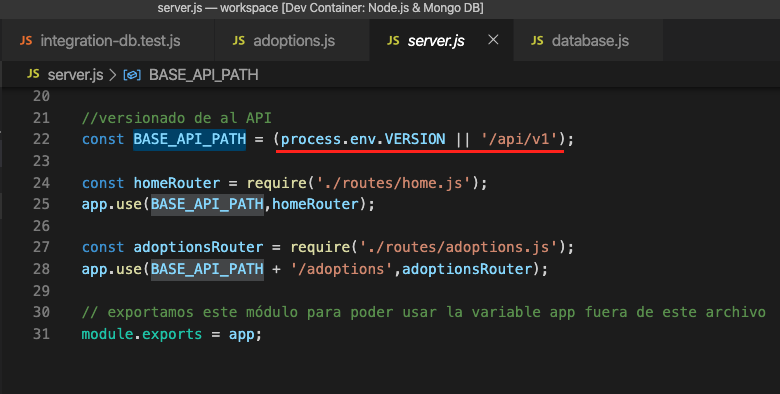
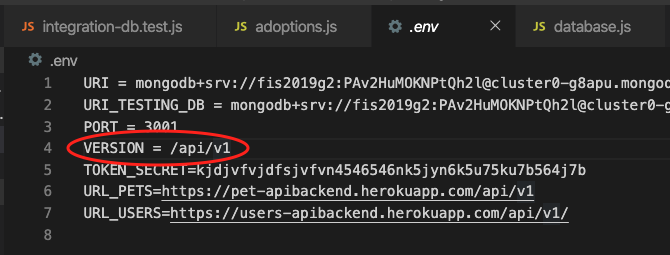
<https://adoptions-apibackend.herokuapp.com/>

Un paso importante a tener en cuenta para realizar el despliegue en Heroku es configurar las variables de entorno necesarias para que el microservicio funcione correctamente.



* **La API que gestione el recurso también debe ser accesible en una dirección bien versionada.**

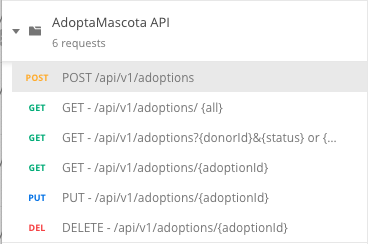
Este requisito se ha realizado, siguiendo las buenas prácticas, insertando en la URL la versión de la misma mediante una variable de entorno.

****

* **Se debe tener un conjunto de ejemplos de uso del API en Postman de todas las operaciones de la API.**

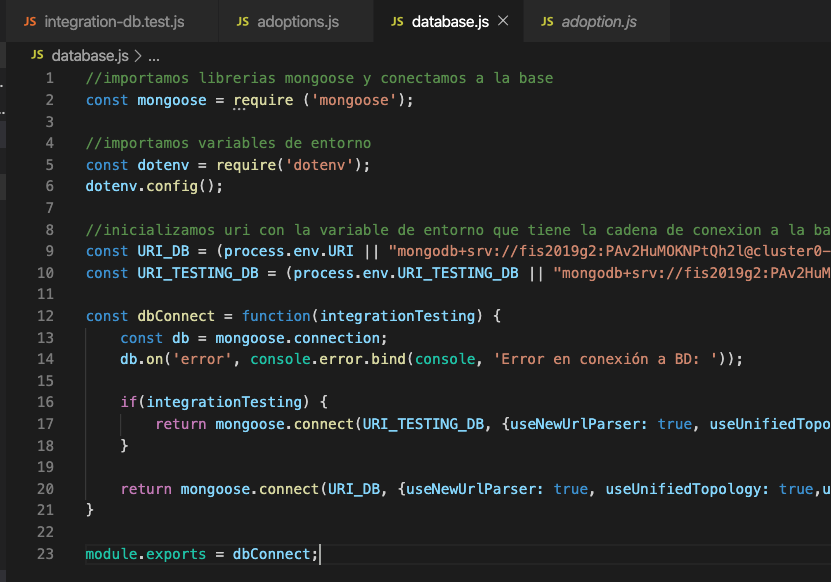
Se adjunta la colección de ejemplos de Postman en el siguiente enlace:

<https://www.getpostman.com/collections/e5fda85846ad6b85e473>

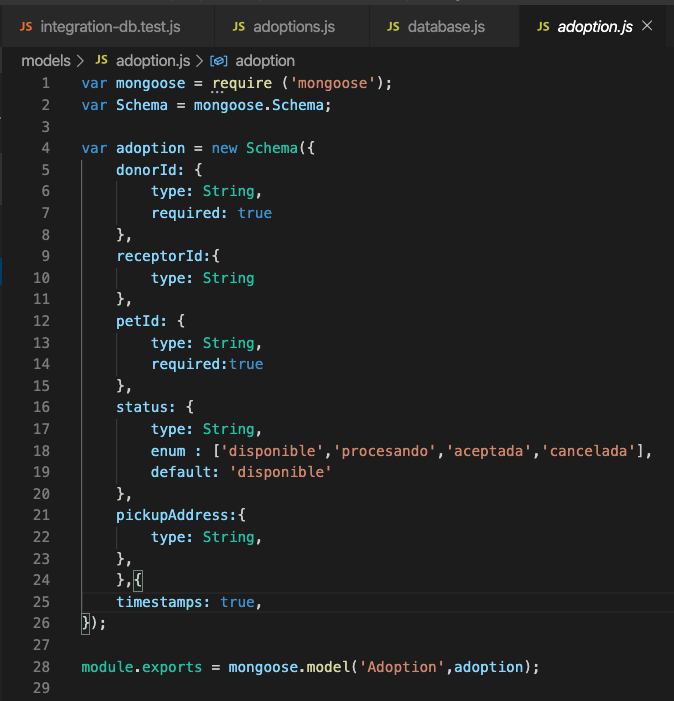


* **Debe tener persistencia utilizando MongoDB**

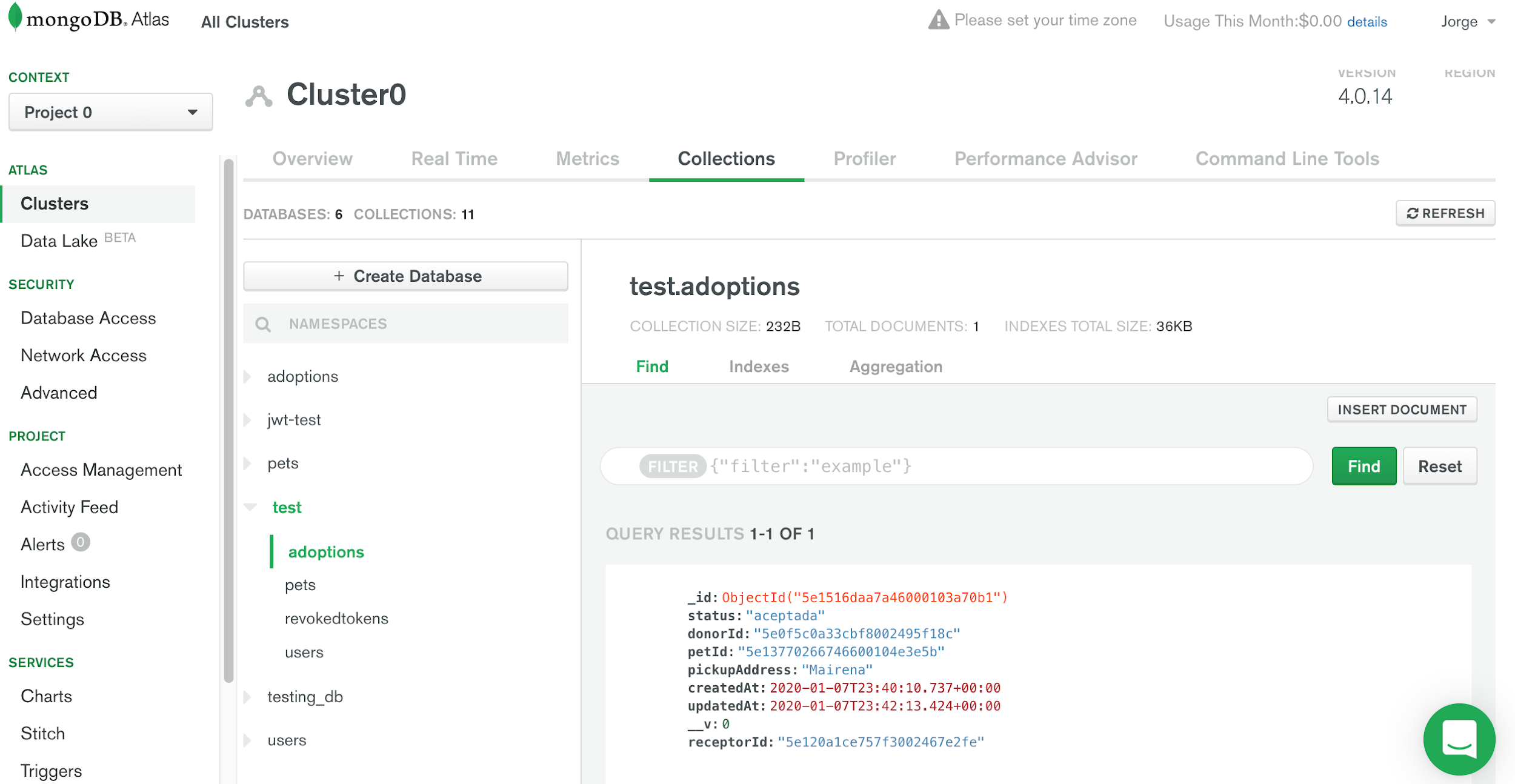
Para integrar MongoDB en el ecosistema de nuestro backend se ha usado la librería “mongoose”. Para gestionar la conexión a la base de datos, se ha definido un fichero específico (database.js) en el cual se indican las rutas de las bases de datos (producción y testing) mediante variables de entorno.



Una vez instalada, esta permite crear modelos (“Schemas”) que definirán los campos que persistirán en la base de datos en forma de documentos:

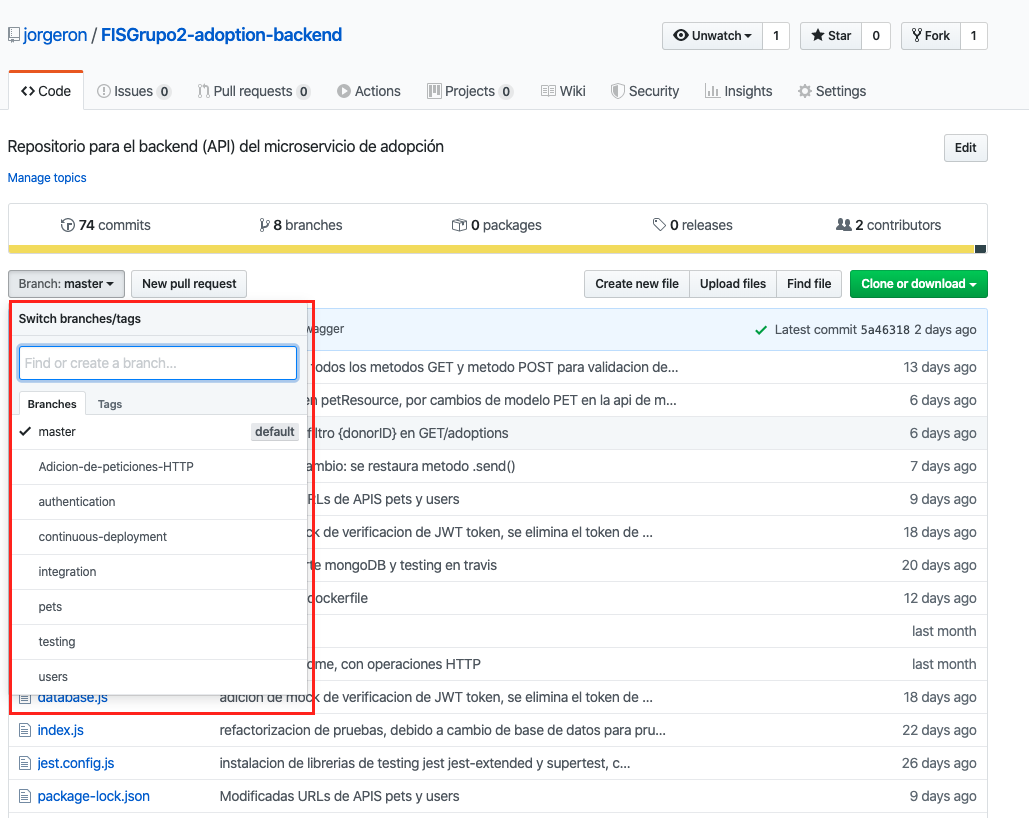


Un aspecto a tener en cuenta en este apartado es la utilización de **mongoDB ATLAS**, el servicio cloud que ofrece mongoDB solucionando así el problema de la localización de la base de datos.

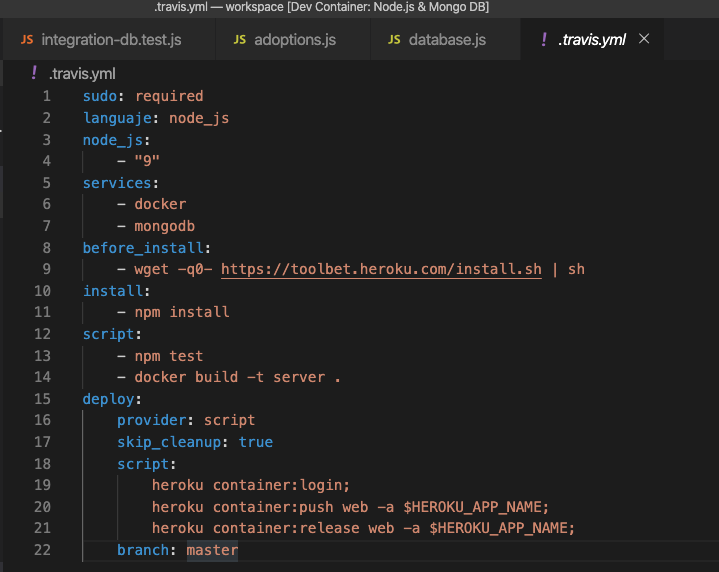


* **Utilizar gestión del código fuente y mecanismos de integración continua:**

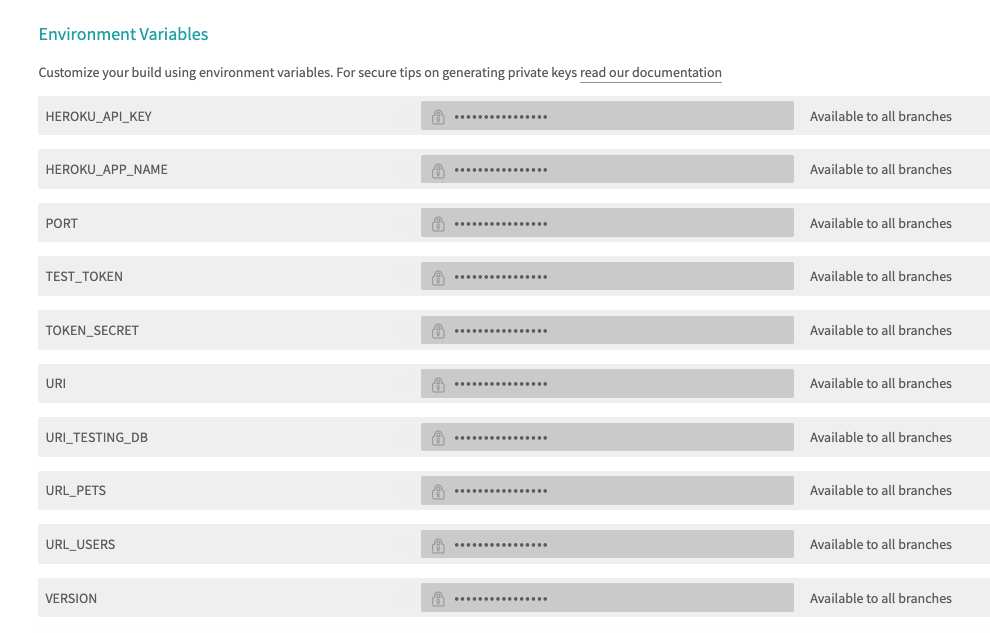
La gestión del código fuente se ha llevado a cabo mediante un repositorio de git siguiendo la metodología propuesta por GitHubFlow. Así, se ha creado una rama por cada funcionalidad que se ha ido añadiendo al proyecto, finalizando estas mismas con un pull-request, el cual se ha aprobado tras la revisión de ambos miembros del equipo.

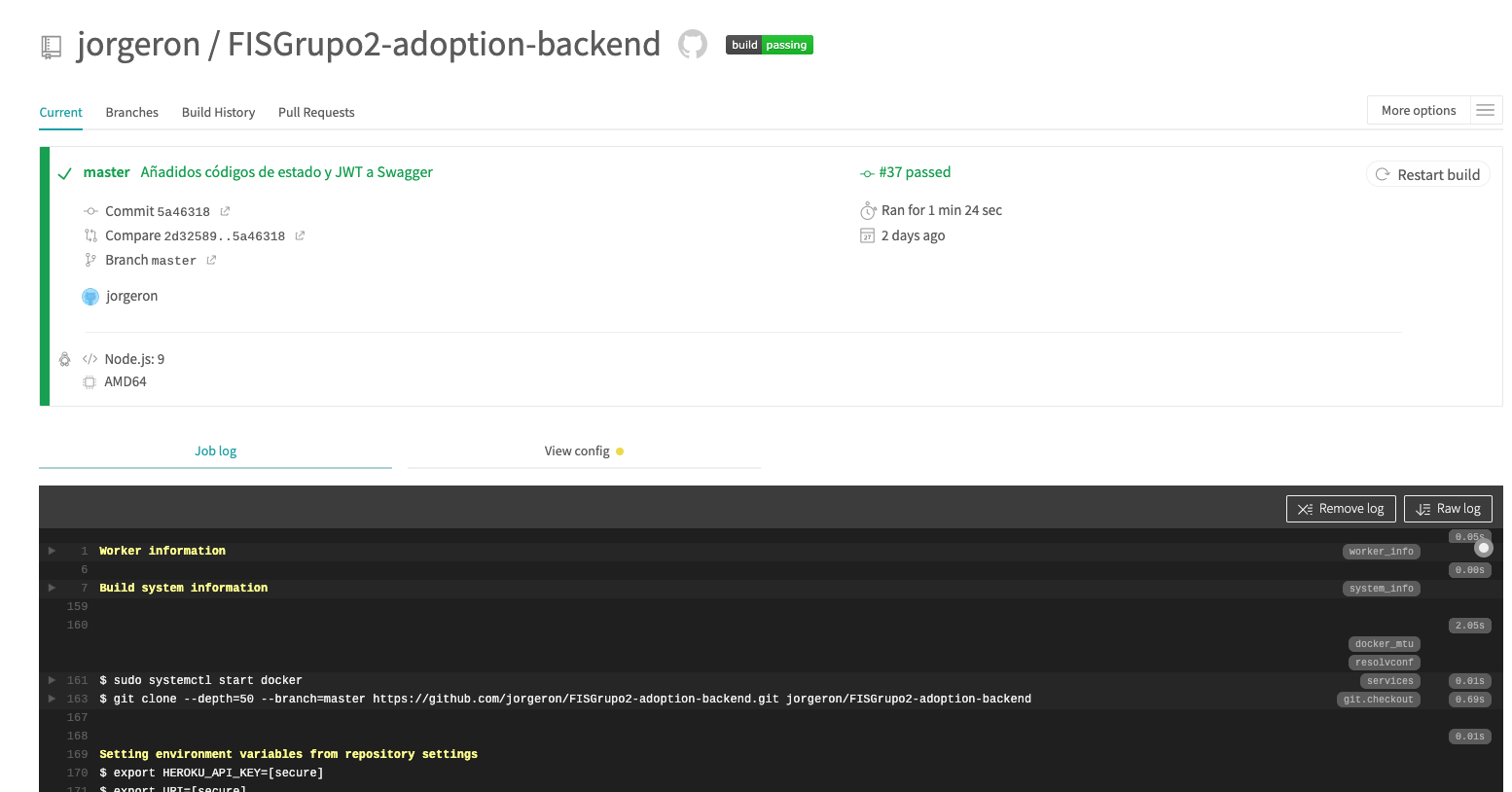


La integración continua, por su parte, se ha realizado con Travis. Mediante la configuración del fichero travis.yml, donde se indican los pasos a realizar cada vez que se ejecuta, además de las instrucciones de despliegue en Heroku. Así, cada vez que se realiza un push a la rama master del repositorio, travis se encarga de ejecutar los test, construir el contenedor Docker y, si todo ha ido bien, desplegar la última versión en Heroku.



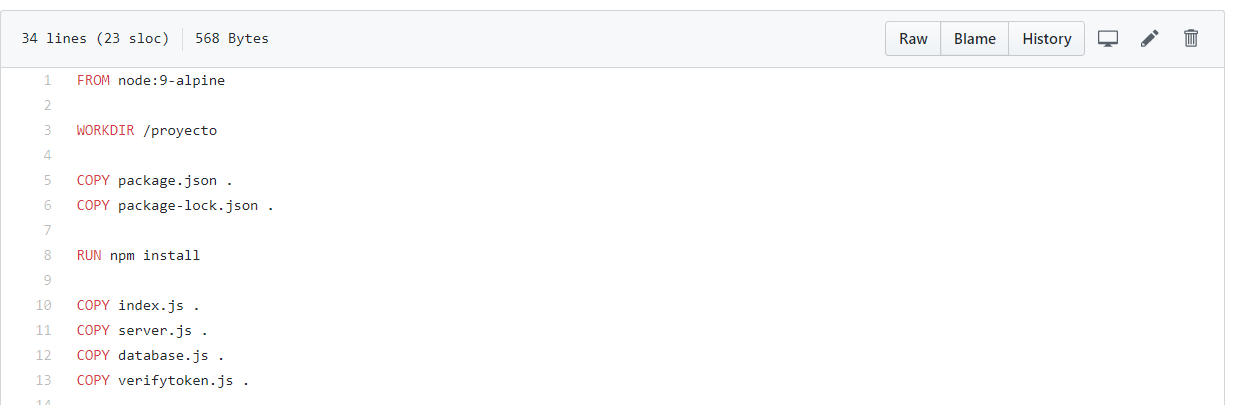
Para que todo funcione correctamente, también se tendrán que configurar todas las variables de entorno necesarias, además de dos variables especiales: $HEROKU\_APP\_NAME y $HEROKU\_API\_KEY, mediante las cuales se indican los datos de la aplicación para su despliegue en Heroku.





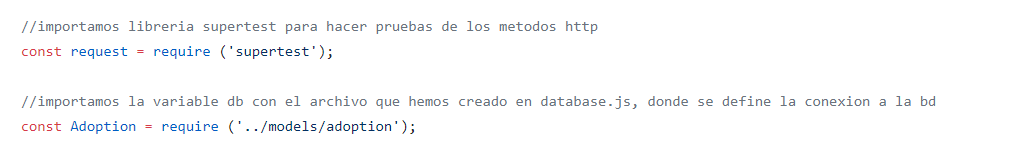
* **Debe haber definida una imagen Docker del proyecto**

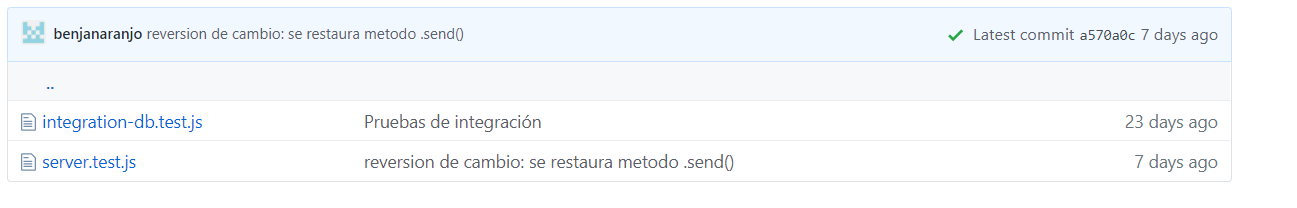
En la raíz del repositorio de GitHub (<https://github.com/jorgeron/FISGrupo2-adoption-backend>), se puede encontrar el Dockerfile completo, con el que se construyó la imagen desplegada en Heroku.



* **Debe haber pruebas unitarias implementadas en javascript para el código del backend utilizando Mocha y Chai o similar**

Las herramientas y librerías utilizadas para las pruebas unitarias han sido JEST y SUPERTEST, los archivos de pruebas pueden encontrarse en el repositorio de GITHUB, en la carpeta /TESTS/, y hay uno correspondiente a los métodos del backend, siguiendo el esquema ARRANGE, ACT, ASSERT



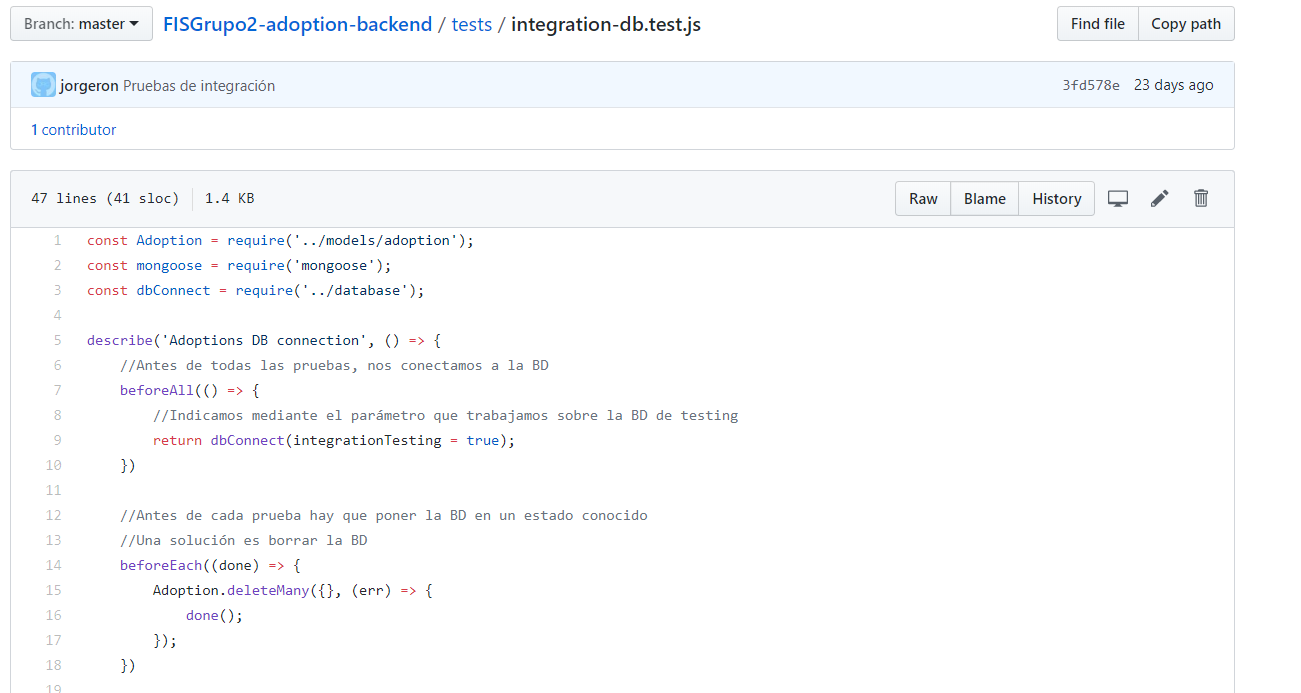


Se muestra un ejemplo de una prueba unitaria tipo mock:



* **Debe haber pruebas de integración con la base de datos.**

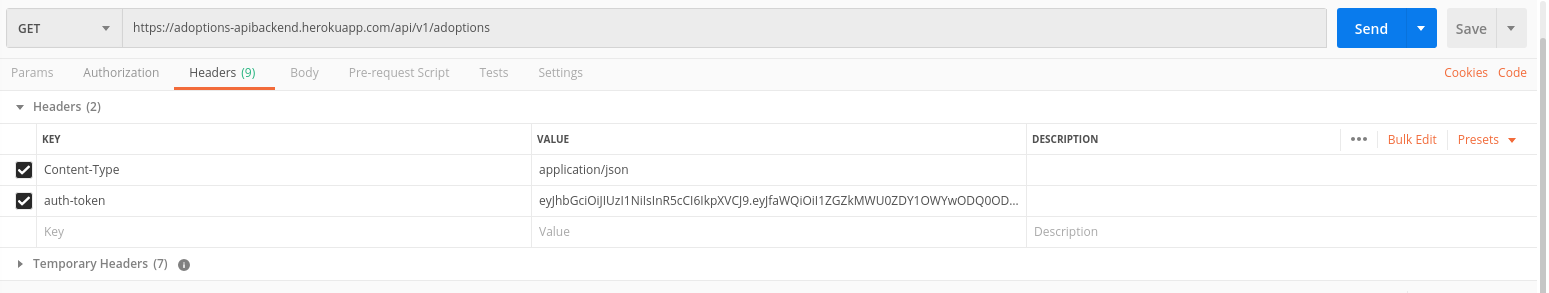
En la ruta /TESTS/ del repositorio de GitHub, se pueden encontrar las pruebas de integración con la base de datos, siguiendo el esquema ARRANGE, ACT, ASSERT

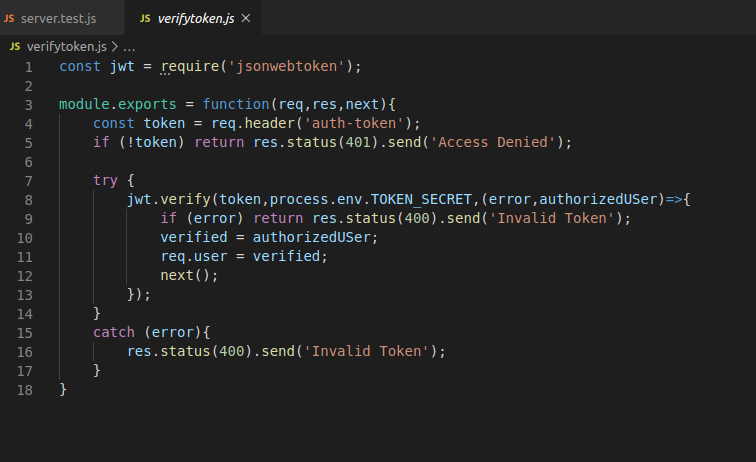


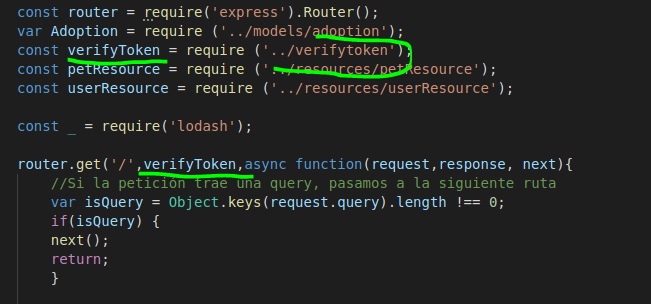
* **Debe tener un mecanismo de autenticación en la API**

Este punto ya fue mencionado anteriormente, el método de autenticación escogido fue de JSON Web Token.

Para implementarlo, se creó una función anónima, que es cargada en el componente de backend a través de una variable llamada VERIFY, que se adiciona como validación a las peticiones de cada método HTTP.







1. **NIVEL DE ESFUERZO**

Para llevar un control de tiempo de desarrollo del proyecto, se ha utilizado una de las herramientas de productividad mencionadas en la materia FUNDAMENTOS DE GESTIÓN DE TI, la cual RESCUE TIME, en su versión gratuita.

Según los reportes obtenidos por la herramienta, el nivel de esfuerzo ha sido calculado, en base al consumo de tiempo de las aplicaciones VMWARE PLAYER y VIRTUALBOX, las cuales contienen las máquinas virtuales sobre las cuales se ha desarrollado el proyecto.

Así mismo se considera el tiempo de la aplicación VIDEO.UI y VLC PLAYER, para la reproducción de los vídeos de la clase.

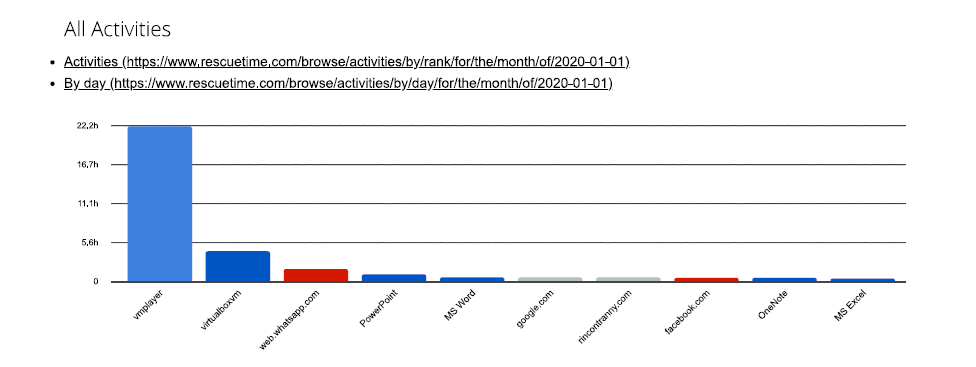
**Tiempo Total**: 120 horas

Benjamín Naranjo: 74 horas aproximadamente

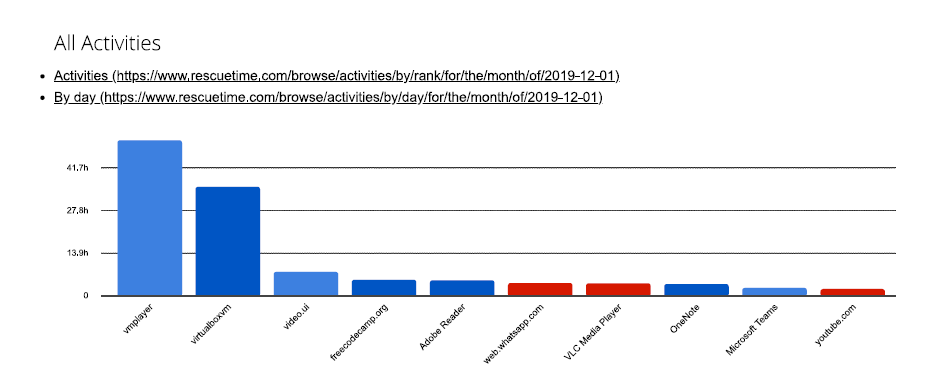
Jorge Roncel: 46 horas aproximadamente

**Esfuerzo en horas (Benjamín Naranjo):**

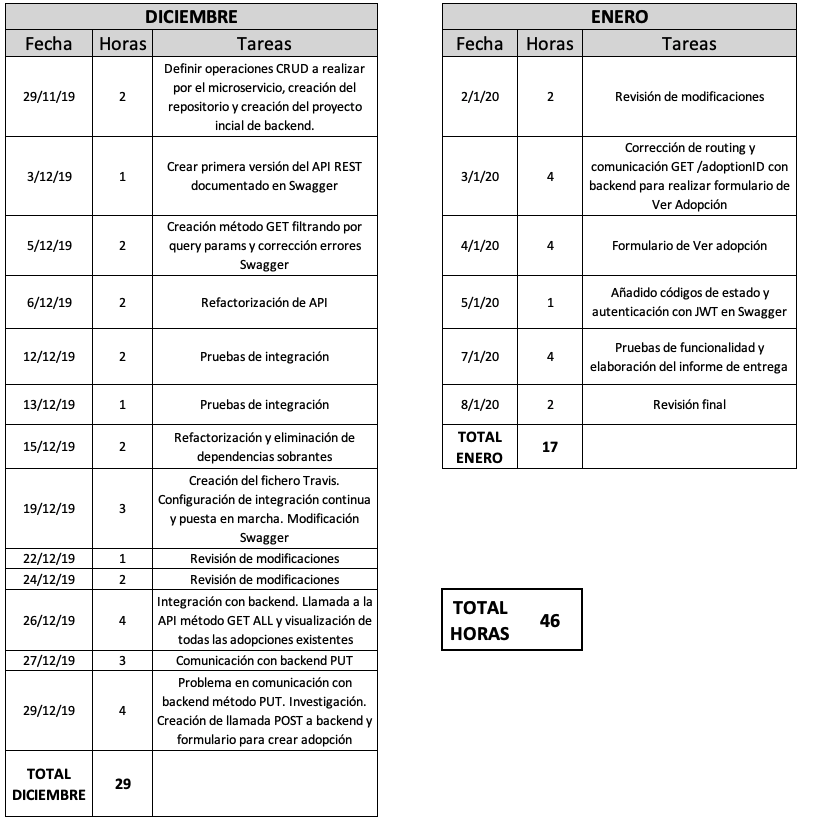
**Enero :** 25 horas aproximadamente

****

**Diciembre**: 50 horas aproximadamente



**Esfuerzo en horas (Jorge Roncel):**

****