|  |  |
| --- | --- |
| **SD** | **Sistemas Distribuidos** |
| **20/21** | Práctica |
|  | Memoria de la práctica |

# Introducción

Esta práctica ha sido realizada utilizando la pila MEAN para su desarrollo. Esto es, se ha utilizado MongoDB como base de datos, Express como servidor y Node.js para la ejecución de código JavaScript. Quedaría el uso de Angular para la parte del front-end, pero no ha sido realizado en esta práctica.

 Para poner en marcha esta práctica, primero se ha procedido con la instalación de los paquetes necesarios. Esta se desarrolló en **Ubuntu 20.10**, utilizando el gestor de paquetería de este SO, apt-get, por lo que los comandos a utilizar en la instalación fueron los siguientes:

Una vez tenemos los paquetes necesarios instalados, procedemos a la instalación del gestor de paquetes para node:



A partir de este momento ya podemos trabajar sobre código que queramos ejecutar, como por ejemplo realizar una pequeña versión de un servidor.

## Express

El siguiente paso es instalar Express mediante npm. Para ello, realizamos el siguiente comando:



En el comando anterior, el parámetro *i* sirve para indicar que se va a proceder a la instalación de un paquete, y el parámetro *-S* sirve para indicar a npm que guarde Express en los archivos de configuración, de forma que si en algún momento, desde otro dispositivo, realizamos *npm init*, se instale express de forma automática.



## Nodemon

El siguiente paquete de npm que vamos a instalar es Nodemon. Este se va a encargar de reiniciar nuestros endpoints cada vez que realicemos un cambio en ellos, agilizando el desarrollo de estos. Para instalaro, realizamos el mismo comando explicado anteriormente.



## Morgan

Ahora vamos a instalar el paquete que se va a encargar de proporcionarnos información, mediante logs, de todo lo que pase en nuestra aplicación, mediante el registro de peticiones y respuestas en nuestras aplicaciones Express. Para instalaro:





## MongoJS y MongoDB

Como hemos explicado antes, vamos a utilizar como base de datos MongoDB. Este simple módulo nos permite conectarnos a la base de datos, realizar diferentes tipos de llamadas, como lecturas, escrituras y modificaciones. Para su instalación realizamos los siguientes comandos:





## Node-Fetch

Como se explicará más adelante, nuestro sistema dispone de un GateWay que se encarga de conectarse a los diferentes endpoints y recibir datos de estos. Para ello requerimos de un módulo que nos permita conectarnos a estos, y hacerles las diferentes llamadas get, post, update y delete. Para instalar este paquete realizamos los mismos pasos que en los anteriores:





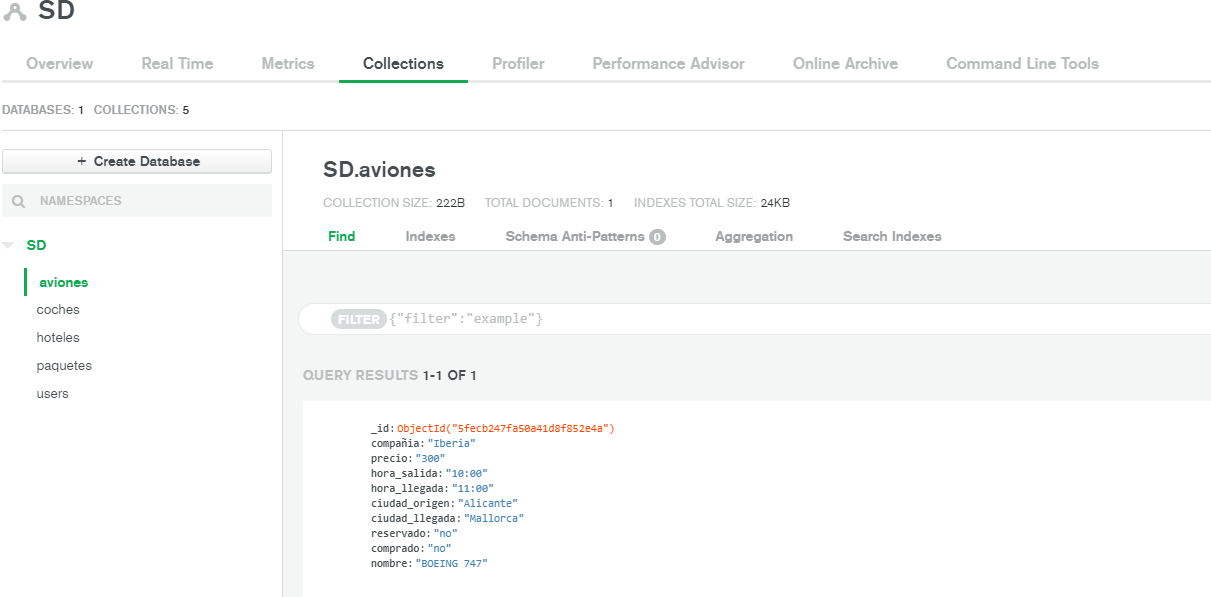
## Url

Si queremos leer los parámetros que nos pasen los usuarios por parámetro necesitamos este módulo, que nos permite extraer la información pasada. Para instalarlo, ejecutamos el siguiente comando y ya lo tenemos:





## Base de Datos

 En los laboratorios de la EPS no viene instalado por defecto MongoDB como base de datos, por lo que, para poder usar una base de datos se ha accedido a [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) y se ha creado un cluster llamado SD, con una base de datos llamada SD, con 5 colecciones: aviones, coches, hoteles, paquetes y users:

# Servicios/End-Points

Para la implementación de la práctica se han implementado varios End-Points que definen la arquitectura de la práctica. De esta forma, se ha implementado un End-Point para los aviones, para los hoteles, para los coches y para la reserva de paquetes.

Los 3 End-Points principales, Coches, Aviones y Hoteles, disponen de una colección en la base de datos en la nube, por lo que cada uno de ellos se conectará a una de estas colecciones para consultar/modificar datos. Además, el End-Point que se encarga de crear, consultar y reservar paquetes, a su vez se conecta a estas bases de datos, con el objetivo de crear estos. La definición gráfica de estos se muestra en la arquitectura conceptual.

Por último, se ha creado un GateWay encargado de conectarse a los anteriores End-Points usando la herramienta fetch.

La siguiente tabla muestra los diferentes servicios que se han creado, con un breve resumen de cada uno de ellos:

| **Servicio** | **Tipo** | **Breve descripción** |
| --- | --- | --- |
| **Reserva de Coches** | WS tipo REST | Permite gestionar las reservas de coches. |
| **Reserva de Hoteles** | WS tipo REST | Permite gestionar las reservas de los hoteles |
| **Reserva de Aviones** | WS tipo REST | Permite gestionar las reservas de aviones |
| **Reserva de Paquetes** | WS tipo REST | Permite gestionar los paquetes |
| **GateWay** | WS tipo REST | Conecta con el resto de servicios. |

A continuación, se pasan a detallar los métodos http de cada servicio:

## API-REST-GW

La siguiente tabla resume los diferentes tipos de llamadas que se pueden realizar al GateWay:

| **Verbo HTTP** | **Ruta** | **Descripción** |
| --- | --- | --- |
| **GET** | /api/paquetes | Obtiene todos los paquetes creados hasta la fecha. |
| **GET** | /api/{colecciones} | Obtenemos todos los elementos de la tabla {coleccion}. Estas colecciones pueden ser coches, aviones o hoteles. |
| **GET** | /api/{colecciones}/{id} | Obtenemos el elemento indicado en {id} de la tabla {coleccion}. Para cada colección, el {id} ha de ser el nombre del objeto a buscar. |
| **GET** | /api/{colecciones}/id/{id} | Obtiene el elemento de la colección indicada, pero a partir de su ID. |
| **POST** | /api/paquetes | Crea un paquete a través de los nombres de las ciudades (de coche, avión y hotel) pasados en la URL de la llamada. Si ya existe un paquete con los objetos seleccionados, devuelve error. |
| **POST** | /api/paquetes/reservar/{id} | Se encarga de reservar un paquete a partir de su {id}. Si el paquete ya está reservado, devuelve error. |
| **POST** | /api/{colecciones} | Este método se encarga de llamar al método post de la {colecciones} pasada y crear nuevos objetos. Se pueden crear coches, aviones y hoteles. |
| **POST** | /login | Este método es el encargado de realizar el proceso de iniciar sesión a los usuarios. Si los datos introducidos son correctos, devuelve el token creado y lo guarda en la base de datos del usuario. |
| **POST** | /register | Este método es el encargado de crear un usuario en la base de datos. Se encarga de comprobar que el email introducido no exista, y guarda el usuario en la BD. |
| **PUT** | /api/{colecciones}/{id} | Se encarga de modificar la colección pasada en la url (coches, aviones, hoteles). El {id} ha de coincidir con el nombre del objeto a modificar. |
| **DELETE** | /api/{colecciones}/{id} | Elimina el elemento {id} de la colección pasada. A diferencia del resto de métodos, este método requiere del ID del producto a eliminar. |

## API-REST-Coches/API-REST-Aviones/API-REST-Hoteles

Para la explicación de las rutas de estos 3 servicios se va a realizar la tabla de uno de ellos, coches, ya que son todos exactamente iguales y no varían sus métodos. La tabla:

| **Verbo HTTP** | **Ruta** | **Descripción** |
| --- | --- | --- |
| **GET** | /api/{colecciones} | Obtiene todos los coches creados hasta la fecha. |
| **GET** | /api/{colecciones}/{nombre} | Obtenemos el coche cuyo nombre es {nombre}. |
| **GET** | /api/{colecciones}/id/{id} | Obtenemos el elemento indicado en {id} de la tabla coches. El {id} ha de ser el id del coche a buscar. |