Picap Training

Este repositorio contiene un proyecto base creado con Ruby on Rails que utiliza MongoDB como base de datos principal.

Rails es un framework que se basa en el patron de arquitectura MVC que permite desarrollar aplicaciones web.

Por otro lado MongoDB es una base de datos no relacional de tipo documental que permite almacenar datos cuyo esquema cambia en el tiempo y es almacenado en formato JSON. Esto ultimo permite que no sea necesario ejecutar migraciones en la base de datos como lo seria en el caso de bases de datos relaciones tal como PostgreSQL.

Tecnologías principales utilizadas en el proyecto

- Ruby versión 2.7.2
- Rails versión 5.2.8
- MongoDB versión 5.0.6

Para poder hacer el build del proyecto es necesario que se haga la instalación de Ruby y MongoDB.

> 💡 Antes de realizar las siguientes instalaciones es recomendable hacerlo en un sistema operativo basado en Unix como Mac OSX o Linux/Ubuntu ya que no todas las herramientas descritas como rvm o Homebrew tienen soporte para Windows.

Instalar Ruby

En el caso de Ruby es recomendable que sea instalado por medio de un manager de versiones tal como rvm o rbenv, siendo rvm el mas recomendable. Una vez instalado se puede instalar la version de ruby de la siguiente forma.

```
rvm install 2.7.2
```

Instalar MongoDB

Por otro lado para instalar Mongo es recomendable hacerlo por medio de HomeBrew

```
brew tap mongodb/brew
```

```
brew update
```

```
brew install mongodb-community@5.0
```

Build del proyecto

Una vez instalado Ruby en el ambiente local se debe ejecutar el siguiente comando en la raíz del proyecto con el propósito de instalar las librerías que se especifican en el archivo Gemfile.

```
bundle install
```

Que también puede ser ejecutado de la siguiente forma

```
bundle i
```

Este proceso debe finalizar sin ningún error.

Correr el proyecto

Dado que el proyecto utiliza como base de datos MongoDB y previamente se hizo su instalación se debe iniciar con el comando

```
brew services start mongodb-community
```

> 💡 Solo a caso de información cuando se vaya a parar la base de datos se debe usar

Ya teniendo la base de datos corriendo se puede ejecutar el servidor.

```
rails server
```

Que también puede ser ejecutado de la siguiente forma

```
rails s
```

Si todo funciono debería verse algo de este estilo indicando que el servidor esta corriendo en el puerto 3000 del computador

```
=> Booting Puma
=> Rails 5.2.8 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.12.6 (ruby 2.7.2-p137), codename: Llamas in Pajamas
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:3000
Use Ctrl-C to stop
```

Ingresando al URL http://localhost:3000/ debería poder visualizarse la pagina de bienvenida de rails.

Crear servicios de prueba

A continuación se presentan instrucciones para crear un servicio REST de creación y listar de prueba en el proyecto. Este servicio se va a encargar de crear un computador y de listarlos.

Paso 1: Crear el modelo

Se va a crear un modelo computer que va a representar un computador, este modelo debe tener los campos.

- model - String
- brand - String
- year - Integer

Para ello se debe crear un nuevo archivo ***computer.rb*** en la carpeta app/models con el siguiente contenido.

```
class Computer
  include Mongoid::Document
  include Mongoid::Timestamps

  field :model, type: String
  field :brand, type: String
  field :year, type: Integer
end
```

Paso 2: Crear controlador

Se va a crear un controlador que es donde se van a declarar los 2 servicios que necesitaremos. Crear un archivo ***app/controllers/computers_controller.rb*** con el siguiente contenido.

```
class ComputersController < ApplicationController
  def create
    computer = Computer.create!(
      model: params[:model],
      brand: params[:brand],
      year: params[:year]
    )
    render json: computer
  end

  def index
    render json: Computer.all
  end
end
```

Antes de probar los servicios se debe modificar el archivo ***routes.rb*** con el siguiente contenido.

```
Rails.application.routes.draw do
  resources :test do
```

```
    get :pokemon, on: :collection
  end

  resources :computers do
    get :index, on: :collection
    post :create, on: :collection
  end
end
```

Compare el contenido de los archivos que se han modificado hasta el momento con lo que se ha pegado para entender el funcionamiento de rails.

Note que se declaran los mismos servicios create y index escritos en el controlador y antes del nombre se usa las palabras ***get*** y ***post,*** esto significa los metodos HTTP que usaremos para acceder a cada servicio.

Paso 3: Probar servicio de crear

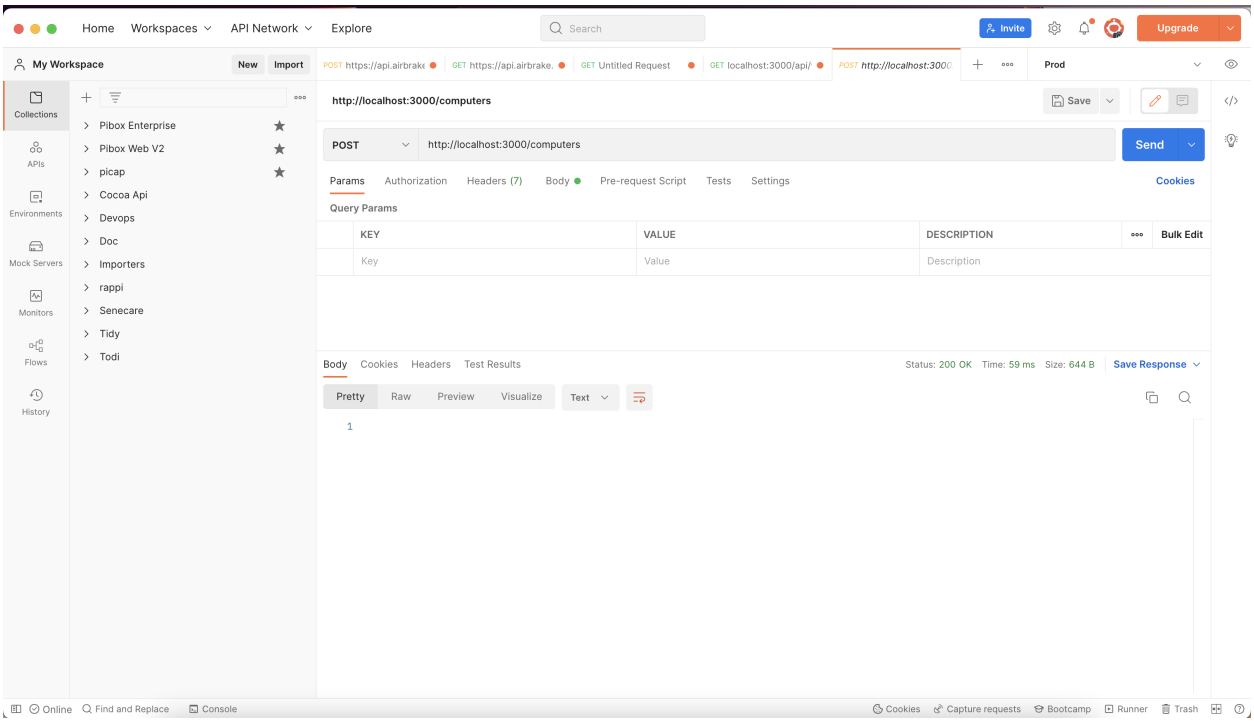
Para poder probar los servicios se debe tener un cliente HTTP con el cual se realizan los servicios para ello instale Postman con anterioridad.

Haga la siguiente petición para probar la creación:

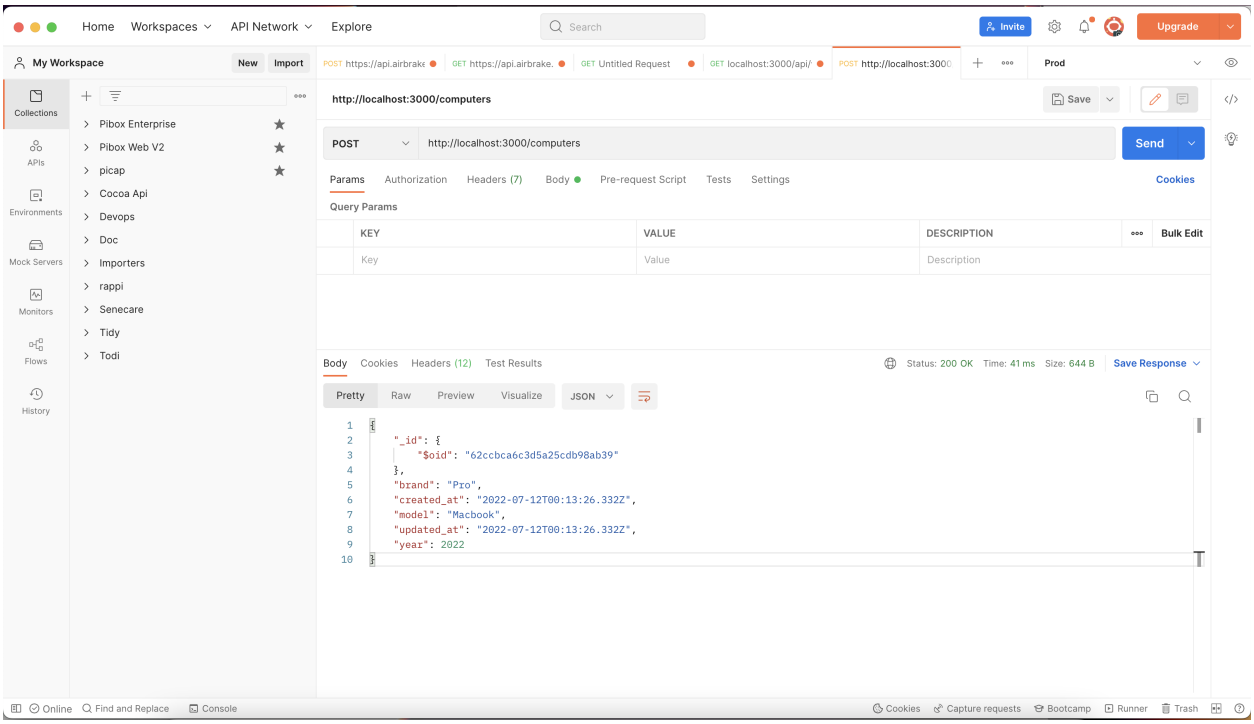
```
POST -> http://localhost:3000/computers
```

Con el siguiente body

```
{
    "model": "Macbook",
    "brand": "Pro",
    "year": 2022
}
```



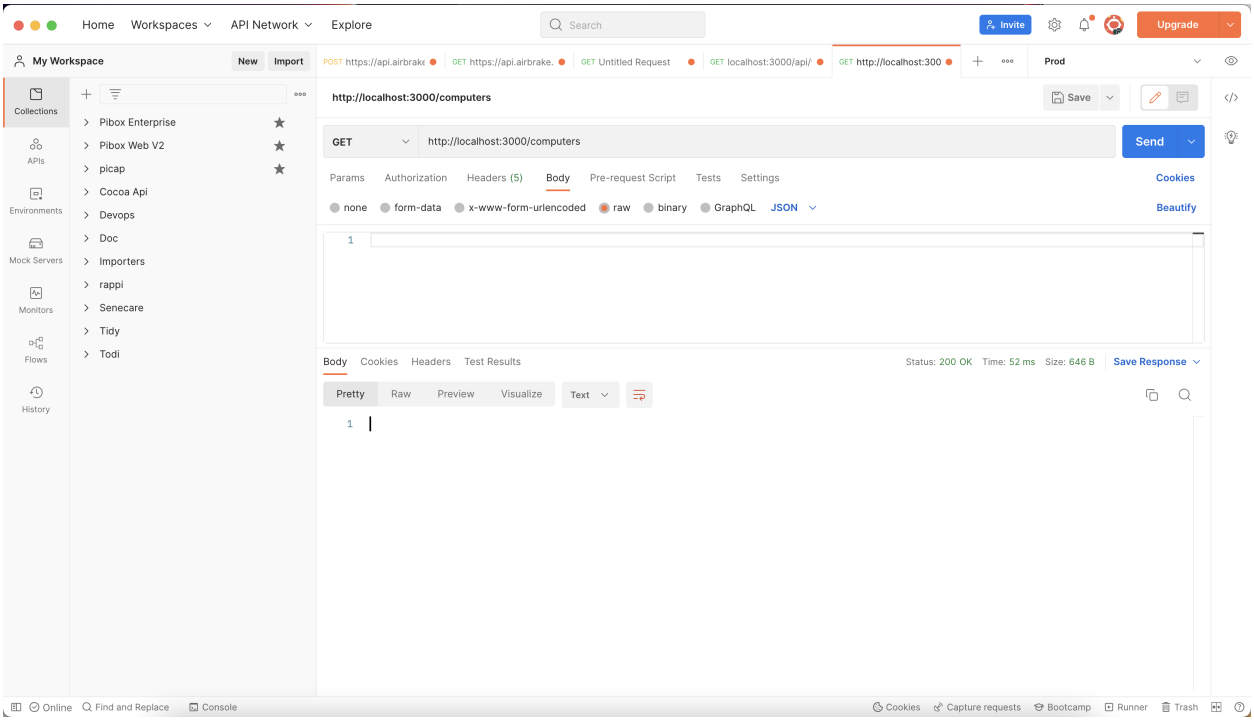
Si todo funciona correctamente debería ver el siguiente resultado después de enviar la petición



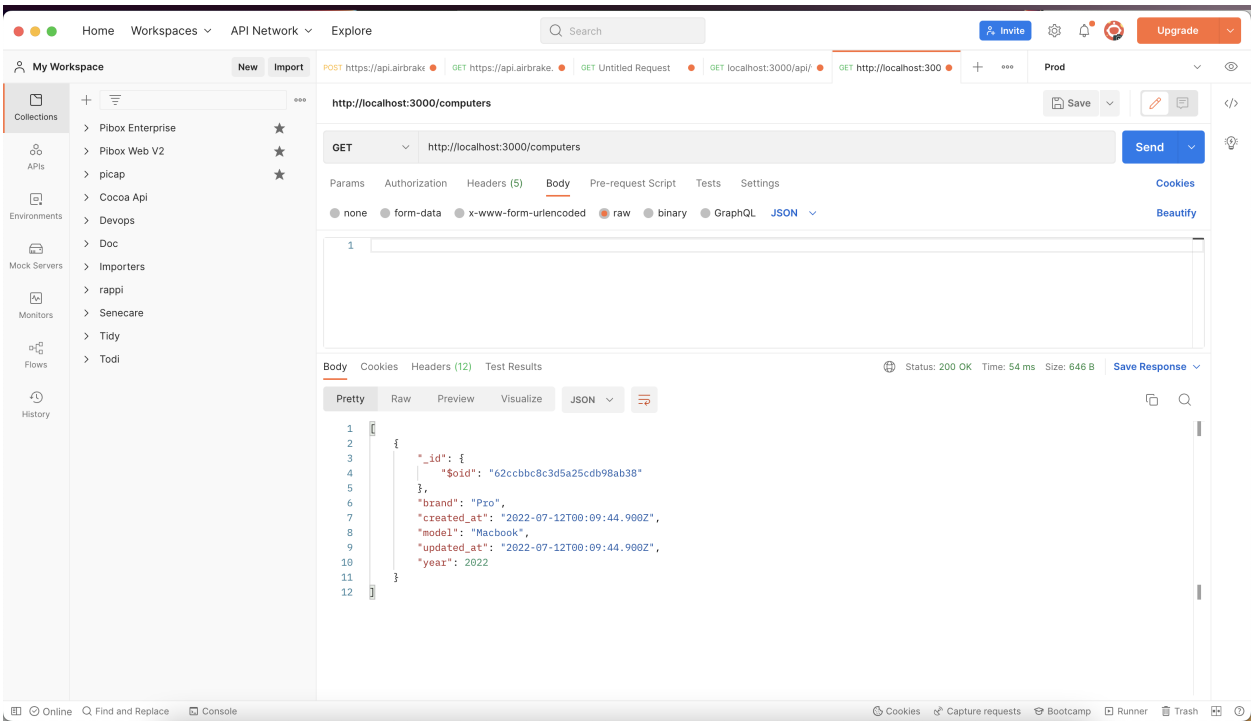
Paso 4: Probar servicio de listar

Haga la siguiente petición para probar la creación:

```
GET -> http://localhost:3000/computers
```



Si todo funciona correctamente debería ver el siguiente resultado después de enviar la petición



Paso 5: Utilizar la consola de rails para verificar creación de registros

Rails cuenta con una consola por la cual se puede ejecutar comandos directamente sobre la base de datos. En otro consola ejecute el siguiente comando.

```
rails console
```

Una vez abierta la consola puede ejecutar comandos tales como contar cuántos registros hay en la base de datos para el modelo computer.

```
2.7.2 :001 > Computer.count
```

Una vez ejecutado la anterior sentencia debería dar un valor mayor a cero.

Reto de entrenamiento

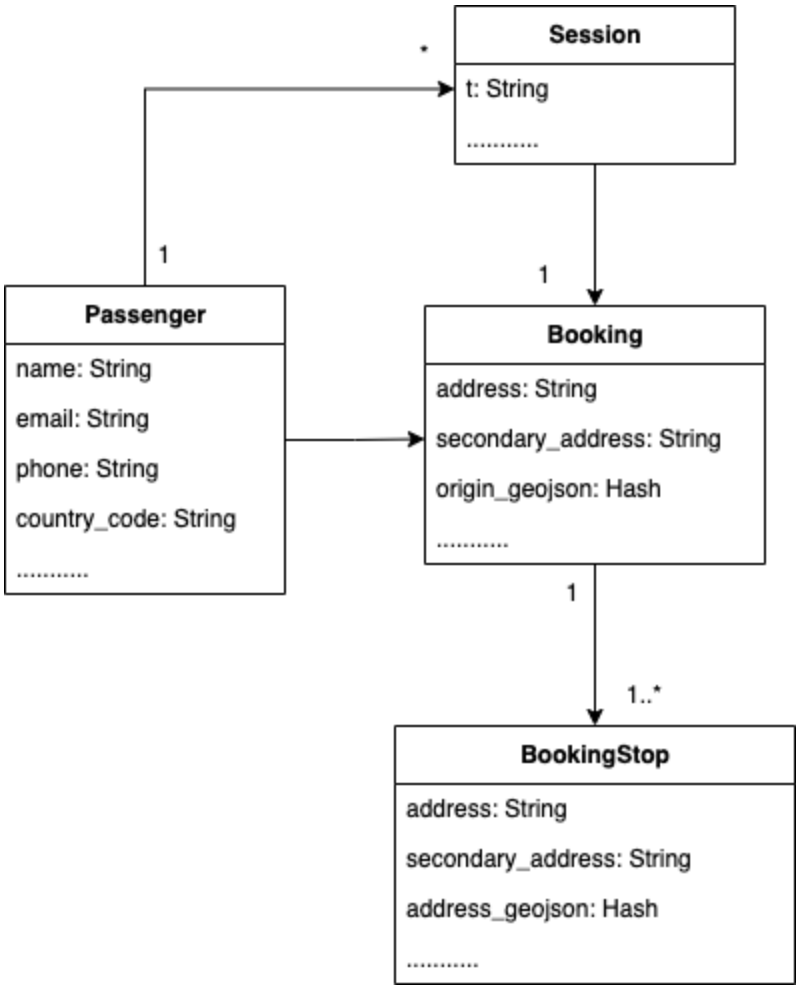
Historia de Pibox

Pibox es un servicio dentro de la compañía que permite a una persona natural o empresa enviar sus paquetes o productos desde un punto A a uno B por medio del grupo de conductores que hacen parte de la plataforma. Muchas de estas empresas tienen sus propios sistemas de software con los cuales reciben sus pedidos y luego se integran con Pibox para realizar las entregas, para lograr esto el equipo desarrollo un API que expone diferentes servicios REST como:

- Crear un Booking
- Listar Bookings solicitados
- Obtener el detalle de un Booking
- Cancelar un Booking

> 💡 Cuando se usa el termino Booking se hace referencia a un pedido o viaje, esto puede ser tanto para recoger un paquete o una persona y que luego sea llevado de un punto A a uno B por parte de un conductor.

Diagrama UML



Para conocer un poco más del sistema se presenta un diagrama UML presentando las clases principales y relaciones entre ellas.

- **Booking:** Representa un viaje o pedido solicitado por un usuario, contiene su dirección de origen donde el conductor debe llegar a recoger.
- **BookingStop:** Representa una parada que tiene que hacer un conductor durante el viaje a entregar.
- **Passenger:** Representa un usuario que puede ser tanto conductor como pasajero.
- **Session:** Representa la vez que un usuario a iniciado sesión, contiene el campo ***t*** que es el token de autenticación generado para la sesión y que debe ser enviado en cada request.

Objetivos del reto

El objetivo del ejercicio de introducción inicial es usar la documentación API donde se especifica los servicios ofrecidos por Pibox y se documenta el significado de cada campo para crear los siguientes servicios. **Ademas de realizar los servicios se debe crear un nuevo modelo llamado Log el cual debe almacenar la respuesta de cada uno de los llamados al API de Pibox que se realice.** Los servicios a realizar son los siguientes.

1. Realizar un servicio con la ruta POST → https://localhost:3000/bookings que debería conectarse con el API de Pibox y crear un nuevo servicio.
2. Realizar un servicio con la ruta GET → https://localhost:3000/bookings que debe conectarse con el API de Pibox y obtener todos los Bookings creados.
3. Realizar un servicio con la ruta GET https://localhost:3000/bookings/{ID_BOOKING} que debe conectarse con el API de Pibox y obtener el detalle de un pedido.

- Para poder acceder a los servicios de Pibox necesita enviar por parámetro un query parameter ***t*** lo cual es un token de autenticación, para ello utilice el siguiente valor RFSIniUVwueoZiBwuy0H13GNUdRWF7vjJThkZtnekHlSrcGicTxHtQ
- Las credenciales para acceder a la documentación son los siguientes.
  - **URL:** https://docs.picap.io
  - **Usuario:** pibox_inicial
  - **Contraseña:** a263e89fa206

Para ver un ejemplo de como se hace un servicio que realiza un HTTP request revise el metodo ***pokemon*** dentro del archivo ***app/controllers/test_controller.rb***