Proyecto Final Backend JS

API para comprar automóviles seminuevos.

Integrantes:

- Asael Hernández (asael.dev@gmail.com)
- Diego Mendoza (diegomer4@gmail.com)
- Ricardo Hernández (ricardo.hernandez.martinez96@gmail.com)
- Salvador Nieves (ing.salvador@hotmail.com)

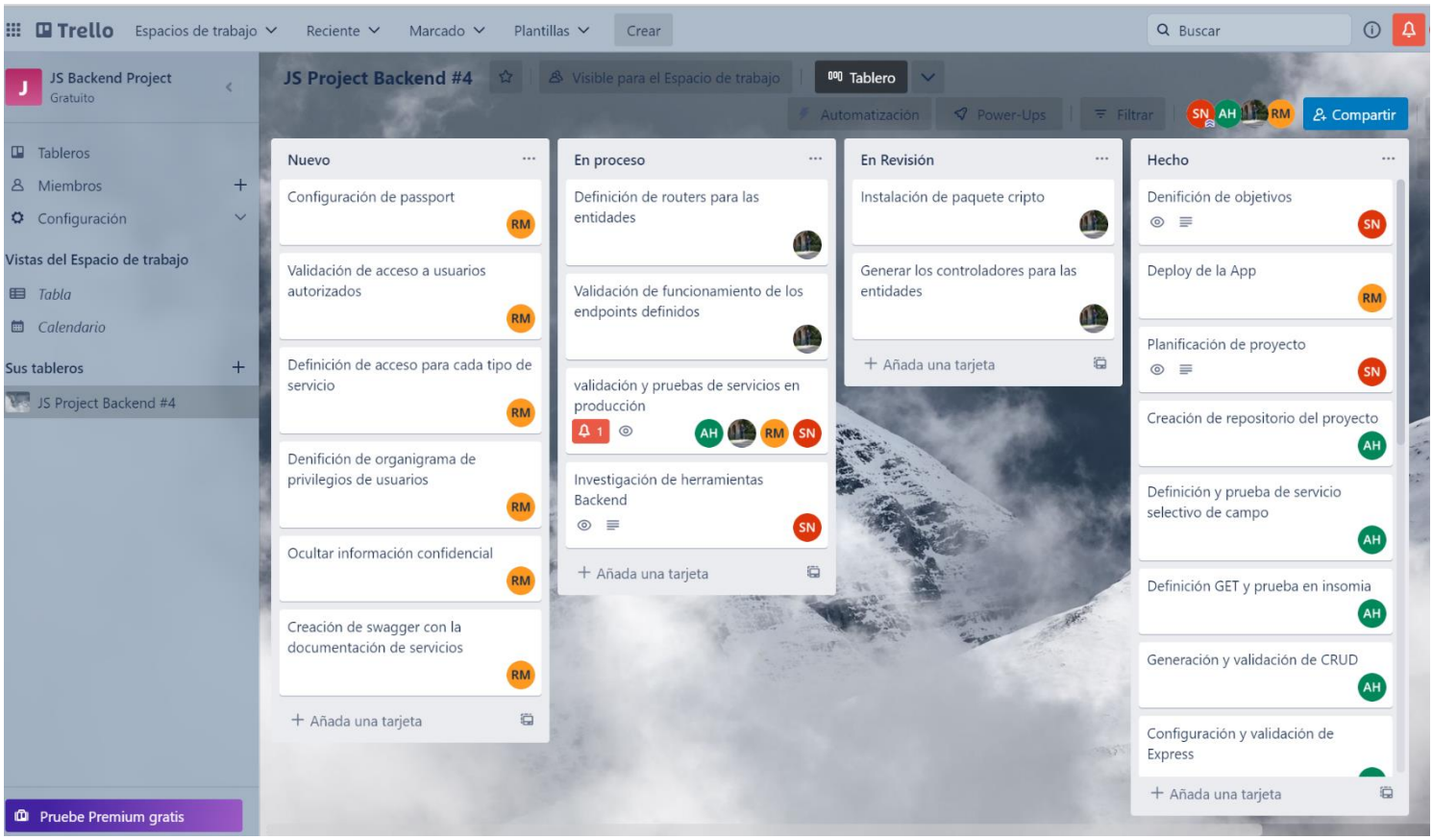
Objetivos del proyecto:

La aplicación a desarrollar tiene como objetivo el poder vender automóviles usados a nuestros distintos clientes, llevar a cabo la venta, gestión de las compras, administración de vendedores y sus ventas para potencializar el negocio y su escalabilidad.

Funcionalidades

- Login de usuarios
- Visualización de autos a elegir
- Compra de autos
- Registro de ventas
- Administración de vendedores y ventas
- Funcionalidades CRUD sobre las tablas establecidas

PLANEACIÓN: Tablero TRELLO, definición y asignación de actividades



Definición de plataformas o herramientas a utilizar

Se anexa documento Excel en repositorio para mejor visualización

Plataformas de desarrollo BackEnd		
Plataforma - Herramienta	Definición	Objetivo
PostgreSQL	PostgreSQL es un sistema de gestión de base de datos relacional open source,que hace énfasis en la extensibilidad y el cumplimiento de SQL.	Gestionar bases de datos SQL de alta concurrencia, cuenta funciones, seguridad integrada, integridad referencial, triggers, Autorizaciones, transacciones y respaldos
JavaScript	Es un lenguaje de alto nivel, dinámico e interpretado. Se define como orientado a objetos, basado en prototipos, imperativo y débilmente tipado.	Plataforma para el desarrollo web dinámico y funcional, compatible con los navegadores modernos, tambien se utiliza en el desarrollo de videojuegos, en la creación de aplicaciones de escritorio y móviles y en la programación de servidores con entornos de ejecución como Node.js. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).
Node.js	Es un entorno de tiempo de ejecución de JavaScript, incluye todo lo que se necesita para ejecutar un programa escrito en Javascript	Tiene como objetivo la creación de Aplicaciones de transmisión de datos (streaming), Aplicaciones intensivas de datos en tiempo real, Aplicaciones vinculadas a E/S, Aplicaciones basadas en JSON:API y en general entonos backend
npm	Node Package Manager o manejador de paquetes de node, es la herramienta por defecto de JavaScript para la tarea de compartir e instalar paquetes.	Maneja las dependencias para una aplicación y permite a los usuarios instalar aplicaciones Node.js que se encuentran en el repositorio
Sequelize	Es un ORM de Node.js basado en promesas para Postgres, MySQL, MariaDB, SQLite y Microsoft SQL Server. Sus características son soporte de transacciones sólido, relaciones, carga ansiosa y perezosa, replicación de lectura.	Permite manipular varias bases de datos SQL de una manera sencilla, entre estas bases de datos podemos encontrar: mysql, sqlite, postgres, mssql.
Express.js	Es un framework de backend Node.js minimalista, rápido y sencillo, que proporciona características y herramientas robustas para desarrollar aplicaciones de backend escalables, el cual ofrece el sistema de enrutamiento.	Permite que se desarrollen funcionalidades tales como el enrutamiento, que se utiliza para almacenar información sobre redes que se encuentren conectadas y permiten que el tráfico de información se transmita de una forma sencilla.
Github	Es un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Éste permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso.	Permite la generación de repositorios de nube con una serie de funcionalidades para la correcta y ordenada colaboración de los proyectos de software y desarrollo.
Heroku	Es una plataforma de servicios en la nube (conocidos como PaaS o Platform as a Service) que permite manejar los servidores y sus configuraciones, escalamiento y la administración.	Ejecución de aplicaciones a través de sus contenedores, también conocidos como Dynos

Épica e Historias de usuario

Diseño Proyecto API para venta de automóviles seminuevos		
Épica	Característica	Historia de usuario
API	Generación de información base de datos	1
	Implementación de base datos Postgres sobre Heroku	2
	Generación de modelo base del proyecto	3
	Generación de CRUD	4
	Definición de servicio de obtención de datos basados en selección	5
	Definición de routers	6
	Identificación y protección de información sensible	7
	Gestión de accesos para usuarios autorizados	8
	Deploy de proyecto	9
	Generación de documentación de servicios	10

Diseño Proyecto API para venta de automóviles seminuevos					
Clave	Historia	Descripción	Escenario	Criterios de aceptación	Reglas de negocio
1	Generación de información base de datos	<Quien> Team Dev <Quiero> Presentar la información de mis autos seminuevos disponibles, usuarios y ventas. <Para> Los usuarios puedan seleccionar y buscar automoviles, ingresar a la plataforma y realizar sus compras	<Dado que>Team Dev genere y cargue la información necesario para el desarrollo y funcionamiento de la plataforma <Cuando>A lo largo del desarrollo del proyecto, fase inicial <Entonces>Team Dev cuenta con información para pruebas y se avanza de forma agil	1.- Basadas en las tablas y relaciones se genera información para cada una de las tablas y campos 2.- La información es consistente y funcional 3.- Se cuenta con mas de 50 registros para las tablas que aplique	Basadas en la relaciones de las tablas y tipos de usuarios
2	Implementación de base datos Postgres sobre Heroku	<Quien> Team Dev <Quiero> Contar con una base de datos relación sobre un servicio de nube para el manejo de datos y transacciones <Para> Los usuarios y administradores puedan llevar a cabo las funcionalidades para las cuales fue desarrollado el proyecto	<Dado que>Team Dev Implemente y publique la base de datos Postgres sobre el servicio de nube <Cuando>Fase inicial del proyecto <Entonces>Team Dev cuenta con el soporte para crear consultas y funcionales de la API	1.- La base de datos se encuentra generada bajo postgres, publicada y accesible para su ointegración y consulta	NA
3	Generación de modelo base del proyecto	<Quien>Team Dev <Quiero> Implementar un modelo base <Para>Poder respresentar la data de nuestra aplicación e interactura de forma adecuada	<Dado que>Team Dev implemente el modelo base con base en las definiciones previas y análisis definido <Cuando>Fase inicial <Entonces>Team Dev cuenta con la estructura basica para poder representar la data, interactura con ella y genrar la funcionalidades posteriores	1.- La representación del modelo concuerda con las definiciones de las estructuras de la data	Basadas en la relaciones de las tablas y tipos de datos
4	Generación de CRUD	<Quien>Team Dev <Quiero> Generar las consultas y funcionalidades CRUD <Para>Administradores puedan usar estas funcionalidades sobre las tablas de productos a vender(automoviles), usuarios y ventas	<Dado que>Team Dev implemente las funcionaldes de CRUD <Cuando>Posterior a contar con la base y modelo establecido <Entonces>usuarios de tipo administrador contaran con las funcionalidades de Alta, Baja, actualización y borrado para las diferentes tablas utilizadas	1.- Team Dev puede ejecutar dichas funcionalidades y consultas sobre las tablas definidas	Ejecución de funcionalidades basada en roles
5	Definición de servicio de obtención de datos basados en selección	<Quien>Team Dev <Quiero> Implementar servicios para la obtención de datos especificos <Para>Los usuarios pueden realizar consultas con base en sus requerimientos	<Dado que>TeamDev Implemente los servicios para la obtención de data especifica <Cuando>Posterior a contar con las funcionalidades de CRUD implementadas <Entonces>Se pueden realizar busquedas por ID y los clientes pueden realizar consultas basandos en campos	1.- La información que se arroja basada en la consulta es coherente e integra	Basados en los privilegios de consultas
6	Definición de routers	<Quien>Team Dev <Quiero> Modularizar la estructura del proyecto <Para>Restructurar la App y permitir el control de nuestros flujos de redireccionamiento	<Dado que>Team Dev Defina e implemente los routers dentro del proyecto <Cuando>Una vez que se cuenten con los directorios definidos para controladores y modelos <Entonces>Contaremos con una estructura reoganizada y un funcionamiento adecuado de los endpoints	1.- Los endpoints definidos tendran un comportamiento esperado adecuado	NA

7	Identificación y protección de información sensible	<Quien>Team Dev <Quiero> Identificar la información sensible <Para> Proteger la integridad de la información sensible de nuestra App	<Dado que>Team Dev implemente el paquete cripto basado en el análisis de información <Cuando>Posterior a contar con un esquema de información reestructurado <Entonces>Podremos asegurar de mejor manera la información sensible detectadaen la definición	1.- La información sensible estará protegida de mejor manera	NA
8	Gestión de accesos para usuarios autorizados	<Quien>Team Dev <Quiero> Definir e implementar el manejo de sesiones <Para>Gestionar y controlar el acceso a los usuarios autorizado	<Dado que>Team Dev implemente la configuración de passport <Cuando>Una vez que se cuenten con los directorios definidos para controladores y modelos <Entonces>Podremos controlar los accesos de los usuarios de acuerdo a sus privilegios	1.- Los usuarios deberan de poder acceder a los recursos basados en sus privilegios	Basados en los privilegios definidos por usuarios
9	Deploy de proyecto	<Quien>Team Dev <Quiero> Publicar proyeto en servicio de nube <Para>App accesible a los clientes para su uso en producción y proveer los servicios para lo que fue creado	<Dado que>Team Dev realicé el deploy de la app en una plataforma de nube <Cuando>Al concluir con la implementación, funcionalidades y pruebas de la App <Entonces>Los servicios estarán disponibles para los usuarios autorizados y clientes cumpliendo con el objetivo definido	1.- Los usuarios y clientes deberan de poder acceder desde internet a los servicios basados en sus roles y privilegios	Basados en los privilegios definidos por usuarios
10	Generación de documentación de servicios	<Quien>Team Dev <Quiero> Realización de Swagger con la documentación de los servicios <Para>Contar con la información estructurada, documentada y accesible de los servicios implementados	<Dado que>Team Dev genere un Swagger con la información de los servicios <Cuando>Al concluir con la implementación, funcionalidades y pruebas de la App <Entonces>Se tendra accesible para su revisión la documentación de los servicios del proyecto	1.- Team Dev y nuevos miembros podran acceden y consultar la documentación y actualizacion de forma adecuada	NA

Se anexa documento Excel en repositorio para mejor visualización

Tipos de usuarios

Los usuarios de la API pueden ser administradores o clientes. A continuación, se detalla que permisos tiene cada tipo de usuario dependiendo de su rol:

- Rol de administrador (administrator):
  1) Crear, editar y eliminar autos.
  2) Obtener, crear, actualizar y eliminar otros usuarios administradores. Debe iniciar sesión para poder realizar estas acciones.
  3) Crear, actualizar, eliminar autos. Debe iniciar sesión para poder realizar estas acciones.
  4) Actualizar el estado de las compras realizadas por los clientes. Los estados válidos son:
     a. Received – Orden nueva creada por cliente y registrada en el sistema.
     b. Processed – Orden ha sido procesada.
     c. Shipped – El auto de la orden ha sido enviado.
     d. Delivered – El auto de la orden ha sido entregado al cliente.
- Rol de cliente (customer):
  1) Ver y filtrar listado de autos.
  2) Comprar autos y dar seguimiento a sus compras.

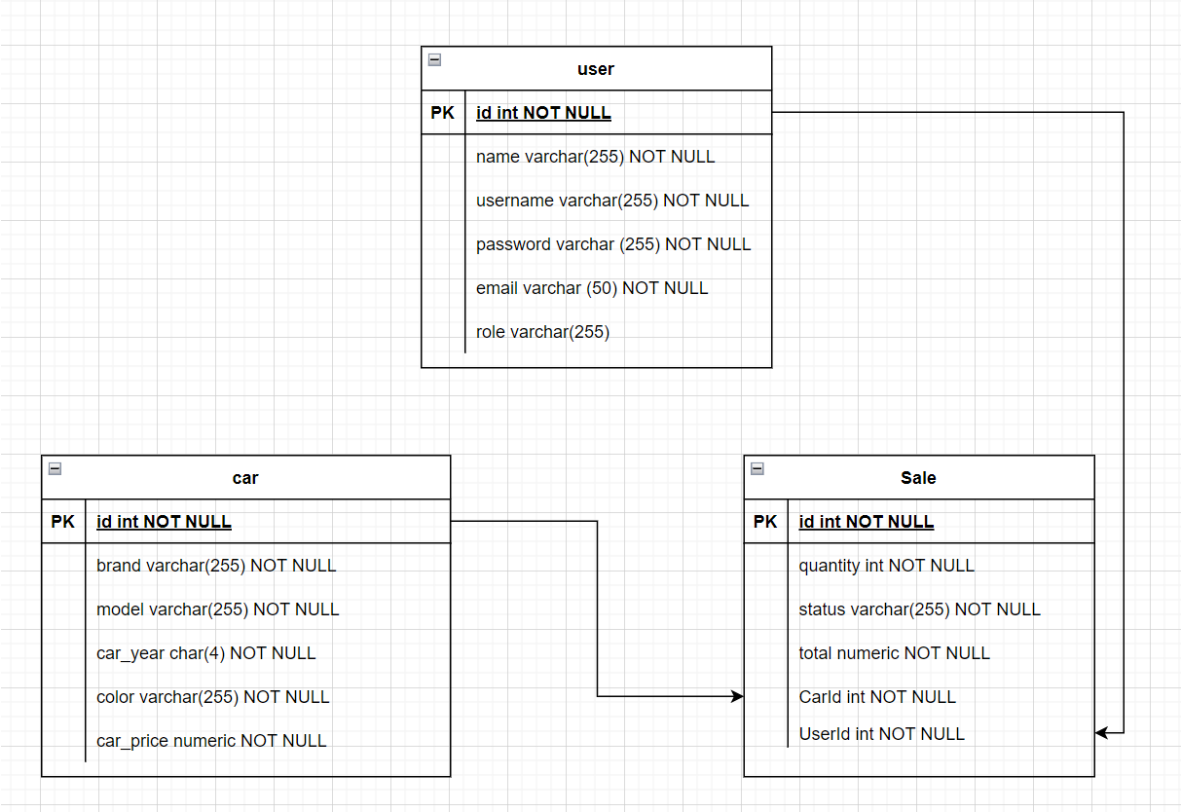
Entidades del proyecto

Entidades generadas para la definición de base de datos

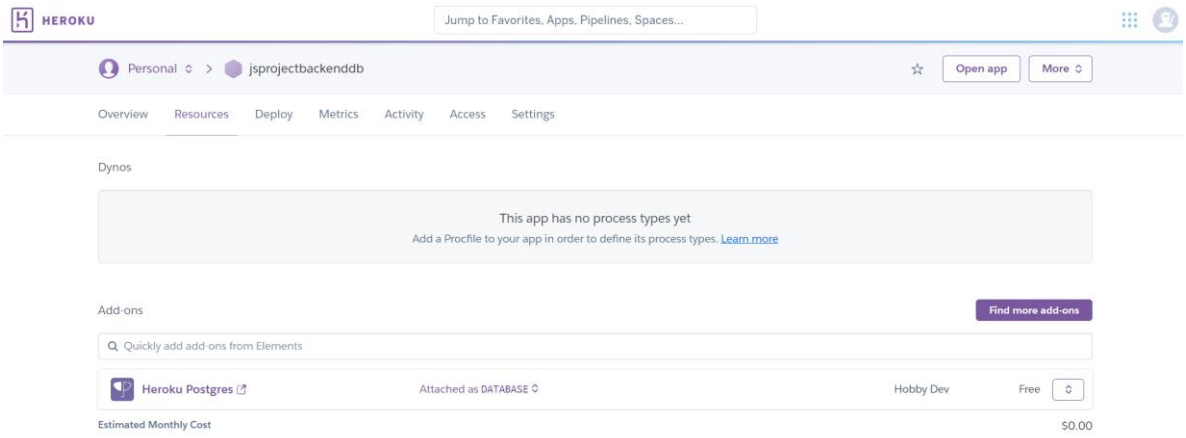
- **Users**. Tabla para almacenar los datos generales de los usuarios de la API, esto incluye:

- **Cars**. Tabla para almacenar datos de los automóviles semi-nuevos que están en venta.
- **Sales**. Tabla para almacenar los datos de una compra de automóvil realizada por un usuario con el rol *customer*.

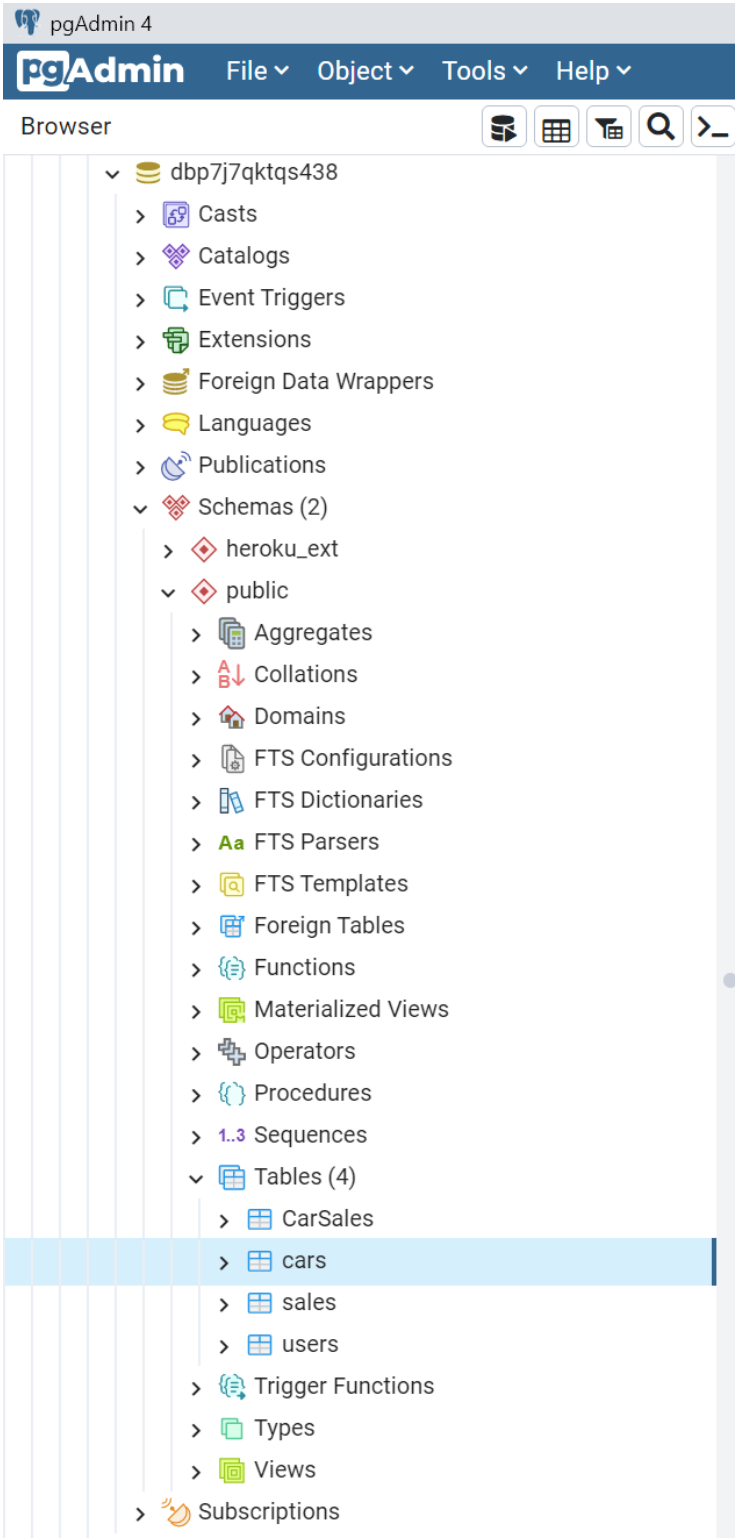
Definición de diagrama Entidad-Relación



Creación de proyecto y DB Posrgres sobre Heruku



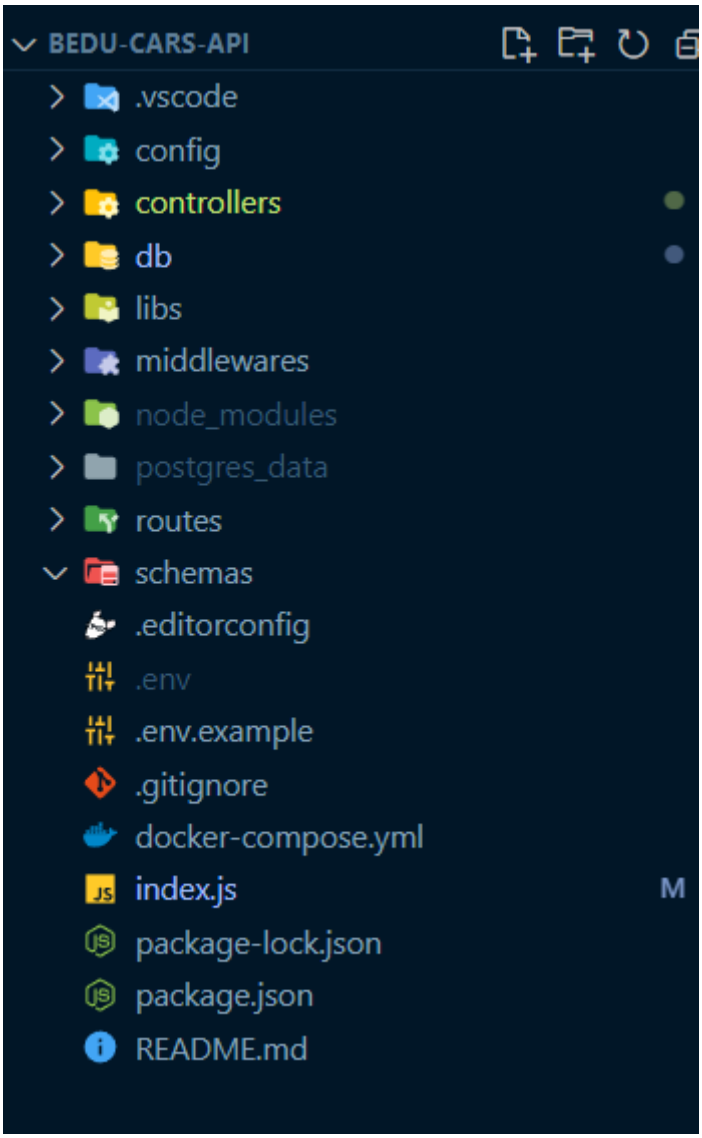
Conexión a la DB Postgres Heroku a través de PgAdmin4



Uso de Mockaroo para poblar base de datos

	A	B	C	D	E
1	carmanufact	carmodel	caryear	carcolor	carprice
2	BMW	X5	2012	Violet	$456,716.19
3	Dodge	Dakota	1998	Red	$336,277.21
4	Lincoln	Blackwood	2002	Red	$328,525.74
5	Saturn	Outlook	2008	Puce	$177,799.96
6	Nissan	Cube	2010	Indigo	$287,036.44
7	Ford	Flex	2009	Crimson	$293,693.15
8	Jeep	Grand Cherokee	2011	Yellow	$133,836.96
9	Acura	CL	1999	Aquamarine	$293,228.06
10	Mercury	Grand Marquis	2002	Teal	$458,301.24
11	Dodge	Caravan	1998	Fuscia	$176,067.36
12	Ford	Fiesta	2013	Crimson	$398,767.42
13	Mercury	Cougar	1991	Khaki	$395,951.40
14	Cadillac	Escalade EXT	2012	Mauv	$168,003.25
15	Chevrolet	Silverado 3500	2009	Turquoise	$192,624.21
16	Ford	E-Series	1993	Crimson	$259,185.81
17	Chevrolet	Silverado 1500	2009	Yellow	$434,535.37
18	Mercury	Cougar	1967	Goldenrod	$342,485.97
19	Hyundai	Genesis	2010	Khaki	$183,775.86
20	GMC	2500	1992	Blue	$167,375.67
21	Volkswagen	Corrado	1994	Turquoise	$178,008.97
22	Mercedes-B	R-Class	2009	Red	$376,878.72
23	Mazda	929	1993	Puce	$183,350.42
24	Mercury	Grand Marquis	1993	Mauv	$165,060.23
25	Jaguar	XJ Series	2003	Khaki	$342,001.81
26	Mitsubishi	Pajero	2000	Indigo	$450,481.83
27	Mazda	626	1997	Puce	$456,621.54
28	Audi	5000S	1985	Yellow	$334,056.09
29	Lincoln	Continental	2002	Blue	$174,442.94
30	Dodge	Ram Van B350	1993	Pink	$293,277.93
31	Land Rover	Freelander	2010	Khaki	$161,075.77
32	Mercedes-B	W126	1981	Fuscia	$343,445.28
33	Lexus	ES	2005	Pink	$274,410.97
34	Chevrolet	Astro	2002	Red	$234,184.18
35	Porsche	911	2009	Yellow	$478,400.00
36	Chevrolet	Prizm	1999	Maroon	$388,042.99
37	Hyundai	Sonata	1994	Maroon	$291,402.97
38	BMW	X6	2010	Teal	$390,735.88
39	Acura	NSX	1993	Green	$385,016.26
40	Mercedes-B	S-Class	1994	Goldenrod	$394,170.71

Estructura del proyecto



Dependencias del proyecto

package.json > ...

```json
{
  "name": "bedu-cars-api",
  "version": "1.0.0",
  "description": "BEDU Backend Fundamentals project. Used Cars for Sale API.",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "dev": "nodemon index.js",
    "start": "node index.js"
  },
  "keywords": [
    "expressjs",
    "nodejs",
    "javascript",
    "api",
    "cars"
  ],
  "author": "Asael Hernández <asael.dev@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "@hapi/boom": "^10.0.0",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.1",
    "pg": "^8.8.0",
    "pg-hstore": "^2.3.4",
    "sequelize": "^6.24.0"
  }
}
```

Definir consultas usando los métodos de Sequelize

```
const publicAttributes = ['id', 'name', 'userName', 'email'];

class UsersController {
  constructor() {}

  async all(role = 'customer') {
    return await models.Users.findAll({
      attributes: publicAttributes,
      where: {
        role: role,
      },
    });
  }
```

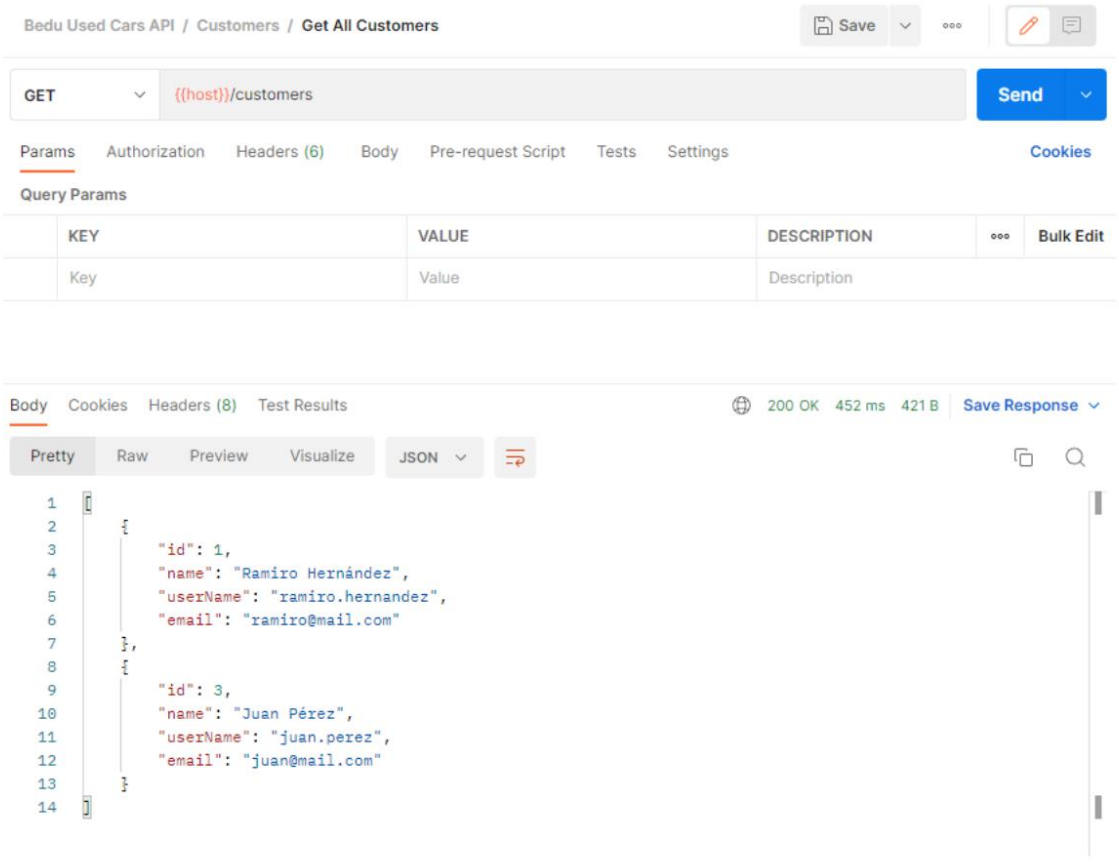
```
async create(data, role = 'customer') {
  try {
    const { password } = data;
    const { salt, hash } = Users.createPassword(password);
    return await models.Users.create({
      ...data,
      role,
      password_hash: hash,
      password_salt: salt,
    });
  } catch (error) {
    throw new boom.internal(error.message);
  }
}
```

Definición y configuración de Express

index.js > app.get('/') callback

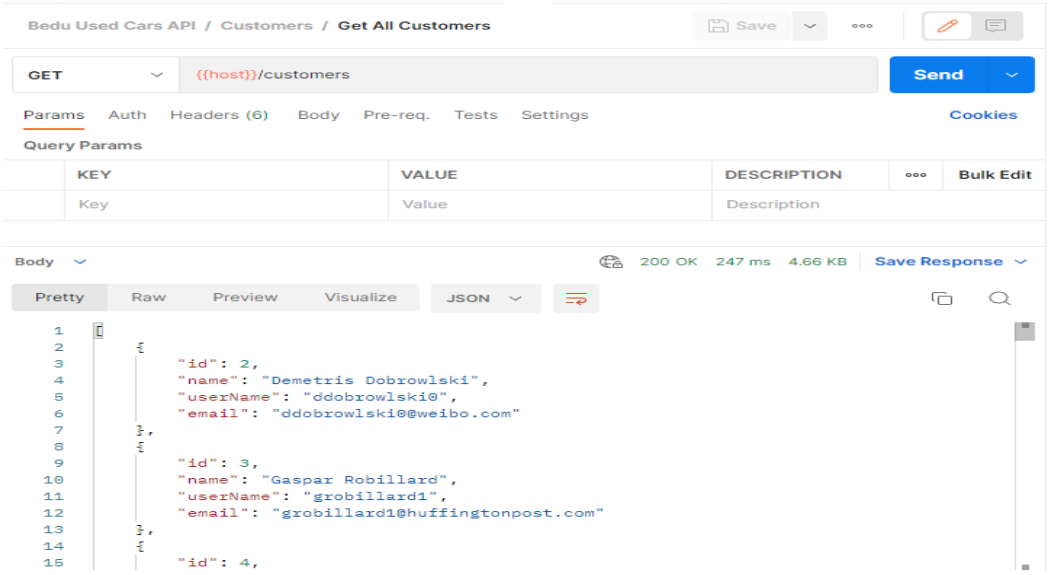
```
const config = require('./config/config');
const express = require('express');
const cors = require('cors');
const {
  boomErrorHandler,
  ormErrorHandler,
} = require('./middlewares/error.handler');
const routerApi = require('./routes');

const app = express();

const allowedDomains = ['https://localhost:8080'];

const corsSettings = {
  origin: (origin, callback) => {
    if (allowedDomains.includes(origin) || !origin) {
      callback(null, true);
    } else {
      callback(new Error('Domain not allowed.', false));
    }
  },
};

app.use(express.json());
app.use(cors(corsSettings));

app.get('/', (req, res) => {
  res.send('Welcome to Bedu Used Cars for Sale API!');
});

routerApi(app);

app.use(ormErrorHandler);
app.use(boomErrorHandler);

app.listen(config.port, () => {
  console.log('App running on port:', config.port);
});

```

Validación y petición GET a URL

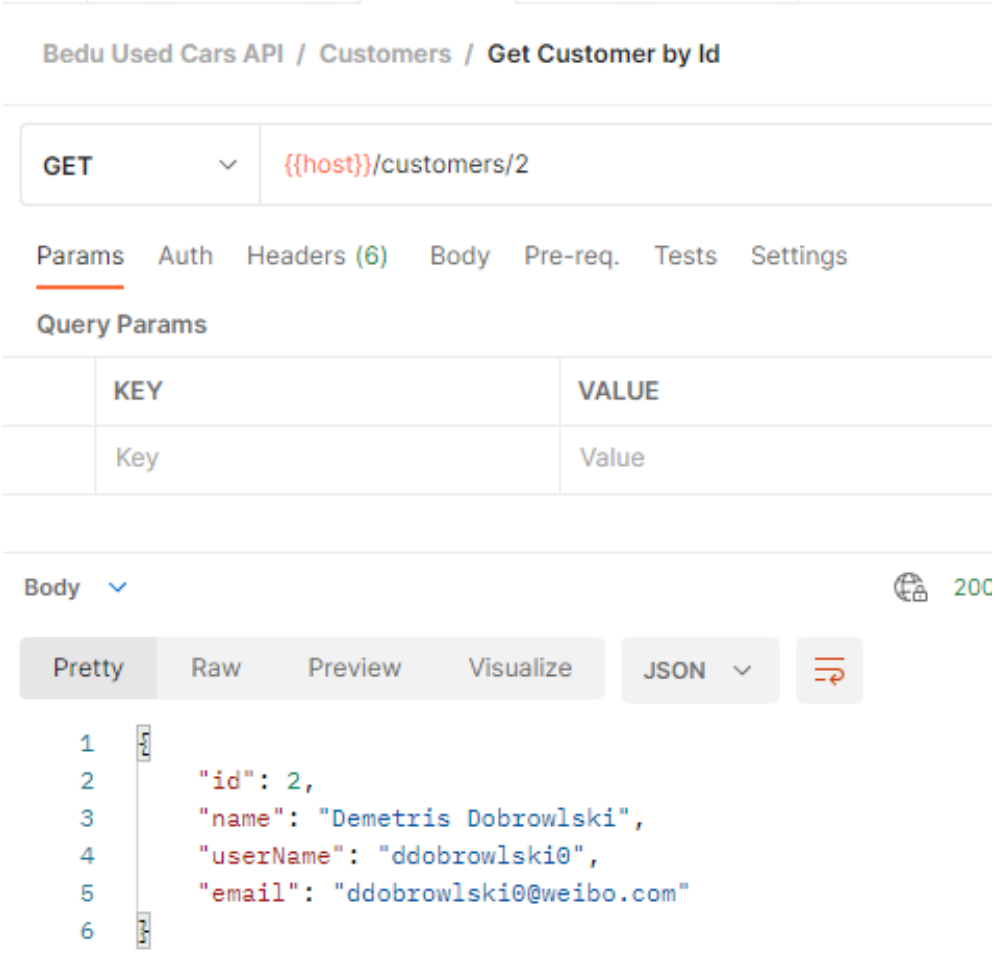


Validación de servicios CRUD

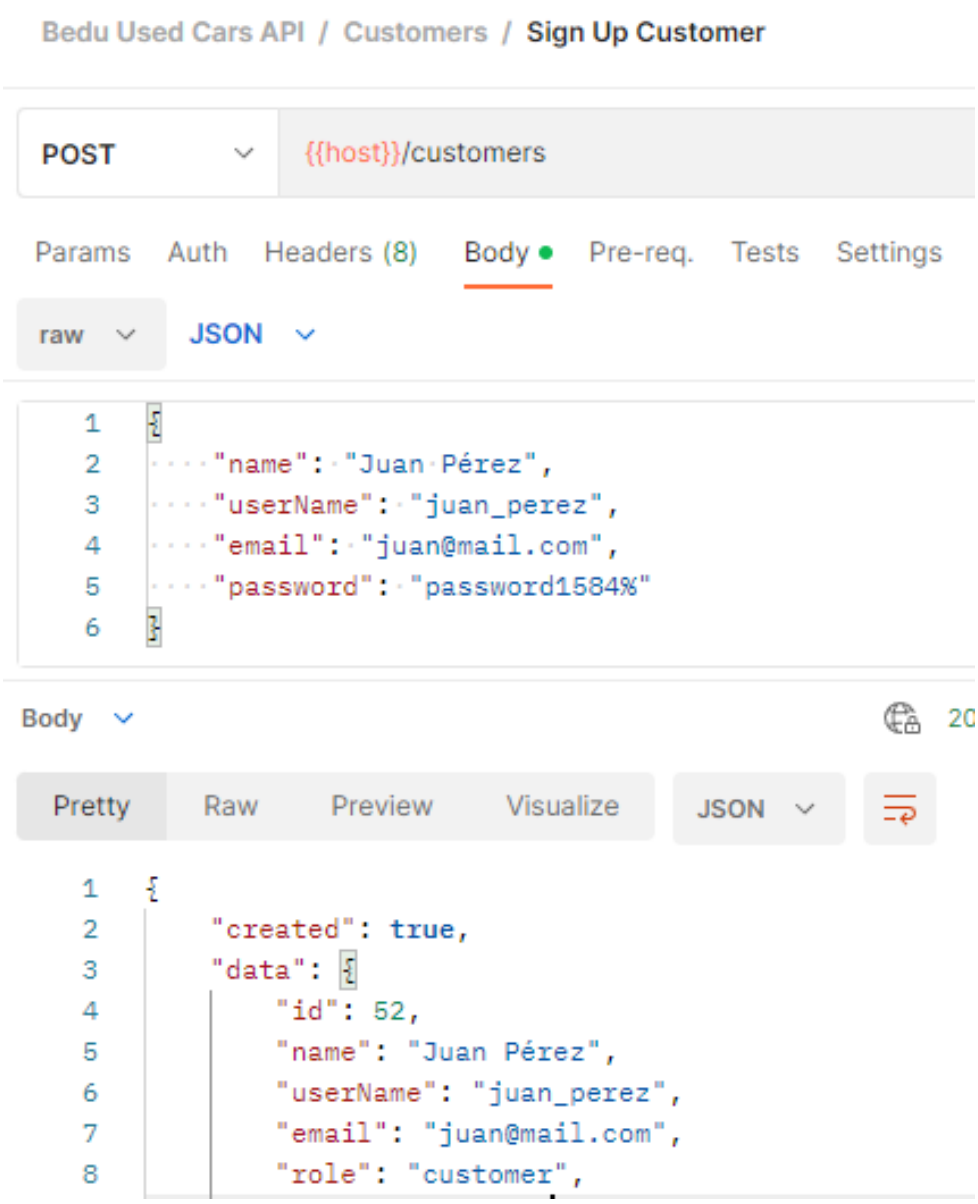
Get All Customers



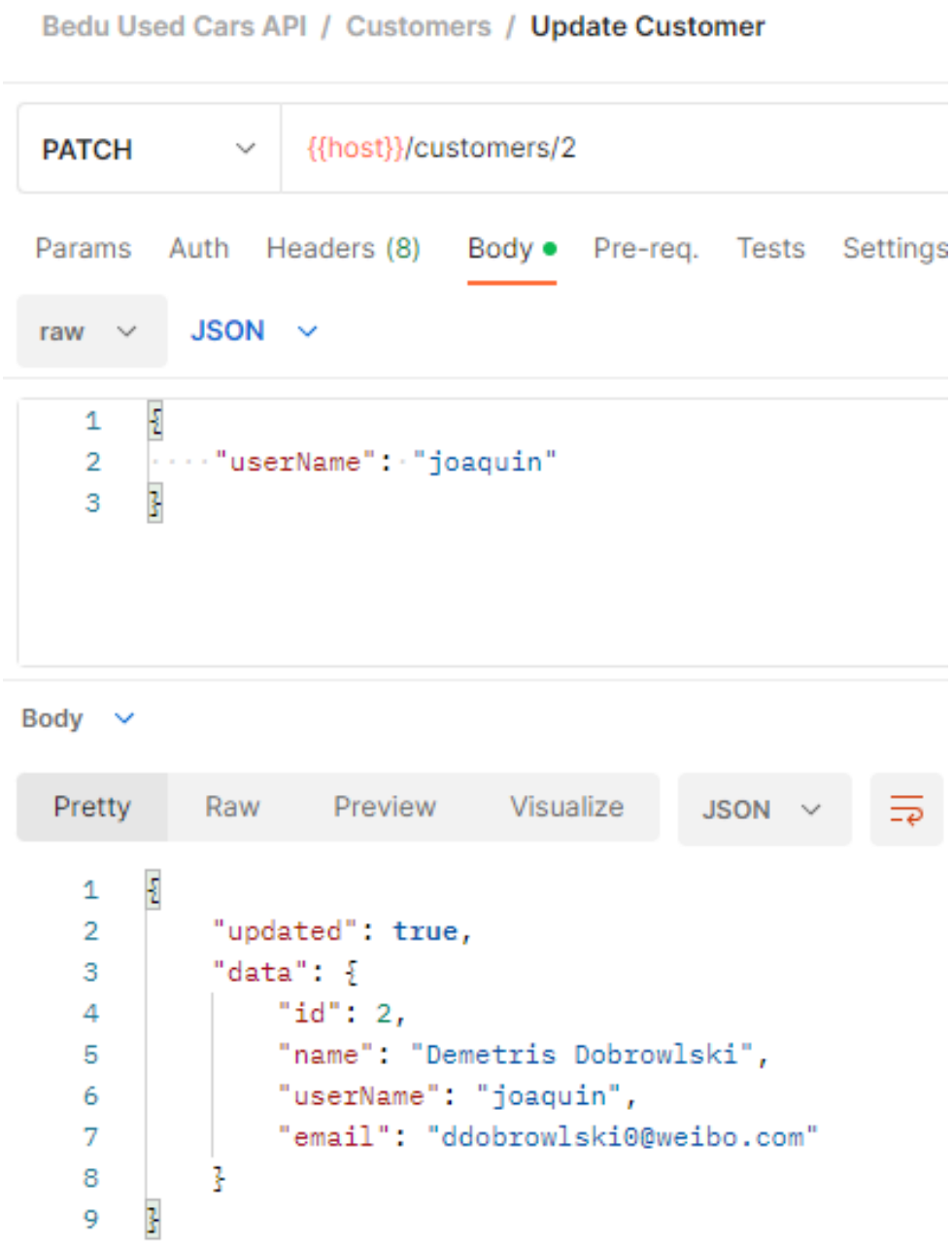
Get Customer by Id



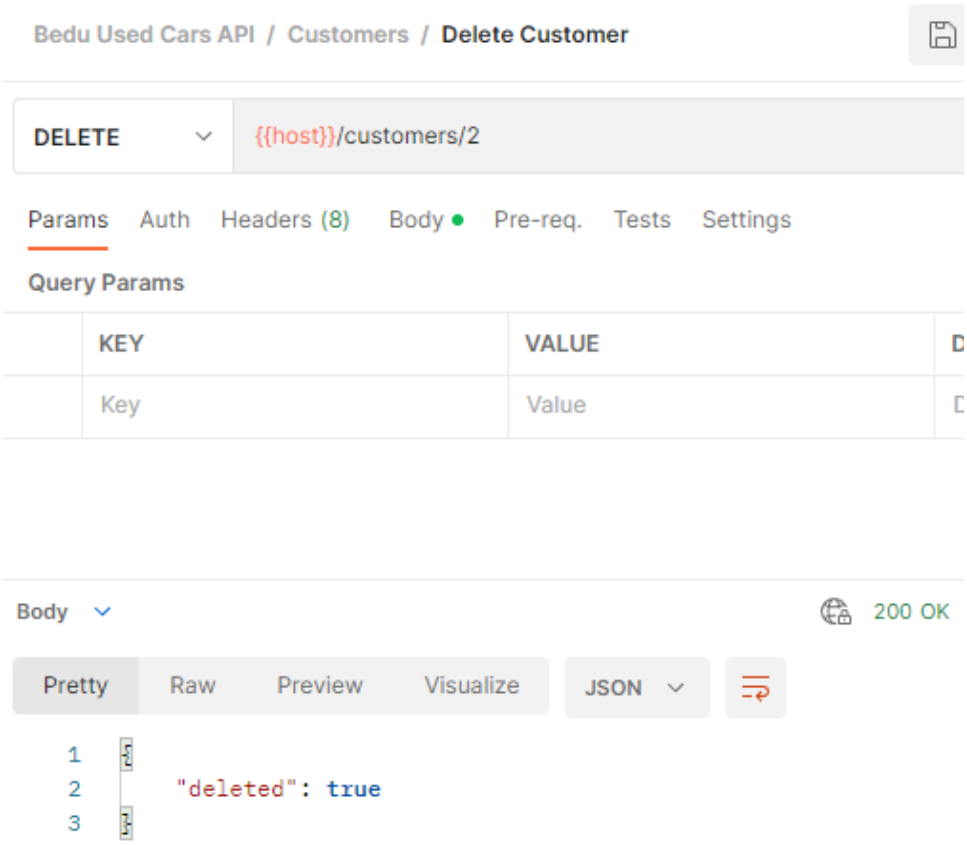
Sign Up Customer (Create Customer)



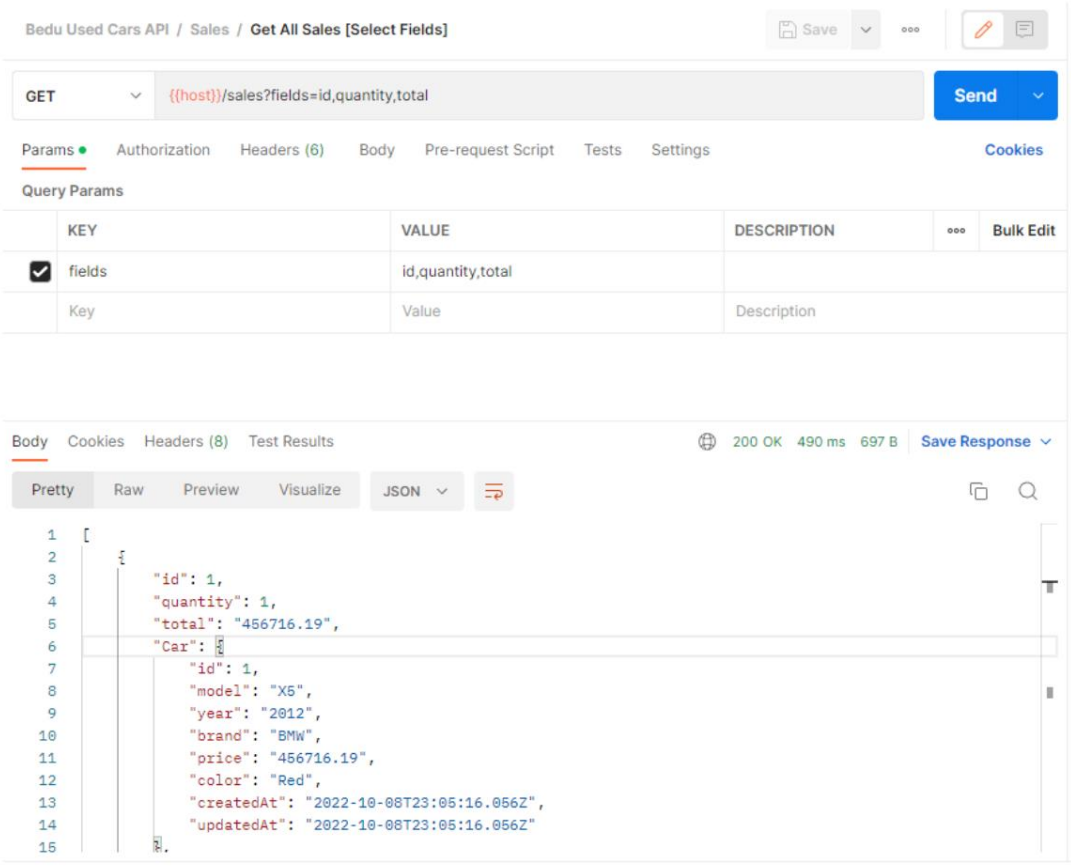
Update Customer



Delete Customer

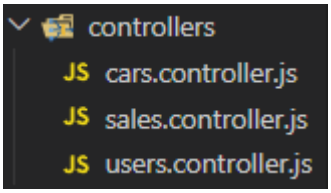


Definición y validación de servicios por campo



Controladores, Modelos y Routers por entidad

- **Controladores**



- Cars.controller.js

```
controllers > JS cars.controller.js > ...
1   const boom = require('@hapi/boom');
2   const { models } = require('../libs/sequelize');
3
4   class CarsController {
5     constructor() {}
6
7     async all() {
8       return await models.Cars.findAll();
9     }
10
11    async findOne(id) {
12      const car = await models.Cars.findOne({
13        where: {
14          id,
15        },
16      });
17
18      if (!car) {
19        throw new boom.notFound('Car was not found.');
20      }
21
22      return car;
23    }
```

- Sales.controller.js

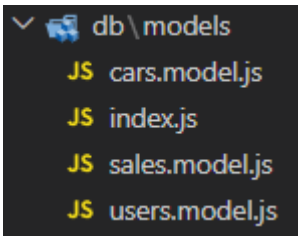
```
controllers > JS sales.controller.js > SalesController > findOne
1   const boom = require('@hapi/boom');
2   const { models } = require('../libs/sequelize');
3
4   const publicAttributes = ['id', 'quantity', 'status', 'total'];
5
6   class SalesController {
7     constructor() {}
8
9     async all(fields = null) {
10      const attributes = fields !== null ? fields : publicAttributes;
11      return await models.Sales.findAll({
12        attributes: attributes,
13        include: ['Car', 'User'],
14      });
15    }
16
17    async findOne(id) {
18      const sale = await models.Sales.findOne({
19        attributes: publicAttributes,
20        where: {
21          id,
22        },
23        include: ['Car', 'User'],
```

- Users.controller.js

controllers > JS users.controller.js > logIn

```
const boom = require('@hapi/boom');
const { Users } = require('../db/models/users.model');
const { models } = require('../libs/sequelize');

const publicAttributes = ['id', 'name', 'userName', 'email'];

class UsersController {
  constructor() {}

  async all(role = 'customer') {
    return await models.Users.findAll({
      attributes: publicAttributes,
      where: {
        role: role,
      },
    });
  }

  async findOne(id, role = 'customer') {
    const user = await models.Users.findOne({
      attributes: publicAttributes,
      where: { id, role },
    });
```

- **Modelos**



- Cars.model.js

db > models > JS cars.model.js > ...

```
const { Model, DataTypes, Sequelize } = require('sequelize');

const CARS_TABLE = 'cars';

const CarsSchema = {
  id: {
    allowNull: false,
    autoIncrement: true,
    primaryKey: true,
    type: DataTypes.INTEGER,
  },
  model: {
    allowNull: false,
    type: DataTypes.STRING,
  },
  year: {
    type: DataTypes.CHAR(4),
    validate: {
      isNumeric: true,
      len: [4, 4],
    },
  },
  brand: {
```

- Index.js

db > models > JS index.js > ...

```
const { Users, UsersSchema } = require('./users.model');
const { Cars, CarsSchema } = require('./cars.model');
const { Sales, SalesSchema } = require('./sales.model');

function setupModels(sequelize) {
  Users.init(UsersSchema, Users.config(sequelize));
  Cars.init(CarsSchema, Cars.config(sequelize));
  Sales.init(SalesSchema, Sales.config(sequelize));
  Users.associate(sequelize.models);
  Sales.associate(sequelize.models);
  Cars.associate(sequelize.models);

  Sales.addHook('beforeCreate', 'setTotal', async (sale, options) => {
    const car = await sequelize.models.Cars.findByPk(sale.CarId);
    sale.total = parseFloat(sale.quantity * car.price).toFixed(2);
  });
}

module.exports = setupModels;
```

- Sales.model.js

db > models > JS sales.model.js > ...

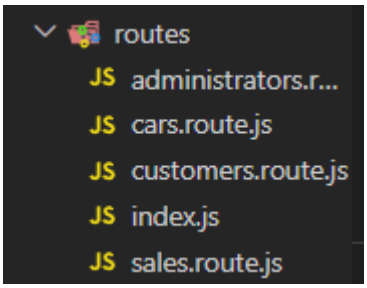
```
const { Model, DataTypes } = require('sequelize');

const SALES_TABLE = 'sales';

const SalesSchema = {
  id: {
    allowNull: false,
    autoIncrement: true,
    type: DataTypes.INTEGER,
    primaryKey: true,
  },
  quantity: {
    type: DataTypes.SMALLINT,
    defaultValue: 1,
    min: 1,
    max: 3,
  },
  status: {
    type: DataTypes.STRING,
    defaultValue: 'Received',
    allowNull: false,
    validate: {
      isIn: [['Received', 'Processed', 'Shipped', 'Delivered', 'Canceled']],
```

- Users.model.js

db > models > JS users.model.js > ...

```
const { Model, DataTypes } = require('sequelize');
const crypto = require('crypto');

const USERS_TABLE = 'users';

const UsersSchema = {
  id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    autoIncrement: true,
    primaryKey: true,
  },
  name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  userName: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
    validate: {
      is: /^[a-zA-Z0-9_-]+$/,
    },
```

- **Routers**



  - Adminstrators.route.js

routes > JS administrators.route.js > <unknown>

```
const { UsersController, logIn } = require('../controllers/users.controller');
const router = require('express').Router();

const usersController = new UsersController();

router.get('/', async (req, res) => {
  const administrators = await usersController.all('administrator');
  return res.status(200).json(administrators);
});

router.get('/:id', async (req, res, next) => {
  const { id } = req.params;

  try {
    const administrator = await usersController.findOne(id, 'administrator');
    return res.status(200).json(administrator);
  } catch (error) {
    next(error);
  }
});

router.post('/', async (req, res, next) => {
  const data = req.body;

  try {
    const administrator = await usersController.create(data, 'administrator');
    return res.status(201).json({
      created: true,
      data: administrator,
    });
  } catch (error) {
    next(error);
  }
});

router.patch('/:id', async (req, res, next) => {
  const data = req.body;
  const { id } = req.params;

  try {
    const rowsUpdated = await usersController.update(id, data, 'administrator');
    const administrator = await usersController.findOne(id, 'administrator');
    return res.status(200).json({
      updated: rowsUpdated > 0,
      data: administrator,
```

- Cars.route.js

routes > JS cars.route.js > ...

```
const CarsController = require('../controllers/cars.controller');
const router = require('express').Router();

const carsController = new CarsController();

router.get('/', async (req, res) => {
  const cars = await carsController.all();
  return res.status(200).json(cars);
});

router.get('/:id', async (req, res, next) => {
  const { id } = req.params;

  try {
    const car = await carsController.findOne(id);
    return res.status(200).json(car);
  } catch (error) {
    next(error);
  }
});

router.post('/', async (req, res, next) => {
  const data = req.body;

  try {
    const car = await carsController.create(data);
    return res.status(201).json({
      created: car !== null,
      data: car,
    });
  } catch (error) {
    next(error);
  }
});

router.patch('/:id', async (req, res, next) => {
  const data = req.body;
  const { id } = req.params;

  try {
    const updatedRecords = await carsController.update(id, data);
    const customer = await carsController.findOne(id);
    return res.status(200).json({
      updated: updatedRecords > 0,
      data: customer,
```

- Customers.route.js

routes > JS customers.route.js > ...

```
const {
  UsersController,
  logIn,
} = require('../controllers/users.controller');
const router = require('express').Router();

const usersController = new UsersController();

router.get('/', async (req, res) => {
  const customers = await usersController.all();

  return res.status(200).json(customers);
});

router.get('/:id', async (req, res, next) => {
  const { id } = req.params;
  try {
    const customer = await usersController.findOne(id);
    return res.status(200).json(customer);
  } catch (error) {
    next(error);
  }
});

router.post('/', async (req, res, next) => {
  const data = req.body;

  try {
    const customer = await usersController.create(data);
    return res.status(201).json({
      created: true,
      data: customer,
    });
  } catch (error) {
    next(error);
  }
});

router.patch('/:id', async (req, res, next) => {
  const data = req.body;
  const { id } = req.params;

  try {
    const updatedRecords = await usersController.update(id, data);
    const customer = await usersController.findOne(id);
```

- Index.js

routes > JS index.js > ...

```
const express = require('express');
const cars = require('./cars.route');
const sales = require('./sales.route');
const customers = require('./customers.route');
const administrators = require('./administrators.route');

const routerApi = (app) => {
    const router = express.Router();
    app.use('/api/v1', router);
    router.use('/cars', cars);
    router.use('/sales', sales);
    router.use('/customers', customers);
    router.use('/administrators', administrators);
}

module.exports = routerApi;
```

- Sales.route.js

routes > JS sales.route.js > ...

```
const router = require('express').Router();
const SalesController = require('../controllers/sales.controller');

const salesController = new SalesController();

router.get('/', async (req, res) => {
  const { fields } = req.query;
  const parsedFields = fields ? fields.split(',') : null;
  const sales = await salesController.all(parsedFields);
  return res.status(200).json(sales);
});

router.get('/:id', async (req, res, next) => {
  const { id } = req.params;

  try {
    const sale = await salesController.findOne(id);
    return res.status(200).json(sale);
  } catch (error) {
    next(error);
  }
});

router.post('/', async (req, res, next) => {
  const data = req.body;

  try {
    const sale = await salesController.create(data);
    return res.status(201).json({
      created: sale !== null,
      data: sale,
    });
  } catch (error) {
    next(error);
  }
});

router.patch('/:id', async (req, res, next) => {
  const { status } = req.body;
  const { id } = req.params;

  try {
    const updatedRecords = await salesController.updateStatus(id, status);
    const sale = await salesController.findOne(id);
    return res.status(200).json({
```

Configuración de Router

```
JS index.js > ...
  8   const routerApi = require('./routes');
  9
```

```
JS index.js > ...
 31   routerApi(app);
 32
```

Definición y configuración de protección de información a través de plataforma Cripto

- Se identificó como datos sensibles las contraseñas de los *users* dentro de la API por lo que se realizó una configuración de protección de datos.

```
db > models > JS users.model.js > ...
  1   const { Model, DataTypes } = require('sequelize');
  2   const crypto = require('crypto');
```

```
db > models > JS users.model.js > ...
 33     password_hash: {
 34       type: DataTypes.TEXT(512),
 35       allowNull: true,
 36     },
 37     password_salt: {
 38       type: DataTypes.STRING,
 39       allowNull: true,
 40     },
```

```
db > models > JS users.model.js > ...
 70   Users.createPassword = function (plainText) {
 71     const salt = crypto.randomBytes(16).toString('hex');
 72     const hash = crypto
 73       .pbkdf2Sync(plainText, salt, 10000, 512, "sha512")
 74       .toString("hex");
 75     return { salt: salt, hash: hash };
 76   };
 77
 78   Users.validatePassword = function (password) {
 79     const hash = crypto
 80       .pbkdf2Sync(password, salt, 10000, 512, "sha512")
 81       .toString("hex");
 82     return this.password_hash === hash;
 83   };
```

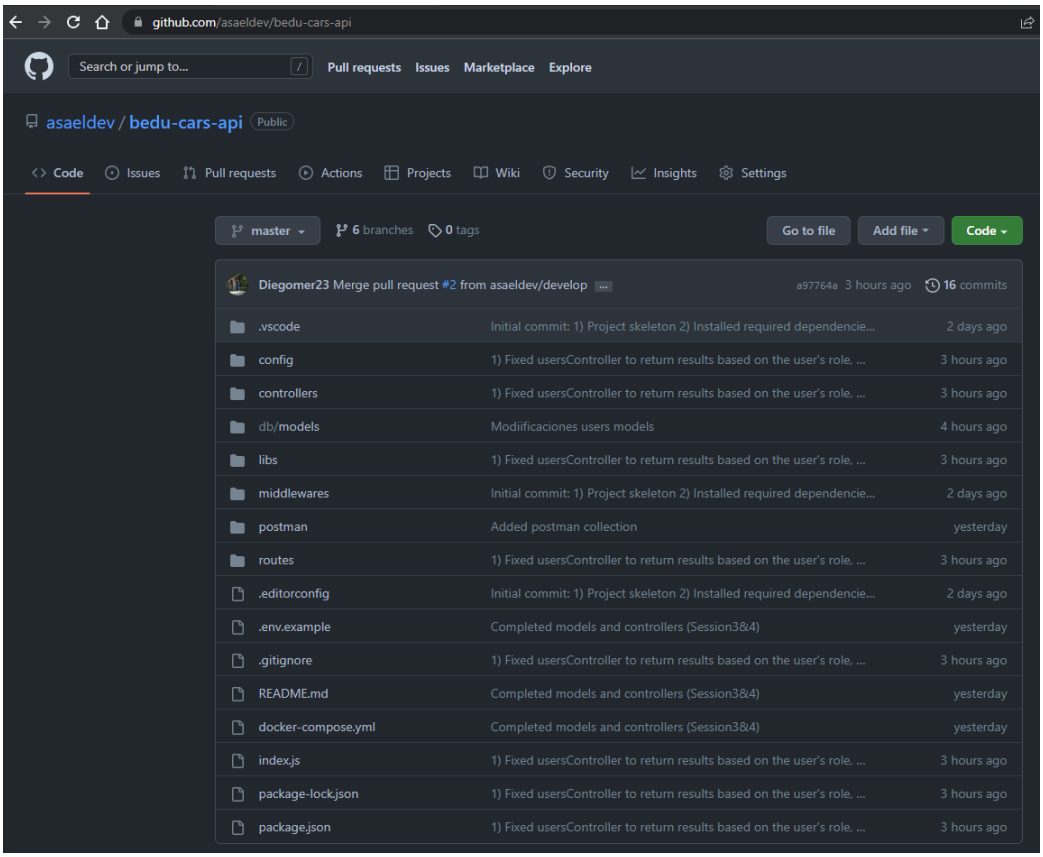
Definición y configuración de Passport

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const User = require('../db/models/users.model');

passport.use(new LocalStrategy({
    usernameField: 'email',
    passwordField: 'password',

}, function (email, password, done) {
    User.findOne({ where: {email: email}}).then(function (user) {
        if (!user || !user.validatePassword(password)) {
            return done(null, false, {errors: {'user or email': 'is incorrect'}})
        }
        return done(null, user);
    }).catch(done)
}));

module.exports = passport;
```

Definición de accesos por servicio

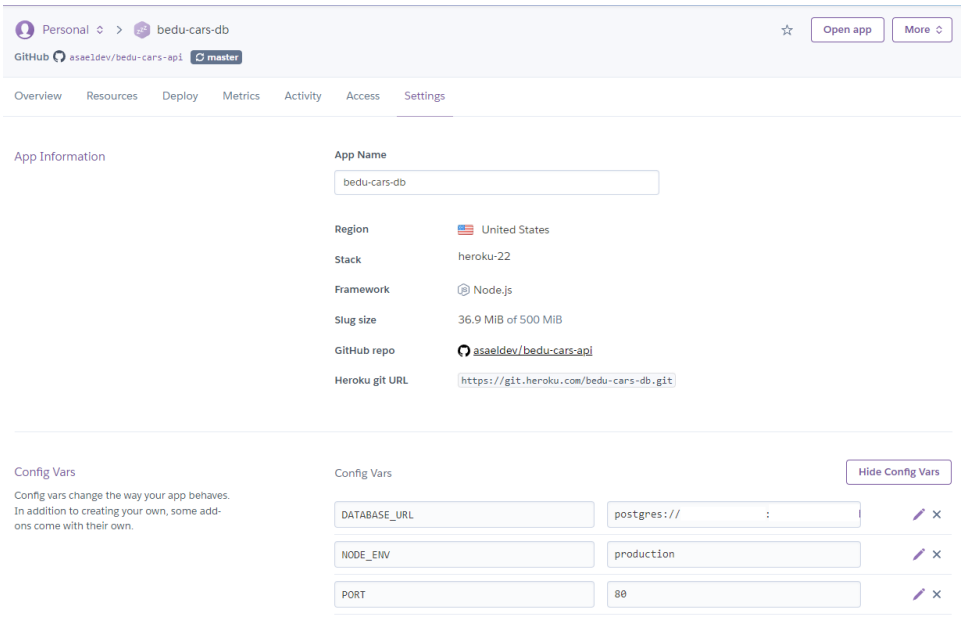
```
const boom = require('@hapi/boom');
const { models } = require('../libs/sequelize');

function checkRoles(...roles) {
  return async (req, res, next) => {
    const { user: userName } = req.user;
    const user = await models.Users.findOne({ where: { userName: userName } });
    if (roles.includes(user.role)) {
      next();
    } else {
      next(
        boom.forbidden(
          'User does not have permissions to perform the requested operation.'
        )
      );
    }
  };
}

module.exports = { checkRoles };
```

- URL del repositorio: https://github.com/asaeldev/bedu-cars-api

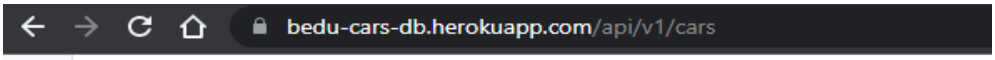


Despliegue de aplicación sobre Heroku

- API URL: https://bedu-cars-db.herokuapp.com

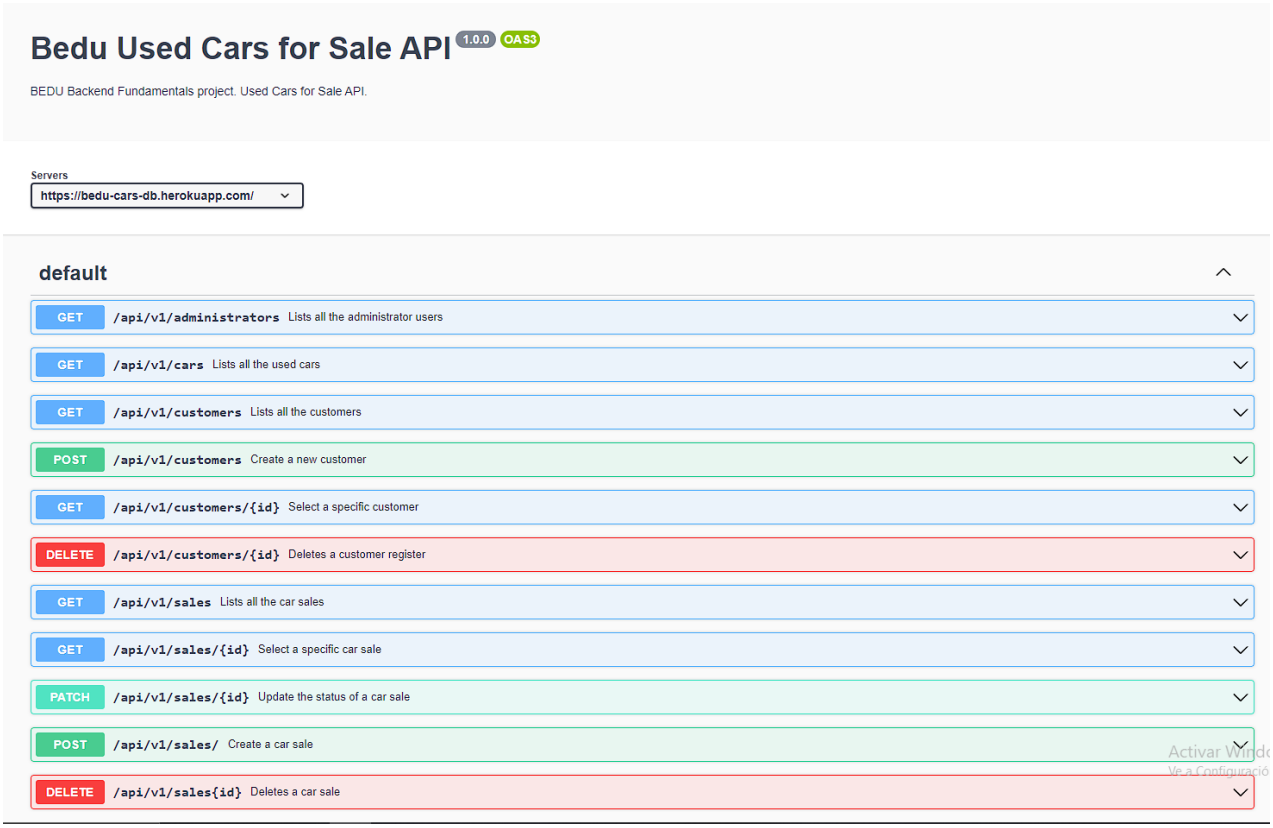


Validación de servicios Producción




```
// 20221009204511
// https://bedu-cars-db.herokuapp.com/api/v1/cars

[
  {
    "id": 1,
    "model": "Quest",
    "year": "1997",
    "brand": "Nissan",
    "price": "32006.49",
    "color": "Turquoise",
    "createdAt": "2022-04-23T00:00:00.000Z",
    "updatedAt": "2021-12-29T00:00:00.000Z"
  },
  {
    "id": 2,
    "model": "E-Series",
    "year": "2004",
    "brand": "Ford",
    "price": "35814.77",
    "color": "Mauv",
    "createdAt": "2021-10-28T00:00:00.000Z",
    "updatedAt": "2022-07-01T00:00:00.000Z"
  },
  {
    "id": 3,
    "model": "F-Series",
    "year": "1990",
    "brand": "Ford",
    "price": "49975.85",
    "color": "Aquamarine",
    "createdAt": "2022-06-25T00:00:00.000Z",
    "updatedAt": "2022-04-10T00:00:00.000Z"
  },
  {
    "id": 4,
    "model": "Savana 3500",
    "year": "1998",
    "brand": "GMC",
    "price": "17962.79",
    "color": "Mauv",
    "createdAt": "2021-11-10T00:00:00.000Z",
    "updatedAt": "2021-11-28T00:00:00.000Z"
  },
  {
    "id": 5,
    "model": "Relay",
    "year": "2006",
    "brand": "Saturn",
    "price": "24023.91",
    "color": "Maroon",
    "createdAt": "2021-12-08T00:00:00.000Z",
    "updatedAt": "2021-12-09T00:00:00.000Z"
  },
  {
```

Definición de documentación de la aplicación



Anexos

Conexión a base de datos Heroku

Datos de conexión a la base de datos de heroku:

Host: ec2-44-209-186-51.compute-1.amazonaws.com
Database: dbp7j7qktqs438
User: wvdrdomdthklqb
Port: 5432
Password:885820e889dba3db8abc229924f089833b6577c9c27699ba1bb392fa8e4223de
URI:postgres://wvdrdomdthklqb:885820e889dba3db8abc229924f089833b6577c9c27699ba1bb392fa8e4223de@ec2-44-209-186-51.compute-1.amazonaws.com:5432/dbp7j7qktqs438
Heroku CLI
heroku pg:psql postgresql-encircled-74502 --app bedu-cars-db

Tablero TRELLO

https://trello.com/invite/b/QxZxFLRV/807fead925816ae7d4ba303e54b90464/js-project-backend-4