Fundación Universitaria Konrad Lorenz.
Desarrollo De Nuevas Tecnologías. API-Zoológico.
Estudiante: David Chaves Cod: 506222728.

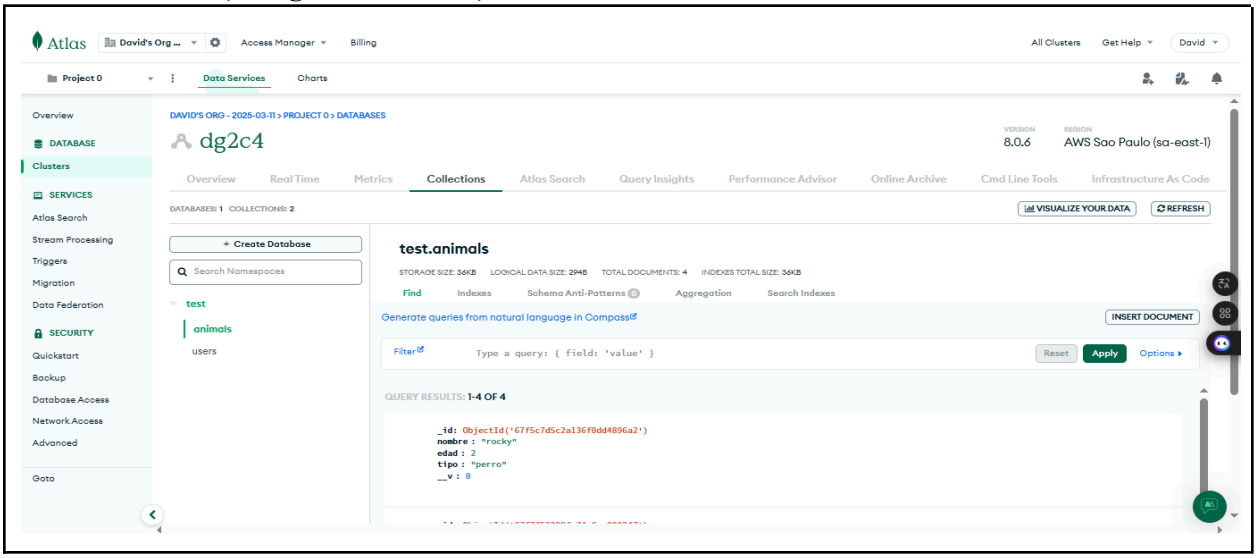
Documentación API-Zoológico

Link Repositorito GitHub: https://github.com/dg2c4/Zoologico-API

Tema abordado:

La creación de una API para un zoológico implica el desarrollo de un sistema backend que permite gestionar información sobre animales, especies y operaciones del zoológico. El sistema se construye utilizando Node.js como servidor, MongoDB como base de datos, y Express.js como framework para crear endpoints RESTful.

Modelo de Datos (MongoDB-Postman):



Estructura del Proyecto:

```
API-Zoológico/
└── node_modules/
└── src/
|   ├── models/
|   |   └── animal.js
|   ├── routes/
|   |   ├── authRoutes.js
|   |   ├── validate_token.js
|   |   └── animal.js
├── .env/
├── .gitignore/
├── README.md/
├── index.js/
├── package-lock.json/
└── package.json/
```

La arquitectura del sistema se organiza en capas:

Fundación Universitaria Konrad Lorenz.
Desarrollo De Nuevas Tecnologías. API-Zoológico.
Estudiante: David Chaves Cod: 506222728.

A. Capa de Rutas (Routes):

- animalRoutes.js: Maneja todas las operaciones CRUD para los animales.
- authRoutes.js: Gestiona la autenticación de usuarios.
- validate_token.js: Verifica los tokens de acceso.

B. Capa de Modelos (Models):

- animal.js: Define la estructura de datos para los animales en la base de datos.
- Especifica campos como nombre, especie, edad, ubicación, etc.

C. Capa de Datos:

- MongoDB Atlas para almacenamiento en la nube.
- Permite escalabilidad y alta disponibilidad.
- Facilita la gestión de datos sin estructura fija.

Implementación de la API (src→models→animal.js):

```
const mongoose = require("mongoose"); // importando el componente mogoose
const animalSchema = mongoose.Schema({
   nombre: {
      type: String,
      required: true,
   },
   edad: {
      type: Number,
      required: true,
   },
   tipo: {
      type: String,
      required: true,
   },
   fecha: {
      type: Date,
      requiered: true,
   }
});
module.exports = mongoose.model("Animal", animalSchema);
```

Implementación de la API (src→models→user.js):

```
const mongoose = require("mongoose"); // importando el componente mongoose
const bcrypt = require("bcrypt"); // importando el componente bcrypt
const userSchema = mongoose.Schema({
   usuario: {
      type: String,
      required: true
   },
   correo: {
      type: String,
      required: true
```

```
    },
    clave: {
        type: String,
        required: true
    }
});
userSchema.methods.encryptClave = async (clave) => {
    const salt = await bcrypt.genSalt(10);
    return bcrypt.hash(clave, salt);
}
module.exports = mongoose.model('User', userSchema);
```

Implementación de la API (src→routes→animal.js):

```
// Validar el Token de seguridad
const verifyToken = require('./validate_token');

const express = require("express");
const router = express.Router(); //manejador de rutas de express
const animalSchema = require("../models/animal");
//Nuevo animal
router.post("/animals", (req, res) => {
    const animal = animalSchema(req.body);
    animal
        .save()
        .then((data) => res.json(data))
        .catch((error) => res.json({ message: error }));
});


//Consultar todos los animales
router.get("/animals", verifyToken, (req, res) => {
    animalSchema
        .find()
        .then((data) => res.json(data))
        .catch((error) => res.json({ message: error }));
});

module.exports = router;
```

Implementación de la API (src→routes→authentication.js):

```
//Validación de inicio de sesión
const bcrypt = require("bcrypt");

//JWT Json Web Toke
const jwt = require("jsonwebtoken");


const express = require("express");
```

```
const router = express.Router(); //manejador de rutas de express
const userSchema = require("../models/user");


//Validación de datos de entrada
router.post('/signup', async (req, res) => {
    const { usuario, correo, clave } = req.body;
    const user = new userSchema({
        usuario: usuario,
        correo: correo,
        clave: clave
    });

    user.clave = await user.encryptClave(user.clave);
    await user.save(); //save es un método de mongoose para guardar datos en MongoDB //segundo parámetro: un texto que hace que el código generado sea único //tercer parámetro: tiempo de expiración (en segundos, 24 horas en segundos)
    //primer parámetro: payload - un dato que se agrega para generar el token
    const token = jwt.sign({ id: user._id }, process.env.SECRET, {
        expiresIn: 60 * 60 * 24, //un día en segundos
    });
    res.json({
        auth: true,
        token,
    });
});



//inicio de sesión
router.post("/login", async (req, res) => {
    // validaciones
    const { error } = userSchema.validate(req.body.correo, req.body.clave);
    if (error) return res.status(400).json({ error: error.details[0].message });
    //Buscando el usuario por su dirección de correo
    const user = await userSchema.findOne({ correo: req.body.correo });
    //validando si no se encuentra
    if (!user) return res.status(400).json({ error: "Usuario no encontrado" });
    //Transformando la contraseña a su valor original para
    //compararla con la clave que se ingresa en el inicio de sesión
    const validPassword = await bcrypt.compare(req.body.clave, user.clave);
    if (!validPassword)
        return res.status(400).json({ error: "Clave no válida" });
    res.json({
        error: null,
        data: "Bienvenido(a)",
    });
});

//Exportando el módulo para usarlo en otros archivos
```

```
module.exports = router;
```

Implementación de la API (src→routes→validate_token.js):

```
const jwt = require('jsonwebtoken')
//función para verificar que el token sea válido
//y si el usuario tiene permiso para acceder
//En el servidor se va a recibir así:
//access-token

const verifyToken = (req, res, next) => {
   const token = req.header('access-token')
   if (!token) return res.status(401).json({ error: '¡Lo sentimos!, pero no tiene permisos para acceder a
esta ruta.' })
   try {
      const verified = jwt.verify(token, process.env.SECRET)
      req.user = verified
      next() // Si en token es correcto, se puede continuar
   } catch (error) {
      res.status(400).json({ error: 'El token no es válido' })
   }
}
module.exports = verifyToken;
//Exportando el módulo para usarlo en otros archivos
//module.exports = verifyToken
```

Implementación de la API (src→index.js):

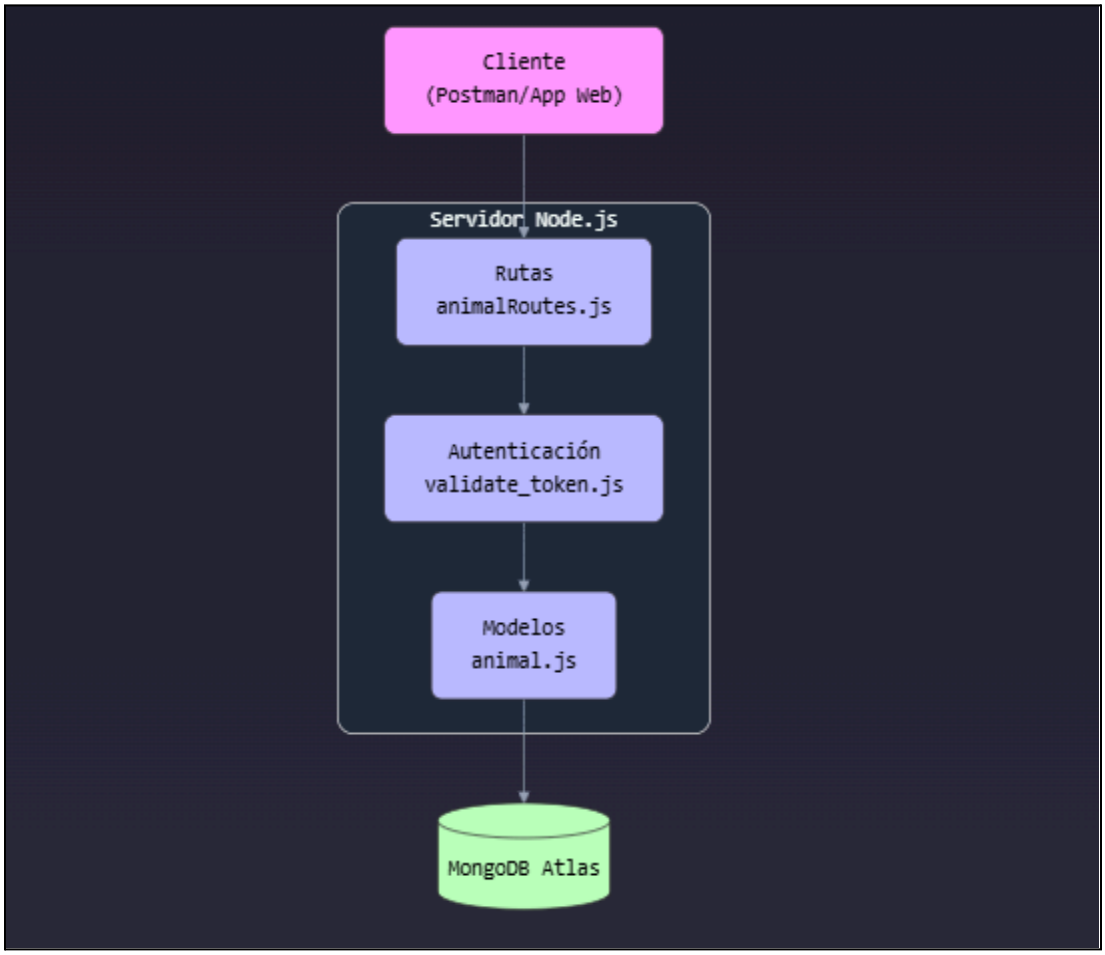
```
const parser = require("body-parser");
const express = require('express');
const app = express();
const port = 3000;
const animalRoutes = require("./routes/animal");
const authRoutes = require("./routes/authentication");
const mongoose = require("mongoose");
require('dotenv').config();
app.use(parser.urlencoded({ extended: false })); //permite leer los datos que vienen en la petición
app.use(parser.json()); // transforma los datos a formato JSON
//Gestión de las rutas usando el middleware
app.use("/api", animalRoutes);
app.use("/api", authRoutes);
//Conexión a la autenticación
app.use(express.json());
//Conexión a la base de datos
mongoose
   .connect(process.env.MONGODB_URI)
   .then(() => console.log("Conexión exitosa"))
   .catch((error) => console.log(error));
//Conexión al puerto
app.listen(port, () => {
```

```
  console.log(`Example app listening on port ${port}`)
});
```

Implementación de la API (package.json):

```
{
 "dependencies": {
  "bcrypt": "^5.1.1",
  "dotenv": "^16.4.7",
  "express": "^5.1.0",
  "jsonwebtoken": "^9.0.2",
  "mongoose": "^8.13.2",
  "nodemon": "^3.1.9"
 },
 "scripts": {
  "dev": "nodemon src/index.js"
 }
}
```

Diagrama de Flujo de la API:



Fundación Universitaria Konrad Lorenz.
Desarrollo De Nuevas Tecnologías. API-Zoológico.
Estudiante: David Chaves Cod: 506222728.

Los endpoints principales de la API incluyen:

- POST /api/animals: Crear nuevo animal.
- GET /api/animals: Listar todos los animales.
- GET /api/animals/:id: Obtener información de un animal específico.
- PATCH /api/animals/:id: Actualizar información de un animal.
- DELETE /api/animals/:id: Eliminar un animal del sistema.

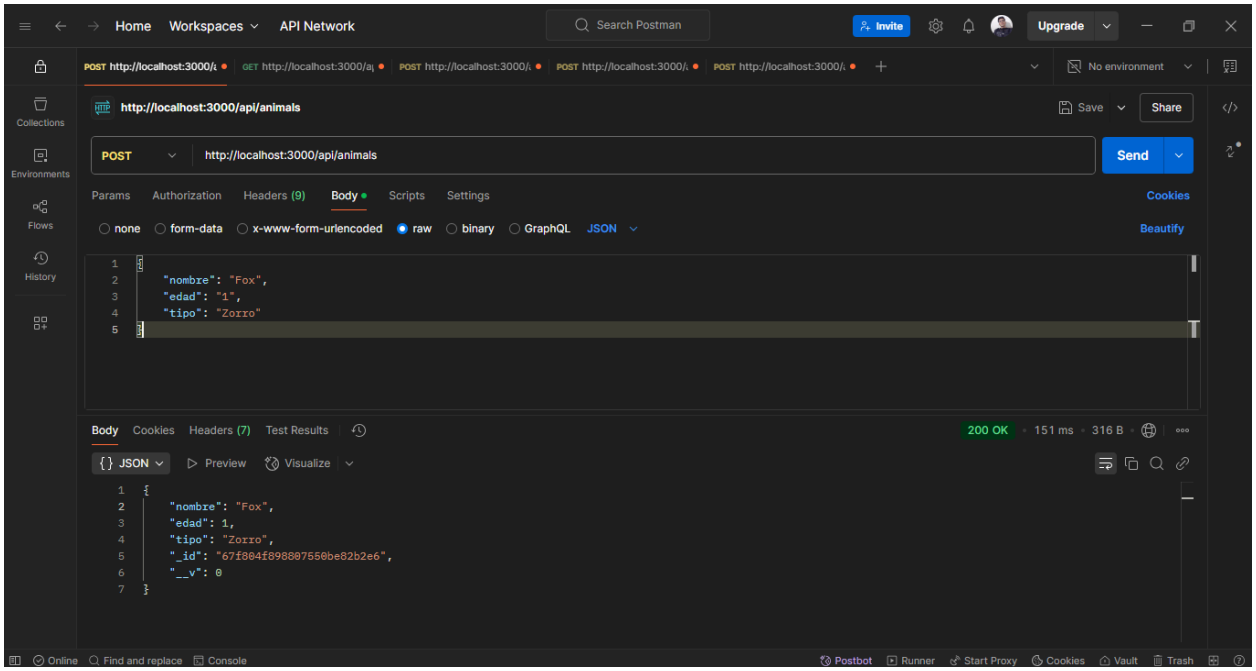
Tecnologías Utilizadas:

- Node.js: Entorno de ejecución para JavaScript
- Express: Framework para construir la API REST
- MongoDB: Base de datos NoSQL para almacenar la información
- Mongoose: ODM para interactuar con MongoDB
- JWT (JSON Web Tokens): Para autenticación stateless
- bcrypt: Para hashing de contraseñas
- Postman/Thunder Client: Para probar los endpoints
- dotenv: Para manejar variables de entorno

Consideraciones de Seguridad

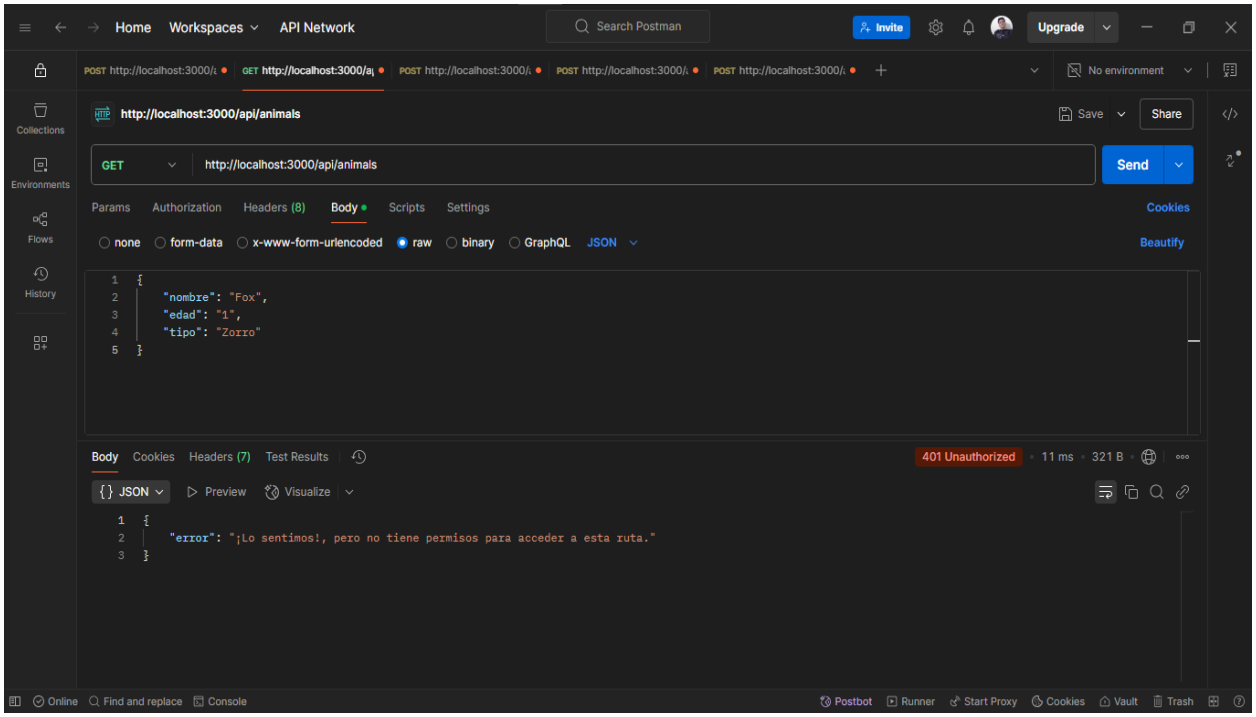
- bcrypt: Se utiliza para hackear las contraseñas antes de almacenarlas en la base de datos.
- JWT: Proporciona autenticación stateless con expiración de tokens.
- Middleware de autorización: Restringe el acceso a ciertas rutas basado en roles.
- Validación de datos: Mongoose valida los datos antes de guardarlos.
- Variables de entorno: Datos sensibles como JWT SECRET y MONGODB URI se almacenan en .env.

Evidencias de Operaciones realizadas (POST):

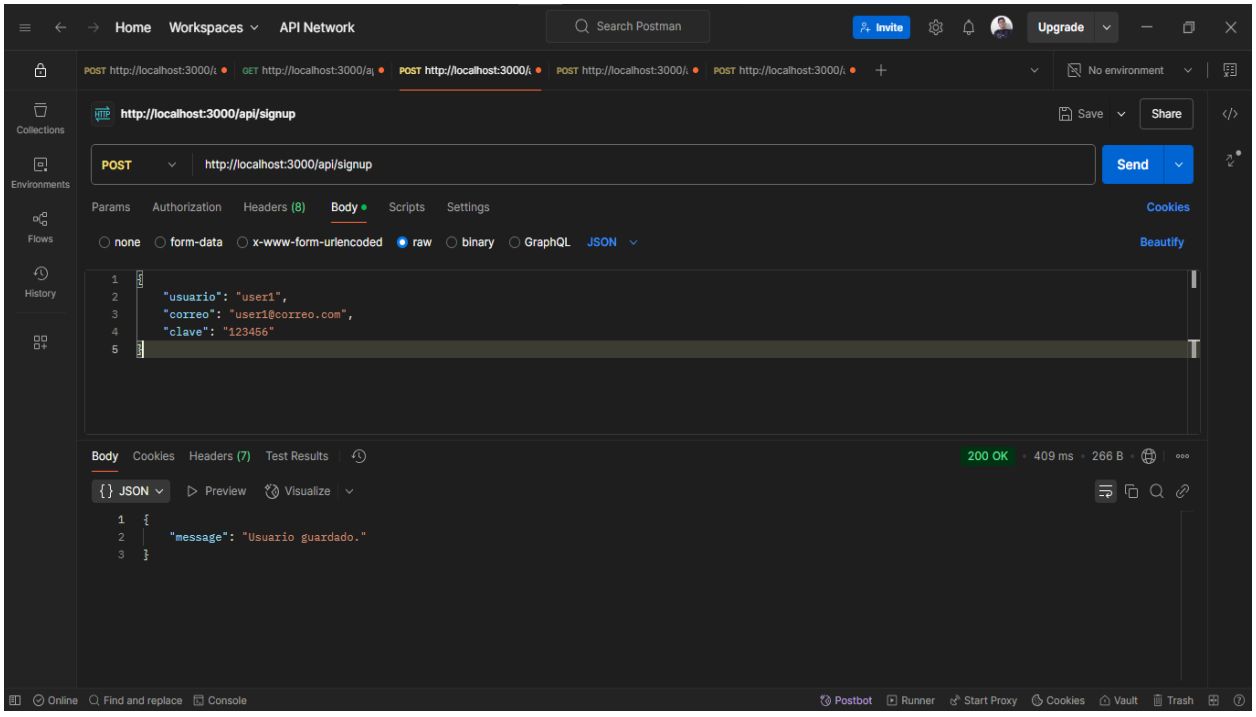


Fundación Universitaria Konrad Lorenz.
Desarrollo De Nuevas Tecnologías. API-Zoológico.
Estudiante: David Chaves Cod: 506222728.

Evidencias de Operaciones realizadas (GET):

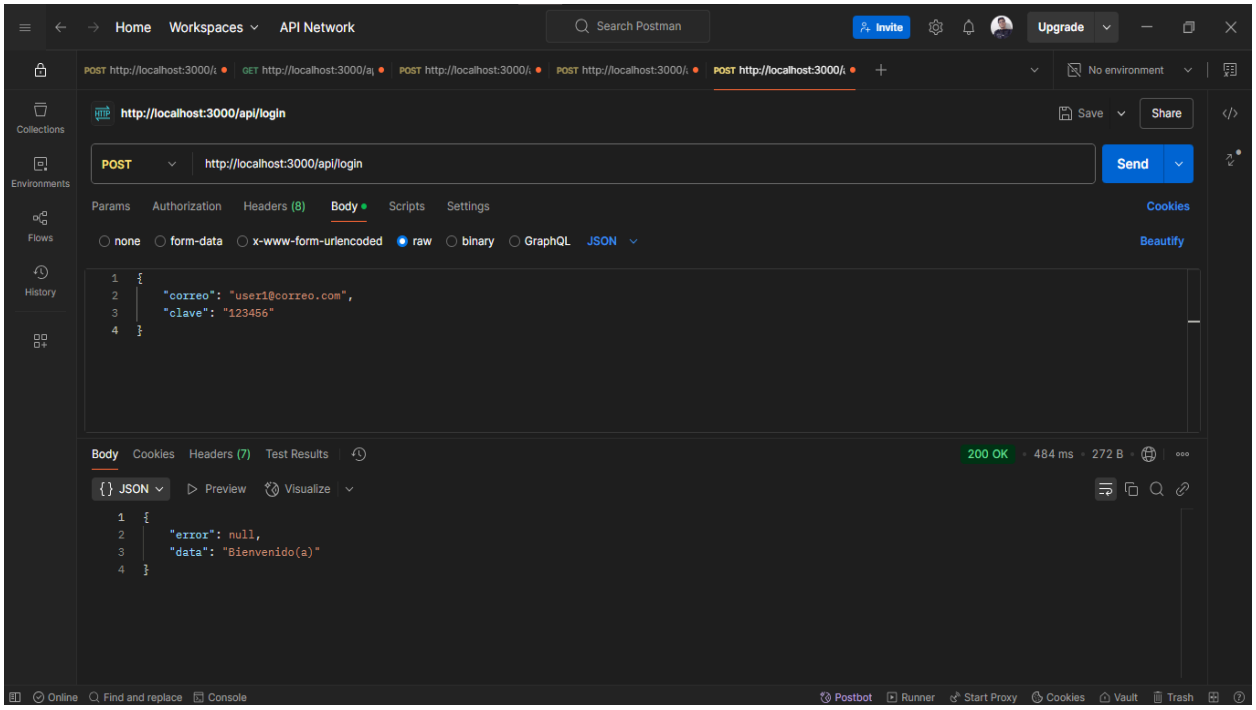


Evidencias de Operaciones realizadas (POST-Signup):



Fundación Universitaria Konrad Lorenz.
Desarrollo De Nuevas Tecnologías. API-Zoológico.
Estudiante: David Chaves Cod: 506222728.

Evidencias de Operaciones realizadas (POST-Login):



Conclusiones:

- Este proyecto se centra en el desarrollo de una API RESTful utilizando Node.js y MongoDB, implementando una arquitectura de software que permite la gestión completa de recursos mediante operaciones CRUD (Crear, Leer, Actualizar y Eliminar).
- La estructura del proyecto sigue un patrón modular organizado en carpetas específicas para modelos de datos, rutas de API y middleware de autenticación, facilitando el mantenimiento y escalabilidad del sistema.
- El backend implementa una conexión robusta con MongoDB Atlas para el almacenamiento persistente de datos, mientras que Node.js proporciona el entorno de ejecución para el servidor, permitiendo la creación de endpoints RESTful que procesan solicitudes HTTP y responden en formato JSON.