

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

COLSEARCH

ENTIDADES – TABLAS – MODELOS DE DATOS



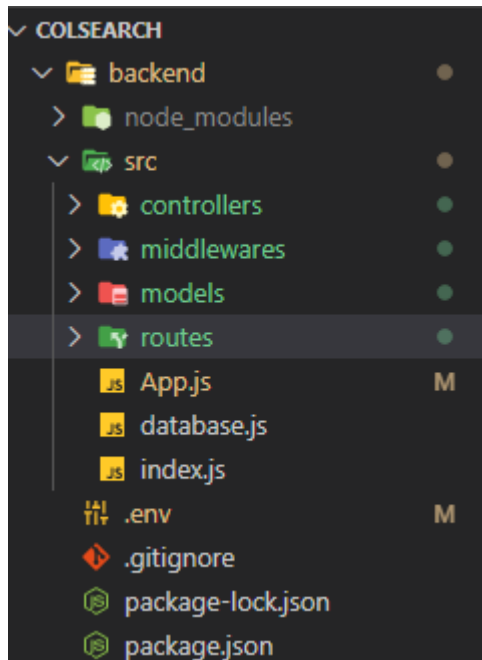
EL MODELO DE DATOS ESTA PENSADO PARA UNA BASE DE DATOS NO RELACIONAL
EN NUESTRO CASO MONGODB.

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

CRUD

Create, Read, Update, Delete

Nuestro proyecto esta construido casi en su totalidad usando Javascript, en el caso del Backend esta construido usando Node.js y su framework Express, con el módulo Mongoose para manipular MongoDB.



Tenemos esta estructura de Carpetas en nuestro Backend, nuestro backend esta construido en torno a una arquitectura REST.

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

- **Controllers:** Es donde van los todos los controladores con sus respectivos métodos que interactúan con la base de datos.
- **Middlewares:** Funcionalidades que nos sirven como un mediador entre nuestro backend – el Frontend y la base de datos respectivamente para que nuestro backend funcione correctamente.
- **Modelos:** Donde están definidos nuestros modelos de datos para el entendimiento de la base de datos y el backend.
- **Routes:** Donde están definidas las rutas en donde se realizarán las respectivas interacciones HTTP entre el frontend y el backend.

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Routes

```
//Middlewares
app.use(cors());
app.use(express.json());

//Routes
app.use("/api/users", require("../routes/users.routes"));
app.use("/api/missing", require("../routes/missingpersons.routes"));
```

```
users.controller.js  missingpersons.routes.js X
backend > src > routes > missingpersons.routes.js > ...
1  const { Router } = require("express");
2  const router = Router();
3  const {
4    createMissingPerson,
5    getMissingPersons,
6    getMissingPerson,
7    updateMissingPerson,
8    deleteMissingPerson,
9  } = require("../controllers/missingpersons.controller");
10
11  router.route("/").get(getMissingPersons).post(createMissingPerson);
12
13  router
14    .route("/:id")
15    .get(getMissingPerson)
16    .put(updateMissingPerson)
17    .delete(deleteMissingPerson);
18
19  module.exports = router;
20
```

```
users.controller.js  users.routes.js X
backend > src > routes > users.routes.js > ...
You, a few seconds ago | 1 author (You)
1  const { Router } = require("express");
2  const router = Router();
3  const {
4    getUsers,
5    getUserById,
6    createUser,
7    updateUser,
8    deleteUser,
9    loginUser,
10   validToken,
11 } = require("../controllers/users.controller");
12
13 const auth = require("../middlewares/auth");
14 router.route("/").get(getUsers).post(createUser);
15 router.route("/login").post(loginUser);
16 router.route("/validToken").post(validToken);
17 router.route("/:id").get(getUserById).put(updateUser).delete(auth, deleteUser);
18
19 module.exports = router;
20
```

/api/users : Acciones referentes a los usuarios

- **/** – Crear Usuario – Obtener Usuarios
 - GET, POST
- **/login** – Efectuar login
 - POST
- **/validToken** – Token Válido
 - POST
- **/id** – Editar, Obtener Usuario, Eliminar Usuario
 - GET, PUT, DELETE

/api/missing: Acciones referentes a las personas desaparecidas

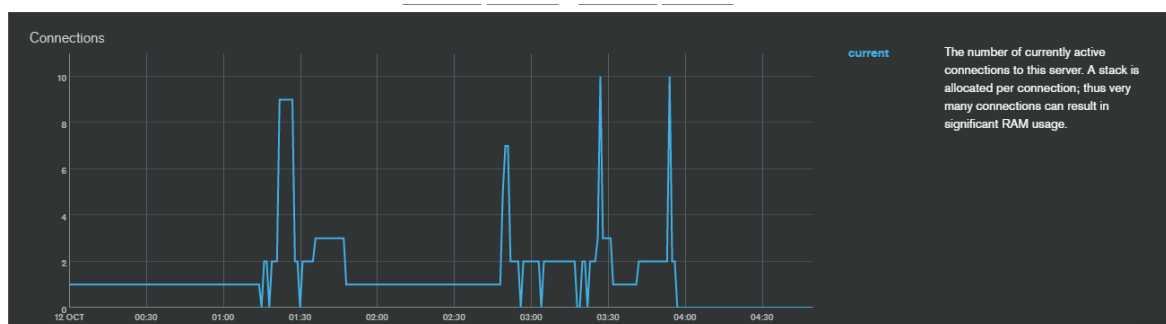
- **/** – Crear Registro – Obtener Registros
 - GET, POST
- **/id** – Editar, Obtener Registro, Eliminar Registro
 - GET, PUT, DELETE

Archivos

Database.js – Donde está alojado el script que permite la conexión con entre Node.js y MongoDB.

```
You, 2 days ago | 1 author (You)
1  const mongoose = require("mongoose");
2  const URI = process.env.MONGODB_URI;
3
4  async function database() {
5    try {
6      mongoose.connect(URI, {
7        useNewUrlParser: true,
8        useCreateIndex: true,
9        useUnifiedTopology: true,
10       useFindAndModify: false,
11     });
12   } catch (error) {
13     console.log(error.message);
14   }
15 }
16
17 database();
18
```

Aquí se visualizan las conexiones que ha tenido el backend con la base de datos



Modelos

Usuarios

```
You, a few seconds ago | 1 author (You)
1  const { Schema, model } = require("mongoose");
2
3  const UserSchema = new Schema(
4    {
5      username: {
6        type: String,
7        required: true,
8        trim: true,
9        unique: true,
10     },
11     password: {
12       type: String,
13       required: true,
14     },
15     name: String,
16   },
17   { timestamps: true }
18 );
19
20 module.exports = model("User", UserSchema);
21
```

Annotations in the image:

- Schema (points to `Schema` in line 3)
- Username (points to `username` in line 5)
- Tipo de dato (points to `String` in line 6)
- requerido (points to `required: true` in line 7)
- auto correcion (points to `trim: true` in line 8)
- único (points to `unique: true` in line 9)
- Password (points to `password` in line 11)
- Name, nombre del usuario (points to `name: String` in line 15)
- Timestamps (points to `{ timestamps: true }` in line 17)
- Modelar el Usuario en base al esquema (points to `model("User", UserSchema)` in line 20)

Debemos definir un Schema usando el módulo Mongoose, este esquema es un Objeto tipo JSON que nos permite definir los atributos que tendrán nuestros modelos.

Ejemplo:

```
Username : {
  Tipo : String,
  Requerido: True,
  Único : True
}
```

Este modelo ayuda a la base de datos a saber que tipo de dato esperar, si es único y si es requerido

Personas Desaparecidas

```
1  const { Schema, model } = require("mongoose");  Módulo mongodb
2
3  const missingPersonSchema = new Schema(  Esquema
4    {
5      name: {
6        type: String,
7        required: true,  Nombre
8        trim: true,
9      },
10     surname: {
11       type: String,
12       required: true,  Apellido
13       trim: true,
14     },
15     identificationCard: {  IdentificationCard
16       type: String,
17     },
18     approved: Boolean,  Aprobado
19     found: Boolean,  Encontrado
20     description: {  Descripción
21       age: Number,
22       height: Number,
23       skin: String,
24       hairColour: String,
25       complexion: String,
26       detailed: String,
27     },
28     imageUrl: String,  URL Imagen
29     neighborhood: String,  Barrio
30     contact: String,  Número de Contacto
31     email: String,  Correo
32     location: {  Ubicación
33       latitude: String,
34       longitude: String,
35     },
36   },
37   {  Timestamps
38     timestamps: true,
39   }
40 );
41
42 module.exports = model("MissingPerson", missingPersonSchema);
43
```

Este modelo es más largo pero es más de lo mismo.

nombreAtributo : tipoDeDato

nombreAtributo : {Opciones}

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

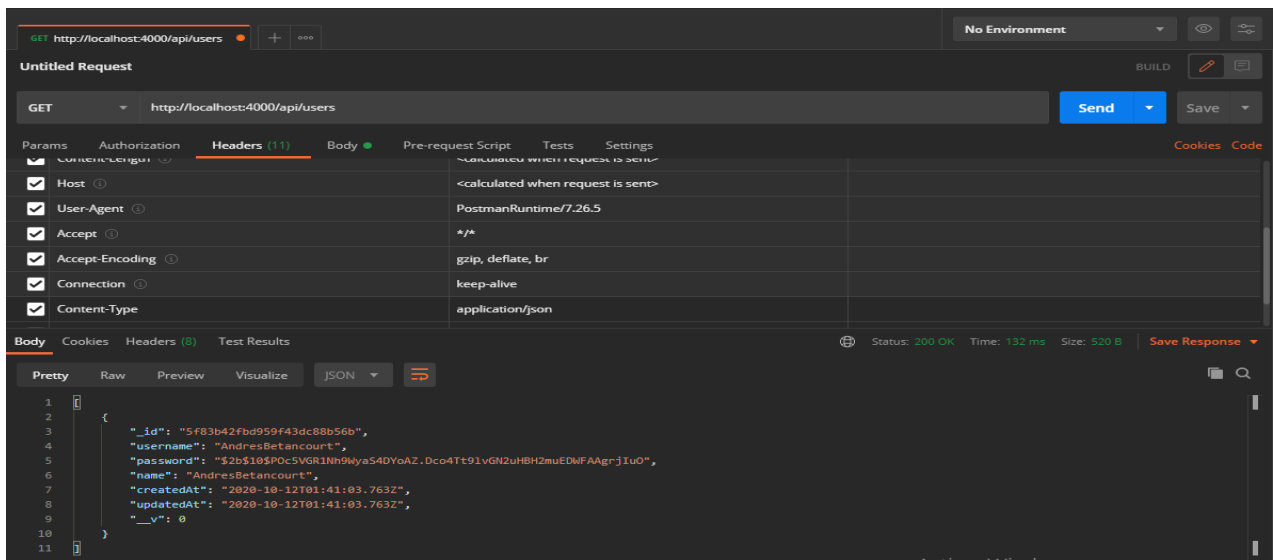
CRUD – Usuarios

Get Users – GET

```
users.controller.js X
backend > src > controllers > users.controller.js > Users.deleteUser > message

You, a few seconds ago | 1 author (You)
1 const User = require("../models/User");           Model - User
2 const jwt = require("jsonwebtoken");               Módulo Authentication
3 const bcrypt = require("bcrypt");                  Password Encrypt
4 const Users = {};
5
6 Users.getUsers = async (req, res) => {
7   try {
8     const users = await User.find();               Obtener todos los registros
9     res.json(users);                               Devolver la respuesta de la colección Users.
10  } catch (error) {
11    res.status(500);
12    res.json({ message: error.message });           Error
13  }
14 };
15
```

EX:



Se hizo una petición GET hacia la ruta definida para obtener todos los registros de usuarios en la Colección Users.

Get User by ID – GET

```
Users.getUserById = async (req, res) => {  
  try {  
    const user = await User.findById(req.params.id); ← Buscar Usuario por ID  
    if (user) {  
      res.json(user); Si existe, lo retorna como respuesta de servidor  
    } else {  
      res.json({ message: "Usuario no Encontrado" }); De lo contrario usuario no encontrado  
    }  
  } catch (error) {  
    res.status(500);  
    res.json({ message: error.message }); Error  
  }  
};
```

EX:

The screenshot shows a Postman interface for a GET request to `http://localhost:4000/api/users/5f83b42fbd959f43dc88b56b`. The request headers are set to `Host: localhost:4000`, `User-Agent: PostmanRuntime/7.26.5`, `Accept: */*`, `Accept-Encoding: gzip, deflate, br`, `Connection: keep-alive`, and `Content-Type: application/json`. The response status is `200 OK` with a time of `95 ms` and a size of `518 B`. The response body is a JSON object:

```
{  
  "_id": "5f83b42fbd959f43dc88b56b",  
  "username": "AndresBetancourt",  
  "password": "$2b$10$POc5VGR1Nh9Mya54DYoaZ.Dco4Tt91vGN2uHBH2muEDWFAAgrjIu0",  
  "name": "AndresBetancourt",  
  "createdAt": "2020-10-12T01:41:03.763Z",  
  "updatedAt": "2020-10-12T01:41:03.763Z",  
  "__v": 0  
}
```

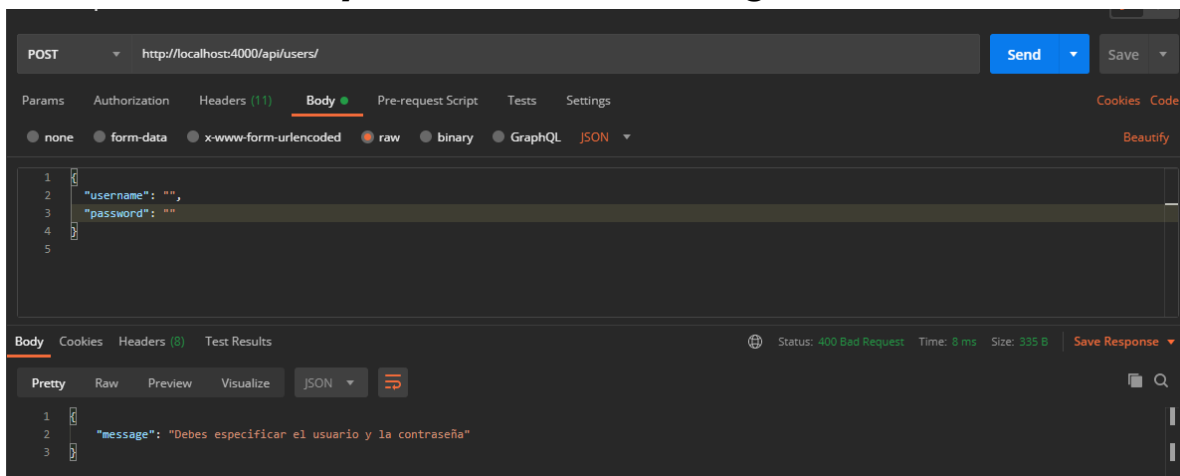
Se obtuvo un usuario por su id.

Create User – POST

```
16 Users.createUser = async (req, res) => {  
17   try {  
18     let { username, password, name } = req.body; ← Obtener valores del Body de la petición HTTP  
19   }  
20   if (!username || !password) { ← Si no hay usuario o contraseña, retorna error  
21     res.status(400);  
22     res.json({ message: "Debes especificar el usuario y la contraseña" });  
23   }  
24   if (password.length < 6) { ← Si la contraseña es muy corta, retorna error  
25     res.status(400);  
26     res.json({ message: "La contraseña es muy corta" });  
27   }  
28   const userExists = await User.findOne({ username: username }); ← Si usuario existe no deja crearlo  
29   if (userExists) {  
30     res.status(400);  
31     res.json({ message: "Este usuario ya existe" });  
32   }  
33   if (!name) { ← Si no se ingreso el nombre se asigna el nombre de usuario como nombre  
34     name = username;  
35   }  
36   const passwordHash = await Users.hashPassword(password); ← Se encripta la contraseña  
37   const newUser = new User({  
38     username,  
39     password: passwordHash, ← Se crea el Usuario, Objeto  
40     name, ← Se guarda, funcion asincrona  
41   });  
42   await newUser.save(); ← Se envia la respuesta del servidor  
43   res.json({ message: "Usuario Creado", newUser });  
44 } catch (error) {  
45   res.status(500);  
46   res.json({ message: error.message }); ← Error  
47 }  
48 };
```

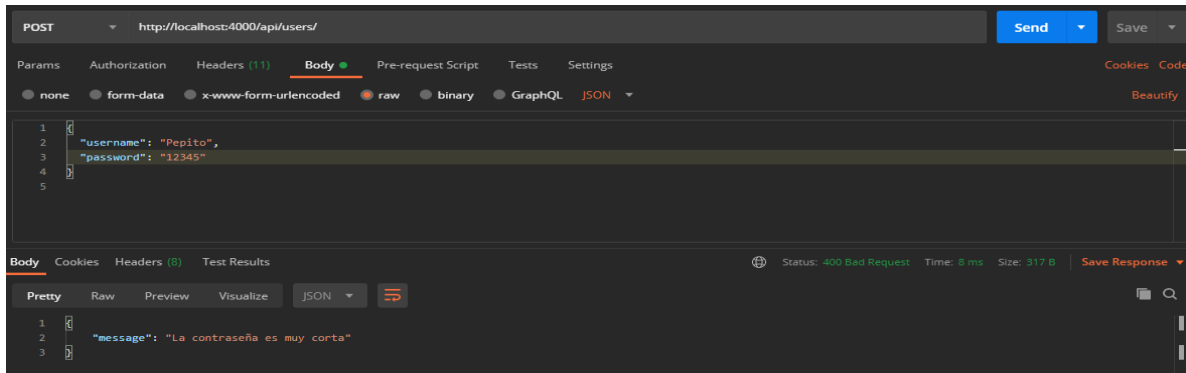
EX:

Error 1: Usuario y contraseña no ingresados

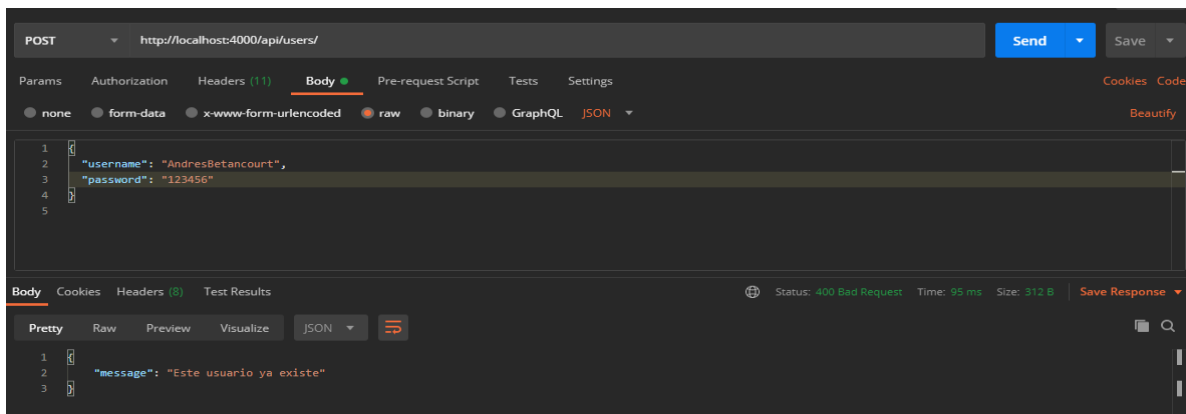


Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Error 2: Contraseña menor a 6 caracteres



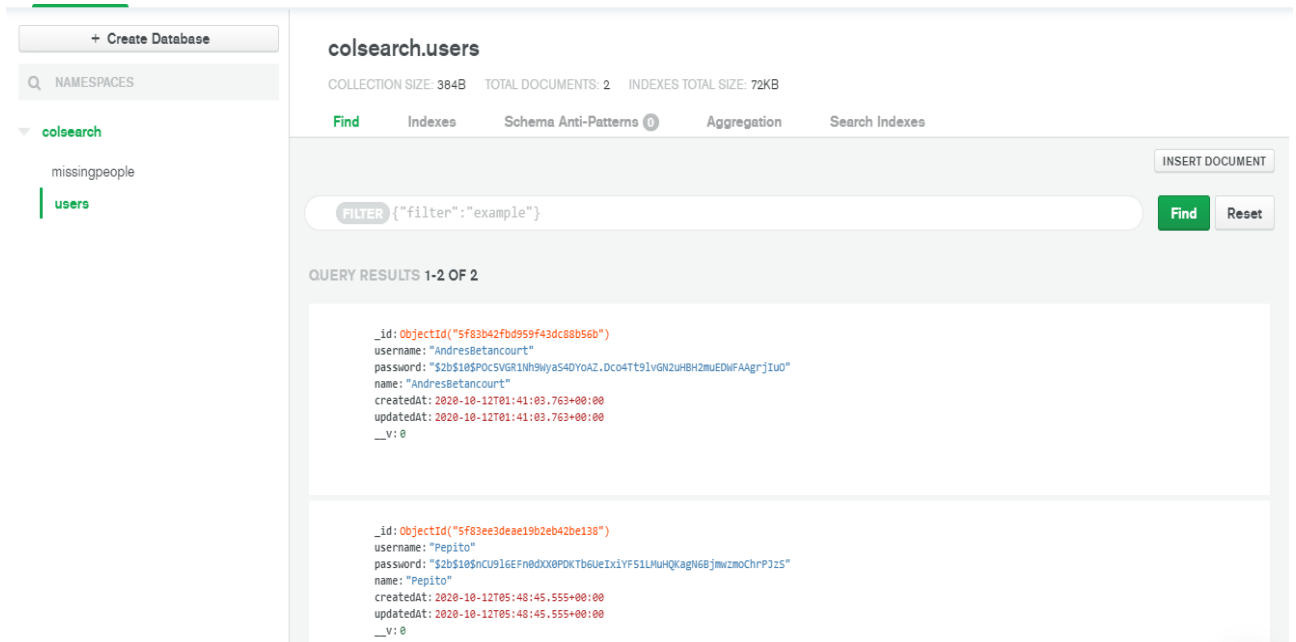
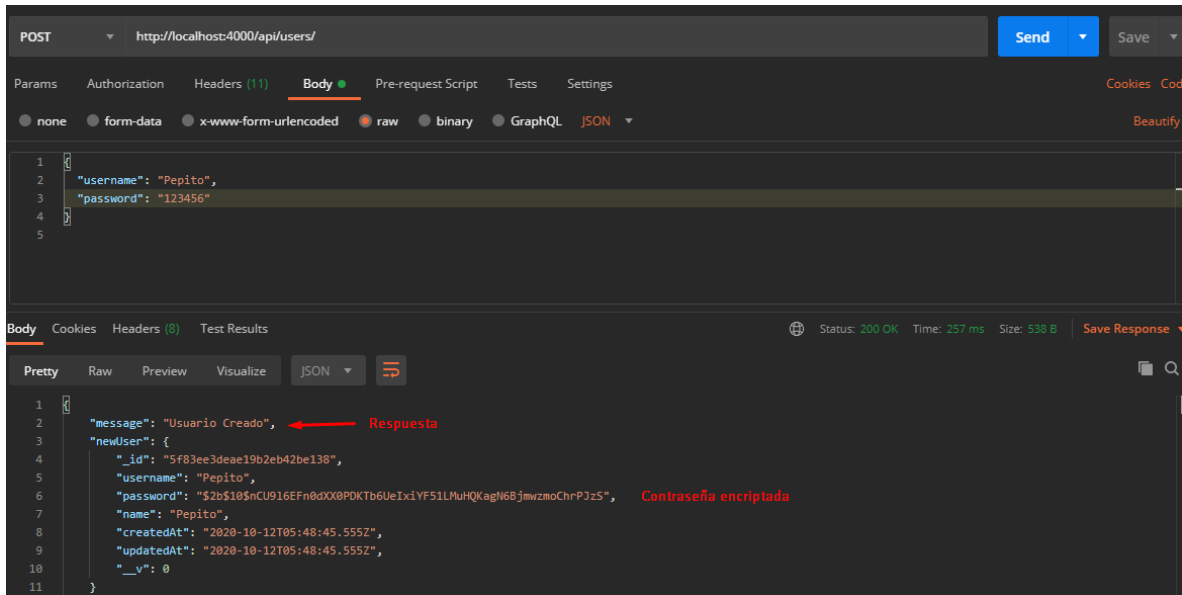
Error 3 : Usuario ya existe



Se hizo una petición POST hacia la ruta de la api de los usuarios y se hizo prueba de las validaciones en los datos.

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Usuario creado satisfactoriamente



Usuarios guardados en la base de datos. Mongo Atlas

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Update User – PUT

```
Users.updateUser = async (req, res) => {  
  try {  
    let { username, password, name } = req.body;   
    if (!username || !password) {  
      res  
        .status(400)  
        .json({ message: "Debes especificar usuario y contraseña" });  
    }  
    if (password.length < 6) {  
      res.status(400).json({ message: "La contraseña es muy corta" });  
    }  
    if (!name) {  
      name = username;  
    }  
  
    const passwordHash = await Users.hashPassword(password);  
  
    await User.findByIdAndUpdate(req.params.id, {  
      username,  
      password: passwordHash,  
      name,  
    });  
    res.json({ message: "Usuario Actualizado" });  
  } catch (error) {  
    res.status(500);  
    res.json({ message: error.message });  
  }  
};
```

Obtener datos del body de la petición

No se especifico usuario y/o contraseña

Contraseña corta

encriptar contraseña

Buscar por id y actualizar

Respuesta del servidor

Error

PUT http://localhost:4000/api/users/5f83ee3deae19b2eb42be138

Send Save

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {  
2   "username": "PepitoPerez",  
3   "password": "123456"  
4 }  
5
```

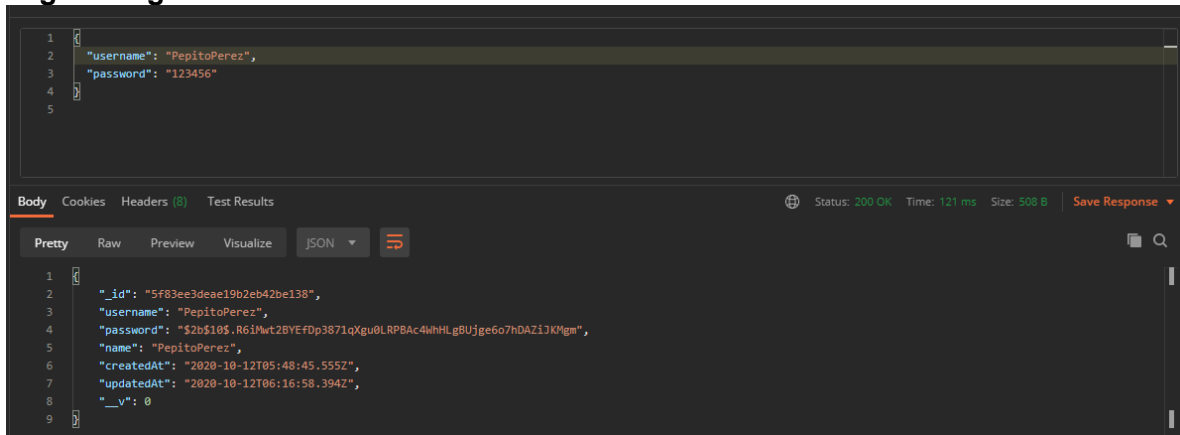
Body Cookies Headers (8) Test Results

Status: 200 OK Time: 159 ms Size: 300 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Usuario Actualizado"  
3 }
```

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán



The screenshot shows a REST client interface. The top section displays the request body as a JSON object: `{ "username": "PepitoPerez", "password": "123456" }`. The bottom section shows the response body in 'Pretty' format, which is a JSON object containing user details: `{ "_id": "5f83ee3deae19b2eb42be138", "username": "PepitoPerez", "password": "$2b$10$.R6iMwt2BYEfDp3871qXgu0LRPBac4WhHLgBUjge6o7hDAZi3KMgm", "name": "PepitoPerez", "createdAt": "2020-10-12T05:48:45.555Z", "updatedAt": "2020-10-12T06:16:58.394Z", "__v": 0 }`. The status bar indicates a 200 OK response.

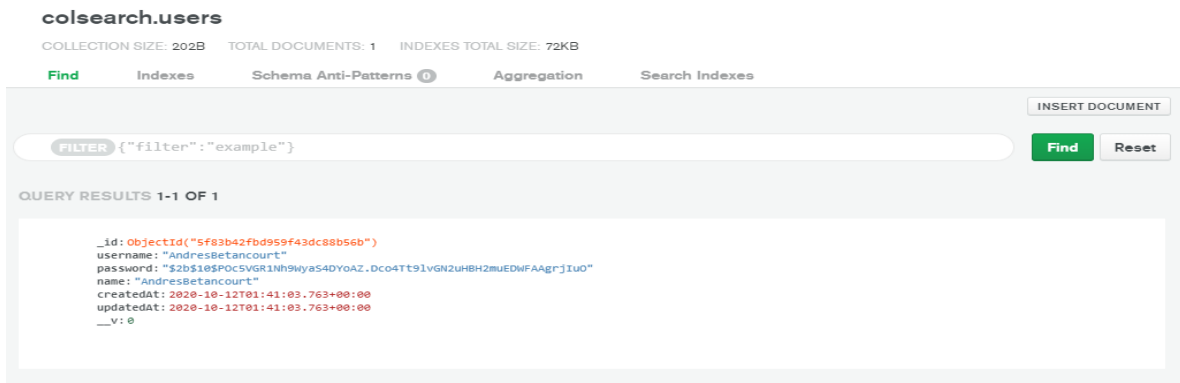
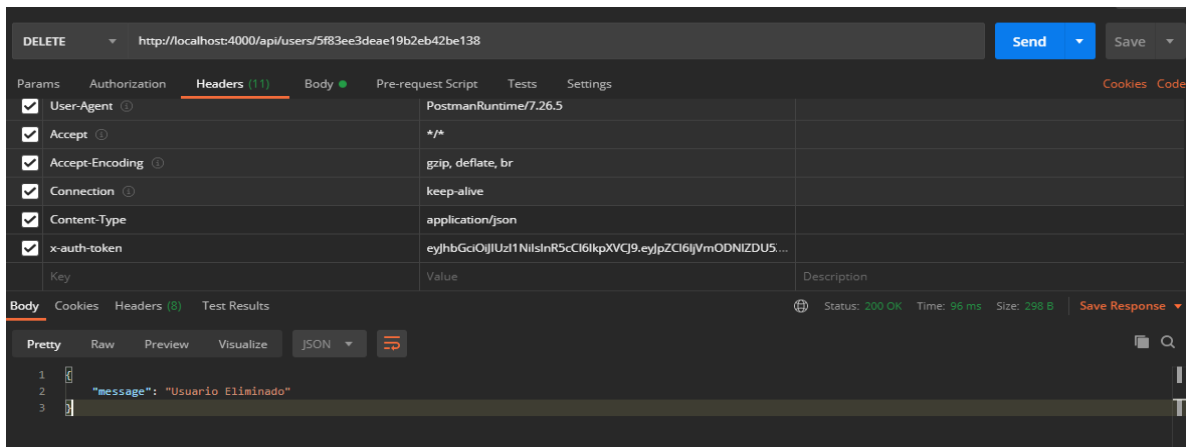
```
_id: ObjectId("5f83ee3deae19b2eb42be138")
username: "PepitoPerez"
password: "$2b$10$.R6iMwt2BYEfDp3871qXgu0LRPBac4WhHLgBUjge6o7hDAZi3KMgm"
name: "PepitoPerez"
createdAt: 2020-10-12T05:48:45.555+00:00
updatedAt: 2020-10-12T06:16:58.394+00:00
__v: 0
```

**Se cambio el nombre de usuario a
PepitoPerez**

**Usuario actualizado mediante el método
PUT HTTP.**

Delete User – DELETE

```
Users.deleteUser = async (req, res) => {  
  try {  
    await User.findByIdAndDelete(req.params.id); ← Buscar por id y eliminar  
    res.json({ message: "Usuario Eliminado" }); ← Respuesta servidor  
  } catch (error) {  
    res.status(500);  
    res.json({ message: error.message }); ← Error  
  }  
};
```



Se elimino a PepitoPerez

Este va más al grano es simplemente una petición DELETE al id
especifico del usuario

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Personas Desaparecidas

Create Register - POST

```
const MissingPerson = require("../models/MissingPerson");  
const MissingPersons = {};  
  
MissingPersons.createMissingPerson = async (req, res) => {  
  try {  
    const {  
      name,  
      surname,  
      identificationCard,  
      approved,  
      found,  
      description,  
      imageUrl,  
      neighborhood,  
      email,  
      location,  
    } = req.body;  
  
    if (!name || !surname || !identificationCard) {  
      res.status(400).json({  
        message:  
          "Debes ingresar el nombre, apellido y al menos un identificador para poder localizar a esta persona",  
      });  
    }  
  
    const newMissingPerson = new MissingPerson({  
      name,  
      surname,  
      identificationCard,  
      approved,  
      found,  
      description,  
      imageUrl,  
      neighborhood,  
      email,  
      location,  
    });  
  
    await newMissingPerson.save();  
    res.json({ message: "Registro agregado" });  
  } catch (error) {  
    res.status(500).json({ message: error.message });  
  }  
};
```

Importar Modelo

Obtener del Body

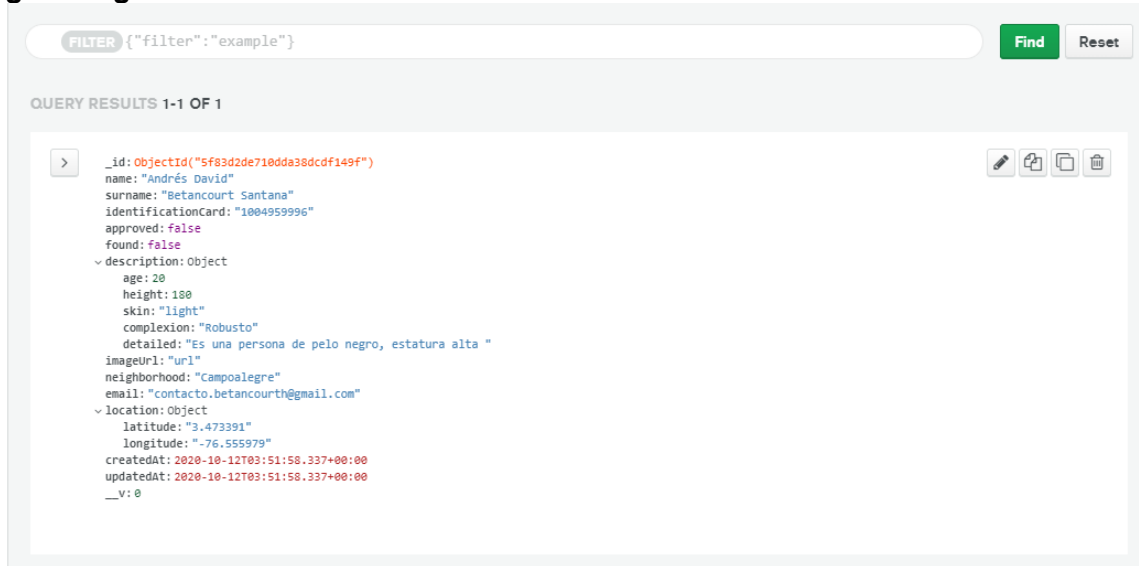
Validar datos

Crear registro y guardar en base de datos

Respuesta servidor

Error

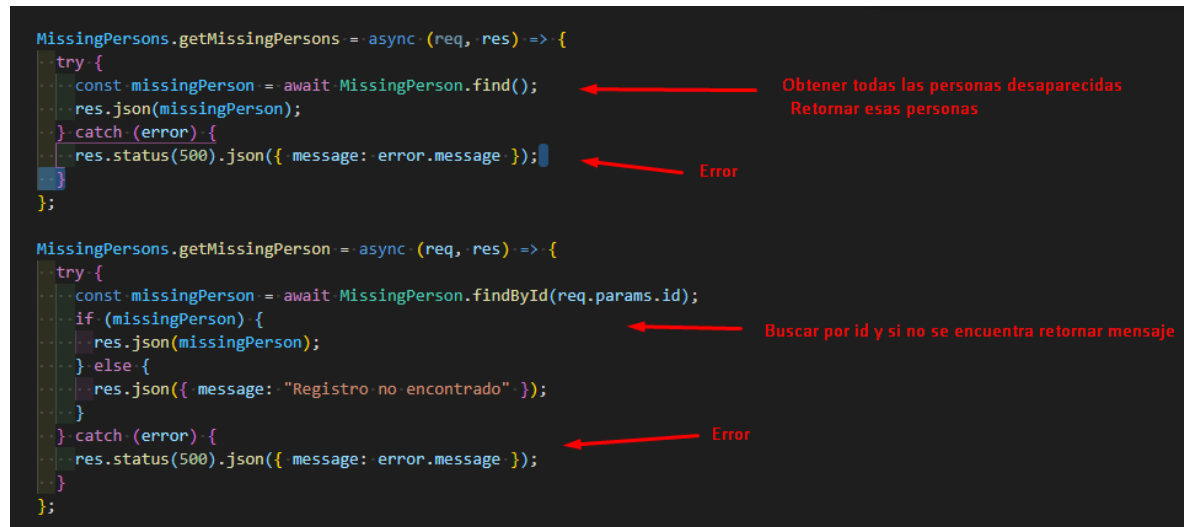
Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán



**Se creo un registro de persona
desaparecida en la base de datos**

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Get Register – Get Register by ID – GET



```
MissingPersons.getMissingPersons = async (req, res) => {
  try {
    const missingPerson = await MissingPerson.find();
    res.json(missingPerson);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

MissingPersons.getMissingPerson = async (req, res) => {
  try {
    const missingPerson = await MissingPerson.findById(req.params.id);
    if (missingPerson) {
      res.json(missingPerson);
    } else {
      res.json({ message: "Registro no encontrado" });
    }
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Obtener todas las personas desaparecidas
Retornar esas personas

Error

Buscar por id y si no se encuentra retornar mensaje

Error

Basicamente como está explicado en las imágenes utilizan el método HTTP GET para obtener los registros de la base de datos.

Update Register – PUT

```
68 MissingPersons.updateMissingPerson = async (req, res) => {  
69   try {  
70     const {  
71       name,  
72       surname,  
73       identificationCard,  
74       approved,  
75       found,  
76       description,  
77       imageUrl,  
78       neighborhood,  
79       email,  
80       location,  
81     } = req.body;  
82  
83     await MissingPerson.findByIdAndUpdate(req.params.id, {  
84       name,  
85       surname,  
86       identificationCard,  
87       approved,  
88       found,  
89       description,  
90       imageUrl,  
91       neighborhood,  
92       email,  
93       location,  
94     });  
95     res.json({ message: "Registro Actualizado" });  
96   } catch (error) {  
97     res.status(500).json({ message: error.message });  
98   }  
99 }  
100
```

Información que llega del body

Actualizar información a Objeto que encuentre por id

Se abstraen los datos del body de la petición HTTP y se actualizan.

Andrés David Betancourt Santana
Kevin Ivan Ulloa Nepas
Miguel Angel Serrate Timarán

Delete Register – DELETE

```
MissingPersons.deleteMissingPerson = async (req, res) => {  
  try {  
    await MissingPerson.findByIdAndDelete(req.params.id);  
    res.json({ message: "Registro Eliminado" });  
  } catch (error) {  
    res.status(500).json({ message: error.message });  
  }  
};
```

Se elimina al registro del id que entre por parámetro en la ruta