

DOCUMENTO PRUEBAS UNITARIAS

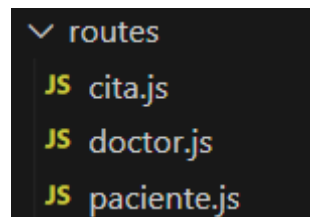
SE REALIZARON EN POSTMAN

Se utiliza `app.use()` para definir los prefijos de las rutas de los endpoints y asociarlos a los controladores correspondientes. En este caso, las rutas de los endpoints están definidas en archivos separados (`paciente.js`, `doctor.js` y `cita.js`) y se importan y se asocian en el archivo principal de la aplicación. Se tienen 3 diferentes una por cada componente grande: paciente, doctor y cita

El manejo de que la información se envíe bien al Back end, se hace desde el formulario del front end mediante Validator, aplica para las diversos endpoints.

```
app.use('/api/paciente', require('./routes/paciente'));
app.use('/api/doctor', require('./routes/doctor'));
app.use('/api/cita', require('./routes/cita'));
```

También se utiliza la opción para definir directamente las rutas de los endpoints en el archivo `cita.js`, `doctor.js` y `paciente.js`, en las rutas. utilizando `router.post()`, `router.get()` y `router.delete()` para especificar las rutas y los controladores asociados.



- **Rutas de paciente.js**
(Esquema)

```
const PacienteSchema = mongoose.Schema({
  cedula: {
    type: String,
    required: true,
    unique: true
  },
  nombre: {
    type: String,
    required: true
  },
  apellido: {
    type: String,
    required: true
  },
  fechaNacimiento: {
    type: Date,
    required: true
  },
  edad: {
    type: Number,
    required: true
  },
  fechaCreacion: {
    type: Date,
    default: Date.now()
  }
});
```

- `router.post('/', pacienteController.crearPaciente);`

Se utiliza el método POST para la creación de un nuevo paciente y se maneja desde el controlador del paciente mediante una función llamada `crearPaciente()`, cuyo código es

```
exports.crearPaciente = async (req, res) => {  
  
  try {  
    let paciente;  
  
    //creamos un paciente  
    paciente = new Paciente(req.body);  
  
    await paciente.save();  
    res.send(paciente);  
  
  } catch (error) {  
    console.log(error);  
    res.status(500).send('hubo un error al crear el paciente');  
  }  
}
```

El manejo de que la información se envíe bien al Back end, se hace desde el formulario del front end mediante Validator. Se espera que se envíe a la DB: una cedula (única), nombre, apellido, fecha de nacimiento en formato DATE y edad (la cual se calcula en el front por medio de la fecha de nacimiento y se envía).

Por lo cual se hará una prueba con los datos enviados de forma correcta, otra con cedula repetida y otra con formato incorrecto de fecha.

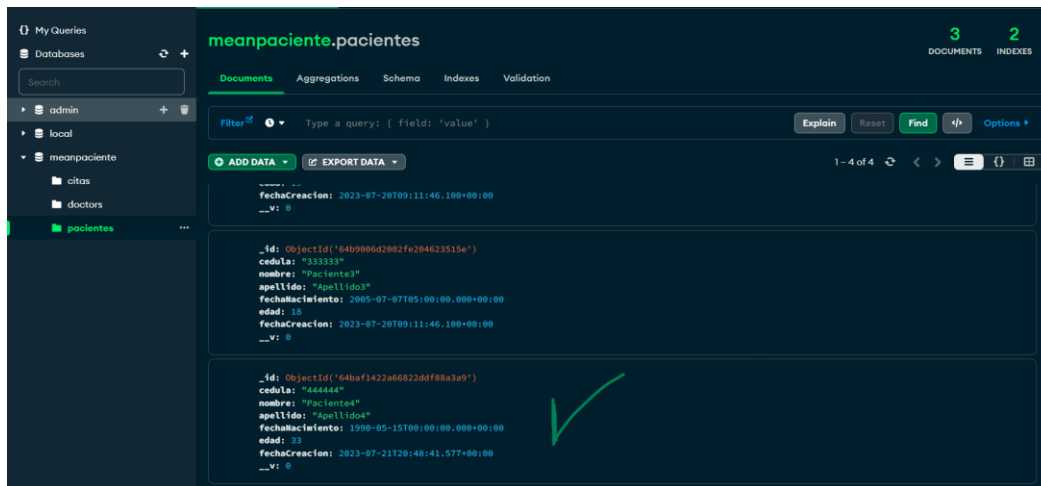
The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:4000/api/paciente
- Body Type:** JSON
- Request Body (JSON):**

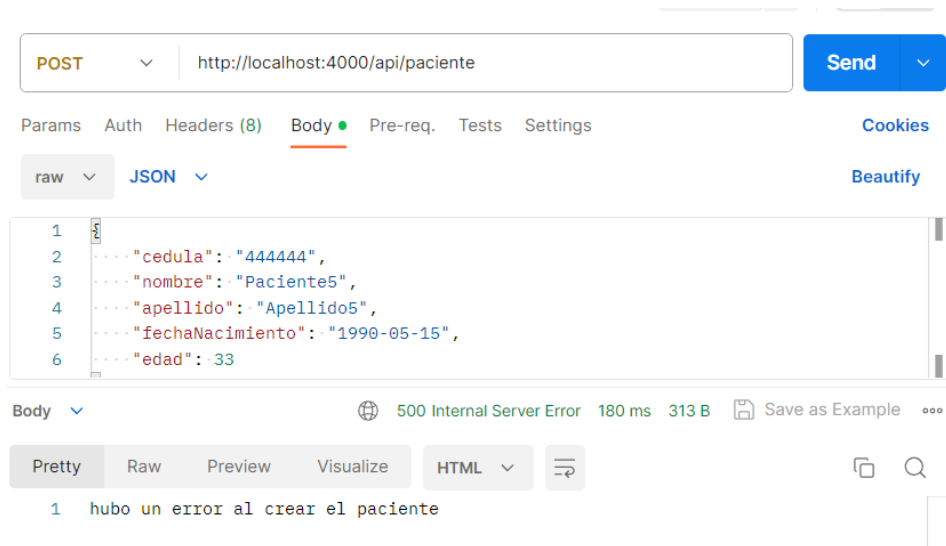
```
{  "cedula": "444444",  "nombre": "Paciente4",  "apellido": "Apellido4",  "fechaNacimiento": "1990-05-15",  "edad": 33}
```
- Status:** 200 OK
- Time:** 217 ms
- Size:** 470 B
- Response Body (JSON):**

```
{  "cedula": "444444",  "nombre": "Paciente4",  "apellido": "Apellido4",  "fechaNacimiento": "1990-05-15T00:00:00.000Z",  "edad": 33,  "fechaCreacion": "2023-07-21T20:48:41.577Z",  "_id": "64baf1422a66822ddf88a3a9",  "__v": 0}
```

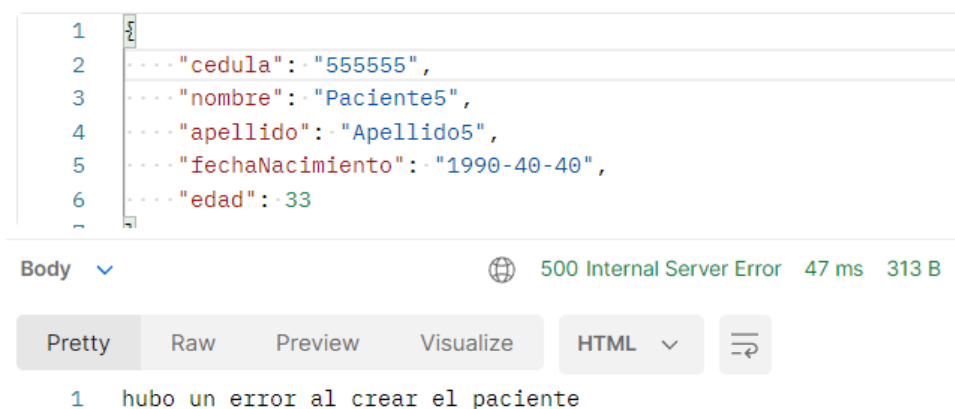
Vemos status 200 ok, y que se envía la información de forma correcta a DB. Verificamos que se haya persistido el registro en la BD



Ahora intentamos enviar una respuesta incorrecta, cedula repetida 444444, vemos un status 500 y la notificación de que hubo un error al momento de crearlo



Ahora con una fecha invalida y el resto de los campos válidos, vemos status 500 y que no puede crear el paciente.

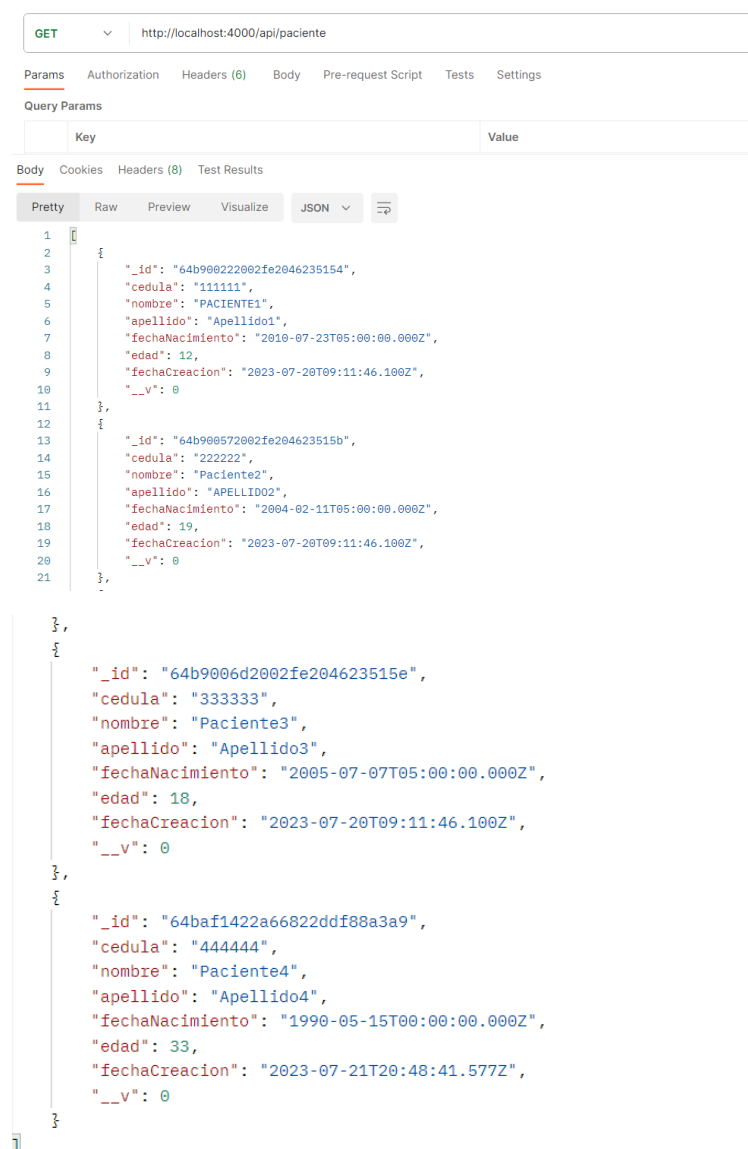


- `router.get('/', pacienteController.obtenerPaciente);`

Se utiliza el método GET para listar todos los pacientes y se maneja desde el controlador del paciente mediante una función llamada obtenerPaciente(), cuyo código es

```
exports.obtenerPaciente = async (req, res) => {
  try {
    const paciente = await Paciente.find();
    res.json(paciente)
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error en obtener al paciente");
  }
}
```

Entonces haremos la petición GET a la ruta adecuada y obtendremos la lista de todos los pacientes que se encuentran en la BD.



The screenshot shows a REST client interface with a GET request to `http://localhost:4000/api/paciente`. The response is a JSON array of 4 patient objects. The response is displayed in the 'Body' tab, formatted as JSON.

```
[{"_id": "64b900222002fe2046235154", "cedula": "111111", "nombre": "PACIENTE1", "apellido": "Apellido1", "fechaNacimiento": "2010-07-23T05:00:00.000Z", "edad": 12, "fechaCreacion": "2023-07-20T09:11:46.100Z", "__v": 0}, {"_id": "64b900572002fe204623515b", "cedula": "222222", "nombre": "Paciente2", "apellido": "APELLIDO2", "fechaNacimiento": "2004-02-11T05:00:00.000Z", "edad": 19, "fechaCreacion": "2023-07-20T09:11:46.100Z", "__v": 0}, {"_id": "64b9006d2002fe204623515e", "cedula": "333333", "nombre": "Paciente3", "apellido": "Apellido3", "fechaNacimiento": "2005-07-07T05:00:00.000Z", "edad": 18, "fechaCreacion": "2023-07-20T09:11:46.100Z", "__v": 0}, {"_id": "64baf1422a66822ddf88a3a9", "cedula": "444444", "nombre": "Paciente4", "apellido": "Apellido4", "fechaNacimiento": "1990-05-15T00:00:00.000Z", "edad": 33, "fechaCreacion": "2023-07-21T20:48:41.577Z", "__v": 0}]
```

Los cuales corresponden con todos los pacientes que hay en dicho momento en la BD.

- `router.put('/:id', pacienteController.actualizarPaciente);`

Se utiliza el método PUT para actualizar un paciente en específico identificado mediante su `_id` y se maneja desde el controlador del paciente mediante una función llamada `actualizarPaciente()`, cuyo código es

```
exports.actualizarPaciente = async (req, res) => {
  try {
    const { cedula, nombre, apellido, fechaNacimiento, edad } = req.body;
    let paciente = await Paciente.findById(req.params.id);

    if (!paciente) {
      res.status(404).json({ msg: 'No existe el paciente solicitado' });
    }
    paciente.cedula = cedula;
    paciente.nombre = nombre;
    paciente.apellido = apellido;
    paciente.fechaNacimiento = fechaNacimiento;
    paciente.edad = edad;

    paciente = await Paciente.findOneAndUpdate({ _id: req.params.id }, paciente, { new: true });
    res.json(paciente);
  } catch (error) {
    console.log(error);
    res.status(500).send('hubo un error en actualizar el paciente');
  }
}
```

Entonces se realiza la petición para editar algún campo, por medio de PUT y se da el `_id` para acceder al paciente en específico

Utilizaremos el siguiente paciente

```
1  _id: ObjectId('64baf1422a66822ddf88a3a9')
2  cedula: "4444444"
3  nombre: "Paciente4"
4  apellido: "Apellido4"
5  fechaNacimiento: 1990-05-15T00:00:00.000+00:00
6  edad: 33
7  fechaCreacion: 2023-07-21T20:48:41.577+00:00
8  __v: 0
```

The screenshot shows a REST client interface with a PUT request to `http://localhost:4000/api/paciente/64baf1422a66822ddf88a3a9`. The request body is a JSON object with the following fields: `"cedula": "444444", "nombre": "Paciente4EDITADO", "apellido": "APELLIDO4EDITADO", "fechaNacimiento": "2000-01-01", "edad": 23`. The response status is 200 OK, and the response body is a JSON object: `{"_id": "64baf1422a66822ddf88a3a9", "cedula": "444444", "nombre": "Paciente4EDITADO", "apellido": "APELLIDO4EDITADO", "fechaNacimiento": "2000-01-01T00:00:00.000Z", "edad": 23, "fechaCreacion": "2023-07-21T20:48:41.577Z", "__v": 0}`.

Cómo podemos apreciar para el paciente con el `_id` 64baf1422a66822ddf88a3a9, se le modificaron los atributos y mediante el PUT y se tiene como respuesta un Status 200 ok y la información actualizada del paciente.

Ahora con un `_id` que no esta dentro de la BD, dará un error.

The screenshot shows a REST client interface with a PUT request to `http://localhost:4000/api/paciente/74baf1422a66822ddf88a3a1`. The request body is the same JSON object as in the previous screenshot. The response status is 404 Not Found, and the response body is a JSON object: `{"msg": "No existe el paciente solicitado"}`.

Vemos que da status 404 no encontrado, porque no hay paciente con ese `_id`, además de esto si se envía una fecha con formato incorrecto o una cedula que ya exista en la BD asociado a otro paciente, no permitirá editar la información del paciente.

- **`router.get('/:id', pacienteController.obtenerPacienteid);`**

Se utiliza el método GET para obtener y listar sus atributos un paciente en específico identificado mediante su `_id` y se maneja desde el controlador del paciente mediante una función llamada `obtenerPaciente()`, cuyo código es

```

exports.obtenerPacienteId = async (req, res) => {
  try {
    let paciente = await Paciente.findById(req.params.id);

    if(!paciente) {
      res.status(404).json({ msg: 'No existe el paciente solicitado' });
    }

    res.json(paciente);
  } catch (error) {
    console.log(error);
    res.status(500).send('hubo un error en obtener al paciente');
  }
}

```

Entonces se realizará la petición con `_id` de paciente valido para que devuelva dicha información. `_id` 64baf1422a66822ddf88a3a9 (anteriormente editado sus atributos)

GET `http://localhost:4000/api/paciente/64baf1422a66822ddf88a3a9` Send

Params Auth Headers (6) **Body** Pre-req. Tests Settings Cookies

none

This request does not have a body

Body 200 OK 206 ms 484 B Save as Example

```

1 {
2   "_id": "64baf1422a66822ddf88a3a9",
3   "cedula": "444444",
4   "nombre": "Paciente4EDITADO",
5   "apellido": "APELLIDO4EDITADO",
6   "fechaNacimiento": "2000-01-01T00:00:00.000Z",
7   "edad": 23,
8   "fechaCreacion": "2023-07-21T20:48:41.577Z",
9   "__v": 0
10 }

```

Como podemos observar se trae el paciente, identificado mediante su `_id` y se listan todos sus atributos. Status 200 OK.

Ahora intentar con `_id` que no se encuentre en los pacientes de la BD

GET `http://localhost:4000/api/paciente/77baf1422a66822ddf88a3a9` Send

Params Auth Headers (6) **Body** Pre-req. Tests Settings Cookies

none

This request does not have a body

Body 404 Not Found 120 ms 316 B Save as Example

```

1 {
2   "msg": "No existe el paciente solicitado"
3 }

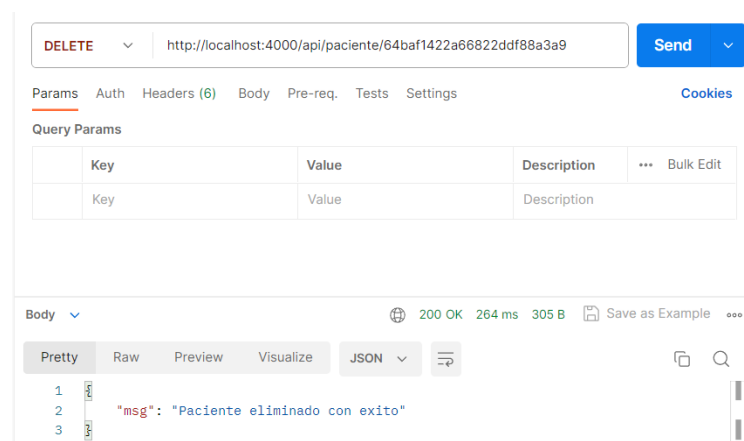
```

- `router.delete('/:id', pacienteController.eliminarPaciente);`

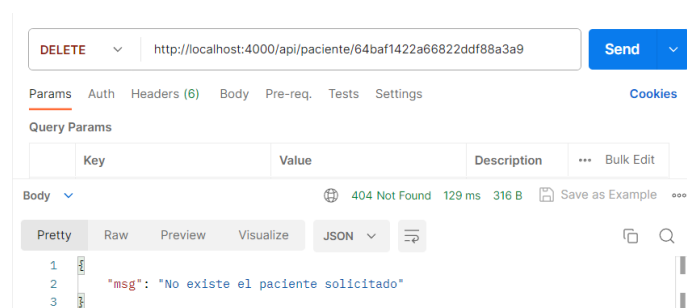
Se utiliza el método DELETE para eliminar un paciente en específico identificado mediante su `_id` y se maneja desde el controlador del paciente mediante una función llamada `obtenerPaciente()`, cuyo código es

```
✓ exports.eliminarPaciente = async (req, res) => {  
  ✓  
    try {  
      let paciente = await Paciente.findById(req.params.id);  
  
      ✓  
      if(!paciente) {  
        res.status(404).json({ msg: 'No existe el paciente solicitado' })  
      }  
  
      await Paciente.findOneAndRemove({ _id: req.params.id})  
      res.json({msg: 'Paciente eliminado con éxito'});  
  
    } catch (error) {  
      console.log(error);  
      res.status(500).send('hubo un error al eliminar el paciente');  
    }  
  }  
}
```

Entonces se realiza la petición y se especifica el `_id` para borrar el paciente de la BD, utilizaremos `_id` 64baf1422a66822ddf88a3a9, cuando `_id` está en la colección de pacientes, elimina el registro, en caso contrario envía un mensaje que no existe el paciente solicitado



Volveremos a enviar ese mismo `_id` y debe mostrar error, ya que dicho `_id` ya no se encuentra dentro de la colección



- **Rutas de doctor.js**

(Esquema)

```
const DoctorSchema = mongoose.Schema({
  nombre: {
    type: String,
    required: true
  },
  apellido: {
    type: String,
    required: true
  },
  especialidad: {
    type: String,
    required: true
  },
  consultorio: {
    type: Number,
    required: true
  },
  correo: {
    type: String,
    required: true
  },
  fechaCreacion: {
    type: Date,
    default: Date.now()
  }
});
```

- **router.post('/', doctorController.crearDoctor);**

Se utiliza el método POST para crear un nuevo doctor y se maneja desde el controlador del doctor mediante una función llamada crearDoctor(), cuyo código es

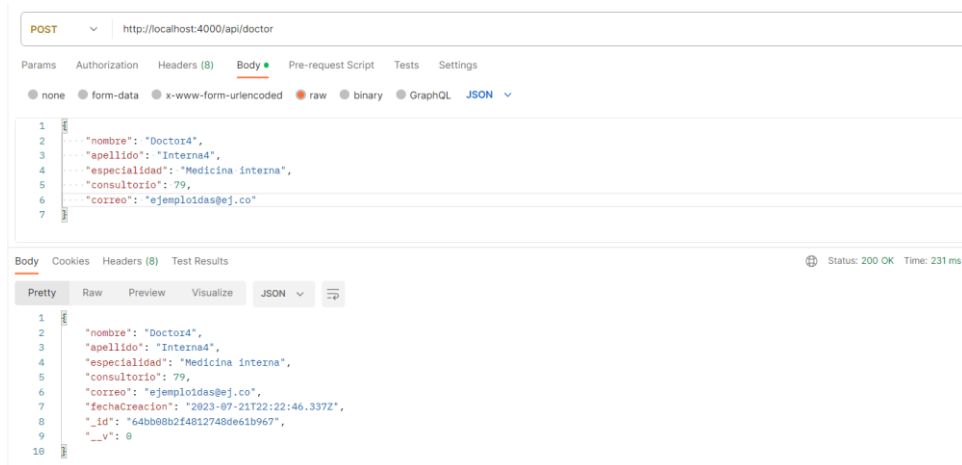
```
exports.crearDoctor = async (req, res) => {
  try {
    let doctor;

    //Se crea el doctor
    doctor = new Doctor(req.body);

    await doctor.save();
    res.send(doctor);
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error en la creacion del doctor");
  }
}
```

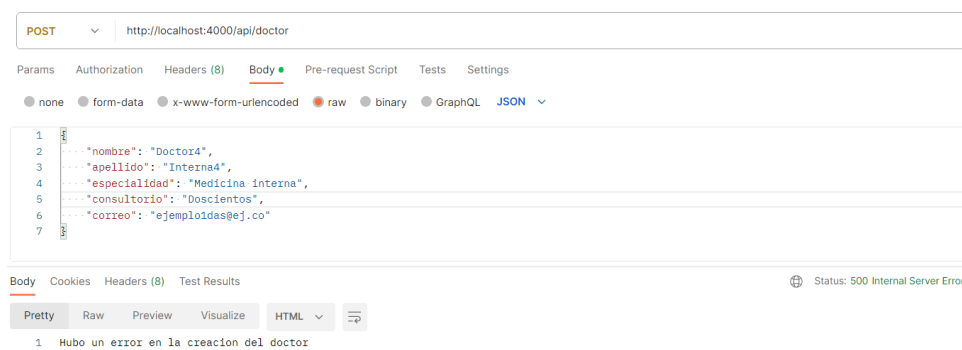
Se debe enviar el nombre, apellido, especialidad, consultorio, correo (y _id que será el identificador único y se genera automáticamente y fechaCreacion que también se genera automáticamente)

Por ejemplo, podemos observar cuando se envían datos correctos que se crea un nuevo doctor.



Status 200 OK. Se crea un nuevo doctor en la colección.

Ahora enviamos un formulario incorrecto, ya que consultorio es number



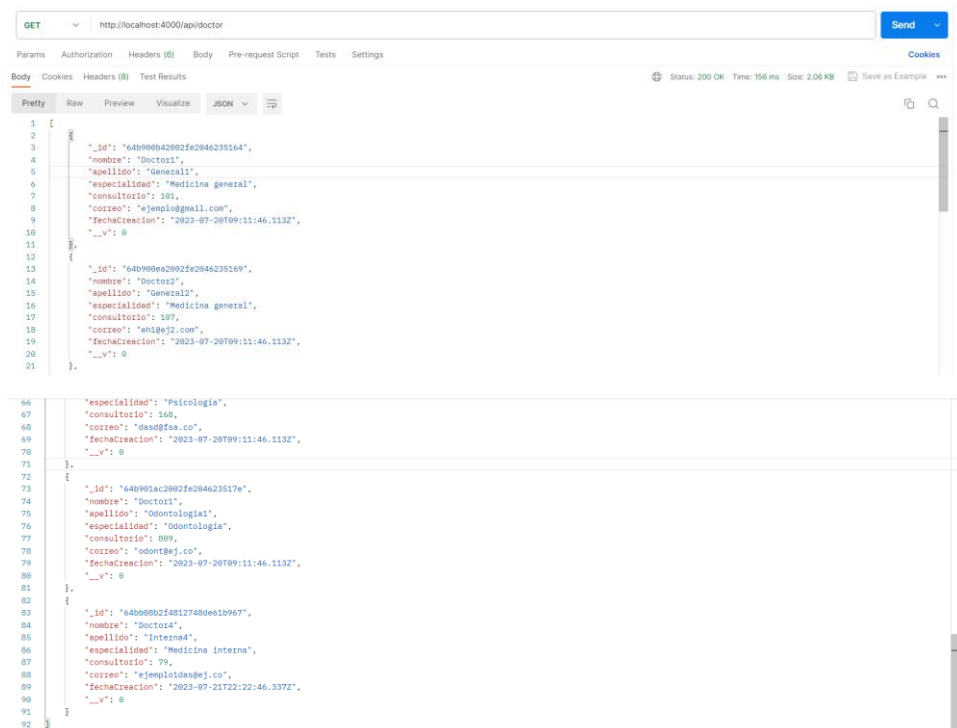
Apreciamos un status 500 error y el mensaje de que hubo un error en la creación del doctor.

- **router.get('/', doctorController.obtenerDoctores);**

Se utiliza el método GET para listar todos los doctores y se maneja desde el controlador del doctor mediante una función llamada obtenerDoctores(), cuyo código es

```
exports.obtenerDoctores = async (req, res) => {
  try {
    const doctores = await Doctor.find();
    res.json(doctores)
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error ..");
  }
}
```

Se realiza la solicitud y se listan todos los doctores dentro de la colección junto con sus atributos.



Vemos un status 200 OK, y la lista de los doctores que están registrados dentro de la colección junto sus atributos.

- **router.put('/:id', doctorController.actualizarDoctor);**

Se utiliza el método PUT para editar un doctor específico, identificado mediante su `_id` y se maneja desde el controlador del doctor mediante una función llamada `crearDoctor()`, cuyo código es

```

exports.actualizarDoctor = async (req, res) => {
  try {
    const {nombre, apellido, especialidad, consultorio, correo} = req.body;
    let doctor = await Doctor.findById(req.params.id);

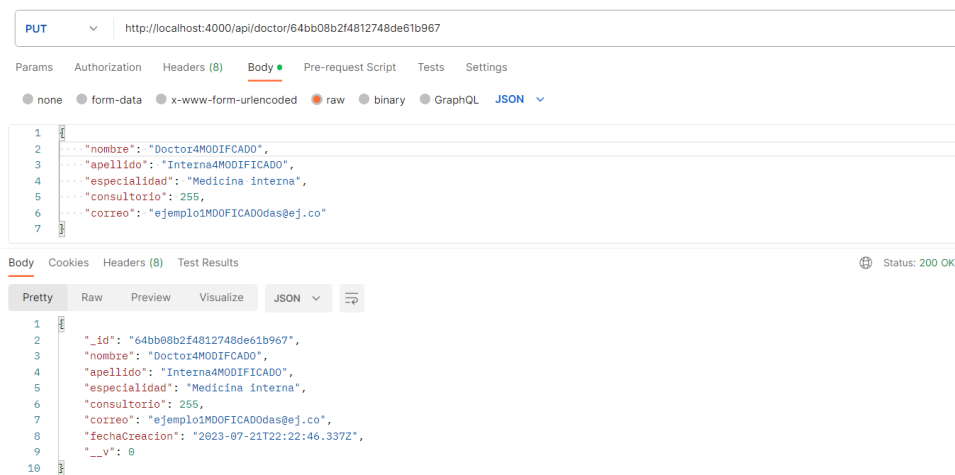
    if(!doctor) {
      res.status(404).json({ msg: "No existe el doctor" });
    }

    doctor.nombre = nombre;
    doctor.apellido = apellido;
    doctor.especialidad = especialidad;
    doctor.consultorio = consultorio;
    doctor.correo = correo;

    doctor = await Doctor.findOneAndUpdate({ _id: req.params.id }, doctor, { new: true });
    res.json(doctor);
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error ..");
  }
}

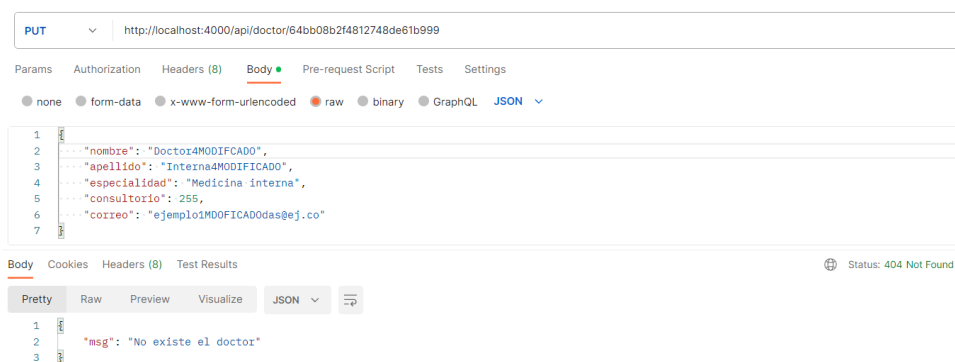
```

Se edita un doctor enviándole el `_id` del doctor al que hacemos referencia, y se pueden modificar sus atributos teniendo las mismas restricciones que al momento de crear un nuevo doctor (respetar el tipo de datos y enviar todos los atributos) editaremos el doctor con `_id` 64bb08b2f4812748de61b967



Como podemos ver un status 200 OK, se envía la solicitud junto con los valores actualizados del doctor y los modifica en la BD y devuelve el resultado para comprobar que se realizaron correctamente los resultados.

Ahora si intentamos con `_id` que no está en la colección de doctors, vemos que se recibe un error



Status 404 no se encuentra el doctor, con el `_id` requerido

- **`router.get('/:id', doctorController.obtenerDoctor);`**

Se utiliza el método GET para obtener un doctor específico y listar sus atributos, identificado mediante su `_id` y se maneja desde el controlador del doctor mediante una función llamada `obtenerDoctor()`, cuyo código es

```
exports.obtenerDoctor = async (req, res) => {
  try {
    let doctor = await Doctor.findById(req.params.id);

    if(!doctor) {
      res.status(404).json({ msg: "No existe el doctor" });
    }

    res.json(doctor);
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error ..");
  }
}
```

Ahora se seleccionará con el método GET un doctor y se listarán sus atributos, identificado mediante el `_id` del doctor, se utilizará el `_id` 64bb08b2f4812748de61b967

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `http://localhost:4000/api/doctor/64bb08b2f4812748de61b967`
- Status:** 200 OK
- Response Time:** 149 ms
- Response Size:** 503 B
- Body (JSON):**

```
{  "id": "64bb08b2f4812748de61b967",  "nombre": "Doctor4MODIFICADO",  "apellido": "Interna4MODIFICADO",  "especialidad": "Medicina interna",  "consultorio": 255,  "correo": "ejemplo1MDOFICAD0das@ej.co",  "fechaCreacion": "2023-07-21T22:22:46.337Z",  "__v": 0}
```

Vemos un status 200 OK y la información del doctor, identificado mediante su `_id`

Ahora si enviamos un `_id` que no se encuentra en la colección un error.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `http://localhost:4000/api/doctor/64bb08b2f4812748de61b999`
- Status:** 404 Not Found
- Response Time:** 109 ms
- Response Size:** 303 B
- Body (JSON):**

```
{  "msg": "No existe el doctor"}
```

Status 404 no encontrado doctor.

- `router.delete('/:id', doctorController.eliminarDoctor);`

Se utiliza el método DELETE para eliminar un doctor específico, identificado mediante su `_id` y se maneja desde el controlador del doctor mediante una función llamada `eliminarDoctor()`, cuyo código es

```

exports.eliminarDoctor = async (req, res) => {
  try {
    let doctor = await Doctor.findById(req.params.id);

    if(!doctor) {
      res.status(404).json({ msg: "No existe el doctor" });
    }

    await Doctor.findOneAndRemove({ _id: req.params.id })
    res.json({ msg: 'El doctor fue eliminado con éxito' });
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error ..");
  }
}

```

Ahora se utilizará el DELETE, para borrar un doctor, se utilizará el _id 64bb08b2f4812748de61b967

The screenshot shows a REST client interface with a DELETE request to `http://localhost:4000/api/doctor/64bb08b2f4812748de61b967`. The response status is 200 OK, with a response time of 233 ms and a body size of 310 B. The response body is a JSON object: `{ "msg": "El doctor fue eliminado con éxito" }`.

Como podemos ver un estatus 200 OK y el mensaje que se eliminó el doctor con éxito, ahora probaremos enviando la misma solicitud, la cual debe dar error ya que ese _id 64bb08b2f4812748de61b967 ya no existe dentro de la colección porque acabo de ser eliminado.

The screenshot shows the same REST client interface with the same DELETE request. The response status is 404 Not Found, with a response time of 105 ms and a body size of 303 B. The response body is a JSON object: `{ "msg": "No existe el doctor" }`.

- `router.get('/especialidad/:especialidad', doctorController.obtenerDoctoresPorEspecialidad);`

Se utiliza el método GET para obtener los doctores que tienen una especialidad específica, identificado mediante su especialidad y se maneja desde el controlador del doctor mediante una función llamada **obtenerDoctoresPorEspecialidad** (), esta función es utilizada para mostrar una lista

con los doctores que tienen una especialidad específica y es utilizada para crear las citas cuyo código es

```
exports.obtenerDoctoresPorEspecialidad = async (req, res) => {
  try {
    const especialidad = req.params.especialidad;
    const doctores = await Doctor.find({ especialidad }, 'nombre apellido _id');
    res.json(doctores);
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error al obtener los doctores");
  }
};
```

Entonces utilizaremos la especialidad 'Cardiología' para ver la lista con los doctores con dicha especialidad

The screenshot shows a REST client interface. At the top, a GET request is defined for the URL `http://localhost:4000/api/doctor/especialidad/'Cardiología'`. Below the URL bar, tabs for 'Params', 'Auth', 'Headers (6)', 'Body', 'Pre-req.', 'Tests', and 'Settings' are visible, with 'Params' selected. A 'Query Params' table is shown with columns 'Key', 'Value', 'Description', and 'Bulk Edit'. The 'Body' tab is also visible, showing a status of '200 OK' and a response size of '101 ms 267 B'. The response is displayed in 'JSON' format.

Key	Value	Description	Bulk Edit
Key	Value	Description	

Vemos un status 200 OK, esta ruta se utiliza para obtener el nombre apellido y _id de los doctores que cumplen con la especialidad específica.

- **Rutas de cita.js**

(Esquema)

```
const CitaSchema = mongoose.Schema({
  idPaciente: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Paciente',
    required: true
  },
  idDoctor: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Doctor',
    required: true
  },
  fechaCita: {
    type: Date,
    required: true
  },
  fechaCreacion: {
    type: Date,
    default: Date.now()
  }
});
```

Solo se requieren de dichos atributos, pero en el front end se muestran los datos asociados al paciente y al doctor, que se pueden identificar mediante idPaciente y idDoctor.

- `router.post('/', citaController.crearCita);`

Se utiliza el método POST para crear una nueva cita, se maneja desde el controlador de la cita mediante una función llamada `crearCita()`, cuyo código es

```
exports.crearCita = async (req, res) => {
  try {
    const { cedula, especialidad, fechaCita } = req.body;

    // Verificar si existe el paciente por cédula
    const paciente = await Paciente.findOne({ cedula });
    if (!paciente) {
      return res.status(404).json({ msg: "No existe el paciente con esa cédula" });
    }

    // Verificar si existe al menos un doctor con la especialidad requerida
    const doctor = await Doctor.findOne({ especialidad });
    if (!doctor) {
      return res.status(404).json({ msg: "No existe doctor con esa especialidad" });
    }

    // Crear la cita utilizando los _id encontrados
    const cita = new Cita({ idPaciente: paciente._id, idDoctor: doctor._id, fechaCita });

    await cita.save();

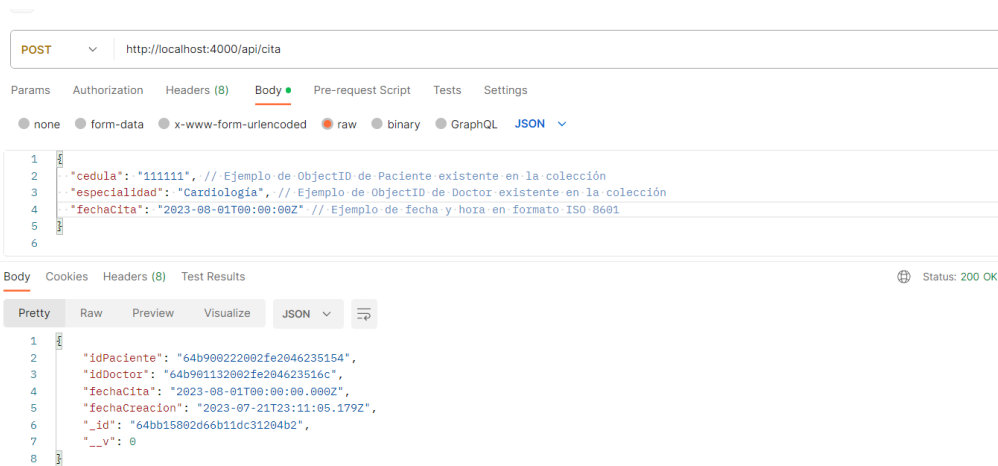
    res.json(cita);
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error en la creación de la cita");
  }
};
```

Se debe enviar una petición con esta estructura

```
{
  "cedula": "CedulaPaciente", // Ejemplo de ObjectID de Paciente existente en la colección
  "especialidad": "EspecialidadDoctor", // Ejemplo de ObjectID de Doctor existente en la colección
  "fechaCita": "AAAA-MM-DDT00:00:00Z" // Ejemplo de fecha y hora en formato ISO 8601
}
```

Con esto la cedula se identifica el `_id` del paciente asociado, con la especialidad se verifica que por lo menos haya 1 doctor con la especialidad (desde el front se muestran los nombres de los doctores que cumplen con la especialidad, para que el usuario decida el doctor con el cuál quiere la cita, así que el `_id` doctor se identifica de forma más acertada cuando el usuario decide el nombre del doctor con el cuál quiere la cita)

Utilizaremos la cedula 111111 que se encuentra dentro de la colección de pacientes, la especialidad Medicina interna y fecha 2023-08-01T00:00:00Z, estos son datos válidos por lo cual creará la cita.

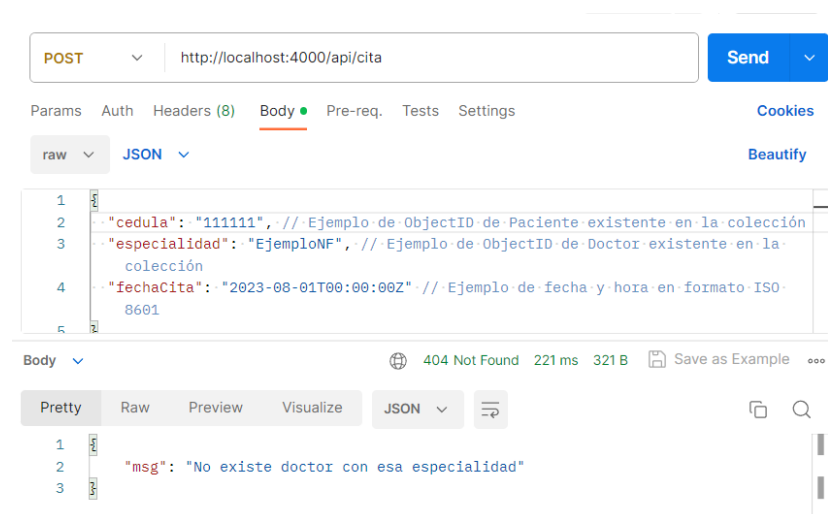


Vemos un status 200 OK, por lo que se creó la cita correctamente.

Ahora utilizaremos una cedula que no existe dentro de la colección de pacientes, para que de error.



Vemos status 404, porque no se encuentra esa cedula dentro de la colección de paciente. Ahora daremos una especialidad que no se encuentra dentro en la colección de doctors.



Como podemos apreciar status 404, no se encuentra con un doctor con dicha especialidad.

- **router.get('/', citaController.obtenerCitas);**

Se utiliza el método GET para obtener una lista completa de todas las citas, se maneja desde el controlador de la cita mediante una función llamada obtenerCitas(), cuyo código es

```
exports.obtenerCitas = async (req, res) => {
  try {
    const citas = await Cita.find()
      .populate('idPaciente', 'cedula nombre apellido') // Muestra solo cedula, nombre y apellido del paciente
      .populate('idDoctor', 'especialidad nombre apellido consultorio') // Muestra solo especialidad, nombre y apellido del doctor
      .select('idPaciente idDoctor fechaCita'); // Muestra solo estos campos de la cita

    console.log(citas);
    res.json(citas);
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error listando las citas");
  }
};
```

Utilizamos para obtener una lista completa de las citas y mostrar atributos de las colecciones paciente y doctors, que serán utilizados en el front end para mostrar de forma más detallada los datos de los involucrados de las citas.

The screenshot shows a REST client interface with a GET request to `http://localhost:4000/api/cita` sent successfully. The response is a JSON array of two appointment objects. Each object contains an `_id`, an `idPaciente` object with `_id`, `cedula`, `nombre`, and `apellido`, an `idDoctor` object with `_id`, `nombre`, `apellido`, `especialidad`, and `consultorio`, and a `fechaCita` string.

```
{
  "_id": "64b9022c2002fe2046235196",
  "idPaciente": {
    "_id": "64b900222002fe2046235154",
    "cedula": "111111",
    "nombre": "PACIENTE1",
    "apellido": "Apellido1"
  },
  "idDoctor": {
    "_id": "64b900b42002fe2046235164",
    "nombre": "Doctor1",
    "apellido": "General1",
    "especialidad": "Medicina general",
    "consultorio": 101
  },
  "fechaCita": "2023-07-28T05:00:00.000Z"
},
{
  "_id": "64b9023d2002fe204623519e",
  "idPaciente": {
    "_id": "64b900572002fe204623515b",
    "cedula": "222222",
    "nombre": "Paciente2",
    "apellido": "APELLIDO02"
  },
  "idDoctor": {
```

```

27     "idDoctor": {
28       "_id": "64b901132002fe204623516c",
29       "nombre": "Doctor1",
30       "apellido": "Cardiologia",
31       "especialidad": "Cardiología",
32       "consultorio": 205
33     },
34     "fechaCita": "2023-07-25T05:00:00.000Z"
35   },
36   {
37     "_id": "64bb15802d66b11dc31204b2",
38     "idPaciente": {
39       "_id": "64b900222002fe2046235154",
40       "cedula": "111111",
41       "nombre": "PACIENTE1",
42       "apellido": "Apellido1"
43     },
44     "idDoctor": {
45       "_id": "64b901132002fe204623516c",
46       "nombre": "Doctor1",
47       "apellido": "Cardiologia",
48       "especialidad": "Cardiología",
49       "consultorio": 205
50     },
51     "fechaCita": "2023-08-01T00:00:00.000Z"
52   }
53 ]

```

Como podemos apreciar status 200 OK y retorna la lista de las citas con información ampliada de cada involucrado en la cita, para poder mostrarlo desde el front end

- **router.delete('/:id', citaController.eliminarCita);**

Se utiliza el método Delete para eliminar una cita en específico, identificada mediante su `_id` se maneja desde el controlador de la cita mediante una función llamada `crearCita()`, cuyo código es

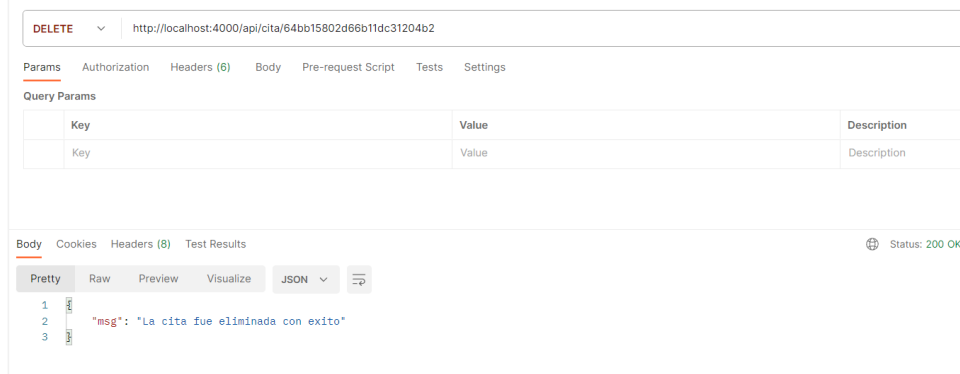
```

exports.eliminarCita = async (req, res) => {
  try {
    let cita = await Cita.findById(req.params.id);

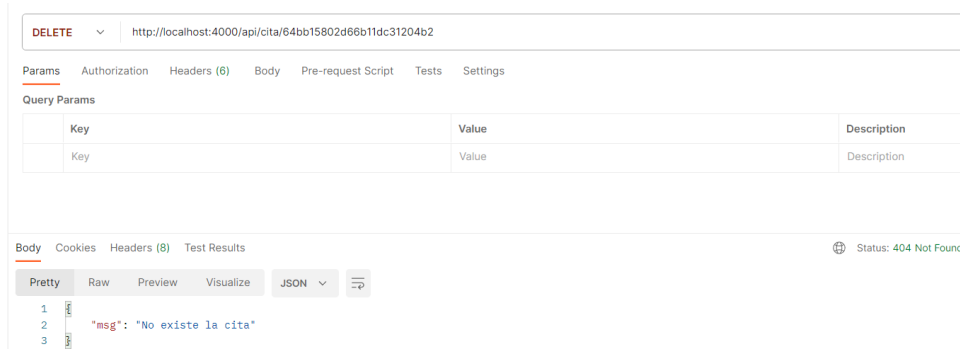
    if(!cita){
      res.status(404).json( {msg : 'No existe la cita'})
    }
    await Cita.findOneAndRemove({ _id: req.params.id })
    res.json({ msg:'La cita fue eliminada con exito' });
  } catch (error) {
    console.log(error);
    res.status(500).send("Hubo un error eliminando la cita");
  }
};

```

Se le da un `_id` de la cita para eliminar la cita, en este caso utilizaremos el `_id` `64bb15802d66b11dc31204b2` que se encuentra dentro de la colección de citas.



Como podemos observar status 200 OK, se elimina la cita con dicho _id. Ahora enviaremos nuevamente dicha petición y será error ya que ese _id ya no se encuentra dentro de la colección de citas.



Como podemos apreciar status 404 Not found, ya que no se encuentra dicho _id dentro de la colección de citas.

(No se crea EDITAR, principalmente porque es la forma tradicional que se manejan las citas, como se consideran compromisos no son tan fáciles de modificar, sino que se puede eliminar la cita o crear una nueva)

AUTOR: JUAN CARLOS GARCIA GUERRERO