# 3. Message-Services

## Unidad 3. Ejercicio final

## Programación de Servicios y Procesos

# Tabla de contenido

# 1. Introducción y primeros pasos

En este ejercicio final, se pide que implemente algunos servicios web utilizando NodeJS (Express) y MongoDB (Mongoose). Estos servicios serán usados por el proyecto que realizarás en la unidad 4. Por ahora, se utilizará ***Postman*** para verificar que todo funcione correctamente.

Estos servicios proporcionarán información simple para manejar a los usuarios y los mensajes enviados y recibidos de esos usuarios. También se debe desarrollar un servicio de inicio de sesión, por lo que deberás implementar un *token* de autenticación (ver Anexo III).

## 1.1. Configurar el proyecto

Crea un proyecto Node (NPM) llamado ***message-services*** (dentro de un directorio con el mismo nombre). Esta será la carpeta que debes entregar comprimida en un archivo zip (recuerda eliminar el directorio ***node_modules*** antes de comprimir).

## 1.2. Configuración de la base de datos

La base de datos almacenará 2 esquemas diferentes:

**User**

- ***name***: *String*, obligatorio, *trim*, longitud mínima = 1, solo caracteres alfanuméricos, único.
- ***password***: *String*, obligatorio, longitud mínima = 4.
- ***image***: *String*, obligatorio.

**Message**

- ***from***: *ObjectId*, referencia al modelo to *User('users')*. Obligatorio.
- ***to***: *ObjectId*, referencia al modelo to *User('users')*. Obligatorio.
- ***message***: *String*, obligatorio, *trim*, longitud mínima = 1.
- ***image***: *String*, no obligatorio.
- ***sent***: *String*, obligatorio, *trim*, longitud mínima = 10.

# 2. Servicios

Implementa los siguientes servicios y pruébalos con *Postman*.

## 2.1. *Users*

Para implementar los servicios de inicio de sesión y registro, debes instalar el paquete NPM **sha256**. Las contraseñas deben almacenarse con este cifrado(instrucciones aquí).

Puedes configurar el vencimiento del *token* para que sea muy largo en el futuro (meses o años), por lo que no sería necesario renovarlo.

**(POST) /login**

Este servicio recibirá un objeto JSON que contiene un nombre de usuario y una contraseña, como este:

```
{
    "name": "test",
    "password": "1234"
}
```

Verificará si existe un usuario con ese nombre y contraseña (encriptada) en la base de datos, y en caso de que la solicitud sea exitosa, devolverá el *token* de autenticación, y también el nombre y la imagen de avatar del usuario:

```
{
    "ok": true,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1YTUyNzk4NmM0YmE0Zjc2NTAzNWY3ZG
              IiLCJuYW1lIjoidGVzdCIsImlhdCI6MTUxNTM1NTEyNCwiZXhwIjoxNTc4NDcwMzI0fQ.BflHN2p9I-
              qN1GeR_MR2rlwNRdfTut_iGBRLPuO7Z9U",
    "name": 'test',
    "image": 'img/1546941995997.jpg'
}
```

Como puedes ver, el campo de la imagen es una referencia a la ruta del archivo que contiene el avatar del usuario. Para que la aplicación cliente pueda mostrar esa imagen, debes permitir que NodeJS proporcione recursos estáticos:

```
app.use(express.static('public'));
app.use('/img', express.static(__dirname + '/img'));
```

Si se produce un error en el proceso de inicio de sesión, este servicio debe devolver el siguiente JSON:

```
{
    "ok": false,
    "error": "User or password incorrect"
}
```

### (POST) /register

Este servicio recibirá un nombre de usuario y una contraseña, y ambos campos deben guardarse en la base de datos. También se debe proporcionar una imagen de avatar para ese usuario. Desde el lado del cliente, se enviará una imagen JPEG (no hay que verificar eso) codificada en base64:

```
{
    "name": "test",
    "password": "1234",
    "image": "JPEG Image in Base64 format"
}
```

Después de recibir esa solicitud, debes crear un archivo que contenga la imagen, para generar la URL de la imagen, que será referenciada y mostrada en el lado del cliente. También deberás tener en cuenta que debes generar un nombre único para cada archivo. Para hacer eso, puedes usar el siguiente código, ya que la fecha actual será diferente cada vez que se ejecute el código:

```
const filePath = `img/${Date.now()}.jpg`;
fs.writeFileSync(filePath, Buffer.from(request.body.image, 'base64'));
```

Si el usuario se registró correctamente, se devolverá el siguiente objeto:

```
{
    "ok": true
}

Si algo ha ido mal:
{
    "ok": false,
    "error": "User couldn't be registered"
}
```

A partir de este momento, para todos los siguientes servicios debes comprobar que el *token* ha sido enviado (encabezado de autorización) y es válido. Si algo sale mal con el *token*, debes devolver una respuesta 401 (No autorizado) → ***response.sendStatus(401)***.

### (GET) /users

Devolverá un array con todos los usuarios registrados que están almacenados en la base de datos. Utiliza el siguiente formato:

```
{
    "ok": true,
    "users": [
        {
            "_id": "5c34762b87a65b4316ddeb18",
            "name": "test1",
            "password": "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4",
            "image": "img/1515355769269.jpg",
            "__v": 0
        },
        {
            "_id": "5c349c110487514a972f598a",
            "name": "test2",
            "password": "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4",
```

```
            "image": "img/1546951697375.jpg",
            "__v": 0
        }
    ]
}
```

### (PUT) /users

Este servicio actualizará la imagen de avatar de un usuario. Si deseas obtener la información actual para cambiar solo el archivo de imagen, en lugar de usar *findByIdAndUpdate*, puedes utilizar el siguiente código donde se obtienen los datos del usuario durante el proceso de autenticación:

```
const user = …

User.findById(user._id).then(user => {
    // Generate a new ".jpg" file where the new avatar image will be saved

    user.save().then(user => {
        // Updated
    }).catch(error => {
        // Error updating
    });

}).catch(error => {
    // Book not found
});
```

Después de recibir la imagen, recuerda que debes generar un nuevo archivo para que la cadena que contiene la ruta también se actualice en la base de datos. Puedes hacerlo de la misma manera que se hizo en el proceso de registro:

```
const filePath = `img/${Date.now()}.jpg`;
fs.writeFileSync(filePath, Buffer.from(request.body.image, 'base64'));
```

Este servicio devolverá al menos dos respuestas como los servicios anteriores. En caso de que la solicitud se procese correctamente:

```
{
    "ok": true
}

En caso de que algo salga mal:
{
    "ok": false,
    "error": "Error updating user: 5c34762b87a65b4316ddeb18"
}
```

## 2.2. *Messages*

En estos servicios también deberás comprobar que se ha enviado el *token*, devolviendo un estado 401 en caso contrario.

### (GET) /messages

Este servicio devuelve todos los mensajes que se enviaron al usuario que está realizando la solicitud. Puedes usar código similar a este:

```
const user = …

Message.find({to: user._id}).then(messages => {
    …
}).catch(error => {
    …
});
```

Ejemplo de respuesta:

```
{
    "ok": true,
    "messages": [
        {
            "_id": "5c34c1557fe11c54bad1625a",
            "from": "5c34762b87a65b4316ddeb18",
            "to": "5c34c2737fe11c54bad1625c",
            "message": "Hello! This is a message without image",
            "sent": "30/01/2019",
            "__v": 0
        },
        {
            "_id": "5c34c16b7fe11c54bad1625b",
            "from": "5c349c110487514a972f598a",
            "to": "5c34c2737fe11c54bad1625c",
            "message": "Hello! This is a message with image",
            "message": "img/1546961259261.jpg",
            "sent": "31/01/2019",
            "__v": 0
        }
    ]
}
```

Devuelve una respuesta de error si no se pueden recibir esos mensajes (por cualquier motivo):

```
{
    "ok": false,
    "error": "Error getting messages for user: 5c34c16b7fe11c54bad1625b"
}
```

### (POST) /messages/:toUserId

Este servicio envía un nuevo mensaje a un usuario. El usuario que envía el mensaje es el que está registrado (puedes obtener esa información del *token*) y la identificación del usuario de destino se recibe como parámetro. Los datos JSON que debes enviar al servicio son el texto, la imagen (opcional) y la fecha actual cuando se envía el mensaje, así:

```
{
    "message": "This is a message",
    "image": "JPEG Image in Base64 format",
    "sent": "31/01/2019"
}
```

El servicio devolverá el mensaje insertado si todo sale como se esperaba:

```
{
    "ok": true,
    "message": {
        "_id": "5a52823b5e2a930529e061e3",
        "from": "5c349c110487514a972f598a",
        "to": "5c34c2737fe11c54bad1625c",
        "messsage": "This is a message",
        "image": "JPEG Image in Base64 format",
        "sent": "31/01/2019",
        "__v": 0
    }
}
```

O un error si algo sale mal:

```
{
    "ok": false,
    "error": "Error sending a message to: 5c34c2737fe11c54bad1625c"
}
```

### (DELETE) /messages/:id

Este servicio elimina el mensaje que coincida con la identificación recibida. Debes proporcionar una respuesta si se elimina el mensaje:

```
{
    "ok": true
}

En caso de no perder realizarse la petición:
{
    "ok": false,
    "error": "Error deleting message: 5a52823b5e2a930529e061e3"
}
```

# 3. Mejoras opcionales

- Complete el campo *“from”* (que identifica a los usuarios que se envían los mensajes) para que tanto el nombre de usuario como las imágenes de avatar también se proporcionen al recuperar los mensajes con el servicio GET. De esta forma, la aplicación cliente puede listar todos los mensajes junto con los usuarios que los enviaron.
- En lugar de recibir la fecha *“sent”* como un *String*, usa un tipo fecha y utiliza la fecha actual en el servidor.

# 4. Reglas de evaluación

## 4.1. Parte obligatoria

Para obtener la nota final, se aplicarán los siguientes criterios:

- Esquemas, conexión a la base de datos e inicialización de la aplicación: **1 punto**.
- Servicio de inicio de sesión: **1 punto**.
- Servicio de registro: **1 punto**.
- Obteniendo todos los usuarios: **1 punto**.
- Actualización de la imagen de un usuario: **1 punto**.
- Recibir los mensajes del usuario registrado: **1 punto**.
- Envío de un mensaje: **1 punto**.
- Eliminar un mensaje: **1 punto**.
- Autenticación del usuario cuando sea necesario: **1 punto**.
- Comprobación y envío de mensajes de error, código limpio y comentarios: **1 punto**.

## 4.2. Acerca de las mejoras opcionales

Los cambios opcionales propuestos pueden mejorar tu nota hasta 1 punto extra, pero solo si tu nota mínima en la parte obligatoria es de al menos 7.