UNIVERSIDAD SAN CARLOS DE GUATEMALA

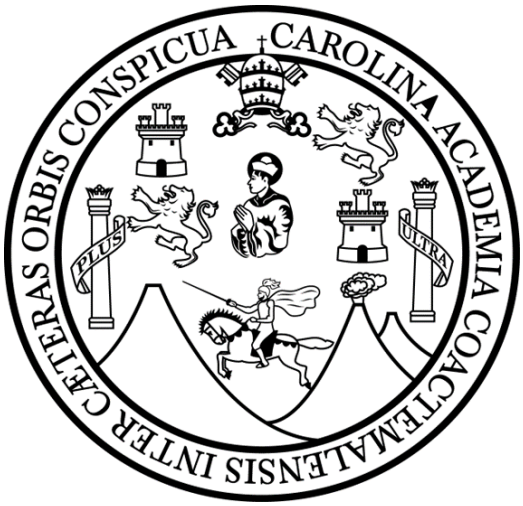
FACULTAD DE CIENCIAS DE LA INGENIERÍA

BASES DE DATOS 2

ING. MARLON FRANCISCO ORELLANA LOPEZ

AUX. JHONATHAN DANIEL TOCAY COTZOJAY

SECCIÓN P



DOCUMENTACIÓN PROYECTO 2

GRUPO 6

POR:

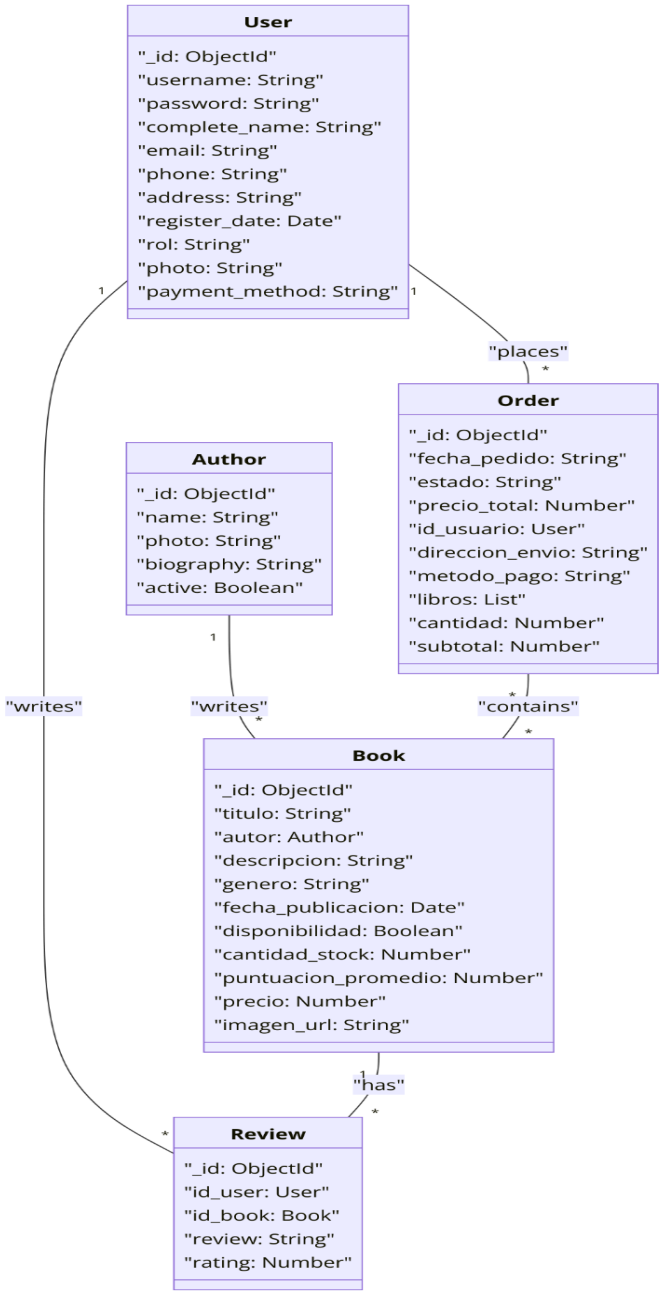
201930693 -	3146	39217	0901	-	HERNANDEZ SAPÓN, LEVÍ ISAAC
201930697	3348	21282	0901		SÁNCHEZ SANTOS, LUIS FERNANDO
201930699 -	3134	21633	0901	-	MORALES XICARÁ, ERICK DANIEL
201230463 -	2253	62503	0901	-	SUM COYOY, RANDY MARCELINO
202031064 -	3133	35931	0901	-	OVALLE DE LEÓN, CARLOS ALEXIS
202031683 -	3353	15267	0901	-	GORDILLO GONZÁLEZ, PEDRO RICARDO

GUATEMALA, GUATEMALA, 28/06/2024

Índice

Estructura de la base de datos

Diagrama:



Casos de Uso

Books

Caso de uso	Obtener todos los libros	CU 001
Actor(es)	Administrador, Usuario	
Poscondición	Se obtienen todos los libros disponibles en el sistema	
Resumen	Se solicita la lista completa de libros desde el sistema	
Tipo	Primario	

Caso de uso	Obtener libro por ID	CU 002
Actor(es)	Administrador, Usuario	
Poscondición	Se obtiene la información del libro correspondiente al ID solicitado	
Resumen	Se solicita información específica de un libro mediante su ID	
Tipo	Primario	

Caso de uso	Agregar libro	CU 003
Actor(es)	Administrador	
Poscondición	Se agrega un nuevo libro al catálogo	
Resumen	Se registran los datos de un nuevo libro en el sistema	
Tipo	Primario	

Caso de uso	Eliminar libro	CU 004
Actor(es)	Administrador	
Poscondición	Se elimina un libro existente del catálogo	
Resumen	Se elimina un libro del sistema mediante su ID	
Tipo	Primario	

Caso de uso	Actualizar libro	CU 005
Actor(es)	Administrador	
Poscondición	Se actualiza la información de un libro existente	
Resumen	Se actualizan los datos de un libro específico mediante su ID	
Tipo	Primario	

Caso de uso	Obtener puntuación por ID	CU 006
Actor(es)	Administrador, Usuario	
Poscondición	Se obtiene la puntuación del libro correspondiente al ID solicitado	
Resumen	Se solicita la puntuación de un libro mediante su ID	
Tipo	Primario	

Caso de uso	Obtener libros por autor	CU 007
Actor(es)	Administrador, Usuario	
Poscondición	Se obtienen todos los libros de un autor específico	
Resumen	Se solicita la lista de libros escritos por un autor específico	
Tipo	Primario	

Caso de uso	Filtrar libros	CU 008
Actor(es)	Administrador, Usuario	
Poscondición	Se obtienen libros filtrados según el criterio especificado	
Resumen	Se filtran los libros en el sistema según un criterio (título, autor, género, precio, puntuación)	
Tipo	Primario	

Uploads

Caso de uso	Añadir imagen	CU 009
Actor(es)	Administrador, Usuario	
Poscondición	La imagen se sube al servidor y se devuelve el nombre del archivo subido	
Resumen	El usuario sube una imagen que se almacena en el servidor y obtiene el nombre del archivo	
Tipo	Primario	

Caso de uso	Actualizar imagen	CU 010
Actor(es)	Administrador, Usuario	
Poscondición	La imagen existente se actualiza en el servidor y se devuelve el nombre del archivo actualizado	
Resumen	El usuario actualiza una imagen existente en el servidor y obtiene el nombre del archivo actualizado	
Tipo	Primario	

Caso de uso	Obtener imagen	CU 011
Actor(es)	Administrador, Usuario	
Poscondición	La imagen solicitada se redirige desde el servidor	
Resumen	El usuario solicita una imagen específica y se redirige a la URL de la imagen almacenada en el servidor	
Tipo	Primario	

Sales

Caso de uso	Crear pedido	CU 012
Actor(es)	Usuario, Administrador	
Poscondición	El pedido se crea en el sistema y se devuelve la información del pedido creado	
Resumen	El usuario crea un nuevo pedido en el sistema proporcionando la información necesaria	
Tipo	Primario	

Caso de uso	**Obtener todos los pedidos**	**CU 013**
Actor(es)	Administrador	
Poscondición	Se obtiene la lista de todos los pedidos registrados en el sistema	
Resumen	El administrador solicita la lista completa de pedidos en el sistema	
Tipo	Primario	

Caso de uso	**Obtener pedido por ID**	**CU 014**
Actor(es)	Usuario, Administrador	
Poscondición	Se obtiene la información del pedido correspondiente al ID solicitado	
Resumen	El usuario o administrador solicita la información de un pedido específico mediante su ID	
Tipo	Primario	

Caso de uso	**Actualizar pedido por ID**	**CU 015**
Actor(es)	Administrador	
Poscondición	Se actualiza el estado de un pedido existente en el sistema	
Resumen	El administrador actualiza el estado de un pedido específico mediante su ID	
Tipo	Primario	

Caso de uso	Obtener pedidos por usuario	CU 016
Actor(es)	Usuario, Administrador	
Poscondición	Se obtiene la lista de pedidos realizados por un usuario específico	
Resumen	El usuario o administrador solicita la lista de pedidos de un usuario específico	
Tipo	Primario	

Caso de uso	Obtener pedidos por estado	CU 017
Actor(es)	Administrador	
Poscondición	Se obtiene la lista de pedidos según el estado solicitado	
Resumen	El administrador solicita la lista de pedidos filtrados por su estado	
Tipo	Primario	

Reviews

Caso de uso	Añadir reseña	CU 018
Actor(es)	Usuario	
Poscondición	La reseña se guarda en el sistema y se actualiza la puntuación promedio del libro	
Resumen	El usuario añade una reseña para un libro específico, lo que actualiza la puntuación promedio del libro	
Tipo	Primario	

Caso de uso	Obtener reseñas por ID de libro	CU 019
Actor(es)	Usuario, Administrador	
Poscondición	Se obtiene la lista de reseñas para un libro específico	
Resumen	El usuario o administrador solicita la lista de reseñas asociadas a un libro mediante su ID	
Tipo	Primario	

Authors

Caso de uso	Obtener todos los autores	CU 020
Actor(es)	Administrador, Usuario	
Poscondición	Se obtiene la lista completa de autores registrados en el sistema	
Resumen	El usuario o administrador solicita la lista completa de autores en el sistema	
Tipo	Primario	

Caso de uso	Obtener autores activos	CU 021
Actor(es)	Administrador, Usuario	
Poscondición	Se obtiene la lista de autores activos en el sistema	
Resumen	El usuario o administrador solicita la lista de autores que están activos en el sistema	
Tipo	Primario	

Caso de uso	Obtener autor por ID	CU 022
Actor(es)	Administrador, Usuario	
Poscondición	Se obtiene la información del autor correspondiente al ID solicitado	
Resumen	El usuario o administrador solicita la información de un autor específico mediante su ID	
Tipo	Primario	

Caso de uso	Añadir autor	CU 023
Actor(es)	Administrador	
Poscondición	Se añade un nuevo autor al sistema y se devuelve la información del autor añadido	
Resumen	El administrador añade un nuevo autor en el sistema proporcionando la información necesaria	
Tipo	Primario	

Caso de uso	Eliminar autor	CU 024
Actor(es)	Administrador	
Poscondición	Se elimina un autor existente del sistema	
Resumen	El administrador elimina un autor del sistema mediante su ID	
Tipo	Primario	

Caso de uso	Actualizar autor	CU 025
Actor(es)	Administrador	
Poscondición	Se actualiza la información de un autor existente en el sistema	
Resumen	El administrador actualiza los datos de un autor específico mediante su ID	
Tipo	Primario	

Users

Caso de uso	Iniciar sesión	CU 026
Actor(es)	Usuario	
Poscondición	El usuario se autentica en el sistema y se devuelve la información del usuario autenticado	
Resumen	El usuario ingresa sus credenciales para acceder al sistema	
Tipo	Primario	

Caso de uso	Obtener todos los usuarios	CU 027
Actor(es)	Administrador	
Poscondición	Se obtiene la lista completa de usuarios registrados en el sistema	
Resumen	El administrador solicita la lista completa de usuarios en el sistema	
Tipo	Primario	

Caso de uso	Crear usuario	CU 028
Actor(es)	Administrador	
Poscondición	Se añade un nuevo usuario al sistema y se devuelve la información del usuario añadido	
Resumen	El administrador crea un nuevo usuario en el sistema proporcionando la información necesaria	
Tipo	Primario	

Caso de uso	Actualizar usuario	CU 029
Actor(es)	Administrador	
Poscondición	Se actualiza la información de un usuario existente en el sistema	
Resumen	El administrador actualiza los datos de un usuario específico mediante su ID	
Tipo	Primario	

Caso de uso	Obtener pedidos por usuario	CU 030
Actor(es)	Administrador	
Poscondición	Se obtiene la lista de pedidos realizados por un usuario específico según su estado	
Resumen	El administrador solicita la lista de pedidos de un usuario específico filtrados por su estado	
Tipo	Primario	

Caso de uso	**Cambiar estado del usuario**	**CU 031**
Actor(es)	Administrador	
Poscondición	Se actualiza el estado de un usuario en el sistema	
Resumen	El administrador cambia el estado de un usuario específico mediante su ID	
Tipo	Primario	

Caso de uso	**Obtener top de libros vendidos**	**CU 032**
Actor(es)	Administrador	
Poscondición	Se obtiene la lista de los libros más vendidos en el sistema	
Resumen	El administrador solicita la lista de los libros más vendidos en el sistema	
Tipo	Primario	

Caso de uso	**Obtener usuario por ID**	**CU 033**
Actor(es)	Administrador	
Poscondición	Se obtiene la información del usuario correspondiente al ID solicitado	
Resumen	El administrador solicita la información de un usuario específico mediante su ID	
Tipo	Primario	

Diagrama de Despliegue

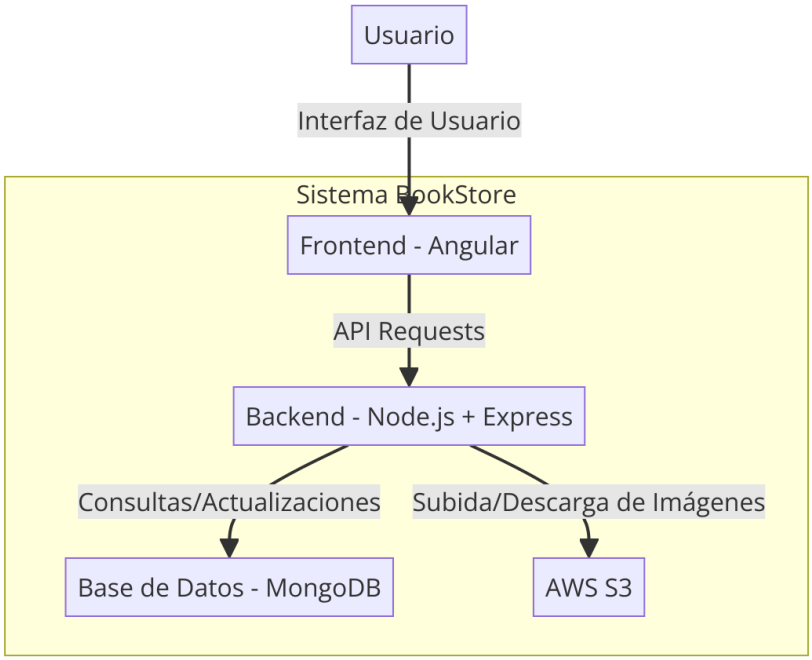
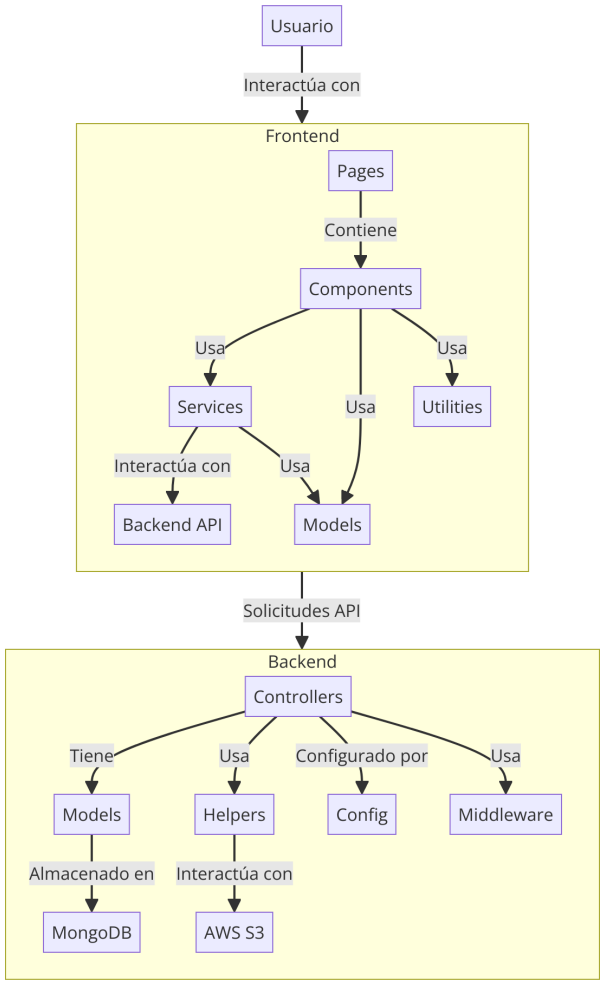


Diagrama de Paquetes



Organigrama

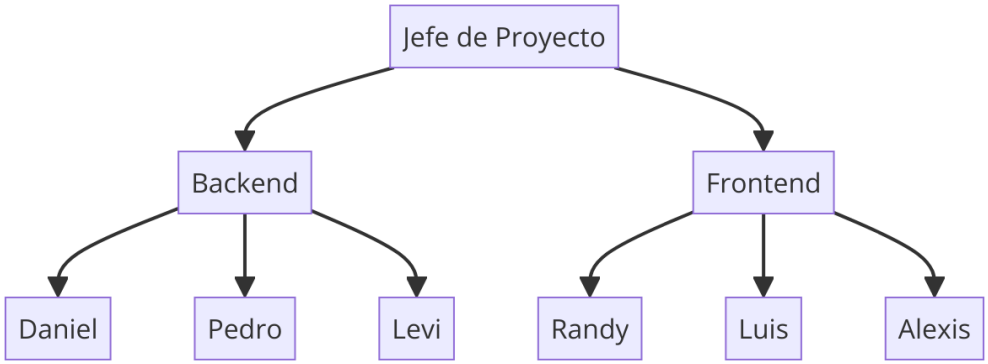
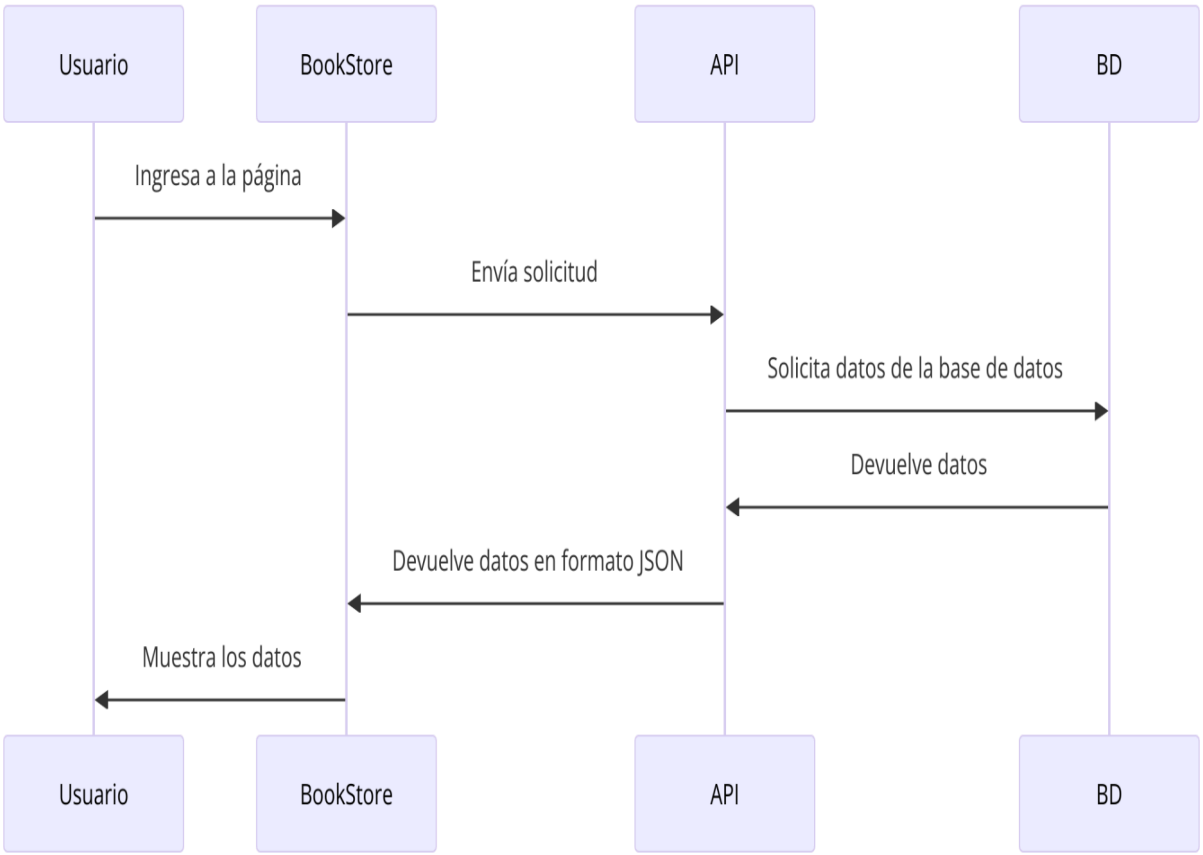


Diagrama de Comunicación

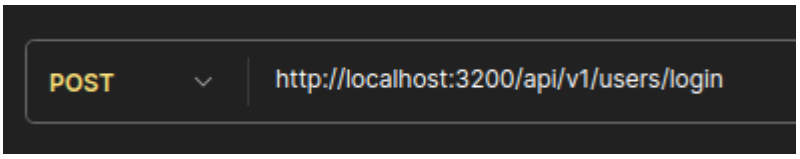


Métodos y funciones utilizados en el backend

Usuarios

login

Ruta



Body

Código

```
const login = async (req, res) => {
    const { email, password } = req.body;
    if (!password) {
        res.status(400).json({ message: 'la contraseña es requerida', 'ok': false});
        return;
    }
        let user = await User.findOne({ email });

        if (!user) {
            user = await User.findOne({ username: email });
            if (!user) {
                res.status(404).json({ message: 'No existe el usuario', 'ok': false});
                return
            }
            if (user.password !== password) {
                res.status(401).json({ message: 'Contraseña incorrecta', 'ok': false});
                return;
            }
            res.status(200).json({ user, 'ok': true });
            return;
        }

        if (user.password !== password) {
            res.status(401).json({ message: 'Contraseña incorrecta', 'ok': false});
            return;
        }
        res.status(200).json({ user, 'ok': true });

}
```

Se implementa una función login asincrónica que maneja solicitudes POST para autenticar a los usuarios.

Extracción de Datos: Extrae el email y password del cuerpo de la solicitud POST (req.body).

Validación de contraseña: Verifica si se proporcionó una contraseña. Si no, devuelve un error 400.

Búsqueda de Usuario: Busca un usuario en la base de datos por email. Si no encuentra al usuario por email, intenta buscarlo por username (que se asume que es el mismo que el email).

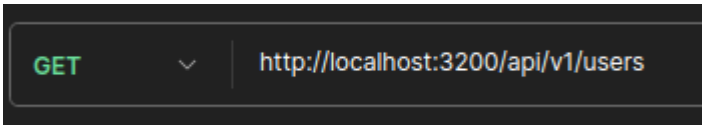
Manejo de Errores de Usuario: Si no encuentra ningún usuario con los datos proporcionados, devuelve un error 404 indicando que el usuario no existe.

Validación de contraseña: Si encuentra un usuario, verifica si la contraseña proporcionada coincide con la contraseña almacenada en la base de datos. Si no coincide, devuelve un error 401.

Respuesta exitosa: Si la contraseña coincide, devuelve un código de estado 200 junto con el objeto JSON que contiene la información del usuario autenticado y un campo ok establecido en true.

getAllUsers

Ruta



Código

```
const getAllUsers = async (req, res) => {
    const users = await User.find();
    if (users==null || users.length==0 || !users) {
        res.status(404).json({message: 'Users not found'});
        return;
    }
    res.json(users);
}
```

Se implementa una función asincrónica que maneja una solicitud GET para recuperar todos los usuarios de la base de datos.

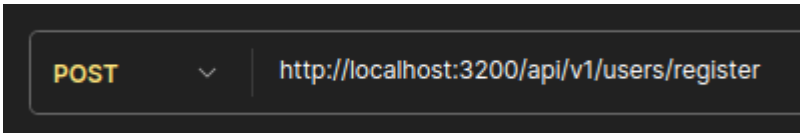
Búsqueda de Usuarios: Utiliza el método find del modelo User para buscar todos los usuarios en la base de datos de forma asincrónica y los almacena en la variable users.

Manejo de Errores: Si users es null, está vacío (users.length == 0), o es undefined (!users), devuelve un código de estado 404 (Not Found) junto con un mensaje JSON que indica que no se encontraron usuarios.

Respuesta exitosa: Si se encuentran usuarios, devuelve un objeto JSON con la lista de usuarios.

createUser

Ruta



Body

	Key	Value
☑	username	fiMd12
☑	password	15adf8
☑	complete_name	Fernanda Isabel Murcia Darodes
☑	email	fim@gmail.com
☑	phone	42698500
☑	address	4ta avenida Sur, Ciudad Real, Guatemala
☑	register_date	2024-06-16
☑	rol	ADMIN_ROLE
☑	photo	red.png ×
☑	payment_method	Efectivo

Código

```
const createUser = async (req, res) => {
    const user = new User(req.body);
    console.log(user);
    await user.save();
    res.json(user);
}
```

Se implementa la función createUser asincrónica que maneja una solicitud POST para crear un nuevo usuario en la base de datos.

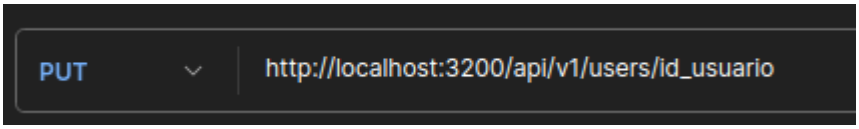
Creación de Usuario: Crea una nueva instancia del modelo User utilizando los datos proporcionados en el cuerpo de la solicitud (req.body). Almacena esta instancia en la variable user.

Guardado en la Base de Datos: Utiliza el método save de la instancia user para guardar el nuevo usuario en la base de datos de forma asincrónica.

Respuesta exitosa: Devuelve un objeto JSON con la información del usuario recién creado.

updateUser

Ruta



body

```
{
    "id": "667b15613331c02511c58c4b",
    "phone": 12569800
}
```

Código

```
const updateUser = async (req, res) => {
    const { id } = req.params;
    const user = await User.findByIdAndUpdate(id, req.body , {new: true});
    res.json(user);
}
```

Se implementa la función updateUser asincrónica que maneja una solicitud PUT para actualizar un usuario existente en la base de datos.

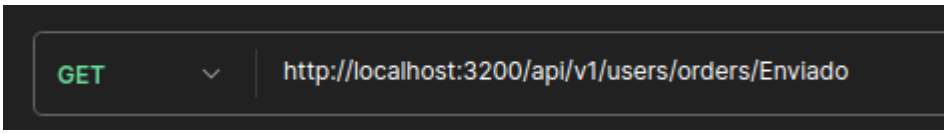
Extracción del ID del Usuario: Extrae el id de los parámetros de la solicitud (req.params).

Actualización del Usuario: Utiliza el método findByIdAndUpdate del modelo User para encontrar y actualizar el usuario con el id proporcionado. El segundo argumento (req.body) contiene los datos actualizados, y el tercer argumento{new: true}) asegura que la función retorna el documento actualizado.

Respuesta exitosa: Devuelve un objeto JSON con la información del usuario actualizado.

getOrders

Ruta



Código

```
const getOrders = async (req, res) => {
    const orders_users = await User.find({status: req.params.status});
    if (orders_users==null || orders_users.length==0 || !orders_users) {
        res.status(404).json({message: 'Orders not found'});
        return;
    }
    res.json(orders_users);
}
```

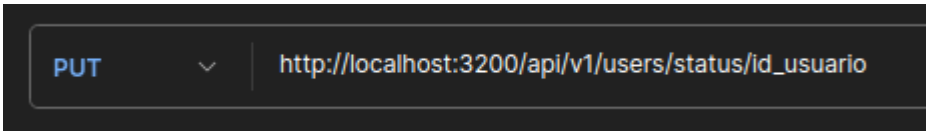
Se implementa la función getOrders asincrónica que maneja una solicitud GET para obtener los pedidos con un estado en específico.

Búsqueda de Usuarios por Estado: Utiliza el método find del modelo User para buscar todos los usuarios que tienen un estado específico (status) proporcionado en los parámetros de la solicitud (req.params.status).

Manejo de Errores: Si no se encuentran usuarios (orders_users es null, vacío o undefined), devuelve un código de estado 404 (Not Found) con un mensaje JSON indicando que no se encontraron pedidos.

Respuesta exitosa: Si se encuentran usuarios, devuelve un objeto JSON con la lista de usuarios que cumplen con el estado especificado.

changeStatus




```
const changeStatus = async (req, res) => {
    const { id } = req.params;
    const user = await User.findByIdAndUpdate (id, req.body , {new: true});
    res.json(user);
}
```

Se implementa la función changeStatus asincrónica que maneja una solicitud PUT para actualizar un estado.

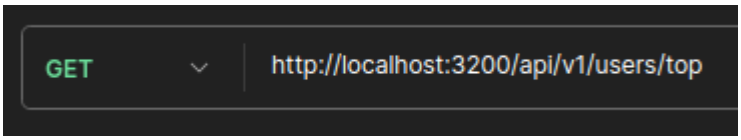
Extracción del ID del Usuario: Extrae el id de los parámetros de la solicitud (req.params).

Actualización del Usuario: Utiliza el método findByIdAndUpdate del modelo User para encontrar y actualizar el usuario con el id proporcionado. El segundo argumento (req.body) contiene los datos actualizados, y el tercer argumento{new: true}) asegura que la función retorna el documento actualizado.

Respuesta exitosa: Devuelve un objeto JSON con la información del usuario actualizado.

topBooks

Ruta




```
const topBooks = async (req, res) => {
    const books = await User.find().sort({books_sold: -1}).limit(15);
    if (books==null || books.length==0 || !books) {
        res.status(404).json({message: 'Books not found'});
        return;
    }
    res.json(books);
}
```

Se implementa la función topBooks asincrónica que maneja una solicitud GET para obtener el top de libros vendidos.

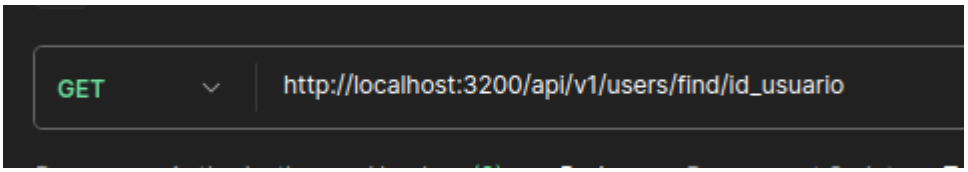
Búsqueda de Libros: Utiliza el método find del modelo User para buscar todos los usuarios y ordenarlos por la cantidad de libros vendidos (books_sold) en orden descendente. Limita el resultado a los 15 usuarios principales.

Manejo de Errores: Si no se encuentran usuarios (books es null, vacío o undefined), devuelve un código de estado 404 (Not Found) con un mensaje JSON indicando que no se encontraron libros.

Respuesta exitosa: Si se encuentran usuarios, devuelve un objeto JSON con la lista de los 15 usuarios principales según la cantidad de libros vendidos.

getUserById

Ruta



```
const getUserById = async (req, res) => {
    try {
        const user = await User.findById(req.params.id);
        if (!user) {
            res.status(404).json({ message: 'Usuario no encontrado' });
            return;
        }

        res.json(user);
    } catch (error) {
        res.status(500).json({ message: 'Error al obtener ', error: error.message });
    }
}
```

Se implementa la función getUserById asincrónica que maneja una solicitud GET para obtener un usuario por id.

Búsqueda de Usuario por ID: Intenta encontrar un usuario utilizando el método findById del modelo User con el id proporcionado en los parámetros de la solicitud (req.params.id).

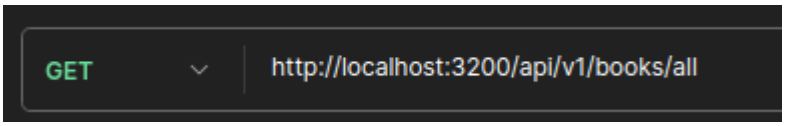
Manejo de Errores: Si no se encuentra un usuario, devuelve un código de estado 404 (Not Found) con un mensaje JSON indicando que el usuario no fue encontrado. Si ocurre un error durante la búsqueda, devuelve un código de estado 500 (Internal Server Error) con un mensaje JSON describiendo el error.

Respuesta exitosa: Si se encuentra el usuario, devuelve un objeto JSON con la información del usuario encontrado.

Libros

getAll

Ruta



Código

```
const getAll = async (req, res) => {
    const books = await Book.find();
    res.json(books);
}
```

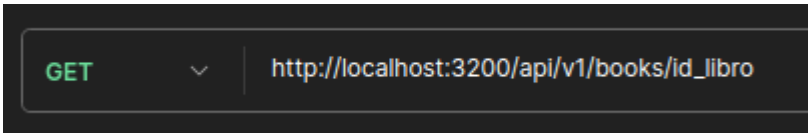
Se implementa la función getAll asincrónica que maneja una solicitud GET para recuperar todos los libros de la base de datos.

Búsqueda de Libros: Utiliza el método find del modelo Book para buscar todos los libros en la base de datos de forma asincrónica y los almacena en la variable books.

Respuesta exitosa: Devuelve un objeto JSON con la lista de libros encontrados.

getBookById

Ruta




```
const getBookById = async (req, res) => {
    const book = await Book.findById(req.params.id);
    res.json(book);
}
```

Se implementa la función getBookById asincrónica que maneja una solicitud GET para recuperar un libro específico de la base de datos utilizando su ID.

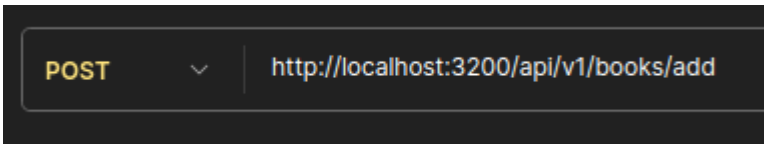
Extracción del ID del Libro: Extrae el id de los parámetros de la solicitud (req.params.id).

Búsqueda del Libro por ID: Utiliza el método findById del modelo Book para buscar el libro con el id proporcionado de forma asincrónica y lo almacena en la variable book.

Respuesta exitosa: Devuelve un objeto JSON con la información del libro encontrado.

addBook

Ruta



Código

```
const addBook = async (req, res) => {
    const { titulo, autor, descripcion, genero, fecha_publicacion, disponibilidad, cantidad_stock, puntuacion_promedio,
    const newBook = new Book(
        { titulo, autor, descripcion, genero, fecha_publicacion, disponibilidad,
            cantidad_stock, puntuacion_promedio, precio,  imagen_url
        });
    const bookSaved = await newBook.save();
    res.json({ message: 'Book added', value: bookSaved });
};
```

Se implementa la función addBook asincrónica que maneja una solicitud POST para agregar un nuevo libro a la base de datos.

Extracción de Datos del Cuerpo de la Solicitud: Extrae los datos del libro desde el cuerpo de la solicitud (req.body).

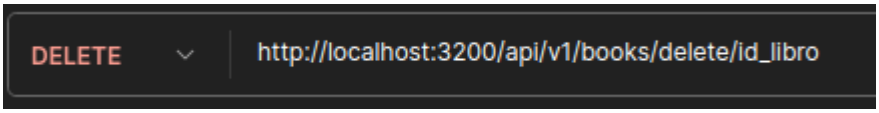
Creación de un Nuevo Objeto Libro: Crea una nueva instancia del modelo Book con los datos extraídos.

Guardado del Nuevo Libro: Guarda el nuevo libro en la base de datos utilizando el método save.

Respuesta exitosa: Devuelve un objeto JSON con un mensaje indicando que el libro ha sido agregado y con la información del libro guardado.

deleteBook

Ruta



```
const deleteBook = async (req, res) => {
    const bookDeleted = await Book.deleteOne({ _id: req.params.id });
    res.json({ deleted: !!bookDeleted, ok: true });
};
```

Se implementa la función deleteBook asincrónica que maneja una solicitud DELETE para eliminar un libro de la base de datos utilizando su ID.

Extracción del ID del Libro: Extrae el id de los parámetros de la solicitud (req.params.id).

Eliminación del Libro: Utiliza el método deleteOne del modelo Book para eliminar el libro con el id proporcionado de forma asincrónica y almacena el resultado en la variable bookDeleted.

Respuesta Exitosa: Devuelve un objeto JSON con una propiedad deleted que indica si el libro fue eliminado correctamente y una propiedad ok establecida en true.

updateBook

Ruta
http://localhost:3200/api/v1/books/update/:id

```
const updateBook = async (req, res) => {
    const updated = await Book.updateOne({ _id: req.params.id }, req.body);
    res.json({ updated: !!updated });
}
```

Esta función actualiza la información de un libro según el id que recibe.

getPuntuacionById

Ruta
http://localhost:3200/api/v1/books/puntuacion/:id

```
const getPuntuacionById = async (req, res) => {
    const book = await Book.findById(req.params.id);
    res.json(book.puntuacion_primedio);
};
```

Obtiene la puntuación de un libro según su Id.

getBooksByAuthorId

http://localhost:3200/api/v1/authors/find/:id

```
const getBooksByAuthorId = async (req, res) => {
    const books = await Book.find({ autor: req.params.id });
    res.json(books);
}
```

Obtiene los libros según el Id del autor que se envía.

getBooksByFilter

http://localhost:3200/api/v1/books/getBooksByFilter

```
const getBooksByFilter = async (req, res) => {
    const { filter, value } = req.params;
    let books;
    switch (filter) {
        case 'titulo':
            books = await Book.find({ titulo:  { $regex: value, $options: 'i' } });
            res.json(books);
            break;
        case 'autor':
            const authors = await Author.find({ name: { $regex: value, $options: 'i' } });
            books = await Book.find({ autor: { $in: authors.map(author => author._id) } });
            res.json(books);
            break;
        case 'genero':
            books = await Book.find({ genero:  { $regex: value, $options: 'i' } });
            res.json(books);
            break;
        case 'precio':
            books = await Book.find({ precio: value });
            res.json(books);
            break;
        case 'puntuacion':
            books = await Book.find({ puntuacion_promedio: value });
            res.json(books);
            break;
        default:
            res.status(400).json({ message: 'Invalid filter'});
    }
}
```

Esta función hace el filtro para buscar un libro según el título, autor, género, precio, o puntuación, de lo contrario devuelve un Invalid Filter

Autores

getAll

http://localhost:3200/api/v1/authors/all

```
const getAll = async (req, res) => {
    const authors = await Author.find();
    res.json(authors);
};
```

Esta función obtiene a todos los autores de las base de datos.

getAuthorsActive

http://localhost:3200/api/v1/authors/active

```
const getAuthorsActive = async (req, res) => {
    const authors = await Author.find({ active: true });
    res.json(authors);
};
```

Esta función verifica con una bandera de true o false si un Autor está en activo o no.

getAuthorById

http://localhost:3200/api/v1/authors/find/id

```
const getAuthorById = async (req, res, next) => {
    const author = await Author.findById(req.params.id);
    res.json(author);
    next()
};
```

Esta función busca a un autor según el _id

addAuthor

http://localhost:3200/api/v1/authors/add

```
const addAuthor = async (req, res) => {
    const { name, photo, biography, active } = req.body;
    const newAuthor = new Author({ name, photo, biography, active });
    const authorSaved = await newAuthor.save();
    res.json({ message: 'Author added', value: authorSaved });
};
```

En esta función se agrega a un autor, siguiendo la estructura de name, photo, biography y si esta activo o no.

deleteAuthor

http://localhost:3200/api/v1/authors/delete/id

```
const deleteAuthor = async (req, res) => {
    const authorDeleted = await Author.deleteOne({ _id: req.params.id });
    res.json({ deleted: !!authorDeleted });
};
```

Esta función elimina un autor según el Id que se le envía.

updateAuthor

http://localhost:3200/api/v1/authors/update/id

```
const updateAuthor = async (req, res) => {
    const { id } = req.params;
    const updated = await Author.findByIdAndUpdate(id, req.body, {new: true});
    res.json({ updated: !!updated });
}
```

Esta función actualiza la información de un autor según el id que reciba.

Pedidos

createOrder

http://localhost:3200/api/v1/pedidos/venta

```
const createOrder = async (req, res) => {
  try {
    const { id_usuario, libros, direccion_envio } = req.body;
    if (!id_usuario || !libros) {
      return res.status(400).json({
        ok: false,
        message: "Campos incompletos, debe enviar id_usuario y arreglo de libros",
      });
    }

    const validStock = await validateStock(libros);

    if (validStock.length > 0 || false) {
      return res.status(400).json({
        ok: false,
        message: "Sin stock suficiente de los siguientes libros",
        books: validStock,
      });
    }

    const currentDate = new Date();
    const formattedDate = currentDate.toLocaleDateString("es-ES");

    const total_price = await calculateTotalPrice(libros);

    if (!(await updateStock(libros))) {
      return res
        .status(500)
        .json({ ok: false, message: "Error al actualizar stock" });
    }


    // para los libros generados
    const libroIds = libros.map(libro => libro.libro._id);
    const nombres = libros.map(libro => libro.libro.titulo);

    await new Order({
      fecha_pedido: formattedDate,
      estado: "En proceso",
      precio_total: total_price,
      id_usuario,
      direccion_envio,
      libros: libros,
      metodo_pago: "Efectivo",
    }).save();

    res.status(200).json({ ok: true, message: "Pedido realizado exitosamente" , libroIds, nombres});
  } catch (error) {
    res
      .status(500)
      .json({
      ok: false, message: "Error al crear el pedido",error: error.message});
  }
};
```

getOrders

```
const getOrders = async (req, res) => {
  try {
    const orders = await Order.find();
    res.status(200).json(orders);
  } catch (error) {
    res
      .status(500)
      .json({
        ok: false,
        message: "Error al obtener los pedidos",
        error: error.message,
      });
  }
};
```

getOrderById

```
const getOrderById = async (req, res) => {
  try {
    const order = await Order.findById(req.params.id_pedido);

    if (!order) {
      res
        .status(404)
        .json({ ok: false, message: "Pedido no encontrado" });
    }

    res.status(200).json(order);
  } catch (error) {
    res
      .status(400)
      .json({
        ok: false,
        message: "Error al obtener el id del pedido",
        error: error.message });
  }
};
```

updateOrderById

```
const updateOrderById = async (req, res) => {
  try {
    const updatedOrder = await Order.findByIdAndUpdate(
      req.params.id_pedido,
      { estado: req.body.estado },
      { new: true }
    );

    if (!updatedOrder) {
      res
        .status(404)
        .json({
          ok: false,
          message: "Pedido no encontrado" });
    }

    res.status(200).json({ ok: true, message: "Pedido actualizado correctamente" });
  } catch (error) {
    res
      .status(500)
      .json({
        ok: false,
        message: "Error al actualizar el estado del pedido" });
  }
};
```

getOrdersByUser

```
const getOrdersByUser = async (req, res) => {
  try {
    const user = await Users.findById(req.params.id_usuario);

    if (!user) {
      res
        .status(404)
        .json({ ok: false, message: "Usuario no encontrado" });
    }

    const orders = await Order.find({ id_usuario: req.params.id_usuario });

    res.status(200).json(orders);
  } catch (error) {
    res
      .status(500)
      .json({
        ok: false,
        message: "Error al obtener los pedidos del usuario",
        error: error.message
      });
  }
};
```

getOrdersByStatus

```
const getOrdersByStatus = async(req, res)=>{
  try {

    const orders = await Order.find({estado: req.params.estado});

    res
      .status(200)
      .json({
        orders
      })

  } catch (error) {
    res
      .status(500)
      .json({
        ok: false,
        message: "Error al obtener los pedidos según el estado",
        error: error.message
      });
  }
}
```

Imágenes

addImage

```
const addImage = async (req, res) => {
    try {
        const fileName = await uploadFile(req.files
            , ['png', 'jpg', 'jpeg', 'gif']
            , 'images');
        res.json({fileName});
    } catch(error) {
        res.status(400).json({ error });
    }
}
```

```
const uploadFile = (files, validExtensions = ['png', 'jpg', 'jpeg', 'gif'], folder = '') => {
    return new Promise((resolve, reject) => {
        const { file } = files;
        const cutName = file.name.split('.');
        const extension = cutName[cutName.length - 1];

        if (!validExtensions.includes(extension)) {
            return reject(`The extension ${extension} is not allowed, valid extensions are: ${validExtensions}`);
        }

        const temporalName = uuidv4() + '.' + extension;
        const stream = fs.createReadStream(file.tempFilePath);

        uploadFileToS3(temporalName, extension, stream)
            .then((result) => {
                resolve(result);
            })
            .catch((err) => {
                reject(err);
            });

    });
}
```

updateImage

```
const updateImage = async (req, res) => {
    try {
        const { image } = req.params;
        const fileName = await updateFile(req.files, image
            , ['png', 'jpg', 'jpeg', 'gif']
            , 'images');
        res.json({fileName});
    } catch(error) {
        res.status(400).json({ error });
    }
}
```

```
const updateFile = (files, fileName, validExtensions = ['png', 'jpg', 'jpeg', 'gif'], folder = '') => {
    return new Promise((resolve, reject) => {
        const { file } = files;
        const cutName = file.name.split('.');
        const extension = cutName[cutName.length - 1];

        if (!validExtensions.includes(extension)) {
            return reject(`The extension ${extension} is not allowed, valid extensions are: ${validExtensions}`);
        }

        const temporalName = fileName.split('.')[0] + '.' + extension;

        deleteFileS3(fileName)
            .then(deleteResult => {
                if (!deleteResult) {
                    return reject(`Error deleting image`);
                }

                const stream = fs.createReadStream(file.tempFilePath);

                uploadFileToS3(temporalName, extension, stream)
                    .then(result => {
                        resolve(result);
                    })
                    .catch(err => {
                        reject(err);
                    });
            })
            .catch(err => {
                reject(err);
            });


    });
}
```

```
async function uploadFileToS3(name, type, file){

    const uploadParam = {
        Bucket: AWS_BUCKET_NAME,
        Key: `${name}`,
        Body: file,
        ContentEncoding: 'base64',
        ContentType: `image/${type}`
    }
    const command = new PutObjectCommand(uploadParam)
    await client.send(command)
    return name;

}
```

```
async function deleteFileS3(name){

    try {
        const deleteParam = {
            Bucket: AWS_BUCKET_NAME,
            Key: `${name}`
        }
        const command = new DeleteObjectCommand(deleteParam)
        await client.send(command)
        return true;
    } catch (error) {
        return false
    }

}
```

```
getImage


const getImage = (req, res) => {
    const { image } = req.params;

    const imageUrl = `https://${AWS_BUCKET_NAME}.s3.amazonaws.com/${image}`;
    res.redirect(imageUrl);
}
```

Routes

```
const express = require('express');
const AuthorsController = require('../controllers/authors.controller');

const router = express.Router();
const api = '/api/v1/authors';

router.post(`${api}/add`, AuthorsController.addAuthor);
router.get(`${api}/all`, AuthorsController.getAll);
router.get(`${api}/active`, AuthorsController.getAuthorsActive);
router.get(`${api}/find/:id`, AuthorsController.getAuthorById);
router.delete(`${api}/delete/:id`, AuthorsController.deleteAuthor);
router.put(`${api}/update/:id`, AuthorsController.updateAuthor);

module.exports = router;
```

```
const express = require('express')
const BooksController = require('../controllers/books.controller');

const router = express.Router();
const api = '/api/v1/books';

router.get(`${api}/all`, BooksController.getAll);
router.get(`${api}/:id`, BooksController.getBookById);
router.post(`${api}/add`, BooksController.addBook);
router.delete(`${api}/delete/:id`, BooksController.deleteBook);
router.put(`${api}/update/:id`, BooksController.updateBook);
router.get(`${api}/puntuacion/:id`, BooksController.getPuntuacionById);
router.get(`${api}/author/:id`, BooksController.getBooksByAuthorId);
router.get(`${api}/filter/:filter/:value`, BooksController.getBooksByFilter);

module.exports = router;
```

```
const express = require('express');
const OrdersController = require('../controllers/sales.controller')

const router = express.Router();
const api = '/api/v1/pedidos';

router.post(`${api}/venta`, OrdersController.createOrder);
router.get(`${api}/getTodosPedidos`, OrdersController.getOrders);
router.get(`${api}/getPedidosById/:id_pedido`, OrdersController.getOrderById);
router.put(`${api}/updatePedidoById/:id_pedido`, OrdersController.updateOrderById);
router.get(`${api}/getPedidosUsuario/:id_usuario`, OrdersController.getOrdersByUser);
router.get(`${api}/getPedidosByState/:estado`, OrdersController.getOrdersByStatus);

module.exports = router;
```

```
const express = require('express')
const ReviewsController = require('../controllers/reviews.controller');

const router = express.Router();
const api = '/api/v1/reviews';

router.get(`${api}/:id`, ReviewsController.getReviewsByBookId);
router.post(`${api}/add`, ReviewsController.addReview);

module.exports = router;
```

```
const { Router } = require('express');
const router = Router();
const UploadsController = require('../controllers/uploads.controller');
const { validateFile } = require('../middlewares/validateFile');

const api = '/api/v1/upload';

router.post(`${api}/add`, validateFile, UploadsController.addImage);
router.get(`${api}/find/:image`, UploadsController.getImage);
router.put(`${api}/update/:image`, validateFile, UploadsController.updateImage);


module.exports = router;
```

```
const express = require('express');
const User = require('../models/Users');
const UserController = require('../controllers/users.controller');

const router = express.Router();
const api = '/api/v1/users';

router.get(api, UserController.getAllUsers);
router.get(`${api}/find/:id`, UserController.getUserById);
router.post(`${api}/register`, UserController.createUser);
router.put(`${api}/:id`, UserController.updateUser);
router.get(`${api}/orders/:status`, UserController.getOrders);
router.put(`${api}/status/:id`, UserController.changeStatus);
router.get(`${api}/top`, UserController.topBooks);
router.post(`${api}/login`, UserController.login);


module.exports = router;
```

```
const express = require('express');
const app = express();
const cors = require('cors');
const fileUpload = require('express-fileupload');
//Routes
const ordersRoutes = require('./routes/orders.routes');
const authorsRoutes = require('./routes/authors.routes');
const uploadRoutes = require('./routes/upload.routes');
const usersRoutes = require('./routes/users.routes');
const booksRoutes = require('./routes/books.routes');
const reviewRoutes = require('./routes/reviews.routes');

app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cors());
app.use(fileUpload({
    useTempFiles : true,
    tempFileDir : '/tmp/',
    createParentPath: true
}));
app.use('/images', express.static('src/uploads/images'));


app.get('/',(req,res)=>{
    res.send('Hello from Backend API')
});

app.use(ordersRoutes);
app.use(usersRoutes);
app.use(authorsRoutes);
app.use(uploadRoutes);
app.use(reviewRoutes);
app.use(booksRoutes);

module.exports = app;
```

```
// creando api rest con express
const app = require('./app');
const {connect} = require("./configs/database.configs");
require('dotenv').config();
connect();

const PORT = process.env.APP_PORT || 3200;

app.listen(PORT, ()=>{
    console.log(`Servidor iniciado en el puerto ${PORT}`);
})
```

```
const mongoose = require('mongoose');
require('dotenv').config();

const host = process.env.DATABASE_HOST;
const port = process.env.DATABASE_PORT;
const database = process.env.DATABASE_NAME;

async function connect() {
    try {
        await mongoose.connect(`mongodb://${host}:${port}/${database}`, {
            family: 4
        });
        console.log('Conectado a la base de datos: '+ database);
    } catch (error) {
        console.error('Error de conexion', error);
    }
}

module.exports = { connect };
```