UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS
LABORATORIO SISTEMAS DE BASES DE DATOS 2
SECCIÓN P
VACACIONES DE JUNIO 2024

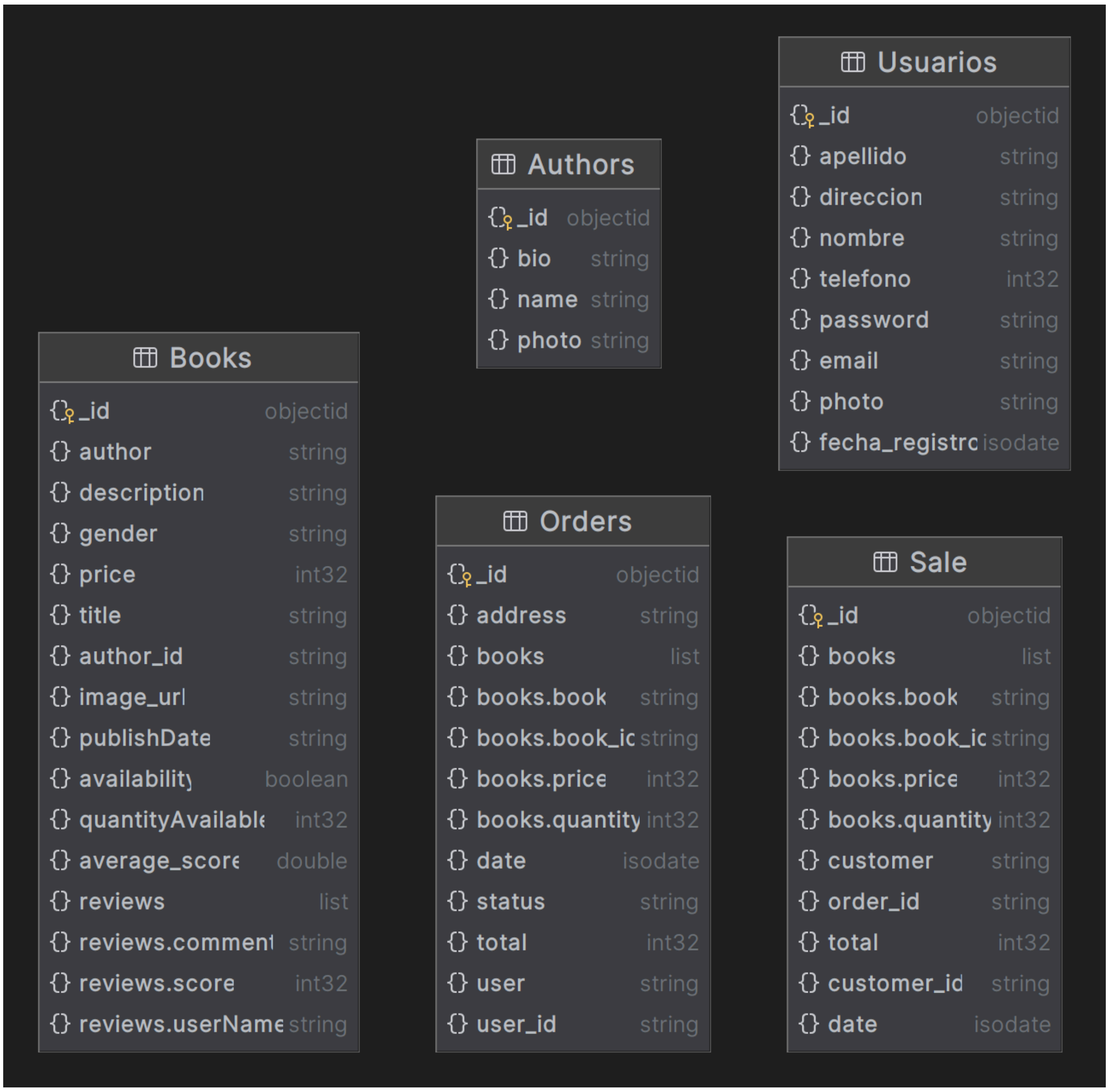
Proyecto 2

Integrantes Grupo 3:

Nombre	Carné
Cristian Daniel Pereira Tezagüic	202010893
Juan Josue Zuleta Beb	202006353
Oward Francisco Alberí Sian Solis	201901807
Jonatan Leonel Garcia Arana	202000424
Dayana Alejandra Reyes Rodríguez	202002364
Christopher Ivan Monterroso Alegria	201902363

GUATEMALA, 27 DE JUNIO DE 2024

Esquema Base de datos



Usuarios

_id	objectid
apellido	string
direccion	string
nombre	string
telefono	int32
password	string
email	string
photo	string
fecha_registrc	isodate

Authors

_id	objectid
bio	string
name	string
photo	string

Books

_id	objectid
author	string
description	string
gender	string
price	int32
title	string
author_id	string
image_url	string
publishDate	string
availabilit	boolean
quantityAvailabl	int32
average_scor	double
reviews	list
reviews.commen	string
reviews.score	int32
reviews.userNam	string

Orders

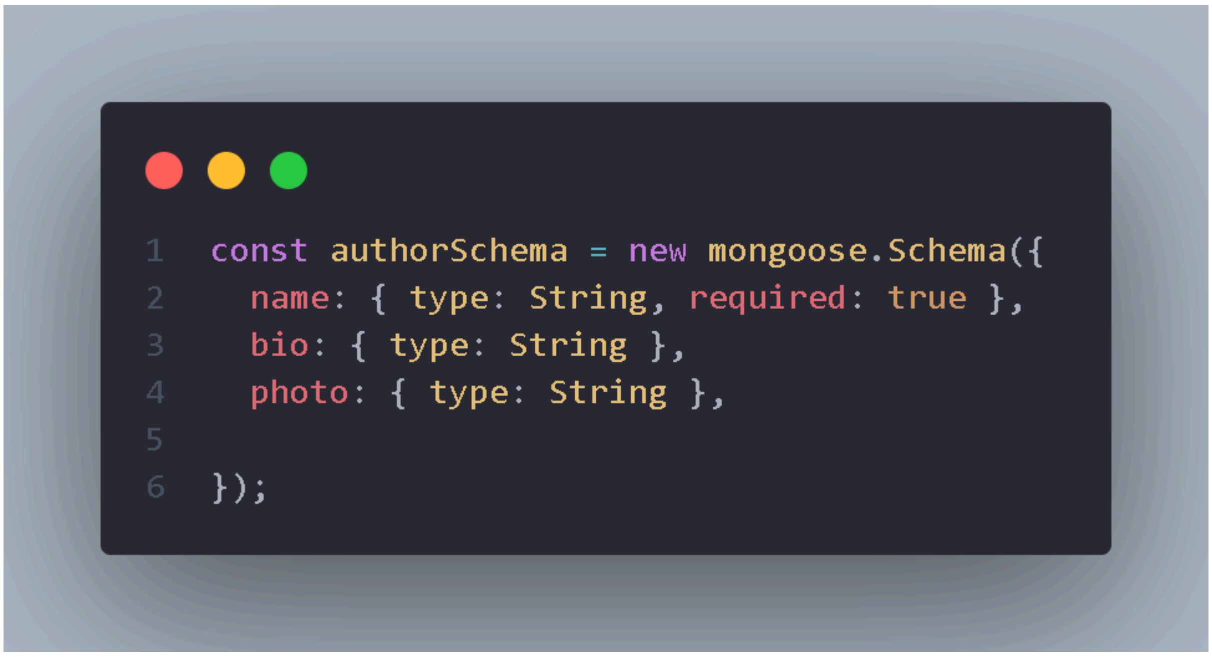
_id	objectid
address	string
books	list
books.book	string
books.book_ic	string
books.price	int32
books.quantity	int32
date	isodate
status	string
total	int32
user	string
user_id	string

Sale

_id	objectid
books	list
books.book	string
books.book_ic	string
books.price	int32
books.quantity	int32
customer	string
order_id	string
total	int32
customer_id	string
date	isodate

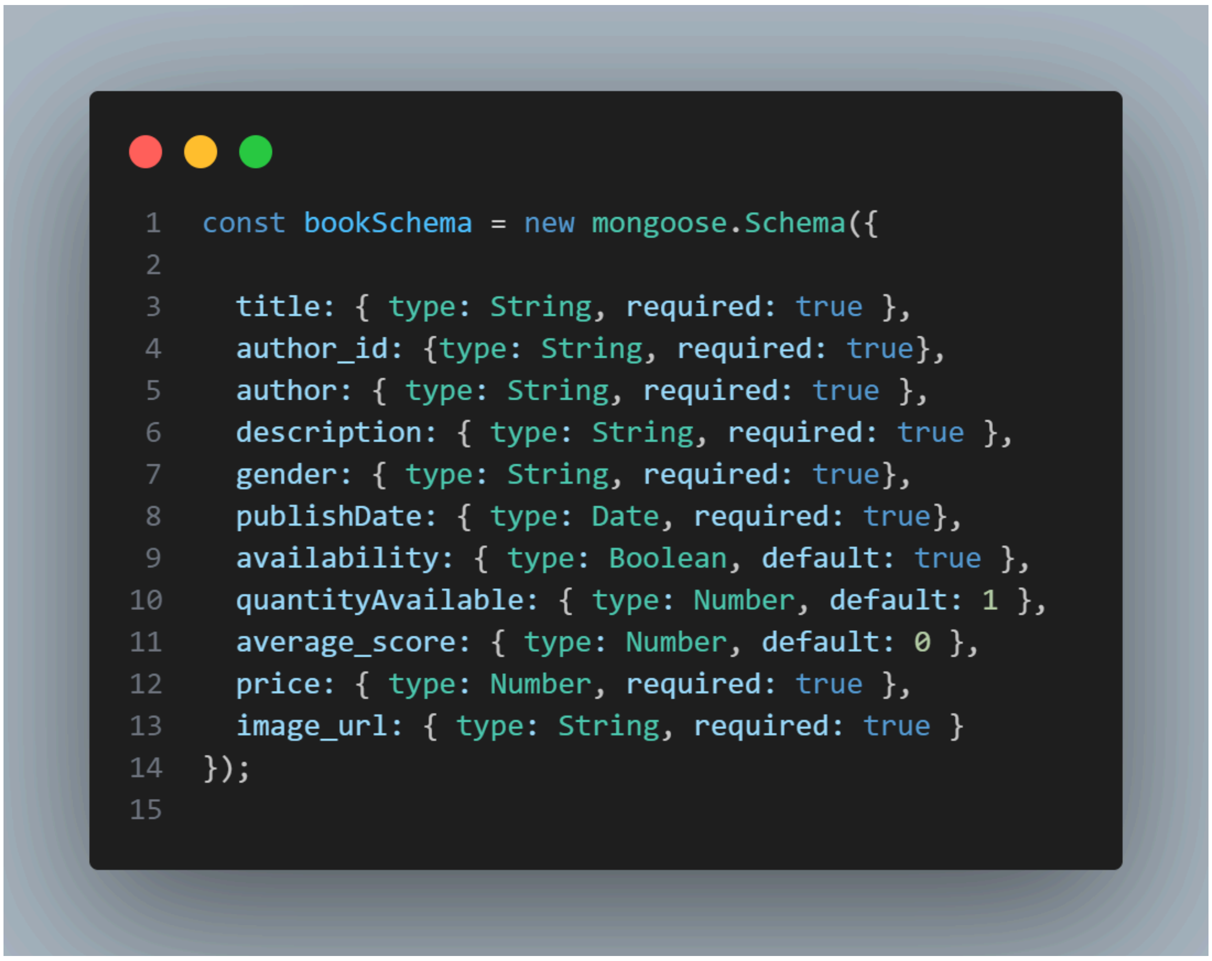
Estructuras

Estructura Autor



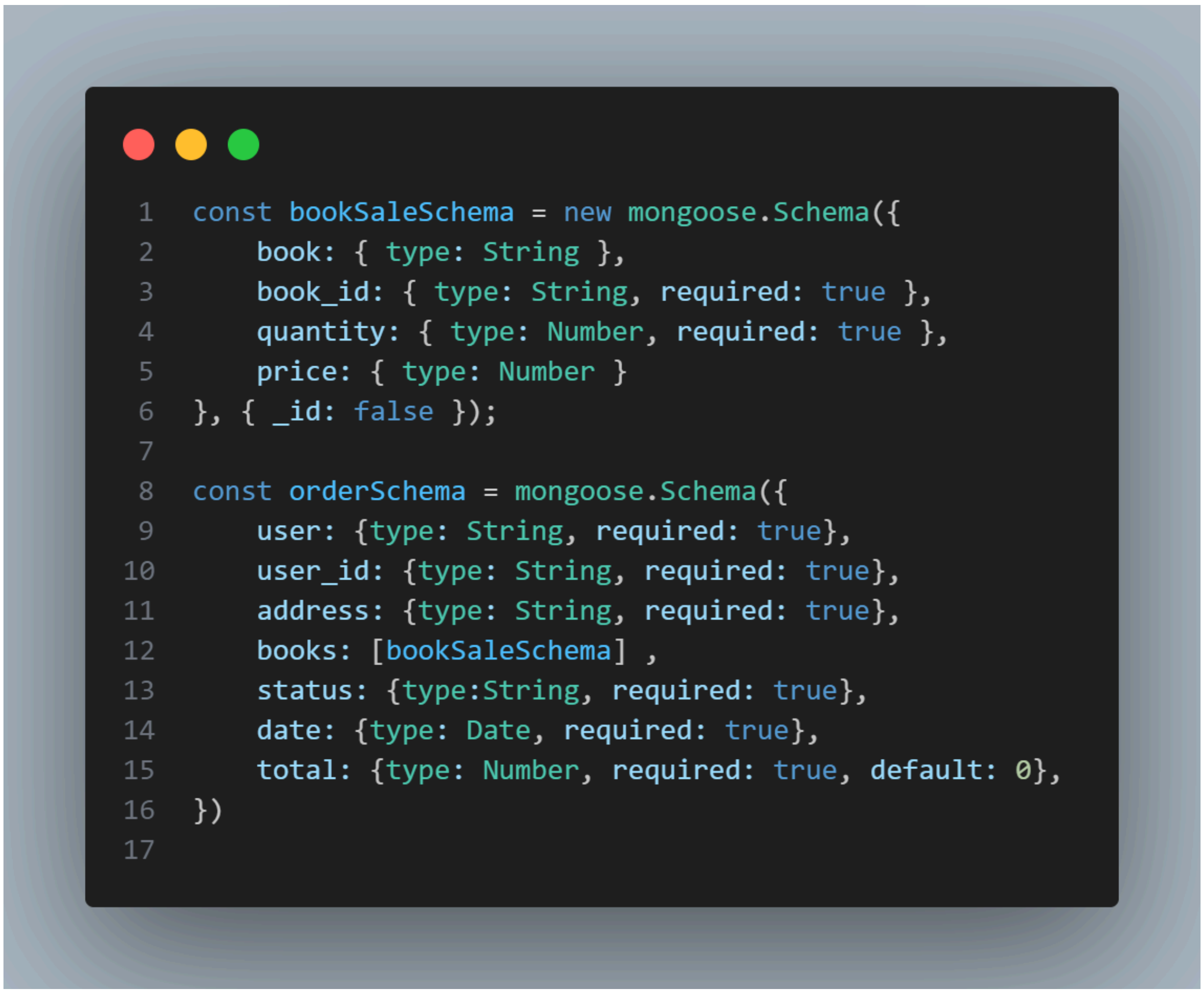
```
const authorSchema = new mongoose.Schema({
  name: { type: String, required: true },
  bio: { type: String },
  photo: { type: String },

});
```

Estructura Book



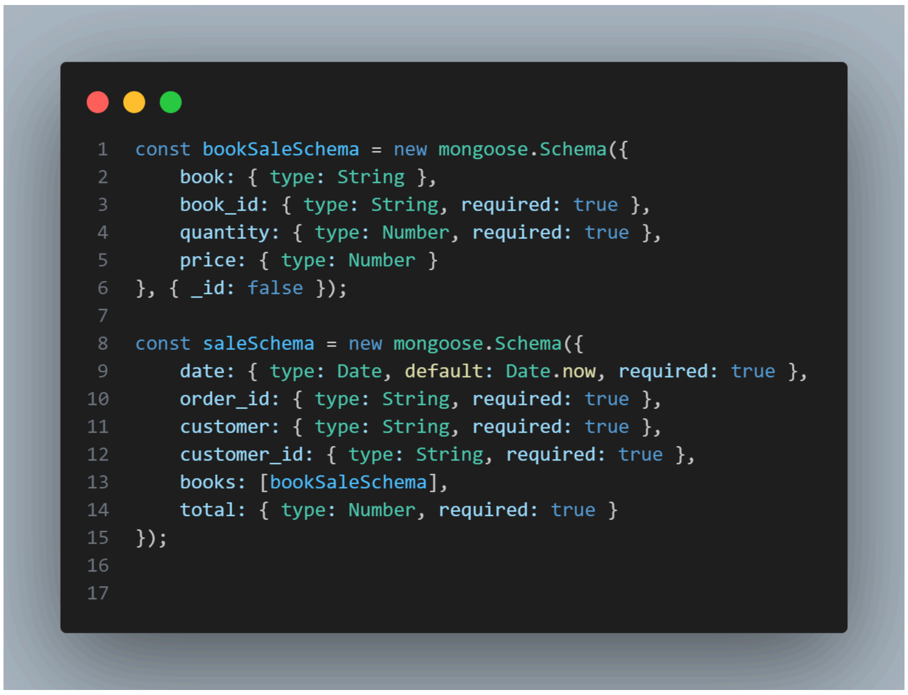
```
const bookSchema = new mongoose.Schema({

  title: { type: String, required: true },
  author_id: {type: String, required: true},
  author: { type: String, required: true },
  description: { type: String, required: true },
  gender: { type: String, required: true},
  publishDate: { type: Date, required: true},
  availability: { type: Boolean, default: true },
  quantityAvailable: { type: Number, default: 1 },
  average_score: { type: Number, default: 0 },
  price: { type: Number, required: true },
  image_url: { type: String, required: true }
});

```

Estructura Order



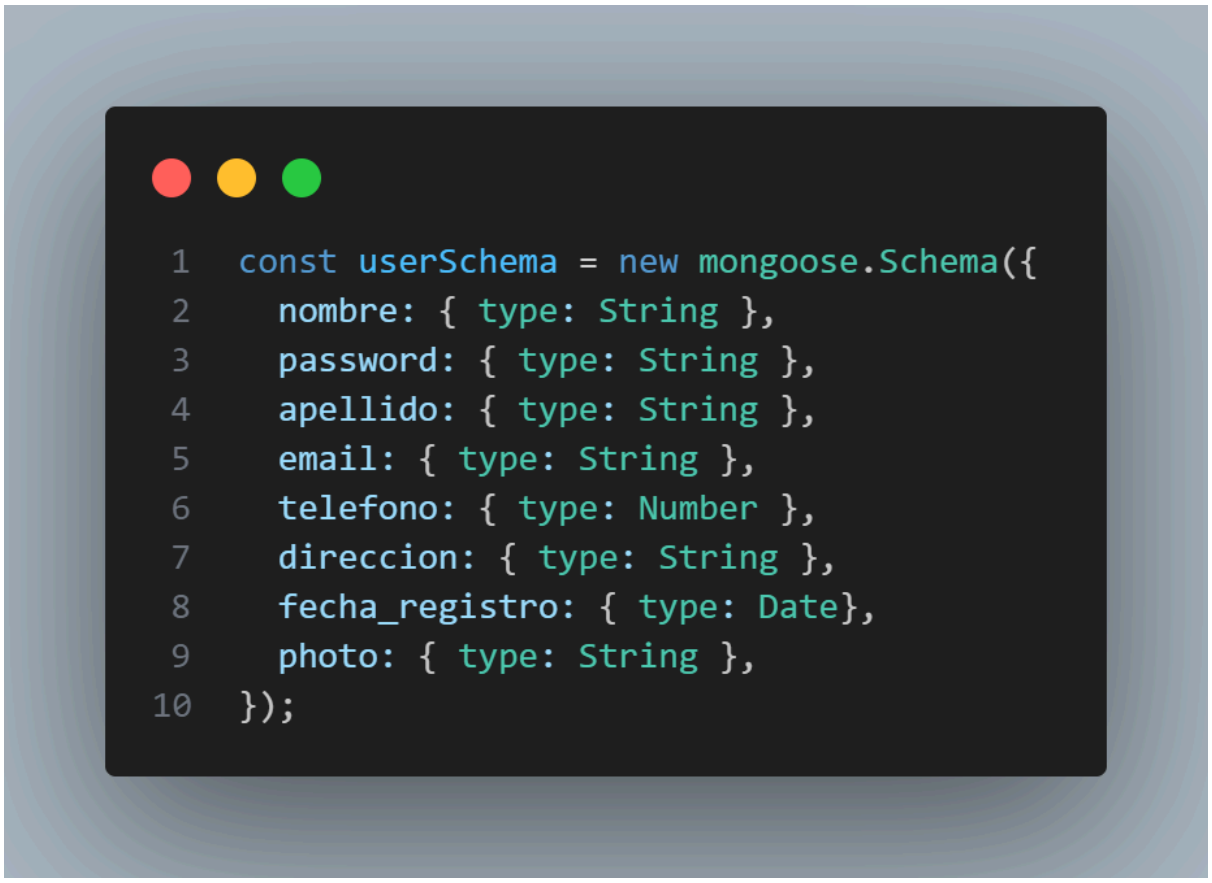
```
const bookSaleSchema = new mongoose.Schema({
    book: { type: String },
    book_id: { type: String, required: true },
    quantity: { type: Number, required: true },
    price: { type: Number }
}, { _id: false });

const orderSchema = mongoose.Schema({
    user: {type: String, required: true},
    user_id: {type: String, required: true},
    address: {type: String, required: true},
    books: [bookSaleSchema] ,
    status: {type:String, required: true},
    date: {type: Date, required: true},
    total: {type: Number, required: true, default: 0},
})

```

Estructura Sale



```
const bookSaleSchema = new mongoose.Schema({
    book: { type: String },
    book_id: { type: String, required: true },
    quantity: { type: Number, required: true },
    price: { type: Number }
}, { _id: false });

const saleSchema = new mongoose.Schema({
    date: { type: Date, default: Date.now, required: true },
    order_id: { type: String, required: true },
    customer: { type: String, required: true },
    customer_id: { type: String, required: true },
    books: [bookSaleSchema],
    total: { type: Number, required: true }
});


```

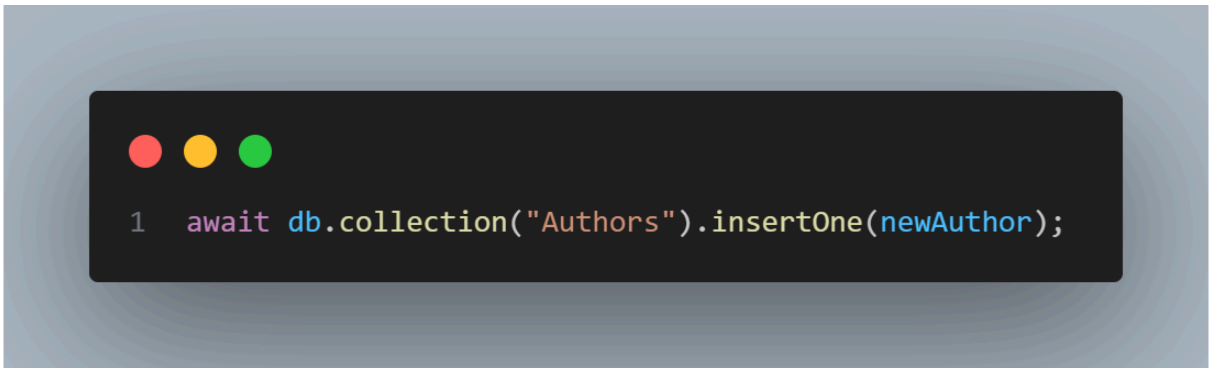
Estructura Users



```javascript
const userSchema = new mongoose.Schema({
  nombre: { type: String },
  password: { type: String },
  apellido: { type: String },
  email: { type: String },
  telefono: { type: Number },
  direccion: { type: String },
  fecha_registro: { type: Date},
  photo: { type: String },
});
```

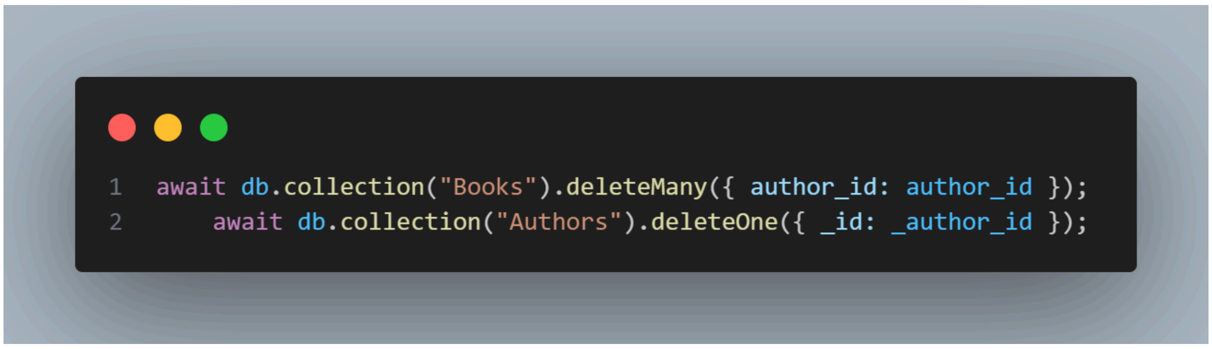
Operaciones de Autor

Creación de un nuevo autor



```javascript
await db.collection("Authors").insertOne(newAuthor);
```

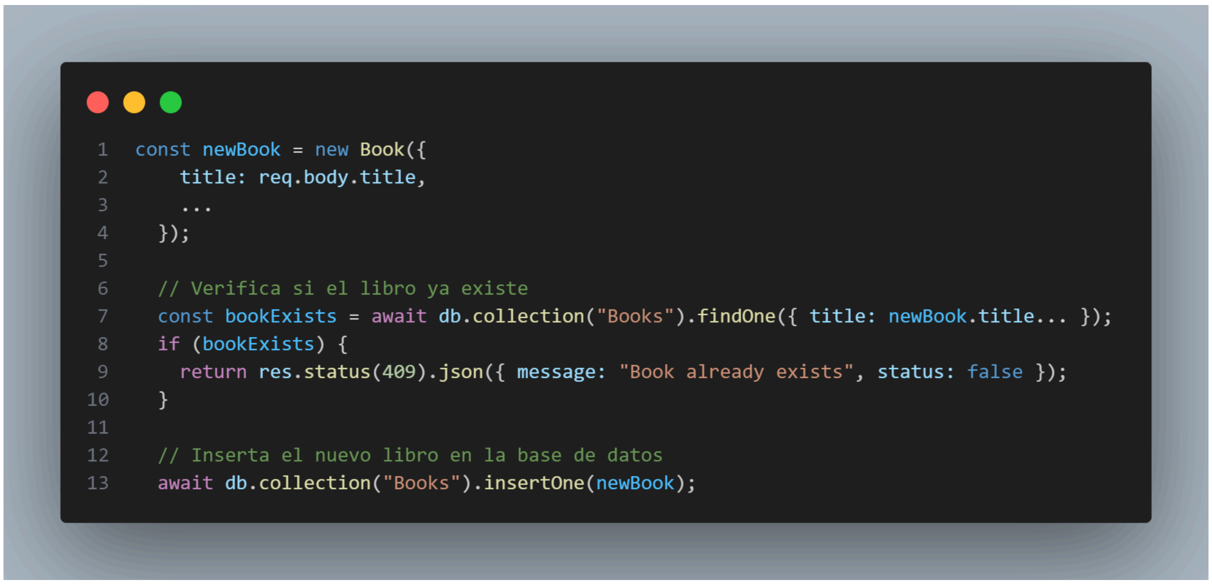
Eliminación de un Autor



```
await db.collection("Books").deleteMany({ author_id: author_id });
    await db.collection("Authors").deleteOne({ _id: _author_id });
```

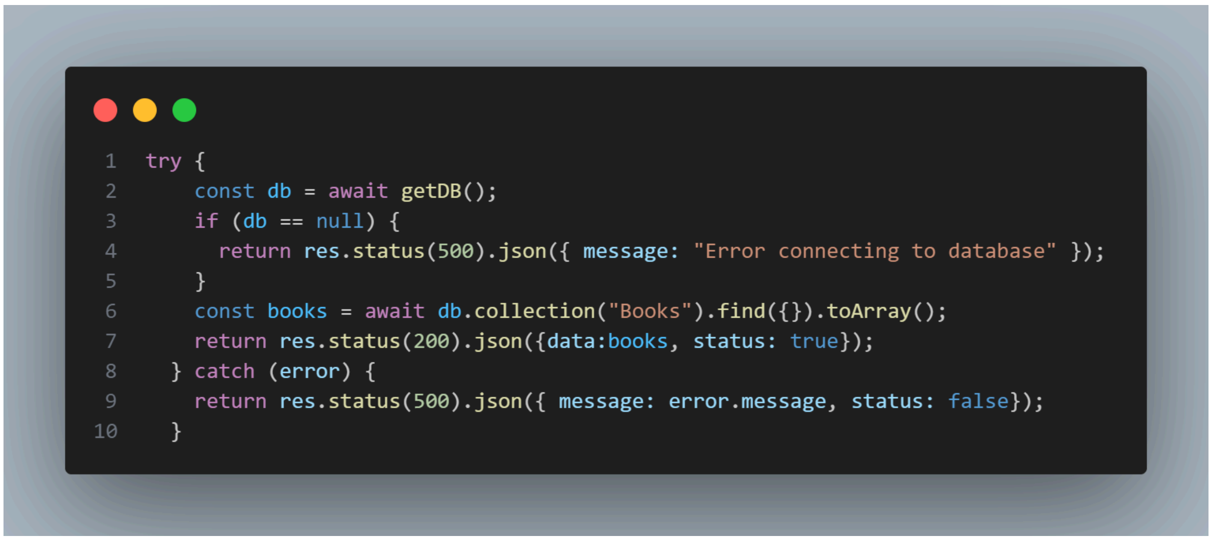
Operaciones de Libro

Creación de un nuevo Libro



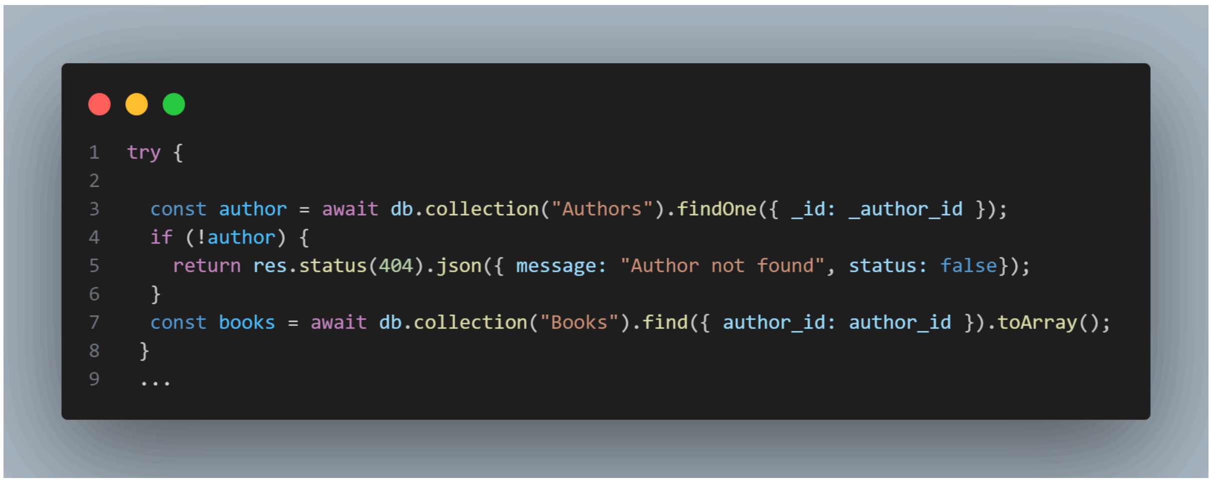
```
const newBook = new Book({
    title: req.body.title,
    ...
  });

  // Verifica si el libro ya existe
  const bookExists = await db.collection("Books").findOne({ title: newBook.title... });
  if (bookExists) {
    return res.status(409).json({ message: "Book already exists", status: false });
  }

  // Inserta el nuevo libro en la base de datos
  await db.collection("Books").insertOne(newBook);
```

Obtener Libros



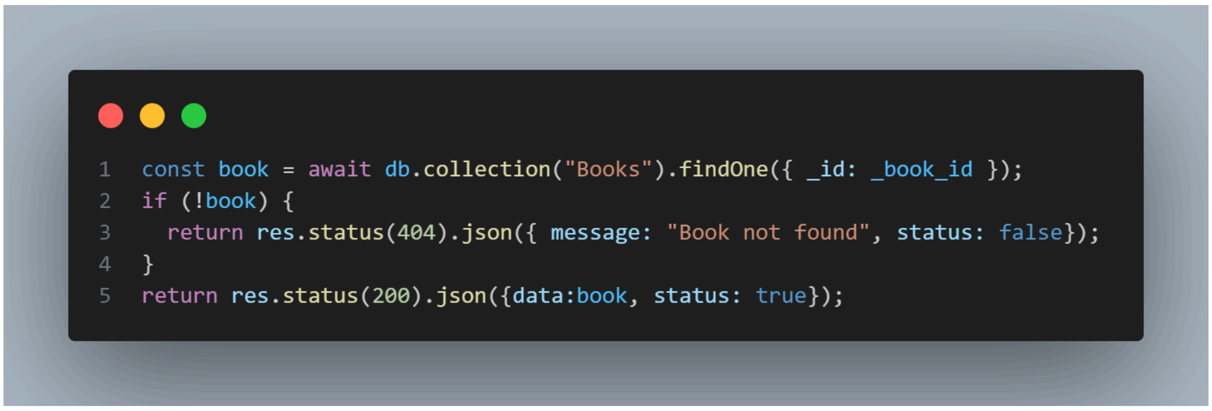
```
try {
    const db = await getDB();
    if (db == null) {
      return res.status(500).json({ message: "Error connecting to database" });
    }
    const books = await db.collection("Books").find({}).toArray();
    return res.status(200).json({data:books, status: true});
  } catch (error) {
    return res.status(500).json({ message: error.message, status: false});
  }
```

Obtener libros por autor



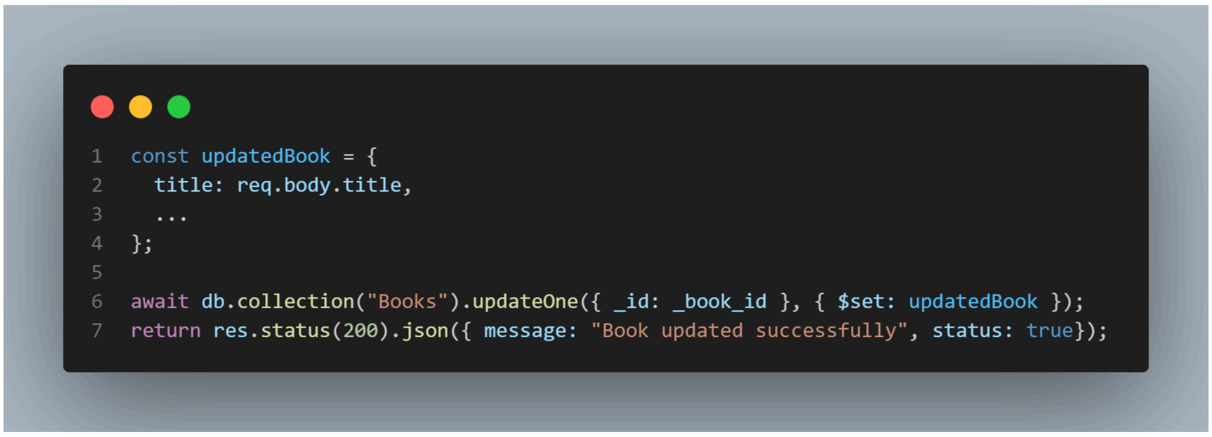
```
try {

  const author = await db.collection("Authors").findOne({ _id: _author_id });
  if (!author) {
    return res.status(404).json({ message: "Author not found", status: false});
  }
  const books = await db.collection("Books").find({ author_id: author_id }).toArray();
 }
 ...
```

Obtener libro por ID



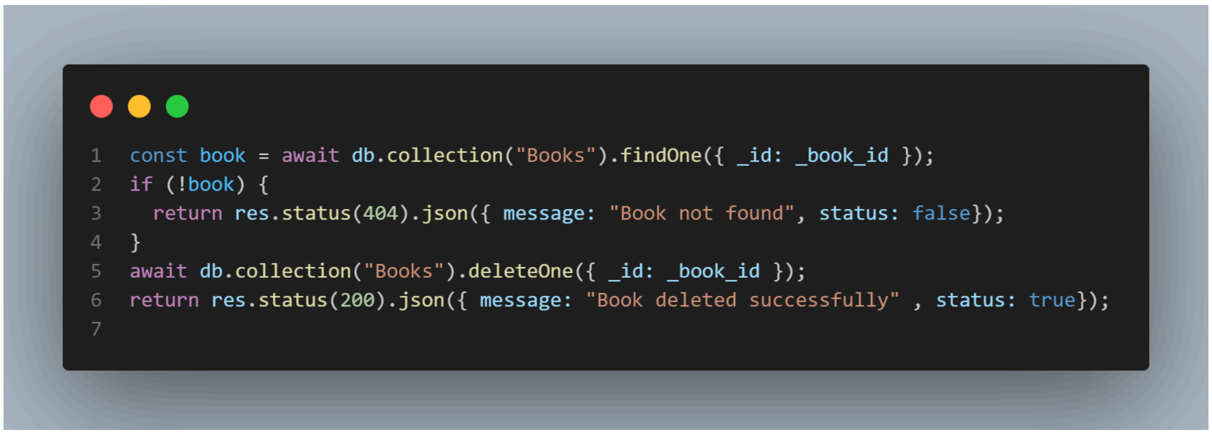
```
const book = await db.collection("Books").findOne({ _id: _book_id });
if (!book) {
  return res.status(404).json({ message: "Book not found", status: false});
}
return res.status(200).json({data:book, status: true});
```

Actualizar libro



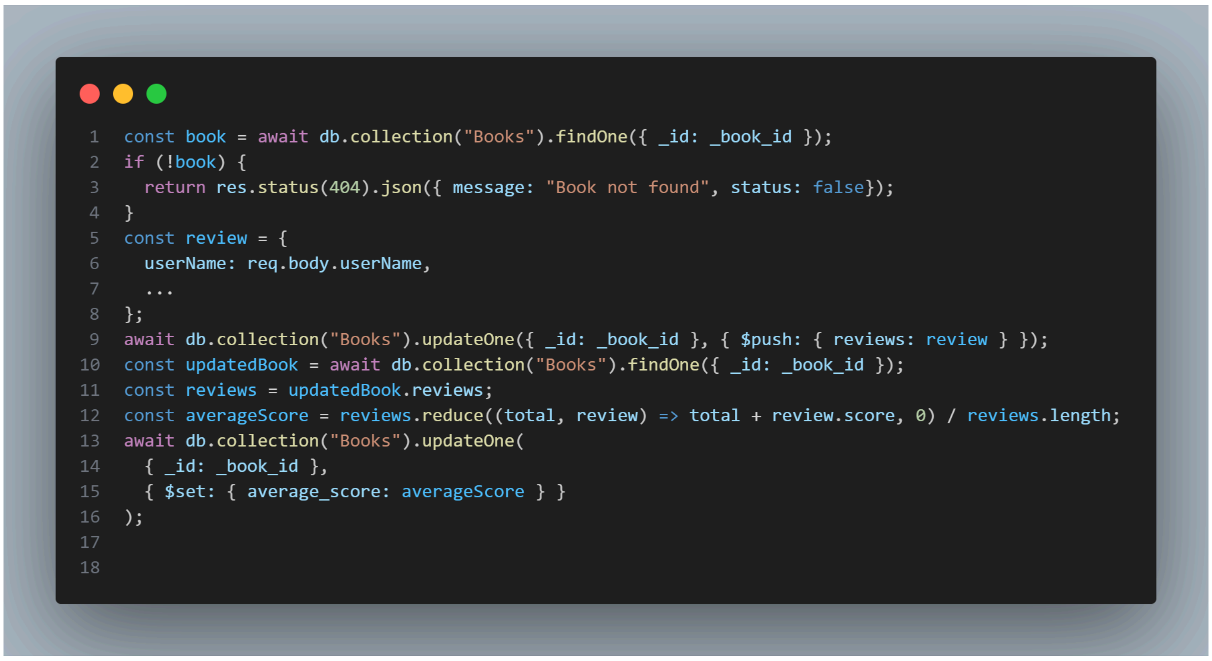
```
const updatedBook = {
  title: req.body.title,
  ...
};

await db.collection("Books").updateOne({ _id: _book_id }, { $set: updatedBook });
return res.status(200).json({ message: "Book updated successfully", status: true});
```

Eliminar libro



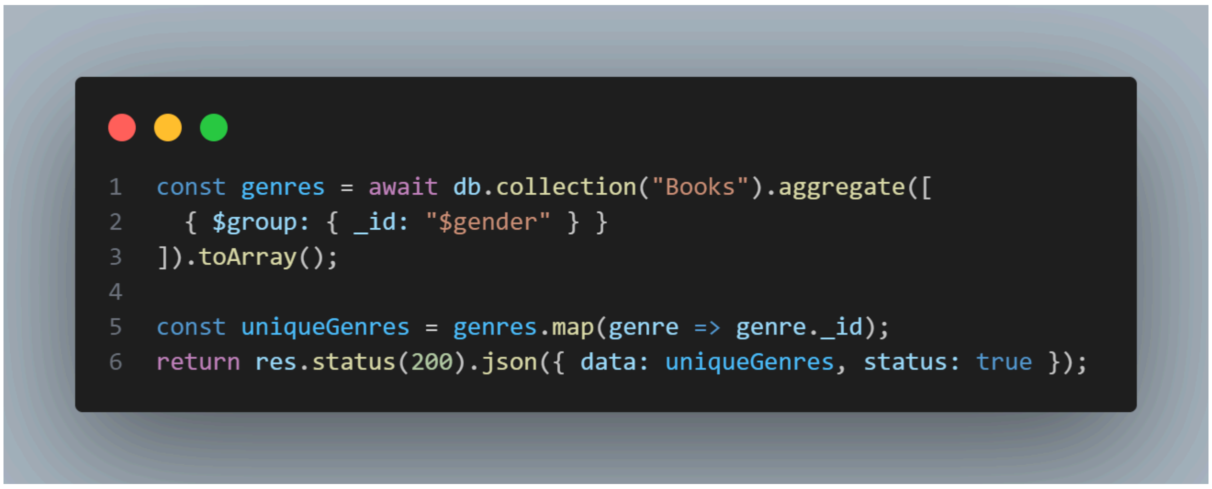
```
const book = await db.collection("Books").findOne({ _id: _book_id });
if (!book) {
  return res.status(404).json({ message: "Book not found", status: false});
}
await db.collection("Books").deleteOne({ _id: _book_id });
return res.status(200).json({ message: "Book deleted successfully" , status: true});
```

Agregar Review al libro



```
const book = await db.collection("Books").findOne({ _id: _book_id });
if (!book) {
  return res.status(404).json({ message: "Book not found", status: false});
}
const review = {
  userName: req.body.userName,
  ...
};
await db.collection("Books").updateOne({ _id: _book_id }, { $push: { reviews: review } });
const updatedBook = await db.collection("Books").findOne({ _id: _book_id });
const reviews = updatedBook.reviews;
const averageScore = reviews.reduce((total, review) => total + review.score, 0) / reviews.length;
await db.collection("Books").updateOne(
  { _id: _book_id },
  { $set: { average_score: averageScore } }
);
```

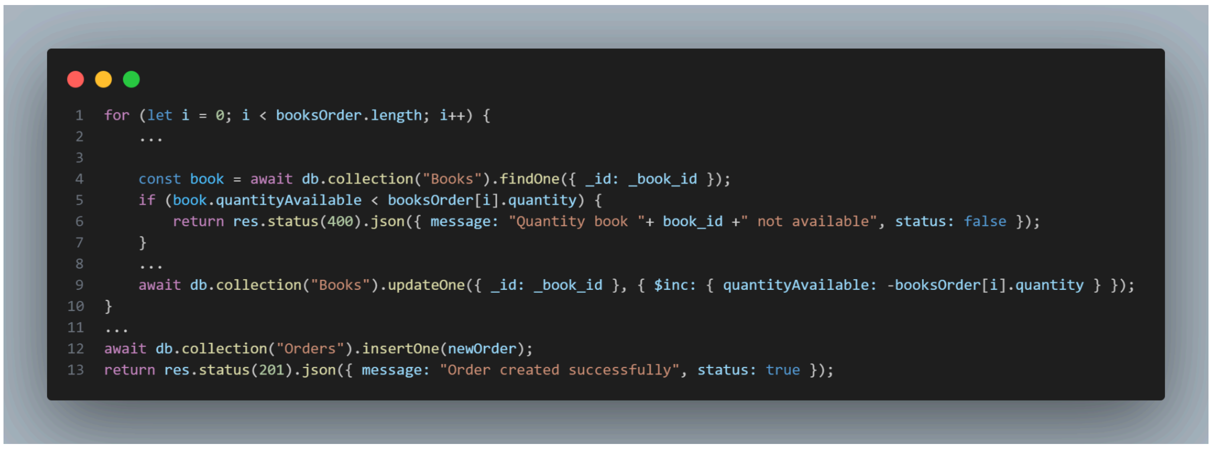
Obtener géneros de los libros



```
const genres = await db.collection("Books").aggregate([
  { $group: { _id: "$gender" } }
]).toArray();

const uniqueGenres = genres.map(genre => genre._id);
return res.status(200).json({ data: uniqueGenres, status: true });
```

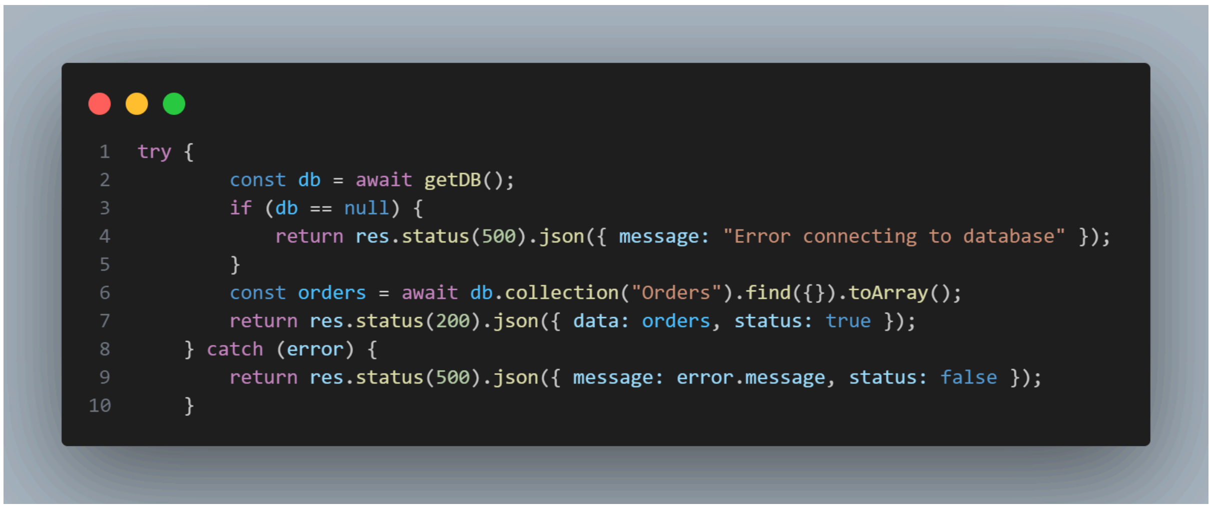
Operaciones de Órdenes

Crear Orden



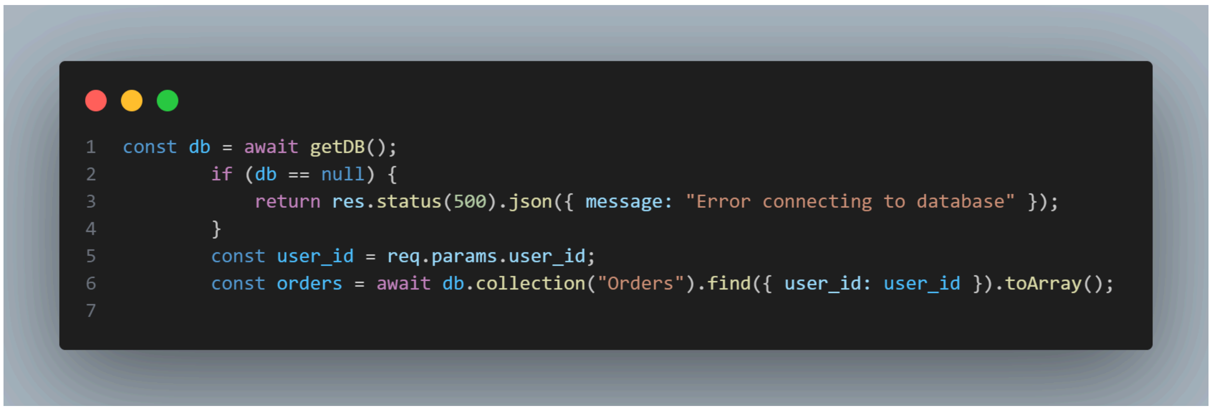
```
for (let i = 0; i < booksOrder.length; i++) {
    ...

    const book = await db.collection("Books").findOne({ _id: _book_id });
    if (book.quantityAvailable < booksOrder[i].quantity) {
        return res.status(400).json({ message: "Quantity book "+ book_id +" not available", status: false });
    }
    ...
    await db.collection("Books").updateOne({ _id: _book_id }, { $inc: { quantityAvailable: -booksOrder[i].quantity } });
}
...
await db.collection("Orders").insertOne(newOrder);
return res.status(201).json({ message: "Order created successfully", status: true });
```

Obtener Órdenes



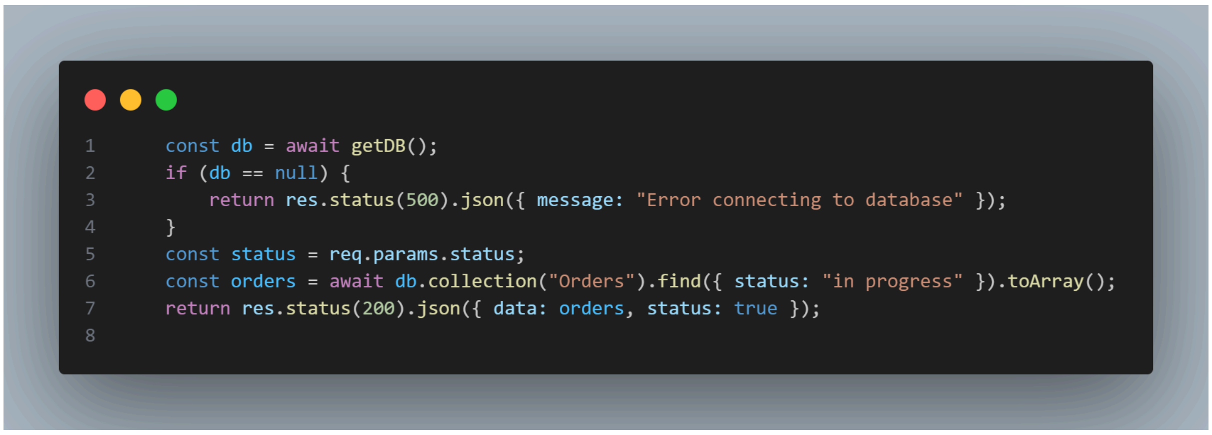
```
try {
        const db = await getDB();
        if (db == null) {
            return res.status(500).json({ message: "Error connecting to database" });
        }
        const orders = await db.collection("Orders").find({}).toArray();
        return res.status(200).json({ data: orders, status: true });
    } catch (error) {
        return res.status(500).json({ message: error.message, status: false });
    }
```

Obtener orden por id de usuario



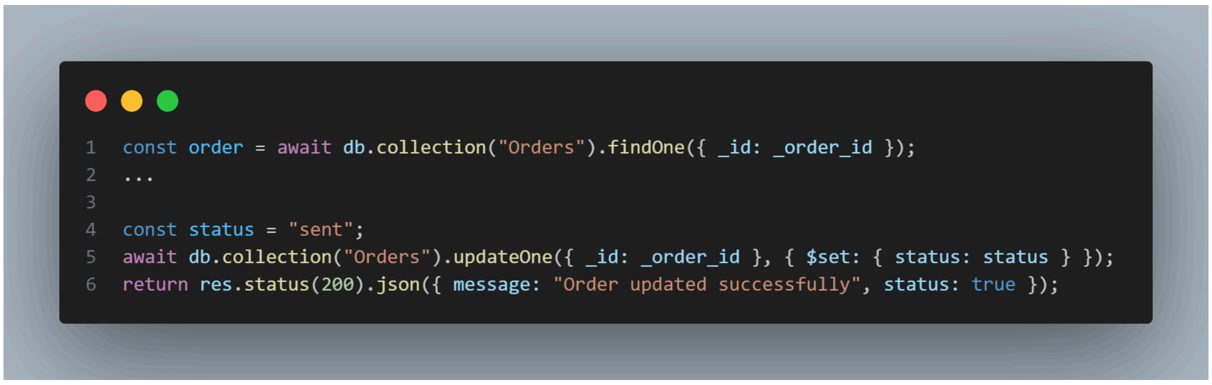
```
const db = await getDB();
        if (db == null) {
            return res.status(500).json({ message: "Error connecting to database" });
        }
        const user_id = req.params.user_id;
        const orders = await db.collection("Orders").find({ user_id: user_id }).toArray();

```

Obtener órdenes en progreso



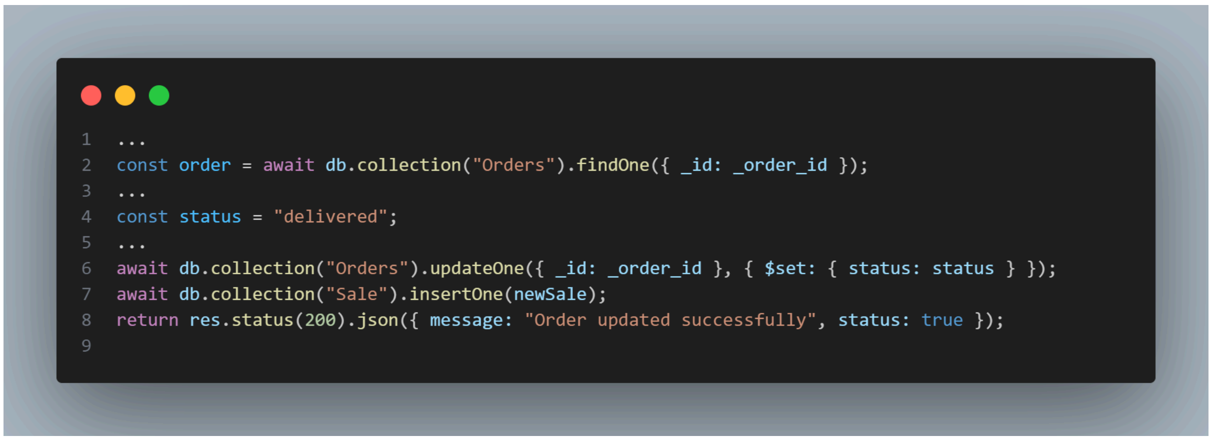
```
    const db = await getDB();
    if (db == null) {
        return res.status(500).json({ message: "Error connecting to database" });
    }
    const status = req.params.status;
    const orders = await db.collection("Orders").find({ status: "in progress" }).toArray();
    return res.status(200).json({ data: orders, status: true });

```

Actualizar estados de orden para enviar



```
const order = await db.collection("Orders").findOne({ _id: _order_id });
...

const status = "sent";
await db.collection("Orders").updateOne({ _id: _order_id }, { $set: { status: status } });
return res.status(200).json({ message: "Order updated successfully", status: true });
```

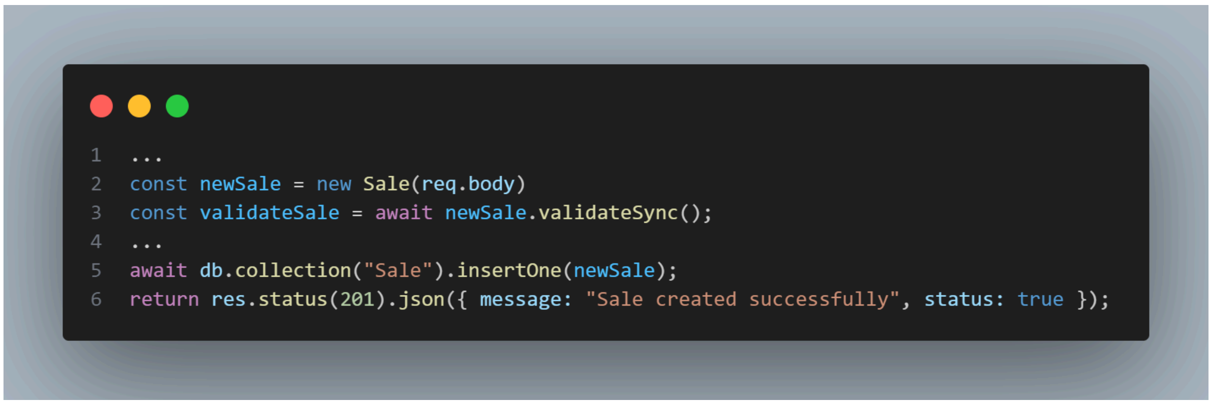
Actualizar estado de órdenes a entregado



```
...
const order = await db.collection("Orders").findOne({ _id: _order_id });
...
const status = "delivered";
...
await db.collection("Orders").updateOne({ _id: _order_id }, { $set: { status: status } });
await db.collection("Sale").insertOne(newSale);
return res.status(200).json({ message: "Order updated successfully", status: true });

```

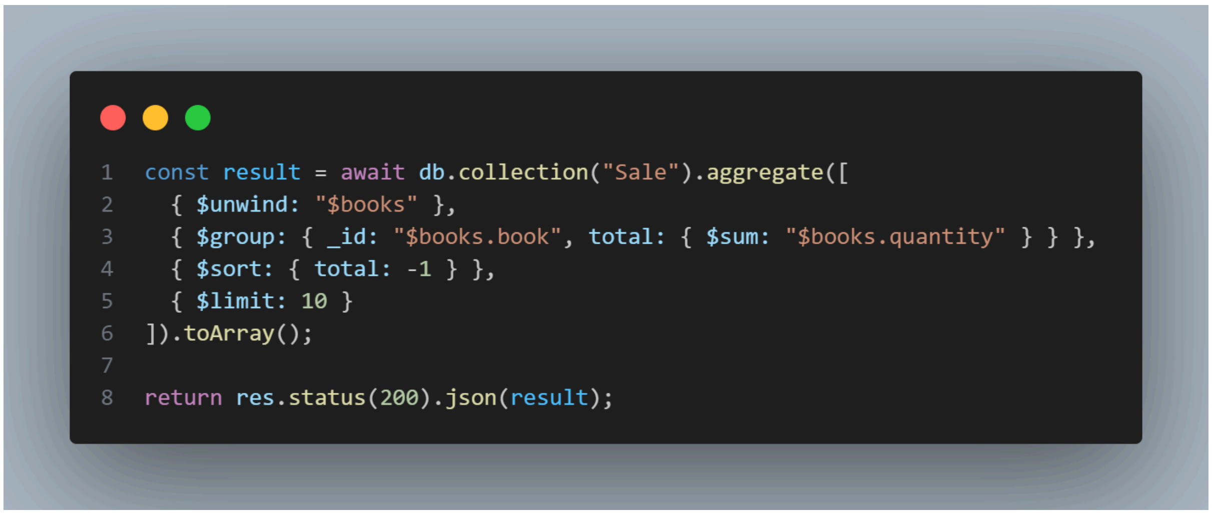
Operaciones de Ventas

Crear nueva Venta



```
...
const newSale = new Sale(req.body)
const validateSale = await newSale.validateSync();
...
await db.collection("Sale").insertOne(newSale);
return res.status(201).json({ message: "Sale created successfully", status: true });
```

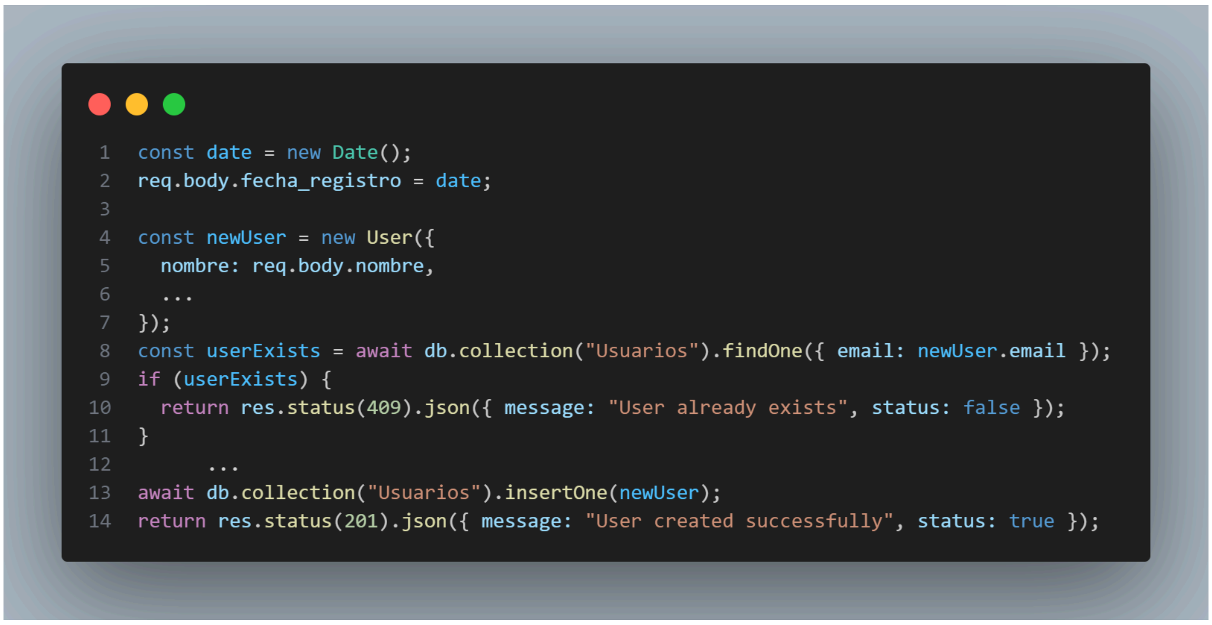
Obtener top de libros más vendidos



```
const result = await db.collection("Sale").aggregate([
  { $unwind: "$books" },
  { $group: { _id: "$books.book", total: { $sum: "$books.quantity" } } },
  { $sort: { total: -1 } },
  { $limit: 10 }
]).toArray();

return res.status(200).json(result);
```

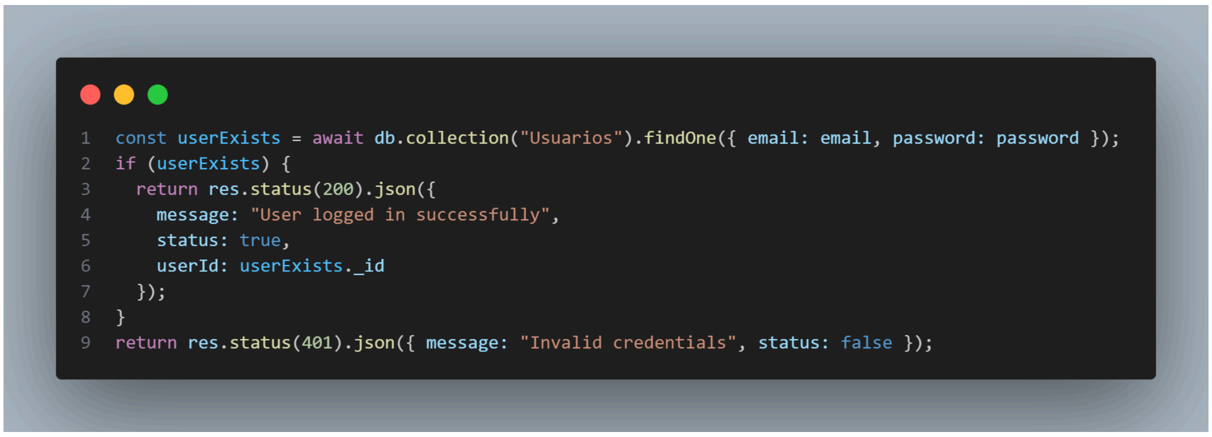
Operaciones de Usuario

Registrar un usuario



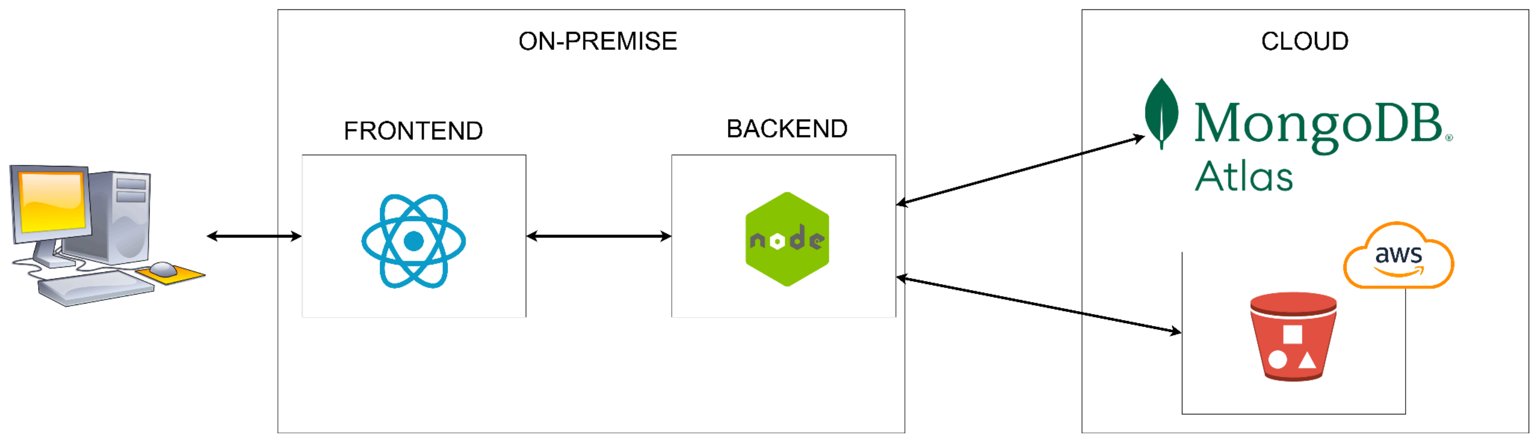
```javascript
const date = new Date();
req.body.fecha_registro = date;

const newUser = new User({
  nombre: req.body.nombre,
  ...
});
const userExists = await db.collection("Usuarios").findOne({ email: newUser.email });
if (userExists) {
  return res.status(409).json({ message: "User already exists", status: false });
}
        ...
await db.collection("Usuarios").insertOne(newUser);
return res.status(201).json({ message: "User created successfully", status: true });
```

Login de Usuario



```javascript
const userExists = await db.collection("Usuarios").findOne({ email: email, password: password });
if (userExists) {
  return res.status(200).json({
    message: "User logged in successfully",
    status: true,
    userId: userExists._id
  });
}
return res.status(401).json({ message: "Invalid credentials", status: false });
```

Arquitectura del Proyecto


ON-PREMISE
FRONTEND
BACKEND
node
CLOUD
MongoDB
Atlas
aws