

*Universidad de San Carlos de Guatemala USAC.*

*Base de datos 2*

*Escuela de Vacaciones Primer semestre*

*Ing. Marlon Francisco Orellana Lopez*

*Auxiliar Jhonathan Tocay*



## **Proyecto 2 - BookStore USAC**

### ***Integrantes:***

#### ***Carnet***

202109754  
201900619  
202109732  
201902128  
202010316  
201700686

#### ***Estudiante***

Aldo Saul Vasquez Moreira  
Yonathan Alexander Hernández Satz  
José Eduardo Galdámez González  
Jorge Mario Cano Blanco  
Joseph Jeferson Marroquin Monroy  
Cristofher Antonio Saquilmer Rodas

## **Proyecto BookStore USAC**

Se trata de una plataforma de venta de libros que permite a los usuarios comprar diferentes libros en línea. Los Administradores pueden registrar libros, gestionar autores y recibir pedidos, mientras que los compradores pueden navegar por los productos, agregar artículos al carrito de compras y realizar pagos seguros.

### **Frontend**

Para el front se utilizó react para una interfaz de usuario intuitiva y receptiva que permite que el usuario navegue por las categorías de los libros y buscar autores.

### **Backend**

Se utilizó TypeScript con Node.js donde se hace toda la gestión de los usuarios, libros, autores integrando con Mongo para almacenar y consultar estos mismos.

### **MongoDB**

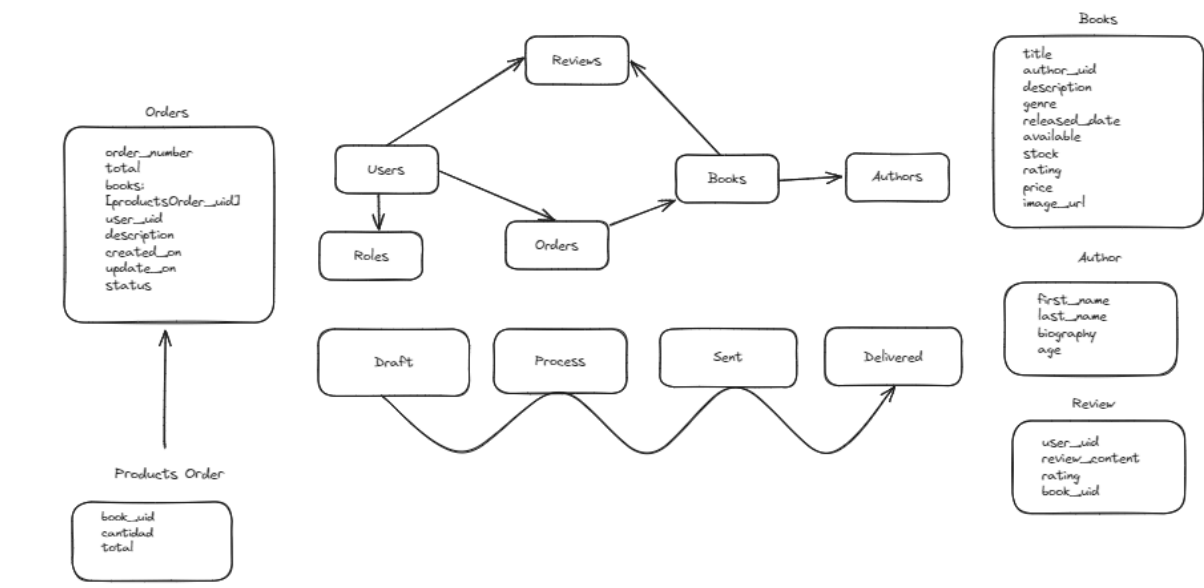
Almacena la información de libros, autores y usuarios.

### **MongoDB Atlas**

Se utilizó MongoDB Atlas para el manejo de la base de datos en la nube.

Esta aplicación combina la robustez y escalabilidad de MongoDB para el almacenamiento de datos, la eficiencia y tipado estático de TypeScript en el backend, y la capacidad de React para crear interfaces de usuario dinámicas y atractivas.

### **Estructura Base de Datos**



## Funciones y Consultas

Función para buscar un usuario en la base de datos que coincida con el correo electrónico y la contraseña. Si se encuentra el usuario, retorna un objeto que indica éxito

```
async login(email:string, password:string) {
    const user = await this.usersModel.find({ email, password });
    const rol = await this.rolesModel.find({ _id:user[0].rol});

    if (user) {
        await this.usersModel.updateOne({ _id: user[0]._id }, {
update_session : new Date()})
        return { success: true, userdata: user, rol: rol[0].name};
    }
    return { success: false };
}
```

Método diseñado para recuperar un usuario específico de la base de datos utilizando su identificador. Utilizar el modelo usersmodel para buscar en la coleccion users.

```
async getUser(id:string) {
    return await this.usersModel.find({ _id: id });
}
```

La función está diseñada para recuperar todos los libros de la base de datos, junto con la información de sus autores.

```

async getAllBooks() {
    return await this.booksModel.aggregate([
        {
            $lookup: {
                from: 'authors', // Nombre de la colección de
autores en la base de datos
                localField: 'author_uid',
                foreignField: '_id',
                as: 'author'
            }
        },
        {
            $unwind: {
                path: '$author',
                preserveNullAndEmptyArrays: true // Esto asegura
que los libros sin autores aún sean incluidos
            }
        },
        {
            $project: {
                _id: 1,
                title: 1,
                author_uid: 1,
                description: 1,
                genre: 1,
                released_date: 1,
                available: 1,
                stock: 1,
                rating: 1,
                price: 1,
                image_url: 1,
                'author._id': 1,
                'author.first_name': 1,
                'author.last_name': 1,
                'author.biography': 1,
                'author.age': 1
            }
        },
        {
            $sort: { _id: 1 } // Orden ascendente por _id
        }
    ]);
}

```

Esta función nos ayuda a buscar libros cuyo título coincida con un patrón proporcionado el nombre del libro, utilizando una expresión regular, y luego recuperar información detallada de esos libros junto con los detalles de sus autores.

```
async getBooksName(nameBook:string) {  
    const regex = new RegExp(nameBook, 'i');  
    return await this.booksModel.aggregate([  
        {  
            $match: { title: regex }  
        },  
        {  
            $lookup: {  
                from: 'authors',  
                localField: 'author_uid',  
                foreignField: '_id',  
                as: 'author'  
            }  
        },  
        {  
            $unwind: {  
                path: '$author',  
                preserveNullAndEmptyArrays: true  
            }  
        },  
        {  
            $project: {  
                _id: 1,  
                title: 1,  
                author_uid: 1,  
                description: 1,  
                genre: 1,  
                released_date: 1,  
                available: 1,  
                stock: 1,  
                rating: 1,  
                price: 1,  
                image_url: 1,  
                'author._id': 1,  
                'author.first_name': 1,  
                'author.last_name': 1,  
                'author.biography': 1,  
                'author.age': 1  
            }  
        }  
    ])  
}
```

```

    }
  }
  });
}

```

Esta función sirve para buscar autores cuyo nombre coincida con uno o más términos proporcionados en name.

```

    async getAuthorName(name: string) {
      const names = name.split(' ').filter(n => n);
      const regexes = names.map(n => new RegExp(n, 'i'));

      let query = [];

      if (regexes.length === 1) {
        query.push({ first_name: regexes[0] });
        query.push({ last_name: regexes[0] });
      } else if (regexes.length >= 2) {
        for (let i = 0; i < regexes.length; i++) {
          for (let j = 0; j < regexes.length; j++) {
            if (i !== j) {
              query.push({ $and: [{ first_name: regexes[i] },
                { last_name: regexes[j] }] });
            }
          }
        }
      }

      return await this.authorModel.find({
        $or: query
      });
    }
  }

```

Función que busca al autor por medio de su id.

```

    async getAuthor(authorId:string) {
      return await this.authorModel.find({ _id: authorId});
    }
  }

```

Función que obtiene todos los géneros de libros disponibles junto con el recuento de libros por cada género. La función retorna una promesa que se resolverá con un arreglo de objetos, donde cada objeto representa un género de libro.

```
async getAllGenres() {
    return await this.booksModel.aggregate([
        {
            $group: {
                _id: "$genre",
                count: { $sum: 1 }
            }
        },
        {
            $project: {
                _id: 0,
                genre: "$_id"
            }
        }
    ]).exec();
}
```

Esta función permite obtener una lista de libros de un género específico junto con detalles adicionales de los autores asociados, si están disponibles, facilitando así la presentación de datos en aplicaciones o servicios que consumen esta información.

```
async getGenreBooks(genreBook:string) {
    const regex = new RegExp(genreBook, 'i');
    return await this.booksModel.aggregate([
        {
            $match: { genre: regex }
        },
        {
            $lookup: {
                from: 'authors',
                localField: 'author_uid',
                foreignField: '_id',
                as: 'author'
            }
        },
        {
            $unwind: {
                path: '$author',
                preserveNullAndEmptyArrays: true
            }
        }
    ])
```

```

    },
    {
      $project: {
        _id: 1,
        title: 1,
        author_uid: 1,
        description: 1,
        genre: 1,
        released_date: 1,
        available: 1,
        stock: 1,
        rating: 1,
        price: 1,
        image_url: 1,
        'author._id': 1,
        'author.first_name': 1,
        'author.last_name': 1,
        'author.biography': 1,
        'author.age': 1
      }
    }
  ]);
}

```

Esta función nos ayuda a obtener libros ordenados por precio ascendente, junto con la información de sus autores mediante una consulta de agregación.

```

async getLowPrices() {
  return await this.booksModel.aggregate([
    {
      $sort: { price: 1 }
    },
    {
      $lookup: {
        from: 'authors',
        localField: 'author_uid',
        foreignField: '_id',
        as: 'author'
      }
    },
    {
      $unwind: {
        path: '$author',

```



```

        preserveNullAndEmptyArrays: true
      }
    },
    {
      $project: {
        _id: 1,
        title: 1,
        author_uid: 1,
        description: 1,
        genre: 1,
        released_date: 1,
        available: 1,
        stock: 1,
        rating: 1,
        price: 1,
        image_url: 1,
        'author._id': 1,
        'author.first_name': 1,
        'author.last_name': 1,
        'author.biography': 1,
        'author.age': 1
      }
    }
  ]);
}

```

Esta función nos ayuda a obtener libros ordenados por precio descendente, junto con la información de sus autores mediante una consulta de agregación.

```

async getHighPrices() {
  return await this.booksModel.aggregate([
    {
      $sort: { price: -1 }
    },
    {
      $lookup: {
        from: 'authors',
        localField: 'author_uid',
        foreignField: '_id',
        as: 'author'
      }
    }
  ],
  {

```

```

        $unwind: {
          path: '$author',
          preserveNullAndEmptyArrays: true
        }
      },
      {
        $project: {
          _id: 1,
          title: 1,
          author_uid: 1,
          description: 1,
          genre: 1,
          released_date: 1,
          available: 1,
          stock: 1,
          rating: 1,
          price: 1,
          image_url: 1,
          'author._id': 1,
          'author.first_name': 1,
          'author.last_name': 1,
          'author.biography': 1,
          'author.age': 1
        }
      }
    ]
  );
}

```

Esta función obtiene los libros ordenados por calificación de manera descendente junto con la información de sus autores.

```

async getHighRatingBooks() {
  return await this.booksModel.aggregate([
    {
      $sort: { rating: -1 }
    },
    {
      $lookup: {
        from: 'authors',
        localField: 'author_uid',
        foreignField: '_id',
        as: 'author'
      }
    }
  ])
}

```

```

    },
    {
      $unwind: {
        path: '$author',
        preserveNullAndEmptyArrays: true
      }
    },
    {
      $project: {
        _id: 1,
        title: 1,
        author_uid: 1,
        description: 1,
        genre: 1,
        released_date: 1,
        available: 1,
        stock: 1,
        rating: 1,
        price: 1,
        image_url: 1,
        'author._id': 1,
        'author.first_name': 1,
        'author.last_name': 1,
        'author.biography': 1,
        'author.age': 1
      }
    }
  ]);
}

```

Esta función obtiene los libros ordenados por calificación de manera ascendente junto con la información de sus autores.

```

async getLowRatingBooks() {
  return await this.booksModel.aggregate([
    {
      $sort: { rating: 1 }
    },
    {
      $lookup: {
        from: 'authors',
        localField: 'author_uid',
        foreignField: '_id',

```

```

        as: 'author'
      }
    },
    {
      $unwind: {
        path: '$author',
        preserveNullAndEmptyArrays: true
      }
    },
    {
      $project: {
        _id: 1,
        title: 1,
        author_uid: 1,
        description: 1,
        genre: 1,
        released_date: 1,
        available: 1,
        stock: 1,
        rating: 1,
        price: 1,
        image_url: 1,
        'author._id': 1,
        'author.first_name': 1,
        'author.last_name': 1,
        'author.biography': 1,
        'author.age': 1
      }
    }
  ]);
}

```

Función para obtener todos los libros por medio de su id.

```

async getBookById(id:string) {
  return await this.booksModel.find({ _id: id});
}

```

Función para obtener todos los autores por medio de su id.

```

async getAuthorById(id:string) {
  return await this.authorModel.find({ _id:id });
}

```

Función para obtener todas las revisiones que tenga un libro si no tiene reseñas lanza un error de que no se ha encontrado reseñas.

```
async getReviewsBook(id: string) {
  try {
    const reviews = await this.reviewsModel.aggregate([
      { $match: { book_uid: new ObjectId(id) } },
      {
        $lookup: {
          from: 'users',
          localField: 'user_uid',
          foreignField: '_id',
          as: 'user',
        },
      },
      {
        $unwind: '$user' // Desenrollar el array 'user'
        // resultado del $lookup
      },
      {
        $project: {
          _id: 1,
          content: 1,
          rating: 1,
          created_on: 1,
          user_name: { $concat: ['$user.first_name', ' ', '$user.last_name'] }, // Concatenar nombres y apellidos
          user_email: '$user.email',
        },
      },
    ]);

    return reviews;
  } catch (error) {
    console.error('Error al obtener reseñas del libro', error);
    throw new Error('No se pudieron obtener las reseñas del libro');
  }
}
```

Función para agregar una reseña a un libro donde se agrega la descripción, el rating que es la calificación que le dio el usuario y también se guarda el nombre del usuario que la publicó.

```
async addReview(idBook: string, idUser: string, content: string, rating: number) {
```

```

    try {
      console.log('idUser:', idUser);
      const review = await this.reviewsModel.create({
        user_uid: idUser,
        content,
        rating,
        book_uid: idBook
      });
      console.log('review:', review);
      if (review) {
        const ratingBook = await this.reviewsModel.aggregate([
          { $match: { book_uid: new ObjectId(idBook) } },
          { $group: { _id: "$book_uid", avgRating: { $avg:
"$rating" } } }
        ]).exec();
        if (ratingBook.length > 0) {
          const avgRating =
ratingBook[0].avgRating.toFixed(2);
          const updatedBook = await
this.booksModel.updateOne(
            { _id: idBook },
            { rating: avgRating }
          ).exec();

          return { success: true };
        } else {
          console.error('No se encontraron reseñas para el
libro especificado.');
```

```

          return { success: false };
        }
      } else {
        console.error('No se pudo crear la reseña.');
```

```

        return { success: false };
      }
    } catch (error) {
      console.error('Error al agregar la reseña y actualizar el
rating:', error);
      throw error;
    }
  }
}

```

Función para agregar un libro donde se pide el título del libro, el id del autor, la descripción, el género, la fecha de publicación, el stock, el precio y la imagen del libro.

```
async addBook(title: string, author_uid: string, description: string,
genre: string, released_date: string, stock: number, price: number,
image_url: string) {
    try {
        var available = false;
        // Verifica la existencia de un libro para no generar
duplicados
        if (stock > 0 ) {
            available = true;
        }
        const existingBook = await this.booksModel.findOne({ title,
author_uid });
        if (existingBook) {
            return null; // Devolver null si el libro ya existe
        }

        const newBook = await this.booksModel.create({
            title,
            author_uid,
            description,
            genre,
            released_date, //string
            available,
            stock,
            rating: 0, // Inicialmente, el rating es 0
            price,
            image_url
        });
        return newBook.toJSON();
    } catch (error) {
        console.error('Error adding book:', error);
        throw new Error('Error adding book');
    }
}
```

Función para actualizar un libro donde se pueden colocar los mismos datos que para agregar un libro pero solo se envían los campos que se quieren actualizar.

```

async updateBook(id: string, updates: Partial<{ title: string,
author_uid: string, description: string, genre: string, released_date:
string, stock: number, price: number, image_url: string }>) {
  try {
    var available = false;
    // Verificar que el libro existe
    const existingBook = await this.booksModel.findById(id);
    if (!existingBook) {
      return null; // Devolver null si el libro no existe
    }

    // Verificar si el nuevo título y autor ya existen en otro
libro

    if (updates.title && updates.author_uid) {
      const duplicateBook = await this.booksModel.findOne({
title: updates.title, author_uid: updates.author_uid, _id: { $ne: id }
});

      if (duplicateBook) {
        throw new Error('El libro con el mismo título y
autor ya existe.');
```



```

        available: available,
        stock: updates.stock !== undefined ? updates.stock :
existingBook.stock,
        price: updates.price !== undefined ? updates.price :
existingBook.price,
        image_url: updates.image_url || existingBook.image_url
    };

    const updatedBook = await
this.booksModel.findByIdAndUpdate(id, updatedData, { new: true });
    if (!updatedBook) {
        throw new Error('Error actualizando el libro.');
```

Función que borra un libro de la base de datos por medio de su id.

```

async deleteBook(id: string) {
    try {
        const deletedBook = await
this.booksModel.findByIdAndDelete(id);
        return deletedBook ? deletedBook.toJSON() : null;
    } catch (error) {
        console.error('Error deleting book:', error);
        throw new Error('Error deleting book');
```

```

async register(first_name: string, last_name: string, email: string,
phone: number, address: string, password: string, rol: string) {
    try {
        // Verificar si el correo electrónico ya está registrado
        const existingUser = await this.usersModel.findOne({ email
});

        if (existingUser) {
            return null; // Devolver null si el correo electrónico
ya está registrado
        }
    }
}
```

```

        const newUser = await this.usersModel.create({
            first_name,
            last_name,
            email,
            phone,
            address,
            password, // falta cifrar la contraseña jeje salu2
            rol,
            shopping_cart: null
        });
        return newUser.toJSON();
    } catch (error) {
        console.error('Error registering user:', error);
        throw new Error('Error registering user');
    }
}

```

Permite a los usuarios actualizar su información de perfil de manera segura y eficiente, manteniendo la integridad de los datos y proporcionando una experiencia fluida al usuario final.

```

async updateProfile(id: string, updates: Partial<{ first_name: string,
last_name: string, email: string, phone: number, address: string,
password: string, rol: string, shopping_cart: any }>) {
    try {
        // Verificar que el usuario existe
        const existingUser = await this.usersModel.findById(id);
        if (!existingUser) {
            return null; // Devolver null si el usuario no existe
        }

        // Verificar si el nuevo correo electrónico ya existe en
        // otro usuario
        if (updates.email && updates.email !== existingUser.email) {
            const duplicateUser = await this.usersModel.findOne({
                email: updates.email, _id: { $ne: id } });
            if (duplicateUser) {
                throw new Error('El correo electrónico ya está
                registrado.');
```

```

        // Mantener los valores existentes si los nuevos valores no
        son proporcionados
        const updatedData = {
            first_name: updates.first_name ||
existingUser.first_name,
            last_name: updates.last_name || existingUser.last_name,
            email: updates.email || existingUser.email,
            phone: updates.phone !== undefined ? updates.phone :
existingUser.phone,
            address: updates.address || existingUser.address,
            password: updates.password || existingUser.password,
            rol: updates.rol || existingUser.rol,
            shopping_cart: updates.shopping_cart !== undefined ?
updates.shopping_cart : existingUser.shopping_cart
        };

        const updatedUser = await
this.usersModel.findByIdAndUpdate(id, updatedData, { new: true });
        if (!updatedUser) {
            throw new Error('Error actualizando el perfil.');
```

Función que ayuda a retornar los roles de la aplicación en este caso se utiliza “Admin” y “Client” si se quiere agregar otro rol se puede hacer sin ningun problema.

```

async getRoles() {
    try {
        const roles = await this.rolesModel.find({});
        return roles.map(role => role.toJSON());
    } catch (error) {
        console.error('Error getting roles:', error);
        throw new Error('Error getting roles');
```

Función que nos muestra todos los autores que están en la base de datos en la coleccion de Autores.

```
async getAllAuthors() {
    return await this.authorModel.find();
}
```

Nos devuelve los libros de los autores buscandolos por medio de su id.

```
async getBooksAuthor(authorId: string) {
    if (!ObjectId.isValid(authorId)) {
        throw new Error('Id inválido.')
    }

    const autorObjectId = new ObjectId(authorId)
    return await this.booksModel.find({author_uid:autorObjectId});
}
```

Agregar a un autor con los campos de nombre, apellido, biografia y edad.

```
async addAuthor(first_name:string, last_name:string, biography:string,
age:number) {
    const newAuthor = await this.authorModel.create({
        first_name,
        last_name,
        biography,
        age
    });
    return newAuthor;
}
```

Borra al autor por medio de su id.

```
async deleteAutor(authorId: string){
    if (!ObjectId.isValid(authorId)) {
        throw new Error('Id inválido.')
    }

    const autorObjectId = new ObjectId(authorId)
    await this.authorModel.deleteOne({_id:autorObjectId});
    await this.booksModel.deleteMany({author_uid:autorObjectId})
    return 'Author and associated books deleted successfully'
}
```

Crea las ordenes de compra, esto después de crear el carrito de compra y donde se pueden agregar productos.

```
async createOrder(user_uid: string) {
```

```

        var order_number=await this.ordersModel.countDocuments({
user_uid: user_uid }).exec();
        if (order_number==0) order_number=1;
        else order_number++;

        const newOrder = await this.ordersModel.create({
            order_number,
            user_uid
        });

        await this.usersModel.updateOne({ _id: user_uid }, {
shopping_cart: newOrder._id }).exec();

        return newOrder;
    }

```

Agrega los productos al carrito, pide el id de la orden, el id del libro que se está agregando y la cantidad de libros.

```

async addProductOrder(order_uid: string, book_uid: string, quantity:
number) {
    try {
        // Buscar el libro por su _id
        const book = await this.booksModel.findById(book_uid);
        if (!book) {
            console.log('El libro no existe.');
```

```

            throw new Error('El libro no existe.');
```

```

        }
        if (book.stock <= quantity) {
            console.log('No hay suficiente stock.');
```

```

            throw new Error('No hay suficiente stock.');
```

```

        }

        // Verificar si el producto ya está en la orden
        const productsOrder = await this.productsOrderModel.find({
order_uid, book_uid });
        if (productsOrder.length > 0) {
            console.log('El libro ya está en la orden.');
```

```

            throw new Error('El libro ya está en la orden.');
```

```

        }

        const total = book.price * quantity;
    }

```

```

        // Crear el nuevo producto de la orden
        const newProductOrder = await
this.productsOrderModel.create({
            order_uid,
            book_uid,
            quantity,
            total
        });

        // Actualizar la orden para agregar el nuevo producto
        await this.ordersModel.updateOne({ _id: order_uid }, { $push:
{ books: newProductOrder._id } });

        // Obtener la orden actualizada con los libros poblados
        const order = await
this.ordersModel.findById(order_uid).populate('books');
        if (!order) {
            console.log('La orden no existe.');
```

```

            throw new Error('La orden no existe.');
```

```

        }

        const orderDescription = await
Promise.all(order.books.map(async (productOrder: any) => {
            const product = await this.productsOrderModel.find({ _id:
productOrder });
            const productBook = await
this.booksModel.find({_id:product[0].book_uid});
            if (!productBook) {
                console.log('El libro del producto en la orden no
existe.');
```

```

                throw new Error('El libro del producto en la orden no
existe.');
```

```

            }
            return `${product[0].quantity} x ${productBook[0].title}`;
        }));
        const orders1=await this.ordersModel.find({ _id:
order_uid});
        var totalOrder=0;
        for (let i=0; i<orders1[0].books.length; i++) {
            const product=await this.productsOrderModel.find({ _id:
orders1[0].books[i]});
            console.log('product:', product[0].total);
            totalOrder+=product[0].total;

```

```

    }

    await this.ordersModel.updateOne({ _id: order_uid }, {
description: orderDescription.join(', '), total: totalOrder }).exec();

    return newProductOrder;
  } catch (error) {
    console.error('Error al agregar el producto a la orden:',
error);
    throw new Error('Error al agregar el producto a la orden.');
```

Función que nos ayuda a borrar un producto de la orden esto por medio del id del producto.

```

async deleteProductOrder(product_uid: string) {
  const product = await
this.productsOrderModel.find({_id:product_uid});
  try {
    const book = await this.booksModel.findByIdAndUpdate({ _id:
product[0].book_uid }, { $inc: { stock: product[0].quantity }}).exec();
    const orderUpdateResult = await this.ordersModel.updateOne(
      { _id: product[0].order_uid },
      { $pull: { books: new ObjectId(product_uid) } }
    ).exec();

    if (orderUpdateResult.modifiedCount === 0) {
      console.log(`No se encontró el pedido con ID
${product[0].order_uid} o el producto ${product_uid} no estaba en el
array 'books'.`);
    } else {
      console.log(`Producto ${product_uid} eliminado del
array 'books' del pedido ${product[0].order_uid}.`);
    }

    const productDeleteResult = await
this.productsOrderModel.findOneAndDelete({ _id: product_uid }).exec();
    const orders=await this.ordersModel.find({ _id:
product[0].order_uid});
    var totalOrder=0;
    for (let i=0; i<orders[0].books.length; i++) {
```

```

        const product=await this.productsOrderModel.find({ _id:
orders[0].books[i]});
        totalOrder+=product[0].total;
    }
    const order = await
this.ordersModel.findById(product[0].order_uid).populate('books');
    if (!order) {
        console.log('La orden no existe.');
```

```

        throw new Error('La orden no existe.');
```

```

    }

    const orderDescription = await
Promise.all(order.books.map(async (productOrder: any) => {
        const product1 = await this.productsOrderModel.find({
_id: productOrder });
        const productBook = await
this.booksModel.find({_id:product1[0].book_uid});
        if (!productBook) {
            console.log('El libro del producto en la orden no
existe.');
```

```

            throw new Error('El libro del producto en la orden no
existe.');
```

```

        }
        return `${product1[0].quantity} x
${productBook[0].title}`;
    }));

    await this.ordersModel.updateOne({ _id:
product[0].order_uid }, {description:orderDescription.join(', '),
total: totalOrder }).exec();
    if (!productDeleteResult) {
        console.log(`No se encontró el producto con ID
${product_uid} en la colección productsOrder.`);
    } else {
        console.log(`Producto ${product_uid} eliminado de la
colección productsOrder.`);
    }
    return orderUpdateResult;
} catch (error) {
    console.error(`Error eliminando el producto ${product_uid}
del pedido ${product[0].order_uid}:`, error);
    throw error;
}

```



```
}
```

Función que ayuda a actualizar la cantidad de productos en el carrito donde se pide el id del producto y la nueva cantidad de productos.

```
async updateProductOrder(product_uid: string, quantity: number) {
    const productOriginal = await
this.productsOrderModel.find({_id:product_uid});
    const book = await
this.booksModel.find({_id:productOriginal[0].book_uid});
    var stock=book[0].stock+productOriginal[0].quantity;
    if ((stock) < quantity) {
        console.log('No hay suficiente stock.');
```

```
        return Error('No hay suficiente stock.');
```

```
    }
    var newTotal=book[0].price*quantity;
    const product = await
this.productsOrderModel.findByIdAndUpdate(product_uid, { quantity:
quantity, total:newTotal }, { new: true }).exec();
    await this.booksModel.updateOne({ _id:
productOriginal[0].book_uid }, { stock: stock-quantity }).exec();
    const products=await this.ordersModel.find({ _id:
productOriginal[0].order_uid});
    // Obtener la orden actualizada con los libros poblados
    const order = await
this.ordersModel.findById(productOriginal[0].order_uid).populate('books
');
```

```
    if (!order) {
        console.log('La orden no existe.');
```

```
        throw new Error('La orden no existe.');
```

```
    }

    const orderDescription = await
Promise.all(order.books.map(async (productOrder: any) => {
        const product1 = await this.productsOrderModel.find({ _id:
productOrder });
        const productBook = await
this.booksModel.find({_id:product1[0].book_uid});
        if (!productBook) {
            console.log('El libro del producto en la orden no
existe.');
```

```
            throw new Error('El libro del producto en la orden no
existe.');
```

```

    }
    return `${product1[0].quantity} x ${productBook[0].title}`;
  }));
  var totalOrder=0;
  for (let i=0; i<products[0].books.length; i++) {
    const product=await this.productsOrderModel.find({ _id:
products[0].books[i]});
    totalOrder+=product[0].total;
  }
  await this.ordersModel.updateOne({ _id:
productOriginal[0].order_uid }, {description: orderDescription.join(',
'), total: totalOrder }).exec();
  return product;
}

```

Llama a la orden por su id y nos muestra el resumen de esta.

```

async getOrderResume(order_uid: string) {
  const order=await this.ordersModel.find({ _id: order_uid });
  const products = await this.productsOrderModel.aggregate([
    {
      $match: { order_uid: new ObjectId(order_uid) } // Filtro
para encontrar las órdenes por order_uid
    },
    {
      $lookup: {
        from: 'books', // Colección books
        localField: 'book_uid',
        foreignField: '_id',
        as: 'book'
      }
    },
    {
      $unwind: '$book' // Desenrollar el array resultado de
$lookup (ya que es un solo libro por orden)
    },
    {
      $lookup: {
        from: 'authors', // Colección authors
        localField: 'book.author_uid',
        foreignField: '_id',
        as: 'author'
      }
    }
  ]

```

```

        },
        {
            $unwind: '$author' // Desenrollar el array resultado de
$lookup (ya que es un solo autor por libro)
        },
        {
            $project: {
                _id: 1, // Opcional: Excluir el _id del resultado final
si no es necesario
                order_uid: 1,
                quantity: 1,
                total: 1,
                book: {
                    title: 1,
                    description: 1,
                    genre: 1,
                    released_date: 1,
                    available: 1,
                    stock: 1,
                    rating: 1,
                    price: 1,
                    image_url: 1
                },
                author: {
                    first_name: 1,
                    last_name: 1,
                    biography: 1,
                    age: 1
                }
            }
        }
    ]));

    return { order, products };
}

```

```

async getOrdersByUser(user_uid: string) {
    return await this.ordersModel.find({ user_uid: user_uid,
status: { $ne: 'DRAFT' }}).sort({ created_on: -1});
}

```

Llama a todas las ordenes que están en la base de datos para que el administrador las pueda ver.

```
async getOrders() {
  try {
    const orders = await this.ordersModel.aggregate([
      {
        $match: { status: { $ne: 'DRAFT' } }
      },
      {
        $sort: { created_on: -1 }
      },
      {
        $lookup: {
          from: 'users', // Nombre de la colección de usuarios en
          // tu base de datos
          localField: 'user_uid',
          foreignField: '_id',
          as: 'user'
        }
      },
      {
        $unwind: '$user'
      },
      {
        $project: {
          _id: 1,
          order_number: 1,
          description: 1,
          status: 1,
          total: 1,
          user: { $mergeObjects: ['$user', {}] }, // Asegura que
          // los campos de usuario se agreguen correctamente
          books: 1,
          created_on: 1,
          update_on: 1
        }
      }
    ]);
    return orders;
  } catch (error) {
    console.error('Error fetching orders:', error);
    throw new Error('Unable to fetch orders');
```

```

    }
  }
}

```

Función que nos ayuda a borrar la orden en dado caso ya no se quiera. donde se pide el id de la orden.

```

async deleteOrder(order_uid: string) {
  const order = await this.ordersModel.find({_id:order_uid});
  const products=await this.productsOrderModel.find({ order_uid:
order_uid });
  for (let i=0; i<products.length; i++) {
    await this.productsOrderModel.deleteOne({ _id:
products[i]._id });
    const book = await
this.booksModel.find({_id:products[i].book_uid});
    await this.booksModel.updateOne({ _id: products[i].book_uid
}, { stock: book[0].stock+products[i].quantity}).exec();
  }
  const orderDeleteResult = await this.ordersModel.deleteOne({
_id: order_uid }).exec();
  return orderDeleteResult;
}

```

Esta funcion se encarga de actualizar el estado de un pedido en funcion de su estado actual ya sea, Draft, Process o Sent. Si el estado es Draft se actualiza a Process y se limpia el carrito, si es Process se pasa a Sent y si es Sent se ppasa a Delivered.

```

async updateStatusOrder(order_uid: string) {
  const order=await this.ordersModel.find({ _id: order_uid });
  if(order[0].status=='DRAFT') {
    await this.ordersModel.updateOne({ _id: order_uid }, {
status: 'PROCESS' }).exec();
    await this.usersModel.updateOne({ _id: order[0].user_uid },
{ shopping_cart: null }).exec();
    return 'Order status updated successfully';
  }else if(order[0].status=='PROCESS') {
    await this.ordersModel.updateOne({ _id: order_uid }, {
status: 'SENT' }).exec();
    return 'Order status updated successfully';
  }else if(order[0].status=='SENT') {
    await this.ordersModel.updateOne({ _id: order_uid }, {
status: 'DELIVERED' }).exec();
    return 'Order status updated successfully';
  }
}

```

```

    }

}

```

Función que nos devuelve el reporte de los libros más vendidos, basado en pedidos que han sido entregados.

```

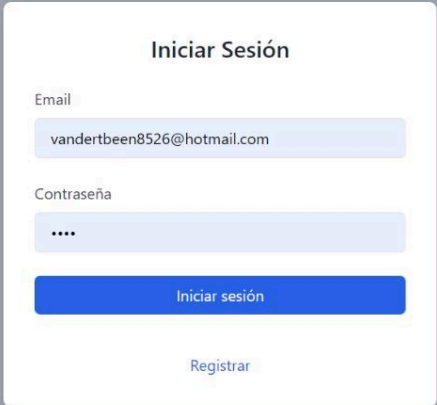
async getReportTopBooks() {
  try {
    const report = await this.ordersModel.aggregate([
      { $match: { status: 'DELIVERED' } },
      { $unwind: '$books' },
      {
        $lookup: {
          from: 'products_orders',
          localField: 'books',
          foreignField: '_id',
          as: 'orderDetails'
        }
      },
      { $unwind: '$orderDetails' },
      {
        $lookup: {
          from: 'books',
          localField: 'orderDetails.book_uid',
          foreignField: '_id',
          as: 'bookDetails'
        }
      },
      { $unwind: '$bookDetails' },
      {
        $group: {
          _id: '$bookDetails.title',
          totalQuantitySold: { $sum: '$orderDetails.quantity' },
          totalSales: { $sum: { $multiply: ['$orderDetails.quantity', '$bookDetails.price'] } }
        }
      },
      {
        $project: {
          _id: 0,
          title: '$_id',

```

```
        totalQuantitySold: 1,  
        totalSales: 1  
      },  
      { $sort: { totalSales: -1 } }  
    ] );  
  
    return report;  
  } catch (error) {  
    console.error('Error generating sales report:', error);  
  }  
}
```

## Vista Login

Ventana donde puede ingresar su correo y contraseña para verificar si existe.



The image shows a login form titled "Iniciar Sesión" centered on a gray background. The form is a white card with a light gray border. It contains two input fields: "Email" with the value "vandertbeen8526@hotmail.com" and "Contraseña" with masked characters "....". Below these fields is a blue button labeled "Iniciar sesión" and a link labeled "Registrar" in blue text.

**Vista Registro:** vista donde se puede registrar un usuario para poder acceder a la aplicación

## Registro

Nombre

Apellido

Email

vandertbeen8526@hotmail

Teléfono

Dirección

Contraseña

....

Rol

Seleccione un rol

Registrar

Iniciar Sesión

## Vista de Perfil de Usuario

Vista donde un usuario puede ver sus datos y puede actualizarlos.

Mi perfil

Autores

Libros

Historial de compras

Reportes

Cerrar sesión

¡Bienvenido a Book Store USAC!

Aquí puedes ver y actualizar tu perfil.

Mi Perfil

Nombre: Milton Manuel

Apellido: Vandert Been

Email: vandertbeen8526@hotmail.com

Teléfono: 45656591

Dirección: Alemania, Alemania

Rol: Client

Editar

Agregar Nuevo Carrito

Resumen de la Orden

No tienes ningún carrito actualmente.



## ¡Bienvenido a Book Store USAC!

Aquí puedes ver y actualizar tu perfil.

### Mi Perfil

### Vista de Carrito

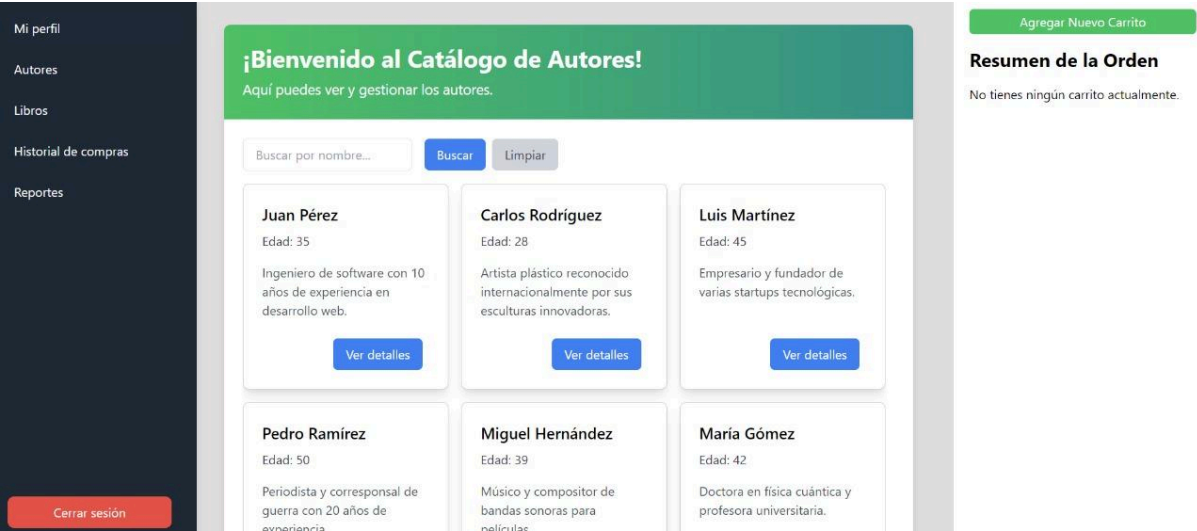
Cuando un usuario acceda no tendrá un carrito de compras, deberá crearlo para después poder agregar productos.

## Resumen de la Orden

No tienes ningún carrito actualmente.

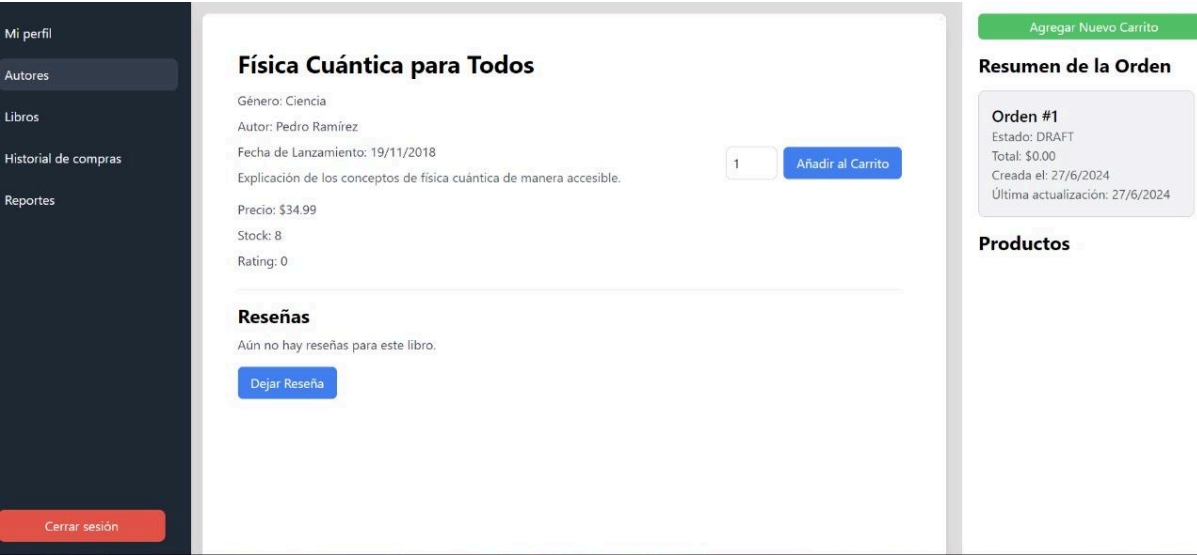
### Vista de Catálogo de Autores

Vista donde se muestran a todos los autores, donde se puede ver su información y también se puede buscar algún autor y también ordenar por filtros.



### Vista Libro

Vista donde se puede ver toda la informacion del libro así como se puede agregar al carrito y tambien se puede dejar una reseña.



### Vista Un Solo Autor

Vista donde se puede ver toda la informacion del autor.

Mi perfil

Autores

Libros

Historial de compras

Reportes

Cerrar sesión

Juan Pérez

Edad: 35

Ingeniero de software con 10 años de experiencia en desarrollo web.

Libros

Reportajes de Guerra

Ocultar detalles

Género: Periodismo

Fecha de lanzamiento: 4/12/2020

Disponibilidad: Disponible

Stock: 7

Precio: \$27.99

Descripción: Compilación de los reportajes más impactantes de zonas de conflicto.

Periodismo de Investigación

Ver detalles

Conflictos Internacionales

Ver detalles

Crónicas de Guerra

Ver detalles

Agregar Nuevo Carrito

Resumen de la Orden

No tienes ningún carrito actualmente.

Vista Catálogo de Libros

Vista donde se pueden ver todos los libros.

Mi perfil

Autores

Libros

Historial de compras

Reportes

Cerrar sesión

¡Bienvenido al Catálogo de Libros!

Aquí puedes ver y gestionar libros.

Buscar por título...

Buscar

Limpiar

Todos los Géneros

Ordenar por

Moda

Diseño de Moda Contemporáneo

Género: Moda

Autor: María Gómez

Precio: \$29.99

Rating: 0

Exploración de las tendencias y técnicas en el diseño de moda actual.

Ver Detalles

Tendencias de la Moda

Género: Moda

Autor: María Gómez

Precio: \$32.99

Rating: 0

Análisis de las tendencias emergentes en el mundo de la moda.

Ver Detalles

Diseño Sostenible

Género: Moda

Autor: María Gómez

Precio: \$26.99

Rating: 0

Guía para el diseño de moda sostenible y eco-friendly.

Ver Detalles

Agregar Nuevo Carrito

Resumen de la Orden

No tienes ningún carrito actualmente.

Vista Libro Agregado al Carrito

Mi perfil

Autores

Libros

Historial de compras

Reportes

Cerrar sesión

Física Cuántica para Todos

Género: Ciencia

Autor: Pedro Ramírez

Fecha de Lanzamiento: 19/11/2018

Explicación de los conceptos de física cuántica de manera accesible.

Precio: \$34.99

Stock: 8

Rating: 0

1

Añadir al Carrito

Reseñas

Aún no hay reseñas para este libro.

Dejar Reseña

Agregar Nuevo Carrito

Resumen de la Orden

Orden #1

Estado: DRAFT

Total: \$34.99

Creada el: 27/6/2024

Última actualización: 27/6/2024

Productos

Física Cuántica para Todos

Explicación de los conceptos de física cuántica de manera accesible.

Cantidad: 1

Total: \$34.99

Autor: Pedro Ramírez

Eliminar

Editar

Confirmar Orden

Vista de Reportes



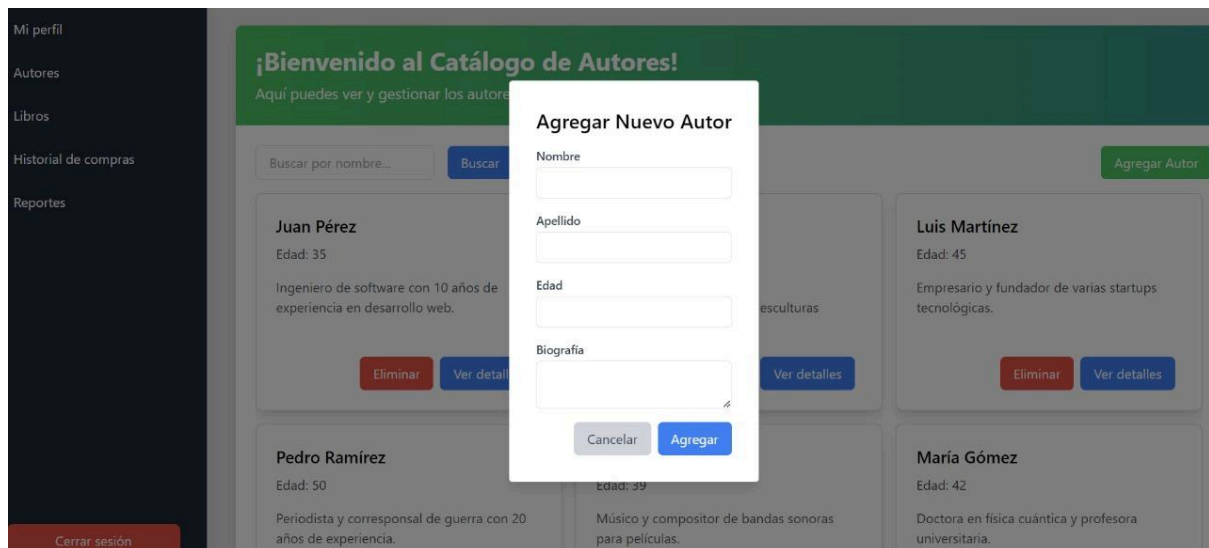
Vista Historial de Compras

Vista donde se puede ver el historial de compras del usuario asi como se puede cambiar el estado de estas.



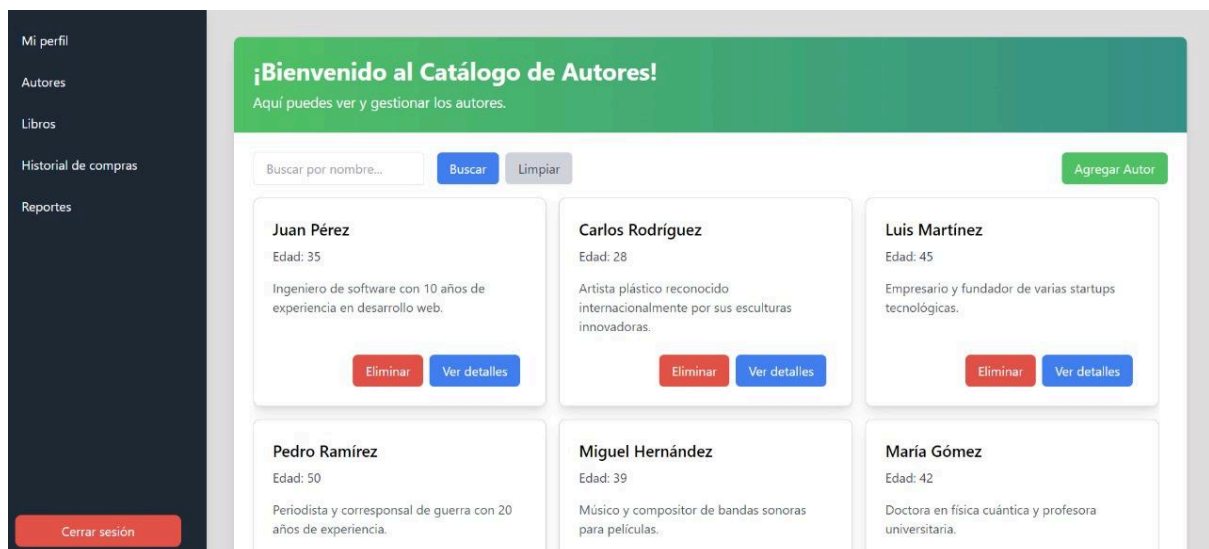
Vista Administrador

Agregar Autor



## Vista Catalogo de Autores Administrador

Vista donde se pueden eliminar a los autores, así como ver su información y agregar nuevos.

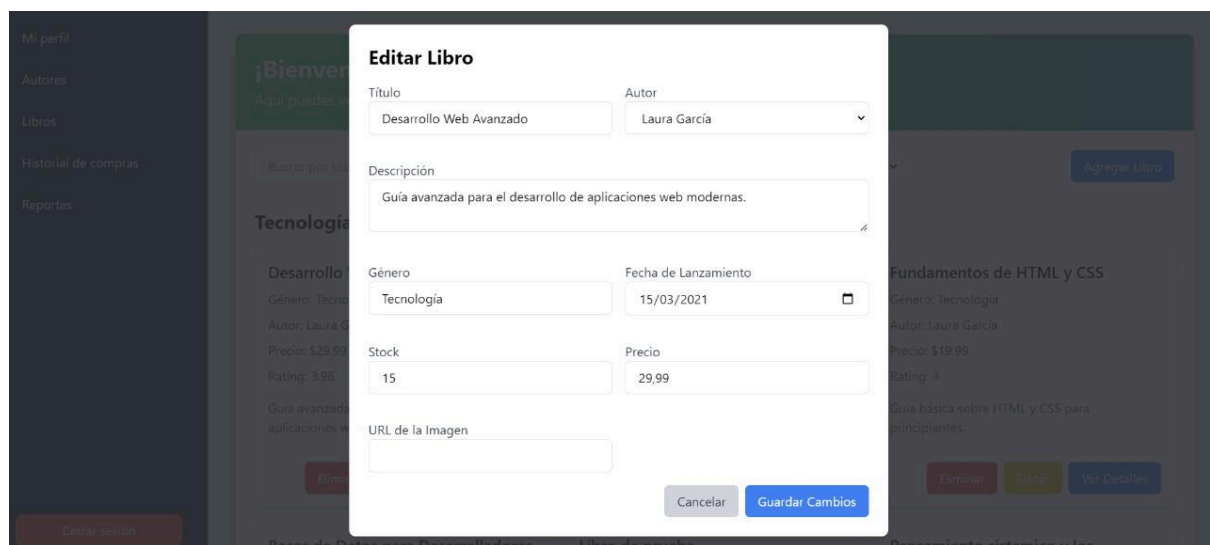


## Vista Catálogos de Libros

Vista donde se pueden agregar libros, editar libros y eliminar libros.



## Vista para Editar un Libro



## Vista Reportes

