Aplicación “La huerta” - Backend

Datos del estudiante
Creación de la base de datos MySQL
    Entidades
    Diagrama
Aplicación express
    Database
    Models
    Routes y Controllers
        userSessionRouter
        shoppingCartRouter
        notificationRouter
        orderRouter
        productRouter
    Rutas del backend

Datos del estudiante

Este taller fue elaborado por Luis Danilo Juajinoy Gálvez con código estudiantil 2130341182.

El código fuente se encuentra en el siguiente repositorio

```
https://github.com/LuisDanilo/FruverApp.git
```

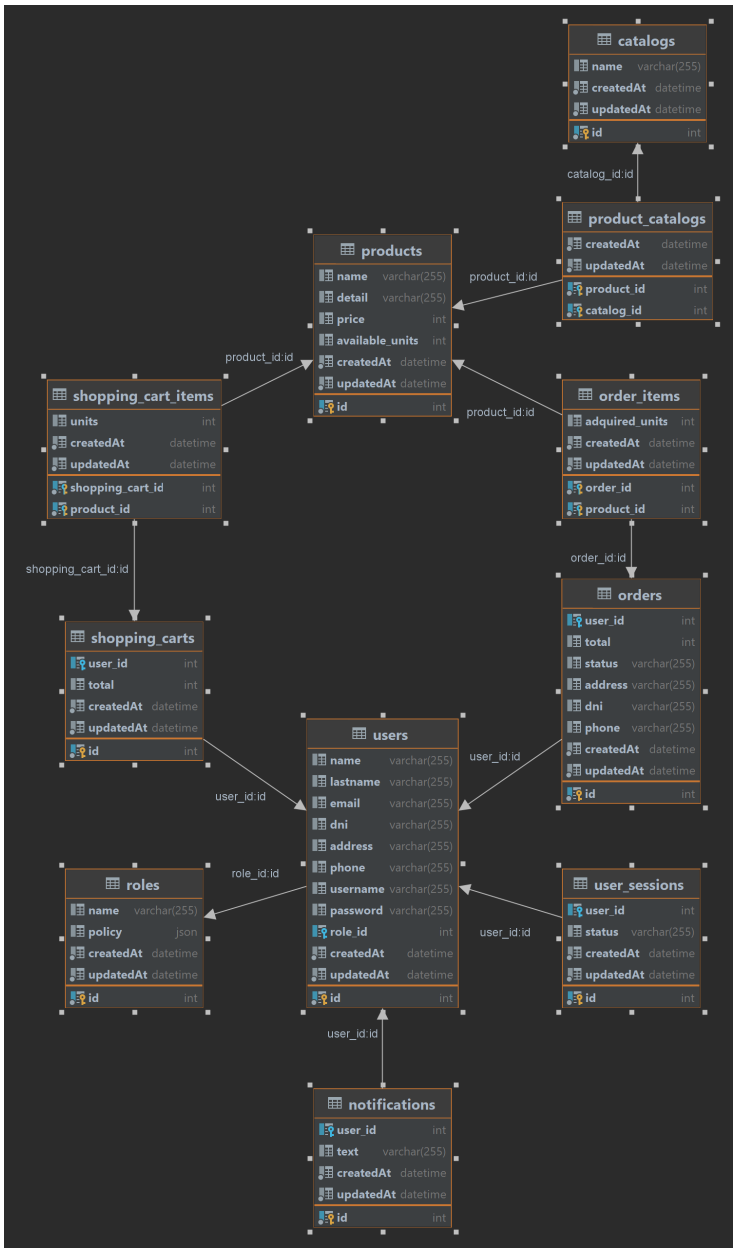
Creación de la base de datos MySQL

El modelado e implementación de la base de datos debe permitir el almacenamiento de datos relacionados a actividades como la visualización de productos, compra de productos, manejo de roles, manejo de carrito de compras, revisión de órdenes, etc.

Entidades

- **Catalogs:** Tabla donde se almacenan los catálogos ofrecidos, cada catálogo consiste en un conjunto de productos.
- **Notifications:** Tabla donde se almacenan las notificaciones a los usuarios creadas por el sistema.
- **Orders:** Tabla donde se almacenan las órdenes creadas por los usuarios, cada orden consiste en un conjunto de productos y los diferentes detalles de la compra como número de productos adquiridos y precio total.
- **Products:** Tabla donde se almacenan los productos ofrecidos y detalles como sus precios unitarios y unidades disponibles.
- **Users:** Tabla donde se almacenan los usuarios del sistema y detalles como información personal y credenciales de ingreso (por facilidad del demo).
- **Shopping carts:** Tabla donde se almacenan los carritos de compra de los usuarios, cada carrito de compra consiste en un conjunto de productos y los diferentes detalles de la compra deseada como número de productos adquiridos y precio total.
- **Roles:** Tabla donde se almacenan los roles de los usuarios y sus respectivos permisos a las diferentes rutas del backend, de este modo los usuarios y administradores están únicamente habilitados para ejecutar acciones respectivas de su rol (EJ: Un usuario no puede administrar órdenes, pero si un administrador).
- **User sessions:** Tabla que almacena las sesiones iniciadas por una persona, con el fin de validar las peticiones posteriores a su ingreso en la aplicación.
- **Shopping cart items:** Tabla intermedia que relaciona un carrito de compras con los productos que este contiene.
- **Product catalogs:** Tabla intermedia que relaciona un catálogo con los productos que este contiene.
- **Order items:** Tabla intermedia que relaciona una orden con los productos que esta contiene.

Diagrama



Aplicación express

Se toma como base la aplicación *express* realizada en clase, por ende conserva la estructura propuesta.

Database

En esta parte se configuran las configuraciones usadas por *sequelize* para la conexión con la base de datos, para esto será necesario **contar con una base de datos MySQL** ejecutándose bajo las configuraciones:

- **host:** localhost
- **puerto:** 3306
- **usuario:** admin
- **contraseña:** Admin123

- **base de datos:** fruverdb

Models

Tal como fue propuesto antes, aquí se crean los modelos referentes a la estructura de la base de datos de la forma que *sequelize* requiere

Routes y Controllers

Se programan los controladores que contienen las funciones que hacen el trabajo principal del backend, es decir consultar datos, procesarlos y retornarlos, de modo que se asigna controlador para cada ruta creada

Para esta aplicación se dividieron las diferentes rutas se la siguiente manera

userSessionRouter

Aquí se crearon las rutas necesarias para autenticación, los controladores de esta ruta contienen las siguientes funciones

```
/**
 * Función que realiza login a un usuario.
 * Retorna al usuario algunos de sus datos y el sessionId si el login fue exitoso, error en caso contrario.
 */
export const performLogin = async (req, res) => {
    try {
        // Recuperar credenciales
        const { username, password } = req.body
        // Buscar usuario con las credenciales anteriores
        const user = await User.findOne({
            where: { username, password }
        })
        if (!user) {
            // Usuario no encontrado
            throw new Error('User not found using given credentials')
        }
        // Usuario encontrado, crear una nueva sesión
        const newSession = await UserSession.create({
            user_id: user.id, status: 'ACTIVE'
        })
        // Información del usuario como respuesta
        res.status(200).json({
            sessionId: `${newSession.id}`,
            roleId: `${user.role_id}`,
            username: user.username,
            address: user.address,
            dni: user.dni,
            phone: user.phone
        })
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

```
/**
 * Función que realiza logout a un usuario.
 */
export const performLogout = async (req, res) => {
    try {
        // Actualizar la sesión del usuario a inactiva (soft delete)
        const updated = await UserSession.update(
            { status: 'INACTIVE' },
            { where: { id: req.sessionId } })
        if (!updated) {
            // Sesión no actualizada
            throw new Error('Couldnt end session')
        }
        res.status(204).json({ message: 'bye' })
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

shoppingCartRouter

Aquí se crearon las rutas necesarias para la funcionalidad de carrito de compras

```
/**
 * Función que agrega un producto al carrito de compras del usuario.
 * Retorna un listado de productos.
 */
export const addShoppingCartItem = async (req, res) => {
    try {
        // Recuperar producto a agregar
        const product = await Product.findByPk(req.body.productId)
        const shoppingCartId = req.user.dataValues.shopping_cart.dataValues.id
        if (!product || product.dataValues.available_units < req.body.desiredUnits) {
            // Error al querer agregar el producto sin unidades disponibles
            throw new Error("Couldnt add product to shopping cart")
        }
        // Crear el producto en el carrito de compras
        await ShoppingCartItem.create({
            shopping_cart_id: shoppingCartId,
            product_id: req.body.productId,
            units: req.body.desiredUnits
        }).catch(err => {
            if (err.name === "SequelizeUniqueConstraintError") {
                // Se intenta agregar un producto que ya existe en el carrito
                // Se procede a actualizar la cantidad de items del producto existente
                return ShoppingCartItem.update(
                    { units: literal(`units + ${req.body.desiredUnits}`) },
                    { where: { shopping_cart_id: shoppingCartId, product_id: req.body.productId } }
                )
            }
            return Promise.reject(err)
        })
        // Actualizar las unidades disponibles del producto
        await Product.update(
            { available_units: literal(`available_units - ${req.body.desiredUnits}`) },
            { where: { id: req.body.productId } }
        )
        // Actualizar el total del carrito
        await ShoppingCart.update(
            { total: literal(`total + (${product.dataValues.price} * ${req.body.desiredUnits})`) },
            { where: { id: shoppingCartId } }
        )
        res.status(204).send()
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

```
/**
 * Función que recupera un listado de productos que estén en el carrito del usuario.
 * Retorna un listado de productos.
 */
export const getShoppingCartItems = async (req, res) => {
    try {
        // Consultar productos del carrito de compras
        const shoppingCart = await ShoppingCart.findOne({
            where: { user_id: req.user.dataValues.id },
            include: [
                { model: ShoppingCartItem, include: [{ model: Product }] }
            ]
        })
        // Mapear productos consultados para retornar una respuesta más limpia
        res.status(200).json(shoppingCart.dataValues.shopping_cart_items.map(ci => ({
            name: ci.dataValues.product.dataValues.name,
            detail: ci.dataValues.product.dataValues.detail,
            price: ci.dataValues.product.dataValues.price,
            adquired_units: ci.dataValues.units
        })))
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

notificationRouter

Aquí se crearon las rutas necesarias para la funcionalidad de notificaciones

```
/**
 * Función para obtener las notificaiones de un usuario.
 * Retorna un listado de notificaciones
 */
```

```
export const getNotifications = async (req, res) => {
    try {
        const notifications = await Notification.findAll({
            where: { user_id: req.user.id }
        })
        res.status(200).json(notifications)
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

```
/**
 * Función que borra una notificación.
 */
export const deleteNotification = async (req, res) => {
    try {
        await Notification.destroy({
            where: { id: req.query.notificationId }
        })
        res.status(204).send()
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

orderRouter

Aquí se crearon las rutas necesarias para las funcionalidades relacionadas a órdenes

```
/**
 * Función que obtiene todas las órdenes.
 * Retorna un listado de órdenes.
 */
export const getOrders = async (req, res) => {
    try {
        // Recuperar órdenes
        const orders = await Order.findAll({
            include: [
                { model: User },
                { model: OrderItem, include: [{ model: Product }] }
            ]
        })
        // Formatear la respuesta
        res.status(200).json(orders.map(o => ({
            id: o.id,
            total: o.total,
            status: o.status,
            user: o.user.name + ' ' + o.user.lastname,
            delivery_address: o.user.address,
            no_available_products: o.order_items.some(oi => oi.product.available_units <= 0)
        })))
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: err })
    }
}
```

```
/**
 * Función que obtiene todos los productos de una orden.
 * Retorna un listado de productos.
 */
export const getOrderItems = async (req, res) => {
    try {
        // Recuperar parámetros
        const { order: orderId } = req.query
        // Recuperar productos de la orden
        const orderItems = await OrderItem.findAll({
            where: { order_id: orderId },
            include: [{ model: Product }]
        })
        // Formatear la respuesta
        res.status(200).json(orderItems.map(oi => ({
            name: oi.product.name,
            detail: oi.product.detail,
            price: oi.product.price,
            adquired_units: oi.adquired_units
        })))
```

```
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: err })
    }
}
```

```
/**
 * Función que crea una orden y notifica al usuario.
 */
export const createOrder = async (req, res) => {
    try {
        // Recuperar los parámetros
        const { address, dni, phone } = req.body
        const shoppingCartId = req.user.dataValues.shopping_cart.dataValues.id
        // Recuperar el carrito del usuario autenticado
        const shoppingCart = await ShoppingCart.findOne({
            where: { id: shoppingCartId },
            include: [
                { model: ShoppingCartItem }
            ]
        })
        // Crear una nueva orden
        const order = await Order.create({
            user_id: req.user.id,
            total: shoppingCart.dataValues.total,
            status: 'IN_PROGRESS',
            address, dni, phone
        })
        // Agregar los productos del carrito a la orden
        shoppingCart.dataValues.shopping_cart_items.forEach(async ci => {
            await OrderItem.create({
                order_id: order.id,
                product_id: ci.dataValues.product_id,
                adquired_units: ci.dataValues.units
            })
        })
        // Vaciar el carrito
        await ShoppingCartItem.destroy({
            where: { shopping_cart_id: shoppingCartId },
        })
        // Reiniciar el conteo total del carrito
        await ShoppingCart.update(
            { total: 0 },
            { where: { id: shoppingCartId } }
        )
        // Crear notificación (al usuario que generó la orden)
        await Notification.create({
            user_id: req.user.id,
            text: `Tu orden #${order.id} fue creada existósamente`
        })
        res.status(204).send()
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

```
/**
 * Función que actualiza el estado de una orden y notifica al usuario.
 */
export const updateOrder = async (req, res) => {
    try {
        // Mensaje amigable para la notificación del usuario
        let friendlyOrderStatus
        if (req.body.status === 'APPROVED') {
            friendlyOrderStatus = 'APROBADA'
        } else if (req.body.status === 'REJECTED') {
            friendlyOrderStatus = 'RECHAZADA'
        } else {
            friendlyOrderStatus = 'PENDIENTE'
        }
        // Recuperar la orden a actualizar
        const order = await Order.findByPk(req.body.orderId)
        // Actualizar el estado de la orden
        await Order.update(
            { status: req.body.status },
            { where: { id: req.body.orderId } }
        )
        if (req.body.status === 'REJECTED') {
            // Recuperar los productos de la orden
            const orderItems = await OrderItem.findAll({
                where: { order_id: req.body.orderId }
```

```
            })
            // Reembolsar los productos de la orden al inventario
            orderItems.forEach(async oi => {
                await Product.update(
                    { available_units: literal(`available_units + ${oi.dataValues.adquired_units}`) },
                    { where: { id: oi.dataValues.product_id } }
                )
            })
        }
        // Crea la notificación al usuario
        await Notification.create({
            user_id: order.user_id,
            text: `Tu orden #${req.body.orderId} fue actualizada a ${friendlyOrderStatus}`
        })
        res.status(200).send()
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

productRouter

Aquí se crearon las rutas necesarias para la funcionalidad de obtener y filtrar productos

```
/**
 * Función que recupera un listado de productos.
 * Permite consultar productos usando filtros de precio y catálogo.
 * Retorna un listado de productos con los filtros indicados.
 */
export const getProducts = async (req, res) => {
    try {
        // Recuperación de los filtros
        const { catalog, min, max } = req.query
        const minPrice = toNumber(min)
        const maxPrice = toNumber(max)
        const catalogId = toNumber(catalog)
        // Opciones de consulta / filtros
        const options = {
            // Existe parámetro catalog? Filtre productos por catálogo
            ...(catalogId > 0 ? { include: [{ model: ProductCatalog, where: { catalog_id: catalogId } }] } : {}),
            where: {
                [Op.and]: [
                    // Existe parámetro min? Filtre productos por limite inferior
                    (minPrice ? { price: { [Op.gte]: minPrice } } : {}),
                    // Existe parámetro max? Filtre productos por limite superior
                    (maxPrice ? { price: { [Op.lte]: maxPrice } } : {}),
                    // Filtre productos con unidades disponibles
                    { available_units: { [Op.gt]: 0 } }
                ]
            }
        }
        // Consultar productos
        const data = await Product.findAll(options)
        // Mapear productos consultados para retornar una respuesta más limpia
        res.status(200).json(data.map(({ id, name, price, available_units, detail, image }) => ({ id, name, price, available_units, de
    } catch (err) {
        console.error(err)
        res.status(400).json({ message: `${err}` })
    }
}
```

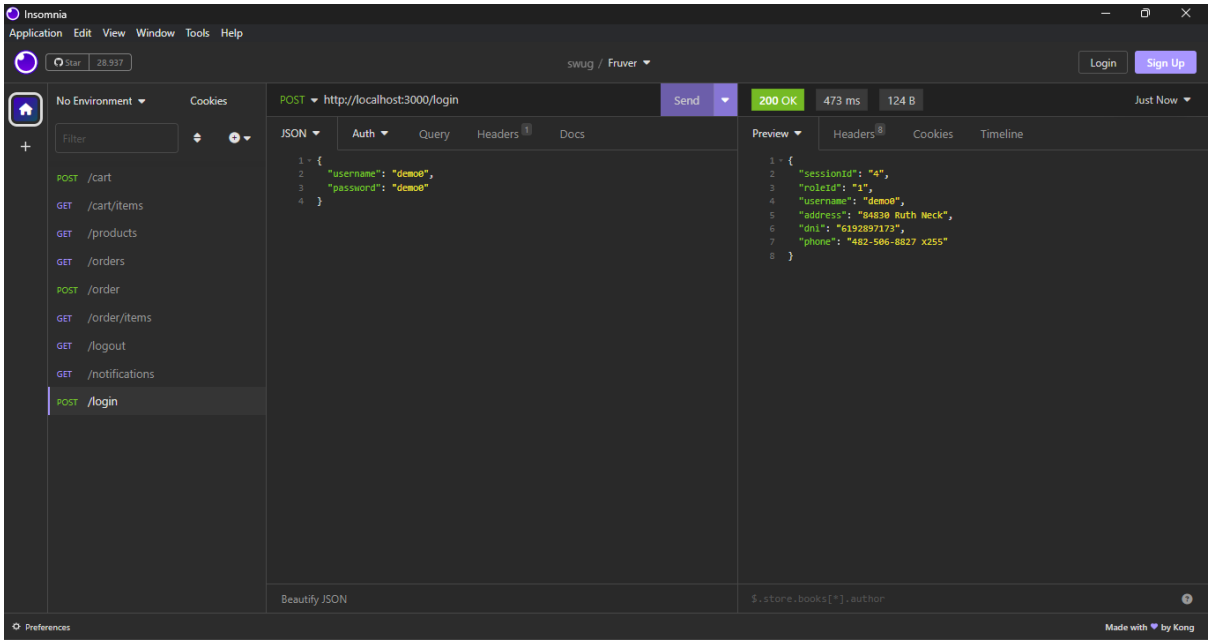
Rutas del backend

Se crearon diferentes rutas para cada funcionalidad requerida tanto del lado de administración como de usuario, además de un sistema de autenticación y autorización con el fin de separar las funcionalidades disponibles para ambos roles.

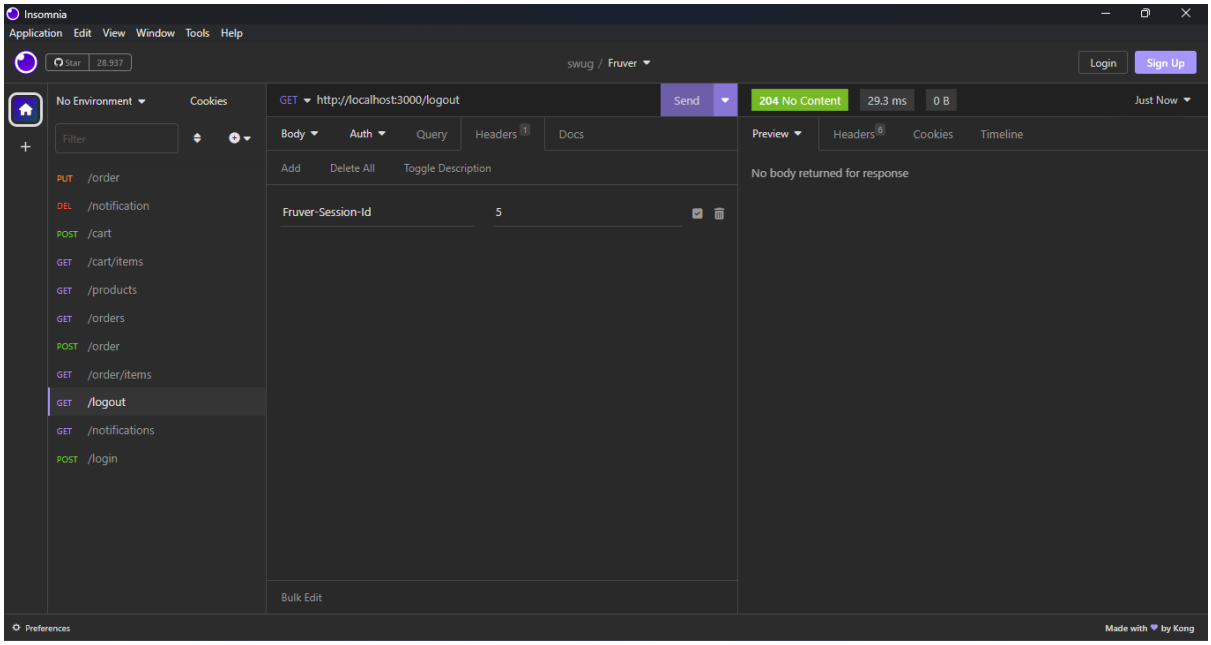
Ruta	Metodo HTTP	Descripción	Requiere autenticación?	Respuesta

Ruta	Metodo HTTP	Descripción	Requiere autenticación?	Respuesta
/login	POST	Ruta designada para que un usuario o administrador pueda ingresar a la aplicación con las credenciales que disponga. Recibe un usuario y contraseña.	No	Retorna información base del usuario autenticado y el **ID de la sesión**, el cual es usado en las demás rutas como mecanismo de autenticación y autorización.
/logout	GET	Ruta designada para que un usuario o administrador pueda terminar su sesión en la aplicación	Si	No tiene retorno (204)
/notifications	GET	Ruta designada para que un usuario pueda obtener sus notificaciones	Si	Retorna un listado de las notificaciones del usuario en sesión
/notification?notificationId=<id>	DELETE	Ruta designada para que un usuario pueda borrar una notificación	Si	No tiene retorno (204)
/orders	GET	Ruta designada para que un administrador pueda listar las órdenes creadas por usuarios	Si	Retorna un listado de las órdenes creadas por los usuarios
/order	POST	Ruta designada para que un usuario pueda crear una orden, la cual posteriormente debe ser aprobada o rechazada por un administrador. Recibe información adicional para la orden (dni, teléfono, dirección de entrega). 💡Los productos a agregar son tomados y/o extraídos del carrito de compras del usuario	Si	No tiene retorno (204)
/products?min=<min>&max=<max>&catalog=<id>	GET	Ruta designada para que un usuario o administrador pueda listar y filtrar los productos ofrecidos	Si	Retorna un listado de los productos registrados, filtrados por rango de precios y catálogo
/order/items?order=<id>	GET	Ruta designada para que un administrador pueda listar los productos de una orden.	Si	Retorna un listado de los productos asociados a la orden indicada
/order	PUT	Ruta designada para que un administrador pueda actualizar el estado de una orden. Recibe el ID de la orden a actualizar y el nuevo estado.	Si	No tiene retorno (204)
/cart	POST	Ruta designada para que un usuario pueda agregar un producto al carrito de compras. Recibe el ID del producto a agregar y las unidades deseadas del mismo.	Si	No tiene retorno (204)
/cart/items	GET	Ruta designada para que un usuario pueda listar los productos de su carrito de compras	Si	Retorna un listado de los productos existentes en el carrito de compras del usuario

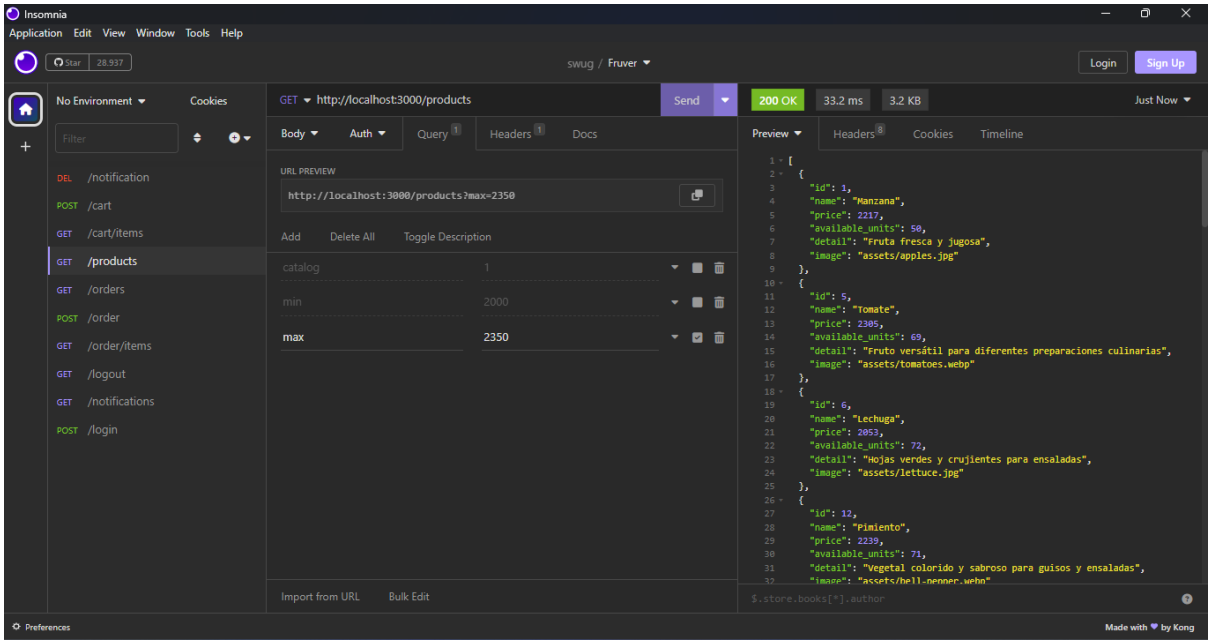
A continuación algunas pruebas realizadas en **Insomnia**



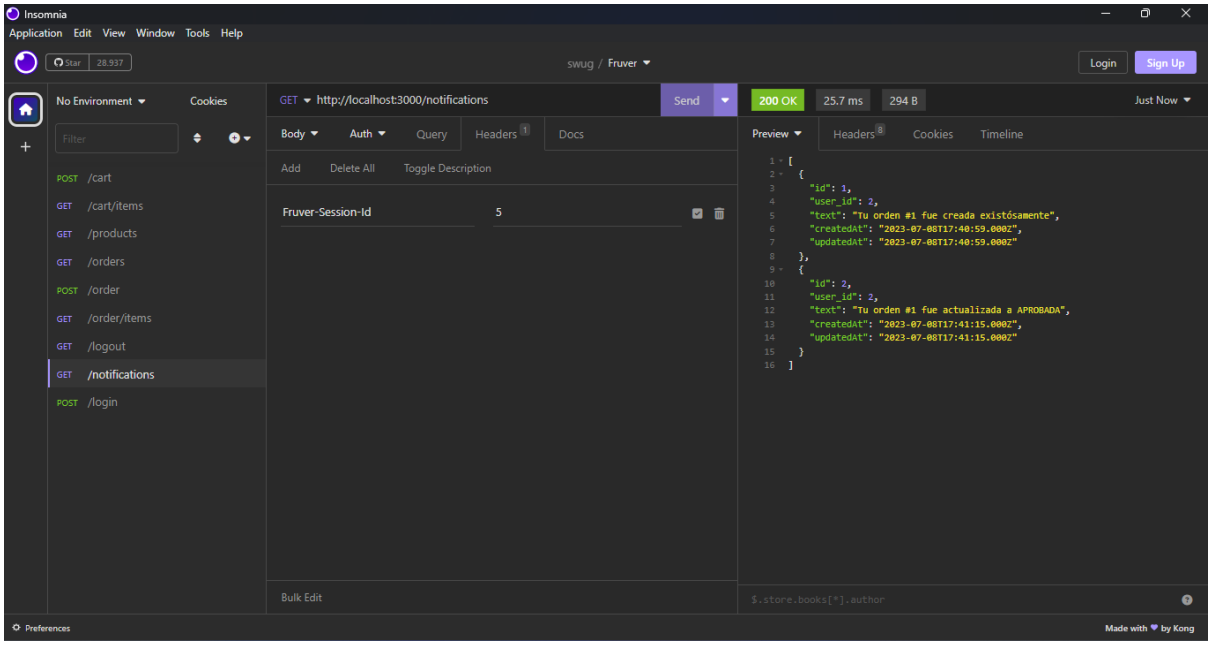
POST /login



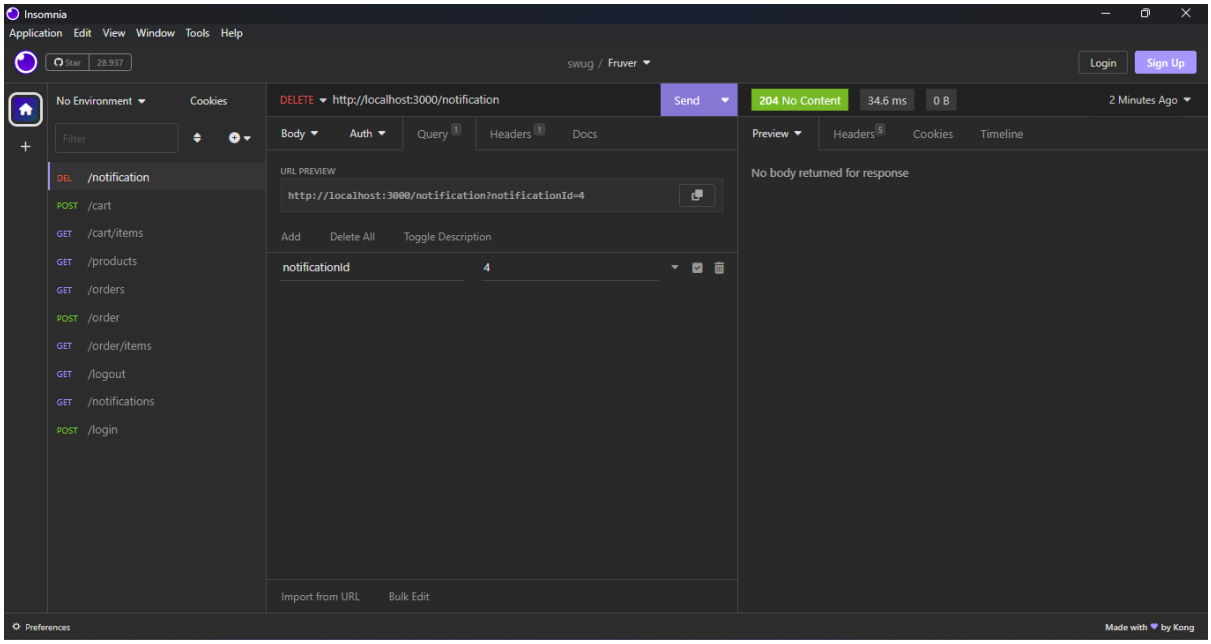
GET /logout



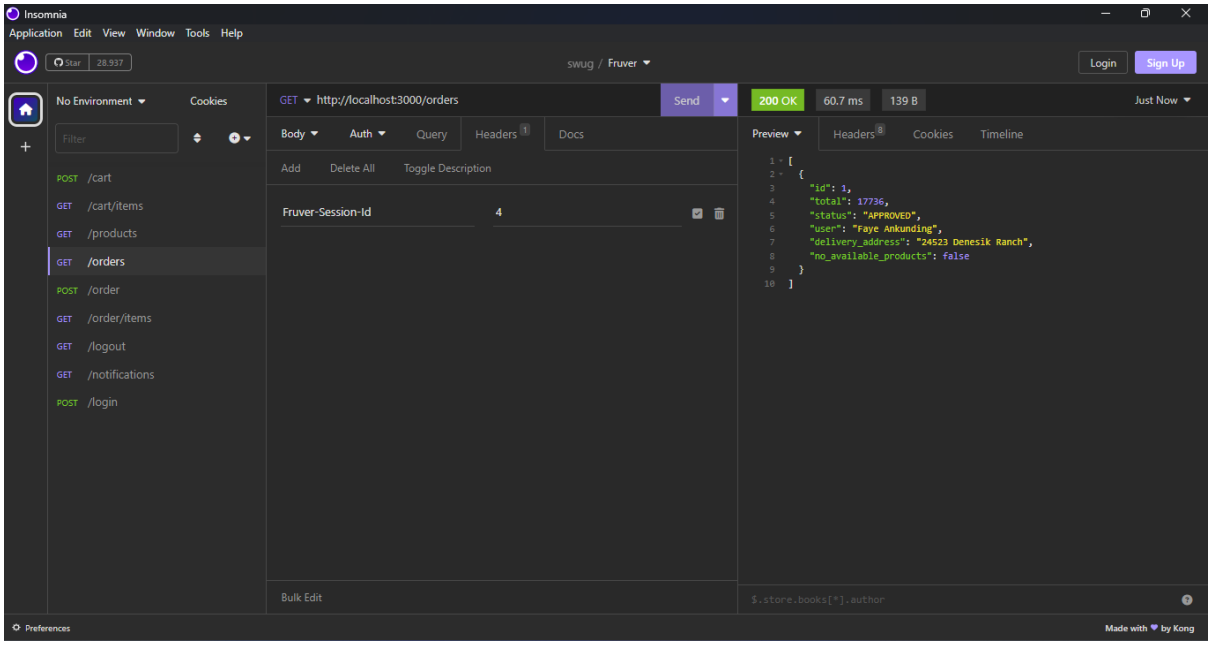
GET /products



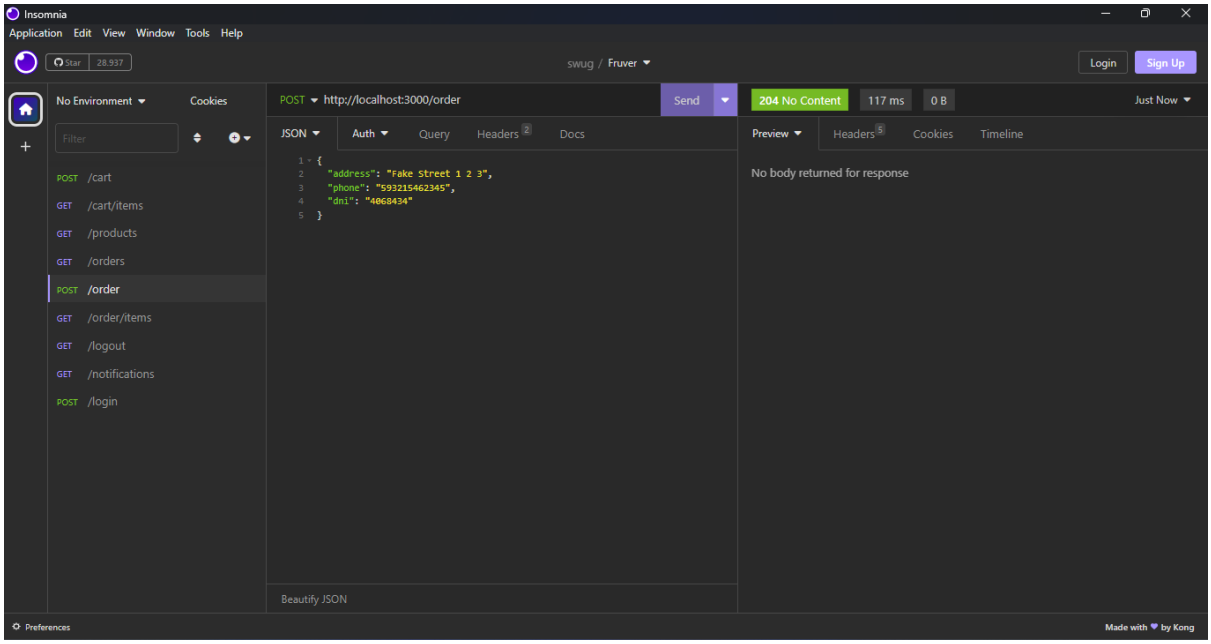
GET /notifications



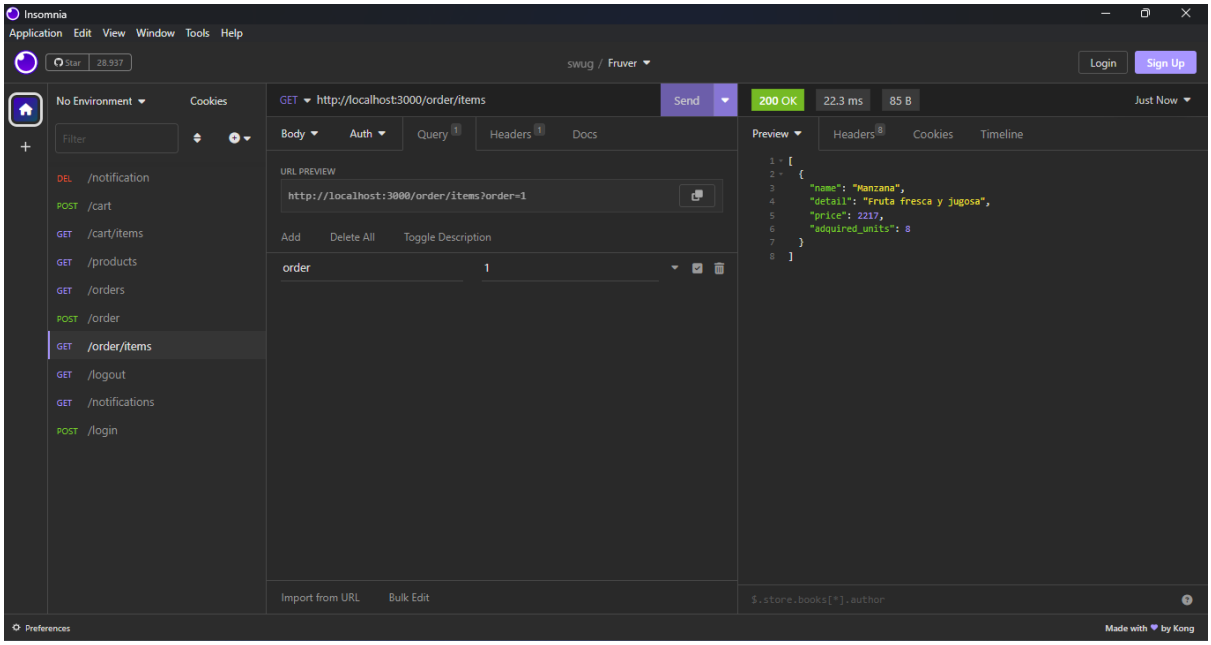
DELETE /notification



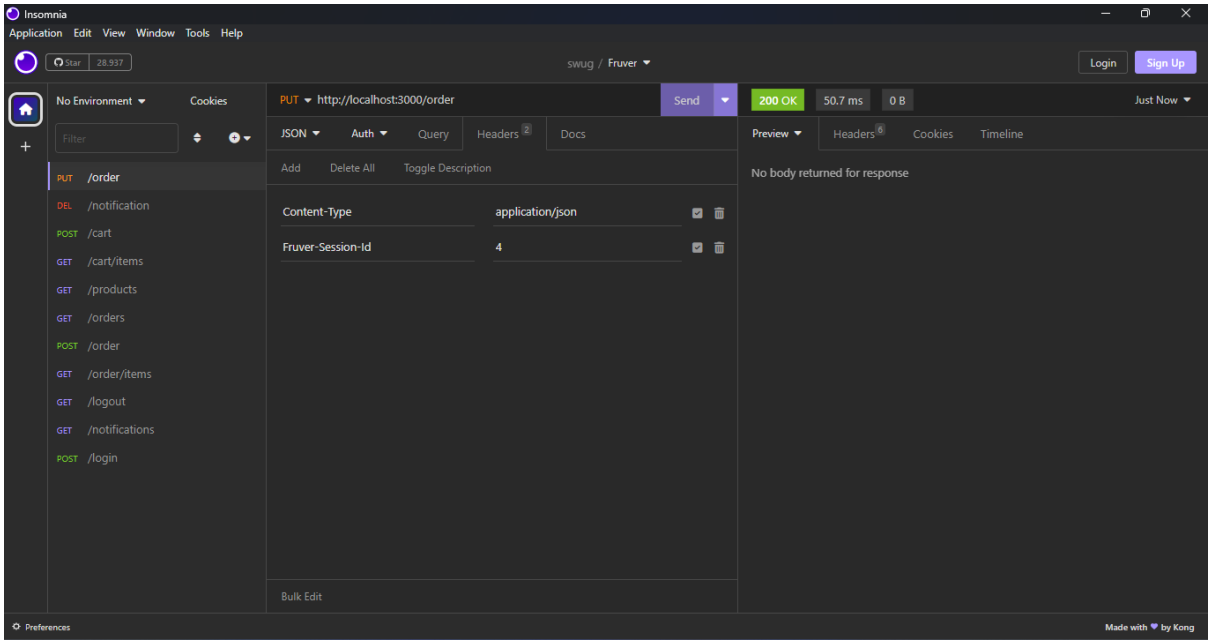
GET /orders



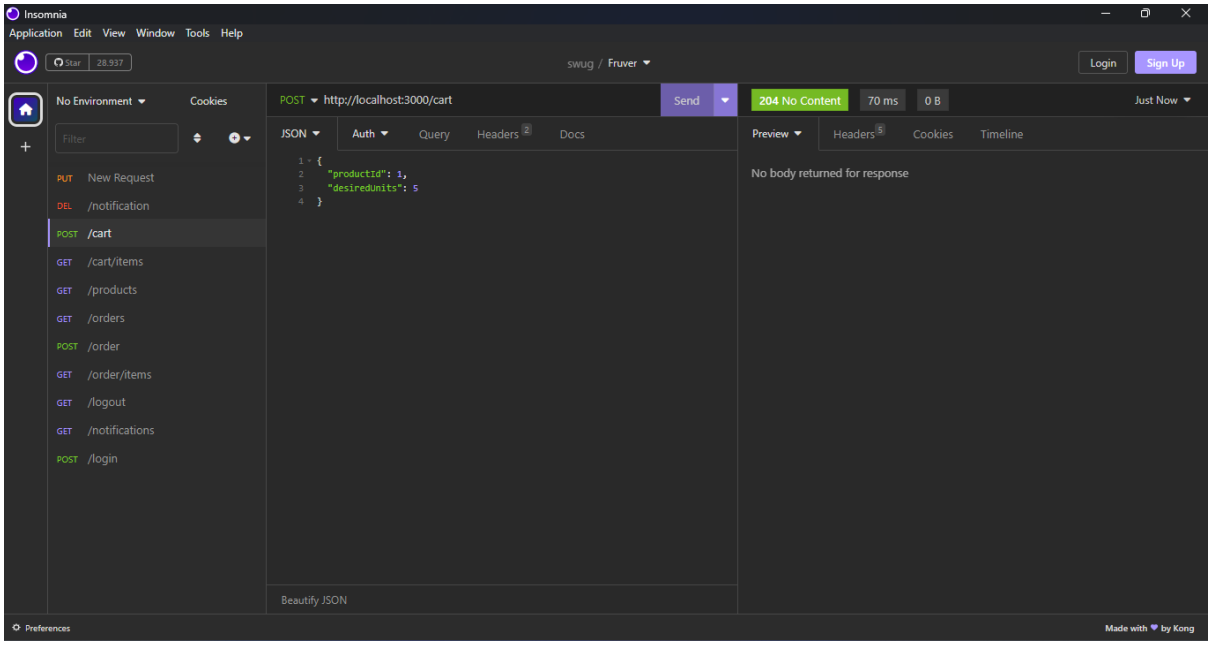
POST /order



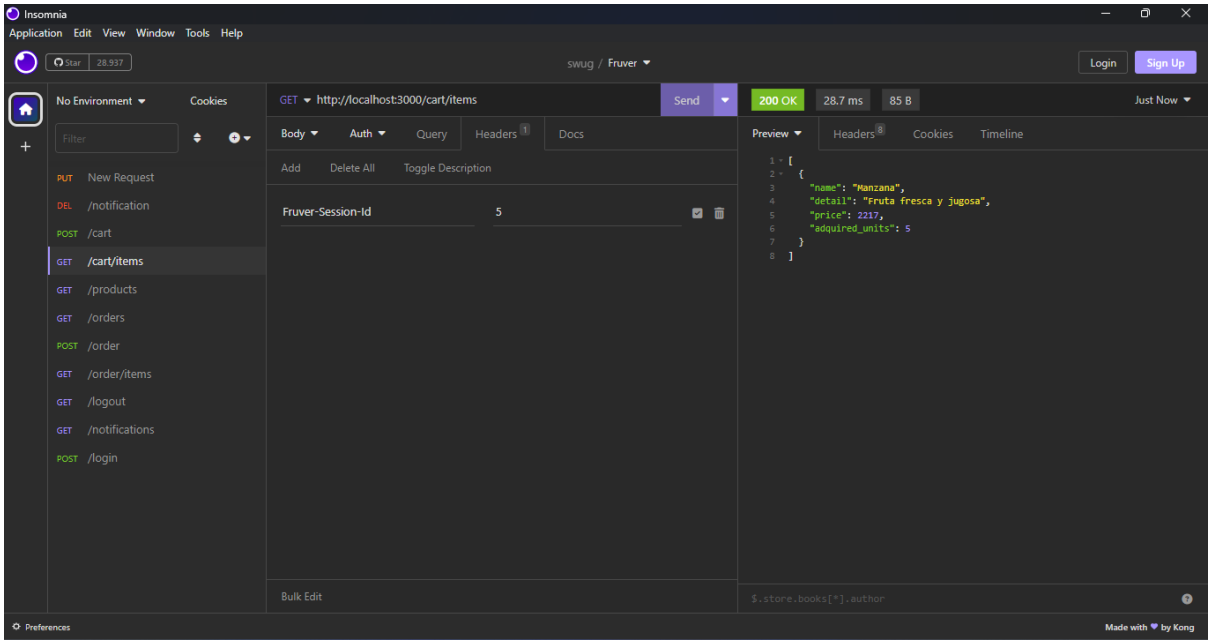
GET /order/items



PUT /order



POST /cart



GET /cart/items