Apuntes sobre el desarrollo:

El proyecto se implementó usando arquitectura limpia y patrones de desarrollo, puntualmente se usaron los patrones **Singleton**, **Repository** y **Model**.

La razón de que en la implementación se haya decidido usar arquitectura limpia e inyección de dependencias es poder utilizar diferentes tecnologías como componentes dentro de la app.

Eso sí, cabe aclarar que la inyección de dependencias en el servicio no se hace desde un módulo o gestor de dependencias como naturalmente se hace. Se hizo desde el constructor por rapidez en cuanto a desarrollo; sin embargo, dentro de la clase se hizo uso de interfaces abstractas para facilitar la migración a módulo y gestor de dependencias.

Mejoras que implementaría:

- Un sistema de Caché con Redis/MemCached. La implementación de caché permitiría a la aplicación escalar sin tener que escalar al mismo nivel la base de datos, teniendo en cuenta que un clúster de Mongo o cualquier base de datos bajo demanda escala bajo demanda como su nombre lo indica, tener un sistema de caché permitiría escalar la aplicación sin escalar tanto la base de datos.
- Manejo de autenticación. La implementación de manejo de autenticación permitiría tener cada carrito asociado con un usuario/persona/entidad de la aplicación, facilitando su uso y la nomenclatura del API de carrito.

Puntos Flacos:

- En la definición no se especifica las propiedades puntuales que tiene cada tipo de item (Evento/Producto) así que se hace uso de un campo de metadatos en una entidad de Item para hacerlo lo más flexible posible.
- No hay un Read de items. El usuario no puede saber que items existen para poder agregarlos al carrito con facilidad, sin embargo se entiende que el API es solo de carrito así que tiene sentido.
- Al no tener autenticación cualquier usuario podría acceder y modificar los items o el carrito de otro usuario, generando una vulnerabilidad.

Gestión de errores

Los errores se gestionan con códigos internos generados por el API para que el frontend pueda gestionarlos de igual manera por medio de API's de localización y traducción. Se brinda en la respuesta del API un detalle del fallo y un código que es el anteriormente mencionado. Ejemplo:

```
{
   "detail": {
       "msg": "The item you trying to add has not enough stock.",
       "code": "CART-003"
   }
}
```

Estructura de URLs o Esquema de la API

Método HTTP	URL	Descripción
`POST`	`/`	Crea un nuevo carrito de compras. Devuelve el carrito creado.
`GET`	`/{id}`	Obtiene los totales y el subtotal del carrito de compras especificado por su ID.
`POST`	`/{id}`	Agrega un nuevo ítem al carrito de compras especificado por su ID.
`PUT`	`/{id}`	Actualiza un ítem existente en el carrito de compras especificado por su ID.
`DELETE`	`/{id}/{item_id}`	Elimina un ítem específico del carrito de compras identificado por el ID del carrito y del ítem.

Descripción del Comportamiento de Cada Endpoint

1. **POST `/`**:
   - **Descripción**: Crea un nuevo carrito de compras.
   - **Función**: Llama al método `create_new_cart` del servicio `CartApiService` para crear un nuevo carrito y devolverlo como respuesta.
   - **Respuesta**: Un objeto `Cart` que representa el carrito recién creado.
2. **GET `/{id}`**:
   - **Descripción**: Obtiene los totales (total y subtotal) del carrito de compras identificado por el parámetro `{id}`.
   - **Función**: Llama al método `get_cart_totals` del servicio `CartApiService` para obtener los totales del carrito especificado.
   - **Respuesta**: Un objeto `CartTotals` que contiene los totales del carrito.
3. **POST `/{id}`**:
   - **Descripción**: Agrega un ítem al carrito de compras especificado por el parámetro `{id}`.
   - **Función**: Llama al método `add_item_to_cart` del servicio `CartApiService`, pasando el `itemId` del cuerpo de la solicitud (`AddItemToCartDTO`), para agregarlo al carrito.
   - **Respuesta**: La respuesta es personalizada dependiendo de la implementación de `add_item_to_cart`.
4. **PUT `/{id}`**:
   - **Descripción**: Actualiza un ítem existente en el carrito de compras especificado por el parámetro `{id}`.
   - **Función**: Llama al método `update_cart_item` del servicio `CartApiService`, pasando el `id` del carrito y los detalles de actualización del cuerpo de la solicitud (`UpdateCartItemDTO`), para actualizar el ítem.
   - **Respuesta**: La respuesta es personalizada dependiendo de la implementación de `update_cart_item`.

5. **DELETE `/{id}/{item_id}`**:

- **Descripción**: Elimina un ítem específico del carrito de compras, identificado por el parámetro `{id}` del carrito y `{item_id}` del ítem.
- **Función**: Llama al método `delete_cart_item` del servicio `CartApiService`, pasando el `id` del carrito y el `item_id` del ítem a eliminar.
- **Respuesta**: La respuesta es personalizada dependiendo de la implementación de `delete_cart_item`.

Clases explicadas (Diagrama de clases):

Clase: `ItemType (Enum)`

Nombre del Atributo	Tipo de Dato	Descripción
`PRODUCT`	`str`	Valor de enumeración "PRODUCT"
`EVENT`	`str`	Valor de enumeración "EVENT"

Clase: `Item (BaseModel)`

Nombre del Atributo	Tipo de Dato	Descripción
`id`	`str`	Identificador del item (alias: `_id`)
`name`	`str`	Nombre del item
`type`	`ItemType`	Tipo del item (`PRODUCT` o `EVENT`)
`price`	`float`	Precio del item
`stock`	`int`	Cantidad de stock disponible
`meta`	`dict[str, Any]`	Metadatos adicionales para el item, pueden variar dependiendo del tipo de item. si es evento, contendrá las propiedades adicionales del evento, si es producto, propiedades adicionales del producto.

Relaciones:

- Utiliza la enumeración `ItemType` para definir el tipo de item.

Clase: `CartItem (Item)`

Nombre del Atributo	Tipo de Dato	Descripción
`id`	`str`	Identificador del item (heredado de `Item`)
`name`	`str`	Nombre del item (heredado de `Item`)
`type`	`ItemType`	Tipo del item (heredado de `Item`)

`price`	`float`	Precio del item (heredado de `Item`)
`stock`	`int`	Cantidad de stock disponible (heredado de `Item`)
`meta`	`dict[str, Any]`	Metadatos adicionales para el item (heredado de `Item`)
`quantity`	`int`	Cantidad del item en el carrito

Relaciones:

- Hereda de `Item`.

Clase: `Cart (BaseModel)`

Nombre del Atributo	Tipo de Dato	Descripción
`id`	`str`	Identificador del carrito (alias: `_id`)
`items`	`list[CartItem]`	Lista de items en el carrito de compras

Relaciones:

- Contiene una lista de objetos `CartItem`.

Relaciones entre Clases

- **`Cart`** contiene una lista de **`CartItem`**.
- **`CartItem`** hereda de **`Item`**.
- **`Item`** utiliza la enumeración **`ItemType`** para definir el tipo de item.