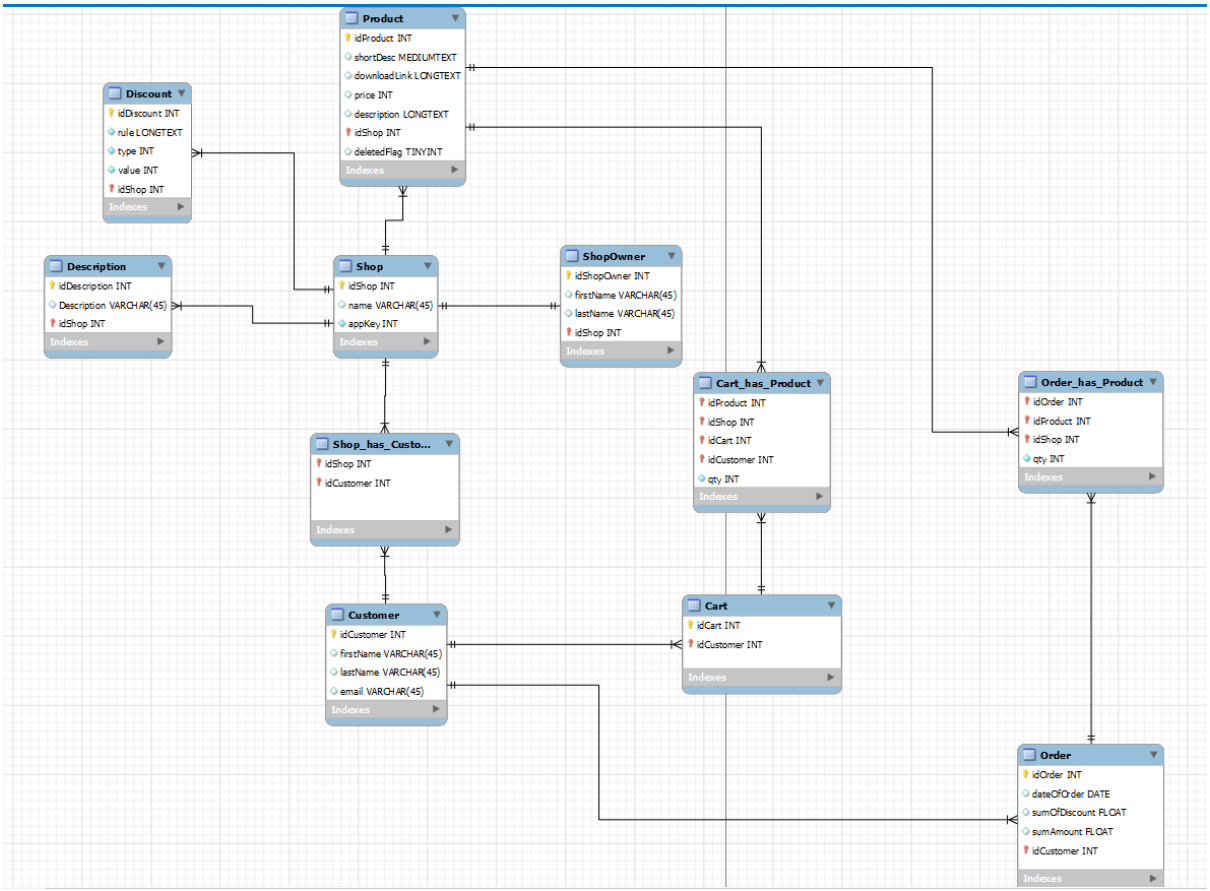
CaaS

Angelos Angelis
11-30-2022

Ausbaustufe 1

Datenbankmodell



Wichtige Design-Entscheidungen:

- AppKey ist ein NotNull und Unique Attribut in der Shop Tabelle
- Man kann nicht einen Cart haben ohne angemeldet zu sein. Das gleiche gilt auch um etwas zu kaufen
- Ein Shop hat mehrere Customer und ein Customer kann in mehreren Shops einkaufen
- Cart und Order sind nicht direkt miteinander in relation. Um eine Order zu erstellen werden im BackEnd die nötigen Daten aus Cart_has_Product und Cart entnommen und im neuen Order & Order_has_Product Eintrag hinzugefügt
- Einträge in den Tabellen Customer, Order, Order_has_Product, Product, Shop dürfen nicht gelöscht werden
- **Discount**: Rule stellt die Regel dar. In dem Fall ist es immer ein SQL IF Statement das ausgelesen und dann ausgeführt wird. Type ist die RabattAktion. In dem Fall falls type 0 ist, ist es eine FixPreis Rabattaktion falls es type 1 ist, ist es prozentueller Preis Nachlass.

Constraints:

- Wenn ein Cart gelöscht wird, werden auch die entsprechende Cart_has_Product Einträge gelöscht

Verbesserungsmöglichkeiten für die nächste Ausbaustufe:

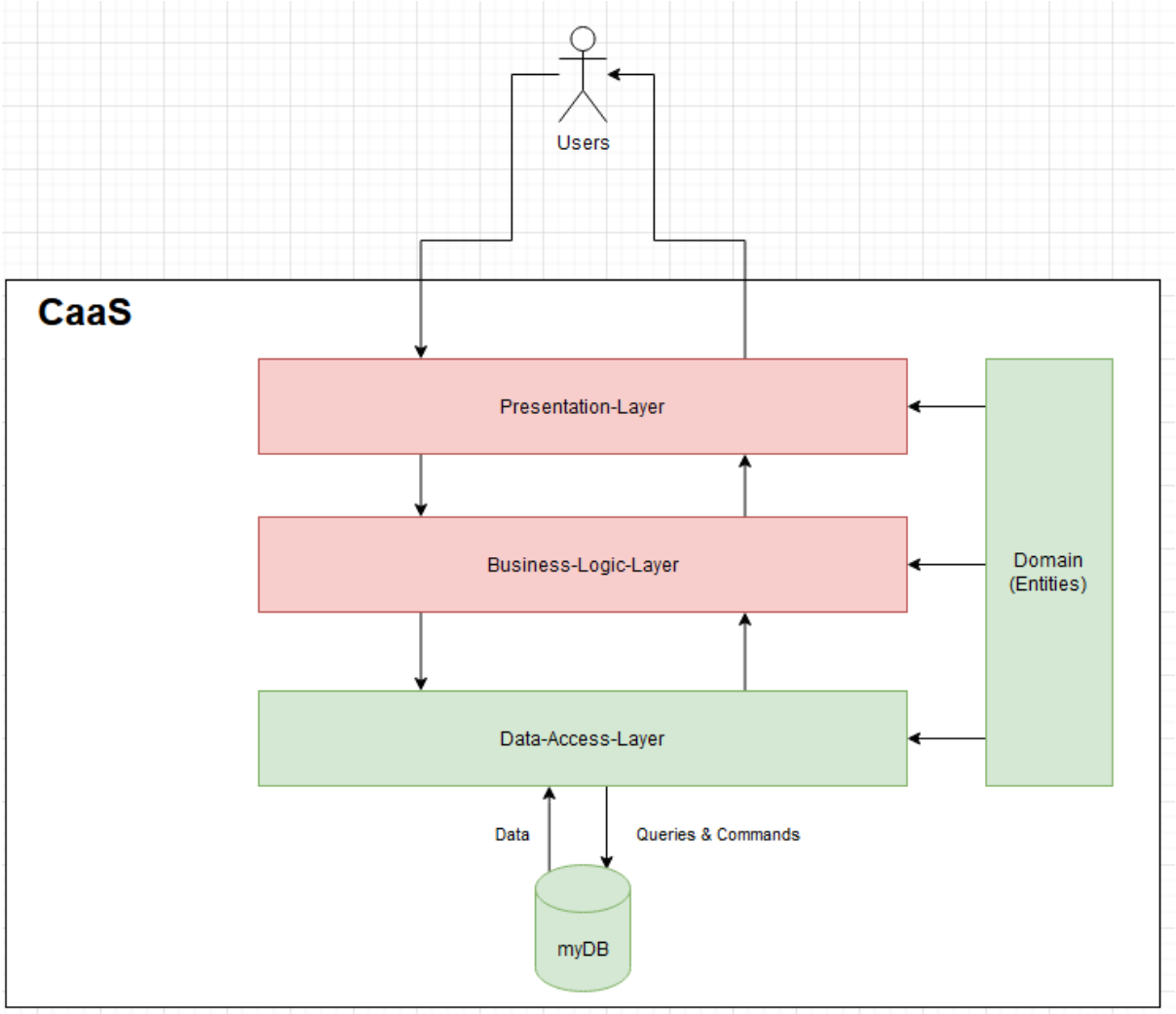
- AppKey zu String
- Ids nicht autoincrement sondern random generated strings.

Create und Insert:

- Es gibt 2 Datenbanken eine ProductionDb und eine TestDb. Die Testdaten für beide wurden mittels Mockaroo erstellt.

Struktur vom Projekt

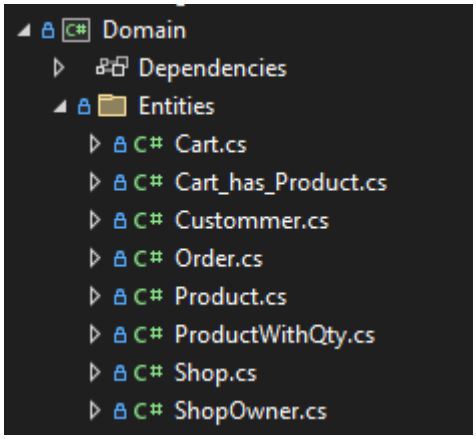
Mein CaaS habe ich in 4 Projekten aufgeteilt



In der ersten Ausbaustufe habe ich die Domain-Layer, die Data-Access-Layer und die Datenbank implementiert und getestet.

Domain-Layer:

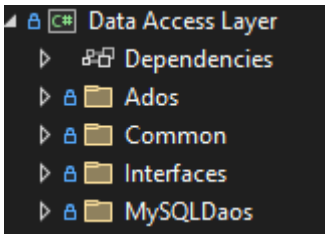
Die Domain Layer besteht aus einem Ordner namens Entities. Dieser enthält alle Transportklassen. Transportklassen sind wichtig da sie zur repräsentation der Daten in der Datenbank dienen.



Hierbei wichtig ist dass ich auch eine Entität ProductWithQty definiert habe die ein Produkt in einem Warenkorb bzw in einer Order darstellen soll.

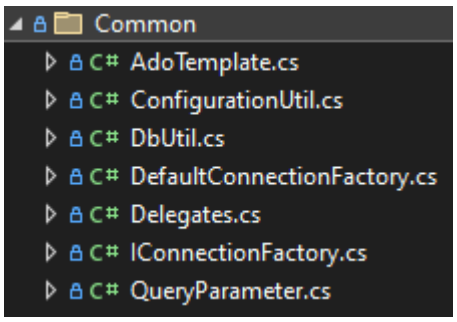
Um diese Objekte von diesen Klassen zu erstellen müssen die Daten die man von der Datenbank bekommt auf die jeweiligen Entitäten gemappt werden. Dafür gibt es dann im DAL Projekt eine Mapper Klasse.

Data-Access-Layer:



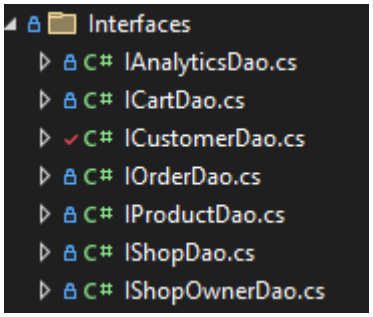
Die DAL soll es ermöglichen so einfach wie möglich auf Daten von der Datenbank zugreifen zu können: In meinem Fall habe ich die DAL in 4 Teile unterteilt:

Common:



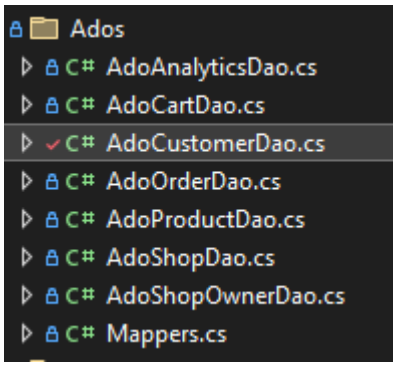
Im Common Ordner findet man die wichtigsten Teile der DAL. Klassen die es uns ermöglichen und im Programm mit der Datenbank zu verbinden und Klassen die es uns ermöglichen async Queries und Commands durchzuführen.

Interfaces:



Im Interfaces Ordner findet man die Intefaces zu allen Dao's. Darin sind die Methodenschnittstellen definiert für die Queries.

Ados:



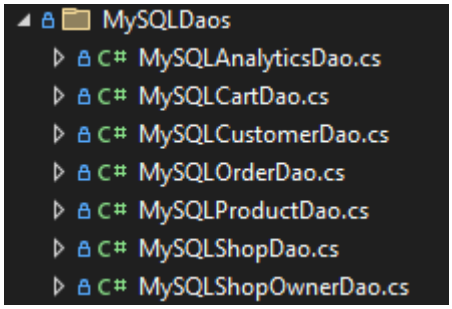
Im Ados Ordner sind die ganzen Dao's gemäß deren Interfaces implementiert. All diese Daos sind abstrakte Klassen die ein AdoTemplate enthalten (sodass man Queries durchführen kann). Die sind abstrakt sodass man egal was für eine Datenbank man benutzt trotzdem die gleiche Funktionalität hat.

Hier ist ein einfaches Beispiel einer insert Query:

```
public virtual async Task<int> InsertAsync(Cart cart)
{
    const string SQL_INSERT = @"insert into Cart (idCustomer) values(@idCustomer)";
    cart.idCart =
        Convert.ToInt32(await template.ExecuteScalarAsync<object>(
            $"{SQL_INSERT};{LastInsertedIdQuery}",
            new QueryParameter("@idCustomer", cart.idCustomer)
            ));
    return cart.idCart;
}
```

- SQL_INSERT definiert die Query
- New QueryParameter ersetzt die „@" im String durch eine Variable
- LastInsertIdQuery gibt die id zurück die hinzugefügt wurde. Diese wird in cart.idCart gespeichert

MySQLDaos:



Die MySQLDaos leiten von den Ados ab. Darin gehört jetzt Datenbankspezifischer code rein. Hier ein Beispiel:

```
public class MySQLCartDao : AdoCartDao
{
    1 reference
    public MySQLCartDao(IConnectionFactory connectionFactory): base(connectionFactory) { }

    3 references
    protected override string LastInsertedIdQuery => "select LAST_INSERT_ID()";
}
```

Mit Datenbankspezifischen code meine ich die LastInstertIdQuery die in einer Tabelle die Letzte eingefügte id zurückgibt. Für jede Datenbank ist diese Query anders weshalb das in den SQLDaos implementiert wurde.

Mapper:

Mapped die Ergebnise einer Query auf die jeweilige Entität und erstellt daraus ein neues Objekt

Beispiel:

```
public static ShopOwner MapRowToShopOwner(IDataRecord row)
{
    return new ShopOwner(idShopOwner_: (int)row["idShopOwner"],
                         firstName_: (string)row["firstName"],
                         lastName_: (string)row["lastName"],
                         idShop_: (int)row["idShop"]);
}
```

Grober Ablauf einer Query:

1.QueryMethode wird in dem entsprechendem Ado definiert -> 2.Ergebnis wird gemapped und ein entsprechendes Objekt wird erstellt. -> 3.Ein MySQLDao wird erstellt mit Hilfe einer connectionFactory -> 4. Das MySQLDao führt dann dann die QueryMethode aus und das Ergebnis wird am besten in ein Objekt von der Entsprechenden Entität gespeichert

Setup:

1.Schritt MySQL Server starten auf localhost mit Port3306 Uid=root pwd=MyNewPass

2.Schritt Datenbank erstellen mit dem CREATE_SCRIPT in .../

3.Schritt Datenbank befüllen mit dem INSERT_SCRIPT in .../

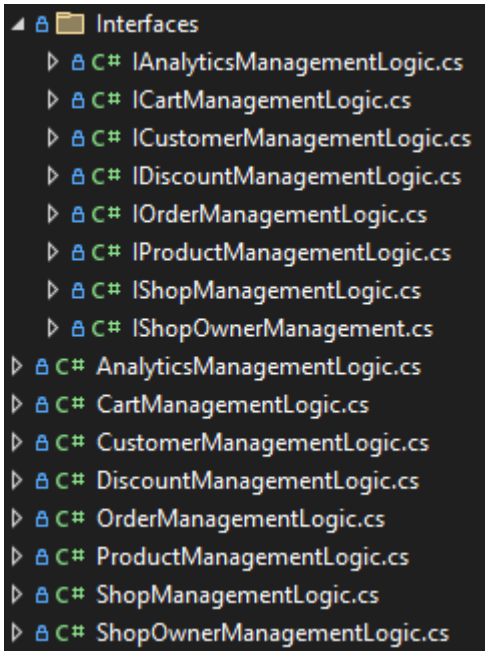
4.Queries per DAL ausführen

Ausbaustufe 2

CaaS.Core

Stellt die Business-Logik des Projektes dar.

Die Logik-Layer besteht aus Interfaces und deren Implementierung.



Am Interessantesten hier ist wahrscheinlich die CartManagementLogic.cs.

```
public async Task<int> CreateOrderFromCart(Cart cart, int customerId)
        {
            //Get all Products From Cart
            List<ProductWithQty> products = (await
cartDao.GetProductsDetailedInCart(cart.idCart)).ToList();
            //check if Cart empty
            if (products.IsNullOrEmpty()) throw new ArgumentException("Your Cart
is Empty");
            //Get All Discount Rules
            Console.WriteLine(products.ElementAt(0).idShop);
            List<Discount> discountList = (await
discountDao.GetDiscountsOfShop(1)).ToList();
            //Check if Rules apply for Products
            float sumPrice = 0;
            float sumDiscount = 0;
            foreach(ProductWithQty product in products)
            {
                sumPrice += product.price * product.qty;
                //Execute Conditions
                foreach (Discount discount in discountList)
                {
                    if (discount.rule.Contains("idProduct"))
                    {
                        if(await commonDao.CheckDiscountCondition(discount.rule,
product.idProduct))
                        {
                            if(discount.type == 0)
                            {
                                //price*qty - discount.value
                                sumPrice -= discount.value;
```

```
                                sumDiscount += discount.value;
                            }
                            else
                            {
                                //price * qty - (price*qty)*(discount.value/100)
                                sumPrice -= ((float)product.price *
(float)product.qty) * ((float)discount.value / 100);
                                sumDiscount += ((float)product.price *
(float)product.qty) * ((float)discount.value / 100);
                            }
                        }
                    }
                    else
                    {
                        if (await commonDao.CheckDiscountCondition(discount.rule,
0))
                        {
                            if (discount.type == 0)
                            {
                                sumPrice -= discount.value;
                                sumDiscount += discount.value;
                            }
                            else
                            {
                                sumPrice -= ((float)product.price *
(float)product.qty) * ((float)discount.value / 100);
                                sumDiscount += ((float)product.price *
(float)product.qty) * ((float)discount.value / 100);
                            }
                        }
                    }
                }
            }
            if (cart == null || await cartDao.CartExist(cart.idCart) == false)
throw new ArgumentException("Cart does not exist");
            //Check if customer owns Cart
            if (await cartDao.CustomerOwnsCart(customerId, cart.idCart) == false)
throw new ArgumentException("You dont own this cart");
            //Console.WriteLine(sumDiscount + " " + sumPrice);
            return await
cartDao.CreateOrderFromCart(cart,sumPrice,sumDiscount);//with sum of price as
param
        }
```

Diese Methode wandelt einen Cart zu einer Order um und berechent den Preis inkl. Preisnachlass.

Ansonsten werden in der Busines-Logic Sachen abgeprüft wie ob die id's stimmen

```
//Check if Cart exists
if (await cartDao.CartExist(cartId) == false) throw new ArgumentException("Cart does not exist");
```

oder z.B auch ob ein valider AppKey angegeben wurde.

```
if (commonDao.CheckAppKeyValidity(product.idShop, AppKey).Result == false) throw new ArgumentException("False AppKey");
```

Grobe Implementierung der ManagementLogic Klassen:

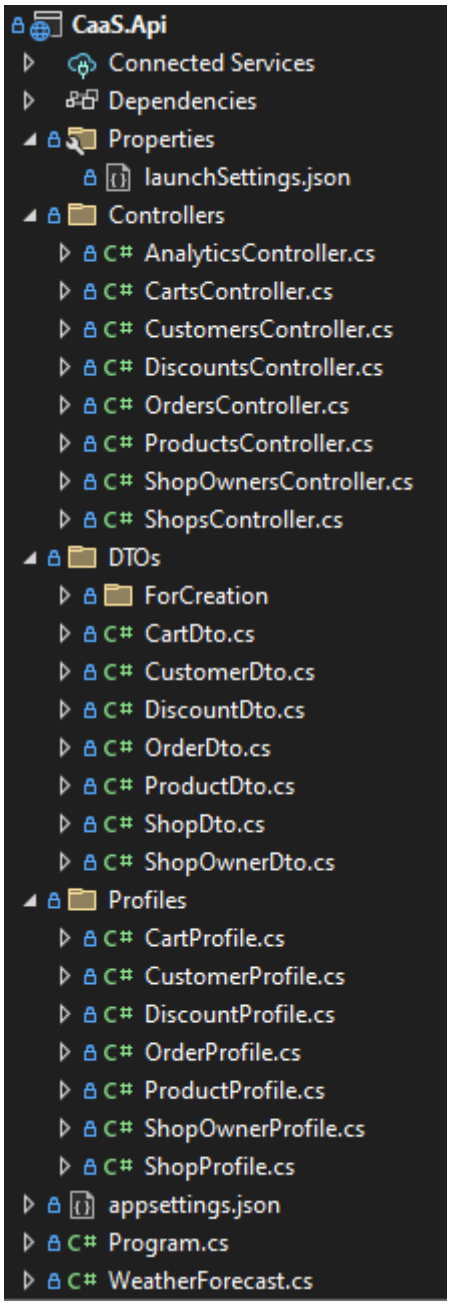
An sich besitzen die Klassen die Dao's die sie brauchen als Properties. Im Konsturktor wird eine Connection zur Datenbank hergestellt und die Dao's werden intialisiert. Ebenfalls hat jede Klasse einen zweiten Konstruktor dieser existiert nur zum Intialisieren der Mocks bei Testen. Hier ein Beispiel:

```
public class CartManagementLogic : ICartManagementLogic
{
        private readonly ICartDao cartDao;
        private readonly IProductDao productDao;
        private readonly ICustomerDao customerDao;
        private readonly IDiscountDao discountDao;
        private readonly ICommonDao commonDao;
        public CartManagementLogic()
        {
            IConfiguration configuration = new
ConfigurationBuilder().AddJsonFile("appsettings.json", optional: false).Build();
            IConnectionFactory connectionFactory =
DefaultConnectionFactory.FromConfiguration("PersonDbConnection");
            this.cartDao = new MySQLCartDao(connectionFactory);
            this.productDao = new MySQLProductDao(connectionFactory);
            this.customerDao = new MySQLCustomerDao(connectionFactory);
            this.discountDao = new MySQLDiscountDao(connectionFactory);
            this.commonDao = new MySQLCommonDao(connectionFactory);
        }

        public CartManagementLogic(ICartDao object1, IProductDao object2,
ICustomerDao object3, IDiscountDao object4, ICommonDao object5)
        {
            this.cartDao = object1;
            this.productDao = object2;
            this.customerDao = object3;
            this.discountDao = object4;
            this.commonDao = object5;
        }

...

}
```

CaaS.Api

Die Rest-API des CaaS-Backends:

Besteht aus 4 Ordnern.



Properties enthält die launchSettings. Da drin ist spezifiziert dass beim Starten der Api, sich ein Tab im Browser mit swagger öffnet.

Der Controllers Ordner enthält die Controller der Api. Ein Controller ist dafür verantwortlich, die Art und Weise zu steuern, wie ein Benutzer mit einer MVC-Anwendung interagiert. Ein Controller enthält die Flusssteuerungslogik für eine die Anwendung. Ein Controller bestimmt, welche Antwort an einen Benutzer zurückgesendet wird, wenn ein Benutzer eine Browseranfrage stellt. Wird im späteren Kapitel noch näher erläutert.

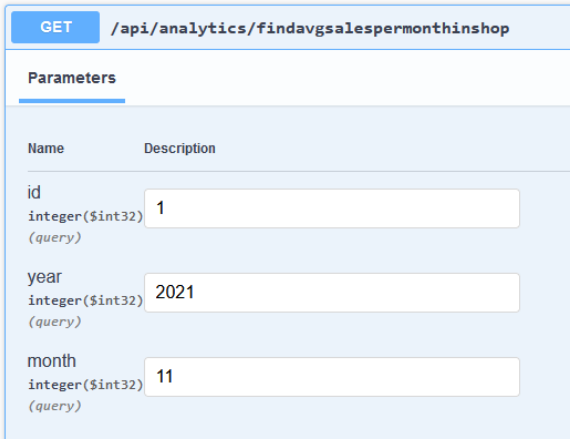
Der Dto´s Ordner enthält die Data Trasfer Objects. Dtos sind Objekte, die Daten zwischen Schichten transportiert

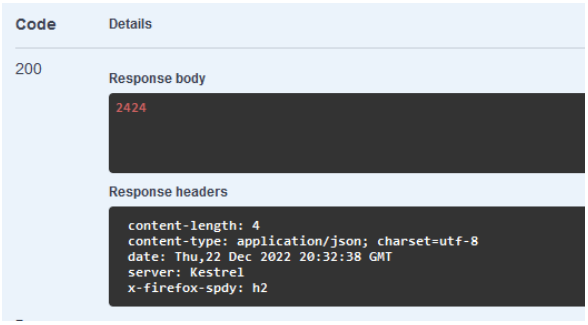
Der Profiles Ordner enthält die „Configs“ für den Automapper damit er die Dto’s auf Domain Objekte mapped.

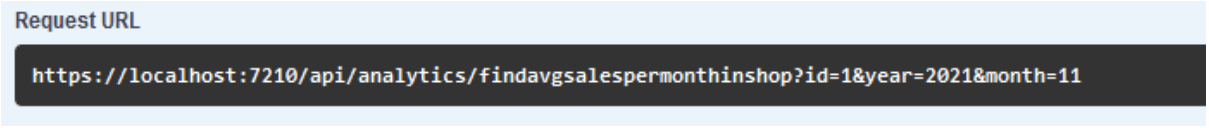
Controller & Endpoints

Analytics:

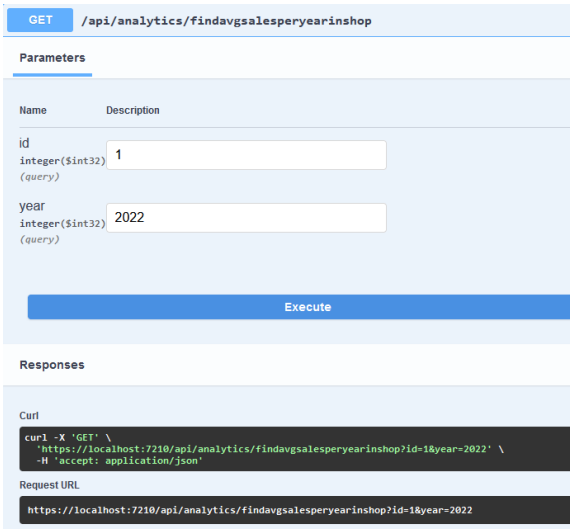
FindAvgSalesPerMonthInShopAsync:

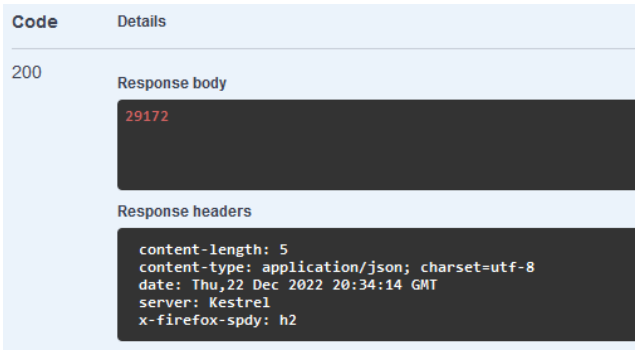




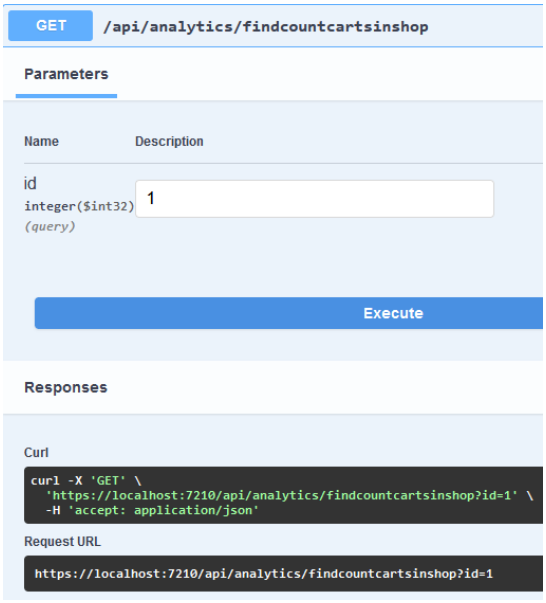


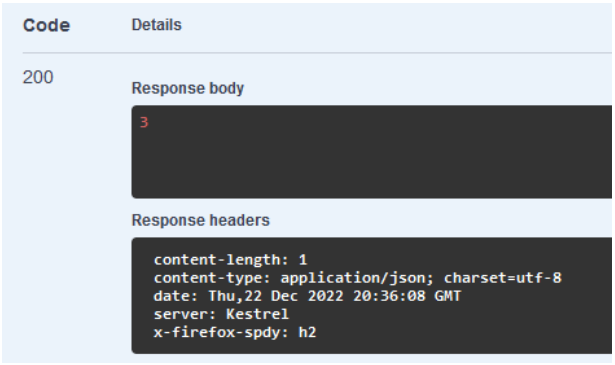
FindAvgSalesPerYearInShopAsync:



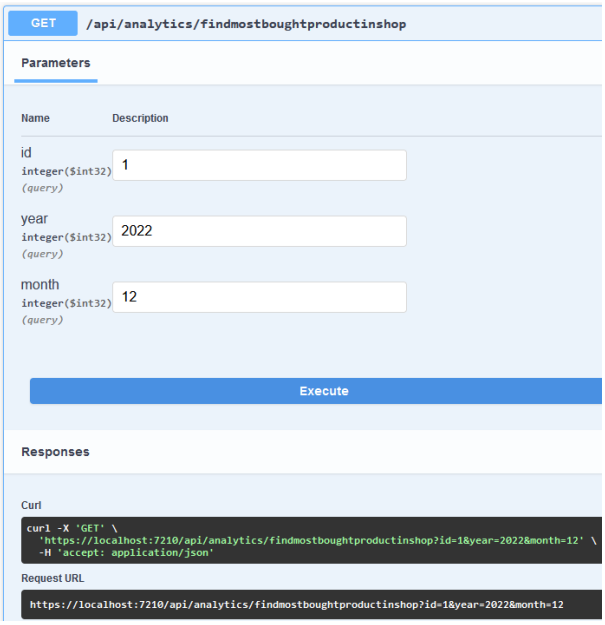


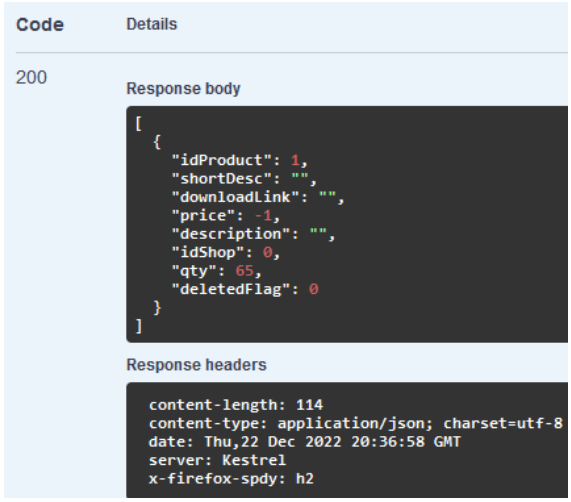
FindCountCartsInShopAsync





FindMostBoughtProductInShopAsync





Carts:

GetCartsOfCustomer

GET /api/carts/customer/{customerId}

Parameters

Name	Description
customerId * required integer($int32) (path)	1

Execut

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7210/api/carts/customer/1' \
  -H 'accept: application/json'
```

Request URL

```
https://localhost:7210/api/carts/customer/1
```

Code Details

200

Response body

```
[
  {
    "idCart": 1,
    "idCustomer": 1
  }
]
```

GetProductsOfCart

GET /api/carts/{cartId}/products

Parameters

Name	Description
cartId * required integer($int32) (path)	1

Exec

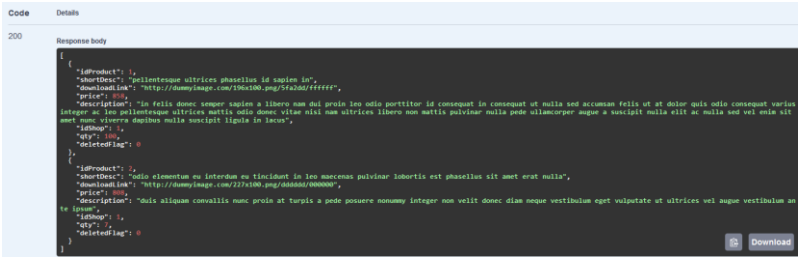
Responses

Curl

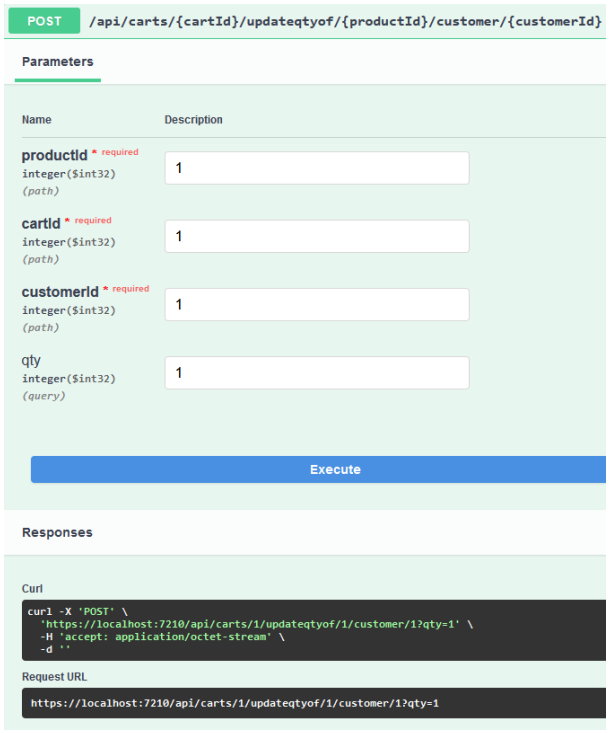
```
curl -X 'GET' \
  'https://localhost:7210/api/carts/1/products' \
  -H 'accept: application/json'
```

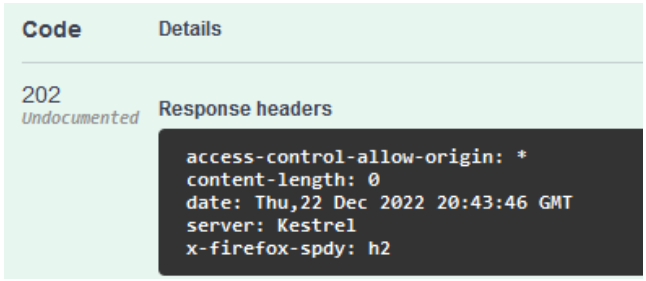
Request URL

```
https://localhost:7210/api/carts/1/products
```

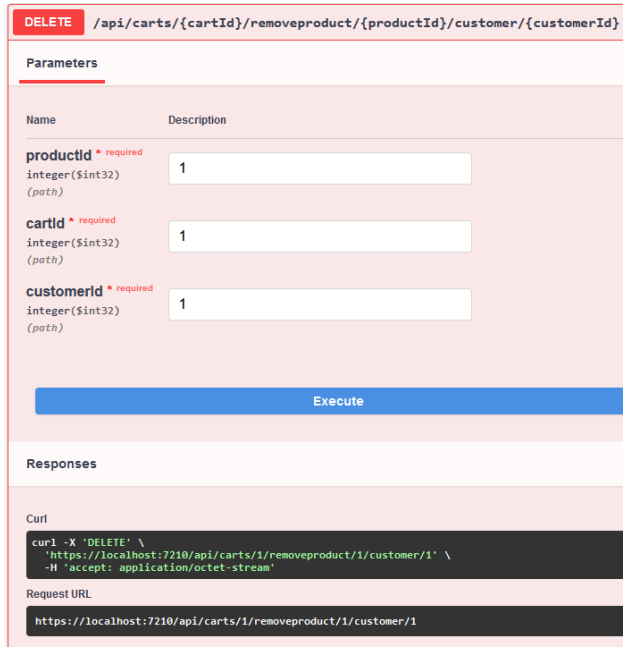


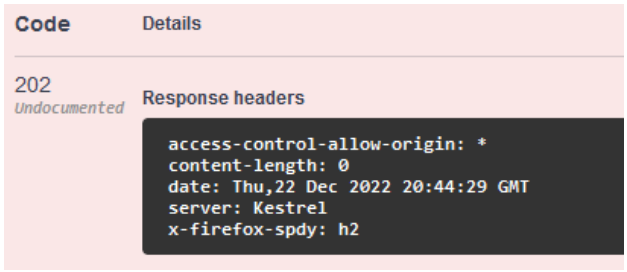
UpdateQtyOfProductInCart



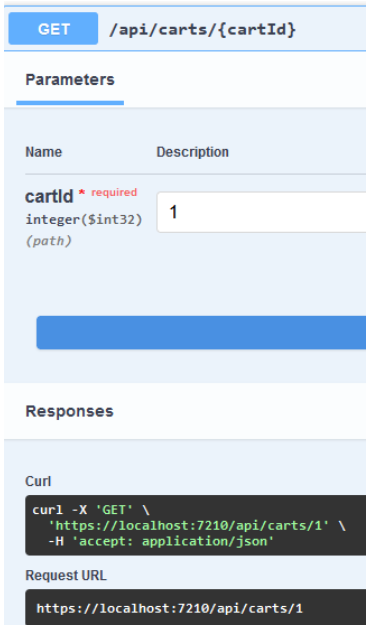


RemoveProductFromCart

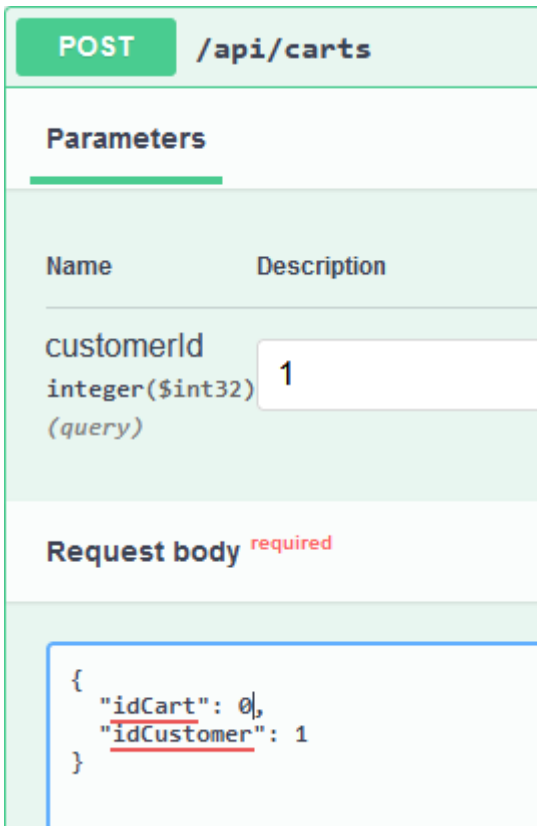




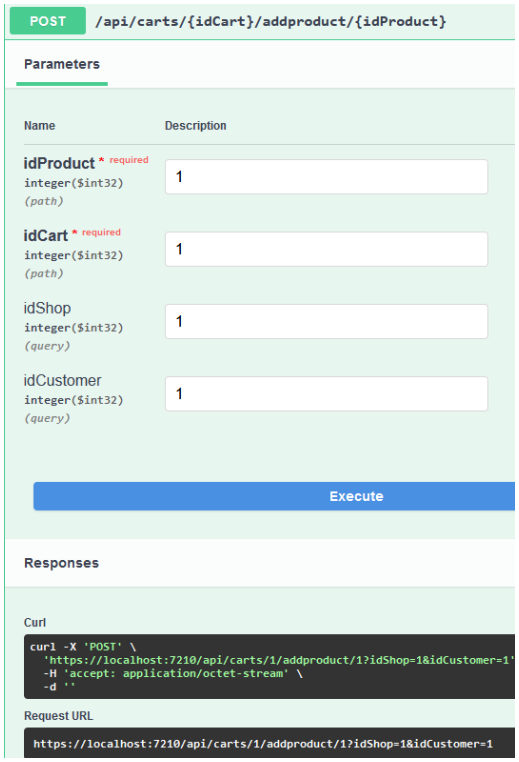
GetCartById

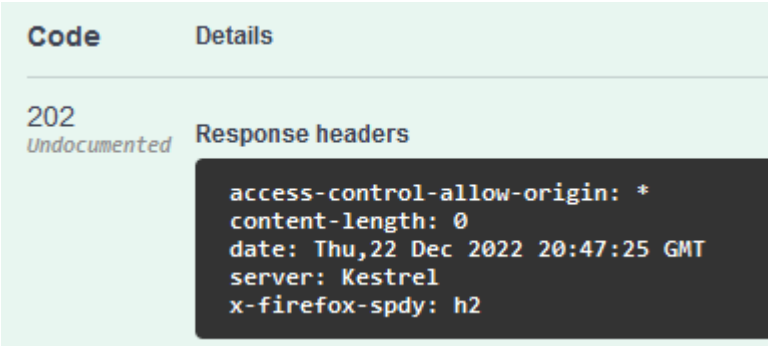


CreateCart

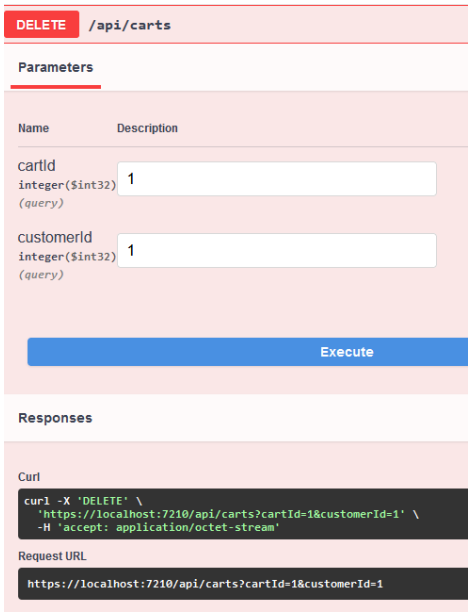


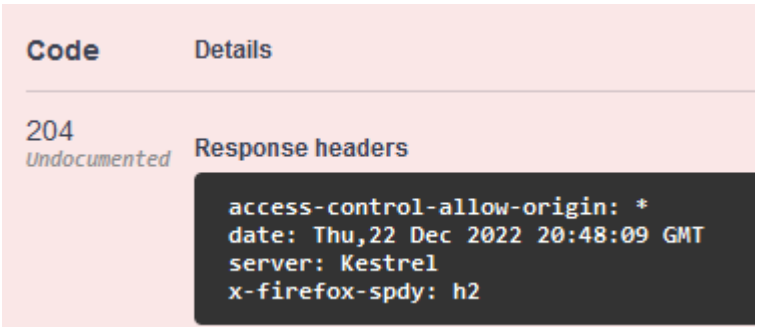
AddProductToCart



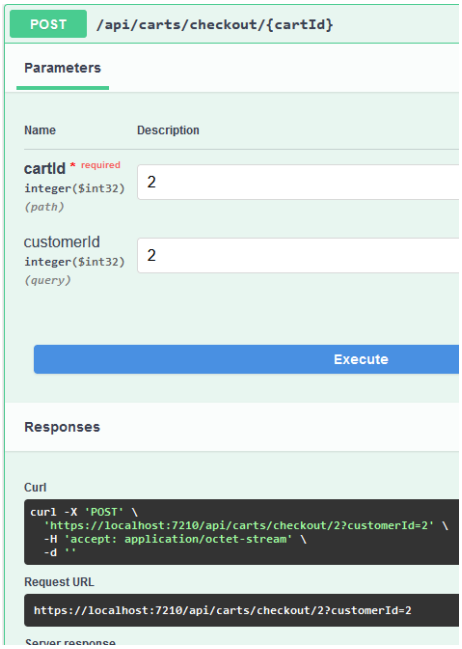


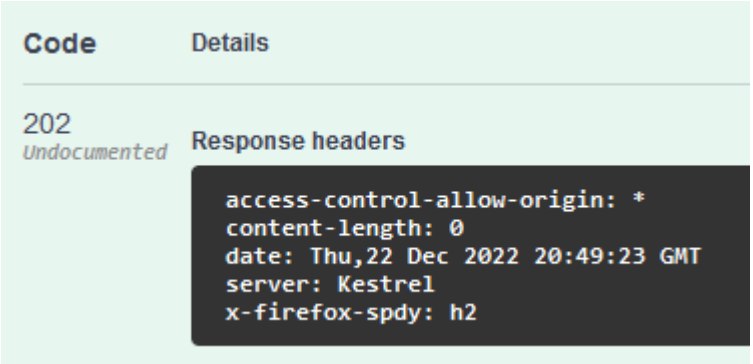
DeleteCart





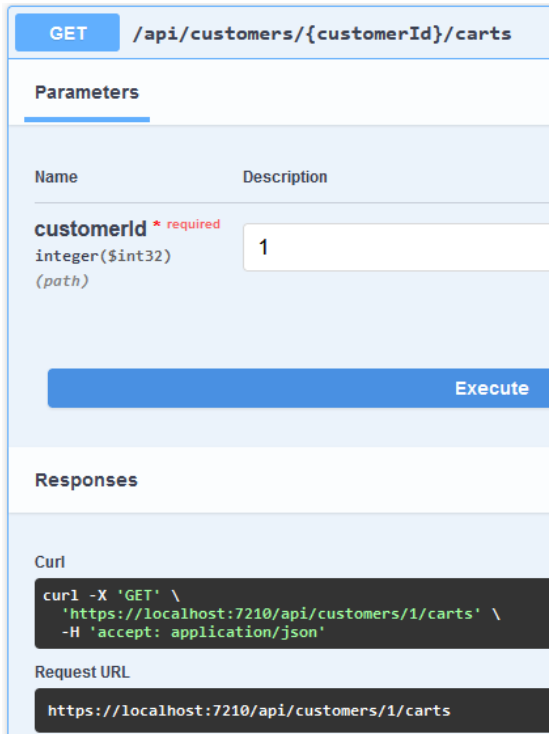
CreateOrderFromCart



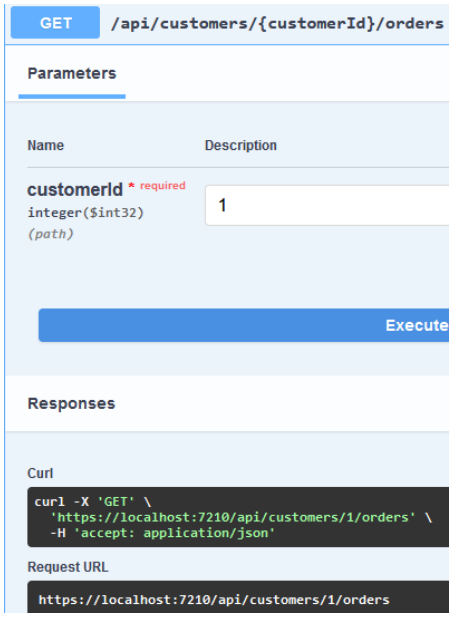


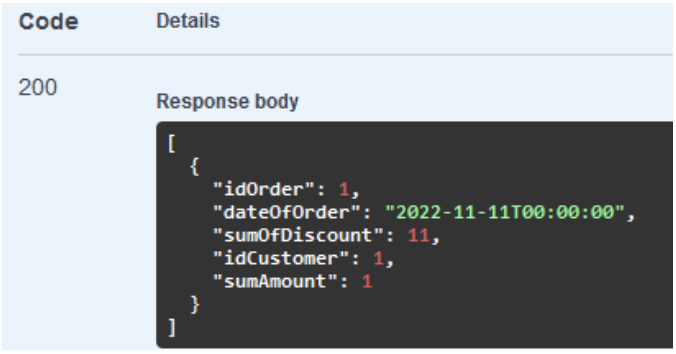
Customers:

GetCartsOfCustomer



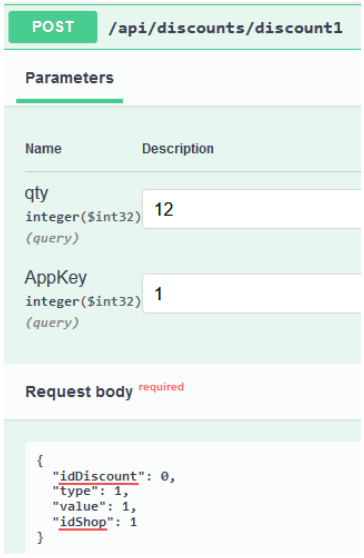
GetOrdersOfCustomer

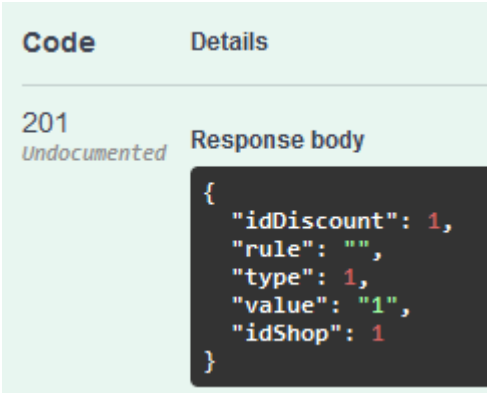




Discounts:

CreateDiscount1





CreateDiscount2

POST /api/discounts/discount2

Parameters

Name	Description
date1 string (query)	2022-01-01
date2 string (query)	2023-01-01
AppKey integer($int32) (query)	1

Request body required

```
{
  "idDiscount": 0,
  "type": 1,
  "value": 11,
  "idShop": 2
}
```

201
Undocumented

Response body

```
{
  "idDiscount": 0,
  "rule": "",
  "type": 1,
  "value": "11",
  "idShop": 2
}
```

GetDiscountsOfShop

GET /api/discounts/shop/{shopId}

Parameters

Name	Description
shopId * required integer($int32) (path)	1

Exec

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7210/api/discounts/shop/1' \
  -H 'accept: application/json'
```

Request URL

```
https://localhost:7210/api/discounts/shop/1
```

Code	Details
200	Response body

```
[
  {
    "idDiscount": 1,
    "rule": "Select IF(qty >= 12,1,0) FROM Cart_has_Product Where idProduct = @idProduct",
    "type": 1,
    "value": "1",
    "idShop": 1
  }
]
```

GetDiscountsById

GET /api/discounts/{discountId}

Parameters

Name	Description
discountId * required integer($int32) (path)	1

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7210/api/discounts/1' \
  -H 'accept: application/json'
```

Request URL

```
https://localhost:7210/api/discounts/1
```

Code	Details
200	Response body

```
{
  "idDiscount": 1,
  "rule": "Select IF(qty >= 12,1,0) FROM Cart_has_Product Where idProduct = @idProduct",
  "type": 1,
  "value": "1",
  "idShop": 1
}
```

Response headers

```
content-length: 133
content-type: application/json; charset=utf-8
date: Thu,22 Dec 2022 20:57:59 GMT
server: Kestrel
x-firefox-spdy: h2
```

DeleteDiscount

DELETE /api/discounts/{discountId}

Parameters

Name	Description
discountId * required integer($int32) (path)	1
AppKey integer($int32) (query)	1

Execute

Responses

Curl

```
curl -X 'DELETE' \
  'https://localhost:7210/api/discounts/1?AppKey=1' \
  -H 'accept: application/octet-stream'
```

Request URL

```
https://localhost:7210/api/discounts/1?AppKey=1
```

Code	Details
204 *Undocumented*	Response headers

```
access-control-allow-origin: *
date: Thu,22 Dec 2022 20:58:36 GMT
server: Kestrel
x-firefox-spdy: h2
```

Orders:

GetOrderById

GET /api/orders/{orderId}

Parameters

Name	Description
orderId * required integer($int32) *(path)*	1

Ex

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7210/api/orders/1' \
  -H 'accept: application/json'
```

Request URL

```
https://localhost:7210/api/orders/1
```

Code	Details
200	Response body


```
{
  "idOrder": 1,
  "dateOfOrder": "2022-11-11T00:00:00",
  "sumOfDiscount": 11,
  "idCustomer": 1,
  "sumAmount": 1
}
```


GetProductsInOrder

GET /api/orders/productsin/{orderId}

Parameters

Name	Description
orderId * required integer($int32) *(path)*	1

Execute

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7210/api/orders/productsin/1' \
  -H 'accept: application/json'
```

Request URL

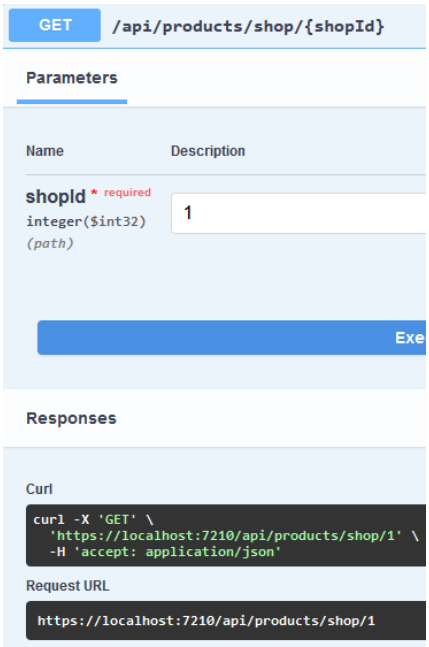
```
https://localhost:7210/api/orders/productsin/1
```

Code	Details
200	Response body

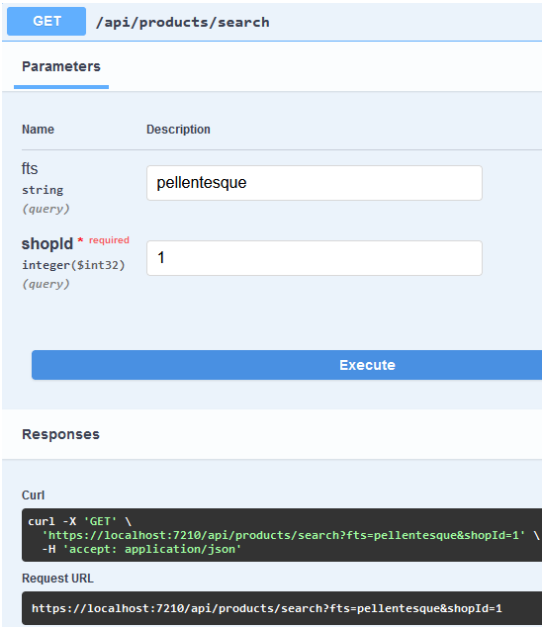

```
[
  {
    "idProduct": 1,
    "shortDesc": "pellentesque ultrices phasellus id sapien in",
    "downloadLink": "http://dummyimage.com/196x100.png/5fa2dd/ffffff",
    "price": 858,
    "description": "in felis donec semper sapien a libero nam dui proin leo
integer ac leo pellentesque ultrices mattis odio donec vitae nisi nam ultric
amet nunc viverra dapibus nulla suscipit ligula in lacus",
    "idShop": 1,
    "qty": 3,
    "deletedFlag": 0
  },
  {
    "idProduct": 4,
    "shortDesc": "justo maecenas rhoncus aliquam lacus",
    "downloadLink": "http://dummyimage.com/144x100.png/cc0000/ffffff",
    "price": 458,
    "description": "dui proin leo odio porttitor id consequat in consequat u
is odio donec vitae nisi nam ultrices libero non mattis pulvinar nulla pede
ligula in lacus curabitur at ipsum ac tellus semper interdum mauris ullamcor
u",
    "idShop": 2,
    "qty": 3,
    "deletedFlag": 0
  }
]
```

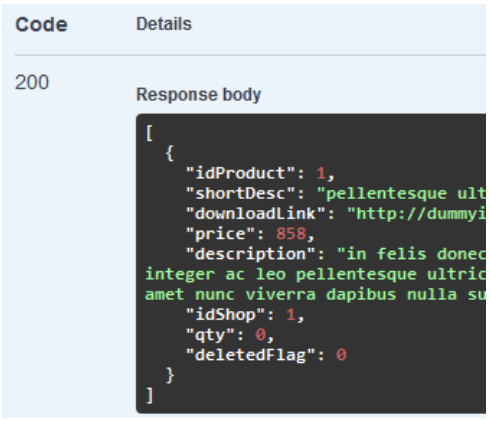
Products:

GetProducts

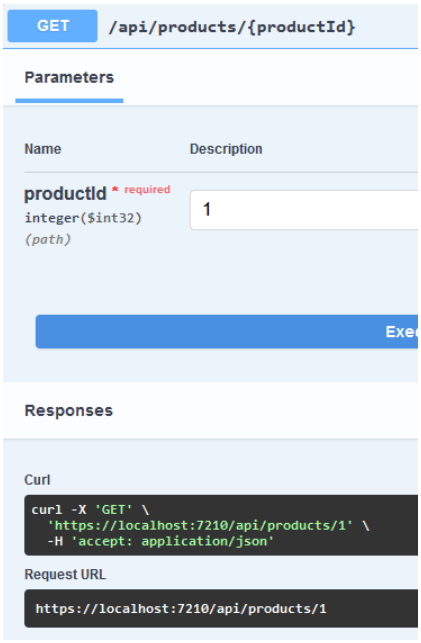


SearchProductsInShop

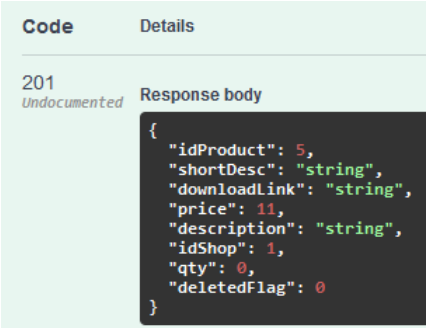




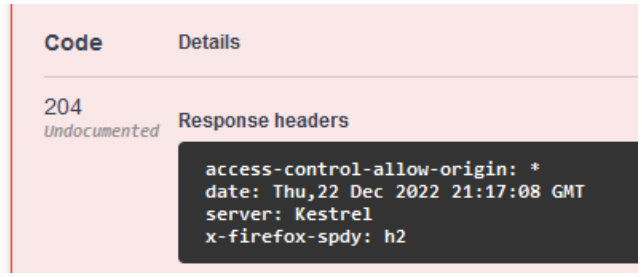
GetProductById



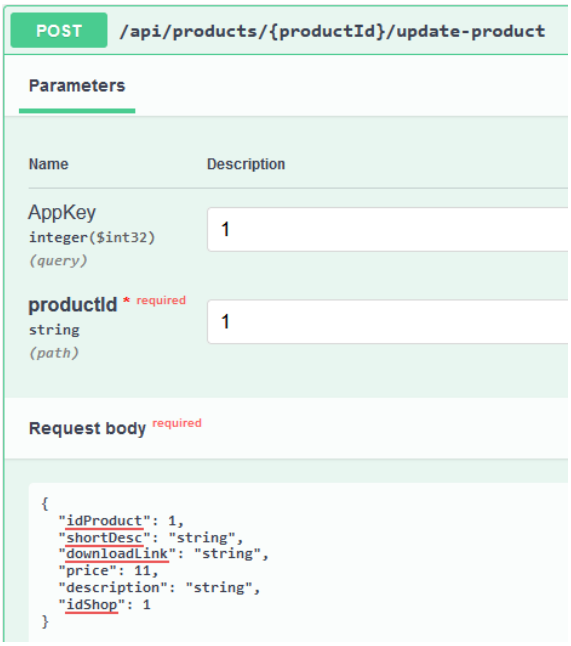
CreateProduct

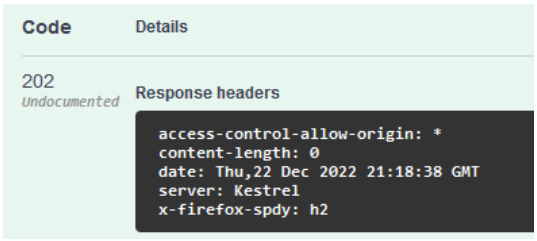


DeleteProduct



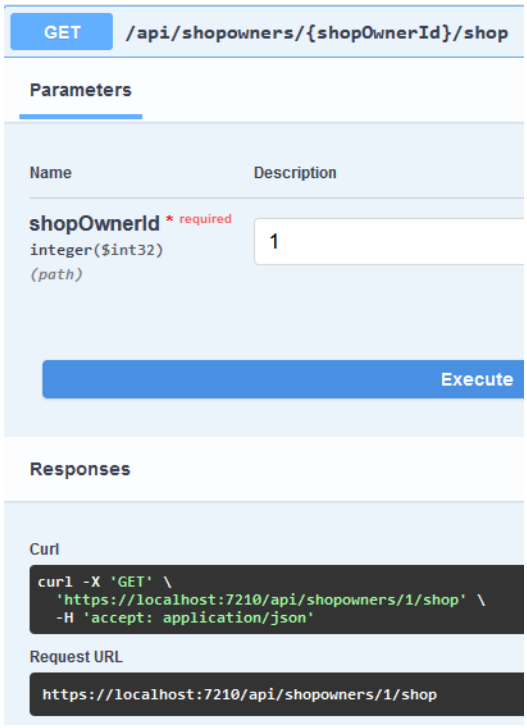
UpdateProduct

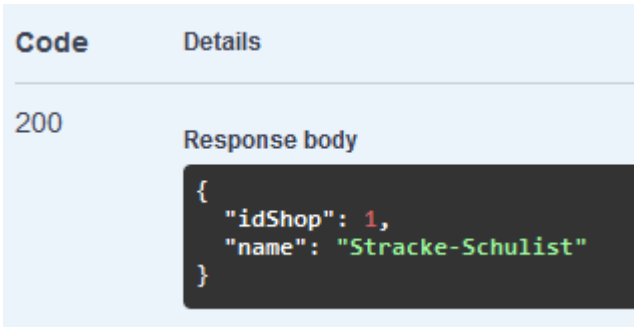




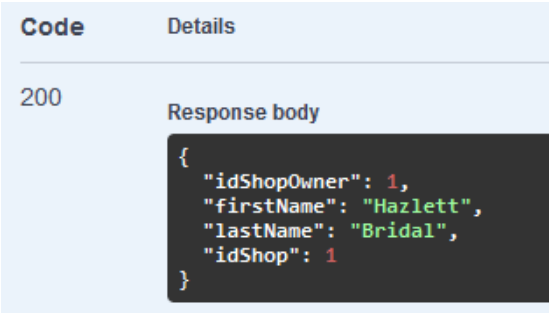
ShopOwners:

GetShopByShopOwnerId





GetShopOwnerById



CreateOwner

POST `/api/shopowners`

Parameters

No parameters

Request body required

```
{
  "idShopOwner": 0,
  "firstName": "string",
  "lastName": "string",
  "idShop": 2
}
```

Code Details

201
Undocumented

Response body

```
{
  "idShopOwner": 3,
  "firstName": "string",
  "lastName": "string",
  "idShop": 2
}
```

UpdateShopOwner

POST `/api/shopowners/update-shopowner`

Parameters

No parameters

Request body required

```
{
  "idShopOwner": 3,
  "firstName": "asdasd",
  "lastName": "string",
  "idShop": 2
}
```

Code Details

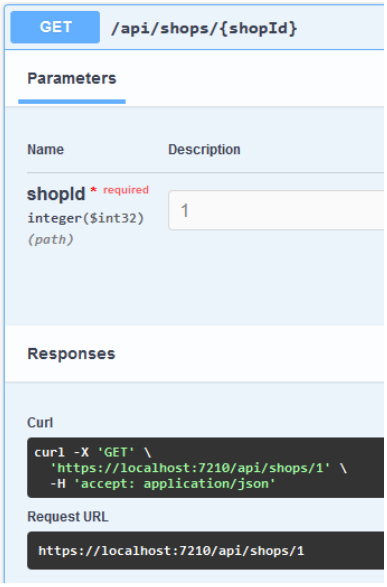
202
Undocumented

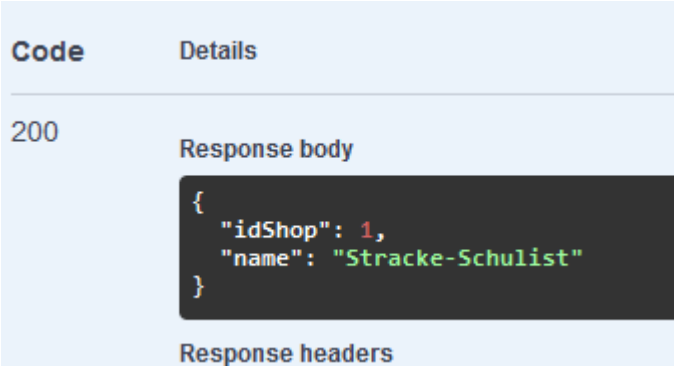
Response headers

```
access-control-allow-origin: *
content-length: 0
date: Thu,22 Dec 2022 21:23:26 GMT
server: Kestrel
x-firefox-spdy: h2
```

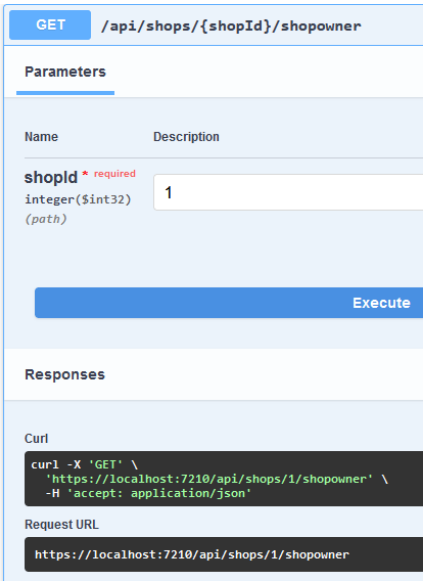
Shops:

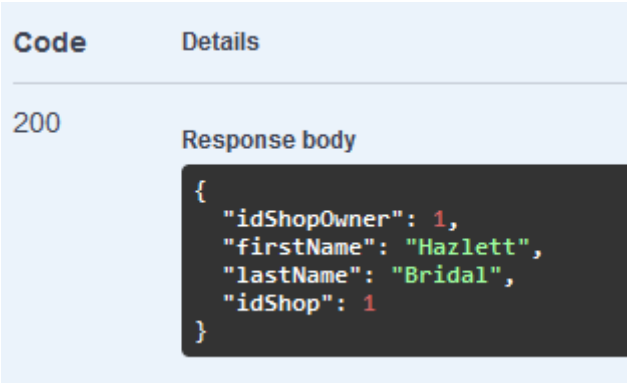
GetShopById



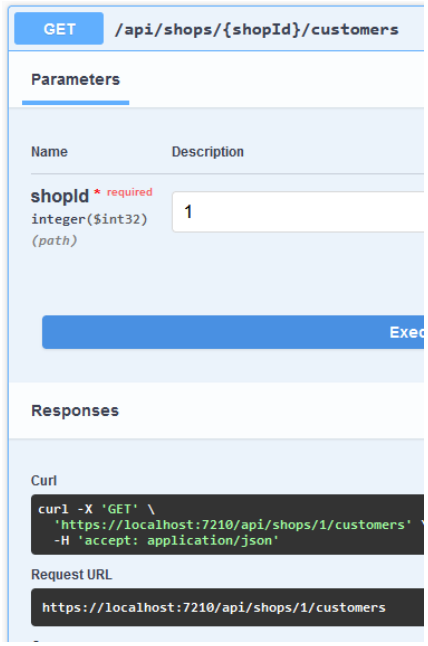


GetShopOwnerByShopId

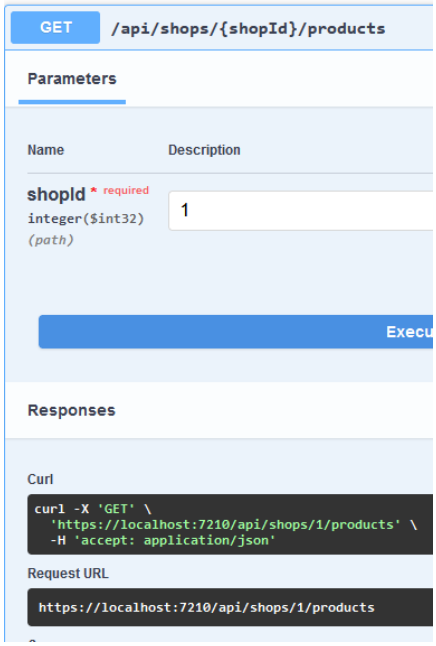


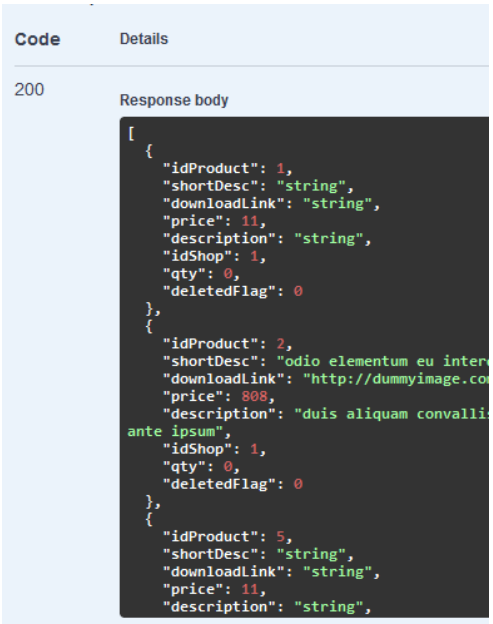


GetCustomersByShopId

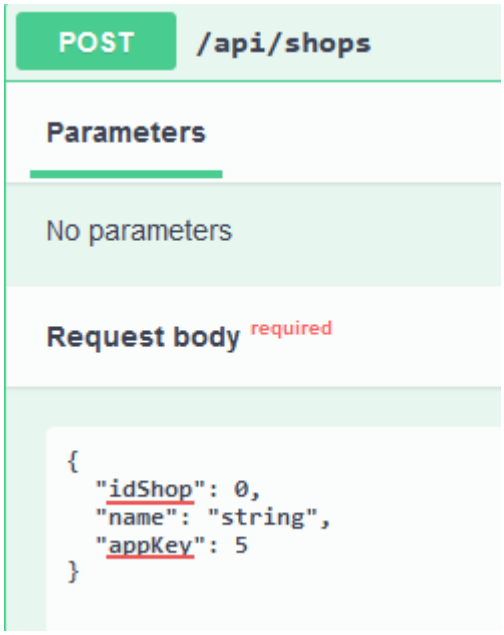


GetProductsByShopId

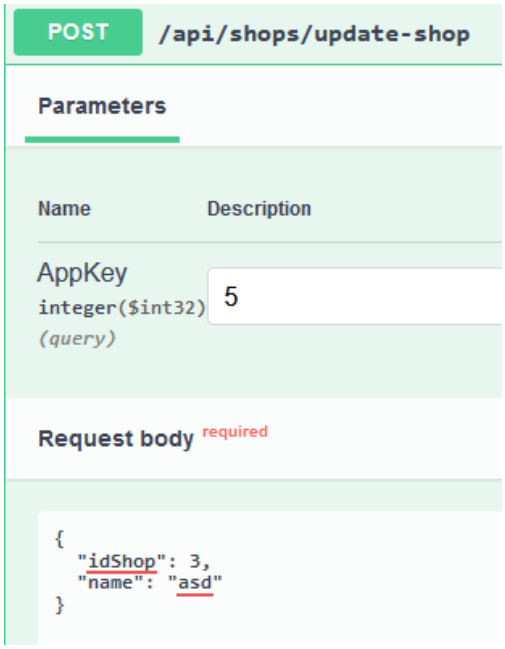


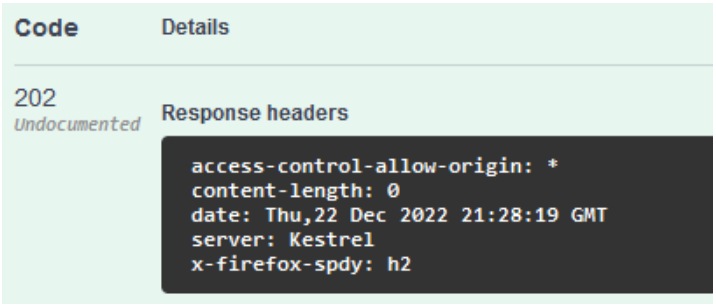


CreateProduct



UpdateProduct





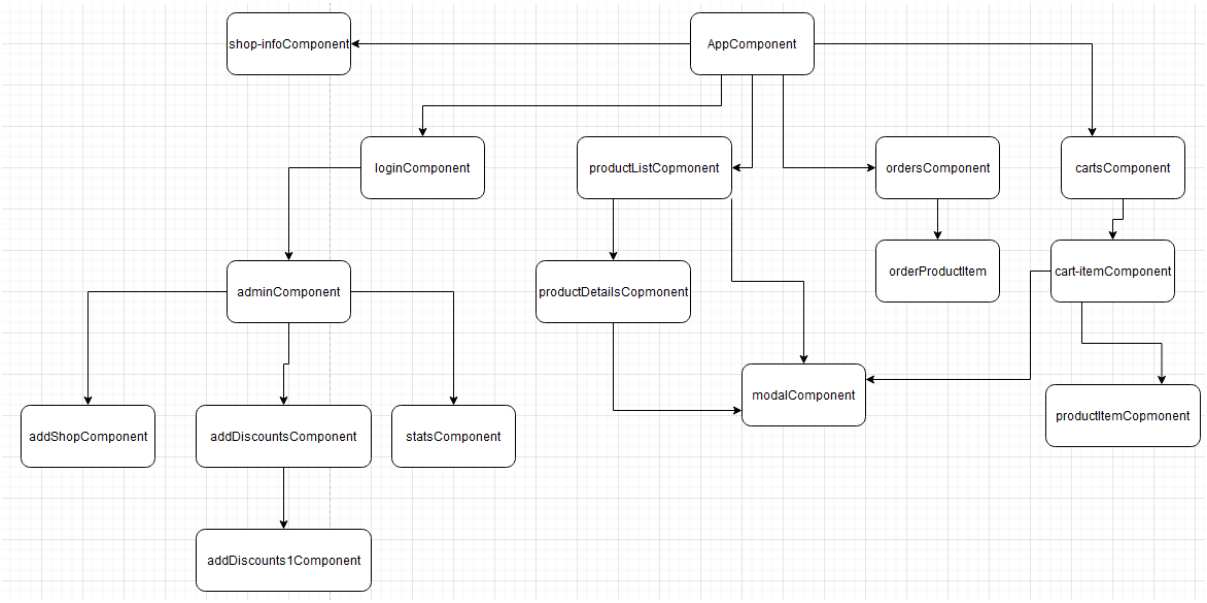
Ausbaustufe 3

Für die dritte ausbaustufe wurde Angular 15 benutzt. Ebenfalls wurden hauptsächlcih zusätzliche Steuerelemente von Material Bootstrap (https://mdbootstrap.com/). Für manche Icons habe ich font-awesome benutzt (https://fontawesome.com/) und für die charts habe ich PrimeNG benutzt (https://www.primefaces.org/primeng/). Für die Authentifizierung habe ich die Lösung von Manfred Steyer benutzt.

Setup:

1. MySQL Datenbank starten
2. CaaS API starten
3. Npm install
4. Ng serve Webserver wird auf localhost:4200 gestartet
5. *in environment.ts kann man die Endpunkte für die API ändern.

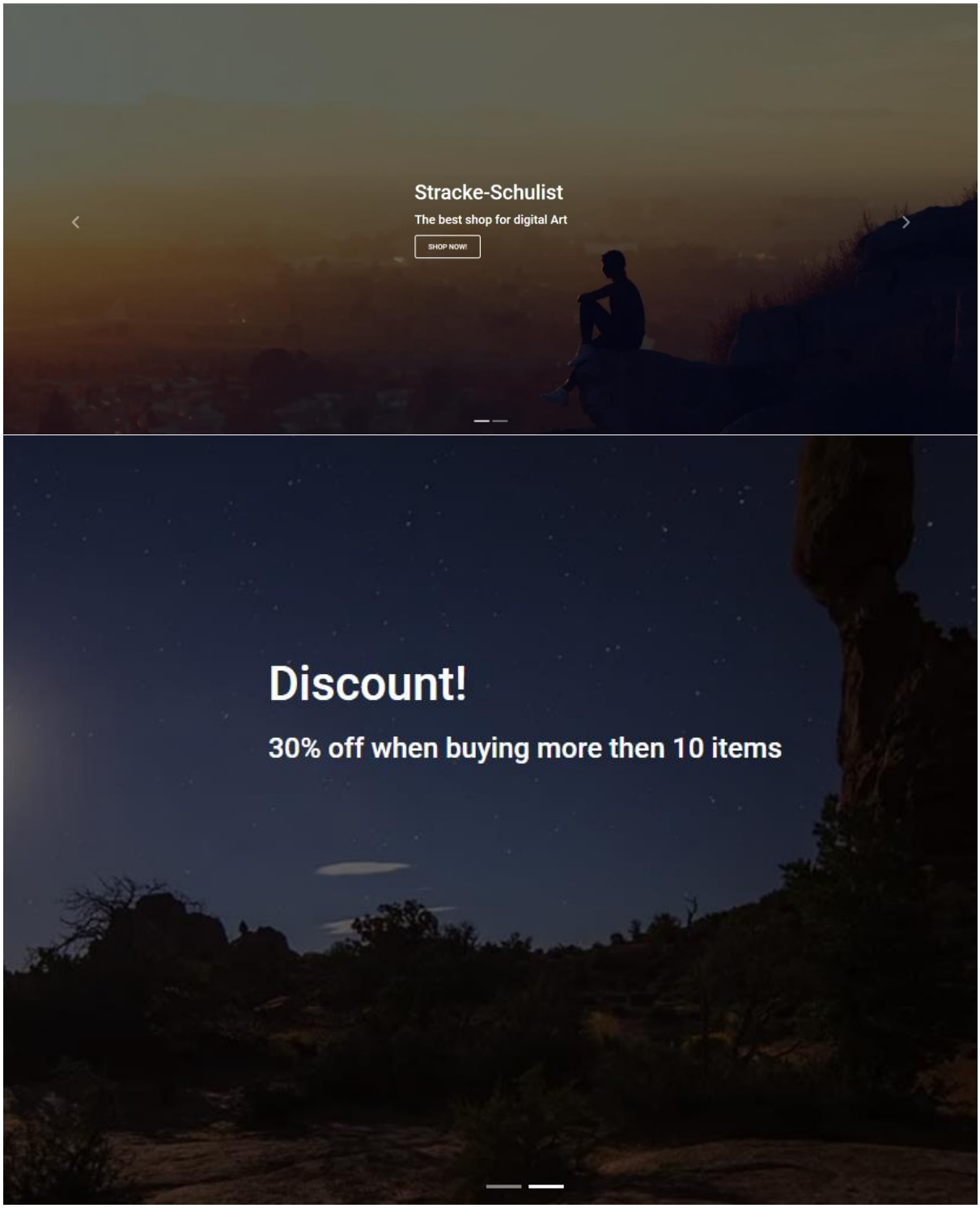
Kopmonenten:



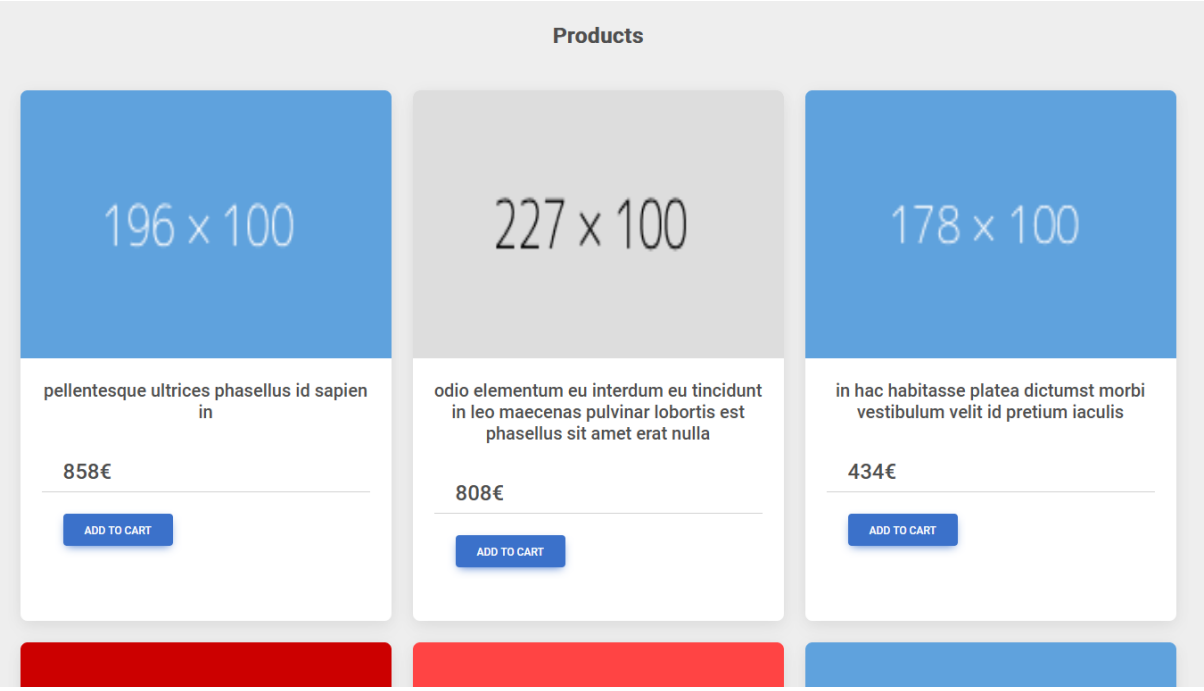
Übersicht der Komponenten

NavBar

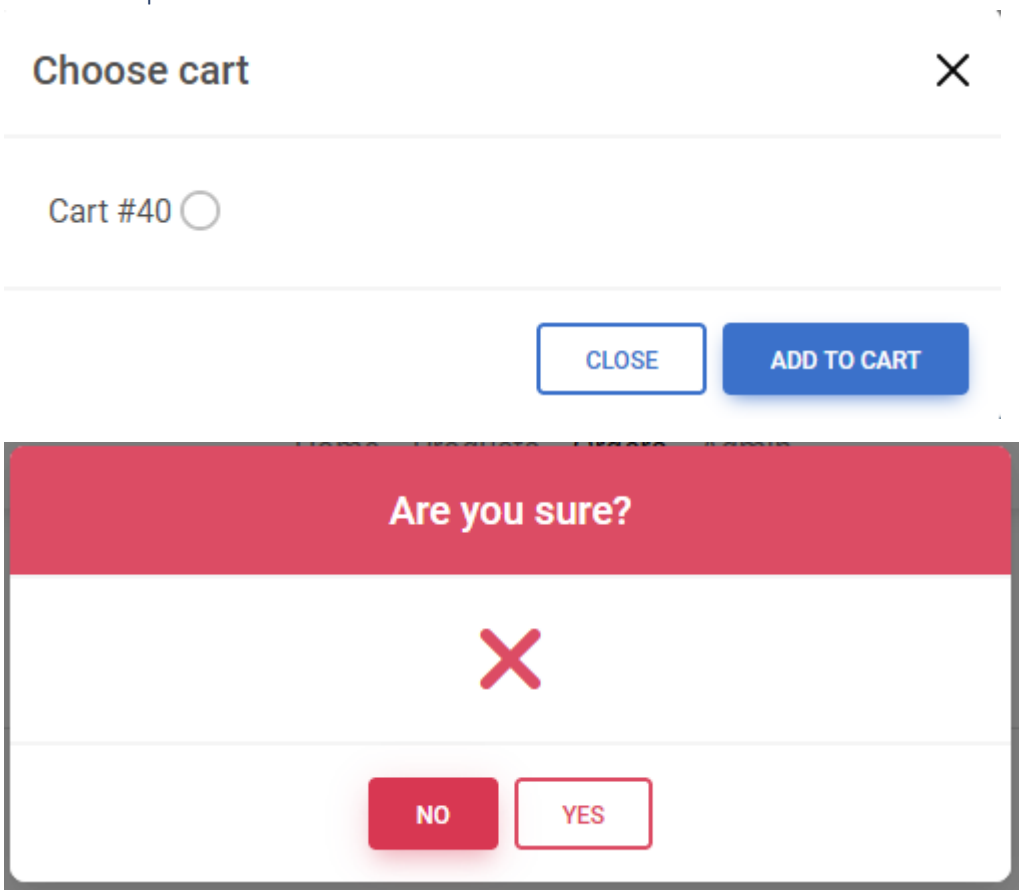
shopInfoComponent(home)

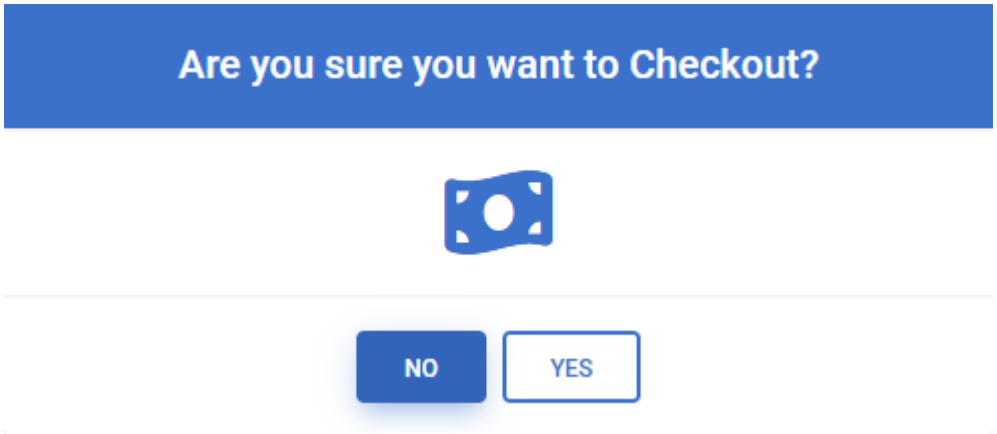


productListComponent(Products)

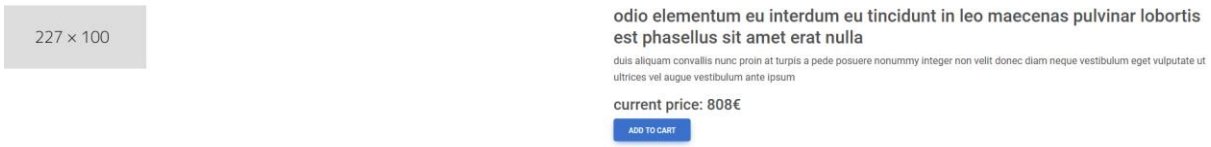


ModalComponent

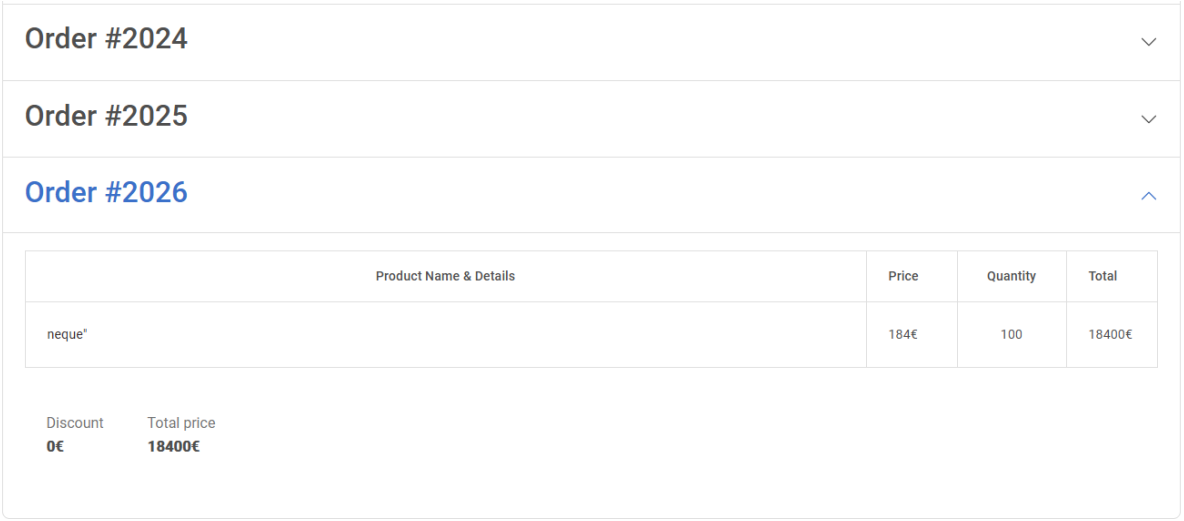




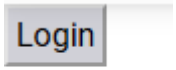
ProductDetailComponent



OrdersComponent

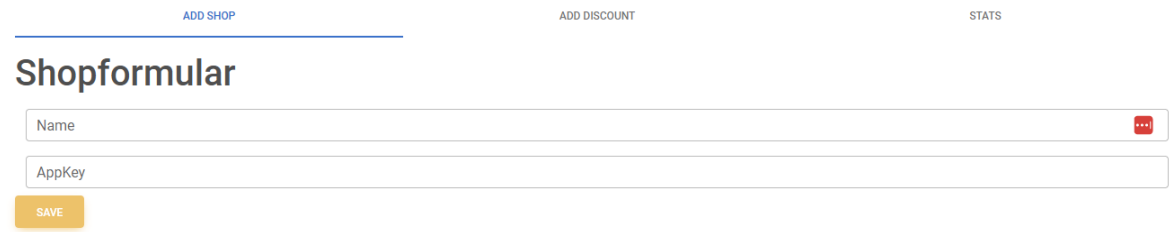


LoginComponent

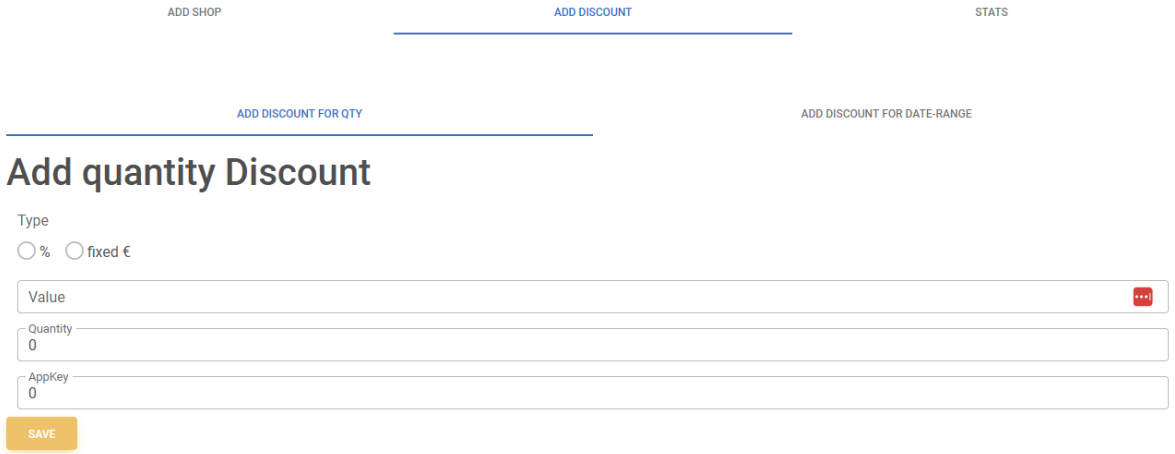


AdminComponent

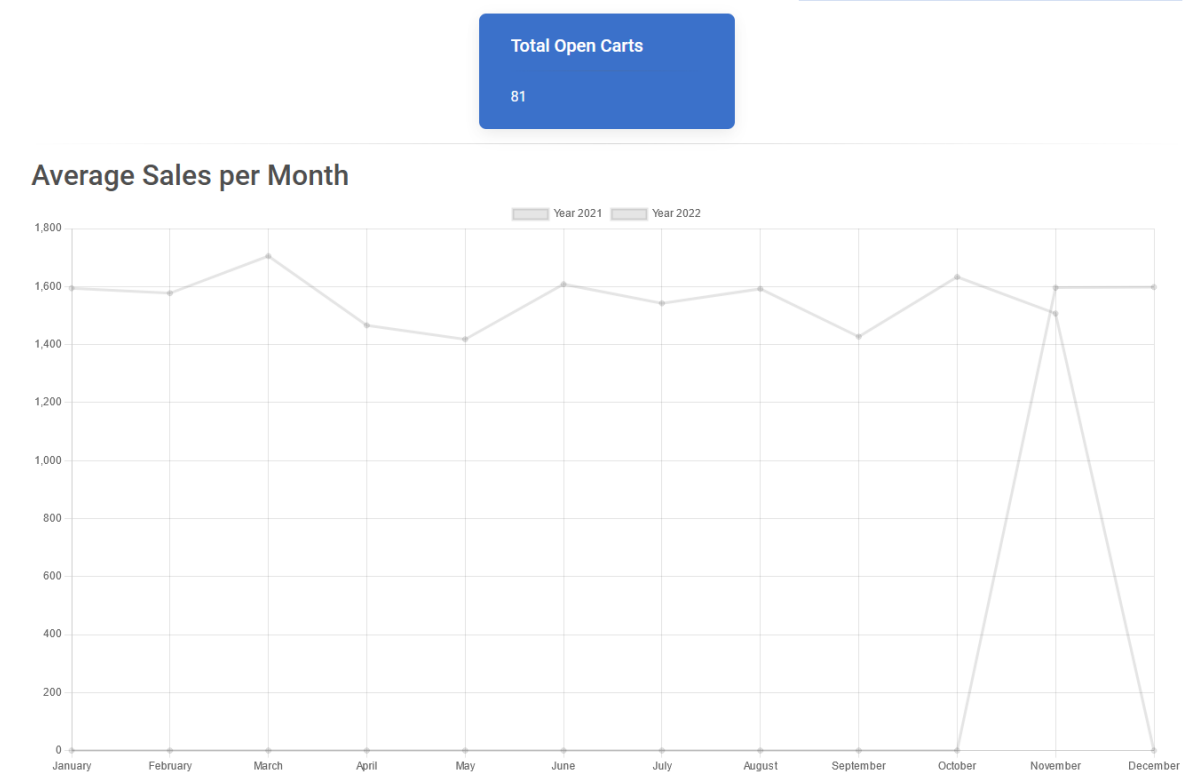
AddShopComponent

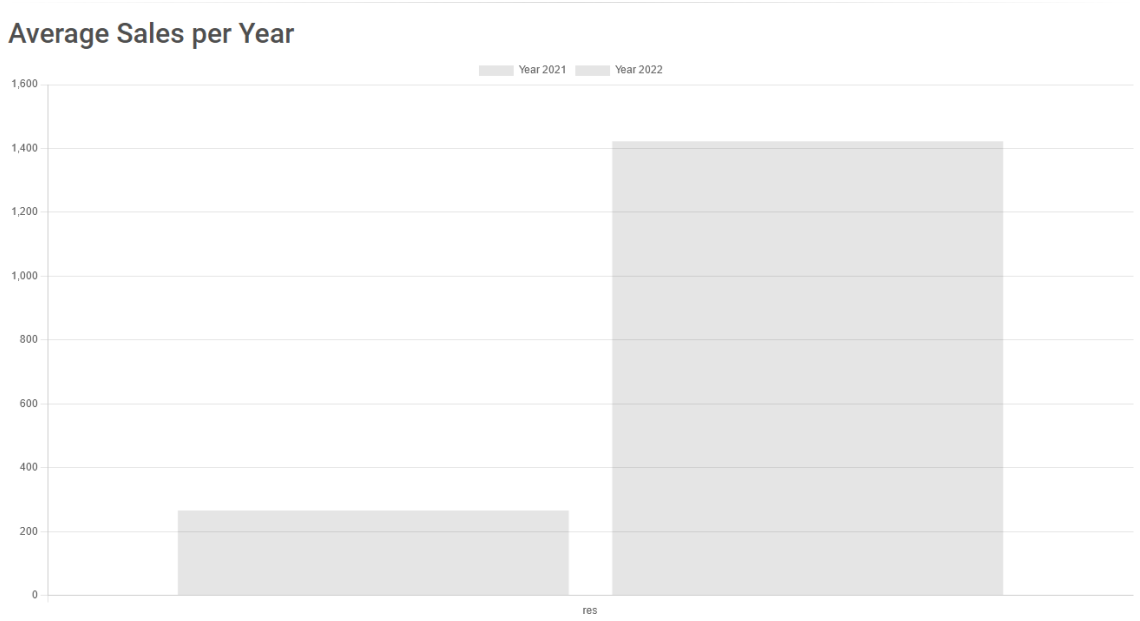


AddDiscountsComponent



StatsComponent





CartsComponent