Project Assignment 1

API specification

Your backend server will be running on port `8099`.

All endpoints need to allow CORS to `http://localhost:3010`, where the frontend server will be running.

The entire API needs to have the context path (i.e. be prefixed with) `/api/v1`. You can set the context path with the `server.servlet.contextPath` setting.

Database container specs

The database Docker container will have the name `database`, and the port mapping will be `5432:5432`.

The database name will be `saltmerch`.

The db username will be `jensenyh`.

The db password will be `password`.

That means that the url of the database will be:

`jdbc:postgresql://localhost:5432/saltmerch`

(bear this in mind when setting up the datasource in `application.yml`/`application.properties`.)

ENDPOINTS

1. Product endpoints
    - Get all products
    - Get all products in category
    - Get one specific product
    - Create new product

- Update product's metadata
- Create new variant for a specific product
- Restock specific size of a variant
- Delete product
- Delete variant of a product

2. Cart endpoints
    - Get cart contents
    - Add item to cart/Remove item from cart
    - Clear cart contents (and potentially restock)

Product endpoints

Get all products

`GET /api/v1/products`

Gets the list of all products, regardless of category. In other words, gets all hats, jackets, tshirts, bags, without details about specific colors, sizes, and stock..

Returns JSON with the following format:

```
[
    {
        "id":"integer",
        "category":"string",
        "title":"string",
        "description":"string",
        "previewImage":"string",
        "variants":"array (empty)"
    }
]
```

Example of a return object:

```
[
    {
        "id": 1,
```

```
        "category": "hats",
        "title": "Salty Hat",
        "description": "This is a salty hat.",
        "previewImage": "images/hats/01.jpg",
        "variants": []
    },
    {
        "id": 10,
        "category": "tshirts",
        "title": "Salty T-shirt",
        "description": "This is a salty t-shirt.",
        "previewImage": "images/tshirts/01.jpg",
        "variants": []
    }
]
```

Get all products in category

```
GET /api/v1/products/hats
GET /api/v1/products/jackets
GET /api/v1/products/tshirts
GET /api/v1/products/bags
```

Gets a list of products of one specific category (hats, jackets, tshirts, or bags).

Returns JSON with the following format:


```
[
    {
        "id":"integer",
        "category":"string",
        "title":"string",
        "description":"string",
        "previewImage":"string",
        "variants":"array (empty)"
    }
]
```


Example of a return object for GET /hats:


```
[
    {
        "id": 1,
        "category": "hats",
        "title": "Salty Hat",
        "description": "This is a salty hat.",
```

```
        "previewImage": "images/hats/01.jpg",
        "variants": []
    },
    {
        "id": 2,
        "category": "hats",
        "title": "Peppery hat",
        "description": "This is a peppery hat.",
        "previewImage": "images/hats/02.jpg",
        "variants": []
    }
]
```

Get one specific product

`GET /api/v1/products/{id}`

Gets all details about the specific product that was requested, including what's in stock.

Returns JSON with the following format:

```
[
    {
        "id":"integer",
        "category":"string",
        "title":"string",
        "description":"string",
        "previewImage":"string",
        "variants":[
            {
                "colorName":"string",
                "images":[ "string" ],
                "sizes":[
                    {
                        "size":"string",
                        "stock":"integer"
                    }
                ]
            }
        ]
    }
]
```

`variants` is an *array* of `Variant` *objects*; `Variant` contains a `colorName` *string*, an `images` *array*, and a `sizes` *array*.

`images` is an *array* of *strings*, each one being a url of an image.

`sizes` is an *array* of `VariantSize` objects; `VariantSize` contains a `size` *string* and a `stock` *integer*.

Example of a return object for `GET /api/v1/products/5`:


```
{
    "id": 5,
    "category": "jackets",
    "title": "Salty jacket",
    "description": "This is a very salty jacket.",
    "previewImage": "images/jackets/black_01.jpg",
    "variants": [
        {
            "colorName": "Black",
            "images": [
                "images/hat/black_01.jpg",
                "images/hat/black_02.jpg"
            ],
            "sizes": [
                {
                    "size": "M",
                    "stock": 12
                },
                {
                    "size": "L",
                    "stock": 8
                }
            ]
        },
        {
            "colorName": "Gray",
            "images": [
                "images/jackets/gray_01.jpg"
            ],
            "sizes": [
                {
                    "size": "S",
                    "stock": 2
                }
            ]
        }
    ]
}
```

Create new product

(OPTIONAL)

```
POST /api/v1/products/hats
POST /api/v1/products/jackets
POST /api/v1/products/tshirts
POST /api/v1/products/bags
```

Creates a new product of the specified category and stores it in the database.

Returns `201` and the created product (including its assigned id) as the response body.

Request body:

```
{
    "title":"string",
    "description":"string",
    "previewImage":"string",
    "variants":[
        {
            "colorName":"string",
            "images":[ "string" ],
            "sizes":[
                {
                    "size":"string",
                    "stock":"integer"
                }
            ]
        }
    ]
}
```

Update product's metadata

(OPTIONAL)

```
PUT /api/v1/products/{id}
```

Updates a specific (identified by `id`) product's basic info (title, description, previewImage). The body can contain `title`, `description`, and `previewImage` (all optional).

Returns `200`.

Request body format:

```
{
    "title":"string",
    "description":"string",
    "previewImage":"string"
}
```

Example body:

```
{
    "title": "A saltier hat",
    "description": "This became an even saltier hat than before!"
}
```

Create new variant for a specific product

(OPTIONAL)

`POST /api/v1/products/{id}/variants`

Creates a new variant for the product with id `{id}`.

A *variant* describes a color and available sizes for that color.

Returns `201`.

Request body format:

```
{
    "colorName":"string",
    "images":[ "string" ],
    "sizes":[
        {
            "size":"string",
            "stock":"integer"
        }
    ]
}
```

Example request body:

```
{
    "colorName": "Gray",
    "images": [
        "images/jackets/01.jpg",
        "images/jackets/02.jpg"
    ],
    "sizes": [
```

```
        {
            "size": "M",
            "stock": 10
        },
        {
            "size": "XL",
            "stock": 1
        }
    ]
}
```

Restock specific size of a variant

(OPTIONAL)

`PUT /api/v1/products/{id}/variants/stock?size={size}&color={green}&quantity={quantity}`

Restocks a specific item of a specific color and size, i.e. increases its quantity by a given number.

Path variable `id` is used to identify the product, and query parameters `size`, `color`, and `quantity` are used to specify the exact item variant.

No request body.

Returns `200`.

Delete product

(OPTIONAL)

`DELETE /api/v1/products/{id}`

Deletes a specific product from the database.

Returns `200`.

Delete variant of a product

(OPTIONAL)

`DELETE /api/v1/products/{productId}/variants/{variantId}`

Deletes a specific variant for the product with id `{id}`.

Returns `200`.

Add item to cart/Remove item from cart

`PATCH /api/v1/carts/{id}?action=add`
`PATCH /api/v1/carts/{id}?action=remove`

Depending on the `action` query *parameter*'s value, this endpoint should either add or remove one piece of the passed item to or from the cart.

The stock quantity should also be updated accordingly in the database (+1 or -1).

Returns status code 200 on success.

If `action=add`, the request body contains 2 extra properties, which don't need to be present when `action=remove`.

Request body for `action=add`:

```
{
    "productId":"integer",
    "color":"string",
    "size":"string",
    "title":"string",
    "previewImage":"string"
}
```

Request body for `action=remove`:

```
{
    "productId":"integer",
    "color":"string",
    "size":"string"
}
```

Example of a request body `action=add`:

```
{
    "productId": 1,
    "color": "Black",
    "size": "onesize",
    "title": "Salty hat",
    "previewImage": "images/salt-store-items/hats/black_01.jpg"
}
```

Example of a request body `action=remove`:

```
{
    "productId": 1,
    "color": "Black",
    "size": "onesize"
}
```

Clear cart contents (and potentially restock)

```
DELETE /api/v1/carts/{id}
DELETE /api/v1/carts/{id}?buyout=true
```

If the `buyout` *query parameter* equals `true`, only clears the user's cart.

Otherwise, clears the cart, but also restocks the items that were in the cart. In other words, reverts the stock amounts to their previous values, by adding back the quantities from the cart.