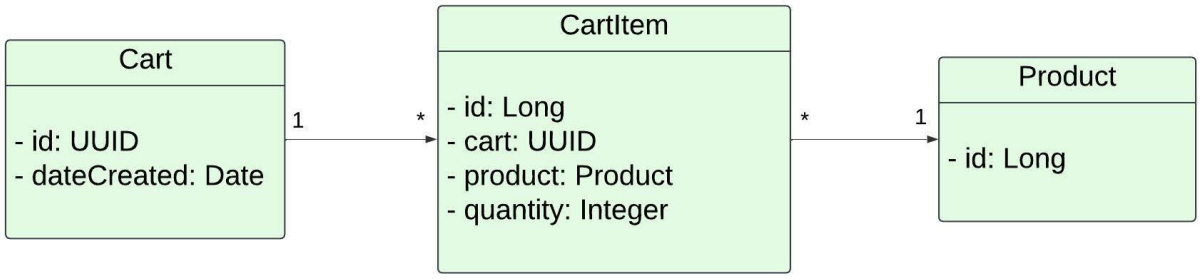
Capstone Project: Building the Shopping Cart API

In this capstone project, you'll apply everything you've learned to **design and implement a Shopping Cart API** from scratch. This will help reinforce your understanding of RESTful API design, database migrations, entity relationships, validation, business logic, and CRUD operations.

Before you watch me build this step by step, I encourage you to take an hour or two to work through the steps on your own. This will give you hands-on experience and help you identify gaps in your understanding.

Step 1: Creating Database Tables

- Below, you'll find a UML diagram representing the logical model for this project.
- Create the corresponding database tables by writing SQL scripts.
- Apply these scripts using Flyway migrations, then run your application to ensure the tables are created.



Key Design Decisions:

- In this design, we're not associating shopping carts with users. This allows users to add items to their shopping cart without having to log in first.
- Later, if a user logs in, we can associate their cart with their account.
- We use `UUID` (Universally Unique Identifier) instead of `Long` for cart.id because UUIDs ensure uniqueness and provide better security. Sequential IDs can expose how many carts have been created, and potentially allow a hacker to modify

someone else's cart. Here's an example of a UUID: `31ac008a-0591-11f0-bb73-2e7ff13c56fc`

Hints for Creating the Database Schema:

- MySQL doesn't have a `UUID` type. We can store them as `VARCHAR(36)` or `BINARY(16)`. Storing them as binary reduces storage space and improves indexing performance.
- To initialize the `id` column use the expression `uuid_to_bin(uuid())`. This converts a UUID from a string format to a compact binary format.

Step 2: Creating Entity Classes

- Now that we have our database tables, create entity classes that map to them.
- Once your entities are created, move on to the next step!

Step 3: Creating a Cart

Example Request:

```
POST /carts
```

Example Response:

```
201 CREATED

{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "items": [],
  "totalPrice": 0
}
```

Step 4: Adding a Product to the Cart

Example Request:

```
POST /carts/{cartId}/items

{

  "productId": 1

}
```

Example Response:

```
201 CREATED

{

  "product": {

     "id": 1,

     "name": “Product 1”,

     "price": 10

  }

  "quantity": 5,

  "totalPrice": 50

}
```

- If cart doesn’t exist, return 404 Not Found.
- If product doesn’t exist, return 400 Bad Request.
- If the product is already in the cart, update the quantity instead of adding a duplicate entry.
- If successful, return 200 OK.

Step 5: Getting a Cart

Example Request:

```
GET /carts/{cartId}
```

Example Response:

```
{
  “id": "550e8400-e29b-41d4-a716-446655440000"
  "items": [
     {
         "product": {
            "id": 1,
            "name": “Product 1”,
            "price": 10
         }
        "quantity": 5,
        "totalPrice": 50,
     }
  ],
  "totalPrice": 50
}
```

- If cart doesn’t exist, return 404 Not Found.
- If successful, return 200 OK.

Step 6: Updating a Cart Item

Example Request:

```
PUT /carts/{cartId}/items/{productId}

{
  "quantity": 5
}
```

Example Response:

```
{
  "product": {
     "id": 1,
     "name": “Product 1”,
     "price": 10
  }
  "quantity": 5,
  "totalPrice": 50
}
```

- If the cart or cart item doesn’t exist, return 404 Not Found.
- Validate that the quantity is between 1 and 100. If validation fails, return 400 Bad Request.
- If successful, return 200 OK.

Step 7: Removing a Product from the Cart

Example Request:

```
DELETE /carts/{cartId}/items/{productId}
```

Example Response:

```
HTTP 204
```

- If the cart or cart item doesn't exist, return 404 Not Found.
- If successful, return 204 No Content.

Step 8: Clearing a Cart

Example Request:

```
DELETE /carts/{cartId}/items
```

Example Response:

```
HTTP 204
```

- If the cart doesn't exist, return 404 Not Found.
- If successful, return 204 No Content.