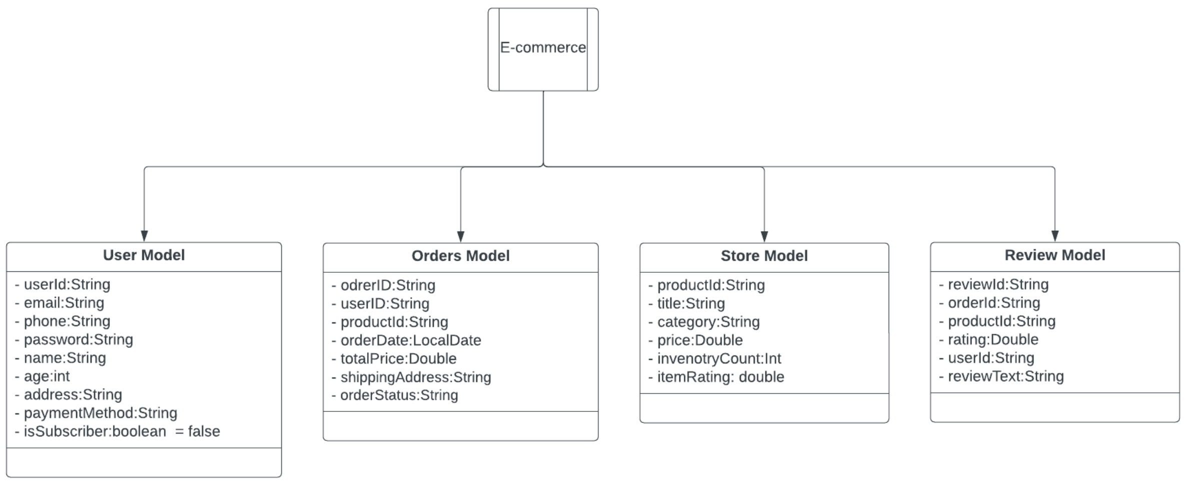
1.1 Scope:
The task is to choose a specific domain or industry and create a validation checklist for the data or processes relevant to that domain. should include at least 3 models from the domain along with their validation rules or criteria.

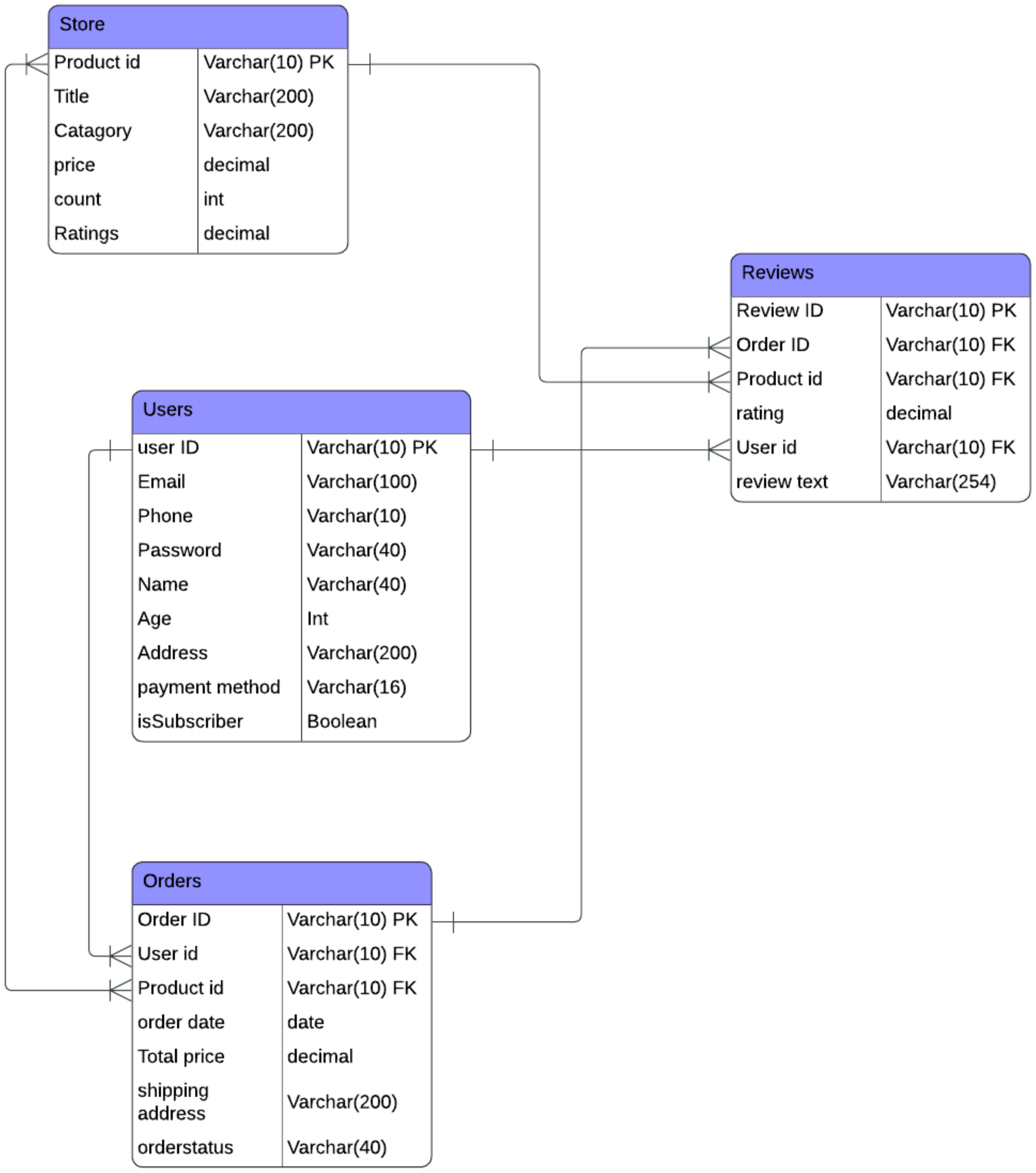
1.2 Chosen Domain:
The online E-commerce domain was chosen and amazon was the example looked at during the writing.

1.3 Purpose:
This domain allows the users to buy items in the store and write reviews for them the process includes order creation, tracking status, reviews and refunds if needed, or allows ths users to sell items on the store this involves creating the listing, setting the price and showing orders.

1.4 Plan:
Splitting and simplifying the models into 4 main models:
1. Users model.
2. Orders model.
3. Store model.
4. Reviews model.



1.5 Database schema:



Store	
Product id	Varchar(10) PK
Title	Varchar(200)
Catagory	Varchar(200)
price	decimal
count	int
Ratings	decimal

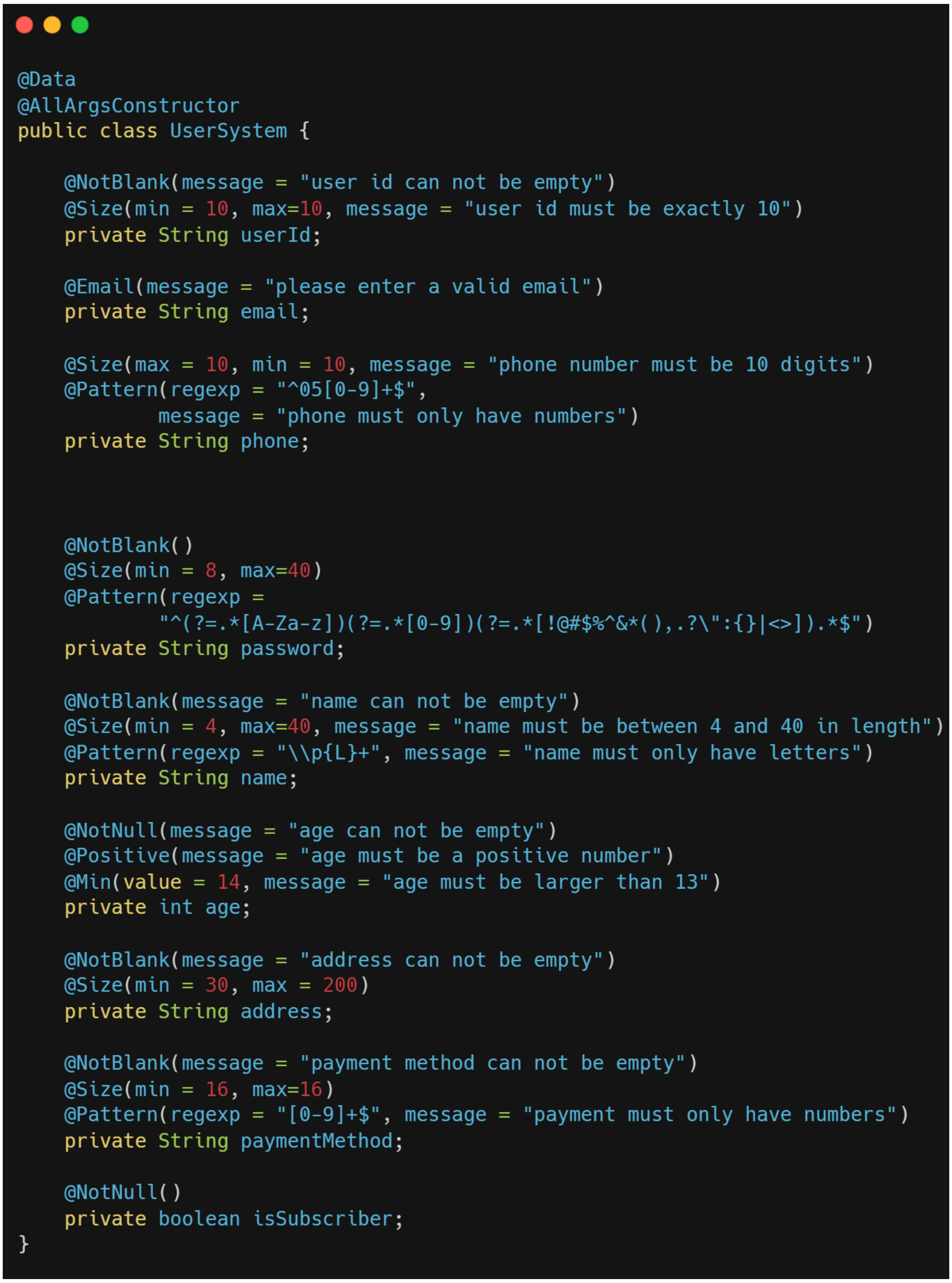
Reviews	
Review ID	Varchar(10) PK
Order ID	Varchar(10) FK
Product id	Varchar(10) FK
rating	decimal
User id	Varchar(10) FK
review text	Varchar(254)

Users	
user ID	Varchar(10) PK
Email	Varchar(100)
Phone	Varchar(10)
Password	Varchar(40)
Name	Varchar(40)
Age	Int
Address	Varchar(200)
payment method	Varchar(16)
isSubscriber	Boolean

Orders	
Order ID	Varchar(10) PK
User id	Varchar(10) FK
Product id	Varchar(10) FK
order date	date
Total price	decimal
shipping address	Varchar(200)
orderstatus	Varchar(40)

2.1 User model:

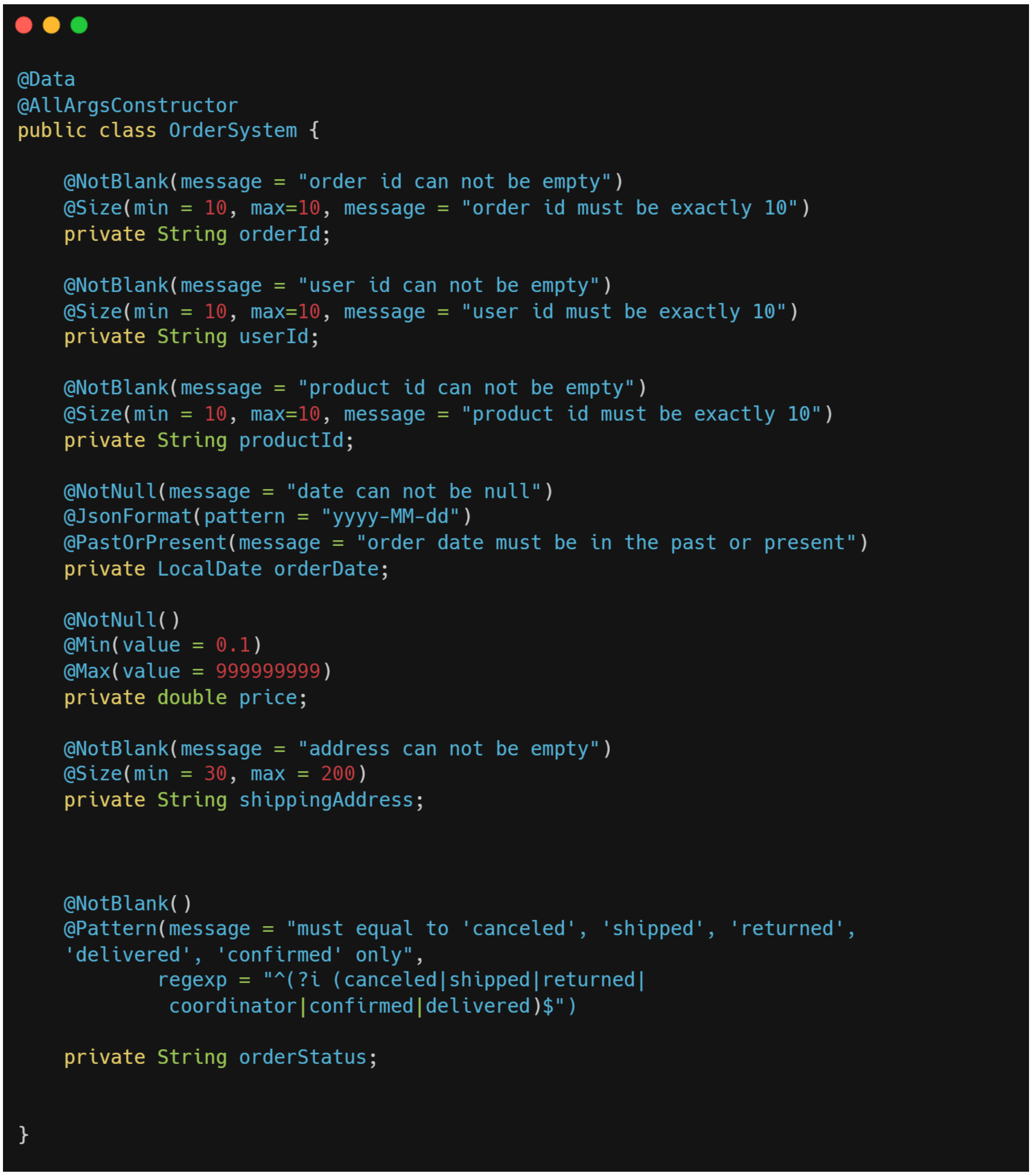
User model i meant to handle users registry, and orders.



```java
@Data
@AllArgsConstructor
public class UserSystem {

    @NotBlank(message = "user id can not be empty")
    @Size(min = 10, max=10, message = "user id must be exactly 10")
    private String userId;

    @Email(message = "please enter a valid email")
    private String email;

    @Size(max = 10, min = 10, message = "phone number must be 10 digits")
    @Pattern(regexp = "^05[0-9]+$",
            message = "phone must only have numbers")
    private String phone;



    @NotBlank()
    @Size(min = 8, max=40)
    @Pattern(regexp =
            "^(?=.*[A-Za-z])(?=.*[0-9])(?=.*[!@#$%^&*(),.?\":{}|<>]).*$")
    private String password;

    @NotBlank(message = "name can not be empty")
    @Size(min = 4, max=40, message = "name must be between 4 and 40 in length")
    @Pattern(regexp = "\\p{L}+", message = "name must only have letters")
    private String name;

    @NotNull(message = "age can not be empty")
    @Positive(message = "age must be a positive number")
    @Min(value = 14, message = "age must be larger than 13")
    private int age;

    @NotBlank(message = "address can not be empty")
    @Size(min = 30, max = 200)
    private String address;

    @NotBlank(message = "payment method can not be empty")
    @Size(min = 16, max=16)
    @Pattern(regexp = "[0-9]+$", message = "payment must only have numbers")
    private String paymentMethod;

    @NotNull()
    private boolean isSubscriber;
}
```

3.1 Orders Model:

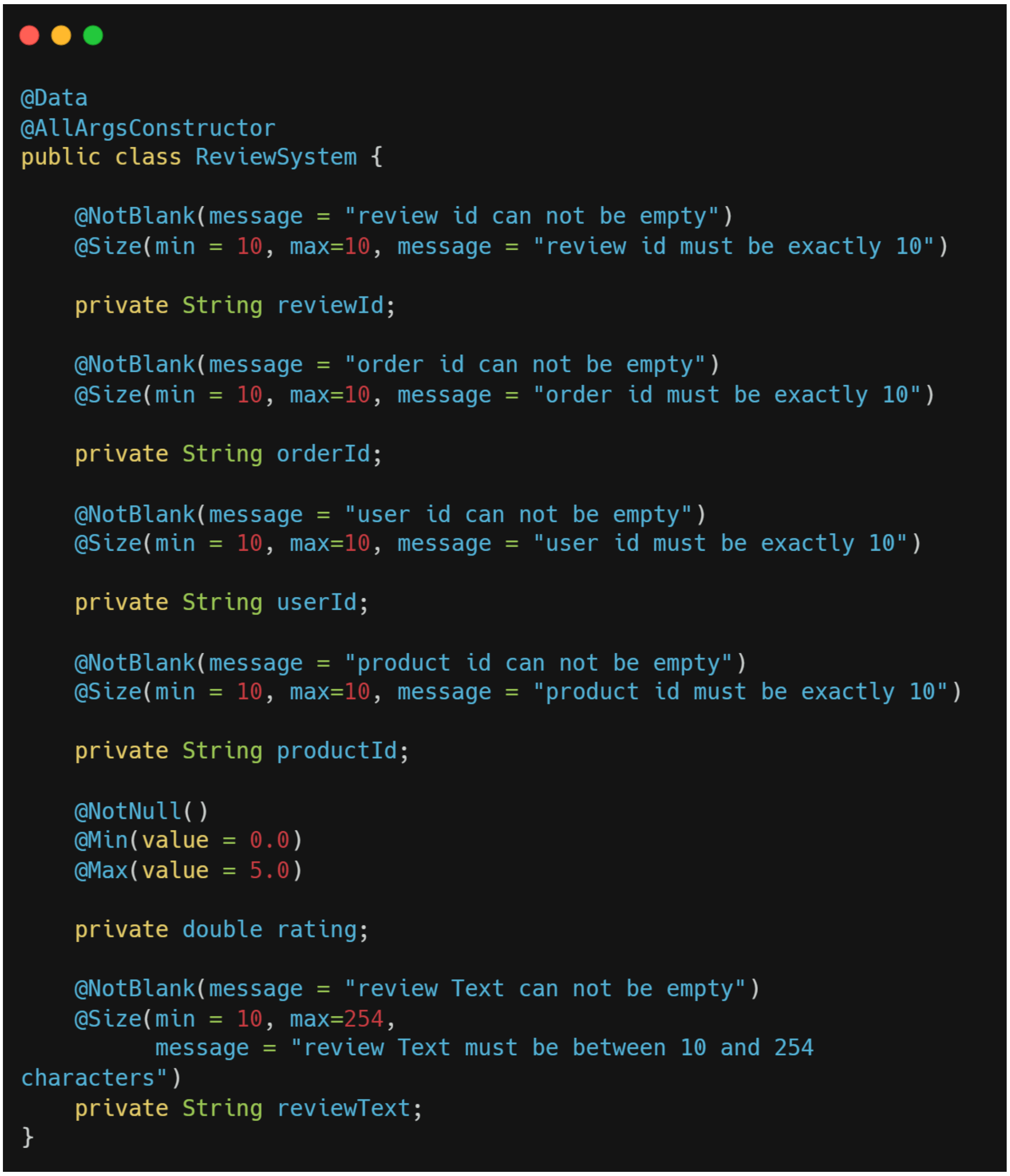
meant to handle the order process along with invoices.



```java
@Data
@AllArgsConstructor
public class OrderSystem {

    @NotBlank(message = "order id can not be empty")
    @Size(min = 10, max=10, message = "order id must be exactly 10")
    private String orderId;

    @NotBlank(message = "user id can not be empty")
    @Size(min = 10, max=10, message = "user id must be exactly 10")
    private String userId;

    @NotBlank(message = "product id can not be empty")
    @Size(min = 10, max=10, message = "product id must be exactly 10")
    private String productId;

    @NotNull(message = "date can not be null")
    @JsonFormat(pattern = "yyyy-MM-dd")
    @PastOrPresent(message = "order date must be in the past or present")
    private LocalDate orderDate;

    @NotNull()
    @Min(value = 0.1)
    @Max(value = 999999999)
    private double price;

    @NotBlank(message = "address can not be empty")
    @Size(min = 30, max = 200)
    private String shippingAddress;



    @NotBlank()
    @Pattern(message = "must equal to 'canceled', 'shipped', 'returned',
    'delivered', 'confirmed' only",
            regexp = "^(?i (canceled|shipped|returned|
             coordinator|confirmed|delivered)$")

    private String orderStatus;


}
```

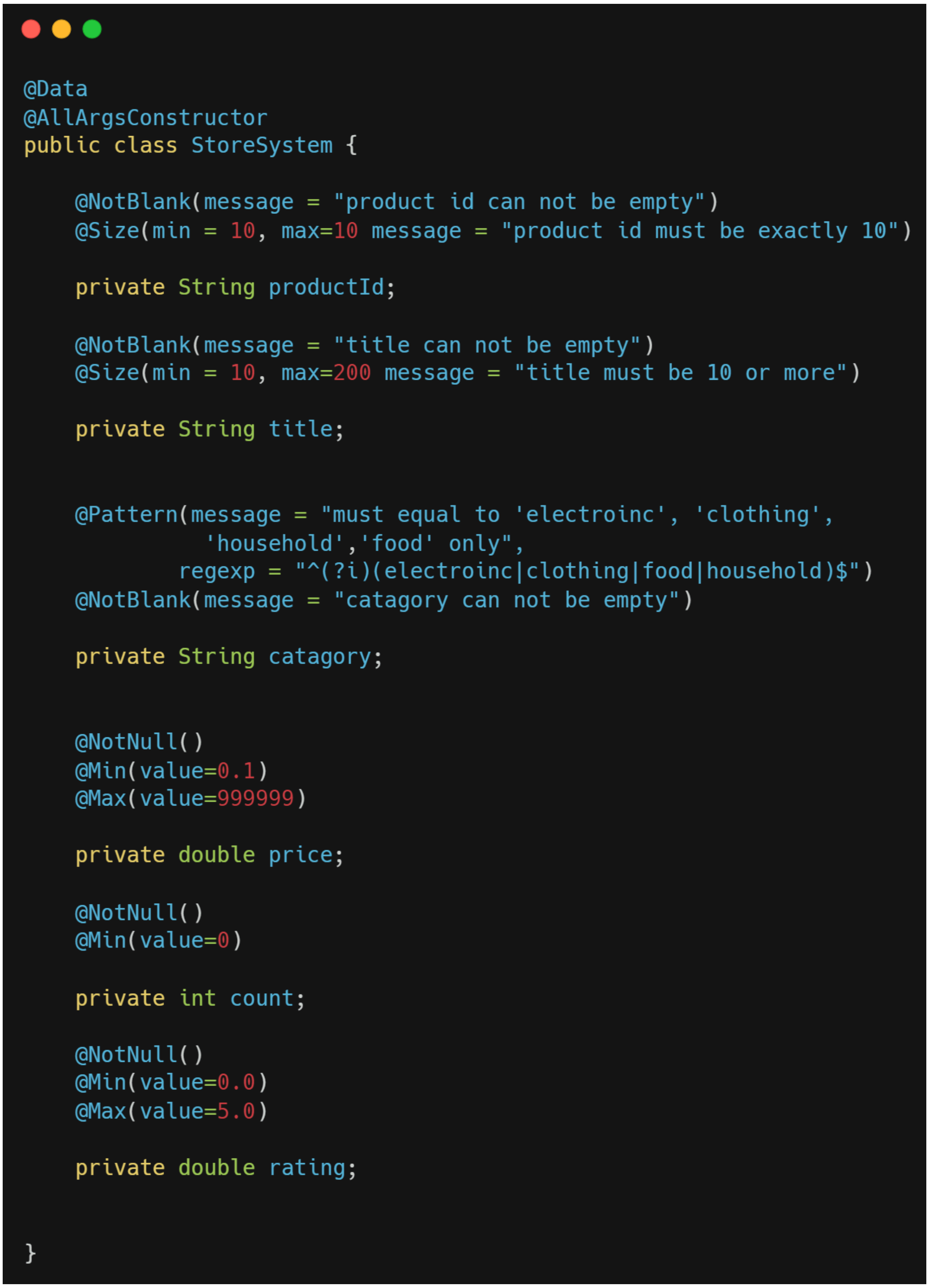
4.1 Reviews Model:

Reviews model is meant to handle review creations



```java
@Data
@AllArgsConstructor
public class ReviewSystem {

    @NotBlank(message = "review id can not be empty")
    @Size(min = 10, max=10, message = "review id must be exactly 10")

    private String reviewId;

    @NotBlank(message = "order id can not be empty")
    @Size(min = 10, max=10, message = "order id must be exactly 10")

    private String orderId;

    @NotBlank(message = "user id can not be empty")
    @Size(min = 10, max=10, message = "user id must be exactly 10")

    private String userId;

    @NotBlank(message = "product id can not be empty")
    @Size(min = 10, max=10, message = "product id must be exactly 10")

    private String productId;

    @NotNull()
    @Min(value = 0.0)
    @Max(value = 5.0)

    private double rating;

    @NotBlank(message = "review Text can not be empty")
    @Size(min = 10, max=254,
          message = "review Text must be between 10 and 254
characters")
    private String reviewText;
}
```

5.1 Store Model:
meant to handle items and their ratings and counts



```java
@Data
@AllArgsConstructor
public class StoreSystem {

    @NotBlank(message = "product id can not be empty")
    @Size(min = 10, max=10 message = "product id must be exactly 10")

    private String productId;

    @NotBlank(message = "title can not be empty")
    @Size(min = 10, max=200 message = "title must be 10 or more")

    private String title;


    @Pattern(message = "must equal to 'electroinc', 'clothing',
              'household','food' only",
            regexp = "^(?i)(electroinc|clothing|food|household)$")
    @NotBlank(message = "catagory can not be empty")

    private String catagory;


    @NotNull()
    @Min(value=0.1)
    @Max(value=999999)

    private double price;

    @NotNull()
    @Min(value=0)

    private int count;

    @NotNull()
    @Min(value=0.0)
    @Max(value=5.0)

    private double rating;


}
```