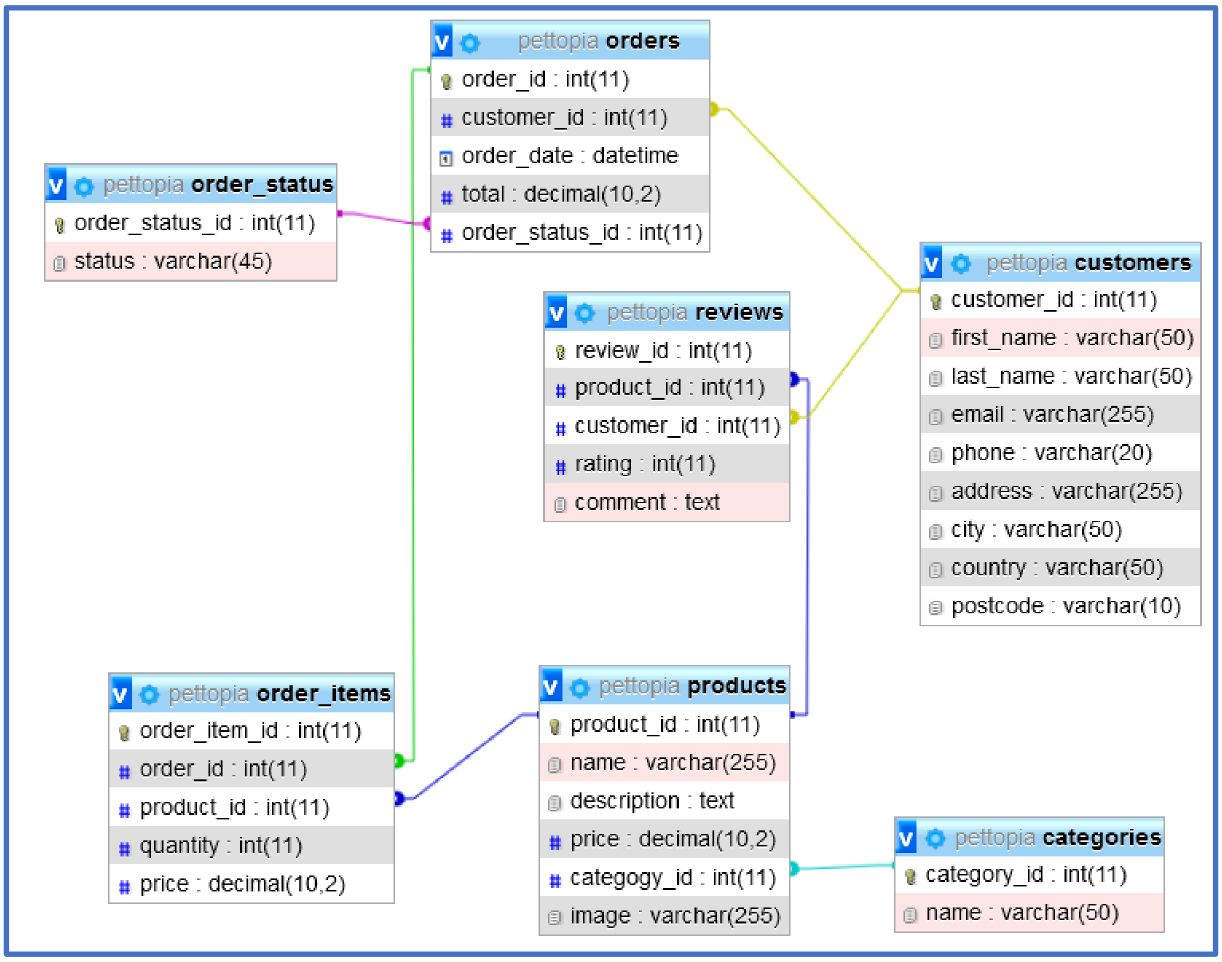
Pettopia is an online pet store that offers a wide range of pet supplies, pet food, and accessories for dogs, cats, birds, and small pets. The store provides pet owners with high-quality products at competitive prices, ensuring the best value for their money. In addition, the store offers convenience and flexibility with online ordering, quick delivery, and easy returns, making it a one-stop-shop for all pet needs.

Pettopia's target market includes pet owners who value their pets as members of their family and are willing to spend money to ensure their well-being. This includes both existing pet owners and potential new pet owners who are looking to adopt a pet. The store will target customers of all ages, with a focus on millennials and Gen Z, who are increasingly adopting pets and are more likely to shop online.

For this assignment, you must develop a REST API for Pettopia.

The database (the script for which is provided in the starter code) I have provided for you looks like the following:



The *products* table has a one-to-many relationship with the *order_items* table, as many products can appear in multiple orders and each order item can only correspond to one product.

The *products* table also has a one-to-many relationship with the *reviews* table, as a product can have multiple reviews, but each review corresponds to only one product.

The *products* table has a many-to-one relationship with the *categories* table, as many products can belong to the same category and each category can contain multiple products.

The *orders* table has a one-to-many relationship with the *order_items* table, as each order can have multiple order items, but each order item corresponds to only one order.

The *orders* table also has a many-to-one relationship with the *customers* table, as many orders can belong to the same customer, but each order corresponds to only one customer.

The *orders* table also has a many-to-one relationship with the *order_status* table, a single order status can be associated with multiple orders, but each order can only have one order status.

The *customers* table has a one-to-many relationship with the *orders* table, as a customer can place multiple orders, but each order corresponds to only one customer.

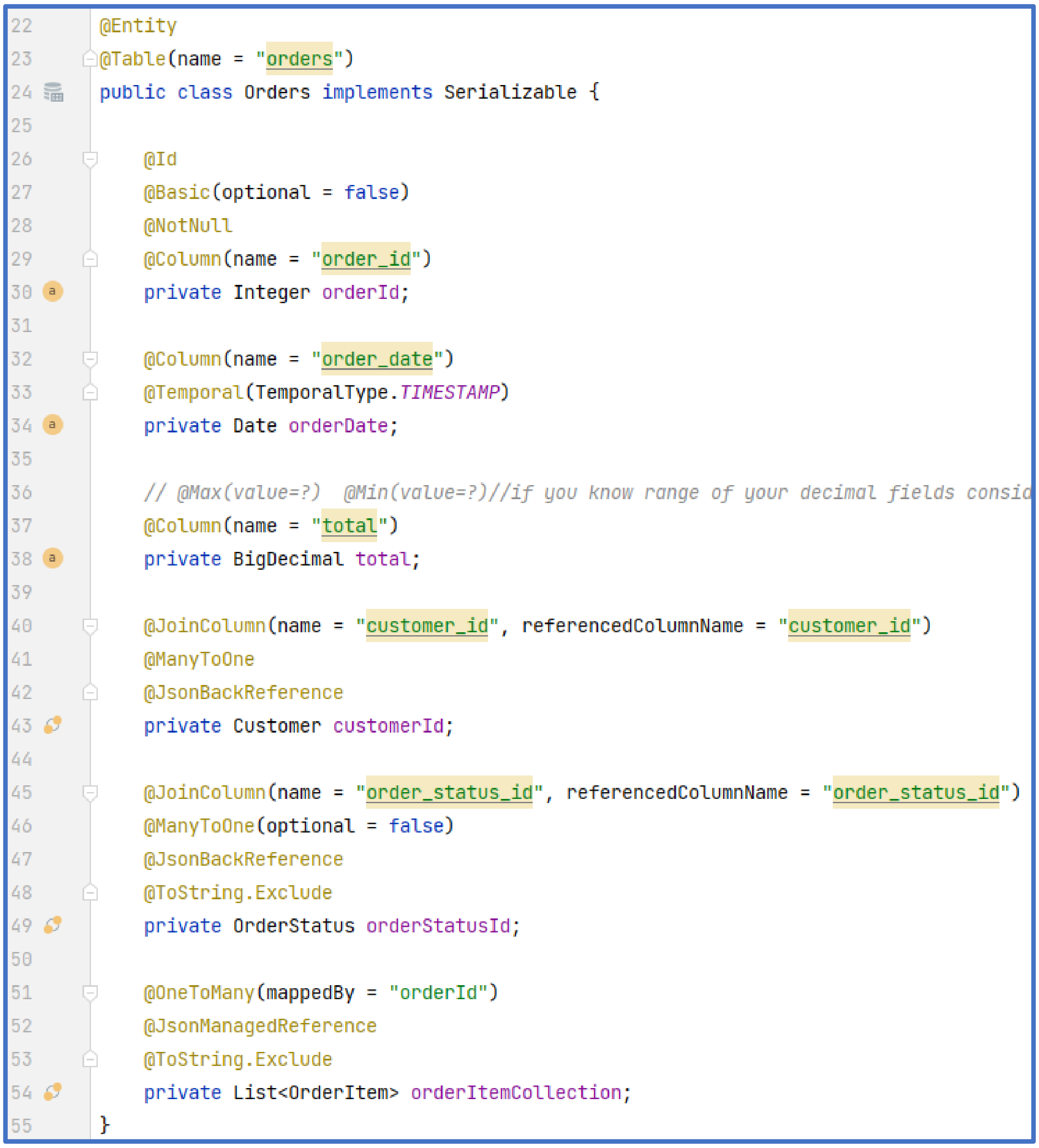
The *customers* table also has a one-to-many relationship with the *reviews* table, as a customer can write multiple reviews but each review corresponds to only one customer.

The *reviews* table has a many-to-one relationship with both the *products* and *customers* tables, as many reviews can correspond to the same product or customer but each review corresponds to only one product or customer.

The *categories* table has a one-to-many relationship with the *products* table, as a category can contain multiple products, but each product can belong to only one category.

There may be occasions where an order does not contain any order_items. Pettopia are aware of such inconsistencies and have earmarked such orders for archival in the future.

For each table in the database, there is a corresponding entity class (in the starter code) where the relationships between the various tables is maintained. For example:



```java
@Entity
@Table(name = "orders")
public class Orders implements Serializable {

    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "order_id")
    private Integer orderId;

    @Column(name = "order_date")
    @Temporal(TemporalType.TIMESTAMP)
    private Date orderDate;

    // @Max(value=?)  @Min(value=?)//if you know range of your decimal fields consid
    @Column(name = "total")
    private BigDecimal total;

    @JoinColumn(name = "customer_id", referencedColumnName = "customer_id")
    @ManyToOne
    @JsonBackReference
    private Customer customerId;

    @JoinColumn(name = "order_status_id", referencedColumnName = "order_status_id")
    @ManyToOne(optional = false)
    @JsonBackReference
    @ToString.Exclude
    private OrderStatus orderStatusId;

    @OneToMany(mappedBy = "orderId")
    @JsonManagedReference
    @ToString.Exclude
    private List<OrderItem> orderItemCollection;
}
```

To Do.

1. You must expose the *Customer* table in the *Petopia* database with a suitable API. The following is **required** functionality where you must support the GET ꜝ, POST, PUT and DELETE HTTP methods. All responses from these HTTP methods should be in JSON by default but XML representations should be available if desired by the client.

   GET an individual *Customer* based on *id*.

   GET all *Customers* (appropriate use of pagination is required).

   Both GET requests should adhere to HATEOAS principles:

   1. The GET an individual Customer should include a link to *all Customers*.
   2. The GET all *Customers* should include two links. One "self" link and another link that when followed will display (any) associated *Orders* that the *Customer* has, along with related *OrderItem* and *Products data*.

   At least one of the GET and one of the POST requests for this task should be consumed by a client application written in a language other than Java.

(30 Marks)

2. For a specified order or customer return an invoice.

   An invoice is a document that outlines the details of a transaction between a seller and a buyer. It is typically used in commercial transactions to request payment from the buyer for goods or services that have been provided. An invoice would not typically show order items that have been previously shipped and paid for, unless they are part of a larger order that includes items that are still outstanding or have yet to be paid for. An invoice typically includes details only for the items that are currently being invoiced and that require payment. If a customer has already paid for some of the items in a previous transaction, those items would not be included in the current invoice.

   However, if a customer has outstanding payments or if there are items that were not fulfilled in a previous order, those items may be included in the current invoice along with any new items being invoiced.

   Invoices differ from organisation to organisation (and from region to region). You can decide on the exact details and format of Pettopia's invoice yourself.

   In the database, there is an *Order_Status* table (essentially a lookup table). This table stores the various order status types (Shipped, Delivered, Processing, Cancelled and Pending) within Pettopia. For the purposes of this exercise, you can assume that invoices need only be produced for orders which are either pending or being processed. Should an invoice be requested for an order which has a status of

"Shipped", "Delivered" or "Cancelled" then no invoice will be produced – how will you handle such an occurrence?

(15 Marks)

3. Pettopia have carried out research to identify possible diversification opportunities. The following has been identified:

    - **Grooming services:** Pettopia could offer pet grooming services in addition to selling pet products. This could include services such as bathing, clipping nails, and grooming.
    - **Boarding and day-care services:** Pettopia could offer pet boarding and day-care services for customers who need a safe and comfortable place for their pets while they're away.
    - **Training services:** Pettopia could offer pet training services to help pet owners train their pets and correct behaviour problems.
    - **Pet adoption services:** Pettopia could partner with local animal shelters and rescue organisations to offer pet adoption services to customers.
    - **Pet-related travel services:** Pettopia could look to offer pet-related travel information, such as pet-friendly holiday rentals, pet-friendly hotels, and pet-friendly transportation options. This can include mapping data and location-specific data.
    - **Pet-related events and activities:** Pettopia could organise and sponsor pet-related events and activities, such as pet-friendly garden parties, charity walks, and adoption events.
    - **Pet-related content and media:** Pettopia could distribute pet-related content and media, such as videos, articles, and podcasts, to engage and educate customers.
    - **Subscription services:** Pettopia could offer pet-related subscription services, such as monthly pet toy or treat boxes.
    - **Loyalty-programme:** Create a loyalty program to retain customers and incentivise repeat purchases.
    - **Personalisation:** Use data from a user's profile and behaviour to personalise content and services via a REST API. This can include personalised recommendations and targeted marketing.
    - **Pet health tracking:** Pettopia could provide a REST API that tracks a pet's health data, including vaccinations, check-ups, and other medical information, and send reminders for future appointments.

    Many of the above could be combined with a newsletter, highlighting to customers new product offerings and services. Consider returning a QR code, that when scanned that will act as a link to where the (latest) newsletter can be downloaded from.

You must complement what you developed for parts 1 and 2 for this assignment with a **comprehensive** extension to the API that will allow Pettopia to realise one of the diversification opportunities outlined above, or alternatively, one that you deem suitable yourself. To implement this aspect of the assignment, you will more than likely have to change the database structure (add new table(s)) and this will necessitate a change to the existing entity classes or the introduction of newer classes.

You should aim to be **creative** and **innovative** around this API and look to avoid rehashing the functionality you developed for parts 1 and 2. Feel free to use 3rd party API's/libraries to help you with this task as well as **varying the request/response types the methods consume/produce.**

As you know, API methods can return JSON/XML/CSV etc, but they can also return images/audio/video/compressed files/PDFs. Custom HTTP headers can also be specified in a request to pass additional information to the API. For example, using a custom header to request that a thumbnail or large image for a specified product is required in or as part of a response.

(55 Marks)

Notes and Stipulations:

1. This assignment is due on Friday, March 31st at 6pm. I am using Github classroom to manage this assignment and you are required to commit your work to the repository during your weekly practical class. The reality is that you will more than likely commit much more regularly than that. For each commit that you miss your final mark will be reduced by 5%.
2. Your API must be documented using Swagger.
3. You must upload your final solution to Moodle. The version you upload to Moodle should correspond to the final commit on GitHub. Your solution must include the client that you developed for task 1).
4. GET requests can be tested in a web browser. All others can be tested using Postman or equivalent.
5. You must adhere to **best practice when developing your REST API**.
6. The API is to be developed using **Spring Boot**.
7. You must use **MAVEN** to manage your projects dependencies.
8. You must use **the H2 Database.**
9. You must use suitable **bean validation** where necessary, some of which is already provided for you.
10. You must use (embedded) **Tomcat** as your server/container.
11. Use appropriate measures to ensure that all **errors are handled** gracefully.

Why would you develop an application as an API and not as an MVC style application?

1. **Scalability:** APIs are designed to be scalable and can handle many requests and users. This makes them ideal for building applications that are expected to grow and handle high volumes of traffic.
2. **Reusability:** APIs are typically built with the goal of making their functionality available to other applications and developers. By building an application as an API, you can make its features and functionality available for reuse in other projects or for integration with other systems. *
3. **Flexibility:** APIs are often more flexible than traditional MVC architectures because they can be accessed from a variety of different platforms and devices, including web browsers, mobile devices, and other applications.
4. **Separation of concerns:** APIs can help to separate the concerns of the backend application from the frontend user interface. This can make it easier to maintain and update the backend functionality without affecting the user interface. You would have seen this in assignment on, where the back end was developed with Spring Boot but students were able to choose their own view technology (some used Thymeleaf, others React, Angular or Vue).
5. **Security:** APIs can provide an additional layer of security by controlling access to the application's data and functionality. By controlling access through the API, you can ensure that only authorised users and applications can interact with the backend system.

*For instance, say you are building a mobile app that allows users to book flights and hotels. You may want to integrate your app with external services, such as airlines or travel booking sites, to provide users with more options and choices. In this case, building your application as an API would be more suitable than using a traditional MVC architecture, as it would allow other developers and services to easily access and use your app's functionality. For example, a travel booking site could use your app's API to allow users to search for and book flights and hotels directly from their site, without needing to build their own booking system from scratch.

Overall, the decision to build an application as an API instead of using an MVC architecture will depend on the specific requirements and goals of the application. In some cases, an MVC architecture may be more appropriate, while in others, an API may be the better choice.