Revision Notes on Product Controller and API Design

Introduction

In this session, we covered the design and implementation of a product controller along with various APIs to manage products within a microservice architecture. The class was structured around implementing basic CRUD (Create, Read, Update, Delete) operations for a product catalog service. A significant focus was placed on understanding how to properly structure APIs using design principles, especially within the Model-View-Controller (MVC) framework.

Key Concepts

API Terminology:

1. **API Wrappers**: These are essentially contracts between different parts of a system or between systems. They describe what functions are available, how they should be called, and what data they return. For our purposes, drafting API wrappers involved defining the endpoint and the basic function without detailing the internal workings【8:12†transcript.txt】.
2. **API Types Discussed**:
   - **GET**: Fetch data from the server.
   - **POST**: Create a new resource.
   - **PATCH**: Update an existing resource, usually partially.
   - **PUT**: Replace an entire resource or create it if not found.
   - **DELETE**: Remove an existing resource【8:6†transcript.txt】.

Data Structures:

DTOs are used to ensure separation between internal data and the data exposed to client applications, hiding any sensitive or unnecessary information【8:4†transcript.txt】【8:7†transcript.txt】.

2. **Product Model and Category Model**: These should include all necessary fields (e.g., ID, name, description, price, image URL for products, and similar properties for categories) and serve as the primary data structure for storing information【8:12†transcript.txt】【8:14†transcript.txt】.

Annotations and Entity Management:

1. **Annotations in Spring**:
   - `@RequestMapping` : Used to map web requests onto methods in request-handling classes.
   - `@RequestBody` : Maps the body of the HTTP request to an object, enabling easy access to incoming data【8:6†transcript.txt】【8:11†transcript.txt】.
2. **Path and Request Variables**:
   - Using path variables (`@PathVariable`) lets you extract variables from the URI for use in method calls, such as mapping a product ID from the URL to a function argument【8:5†transcript.txt】.
   - For `POST` and `PATCH` requests, the body of the request needs to be mapped correctly to DTOs using the `@RequestBody` annotation.

Approach to API Design

Step-by-Step Implementation:

1. **Defining the Models**:
   - Start by creating models for Products and Categories, including relevant fields.
   - Ensure that DTOs are set up for data transfer to reduce method call dependency【8:16†transcript.txt】.
2. **Creating the Product Controller**:

appropriate HTTP method annotations【8:12†transcript.txt】【8:9†transcript.txt】.

3. **Utilizing DTOs:**
   - Use DTOs to map data between layers, ensuring the DTO includes only the necessary fields required for client-server communication.
   - Hide any non-required fields to prevent unnecessary data exposure【8:15†transcript.txt】【8:13†transcript.txt】.
4. **Endpoints and Mapping:**
   - Discuss and map CRUD endpoints, e.g., for POST (`/products`), GET (both singular and plural `/products/{id}` , `/products`), PATCH, PUT, and DELETE using the REST standards【8:11†transcript.txt】.
   - Implement various annotations to map function calls to specific HTTP requests, ensuring proper data flow through the use of request mappings and valid data structures【8:18†transcript.txt】.

Practical Session Guidance

- The session's implementation part required learners to actively engage with setting up their systems. Class time was dedicated to ensuring models were correctly defined and APIs effectively wrapped without delving deeply into the intricacies of service implementation【8:9†transcript.txt】【8:12†transcript.txt】.
- A significant portion of the session was student-centric, focusing on direct implementation practice. Guidance was provided primarily through high-level suggestions, with expectations set around independent repository management and implementation efforts【8:19†transcript.txt】.

These notes summarize the structured approach to designing a robust product controller within a microservice architecture, covering both theoretical and practical aspects as discussed during the class.