**Product Commenting & Likes Module: Technical Documentation 💬**

## **1. Overview**

The **Product Commenting & Likes Module** is a core component that facilitates user engagement with products in the marketplace. It provides the functionality for users to post comments, create threaded replies, and like or unlike comments. The module is engineered using **Clean Architecture** principles, ensuring a modular, maintainable, and highly reusable codebase.

## **2. Module Structure**

The module's codebase is organized into distinct layers, aligning with the principles of Clean Architecture. This structure separates business logic from data persistence and presentation layers.

marketplace/

│── core/

│ ├── entities/

│ │ └── comment.py

│ ├── repositories/

│ │ └── comment\_repository.py

│ └── services/

│ └── comment\_service.py

│

│── api/

│ ├── serializers/

│ │ └── comment\_serializer.py

│ ├── views/

│ │ └── comment\_views.py

│ └── urls/

│ └── comment\_urls.py

## **3. Core Components**

### **3.1 Entities (core/entities/comment.py)**

Entities are pure Python dataclasses representing the core data models of the module.

* CommentEntity: Represents a user comment, linked to a product and a user. It includes a field for a parent comment to support threaded replies.
* CommentLikeEntity: Captures the relationship between a user and a comment they have liked.

### **3.2 Repositories (core/repositories/comment\_repository.py)**

The repository layer abstracts database interactions and provides a clean interface for the service layer.

* get\_comments\_for\_product(product\_id): Fetches all top-level comments and their nested replies for a given product.
* create\_comment(user, product, text, parent=None): Creates a new comment or a reply to an existing one.
* toggle\_like(user, comment): Adds or removes a like for a comment by a specific user. This ensures a user can only have one like per comment.

### **3.3 Services (core/services/comment\_service.py)**

The service layer contains all business logic and orchestrates the flow of data between the API and repository layers.

* **Comment Validation:** Validates that the specified product and parent comment (if any) exist before creating a new comment.
* **Like Logic:** Manages the toggling of likes, preventing duplicate likes from the same user.
* **Threaded Replies:** Implements the logic for creating and retrieving nested comments.

### **3.4 Serializers (api/serializers/comment\_serializer.py)**

Serializers transform entity objects into JSON for API responses, handling data formatting and nesting.

* CommentSerializer: Serializes a CommentEntity, including nested serialization for replies and the total like count.

### **3.5 Views (api/views/comment\_views.py)**

The views expose REST API endpoints and serve as the entry point for requests.

* ProductCommentsView:
  + **GET** /products/<product\_id>/comments/: Retrieves all comments and their replies for a specific product.
  + **POST** /products/<product\_id>/comments/: Adds a new top-level comment or a reply to a comment.
* CommentLikeView:
  + **POST** /comments/<comment\_id>/like/: Toggles a like for a specific comment.

### **3.6 URLs (api/urls/comment\_urls.py)**

This file defines the URL patterns that map requests to their corresponding views.

Python

urlpatterns = [

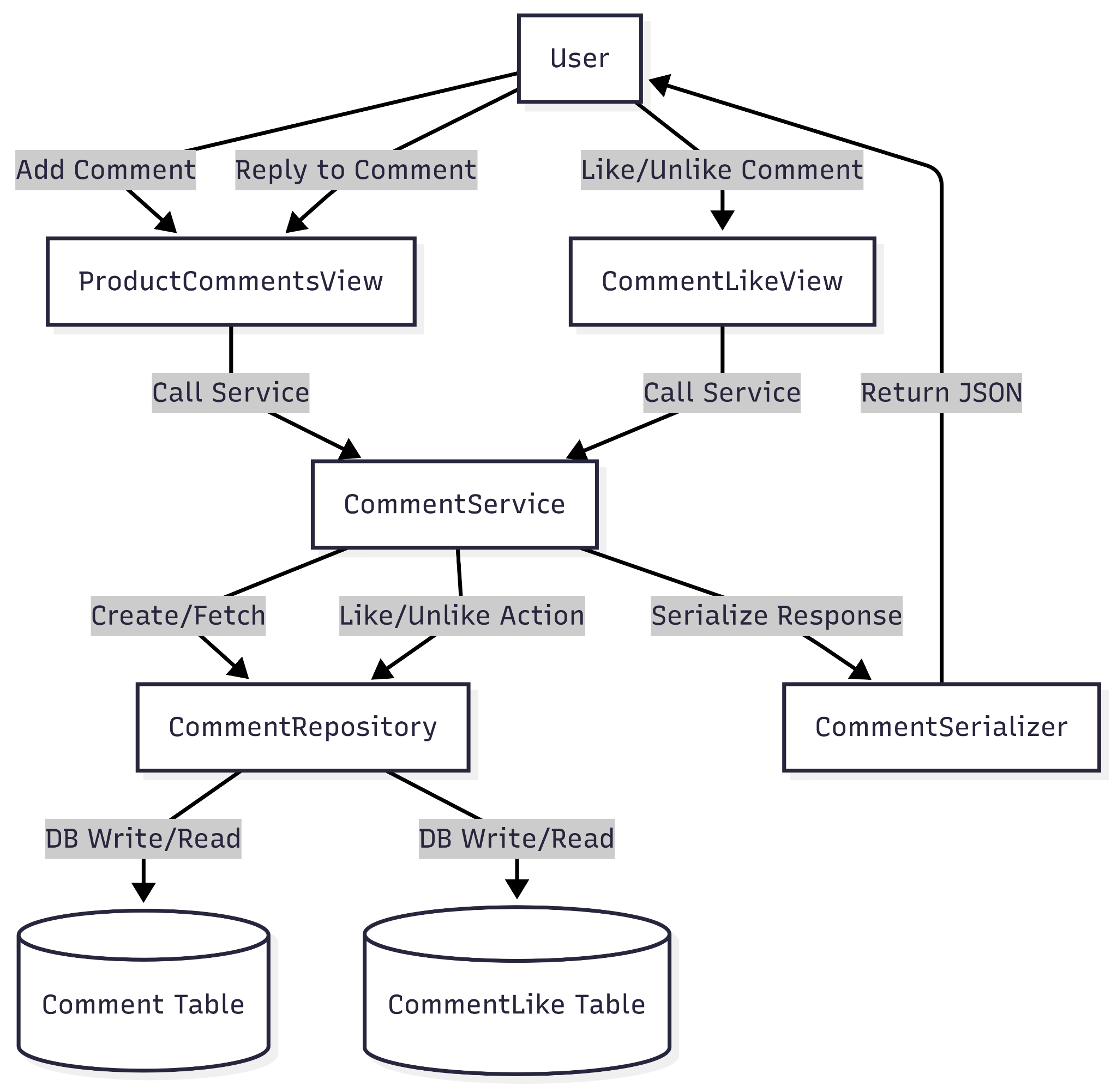
path("products/<int:product\_id>/comments/", ProductCommentsView.as\_view()),

path("comments/<int:comment\_id>/like/", CommentLikeView.as\_view()),

]

## **4. Operational Flow**

The following diagram illustrates the flow of a user interaction, from an API request to a database operation.



## **5. API Examples**

### **5.1 Fetch Comments**

* **Endpoint:** GET /api/products/12/comments/
* **Response:**
* JSON

[

{

"id": 1,

"user": "tino",

"text": "Is this still available?",

"likes": 3,

"replies": [

{

"id": 2,

"user": "maxwell",

"text": "Yes, it is!",

"likes": 1,

"replies": []

}

]

}

]

### **5.2 Add Comment**

* **Endpoint:** POST /api/products/12/comments/
* **Request Body:**
* JSON

{

"text": "Great product!"

}

### **5.3 Like/Unlike Comment**

* **Endpoint:** POST /api/comments/1/like/
* **Result:** This request toggles the like status for the comment with ID 1.

## **6. Future Extensibility**

This module is designed to be easily extensible to support additional features. Potential extensions include:

* **Moderation:** Add administrator APIs to allow for the editing or deletion of comments.
* **Notifications:** Implement a system to notify users when their comments are replied to or liked.
* **Analytics:** Integrate tracking to monitor metrics such as comment and like counts per product over time.