1. **Videos**: Model to store video files or links.
2. **Chapter**: Represents a chapter in a course. It contains a title, description, associated videos, comments, multiple-choice questions (MCQs), likes, etc.
3. **Comment**: Model to store comments made by users on chapters. It's structured using a tree-node model.
4. **Language**: Represents the language used in courses.
5. **Category**: Represents the categories under which courses are classified. It also uses a tree-node model.
6. **Course**: Represents a course, including its name, category, chapters, price, discounts, language, etc.
7. **Likes**: Model to store likes given by users on chapters.
8. **MCQ** and **MCQTest**: Models to handle multiple-choice questions and tests consisting of MCQs.
9. **Subscription**: Model to manage user subscriptions to courses and associated MCQ tests.
10. **Announcement**:
    * Represents announcements with fields like poster, title, description, and redirect\_link.
11. **Content**:
    * Represents various types of content such as "About Us", "Terms and Conditions", etc. with fields like title, description, and category.
12. **Enquiry**:
    * Represents user inquiries with fields like name, mobile, email, message, and is\_read.
13. **Faqs**:
    * Represents frequently asked questions with fields like question and answer.
14. **Roles**:
    * Represents user roles such as admin, moderator, etc.
15. **CustomUserManager** and **User**:
    * Custom user authentication system with fields for user details such as name, email, mobile, gender, profile image, role, status, etc.
16. **Assets**:
    * Represents media assets like images with methods to handle resizing upon saving.
17. **Base**:
    * Abstract base model providing common fields like created\_at and updated\_at to other models.
18. **SocialAccountURL**:
    * Represents social media account URLs with fields for the URL and social media platform.
19. **Teams**:
    * Represents teams with fields for name, image, designation, and social media URLs.
20. **Testimonial**:
    * Represents testimonials with fields for name, designation, message, and optionally, a link to a video testimonial.
21. **AssetsSerializer**:
    * Serializes the **Assets** model with fields for id, sizes, and file\_name.
22. **CategorySerializer**:
    * Serializes the **Category** model with all fields.
23. **ChapterSerializer**:
    * Serializes the **Chapter** model with all fields.
24. **CommentSerializer**:
    * Serializes the **Comment** model with all fields.
25. **ContentSerializer**:
    * Serializes the **Content** model with all fields.
26. **CourseSerializer**:
    * Serializes the **Course** model with all fields and includes extra validation for certain fields.
27. **EnquirySerializer**:
    * Serializes the **Enquiry** model with fields for id, name, mobile, email, and message. It also includes extra validation for the mobile field.
28. **FaqSerializer**:
    * Serializes the **Faqs** model with all fields.
29. **LanguageSerializer**:
    * Serializes the **Language** model with all fields.
30. **LikesSerializer**:
    * Serializes the **Likes** model with all fields.

**ContentTest**

* **test\_list**: Checks if the API endpoint for listing content returns the expected data and HTTP status code.
* **test\_create**: Verifies if creating a new content item through the API works correctly and returns the expected data and HTTP status code.
* **test\_update**: Ensures that updating a content item via the API produces the desired changes and returns the correct HTTP status code.
* **test\_delete**: Tests the deletion of a content item through the API and verifies the HTTP status code and response data.

**EnquiryTest**

* **test\_list**: Checks if the API endpoint for listing enquiries returns the expected data and HTTP status code.
* **test\_create**: Verifies if creating a new enquiry via the API works correctly and returns the expected data and HTTP status code.
* **test\_update**: Ensures that updating an enquiry item via the API produces the desired changes and returns the correct HTTP status code.
* **test\_delete**: Tests the deletion of an enquiry item through the API and verifies the HTTP status code and response data.

**FaqsTest**

* **test\_list**: Checks if the API endpoint for listing FAQs returns the expected data and HTTP status code.
* **test\_create**: Verifies if creating a new FAQ item via the API works correctly and returns the expected data and HTTP status code.
* **test\_update**: Ensures that updating an FAQ item via the API produces the desired changes and returns the correct HTTP status code.
* **test\_delete**: Tests the deletion of an FAQ item through the API and verifies the HTTP status code and response data.

get\_object(self, pk): Retrieves a Category instance by its primary key (pk).

retrieve(self, request, \*args, \*\*kwargs): Retrieves a single Category instance by ID. If the Category is found, it transforms the instance into a dictionary format and returns it.

list(self, request, \*args, \*\*kwargs): Retrieves a list of Category instances, optionally filtered and sorted based on query parameters. It supports pagination and transforms the list of instances into a list of dictionaries.

partial\_update(self, request, \*args, \*\*kwargs): Updates a Category instance partially. It validates the input data, performs the partial update, and returns the updated instance.

create(self, request, \*args, \*\*kwargs): Creates a new Category instance. It validates the input data, creates the instance, and returns the created instance.

delete(self, request, \*args, \*\*kwargs): Deletes a Category instance by ID.

transform\_single(self, instance): Transforms a single Category instance into a dictionary format.

transform\_list(self, data): Transforms a list of Category instances into a list of dictionaries.