Ex1:

* **What is REST?**

REST (Representational State Transfer) is an architectural style used for designing networked applications. It relies on a stateless, client-server communication model and uses standard HTTP methods such as GET, POST, PUT, and DELETE.

* **Why Do We Need REST?**

We use REST because it provides numerous benefits when it comes to building scalable, maintainable, and efficient web services. Some key reasons to use REST are:

1. **Scalability:** Because REST relies on stateless communication, servers can handle a higher number of concurrent requests as there’s no need to store session information between requests.
2. **Flexibility:** REST allows clients to choose the data they need, making it flexible to use for different devices and applications, including mobile apps, web apps, and IoT devices.
3. **Separation of Concerns:** REST allows a clear separation between the client and the server. The client sends requests and receives responses, while the server handles processing. This makes the system more modular and easier to scale.
4. **Standardization and Easy Integration:** Since REST uses standard HTTP methods, it can be easily integrated with most programming languages and frameworks, making it widely adopted.
5. **Human-Readable:** URLs in REST are often human-readable and self-descriptive, making it easy to understand the request and the resources being queried.
6. **Cache:** REST supports caching, which improves the performance of the application by reducing the number of requests that need to be made to the server.

* **Steps for Building REST in Django**
* **Step 1**: Install Django and Django REST Framework

pip install django djangorestframework

Add rest\_framework to INSTALLED\_APPS in settings.py

* **Step 2**: Create a Django Project and App

django-admin startproject myproject

cd myproject

django-admin startapp myapp

* **Step 3**: Define Models

Create a model in models.py:

from django.db import models

class Product(models.Model):

name = models.CharField(max\_length=100)

price = models.DecimalField(max\_digits=10, decimal\_places=2)

description = models.TextField()

Run migrations:

python manage.py makemigrations

python manage.py migrate

* **Step 4**: Create Serializers
* **What is Serialization?**

Serialization is the process of converting complex data (e.g., model instances) into JSON format for easy transmission over the web.

* **Why Do We Need Serializers?**

Django’s models store data in a format that is not directly compatible with JSON. Serializers help convert this data into a format that clients can understand.

Example serializers.py:

from rest\_framework import serializers

from .models import Product

class ProductSerializer(serializers.ModelSerializer):

class Meta:

model = Product

fields = '\_\_all\_\_'

* **Step 5**: Create Views

Create API views using Django REST Framework in views.py:

from rest\_framework import generics

from .models import Product

from .serializers import ProductSerializer

class ProductListCreateView(generics.ListCreateAPIView):

queryset = Product.objects.all()

serializer\_class = ProductSerializer

class ProductDetailView(generics.RetrieveUpdateDestroyAPIView):

queryset = Product.objects.all()

serializer\_class = ProductSerializer

* **Step 6**: Define URLs

In urls.py:

from django.urls import path

from .views import ProductListCreateView, ProductDetailView

urlpatterns = [

path('products/', ProductListCreateView.as\_view(), name='product-list'),

path('products/<int:pk>/', ProductDetailView.as\_view(), name='product-detail'),

]

* **Step 7**: Test the API

Run the Django development server:

python manage.py runserver

Use tools like Postman or CURL to test the endpoints.

**Examples**

***Example 1: Basic REST API for Books***

Let’s say you want to create a simple API that lists all books and allows clients to add new books. You would create a Book model, a serializer, and a view that handles GET and POST requests. Here’s the implementation:

1. **Model (models.py):**

from django.db import models

class Book(models.Model):

    name=models.CharField(max\_length=100)

    price=models.IntegerField()

    author=models.CharField(max\_length=50)

    quantity=models.IntegerField()

    image = models.ImageField(upload\_to='book\_images/', null=True, blank=True)

    def \_\_str\_\_(self):

        return self.name

    class Meta:

        app\_label = 'book'

1. **Serializer (serializers.py):**

class BookSerializer(serializers.ModelSerializer):

    class Meta:

        model = Book

        fields = "\_\_all\_\_"

1. **View (views.py):**

@api\_view(["GET"])

def getBooks(request):

  books=Book.objects.all()

  serializer=BookSerializer(books,many=True)

  return Response(serializer.data)

1. **URL (urls.py):**

urlpatterns=[

    path('login/',views.login2,name="login"),

    path('books/',views.getBooks,name="allBook"),

    path('orders/<int:customer\_id>/', views.get\_orders, name="orders"),

    path('cart/<int:bookID>/',views.cart\_changing,name="to\_cart"),

    path('buying/<int:customer\_id>/',views.buying,name="buying"),

]

***Example 2: Handling Requests***

@api\_view(["GET","POST"])

def cart\_changing(request,bookID):

    book=get\_object\_or\_404(Book,pk=bookID)

    if request.method=="POST":

        customerID=request.data.get("customer")

        customer=get\_object\_or\_404(Customer,pk=customerID)

        quantity=request.data.get("quantity")

        if book.quantity>=quantity:

            cart, created=Cart.objects.get\_or\_create(customer=customer, book\_pk=book, defaults={"quantity": 0})

            cart.quantity+= quantity

            cart.save()

            book.quantity-=quantity

            book.save()

            return Response({"message":"Add ok"},status=status.HTTP\_200\_OK)

        else:

            return Response({"message":"Not enough in stock"},status=status.HTTP\_400\_BAD\_REQUEST)

    elif request.method=="GET":

        book\_send=BookSerializer(book)

        return Response({

            "book":book\_send.data,

            "message":"This is your book"

        },status=status.HTTP\_200\_OK)

***Example 3: Authentication in REST APIs***

from rest\_framework.permissions import IsAuthenticated

from rest\_framework\_simplejwt.authentication import JWTAuthentication

class SecureBookList(APIView):

    authentication\_classes = [JWTAuthentication]

    permission\_classes = [IsAuthenticated]

    def get(self, request):

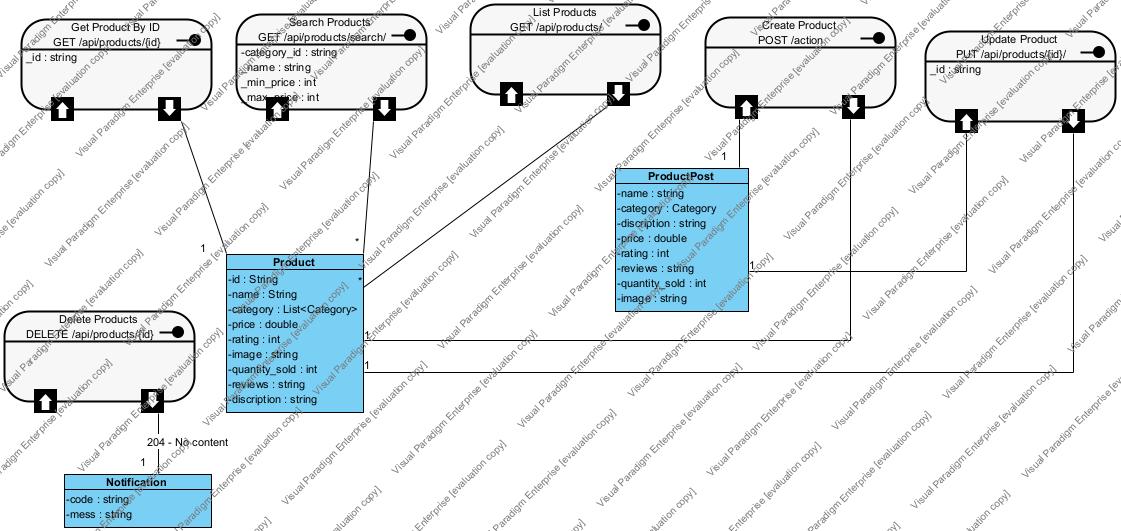
        books = Book.objects.all()

        serializer = BookSerializer(books, many=True)

        return Response(serializer.data)

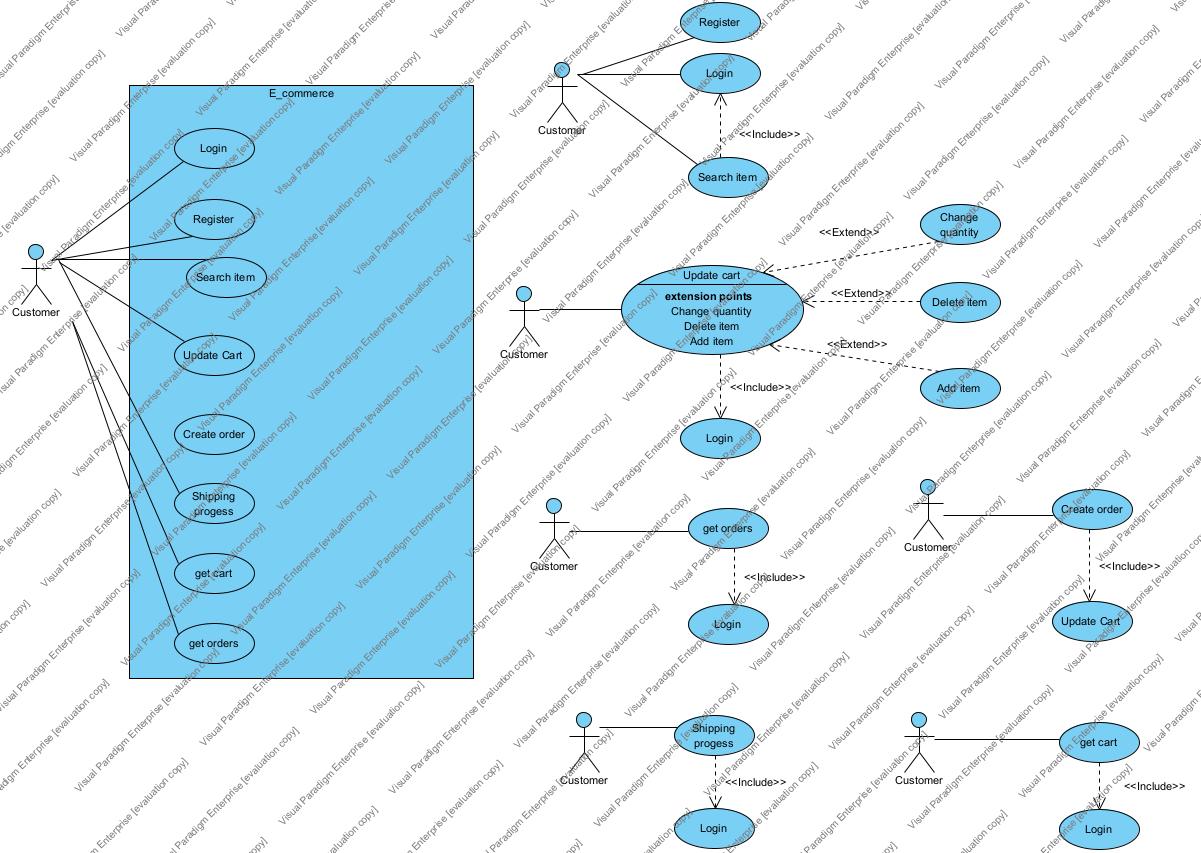
**Conclusion**

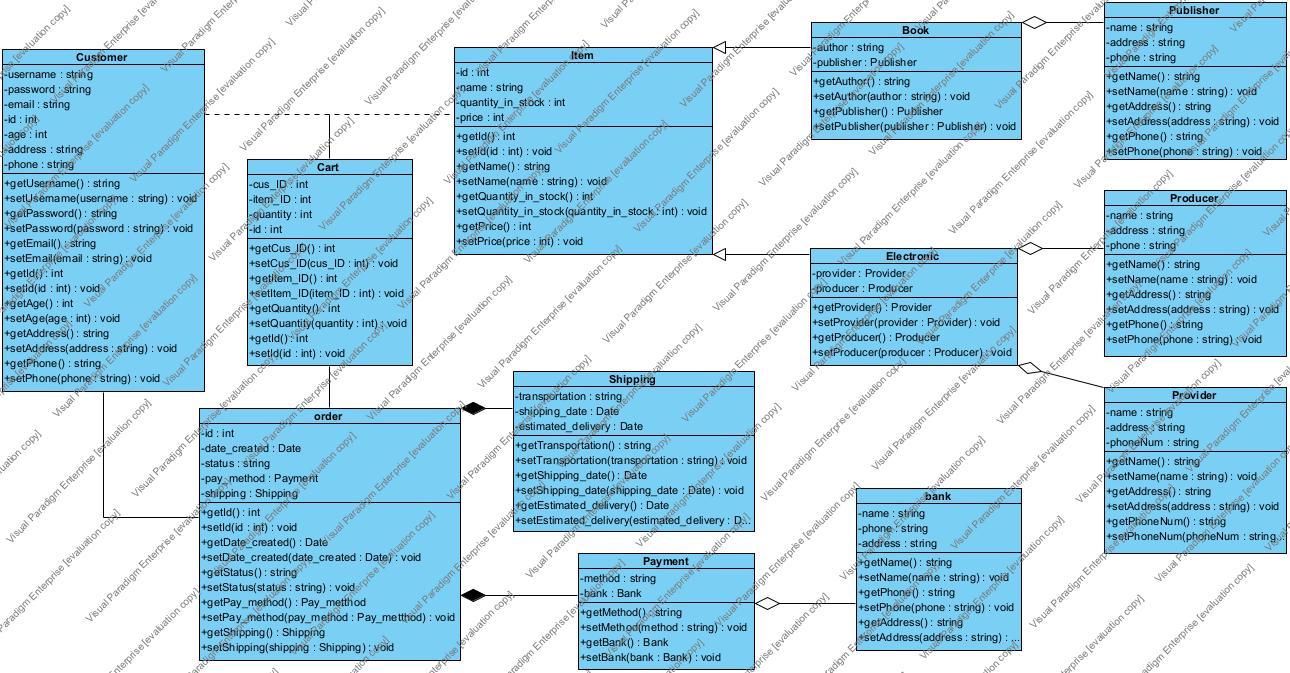
REST is an essential architectural style for building scalable and flexible APIs. By following the principles of REST, you can design services that work seamlessly across different platforms. Django REST Framework simplifies the process of building REST APIs by providing tools like serializers and views, making it easy to handle requests and send responses. Serialization is an essential part of this process, helping to transform complex data types into a format that can be transferred over the network and used by clients.



**Ex2:**

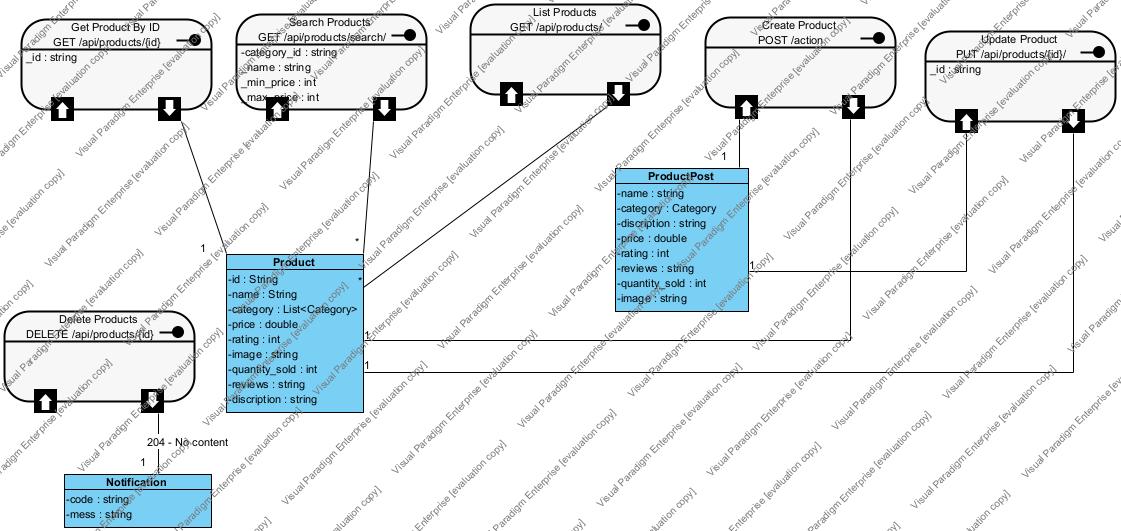
**a,**



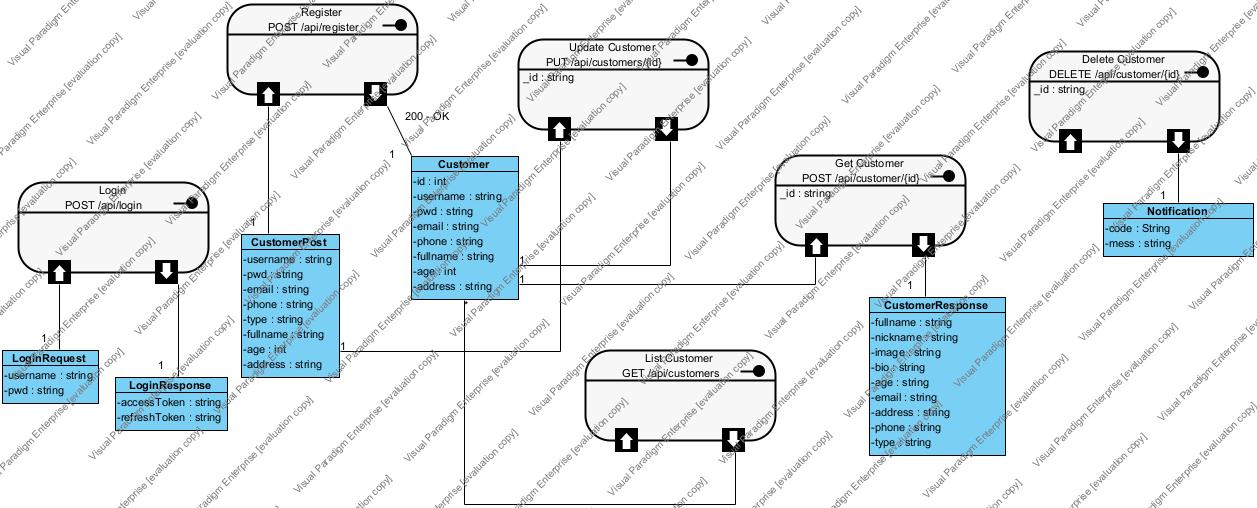
****

**b,**

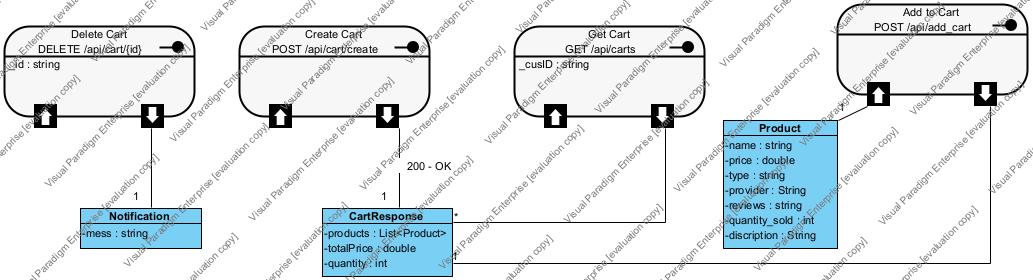
**Product/Item:**



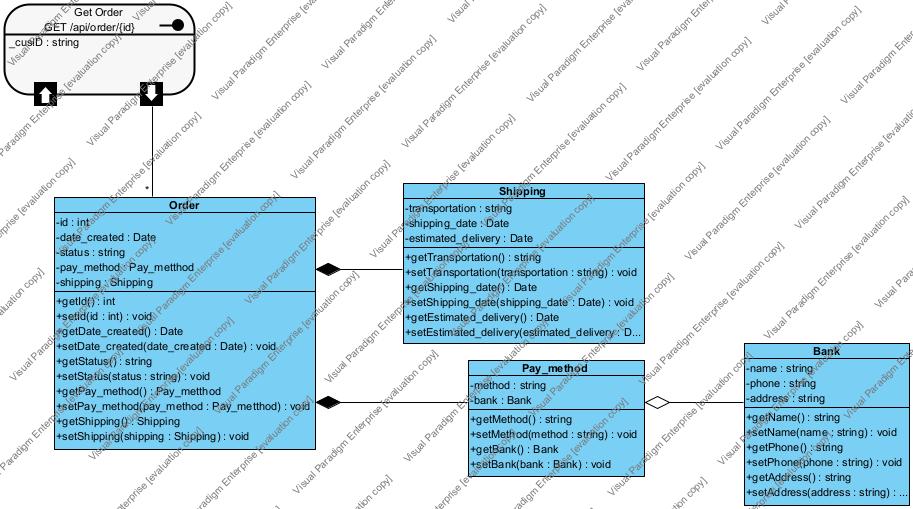
**Customer:**

****

**Cart:**

****

**Order:**

****