**Phạm Đức Chính – B21DCCN181**

**1.**

**What is REST?**

REST (Representational State Transfer) is an architectural style that defines a set of constraints for designing networked applications. It was introduced by Roy Fielding in his dissertation in 2000. REST is widely used for building web services due to its simplicity, scalability, and stateless nature. RESTful APIs use standard HTTP methods such as GET, POST, PUT, DELETE, and PATCH to interact with resources, making it an ideal choice for modern web applications and microservices.

**Key Characteristics of REST**

1. Client-Server Architecture: REST separates the client and server to allow independent evolution of both components.
2. Statelessness: Each request from a client must contain all necessary information, and the server does not store any client session state.
3. Cacheability: Responses can be cached to improve performance and reduce server load.
4. Layered System: REST APIs can be designed with multiple layers, such as authentication, load balancing, and caching layers, without the client knowing about them.
5. Uniform Interface: REST follows a standard set of operations to interact with resources, ensuring simplicity and consistency.
6. Code on Demand (optional): REST allows servers to send executable code (e.g., JavaScript) to clients when needed.

**Why do we need to use REST?**

1. Scalability: REST APIs can handle thousands of simultaneous requests efficiently by leveraging caching and load balancing techniques.
2. Interoperability: REST enables communication between different platforms, including web, mobile, and IoT devices, using standard formats like JSON and XML.
3. Flexibility and Extensibility: REST allows developers to modify and expand APIs without breaking existing client implementations.
4. Improved Performance: Due to statelessness and caching mechanisms, REST APIs offer improved response times and lower latency.
5. Simplicity and Maintainability: REST APIs use standard HTTP methods, making them easier to develop, test, and maintain.
6. Security: REST supports authentication and authorization mechanisms like OAuth, JWT, and API keys to secure access to resources.

**Steps for Building REST in Django**

Step 1: Install Django and Django REST Framework (DRF)

Django REST Framework (DRF) is a powerful toolkit for building RESTful APIs in Django. First, install Django and DRF using pip:

pip install django djangorestframework

Step 2: Create a Django Project and an App

Run the following commands to set up a Django project and app:

django-admin startproject myproject

cd myproject

django-admin startapp myapp

Step 3: Configure Django Settings

Modify settings.py to include Django REST Framework:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'rest\_framework',

'myapp',

]

Step 4: Define a Model in models.py

Create a simple Student model:

from django.db import models

class Student(models.Model):

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

age = models.IntegerField()

email = models.EmailField()

created\_at = models.DateTimeField(auto\_now\_add=True)

Run migrations to create the database table:

python manage.py makemigrations

python manage.py migrate

Step 5: Create a Serializer in serializers.py

Serialization converts complex data types into JSON format for API communication.

from rest\_framework import serializers

from .models import Student

class StudentSerializer(serializers.ModelSerializer):

class Meta:

model = Student

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

Step 6: Create API Views in views.py

Define a view using Django REST Framework’s ModelViewSet:

from rest\_framework import viewsets

from .models import Student

from .serializers import StudentSerializer

class StudentViewSet(viewsets.ModelViewSet):

queryset = Student.objects.all()

serializer\_class = StudentSerializer

Step 7: Configure URLs in urls.py

Use Django’s router system for API endpoints:

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import StudentViewSet

router = DefaultRouter()

router.register(r'students', StudentViewSet)

urlpatterns = [

path('', include(router.urls)),

]

Step 8: Run the Server and Test the API

Start the Django development server:

python manage.py runserver

Test the API using Postman or cURL:

curl -X GET http://127.0.0.1:8000/students/

**What is Serialization? Why do we need it?**

Serialization is the process of converting Django model instances into JSON or XML format so that they can be sent over the web. Deserialization converts this data back into Django model instances.

Why We Need Serialization?

1. Data Exchange: Allows web applications to communicate by sending structured data.
2. Database Interaction: Translates Django models into a format suitable for API consumption.
3. Validation: Ensures data consistency and prevents invalid entries.

**Examples of REST API Usage**

Example 1: Retrieving All Students (GET Request)

Endpoint: GET /students/ Response:

[

{

"id": 1,

"name": "John Doe",

"age": 22,

"email": "john@example.com",

"created\_at": "2024-03-05T12:00:00Z"

}

]

Example 2: Creating a New Student (POST Request)

Endpoint: POST /students/ Request Body:

{

"name": "Jane Doe",

"age": 21,

"email": "jane@example.com"

}

Response:

{

"id": 2,

"name": "Jane Doe",

"age": 21,

"email": "jane@example.com",

"created\_at": "2024-03-05T12:10:00Z"

}

Example 3: Updating a Student Record (PUT Request)

Endpoint: PUT /students/2/ Request Body:

{

"name": "Jane Smith",

"age": 22,

"email": "janesmith@example.com"

}

Response:

{

"id": 2,

"name": "Jane Smith",

"age": 22,

"email": "janesmith@example.com",

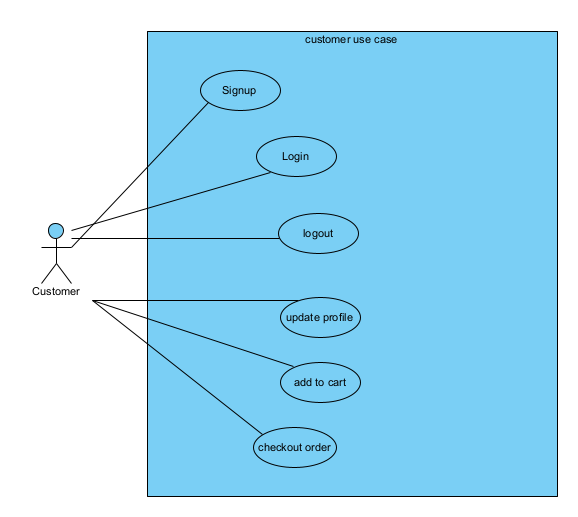
"created\_at": "2024-03-05T12:10:00Z"

}

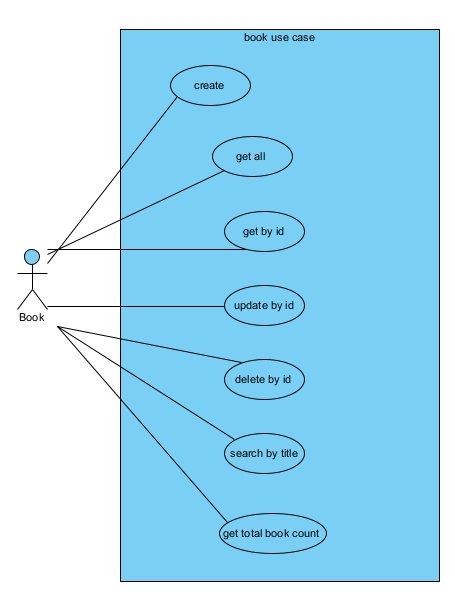
2.

a)

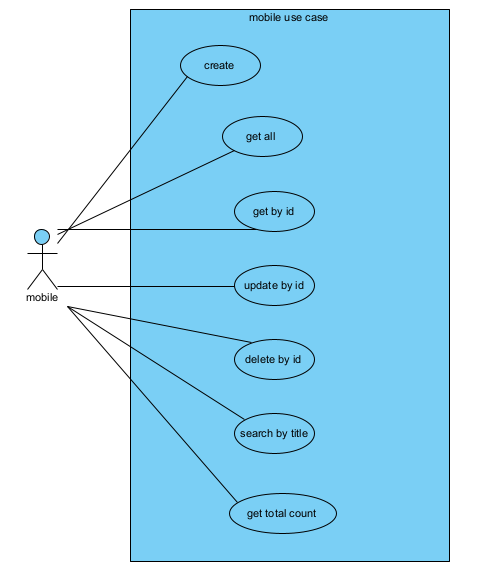
Customer use case diagram



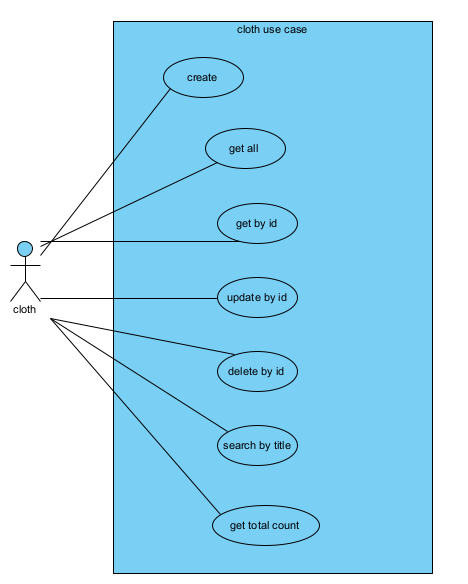
Book use case diagram



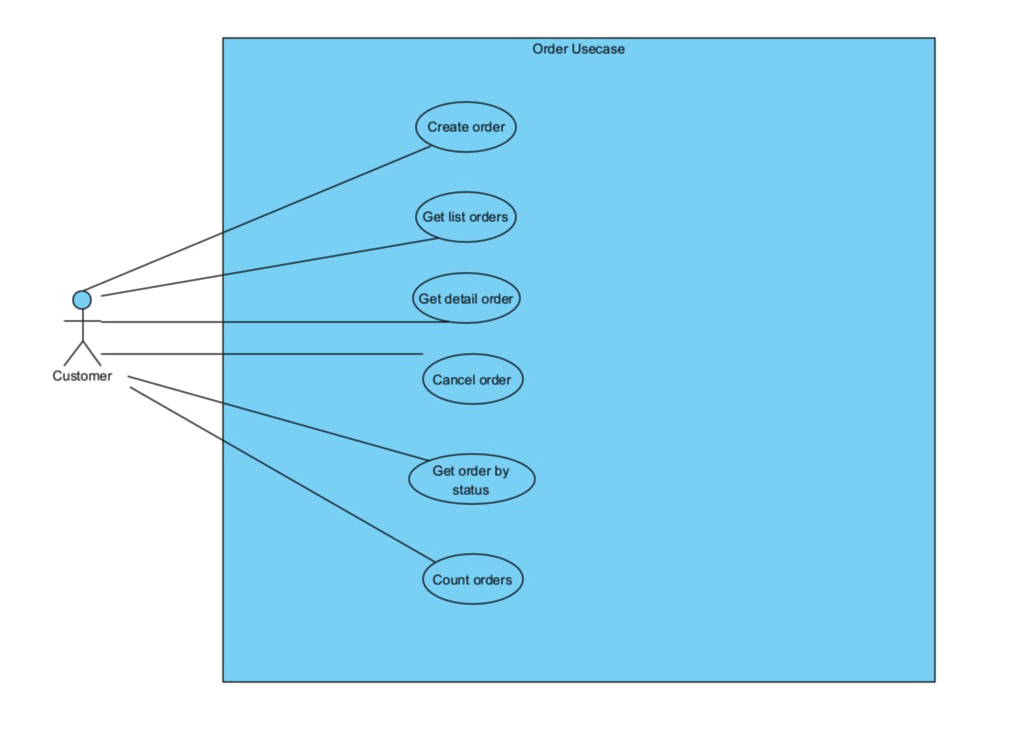
Mobile use case diagram



Cloth usecase diagram



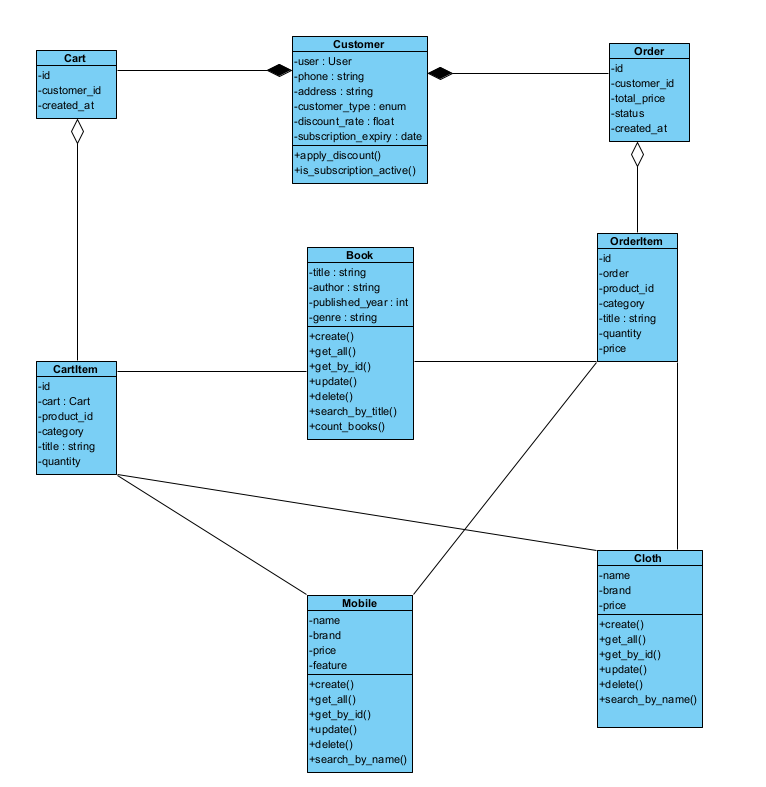
Order usecase



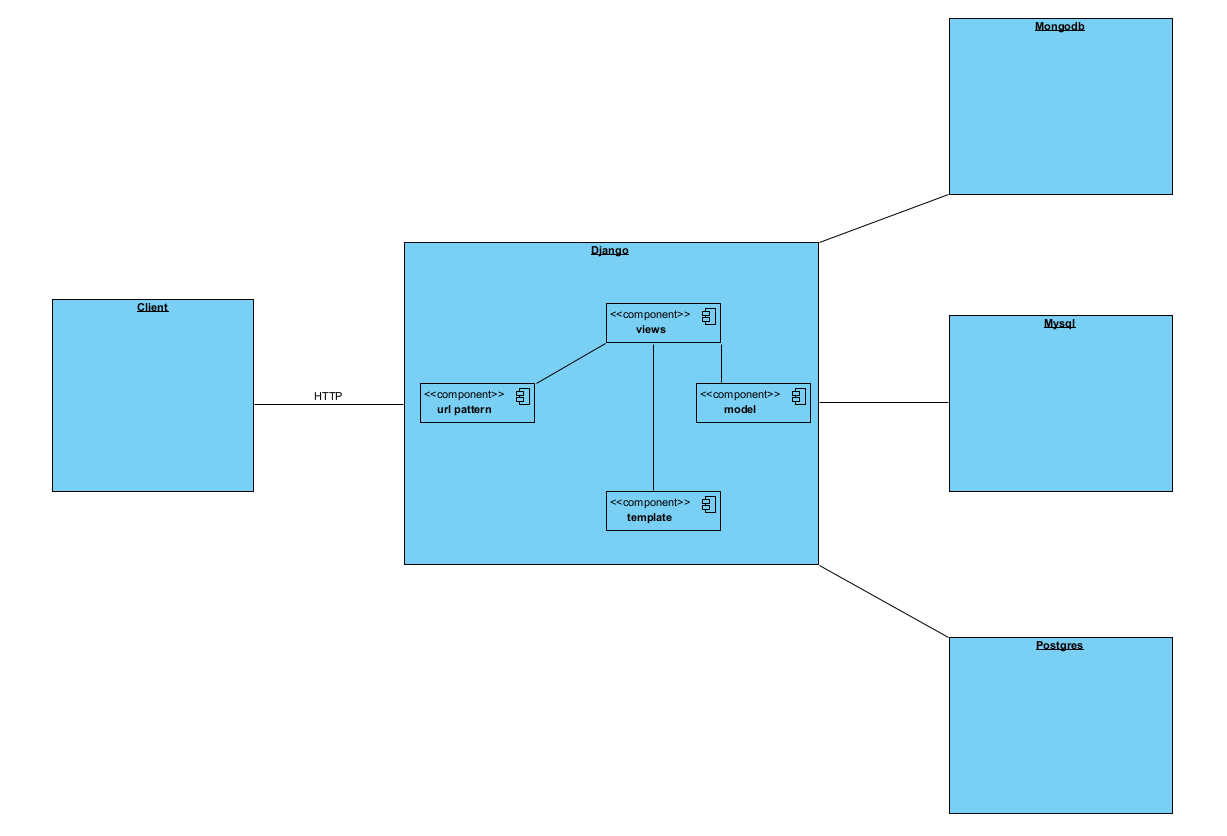
Cart usecase



b)



c)



d) code in github [deltaDC/django-ecommerce](https://github.com/deltaDC/django-ecommerce)