Curso de FastAPI

Diego Saavedra Miguel Amaya

Jul 24, 2024

Table of contents

1 Bienvenido

¡Bienvenido al Curso Completo de FastAPI!

En este curso, exploraremos todo, desde los fundamentos hasta las aplicaciones prácticas.

1.1 ¿De qué trata este curso?

Este curso completo me llevará desde los fundamentos básicos de la programación hasta la construcción de aplicaciones prácticas utilizando los frameworks Django y la biblioteca de React.

A través de una combinación de teoría y ejercicios prácticos, me sumergiré en los conceptos esenciales del desarrollo web y avanzaré hacia la creación de proyectos del mundo real.

Desde la configuración del entorno de desarrollo hasta la construcción de una aplicación web de pila completa, este curso me proporcionará una comprensión sólida y experiencia práctica con FastAPI.

1.2 ¿Para quién es este curso?

Este curso está diseñado para principiantes y aquellos con poca o ninguna experiencia en programación.

Ya sea que sea un estudiante curioso, un profesional que busca cambiar de carrera o simplemente alguien que quiere aprender desarrollo web, este curso es para usted. Desde adolescentes hasta adultos, todos son bienvenidos a participar y explorar el emocionante mundo del desarrollo web con FastAPI.

1.3 ¿Cómo contribuir?

Valoramos su contribución a este curso. Si encuentra algún error, desea sugerir mejoras o agregar contenido adicional, me encantaría saber de usted.

Puede contribuir a través del repositorio en linea, donde puede compartir sus comentarios y sugerencias.

Juntos, podemos mejorar continuamente este recurso educativo para beneficiar a la comunidad de estudiantes y entusiastas de la programación.

Este ebook ha sido creado con el objetivo de proporcionar acceso gratuito y universal al conocimiento.

Estará disponible en línea para cualquier persona, sin importar su ubicación o circunstancias, para acceder y aprender a su propio ritmo.

Puede descargarlo en formato PDF, Epub o verlo en línea en cualquier momento y lugar.

Esperamos que disfrute este emocionante viaje de aprendizaje y descubrimiento en el mundo del desarrollo web con FastAPI!

Part I

Unidad 1: Introducción a Python

2 Configuración y Sintaxis básica

3 Variables y Control de flujo

4 Funciones y Parámetros

Part II

Unidad 2: Estructura de Datos en Python

5 Listas y Tuplas

6 Diccionarios y Conjuntos

Part III

Unidad 3: Programación Orientada a Objetos en Python

7 Conceptos Básicos

8 Herencia y Polimorfismo

Part IV

Unidad 4: Herramientas de Desarrollo

9 Git y Github

Part V

Unidad 5: Introducción a FastAPI

10 Configuraciòn y Estructura

11 Modelos en FastAPI

Part VI

Unidad 6: Desarrollo Avanzado en FastAPI

12 Rutas y Validaciones en FastAPI

13 APIs RESTful con FastAPI

14 Pruebas Unitarias en FastAPI

15 Optimización y Rendimiento

16 Creación de una API de gestión de tareas utilizando FastAPI

Part VII

Unidad 7: Consumo API con FastAPI

17 Configuración y Uso de FastAPI para consumir APIs

18 Integración de APIs de terceros con FastAPI

Part VIII

Unidad 8: Desarrollo Avanzado

19 Manejo de Estado en FastAPI, Context API

20 Navegación en FastAPI con FastRouter

21 Aplicación en FastAPI que consume APIs de terceros

Part IX

Unidad 9: Prácticas Avanadas de Programación

22 Patrones de Diseño en FastAPI

23 Arquitectura de Software y Diseño Modular

Part X

Proyecto Final

24

25

26

Part XI

Laboratorios

27 Desarrollo del Backend para un E-Commerce con Django Rest Framework

27.1 1. Configuración Inicial del Proyecto

27.1.1 1.1. Crear un Proyecto Django

Abre tu terminal y ejecuta los siguientes comandos para crear un nuevo proyecto Django:

```
python -m venv env
source env/bin/activate
pip install django==4.2
django-admin startproject ecommerce_project .
cd ecommerce_project
```

27.1.2 1.2 Crear una Aplicación Django

Dentro del directorio del proyecto, crea una aplicación para manejar el e-commerce:

```
python manage.py startapp products
```

27.1.3 1.3 Instalar Django Rest Framework

Instala DRF usando pip:

```
pip install djangorestframework
```

27.1.4 1.4 Configurar el Proyecto

Añade 'rest_framework' y tu nueva aplicación 'products' a la lista INSTALLED_APPS en ecommerce_project/settings.py:

```
INSTALLED_APPS = [
    # ... otras apps
    'rest_framework',
    'products',
]
```

27.2 2. Definir el Modelo de Datos

27.2.1 2.1 Crear Modelos en products/models.py

Define los modelos para el e-commerce, como Product, Category, y Order:

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Product(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.ForeignKey(Category, related_name='products', on_delete=models.CASC
    stock = models.PositiveIntegerField()

    def __str__(self):
        return self.name

class Order(models.Model):
    product = models.ForeignKey(Product, related_name='orders', on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField()
    total_price = models.DecimalField(max_digits=10, decimal_places=2)
    order_date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Order {self.id} - {self.product.name}"
```

27.2.2 2.2 Crear y Aplicar Migraciones

Genera y aplica las migraciones para los modelos:

```
python manage.py makemigrations
python manage.py migrate
```

27.3 3. Crear Serializers

27.3.1 3.1 Definir Serializers en products/serializers.py

Los serializers se encargan de transformar los modelos en formatos JSON y viceversa:

```python
from rest_framework import serializers
from .models import Category, Product, Order

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = '__all__'

class ProductSerializer(serializers.ModelSerializer):
    category = CategorySerializer()

    class Meta:
        model = Product
        fields = '__all__'

class OrderSerializer(serializers.ModelSerializer):
    product = ProductSerializer()

    class Meta:
        model = Order
        fields = '__all__'
```

27.4 4. Crear Vistas y Rutas

27.4.1 4.1 Definir Vistas en products/views.py

Utiliza las vistas basadas en clases de DRF para crear y manejar las operaciones CRUD:

```python
from rest_framework import generics
from .models import Category, Product, Order
from .serializers import CategorySerializer, ProductSerializer, OrderSerializer

class CategoryListCreate(generics.ListCreateAPIView):
    queryset = Category.objects.all()
    serializer_class = CategorySerializer

class CategoryDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Category.objects.all()
    serializer_class = CategorySerializer

class ProductListCreate(generics.ListCreateAPIView):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer

class ProductDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Product.objects.all()
```

```
    serializer_class = ProductSerializer

class OrderListCreate(generics.ListCreateAPIView):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer

class OrderDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer
```

27.5 4.2 Configurar Rutas en products/urls.py

Define las rutas para acceder a las vistas:

```
from django.urls import path
from . import views

urlpatterns = [
    path('categories/', views.CategoryListCreate.as_view(), name='category-list-create'),
    path('categories/<int:pk>/', views.CategoryDetail.as_view(), name='category-detail'),
    path('products/', views.ProductListCreate.as_view(), name='product-list-create'),
    path('products/<int:pk>/', views.ProductDetail.as_view(), name='product-detail'),
    path('orders/', views.OrderListCreate.as_view(), name='order-list-create'),
    path('orders/<int:pk>/', views.OrderDetail.as_view(), name='order-detail'),
]
```

27.6 4.3 Incluir las URLs en ecommerce_project/urls.py

Añade las URLs de la aplicación al archivo principal de URLs del proyecto:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('products.urls')),
]
```

27.7 5. Probar la API

27.7.1 5.1 Ejecutar el Servidor de Desarrollo

Inicia el servidor de desarrollo de Django:

```
python manage.py runserver
```

27.8 5.2 Probar los Endpoints

Utiliza herramientas como Postman o cURL para probar los endpoints:

- Listar categorías: GET /api/categories/
- Crear categoría: POST /api/categories/
- Obtener categoría específica: GET /api/categories/{id}/
- Actualizar categoría: PUT /api/categories/{id}/
- Eliminar categoría: DELETE /api/categories/{id}/

Y lo mismo para productos y pedidos.

- Listar productos: GET /api/products/
- Crear producto: POST /api/products/
- Obtener producto específico: GET /api/products/{id}/
- Actualizar producto: PUT /api/products/{id}/
- Eliminar producto: DELETE /api/products/{id}/

28 Extra

Agreguemos Swagger a nuestro proyecto para tener una documentación de nuestra API.

28.1 1. Instalar Django Rest Swagger

Instala Django Rest Swagger usando pip:

```
pip install drf-yasg
```

28.2 2. Configurar Django Rest Swagger

Añade 'rest_framework_swagger' a la lista INSTALLED_APPS en ecommerce_project/settings.py:

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi

schema_view = get_schema_view(
    openapi.Info(
        title="E-commerce API",
        default_version='v1',
        description="API documentation for the E-commerce project",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="contact@ecommerce.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('products.urls')),
    path('docs/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-u
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
]
```

28.3 3. Configurar las URLs

Añade las URLs de Swagger al archivo principal de URLs del proyecto:

```
```
from rest_framework_swagger.views import get_swagger_view

schema_view = get_swagger_view(title='E-Commerce API')

urlpatterns = [
    ```
    path('docs/', schema_view),
]
```

> **Tip**
>
> Para evitar un error común es necesario instalr **setuptools** con el siguiente comando:
>
> ```
> pip install setuptools
> ```

Finalmente es necesario agregar el siguiente código al final del archivo settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'drf_yasg',
    'rest_framework',
    'products',
]

```
REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS': 'rest_framework.schemas.coreapi.AutoSchema'
}

CORS_ALLOWED_ORIGINS = [
    "http://localhost:8000",
    "http://127.0.0.1:8000",
    # Añade otros orígenes permitidos aquí
]
```

28.4 4. Probar la Documentación
```