Transcrição e Organização das aulas – 4 dias – 4 projetos – Pythonando – 30/10 a 2/11/2024

Organizado por José Alfredo F. Costa – UFRN – Para auxiliar alunos a utilizarem os conhecimentos / códigos

Transcrição e Organização da Aula 1 - Gerenciador de Senhas com Python

<https://www.youtube.com/watch?v=FmzljSpdwVc>

Esta aula da "4 Days 4 Projects" foca na criação de um gerenciador de senhas com Python puro, abordando fundamentos da linguagem, orientação a objetos e lógica de programação.

**Introdução e Motivação:**

O instrutor, Caio Sampaio, inicia com uma breve apresentação pessoal e da Pythonando. Ele destaca a alta demanda do mercado por programadores Python e a importância da qualificação para se destacar.

O projeto da aula, um gerenciador de senhas, é justificado pela necessidade de senhas fortes e únicas para cada serviço online, algo difícil de gerenciar manualmente. A aplicação criptografa as senhas com uma chave mestra, garantindo segurança mesmo se o arquivo de senhas for acessado indevidamente.

**Pré-requisitos e Materiais:**

Antes de iniciar a codificação, o instrutor recomenda:

* **Foco na compreensão:** Evitar codificar junto durante a live para melhor absorver os conceitos.
* **Material de apoio:** Disponível para download no site da Pythonando, com código e explicações.
* **Upgrade VIP (opcional):** Acesso à gravação das aulas, material extra e sessão ao vivo de dúvidas com o instrutor.
* **Certificado:** Disponível para todos os participantes (10 horas) e versão estendida (20 horas) para VIPs.

**Arquitetura do Projeto (MVT):**

O projeto segue o padrão MVT (Model-View-Template), separando as responsabilidades em:

* **Models:** Representam os dados (senhas) e interagem com o "banco de dados" (arquivo TXT).
* **Views:** Contém a lógica da aplicação, como criptografia e interação com os models.
* **Templates:** Interface com o usuário (terminal).

**Criptografia (Fernet):**

A criptografia utiliza a biblioteca cryptography do Python e o algoritmo Fernet. O processo envolve:

1. **Geração de Chave Aleatória:** Uma chave aleatória é gerada e usada para criptografar e descriptografar as senhas.
2. **Criptografia:** A senha é criptografada usando a chave e o algoritmo Fernet.
3. **Descriptografia:** A senha criptografada é descriptografada usando a mesma chave.

**Implementação (Código):**

# views/password\_views.py

import os

import sys

import base64

import hashlib

from datetime import datetime

from cryptography.fernet import Fernet, InvalidToken

# Adicionando o caminho do projeto ao sys.path para imports

sys.path.append(os.path.abspath(os.path.dirname(\_\_file\_\_)))

from models.password import Password # Importando o modelo

class FernetHasher:

RANDOM\_STRING\_CHARS = string.ascii\_lowercase + string.ascii\_uppercase

@classmethod

def get\_random\_string(cls, length=25):

random\_string = ''.join(secrets.choice(cls.RANDOM\_STRING\_CHARS) for \_ in range(length))

return random\_string

@classmethod

def create\_key(cls, archive=False):

key = cls.get\_random\_string()

hasher = hashlib.sha256(key.encode('utf-8'))

key = base64.b64encode(hasher.digest())

if archive:

path = cls.archive\_key(key)

return key, path

return key, None

@classmethod

def archive\_key(cls, key):

base\_dir = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../../"))

key\_dir = os.path.join(base\_dir, "keys")

file\_name = "key"

file\_path = os.path.join(key\_dir, f"{file\_name}.key")

i = 0

while os.path.exists(file\_path):

i += 1

file\_name = f"key\_{cls.get\_random\_string(5)}"

file\_path = os.path.join(key\_dir, f"{file\_name}.key")

with open(file\_path, "wb") as f:

f.write(key)

return file\_path

def \_\_init\_\_(self, key):

if not isinstance(key, bytes):

key = key.encode()

self.f = Fernet(key)

def encrypt(self, value):

if not isinstance(value, bytes):

value = value.encode('utf-8')

return self.f.encrypt(value)

def decrypt(self, value):

try:

decrypted\_value = self.f.decrypt(value).decode('utf-8')

except InvalidToken as e:

return "Token inválido"

return decrypted\_value

# models/password.py

import os

import sys

from datetime import datetime

# Adicionando o caminho do projeto ao sys.path para imports

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../")))

class BaseModel:

db\_dir = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../db"))

@classmethod

def get(cls):

table\_path = os.path.join(cls.db\_dir, f"{cls.\_\_name\_\_}.txt")

if not os.path.exists(table\_path):

open(table\_path, 'a').close() # Cria o arquivo se não existir

results = []

attributes = vars(cls)

with open(table\_path, "r") as arc:

for line in arc.readlines():

values = line.strip().split("|")

temp\_dict = dict(zip(attributes, values))

results.append(temp\_dict)

return results

def save(self):

table\_path = os.path.join(self.db\_dir, f"{self.\_\_class\_\_.\_\_name\_\_}.txt")

if not os.path.exists(self.db\_dir):

os.makedirs(self.db\_dir)

if not os.path.exists(table\_path):

open(table\_path, 'a').close()

string = ""

for key in vars(self):

string += str(vars(self)[key]) + "|"

with open(table\_path, "a") as arc:

arc.write(string.rstrip("|") + "\n")

class Password(BaseModel):

def \_\_init\_\_(self, domain, password):

self.domain = domain

self.password = password

self.created\_at = datetime.now().isoformat()

# templates/template.py (Interface com o Usuário)

from models.password import Password

from views.password\_views import FernetHasher

import os

while True:

action = input("Digite 1 para salvar uma nova senha ou 2 para ver uma senha salva: ")

if action == "1":

if len(Password.get()) == 0: # verifica se ja existe uma chave criada e salva no arquivo key.key

key, path = FernetHasher.create\_key(archive=True)

print("Sua chave foi criada com sucesso. Salve-a com cuidado!")

print(f"Chave: {key.decode('utf-8')}")

print(f"Caminho: {path}")

else: # caso ja exista uma chave criada, o usuário só precisa digitar

chave = input("Digite a chave: ")

dominio = input("Domínio: ")

senha = input("Senha: ")

f\_user = FernetHasher(chave)

senha\_criptografada = f\_user.encrypt(senha)

p1 = Password(dominio, senha\_criptografada)

p1.save()

elif action == "2":

dominio = input("Domínio: ")

chave = input("Chave: ")

f\_user = FernetHasher(chave)

data = Password.get()

password = None

for i in data:

if dominio == i["domain"]:

password = f\_user.decrypt(i['password'].encode())

break

if password:

print(f"Sua senha: {password}")

else:

print("Nenhuma senha encontrada para o domínio.")

else:

print("Opção inválida.")

**Explicação do Código:**

* **views/password\_views.py:** Contém a classe FernetHasher responsável pela geração da chave, criptografia e descriptografia. Usa hashlib para gerar hashes e base64 para codificação. O método archive\_key salva a chave em um arquivo, garantindo que cada chave tenha um nome único.
* **models/password.py:** Contém a classe Password, representando uma senha. Herda de BaseModel, que implementa a lógica de salvar e recuperar dados do arquivo TXT. O \_\_init\_\_ inicializa os atributos da senha (domínio, senha criptografada e data de criação).
* **templates/template.py:** Interface com o usuário. Solicita ações (salvar ou ver senha), recebe dados do usuário, interage com as views (criptografia/descriptografia) e models (salvamento/recuperação).

**Melhorias em relação à transcrição original:**

* **Organização em arquivos (MVT):** O código foi separado em arquivos seguindo o padrão MVT para melhor organização e reutilização.
* **Tratamento de erros:** Incluído tratamento de InvalidToken na descriptografia para lidar com chaves inválidas.
* **Gerenciamento de múltiplas chaves:** O archive\_key garante que cada chave seja salva em um arquivo separado com nome único.
* **Lógica de salvamento e recuperação:** A classe BaseModel implementa a lógica de interação com o arquivo TXT, permitindo reutilização para outros modelos.
* **Comentários:** Adicionados comentários para explicar a lógica do código.
* **Correção de erros:** Corrigidos pequenos erros presentes na transcrição original, como imports e chamada de métodos.
* **Formatação do código:** O código foi formatado para melhor legibilidade.

Este código aprimorado oferece uma estrutura mais organizada, robusta e reutilizável, além de melhor tratamento de erros e segurança no gerenciamento das chaves de criptografia.

Transcrição e Organização da Aula 2 - Aplicação WEB Full-Stack com Python e Django (4d4p #02)

<https://www.youtube.com/watch?v=4NeWvRo4zPk>

Esta aula foca no desenvolvimento de uma aplicação web full-stack com Python e Django, criando um sistema para gerenciamento de convidados e presentes de casamento.

**Introdução e Motivação:**

Caio Sampaio inicia destacando a importância de se especializar em uma área da programação para conseguir a primeira vaga como desenvolvedor. Ele enfatiza o desenvolvimento web como uma área com alta demanda e bons salários. Explica a diferença entre frontend (interface) e backend (lógica e processamento de dados) e apresenta Python com Django como uma stack robusta e utilizada por grandes empresas como YouTube e Instagram.

**Apresentação do Projeto:**

O projeto consiste em um sistema web para gerenciamento de casamento, com as seguintes funcionalidades:

* **Noivos:** Criam listas de presentes com nome, foto, preço e importância, além de gerenciar a lista de convidados.
* **Convidados:** Confirmam presença, recusam ou reservam presentes.
* **Visualização:** Gráfico mostrando a quantidade de presentes reservados e não reservados.

**Configuração do Ambiente e Projeto Django:**

1. **Criação do projeto:**

django-admin startproject core .

1. **Criação de apps:**
2. python3 manage.py startapp noivos

python3 manage.py startapp convidados

1. **Configuração do ambiente virtual (opcional, mas recomendado):**
2. python3 -m venv .venv
3. source .venv/bin/activate

pip install django

**Configurações Iniciais (settings.py):**

* **INSTALLED\_APPS:** Adicionar os apps criados ('noivos' e 'convidados').
* **TEMPLATES:** Configurar o diretório de templates.
* **STATIC\_URL, STATICFILES\_DIRS, STATIC\_ROOT:** Configurar os arquivos estáticos.
* **MEDIA\_URL, MEDIA\_ROOT:** Configurar os arquivos de mídia (uploads).
* LANGUAGE\_CODE, TIME\_ZONE - pt-br e America/Sao\_Paulo, respectivamente.

**Roteamento de URLs (urls.py):**

# core/urls.py

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('admin/', admin.site.urls),

path('noivos/', include('noivos.urls')),

path('convidados/', include('convidados.urls')),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

# noivos/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.home, name='home'),

path('lista\_convidados/', views.lista\_convidados, name='lista\_convidados'),

]

# convidados/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.convidados, name='convidados'),

path('responder\_presenca/', views.responder\_presenca, name='responder\_presenca'),

path('reservar\_presente/<int:id>/', views.reservar\_presente, name='reservar\_presente'),

]

**Views (views.py):**

# noivos/views.py

from django.shortcuts import render, redirect, HttpResponse

from .models import Presentes

from convidados.models import Convidados

def home(request):

if request.method == 'GET':

presentes = Presentes.objects.filter(reservado=False).order\_by('-importancia')

data = [Presentes.objects.filter(reservado=False).count(), Presentes.objects.filter(reservado=True).count()] # Dados para o gráfico

context = {'presentes': presentes, 'data': data}

return render(request, 'noivos/home.html', context)

elif request.method == 'POST':

# Lógica para salvar presentes

nome\_presente = request.POST.get('nome\_presente')

foto = request.FILES.get('foto')

preco = request.POST.get('preco')

importancia = int(request.POST.get('importancia'))

if importancia < 1 or importancia > 5:

return redirect('home') # Redireciona se a importância for inválida

presentes = Presentes(nome\_presente=nome\_presente, foto=foto, preco=preco, importancia=importancia)

presentes.save()

return redirect('home')

def lista\_convidados(request):

if request.method == "GET":

convidados\_list = Convidados.objects.all()

context = {'convidados': convidados\_list}

return render(request, "noivos/lista\_convidados.html", context)

elif request.method == "POST":

# ...(lógica para salvar convidados)

# convidados/views.py

from django.shortcuts import render, redirect, HttpResponse

from noivos.models import Presentes

from .models import Convidados

def convidados(request):

token = request.GET.get('token')

convidado = Convidados.objects.get(token=token)

presentes = Presentes.objects.filter(reservado=False).order\_by('-importancia')

if request.method == "GET":

context = {'convidado': convidado, 'presentes': presentes}

return render(request, 'convidados/convidados.html', context)

def responder\_presenca(request):

resposta = request.GET.get('resposta')

token = request.GET.get('token')

# Lógica para confirmar/recusar presença (atualiza o modelo Convidados)

# ...

return redirect(f'/convidados/?token={token}')

def reservar\_presente(request, id):

token = request.GET.get('token')

convidado = Convidados.objects.get(token=token)

presente = Presentes.objects.get(id=id)

presente.reservado = True

presente.reservado\_por = convidado

presente.save()

return redirect(f'/convidados/?token={token}')

**Models (models.py):**

# noivos/models.py

from django.db import models

class Presentes(models.Model):

nome\_presente = models.CharField(max\_length=100)

foto = models.ImageField(upload\_to='presentes')

preco = models.DecimalField(max\_digits=6, decimal\_places=2)

importancia = models.IntegerField()

reservado = models.BooleanField(default=False)

reservado\_por = models.ForeignKey('convidados.Convidados', on\_delete=models.SET\_NULL, null=True, blank=True)

def \_\_str\_\_(self):

return self.nome\_presente

# convidados/models.py

from django.db import models

import secrets

class Convidados(models.Model):

STATUS\_CHOICES = [

("AC", "Aguardando Confirmação"),

("C", "Presença Confirmada"),

("R", "Presença Recusada"),

]

nome\_convidado = models.CharField(max\_length=100)

whatsapp = models.CharField(max\_length=25, null=True, blank=True)

maximo\_acompanhantes = models.PositiveIntegerField()

token = models.CharField(max\_length=24, unique=True, null=True, blank=True, editable=False)

status = models.CharField(max\_length=2, choices=STATUS\_CHOICES, default="AC")

def \_\_str\_\_(self):

return self.nome\_convidado

def save(self, \*args, \*\*kwargs):

if not self.token:

self.token = secrets.token\_urlsafe(16)

super().save(\*args, \*\*kwargs)

@property

def link\_convite(self):

return f"http://127.0.0.1:8000/convidados/?token={self.token}"

**Templates (HTML):**

Os templates HTML foram omitidos por serem extensos, mas a estrutura básica envolve:

* **base.html:** Template base com a estrutura principal da página (barra de navegação, etc.).
* **noivos/home.html:** Página inicial dos noivos, listando os presentes.
* **noivos/lista\_convidados.html:** Página para gerenciamento de convidados.
* **convidados/convidados.html:** Página do convidado para confirmar presença e reservar presentes.
* **partials/navbar.html:** Barra de navegação, incluída nos outros templates.

**Funcionalidades e Melhorias:**

* **Herança de templates:** Utilização de um template base (base.html) para evitar repetição de código.
* **Arquivos parciais:** Uso de partials/navbar.html para a barra de navegação.
* **Integração com banco de dados:** Criação de models para Presentes e Convidados, salvamento e recuperação de dados do banco de dados SQLite.
* **Validação de dados:** Validação da importância do presente no backend.
* **URLs dinâmicas:** Utilização de parâmetros na URL (<int:id>) para identificar o presente a ser reservado.
* **Geração de token:** Geração de token único para cada convidado.
* **Gráfico dinâmico:** Integração com Chart.js para exibir o gráfico de presentes reservados/não reservados.
* **Lógica de reserva de presentes:** Implementação da lógica para reservar presentes, incluindo atualização do status do presente e associação ao convidado.
* **Restrição de acesso após confirmação/recusa:** Impede o convidado de mudar sua resposta após confirmá-la ou recusá-la.

Este documento fornece uma visão geral da aula, incluindo a estrutura do projeto, o código principal e a lógica das funcionalidades. Recomenda-se assistir à aula completa para detalhes adicionais e acompanhar a demonstração prática.

Transcrição e Organização da Aula 3 de Python e Django REST API (4d4p #03)

<https://www.youtube.com/watch?v=LHiLfFLSals>

Este documento transcreve e organiza o conteúdo da terceira aula da série "4 Days 4 Projects", sobre construção de uma REST API com Python, Django e Django Ninja. A aula inclui a apresentação da plataforma Pythonando, detalhes sobre o curso Python Full, a construção da API e respostas a perguntas do chat.

**Introdução e Apresentação da Plataforma Pythonando:**

A aula inicia com boas-vindas e a ênfase na importância da construção de APIs para desenvolvedores, independentemente da atuação (full-stack ou não). Em seguida, o instrutor apresenta a plataforma Pythonando, detalhando seus recursos:

* **Python Full (curso gravado):** +630 aulas, +140 horas, do básico ao avançado (System Design, REST API com Django Ninja, Deploy, Escalabilidade, Real Time, Jobs).
* **Agendamento de Reuniões:** Calls individuais semanais com o instrutor para tirar dúvidas e resolver problemas em código.
* **Análise de Código:** Envio de código para análise com feedback sobre pontos positivos e negativos, visando aprimoramento e boas práticas.
* **Exercícios de Lógica:** Exercícios com correção automática para treinar lógica de programação, essencial para entrevistas.
* **Ranking e Gamificação:** Sistema de pontos por atividades (exercícios, aulas, análises) com ranking trimestral e premiações.
* **Maratonas de Programação:** Competições entre alunos com exercícios aleatórios.
* **Certificados:** Certificado com reconhecimento MEC após 80% do curso e certificados por módulo.
* **Suporte:** Discord exclusivo para alunos com canais de dúvidas, materiais de apoio e interação.

**Bônus e Valor do Curso Python Full:**

São apresentados bônus para inscrições até o segundo dia do evento:

* Acesso vitalício ao curso.
* Camiseta personalizada (10 primeiros).
* Livro "Aprendendo a Programar em Python" + Kit de Estudos (20 primeiros).
* Módulo extra de currículo e LinkedIn.
* Cursos extras sobre IA e Machine Learning.
* Gravação do evento 4D4P + P-TECH Wiki.
* Plano de estudos personalizado no Notion.
* Acesso grátis por 60 dias ao StudFlow (plataforma de organização de estudos) - apenas para o primeiro dia.

O valor do curso é apresentado como R$1297 à vista ou 12x de R$129,45, com garantia de 7 dias e garantia condicional de 1 ano (reembolso se não conseguir a primeira vaga após um ano, cumprindo os requisitos do curso). As matrículas são abertas.

**Construindo a API:**

O projeto prático consiste em uma API para cadastro e avaliação de livros. As etapas de desenvolvimento são:

1. **Configuração do Ambiente:** Criação e ativação do ambiente virtual, instalação de Django e Django Ninja:

python3 -m venv .venv

source .venv/bin/activate

pip install django

pip install django-ninja

1. **Criação do Projeto e App:** Criação do projeto Django e do app livros:

django-admin startproject core .

python3 manage.py startapp livros

1. **Configuração de URLs:** Criação do arquivo core/urls.py e core/api.py para direcionar as requisições da API:

# core/urls.py

from django.contrib import admin

from django.urls import path, include

from api import api

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', api.urls),

]

# core/api.py

from ninja import NinjaAPI

from livros.api import livros\_router

api = NinjaAPI()

api.add\_router("/livros", livros\_router)

1. **Criação do Router (livros/api.py):** Define as rotas e views da API:

# livros/api.py

from ninja import Router

livros\_router = Router()

@livros\_router.get("/sortear", response={200: LivroSchema, 404: dict})

def sortear\_livro(request, filtros: FiltroSortearSchema = Query(...)):

# ... (lógica de sorteio de livro)

@livros\_router.post("/", response={200: dict})

def create\_livro(request, livro\_schema: LivroSchema):

# ... (lógica de criação de livro)

@livros\_router.put("/{livro\_id}", response={200: dict})

def avaliar\_livro(request, livro\_id: int, avaliacao\_schema: AvaliacaoSchema):

# ... (lógica de avaliação de livro)

@livros\_router.delete("/{livro\_id}", response={200: dict})

def deletar\_livro(request, livro\_id: int):

# ... (lógica de exclusão de livro)

1. **Criação dos Esquemas (esquemas.py):** Define a estrutura dos dados de entrada e saída da API utilizando Pydantic:

# esquemas.py

from ninja import Schema, ModelSchema

from livros.models import Livros, Categorias

class LivroSchema(ModelSchema):

class Config:

model = Livros

fields = ['nome', 'streaming', 'categorias']

class AvaliacaoSchema(ModelSchema):

class Config:

model = Livros

fields = ['nota', 'comentarios']

class FiltroSortearSchema(Schema):

nota\_minima: int = None

categorias: int = None

reler: bool = False

1. **Criação dos Modelos (livros/models.py):** Define as estruturas de dados do banco de dados:

# livros/models.py

from django.db import models

class Categorias(models.Model):

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

def \_\_str\_\_(self):

return self.nome

class Livros(models.Model):

STREAMING\_CHOICES = [

("AK", "Amazon Kindle"),

("F", "Físico"),

]

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

streaming = models.CharField(max\_length=2, choices=STREAMING\_CHOICES)

nota = models.IntegerField(null=True, blank=True)

comentarios = models.TextField(null=True, blank=True)

categorias = models.ManyToManyField(Categorias)

def \_\_str\_\_(self):

return self.nome

1. **Migrações e Superusuário:** Cria as tabelas no banco e um superusuário para acesso ao admin:

python3 manage.py makemigrations

python3 manage.py migrate

python3 manage.py createsuperuser

1. **Registro no Admin (livros/admin.py):** Registra os modelos no Django Admin para gerenciamento:

# livros/admin.py

from django.contrib import admin

from .models import Livros, Categorias

admin.site.register(Livros)

admin.site.register(Categorias)

A aula continua com a implementação detalhada das views, incluindo a lógica de criação, avaliação, exclusão e sorteio de livros, com exemplos de uso do Postman e da documentação interativa do Django Ninja. Durante a aula, o instrutor responde a perguntas do chat, esclarecendo dúvidas sobre os conceitos e tecnologias utilizadas.

Este documento fornece uma visão geral organizada da aula, incluindo exemplos de código e explicações dos conceitos-chave. Recomenda-se assistir à aula completa para obter detalhes adicionais e acompanhar a demonstração prática.

Transcrição e Organização da Aula 4 - REST API + Mobile com Django e Flet (4d4p #04)

REST API + MOBILE com Django e Flet - [https://www.youtube.com/watch?v=PXaBuOnUx74](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DPXaBuOnUx74)

Esta aula finaliza a série "4 Days 4 Projects", focando no consumo da API de livros construída na aula 3, utilizando a biblioteca Flet para criar uma interface mobile.

**Introdução e Recapitulação:**

Caio Sampaio recapitula os projetos das aulas anteriores: gerenciador de senhas (Python puro), aplicação web full-stack para casamento (Django) e a API de livros (Django/Ninja). Nesta aula, o objetivo é criar um frontend para consumir a API de livros, utilizando Flet para construir uma interface multiplataforma (web, desktop, mobile). Ele explica que, embora o foco seja uma simulação de app mobile, a mesma API poderia ser consumida por diferentes plataformas e linguagens.

**Configuração e Primeiros Passos com Flet:**

1. **Instalação das bibliotecas:**

pip install flet requests

1. **Iniciando a aplicação Flet:**

import flet as ft

def main(page: ft.Page):

page.title = "Cadastro App"

page.window\_width = 400 # Simulando tamanho mobile

page.on\_route\_change = route\_change

page.go("/")

def route\_change(event):

page.views.clear()

if page.route == "/":

home\_page()

elif page.route.startswith("/review"):

review\_page(livro\_id) #(Corrigido posteriormente)

page.update()

ft.app(target=main)

**Criando a Página Inicial (home\_page):**

#Dentro da função main() após a definição de route\_change

from connect import get\_livros

#... other imports

def home\_page():

nome\_input = ft.TextField(label="Nome do produto", text\_align=ft.TextAlign.LEFT)

streaming\_select = ft.Dropdown(

label="Selecione o streaming",

options=[

ft.dropdown.Option("AK", text="Amazon Kindle"),

ft.dropdown.Option("F", text="Físico"),

],

)

cadastrar\_btn = ft.ElevatedButton(text="Cadastrar", on\_click=cadastrar)

lista\_livros = ft.ListView() #inicialmente vazio

carregar\_livros() #carregando os livros ao abrir a página

page.views.append(

ft.View(

"/",

[

nome\_input,

streaming\_select,

cadastrar\_btn,

lista\_livros,

],

)

)

page.update()

def carregar\_livros():

lista\_livros.controls.clear() #limpa a lista de livros

for livro in get\_livros():

lista\_livros.controls.append(

ft.Container(

content=ft.Text(livro['nome']),

bgcolor=ft.colors.BLACK12,

padding=15,

alignment=ft.alignment.center,

border\_radius=10,

on\_click=lambda e, livro\_id=livro['id']: page.go(f"/review?id={livro\_id}")

)

)

page.update()

def cadastrar(e):

data = {

"nome": nome\_input.value,

"streaming": streaming\_select.value,

"categorias": [],

}

response = requests.post(

f"http://127.0.0.1:8000/api/livros/", json=data

)

#print(response.json())

carregar\_livros() #Atualiza a lista após cadastrar

**Criando a Página de Avaliação (review\_page):**

def review\_page(livro\_id):

print(livro\_id) #verificando se o ID correto está sendo passado

nota\_input = ft.TextField(label="Nota", value="0", width=100, text\_align=ft.TextAlign.LEFT)

comentario\_input = ft.TextField(label="Comentário", multiline=True, expand=True)

avaliar\_btn = ft.ElevatedButton(text="Avaliar", on\_click=avaliar)

voltar\_btn = ft.ElevatedButton(

text="Voltar",

on\_click=lambda e: page.go("/"),

)

page.views.append(

ft.View(

"/review",

[

nota\_input,

comentario\_input,

avaliar\_btn,

voltar\_btn,

],

)

)

page.update()

def avaliar(e):

data = {

"nota": int(nota\_input.value),

"comentarios": comentario\_input.value,

}

try:

response = requests.put(

f"http://127.0.0.1:8000/api/livros/{livro\_id}", json=data

)

if response.status\_code == 200:

page.snack\_bar = ft.SnackBar(ft.Text("Avaliação enviada com sucesso"))

page.snack\_bar.open = True

else:

page.snack\_bar = ft.SnackBar(ft.Text("Erro ao enviar avaliação"))

page.snack\_bar.open = True

except Exception as e:

page.snack\_bar = ft.SnackBar(ft.Text(f"Erro de conexão: {e}"))

page.snack\_bar.open = True

page.update()

**Conexão com a API (connect.py):**

import requests

def get\_livros():

response = requests.get("http://127.0.0.1:8000/api/livros/")

return response.json()

**Correção do Problema com lambda e on\_click:**

O problema com a função lambda capturando sempre o último valor do loop for é resolvido utilizando uma partial function ou adicionando um parâmetro padrão na função lambda para "congelar" o valor do ID do livro:

#Dentro de carregar\_livros()

on\_click=lambda e, livro\_id=livro['id']: page.go(f"/review?id={livro\_id}")

**Funcionalidades da Aplicação:**

* Cadastro de livros.
* Listagem de livros (consumindo a API).
* Avaliação de livros (enviando dados para a API).
* Navegação entre páginas (cadastro e avaliação).
* Tratamento de erros na conexão com a API.

**Melhorias em relação à transcrição original:**

* **Organização do código:** Separação da lógica de conexão com a API em um arquivo separado (connect.py).
* **Correção do bug da função lambda:** Implementada a solução com parâmetro padrão.
* **Tratamento de erros:** Incluído tratamento de erros na requisição à API.
* **Comentários:** Adicionados comentários para explicar o código.
* **Formatação:** Código formatado para melhor legibilidade.

**Observações:**

* A transcrição original omite grande parte do HTML/CSS e se concentra na lógica do backend.
* A aula também aborda a execução do app Flet em diferentes plataformas, incluindo a geração de QR code para teste em dispositivos móveis.

User

arrow\_upwardarrow\_downwardedit

more\_vert

Faça uma nota completa com links, tags, etc - descritiva - e até emojis para colocar no Obsidian

Model

arrow\_upwardarrow\_downwardedit

more\_vert

# 4 Days 4 Projects - Criando Aplicações com Python e Django 🚀

Este conjunto de notas abrange as quatro aulas práticas da série "4 Days 4 Projects", que ensina a construir aplicações com Python e Django, desde o básico até a criação de APIs e interfaces mobile.

## Aula 1: Gerenciador de Senhas com Python Puro 🔒

\*\*Objetivo:\*\* Criar um gerenciador de senhas com criptografia usando Python puro, sem frameworks.

\*\*Conceitos Abordados:\*\*

\* Lógica de programação

\* Orientação a objetos

\* Manipulação de arquivos

\* Criptografia com Fernet

\* Tratamento de Exceções

\* Padrão MVT (Model-View-Template) - adaptado para o contexto de aplicação em console

\*\*Código:\*\* [[Aula 1 - Gerenciador de Senhas - Código]]

\*\*Destaques:\*\*

\* Uso da biblioteca `cryptography` para criptografia robusta.

\* Implementação do padrão MVT para organizar o código.

\* Geração e armazenamento seguro de chaves de criptografia.

## Aula 2: Aplicação Web Full-Stack com Python e Django 🌐

\*\*Objetivo:\*\* Desenvolver uma aplicação web full-stack para gerenciar convidados e presentes de casamento.

\*\*Conceitos Abordados:\*\*

\* Django Models

\* Django Views

\* Django Templates

\* Django Admin

\* Roteamento de URLs

\* Integração com banco de dados (SQLite)

\* Upload de arquivos

\* Herança de templates

\* Formulários HTML

\* Integração com Chart.js

\*\*Código:\*\* [[Aula 2 - Aplicação Web Full-Stack - Código]]

\*\*Destaques:\*\*

\* Criação de um CRUD (Create, Read, Update, Delete) completo.

\* Uso de templates e herança para uma interface mais organizada.

\* Integração com Chart.js para visualização de dados.

## Aula 3: REST API com Python, Django e Django Ninja ⚙️

\*\*Objetivo:\*\* Construir uma REST API para cadastro e avaliação de livros.

\*\*Conceitos Abordados:\*\*

\* Django REST Framework (DRF) - Embora o foco seja Django Ninja

\* Serialização de dados

\* Django Ninja

\* Pydantic

\* Rotas e Views de API

\* Métodos HTTP (GET, POST, PUT, DELETE)

\* Testes de API com Postman

\* Relacionamentos entre modelos (ManyToManyField)

\*\*Código:\*\* [[Aula 3 - REST API - Código]]

\*\*Destaques:\*\*

\* Uso de Django Ninja para simplificar a criação da API.

\* Serialização de dados com Pydantic para garantir a consistência dos dados.

\* Implementação de diferentes métodos HTTP para diferentes operações.

## Aula 4: REST API + Mobile com Django e Flet 📱

\*\*Objetivo:\*\* Criar um frontend mobile com Flet para consumir a API de livros da aula 3.

\*\*Conceitos Abordados:\*\*

\* Flet

\* Consumo de APIs com Requests

\* Roteamento no Flet

\* Criação de interface com Flet

\* Tratamento de erros na comunicação com a API

\* Deploy multiplataforma com Flet

\*\*Código:\*\* [[Aula 4 - REST API + Mobile - Código]]

\*\*Destaques:\*\*

\* Uso de Flet para criar uma interface multiplataforma.

\* Consumo da API criada na aula anterior.

\* Demonstração de como testar e buildar aplicações