

Desenvolvimento de Sistemas Web
Área de concentração
TI-SOFTWARE
Competência(a) a ser(em) desenvolvidas
Criar um aplicativo web para avaliar a unidade de competência
Passos
(1) Contextualização e Mobilização Professores ministram disciplinas. As disciplinas geralmente fazem parte de um curso. Porém, temos algumas escolas que chamam disciplina de unidade de competência, as famosas UCs. Temos diversos tipos de cursos em uma escola: qualificação, aperfeiçoamento, técnico, superior entre outros. Nesse contexto vamos focar no curso técnico. O curso técnico em questão é o curso técnico em informática. Ele possui uma variedade de unidade de competência. Devido a isso, o software precisa de um SGBD para armazenar os dados.
(2) Atividade de Aprendizagem Desenvolver um projeto web com o “ <i>microframework flask</i> ” para cadastrar unidade de competência em um banco de dados.
(3) Organização da Atividade de Aprendizagem Etapa 1: preparar ambiente Etapa 2: relembrar fundamentos de ambiente virtual Etapa 3: explicar a utilização do protocolo HTTP Etapa 4: criar a aplicação (estrutura de pastas e design patterns) Etapa 5: preparando o flask Etapa 6: instalar pacotes Etapa 7: conectar com banco de dados Etapa 8: criar um models com migrations Etapa 9: criar rotas Etapa 10: criar rotas com parâmetros Etapa 11: criar rotas com métodos http Etapa 12: criar views Etapa 13: criar templates com html Etapa 14: criar templates com bootstrap Etapa 15: criar template com jinja 2 Etapa 16: persistir (inserir) dados no banco de dados

(4) Coordenação e Acompanhamento

A cada etapa deverá ser feita uma entrega e um feedback

Etapa 1: rodou um “alô mundo rota”

Etapa 2: resolveu problemas de ambientes virtuais?

Etapa 3: quais os principais métodos do protocolo HTTP? O que é REST? O que é RESTFull?

Etapa 4: o que significa programação em camadas MTV?

Etapa 5: rodou a aplicação “alô mundo rota”?

Etapa 6: verificar as versões instalados dos pacotes

Etapa 7: conseguiu conectar? Roda a aplicação? Rodou?

Etapa 8: criou o modelo ? deu um init? deu um migrate? deu um upgrade?

Etapa 9: mostre a rota básica funcionando?

Etapa 10: mostre a rota com parâmetro funcionando?

Etapa 11: teste com postman métodos http diferentes modificando também o tipo de HTTP aceito na rota

Etapa 12: você entendeu o que são views?

Etapa 13: testar a página .html

Etapa 14: testar a página .html com bootstrap

Etapa 15: testar a página .html com jinja2 (templa engine)

Etapa 16: abra o banco de dados e faça um select * from

na tabela em questão

(5) Análise e Avaliação da Atividade de Aprendizagem;

Verificar cada etapa de 1 a 16 com arguição oral

Realizar um questionário online para verificação de conhecimentos

Verificar o funcionamento do software com funcionalidades básicas de inserção no banco de dados.

(6) Outras Referências

<https://github.com/romulosilvestre/gabaritoflask>

(7) Síntese e Generalização

Utilizando o projeto base e modificado como exemplo, escolha um caso de uso do seu projeto integrador e faça o aplicativo web inserindo dados.

Recursos necessários:

Projeto base de exemplo

Diagrama de Caso de Uso

Documento Textual de Caso de Uso

Modelagem Conceitual

Modelo Lógico

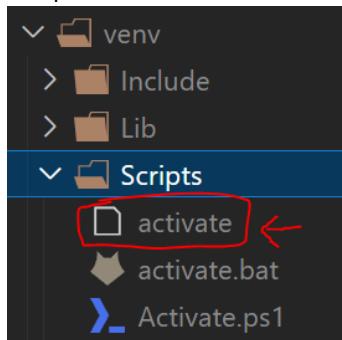
Diagrama de Classe

.

Etapa 1: preparar ambiente

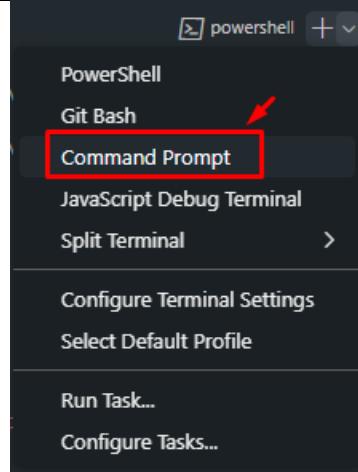
O que é um ambiente virtual?

No python o ambiente virtual é uma instalação que não conflita com as versões que estão instaladas no computador.



Habilidade:

A	Criar uma pasta na área de trabalho :
B	Abrir a pasta no Microsoft Visual Studio Code:
C	Instalar o ambiente virtual: <code>python -m venv venv</code>
D	Verificar a pasta criada:
E	Abrir a pasta venv :
F	Mudar o tipo de terminal para Command Prompt Verifique é necessário mudar de terminal

	 <p>Leia no canto inferior direito que tipo de interpretador está selecionado. Caso não seja a venv selecione na pasta quando necessário.</p>
G	<p>Ativar o ambiente virtual</p> <pre>C:\Users\BIBLIOTECA\Desktop\projectscore\venv>cd Scripts C:\Users\BIBLIOTECA\Desktop\projectscore\venv\Scripts>activate (venv) C:\Users\BIBLIOTECA\Desktop\projectscore\venv\Scripts></pre> <p>Volte para pasta original principal.</p> <pre>(venv) C:\Users\BIBLIOTECA\Desktop\projectscore\venv\Scripts>cd .. (venv) C:\Users\BIBLIOTECA\Desktop\projectscore\venv>cd .. (venv) C:\Users\BIBLIOTECA\Desktop\projectscore></pre>

Etapa 2: relembrar fundamentos de ambientes virtuais

Quais problemas têm enfrentado com ambientes virtuais?

O **objetivo desses ambientes** é que você possa trabalhar com versões diferentes na mesma máquina.

Qual versão você está utilizando?

Habilidade:

A	<pre>(venv) C:\Users\BIBLIOTECA\Desktop\projectscore>python --version Python 3.12.3</pre>	
---	--	--

Etapa 3: explicar a utilização do protocolo HTTP

O protocolo HTTP tem como base a comunicação entre máquinas. Uma máquina cliente faz requisições para uma outra máquina servidora. Quando a máquina cliente faz um pedido para a máquina servidora chamados de **request**. Agora quando o servidor da a resposta ao cliente chamamos de **response**.

A tabela abaixo mostra os métodos HTTP:

Método	Significado semântico
GET	“Pega” do servidor
POST	Incluir “alguma coisa” no servidor
PUT	Atualizar “alguma coisa” no servidor
DELETE	Apagar “alguma coisa” no servidor

Nessa comunicação entre cliente e servidor ocorre vários problemas de comunicação, para cada problema temos um código específico:

Método	Significado semântico
200	OK. Deu tudo certo.
302	Found. O recurso existe no servidor (típico de GET)
401	Unauthorized. Não realizou autenticação para acessar o recurso
404	Not Found. O recurso que você solicitou não existe no servidor.
500	Internal Server Error. O servidor encontrou um erro durante o processamento da requisição.

Conhecimentos:

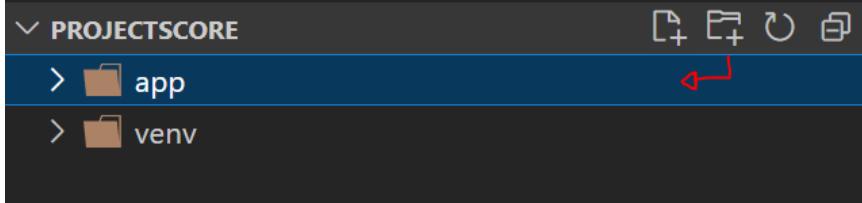
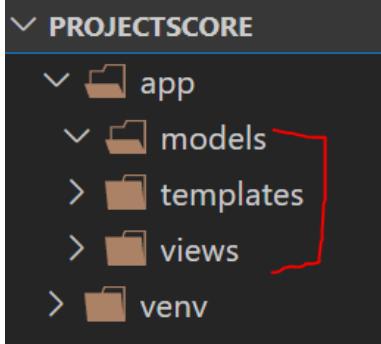
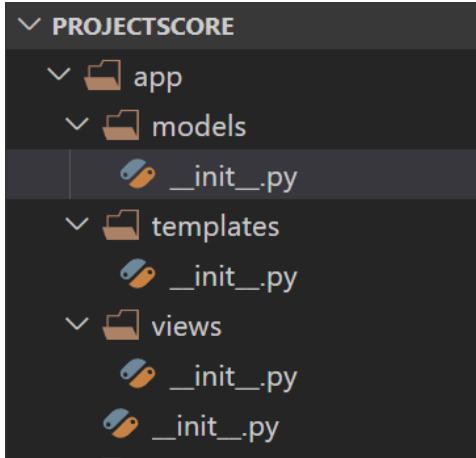
A	Os principais métodos HTTP são: GET, POST, PUT, DELETE
B	Existem diversos números que possuem significados semânticos no mundo do protocolo HTTP. Cada número representa uma mensagem do que ocorreu na comunicação entre cliente e servidor.
C	O padrão REST traz boas práticas de utilização do HTTP

D

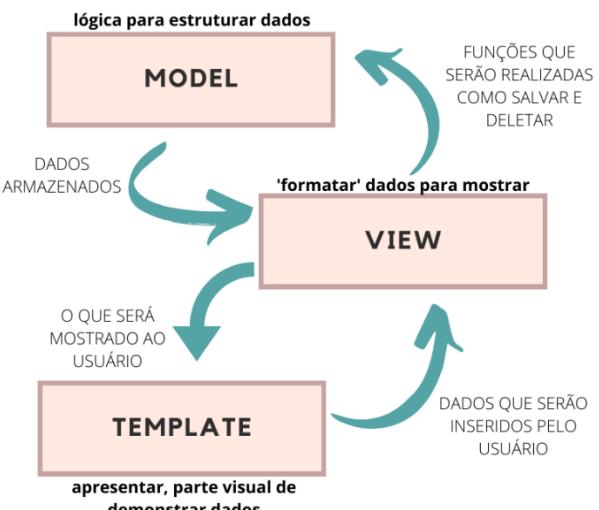
Quando uma API aplica o padrão REST, chamamos essa API de API RESTFull

Etapa 4: criar a aplicação (estrutura de pastas e design patterns)

Habilidades:

A	Criar a pasta app dentro da pasta projectscore
	
B	Dentro da pasta app criar mais três pastas:
	
C	Criar um arquivo de inicialização dentro da pasta app e dentro de cada pasta criada na pasta app.
	

Conhecimentos

A	Design Patterns são padrões de projetos
B	Padrões de projetos em programação é criar pastas para dividir a programação
C	Cada pasta fica responsável por uma camada
D	Nas linguagens usa-se pastas, pacotes, módulos, a nomenclatura varia, mas o objetivo é o mesmo “separar para organizar”
C	Nessa organização as aplicações web podem e devem ser escalonáveis, facilitando a manutenção no futuro
E	Ela no início aparenta ter mais trabalho, depois você consegue usufruir da organização.
F	No mundo web, o padrão mais conhecido é MVC
G	No python, o mais conhecido é MTV usado no framework Django
H	MVC (model -view – controller)
I	MTV (model – template – view)
J	Model – é o modelo dos dados
K	Template – é o html (front-end) da aplicação
L	View – trata as requisições da aplicação web (aplicando HTTP e rotas)
M	O design patterns tanto MVC como MTV são muito utilizados no mercado de trabalho devido sua aplicabilidade nos frameworks e softwares de grande escala (corporativo)
N	Analisa imagem do funcionamento de design pattern:
	 <pre> graph TD MODEL[MODEL] -- "DADOS ARMAZENADOS" --> VIEW[VIEW] MODEL -- "FUNÇÕES QUE SERÃO REALIZADAS COMO SALVAR E DELETAR" --> VIEW VIEW -- "O QUE SERÁ MOSTRADO AO USUÁRIO" --> TEMPLATE[TEMPLATE] TEMPLATE -- "DADOS QUE SERÃO INSERIDOS PELO USUÁRIO" --> VIEW </pre> <p>lógica para estruturar dados MODEL 'formatar' dados para mostrar VIEW apresentar, parte visual de demonstrar dados TEMPLATE</p>

Etapa 5: preparando o flask

O framework que iremos utilizar é o Flask. Ele é um microframework, devido sua simplicidade sua popularidade cresce a cada dia. É preciso entender que dentro do flask possui outras bibliotecas que o apoiam, cito: Werkzeug e a Jinja2.

A Werkzeug tem o objetivo de oferecer recursos para o programador utilizar o HTTP.

- Analisar cabeçalho HTTP;
- Facilitar o trabalho com requisições e respostas;
- Depurar interativamente;
- Ter suporte ao Unicode;
- Ter suporte a sessões e cookies;
- Ter um sistema de roteamento integrado.

Por outro lado, o Jinja2 é uma biblioteca que tem como objetivo trabalhar com template.

Com ela você pode:

- ✓ Ter herança de template
- ✓ Prevenção de XSS
- ✓ Sintaxe configurável
- ✓ Execução de código Python em páginas HTML

Conhecimentos

A	O que é um framework?
B	O que é um template engine?
C	O que é jinja2?
D	O que é Werkzeug?
E	Quais as vantagens de utilizar o Jinja2?

Habilidades

A	Instalar o Flask (com a venv ativada e posicionado no diretório principal): <code>pip install flask</code>
C	Gerar uma lista dos pacotes instalados juntamente com o flask:

PROJECTSCORE

- app
- models
- templates
- views
- venv

requirements.txt

```
1 blinker==1.7.0
2 click==8.1.7
3 colorama==0.4.6
4 Flask==3.0.3
5 itsdangerous==2.2.0
6 Jinja2==3.1.3
7 MarkupSafe==2.1.5
8 Werkzeug==3.0.2
9
```

TERMINAL

```
(venv) C:\Users\BIBLIOTECA\Desktop\projectscore>pip freeze > requirements.txt
(venv) C:\Users\BIBLIOTECA\Desktop\projectscore>[]
```

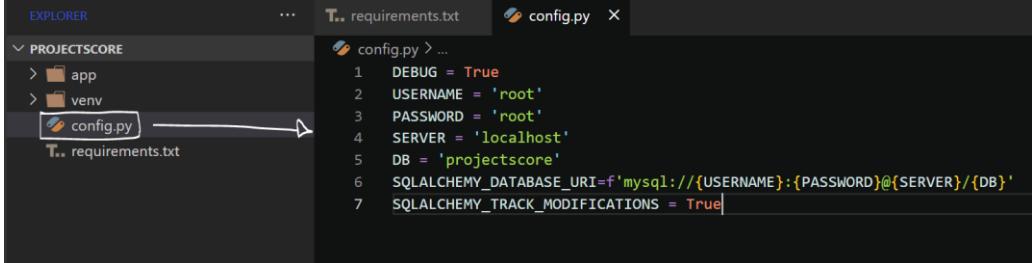
Etapa 6: instalar pacotes

Habilidades

A	Instalar o “mysql-connector-python 8.3.0” : pip install mysql-connector-python
B	Dê um pip freeze > requirements.txt <pre>1 blinker==1.7.0 2 click==8.1.7 3 colorama==0.4.6 4 Flask==3.0.3 5 itsdangerous==2.2.0 6 Jinja2==3.1.3 7 MarkupSafe==2.1.5 8 mysql-connector-python==8.3.0 ↗ 9 Werkzeug==3.0.2 10</pre>
C	Instalar: pip install flask_sqlalchemy
D	Verificar as instalações:

	<pre> 1 blinker==1.7.0 2 click==8.1.7 3 colorama==0.4.6 4 Flask==3.0.3 5 Flask-SQLAlchemy==3.1.1 ↪ 6 greenlet==3.0.3 7 itsdangerous==2.2.0 8 Jinja2==3.1.3 9 MarkupSafe==2.1.5 10 mysql-connector-python==8.3.0 11 SQLAlchemy==2.0.29 ↪ 12 typing_extensions==4.11.0 13 Werkzeug==3.0.2 </pre>
E	<code>pip install mysqlclient</code>

Etapa 7 e 8: criar models , conectar com banco e realizar migrations
Habilidades

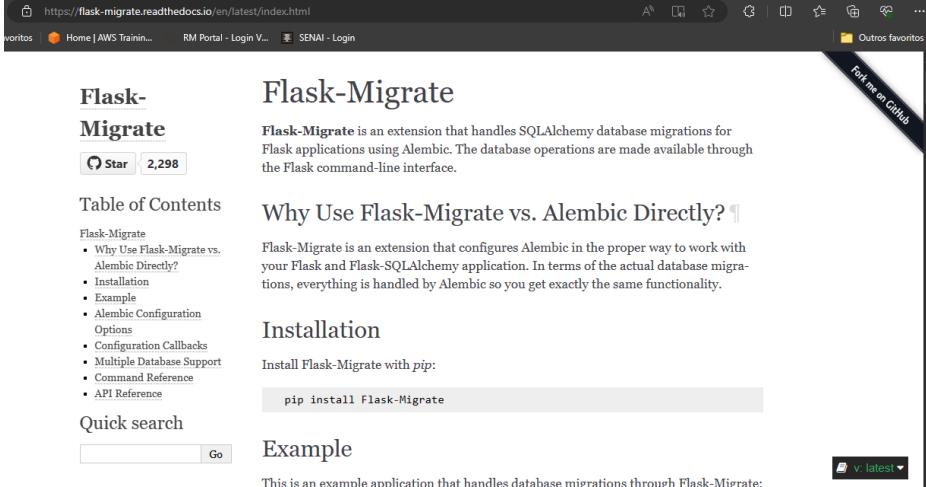
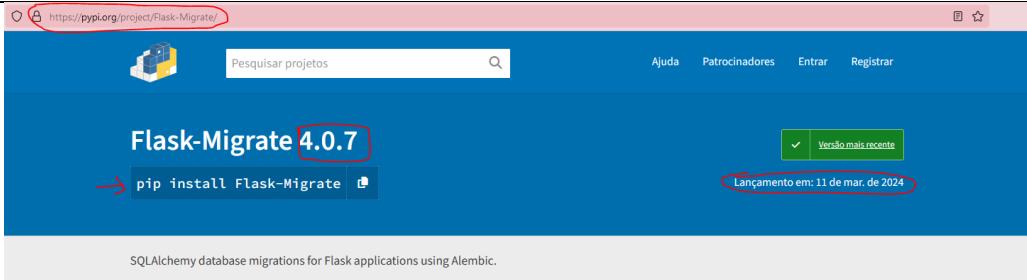
A	Criar um arquivo config.py no projeto raiz:
	 <pre> PROJECTSCORE app venv config.py requirements.txt </pre> <pre> config.py > ... 1 DEBUG = True 2 USERNAME = 'root' 3 PASSWORD = 'root' 4 SERVER = 'localhost' 5 DB = 'projectscore' 6 SQLALCHEMY_DATABASE_URI=f'{username}:{password}@{server}/{db}' 7 SQLALCHEMY_TRACK_MODIFICATIONS = True </pre>
B	Estudar, corrigir e analisar codificação: Lista de Codificação 01:

```

DEBUG = True
USERNAME = 'root'
PASSWORD = 'root'
SERVER = 'localhost'
DB = 'projectscore'
SQLALCHEMY_DATABASE_URI=f'{username}:{password}@{server}/{db}'
SQLALCHEMY_TRACK_MODIFICATIONS = True

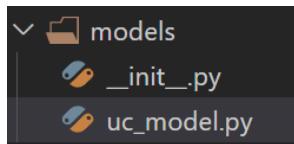
```

C	<pre> 1 blinker==1.7.0 2 click==8.1.7 3 colorama==0.4.6 4 Flask==3.0.3 5 <u>Flask-SQLAlchemy==3.1.1</u> 6 greenlet==3.0.3 7 itsdangerous==2.2.0 8 Jinja2==3.1.3 9 MarkupSafe==2.1.5 10 mysql-connector-python==8.3.0 11 <u>SQLAlchemy==2.0.29</u> 12 typing_extensions==4.11.0 13 Werkzeug==3.0.2 14 ... </pre>	<p>Verificar pacotes :</p> <p>Flask-SQLAlchemy: trabalhar com ORM e Flask</p> <p>mysql-connector-python : conector com banco</p> <p>SQLAlchemy: ORM escolhido</p>
D	O SQLAlchemy é uma biblioteca de mapeamento objeto-relacional (ORM) que fornece um conjunto completo de comandos para manipular bancos de dados utilizando o Python.	
E	Frameworks ORM tentam reduzir o tamanho do trabalho que temos ao nos conectar principalmente com bases de dados relacionais.	
F	<p>ORM é um acrônimo para object-relational mapping – mapeamento objeto-relacional.</p> <ul style="list-style-type: none"> • Cada classe acaba sendo interpretada como uma tabela; • Cada linha de uma tabela, bem como seus relacionamentos, é tratada como instância do objeto relacionado à tabela em questão. 	
G	<p>DEBUG = True</p> <p>DEBUG = False</p> <p>Faça um teste utilizando os dois e escolha qual o melhor para você</p> <pre> 1 DEBUG = True 2 3 USERNAME = 'root' 4 PASSWORD = 'root' 5 SERVER = 'localhost' 6 DB = 'projectscore' 7 8 SQLALCHEMY_DATABASE_URI=f'mysql://'{USERNAME}:{PASSWORD}@{SERVER}/{DB} 9 SQLALCHEMY_TRACK_MODIFICATIONS = True 10 11 12 </pre>	
H	O que são models?	
I	Como vimos anteriormente, iremos separar nosso projeto Flask em três camadas principais (Model-Template-View).	
J	A camada Model tem um papel fundamental nessa arquitetura.	

K	É ela quem se comunica, através do SQLAlchemy, com o banco de dados para criar os modelos da aplicação diretamente no banco de dados sem a necessidade de usar código SQL.
L	Precisamos do Migrations
	 <p>Assim como o GitHub faz o versionamento do código o Flask-Migrate faz o versionamento do banco.</p>
M	
N	<pre>> pip install flask-migrate</pre>

O	<pre> 1 alembic==1.13.1 2 blinker==1.7.0 3 click==8.1.7 4 colorama==0.4.6 5 Flask==3.0.3 6 Flask-Migrate==4.0.7 ↗ 7 Flask-SQLAlchemy==3.1.1 8 greenlet==3.0.3 9 itsdangerous==2.2.0 10 Jinja2==3.1.3 11 Mako==1.3.3 12 MarkupSafe==2.1.5 13 mysql-connector-python==8.3.0 14 mysqlclient==2.2.4 15 SQLAlchemy==2.0.29 16 typing_extensions==4.11.0 17 Werkzeug==3.0.2 </pre>
---	---

P Em models crie um arquivo uc_model.py:



Q	<pre> 1 from app import db 2 class UnidadeCompetencia(db.Model): 3 __tablename__ = "unidadecompetencia" 4 id = db.Column(db.Integer,primary_key=True,autoincrement=True) 5 numero = db.Column(db.Integer) 6 nome = db.Column(db.String(200)) 7 carga_horaria = db.Column(db.Integer) 8 competencia_geral= db.Column(db.String(255)) </pre>
---	--

Segue listagem da codificação:

```

from app import db

class UnidadeCompetencia(db.Model):

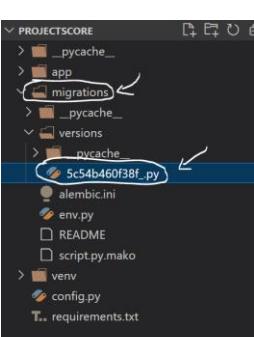
    __tablename__ = "unidadecompetencia"

    id = db.Column(db.Integer,primary_key=True,autoincrement=True)

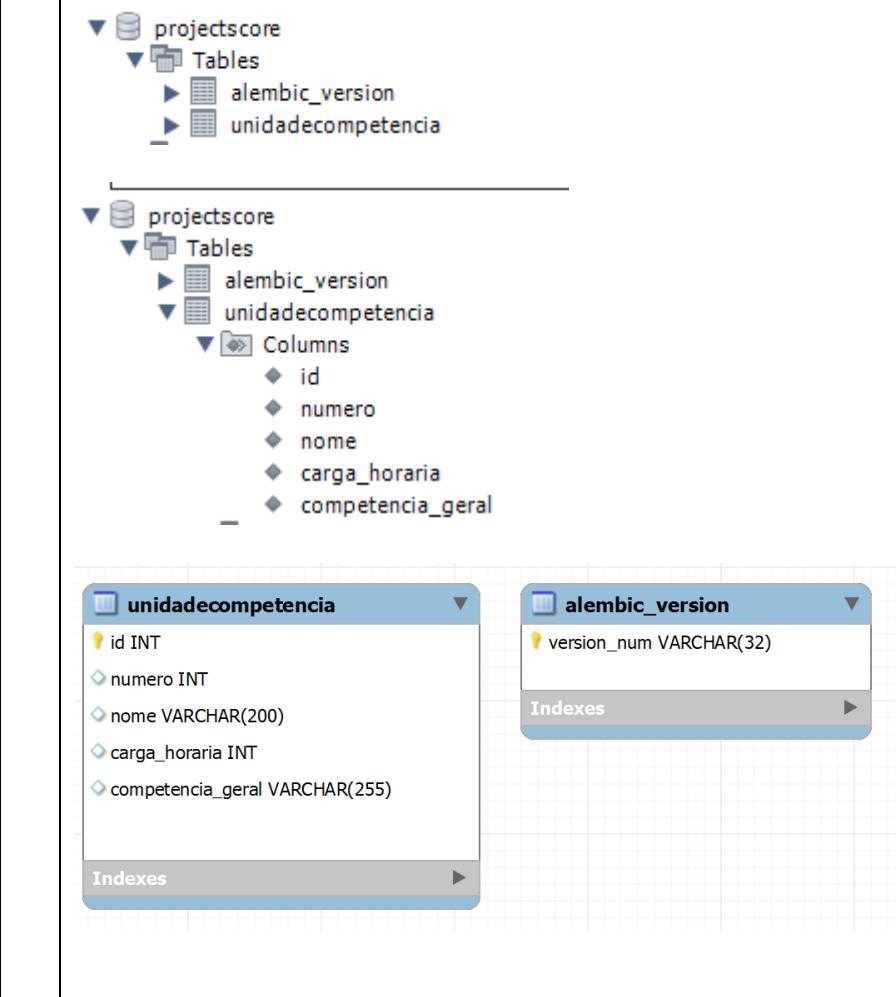
    numero = db.Column(db.Integer)

```

	<pre> nome = db.Column(db.String(200)) carga_horaria = db.Column(db.Integer) competencia_geral= db.Column(db.String(255)) </pre>
R	Programando o <code>__init__.py</code> do app
	<pre> 1 #importando o flask 2 from flask import Flask 3 #importando o SQLAlchemy 4 from flask_sqlalchemy import SQLAlchemy 5 from flask_migrate import Migrate,upgrade 6 #criando o aplicativo 7 app = Flask(__name__) 8 #puxando o arquivo config.py 9 app.config.from_object('config') 10 #criando um objeto db da classe SQLAlchemy 11 db = SQLAlchemy(app) 12 #criar uma variável migrate e passar a instância da aplicação e do db 13 migrate = Migrate(app,db) 14 15 #determinar o que vai ter no projeto 16 from .models import uc_model </pre>
S	Verifique a linha do Migrate (importante)
	<pre> #criar uma variável migrate e passar a instância da aplicação e do db migrate = Migrate(app,db) </pre>
R	<pre> (venv) C:\Users\BIBLIOTECA\Desktop\avaliapi>pip install flask-script Defaulting to user installation because normal site-packages is not writeable Collecting flask-script Downloading Flask-Script-2.0.6.tar.gz (43 kB) ━━━━━━━━━━━━━━━━━━━━━━━━ 43.1/43.1 kB 350.1 kB/s eta 0:00:00 Installing build dependencies ... done </pre>

S	<pre> 1 alembic==1.13.1 2 blinker==1.7.0 3 click==8.1.7 4 colorama==0.4.6 5 Flask==3.0.3 6 Flask-Migrate==4.0.7 7 Flask-Script==2.0.6 ↗ 8 Flask-SQLAlchemy==3.1.1 9 greenlet==3.0.3 10 itsdangerous==2.2.0 11 Jinja2==3.1.3 12 Mako==1.3.3 13 MarkupSafe==2.1.5 14 mysql-connector-python==8.3.0 15 mysqlclient==2.2.4 16 SQLAlchemy==2.0.29 17 typing_extensions==4.11.0 18 Werkzeug==3.0.2 19 .. </pre>
T	Crie um banco de dados conforme o config.py
	<pre> config.py > ... 1 DEBUG = True 2 USERNAME = 'root' 3 PASSWORD = 'root' 4 SERVER = 'localhost' 5 DB = 'projectscore' ↗ 6 SQLALCHEMY_DATABASE_URI=f'mysql://{{USERNAME}}:{{PASSWORD}}@{{SERVER}}/{{DB}}' 7 SQLALCHEMY_TRACK_MODIFICATIONS = True </pre> <p>1 • CREATE DATABASE projectscore;</p> <p>2</p> <p>3 </p>
U	Inicie o migrations
	<pre>python3 -m flask db init</pre>
V	<pre>python3 -m flask db migrate</pre>
X	Avaliando se os dois comandos anteriores deram certo:
	 <pre> PROJECTSCORE > __pycache__ > app > migrations ↗ > __pycache__ > versions > __pycache__ 5c54b460f38f.py ↗ alembic.ini env.py README script.py.mako venv config.py requirements.txt </pre> <pre> migrations > versions > 5c54b460f38f.py > ... 13 revision = '5c54b460f38f' 14 down_revision = None 15 branch_labels = None 16 depends_on = None 17 18 19 def upgrade(): ↗ 20 # ## commands auto generated by Alembic - please adjust! ## 21 op.create_table('unidadecompetencia', 22 sa.Column('id', sa.Integer(), autoincrement=True, nullable=False), 23 sa.Column('numero', sa.Integer(), nullable=True), 24 sa.Column('nome', sa.String(length=200), nullable=True), 25 sa.Column('carga_horaria', sa.Integer(), nullable=True), 26 sa.Column('competencia_geral', sa.String(length=255), nullable=True), 27 sa.PrimaryKeyConstraint('id') 28) 29 # ## end Alembic commands ## 30 </pre>

```
z | python3 -m flask db upgrade
```



Etapa 9 – Criar novas rotas

Maravilha, você chegou em um momento importante do treinamento. Vamos aprender hoje sobre rotas, a camada template e bootstrap.

Antes de iniciarmos vamos preparar o ambiente do nosso laboratório.

Crie uma pasta: **projetosflask**

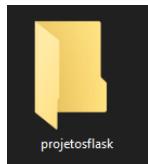
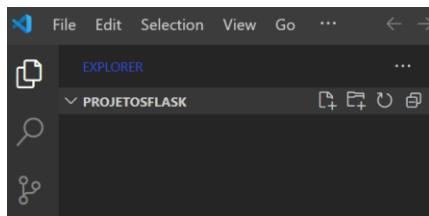


Figura 1 - entre no github do treinamento

Abra a pasta no Visual Studio Code.



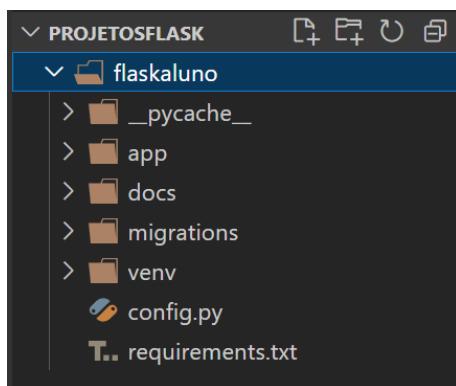
Abra o terminal e digite o seguinte código:

```
PS C:\Users\BIBLIOTECA\Desktop\projetosflask> git clone https://github.com/romulosilvestre/flaskaluno.git
Cloning into 'flaskaluno'... 3
remote: Enumerating objects: 3013, done. 1
remote: Counting objects: 100% (3013/3013), done.
remote: Compressing objects: 100% (2764/2764), done.
remote: Total 3013 (delta 208), reused 3013 (delta 208), pack-reused 0
Receiving objects: 100% (3013/3013), 23.89 MiB | 880.00 KiB/s, done.
Resolving deltas: 100% (208/208), done.
Updating files: 100% (2744/2744), done.
PS C:\Users\BIBLIOTECA\Desktop\projetosflask>
```

O git clone não mais é do que um download (baixar). O comando é simples:

- 1) É o comando
- 2) É o endereço do github do projeto base do aluno
- 3) Iniciou a clonagem do repositório público “flaskaluno”

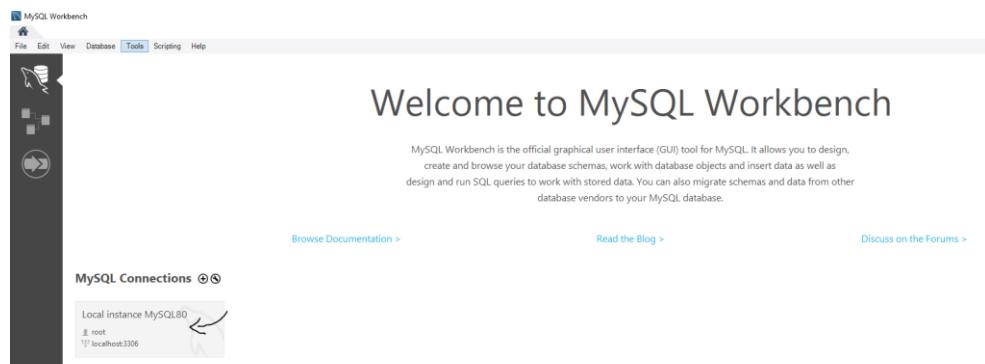
Visualize a estrutura de pastas:



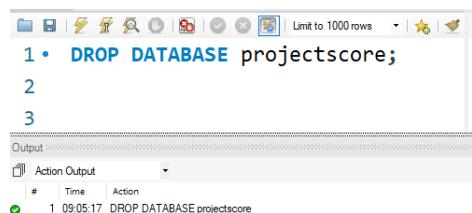
Abra o arquivo config.py e confira o nome do banco de dados:

```
1 DEBUG = True
2 USERNAME = 'root'
3 PASSWORD = 'root'
4 SERVER = 'localhost'
5 DB = 'projectscore' ↴
6 SQLALCHEMY_DATABASE_URI=f'mysql://{{USERNAME}}:{{PASSWORD}}@{{SERVER}}/{{DB}}'
7 SQLALCHEMY_TRACK_MODIFICATIONS = True|
```

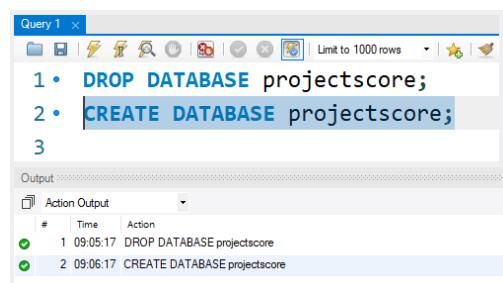
Abra o MySQL Workbench e clique em root:



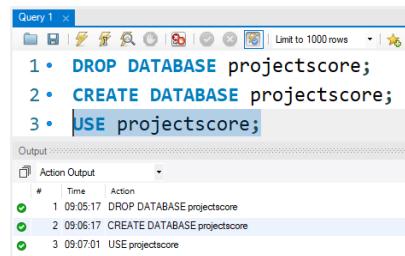
Dê um DROP DATABASE:



Dê um CREATE DATABASE:



Use o banco de dados criado:



```

3 • USE projectscore;
4 • show tables;
5

```

The screenshot shows a MySQL Workbench interface. A query window is open with the following SQL code:

```

3 • USE projectscore;
4 • show tables;
5

```

The results grid shows a single row: "Tables_in_projectscore". There is a question mark icon next to the result.

Figura 2 - Veja que não há nenhuma tabela criada. Maravilha! É isso mesmo!

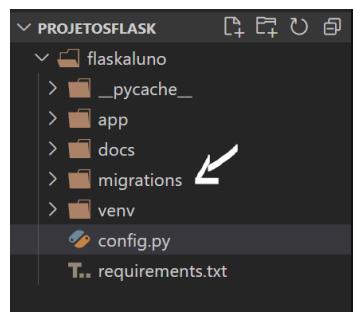


Figura 3- Veja que tem uma pasta chamada migrations (gerada nas etapas 7 e 8)

Ative a máquina virtual:

```

C:\Users\BIBLIOTECA\Desktop\projetosflask>cd flaskaluno

C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>cd venv

C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno\venv>cd Scripts

C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno\venv\Scripts>activate

(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno\venv\Scripts>█

```

Volte para a pasta flaskaluno (clonada)

```

(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno\venv\Scripts>cd ..

(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno\venv>cd ..

(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>█ ↵

```

```

(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>python -m flask db init
Error: Directory migrations already exists and is not empty

```

Figura 4 - Dê um db init

```
(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>python -m flask db migrate
INFO [alembic.runtime.migration] Context impl MySQLImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
ERROR [flask_migrate] Error: Target database is not up to date.
```

Figura 5 - Dê um db migrate

```
(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>python -m flask db upgrade
INFO [alembic.runtime.migration] Context impl MySQLImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.runtime.migration] Running upgrade  -> 5c54b460f38f, empty message
```

Figura 6 - Dê um db upgrade

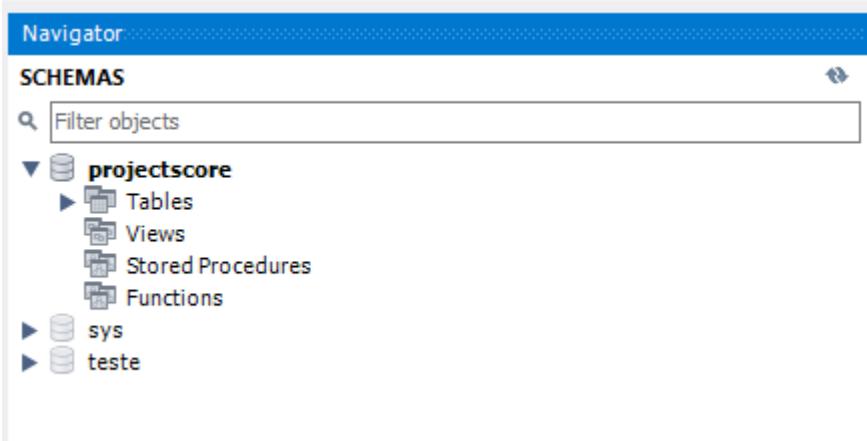
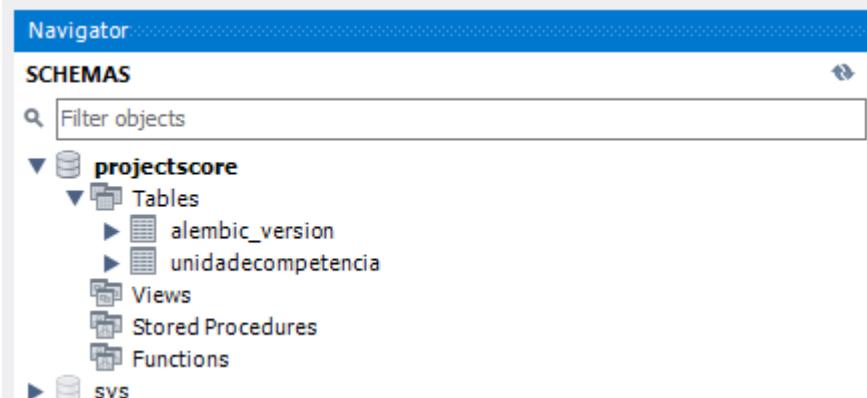
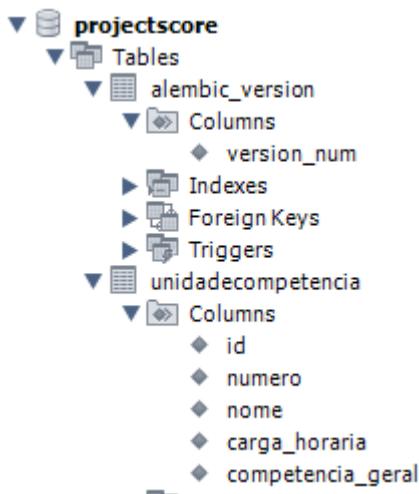


Figura 7 - Atualize o Navigator

Visualize as tabelas no Navigator:



Vai expandindo e verificando quais campos foram gerados em ambas as tabelas



A tabela **alembic_version** é a tabela de versão do Flask-Migrate.

Para visualizar como diagrama podemos fazer a engenharia reversa.

Aperte CTRL+R

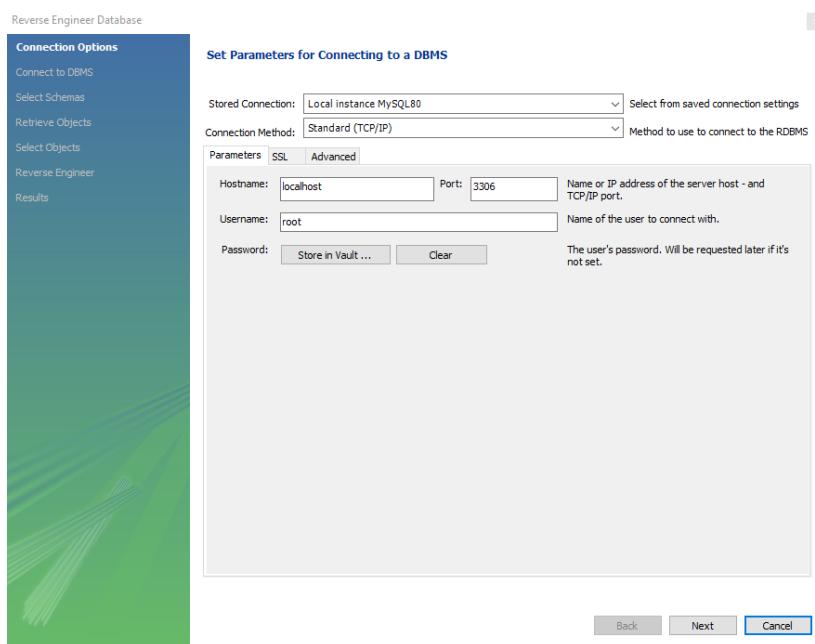


Figura 8 - Mantenha as configurações e clique em Next.

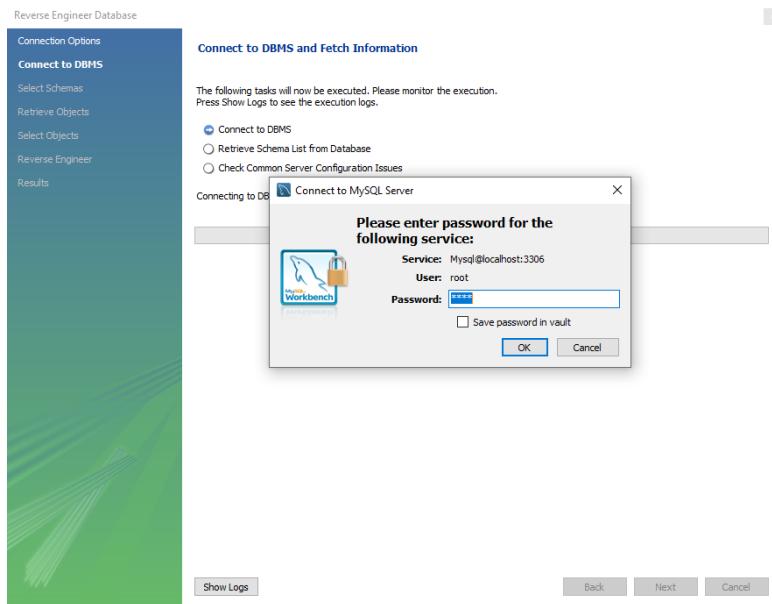


Figura 9 - Digite a senha

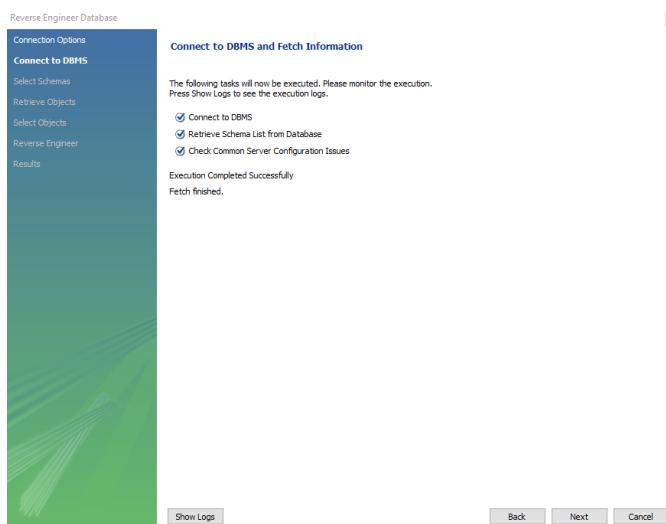


Figura 10 - Clique em Next

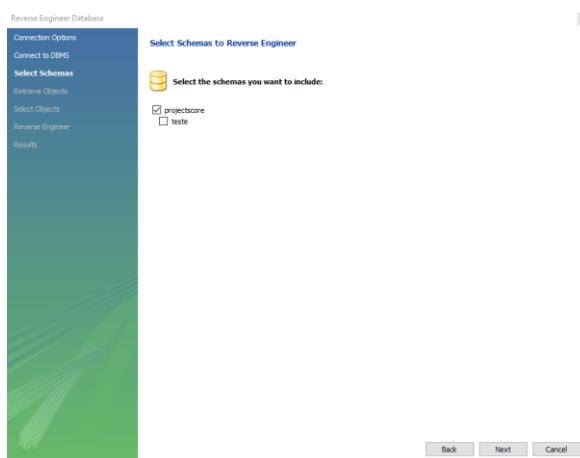
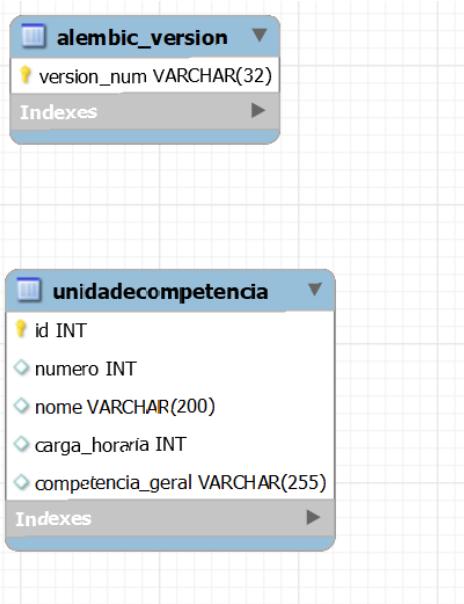


Figura 11 - Clique em next (projectscore)

Veja o resultado da engenharia reversa:



Encerramos a ambientação vamos partir para o nosso estudo de rotas.

Etapa 9: criar rotas

Etapa 10: criar rotas com parâmetros

Etapa 11: criar rotas com métodos http

Os passos a seguir terá um laboratório sobre rotas.

Ele vai abranger a criação de rotas simples, rotas com parâmetros e formas de personalizar e restringir rotas utilizando o protocolo HTTP.

Você deve abrir (caso o projeto seja clonado) ou criar (caso esteja criando passo a passo)

arquivo: uc_view.py

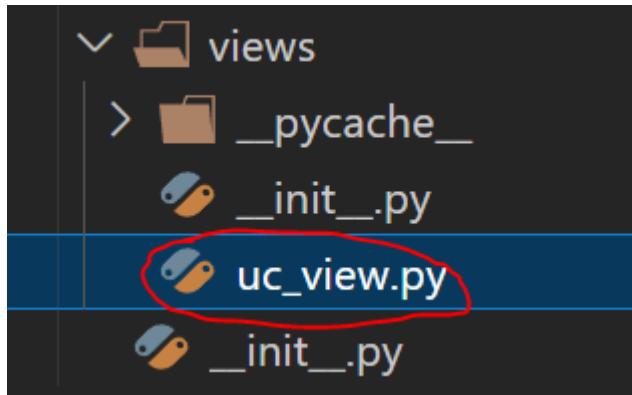
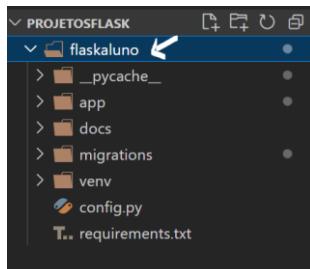


Figura 12 - Vamos criar uma tela de login. Para isso precisaremos de uma rota de login.

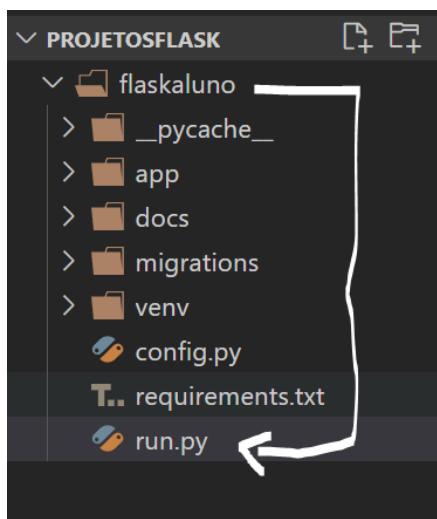
Vamos codar:

```
10  from app import app
11  from flask import render_template
12
13  @app.route("/")
14  def index():
15      return "página principal aqui vai ficar o login"
16
```

Agora vamos preparar para a execução:



Crie um arquivo dentro de flaskaluno com o nome run.py



Clique no arquivo e crie a seguinte codificação:

```
flaskaluno > run.py
  1  from app import app
  2
  3  if __name__ == "__main__":
→ 4      app.run()
  5
  6
```

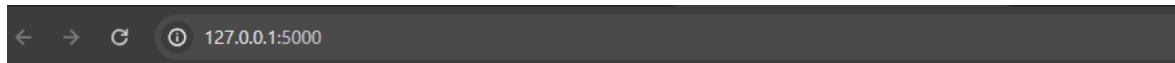
Faça a importação da uc_view

```
  9  app.config.from_object('config')
10  #criando um objeto db da classe SQLAlchemy
11  db = SQLAlchemy(app)
12  #criar uma variável migrate e passar a instância da app
13  migrate = Migrate(app,db)
14
15  #determinar o que vai ter no projeto
16  from .models import uc_model
17  from .views import uc_view
```

Clique em cima do arquivo run.py e execute:

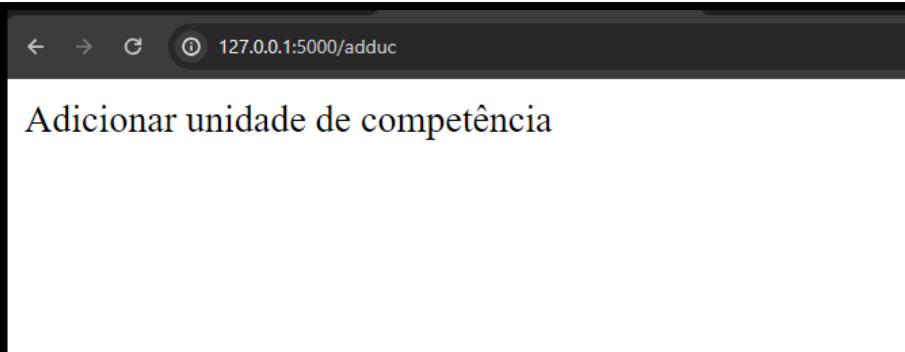
```
PS C:\Users\BIBLIOTECA\Desktop\projetosflask> & C:/Users/BIBLIO
rosoft/WindowsApps/python3.12.exe c:/Users/BIBLIOTECA/Desktop/p
o/run.py
  * Serving Flask app 'app'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a produ
a production WSGI server instead.
  * Running on http://127.0.0.1:5000
Press CTRL+C to quit
  * Restarting with watchdog (windowsapi)
  * Debugger is active!
  * Debugger PIN: 117-803-932
```

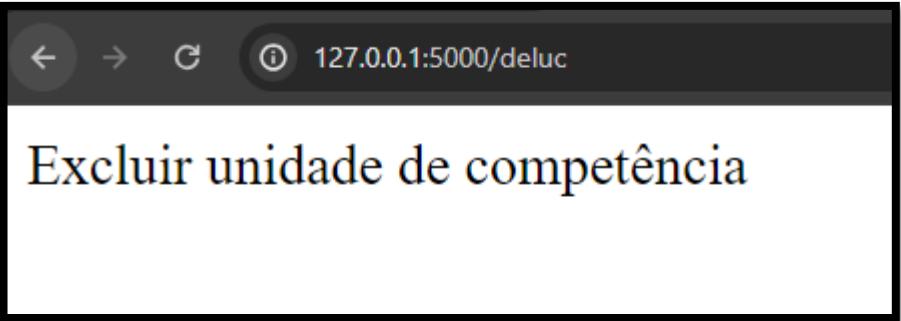
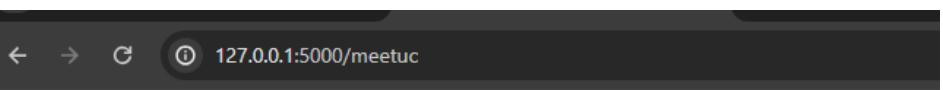
Veja agora o resultado:



página principal aqui vai ficar o login

Pronto! Fizemos uma ambientação, e nosso projeto está pronto para criação de novas rotas.
Abaixo a rota que iremos criar:

rota	descrição
/listuc	Listar unidades de competências  A screenshot of a web browser window. The address bar at the top shows the URL "127.0.0.1:5000/listuc". The main content area of the browser displays the text "Listar Unidades de Competências" in a large, dark font.
/adduc	Adicionar unidade de competência  A screenshot of a web browser window. The address bar at the top shows the URL "127.0.0.1:5000/adduc". The main content area of the browser displays the text "Adicionar unidade de competência" in a large, dark font.
/deluc	Deletar unidade de competência

	
/upuc	<p>Atualizar a unidade de competência</p> 
/meetuc	<p>Encontrar uma unidade de competência</p> 

habilidades

A	Inserir comentário de uma linha:
---	----------------------------------

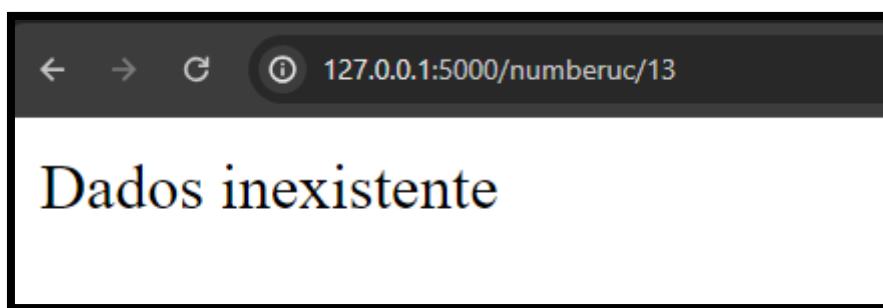
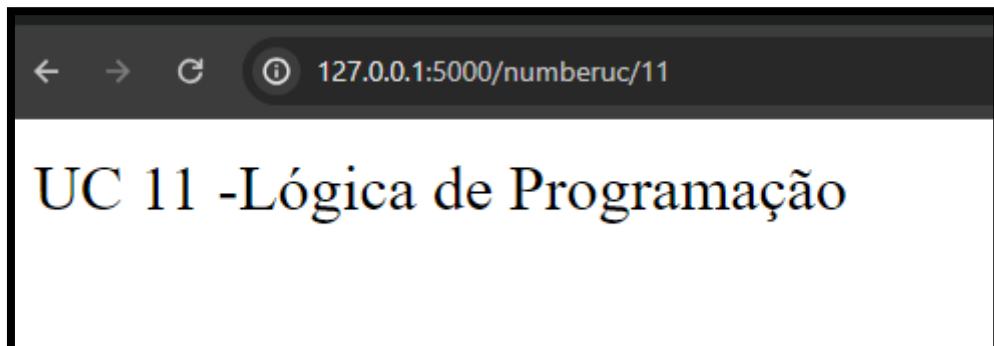
	<pre>#Nela vou criar os métodos que minha aplicação vai executar</pre>
B	<p>Inserir comentários de múltiplas linhas:</p> <pre>""" rota descrição /listuc Listar unidades de competências /adduc Adicionar unidade de competência /deluc Deletar unidade de competência /upuc Atualizar a unidade de competência /meetuc Encontrar uma unidade de competência """</pre>
C	<p>Realizar a importação do app:</p> <pre>from app import app</pre>
D	<p>Realizar a importação do render_template</p> <pre>from flask import render_template</pre>
E	<p>Criar uma rota para listar competências:</p> <pre>@app.route("/listuc") def listar_uc(): return render_template("uncompetencias/uc_template.html")</pre>
F	<p>Criar uma rota para adicionar uma competência:</p> <pre>@app.route("/adduc") def adicionar_uc(): return "Adicionar unidade de competência"</pre>
G	<p>Criar uma rota para apagar uma uc</p> <pre>@app.route("/deluc") def excluir_uc(): return "Excluir unidade de competência"</pre>

H	<p>Criar uma rota para atualizar uma uc</p> <pre>@app.route("/upuc") def atualizar_uc(): return "Atualizar unidade de competência"</pre>
I	<p>Definir duas rotas no mesmo método:</p> <pre>@app.route("/meetuc", defaults={'nome':None}, methods=['PUT']) @app.route("/meetuc/<string:nome>") def encontrar_uc(nome): if nome: return render_template("ucs/uc_temp.html", nome_uc=nome) else: return f"Navegue pelas UCs de forma personalizada"</pre>
J	<p>Resolver desafio lógico dentro de uma rota:</p> <pre>@app.route("/numberuc/<int:numero>") def number_uc(numero): match(numero): case 11: return f"UC {numero} -Lógica de Programação" case 12: return f"UC {numero} -Informática Básica" case _:return f" Dados inexistente"</pre>
K	<p>Observar o código e corrigir erros.</p> <p>Segue lista completa do código para analisar e corrigir erros:</p> <pre>#Nela vou criar os métodos que minha aplicação vai executar """ rota descrição /listuc Listar unidades de competências /adduc Adicionar unidade de competência /deluc Deletar unidade de competência</pre>

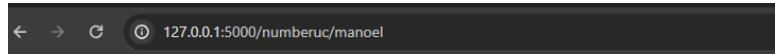
	<pre> /upuc Atualizar a unidade de competência /meetuc Encontrar uma unidade de competência """ from app import app from flask import render_template @app.route("/listuc") def listar_uc(): return render_template("uncompetencias/uc_template.html") @app.route("/adduc") def adicionar_uc(): return "Adicionar unidade de competência" @app.route("/deluc") def excluir_uc(): return "Excluir unidade de competência" @app.route("/upuc") def atualizar_uc(): return "Atualizar unidade de competência" @app.route("/meetuc", defaults={"nome":None}, methods=['PUT']) @app.route("/meetuc/<string:nome>") def encontrar_uc(nome): if nome: return render_template("ucs/uc_temp.html", nome_uc=nome) else: return f"Navegue pelas UCs de forma personalizada" @app.route("/numberuc/<int:numero>") def number_uc(numero): match(numero): case 11: return f"UC {numero} -Lógica de Programação" case 12: return f"UC {numero} -Informática Básica" case _: return f" Dados inexistente" </pre>
L	<p>Entender e explicar sobre o funcionamento de rotas com parâmetros criados:</p> <pre> @app.route("/numberuc/<int:numero>") def number_uc(numero): return "Encontrar unidade de competência" </pre>
M	Entender e explicar sobre o funcionamento da estrutura de seleção múltipla match:

```
@app.route("/numberuc/<int:numero>")  
def number_uc(numero):  
    return "Encontrar unidade de competência"
```

N Teste a rota `/numberuc` passando o valor 11, 12 e 13 para o parâmetro número:



Tente passar algo fora das condições do match:



The requested URL was not found on the server. If you &

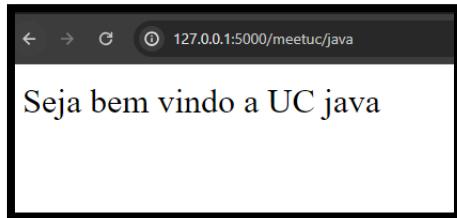
O Trabalhar com duas rotas no mesmo método:

Rota original não modificada:

```
@app.route("/meetuc")
def encontrar_uc():
    return "Encontrar unidade de competência"
```

Solução modificada:

```
@app.route("/meetuc", defaults={'nome':None})
@app.route("/meetuc/<string:nome>")
def encontrar_uc(nome):
    if nome:
        return f"Seja bem vindo a UC {nome}"
    else:
        return f"Navegue pelas UCs de forma personalizada"
```



P Entender os tipos de dados nas rotas:

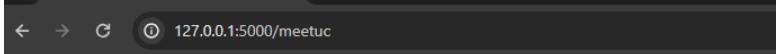
Cada parâmetro de uma rota deve possuir um tipo válido e, assim, evitar que uma rota aceita parâmetros com que o método da view não saiba trabalhar. Sendo assim, na tabela abaixo podemos ver todos os tipos de parâmetros que o Flask suporta:

- **string**: aceita qualquer texto sem barra (padrão);
- **int**: aceita valores positivos inteiros;
- **float**: aceita valores positivos ponto flutuantes;
- **path**: como string, mas aceita barra;
- **uuid**: aceita strings UUID.

Q

Você pode restringir rotas utilizando os métodos HTTP:

```
@app.route("/meetuc", defaults={'nome':None}, methods=['DELETE'])
@app.route("/meetuc/<string:nome>")
def encontrar_uc(nome):
    if nome:
        return f"Seja bem vindo a UC {nome}"
    else:
        return f"Navegue pelas UCs de forma personalizada"
```



Method Not Allowed

The method is not allowed for the requested URL.

Você vai ficar proibido de acessar a rota.

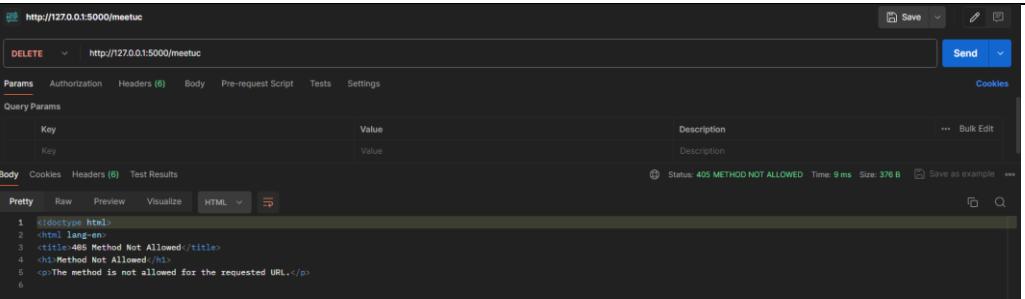
Mude agora para ele aceitar via GET e POST



```
@app.route("/meetuc", defaults={'nome':None}, methods=['GET', 'POST'])
@app.route("/meetuc/<string:nome>")
def encontrar_uc(nome):
    if nome:
        return f"Seja bem vindo a UC {nome}"
    else:
        return f"Navegue pelas UCs de forma personalizada"
```

R

Testar as rotas no postman:



http://127.0.0.1:5000/meetuc

DELETE http://127.0.0.1:5000/meetuc

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (6) Test Results

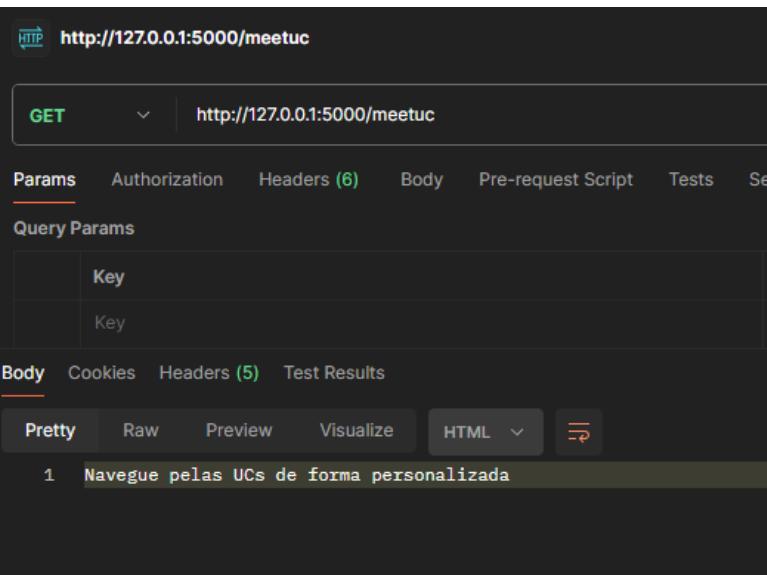
Pretty Raw Preview Visualize HTML

```
1 <!doctype html>
2 <html lang="en">
3 <title>405 Method Not Allowed</title>
4 <h1>Method Not Allowed</h1>
5 <p>The method is not allowed for the requested URL.</p>
```

Status: 405 METHOD NOT ALLOWED Time: 9 ms Size: 376 B Save as example

Teste as rotas no POSTMAN veja que deu o erro pois ele aceita só GET e POST

Se ele usar o get



HTTP http://127.0.0.1:5000/meetuc

GET http://127.0.0.1:5000/meetuc

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

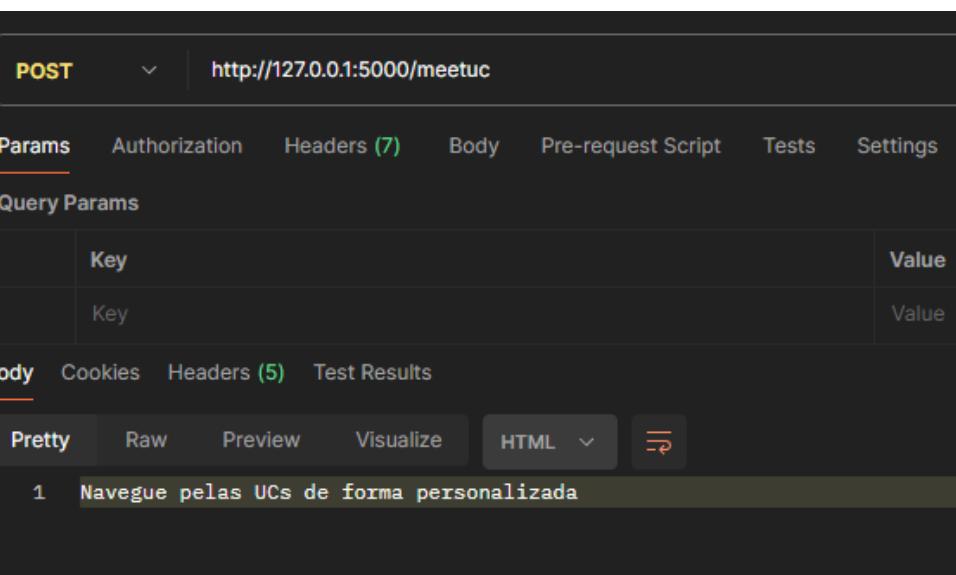
Query Params

Key
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize HTML

```
1 Navegue pelas UCs de forma personalizada
```



POST http://127.0.0.1:5000/meetuc

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize HTML

```
1 Navegue pelas UCs de forma personalizada
```

Mude para PUT

```
@app.route("/meetuc", defaults={'nome':None}, methods=['PUT'])
@app.route("/meetuc/<string:nome>")
def encontrar_uc(nome):
    if nome:
        return f"Seja bem vindo a UC {nome}"
    else:
        return f"Navegue pelas UCs de forma personalizada"
```

POST <http://127.0.0.1:5000/meetuc>

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1 <!doctype html>
2 <html lang=en>
3 <title>405 Method Not Allowed</title>
4 <h1>Method Not Allowed</h1>
5 <p>The method is not allowed for the requested URL.</p>
```

GET <http://127.0.0.1:5000/meetuc>

Params Authorization Headers (6) Body Pre-request Script Tests Settings

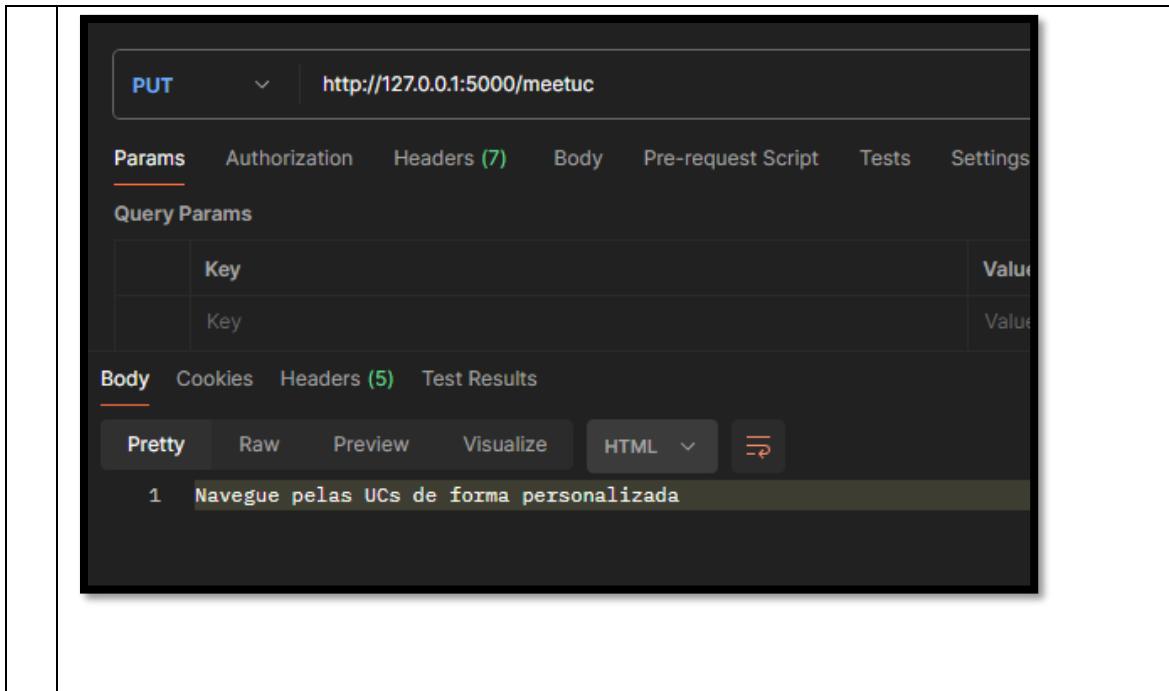
Query Params

Key	Value
Key	Value

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1 <!doctype html>
2 <html lang=en>
3 <title>405 Method Not Allowed</title>
4 <h1>Method Not Allowed</h1>
5 <p>The method is not allowed for the requested URL.</p>
```



Parabéns! você encerrou a etapa 9,10 e 11.

Agora vamos para as seguintes etapas:

Etapa 12: criar views

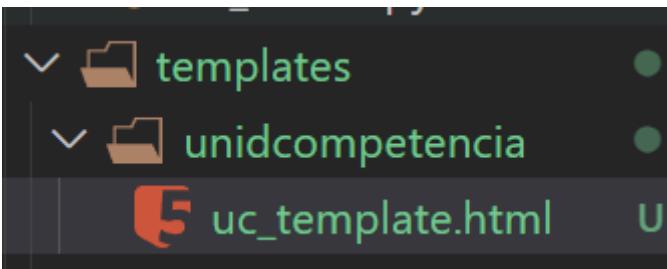
Conhecimentos:

A	Views é uma das camadas do design pattern mtv (padrão utilizado no Django)
B	A camada View é responsável por tramar as informações obtidas pelo model e exibi-las ao usuário (seja através de um template ou não).
C	É ela quem faz o “meio de campo” para que as informações sejam exibidas para quem está acessando a aplicação.
D	Uma view é composta por uma ou mais ações. Cada ação representa um recurso que a aplicação dispõe.

Etapa 13: criar templates com html

Conhecimentos:

A	Templates é uma camada do padrão mtv
B	Após todos os dados estiverem prontos para serem exibidos ao usuário, eles são repassados para o template.
C	Esta camada é responsável por exibir as informações para o usuário utilizando páginas HTML.
D	O Flask utiliza o Jinja2 como sistema de templates. Ele é bem completo e que provê diversas facilidades para a renderização das informações em páginas HTML.

E	<p>Views (trata as requisições e envia os dados para a camada Templates)</p>  <p>TEMPLATES (mostra os dados para os usuário e interage com o mesmo)</p>
F	<p>Observe que as informações que quero apresentar para o usuário no código anterior encontra-se dentro da VIEW:</p> <pre>@app.route("/meetuc", defaults={'nome':None}, methods=['PUT']) @app.route("/meetuc/<string:nome>") def encontrar_uc(nome): if nome: return f"Seja bem vindo a UC {nome}" ↴ else: return f"Navegue pelas UCs de forma personalizada" ↴</pre>
G	<p>Na camada templates, crie uma pasta (unidcompetencia) e um arquivo (uc_template.html):</p> 
H	<p>No arquivo html crie uma página html simples (use !) para gerar um código base html:</p> <pre>1 <!DOCTYPE html> 2 <html lang="en"> 3 <head> 4 <meta charset="UTF-8"> 5 <meta name="viewport" content="width=device-width, initial-scale=1.0"> 6 <title>Document</title> 7 </head> 8 <body> 9 10 </body> 11 </html></pre>

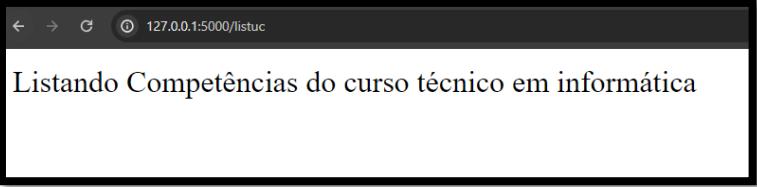
```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Unidade de Competência</title>
7  </head>
8  <body>
9
10 </body>
11 </html>

```

Figura 13 - mudar o idioma para pt-br

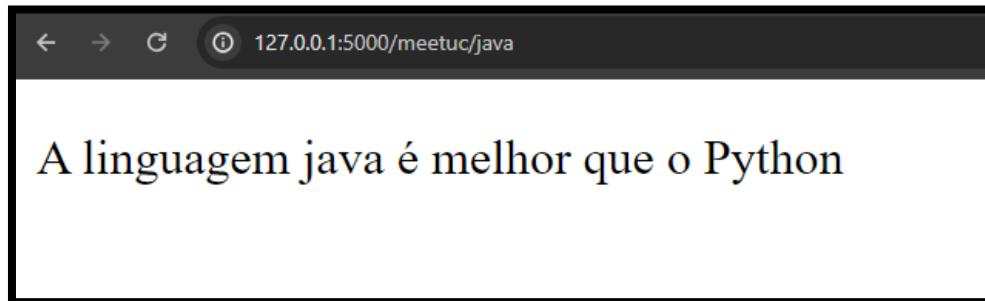
I	Vamos agora para a nossa view tratar a requisição para enviar dados para o template.
J	Vou escolher esse aqui:
	<pre> from app import app @app.route("/listuc") def listar_uc(): return "Listar Unidades de Competências" </pre>
K	Primeiro você tem que importar: from flask import render_template <pre> from app import app from flask import render_template </pre>
L	Coloque um texto na página HTML:
M	Na rota use a função render_template() passe a localização do arquivo (endereço relativo) para enviar os dados (nesse exemplo, retorna a própria página) para renderização na página html.

	<pre>from app import app from flask import render_template @app.route("/listuc") def listar_uc(): → return render_template("uncompetencias/uc_template.html")</pre>	
N		
	<p>Figura 14 - teste a rota enviando mensagens para a camada template</p>	
O	<pre>app > templates > uncompetencias > uc_template.html > html 2 <html lang="pt-br"> 3 <head> 6 <title>Unidade de Competência</title> 7 </head> 8 <body> 9 Listando Competências do curso técnico em informática 10 </body> 11 </html></pre> <pre>@app.route("/listuc") def listar_uc(): return render_template("uncompetencias/uc_template.html")</pre>	
P	<p>Inspecione os códigos no seu navegador preferido (os desenvolvedores web e front-end, gostam muito do Firefox)</p>	
Q	<p>Mas com certeza não usaremos as rotas apenas para chamar a página e não trafegar dados. Precisamos aprender a passar os dados, isso é muito importante e essencial para o nosso trabalho:</p> <p>Vamos escolher o seguinte método para fazer o experimento:</p> <pre>@app.route("/meetuc", defaults={'nome':None}, methods=['PUT']) @app.route("/meetuc/<string:nome>") def encontrar_uc(nome): if nome: return f"Seja bem vindo a UC {nome}" else: return f"Navegue pelas UCs de forma personalizada"</pre> <p>Veja a modificação que você vai ter que fazer:</p>	

```

@app.route("/meetuc", defaults={'nome':None}, methods=['PUT'])
@app.route("/meetuc/<string:nome>")
def encontrar_uc(nome):
    if nome:
        return render_template("uncompetencias/uc_template.html", nome_uc=nome)
    else:
        return f"Navegue pelas UCs de forma personalizada"

```



```

@app.route("/meetuc/<string:nome>")
def encontrar_uc(nome):
    if nome:
        return render_template("ucs/uc_temp.html", nome_uc=nome)
    else:
        return f"Navegue pelas UCs de forma personalizada"

```

Agora vamos avançar para as próximas etapas.

Etapa 14: criar templates com bootstrap

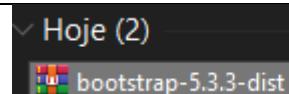
Etapa 15: criar template com jinja 2

Etapa 16: persistir (inserir) dados no banco de dados

Etapa 14: criar templates com bootstrap

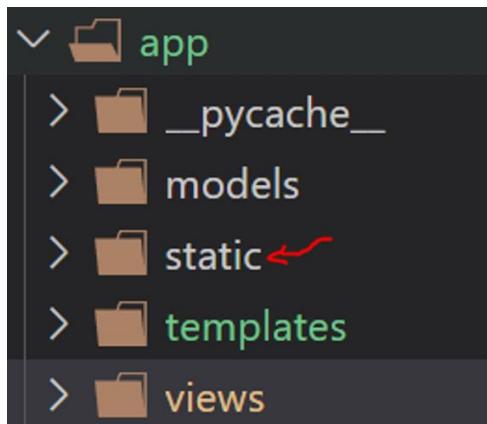
Habilidades:

A	Framework CSS com fontes, estilos e grids personalizados e é muito utilizado no mercado.
B	Primeiros passos baixar os arquivos download
C	<h3>Compiled CSS and JS</h3> <p>Download ready-to-use compiled code for Bootstrap v5.3.3 to easily drop into your project, which includes:</p> <ul style="list-style-type: none"> Compiled and minified CSS bundles (see CSS files comparison) Compiled and minified JavaScript plugins (see JS files comparison) <p>This doesn't include documentation, source files, or any optional JavaScript dependencies like Popper.</p> <p>Download</p>
D	Descompacte o arquivo baixado:

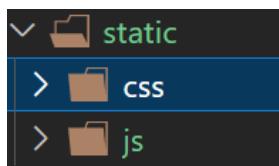


Nela temos duas pastas (css e js)

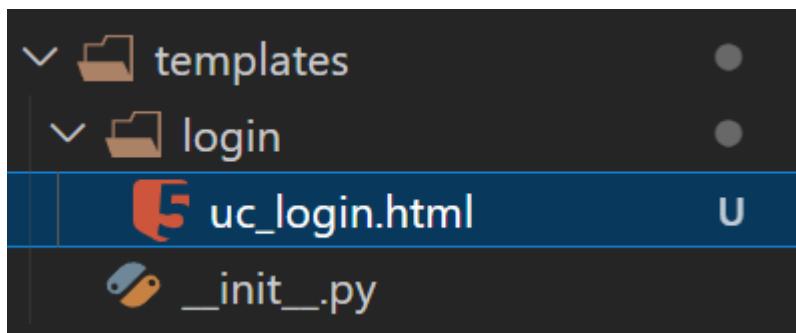
E Agora no seu Visual Studio Code crie uma pasta static:



F Coloque as duas pastas do bootstrap dentro da pasta static:



G Criar um arquivo login_uc.html



Gerar o html:

```

1  <!DOCTYPE html>
2  <html lang="pt-br"> ↵
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Login - ProjectScorte</title> ↵
7  </head>
8  <body>
9
10 </body>
11 </html>

```

Figura 15 - mude o idioma e o title

H Crie um comentário html

```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Login - ProjectScorte</title>
7      <!--comentário html-->
8  </head>
9  <body>
10 </body>
11 </html>

```

I

```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Login - ProjectScorte</title>
7      <!--comentário html-->
8      <!--insira o bootstrap aqui!!-->
9  </head>
10 <body>
11
12 </body>
13 </html>

```

J Faça a importação:

```
<link rel="stylesheet" href="{{url_for('static',filename='css/bootstrap.css')}}">
```

K

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Login - ProjectScorte</title>
7      <!--comentário html-->
8      <!--insira o bootstrap aqui!-->
9      <link rel="stylesheet" href="{{url_for('static',filename='css/bootstrap.css')}}">
10 </head>
11 <body>
12
13 </body>
14 </html>
```



L Versão: 5.3.3

M Grid:

```
11 <body>
12
13     <!--O grid do bootstrap é uma tabela com 12 colunas-->
14     <!--Você vai personalizando a quantidade de linhas-->
15     <!--Você vai personalizando a quantidade de colunas que os elementos vão ocupar-->
16
17     <div class="container"> ┌─────────┐
18         <div class="row">   ┌─────────┐
19             <div class="col">   | Column
20                 Column    |   ┌─────────┐
21             </div>       |   | Column
22             <div class="col">   |   |   ┌─────────┐
23                 Column    |   |   | Column
24             </div>       |   |   |   ┌─────────┐
25                 Column    |   |   |   | Column
26             </div>       |   |   |   |   ┌─────────┐
27                 Column    |   |   |   |   | Column
28             </div>       |   |   |   |   |   ┌─────────┐
29         </div>       |   |   |   |   |   | Column
30     </div>       |   |   |   |   |   |   ┌─────────┐
31     </body>       |   |   |   |   |   |   | Column
```

N Coloque o texto **célula**

```
<div class="container">
    <div class="row">
        <div class="col">
            célula 1
        </div>
        <div class="col">
            célula 2
        </div>
        <div class="col">
            célula 3
        </div>
    </div>
</div>
```

O Tínhamos criado uma rota para login lembra?

`<div class="row">
 <button type="button" class="btn btn-primary">Primary</button>
</div>`

O botão pegou toda a área.

Você pode configurar dizendo que ele vai pegar :

12/3 = 4 (4 colunas)

Nesse caso meu objetivo é que o botão acompanhe a célula 1 da linha anterior.

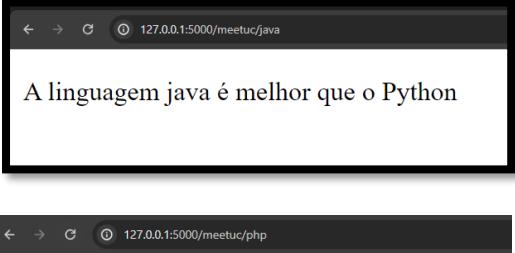
`<div class="row">
 <button type="button" class="btn btn-primary col-4">Primary</button>
</div>`

Etapa 15: criar template com jinja 2

Conhecimentos

A	Jinja2 é um template engine. Essas tecnologias permitem misturar HTML com uma linguagem de back end.
B	No caso o Jinja2 permite misturar código Python com HTML

Habilidades

A	<p><code>{{ }}</code> - usado para embutir valores de variáveis python.</p> <pre>2 <html lang="pt-br"> 3 <head> 4 <title>Unidade de Competência</title> 5 </head> 6 <body> 7 8 9 10 <p>A linguagem {{nome_uc}} é melhor que o Python</p> 11 12 13 </body> 14 </html></pre>
B	<p><code>{% %}</code> – usado para embutir codificação python – possui uma sintaxe especial</p> <pre>8 <body> 9 {% if nome_uc == "java" %} 10 <p>A linguagem {{nome_uc}} é melhor que o Python</p> 11 {% else %} 12 <p>A linguagem {{nome_uc}} deve ser igual a Python</p> 13 {% endif %}</pre>
C	<p>Resultado:</p>  <p>127.0.0.1:5000/meetuc/java</p> <p>A linguagem java é melhor que o Python</p> <p>127.0.0.1:5000/meetuc/php</p> <p>A linguagem php deve ser igual a Python</p>

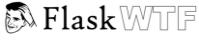
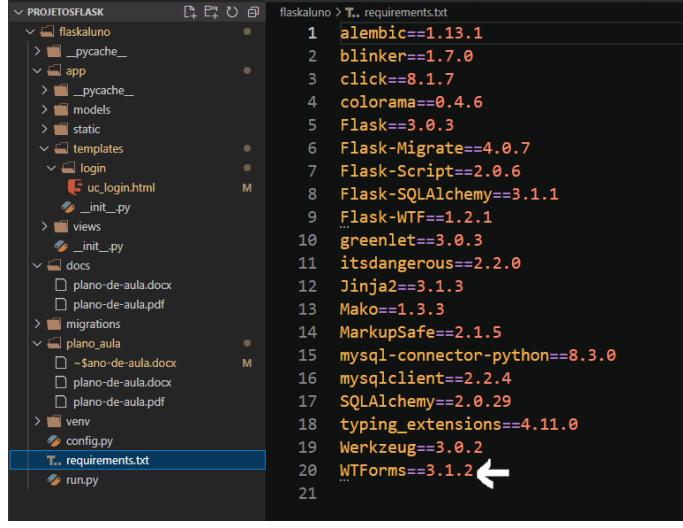
Etapa 16: persistir (inserir) dados no banco de dados

Para inserir em um banco de dados temos que realizar algumas validações de formulários.

Conhecimentos

A	A definição de formulário no Flask é uma camada extra na qual definimos regras de validação para os nossos models.
B	Essa camada extra, certifica que todos os dados, antes de serem inseridos no BD, serão validados conforme regras que o programador determinou seguindo o diagrama de caso e uso e documento textual de caso de uso, bem como outras documentações afins.
C	A validação, ocorre, portanto, no front-end e no back-end.

Habilidades

A	<pre>pip install Flask-WTF</pre> <p>Flask-WTF é uma ponte do WTForms e o Flask</p>
B	 
C	<pre>1 alembic==1.13.1 2 blinker==1.7.0 3 click==8.1.7 4 colorama==0.4.6 5 Flask==3.0.3 6 Flask-Migrate==4.0.7 7 Flask-Script==2.0.6 8 Flask-SQLAlchemy==3.1.1 9 Flask-WTF==1.2.1 ← 10 greenlet==3.0.3 11 itsdangerous==2.2.0 12 Jinja2==3.1.3 13 Mako==1.3.3 14 MarkupSafe==2.1.5 15 mysql-connector-python==8.3.0 16 mysqlclient==2.2.4 17 SQLAlchemy==2.0.29 18 typing_extensions==4.11.0 19 Werkzeug==3.0.2 20 WTForms==3.1.2 ← 21</pre>
D	Após o clone verifique o arquivo requirements.txt  <p>The screenshot shows a file explorer window with the following structure:</p> <ul style="list-style-type: none">PROJETOSFLASKflaskalunoapptemplatesdocsmigrationsplano.aulavenvrequirements.txtrun.py <p>The requirements.txt file contains the following dependencies:</p> <pre>1 alembic==1.13.1 2 blinker==1.7.0 3 click==8.1.7 4 colorama==0.4.6 5 Flask==3.0.3 6 Flask-Migrate==4.0.7 7 Flask-Script==2.0.6 8 Flask-SQLAlchemy==3.1.1 9 Flask-WTF==1.2.1 ← 10 greenlet==3.0.3 11 itsdangerous==2.2.0 12 Jinja2==3.1.3 13 Mako==1.3.3 14 MarkupSafe==2.1.5 15 mysql-connector-python==8.3.0 16 mysqlclient==2.2.4 17 SQLAlchemy==2.0.29 18 typing_extensions==4.11.0 19 Werkzeug==3.0.2 20 WTForms==3.1.2 ← 21</pre>

```
1 alembic==1.13.1
2 blinker==1.7.0
3 click==8.1.7
4 colorama==0.4.6
5 Flask==3.0.3
6 Flask-Migrate==4.0.7
7 Flask-Script==2.0.6
8 Flask-SQLAlchemy==3.1.1
9 Flask-WTF==1.2.1 ←
10 greenlet==3.0.3
11 itsdangerous==2.2.0
12 Jinja2==3.1.3
13 Mako==1.3.3
14 MarkupSafe==2.1.5
15 mysql-connector-python==8.3.0
16 mysqlclient==2.2.4
17 SQLAlchemy==2.0.29
18 typing_extensions==4.11.0
19 Werkzeug==3.0.2
20 WTForms==3.1.2
21
```

Se ambos estiverem instalados.

Repita os passos de ambientação do Migrate, caso ainda não tenha feito.

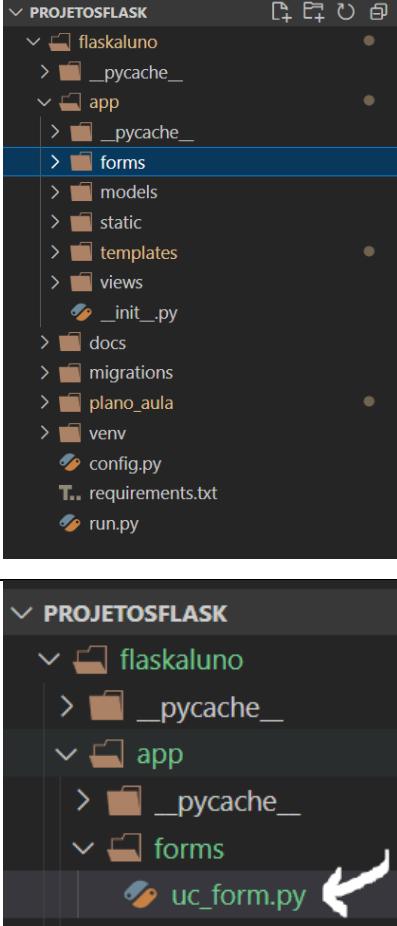
Após o init:

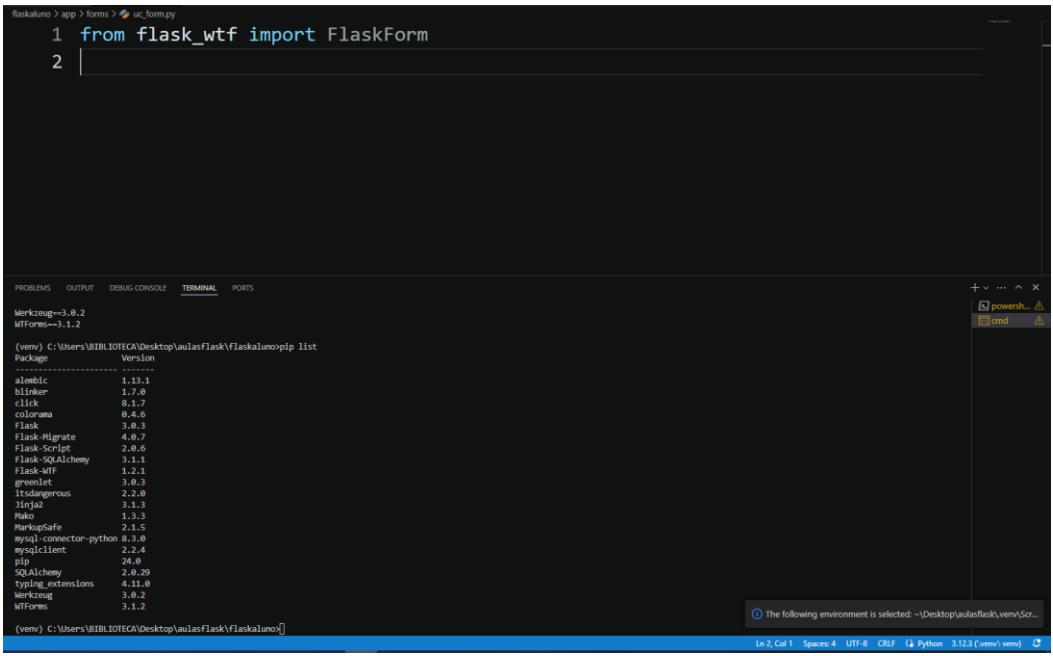
```
(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>python3 -m flask db migrate
INFO [alembic.runtime.migration] Context impl MySQLImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
ERROR [flask_migrate] Error: Target database is not up to date.

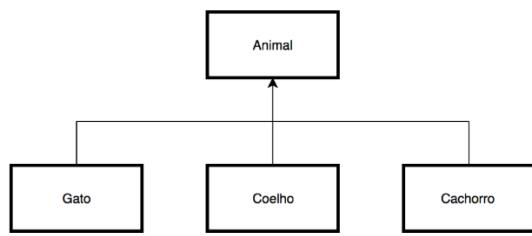
(venv) C:\Users\BIBLIOTECA\Desktop\projetosflask\flaskaluno>python3 -m flask db upgrade
INFO [alembic.runtime.migration] Context impl MySQLImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.runtime.migration] Running upgrade  -> 5c54b460f38f, empty message
```

Verifique o banco:

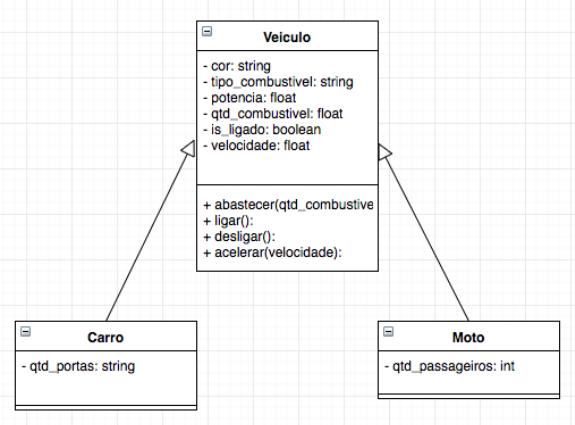


E	Criar as regras de validação para a classe UnidadeCompetencia
F	Criar um novo pacote chamado forms
G	 <p>Figura 16 - Criar o arquivo que vai validar a unidade de competência</p>

H	<p>Verificar importações e ambientes virtuais</p>  <pre>flaskaluno > app > forms > uc_form.py 1 from flask_wtf import FlaskForm 2 </pre> <p>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS</p> <p>Werkzeug==3.0.2 WTForms==3.1.2</p> <p>(venv) C:\Users\BIBLIOTECA\Desktop\aulasFlask\Flaskaluno>pip list</p> <table border="1"> <thead> <tr> <th>Package</th> <th>Version</th> </tr> </thead> <tbody> <tr><td>alembic</td><td>1.13.1</td></tr> <tr><td>boto3</td><td>1.20.0</td></tr> <tr><td>clics</td><td>0.1.7</td></tr> <tr><td>colorama</td><td>0.4.6</td></tr> <tr><td>Flask</td><td>3.0.3</td></tr> <tr><td>Flask-Migrate</td><td>4.0.7</td></tr> <tr><td>Flask-SQLAlchemy</td><td>3.0.1</td></tr> <tr><td>Flask-WTF</td><td>1.2.1</td></tr> <tr><td>greenlet</td><td>3.0.3</td></tr> <tr><td>itsdangerous</td><td>2.2.0</td></tr> <tr><td>Jinja2</td><td>3.1.3</td></tr> <tr><td>MarkupSafe</td><td>2.1.5</td></tr> <tr><td>mysql-connector-python</td><td>8.3.0</td></tr> <tr><td>mysqlclient</td><td>2.2.4</td></tr> <tr><td>pandas</td><td>240.0</td></tr> <tr><td>SQLAlchemy</td><td>2.0.29</td></tr> <tr><td>typing_extensions</td><td>4.11.0</td></tr> <tr><td>Werkzeug</td><td>3.0.2</td></tr> <tr><td>WTForms</td><td>3.1.2</td></tr> </tbody> </table> <p>(venv) C:\Users\BIBLIOTECA\Desktop\aulasFlask\Flaskaluno></p> <p>The following environment is selected: - (Desktop\aulasFlask\venv) venv</p> <p>Le 2, Col 1 Spaces: 4 UTF-8 CR/LF Python 3.12.3 (venv) venv</p>	Package	Version	alembic	1.13.1	boto3	1.20.0	clics	0.1.7	colorama	0.4.6	Flask	3.0.3	Flask-Migrate	4.0.7	Flask-SQLAlchemy	3.0.1	Flask-WTF	1.2.1	greenlet	3.0.3	itsdangerous	2.2.0	Jinja2	3.1.3	MarkupSafe	2.1.5	mysql-connector-python	8.3.0	mysqlclient	2.2.4	pandas	240.0	SQLAlchemy	2.0.29	typing_extensions	4.11.0	Werkzeug	3.0.2	WTForms	3.1.2
Package	Version																																								
alembic	1.13.1																																								
boto3	1.20.0																																								
clics	0.1.7																																								
colorama	0.4.6																																								
Flask	3.0.3																																								
Flask-Migrate	4.0.7																																								
Flask-SQLAlchemy	3.0.1																																								
Flask-WTF	1.2.1																																								
greenlet	3.0.3																																								
itsdangerous	2.2.0																																								
Jinja2	3.1.3																																								
MarkupSafe	2.1.5																																								
mysql-connector-python	8.3.0																																								
mysqlclient	2.2.4																																								
pandas	240.0																																								
SQLAlchemy	2.0.29																																								
typing_extensions	4.11.0																																								
Werkzeug	3.0.2																																								
WTForms	3.1.2																																								
I	<pre>1 from flask_wtf import FlaskForm 2 3 class UcForm(FlaskForm): ↵ 4 pass</pre>																																								
J	<p>Entender o conceito e prática de herança:</p> <ul style="list-style-type: none"> ✓ Herança é um pilar da orientação a objetos <p>A herança em orientação a objetos permite que uma classe herde métodos e atributos de outra classe, promovendo o reaproveitamento de código ao definir uma classe "pai" com características e métodos comuns e suas classes "filhas" que herdam essas informações. Embora as classes "filhas" possam ter métodos e atributos próprios, a herança facilita o compartilhamento de informações da classe "pai" para suas subclasses.</p>																																								



Herança na UML



K Como usar herança no Python

```

import veiculo
class Carro(veiculo.Veiculo):
  
```

L

```

1 from flask_wtf import FlaskForm
2
3 class UCForm(FlaskForm):
4     pass
  
```

A classe UCForm herda FlaskForm

M	<pre> 1 from flask_wtf import FlaskForm 2 from wtforms import StringField 3 from wtforms import IntegerField 4 from wtforms.validators import DataRequired 5 class UcForm(FlaskForm): 6 numero = IntegerField("numero",validators=[DataRequired()]) 7 nome = StringField("nome",validators=[DataRequired()]) 8 carga_horaria = IntegerField("carga_horaria",validators=[DataRequired()]) 9 competencia_geral = StringField("competencia_geral",validators=[DataRequired()]) 10 11 </pre>
---	--

N		<pre> 59 60 #Criando o CRUD de Unidade de Competência 61 </pre>
---	--	---

O	Bate papo : O que é CRUD?
---	---------------------------

P	<pre> from flask import render_template from app.forms import cliente_form from app import app # @app.route("/ola", defaults={'nome': None}) # @app.route("/ola/<string:nome>") # def teste(nome): # return render_template("clientes/teste.html", nome_usuario=n) # # @app.route("/oi") # def oi(): # return "Oi, mundo em Flask!!" @app.route("/cadastrar_cliente") def cadastrar_cliente(): form = cliente_form.ClienteForm() return render_template("clientes/form.html", form=form) </pre> <p>CRIAR</p>
---	--

Figura 17 - Comente sobre a imagem. Bate Papo Views+Forms

Anexo I- Os tipos na Linguagem Python

Python 3 The standard type hierarchy

