Professor Rômulo Cesar Silvestre

|  |
| --- |
| 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 *“microframework flask”* 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.

Interface gráfica do usuário

Descrição gerada automaticamente

**Habilidade:**

|  |  |
| --- | --- |
| A | Criar uma pasta na **área de trabalho:** |
| B | Abrir a pasta no Microsoft Visual Studio Code: |
| C | Instalar o ambiente virtual: |
| 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**    Leia no canto inferior direito que tipo de interpretador está selecionado. Caso não seja a venv selecione na pasta quando necessário. |
| G | Ativar o ambiente virtual    Volte para pasta original principal. |

# **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 |  |

# **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 comuniçaõ 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 vária, 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 | Analisar imagem do funcionamento de design pattern:  Como funciona a arquitetura MTV (Django) | by Diandra Andrade e Silva |  Medium |

# **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): |
| C | Gerar uma lista dos pacotes instalados juntamente com o flask: |

# Etapa 6: instalar pacotes

Habilidades

|  |  |
| --- | --- |
| A | Instalar o “mysql-connector-python 8.3.0” : |
| B | Dê um pip freeze > requirements.txt |
| C | Instalar: |
| D | Verificar as instalações: |
| E |  |

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

Habilidades

|  |  |
| --- | --- |
| A | Criar um arquivo ***config.py*** no projeto raiz:  Tela de computador com texto preto sobre fundo branco  Descrição gerada automaticamente |
| B | Estudar, corrigir e analisar codificação:  **Lista de Codificação 01:**  DEBUG = True  USERNAME = 'root'  PASSWORD = 'root'  SERVER = 'localhost'  DB = 'projectscore'  SQLALCHEMY\_DATABASE\_URI=f'mysql://{USERNAME}:{PASSWORD}@{SERVER}/{DB}'  SQLALCHEMY\_TRACK\_MODIFICATIONS = True |
| C | Verificar pacotes :  Flask-SQLAlchemy: trabalhar com ORM e Flask  mysql-connector-python : conector com banco  SQLAlchemy: ORM escolhido |
| 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 | ORM é um acrônimo para object-relational mapping – mapeamento objeto-relacional.   * 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 | DEBUG = True  DEBUG = False  Faça um teste utilizando os dois e escolha qual o melhor para você |
| 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    Assim como o GitHub faz o versionamento do código o Flask-Migrate faz o versionamento do banco. |
| M | Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |
| N |  |
| O |  |
| P | Em models crie um arquivo uc\_model.py: |
| Q | **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)  nome = db.Column(db.String(200))  carga\_horaria = db.Column(db.Integer)  competencia\_geral= db.Column(db.String(255)) |
| R | Programando o **\_\_init\_\_.py** do app |
| S | Verifique a linha do Migrate (importante) |
| R | Texto  Descrição gerada automaticamente |
| S |  |
| T | Crie um banco de dados conforme o **config.py** |
| U | Inicie o migrations |
| V |  |
| X | Avaliando se os dois comandos anteriores deram certo: |
| Z |  |

# 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**

Ícone

Descrição gerada automaticamente

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Figura - entre no github do treinamento

Abra a pasta no Visual Studio Code.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Abra o terminal e digite o seguinte código:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

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:

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

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

Texto

Descrição gerada automaticamente

Abra o MySQL Workbench e clique em root:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Dê um DROP DATABASE:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Dê um CREATE DATABASE:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Use o banco de dados criado:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

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

Interface gráfica do usuário

Descrição gerada automaticamente

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

Ative a máquina virtual:

Texto

Descrição gerada automaticamente

Volte para a pasta flaskaluno (clonada)

Texto

Descrição gerada automaticamente

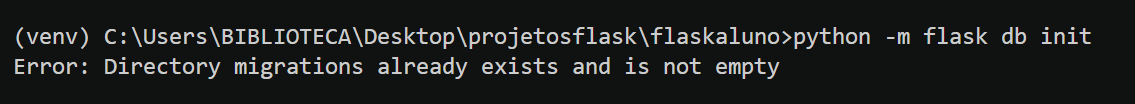


Figura - Dê um db init

Texto

Descrição gerada automaticamente

Figura - Dê um db migrate

Texto

Descrição gerada automaticamente

Figura - Dê um db upgrade

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura - Atualize o Navigator

Visualize as tabelas no Navigator:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

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

Gráfico

Descrição gerada automaticamente

A tabela **alembic\_version** é a tabela de versão do Flask-Migrate.

Para visualizar como diagrama podemos fazer a engenharia reversa.

Aperte CTRL+R

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura - Mantenha as configurações e clique em Next.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura - Digite a senha

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente

Figura - Clique em Next

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente

Figura - Clique em next (projectscore)

Veja o resultado da engenharia reversa:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média

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

Uma imagem contendo Aplicativo

Descrição gerada automaticamente

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

Vamos codar:

Texto

Descrição gerada automaticamente

Agora vamos preparar para a execução:

Texto

Descrição gerada automaticamente com confiança média

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

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Clique no arquivo e crie a **seguinte codificação**:

Interface gráfica do usuário

Descrição gerada automaticamente

Faça a importação da **uc\_view**

Texto

Descrição gerada automaticamente

Clique em cima do arquivo run.py e execute:

Texto

Descrição gerada automaticamente

Veja agora o resultado:

Interface gráfica do usuário

Descrição gerada automaticamente

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

|  |  |
| --- | --- |
| **rota** | **descrição** |
| /listuc | Listar unidades de competências  Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto  Descrição gerada automaticamente |
| /adduc | Adicionar unidade de competência  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |
| /deluc | Deletar unidade de competência  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |
| /upuc | Atualizar a unidade de competência  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |
| /meetuc | Encontrar uma unidade de competência  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |

habilidades

|  |  |
| --- | --- |
| A | Inserir comentário de uma linha:  *#Nela vou criar os métodos que minha aplicação vai executar* |
| B | Inserir comentários de múltiplas linhas:  """  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  """ |
| C | Realizar a importação do app:  from app import app |
| D | Realizar a importação do render\_template  from flask import render\_template |
| E | Criar uma rota para listar competências:  @app.route("/listuc")  def listar\_uc():      return render\_template("uncompetencias/uc\_template.html") |
| F | Criar uma rota para adicionar uma competência:  @app.route("/adduc")  def adicionar\_uc():      return "Adicionar unidade de competência" |
| G | Criar uma rota para apagar uma uc  @app.route("/deluc")  def excluir\_uc():      return "Excluir unidade de competência" |
| H | Criar uma rota para atualizar uma uc  @app.route("/upuc")  def atualizar\_uc():      return "Atualizar unidade de competência" |
| I | Definir duas rotas no mesmo método:  @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" |
| J | Resolver desafio lógico dentro de uma rota:  @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" |
| K | Observar o código e corrigir erros.  Segue lista completa do código para analisar e corrigir erros:  #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  /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" |
| L | Entender e explicar sobre o funcionamento de rotas com parâmetros criados:  Texto  Descrição gerada automaticamente |
| M | Entender e explicar sobre o funcionamento da estrutura de seleção múltipla match:  Texto  Descrição gerada automaticamente |
| N | Teste a rota ***/numberuc*** passando o valor 11, 12 e 13 para o parâmetro número:  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente  Interface gráfica do usuário, Texto  Descrição gerada automaticamente  Tente passar algo fora das condições do match:  Interface gráfica do usuário, Texto, Aplicativo, Site  Descrição gerada automaticamente |
| O | Trabalhar com duas rotas no mesmo método:  Rota original não modificada:  Interface gráfica do usuário  Descrição gerada automaticamente com confiança média  Solução modificada:  Texto  Descrição gerada automaticamente  Interface gráfica do usuário, Texto, Aplicativo, Email, Site  Descrição gerada automaticamente  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |
| 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:  Texto  Descrição gerada automaticamente  Interface gráfica do usuário, Site  Descrição gerada automaticamente  Você vai ficar proibido de acessar a rota.  Mude agora para ele aceitar via GET e POST  Interface gráfica do usuário, Texto, Aplicativo, Site  Descrição gerada automaticamente  Tela preta com letras brancas  Descrição gerada automaticamente |
| R | Testar as rotas no postman:    Teste as rotas no POSTMAN veja que deu o erro pois ele aceita só GET e POST  **Se ele usar o get**  Tela de celular  Descrição gerada automaticamente  Tela de celular  Descrição gerada automaticamente  Mude para PUT  Texto  Descrição gerada automaticamente  Tela de celular com aplicativo aberto  Descrição gerada automaticamente  Tela de celular com aplicativo aberto  Descrição gerada automaticamente  Tela de celular  Descrição gerada automaticamente |

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 tramitar 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 | **Views**  (trata as requisições e envia os dados para a camada Templates)  **TEMPLATES**  (mostra os dados para os usuário e interage com o mesmo) |
| F | Observe que as informações que quero apresentar para o usuário no código anterior encontra-se dentro da VIEW:  Texto  Descrição gerada automaticamente |
| G | Na camada templates, crie uma pasta (**unidcompetencia**) e um arquivo (**uc\_template.html**):  Interface gráfica do usuário, Aplicativo  Descrição gerada automaticamente |
| H | No arquivo html crie uma página html simples (use !) para gerar um código base html:  Texto  Descrição gerada automaticamente  Texto  Descrição gerada automaticamente  Figura - 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:  Interface gráfica do usuário, Texto  Descrição gerada automaticamente |
| K | Primeiro você tem que importar:  from flask import **render\_template** |
| L | Coloque um texto na página HTML:  Texto  Descrição gerada automaticamente |
| 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.  Texto  Descrição gerada automaticamente |
| N | Interface gráfica do usuário  Descrição gerada automaticamente  Figura - teste a rota enviando mensagens para a camada template |
| O | Texto  Descrição gerada automaticamente  Texto  Descrição gerada automaticamente com confiança média |
| P | Inspecione os códigos no seu navegador preferido (os desenvolvedores web e front-end, gostam muito do Firefox) |
| Q | 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:  Vamos escolher o seguinte método para fazer o experimento:  Texto  Descrição gerada automaticamente  Veja a modificação que você vai ter que fazer:    Interface gráfica do usuário, Texto, Site  Descrição gerada automaticamente  Texto  Descrição gerada automaticamente |

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 | Texto  Descrição gerada automaticamente |
| D | Descompacte o arquivo baixado:  Interface gráfica do usuário, Aplicativo  Descrição gerada automaticamente  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:  Interface gráfica do usuário, Aplicativo  Descrição gerada automaticamente |
| G | Criar um arquivo login\_uc.html    Gerar o html:    Figura - mude o idioma e o title |
| H | Crie um comentário html |
| I |  |
| J | Faça a importação:    <link rel="stylesheet"  href="{{url\_for('static',filename='css/bootstrap.css')}}"> |
| K |  |
| L | Versão: 5.3.3 |
| M | Grid: |
| N | Coloque o texto **célula** |
| O | Tinhamos criado uma rota para login lembra?    Vamos modificá-la: |
| P | Vamos executar: |
| Q | Agora vamos criar uma segunda linha que terá um botão pegando as doze colunas.  Nova linha:    Entrando no site para escolher um botão:      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. |

# **Etapa 15**: criar template com jinja 2

**Conhecimentos**

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

**Habilidades**

|  |  |
| --- | --- |
| A | {{ }} - usado para embutir valores de variáveis python.  Texto  Descrição gerada automaticamente |
| B | {% %} – usado para embutir codificação python – possui uma sintaxe especial  Tela preta com letras brancas  Descrição gerada automaticamente |
| C | Resultado:  Interface gráfica do usuário, Texto, Aplicativo, Email  Descrição gerada automaticamente  Interface gráfica do usuário, Texto, Aplicativo  Descrição gerada automaticamente |

# **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 | Flask-WTF é uma ponte do WTForms e o Flask |
| B |  |
| C |  |
| D | Após o clone verifique o arquivo **requeriments.txt**      Se ambos estiverem instalados.  Repita os passos de ambientação do Migrate, caso ainda não tenha feito.  Após o init:    Verifique o banco: |
| E | Criar as regras de validação para a classe **UnidadeCompetencia** |
| F | Criar um novo pacote chamado **forms** |
| G | Figura - Criar o arquivo que vai validar a unidade de competência |
| H | Verificar importações e ambientes virtuais |
| I |  |
| J | Entender o ***conceito e prática*** de herança:   * Herança é um pilar da orientação a objetos   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.    Herança na UML |
| K | Como usar herança no Python |
| L | A classe UCForm herda FlaskForm |
| M |  |
| N |  |
| O | Bate papo : O que é CRUD? |
| P | Figura - Comente sobre a imagem. Bate Papo Views+Forms |
|  |  |
|  |  |
|  |  |

# Anexo I - Os tipos na Linguagem Python

Tela de celular com texto branco sobre fundo preto

Descrição gerada automaticamente