Quick Report – Dúvida- quero estilizar de forma personalizada no framework, mas não atrelado 100% aos frameworks?

Parabéns, se você está lendo esse tutorial com certeza se encontra na parte mais emocionante do projeto. Porém, a mais desafiadora, que é ligar múltiplas tabelas em um único front-end. Isso, pode parecer confuso no início, mas com a prática logo estará apto a criar as suas próprias telas com múltiplos dados.

Vamos atender a primeira dúvida, e nesse suporte vou explicar uma forma de personalização dos seus template, usando a parte “static” do projeto. E também uma técnica de sobrepor os frameworks css.

Então vamos começar.

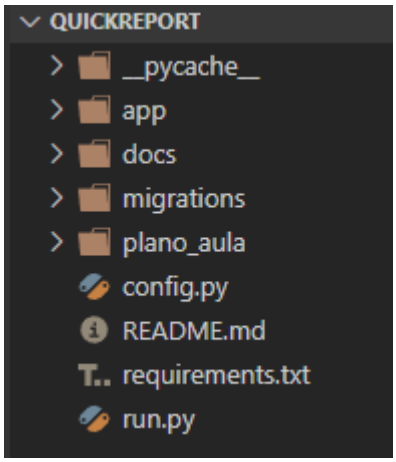
O projeto base do professor para apoio se encontra no seguinte link:

https://github.com/romulosilvestre/quickreport.git

Ele possui apenas o CRUD de escola.

Vamos revisitar agora o modelo de banco de dados construído por você, vamos?

Estrutura clonada



Vamos focar no curso

Observe que conforme orientação do modelo conceitual de dados, o curso deve ter um id estrangeiro, chamamos de foreign key.

```python
from app import db
class Curso(db.Model):
    __tablename__ = "curso"
    id = db.Column(id.Integer,primary_key=True,autoincrement=True)
    nome = db.Column(db.String(225))
    fk_Escola_id= db.Column(db.Integer,db.ForeignKey('escola.id'))

```

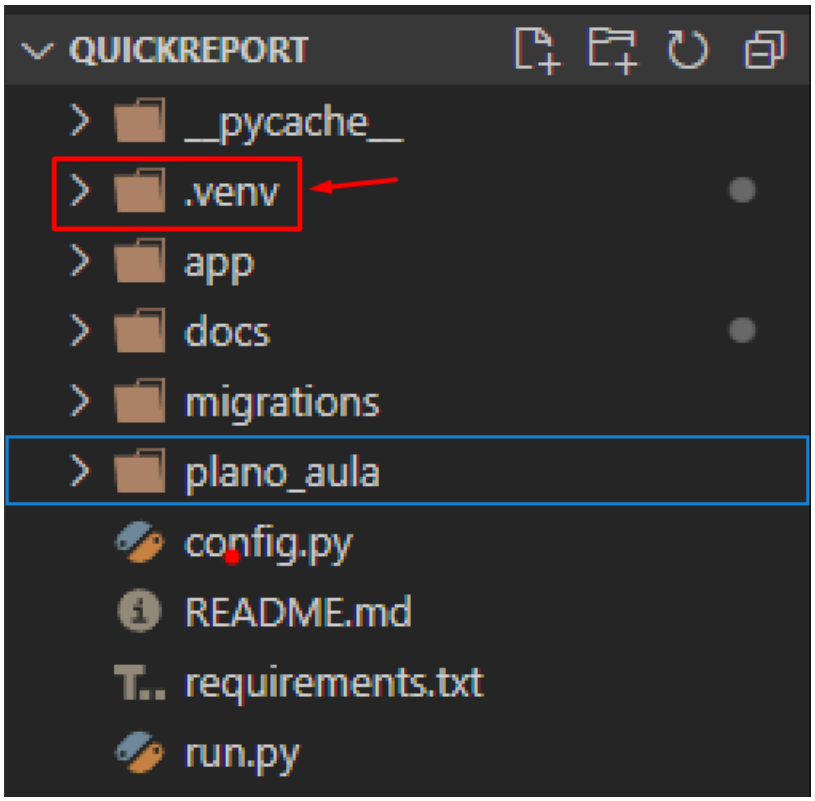
Esse é o desafio. Em um mesmo template de curso teremos que ter informações do curso para cadastrar, mas trazer algumas informações da tabela escola.

Como só tem duas informações da escola (id, nome) são os dois necessários para listarmos dentro do select html.

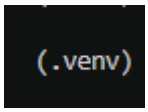
Vamos executar o projeto para verificar o cadastro.

Iniciamos criando um novo ambiente virtual. Não é necessário criar caso já tenha. Esse processo é só caso tenha que clonar a primeira vez, o quer clonar a versão do professoro pela primeira vez, ou perdeu o arquivo em alguma máquina.

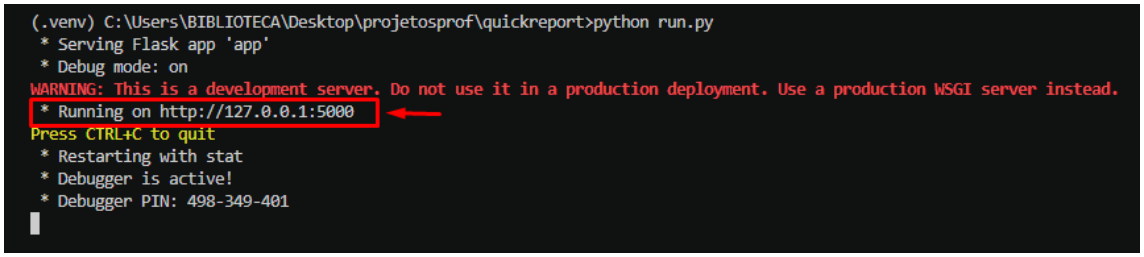
Vamos criar a venv.



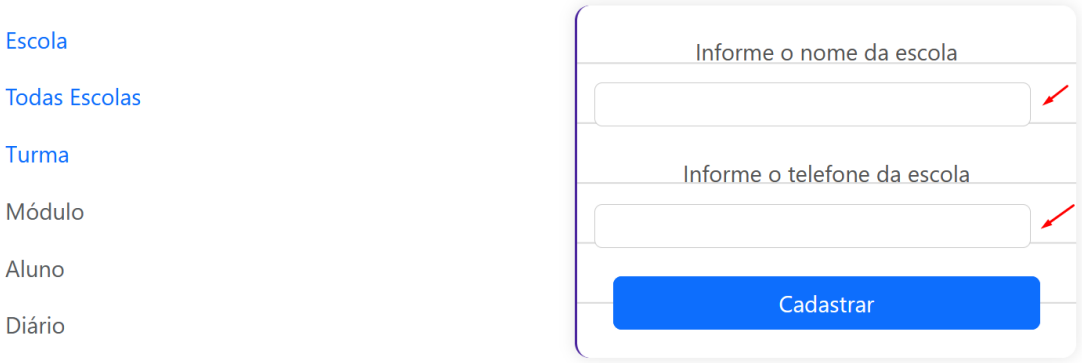
Ative a .venv

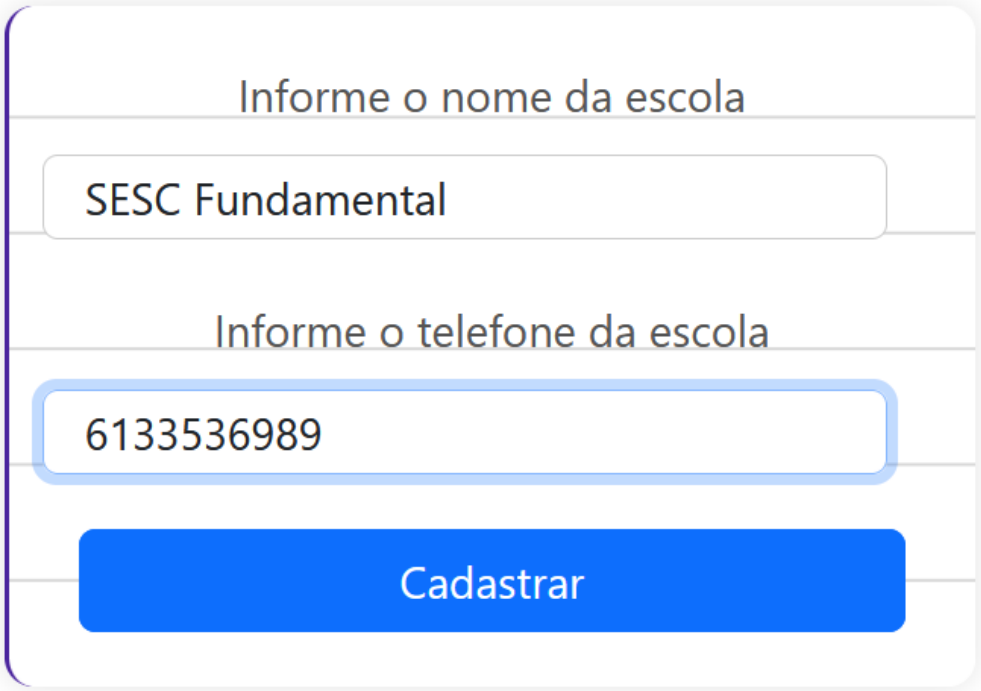


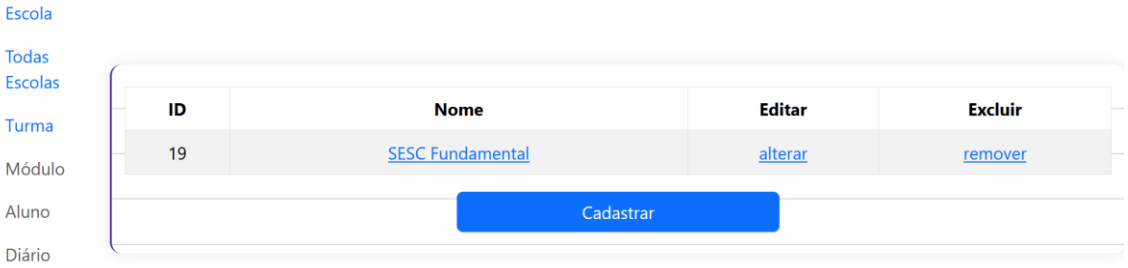
Execute o projeto



Vou cadastrar a escola.







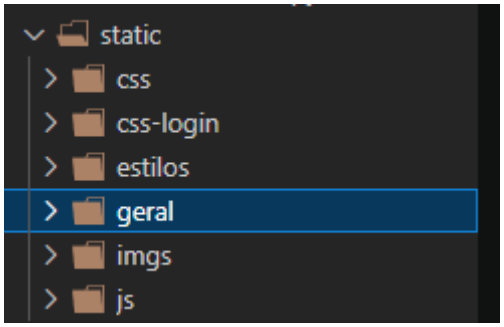
Parte 1 – CSS personalizado

Uma dúvida que foi levantada na sala de aula é sobre o visual da aplicação. Se a estrutura ficar dividida conforme se encontra na estrutura do projeto fica tudo tranquilo e fácil a utilização de recursos estáticos.

Vou dar um exemplo simples de um css para formatar um table html.

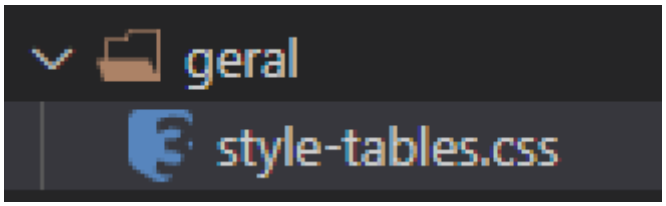
```
{% extends "escola/base.html"  %}



{% block conteudo %}


    <div class="container mt-5">
        <table class="table table-bordered table-striped">
            <thead class="thead-dark">
                <tr>
                    <th>ID</th>
                    <th>Nome</th>
                    <th>Editar</th>
                    <th>Excluir</th>
                </tr>
            </thead>
            <tbody>
                {% for escola in escolas %}
                    <tr>
                        <td>{{ escola.id }}</td>
                        <td><a href="{{url_for('listar_escola',id=escola.id)}}">{{ escola.nome }}</a></td>
                        <td><a href="{{url_for('editar_escola',id=escola.id)}}">alterar</a></td>
                        <td><a href="{{url_for('remover_escola',id=escola.id)}}">remover</a></td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
        <a href="{{url_for('cadastrar_escola')}}"> <button type="submit" class="btn btn-primary">Cadastrar</button></a>

    </div>
{% endblock conteudo %}
```

Dentro da static, pasta criada especificamente para trabalhar com o “site”, o front-end puro em suas estilizações bootstrap ou css autoral.



Crie uma pasta geral, para um css autoral, apesar que no bootstrap posso formatar a tabela também.

Criei um arquivo:



Nele vou criar uma classe css para formatar linhas e bordas de uma tabela.

Criei um css que estiliza a tabela para ficar com a aparência de um carro.

```css
.notebook-table{
    table {
        width: 100%;
        border-collapse: collapse;
    }

    table, th, td {
        border: 1px solid black;
    }

    th, td {
        padding: 10px;
        text-align: left;
    }

    th {
        background-color: #f2f2f2;
    }
}
```

Agora precisamos linkar esse arquivo na tela de listagem de escola.

Lembre que nosso projeto esta vinculado a uma base.

```
{% extends "escola/base.html"  %}
```

De forma bem-organizada dividi em três arquivos css:

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

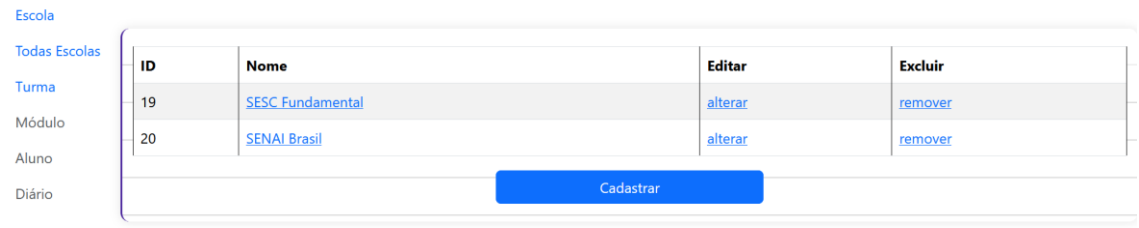
Agora temos mais um link que é o geral/style-tables.css

Como ele tem herança de templates automaticamente ele vai utilizar o css da base.

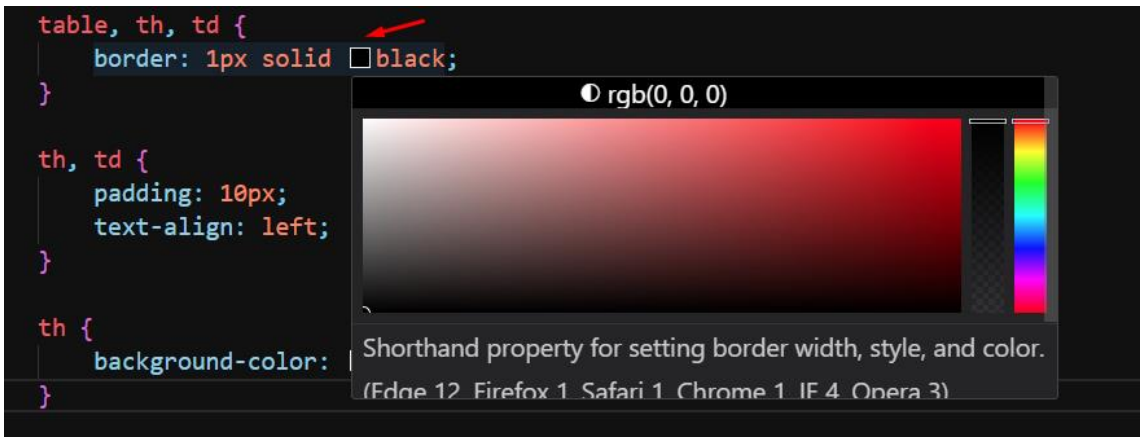
Agora vamos acrescentar a classe para verificar o que aconteceu!

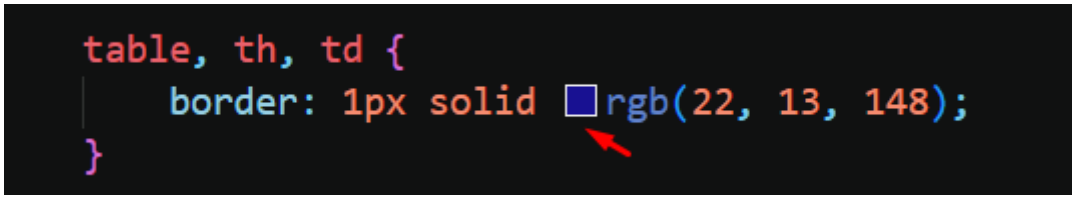
```
<div class="container mt-5">
    <table class="table table-bordered table-striped notebook-table">
        <thead class="thead-dark">
            <tr>
                <th>ID</th>
                <th>Nome</th>
                <th>Editar</th>
                <th>Excluir</th>
            </tr>
        </thead>
        <tbody>
            {% for escola in escolas %}
                <tr>
                    <td>{{ escola.id }}</td>
                    <td><a href="{{url_for('listar_escola',id=escola.id)}}">{{ escola.nome }}</a></td>
                    <td><a href="{{url_for('editar_escola',id=escola.id)}}">alterar</a></td>
                    <td><a href="{{url_for('remover_escola',id=escola.id)}}">remover</a></td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
    <a href="{{url_for('cadastrar_escola')}}"> <button type="submit" class="btn btn-primary">Cadastrar</button></a>
```

Olá, o resultado ficou da seguinte forma:



Agora você quer mudar a cor da borda.

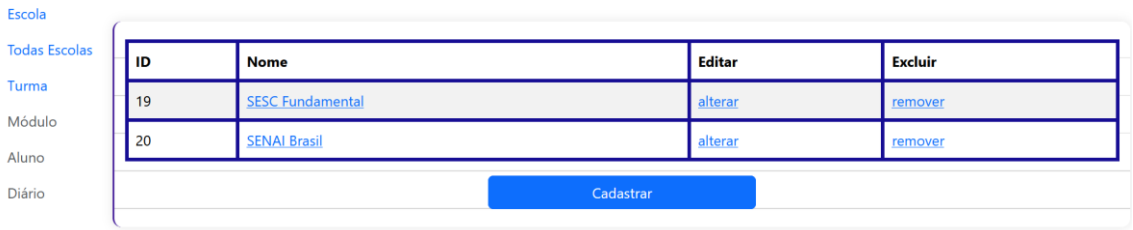




Vou deixar a borda maior:

```css
table, th, td {
    border: 3px solid rgb(22, 13, 148);
}
```

Se você quer sobrepor outros css inclusive o css do bootstrap pode usar !important



Para ele ficar com as bordas conforme o exemplo acima, tive que inserir o seguinte código no css:

```css
table, th, td {
    border: 4px solid rgb(22, 13, 148) !important;
}
```

Assim, você fica a vontade para decidir se usa o css personalizado ou se o css do bootstrap. Apesar do bootstrap ter muitos recursos, mas sei que você gosta de personalizar e deixar do seu jeito.

Espero ter ajudado na dúvida específica de trabalhar com o visual e uma introdução a utilização de herança de templates.

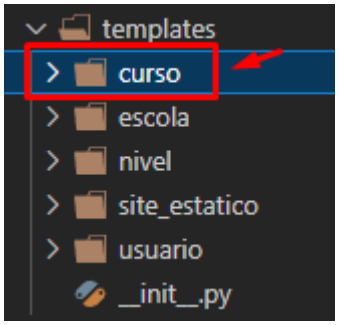
Assim, na camada estática você esta livre para estilizar usando! important. Infinitas possibilidades para estilização de front-end surgem independente do framework utilizado.

Espero ter ajudado nesse suporte escrito!

A disposição!

Parte 2 – Criando a tela curso trabalhando com múltiplas tabelas do banco de dados

Crie uma pasta chamada curso



No template curso na página index.html vamos criar uma página que exibe 10 vezes uma <h1>, veja também explicações sobre os delimitadores:

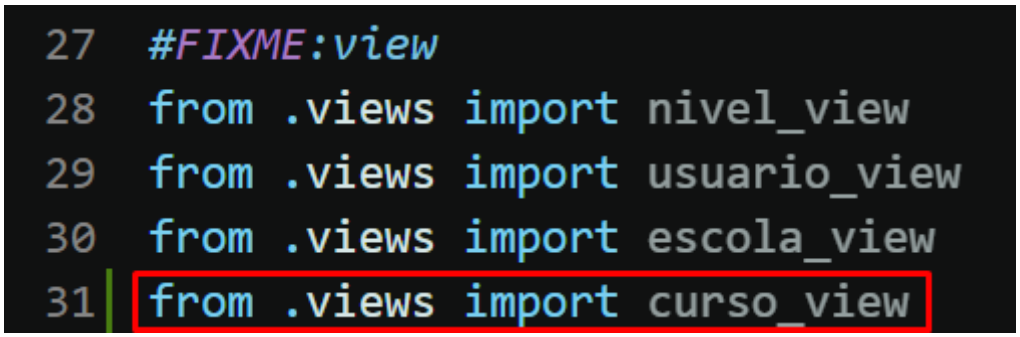
```
<!--Como a escola é nossa primeira entidade, vou considera-la como base para as outras-->
<!--Considerações sobre delimitadores do Jinja 2-->

{#

    Pode utilizar esses delimitadores para comentar nos templates
    Outros delimitadores:
    {%   %}  : usado para utilizar métodos python nos templates
    {{   }}  : serve para concatenar variáveis aos templates

#}

{# vamos dar um exemplo simples: #}

{% for i in range(10): %}

    <h1>estou no loop {{ i }}</h1>

{% endfor%}

```

Para essa página funcionar vamos criar uma rota especifica para curso:

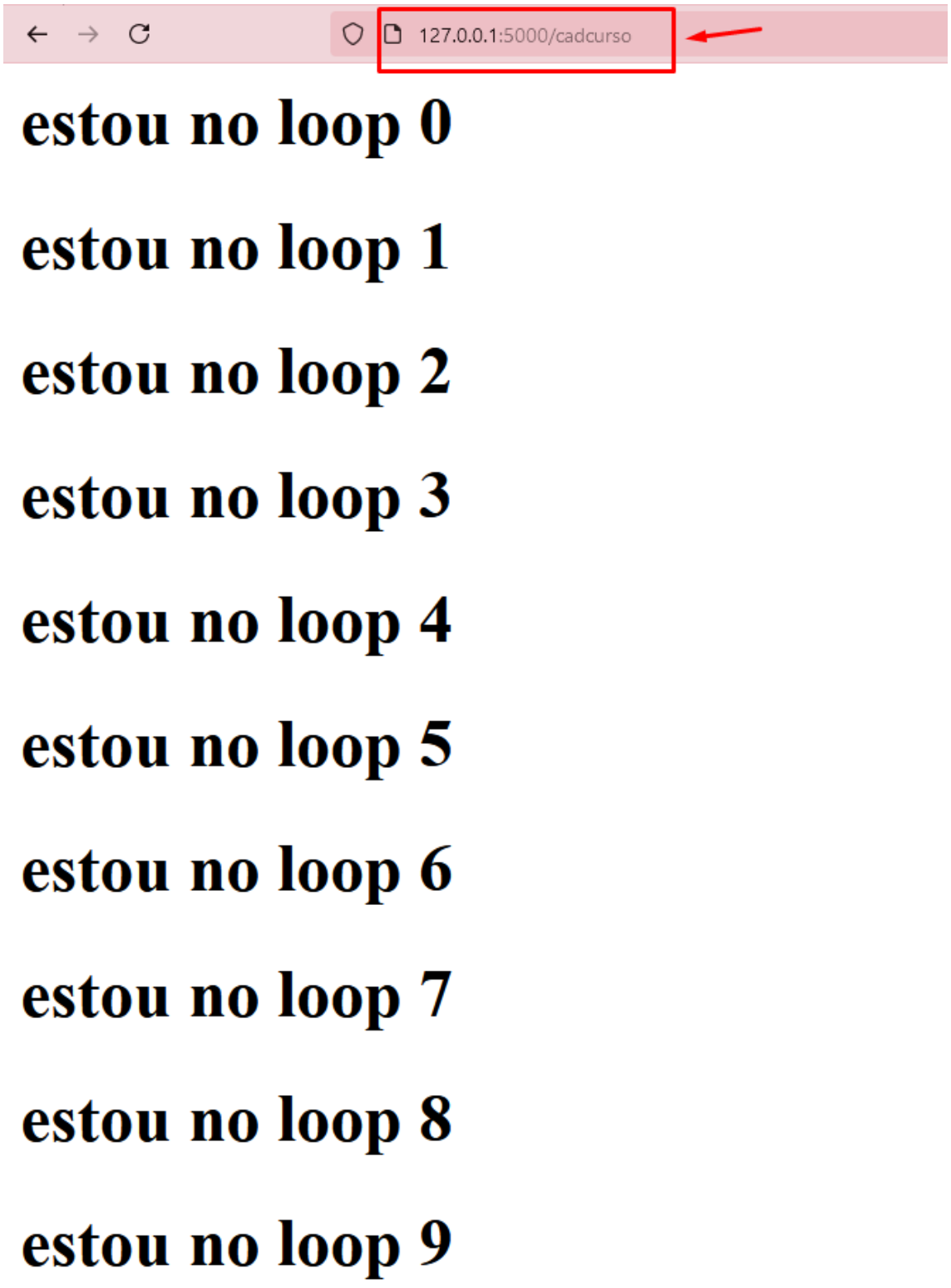
```
from app import app
from flask import render_template

@app.route("/cadcurso",methods=["POST","GET"])
def cadastrar_curso():
    return render_template("curso/index.html")

```

Assim, temos uma rota que nos auxilia no cadastro de curso. Por enquanto o objetivo é só abrir a tela index.html e ver se os delimitadores do jinja 2 está funcionando corretamente.

Não esqueça de trazer para __init__.py o que você acabou de criar:



Vamos rodar a aplicação:



Como esse exemplo foi apenas didático vou comentar o código ok.

```
15  {#
16  {% for i in range(10): %}
17
18      <h1>estou no loop {{ i }}</h1>
19
20  {% endfor%}
21
22  #}
```

Pronto o loop sumiu

Agora que você sabe comentar e programar no template usando jinja 2.

Vamos importar a base da escola, reaproveitando o arquivo base principal , vamos:

```
24  {% extends escola/base.html %}
25
26
27
```

Então, aqui estou herdando o html da outra pasta.

Vamos executar a aplicação?

Ele deu um erro

Isso acontece porque temos que voltar o diretório para usar outra pasta como herança:

```
{% extends "./escola/base.html" %}
```

E é necessário usar “ “.

Lembre-se, o template engine converte os códigos jinja 2 para o que o navegador realmente entende: HTML, CSS e JS.

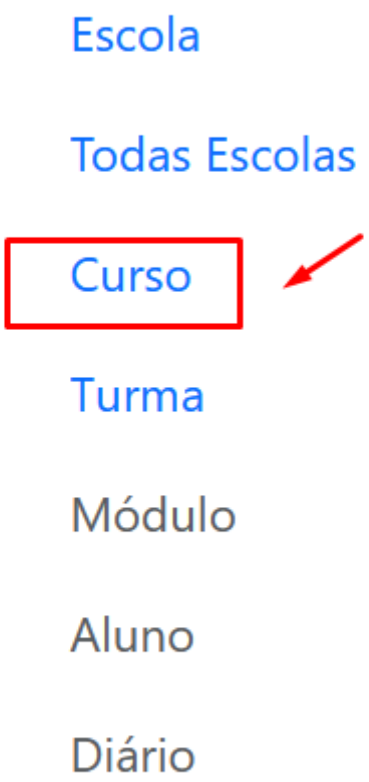
Veja que não tem curso. Olha só que legal a velocidade de atualização usando Jinja.

Você vai atualizar em apenas um lugar e ele vai atualizar em todos. Vamos?

Abaixo de todas as escolas vou criar um link:

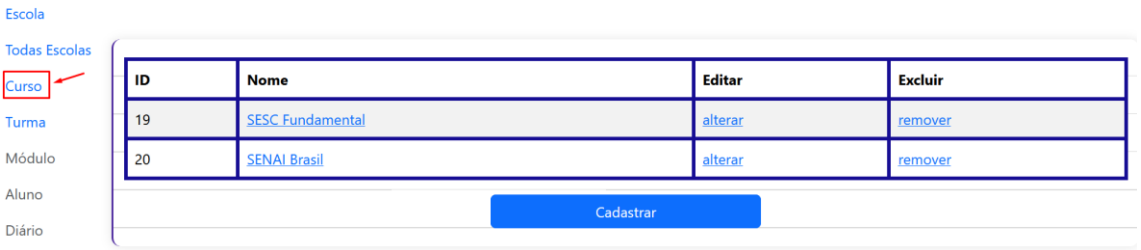
```
<li class="nav-item">
    <a class="nav-link" href="{{url_for('listar_escolas')}}">Todas Escolas</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="#">Curso</a>
</li>
```

Agora atualize!!!



Todas as páginas receberam a atualização.

Vamos para todas as escolas por exemplo.



Veja que já foi atualizado automaticamente.

Agora que você aprendeu a usar o Jinja 2 com uma introdução de herança de template. Vamos ajustar a rota.

Vamos?

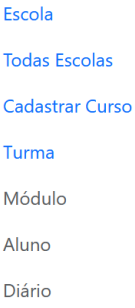
Entre no template escola/base.html

Atualize o texto e a url do hiperlink :

```
<li class="nav-item">
    <a class="nav-link" href="{{url_for('cadastrar_curso')}}">Cadastrar Curso</a>
</li>
```

Atualize e veja o resultado!

Estou em listar escolas e vou clicar curso.



Assim, foi direcionado para a tela de cadastro de curso.

Agora vamos colocar um select dinâmico que busque os dados de escolas de forma personalizada para mostrar os dados em uma select html. Vamos?

Vamos criar um form curso:

```python
from flask_wtf import FlaskForm
from wtforms import StringField,SelectField
from wtforms.validators import DataRequired
class CursoForm(FlaskForm):
    nome = StringField("nome",validators=[DataRequired()])
    escola_id = SelectField('Escola', coerce=int, validators=[DataRequired()])
```

Importe na rota

```python
from app import app
from flask import render_template
from app.models.escola_model import Escola
from app.forms.curso_form import CursoForm

```

Na linha: **from app import app**

Importação do app

==========

Na linha: **from flask import render_template**

Importação flask, importo o render_template para renderizar a página web

==========

Na linha: **from app.models.escola_model import Escola**

Faço uma importação direta como você gostou na aula

==========

Na linha: **from app.forms.curso_form import CursoForm**

Faço a importação da camada form

==========

Na linha: @app.route("/cadcurso",methods=["POST","GET"])

Programo a rota

==========

def cadastrar_curso():

form_curso = CursoForm()

Programo a instância do objeto form_curso da classe CursoForm( )

 # Carregar os níveis do banco de dados

escolas = Escola.query.all()

==========

 # Formatar os níveis para o formato necessário para o campo de seleção

escolas_choices = [(escola.id, escola.nome) for escola in escolas]

Vamos a explicação detalhada linha a linha:

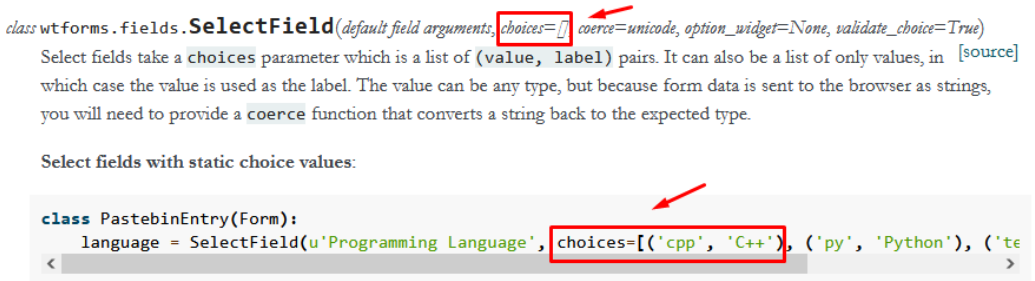
1. Criamos uma variável escolas_choices.
2. Essa variável recebe os dados da tabela escola de forma personalizada
3. Para personalizar tivemos que criar um for onde a variável escola percorre escolas (que a lista que veio do banco)
4. Criamos duas outras variáveis escola.id e escola.nome que representa os dois campos que virão do banco. Assim, conseguimos personalizar.
5. Armazenamos essa personalização dentro da variável escolas_choices.

==========

form_curso.escola_id.choices = escolas_choices

Vamos a explicação detalhada linha a linha:

1. A variável escolas_choices possui a personalização lembra (id, nome) para carregar o select html lá no template curso/index.html
2. Então, para que o form_curso entenda precisamos , pegar a variável que criamos para a chave estrangeira escola_id.
3. form.curso.escola_id , assim acessamos o form e a variável responsável pelo relacionamento.
4. Form_curso.escola_id.choices : choices é uma propriedade do SelectField conforme documentação do mesmo.
5. Armazenamos essa personalização dentro da variável escolas_choices.



class wtforms.fields.**SelectField**(*default field arguments, choices=[], coerce=unicode, option_widget=None, validate_choice=True*)

Select fields take a `choices` parameter which is a list of `(value, label)` pairs. It can also be a list of only values, in which case the value is used as the label. The value can be any type, but because form data is sent to the browser as strings, you will need to provide a `coerce` function that converts a string back to the expected type. [source]

Select fields with static choice values:

```
class PastebinEntry(Form):
    language = SelectField(u'Programming Language', choices=[('cpp', 'C++'), ('py', 'Python'), ('te
```

if form_curso.validate_on_submit():

pass

Vamos a explicação detalhada linha a linha:

1. Aqui eu faço uma verificação condicional se o formulário foi enviado, no caso o formulário form_curso.
2. Nesse caso coloquei “pass”, porque meu objetivo agora é apenas mostrar como carregar o SelectField do WtForms e também no Select HTML que esta no template curso/index.html.

==========

Por último, e não menos importante o retorno do formulário como parâmetro da função render_template (envio do template, envio do formulário)

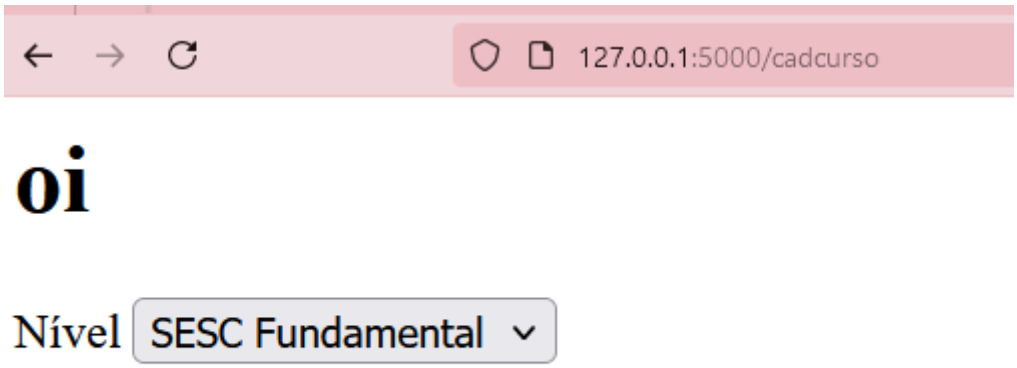
return render_template("curso/index.html",form_curso=form_curso)

Para rodar aplicação inicialmente tire a herança de template para testar:

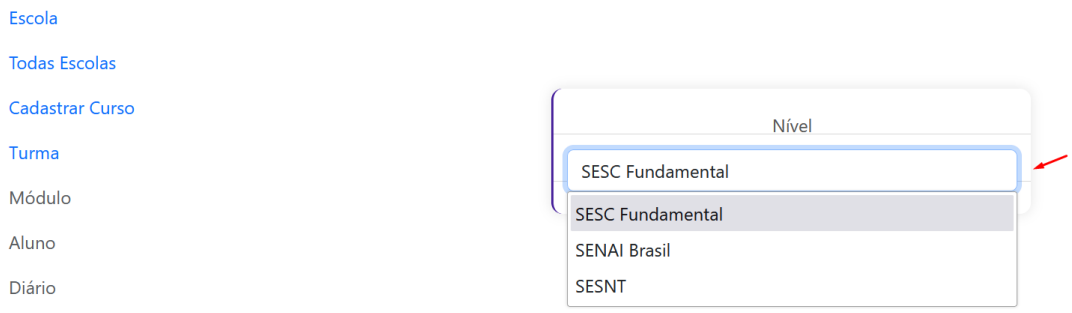
```
<!--Como a escola é nossa primeira entidade, vou considera-la

como base para as outras-->

<!--Considerações sobre delimitadores do Jinja 2-->



{#



    Pode utilizar esses delimitadores para comentar nos

templates

    Outros delimitadores:

    {%   %}  : usado para utilizar métodos python nos templates
```

```
    {{   }}  : serve para concatenar variáveis aos templates


#}


{# vamos dar um exemplo simples: #}


{#
{% for i in range(10): %}


    <h1>estou no loop {{ i }}</h1>


{% endfor%}


#}
<h1>oi</h1>
<div class="form-group">
    <label>Nível</label>
    <select id="escola_id" name="escola_id" class="form-
control">
        {% for id, nome in form_curso.escola_id.choices %}
            <option value="{{ id }}">{{ nome }}</option>
        {% endfor %}
    </select>
```

```
</div>
```



Para ficar com o visual elegante coloquei o boostrap e no container na classe coloquei col-4



Segue a codificação do template, ficou grande pois tem outras explicações didáticas que fiz pra você.

<!--Como a escola é nossa primeira entidade, vou considera-la como base para as outras-->

<!--Considerações sobre delimitadores do Jinja 2-->

{% extends "./escola/base.html" %}

{#

Pode utilizar esses delimitadores para comentar nos templates

Outros delimitadores:

{% %} : usado para utilizar métodos python nos templates

{{ }} : serve para concatenar variáveis aos templates

#}

{# vamos dar um exemplo simples: #}

{#

{% for i in range(10): %}

<h1>estou no loop {{ i }}</h1>

{% endfor%}

#}

{% block conteudo %}

<label>Nível</label>

<select id="escola_id" name="escola_id" class="form-control">

{% for id, nome in form_curso.escola_id.choices %}

<option value="{{ id }}">{{ nome }}</option>

{% endfor %}

</select>

{% endblock conteudo %}

Está marcado de amarelo os códigos que alterei para melhorar o visual herando templates.