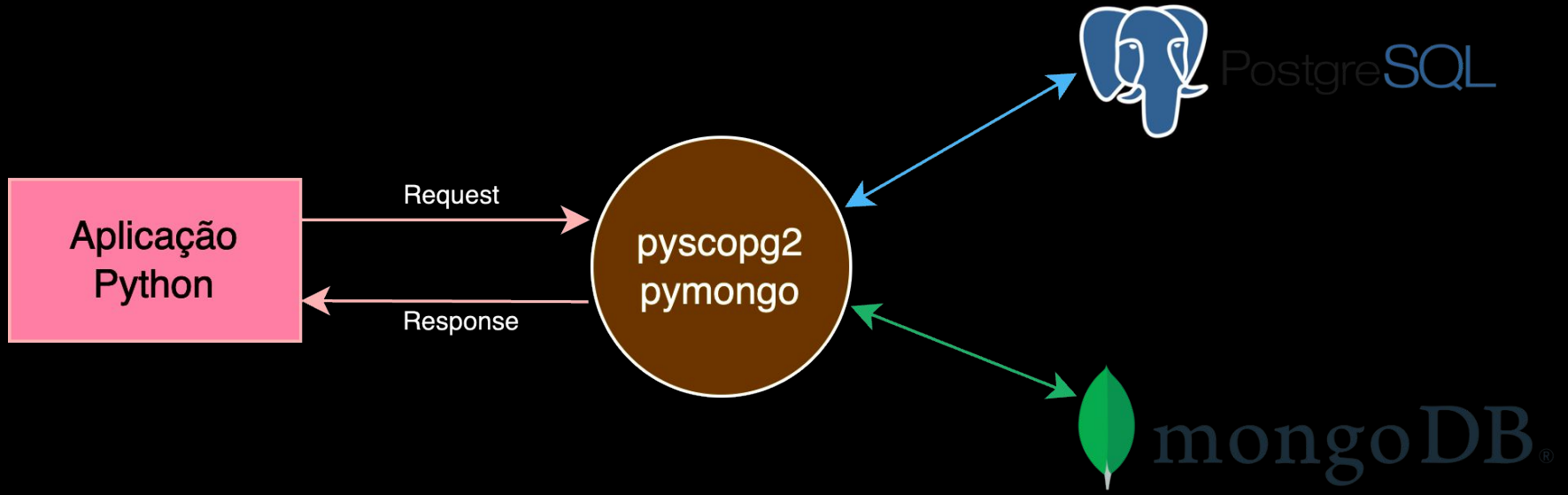
Aula 07

- Integração do Python com MongoDB parte 1

Integração do Python com MongoDB parte 1

Python Conectores



Ambientes virtuais

- Crie um novo ambiente virtual executando o comando venv:

Ambientes virtuais

- Para ativar o ambiente virtual:

```
# No Linux/Mac use o seguinte comando:
source venv/bin/activate

# No Windows:
venv\Scripts\activate
```

PyMongo

- O PyMongo é um conjunto de ferramentas para interagir com o banco de dados MongoDB no Python;
- O pacote PyMongo é um driver Python nativo para MongoDB;
- Você pode encontrar mais detalhes na página da lib no Python Package Index;
- Faça a instalação do pymongo usando o pip:

```
1  pip install pymongo
```

Conectando-se ao MongoDB com Python

- Também precisamos criar uma conexão com o MongoDB;
- Crie um arquivo client_db.py com o seguinte código:

```python
from pymongo import MongoClient
from pymongo.server_api import ServerApi


if __name__ == '__main__':
    client = MongoClient('mongodb+srv://jonilsonds9:<password>@cluster0.da2icrh.mongodb.net/?retryWrites=true&w=majority',
                         server_api=ServerApi('1'))

    try:
        client.admin.command('ping')
        print('Conectado ao banco de dados')
    except Exception as e:
        print(e)
```

Isolando client com o padrão Factory

- Vamos renomear nosso módulo client_db.py para client_factory.py e alterarmos o código para:

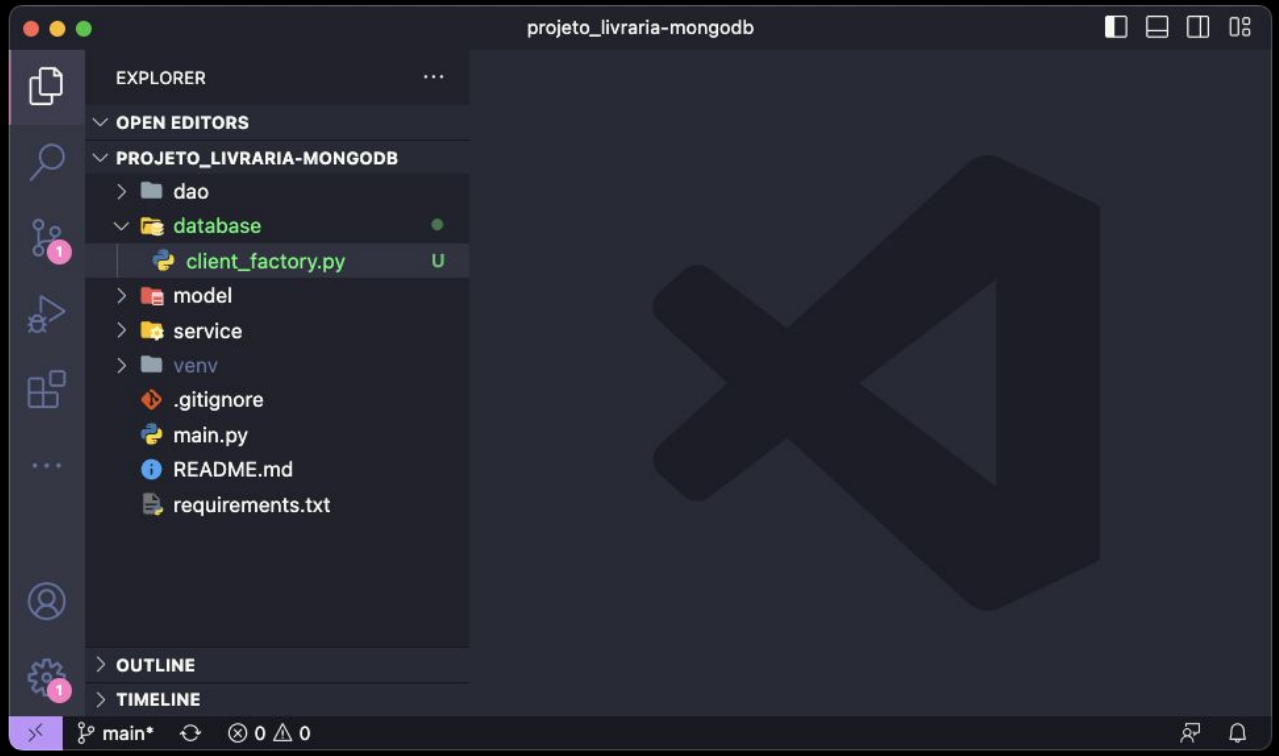
```python
from pymongo import MongoClient
from pymongo.server_api import ServerApi

class ClientFactory:

    def get_client(self):
        return MongoClient(
            'mongodb+srv://jonilsonds9:QmENRV1ddPg139Mk@cluster0.da2icrh.mongodb.net/?retryWrites=true&w=majority',
            server_api=ServerApi('1'))

```

Camada database



Alterando model Categoria

- Antes de mais nada precisamos remover o campo Id do construtor do model Categoria:

```python
class Categoria:

    def __init__(self, nome: str): # alterado
        self.__id: int = 0 # alterado
        self.__nome: str = nome

    def __str__(self):
        return f'{self.__id} | {self.__nome}'

    @property
    def id(self) -> int:
        return self.__id

    @id.setter
    def id(self, id: int):
        self.__id = id

    @property
    def nome(self) -> str:
        return self.__nome

    @nome.setter
    def nome(self, nome: str):
        self.__nome = nome

```

CategoriaDAO.adicionar

- Agora precisamos utilizar o nosso client, para usarmos o insert_one da coleção categorias para salvar uma nova categoria no CategoriaDAO.

```python
from model.categoria import Categoria
from database.client_factory import ClientFactory

class CategoriaDAO:

    def __init__(self):
        self.__client: ClientFactory = ClientFactory()

    # ...

    def adicionar(self, categoria: Categoria) -> None:
        client = self.__client.get_client()
        db = client.livraria
        db.categorias.insert_one({'nome': categoria.nome})
        client.close()
```

CategoriaService.adicionar

- Precisamos alterar o método adicionar do CategoriaService para deixarmos de lidar com id e assim passar essa responsabilidade para MongoDB.

```python
class CategoriaService:

    # ...

    def adicionar(self):
        print('\nAdicionando categoria ... ')

        try:
            # último id removido
            nome = input('Digite o nome da categoria: ')
            nova_categoria = Categoria(nome) # alterado
            self.__categoria_dao.adicionar(nova_categoria)
            print('Categoria adicionada com sucesso!')
        except Exception as e:
            print(f'Erro ao inserir categoria! - {e}')
            return

        input('Pressione uma tecla para continuar ... ')
```

Alterando o listar da CategoriaDAO

- Agora precisamos alterar o método listar do CategoriaDAO para buscar todos os documentos da coleção de categorias, e recuperar todas as categorias salvas no MongoDB, e continuar retornando um list[Categoria] para o service.

```python
class CategoriaDAO:

    def __init__(self):
        self.__client: ClientFactory = ClientFactory()

    def listar(self) -> list[Categoria]:
        categorias = list()

        client = self.__client.get_client()
        db = client.livraria
        for documento in db.categorias.find():
            cat = Categoria(documento['nome'])
            cat.id = documento['_id']
            categorias.append(cat)
        client.close()

        return categorias
```

Alterando model Categoria

- Agora para trabalharmos conforme o MongoDB funciona precisamos alterar o atributo id do nosso model Categoria de **int** para **ObjectId** do bson, que representa um Id no MongoDB, para assim podermos utilizar o MongoDB corretamente.

```python
from bson import ObjectId

class Categoria:

    def __init__(self, nome: str):
        self.__id: ObjectId = None # alterado
        self.__nome: str = nome

    def __str__(self):
        return f'{self.__id} | {self.__nome}'

    @property
    def id(self) -> ObjectId: # alterado
        return self.__id

    @id.setter
    def id(self, id: ObjectId):  # alterado
        self.__id = id

    @property
    def nome(self) -> str:
        return self.__nome

    @nome.setter
    def nome(self, nome: str):
        self.__nome = nome

```

CategoriaDAO.remover

- Agora precisamos remover o documento que tiver o categoria_id informado.

```python
from model.categoria import Categoria
from database.client_factory import ClientFactory
from bson import ObjectId

class CategoriaDAO:

    # ...

    def remover(self, categoria_id: str) -> bool: # alterado: int para str
        client = self.__client.get_client()
        db = client.livraria
        resultado = db.categorias.delete_one({'_id': ObjectId(categoria_id)}) # alterado
        if (resultado.deleted_count == 1):
            return True
        return False
```

CategoriaService.remover

- Agora precisamos modificar o nosso input do método remover do CategoriaService, para que aceite string (str), já que o campo Id do MongoDB é um ObjectId (que usa numéros e letras), assim precisamos receber como string e depois converter para ObjectId no nosso CategoriaDAO como fizemos no passo anterior.

```python
class CategoriaService:

    # ...

    def remover(self):
        print('\nRemovendo categoria ... ')

        try:
            categoria_id = input('Digite o ID da categoria para excluir: ') # alterado
            if (self.__categoria_dao.remover(categoria_id)):
                print('Categoria excluída com sucesso!')
            else:
                print('Categoria não encontrada!')
        except Exception as e:
            print(f'Erro ao excluir categoria! - {e}')
            return

        input('Pressione uma tecla para continuar ... ')
```

CategoriaDAO.buscar_por_id

- E por fim podemos implementar o buscar_por_id usando também o ObjectId.

```python
from bson import ObjectId

class CategoriaDAO:

    # ...

    def buscar_por_id(self, categoria_id: str) -> Categoria:
        cat = None
        client = self.__client.get_client()
        db = client.livraria
        resultado = db.categorias.find_one({'_id': ObjectId(categoria_id)})
        if (resultado):
            cat = Categoria(resultado['nome'])
            cat.id = resultado['_id']
        return cat
```

CategoriaService.mostrar_por_id

- Para finalizar também precisamos fazer a mesma coisa que fizemos no remover e pelos mesmos motivos.

```python
class CategoriaService:

    # ...

    def mostrar_por_id(self):
        print('\nCategoria por Id ... ')

        try:
            id = input('Digite o Id da categoria para buscar: ') # alterado
            cat = self.__categoria_dao.buscar_por_id(id)

            if (cat == None):
                print('Categoria não encontrada!')
            else:
                print(f'Id: {cat.id} | Categoria: {cat.nome}')
        except Exception as e:
            print(f'Erro ao exibir categoria! - {e}')
            return

        input('Pressione uma tecla para continuar ... ')
```