WebMAC

Com FastAPI e HTMX

Aula 7

SQLModel e Integração com FastAPI

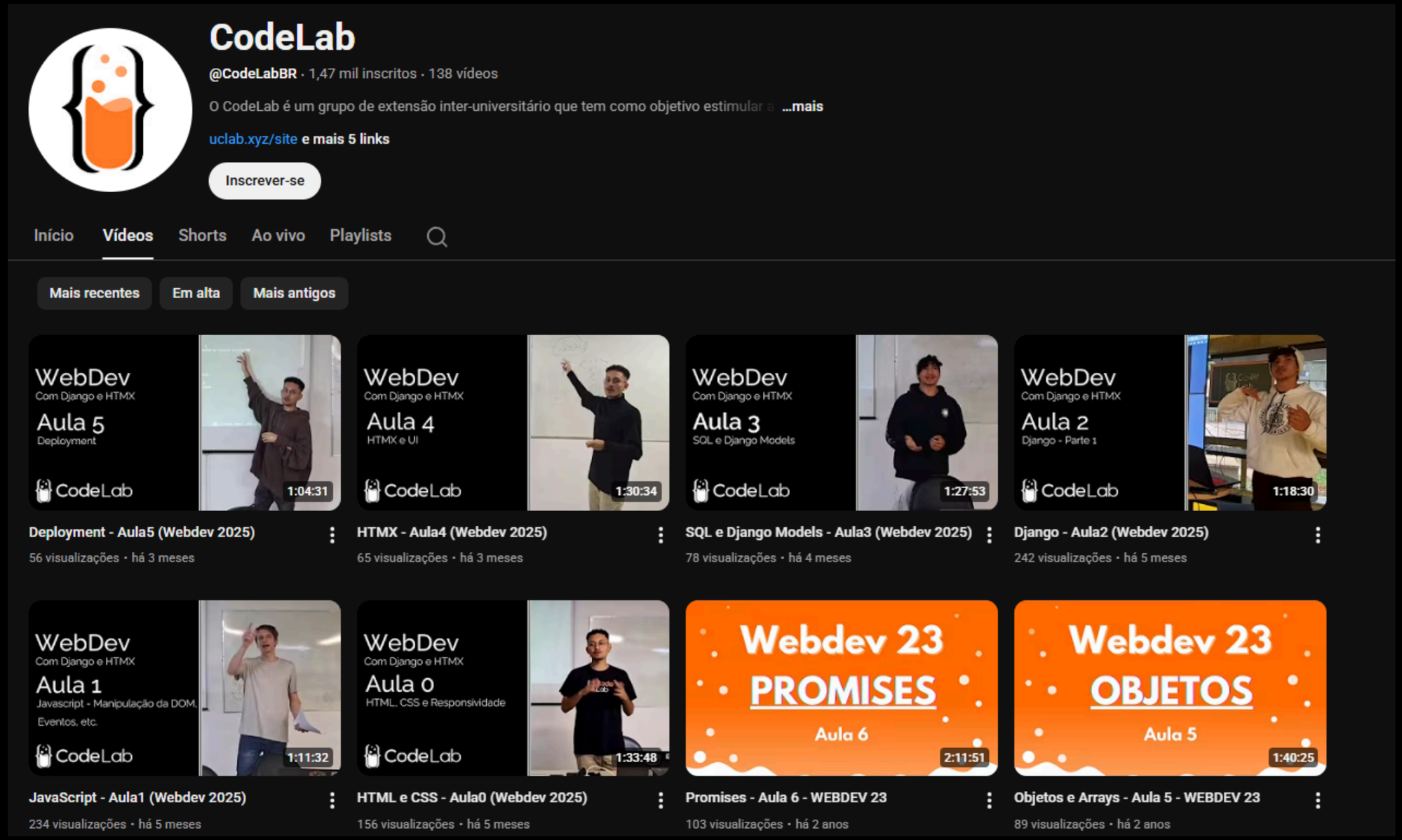
Semanas do Curso

1. HTML, CSS e Responsividade
2. Javascript
3. FastAPI
4. **SQL e SQLModel**
5. HTMX e UI
6. Docker

Objetivos das semanas

1. Criar páginas web
2. Tornar páginas dinâmicas
3. Criar um back-end
4. **Gerenciar dados**
5. Criar interfaces modernas
6. Rodar com containers

Canal do Codelab no Youtube



Resumo da Aula:

Tópicos:

- Introdução ao SQLModel;
- Revisão rápida sobre relações;
- Integrando SQLModel com FastAPI;

O que é o SQLModel?

É uma biblioteca em Python para abstrair o BD:

- Transforma entradas da tabela em objetos;
- Tem fácil integração com FastAPI;
- Simplifica a criação dos dados;
- Facilita o tratamento de erros nas operações.

Receitinha para criar um BD com SQLModel

Podemos simplificar isso em 3 etapas:

1. Criar os modelos de cada tipo de dado;
2. Inicializar a engine para acesso ao banco;
3. Sincronizar as tabelas com os modelos.

Exemplo Simples (Modelo)

```python
class Estado(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    nome: str
    sigla: str

    cidades: List["Cidade"] = Relationship(back_populates="estado")


class Cidade(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    nome: str
    estado_id: int = Field(foreign_key="estado.id")

    estado: Estado | None = Relationship(back_populates="cidades")
```

Qual o Tipo de relação entre Estado e Cidade?

Exemplo Simples (Criar engine)

```
arquivo_sqlite = "aula7.db"
url_sqlite = f"sqlite:///{arquivo_sqlite}"

engine = create_engine(url_sqlite)
```

Exemplo Simples (Sincronizar o BD)

```python
def create_db_and_tables():
    SQLModel.metadata.create_all(engine)

@app.on_event("startup")
def on_startup() -> None:
    create_db_and_tables()
```

Principais Atributos do SQLModel

- **Field:** É onde expressamos as restrições e propriedades da variável, como exemplos, temos:
  - **unique:** Que pode ser True/False.
  - **default:** Define um valor padrão para a entrada.
  - **primary_key:** Que também pode ser True/False.
  - **foreign_key:** indica o parâmetro que ele aponta.
- **Relationship:**
  - **back_populates:** Preenche dados por relação.
  - **link_model:** Estabelece relação intermediária.

Revisando Relações - **Relação 1:1**

```python
class Pessoa(SQLModel, table=True):
    nusp:int = Field(primary_key=True)
    nome: str
    idade: int

    dados_usp: "DadosUSP"       | None = Relationship(back_populates="usp_pessoa_ponteiro")
    dados_rf: "ReceitaFederal" | None = Relationship(back_populates="rf_pessoa_ponteiro")

class ReceitaFederal(SQLModel, table=True):
    cpf: str = Field(primary_key=True)
    valor_devendo_no_pix: float = Field(default=999.99)

    pessoa_id: int = Field(foreign_key="pessoa.nusp", unique=True)
    rf_pessoa_ponteiro: Pessoa | None = Relationship(back_populates="dados_rf")

class DadosUSP(SQLModel, table=True):
    id: int = Field(primary_key=True, foreign_key="pessoa.nusp")
    media: float
    reprovacoes: int
    usp_pessoa_ponteiro: Pessoa | None = Relationship(back_populates="dados_usp")
```

Revisando Relações - **Relação 1:n**

```python
class Estado(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    nome: str
    sigla: str

    cidades: List["Cidade"] = Relationship(back_populates="estado")


class Cidade(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    nome: str
    estado_id: int = Field(foreign_key="estado.id")

    estado: Estado | None = Relationship(back_populates="cidades")
```

Revisando Relações - **Relação n:m**

```python
class Matricula(SQLModel, table=True):
    aluno_id: int = Field(foreign_key="aluno.id", primary_key=True)
    disciplina_id: int = Field(foreign_key="disciplina.id", primary_key=True)

class Aluno(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    nome: str

    disciplinas: List["Disciplina"] = Relationship(back_populates="alunos", link_model=Matricula)


class Disciplina(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    nome: str

    alunos: List["Aluno"] = Relationship(back_populates="disciplinas", link_model=Matricula)
```

Usando as Relações - **Relação 1:1**

```
pessoa = Pessoa(nusp=1, nome="Frieren", idade=1000)

dados = DadosUSP(media=math.inf, reprovacoes=0)

pessoa.dados_usp = dados

session.add(pessoa)
session.commit()
```

Usando as Relações - **Relação 1:1**

```python
pessoa = Pessoa(nusp=201852361, nome="Claudin do Motocross", idade=15)
session.add(pessoa)
session.commit()

receita = ReceitaFederal(
    cpf="123.456.789-00",
    valor_devendo_no_pix=8000.00,
    pessoa_id=pessoa.nusp
)

session.add(receita)
session.commit()
```

Usando as Relações - **Relação n:m**

```
disciplina = Disciplina(nome="WebMAC")
session.add(disciplina)

aluno = Aluno(nome="Agnés Claudel")
session.add(aluno)
session.commit()

matricula = Matricula(
    aluno_id=aluno.id,
    disciplina_id=disciplina.id
)
session.add(matricula)

session.commit()
```

SQLModel

+

FastAPI

Exemplo (Integração com FastAPI)

Criação de Dados:

```python
@app.post("/estados")
def criar_estado(estado: Estado):
    with Session(engine) as session:
        session.add(estado)
        session.commit()
        session.refresh(estado)
        return estado
```

Exemplo (Integração com FastAPI)

Obtenção de Dados:

```python
@app.get("/estados")
def estado_por_nome(nome: str):
    with Session(engine) as session:
        query = select(Estado)
        query = query.where(Estado.nome == nome)
        return session.exec(query).first()

@app.get("/estados/{nome}")
def estado_por_nome(nome: str):
    with Session(engine) as session:
        query = select(Estado)
        query = query.where(Estado.nome == nome)
        return session.exec(query).first()
```

Consegue enxergar algum problema neste código?

Exemplo (Integração com FastAPI)

Obtenção de Dados (retornando listas):

```python
@app.get("/estados")
def lista_todos_os_estados():
    with Session(engine) as session:
        return session.exec(select(Estado)).all()
```

Exemplo (Integração com FastAPI)

Deleção de dados (por ID):

```python
@app.delete("/cidades/{cidade_id}")
def remover_cidade(cidade_id: int):
    with Session(engine) as session:
        cidade = session.get(Cidade, cidade_id)
        session.delete(cidade)
        session.commit()
        return {"ok": True}
```

Exemplo (Integração com FastAPI)

Deleção de dados (por qualquer outro parâmetro):

```python
@app.delete("/cidades")
def remover_cidade_por_nome(nome: str):
    with Session(engine) as session:
        query = select(Cidade).where(Cidade.nome == nome)
        cidade_a_ser_deletada = session.exec(query).first()
        session.delete(cidade_a_ser_deletada)
        session.commit()
        return {"ok": True}
```

Exemplo (Integração com FastAPI)

Atualização de Dados:

```python
@app.patch("/estados")
def atualizar_estado(nome_atual: str, nome: str | None = None, sigla: str | None = None):
    with Session(engine) as session:
        estado = session.exec(select(Estado).where(Estado.nome == nome_atual)).first()

        if nome is not None:  estado.nome = nome
        if sigla is not None: estado.sigla = sigla

        session.add(estado)
        session.commit()
        session.refresh(estado)
        return estado
```

Exercício CEC

Complete o Modelo já iniciado na página do WebMAC e desenvolva a API para manipular o banco de dados.

A API deverá permitir

- Listagem de Todas os alunos do sistema
- Listagem de todas as tarefas de um Aluno específico
- Criação de Novos alunos

Use os modelos e funções já implementadas como referência.