EduCollab

Maria Eduarda Romana
Nicolas Romano
Rafael Sanches

Objetivo

EduCollab é uma plataforma web desenvolvida para conectar professores e alunos em um ambiente de aprendizado colaborativo.

O objetivo do projeto é fornecer ferramentas para que professores possam criar e gerenciar conteúdo educacional, como bancos de questões e provas, enquanto os alunos podem utilizar esses recursos para estudar e testar seus conhecimentos.

Modelagem do Banco de Dados

models.py

```python
from typing import Optional, List
from datetime import datetime
from sqlmodel import SQLModel, Field, Relationship, Column, JSON
from enum import Enum

# --- ENUMS ---
class AnswerOptions(str, Enum):
    A = "A"
    B = "B"
    C = "C"
    D = "D"

# --- USUÁRIOS ---
class UserBase(SQLModel):
    name: str
    phone: str
    area: Optional[str] = None
    level: Optional[str] = None

class User(UserBase, table=True):
    # 'email' é chave primária lógica ou unique index
    email: str = Field(primary_key=True, index=True)
    hashed_password: str
    user_type: str  # 'student' ou 'teacher'

# --- QUESTÕES ---
class QuestionBase(SQLModel):
    materia: str
    enunciado: str
    # SQLite não tem array nativo, usamos JSON type do SQLAlchemy
    alternativas: List[str] = Field(sa_column=Column(JSON))
    resposta: AnswerOptions
```

Integração back-BD

Autenticação e Segurança:
Persistência de usuários e verificação segura de credenciais via Banco de Dados. Busca otimizada com select, validação de hash de senha (Bcrypt) e geração de token JWT.

Arquivo: security.py

```python
@router.post("/token", response_model=Token)
async def login_for_access_token(
    form_data: OAuth2PasswordRequestForm = Depends(),
    session: Session = Depends(get_session)
):
    # Busca usuário no banco
    statement = select(User).where(User.email == form_data.username)
    user = session.exec(statement).first()

    if not user or not verify_password(form_data.password, user.hashed_password):
        raise HTTPException(
            status_code=401,
            detail="Email ou senha incorretos",
            headers={"WWW-Authenticate": "Bearer"},
        )

    access_token = create_access_token(
        data={"sub": user.email, "user_type": user.user_type},
        expires_delta=timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    )

    return {
        "access_token": access_token,
        "token_type": "bearer",
        "user_type": user.user_type
    }
```

Madu

Integração back-BD

Relacionamentos no Fórum:
Gerenciamento de chaves estrangeiras (Foreign Key) e integridade referencial entre Tópicos e Respostas. Criar uma resposta vinculada a um tópico existente e recuperar esses dados aninhados.

Arquivo: forum.py

```python
@router.post("/topics/{topic_id}/replies", response_model=ReplyDisplay)
async def add_reply_to_topic(
    topic_id: str,
    reply_data: ReplyBase,
    current_user: users.User = Depends(users.get_current_user),
    session: Session = Depends(get_session)
):
    context = 'teacher' if current_user.user_type == 'teacher' else 'student'
    statement = select(Topic).where(Topic.id == topic_id, Topic.context_type == con
    topic = session.exec(statement).first()

    if not topic:
        raise HTTPException(status_code=404, detail="Tópico não encontrado")

    autor_nome = current_user.name

    new_reply = Reply(
        id=f"resp-{int(datetime.now().timestamp())}",
        topic_id=topic_id,
        texto=reply_data.texto,
        autor_nome=autor_nome,
        data=datetime.now()
    )

    session.add(new_reply)
    session.commit()
```

Nicolas

Integração back-BD

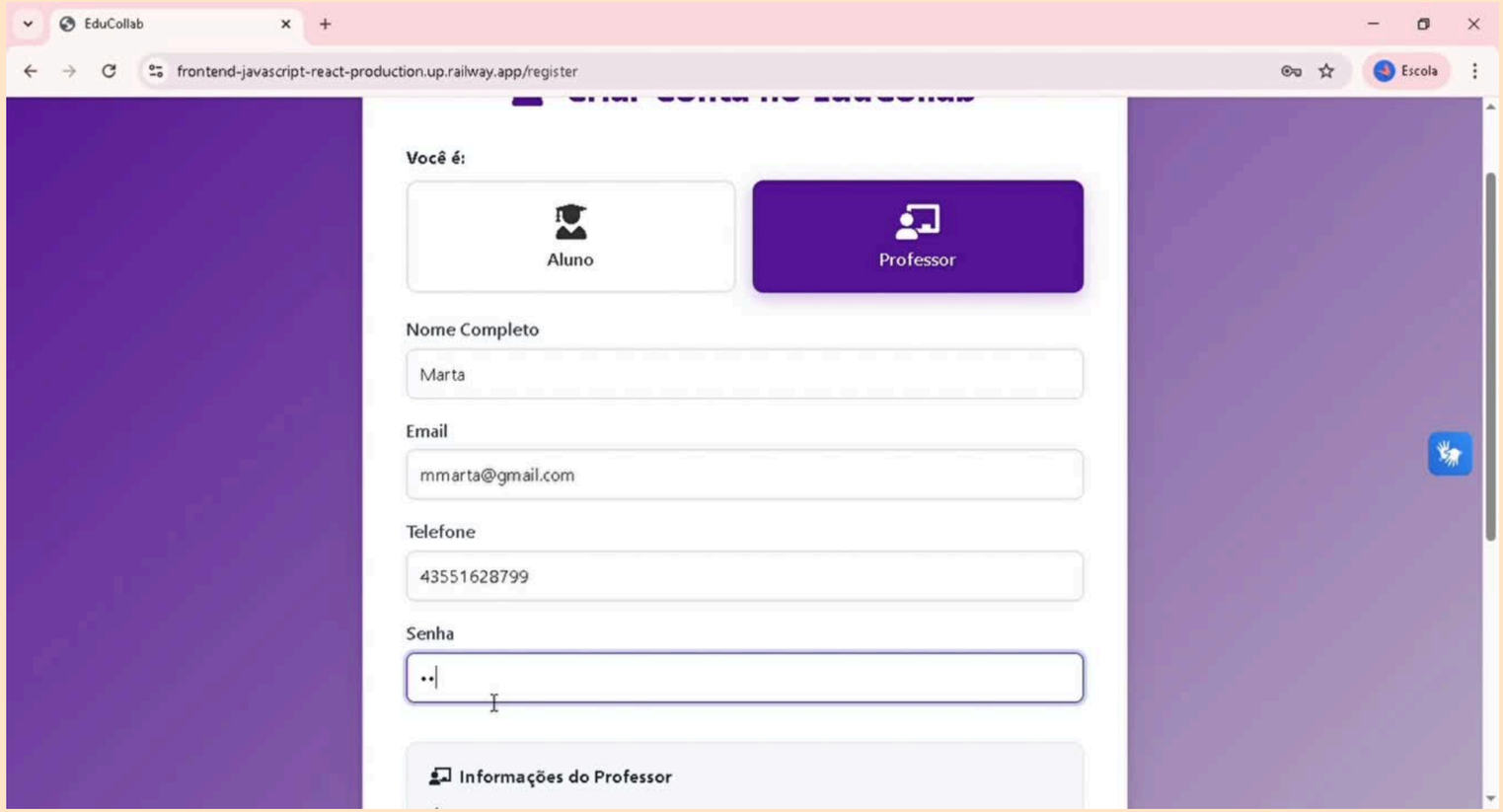
Persistência de Estruturas Complexas: Armazenamento de listas em um banco relacional que espera tipos primitivos. Uso de tipos avançados do SQLAlchemy (JSON) dentro do SQLModel para permitir que uma coluna armazene uma lista ["A", "B", "C"].

Arquivo: questions.py

```python
@router.post("/question", response_model=QuestionDisplay, status_code=status.HTTP_2
async def create_question(
    question_data: QuestionCreate,
    current_user: users.User = Depends(users.get_current_user),
    session: Session = Depends(get_session)
):
    if current_user.user_type != 'teacher':
        raise HTTPException(status_code=404, detail="Apenas professores podem criar

    # SQLModel lida com o mapeamento para JSON automaticamente via models.py
    new_question = Question.model_validate(question_data)

    session.add(new_question)
    session.commit()
    session.refresh(new_question)

    return new_question

@router.delete("/question/{question_id}", response_model=QuestionDisplay)
async def delete_question(
    question_id: int,
    current_user: users.User = Depends(users.get_current_user),
    session: Session = Depends(get_session)
):
    if current_user.user_type != 'teacher':
        raise HTTPException(status_code=404, detail="Apenas professores podem delet
```

Rafael

Desmonstração



vídeo completo

OBRIGADO !

GitHub do projeto