STRESS AI

Guilherme Almeida, Lucas Castilho, Matheus Hirata

Desenvolvimento Web 2025.2 - UTFPR Campus Apucarana

OBJETIVOS

Desenvolver uma aplicação web capaz de avaliar o nível de estresse de estudantes por meio de um questionário interativo e fornecer recomendações personalizadas com o auxílio de inteligência artificial, promovendo autoconhecimento e apoio à saúde mental.

INTEGRAÇÃO COM API EXTERNA

Integração com a API Gemini para gerar recomendações personalizadas com base nas respostas do usuário.

```python
def generate_recommendations(percent: int, level: str, dims: list[dict]) -> list[dict]:
    prompt = f"""
    Você é um assistente que gera recomendações breves de bem-estar para estudantes.
    Avalie o seguinte resultado de teste de estresse:
    - Índice Geral: {percent}/100
    - Nível de Estresse: {level}
    - Dimensões:
    {dims}

    Gere de 3 recomendações práticas, curtas e em português focadas nos pontos onde o usuário está com o nível mais alto de estresse.
    Cada recomendação deve ter uma "tag" (palavra-chave) e um "text" (descrição).
    Retorne as recomendações no formato JSON, como no exemplo:
    [
      {{ "tag": "Sono", "text": "Durma 7–9h por noite e evite telas antes de dormir." }},
      {{ "tag": "Pausas", "text": "Faça pequenas pausas de 5 minutos a cada hora de estudo." }}
    ]

    Certifique-se de que o JSON seja válido e bem formatado.
    """

    # Gera o conteúdo usando o modelo Gemini
    response = model.generate_content(prompt)
```

ENDPOINTS

PUT /users/me/email – Atualização de E-mail com Validação

Permite ao usuário autenticado alterar seu e-mail, confirmando a senha atual.

O que faz:

- Verifica se a senha atual está correta.
- Impede troca para um e-mail já cadastrado.
- Atualiza o índice de e-mails (DB_EMAIL_INDEX) e o registro principal (DB_USERS).
- Retorna os novos dados públicos do usuário.

```python
@router.put("/me/email", response_model=UserBase)
def update_email(payload: EmailUpdate, current_user: Annotated[dict, Depends(get_current_active_user)]):
    if not pwd_hasher.verify(payload.current_password, current_user["hashed_password"]):
        raise HTTPException(status_code=400, detail="Senha atual incorreta")

    existing = get_user_by_email(payload.email)
    if existing and existing["id"] != current_user["id"]:
        raise HTTPException(status_code=400, detail="E-mail já em uso")

    old_email = current_user["email"]
    if old_email != payload.email:
        if old_email in DB_EMAIL_INDEX:
            del DB_EMAIL_INDEX[old_email]
        DB_EMAIL_INDEX[payload.email] = current_user["id"]
        current_user["email"] = payload.email
        DB_USERS[current_user["id"]] = current_user

    return UserBase(**{k: current_user[k] for k in ["id", "name", "email", "disabled", "scopes"]})
```

ENDPOINTS

POST /auth/token – Login com JWT

Realiza a autenticação do usuário e retorna um token de acesso.

O que faz:

- Recebe username e password via OAuth2.
- Verifica as credenciais e gera um JWT.
- Inclui no token os escopos de acesso concedidos ao usuário.
- O token é usado para autenticar rotas protegidas (/users, /assessments).

```python
@router.post("/token", response_model=Token)
def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user = verify_credentials(form_data.username, form_data.password)
    if not user:
        raise HTTPException(status_code=401, detail="Credenciais inválidas")

    granted_scopes = grant_scopes(user, getattr(form_data, "scopes", []))
    access_token = create_access_token(subject=user["id"], scopes=granted_scopes)
    return {"access_token": access_token, "token_type": "bearer"}
```

ENDPOINTS

POST /assessments - Criação de Recomendações com Gemini AI

Realiza o cálculo do resultado e gera recomendações automáticas com base nas respostas do usuário, utilizando a integração com Gemini AI.

1. Recebe respostas (AssessmentIn)
2. Identifica o usuário autenticado
3. processo com compute_assessment()
    a. Percentual
    b. Nível do estresse
    c. Dimensões avaliadas
4. Gera recomendações via Gemini AI
5. Armazena o resultado no banco (DB_ASSESSMENTS)

```python
@router.post("", response_model=AssessmentOut)
def create_assessment(
    payload: AssessmentIn,
    current_user: Annotated[dict, Depends(get_current_active_user)],
):
    user_id = current_user["id"]
    percent, level, dims = compute_assessment(payload.answers)

    rec = {
        "id": str(uuid.uuid4()),
        "created_at": datetime.utcnow(),
        "percent": percent,
        "level": level,
        "dims": [d.model_dump() for d in dims],
    }

    # Gera recomendações usando o Gemini AI
    rec["recommendations"] = generate_recommendations(percent, level, rec["dims"])

    DB_ASSESSMENTS.setdefault(user_id, []).append(rec)
    return AssessmentOut(**rec)
```

Resultado Esperado

```
{
  "id": "...",
  "created_at": "...",
  "percent": 85,
  "level": "Intermediário",
  "dims": [...],
  "recommendations": [...]
}
```