Код проекта IOT

## Файл: app\database.py

# app/database.py

from sqlalchemy.ext.asyncio import AsyncSession, create\_async\_engine

from app.models import Base

DATABASE\_URL = "postgresql+asyncpg://postgres:postgres@localhost:5432/monitoring"

# DATABASE\_URL = "postgresql+asyncpg://mon\_admin@127.0.0.1/monitoring"

engine = create\_async\_engine(DATABASE\_URL, echo=True)

## Файл: app\dependencies.py

# app/dependencies

from sqlalchemy.ext.asyncio import AsyncSession # , create\_async\_engine

from sqlalchemy.orm import sessionmaker

from app.database import engine

def get\_db():

"""Генератор, предоставляющий асинхронную сессию базы данных."""

SessionLocal = sessionmaker(engine, class\_=AsyncSession, expire\_on\_commit=False)

db = SessionLocal()

try:

yield db

finally:

db.close()

## Файл: app\main.py

# app/main.py

from datetime import datetime

from fastapi import FastAPI, Depends, Form, HTTPException, Request

from pydantic import BaseModel

from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy import text

from fastapi.responses import RedirectResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

from starlette.middleware.sessions import SessionMiddleware

from werkzeug.security import check\_password\_hash

from app.database import engine

from app.dependencies import get\_db

app = FastAPI()

app.add\_middleware(SessionMiddleware, secret\_key="your\_secret\_key")

templates = Jinja2Templates(directory="templates")

app.mount("/static", StaticFiles(directory="static"), name="static")

class DowntimeUpdateRequest(BaseModel):

"""Модель для обновления информации о простое оборудования."""

answer\_id: int

@app.get("/")

async def welcome(request: Request):

"""Отображает приветственную страницу с ссылкой на выбор группы."""

return templates.TemplateResponse("welcome.html", {"request": request})

@app.get("/select-group")

async def select\_group(request: Request, db: AsyncSession = Depends(get\_db)):

"""Возвращает список доступных групп пользователей."""

query = text("SELECT group\_id, group\_name FROM groups") # Убедитесь, что поля называются именно так в вашей БД

async with db.begin():

result = await db.execute(query)

groups = result.all()

return templates.TemplateResponse("select\_group.html", {"request": request, "groups": groups})

@app.post("/set-group")

async def set\_group(request: Request, db: AsyncSession = Depends(get\_db)):

"""Устанавливает группу пользователя в сессии и перенаправляет на страницу выбора пользователя."""

form = await request.form()

group\_id = form.get("group\_id")

if not group\_id:

raise HTTPException(status\_code=400, detail="Не указан ID группы")

request.session['group\_id'] = int(group\_id)

return RedirectResponse(url="/select-user", status\_code=303)

@app.get("/select-user")

async def select\_user(request: Request, db: AsyncSession = Depends(get\_db)):

"""Возвращает страницу для выбора пользователя на основе выбранной группы."""

group\_id = request.session.get('group\_id')

query = text("SELECT users.\* FROM users JOIN users\_groups ON users.user\_id = users\_groups.user\_id WHERE users\_groups.group\_id = :group\_id")

async with db.begin():

result = await db.execute(query, {'group\_id': group\_id})

users = result.all()

return templates.TemplateResponse("select\_user.html", {"request": request, "users": users, "group\_id": group\_id})

@app.get("/login")

async def login\_form(request: Request):

"""Предоставляет форму входа."""

username = request.query\_params.get('username')

group\_id = request.session.get('group\_id')

return templates.TemplateResponse("login.html", {"request": request, "username": username, "group\_id": group\_id})

@app.post("/login")

async def login(request: Request, username: str = Form(...), password: str = Form(...), group\_id: int = Form(...), db: AsyncSession = Depends(get\_db)):

"""Аутентификация пользователя и установка сессии после успешного входа."""

# SQL запрос для получения информации о пользователе и его пароле

query = text("""

SELECT users.user\_id, users.user\_password

FROM users

JOIN users\_groups ON users.user\_id = users\_groups.user\_id

WHERE users.user\_name = :username AND users\_groups.group\_id = :group\_id

""")

async with db.begin():

result = await db.execute(query, {"username": username, "group\_id": group\_id})

user = result.first()

# Проверяем, существует ли пользователь и верный ли пароль

if not user or not check\_password\_hash(user.user\_password, password):

error\_message = "Неверное имя пользователя или пароль"

return templates.TemplateResponse(

"login.html", {

"request": request, "error\_message": error\_message,

"username": username, "group\_id": group\_id

}

)

# raise HTTPException(status\_code=401, detail="Неверное имя пользователя или пароль")

# Установка user\_id и group\_id в сессию

request.session['user\_id'] = user.user\_id

request.session['group\_id'] = group\_id

# Перенаправление пользователя на страницу панели управления

return RedirectResponse(url=f"/dashboard/{group\_id}", status\_code=303)

@app.get("/logout")

async def logout(request: Request):

"""Выход пользователя и очистка сессии."""

request.session.clear()

return RedirectResponse(url='/', status\_code=303)

@app.get("/dashboard/{group\_id}")

async def dashboard(request: Request, group\_id: int, db: AsyncSession = Depends(get\_db)):

"""Отображает панель управления, показывая все оборудование, связанное с выбранной группой."""

query = text("SELECT \* FROM equipment WHERE group\_id = :group\_id")

async with db.begin():

result = await db.execute(query, {'group\_id': group\_id})

equipments = result.fetchall()

return templates.TemplateResponse("dashboard.html", {"request": request, "equipments": equipments, "group\_id": group\_id})

@app.get("/equipment/{group\_id}")

async def get\_equipment(group\_id: int, db: AsyncSession = Depends(get\_db)):

"""Возвращает список оборудования вместе с их текущим статусом в выбранной группе."""

equipment\_query = text("""

SELECT e.equipment\_id, e.equipment\_name, COALESCE(a.active, FALSE) AS active

FROM equipment e

LEFT JOIN alerts\_subscription a ON e.equipment\_id = a.equipment\_id AND a.active = TRUE

WHERE e.group\_id = :group\_id

""")

async with db.begin():

result = await db.execute(equipment\_query, {'group\_id': group\_id})

equipments = result.all() # Используем метод all() для получения всех результатов

# Обработка результатов как списка кортежей

equipment\_list = [

{"id": eq[0], "name": eq[1], "active": eq[2]}

for eq in equipments

]

return equipment\_list

def get\_current\_user(request: Request):

"""Извлекает и возвращает ID текущего пользователя из сессии."""

user\_id = request.session.get('user\_id')

if not user\_id:

raise HTTPException(status\_code=400, detail="Пользователь не вошел в систему")

return user\_id

@app.post("/toggle-equipment/{equipment\_id}")

async def toggle\_equipment(equipment\_id: int, user\_id: str = Depends(get\_current\_user), db: AsyncSession = Depends(get\_db)):

"""Переключает статус активности оборудования для пользователя."""

async with db.begin():

# Проверяем текущий статус подписки

subscription\_query = text("""

SELECT id, active FROM alerts\_subscription

WHERE equipment\_id = :equipment\_id AND user\_id = :user\_id

ORDER BY subscribe\_time DESC

LIMIT 1

""")

subscription\_result = await db.execute(subscription\_query, {'equipment\_id': equipment\_id, 'user\_id': user\_id})

subscription = subscription\_result.fetchone()

if subscription:

# Получаем значения через RowProxy.\_mapping для безопасного доступа к данным как к словарю

subscription\_id = subscription.\_mapping['id']

current\_active = subscription.\_mapping['active']

if current\_active:

# Если активная, деактивируем

update\_query = text("""

UPDATE alerts\_subscription SET active = FALSE, unsubscribe\_time = timezone('utc', now())

WHERE id = :id

""")

await db.execute(update\_query, {'id': subscription\_id})

active\_status = False

else:

# Если неактивная, активируем

update\_query = text("""

UPDATE alerts\_subscription SET active = TRUE, subscribe\_time = timezone('utc', now())

WHERE id = :id

""")

await db.execute(update\_query, {'id': subscription\_id})

active\_status = True

else:

# Создаем новую подписку

insert\_query = text("""

INSERT INTO alerts\_subscription (equipment\_id, user\_id, active, subscribe\_time, minutes\_to\_live)

VALUES (:equipment\_id, :user\_id, TRUE, timezone('utc', now()), 480)

""")

await db.execute(insert\_query, {'equipment\_id': equipment\_id, 'user\_id': user\_id})

active\_status = True

await db.commit()

return {"status": "success", "active": active\_status, "equipment\_id": equipment\_id}

@app.get("/downtimes/{equipment\_id}")

async def get\_downtimes(equipment\_id: int, db: AsyncSession = Depends(get\_db)):

"""Получает список всех простоев для указанного оборудования."""

query = text("""

SELECT equipment\_id, start\_id, stop\_id, answer\_id

FROM workflow

WHERE equipment\_id = :equipment\_id

""")

async with db.begin():

result = await db.execute(query, {'equipment\_id': equipment\_id})

downtimes = result.all()

# Используем названия колонок в ответе, как они определены в таблице

return [

{'equipment\_id': dt.equipment\_id, 'start\_id': dt.start\_id, 'stop\_id': dt.stop\_id, 'answer\_id': dt.answer\_id}

for dt in downtimes

]

@app.post("/update-downtime/{equipment\_id}/{start\_id}")

async def update\_downtime(equipment\_id: int, start\_id: int, request: DowntimeUpdateRequest, db: AsyncSession = Depends(get\_db)):

"""Обновляет информацию о простое, связывая его с ответом оператора."""

query = text("""

UPDATE workflow

SET answer\_id = :answer\_id

WHERE equipment\_id = :equipment\_id AND start\_id = :start\_id

RETURNING equipment\_id, start\_id, stop\_id, answer\_id

""")

async with db.begin():

result = await db.execute(query, {'answer\_id': request.answer\_id, 'equipment\_id': equipment\_id, 'start\_id': start\_id})

downtime = result.mappings().first() # Используем mappings(), чтобы получить результат в виде словаря

if downtime:

# Возвращаем информацию об обновлённом простое

return {

"status": "success",

"message": "Простой обновлен",

"data": downtime # Отправляем данные как словарь

}

else:

# Если строка не была найдена или обновлена, возвращаем ошибку

raise HTTPException(status\_code=404, detail="Простой не найден")

@app.get("/answers")

async def get\_answers(db: AsyncSession = Depends(get\_db)):

"""Возвращает список всех доступных ответов для использования в системе."""

query = text("SELECT answer\_id, answer\_text FROM answers\_list")

async with db.begin():

result = await db.execute(query)

answers = result.all() # Получаем все строки, каждая строка будет в виде кортежа

# Преобразуем каждую строку (кортеж) в словарь

return [{'answer\_id': ans[0], 'answer\_text': ans[1]} for ans in answers]

## Файл: app\models.py

# app/models

# Файл носит справочный характер для настройки SQL запросов

from sqlalchemy import (

BigInteger, Boolean, CHAR, Column, DateTime,

Float, ForeignKey, Index, Integer, SmallInteger,

String, Table, Text, Time, text # noqa

)

from sqlalchemy.dialects.postgresql import OID, TIMESTAMP

from sqlalchemy.orm import relationship

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

metadata = Base.metadata

class Alert(Base):

\_\_tablename\_\_ = 'alerts'

\_\_table\_args\_\_ = (

Index('unique\_equipment\_user\_start', 'equipment\_id', 'start\_id', 'user\_id', unique=True),

)

id = Column(BigInteger, primary\_key=True, server\_default=text("nextval('alerts\_id\_seq'::regclass)"))

equipment\_id = Column(Integer, nullable=False)

start\_id = Column(BigInteger, nullable=False)

user\_id = Column(CHAR(32), nullable=False)

open\_time = Column(DateTime, nullable=False, server\_default=text("timezone('utc'::text, now())"))

close\_time = Column(DateTime)

answer\_id = Column(Integer)

alarm\_type = Column(Integer, nullable=False, server\_default=text("0"))

minutes\_to\_live = Column(Integer, server\_default=text("30"))

class AlertsSubscription(Base):

\_\_tablename\_\_ = 'alerts\_subscription'

id = Column(BigInteger, primary\_key=True, server\_default=text("nextval('alerts\_subscription\_id\_seq'::regclass)"))

equipment\_id = Column(Integer, nullable=False)

user\_id = Column(CHAR(32), nullable=False)

active = Column(Boolean, nullable=False, server\_default=text("true"))

subscribe\_time = Column(DateTime, nullable=False, server\_default=text("timezone('utc'::text, now())"))

unsubscribe\_time = Column(DateTime)

minutes\_to\_live = Column(Integer, nullable=False, server\_default=text("480"))

subscribe\_action = Column(Integer, server\_default=text("0"))

t\_all\_db\_volume = Table(

'all\_db\_volume', metadata,

Column('total', Text)

)

class AnswersCategory(Base):

\_\_tablename\_\_ = 'answers\_categories'

answer\_category = Column(Integer, primary\_key=True)

name = Column(Text)

t\_bad\_workflows = Table(

'bad\_workflows', metadata,

Column('dt', DateTime(True)),

Column('equipment\_id', BigInteger),

Column('bad\_start\_id', BigInteger),

Column('bad\_stop\_id', BigInteger),

Column('duration1', BigInteger),

Column('start\_id', BigInteger),

Column('stop\_id', BigInteger),

Column('duration2', BigInteger)

)

class Equipment(Base):

\_\_tablename\_\_ = 'equipment'

equipment\_id = Column(Integer, primary\_key=True)

group\_id = Column(Integer)

equipment\_name = Column(String(200))

equipment\_status = Column(Integer, nullable=False, server\_default=text("0"))

plan\_val = Column(Float(53))

mac\_address = Column(String(50))

use\_align\_filter = Column(Boolean, server\_default=text("false"))

align\_filter\_secs = Column(BigInteger, server\_default=text("15"))

std\_window\_secs = Column(BigInteger, server\_default=text("5"))

sort\_order = Column(Integer, server\_default=text("0"))

t\_equipment\_and\_groups = Table(

'equipment\_and\_groups', metadata,

Column('equipment\_id', Integer),

Column('equipment\_name', String(200)),

Column('group\_id', Integer),

Column('group\_name', String(200)),

Column('channel\_id', Integer),

Column('channel\_alias', String),

Column('is\_active', Boolean),

Column('sens\_level', Float(53)),

Column('use\_std', Boolean),

Column('std\_level', Float),

Column('mac\_address', String(50))

)

class Group(Base):

\_\_tablename\_\_ = 'groups'

group\_id = Column(Integer, primary\_key=True)

parent\_id = Column(Integer)

group\_name = Column(String(200))

group\_status = Column(Integer, nullable=False, server\_default=text("1"))

class User(Base):

\_\_tablename\_\_ = 'users'

user\_id = Column(CHAR(32), primary\_key=True)

user\_name = Column(String(50))

user\_full\_name = Column(String(250))

user\_mail = Column(String(250))

user\_role = Column(Integer, nullable=False, server\_default=text("0"))

user\_auth\_type = Column(Integer, nullable=False, server\_default=text("0"))

user\_status = Column(Integer, nullable=False, server\_default=text("1"))

user\_password = Column(String(400))

salt = Column(String(400))

last\_device\_id = Column(String(400))

create\_time = Column(DateTime, server\_default=text("timezone('utc'::text, now())"))

update\_time = Column(DateTime, server\_default=text("timezone('utc'::text, now())"))

bad\_tries = Column(SmallInteger, nullable=False, server\_default=text("0"))

class Workflow(Base):

\_\_tablename\_\_ = 'workflow'

equipment\_id = Column(BigInteger, primary\_key=True, nullable=False)

start\_id = Column(BigInteger, primary\_key=True, nullable=False)

stop\_id = Column(BigInteger)

answer\_id = Column(Integer, server\_default=text("0"))

is\_alerted = Column(Boolean, server\_default=text("false"))

class AnswersList(Base):

\_\_tablename\_\_ = 'answers\_list'

answer\_id = Column(Integer, primary\_key=True)

answer\_text = Column(String(400), nullable=False)

answer\_action = Column(SmallInteger)

is\_system = Column(Boolean, server\_default=text("false"))

answer\_category = Column(ForeignKey('answers\_categories.answer\_category'), nullable=False, index=True, server\_default=text("1"))

answer\_color = Column(Text, nullable=False, server\_default=text("'#BDF4A8'::text"))

answers\_category = relationship('AnswersCategory')

class UsersGroup(Base):

\_\_tablename\_\_ = 'users\_groups'

user\_id = Column(ForeignKey('users.user\_id', ondelete='CASCADE', onupdate='CASCADE'), primary\_key=True, nullable=False)

group\_id = Column(ForeignKey('groups.group\_id', ondelete='CASCADE', onupdate='CASCADE'), primary\_key=True, nullable=False)

user\_role = Column(Integer, nullable=False, server\_default=text("0"))

group = relationship('Group')

user = relationship('User')

## Файл: app\\_\_init\_\_.py

# app/\_\_init\_\_.py

## Файл: templates\dashboard.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Панель управления</title>

<link rel="stylesheet" href="/static/css/style.css">

</head>

<body>

<div class="dashboard-container">

<h1>Панель управления</h1>

<a href="/logout" class="home-button">На главную</a>

<div id="equipment-list">

<!-- Список оборудования заполняется динамически через JavaScript -->

</div>

<script>

document.addEventListener("DOMContentLoaded", function() {

const groupId = "{{ group\_id }}"; // Используем переданный group\_id

fetch(`/equipment/${groupId}`)

.then(response => response.json())

.then(data => {

const list = document.getElementById('equipment-list');

data.forEach(equipment => {

const item = document.createElement('div');

item.className = 'equipment-item ' + (equipment.active ? 'active' : 'inactive');

item.innerHTML = `

<span>${equipment.name}</span>

<button data-equipment-id="${equipment.id}" class="toggle-equipment">Переключить</button>

<button data-equipment-id="${equipment.id}" class="downtime-button">Простои</button>

<div id="downtimes-${equipment.id}" class="downtime-container"></div>

`;

list.appendChild(item);

});

})

.catch(error => console.error('Error loading equipment:', error));

});

document.addEventListener('click', function(event) {

if (event.target.matches('.toggle-equipment')) {

toggleEquipment(event.target.getAttribute('data-equipment-id'));

}

if (event.target.matches('.downtime-button')) {

loadDowntimes(event.target.getAttribute('data-equipment-id'));

}

});

function toggleEquipment(equipmentId) {

fetch(`/toggle-equipment/${equipmentId}`, {

method: 'POST',

headers: {'Content-Type': 'application/json'}

})

.then(response => response.json())

.then(data => {

if (data.status === 'success') {

alert('Статус оборудования успешно переключен!');

location.reload(); // Перезагрузка страницы для обновления статусов

} else {

throw new Error(data.message || 'Не удалось переключить статус оборудования');

}

})

.catch(error => console.error('Error toggling equipment:', error));

}

function loadDowntimes(equipmentId) {

console.log(`Loading downtimes for equipment ID: ${equipmentId}`);

fetch(`/downtimes/${equipmentId}`)

.then(response => response.json())

.then(data => {

const container = document.getElementById(`downtimes-${equipmentId}`);

if (container) {

container.style.display = data.length > 0 ? 'block' : 'none';

container.innerHTML = data.map(downtime => `

<div class='downtime-entry' id='downtime-${downtime.equipment\_id}-${downtime.start\_id}'>

Start: ${downtime.start\_id}, Stop: ${downtime.stop\_id || 'Ongoing'}

<button data-equipment-id="${downtime.equipment\_id}" data-start-id="${downtime.start\_id}" class="update-reason">Выбрать причину простоя</button>

</div>

`).join('');

}

})

.catch(error => {

console.error('Error loading downtimes:', error);

if (container) {

container.style.display = 'none'; // Скрыть контейнер при ошибке

}

});

}

document.addEventListener('click', function(event) {

if (event.target.matches('.update-reason')) {

showAnswers(event.target.getAttribute('data-equipment-id'), event.target.getAttribute('data-start-id'));

}

});

function showAnswers(equipmentId, startId) {

const containerId = `downtime-${equipmentId}-${startId}`;

const container = document.getElementById(containerId);

if (!container) {

console.error('Container not found for ID:', containerId);

return;

}

// Очистка контейнера от предыдущего списка перед добавлением нового

container.innerHTML = '';

fetch(`/answers`)

.then(response => response.json())

.then(data => {

const selector = document.createElement('select');

selector.onchange = () => { // Добавление обработчика события на изменение выбора

updateDowntime(equipmentId, startId, selector.value);

};

data.forEach(answer => {

const option = document.createElement('option');

option.value = answer.answer\_id;

option.textContent = answer.answer\_text;

selector.appendChild(option);

});

container.appendChild(selector); // Добавление селектора в контейнер

})

.catch(error => {

console.error('Error fetching answers:', error);

});

}

function updateDowntime(equipmentId, startId, answerId) {

console.log("Original answerId:", answerId); // Для проверки исходного значения

const numericAnswerId = parseInt(answerId, 10);

console.log("Converted numericAnswerId:", numericAnswerId);

fetch(`/update-downtime/${equipmentId}/${startId}`, {

method: 'POST',

headers: {'Content-Type': 'application/json'},

body: JSON.stringify({ answer\_id: parseInt(answerId, 10) }) // Преобразование в число

})

.then(response => response.json())

.then(data => {

if (data.status === 'success') {

alert('Downtime updated successfully!');

} else {

throw new Error(data.message || 'Failed to update downtime');

}

})

.catch(error => console.error('Error updating downtime:', error));

}

let idleTime = 0;

function timerIncrement() {

idleTime += 1;

if (idleTime > 5) { // 5 минут бездействия

window.location.href = '/logout';

}

}

// Обнуление таймера при действиях пользователя

document.addEventListener('mousemove', resetTimer);

document.addEventListener('keypress', resetTimer);

function resetTimer() {

idleTime = 0;

}

setInterval(timerIncrement, 60000); // Ежеминутная проверка

</script>

</div>

</body>

</html>

## Файл: templates\login.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Вход в систему</title>

<link rel="stylesheet" href="/static/css/style.css">

<link rel="stylesheet" href="/static/css/index.css">

</head>

<body>

<div class="login-container">

<h2>Вход в систему</h2>

{% if error\_message %}

<div class="error-message">{{ error\_message }}</div>

{% endif %}

<form action="/login" method="post">

<input type="hidden" name="group\_id" value="{{group\_id}}" />

<div class="form-group">

<label for="username">Имя пользователя:</label>

<input type="text" id="username" name="username" value="{{ username }}" required>

</div>

<div class="form-group">

<label for="password">Пароль:</label>

<input type="password" id="password" name="password" required>

</div>

<div class="form-group">

<button type="submit">Вход</button>

</div>

</form>

</div>

<!-- Добавляем элемент для отображения клавиатуры -->

<div class="simple-keyboard">

<!-- Скрипт для работы клавиатуры -->

<script src="/static/index.js"></script>

<script>

const Keyboard = window.SimpleKeyboard.default;

let currentInput = null;

// Инициализация клавиатуры

const keyboard = new Keyboard({

onChange: input => {

if (currentInput) {

currentInput.value = input;

}

},

onKeyPress: button => onKeyPress(button),

inputPattern: {

// Включение всех символов

default: /./

},

layout: {

default: [

"1 2 3 4 5 6 7 8 9 0",

"q w e r t y u i o p",

"a s d f g h j k l",

"z x c v b n m",

"{shift} {space} {backspace}"

]

}

});

// Привязка клавиатуры к полям ввода

document.querySelector("#username").addEventListener("focus", () => {

currentInput = document.querySelector("#username");

keyboard.setInput(currentInput.value);

});

document.querySelector("#password").addEventListener("focus", () => {

currentInput = document.querySelector("#password");

keyboard.setInput(currentInput.value);

});

function onKeyPress(button) {

console.log("Button pressed", button);

}

</script>

</div>

</body>

</html>

## Файл: templates\select\_group.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Выбор группы</title>

<link rel="stylesheet" href="/static/css/style.css"> <!-- Убедитесь, что путь к CSS правильный -->

</head>

<body>

<div class="select-group-container">

<h1>Выберите группу</h1>

<form action="/set-group" method="post">

<label for="group\_id" hidden>Выберите группу</label> <!-- Доступность: добавлен скрытый label -->

<select name="group\_id" id="group\_id" required aria-label="Выбор группы">

{% for group in groups %}

<option value="{{ group.group\_id }}">{{ group.group\_name }}</option>

{% endfor %}

</select>

<button type="submit">Продолжить</button>

</form>

</div>

</body>

</html>

## Файл: templates\select\_user.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Выбор пользователя</title>

<link rel="stylesheet" href="/static/css/style.css">

</head>

<body>

<div class="user-select-container">

<h1>Выберите пользователя</h1>

<form action="/login" method="get">

<select name="username" required>

{% for user in users %}

<option value="{{ user.user\_name }}">{{ user.user\_name }}</option>

{% endfor %}

</select>

<button type="submit">Продолжить</button>

</form>

</div>

</body>

</html>

## Файл: templates\welcome.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Приветствие</title>

<link rel="stylesheet" href="/static/css/style.css">

</head>

<body>

<div class="container">

<h1>Привет!</h1>

<p>Добро пожаловать в наше приложение. Пожалуйста, начните с выбора вашей группы.</p>

<a href="/select-group"><button>Выбрать группу</button></a>

</div>

</body>

</html>