Код проекта IOT

## Файл: app\database.py

# app/database.py

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

from app.logging\_config import logger

# 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"

# logger.info("Создание движка базы данных с URL: %s", DATABASE\_URL)

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

from app.logging\_config import logger

async def get\_db():

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

#logger.info("Создание новой сессии базы данных")

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

db = SessionLocal()

try:

yield db

finally:

await db.close()

logger.info("Сессия базы данных закрыта")

## Файл: app\logging\_config.py

# app/logging\_config.py

import logging

from logging.handlers import RotatingFileHandler

# Настройка логгера

log\_formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")

# Настройка файла логов

LOG\_FILE = "app.log"

file\_handler = RotatingFileHandler(LOG\_FILE, maxBytes=5\*1024\*1024, backupCount=5)

file\_handler.setFormatter(log\_formatter)

# Настройка консольного вывода логов

console\_handler = logging.StreamHandler()

console\_handler.setFormatter(log\_formatter)

# Общий логгер

logger = logging.getLogger("app\_logger")

logger.setLevel(logging.INFO)

logger.addHandler(file\_handler)

logger.addHandler(console\_handler)

## Файл: app\main.py

# app/main.py

"""

Модуль `main.py` является основной точкой входа для приложения на базе FastAPI.

Функции и маршруты, определенные в этом модуле, предоставляют интерфейс для работы с пользователями,

оборудованием и простоями на производстве. Модуль включает аутентификацию пользователей, выбор

производственного цеха, работу с сессиями и отображение информации через HTML-шаблоны.

Основные компоненты:

- FastAPI приложение и его конфигурация.

- Маршруты для обработки запросов пользователей, включая выбор группы (цеха), выбор пользователя и

аутентификацию.

- Функции для работы с базой данных через SQLAlchemy, включая асинхронные запросы и транзакции.

- Обработка сессий пользователей и хранение информации о текущем состоянии в сессии.

- Шаблоны Jinja2 для отображения страниц пользователям.

Модуль использует следующие ключевые библиотеки:

- FastAPI: Основной веб-фреймворк для создания RESTful API.

- SQLAlchemy: ORM для взаимодействия с базой данных.

- Jinja2: Шаблонизатор для рендеринга HTML-страниц.

- Starlette: Для работы с сессиями и статическими файлами.

- Pydantic: Для валидации данных, поступающих в запросах.

- Werkzeug: Для обработки хеширования паролей.

Общие шаги, которые выполняет модуль:

1. Инициализация приложения FastAPI с подключением middleware для сессий.

2. Обработка GET и POST запросов для различных страниц, таких как выбор группы, выбор пользователя,

панель управления, и т.д.

3. Взаимодействие с базой данных для извлечения и обновления данных

о пользователях, оборудовании и простоях.

4. Управление сессиями для хранения информации о текущем пользователе и выбранной группе.

"""

import os

from datetime import datetime, timezone, timedelta

from fastapi import (

FastAPI, Depends, Form, HTTPException,

Query, Request, status

)

from fastapi.responses import RedirectResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

from pydantic import BaseModel

from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy import text

from starlette.middleware.sessions import SessionMiddleware

from werkzeug.security import check\_password\_hash

# from app.database import engine

from app.dependencies import get\_db

from app.logging\_config import logger

app = FastAPI()

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

templates = Jinja2Templates(directory="templates")

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

KALININGRAD\_TZ = timezone(timedelta(hours=2))

USER\_ROLE = 1

PAGE\_SIZE = 4

PAGE = 1

class DowntimeUpdateRequest(BaseModel):

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

answer\_id: int

def convert\_timestamp(ts):

"""Конвертация времени из int"""

if ts:

dt = datetime.fromtimestamp(ts, tz=KALININGRAD\_TZ)

return dt.strftime("%H:%M:%S %d-%m-%Y")

return "Unknown"

def get\_default\_group\_id():

"""Функция для получения group\_id из переменной окружения"""

group\_id = os.getenv("GROUP\_ID")

if group\_id:

logger.info("Получен GROUP\_ID из переменной окружения: %s", group\_id)

else:

logger.warning("GROUP\_ID не установлен в переменных окружения")

return group\_id

@app.get("/")

async def welcome(request: Request):

"""

Отображает приветственную страницу или перенаправляет на выбор пользователя,

если group\_id установлен.

"""

logger.info("Отображение приветственной страницы")

group\_id = get\_default\_group\_id()

if group\_id:

# Если group\_id установлен, сохраняем его в сессии и перенаправляем на выбор пользователя

logger.info("Перенаправление на выбор пользователя с group\_id=%s", group\_id)

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

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

else:

# Если group\_id не установлен, отображаем приветственную страницу

logger.info("Отображение приветственной страницы без group\_id")

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

@app.get("/auto-set-group")

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

"""

Получаем group\_id из переменной окружения

и перенаправляем на выбор пользователя, если установлен

"""

logger.info("Автоматическая установка group\_id")

group\_id = get\_default\_group\_id()

if group\_id:

logger.info("group\_id найден в окружении %s:", group\_id)

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

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

else:

logger.warning("group\_id не найден в окружении")

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

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

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

"""

Возвращает список доступных групп пользователей или

перенаправляет на выбор пользователя, если group\_id уже установлен.

"""

logger.info("Получение списка доступных групп")

query = text("SELECT group\_id, group\_name FROM groups")

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:

logger.error("Не указан ID группы")

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

logger.info("Установка group\_id в сессии: %s", group\_id)

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')

logger.info("Выбор пользователя для group\_id: %s", group\_id)

# auto\_set = group\_id == int(os.getenv("GROUP\_ID", 0))

# Проверяем, был ли установлен group\_id из переменной окружения

# Получаем значение group\_id из переменной окружения как строку и преобразуем в int

env\_group\_id = os.getenv("GROUP\_ID")

if env\_group\_id is not None:

env\_group\_id = int(env\_group\_id)

# Проверяем, был ли установлен group\_id из переменной окружения

auto\_set = group\_id == env\_group\_id

query = text("""

SELECT users.user\_id, users.user\_name, users.user\_full\_name

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.fetchall()

user\_data = [

{"user\_name": user.user\_name, "user\_full\_name": user.user\_full\_name} for user in users

]

return templates.TemplateResponse(

"select\_user.html",

{"request": request, "users": user\_data, "group\_id": group\_id, "auto\_set": auto\_set}

)

@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')

logger.info("Отображение формы входа для пользователя %s в группе %s", username, 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."""

logger.info("Попытка входа пользователя %s в группу %s", username, group\_id)

# 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"

logger.warning(

"Ошибка аутентификации для пользователя %s: %s", username, error\_message

)

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

logger.info("Пользователь %s успешно аутентифицирован", username)

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

@app.get("/logout")

async def logout(request: Request):

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

logger.info("Пользователь выходит из системы")

request.session.clear()

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

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:

logger.error("Пользователь не вошел в систему")

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

return user\_id

@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."""

logger.info("Отображение панели управления для group\_id: %s", group\_id)

# Получаем текущего пользователя

user\_id = get\_current\_user(request)

if not user\_id:

raise HTTPException(status\_code=400, detail="Пользователь не найден")

# Получаем имя текущего пользователя

user\_query = text("SELECT user\_name FROM users WHERE user\_id = :user\_id")

# Получаем список оборудования

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

async with db.begin():

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

user\_name = user\_result.scalar()

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

equipments = result.fetchall()

return templates.TemplateResponse(

"dashboard.html",

{

"request": request,

"equipments": equipments,

"group\_id": group\_id,

"user\_name": user\_name, # Передаем имя пользователя в шаблон

"PAGE": PAGE,

"PAGE\_SIZE": PAGE\_SIZE

}

)

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

async def get\_equipment(

group\_id: int,

page: int = Query(PAGE, alias="page", ge=PAGE),

page\_size: int = Query(PAGE\_SIZE, alias="page\_size", ge=PAGE),

db: AsyncSession = Depends(get\_db)

):

"""

Возвращает список оборудования с постраничным выводом для выбранной группы.

"""

logger.info(

"Получение списка оборудования для group\_id: %s, страница: %s, размер страницы: %s",

group\_id,

page,

page\_size

)

offset = (page - 1) \* page\_size

# Запрос для получения общего количества записей

count\_query = text("""

SELECT COUNT(\*)

FROM equipment

WHERE group\_id = :group\_id

""")

async with db.begin():

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

total\_records = count\_result.scalar() # Получаем общее количество записей

total\_pages = (total\_records + page\_size - 1) // page\_size # Рассчитываем количество страниц

equipment\_query = text("""

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

FROM equipment e

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

LEFT JOIN users u ON a.user\_id = u.user\_id

WHERE e.group\_id = :group\_id

ORDER BY e.equipment\_id DESC

LIMIT :limit OFFSET :offset

""")

async with db.begin():

result = await db.execute(

equipment\_query,

{'group\_id': group\_id, 'limit': page\_size, 'offset': offset}

)

equipments = result.all()

logger.info(

"Список оборудования получен для group\_id: %s, количество записей: %s",

group\_id,

len(equipments)

)

equipment\_list = [

{"id": eq[0], "name": eq[1], "active": eq[2], "user\_name": eq[3]}

for eq in equipments

]

return {

"equipments": equipment\_list,

"current\_page": page,

"total\_pages": total\_pages

}

async def can\_toggle\_equipment(user\_id: str, equipment\_id: int, db: AsyncSession):

"""

Функция для проверки, может ли пользователь переключить статус оборудования.

"""

logger.info(

"Проверка прав пользователя user\_id: %s на переключение оборудования equipment\_id: %s",

user\_id,

equipment\_id

)

async with db.begin(): # Начинаем транзакцию для проверки прав

# Получаем роль текущего пользователя

user\_role\_query = text("""

SELECT user\_role FROM users

WHERE user\_id = :user\_id

""")

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

user\_role = user\_role\_result.scalar()

if user\_role is None:

logger.error("Пользователь user\_id: %s не найден", user\_id)

raise HTTPException(

status\_code=status.HTTP\_404\_NOT\_FOUND,

detail="Пользователь не найден."

)

# Проверяем, занят ли станок другим пользователем

occupation\_check\_query = text("""

SELECT a.id, a.user\_id, u.user\_name FROM alerts\_subscription a

JOIN users u ON a.user\_id = u.user\_id

WHERE a.equipment\_id = :equipment\_id AND a.active = TRUE

LIMIT 1

""")

occupation\_check\_result = await db.execute(

occupation\_check\_query, {'equipment\_id': equipment\_id}

)

occupation = occupation\_check\_result.fetchone()

# Если оборудование уже занято другим пользователем, проверяем роль текущего пользователя

if occupation:

occupied\_by\_user\_id = occupation.user\_id

occupied\_by\_user\_name = occupation.user\_name

logger.info(

"Оборудование занято пользователем %s (user\_id: %s)",

occupied\_by\_user\_name,

occupied\_by\_user\_id

)

# Проверяем, имеет ли текущий пользователь права на изменение статуса

if occupied\_by\_user\_id != user\_id and user\_role != USER\_ROLE:

logger.warning(

"Пользователь user\_id: %s не имеет прав на переключение оборудования",

user\_id

)

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail=f"Оборудование занято пользователем {occupied\_by\_user\_name}. У вас нет прав на его переключение."

)

return True

@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."""

logger.info(

"Переключение статуса оборудования equipment\_id: %s пользователем user\_id: %s",

equipment\_id,

user\_id

)

# Проверяем, может ли текущий пользователь переключить оборудование

await can\_toggle\_equipment(user\_id, equipment\_id, db)

# Теперь переключаем оборудование

async with db.begin(): # Начинаем транзакцию только для изменения статуса оборудования

# Проверяем, занят ли станок кем-либо

occupation\_check\_query = text("""

SELECT a.id, a.user\_id, u.user\_name FROM alerts\_subscription a

JOIN users u ON a.user\_id = u.user\_id

WHERE a.equipment\_id = :equipment\_id AND a.active = TRUE

LIMIT 1

""")

occupation\_check\_result = await db.execute(

occupation\_check\_query, {'equipment\_id': equipment\_id}

)

occupation = occupation\_check\_result.fetchone()

if occupation:

occupied\_by\_user\_id = occupation.user\_id

if occupied\_by\_user\_id == user\_id:

# Если текущий пользователь занял оборудование, освобождаем его

deactivate\_query = text("""

UPDATE alerts\_subscription

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

WHERE id = :subscription\_id

""")

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

active\_status = False

logger.info(

"Оборудование equipment\_id: %s освобождено пользователем user\_id: %s",

equipment\_id,

user\_id

)

else:

# Если текущий пользователь - мастер, освобождаем предыдущее занятие и занимаем оборудование

deactivate\_query = text("""

UPDATE alerts\_subscription

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

WHERE id = :subscription\_id

""")

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

activate\_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(activate\_query, {'equipment\_id': equipment\_id, 'user\_id': user\_id})

active\_status = True

logger.info(

"Оборудование equipment\_id: %s занято пользователем user\_id: %s",

equipment\_id,

user\_id

)

else:

# Если оборудование не занято, занимаем его текущим пользователем

activate\_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(activate\_query, {'equipment\_id': equipment\_id, 'user\_id': user\_id})

active\_status = True

logger.info(

"Оборудование equipment\_id: %s занято пользователем user\_id: %s",

equipment\_id,

user\_id

)

await db.commit() # Завершаем транзакцию

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

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

async def get\_downtimes(

equipment\_id: int,

page: int = Query(PAGE, alias="page", ge=PAGE),

page\_size: int = Query(PAGE\_SIZE, alias="page\_size", ge=PAGE),

db: AsyncSession = Depends(get\_db)

):

"""Получает список всех простоев для указанного оборудования с постраничным выводом."""

logger.info(

"Получение списка простоев для оборудования equipment\_id: %s, страница: %s, размер страницы: %s",

equipment\_id,

page,

page\_size

)

offset = (page - 1) \* page\_size

# Запрос для получения общего количества простоев

count\_query = text("""

SELECT COUNT(\*)

FROM workflow

WHERE equipment\_id = :equipment\_id

""")

async with db.begin():

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

total\_records = count\_result.scalar() # Получаем общее количество простоев

total\_pages = (total\_records + page\_size - 1) // page\_size # Рассчитываем количество страниц

query = text("""

SELECT w.equipment\_id, w.start\_id, w.stop\_id, w.answer\_id, al.answer\_text

FROM workflow w

LEFT JOIN answers\_list al ON w.answer\_id = al.answer\_id

WHERE w.equipment\_id = :equipment\_id

ORDER BY w.start\_id DESC

LIMIT :limit OFFSET :offset

""")

async with db.begin():

result = await db.execute(query, {

'equipment\_id': equipment\_id,

'limit': page\_size,

'offset': offset

})

downtimes = result.all()

logger.info(

"Список простоев получен для equipment\_id: %s, количество записей: %s",

equipment\_id,

len(downtimes)

)

return {

"downtimes": [

{

'equipment\_id': dt.equipment\_id,

'start\_id': dt.start\_id, # Сохраняем как int

'start\_time': convert\_timestamp(dt.start\_id), # Для отображения

'stop\_id': dt.stop\_id,

'stop\_time': convert\_timestamp(dt.stop\_id), # Для отображения

'answer\_id': dt.answer\_id,

'answer\_text': dt.answer\_text if dt.answer\_id else None

}

for dt in downtimes

],

"current\_page": page,

"total\_pages": total\_pages

}

@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."""

logger.info(

"Обновление простоя для оборудования equipment\_id: %s, start\_id: %s",

equipment\_id,

start\_id

)

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

logger.info(

"Простой обновлен для equipment\_id: %s, start\_id: %s", equipment\_id, start\_id

)

return {

"status": "success",

"message": "Простой обновлен",

"data": downtime # Отправляем данные как словарь

}

else:

# Если строка не была найдена или обновлена, возвращаем ошибку

logger.error(

"Простой не найден для equipment\_id: %s, start\_id: %s", equipment\_id, start\_id

)

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."""

logger.info("Получение списка всех доступных ответов")

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

logger.info("Получено %d ответов", len(answers))

# 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

"""

Модуль `models` содержит определения моделей SQLAlchemy для работы с базой данных.

Этот файл используется для настройки и работы с SQL-запросами, а также для описания

структуры таблиц и связей между ними в базе данных.

"""

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):

"""

Модель для таблицы `alerts`, представляющей собой оповещения по оборудованию.

Атрибуты:

id (BigInteger): Уникальный идентификатор оповещения.

equipment\_id (Integer): Идентификатор оборудования.

start\_id (BigInteger): Идентификатор начала оповещения.

user\_id (CHAR(32)): Идентификатор пользователя, связанного с оповещением.

open\_time (DateTime): Время открытия оповещения.

close\_time (DateTime): Время закрытия оповещения.

answer\_id (Integer): Идентификатор ответа.

alarm\_type (Integer): Тип тревоги.

minutes\_to\_live (Integer): Время жизни оповещения в минутах.

"""

\_\_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):

"""

Модель для таблицы `alerts\_subscription`, представляющей собой подписки на оповещения.

Атрибуты:

id (BigInteger): Уникальный идентификатор подписки.

equipment\_id (Integer): Идентификатор оборудования.

user\_id (CHAR(32)): Идентификатор пользователя, связанного с подпиской.

active (Boolean): Статус подписки (активная/неактивная).

subscribe\_time (DateTime): Время подписки.

unsubscribe\_time (DateTime): Время отписки.

minutes\_to\_live (Integer): Время жизни подписки в минутах.

subscribe\_action (Integer): Действие при подписке.

"""

\_\_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):

"""

Модель для таблицы `answers\_categories`, представляющей категории простоев.

Атрибуты:

answer\_category (Integer): Идентификатор категории.

name (Text): Название категории.

"""

\_\_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):

"""

Модель для таблицы `equipment`, представляющей оборудование.

Атрибуты:

equipment\_id (Integer): Уникальный идентификатор оборудования.

group\_id (Integer): Идентификатор группы (цеха).

equipment\_name (String): Название оборудования.

equipment\_status (Integer): Статус оборудования.

plan\_val (Float): Плановое значение.

mac\_address (String): MAC-адрес оборудования.

use\_align\_filter (Boolean): Использование фильтра выравнивания.

align\_filter\_secs (BigInteger): Время выравнивания в секундах.

std\_window\_secs (BigInteger): Время стандартного окна в секундах.

sort\_order (Integer): Порядок сортировки.

"""

\_\_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):

"""

Модель для таблицы `groups`, представляющей группы (цеха).

Атрибуты:

group\_id (Integer): Уникальный идентификатор группы.

parent\_id (Integer): Идентификатор родительской группы.

group\_name (String): Название группы.

group\_status (Integer): Статус группы.

"""

\_\_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):

"""

Модель для таблицы `users`, представляющей пользователей.

Атрибуты:

user\_id (CHAR(32)): Уникальный идентификатор пользователя.

user\_name (String): Имя пользователя.

user\_full\_name (String): Полное имя пользователя.

user\_mail (String): Электронная почта пользователя.

user\_role (Integer): Роль пользователя.

user\_auth\_type (Integer): Тип аутентификации пользователя.

user\_status (Integer): Статус пользователя.

user\_password (String): Хэш пароля пользователя.

salt (String): Соль для хэша пароля.

last\_device\_id (String): Идентификатор последнего устройства.

create\_time (DateTime): Время создания пользователя.

update\_time (DateTime): Время последнего обновления данных пользователя.

bad\_tries (SmallInteger): Количество неудачных попыток входа.

"""

\_\_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):

"""

Модель для таблицы `workflow`, представляющей рабочие процессы.

Атрибуты:

equipment\_id (BigInteger): Уникальный идентификатор оборудования.

start\_id (BigInteger): Идентификатор начала процесса.

stop\_id (BigInteger): Идентификатор завершения процесса.

answer\_id (Integer): Идентификатор ответа.

is\_alerted (Boolean): Признак того, было ли оповещение.

"""

\_\_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):

"""

Модель для таблицы `answers\_list`, представляющей список ответов.

Атрибуты:

answer\_id (Integer): Уникальный идентификатор ответа.

answer\_text (String): Текст ответа.

answer\_action (SmallInteger): Действие, связанное с ответом.

is\_system (Boolean): Признак системного ответа.

answer\_category (Integer): Идентификатор категории ответа.

answer\_color (Text): Цвет ответа.

"""

\_\_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):

"""

Модель для таблицы `users\_groups`, представляющей связь между пользователями и группами.

Атрибуты:

user\_id (CHAR(32)): Идентификатор пользователя.

group\_id (Integer): Идентификатор группы.

user\_role (Integer): Роль пользователя в группе.

"""

\_\_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">

<div id="info-message"></div>

</head>

<body>

<div class="dashboard-container">

<div class="user-info" style="text-align: right;">

<h2>Текущий пользователь: {{ user\_name }}</h2>

</div>

<h1>Панель управления</h1>

<div id="equipment-list">

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

</div>

<div id="error-message" style="color: red; display: none;"></div>

<script>

const PAGE = Number({{ PAGE }});

const PAGE\_SIZE = Number({{ PAGE\_SIZE }});

console.log("PAGE:", PAGE);

console.log("PAGE\_SIZE:", PAGE\_SIZE);

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

const groupId = "{{ group\_id }}";

loadEquipment(groupId, PAGE, PAGE\_SIZE);

});

function loadEquipment(groupId, page, pageSize) {

fetch(`/equipment/${groupId}?page=${page}&page\_size=${pageSize}`)

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

.then(data => {

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

list.innerHTML = ''; // Очистка списка перед добавлением новых элементов

if (data.equipments && Array.isArray(data.equipments)) {

data.equipments.forEach(equipment => {

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

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

const toggleButtonText = equipment.active ? 'Снять со смены' : 'Поставить на смену';

item.innerHTML = `

<div class="equipment-downtime-container">

<span class="equipment-name">${equipment.name}</span>

</div>

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

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

<span class="responsible-person">Табельный №: ${equipment.user\_name || 'Отсутствует'}</span>

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

`;

list.appendChild(item);

});

// Добавление элементов управления пагинацией

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

paginationControls.className = 'pagination-controls';

// Первая страница

if (data.current\_page > 1) {

paginationControls.innerHTML += `

<button onclick="loadEquipment(${groupId}, 1, ${pageSize})">Начало списка оборудования</button>

<button onclick="loadEquipment(${groupId}, ${data.current\_page - 1}, ${pageSize})"><<<</button>

`;

}

// Последняя страница

if (data.current\_page < data.total\_pages) {

paginationControls.innerHTML += `

<button onclick="loadEquipment(${groupId}, ${data.current\_page + 1}, ${pageSize})">>>></button>

<button onclick="loadEquipment(${groupId}, ${data.total\_pages}, ${pageSize})">Конец списка оборудования</button>

`;

}

// Если всего одна страница

if (data.current\_page === 1 && data.total\_pages === 1) {

paginationControls.innerHTML = '';

}

list.appendChild(paginationControls);

} else {

console.error('Некорректный формат данных:', data);

}

})

.catch(error => console.error('Ошибка загрузки оборудования:', 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')) {

const equipmentId = event.target.getAttribute('data-equipment-id');

toggleDowntimesVisibility(equipmentId);

}

});

function toggleDowntimesVisibility(equipmentId) {

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

if (container.style.display === 'block') {

container.style.display = 'none'; // Скрываем список, если он уже был показан

} else {

loadDowntimes(equipmentId); // Загружаем и показываем список, если он был скрыт

}

}

function toggleEquipment(equipmentId) {

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

method: 'POST',

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

})

.then(response => {

if (!response.ok) {

return response.json().then(data => {

throw new Error(data.detail || 'Произошла ошибка при переключении оборудования');

});

}

return response.json();

})

.then(data => {

const infoMessage = document.getElementById('info-message');

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

const button = document.querySelector(`.toggle-equipment[data-equipment-id="${equipmentId}"]`);

const equipmentItem = button.closest('.equipment-item');

// Переключение текста на кнопке и изменение класса

if (button.textContent.trim() === "Поставить на смену") {

button.textContent = "Снять со смены";

equipmentItem.classList.add('active');

equipmentItem.classList.remove('inactive');

} else {

button.textContent = "Поставить на смену";

equipmentItem.classList.add('inactive');

equipmentItem.classList.remove('active');

}

// Отображаем сообщение об успешном переключении

infoMessage.textContent = 'Статус оборудования успешно переключен!';

infoMessage.style.display = 'block';

// Скрываем сообщение через 3 секунды

setTimeout(() => {

infoMessage.style.display = 'none';

}, 1000);

} else if (data.status === 'error') {

// Отображаем сообщение об ошибке

infoMessage.textContent = `Ошибка: ${data.message}`;

infoMessage.style.display = 'block';

// Скрываем сообщение через 3 секунды

setTimeout(() => {

infoMessage.style.display = 'none';

}, 1000);

} else {

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

}

})

.catch(error => {

alert(`Ошибка: ${error.message}`);

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

});

}

function loadDowntimes(equipmentId, page = PAGE, pageSize = PAGE\_SIZE) {

fetch(`/downtimes/${equipmentId}?page=${page}&page\_size=${pageSize}`)

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

.then(data => {

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

if (container) {

container.innerHTML = ''; // Очистка контейнера перед добавлением новых элементов

if (data.downtimes && Array.isArray(data.downtimes)) {

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

data.downtimes.forEach(downtime => {

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

downtimeEntry.className = 'downtime-entry';

downtimeEntry.id = `downtime-${downtime.equipment\_id}-${downtime.start\_id}`;

downtimeEntry.innerHTML = `

<div>

<h3>Начало: ${downtime.start\_time || 'Неизвестно'}, Окончание простоя: ${downtime.stop\_time || 'Продолжается'}</h3>

<h3><strong>Причина простоя:</strong> ${downtime.answer\_text || 'Не указана'}</h3>

</div>

<div>

<button class="edit-downtime-btn" style="float: right;" onclick="editDowntime(${downtime.equipment\_id}, ${downtime.start\_id})">Изменить</button>

</div>

`;

container.appendChild(downtimeEntry);

});

// Добавление элементов управления пагинацией

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

paginationControls.className = 'pagination-controls';

// Первая страница

if (data.current\_page > 1) {

paginationControls.innerHTML += `

<button onclick="loadDowntimes(${equipmentId}, 1, ${pageSize})">Начало списка простоев</button>

<button onclick="loadDowntimes(${equipmentId}, ${data.current\_page - 1}, ${pageSize})"><<<</button>

`;

}

// Последняя страница

if (data.current\_page < data.total\_pages) {

paginationControls.innerHTML += `

<button onclick="loadDowntimes(${equipmentId}, ${data.current\_page + 1}, ${pageSize})">>>></button>

<button onclick="loadDowntimes(${equipmentId}, ${data.total\_pages}, ${pageSize})">Конец списка простоев</button>

`;

}

// Если всего одна страница

if (data.current\_page === 1 && data.total\_pages === 1) {

paginationControls.innerHTML = '';

}

container.appendChild(paginationControls);

} else {

console.error('Некорректный формат данных:', data);

}

}

})

.catch(error => console.error('Ошибка загрузки простоев:', error));

}

function editDowntime(equipmentId, startId) {

const downtimeEntry = document.getElementById(`downtime-${equipmentId}-${startId}`);

downtimeEntry.innerHTML = `

<div>Начало: ${downtimeEntry.querySelector('div').textContent}</div>

<label for="reason-${equipmentId}-${startId}">Выбрать причину простоя:</label>

<select id="reason-${equipmentId}-${startId}" data-equipment-id="${equipmentId}" data-start-id="${startId}">

<option value="">------</option>

</select>

<button class="save-downtime-btn" onclick="updateDowntime(${equipmentId}, ${startId}, document.getElementById('reason-${equipmentId}-${startId}').value)">Сохранить</button>

`;

showAnswers(equipmentId, startId);

}

function showAnswers(equipmentId, startId) {

const selector = document.getElementById(`reason-${equipmentId}-${startId}`);

if (!selector) {

console.error('Selector not found for ID:', `reason-${equipmentId}-${startId}`);

return;

}

fetch(`/answers`)

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

.then(data => {

data.forEach(answer => {

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

option.value = answer.answer\_id;

option.textContent = answer.answer\_text;

selector.appendChild(option);

});

})

.catch(error => {

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

});

}

function updateDowntime(equipmentId, startId, answerId) {

console.log("Original answerId:", answerId);

const numericAnswerId = parseInt(answerId, 10);

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

method: 'POST',

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

body: JSON.stringify({ answer\_id: numericAnswerId })

})

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

.then(data => {

const infoMessage = document.getElementById('info-message'); // Добавляем обработку сообщений

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

// Отображаем сообщение об успешном обновлении

infoMessage.textContent = 'Данные о простое обновлены!';

infoMessage.style.display = 'block';

// Скрываем сообщение через 3 секунды

setTimeout(() => {

infoMessage.style.display = 'none';

}, 1000);

loadDowntimes(equipmentId); // Перезагружаем список простоев

} else {

throw new Error(data.message || 'Данные о простое не обновились');

}

})

.catch(error => {

const infoMessage = document.getElementById('info-message'); // Обработка ошибки

// Отображаем сообщение об ошибке

infoMessage.textContent = `Ошибка: ${error.message}`;

infoMessage.style.display = 'block';

// Скрываем сообщение через 3 секунды

setTimeout(() => {

infoMessage.style.display = 'none';

}, 1000);

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

});

}

let idleTime = 0;

function timerIncrement() {

idleTime += 1;

if (idleTime > 5) {

window.location.href = '/logout';

}

}

document.addEventListener('mousemove', resetTimer);

document.addEventListener('keypress', resetTimer);

function resetTimer() {

idleTime = 0;

}

setInterval(timerIncrement, 60000);

</script>

</div>

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

</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"> <!-- для полной клавиатуры -->

</head>

<body>

<div class="login-container">

<h1>Вход в систему</h1>

<a href="/select-user" class="home-button">На предыдущую страницу</a>

{% 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}}" />

<table>

<tr>

<td><label for="username">Табельный номер:</label></td>

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

</tr>

<tr>

<td><label for="password">Пароль:</label></td>

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

</tr>

<tr>

<td>

<button type="submit" class="large-button">Вход</button>

</td>

</tr>

</table>

</form>

</div>

<!-- Таблица для цифровой клавиатуры -->

<table class="numeric-keyboard">

<tr>

<td><button type="button" class="key">1</button></td>

<td><button type="button" class="key">2</button></td>

<td><button type="button" class="key">3</button></td>

</tr>

<tr>

<td><button type="button" class="key">4</button></td>

<td><button type="button" class="key">5</button></td>

<td><button type="button" class="key">6</button></td>

</tr>

<tr>

<td><button type="button" class="key">7</button></td>

<td><button type="button" class="key">8</button></td>

<td><button type="button" class="key">9</button></td>

</tr>

<tr>

<td colspan="2"><button type="button" class="key">0</button></td>

<td><button type="button" class="key">X</button></td> <!-- X для удаления символов -->

</tr>

</table>

<script>

let currentInput = document.querySelector("#password"); // По умолчанию фокус на поле пароля

document.addEventListener("DOMContentLoaded", () => {

currentInput.focus(); // Устанавливаем фокус на поле пароля при загрузке страницы

});

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

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

});

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

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

});

document.querySelectorAll('.key').forEach(button => {

button.addEventListener('click', () => {

const keyValue = button.innerText;

if (currentInput) {

if (keyValue === 'X') {

currentInput.value = currentInput.value.slice(0, -1); // Удаление последнего символа

} else {

currentInput.value += keyValue; // Добавление символа в текущее поле

}

}

});

});

</script>

</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"> <!-- Подключение вашего основного CSS файла -->

</head>

<body>

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

<h1>Выберите цех</h1>

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

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

<div class="form-row">

<label for="group\_id">Цех:</label>

<select name="group\_id" id="group\_id" required>

<option value="" disabled selected>------</option>

{% for group in groups %}

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

{% endfor %}

</select>

</div>

<div class="form-row">

<button type="submit" class="large-button">Продолжить</button>

</div>

</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>

{% if not auto\_set %}

<a href="/select-group" class="home-button">На предыдущую страницу</a>

{% endif %}

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

<table>

<tr>

<td><label for="username">Табельный номер:</label></td>

<td>

<select name="username" id="username-select" required>

<!-- Опция с прочерком по умолчанию -->

<option value="" disabled selected>------</option>

{% for user in users %}

<option value="{{ user.user\_name }}" data-fullname="{{ user.user\_full\_name }}">{{ user.user\_name }}</option>

{% endfor %}

</select>

</td>

</tr>

<tr>

<td colspan="2">

<div id="fullname-display" style="font-size: 24px; margin-top: 10px;"></div>

</td>

</tr>

<tr>

<td>

<button type="submit" class="large-button">Продолжить</button>

</td>

</tr>

</table>

</form>

</div>

<script>

// Обработка изменения выбора пользователя

document.getElementById('username-select').addEventListener('change', function() {

const selectedOption = this.options[this.selectedIndex];

const fullName = selectedOption.getAttribute('data-fullname');

document.getElementById('fullname-display').innerText = fullName ? `Ф.И.О: ${fullName}` : '';

});

// Инициализация отображения полного имени при загрузке страницы

window.onload = function() {

const selectElement = document.getElementById('username-select');

const selectedOption = selectElement.options[selectElement.selectedIndex];

const fullName = selectedOption.getAttribute('data-fullname');

document.getElementById('fullname-display').innerText = fullName ? `Ф.И.О: ${fullName}` : '';

};

</script>

</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>Зравствуйте!</h1>

<h2>Добро пожаловать в наше приложение. Пожалуйста, начните с выбора вашего цеха.</h2>

<a href="/select-group" class="large-button">Выбрать цех</a>

</div>

</body>

</html>