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

async def get\_db():

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

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

db = SessionLocal()

try:

yield db

finally:

await db.close()

## Файл: app\main.py

# app/main.py

from datetime import datetime, timezone, timedelta

from fastapi import (

FastAPI, Depends, Form, HTTPException,

Query, Request, status

)

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")

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

USER\_ROLE = 1

PAGE\_SIZE = 5

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 / 1000, tz=KALININGRAD\_TZ)

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

return "Unknown"

@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")

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.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}

)

@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)

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.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."""

# 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, db: AsyncSession = Depends(get\_db)):

"""

Возвращает список оборудования вместе с их текущим статусом в выбранной группе,

включая пользователя.

"""

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

""")

async with db.begin():

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

equipments = result.all()

# Обработка результатов как списка словарей

equipment\_list = [

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

for eq in equipments

]

return equipment\_list

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

"""

Функция для проверки, может ли пользователь переключить статус оборудования.

"""

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:

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

# Проверяем, имеет ли текущий пользователь права на изменение статуса

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

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."""

# 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

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

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

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."""

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

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()

return [

{

'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

]

@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">

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

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

});

})

.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')) {

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 => {

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

alert('Статус оборудования успешно переключен!');

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

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

alert(`Ошибка: ${data.message}`);

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

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

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

<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>

${

downtime.answer\_id

? `<div>

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

</div>`

: `<div>

<label for="reason-${downtime.equipment\_id}-${downtime.start\_id}">Выбрать причину простоя:</label>

<select id="reason-${downtime.equipment\_id}-${downtime.start\_id}" data-equipment-id="${downtime.equipment\_id}" data-start-id="${downtime.start\_id}">

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

</select>

</div>`

}

</div>

`).join('');

data.forEach(downtime => {

if (!downtime.answer\_id) {

showAnswers(downtime.equipment\_id, downtime.start\_id);

}

});

container.innerHTML += `

<div class="pagination-controls">

<button onclick="loadDowntimes(${equipmentId}, ${page - 1}, ${pageSize})" ${page <= 1 ? 'disabled' : ''}><<<</button>

<button onclick="loadDowntimes(${equipmentId}, ${page + 1}, ${pageSize})">>>></button>

</div>

`;

}

})

.catch(error => {

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

if (container) {

container.style.display = 'none';

}

});

}

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 => {

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

alert('Данные о простое обновлены!');

loadDowntimes(equipmentId);

} else {

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

}

})

.catch(error => 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 = null;

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"> <!-- Убедитесь, что путь к CSS правильный -->

</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">

<table>

<tr>

<td><label for="group\_id">Цех:</label></td>

<td>

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

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

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

{% for group in groups %}

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

{% endfor %}

</select>

</td>

</tr>

<tr>

<td>

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

</td>

</tr>

</table>

</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>

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

<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>