**Содержание**

Введение

**Предметная область:** Система учета рабочего времени

1. Разработка технического задания

Приложение для учета рабочего времени на FastAPI с PostgreSQL базой данных для отслеживания задач и времени их выполнения. Программа позволяет создавать задачи с описанием, запускать и останавливать таймер для каждой задачи, автоматически рассчитывает продолжительность работы и сохраняет все данные в локальной базе. Интерфейс включает таблицу с полной информацией о задачах, индикатор текущей активной задачи и детальную статистику. Проект демонстрирует работу с графическим интерфейсом, базами данных PostgreSQL, таймерами и модульной архитектурой. Реализованы функции добавления, удаления задач, ведения статистики по дням, визуального оформления активных задач и экспорта данных о рабочем времени сотрудника в удобном формате.

2. Проектирование интерфейса приложения

Проектирование интерфейса приложения было реализовано в Figma.

**Figma** — это онлайн-платформа для совместного проектирования интерфейсов, прототипирования и совместной работы над дизайном. Она доступна через веб-интерфейс и поддерживает совместную работу в режиме реального времени, позволяя нескольким участникам одновременно редактировать проект и взаимодействовать друг с другом. Мы выбрали Figma для проектирования интерфейса нашего продукта по следующим причинам:

* **Совместная работа в реальном времени**: Возможность одновременной работы нескольких дизайнеров над одним проектом ускоряет процесс разработки и улучшает коммуникацию внутри команды.
* **Облачное хранение проектов**: Файлы хранятся в облаке, что упрощает доступ к ним с любого устройства и устраняет необходимость синхронизации версий проекта вручную.
* **Поддержка прототипирования**: Figma позволяет быстро создать интерактивные прототипы, имитирующие поведение будущего приложения, что полезно для тестирования юзабилити и демонстрации заказчику.
* **Простота интеграции с другими инструментами**: Поддерживаются плагины и интеграция с различными сервисами, такими как Zeplin, Marvel и др., что облегчает передачу дизайна разработчикам и тестирование готового продукта.

Таблица 1.1. Преимущества и недостатки использования Figma

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| * Простое обучение и интуитивно понятный интерфейс. * Высокая скорость работы благодаря облачной инфраструктуре. * Совместная работа в реальном времени. * Удобные инструменты для быстрого прототипирования и анимации. * Гибкость настройки компонентов и стилей. * Наличие обширной экосистемы плагинов и ресурсов. | * Необходимость постоянного подключения к Интернету, так как приложение работает исключительно в браузере. * Возможна задержка при работе с большими проектами из-за загрузки элементов интерфейса. * Ограниченные возможности экспорта некоторых сложных анимаций и эффектов. * Иногда возникают проблемы с отображением векторных изображений при импорте из сторонних инструментов. |

Реализация в Figma:

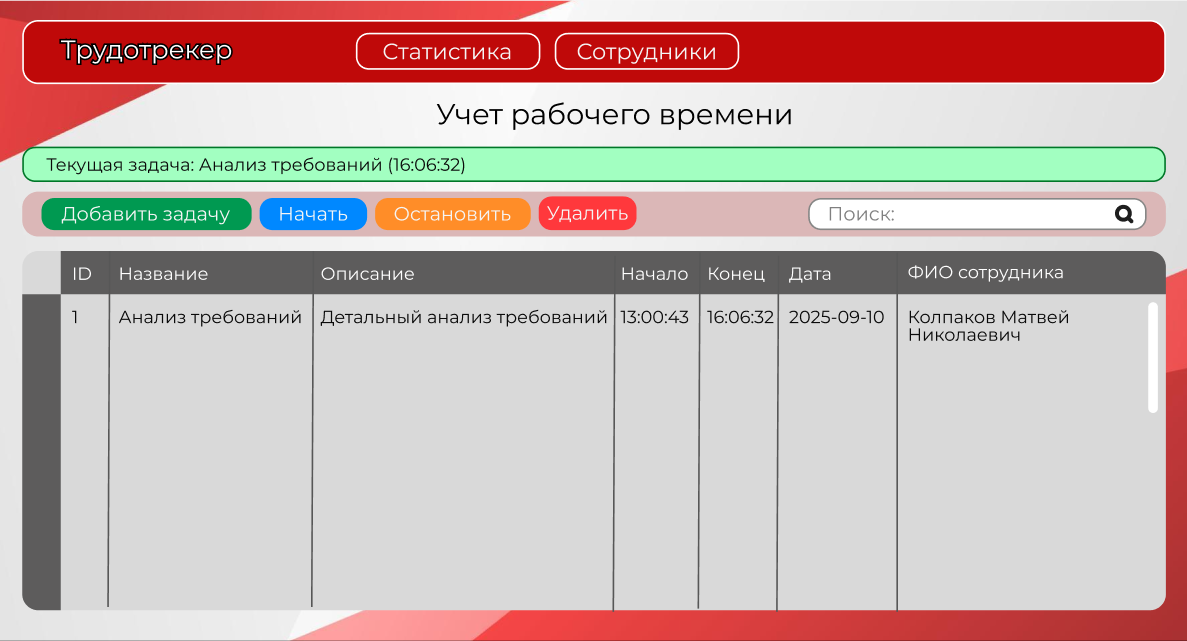


Рис. 1.1. Главная страница

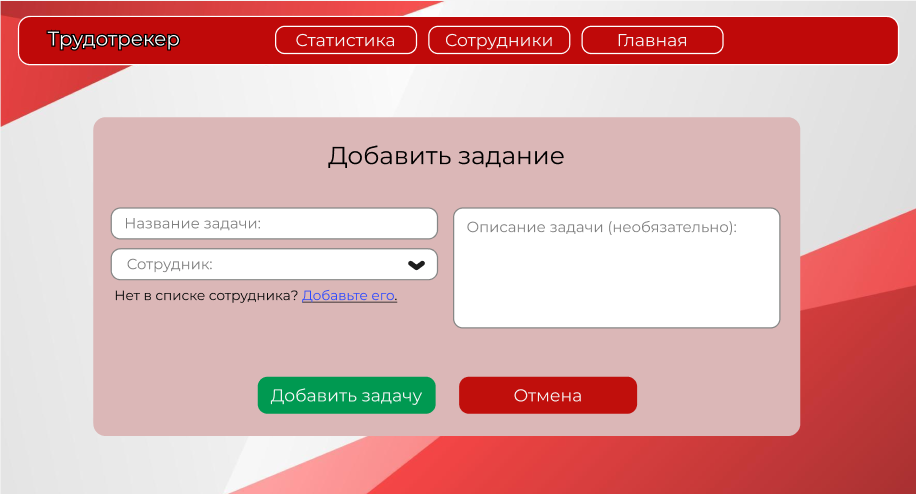


Рис. 1.2. Добавление задания

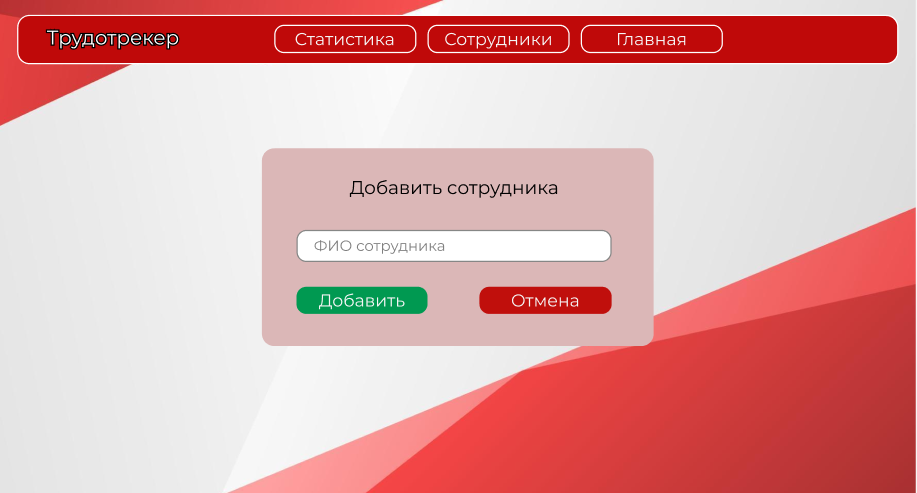


Рис. 1.3. Добавление сотрудника



Рис. 1.4. Список сотрудников



Рис. 1.5. Статистика

3. Проектирование и создание базы данных

3.1. Разработка структуры БД

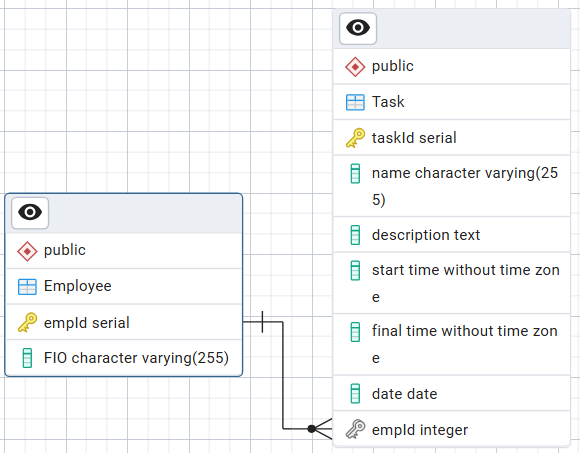


Рис. 3.1. Схема данных



Рис. 3.2. Таблица “Employee”

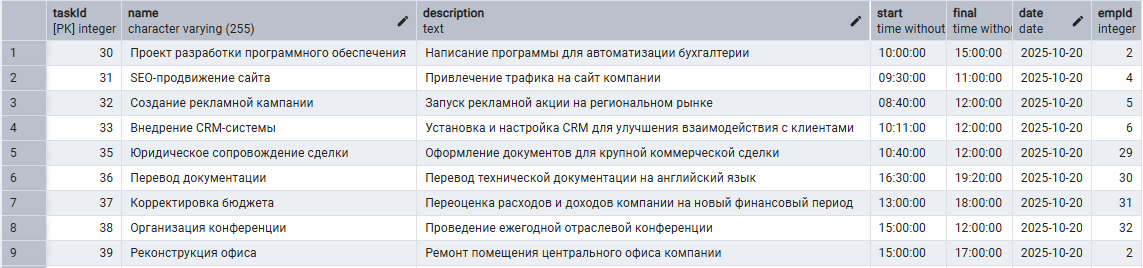


Рис. 3.3. Таблица “Task”

3.2. Создание модели базы данных

Листинг 1 - модуль классов БД

class Employee(Base):  
 \_\_tablename\_\_ = "Employee"  
  
 empId = Column(Integer, primary\_key=True, index=True)  
 FIO = Column(String)  
  
class Task(Base):  
 \_\_tablename\_\_ = "Task"  
  
 taskId = Column(Integer, primary\_key=True, index=True)  
 name = Column(String)  
 description = Column(String)  
 start = Column(Time)  
 final = Column(Time)  
 date = Column(Date)  
 empId = Column(Integer, ForeignKey('Employee.empId'))

4. Разработка программных модулей

4.1. Создание сервера

4.1.1. Модуль подключения к базе данных

Листинг 2 - модуль **database.py** для подключения к БД

from sqlalchemy import create\_engine, Column, Integer, String, Date, Time, ForeignKey  
from sqlalchemy.orm import DeclarativeBase, relationship  
from sqlalchemy.orm import sessionmaker  
  
POSTGRESQL\_DATABASE\_URL = "postgresql+psycopg2://postgres:1234@localhost:5432/LaborTracker"  
engine = create\_engine(POSTGRESQL\_DATABASE\_URL)  
  
class Base(DeclarativeBase):  
 pass  
  
class Employee(Base):  
 \_\_tablename\_\_ = "Employee"  
  
 empId = Column(Integer, primary\_key=True, index=True)  
 FIO = Column(String)  
  
  
class Task(Base):  
 \_\_tablename\_\_ = "Task"  
  
 taskId = Column(Integer, primary\_key=True, index=True)  
 name = Column(String)  
 description = Column(String)  
 start = Column(Time)  
 final = Column(Time)  
 date = Column(Date)  
 empId = Column(Integer, ForeignKey('Employee.empId')) # Исправлено на внешний ключ  
  
 # Связь с сотрудником  
 employee = relationship("Employee")  
  
SessionLocal = sessionmaker(autoflush=False, bind=engine)  
  
try:  
 with engine.connect() as connection:  
 print("✅ Подключение к PostgreSQL успешно")  
 # Создаем таблицы  
 Base.metadata.create\_all(bind=engine)  
except Exception as e:  
 print(f"❌ Ошибка подключения: {e}")

4.1.2. Модуль функций доступа к данным API (GET, POST, PUT, DELETE) на примере таблицы Task:

Листинг 3 - Модуль GET для получения строк из таблицы Task с возможностью фильтрации

@app.get("/api/tasks")  
def get\_tasks(db: Session = Depends(get\_db)):  
 tasks = db.query(Task).join(Employee).all()  
 result = []  
 for task in tasks:  
 # Форматируем время в HH:MM:SS  
 start\_time = task.start.strftime("%H:%M:%S") if task.start else None  
 final\_time = task.final.strftime("%H:%M:%S") if task.final else None  
  
 # Вычисляем общее время  
 total\_time = "00:00:00"  
 if task.start and task.final:  
 start\_dt = dt.datetime.combine(dt.date.today(), task.start)  
 final\_dt = dt.datetime.combine(dt.date.today(), task.final)  
 duration = final\_dt - start\_dt  
 total\_seconds = int(duration.total\_seconds())  
 hours = total\_seconds // 3600  
 minutes = (total\_seconds % 3600) // 60  
 seconds = total\_seconds % 60  
 total\_time = f"{hours:02d}:{minutes:02d}:{seconds:02d}"  
  
 result.append({  
 "taskId": task.taskId,  
 "name": task.name,  
 "description": task.description,  
 "start": start\_time,  
 "final": final\_time,  
 "total\_time": total\_time,  
 "date": str(task.date) if task.date else None,  
 "FIO": task.employee.FIO  
 })  
 return result

Листинг 4 - Модуль POST для создания задачи в Task с указанием неполных данных

@app.post("/api/tasks")  
def create\_task(task: TaskCreate, db: Session = Depends(get\_db)):  
 # Проверяем существование сотрудника  
 employee = db.query(Employee).filter(Employee.empId == task.empId).first()  
 if not employee:  
 raise HTTPException(status\_code=404, detail="Сотрудник не найден")  
  
 db\_task = Task(  
 name=task.name,  
 description=task.description,  
 empId=task.empId,  
 date=dt.date.today()  
 )  
 db.add(db\_task)  
 db.commit()  
 db.refresh(db\_task)  
 return {"message": "Задача создана", "taskId": db\_task.taskId}

Листинг 5 - Модули PUT для инициализации/остановки времени задания

@app.put("/api/tasks/{task\_id}/start")  
def start\_task(task\_id: int, db: Session = Depends(get\_db)):  
 task = db.query(Task).filter(Task.taskId == task\_id).first()  
 if not task:  
 raise HTTPException(status\_code=404, detail="Задача не найдена")  
  
 task.start = dt.datetime.now().time()  
 db.commit()  
 return {"message": "Задача начата"}

@app.put("/api/tasks/{task\_id}/stop")  
def stop\_task(task\_id: int, db: Session = Depends(get\_db)):  
 task = db.query(Task).filter(Task.taskId == task\_id).first()  
 if not task:  
 raise HTTPException(status\_code=404, detail="Задача не найдена")  
  
 task.final = dt.datetime.now().time()  
 db.commit()  
 return {"message": "Задача остановлена"}

Листинг 6 - Модуль DELETE для удаления задания

@app.delete("/api/tasks/{task\_id}")  
def delete\_task(task\_id: int, db: Session = Depends(get\_db)):  
 task = db.query(Task).filter(Task.taskId == task\_id).first()  
 if not task:  
 raise HTTPException(status\_code=404, detail="Задача не найдена")  
  
 db.delete(task)  
 db.commit()  
 return {"message": "Задача удалена"}

4.2. Создание клиента

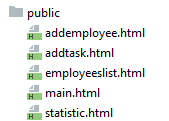


Рис. 4.1. Структура страниц

4.2.1. **main.html**. Данный макет содержит верхнюю панель с заголовкой, кнопками. В центре приведены таблицы с атрибутами, на шапке таблицы отображены кнопки для управления строками и поле для поиска строки, ещё на вверху отображается зеленый индикатор состояния задачи или сообщений:

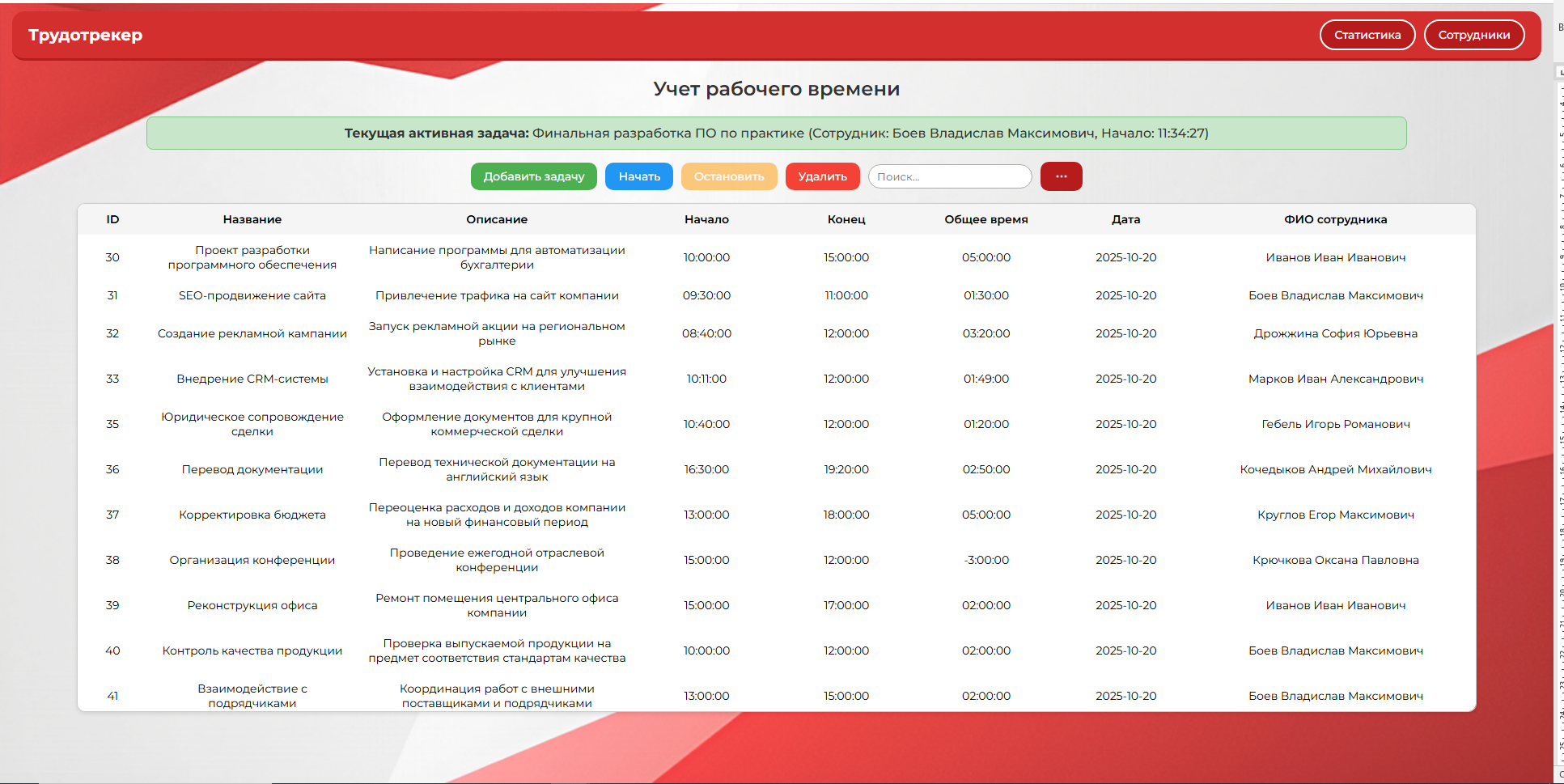


Рис. 4.2. Главная страница

Листинг 7 - Пример модуля для данной страницы:

function displayTasks(tasksToDisplay) {  
 const tbody = document.getElementById('tasksTableBody');  
 tbody.innerHTML = '';  
  
 tasksToDisplay.forEach(task => {  
 const row = document.createElement('tr');  
 row.dataset.id = task.taskId;  
  
 if (task.start && !task.final) {  
 row.classList.add('active-task-row');  
 }  
  
 row.innerHTML = `  
 <td>${task.taskId}</td>  
 <td>${task.name}</td>  
 <td>${task.description || ''}</td>  
 <td>${task.start || ''}</td>  
 <td>${task.final || ''}</td>  
 <td>${task.total\_time}</td>  
 <td>${task.date || ''}</td>  
 <td>${task.FIO}</td>  
 `;  
  
 // Обработчик выбора строки  
 row.addEventListener('click', function() {  
 // Снимаем выделение со всех строк  
 document.querySelectorAll('#tasksTable tr').forEach(r => {  
 r.classList.remove('selected');  
 });  
 // Выделяем текущую строку  
 this.classList.add('selected');  
 selectedTaskId = parseInt(this.dataset.id);  
 updateButtonState(selectedTaskId);  
 });  
  
 tbody.appendChild(row);  
 });  
}

Принцип данной функции заключается в том, что она принимает массив задач и рендерит их в HTML-таблице, обновляя интерфейс в реальном времени.

4.2.2. **statistic.html**: На странице также есть верхняя панель. Ниже расположены заголовок и панель со статистиками, там отображаются общее время работы, кол-во задач, кол-во завершенных задач и список работы по дням.

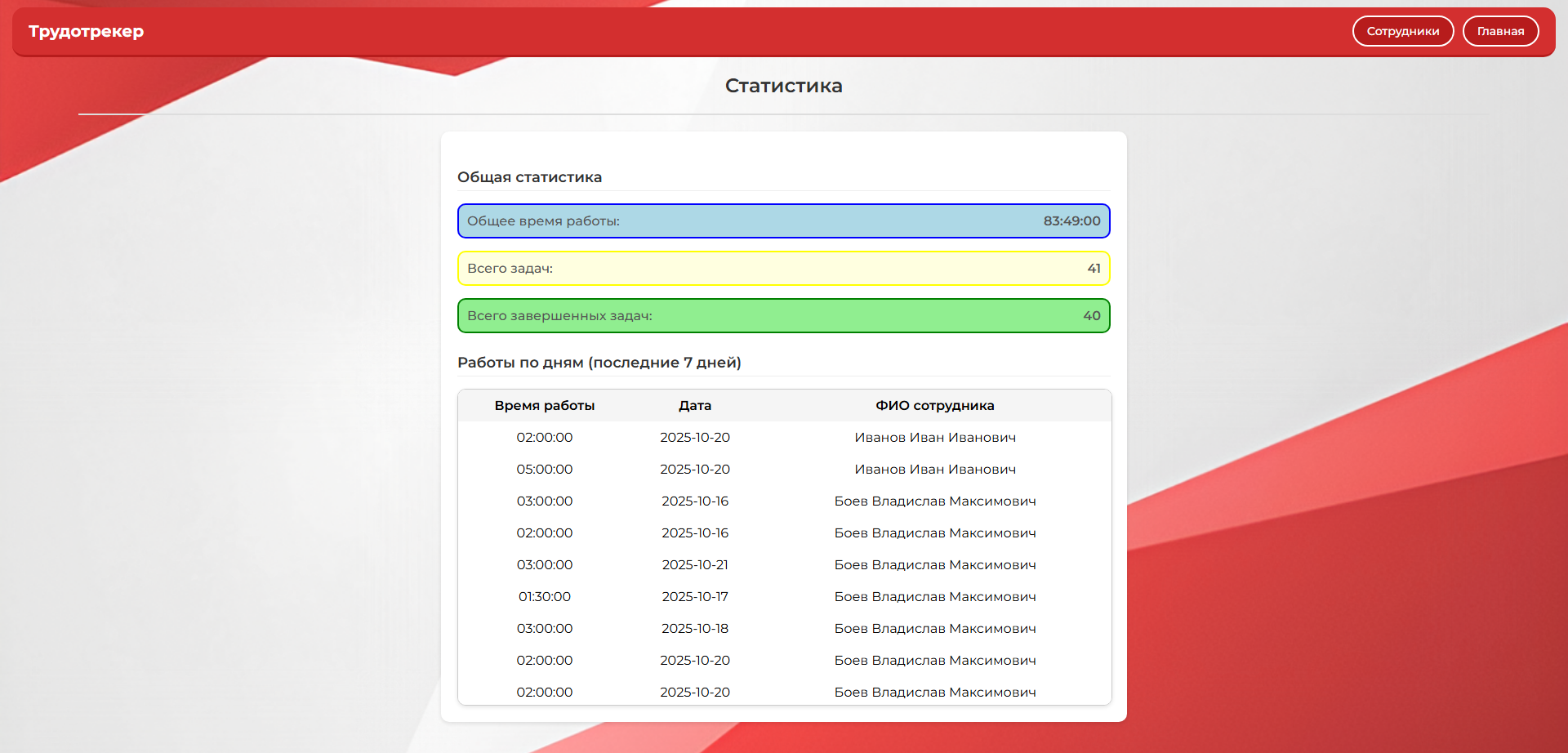


Рис. 4.3. Страница статистики

Листинг 8 - Пример модуля для данной страницы (Рис. 4.3):

async function loadStatistics() {  
 try {  
 const response = await fetch('/api/statistics');  
 const statistics = await response.json();  
  
 // Обновляем общую статистику  
 document.getElementById('totalTime').textContent = statistics.total\_time;  
 document.getElementById('totalTasks').textContent = statistics.total\_tasks;  
 document.getElementById('completedTasks').textContent = statistics.completed\_tasks;  
  
 // Обновляем таблицу работ по дням  
 const tbody = document.getElementById('dailyWorkBody');  
 tbody.innerHTML = '';  
  
 statistics.daily\_work.forEach(work => {  
 const row = document.createElement('tr');  
 row.innerHTML = `  
 <td>${work.time}</td>  
 <td>${work.date}</td>  
 <td>${work.FIO}</td>  
 `;  
 tbody.appendChild(row);  
 });  
 } catch (error) {  
 console.error('Ошибка загрузки статистики:', error);  
 }  
}

Данная функция выполняет **загрузку и отображение статистики. Её принцип в том, что** асинхронная функция получает статистические данные с сервера и динамически обновляет страницу.

4.2.3. **employeeslist.html**: На странице также есть верхняя панель. Ниже расположены заголовок и ниже панель c шапками для управления таблицей, там отображаются записи сотрудников с отображением 3 кнопок на выбор (docx, xlsx, pdf) для скачивания отчета соответствующего сотрудника. Через кнопку Три точки на шапке можно экспортировать/импортировать записи или удалить все записи.

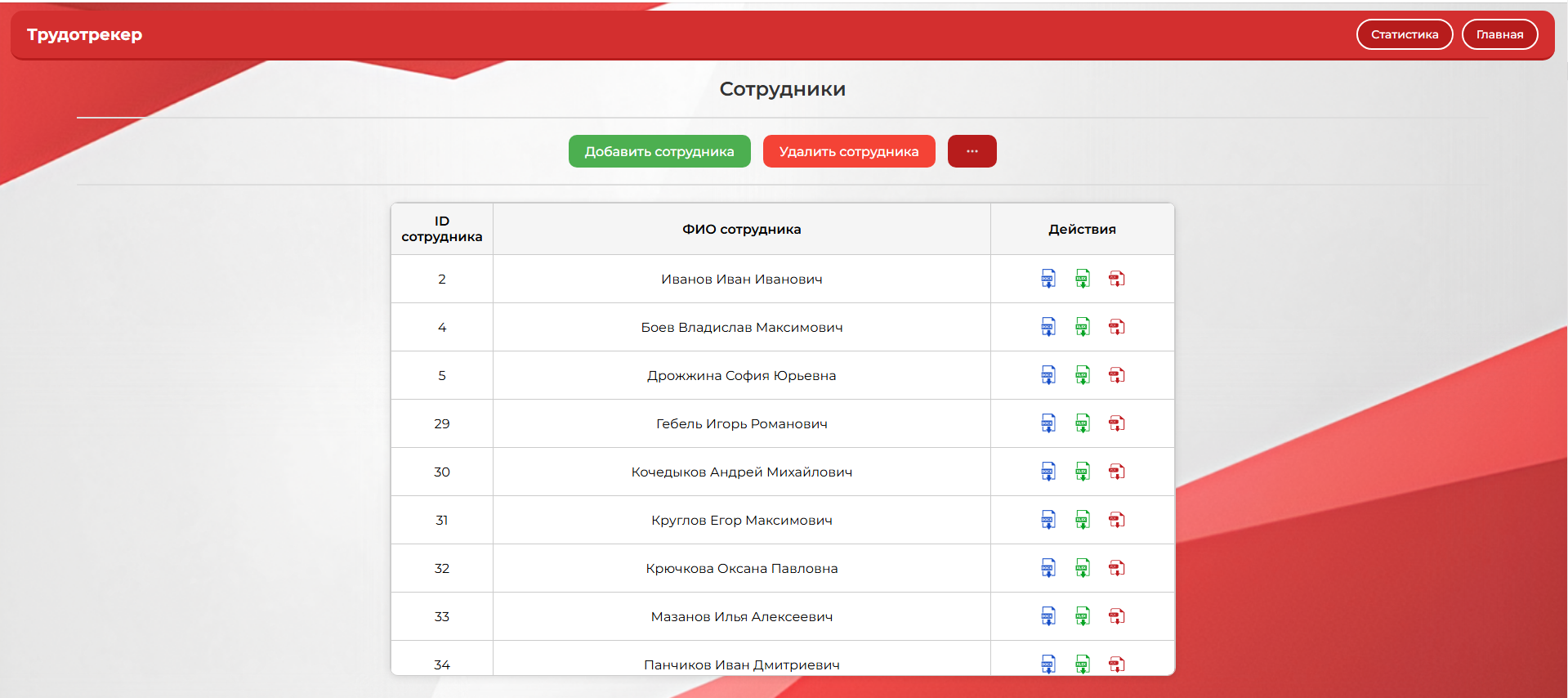


Рис. 4.4. Страница списка сотрудников

Листинг 9 - Пример модуля для данной страницы (Рис. 4.4):

async function importEmployees() {  
 const fileInput = document.createElement('input');  
 fileInput.type = 'file';  
 fileInput.accept = '.xlsx, .xls';  
  
 fileInput.onchange = async (e) => {  
 const file = e.target.files[0];  
 if (!file) return;  
  
 const formData = new FormData();  
 formData.append('file', file);  
 formData.append('ignore\_duplicates', 'true');  
  
 try {  
 const response = await fetch('/api/import/employees', {  
 method: 'POST',  
 body: formData  
 });  
  
 if (response.ok) {  
 const result = await response.json();  
 let message = `Импорт завершен!\nВсего строк: ${result.total\_rows}\nИмпортировано: ${result.imported\_rows}\nПропущено: ${result.skipped\_rows}`;  
  
 if (result.errors.length > 0) {  
 message += `\n\nОшибки:\n${result.errors.slice(0, 5).join('\n')}`;  
 if (result.errors.length > 5) {  
 message += `\n... и еще ${result.errors.length - 5} ошибок`;  
 }  
 }  
  
 alert(message);  
 await loadEmployees(); // Обновляем список  
 } else {  
 const error = await response.json();  
 alert(`Ошибка импорта: ${error.detail}`);  
 }  
 } catch (error) {  
 console.error('Ошибка импорта:', error);  
 alert('Ошибка при импорте файла');  
 }  
 };  
  
 fileInput.click();  
}

Данная функция реализует импорт сотрудников из Excel файла. Принцип: Функция создает скрытый файловый input, позволяет пользователю выбрать Excel-файл и загружает данные на сервер.

4.2.4. **addtask.html**: На странице также есть верхняя панель. Ниже расположен панель c заголовком, полями для заполнения и листом для выбора сотрудников. По заполнению можно добавить задачу.

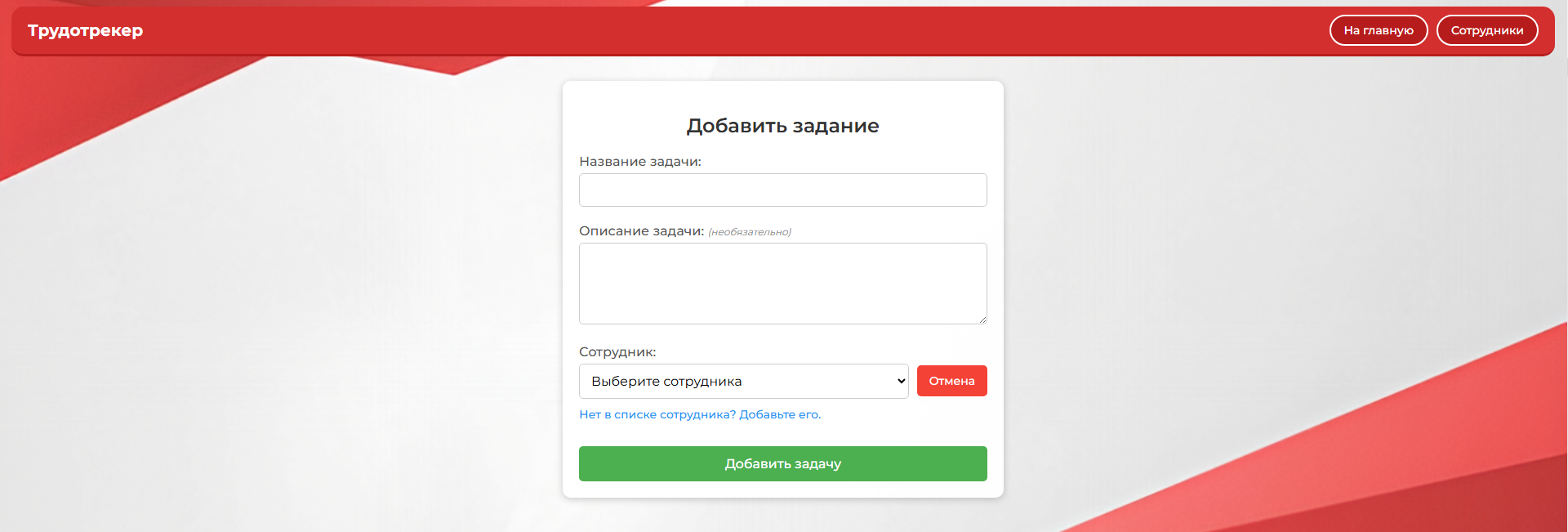


Рис. 4.5. Страница добавления задания

Листинг 10 - Пример модуля для данной страницы (Рис. 4.5):

async function loadEmployees() {  
 try {  
 const response = await fetch('/api/employees');  
 const employees = await response.json();  
  
 const select = document.getElementById('employee');  
 select.innerHTML = '<option value="">Выберите сотрудника</option>';  
  
 employees.forEach(emp => {  
 const option = document.createElement('option');  
 option.value = emp.empId;  
 option.textContent = emp.FIO;  
 select.appendChild(option);  
 });  
 } catch (error) {  
 console.error('Ошибка загрузки сотрудников:', error);  
 }  
}

Данная функция выполняет загрузку списка сотрудников для выпадающего меню. Асинхронная функция, которая получает список сотрудников с сервера и заполняет select-элемент опциями для выбора.

4.2.5. **addemployee.html**: На странице также есть верхняя панель. Ниже расположен панель c заголовком, полями для заполнения

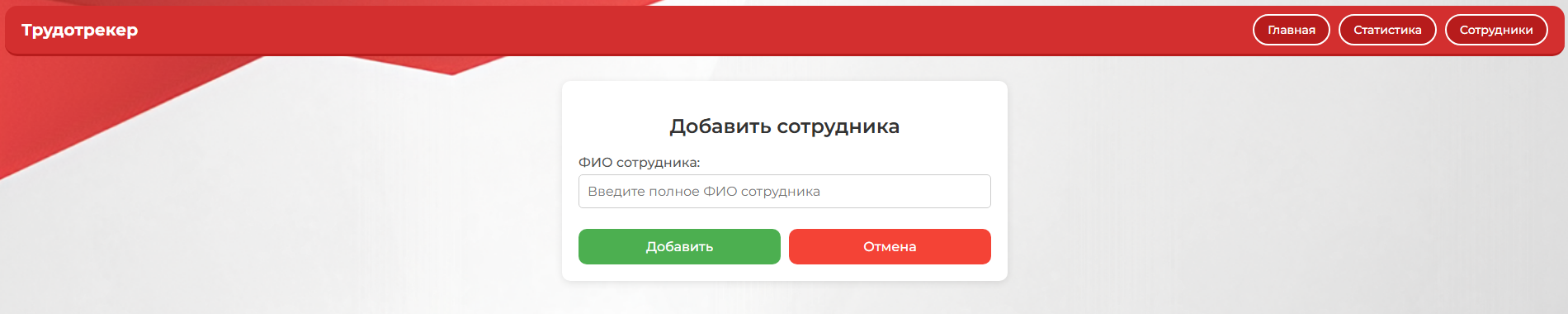


Рис. 4.6. Страница добавления сотрудника

Листинг 11 - Пример модуля для данной страницы (Рис. 4.6):

document.getElementById('add-employee-form').addEventListener('submit', async function(e) {  
 e.preventDefault(); // Предотвращаем стандартную отправку формы  
  
 const employeeName = document.getElementById('employee-name').value;  
  
 try {  
 // Отправляем данные на сервер с помощью AJAX  
 const response = await fetch('/api/employees', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({  
 FIO: employeeName  
 })  
 });  
  
 // Проверяем, успешен ли ответ сервера  
 if (response.ok) {  
 const result = await response.json();  
 alert(`Сотрудник "${employeeName}" успешно добавлен! (ID: ${result.empId})`);  
 // Перенаправляем на страницу со списком сотрудников после успеха  
 window.location.href = '/employees'; // Исправленный URL  
 } else {  
 // Обрабатываем ошибку от сервера (например, 500)  
 const errorData = await response.json();  
 alert(`Ошибка: ${errorData.detail}`);  
 }  
 } catch (error) {  
 // Обрабатываем ошибки сети или другие исключения  
 console.error('Ошибка при отправке запроса:', error);  
 alert('Произошла сетевая ошибка при добавлении сотрудника.');  
 }  
});

Данный код реализует обработку добавления нового сотрудника через AJAX, перехватывает отправку формы, отправляет данные асинхронно и обрабатывает ответ без перезагрузки страницы.

4.3. Реализация модулей вывода на печать

4.3.1. Вывод в шаблон .docx

За данную функцию отвечают следующие библиотеки:

* **from database import \*** - Модели БД (Employee, Task) и подключение к базе данных
* **from sqlalchemy.orm import Session** - Работа с сессиями БД для выполнения запросов
* **from fastapi import Depends, HTTPException, BackgroundTasks** -
* **Depends**: внедрение зависимостей (сессия БД)
* **HTTPException**: обработка ошибок API
* **BackgroundTasks**: фоновая задача удаления файла
* **from fastapi.responses import FileResponse** - Отправка файлов клиенту
* **import datetime as dt** - Работа с датами и временем для расчетов длительности задач
* **from docx import Document** - Создание и редактирование Word документов
* **import o**s - Работа с файловой системой (проверка существования файлов, пути)
* **import uuid** - Генерация уникальных имен файлов
* **from docx.oxml.ns import qn** - Поддержка кириллицы в Word документах
* **from docx.shared import Pt** - Установка размера шрифта в пунктах

Листинг 12 - Модуль вывода в шаблон docx

from database import \*

from sqlalchemy.orm import Session

from fastapi import Depends, HTTPException, BackgroundTasks

from fastapi.responses import FileResponse

import datetime as dt

from docx import Document

import os

import uuid

from docx.oxml.ns import qn

from docx.shared import Pt

@app.get("/api/employees/{emp\_id}/report")  
def generate\_employee\_report(  
 emp\_id: int,  
 background\_tasks: BackgroundTasks,  
 db: Session = Depends(get\_db)  
):  
 try:  
 employee = db.query(Employee).filter(Employee.empId == emp\_id).first()  
 if not employee:  
 raise HTTPException(status\_code=404, detail="Сотрудник не найден")  
  
 tasks = db.query(Task).filter(  
 Task.empId == emp\_id,  
 Task.start.isnot(None),  
 Task.final.isnot(None)  
 ).order\_by(Task.date).all()  
  
 if not tasks:  
 raise HTTPException(status\_code=400, detail="Нет завершенных задач для сотрудника")  
  
 # Собираем данные  
 task\_list = []  
 total\_seconds = 0  
 dates = []  
 for task in tasks:  
 start\_dt = dt.datetime.combine(dt.date.today(), task.start)  
 final\_dt = dt.datetime.combine(dt.date.today(), task.final)  
 duration = final\_dt - start\_dt  
 hours = duration.total\_seconds() / 3600  
 total\_seconds += duration.total\_seconds()  
 if task.date:  
 dates.append(task.date)  
 task\_list.append({  
 "id": task.taskId,  
 "name": task.name,  
 "description": task.description or "",  
 "hours": f"{hours:.2f}"  
 })  
  
 total\_hours = total\_seconds / 3600  
 start\_date = min(dates).strftime('%d.%m.%Y') if dates else 'Н/Д'  
 end\_date = max(dates).strftime('%d.%m.%Y') if dates else 'Н/Д'  
  
 # Получаем фамилию с инициалами  
 family\_with\_inits = get\_family\_with\_initials(employee.FIO)  
  
 # Загружаем шаблон  
 template\_path = "report\_template.docx"  
 if not os.path.exists(template\_path):  
 raise HTTPException(status\_code=500, detail="Шаблон report\_template.docx не найден")  
  
 doc = Document(template\_path)  
  
 # === Замена простых переменных ===  
 def replace\_in\_paragraph(paragraph, old, new):  
 """Замена текста с явным указанием размера шрифта"""  
 if old in paragraph.text:  
 original\_text = paragraph.text  
 paragraph.text = paragraph.text.replace(old, str(new))  
  
 # Устанавливаем размер шрифта для всех runs в параграфе  
 for run in paragraph.runs:  
 run.font.size = Pt(14)  
 run.font.name = "Times New Roman"  
 # Для поддержки кириллицы  
 run.\_element.rPr.rFonts.set(qn('w:eastAsia'), 'Times New Roman')  
  
 for paragraph in doc.paragraphs:  
 replace\_in\_paragraph(paragraph, "{{ fio }}", employee.FIO)  
 replace\_in\_paragraph(paragraph, "{{ start\_date }}", start\_date)  
 replace\_in\_paragraph(paragraph, "{{ end\_date }}", end\_date)  
 replace\_in\_paragraph(paragraph, "{{ total\_hours }}", f"{total\_hours:.2f}")  
 replace\_in\_paragraph(paragraph, "{{ family\_with\_inits }}", family\_with\_inits)  
  
 # === Работа с таблицей ===  
 if len(doc.tables) == 0:  
 raise HTTPException(status\_code=500, detail="В шаблоне не найдена таблица")  
  
 table = doc.tables[0]  
  
 # Удаляем первую строку данных (если она есть и не является заголовком)  
 if len(table.rows) > 1:  
 row = table.rows[1].\_element  
 row.getparent().remove(row)  
  
 # Функция для установки шрифта в ячейке таблицы  
 def set\_cell\_font(cell, font\_name="Times New Roman", font\_size=14):  
 """Устанавливает шрифт для всех параграфов в ячейке таблицы"""  
 for paragraph in cell.paragraphs:  
 for run in paragraph.runs:  
 run.font.name = font\_name  
 run.font.size = Pt(font\_size)  
 run.\_element.rPr.rFonts.set(qn('w:eastAsia'), font\_name)  
  
 # Добавляем строки с данными и устанавливаем шрифт  
 for task in task\_list:  
 row\_cells = table.add\_row().cells  
 row\_cells[0].text = str(task["id"])  
 row\_cells[1].text = task["name"]  
 row\_cells[2].text = task["description"]  
 row\_cells[3].text = task["hours"]  
  
 # Устанавливаем шрифт для всех ячеек новой строки  
 for cell in row\_cells:  
 set\_cell\_font(cell, "Times New Roman", 14)  
  
 # Дополнительно: устанавливаем шрифт для заголовков таблицы (если нужно)  
 if len(table.rows) > 0:  
 header\_cells = table.rows[0].cells  
 for cell in header\_cells:  
 set\_cell\_font(cell, "Times New Roman", 14)  
 # Можно сделать заголовки жирными  
 for paragraph in cell.paragraphs:  
 for run in paragraph.runs:  
 run.bold = True  
  
 # Сохраняем  
 filename = f"report\_{uuid.uuid4().hex}.docx"  
 filepath = os.path.join(BUFFER\_DIR, filename)  
 doc.save(filepath)  
  
 background\_tasks.add\_task(delete\_file, filepath)  
 download\_filename = f"report\_{employee.FIO.replace(' ', '\_')}.docx"  
 return FileResponse(  
 path=filepath,  
 filename=download\_filename,  
 media\_type="application/vnd.openxmlformats-officedocument.wordprocessingml.document"  
 )  
  
 except Exception as e:  
 import traceback  
 print(traceback.format\_exc())  
 raise HTTPException(status\_code=500, detail=f"Ошибка генерации отчёта: {str(e)}")

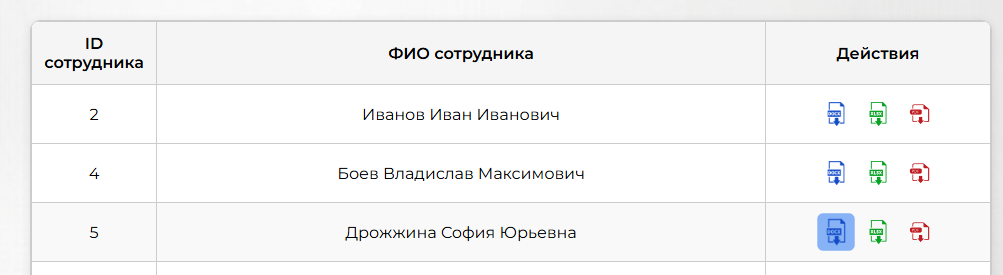


Рис. 4.7. Сохранение отчета docx соотвествующего сотрудника.

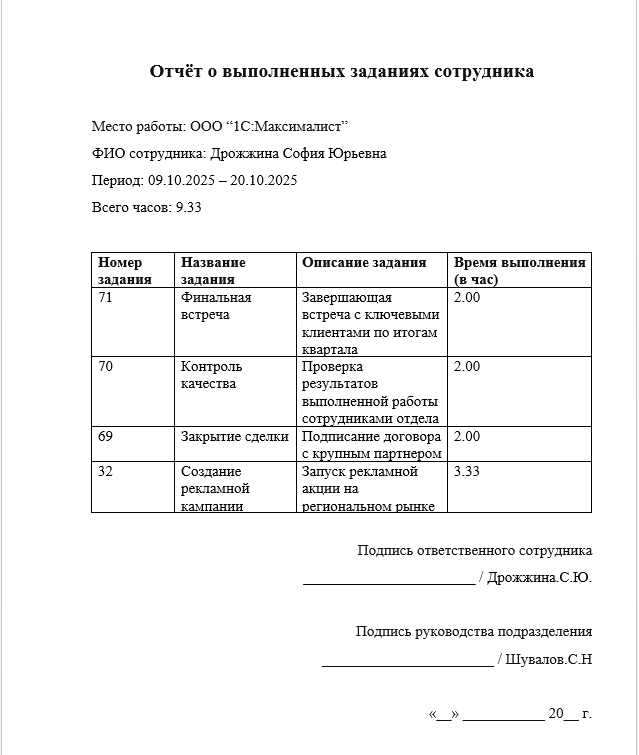


Рис. 4.8. Результат сохранения отчета docx.

4.3.2. Создание документа .xlsx

Для этого используются следующие библиотеки:

* **from database import \*** - Модели БД (Employee, Task) для запросов к базе данных
* **from sqlalchemy.orm import Session** - Работа с сессиями БД
* **from fastapi import Depends, HTTPException, BackgroundTasks** -
  + **Depends**: внедрение зависимостей
  + **HTTPException**: обработка ошибок
  + **BackgroundTasks**: фоновая задача удаления файла
* **from fastapi.responses import FileResponse**- Отправка Excel файлов клиенту
* **import datetime as dt** - Расчет времени выполнения задач
* **import os** - Работа с файловой системой
* **import uuid** - Генерация уникальных имен файлов
* **from openpyxl import load\_workbook** - Загрузка Excel шаблона
* **from openpyxl.styles import Font, PatternFill, Alignment, Border,** **Side** - Стилизация ячеек:
  + **Font**: жирные заголовки
  + **PatternFill**: заливка фона
  + **Alignment**: выравнивание текста
  + **Border**: границы ячеек
* **from openpyxl.utils import get\_column\_letter** - Настройка ширины колонок

Листинг 13 – Функция сохранения отчета в Excel:

from database import \*

from sqlalchemy.orm import Session

from fastapi import Depends, HTTPException, BackgroundTasks

from fastapi.responses import FileResponse

import datetime as dt

import os

import uuid

from openpyxl import load\_workbook

from openpyxl.styles import Font, PatternFill, Alignment, Border, Side

from openpyxl.utils import get\_column\_letter

@app.get("/api/employees/{emp\_id}/excel-report")

def generate\_employee\_excel\_report(

emp\_id: int,

background\_tasks: BackgroundTasks,

db: Session = Depends(get\_db)

):

try:

employee = db.query(Employee).filter(Employee.empId == emp\_id).first()

if not employee:

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

# Получаем ВСЕ задачи сотрудника (не только завершенные)

all\_tasks = db.query(Task).filter(Task.empId == emp\_id).order\_by(Task.date).all()

if not all\_tasks:

raise HTTPException(status\_code=400, detail="Нет задач для сотрудника")

# Собираем данные для таблицы задач

task\_data = []

total\_seconds = 0

completed\_tasks\_count = 0

dates = []

for task in all\_tasks:

hours = 0

if task.start and task.final:

if task.date:

start\_dt = dt.datetime.combine(task.date, task.start)

final\_dt = dt.datetime.combine(task.date, task.final)

else:

start\_dt = dt.datetime.combine(dt.date.today(), task.start)

final\_dt = dt.datetime.combine(dt.date.today(), task.final)

duration = final\_dt - start\_dt

hours = duration.total\_seconds() / 3600

total\_seconds += duration.total\_seconds()

completed\_tasks\_count += 1

if task.date:

dates.append(task.date)

task\_data.append({

"№": task.taskId,

"Название задания": task.name,

"Описание задачи": task.description or "",

"Дата выполнения": task.date.strftime('%d.%m.%Y') if task.date else '',

"Время выполнения (в час)": round(hours, 2)

})

# Рассчитываем показатели

total\_hours = total\_seconds / 3600

total\_tasks\_count = len(all\_tasks)

completion\_rate = (completed\_tasks\_count / total\_tasks\_count \* 100) if total\_tasks\_count > 0 else 0

start\_date = min(dates).strftime('%d.%m.%Y') if dates else 'Н/Д'

end\_date = max(dates).strftime('%d.%m.%Y') if dates else 'Н/Д'

# Получаем фамилию с инициалами

family\_with\_inits = get\_family\_with\_initials(employee.FIO)

# Загружаем шаблон из static папки

template\_path = "report\_template.xlsx"

if not os.path.exists(template\_path):

raise HTTPException(status\_code=500, detail="Шаблон static/report\_template.xlsx не найден")

# Открываем шаблон

wb = load\_workbook(template\_path)

ws = wb.active

# Заполняем основные данные согласно шаблону

ws['C9'] = employee.FIO # [FIO] - ФИО сотрудника

ws['C10'] = f"{start\_date} - {end\_date}" # [min(date)-max(date)] - Период

ws['C11'] = round(total\_hours, 2) # [total\_hours] - Общее кол-во часов

ws['C12'] = total\_tasks\_count # [total count tasks] - Общее кол-во задач

ws['C13'] = f"{completion\_rate:.1f}%" # [percent of completed tasks] - Процент выполнения

ws['C14'] = completed\_tasks\_count # [total count completed tasks] - Выполнено задач

# Выравниваем по левому краю ячейки A5:B5 (адрес)

ws['A5'].alignment = Alignment(horizontal='left')

ws['B5'].alignment = Alignment(horizontal='left')

# Выравниваем по левому краю числовые ячейки C11, C12, C14

ws['C11'].alignment = Alignment(horizontal='left')

ws['C12'].alignment = Alignment(horizontal='left')

ws['C14'].alignment = Alignment(horizontal='left')

# Заполняем таблицу задач начиная с 17 строки

start\_row = 17

# Устанавливаем заголовки таблицы (строка 17)

headers = ["№", "Название задания", "Описание задачи", "Дата выполнения", "Время выполнения (в час)"]

for col, header in enumerate(headers, 1):

cell = ws.cell(row=start\_row, column=col, value=header)

# Стили для заголовков

cell.font = Font(bold=True)

cell.fill = PatternFill(start\_color="E0E0E0", end\_color="E0E0E0", fill\_type="solid")

cell.border = Border(

left=Side(style='thin'),

right=Side(style='thin'),

top=Side(style='thin'),

bottom=Side(style='thin')

)

cell.alignment = Alignment(horizontal='center', vertical='center')

# Заполняем данные задач (начиная со строки 18)

for i, task in enumerate(task\_data):

row = start\_row + 1 + i

# Заполняем ячейки

ws.cell(row=row, column=1, value=task["№"])

ws.cell(row=row, column=2, value=task["Название задания"])

ws.cell(row=row, column=3, value=task["Описание задачи"])

ws.cell(row=row, column=4, value=task["Дата выполнения"])

ws.cell(row=row, column=5, value=task["Время выполнения (в час)"])

# Добавляем границы для всех ячеек

for col in range(1, 6):

cell = ws.cell(row=row, column=col)

cell.border = Border(

left=Side(style='thin'),

right=Side(style='thin'),

top=Side(style='thin'),

bottom=Side(style='thin')

)

# Выравнивание по центру для №, даты и времени

if col in [1, 4, 5]:

cell.alignment = Alignment(horizontal='center', vertical='center')

else:

cell.alignment = Alignment(horizontal='left', vertical='center', wrap\_text=True)

# Добавляем подпись после таблицы задач

last\_task\_row = start\_row + len(task\_data) + 2

# Подпись ответственного сотрудника (выравниваем по правому краю в столбце E)

signature\_cell = ws.cell(row=last\_task\_row, column=5, value="Подпись ответственного сотрудника")

signature\_cell.alignment = Alignment(horizontal='right')

# ФИО с инициалами (выравниваем по правому краю в столбце E)

name\_cell = ws.cell(row=last\_task\_row + 1, column=5, value=f"\_\_\_\_\_\_\_\_\_\_ / {family\_with\_inits}")

name\_cell.alignment = Alignment(horizontal='right')

# Настраиваем ширину колонок для лучшего отображения

column\_widths = [8, 30, 40, 17, 24] # Обновленные ширины для столбцов D и E

for i, width in enumerate(column\_widths, 1):

ws.column\_dimensions[get\_column\_letter(i)].width = width

# Сохраняем файл

filename = f"excel\_report\_{uuid.uuid4().hex}.xlsx"

filepath = os.path.join(BUFFER\_DIR, filename)

wb.save(filepath)

background\_tasks.add\_task(delete\_file, filepath)

download\_filename = f"excel\_report\_{employee.FIO.replace(' ', '\_')}.xlsx"

return FileResponse(

path=filepath,

filename=download\_filename,

media\_type="application/vnd.openxmlformats-officedocument.spreadsheetml.document"

)

except Exception as e:

import traceback

print(traceback.format\_exc())

raise HTTPException(status\_code=500, detail=f"Ошибка генерации Excel отчёта: {str(e)}")

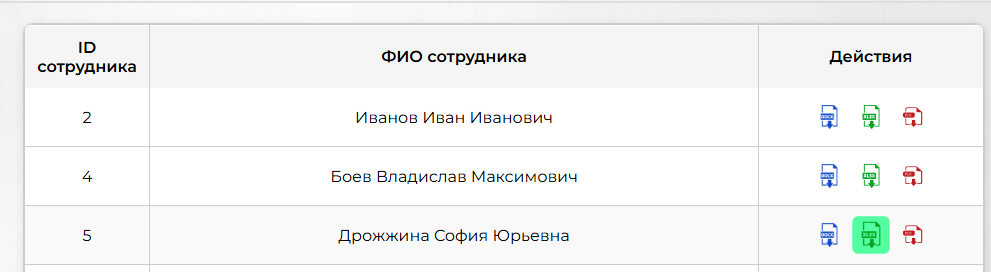


Рис. 4.9. Сохранение отчета xlsx соотвествующего сотрудника.



Рис. 4.10. Результат сохранения отчета xlsx.

4.3.3. Создание документа .pdf

Для этого используются следующие библиотеки:

* **from database import \*** - Модели БД (Employee, Task)
* **from sqlalchemy.orm import Session** - Работа с сессиями БД
* **from fastapi import Depends, HTTPException, BackgroundTasks**-
  + **Depends**: внедрение зависимостей
  + **HTTPException**: обработка ошибок
  + **BackgroundTasks**: фоновая задача удаления файла
* **from fastapi.responses import FileResponse** - Отправка PDF файлов
* **import datetime as dt** - Работа с датами и временем
* **import os** - Работа с файловой системой
* **import uuid** - Генерация уникальных имен файлов
* **from reportlab.lib.pagesizes import A4** - Размер страницы
* **from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle** - Стили текста
* **from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer** -
  + **SimpleDocTemplate**: создание PDF документа
  + **Table**: таблицы с данными
  + **Paragraph**: форматированный текст
  + **Spacer**: отступы
* **from reportlab.lib.units import cm** - Единицы измерения
* **from reportlab.lib import colors** - Цвета для оформления
* **from reportlab.graphics.shapes import Drawing** - Создание рисунков
* **from reportlab.graphics.charts.barcharts import VerticalBarChart** - Вертикальная гистограмма
* **from reportlab.pdfbase import pdfmetrics** - Регистрация шрифтов
* **from reportlab.pdfbase.ttfonts import TTFont** - Работа с TTF шрифтами

Листинг 14 – Функция сохранения отчета в PDF:

from database import \*

from sqlalchemy.orm import Session

from fastapi import Depends, HTTPException, BackgroundTasks

from fastapi.responses import FileResponse

import datetime as dt

import os

import uuid

from reportlab.lib.pagesizes import A4

from reportlab.pdfgen import canvas

from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle

from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer, Image

from reportlab.lib.units import inch, cm from reportlab.lib import colors

from reportlab.graphics.shapes import Drawing

from reportlab.graphics.charts.barcharts import VerticalBarChart

from reportlab.graphics.charts.piecharts import Pie

from reportlab.graphics.charts.legends import Legend from reportlab.pdfbase import pdfmetrics

from reportlab.pdfbase.ttfonts import TTFont

@app.get("/api/employees/{emp\_id}/pdf-report")

def generate\_employee\_pdf\_report(

emp\_id: int,

background\_tasks: BackgroundTasks,

db: Session = Depends(get\_db)

):

try:

employee = db.query(Employee).filter(Employee.empId == emp\_id).first()

if not employee:

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

# Получаем задачи за последние 30 дней

thirty\_days\_ago = dt.date.today() - dt.timedelta(days=30)

tasks = db.query(Task).filter(

Task.empId == emp\_id,

Task.start.isnot(None),

Task.final.isnot(None),

Task.date >= thirty\_days\_ago

).order\_by(Task.date).all()

if not tasks:

raise HTTPException(status\_code=400, detail="Нет завершенных задач для сотрудника за последний месяц")

# Собираем данные

task\_data = []

total\_seconds = 0

dates = []

# Группируем задачи по датам для гистограммы

date\_hours = {}

for task in tasks:

if task.start and task.final:

if task.date:

start\_dt = dt.datetime.combine(task.date, task.start)

final\_dt = dt.datetime.combine(task.date, task.final)

else:

start\_dt = dt.datetime.combine(dt.date.today(), task.start)

final\_dt = dt.datetime.combine(dt.date.today(), task.final)

duration = final\_dt - start\_dt

hours = duration.total\_seconds() / 3600

total\_seconds += duration.total\_seconds()

if task.date:

dates.append(task.date)

date\_str = task.date.strftime('%d.%m.%Y')

# Суммируем часы по датам

date\_hours[date\_str] = date\_hours.get(date\_str, 0) + hours

task\_data.append({

"№": task.taskId,

"Название задания": task.name,

"Описание": task.description or "",

"Дата выполнения": task.date.strftime('%d.%m.%Y') if task.date else '',

"Время выполнения (в час)": round(hours, 2)

})

total\_hours = total\_seconds / 3600

start\_date = min(dates).strftime('%d.%m.%Y') if dates else 'Н/Д'

end\_date = max(dates).strftime('%d.%m.%Y') if dates else 'Н/Д'

# Рассчитываем продолжительность периода в днях

count\_days = 0

if dates:

start\_dt = min(dates)

end\_dt = max(dates)

count\_days = (end\_dt - start\_dt).days + 1 # +1 чтобы включить начальный день

# Функция для правильного склонения слова "день"

def get\_day\_form(days):

if days % 10 == 1 and days % 100 != 11:

return "день"

elif 2 <= days % 10 <= 4 and (days % 100 < 10 or days % 100 >= 20):

return "дня"

else:

return "дней"

# Формируем строку периода с продолжительностью

if dates:

day\_form = get\_day\_form(count\_days)

period\_text = f"{start\_date} - {end\_date} ({count\_days} {day\_form})"

else:

period\_text = "Н/Д"

family\_with\_inits = get\_family\_with\_initials(employee.FIO)

# Создаем PDF файл

filename = f"pdf\_report\_{uuid.uuid4().hex}.pdf"

filepath = os.path.join(BUFFER\_DIR, filename)

# Регистрируем шрифты Times New Roman

try:

# Пути к файлам шрифтов Times New Roman

times\_font\_paths = [

"static/ttf/times.ttf",

"static/ttf/times-new-roman.ttf",

"static/ttf/times-roman.ttf",

"C:/Windows/Fonts/times.ttf",

"C:/Windows/Fonts/timesbd.ttf",

"/usr/share/fonts/truetype/liberation/LiberationSerif-Regular.ttf"

]

times\_found = False

times\_bold\_found = False

for font\_path in times\_font\_paths:

if os.path.exists(font\_path):

if "bd" in font\_path.lower() or "bold" in font\_path.lower():

pdfmetrics.registerFont(TTFont('Times-Bold', font\_path))

times\_bold\_found = True

else:

pdfmetrics.registerFont(TTFont('Times-Roman', font\_path))

times\_found = True

# Если не нашли шрифты, используем DejaVuSans

if not times\_found:

if os.path.exists("static/ttf/DejaVoSans.ttf"):

pdfmetrics.registerFont(TTFont('Times-Roman', "static/ttf/DejaVoSans.ttf"))

times\_found = True

if not times\_bold\_found:

if os.path.exists("static/ttf/DejaVoSans-Bold.ttf"):

pdfmetrics.registerFont(TTFont('Times-Bold', "static/ttf/DejaVoSans-Bold.ttf"))

times\_bold\_found = True

except Exception as e:

print(f"Ошибка регистрации шрифтов: {e}")

# Создаем документ

doc = SimpleDocTemplate(filepath, pagesize=A4,

leftMargin=2 \* cm, rightMargin=2 \* cm,

topMargin=2 \* cm, bottomMargin=2 \* cm)

elements = []

styles = getSampleStyleSheet()

# Создаем кастомные стили с Times New Roman размером 14 и межстрочным интервалом 1.15

line\_height = 14 \* 1.15 # 1.15 межстрочный интервал

try:

title\_style = ParagraphStyle(

'CustomTitle',

parent=styles['Heading1'],

fontName='Times-Bold',

fontSize=14,

leading=line\_height,

spaceAfter=20,

alignment=1, # CENTER

textColor=colors.black

)

header\_left\_style = ParagraphStyle(

'HeaderLeftStyle',

parent=styles['Normal'],

fontName='Times-Roman',

fontSize=14,

leading=line\_height,

spaceAfter=30,

alignment=0 # LEFT

)

header\_right\_style = ParagraphStyle(

'HeaderRightStyle',

parent=styles['Normal'],

fontName='Times-Roman',

fontSize=14,

leading=line\_height,

spaceAfter=30,

alignment=2 # RIGHT

)

header\_bold\_style = ParagraphStyle(

'HeaderBoldStyle',

parent=styles['Normal'],

fontName='Times-Bold',

fontSize=14,

leading=line\_height,

spaceAfter=6

)

normal\_style = ParagraphStyle(

'CustomNormal',

parent=styles['Normal'],

fontName='Times-Roman',

fontSize=14,

leading=line\_height,

spaceAfter=6

)

table\_header\_style = ParagraphStyle(

'TableHeader',

parent=styles['Normal'],

fontName='Times-Bold',

fontSize=14,

leading=line\_height,

alignment=1 # CENTER

)

table\_data\_style = ParagraphStyle(

'TableData',

parent=styles['Normal'],

fontName='Times-Roman',

fontSize=14,

leading=line\_height

)

signature\_style = ParagraphStyle(

'SignatureStyle',

parent=styles['Normal'],

fontName='Times-Roman',

fontSize=14,

leading=line\_height,

alignment=2, # RIGHT

spaceBefore=20

)

except:

# Резервные стили если Times не доступен

title\_style = ParagraphStyle(

'CustomTitle',

parent=styles['Heading1'],

fontSize=14,

leading=line\_height,

spaceAfter=20,

alignment=1,

textColor=colors.black

)

header\_left\_style = ParagraphStyle(

'HeaderLeftStyle',

parent=styles['Normal'],

fontSize=14,

leading=line\_height,

spaceAfter=20,

alignment=0

)

header\_right\_style = ParagraphStyle(

'HeaderRightStyle',

parent=styles['Normal'],

fontSize=14,

leading=line\_height,

spaceAfter=20,

alignment=2

)

header\_bold\_style = ParagraphStyle(

'HeaderBoldStyle',

parent=styles['Normal'],

fontSize=14,

leading=line\_height,

spaceAfter=6

)

normal\_style = ParagraphStyle(

'CustomNormal',

parent=styles['Normal'],

fontSize=14,

leading=line\_height,

spaceAfter=6

)

table\_header\_style = ParagraphStyle(

'TableHeader',

parent=styles['Normal'],

fontSize=14,

leading=line\_height,

alignment=1

)

table\_data\_style = ParagraphStyle(

'TableData',

parent=styles['Normal'],

fontSize=14,

leading=line\_height

)

signature\_style = ParagraphStyle(

'SignatureStyle',

parent=styles['Normal'],

fontSize=14,

leading=line\_height,

alignment=2,

spaceBefore=20

)

# === ВЕРХНИЕ ШАПКИ ===

header\_data = [

[

# Левая шапка - выровнена по левому краю

Paragraph('ООО "1C:Максималист"<br/>Адрес:<br/>г.Москва, проспект Мира, д.126<br/>ИНН 2332543567',

header\_left\_style),

# Правая шапка - выровнена по правому краю

Paragraph(

'УТВЕРЖДАЮ<br/>Руководитель подразделения<br/>\_\_\_\_\_\_\_\_\_\_\_\_\_ / Шувалов.С.Н<br/>от «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.',

header\_right\_style)

]

]

header\_table = Table(header\_data, colWidths=[10 \* cm, 10 \* cm])

header\_table.setStyle(TableStyle([

('VALIGN', (0, 0), (-1, -1), 'TOP'),

]))

elements.append(header\_table)

elements.append(Spacer(1, 1 \* cm))

# === ЗАГОЛОВОК ===

title = Paragraph("ОТЧЕТ О ВЫПОЛНЕННЫХ ЗАДАЧАХ СОТРУДНИКА", title\_style)

elements.append(title)

elements.append(Spacer(1, 1 \* cm))

# === ИНФОРМАЦИЯ О СОТРУДНИКЕ ===

employee\_info\_data = [

[Paragraph('ФИО сотрудника:', header\_bold\_style), Paragraph(employee.FIO, normal\_style)],

[Paragraph('Период анализа:', header\_bold\_style), Paragraph(period\_text, normal\_style)],

[Paragraph('Общее время работы:', header\_bold\_style), Paragraph(f"{total\_hours:.2f} часов", normal\_style)],

[Paragraph('Количество задач:', header\_bold\_style), Paragraph(str(len(tasks)), normal\_style)]

]

employee\_table = Table(employee\_info\_data, colWidths=[5 \* cm, 10 \* cm])

employee\_table.setStyle(TableStyle([

('VALIGN', (0, 0), (-1, -1), 'TOP'),

]))

elements.append(employee\_table)

elements.append(Spacer(1, 1 \* cm))

# === ГИСТОГРАММА ===

if date\_hours:

elements.append(Paragraph("График выполнения задач по датам", header\_bold\_style))

elements.append(Spacer(1, 0.5 \* cm))

# Правильно сортируем даты по хронологии

def parse\_date(date\_str):

return dt.datetime.strptime(date\_str, '%d.%m.%Y')

# Сортируем даты по возрастанию (от старых к новым)

dates\_sorted = sorted(date\_hours.keys(), key=parse\_date)

hours\_data = [date\_hours[date] for date in dates\_sorted]

# Создаем рисунок для гистограммы

drawing = Drawing(400, 200)

chart = VerticalBarChart()

chart.x = 30

chart.y = 50

chart.height = 150

chart.width = 420

chart.data = [hours\_data]

chart.categoryAxis.categoryNames = dates\_sorted

chart.categoryAxis.labels.angle = 45

chart.categoryAxis.labels.dy = -12

chart.categoryAxis.labels.fontName = 'Times-Roman'

chart.categoryAxis.labels.fontSize = 8

chart.valueAxis.valueMin = 0

chart.valueAxis.valueMax = max(hours\_data) \* 1.2 if hours\_data else 10

chart.valueAxis.labels.fontName = 'Times-Roman'

chart.valueAxis.labels.fontSize = 8

chart.bars[0].fillColor = colors.HexColor('#FF0000')

chart.barLabels.fontName = 'Times-Roman'

chart.barLabels.fontSize = 8

drawing.add(chart)

elements.append(drawing)

elements.append(Spacer(1, 0.5 \* cm))

# === ТАБЛИЦА ЗАДАЧ ===

elements.append(Paragraph("Список выполненных задач", header\_bold\_style))

elements.append(Spacer(1, 0.5 \* cm))

# Заголовки таблицы

task\_table\_data = [[

Paragraph('№', table\_header\_style),

Paragraph('Название задания', table\_header\_style),

Paragraph('Описание', table\_header\_style),

Paragraph('Дата выполнения', table\_header\_style),

Paragraph('Время (часы)', table\_header\_style)

]]

# Данные таблицы

for task in task\_data:

task\_table\_data.append([

Paragraph(str(task["№"]), table\_data\_style),

Paragraph(task["Название задания"], table\_data\_style),

Paragraph(task["Описание"], table\_data\_style),

Paragraph(task["Дата выполнения"], table\_data\_style),

Paragraph(str(task["Время выполнения (в час)"]), table\_data\_style)

])

task\_table = Table(task\_table\_data, colWidths=[1.5 \* cm, 5 \* cm, 5 \* cm, 3 \* cm, 2.5 \* cm])

task\_table.setStyle(TableStyle([

('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#F0F0F0')), # Светло-серый фон заголовков

('TEXTCOLOR', (0, 0), (-1, 0), colors.black), # Черный текст для заголовков

('ALIGN', (0, 0), (-1, 0), 'CENTER'),

('ALIGN', (0, 1), (-1, -1), 'LEFT'),

('ALIGN', (4, 1), (4, -1), 'CENTER'),

('FONTNAME', (0, 0), (-1, 0), 'Times-Bold'),

('FONTNAME', (0, 1), (-1, -1), 'Times-Roman'),

('FONTSIZE', (0, 0), (-1, -1), 14),

('BOTTOMPADDING', (0, 0), (-1, -1), 6),

('TOPPADDING', (0, 0), (-1, -1), 6),

('GRID', (0, 0), (-1, -1), 0.5, colors.HexColor('#DEE2E6')),

('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),

]))

elements.append(task\_table)

# === ПОДПИСЬ ===

elements.append(Spacer(1, 1 \* cm))

signature = Paragraph(f"Подпись ответственного сотрудника<br/>\_\_\_\_\_\_\_\_\_\_\_\_\_ / {family\_with\_inits}",

signature\_style)

elements.append(signature)

# Строим PDF

doc.build(elements)

background\_tasks.add\_task(delete\_file, filepath)

download\_filename = f"pdf\_report\_{employee.FIO.replace(' ', '\_')}.pdf"

return FileResponse(

path=filepath,

filename=download\_filename,

media\_type="application/pdf"

)

except Exception as e:

import traceback

print(traceback.format\_exc())

raise HTTPException(status\_code=500, detail=f"Ошибка генерации PDF отчёта: {str(e)}")

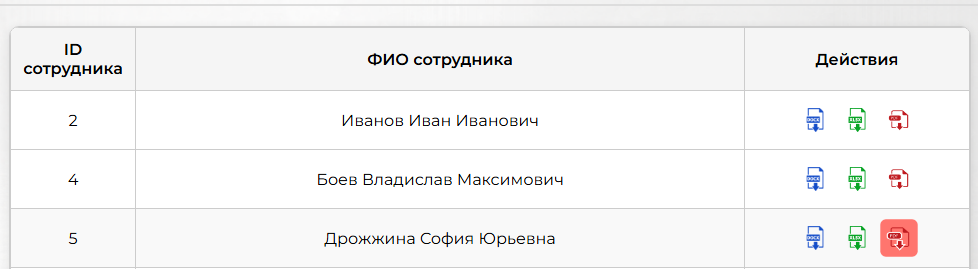


Рис. 4.11. Сохранение отчета PDF соответствующего сотрудника.

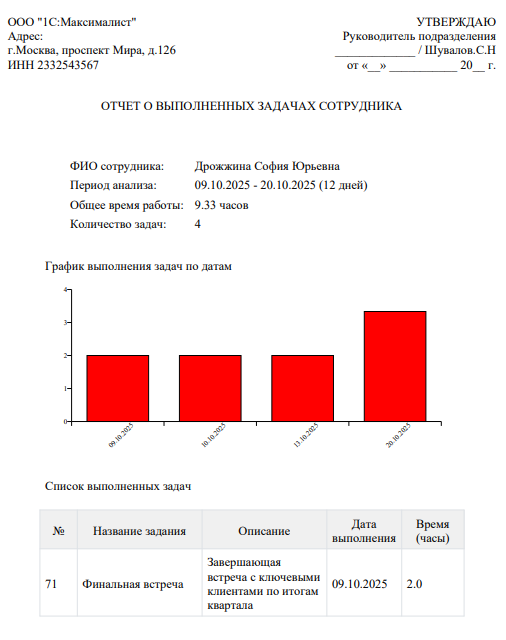


Рис. 4.12. Результат сохранения отчета PDF.

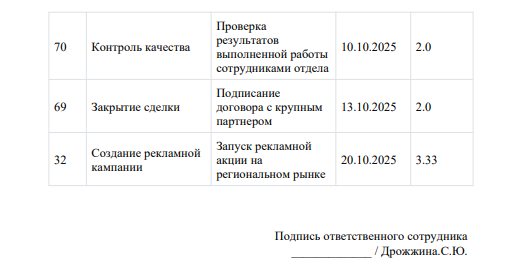


Рис. 4.13. Продолжение результата сохранения отчета PDF.

4.4. Реализация импорта данных из файла в базу данных

Листинг 15 – Функция импорта из Excel в таблицу Task:

@app.post("/api/import/tasks")  
async def import\_tasks(  
 file: UploadFile = File(...),  
 ignore\_duplicates: bool = Form(True),  
 db: Session = Depends(get\_db)  
):  
 """  
 Импорт задач из Excel файла  
 """  
 if not file.filename.endswith(('.xlsx', '.xls')):  
 raise HTTPException(status\_code=400, detail="Формат файла должен быть .xlsx или .xls")  
  
 try:  
 # Читаем файл  
 contents = await file.read()  
 df = pd.read\_excel(io.BytesIO(contents))  
  
 # Нормализуем названия колонок  
 df.columns = df.columns.str.strip().str.lower()  
  
 # Ищем необходимые колонки  
 name\_columns = [col for col in df.columns if any(x in col for x in ['name', 'название', 'задача'])]  
 emp\_columns = [col for col in df.columns if  
 any(x in col for x in ['emp', 'employee', 'сотрудник', 'fio', 'фио'])]  
  
 if not name\_columns:  
 raise HTTPException(status\_code=400, detail="Не найдена колонка с названием задачи")  
 if not emp\_columns:  
 raise HTTPException(status\_code=400, detail="Не найдена колонка с сотрудником")  
  
 name\_column = name\_columns[0]  
 emp\_column = emp\_columns[0]  
  
 # Ищем опциональные колонки  
 desc\_column = next((col for col in df.columns if any(x in col for x in ['desc', 'description', 'описание'])),  
 None)  
 start\_column = next((col for col in df.columns if any(x in col for x in ['start', 'начало'])), None)  
 final\_column = next((col for col in df.columns if any(x in col for x in ['final', 'end', 'конец'])), None)  
 date\_column = next((col for col in df.columns if any(x in col for x in ['date', 'дата'])), None)  
  
 imported\_count = 0  
 skipped\_count = 0  
 errors = []  
  
 for index, row in df.iterrows():  
 try:  
 name\_value = str(row[name\_column]).strip()  
 emp\_value = str(row[emp\_column]).strip()  
  
 if not name\_value or pd.isna(name\_value) or not emp\_value or pd.isna(emp\_value):  
 skipped\_count += 1  
 continue  
  
 # Ищем сотрудника по ФИО или ID  
 employee = None  
 if emp\_value.isdigit():  
 employee = db.query(Employee).filter(Employee.empId == int(emp\_value)).first()  
 else:  
 employee = db.query(Employee).filter(  
 func.lower(Employee.FIO) == func.lower(emp\_value)  
 ).first()  
  
 if not employee:  
 errors.append(f"Строка {index + 2}: Сотрудник '{emp\_value}' не найден")  
 skipped\_count += 1  
 continue  
  
 # Проверяем дубликаты  
 if ignore\_duplicates:  
 existing\_task = db.query(Task).filter(  
 func.lower(Task.name) == func.lower(name\_value),  
 Task.empId == employee.empId  
 ).first()  
 if existing\_task:  
 skipped\_count += 1  
 continue  
  
 # Создаем задачу  
 new\_task = Task(  
 name=name\_value,  
 description=str(row[desc\_column]).strip() if desc\_column and not pd.isna(  
 row[desc\_column]) else None,  
 empId=employee.empId,  
 date=dt.date.today()  
 )  
  
 # Обрабатываем время начала  
 if start\_column and not pd.isna(row[start\_column]):  
 if isinstance(row[start\_column], str):  
 try:  
 new\_task.start = datetime.strptime(row[start\_column], '%H:%M:%S').time()  
 except ValueError:  
 try:  
 new\_task.start = datetime.strptime(row[start\_column], '%H:%M').time()  
 except ValueError:  
 pass  
 elif isinstance(row[start\_column], datetime):  
 new\_task.start = row[start\_column].time()  
  
 # Обрабатываем время окончания  
 if final\_column and not pd.isna(row[final\_column]):  
 if isinstance(row[final\_column], str):  
 try:  
 new\_task.final = datetime.strptime(row[final\_column], '%H:%M:%S').time()  
 except ValueError:  
 try:  
 new\_task.final = datetime.strptime(row[final\_column], '%H:%M').time()  
 except ValueError:  
 pass  
 elif isinstance(row[final\_column], datetime):  
 new\_task.final = row[final\_column].time()  
  
 # Обрабатываем дату  
 if date\_column and not pd.isna(row[date\_column]):  
 if isinstance(row[date\_column], str):  
 try:  
 new\_task.date = datetime.strptime(row[date\_column], '%Y-%m-%d').date()  
 except ValueError:  
 try:  
 new\_task.date = datetime.strptime(row[date\_column], '%d.%m.%Y').date()  
 except ValueError:  
 pass  
 elif isinstance(row[date\_column], datetime):  
 new\_task.date = row[date\_column].date()  
  
 db.add(new\_task)  
 imported\_count += 1  
  
 except Exception as e:  
 errors.append(f"Строка {index + 2}: {str(e)}")  
 skipped\_count += 1  
  
 db.commit()  
  
 return ImportResult(  
 total\_rows=len(df),  
 imported\_rows=imported\_count,  
 skipped\_rows=skipped\_count,  
 errors=errors  
 )  
  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=500, detail=f"Ошибка импорта: {str(e)}")

4.5. Загрузка приложения в репозиторий GitHub



Рис. 4.14. Загруженное приложение в репозиторий

5. Тестирование и отладка

5.1. Проведение модульного тестирования

5.2. Отладка программы

6. Разработка мобильной версии

7. Разработка технической документации

7.1. Руководство пользователя

7.2. Руководство программиста

7.3. Руководство системного программиста

7.4. Методика тестирования и испытания программы

Заключение

Список литературы