**Содержание**

# Введение

Предметная область

# 1. Разработка технического задания

# 2. Проектирование интерфейса приложения

Макет был создан в приложении Figma.

Figma — онлайн-программа для дизайна интерфейсов для сайтов, мобильных приложений, презентаций и маркетинговых материалов.

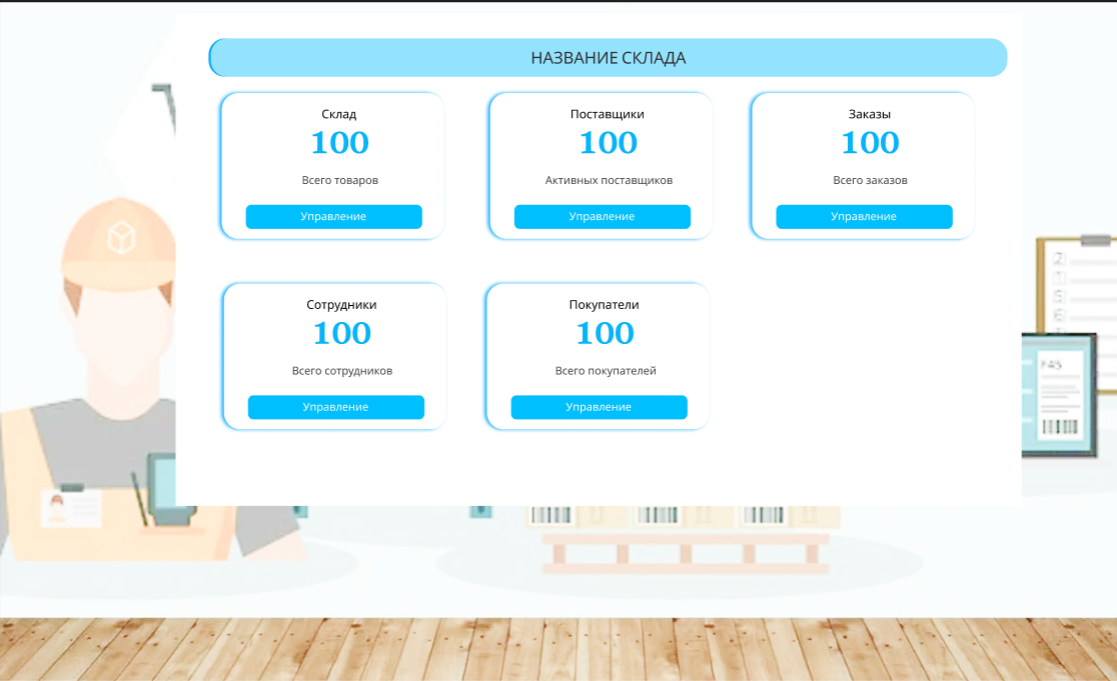


Рис.1 «Главное меню управления складом»

Главное меню представляет собой страницу с 5 разделами, которые переводят на другие страницы. На заднем плане расположено изображение склада. Также в верхней части экрана расположено название.

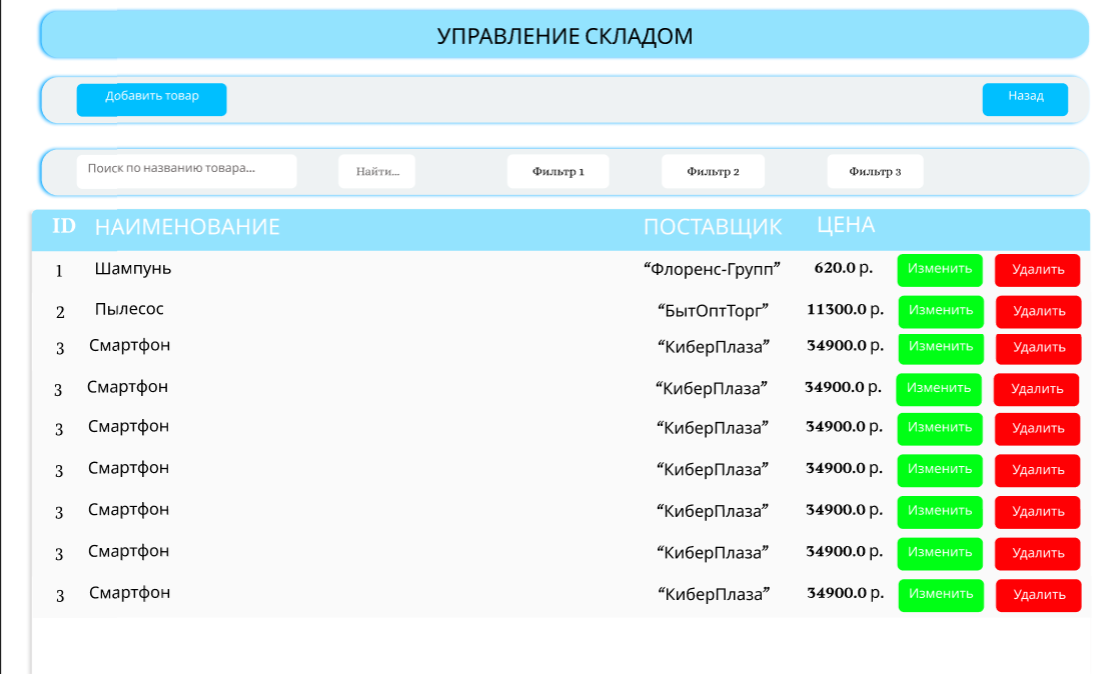


Рис.2 «Управление товарами»

На данной странице можно управлять товарами, добавлять новые, редактировать и удалять имеющиеся. В таблице расположены названия товара, их цена и поставщик. Также сверху расположен поиск для удобства поиска товаров с фильтрацией.

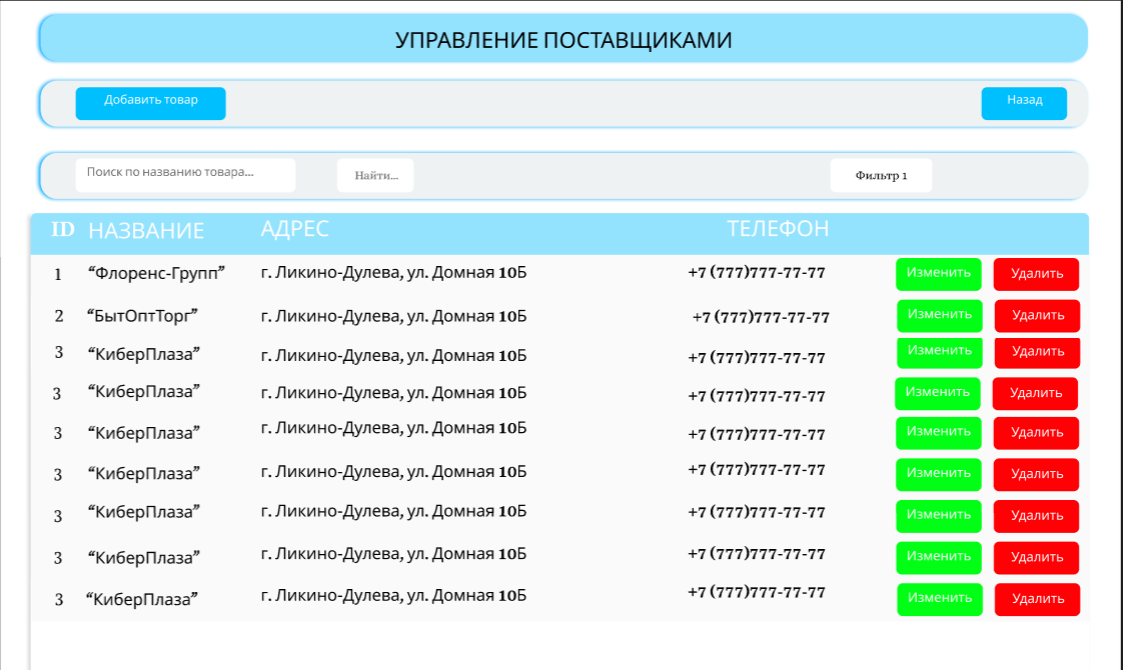


Рис.3 «Управление поставщиками»

На данной странице можно управлять поставщиками. Когда заключается контракт, вносится запись с адресом, номером и названием компании, что поставляет товар. Имеющиеся данные можно редактировать и удалять. Сверху также расположены поиск и фильтрация.

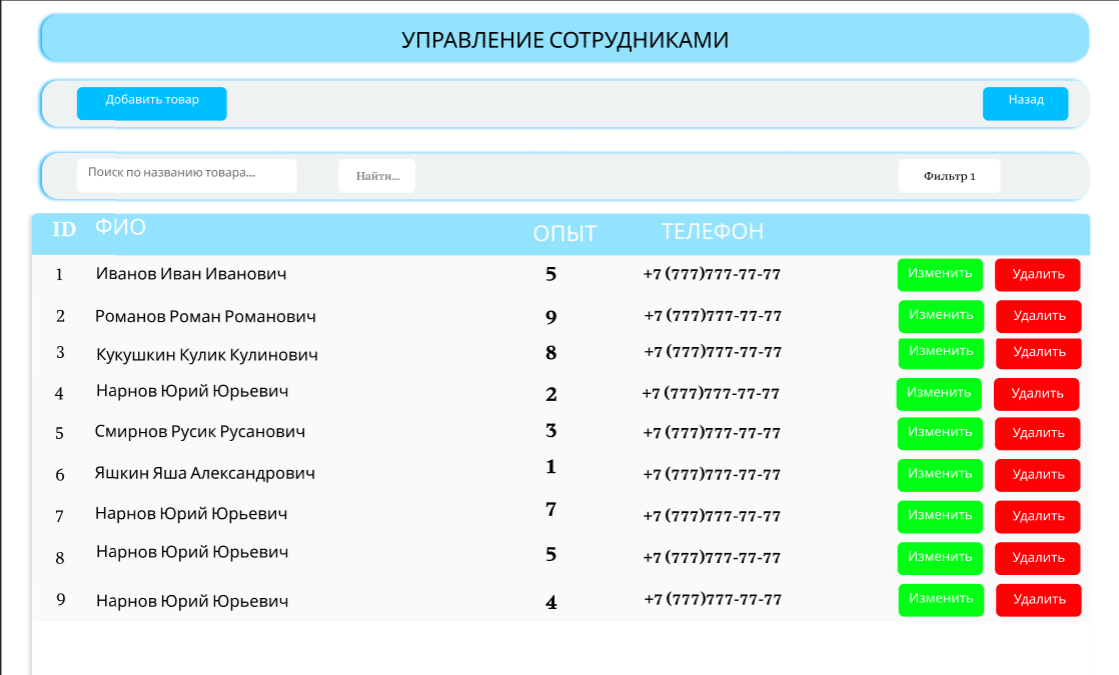


Рис.4 «Управление сотрудниками»

На данной странице можно управлять данными о сотрудниках. При поступлении на работу нового человека, вносится запись с ФИО, номером телефона. Также по мере приобретения стажа, он будет записываться в соответствующую колонку. Сверху также расположен поиск и фильтрация.

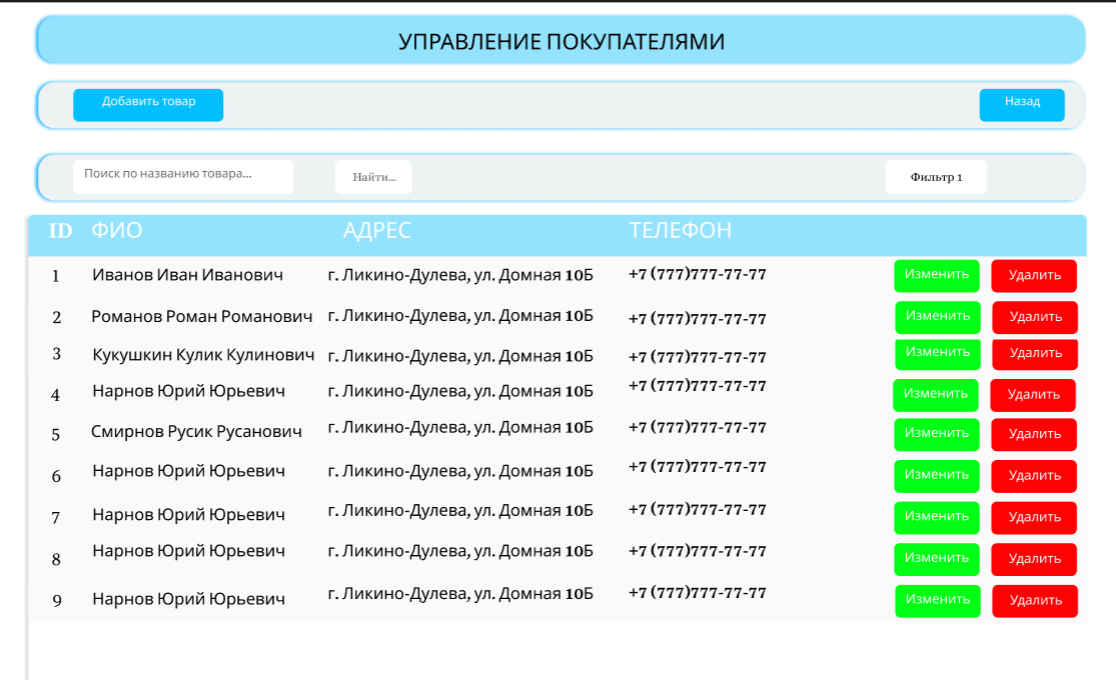


Рис.5 «Управление покупателями»

На данной странице можно управлять данными о покупателях. При поступлении заказа вносится данные о ФИО, адресе, номере телефона клиента. Можно изменять данные и удалять. Также сверху находятся поиск и фильтрация.

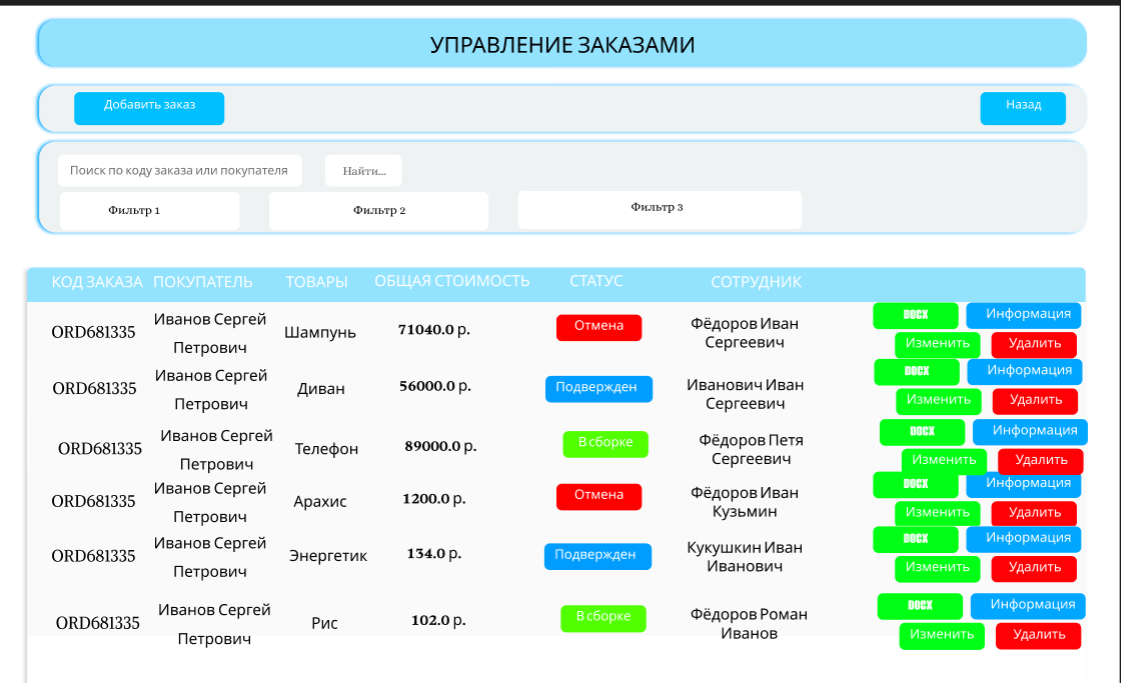


Рис.6 «Управление заказами»

На данной странице можно управлять данными о заказе. Когда клиент заказывает определенный товар, он вносится в таблицу. Также в таблицу вносится ФИО клиента, затем случайно выбирается абсолютно новый идентификатор. После разговора с клиентом, вписывается ФИО сотрудника, что проводит данный заказ, а также нажав на кнопку «Информация», можно поставить определенный статус заказа и написать дополнительные сведения к заказу для других работников. Также в дополнительной информации можно увидеть номер телефона с адресом клиента. Можно также скачать отчет по заказу в word. Сверху страницы находится кнопка «Назад» для возврата на главный экран, фильтрация и функция поиска.

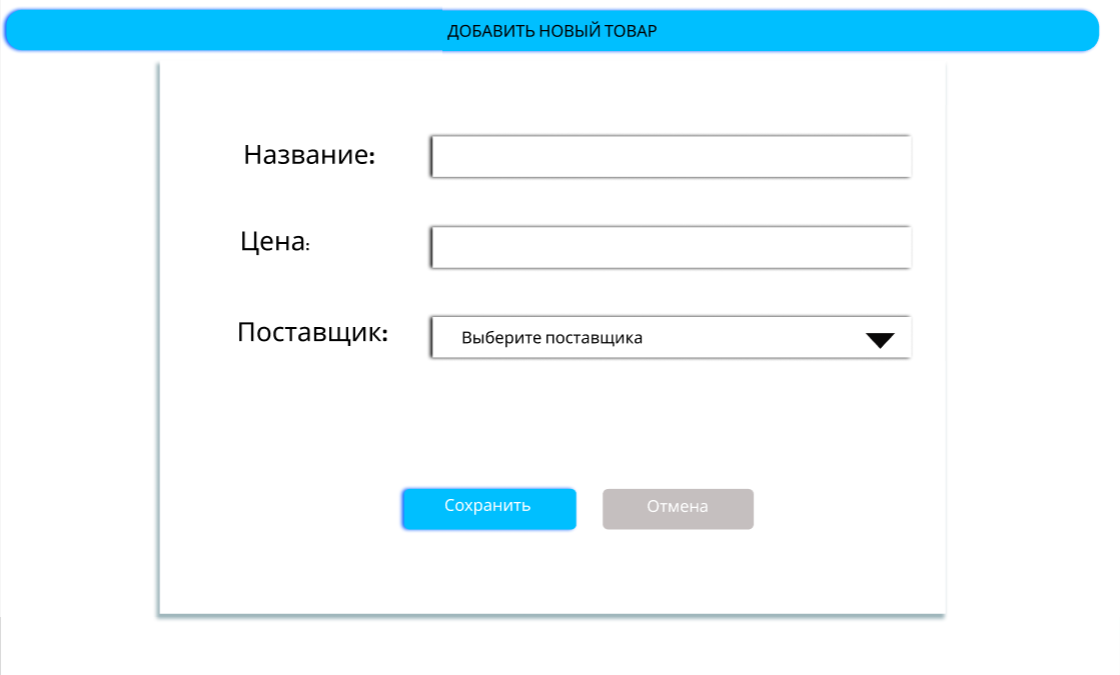


Рис.7 «Добавление новой записи»

На данной страниц отображена система добавления новой записи, которая будет отличаться лишь заполняемыми данными для всех остальных категорий. Когда данные будут заполнены, нужно нажать кнопку «Сохранить» и новые данные выведутся в таблице. Кнопка «Отмена» нужна для отмены добавления новой записи и перехода на страницу заполняемой категории.

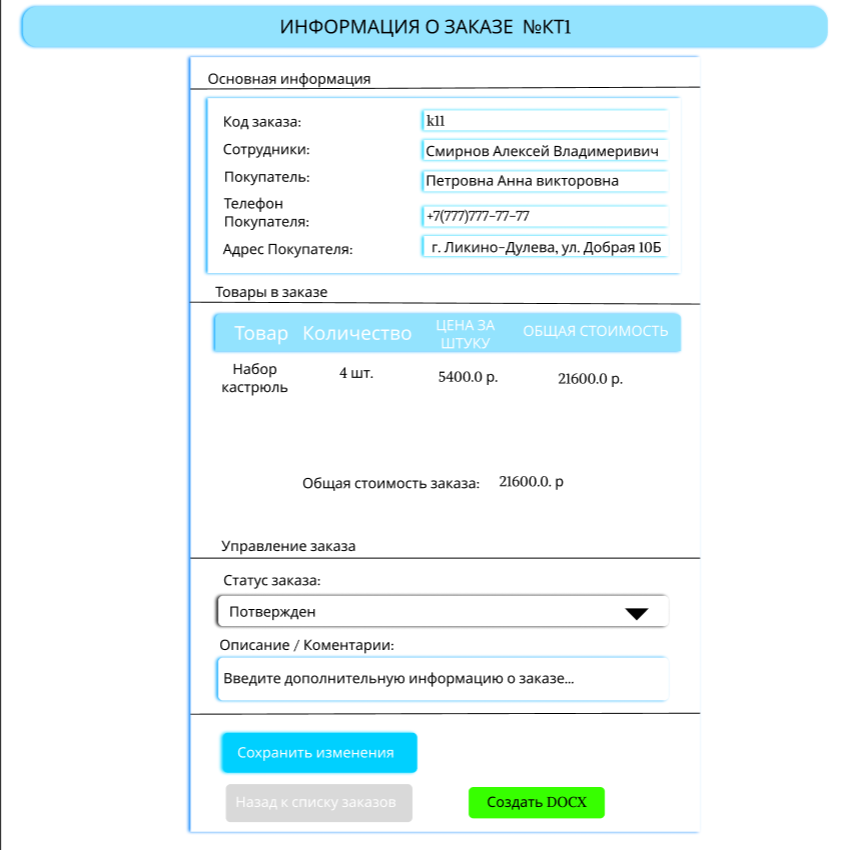


Рис.8 «Информация о заказе»

На данной странице можно увидеть сведения о заказе. Здесь можно заполнить сведения/дополнительную информацию и установить определенный статус.

# 3. Проектирование и создание базы данных

## 3.1. Разработка структуры БД

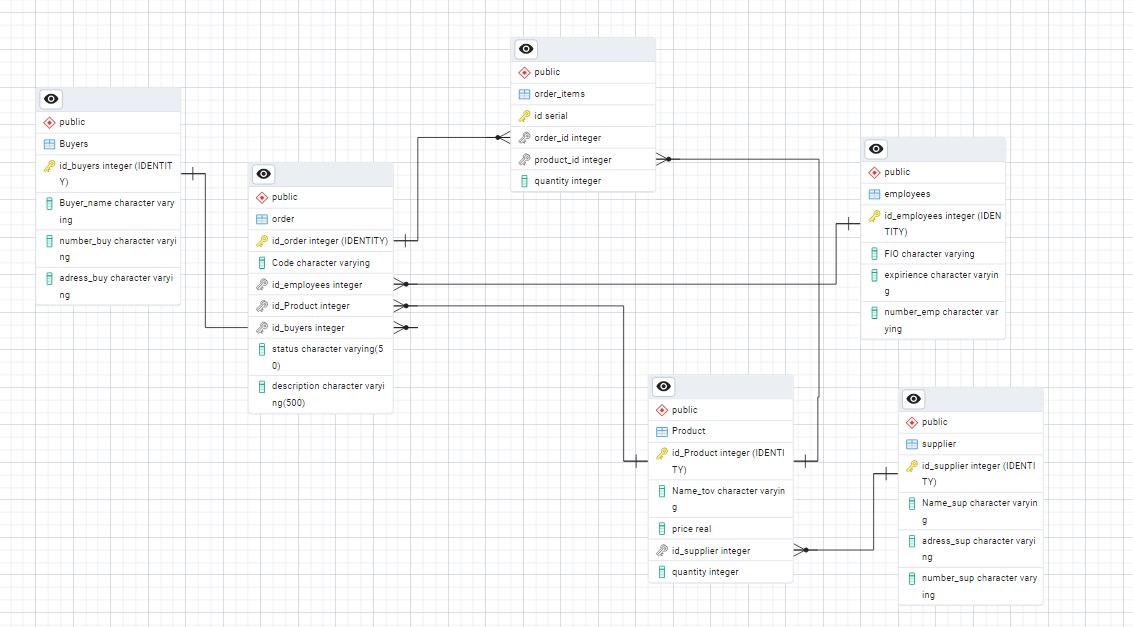


Рис.9 «ER-диаграмма: управление складом»

Таблица «Покупатели» нужна для хранения данных о всех клиентах, что обслужил склад. Он состоит из:

Id\_buyers – первичный ключ в таблице. Тип данных integer, not null, primary key

Buyer\_name – ФИО покупателя. Тип данных character varying(255)

Number\_buy — Номер телефона покупателя. Тип данных character varying(12)

Address\_buy — Адрес покупателя. Тип данных character varying(255)

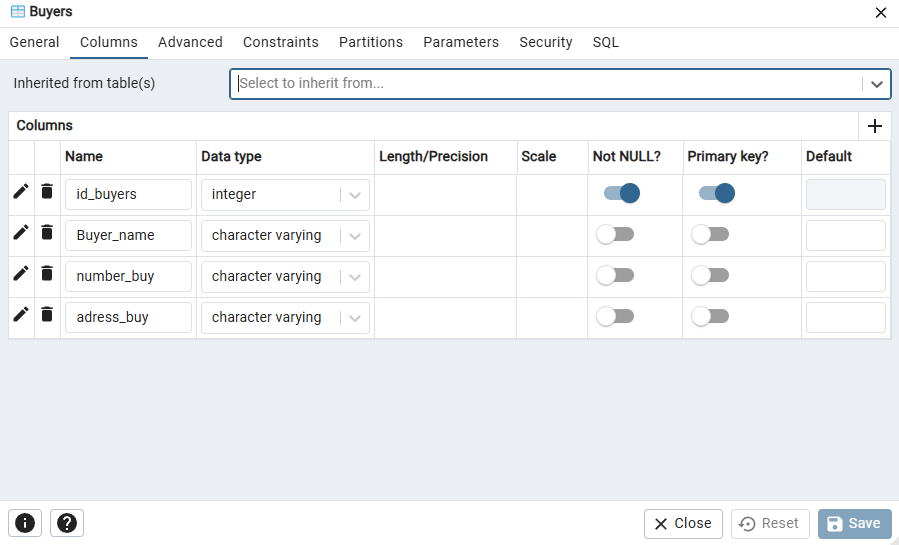


Рис.10 «Таблица “Покупатели”»

Таблица «Товары» нужна для хранения списка товаров. Состоит из:

Id\_Product – Первичный ключ товара. Тип данных integer, not null, primary key

Name\_tov – Название товара. Тип данных character varying(255)

Price – Цена товара. Тип данных real

Id\_supplier – Вторичный ключ, принадлежащий таблице «Поставщики». Тип данных integer.

Quantity – Количество товара на складе. Тип данных integer, default 0

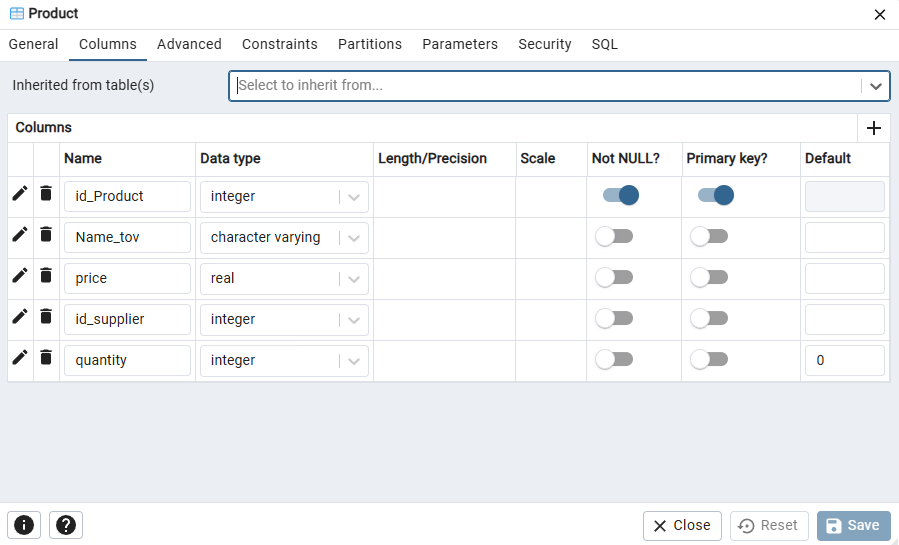


Рис.11 «Таблица “Товары”»

Таблица «Сотрудники» нужна для хранения данных о сотрудниках. Состоит из:

Id\_employees – Первичный ключ таблицы. Тип данных integer, not null, primary key

FIO – ФИО сотрудника. Тип данных character varying(255)

Experience – Стаж сотрудника. Тип данных character varying(2)

Number\_emp – Номер телефона сотрудника. Тип данных character varying(12)

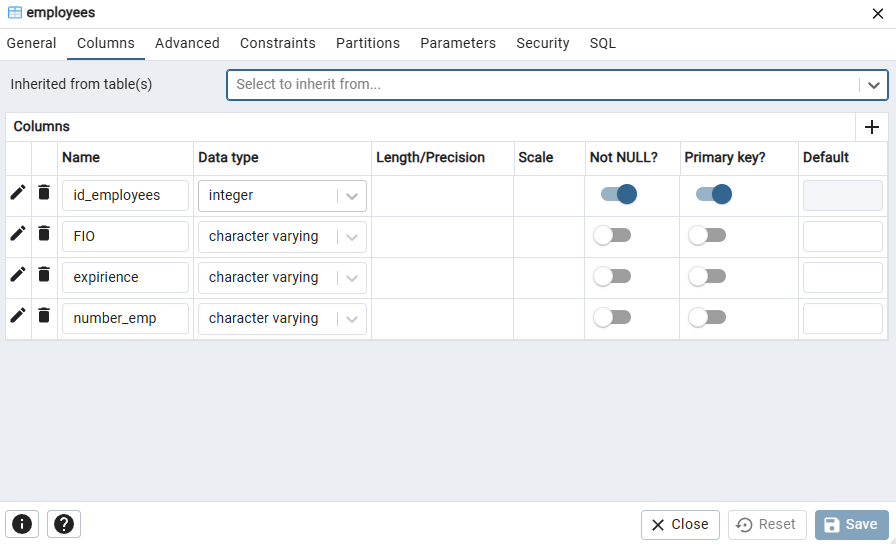


Рис.12 «Таблица “Сотрудники”»

Таблица «Заказы» нужна для хранения данных о заказах клиентов. Состоит из:

Id\_order – Первичный ключ таблицы. Тип данных integer, not null, primary key

Code – Уникальный идентификатор заказа. Тип данных character varying(255)

Id\_employees – Вторичный ключ, принадлежащий таблице «Сотрудники». Тип данных integer.

Id\_Product – Вторичный ключ, принадлежащий таблице «Товары». Тип данных integer.

Id\_buyers – Вторичный ключ, принадлежащий таблице «Покупатели». Тип данных integer.

Status – Статус заказа. Тип данных character varying(255), length/precision 50, default 'В обработке'::character varying

Descriprtion – Дополнительная информация о заказе Тип данных character varying(255), length/precision 500

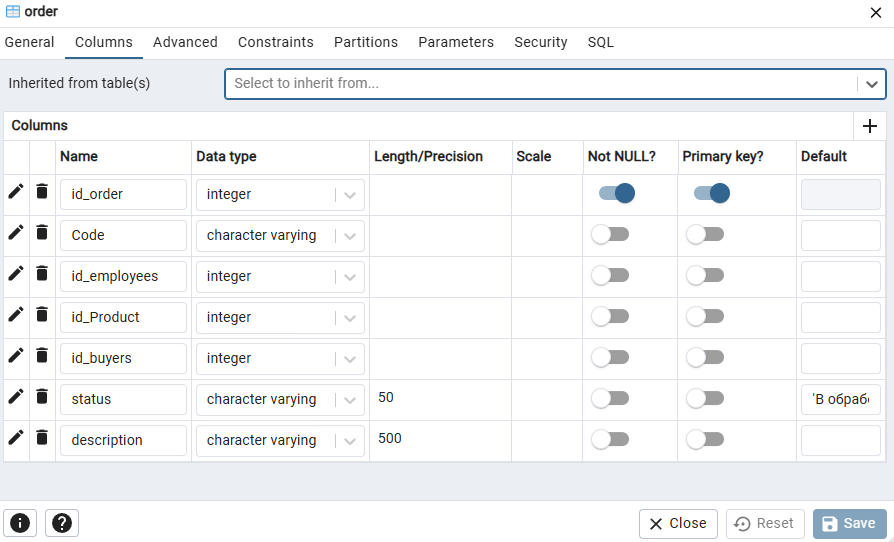


Рис.13 «Таблица “Заказы”»

Таблица «Товары в заказе» нужна для хранения связки "заказ-товар-количество". То есть один заказ может содержать много товаров, и один товар может присутствовать во многих заказах. Состоит из:

Id – Первичный ключ таблицы. Тип данных integer, not null, primary key, default nextval('order\_items\_id\_seq'::regclass)

Id\_order - Вторичный ключ, принадлежащий таблице «Заказы». Тип данных integer.

Id\_Product - Вторичный ключ, принадлежащий таблице «Товары». Тип данных integer.

Quantity - Количество конкретного товара в конкретном заказе. Тип данных integer.

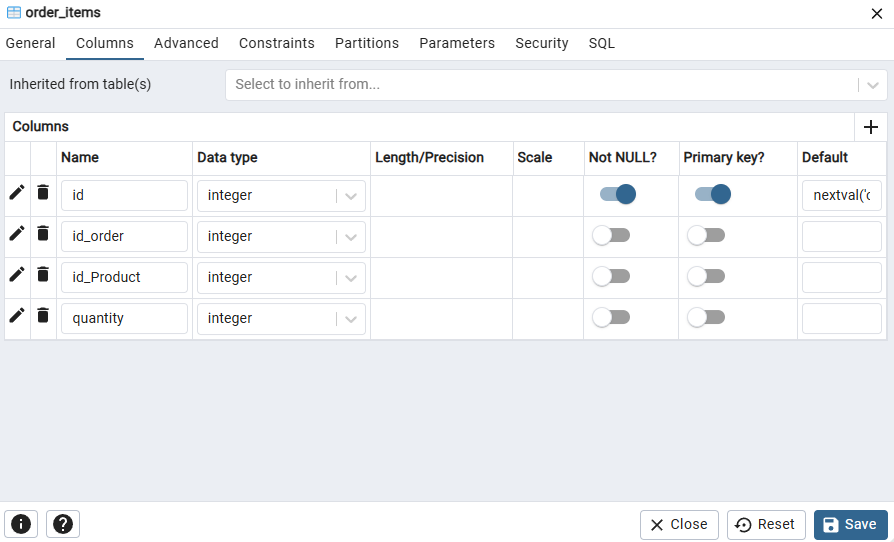


Рис.14 «Таблица “Товары в заказе”»

Таблица «Поставщики» нужна для хранения данных о поставщиках. Состоит из:

Id\_supplier - Первичный ключ таблицы. Тип данных integer, not null, primary key

Name\_sup – Название компании. Тип данных character varying(255), not null

Adress\_sup – Адрес компании. Тип данных character varying(255)

Number\_sup – рабочий номер телефона компании. Тип данных character varying(12)

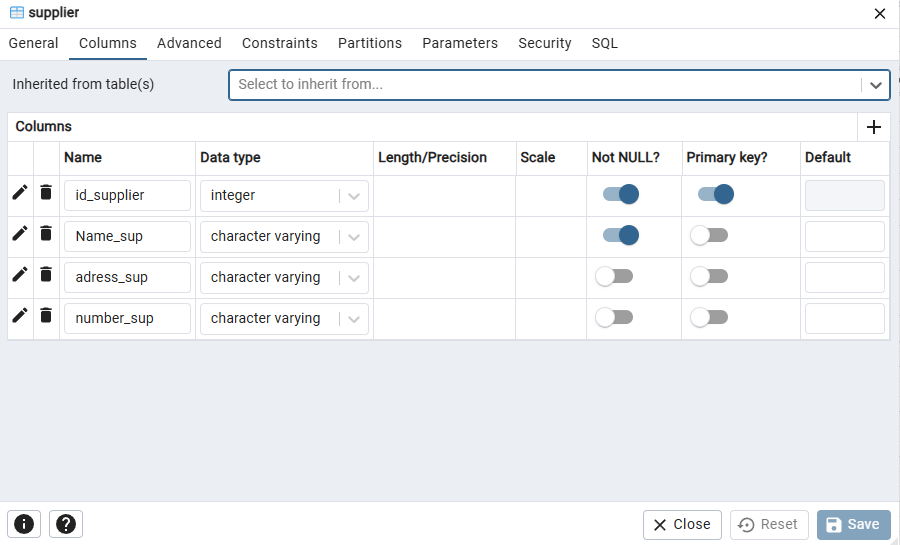


Рис.15 «Таблица “Поставщики”»

## 3.2. Создание модели базы данных

- модуль классов. Файл «models.py».

Листинг 1 – класс «Поставщик»

class Supplier(Base):  
 \_\_tablename\_\_ = "supplier"  
 id\_supplier = Column(Integer, primary\_key=True, index=True, autoincrement=True)  
 Name\_sup = Column(String(45), nullable=False)  
 adress\_sup = Column(String(80))  
 number\_sup = Column(String(12))

Листинг 2 – класс «Товар»

class Product(Base):  
 \_\_tablename\_\_ = "Product"  
 id\_Product = Column(Integer, primary\_key=True, index=True, autoincrement=True)  
 Name\_tov = Column(String(45), nullable=False)  
 price = Column(Float)  
 id\_supplier = Column(Integer, ForeignKey("supplier.id\_supplier"))  
 quantity = Column(Integer, default=0)  
  
 supplier = relationship("Supplier")

Листинг 3 – класс «Покупатели»

class Buyer(Base):  
 \_\_tablename\_\_ = "Buyers"  
 id\_buyers = Column(Integer, primary\_key=True, index=True)  
 Buyer\_name = Column(String(45), nullable=False)  
 number\_buy = Column(String(12))  
 adress\_buy = Column(String(45))

Листинг 4 – класс «Сотрудники»

class Employee(Base):  
 \_\_tablename\_\_ = "employees"  
 id\_employees = Column(Integer, primary\_key=True, index=True, autoincrement=True)  
 FIO = Column(String(45), nullable=False)  
 expirience = Column(String(45))  
 number\_emp = Column(String(12))

Листинг 5 – класс «Заказы»

class Order(Base):  
 \_\_tablename\_\_ = "order"  
 id\_order = Column(Integer, primary\_key=True, index=True, autoincrement=True)  
 Code = Column(String(45))  
 id\_employees = Column(Integer, ForeignKey("employees.id\_employees"))  
 id\_buyers = Column(Integer, ForeignKey("Buyers.id\_buyers"))  
 status = Column(String(50), default="В обработке")  
 description = Column(String(500))  
 employee = relationship("Employee")  
 buyer = relationship("Buyer")  
 items = relationship("OrderItem", back\_populates="order")

Листинг 6 – класс «Товары в заказе»

class OrderItem(Base):  
 \_\_tablename\_\_ = "order\_items"  
 id = Column(Integer, primary\_key=True, index=True, autoincrement=True)  
 order\_id = Column(Integer, ForeignKey("order.id\_order"))  
 product\_id = Column(Integer, ForeignKey("Product.id\_Product"))  
 quantity = Column(Integer, default=1)  
 product = relationship("Product")  
 order = relationship("Order")

# 4. Разработка программных модулей

## 4.1. Создание сервера

- модуль подключения к базе данных находится в файле «database.py»

Листинг 7 – Подключение базы данных к коду

from sqlalchemy import create\_engine  
from sqlalchemy.ext.declarative import declarative\_base  
from sqlalchemy.orm import sessionmaker  
DATABASE\_URL = "postgresql+psycopg2://postgres:1234@localhost:5432/Warehouse"  
engine = create\_engine(DATABASE\_URL)  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
Base = declarative\_base()

- модуль функций доступа к данным API (GET, POST, PUT, DELETE), находится в файле «api.py»

Главное меню

@app.get("/", response\_class=HTMLResponse)  
def home(request: Request, db: Session = Depends(get\_db)):  
 warehouse\_count = db.query(models.Product).count()  
 supplier\_count = db.query(models.Supplier).count()  
 order\_count = db.query(models.Order).count()  
 employee\_count = db.query(models.Employee).count()  
 buyer\_count = db.query(models.Buyer).count()  
 return templates.TemplateResponse("index.html", {  
 "request": request,  
 "warehouse\_count": warehouse\_count,  
 "supplier\_count": supplier\_count,  
 "order\_count": order\_count,  
 "employee\_count": employee\_count,  
 "buyer\_count": buyer\_count  
 })

## Товары

1. Получение списка товаров (get)

@app.get("/warehouse", response\_class=HTMLResponse)  
def warehouse\_page(request: Request, db: Session = Depends(get\_db)):  
 products = db.query(models.Product).all()  
 *# Получаем уникальных поставщиков для фильтра* unique\_suppliers = db.query(models.Supplier.Name\_sup).distinct().all()  
 unique\_suppliers = [supplier[0] for supplier in unique\_suppliers if supplier[0]]  
  
 return templates.TemplateResponse("warehouse.html", {  
 "request": request,  
 "products": products,  
 "unique\_suppliers": sorted(unique\_suppliers)  
 })

2. Получение данных для формы добавления товара (get)

@app.get("/warehouse/add", response\_class=HTMLResponse)  
def add\_warehouse\_page(request: Request, db: Session = Depends(get\_db)):  
 suppliers = db.query(models.Supplier).all()  
 return templates.TemplateResponse("warehouse\_add.html", {  
 "request": request,  
 "suppliers": suppliers  
 })

3. Создание нового товара (post)

@app.post("/warehouse/add")  
async def add\_product(  
 request: Request,  
 db: Session = Depends(get\_db)  
):  
 try:  
 form\_data = await request.form()  
 name = form\_data.get("name")  
 price = float(form\_data.get("price"))  
 supplier\_id = int(form\_data.get("supplier\_id"))  
 quantity = int(form\_data.get("quantity", 0))  
 product = models.Product(  
 Name\_tov=name,  
 price=price,  
 id\_supplier=supplier\_id,  
 quantity=quantity  
 )  
 db.add(product)  
 db.commit()  
 return RedirectResponse(url="/warehouse", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

4. Получение данных для формы редактирования товара (get)

@app.get("/product/{product\_id}/change", response\_class=HTMLResponse)  
def change\_product\_page(request: Request, product\_id: int, db: Session = Depends(get\_db)):  
 product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 suppliers = db.query(models.Supplier).all()  
 if not product:  
 raise HTTPException(status\_code=404, detail="Product not found")  
 return templates.TemplateResponse("product\_change.html", {  
 "request": request,  
 "product": product,  
 "suppliers": suppliers  
 })

5. Редактирование нового товара (post)

@app.post("/product/{product\_id}/change")  
async def change\_product(  
 product\_id: int,  
 request: Request,  
 db: Session = Depends(get\_db)  
):  
 try:  
 form\_data = await request.form()  
 name = form\_data.get("name")  
 price = float(form\_data.get("price"))  
 quantity = int(form\_data.get("quantity", 0))  
 supplier\_id = int(form\_data.get("supplier\_id"))  
 product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 if product:  
 product.Name\_tov = name  
 product.price = price  
 product.quantity = quantity  
 product.id\_supplier = supplier\_id  
 db.commit()  
 return RedirectResponse(url="/warehouse", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

6. Удаление товара

@app.post("/product/{product\_id}/delete")  
def delete\_product(product\_id: int, db: Session = Depends(get\_db)):  
 try:  
 product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 if product:  
 *# Проверяем, нет ли заказов с этим товаром* order\_items\_count = db.query(models.OrderItem).filter(models.OrderItem.product\_id == product\_id).count()  
 if order\_items\_count > 0:  
 raise HTTPException(status\_code=400, detail="Нельзя удалить товар, который есть в заказах")  
  
 db.delete(product)  
 db.commit()  
 return RedirectResponse(url="/warehouse", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

Поставщики

1. Получение списка поставщиков (get)

@app.get("/supplier", response\_class=HTMLResponse)  
def suppliers\_page(request: Request, db: Session = Depends(get\_db)):  
 suppliers = db.query(models.Supplier).all()  
  
 *# Получаем уникальные города из адресов для фильтра* unique\_cities = set()  
 for supplier in suppliers:  
 if supplier.adress\_sup:  
 *# Берем первую часть адреса (обычно город)* city = supplier.adress\_sup.split(',')[0].strip()  
 if city:  
 unique\_cities.add(city)  
  
 return templates.TemplateResponse("supplier.html", {  
 "request": request,  
 "suppliers": suppliers,  
 "unique\_cities": sorted(unique\_cities)  
 })

2. Получение данных для формы добавления нового поставщика (get)

@app.get("/supplier/add", response\_class=HTMLResponse)  
def add\_supplier\_page(request: Request):  
 return templates.TemplateResponse("supplier\_add.html", {"request": request})

3. Модуль функций для добавления нового поставщика (post)

@app.post(“/supplier/add”)  
def add\_supplier(  
 name: str = Form(...),  
 address: str = Form(...),  
 number: str = Form(...),  
 db: Session = Depends(get\_db)  
):  
 try:  
 supplier = models.Supplier(Name\_sup=name, ddress\_sup=address, number\_sup=number)  
 db.add(supplier)  
 db.commit()  
 return RedirectResponse(url=”/supplier”, status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str€)

4. Получение данных для формы редактирования поставщиков (get)

@app.get("/supplier/{supplier\_id}/change", response\_class=HTMLResponse)  
def change\_supplier\_page(request: Request, supplier\_id: int, db: Session = Depends(get\_db)):  
 supplier = db.query(models.Supplier).filter(models.Supplier.id\_supplier == supplier\_id).first()  
 if not supplier:  
 raise HTTPException(status\_code=404, detail="Supplier not found")  
 return templates.TemplateResponse("supplier\_change.html", {  
 "request": request,  
 "supplier": supplier  
 })

5. Модуль функций для редактирования данных о поставщиков (post)

@app.post("/supplier/{supplier\_id}/change")  
def change\_supplier(  
 supplier\_id: int,  
 name: str = Form(...),  
 address: str = Form(...),  
 number: str = Form(...),  
 db: Session = Depends(get\_db)  
):  
 try:  
 supplier = db.query(models.Supplier).filter(models.Supplier.id\_supplier == supplier\_id).first()  
 if supplier:  
 supplier.Name\_sup = name  
 supplier.adress\_sup = address  
 supplier.number\_sup = number  
 db.commit()  
 return RedirectResponse(url="/supplier", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

6. Удаление данных о поставщике

@app.post("/supplier/{supplier\_id}/delete")  
def delete\_supplier(supplier\_id: int, db: Session = Depends(get\_db)):  
 try:  
 supplier = db.query(models.Supplier).filter(models.Supplier.id\_supplier == supplier\_id).first()  
 if supplier:  
 *# Проверяем, нет ли товаров у этого поставщика* products\_count = db.query(models.Product).filter(models.Product.id\_supplier == supplier\_id).count()  
 if products\_count > 0:  
 raise HTTPException(status\_code=400, detail="Нельзя удалить поставщика, у которого есть товары")  
 db.delete(supplier)  
 db.commit()  
 return RedirectResponse(url="/supplier", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

Сотрудники

1. Получаем данные об сотрудниках (get)

@app.get("/employees", response\_class=HTMLResponse)  
def employees\_page(request: Request, db: Session = Depends(get\_db)):  
 employees = db.query(models.Employee).all()  
  
 return templates.TemplateResponse("employees.html", {  
 "request": request,  
 "employees": employees  
 })

2. Получение данных для формы добавления нового сотрудника (get)

@app.get("/employees/add", response\_class=HTMLResponse)  
def add\_employee\_page(request: Request):  
 return templates.TemplateResponse("employee\_add.html", {"request": request})

3. Модуль функций для добавления нового сотрудника (post)

@app.post("/employees/add")  
def add\_employee(  
 fio: str = Form(...),  
 experience: str = Form(...),  
 number: str = Form(...),  
 db: Session = Depends(get\_db)  
):  
 try:  
 employee = models.Employee(FIO=fio, expirience=experience, number\_emp=number)  
 db.add(employee)  
 db.commit()  
 return RedirectResponse(url="/employees", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

4. Получение данных для формы редактирования сотрудников (get)

@app.get("/employee/{employee\_id}/change", response\_class=HTMLResponse)  
def change\_employee\_page(request: Request, employee\_id: int, db: Session = Depends(get\_db)):  
 employee = db.query(models.Employee).filter(models.Employee.id\_employees == employee\_id).first()  
 if not employee:  
 raise HTTPException(status\_code=404, detail="Employee not found")  
 return templates.TemplateResponse("employee\_change.html", {  
 "request": request,  
 "employee": employee  
 })

5. Модуль функций для редактирования данных о сотрудниках (post)

@app.post("/employee/{employee\_id}/change")  
def change\_employee(  
 employee\_id: int,  
 fio: str = Form(...),  
 experience: str = Form(...),  
 number: str = Form(...),  
 db: Session = Depends(get\_db)  
):  
 try:  
 employee = db.query(models.Employee).filter(models.Employee.id\_employees == employee\_id).first()  
 if employee:  
 employee.FIO = fio  
 employee.expirience = experience  
 employee.number\_emp = number  
 db.commit()  
 return RedirectResponse(url="/employees", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

6. Удаление данных о сотруднике

@app.post("/employee/{employee\_id}/delete")  
def delete\_employee(employee\_id: int, db: Session = Depends(get\_db)):  
 try:  
 employee = db.query(models.Employee).filter(models.Employee.id\_employees == employee\_id).first()  
 if employee:  
 *# Проверяем, нет ли заказов у этого сотрудника* orders\_count = db.query(models.Order).filter(models.Order.id\_employees == employee\_id).count()  
 if orders\_count > 0:  
 raise HTTPException(status\_code=400, detail="Нельзя удалить сотрудника, у которого есть заказы")  
 db.delete(employee)  
 db.commit()  
 return RedirectResponse(url="/employees", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

Покупатели

1. Получение списка покупателей

@app.get("/buyers", response\_class=HTMLResponse)  
def buyers\_page(request: Request, db: Session = Depends(get\_db)):  
 buyers = db.query(models.Buyer).all()  
 *# Получаем уникальные города из адресов для фильтра* unique\_cities = set()  
 for buyer in buyers:  
 if buyer.adress\_buy:  
 *# Берем первую часть адреса (обычно город)* city = buyer.adress\_buy.split(',')[0].strip()  
 if city:  
 unique\_cities.add(city)  
 return templates.TemplateResponse("buyers.html", {  
 "request": request,  
 "buyers": buyers,  
 "unique\_cities": sorted(unique\_cities)  
 })

2. Получение данных для формы добавления нового покупателя (get)

@app.get("/buyers/add", response\_class=HTMLResponse)  
def add\_buyer\_page(request: Request):  
 return templates.TemplateResponse("buyer\_add.html", {"request": request})

3. Модуль функций для добавления нового покупателя (post)

@app.post("/buyers/add")  
def add\_buyer(  
 name: str = Form(...),  
 number: str = Form(...),  
 address: str = Form(...),  
 db: Session = Depends(get\_db)  
):  
 try:  
 buyer = models.Buyer(Buyer\_name=name, number\_buy=number, adress\_buy=address)  
 db.add(buyer)  
 db.commit()  
 return RedirectResponse(url="/buyers", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

4. Получение данных для формы редактирования покупателей (get)

@app.get("/buyer/{buyer\_id}/change", response\_class=HTMLResponse)  
def change\_buyer\_page(request: Request, buyer\_id: int, db: Session = Depends(get\_db)):  
 buyer = db.query(models.Buyer).filter(models.Buyer.id\_buyers == buyer\_id).first()  
 if not buyer:  
 raise HTTPException(status\_code=404, detail="Buyer not found")  
 return templates.TemplateResponse("buyer\_change.html", {  
 "request": request,  
 "buyer": buyer  
 })

5. Модуль функций для редактирования данных о покупателей (post)

@app.post("/buyer/{buyer\_id}/change")  
def change\_buyer(  
 buyer\_id: int,  
 name: str = Form(...),  
 number: str = Form(...),  
 address: str = Form(...),  
 db: Session = Depends(get\_db)  
):  
 try:  
 buyer = db.query(models.Buyer).filter(models.Buyer.id\_buyers == buyer\_id).first()  
 if buyer:  
 buyer.Buyer\_name = name  
 buyer.number\_buy = number  
 buyer.adress\_buy = address  
 db.commit()  
 return RedirectResponse(url="/buyers", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

6. Удаление данных о покупателе

@app.post("/buyer/{buyer\_id}/delete")  
def delete\_buyer(buyer\_id: int, db: Session = Depends(get\_db)):  
 try:  
 buyer = db.query(models.Buyer).filter(models.Buyer.id\_buyers == buyer\_id).first()  
 if buyer:  
 *# Проверяем, нет ли заказов у этого покупателя* orders\_count = db.query(models.Order).filter(models.Order.id\_buyers == buyer\_id).count()  
 if orders\_count > 0:  
 raise HTTPException(status\_code=400, detail="Нельзя удалить покупателя, у которого есть заказы")  
 db.delete(buyer)  
 db.commit()  
 return RedirectResponse(url="/buyers", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

Заказы

1. Получение списка заказов (get)

@app.get("/orders", response\_class=HTMLResponse)  
def orders\_page(request: Request, db: Session = Depends(get\_db)):  
 orders = db.query(models.Order).options(  
 joinedload(models.Order.employee),  
 joinedload(models.Order.buyer),  
 joinedload(models.Order.items).joinedload(models.OrderItem.product)  
 ).all()  
 *# Получаем уникальных покупателей и сотрудников для фильтров* unique\_buyers = db.query(models.Buyer.Buyer\_name).distinct().all()  
 unique\_buyers = [buyer[0] for buyer in unique\_buyers if buyer[0]]  
  
 unique\_employees = db.query(models.Employee.FIO).distinct().all()  
 unique\_employees = [employee[0] for employee in unique\_employees if employee[0]]  
 return templates.TemplateResponse("orders.html", {  
 "request": request,  
 "orders": orders,  
 "unique\_buyers": sorted(unique\_buyers),  
 "unique\_employees": sorted(unique\_employees)  
 })

2. Получение данных для формы добавления нового заказа (get)

@app.get("/orders/add", response\_class=HTMLResponse)  
def add\_order\_page(request: Request, db: Session = Depends(get\_db)):  
 employees = db.query(models.Employee).all()  
 products = db.query(models.Product).all()  
 buyers = db.query(models.Buyer).all()  
 *# Генерируем случайный код для формы* random\_code = generate\_order\_code()  
 return templates.TemplateResponse("order\_add.html", {  
 "request": request,  
 "employees": employees,  
 "products": products,  
 "buyers": buyers,  
 "random\_code": random\_code  
 })

3. Модуль функций для добавления нового заказа (post)

@app.post("/orders/add")  
async def add\_order(  
 request: Request,  
 db: Session = Depends(get\_db)  
):  
 try:  
 form\_data = await request.form()  
 *# Извлекаем данные из формы* code = form\_data.get("code")  
 employee\_id = form\_data.get("employee\_id")  
 buyer\_id = form\_data.get("buyer\_id")  
 if not all([code, employee\_id, buyer\_id]):  
 raise HTTPException(status\_code=400, detail="Заполните все обязательные поля")  
 *# Получаем массивы товаров и количеств* product\_ids = form\_data.getlist("product\_id")  
 quantities = form\_data.getlist("quantity")  
 if not product\_ids or not quantities or not product\_ids[0] or not quantities[0]:  
 raise HTTPException(status\_code=400, detail="Необходимо добавить хотя бы один товар")  
 *# Проверка наличия товаров* for i in range(len(product\_ids)):  
 if product\_ids[i] and quantities[i]:  
 product\_id = int(product\_ids[i])  
 order\_quantity = int(quantities[i])  
 product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 if not product:  
 raise HTTPException(status\_code=400, detail=f"Товар с ID {product\_id} не найден")  
 if product.quantity < order\_quantity:  
 raise HTTPException(  
 status\_code=400,  
 detail=f"Недостаточно товара '{product.Name\_tov}'. Доступно: {product.quantity}, заказано: {order\_quantity}"  
 )  
 *# Если все проверки пройдены, создаем заказ* order = models.Order(  
 Code=code,  
 id\_employees=int(employee\_id),  
 id\_buyers=int(buyer\_id)  
 )  
 db.add(order)  
 db.flush() *# Получаем ID заказа  
 # Добавляем товары в заказ и вычитаем из наличия* for i in range(len(product\_ids)):  
 if product\_ids[i] and quantities[i]:  
 product\_id = int(product\_ids[i])  
 order\_quantity = int(quantities[i])  
 product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 product.quantity -= order\_quantity  
 order\_item = models.OrderItem(  
 order\_id=order.id\_order,  
 product\_id=product\_id,  
 quantity=order\_quantity  
 )  
 db.add(order\_item)  
 db.commit()  
 return RedirectResponse(url="/orders", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 print(f"Error: {str(e)}")  
 raise HTTPException(status\_code=400, detail=str(e))

4. Получение данных для формы редактирования заказа (get)

@app.get("/order/{order\_id}/change", response\_class=HTMLResponse)  
def change\_order\_page(request: Request, order\_id: int, db: Session = Depends(get\_db)):  
 order = db.query(models.Order).options(  
 joinedload(models.Order.items)  
 ).filter(models.Order.id\_order == order\_id).first()  
 employees = db.query(models.Employee).all()  
 products = db.query(models.Product).all()  
 buyers = db.query(models.Buyer).all()  
 if not order:  
 raise HTTPException(status\_code=404, detail="Order not found")  
 return templates.TemplateResponse("order\_change.html", {  
 "request": request,  
 "order": order,  
 "employees": employees,  
 "products": products,  
 "buyers": buyers  
 })

5. Модуль функций для редактирования данных о заказе (post)

@app.post("/order/{order\_id}/change")  
async def change\_order(  
 order\_id: int,  
 request: Request,  
 db: Session = Depends(get\_db)  
):  
 try:  
 form\_data = await request.form()  
 *# Извлекаем данные из формы* code = form\_data.get("code")  
 employee\_id = form\_data.get("employee\_id")  
 buyer\_id = form\_data.get("buyer\_id")  
 if not all([code, employee\_id, buyer\_id]):  
 raise HTTPException(status\_code=400, detail="Заполните все обязательные поля")  
 *# Получаем массивы товаров и количеств* product\_ids = form\_data.getlist("product\_id")  
 quantities = form\_data.getlist("quantity")  
 if not product\_ids or not quantities or not product\_ids[0] or not quantities[0]:  
 raise HTTPException(status\_code=400, detail="Необходимо добавить хотя бы один товар")  
 order = db.query(models.Order).options(  
 joinedload(models.Order.items).joinedload(models.OrderItem.product)  
 ).filter(models.Order.id\_order == order\_id).first()  
 if order:  
 *# Возвращаем старые товара на склад* for old\_item in order.items:  
 if old\_item.product:  
 old\_item.product.quantity += old\_item.quantity  
 *# Проверяем наличие новых товаров* for i in range(len(product\_ids)):  
 if product\_ids[i] and quantities[i]:  
 product\_id = int(product\_ids[i])  
 order\_quantity = int(quantities[i])  
 product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 if not product:  
 raise HTTPException(status\_code=400, detail=f"Товар с ID {product\_id} не найден")  
 if product.quantity < order\_quantity:  
 raise HTTPException(  
 status\_code=400,  
 detail=f"Недостаточно товара '{product.Name\_tov}'. Доступно: {product.quantity}, заказано: {order\_quantity}"  
 )  
 order.Code = code  
 order.id\_employees = int(employee\_id)  
 order.id\_buyers = int(buyer\_id)  
 *# Удаляем старые товары* db.query(models.OrderItem).filter(models.OrderItem.order\_id == order\_id).delete()  
 *# Добавляем новые товары и ВЫЧИТАЕМ ИЗ НАЛИЧИЯ* for i in range(len(product\_ids)):  
 if product\_ids[i] and quantities[i]:  
 product\_id = int(product\_ids[i])  
 order\_quantity = int(quantities[i])  
 *# Вычитаем из наличия* product = db.query(models.Product).filter(models.Product.id\_Product == product\_id).first()  
 product.quantity -= order\_quantity  
 order\_item = models.OrderItem(  
 order\_id=order.id\_order,  
 product\_id=product\_id,  
 quantity=order\_quantity  
 )  
 db.add(order\_item)  
 db.commit()  
 return RedirectResponse(url="/orders", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 print(f"Error: {str(e)}")  
 raise HTTPException(status\_code=400, detail=str(e))

6. Модуль функции для вывода всей информации о заказе (get)

@app.get("/order/{order\_id}/intel", response\_class=HTMLResponse)  
def order\_intel\_page(request: Request, order\_id: int, db: Session = Depends(get\_db)):  
 order = db.query(models.Order).options(  
 joinedload(models.Order.employee),  
 joinedload(models.Order.buyer),  
 joinedload(models.Order.items).joinedload(models.OrderItem.product)  
 ).filter(models.Order.id\_order == order\_id).first()  
 if not order:  
 raise HTTPException(status\_code=404, detail="Order not found")  
 return templates.TemplateResponse("order\_intel.html", {  
 "request": request,  
 "order": order  
 })

7. Модуль для редактирования всей информации о заказе (post)

@app.post("/order/{order\_id}/intel")  
def update\_order\_intel(  
 order\_id: int,  
 status: str = Form(...),  
 description: str = Form(None),  
 db: Session = Depends(get\_db)  
):  
 try:  
 order = db.query(models.Order).filter(models.Order.id\_order == order\_id).first()  
 if order:  
 order.status = status  
 order.description = description  
 db.commit()  
  
 return RedirectResponse(url=f"/order/{order\_id}/intel", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

8. Удаление заказа

@app.post("/order/{order\_id}/delete")  
def delete\_order(order\_id: int, db: Session = Depends(get\_db)):  
 try:  
 order = db.query(models.Order).options(  
 joinedload(models.Order.items).joinedload(models.OrderItem.product)  
 ).filter(models.Order.id\_order == order\_id).first()  
 if order:  
 *# Возвращаем товары на склад* for item in order.items:  
 if item.product:  
 item.product.quantity += item.quantity  
 *# Удаляем связанные товары* db.query(models.OrderItem).filter(models.OrderItem.order\_id == order\_id).delete()  
 *# Затем удаляем заказ* db.delete(order)  
 db.commit()  
 return RedirectResponse(url="/orders", status\_code=303)  
 except Exception as e:  
 db.rollback()  
 raise HTTPException(status\_code=400, detail=str(e))

Word

1. Генерация файла формата docx (get)

@app.get("/order/{order\_id}/docx")  
def download\_order\_docx(order\_id: int, db: Session = Depends(get\_db)):  
 try:  
 *# Получаем заказ с связанными данными* order = db.query(models.Order).options( joinedload(models.Order.items).joinedload(models.OrderItem.product),  
 joinedload(models.Order.buyer),  
 joinedload(models.Order.employee)  
 ).filter(models.Order.id\_order == order\_id).first()  
 if not order:  
 raise HTTPException(status\_code=404, detail="Order not found")  
 *# Подготавливаем данные* order\_data = prepare\_order\_data(order, db)  
 *# Генерируем DOCX из шаблона* template\_path = "templates/primer.docx"  
 output\_path = generate\_order\_docx(order\_data, template\_path)  
 *# Возвращаем файл для скачивания* filename = f"order\_{order.Code}.docx"  
 return FileResponse(  
 path=output\_path,  
 filename=filename,  
 media\_type='application/vnd.openxmlformats-officedocument.wordprocessingml.document'  
 )  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error generating document: {str(e)}")

Excel

1. Генерация файла формата xlcx (get)

@app.get("/orders/excel")  
def download\_orders\_excel(db: Session = Depends(get\_db)):  
 try:  
 *# Получаем все заказы с связанными данными* orders = db.query(models.Order).options(  
 joinedload(models.Order.items).joinedload(models.OrderItem.product),  
 joinedload(models.Order.buyer),  
 joinedload(models.Order.employee)  
 ).all()  
 if not orders:  
 raise HTTPException(status\_code=404, detail="No orders found")  
 *# Подготавливаем данные для Excel* orders\_data = prepare\_all\_orders\_data(orders, db)  
 *# Генерируем Excel файл* output\_path = generate\_orders\_excel(orders\_data)  
 *# Возвращаем файл для скачивания* filename = f"orders\_export\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}.xlsx"  
 return FileResponse(  
 path=output\_path,  
 filename=filename,  
 media\_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'  
 )  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error generating Excel file: {str(e)}")

PDF

1. Генерация файла о данных товаров формата pdf (get)

@app.get("/warehouse/catalog/pdf")  
def download\_products\_catalog\_pdf(db: Session = Depends(get\_db)):  
 try:  
 *# Получаем все товары с поставщиками* products = db.query(models.Product).options(  
 joinedload(models.Product.supplier)  
 ).all()  
 if not products:  
 raise HTTPException(status\_code=404, detail="No products found")  
 *# Подготавливаем данные* products\_data = prepare\_products\_data(products, db)  
 *# Генерируем PDF* output\_path = generate\_products\_catalog(products\_data)  
 *# Возвращаем файл для скачивания* filename = f"products\_catalog\_{datetime.now().strftime('%Y%m%d')}.pdf"  
 return FileResponse(  
 path=output\_path,  
 filename=filename,  
 media\_type='application/pdf'  
 )  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error generating PDF catalog: {str(e)}")

2. Генерация файла о данных заказов в формате pdf (get)

@app.get("/orders/pdf")  
def download\_orders\_pdf(db: Session = Depends(get\_db)):  
 try:  
 *# Получаем все заказы* orders = db.query(models.Order).options(  
 joinedload(models.Order.items),  
 joinedload(models.Order.buyer)  
 ).all()  
 if not orders:  
 raise HTTPException(status\_code=404, detail="No orders found")  
  
 *# Подготавливаем данные заказов* orders\_data = []  
 for order in orders:  
 total\_cost = sum(item.product.price \* item.quantity for item in order.items if item.product)  
 orders\_data.append({  
 'code': order.Code,  
 'buyer\_name': order.buyer.Buyer\_name if order.buyer else 'Ne ukazan',  
 'items': order.items,  
 'total\_cost': total\_cost,  
 'status': order.status if order.status else 'V obrabotke'  
 })  
 *# Генерируем PDF* output\_path = generate\_orders\_pdf(orders\_data)  
 *# Возвращаем файл для скачивания* filename = f"orders\_report\_{datetime.now().strftime('%Y%m%d')}.pdf"  
 return FileResponse(  
 path=output\_path,  
 filename=filename,  
 media\_type='application/pdf'  
 )  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error generating PDF report: {str(e)}")

## 4.2. Создание клиента

## 4.3. Реализация модулей вывода на печать

- вывод в шаблон .docx

from docx import Document  
from docx.shared import Inches  
from docx.enum.table import WD\_TABLE\_ALIGNMENT  
from docx.enum.text import WD\_ALIGN\_PARAGRAPH  
from docx.oxml import parse\_xml  
from docx.oxml.ns import nsdecls  
from datetime import datetime  
import tempfile  
import os  
  
  
def generate\_order\_docx(order\_data, template\_path="templates/primer.docx"):  
  
 *# Генерирует DOCX файл заказа из шаблона с правильными границами таблицы* try:  
 if not os.path.exists(template\_path):  
 raise FileNotFoundError(f"Шаблон не найден: {template\_path}")  
  
 doc = Document(template\_path)  
  
 *# Заменяем плейсхолдеры в документе* replace\_placeholders(doc, order\_data)  
  
 *# Создаем временный файл* with tempfile.NamedTemporaryFile(delete=False, suffix='.docx') as tmp\_file:  
 output\_path = tmp\_file.name  
  
 *# Сохраняем документ* doc.save(output\_path)  
 return output\_path  
  
 except Exception as e:  
 raise Exception(f"Ошибка при генерации документа: {str(e)}")  
  
  
def replace\_placeholders(doc, order\_data):  
  
 *# Заменяем простые плейсхолдеры* for paragraph in doc.paragraphs:  
 replace\_in\_paragraph(paragraph, order\_data)  
  
 *# Обрабатываем таблицы* for table in doc.tables:  
 replace\_in\_table(table, order\_data)  
  
  
def replace\_in\_paragraph(paragraph, order\_data):  
  
 *# Заменяет плейсхолдеры в одном параграфе* for key, value in order\_data.items():  
 if key == 'items':  
 continue  
  
 placeholder = f"{{{{{key}}}}}"  
 if placeholder in paragraph.text:  
 paragraph.text = paragraph.text.replace(placeholder, str(value))  
  
  
def replace\_in\_table(table, order\_data):  
  
 *# Ищем строку с плейсхолдером товара* for row\_idx, row in enumerate(table.rows):  
 for cell in row.cells:  
 if "{{quantity}}" in cell.text or "{{product\_id}}" in cell.text:  
 *# Нашли строку с товаром - заменяем её на все товары* replace\_items\_in\_table(table, row\_idx, order\_data['items'])  
 return  
  
  
def replace\_items\_in\_table(table, template\_row\_idx, items):  
  
 *# Удаляем строку-шаблон* table.\_tbl.remove(table.rows[template\_row\_idx].\_tr)  
  
 *# Добавляем строки для каждого товара* for item in items:  
 new\_row = table.add\_row()  
  
 *# Устанавливаем границы для всей строки* for cell in new\_row.cells:  
 *# Устанавливаем границы для ячейки* cell.\_element.tcPr.append(create\_border\_element())  
  
 *# Заполняем ячейки данными товара* cells = new\_row.cells  
 if len(cells) >= 6:  
 cells[0].text = str(item['quantity'])  
 cells[1].text = str(item['product\_id'])  
 cells[2].text = item['product\_name']  
 cells[3].text = f"{item['wholesale\_price']:.2f} руб."  
 cells[4].text = f"{item['total\_cost']:.2f} руб."  
 cells[5].text = f"{item['retail\_price']:.2f} руб."  
  
 *# Центрируем текст в ячейках* for cell in cells:  
 for paragraph in cell.paragraphs:  
 paragraph.alignment = WD\_ALIGN\_PARAGRAPH.CENTER  
  
  
def create\_border\_element():  
  
 *# Создает xml элемент с границами для ячейки таблицы* border\_xml = '''  
 <w:tcPr xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">  
 <w:tcBorders>  
 <w:top w:val="single" w:sz="4" w:space="0" w:color="000000"/>  
 <w:left w:val="single" w:sz="4" w:space="0" w:color="000000"/>  
 <w:bottom w:val="single" w:sz="4" w:space="0" w:color="000000"/>  
 <w:right w:val="single" w:sz="4" w:space="0" w:color="000000"/>  
 </w:tcBorders>  
 </w:tcPr>  
 '''  
 return parse\_xml(border\_xml)  
  
  
def prepare\_order\_data(order, db\_session):  
  
 *# Вычисляем общую стоимость всех товаров* total\_cost = 0  
 items\_data = []  
  
 for item in order.items:  
 item\_total = item.product.price \* item.quantity  
 total\_cost += item\_total  
  
 items\_data.append({  
 'quantity': item.quantity,  
 'product\_id': item.product.id\_Product,  
 'product\_name': item.product.Name\_tov,  
 'wholesale\_price': item.product.price,  
 'total\_cost': item\_total,  
 'retail\_price': item.product.price \* 1.2  
 })  
  
 order\_data = {  
 'order\_code': order.Code,  
 'buyer\_name': order.buyer.Buyer\_name if order.buyer else 'Не указан',  
 'buyer\_phone': order.buyer.number\_buy if order.buyer else 'Не указан',  
 'buyer\_address': order.buyer.adress\_buy if order.buyer else 'Не указан',  
 'employee\_name': order.employee.FIO if order.employee else 'Не указан',  
 'order\_date': datetime.now().strftime('%d.%m.%Y'),  
 'order\_status': order.status if order.status else 'В обработке',  
 'order\_total': f"{total\_cost:.2f}",  
 'description': order.description if order.description else 'Без дополнительных комментариев',  
 'items': items\_data  
 }  
  
 return order\_data



Рис.16 «Пример отчета об заказе в Word»

- создание документа .xlsx

from openpyxl import Workbook  
from openpyxl.styles import Font, Alignment, PatternFill, Border, Side  
from openpyxl.utils import get\_column\_letter  
from datetime import datetime  
import tempfile  
import os  
from collections import Counter  
  
  
def generate\_orders\_excel(orders\_data, output\_path=None):  
  
 *# Генерирует Excel файл со всеми заказами и аналитикой* try:  
 wb = Workbook()  
 ws = wb.active  
 ws.title = "Заказы"  
  
 *# Стили* header\_font = Font(bold=True, size=12, color="FFFFFF")  
 header\_fill = PatternFill(start\_color="366092", end\_color="366092", fill\_type="solid")  
 company\_font = Font(bold=True, size=16, color="2F5496")  
 summary\_font = Font(bold=True, size=14, color="2F5496")  
 border\_style = Border(  
 left=Side(style='thin'),  
 right=Side(style='thin'),  
 top=Side(style='thin'),  
 bottom=Side(style='thin')  
 )  
 center\_align = Alignment(horizontal='center', vertical='center')  
 left\_align = Alignment(horizontal='left', vertical='center')  
  
 *# Заголовок компании* ws.merge\_cells('A1:H1')  
 ws['A1'] = 'ООО "Warehouse"'  
 ws['A1'].font = company\_font  
 ws['A1'].alignment = center\_align  
  
 *# Заголовки столбцов* headers = [  
 'Заказываемое количество',  
 'Номер товара',  
 'Описание',  
 'Общая стоимость',  
 'ФИО покупателя',  
 'ФИО сотрудника',  
 'Дата заказа',  
 'Статус'  
 ]  
  
 *# Записываем заголовки* for col, header in enumerate(headers, 1):  
 cell = ws.cell(row=3, column=col, value=header)  
 cell.font = header\_font  
 cell.fill = header\_fill  
 cell.alignment = center\_align  
 cell.border = border\_style  
  
 *# Заполняем данными заказов* row\_num = 4  
 total\_revenue = 0  
 total\_items = 0  
 status\_counter = Counter()  
 buyer\_counter = Counter()  
  
 for order in orders\_data:  
 for item in order['items']:  
 total\_revenue += item['total\_cost']  
 total\_items += item['quantity']  
 status\_counter[order['order\_status']] += 1  
 buyer\_counter[order['buyer\_name']] += 1  
  
 *# Заказываемое количество* ws.cell(row=row\_num, column=1, value=item['quantity']).alignment = center\_align  
 ws.cell(row=row\_num, column=1).border = border\_style  
  
 *# Номер товара* ws.cell(row=row\_num, column=2, value=item['product\_id']).alignment = center\_align  
 ws.cell(row=row\_num, column=2).border = border\_style  
  
 *# Описание* ws.cell(row=row\_num, column=3, value=item['product\_name']).alignment = left\_align  
 ws.cell(row=row\_num, column=3).border = border\_style  
  
 *# Общая стоимость* ws.cell(row=row\_num, column=4, value=f"{item['total\_cost']:.2f} руб.").alignment = center\_align  
 ws.cell(row=row\_num, column=4).border = border\_style  
  
 *# ФИО покупателя* ws.cell(row=row\_num, column=5, value=order['buyer\_name']).alignment = left\_align  
 ws.cell(row=row\_num, column=5).border = border\_style  
  
 *# ФИО сотрудника* ws.cell(row=row\_num, column=6, value=order['employee\_name']).alignment = left\_align  
 ws.cell(row=row\_num, column=6).border = border\_style  
  
 *# Дата заказа* ws.cell(row=row\_num, column=7, value=order['order\_date']).alignment = center\_align  
 ws.cell(row=row\_num, column=7).border = border\_style  
  
 *# Статус* ws.cell(row=row\_num, column=8, value=order['order\_status']).alignment = center\_align  
 ws.cell(row=row\_num, column=8).border = border\_style  
  
 row\_num += 1  
  
 *# Итоговая строка для таблицы заказов* ws.cell(row=row\_num, column=3, value="ИТОГО:").font = Font(bold=True)  
 ws.cell(row=row\_num, column=3).alignment = right\_align  
 ws.cell(row=row\_num, column=4, value=f"{total\_revenue:.2f} руб.").font = Font(bold=True)  
 ws.cell(row=row\_num, column=4).alignment = center\_align  
 ws.cell(row=row\_num, column=4).fill = PatternFill(start\_color="FFFF00", fill\_type="solid")  
  
 *# Добавляем аналитику ниже таблицы* analytics\_start\_row = row\_num + 3  
  
 *# Заголовок аналитики* ws.merge\_cells(f'A{analytics\_start\_row}:H{analytics\_start\_row}')  
 ws[f'A{analytics\_start\_row}'] = '📊 СВОДНАЯ АНАЛИТИКА'  
 ws[f'A{analytics\_start\_row}'].font = summary\_font  
 ws[f'A{analytics\_start\_row}'].alignment = center\_align  
 ws[f'A{analytics\_start\_row}'].fill = PatternFill(start\_color="E6E6FA", fill\_type="solid")  
  
 *# Основные показатели* analytics\_data = [  
 ["Показатель", "Значение", "", "Статистика по статусам", "Количество"],  
 ["Всего заказов", len(orders\_data), "", "В обработке", status\_counter.get("В обработке", 0)],  
 ["Всего товаров", total\_items, "", "Подтвержден", status\_counter.get("Подтвержден", 0)],  
 ["Общая выручка", f"{total\_revenue:.2f} руб.", "", "В сборке", status\_counter.get("В сборке", 0)],  
 ["Средний чек", f"{total\_revenue / max(len(orders\_data), 1):.2f} руб.", "", "Готов к отгрузке",  
 status\_counter.get("Готов к отгрузке", 0)],  
 ["Топ покупатель", get\_top\_buyer(buyer\_counter), "", "Отгружен", status\_counter.get("Отгружен", 0)],  
 ["", "", "", "Доставлен", status\_counter.get("Доставлен", 0)],  
 ["", "", "", "Отменен", status\_counter.get("Отменен", 0)]  
 ]  
  
 *# Записываем аналитику* for row\_offset, row\_data in enumerate(analytics\_data):  
 current\_row = analytics\_start\_row + 1 + row\_offset  
 for col, value in enumerate(row\_data, 1):  
 cell = ws.cell(row=current\_row, column=col, value=value)  
 if row\_offset == 0: *# Заголовки* cell.font = Font(bold=True)  
 cell.fill = PatternFill(start\_color="F0F0F0", fill\_type="solid")  
 elif value and col in [2, 5]: *# Значения* cell.font = Font(bold=True)  
  
 *# Настраиваем ширину столбцов* column\_widths = {  
 'A': 18, *# Заказываемое количество* 'B': 12, *# Номер товара* 'C': 30, *# Описание* 'D': 16, *# Общая стоимость* 'E': 25, *# ФИО покупателя* 'F': 25, *# ФИО сотрудника* 'G': 12, *# Дата заказа* 'H': 15 *# Статус* }  
  
 for col, width in column\_widths.items():  
 ws.column\_dimensions[col].width = width  
  
 *# Автофильтр для заголовков* ws.auto\_filter.ref = f"A3:H{row\_num - 1}"  
  
 *# Замораживаем заголовки* ws.freeze\_panes = "A4"  
  
 *# Создаем временный файл если путь не указан* if not output\_path:  
 with tempfile.NamedTemporaryFile(delete=False, suffix='.xlsx') as tmp\_file:  
 output\_path = tmp\_file.name  
  
 *# Сохраняем файл* wb.save(output\_path)  
 return output\_path  
  
 except Exception as e:  
 raise Exception(f"Ошибка при генерации Excel файла: {str(e)}")  
  
  
def get\_top\_buyer(buyer\_counter):  
 *# Возвращает топ покупателя* if not buyer\_counter:  
 return "Нет данных"  
 top\_buyer = buyer\_counter.most\_common(1)[0]  
 return f"{top\_buyer[0]} ({top\_buyer[1]} зак.)"  
  
  
def prepare\_all\_orders\_data(orders, db\_session):  
  
 *# Подготавливает данные всех заказов для экспорта* orders\_data = []  
  
 for order in orders:  
 *# Вычисляем общую стоимость заказа* items\_data = []  
  
 for item in order.items:  
 item\_total = item.product.price \* item.quantity  
  
 items\_data.append({  
 'quantity': item.quantity,  
 'product\_id': item.product.id\_Product,  
 'product\_name': item.product.Name\_tov,  
 'total\_cost': item\_total  
 })  
  
 order\_data = {  
 'order\_code': order.Code,  
 'buyer\_name': order.buyer.Buyer\_name if order.buyer else 'Не указан',  
 'employee\_name': order.employee.FIO if order.employee else 'Не указан',  
 'order\_date': datetime.now().strftime('%d.%m.%Y'),  
 'order\_status': order.status if order.status else 'В обработке',  
 'items': items\_data  
 }  
  
 orders\_data.append(order\_data)  
  
 return orders\_data  
  
  
*# Добавляем выравнивание по правому краю*right\_align = Alignment(horizontal='right', vertical='center')

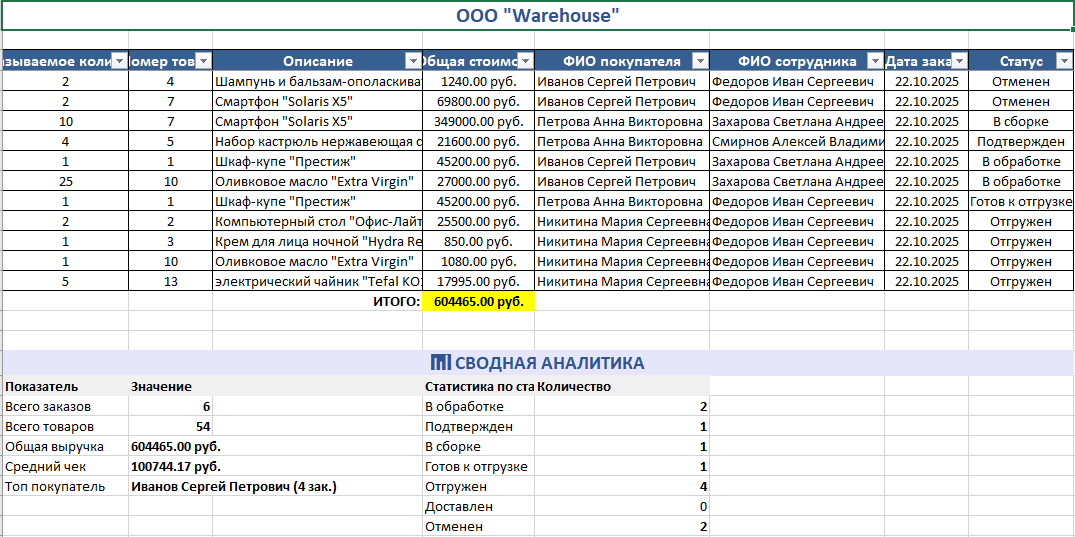


Рис.17 «Отчет о всех заказах в Excel»

- создание документа .pdf

Листинг

from reportlab.pdfgen import canvas  
from reportlab.lib.pagesizes import A4  
from reportlab.lib.units import cm  
from reportlab.pdfbase import pdfmetrics  
from reportlab.pdfbase.ttfonts import TTFont  
from datetime import datetime  
import tempfile  
import os  
  
  
def register\_russian\_font():  
 *# Регистрирует шрифт с поддержкой русского языка* try:  
 font\_paths = [  
 "C:/Windows/Fonts/arial.ttf",  
 ]  
  
 registered\_fonts = {  
 'regular': 'Helvetica',  
 'bold': 'Helvetica-Bold'  
 }  
  
 for font\_path in font\_paths:  
 if os.path.exists(font\_path):  
 try:  
 font\_name = os.path.basename(font\_path).replace('.ttf', '').replace('-Regular', '')  
 pdfmetrics.registerFont(TTFont(font\_name, font\_path))  
 pdfmetrics.registerFont(TTFont(font\_name + '-Bold', font\_path))  
 registered\_fonts['regular'] = font\_name  
 registered\_fonts['bold'] = font\_name + '-Bold'  
 print(f"Зарегистрирован шрифт: {font\_name}")  
 break  
 except Exception as e:  
 print(f"Ошибка регистрации шрифта {font\_path}: {e}")  
 continue  
  
 return registered\_fonts  
  
 except Exception as e:  
 print(f"Не удалось зарегистрировать русский шрифт: {e}")  
 return {'regular': 'Helvetica', 'bold': 'Helvetica-Bold'}  
  
  
def generate\_products\_catalog(products\_data, output\_path=None):  
  
 *# Генерирует PDF каталог товаров* try:  
 if not output\_path:  
 with tempfile.NamedTemporaryFile(delete=False, suffix='.pdf') as tmp\_file:  
 output\_path = tmp\_file.name  
  
 *# Регистрируем шрифты* fonts = register\_russian\_font()  
  
 *# Создаем PDF* c = canvas.Canvas(output\_path, pagesize=A4)  
 width, height = A4  
  
 *# Устанавливаем координаты для начала текста* y\_position = height - 2 \* cm  
  
 *# Заголовок* c.setFont(fonts['bold'], 16)  
 c.drawCentredString(width / 2, y\_position, "КАТАЛОГ ТОВАРОВ")  
 y\_position -= 1 \* cm  
  
 c.setFont(fonts['bold'], 14)  
 c.drawCentredString(width / 2, y\_position, 'ООО "Warehouse"')  
 y\_position -= 0.7 \* cm  
  
 c.setFont(fonts['regular'], 10)  
 date\_text = f"на {datetime.now().strftime('%d.%m.%Y г.')}"  
 c.drawCentredString(width / 2, y\_position, date\_text)  
 y\_position -= 1.5 \* cm  
  
 *# Группируем товары по поставщикам* suppliers = {}  
 for product in products\_data:  
 supplier\_name = product['supplier\_name']  
 if supplier\_name not in suppliers:  
 suppliers[supplier\_name] = []  
 suppliers[supplier\_name].append(product)  
  
 *# Создаем таблицу для каждого поставщика* for supplier\_name, supplier\_products in suppliers.items():  
 *# Проверяем, нужно ли создать новую страницу* if y\_position < 5 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
  
 *# Заголовок поставщика (на русском)* c.setFont(fonts['bold'], 12)  
 c.setFillColorRGB(0.2, 0.4, 0.6) *# Синий цвет* supplier\_title = f"Поставщик: {supplier\_name}"  
 c.drawString(2 \* cm, y\_position, supplier\_title)  
 y\_position -= 0.8 \* cm  
  
 *# Заголовки таблицы* c.setFont(fonts['bold'], 10)  
 c.setFillColorRGB(0, 0, 0) *# Черный цвет  
  
 # Рисуем заголовки столбцов* c.drawString(2 \* cm, y\_position, "Арт.")  
 c.drawString(4 \* cm, y\_position, "Наименование")  
 c.drawString(12 \* cm, y\_position, "Цена")  
 c.drawString(16 \* cm, y\_position, "Наличие")  
  
 *# Линия под заголовками* y\_position -= 0.3 \* cm  
 c.line(2 \* cm, y\_position, 18 \* cm, y\_position)  
 y\_position -= 0.5 \* cm  
  
 *# Товары поставщика* c.setFont(fonts['regular'], 9)  
 for product in supplier\_products:  
 *# Проверяем, нужно ли создать новую страницу* if y\_position < 3 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
  
 *# Повторяем заголовки на новой странице* c.setFont(fonts['bold'], 10)  
 c.drawString(2 \* cm, y\_position, "Арт.")  
 c.drawString(4 \* cm, y\_position, "Наименование")  
 c.drawString(12 \* cm, y\_position, "Цена")  
 c.drawString(16 \* cm, y\_position, "Наличие")  
 y\_position -= 0.8 \* cm  
 c.line(2 \* cm, y\_position, 18 \* cm, y\_position)  
 y\_position -= 0.5 \* cm  
 c.setFont(fonts['regular'], 9)  
  
 *# Данные товара* c.drawString(2 \* cm, y\_position, str(product['id']))  
  
 *# Название товара (обрезанное если слишком длинное)* name = product['name']  
 if len(name) > 35:  
 name = name[:32] + "..."  
 c.drawString(4 \* cm, y\_position, name)  
  
 c.drawString(12 \* cm, y\_position, f"{product['price']:,.2f} руб.".replace(',', ' '))  
 c.drawString(16 \* cm, y\_position, f"{product['quantity']} шт.")  
  
 y\_position -= 0.5 \* cm  
  
 *# Отступ между поставщиками* y\_position -= 0.5 \* cm  
  
 *# Добавляем статистику в конце* if y\_position < 8 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
  
 *# Статистика (на русском)* total\_products = len(products\_data)  
 total\_quantity = sum(product['quantity'] for product in products\_data)  
 total\_suppliers = len(suppliers)  
 avg\_price = sum(product['price'] for product in products\_data) / total\_products if total\_products > 0 else 0  
  
 c.setFont(fonts['bold'], 12)  
 c.drawString(2 \* cm, y\_position, "СТАТИСТИКА КАТАЛОГА:")  
 y\_position -= 0.7 \* cm  
  
 c.setFont(fonts['regular'], 10)  
 stats = [  
 f"• Всего товаров: {total\_products} позиций",  
 f"• Поставщиков: {total\_suppliers} компаний",  
 f"• Товаров на складе: {total\_quantity} ед.",  
 f"• Средняя цена: {avg\_price:,.2f} руб.",  
 "",  
 "Контакты:",  
 "Телефон: +7 (495) 123-45-67",  
 "Email: order@warehouse.ru"  
 ]  
  
 for stat in stats:  
 if y\_position < 3 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
 c.setFont(fonts['regular'], 10)  
  
 c.drawString(2 \* cm, y\_position, stat)  
 y\_position -= 0.5 \* cm  
  
 *# Сохраняем PDF* c.save()  
 return output\_path  
  
 except Exception as e:  
 raise Exception(f"Ошибка при генерации PDF: {str(e)}")  
  
  
def generate\_orders\_pdf(orders\_data, output\_path=None):  
 *# Генерирует PDF отчет по заказам с русскими символами* try:  
 if not output\_path:  
 with tempfile.NamedTemporaryFile(delete=False, suffix='.pdf') as tmp\_file:  
 output\_path = tmp\_file.name  
  
 *# Регистрируем шрифты* fonts = register\_russian\_font()  
  
 *# Создаем PDF* c = canvas.Canvas(output\_path, pagesize=A4)  
 width, height = A4  
  
 *# Устанавливаем координаты для начала текста* y\_position = height - 2 \* cm  
  
 *# Заголовок* c.setFont(fonts['bold'], 16)  
 c.drawCentredString(width / 2, y\_position, "ОТЧЕТ ПО ЗАКАЗАМ")  
 y\_position -= 1 \* cm  
  
 c.setFont(fonts['bold'], 14)  
 c.drawCentredString(width / 2, y\_position, 'ООО "Warehouse"')  
 y\_position -= 0.7 \* cm  
  
 c.setFont(fonts['regular'], 10)  
 date\_text = f"за период: {datetime.now().strftime('%d.%m.%Y г.')}"  
 c.drawCentredString(width / 2, y\_position, date\_text)  
 y\_position -= 1.5 \* cm  
  
 *# Заголовки таблицы заказов* c.setFont(fonts['bold'], 10)  
 headers = ["Код заказа", "Покупатель", "Товары", "Общая стоимость", "Статус"]  
  
 c.drawString(1 \* cm, y\_position, headers[0])  
 c.drawString(4 \* cm, y\_position, headers[1])  
 c.drawString(8 \* cm, y\_position, headers[2])  
 c.drawString(13 \* cm, y\_position, headers[3])  
 c.drawString(17 \* cm, y\_position, headers[4])  
  
 y\_position -= 0.3 \* cm  
 c.line(1 \* cm, y\_position, 19 \* cm, y\_position)  
 y\_position -= 0.5 \* cm  
  
 *# Данные заказов* c.setFont(fonts['regular'], 8)  
 for order in orders\_data:  
 if y\_position < 3 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
 c.setFont(fonts['bold'], 10)  
 c.drawString(1 \* cm, y\_position, headers[0])  
 c.drawString(4 \* cm, y\_position, headers[1])  
 c.drawString(8 \* cm, y\_position, headers[2])  
 c.drawString(13 \* cm, y\_position, headers[3])  
 c.drawString(17 \* cm, y\_position, headers[4])  
 y\_position -= 0.8 \* cm  
 c.line(1 \* cm, y\_position, 19 \* cm, y\_position)  
 y\_position -= 0.5 \* cm  
 c.setFont(fonts['regular'], 8)  
  
 *# Код заказа* c.drawString(1 \* cm, y\_position, order['code'])  
  
 *# Покупатель* buyer = order['buyer\_name'][:20]  
 c.drawString(4 \* cm, y\_position, buyer)  
  
 *# Товары* products\_text = f"{len(order['items'])} товаров"  
 c.drawString(8 \* cm, y\_position, products\_text)  
  
 *# Общая стоимость* total = f"{order['total\_cost']:,.2f} руб.".replace(',', ' ')  
 c.drawString(13 \* cm, y\_position, total)  
  
 *# Статус* status = order['status']  
 c.drawString(17 \* cm, y\_position, status)  
  
 y\_position -= 0.5 \* cm  
  
 *# Статистика заказов* if y\_position < 6 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
  
 total\_orders = len(orders\_data)  
 total\_revenue = sum(order['total\_cost'] for order in orders\_data)  
  
 c.setFont(fonts['bold'], 12)  
 c.drawString(2 \* cm, y\_position, "ИТОГИ:")  
 y\_position -= 0.7 \* cm  
  
 c.setFont(fonts['regular'], 10)  
 stats = [  
 f"Всего заказов: {total\_orders}",  
 f"Общая выручка: {total\_revenue:,.2f} руб.",  
 f"Средний чек: {total\_revenue / total\_orders:,.2f} руб." if total\_orders > 0 else "Средний чек: 0 руб."  
 ]  
  
 for stat in stats:  
 if y\_position < 3 \* cm:  
 c.showPage()  
 y\_position = height - 2 \* cm  
 c.setFont(fonts['regular'], 10)  
  
 c.drawString(2 \* cm, y\_position, stat)  
 y\_position -= 0.5 \* cm  
  
 c.save()  
 return output\_path  
  
 except Exception as e:  
 raise Exception(f"Ошибка при генерации PDF отчета: {str(e)}")  
  
  
def prepare\_products\_data(products, db\_session):  
 *# Подготавливает данные товаров для генерации PDF* products\_data = []  
  
 for product in products:  
 product\_data = {  
 'id': product.id\_Product,  
 'name': product.Name\_tov,  
 'price': product.price,  
 'quantity': product.quantity,  
 'supplier\_name': product.supplier.Name\_sup if product.supplier else 'Без поставщика'  
 }  
 products\_data.append(product\_data)  
  
 *# Сортируем по поставщикам и названию товара* products\_data.sort(key=lambda x: (x['supplier\_name'], x['name']))  
  
 return products\_data



Рис.18 «1 страница каталога товаров в pdf формате»



Рис.19 «2 страница каталога товаров с статистикой»

4.4. Реализация импорта данных из файла в базу данных

4.5. Загрузка приложения в репозиторий GitHub

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