**Содержание**

Введение

1. Разработка технического задания

2. Проектирование интерфейса приложения

Макет представляет собой систему, созданную для преподавателей и администраторов, которая упрощает организацию учебного процесса. Приложение позволяет управлять студентами, учебными группами и предметами, а также просматривать расписание занятий. Основные возможности включают добавление новых записей, редактирование существующих данных и удаление устаревшей информации о студентах, группах и предметах.

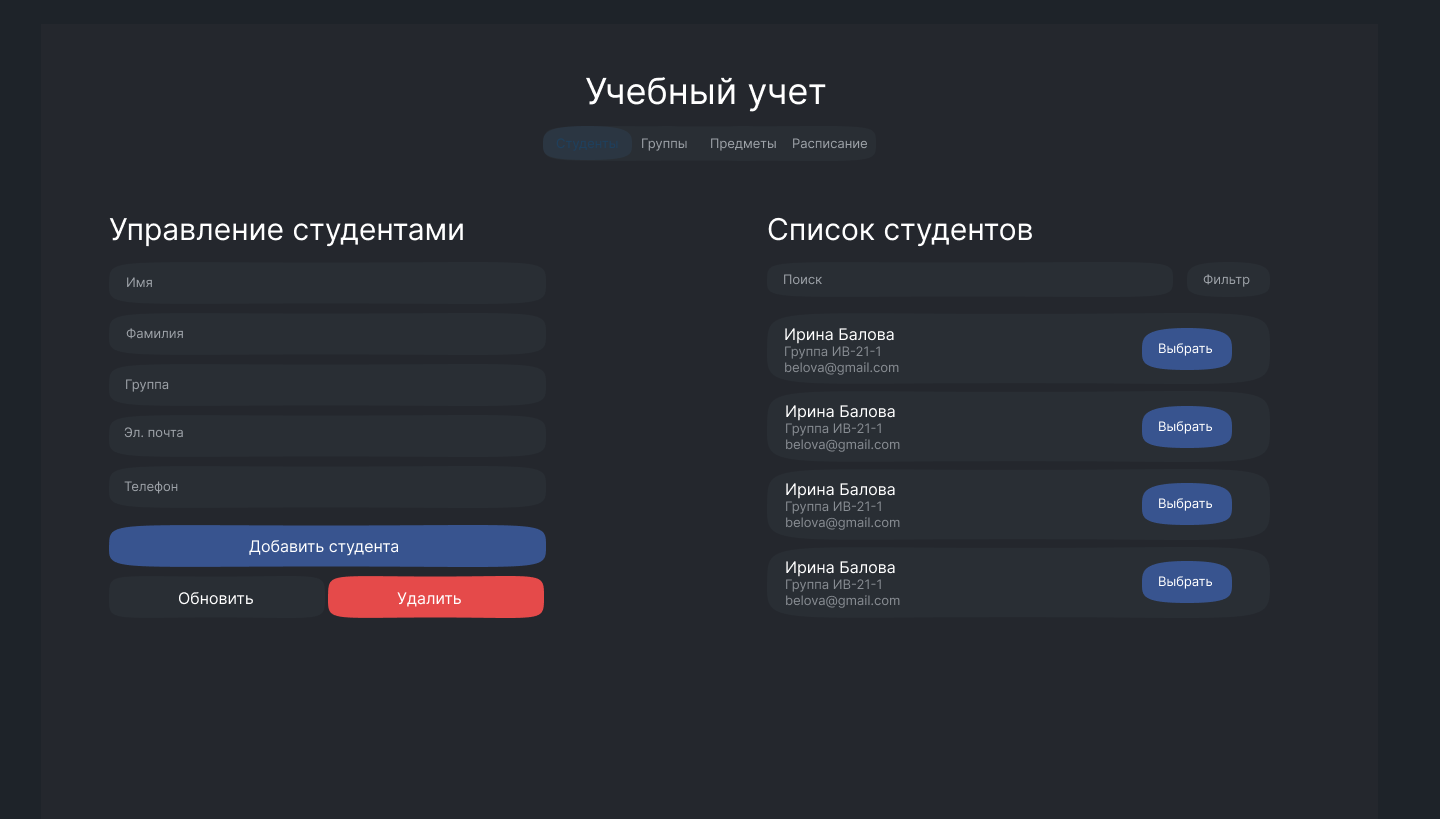


Рис. 1 «Студенты»

На данной странице преподаватель может управлять данными студентов: добавлять новых, редактировать или удалять существующие записи. Также реализована возможность поиска студентов по различным параметрам. В правой части интерфейса отображается список студентов с краткой информацией о каждом из них.

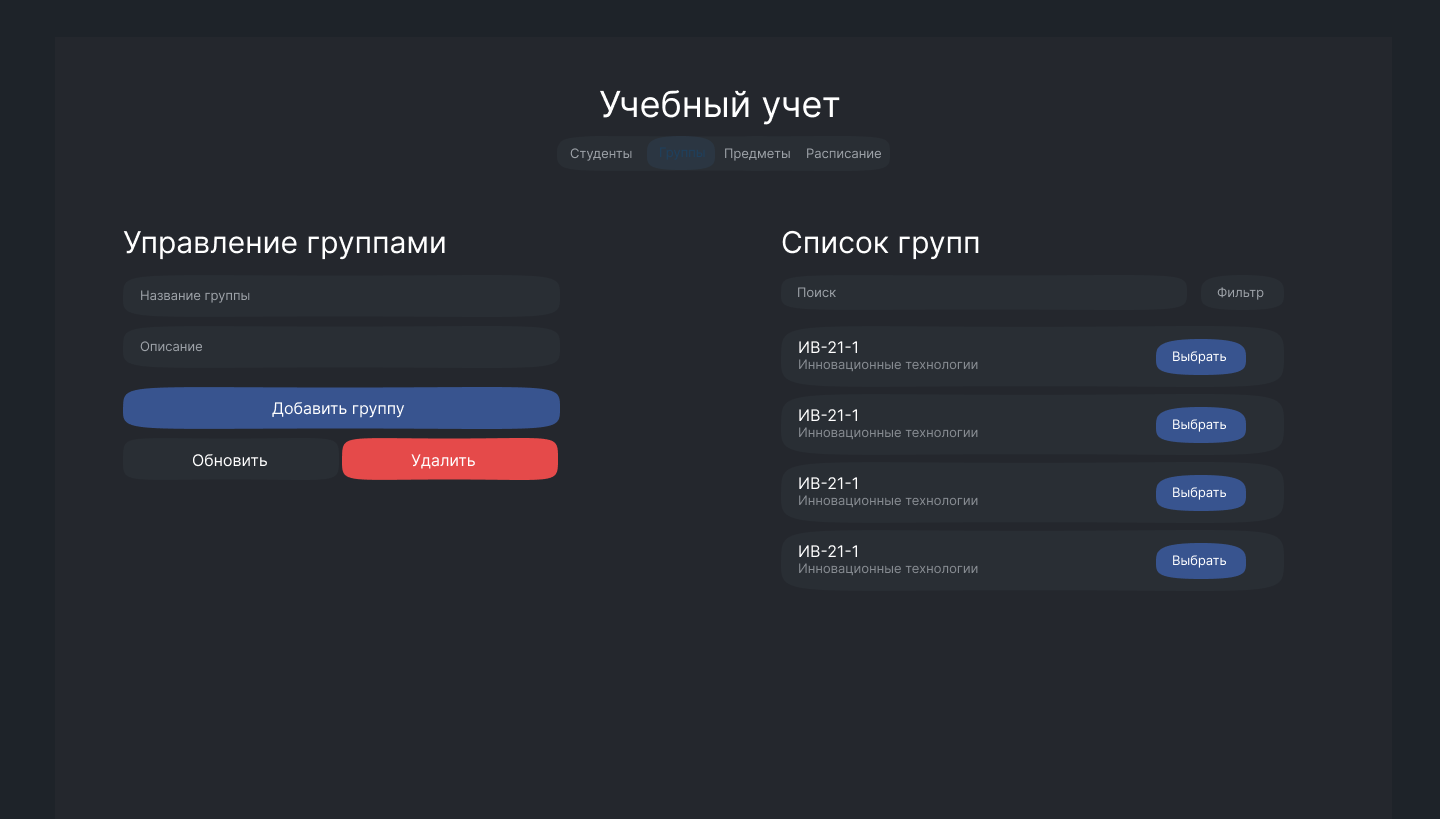


Рис. 2 «Группы»

На этой странице администратор может создавать, изменять и удалять учебные группы. Для каждой группы указываются её название и краткое описание. Справа представлен список всех групп с возможностью поиска, что позволяет быстро находить нужную запись.

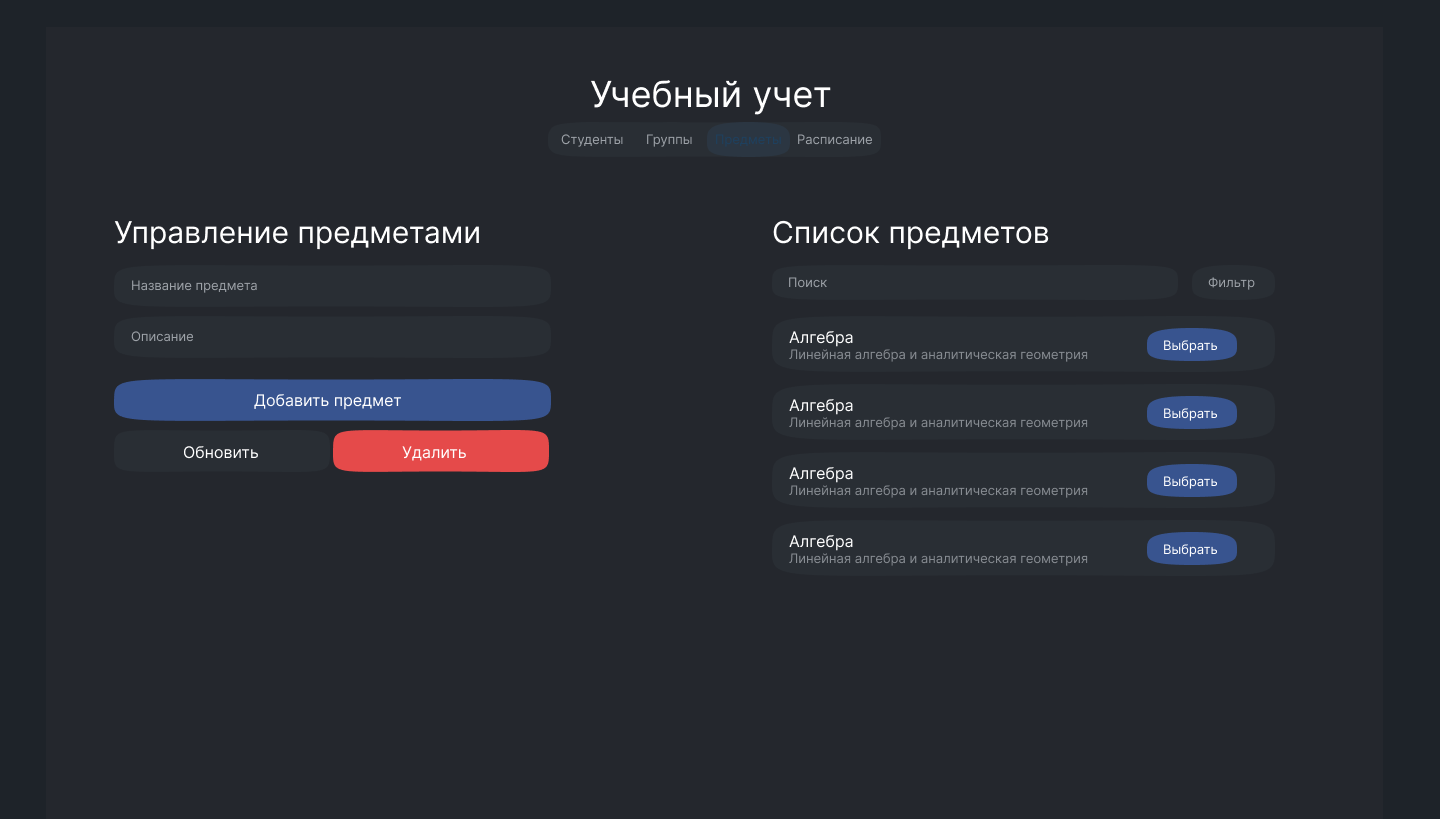


Рис. 3 «Предметы»

На этой странице преподаватель может создавать, изменять и удалять предметы. Для каждого предмета указывается её название и краткое описание. Справа представлен список всех предметов с возможностью поиска, что позволяет быстро находить нужную запись.

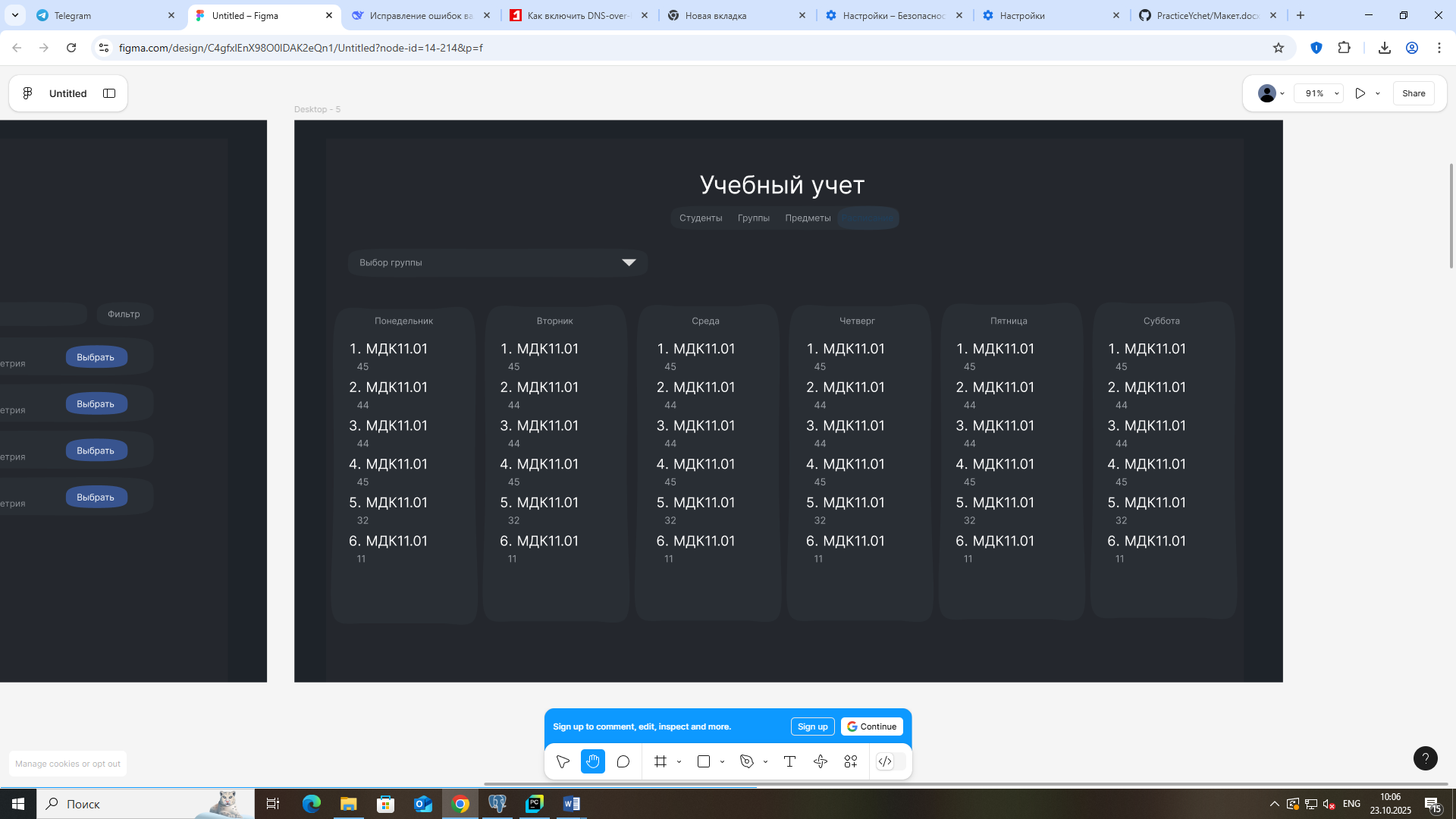


Рис. 4 «Расписание»

Данная страница предназначена для просмотра расписания занятий по выбранной группе. Преподаватель может выбрать нужную группу из выпадающего списка и ознакомиться с расписанием на каждый день недели. Интерфейс структурирован по дням, что обеспечивает удобное восприятие и навигацию.

3. Проектирование и создание базы данных

3.1. Разработка структуры БД

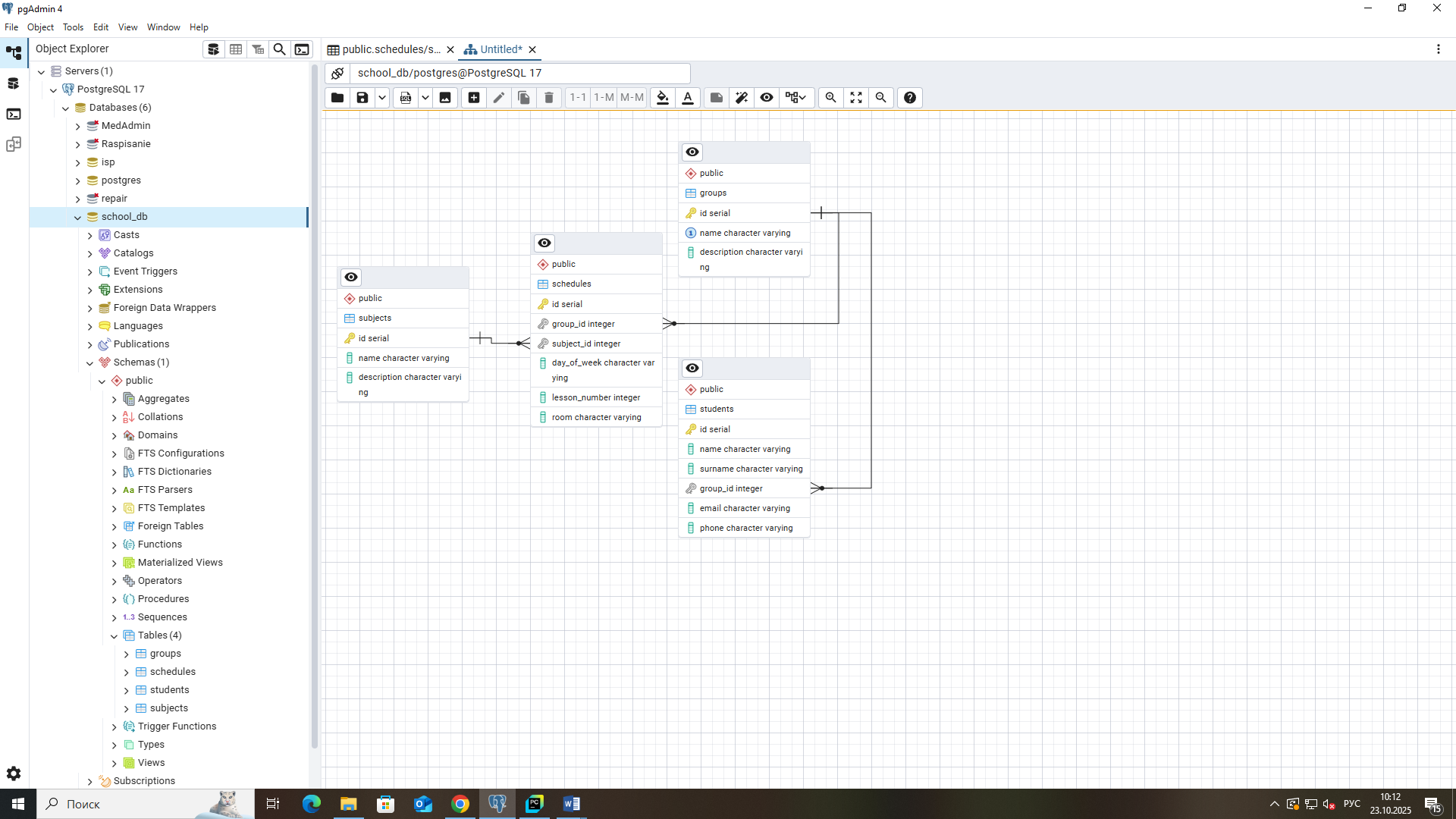


Рис. 5 «ER-диаграмма»

Таблица students предназначена для хранения информации о студентах. В её структуре предусмотрены следующие атрибуты:

* id – первичный ключ. Тип данных integer, not null, primary key, nextval nextval('students\_id\_seq'::regclass)
* name – имя студента. Тип данных character varying, not null
* surname – фамилия студента. Тип данных character varying, not null
* group\_id – внешний ключ таблицы groups. Тип данных integer
* email – почта студента. Тип данных character varying
* phone – телефон студента. Тип данных character varying

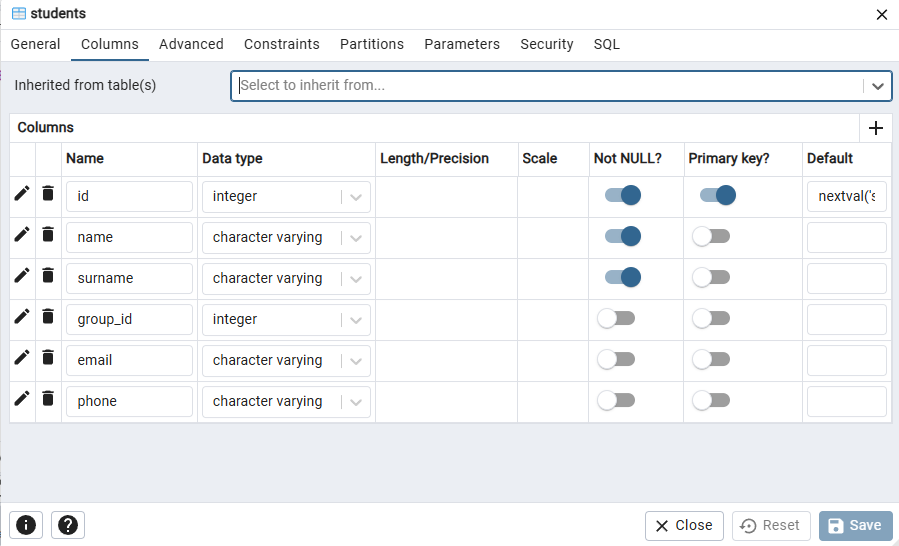


Рис. 6 «Таблица students»

Таблица subjects предназначена для хранения информации о предметах. Она включает следующие атрибуты:

* id – первичный ключ. Тип данных integer, not null, primary key, nextval('subjects\_id\_seq'::regclass)
* name – имя предмета. Тип данных character varying, not null
* description – описание предмета. Тип данных character varying

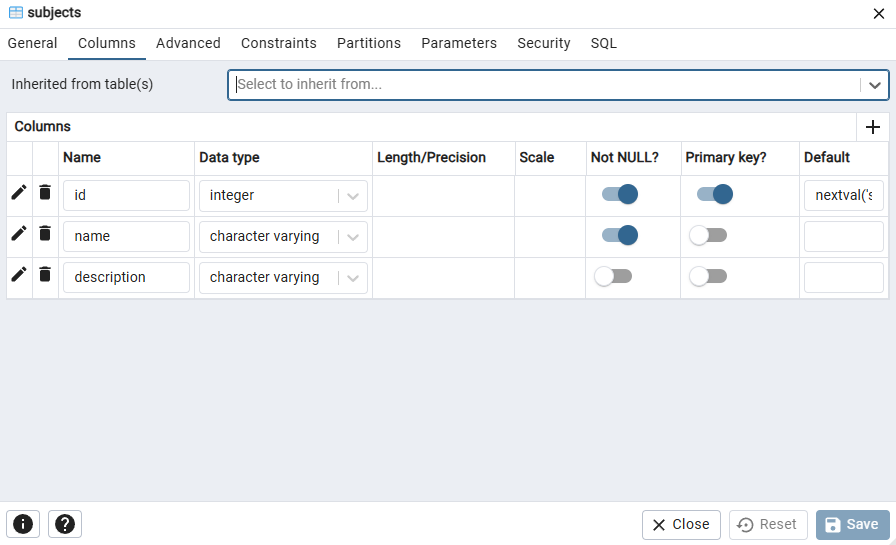


Рис. 7 «Таблица subjects»

Таблица groups служит для хранения сведений об учебных группах. В её структуру входят следующие атрибуты:

* id – первичный ключ. Тип данных integer, not null, primary key, nextval('groups\_id\_seq'::regclass).
* name – имя группы. Тип данных character varying, not null
* description – описание группы. Тип данных character varying

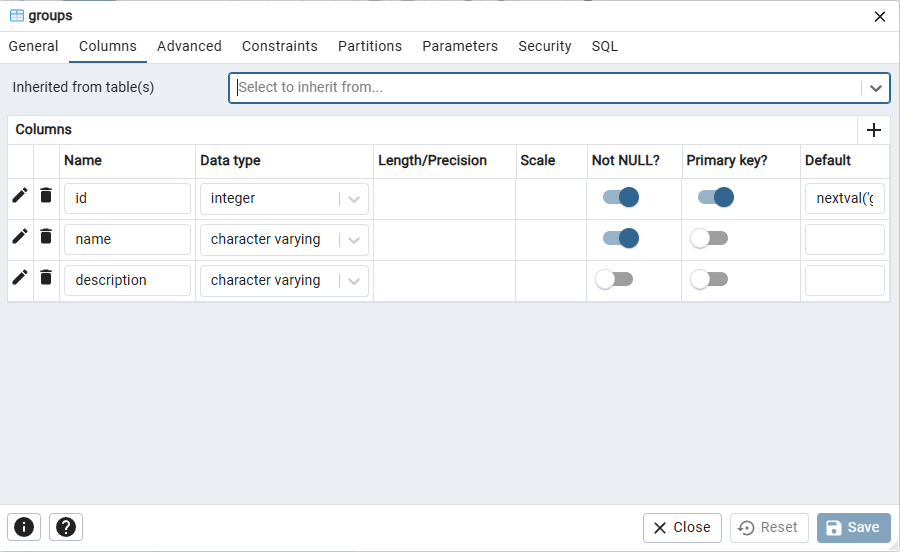


Рис. 8

Таблица schedules используется для формирования расписания занятий. Она содержит следующие атрибуты:

* id – первичный ключ. Тип данных integer, not null, primary key, nextval('schedules\_id\_seq'::regclass)
* group\_id – внешний ключ таблицы groups. Тип данных integer, not null
* subject\_id – внешний ключ таюлицы subjects. Тип данных integer, not null
* day\_of\_week – день недели в таблице расписания. Тип данных character varying, not null
* lesson\_number – номер пары в таблице расписания. Тип данных integer, not null
* room – номер аудитории в таблице расписания. Тип данных character varying

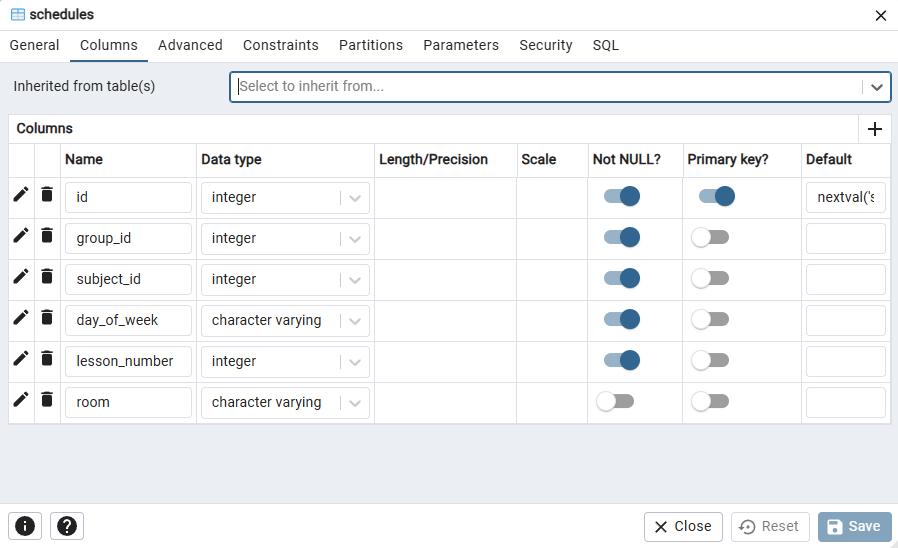


Рис. 9 «Таблица schedules»

3.2. Создание модели базы данных

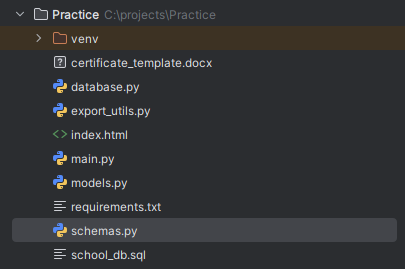


Рис. 10 «Иерархия файлов»

- модуль классов

Листинг 1 – класс Student

class Student(Base):  
 \_\_tablename\_\_ = "students"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 name = Column(String, nullable=False)  
 surname = Column(String, nullable=False)  
 group\_id = Column(Integer, ForeignKey("groups.id"), nullable=True)  
 email = Column(String, nullable=True)  
 phone = Column(String, nullable=True)  
  
 group = relationship("Group", back\_populates="students")

Листинг 2 – класс Group

class Group(Base):  
 \_\_tablename\_\_ = "groups"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 name = Column(String, unique=True, nullable=False)  
 description = Column(String, nullable=True)  
  
 students = relationship("Student", back\_populates="group")  
 schedules = relationship("Schedule", back\_populates="group")

Листинг 3 – класс Subject

class Subject(Base):  
 \_\_tablename\_\_ = "subjects"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 name = Column(String, nullable=False)  
 description = Column(String, nullable=True)  
  
 schedules = relationship("Schedule", back\_populates="subject")

Листинг 4 – класс Schedule

class Schedule(Base):  
 \_\_tablename\_\_ = "schedules"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 group\_id = Column(Integer, ForeignKey("groups.id"), nullable=False)  
 subject\_id = Column(Integer, ForeignKey("subjects.id"), nullable=False)  
 day\_of\_week = Column(String, nullable=False)   
 lesson\_number = Column(Integer, nullable=False)  
 room = Column(String, nullable=True)  
  
 group = relationship("Group", back\_populates="schedules")  
 subject = relationship("Subject", back\_populates="schedules")

4. Разработка программных модулей

4.1. Создание сервера

- модуль подключения к базе данных

Листинг 5 – подключение к базе данных

DATABASE\_URL = "postgresql://postgres:1234@localhost:5432/school\_db"  
  
*# Создание движка базы данных*engine = create\_engine(  
 DATABASE\_URL,  
 pool\_pre\_ping=True,  
 echo=False  
)  
  
*# Создание сессии*SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
  
*# Базовый класс для моделей*Base = declarative\_base()  
  
*# Зависимость для получения сессии БД*def get\_db():  
 db = SessionLocal()  
 try:  
 yield db  
 finally:  
 db.close()

- модуль функций доступа к данным API (GET, POST, PUT, DELETE)

1. API для студентов

Листинг 6 — Получение списка студентов

@app.get("/api/students", response\_model=List[schemas.Student])  
def get\_students(db: Session = Depends(get\_db)):  
 return db.query(models.Student).all()

Листинг 7 — Создание нового студента

@app.post("/api/students", response\_model=schemas.Student)  
def create\_student(student: schemas.StudentCreate, db: Session = Depends(get\_db)):  
 db\_student = models.Student(\*\*student.dict())  
 db.add(db\_student)  
 db.commit()  
 db.refresh(db\_student)  
 return db\_student

Листинг 8 — Обновление данных студента

@app.put("/api/students/{student\_id}", response\_model=schemas.Student)  
def update\_student(student\_id: int, student: schemas.StudentCreate, db: Session = Depends(get\_db)):  
 db\_student = db.query(models.Student).filter(models.Student.id == student\_id).first()  
 if not db\_student:  
 raise HTTPException(status\_code=404, detail="Студент не найден")  
 for key, value in student.dict().items():  
 setattr(db\_student, key, value)  
 db.commit()  
 db.refresh(db\_student)  
 return db\_student

Листинг 9 — Удаление студента

@app.delete("/api/students/{student\_id}")  
def delete\_student(student\_id: int, db: Session = Depends(get\_db)):  
 db\_student = db.query(models.Student).filter(models.Student.id == student\_id).first()  
 if not db\_student:  
 raise HTTPException(status\_code=404, detail="Студент не найден")  
 db.delete(db\_student)  
 db.commit()  
 return {"message": "Студент удален"}

2. API для групп

Листинг 10 — Получение списка групп

@app.get("/api/groups", response\_model=List[schemas.Group])  
def get\_groups(db: Session = Depends(get\_db)):  
 return db.query(models.Group).all()

Листинг 11 — Создание новой группы

@app.post("/api/groups", response\_model=schemas.Group)  
def create\_group(group: schemas.GroupCreate, db: Session = Depends(get\_db)):  
 db\_group = models.Group(\*\*group.dict())  
 db.add(db\_group)  
 db.commit()  
 db.refresh(db\_group)  
 return db\_group

Листинг 12 — Обновление группы

@app.put("/api/groups/{group\_id}", response\_model=schemas.Group)  
def update\_group(group\_id: int, group: schemas.GroupCreate, db: Session = Depends(get\_db)):  
 db\_group = db.query(models.Group).filter(models.Group.id == group\_id).first()  
 if not db\_group:  
 raise HTTPException(status\_code=404, detail="Группа не найдена")  
 for key, value in group.dict().items():  
 setattr(db\_group, key, value)  
 db.commit()  
 db.refresh(db\_group)  
 return db\_group

Листинг 13 — Удаление группы

@app.delete("/api/groups/{group\_id}")  
def delete\_group(group\_id: int, db: Session = Depends(get\_db)):  
 db\_group = db.query(models.Group).filter(models.Group.id == group\_id).first()  
 if not db\_group:  
 raise HTTPException(status\_code=404, detail="Группа не найдена")  
 db.delete(db\_group)  
 db.commit()  
 return {"message": "Группа удалена"}

3. API для предметов

Листинг 14 — Получение списка предметов

@app.get("/api/subjects", response\_model=List[schemas.Subject])  
def get\_subjects(db: Session = Depends(get\_db)):  
 return db.query(models.Subject).all()

Листинг 15 — Создание нового предмета

@app.post("/api/subjects", response\_model=schemas.Subject)  
def create\_subject(subject: schemas.SubjectCreate, db: Session = Depends(get\_db)):  
 db\_subject = models.Subject(\*\*subject.dict())  
 db.add(db\_subject)  
 db.commit()  
 db.refresh(db\_subject)  
 return db\_subject

Листинг 16 — Обновление предмета

@app.put("/api/subjects/{subject\_id}", response\_model=schemas.Subject)  
def update\_subject(subject\_id: int, subject: schemas.SubjectCreate, db: Session = Depends(get\_db)):  
 db\_subject = db.query(models.Subject).filter(models.Subject.id == subject\_id).first()  
 if not db\_subject:  
 raise HTTPException(status\_code=404, detail="Предмет не найден")  
 for key, value in subject.dict().items():  
 setattr(db\_subject, key, value)  
 db.commit()  
 db.refresh(db\_subject)  
 return db\_subject

Листинг 17 — Удаление предмета

@app.delete("/api/subjects/{subject\_id}")  
def delete\_subject(subject\_id: int, db: Session = Depends(get\_db)):  
 db\_subject = db.query(models.Subject).filter(models.Subject.id == subject\_id).first()  
 if not db\_subject:  
 raise HTTPException(status\_code=404, detail="Предмет не найден")  
 db.delete(db\_subject)  
 db.commit()  
 return {"message": "Предмет удален"}

4. API для расписания

Листинг 18 — Получение расписания

@app.get("/api/schedule", response\_model=List[schemas.Schedule])  
def get\_schedule(group\_id: int = None, db: Session = Depends(get\_db)):  
 query = db.query(models.Schedule)  
 if group\_id:  
 query = query.filter(models.Schedule.group\_id == group\_id)  
 return query.all()

Листинг 19 — Создание записи расписания

@app.post("/api/schedule", response\_model=schemas.Schedule)  
def create\_schedule(schedule: schemas.ScheduleCreate, db: Session = Depends(get\_db)):  
 db\_schedule = models.Schedule(\*\*schedule.dict())  
 db.add(db\_schedule)  
 db.commit()  
 db.refresh(db\_schedule)  
 return db\_schedule

Листинг 20 — Обновление записи расписания

@app.put("/api/schedule/{schedule\_id}", response\_model=schemas.Schedule)  
def update\_schedule(schedule\_id: int, schedule: schemas.ScheduleCreate, db: Session = Depends(get\_db)):  
 db\_schedule = db.query(models.Schedule).filter(models.Schedule.id == schedule\_id).first()  
 if not db\_schedule:  
 raise HTTPException(status\_code=404, detail="Расписание не найдено")  
 for key, value in schedule.dict().items():  
 setattr(db\_schedule, key, value)  
 db.commit()  
 db.refresh(db\_schedule)  
 return db\_schedule

Листинг 21 — Удаление записи расписания

@app.delete("/api/schedule/{schedule\_id}")  
def delete\_schedule(schedule\_id: int, db: Session = Depends(get\_db)):  
 db\_schedule = db.query(models.Schedule).filter(models.Schedule.id == schedule\_id).first()  
 if not db\_schedule:  
 raise HTTPException(status\_code=404, detail="Расписание не найдено")  
 db.delete(db\_schedule)  
 db.commit()  
 return {"message": "Расписание удалено"}

5. API для вывода на печать

Листинг 22 — Экспорт справки студента в Word

@app.get("/api/export/student/{student\_id}/certificate-word")  
def export\_student\_certificate\_word(student\_id: int, db: Session = Depends(get\_db)):  
 *"""Экспорт справки студента в Word"""* student = db.query(models.Student).filter(models.Student.id == student\_id).first()  
 if not student:  
 raise HTTPException(status\_code=404, detail="Студент не найден")  
  
 group\_name = None  
 if student.group\_id:  
 group = db.query(models.Group).filter(models.Group.id == student.group\_id).first()  
 group\_name = group.name if group else None  
  
 buffer = create\_student\_certificate(student, group\_name)  
  
 *# Безопасное имя файла без русских символов* filename = f"certificate\_{student.surname}\_{student.name}.docx"  
 safe\_filename = urllib.parse.quote(filename)  
  
 return StreamingResponse(  
 buffer,  
 media\_type="application/vnd.openxmlformats-officedocument.wordprocessingml.document",  
 headers={  
 "Content-Disposition": f"attachment; filename={safe\_filename}; filename\*=UTF-8''{safe\_filename}"  
 }  
 )

Листинг 23 — Экспорт расписания в Excel

@app.get("/api/export/schedule/{group\_id}/excel")  
def export\_schedule\_excel(group\_id: int, db: Session = Depends(get\_db)):  
 *"""Экспорт расписания группы в Excel"""* group = db.query(models.Group).filter(models.Group.id == group\_id).first()  
 if not group:  
 raise HTTPException(status\_code=404, detail="Группа не найдена")  
  
 schedules = db.query(models.Schedule).filter(models.Schedule.group\_id == group\_id).all()  
 subjects = db.query(models.Subject).all()  
 subjects\_dict = {s.id: s.name for s in subjects}  
  
 buffer = create\_schedule\_excel(group, schedules, subjects\_dict)  
  
 *# Безопасное имя файла без русских символов* filename = f"schedule\_{group.name}.xlsx"  
 safe\_filename = urllib.parse.quote(filename)  
  
 return StreamingResponse(  
 buffer,  
 media\_type="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",  
 headers={  
 "Content-Disposition": f"attachment; filename={safe\_filename}; filename\*=UTF-8''{safe\_filename}"  
 }  
 )

Листинг 24 — Экспорт справки студента в PDF

@app.get("/api/export/student/{student\_id}/certificate-pdf")  
def export\_student\_certificate\_pdf(student\_id: int, db: Session = Depends(get\_db)):  
 *"""Экспорт справки студента в PDF"""* student = db.query(models.Student).filter(models.Student.id == student\_id).first()  
 if not student:  
 raise HTTPException(status\_code=404, detail="Студент не найден")  
  
 group\_name = None  
 if student.group\_id:  
 group = db.query(models.Group).filter(models.Group.id == student.group\_id).first()  
 group\_name = group.name if group else None  
  
 buffer = create\_student\_certificate\_pdf(student, group\_name)  
  
 *# Безопасное имя файла без русских символов* filename = f"certificate\_{student.surname}\_{student.name}.pdf"  
 safe\_filename = urllib.parse.quote(filename)  
  
 return StreamingResponse(  
 buffer,  
 media\_type="application/pdf",  
 headers={  
 "Content-Disposition": f"attachment; filename={safe\_filename}; filename\*=UTF-8''{safe\_filename}"  
 }  
 )

4.2. Создание клиента

4.3. Реализация модулей вывода на печать

- вывод в шаблон .docx

Для создания документов Word используется библиотека **python-docx.** Само создание word-файла делается шаблонно в модуле export\_utils.py, функция create\_student\_certificate()

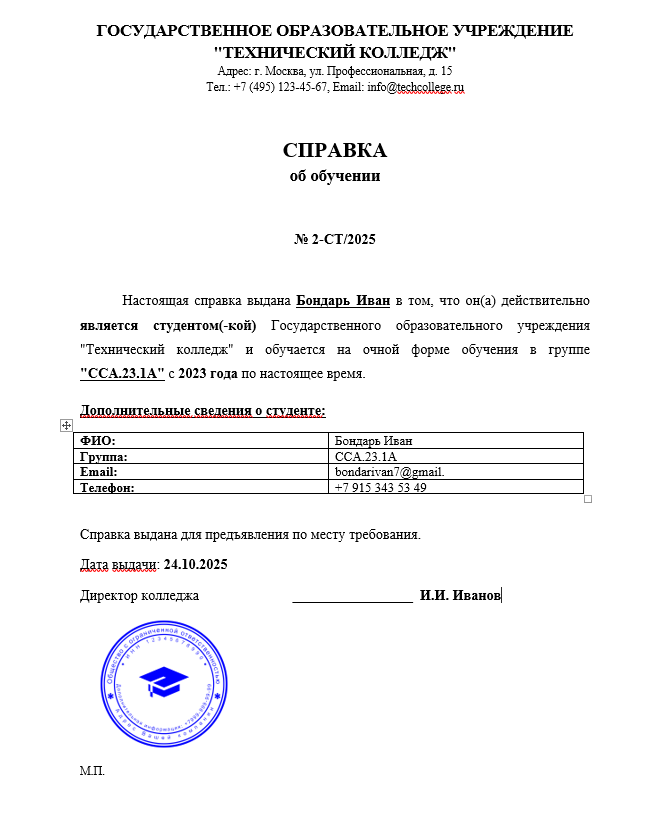


Рис. 11 «Шаблон docx»

Листинг 25 – создание docx-файла

def create\_student\_certificate(student, group\_name):  
 *"""Создание справки об обучении студента из Word-шаблона"""  
  
 # Путь к шаблону* template\_path = 'certificate\_template.docx'  
  
 *# ЗАГРУЖАЕМ ШАБЛОН (НЕ СОЗДАЕМ НОВЫЙ ДОКУМЕНТ!)* doc = Document(template\_path)  
  
 *# Подготовка данных для замены плейсхолдеров* replacements = {  
 '{{ref\_number}}': f'{student.id}-СТ/{datetime.now().year}',  
 '{{student\_fio}}': f'{student.surname} {student.name}',  
 '{{group\_name}}': group\_name or 'Не указана',  
 '{{study\_year}}': str(datetime.now().year - 2),  
 '{{email}}': student.email or 'Не указан',  
 '{{phone}}': student.phone or 'Не указан',  
 '{{issue\_date}}': datetime.now().strftime('%d.%m.%Y')  
 }  
  
 *# Замена плейсхолдеров в параграфах* for paragraph in doc.paragraphs:  
 for key, value in replacements.items():  
 if key in paragraph.text:  
 for run in paragraph.runs:  
 if key in run.text:  
 run.text = run.text.replace(key, value)  
  
 *# Замена плейсхолдеров в таблицах* for table in doc.tables:  
 for row in table.rows:  
 for cell in row.cells:  
 for paragraph in cell.paragraphs:  
 for key, value in replacements.items():  
 if key in paragraph.text:  
 for run in paragraph.runs:  
 if key in run.text:  
 run.text = run.text.replace(key, value)  
  
 *# === ДОБАВЛЕНИЕ ПЕЧАТИ ===  
 # Ищем параграф с текстом "М.П." или добавляем печать в конец документа* stamp\_path = 'stamp.png' *# Путь к файлу печати* if os.path.exists(stamp\_path):last\_paragraph = doc.paragraphs[-1]  
  
 *# Создаем новый параграф для печати* stamp\_paragraph = last\_paragraph.insert\_paragraph\_before()  
 stamp\_run = stamp\_paragraph.add\_run()  
  
 *# Вставляем изображение печати  
 # Размер: 4см x 4см (компактная печать)* stamp\_run.add\_picture(stamp\_path, width=Cm(4), height=Cm(4))  
  
 *# Выравнивание по левому краю (где обычно ставится М.П.)* stamp\_paragraph.alignment = WD\_ALIGN\_PARAGRAPH.LEFT  
  
 *# Добавляем отступ слева для выравнивания с текстом "М.П."* stamp\_paragraph.paragraph\_format.left\_indent = Cm(0.5)  
  
 *# Сохранение в буфер* buffer = io.BytesIO()  
 doc.save(buffer)  
 buffer.seek(0)  
  
 return buffer

Листинг 26 – API для экспорта docx-файла

@app.get("/api/export/student/{student\_id}/certificate-word")  
def export\_student\_certificate\_word(student\_id: int, db: Session = Depends(get\_db)):  
 *"""Экспорт справки студента в Word"""* student = db.query(models.Student).filter(models.Student.id == student\_id).first()  
 if not student:  
 raise HTTPException(status\_code=404, detail="Студент не найден")  
  
 group\_name = None  
 if student.group\_id:  
 group = db.query(models.Group).filter(models.Group.id == student.group\_id).first()  
 group\_name = group.name if group else None  
  
 buffer = create\_student\_certificate(student, group\_name)  
  
 *# Безопасное имя файла без русских символов* filename = f"certificate\_{student.surname}\_{student.name}.docx"  
 safe\_filename = urllib.parse.quote(filename)  
  
 return StreamingResponse(  
 buffer,  
 media\_type="application/vnd.openxmlformats-officedocument.wordprocessingml.document",  
 headers={  
 "Content-Disposition": f"attachment; filename={safe\_filename}; filename\*=UTF-8''{safe\_filename}"  
 }  
 )

- создание документа .xlsx

Для работы с excel используется библиотека **openpyxl.** Само создание excel-файла делается в модуле export\_utils.py, функция create\_schelude\_excel()



Рис. 12 «Пример excel»

Листинг 27 – создание excel-файла

def create\_schedule\_excel(group, schedules, subjects\_dict):  
 *"""Создание расписания в Excel с профессиональным оформлением"""* wb = Workbook()  
 ws = wb.active  
 ws.title = "Расписание"  
  
 *# === СТИЛИ ===* header\_font = Font(name='Arial', size=14, bold=True, color='FFFFFF')  
 header\_fill = PatternFill(start\_color='4472C4', end\_color='4472C4', fill\_type='solid')  
 cell\_font = Font(name='Arial', size=11)  
 title\_font = Font(name='Arial', size=16, bold=True)  
 subtitle\_font = Font(name='Arial', size=12, bold=True)  
  
 border = Border(  
 left=Side(style='thin'),  
 right=Side(style='thin'),  
 top=Side(style='thin'),  
 bottom=Side(style='thin')  
 )  
  
 *# === ШАПКА ДОКУМЕНТА ===* ws.merge\_cells('A1:G1')  
 cell = ws['A1']  
 cell.value = 'ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ'  
 cell.font = title\_font  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
 ws.row\_dimensions[1].height = 25  
  
 ws.merge\_cells('A2:G2')  
 cell = ws['A2']  
 cell.value = '"ТЕХНИЧЕСКИЙ КОЛЛЕДЖ"'  
 cell.font = Font(name='Arial', size=14, bold=True)  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
 ws.row\_dimensions[2].height = 20  
  
 ws.merge\_cells('A3:G3')  
 cell = ws['A3']  
 cell.value = 'г. Москва, ул. Профессиональная, д. 15'  
 cell.font = Font(name='Arial', size=10)  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
  
 ws.merge\_cells('A5:G5')  
 cell = ws['A5']  
 cell.value = f'РАСПИСАНИЕ ЗАНЯТИЙ'  
 cell.font = Font(name='Arial', size=14, bold=True)  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
 ws.row\_dimensions[5].height = 25  
  
 ws.merge\_cells('A6:G6')  
 cell = ws['A6']  
 cell.value = f'Группа: {group.name}'  
 cell.font = subtitle\_font  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
 ws.row\_dimensions[6].height = 20  
  
 ws.merge\_cells('A7:G7')  
 cell = ws['A7']  
 cell.value = f'Учебный год: {datetime.now().year}-{datetime.now().year + 1}'  
 cell.font = Font(name='Arial', size=10, italic=True)  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
  
 *# === ЗАГОЛОВКИ ТАБЛИЦЫ ===* days = ['Пара', 'Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота']  
 row = 9  
  
 for col, day in enumerate(days, start=1):  
 cell = ws.cell(row=row, column=col)  
 cell.value = day  
 cell.font = header\_font  
 cell.fill = header\_fill  
 cell.alignment = Alignment(horizontal='center', vertical='center')  
 cell.border = border  
  
 ws.row\_dimensions[row].height = 30  
  
 *# === НАСТРОЙКА ШИРИНЫ КОЛОНОК ===* ws.column\_dimensions['A'].width = 10  
 for col in ['B', 'C', 'D', 'E', 'F', 'G']:  
 ws.column\_dimensions[col].width = 22  
  
 *# === ЗАПОЛНЕНИЕ РАСПИСАНИЯ ===* lesson\_times = [  
 '08:30 - 10:00',  
 '10:10 - 11:40',  
 '12:00 - 13:30',  
 '13:40 - 15:10'  
 ]  
  
 for lesson\_num in range(1, 5):  
 row += 1  
  
 *# Номер пары с временем* cell = ws.cell(row=row, column=1)  
 cell.value = f'{lesson\_num}\n{lesson\_times[lesson\_num - 1]}'  
 cell.alignment = Alignment(horizontal='center', vertical='center', wrap\_text=True)  
 cell.border = border  
 cell.font = Font(name='Arial', size=10, bold=True)  
  
 for col, day in enumerate(days[1:], start=2):  
 cell = ws.cell(row=row, column=col)  
  
 *# Поиск занятия* schedule\_item = next((s for s in schedules  
 if s.lesson\_number == lesson\_num and s.day\_of\_week == day), None)  
  
 if schedule\_item:  
 subject\_name = subjects\_dict.get(schedule\_item.subject\_id, 'Неизвестно')  
 room = f'\n(ауд. {schedule\_item.room})' if schedule\_item.room else ''  
 cell.value = f'{subject\_name}{room}'  
 cell.fill = PatternFill(start\_color='E7E6E6', end\_color='E7E6E6', fill\_type='solid')  
 else:  
 cell.value = '-'  
  
 cell.alignment = Alignment(horizontal='center', vertical='center', wrap\_text=True)  
 cell.border = border  
 cell.font = cell\_font  
  
 ws.row\_dimensions[row].height = 50  
  
 *# === ПОДПИСЬ И ДАТА ===* row += 3  
 ws.merge\_cells(f'A{row}:G{row}')  
 cell = ws[f'A{row}']  
 cell.value = f'Дата формирования расписания: {datetime.now().strftime("%d.%m.%Y")}'  
 cell.font = Font(name='Arial', size=10, italic=True)  
 cell.alignment = Alignment(horizontal='center')  
  
 row += 2  
 ws.merge\_cells(f'A{row}:D{row}')  
 cell = ws[f'A{row}']  
 cell.value = 'Заместитель директора по УР:'  
 cell.font = Font(name='Arial', size=11)  
  
 ws.merge\_cells(f'E{row}:F{row}')  
 cell = ws[f'E{row}']  
 cell.value = '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'  
 cell.font = Font(name='Arial', size=11)  
 cell.alignment = Alignment(horizontal='center')  
  
 cell = ws[f'G{row}']  
 cell.value = 'С.П. Сидоров'  
 cell.font = Font(name='Arial', size=11, bold=True)  
  
 *# Сохранение в буфер* buffer = io.BytesIO()  
 wb.save(buffer)  
 buffer.seek(0)  
  
 return buffer

Листинг 28 – API для экспорта excel-файла

@app.get("/api/export/schedule/{group\_id}/excel")  
def export\_schedule\_excel(group\_id: int, db: Session = Depends(get\_db)):  
 *"""Экспорт расписания группы в Excel"""* group = db.query(models.Group).filter(models.Group.id == group\_id).first()  
 if not group:  
 raise HTTPException(status\_code=404, detail="Группа не найдена")  
  
 schedules = db.query(models.Schedule).filter(models.Schedule.group\_id == group\_id).all()  
 subjects = db.query(models.Subject).all()  
 subjects\_dict = {s.id: s.name for s in subjects}  
  
 buffer = create\_schedule\_excel(group, schedules, subjects\_dict)  
  
 *# Безопасное имя файла без русских символов* filename = f"schedule\_{group.name}.xlsx"  
 safe\_filename = urllib.parse.quote(filename)  
  
 return StreamingResponse(  
 buffer,  
 media\_type="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",  
 headers={  
 "Content-Disposition": f"attachment; filename={safe\_filename}; filename\*=UTF-8''{safe\_filename}"  
 }  
 )

- создание документа .pdf

Для генерации pdf-файла используется библиотека ReportLab. Сама генерация принимает форму листа А4 с красивой рамкой и вставленной информацией с помощью модуля export\_utils.py, функция create\_student\_certificate\_pdf().



Рис. 13 «Пример pdf»

Листинг 29 – создание pdf-файла

def create\_student\_certificate\_pdf(student, group\_name):  
 *"""Создание справки студента в PDF - отличается от Word версии"""* buffer = io.BytesIO()  
 c = canvas.Canvas(buffer, pagesize=A4)  
 width, height = A4  
  
 *# Регистрация русского шрифта (DejaVu Sans поддерживает кириллицу)* try:  
 *# Пробуем загрузить DejaVu Sans* pdfmetrics.registerFont(TTFont('DejaVu', '/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf'))  
 pdfmetrics.registerFont(TTFont('DejaVu-Bold', '/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf'))  
 font\_regular = 'DejaVu'  
 font\_bold = 'DejaVu-Bold'  
 except:  
 *# Если не получилось, пробуем Arial (Windows)* try:  
 pdfmetrics.registerFont(TTFont('Arial', 'arial.ttf'))  
 pdfmetrics.registerFont(TTFont('Arial-Bold', 'arialbd.ttf'))  
 font\_regular = 'Arial'  
 font\_bold = 'Arial-Bold'  
 except:  
 *# В крайнем случае используем встроенный Helvetica (но он не поддерживает кириллицу)* font\_regular = 'Helvetica'  
 font\_bold = 'Helvetica-Bold'  
  
 *# === РАМКА ДОКУМЕНТА ===* c.setStrokeColorRGB(0.2, 0.2, 0.8)  
 c.setLineWidth(2)  
 c.rect(40, 40, width - 80, height - 80, stroke=1, fill=0)  
  
 *# === ШАПКА ===* c.setFont(font\_bold, 16)  
 c.drawCentredString(width / 2, height - 80, "МИНИСТЕРСТВО ОБРАЗОВАНИЯ")  
  
 c.setFont(font\_bold, 14)  
 c.drawCentredString(width / 2, height - 105, "ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ")  
 c.drawCentredString(width / 2, height - 125, '"ТЕХНИЧЕСКИЙ КОЛЛЕДЖ"')  
  
 *# Линия разделителя* c.setStrokeColorRGB(0.5, 0.5, 0.5)  
 c.setLineWidth(1)  
 c.line(80, height - 140, width - 80, height - 140)  
  
 *# === ЗАГОЛОВОК ===* c.setFont(font\_bold, 20)  
 c.drawCentredString(width / 2, height - 180, "СПРАВКА")  
  
 c.setFont(font\_bold, 12)  
 ref\_number = f"№ {student.id}-PDF/{datetime.now().year}"  
 c.drawCentredString(width / 2, height - 205, ref\_number)  
  
 *# === ОСНОВНОЙ ТЕКСТ ===* c.setFont(font\_regular, 12)  
 y = height - 250  
  
 *# Многострочный текст с переносами* text\_width = width - 160  
  
 text1 = f"Выдана студенту(ке) {student.surname} {student.name}"  
 c.drawString(80, y, text1)  
 y -= 25  
  
 text2 = "в том, что он(а) обучается в"  
 c.drawString(80, y, text2)  
 y -= 20  
  
 c.setFont(font\_bold, 12)  
 text3 = 'Государственном образовательном учреждении'  
 c.drawString(80, y, text3)  
 y -= 20  
  
 text4 = '"Технический колледж"'  
 c.drawString(80, y, text4)  
 y -= 25  
  
 c.setFont(font\_regular, 12)  
 text5 = f"по программе среднего профессионального образования"  
 c.drawString(80, y, text5)  
 y -= 20  
  
 text6 = f"в группе: {group\_name or 'Не указана'}"  
 c.drawString(80, y, text6)  
 y -= 20  
  
 text7 = f"с {datetime.now().year - 2} года по настоящее время."  
 c.drawString(80, y, text7)  
  
 *# === ТАБЛИЦА С ДАННЫМИ ===* y -= 40  
 c.setFont(font\_bold, 11)  
 c.drawString(80, y, "Контактные данные студента:")  
 y -= 25  
  
 *# Рисуем таблицу* c.setStrokeColorRGB(0.3, 0.3, 0.3)  
 c.setLineWidth(0.5)  
  
 *# Заголовки* c.setFont(font\_bold, 10)  
 c.drawString(90, y, "ФИО:")  
 c.drawString(90, y - 20, "Группа:")  
 c.drawString(90, y - 40, "Email:")  
 c.drawString(90, y - 60, "Телефон:")  
  
 *# Значения* c.setFont(font\_regular, 10)  
 c.drawString(200, y, f"{student.surname} {student.name}")  
 c.drawString(200, y - 20, f"{group\_name or 'Не указана'}")  
 c.drawString(200, y - 40, f"{student.email or 'Не указан'}")  
 c.drawString(200, y - 60, f"{student.phone or 'Не указан'}")  
  
 *# Рамка таблицы* c.rect(85, y - 75, width - 170, 90, stroke=1, fill=0)  
  
 *# === НАЗНАЧЕНИЕ ===* y -= 100  
 c.setFont(font\_regular, 11)  
 c.drawString(80, y, "Справка дана для предъявления по месту требования.")  
  
 *# === ДАТА ===* y -= 40  
 c.setFont(font\_bold, 11)  
 date\_str = f"Дата выдачи: {datetime.now().strftime('%d.%m.%Y')}"  
 c.drawString(80, y, date\_str)  
  
 *# === ПОДПИСИ ===* y -= 60  
 c.setFont(font\_regular, 11)  
 c.drawString(80, y, "Директор колледжа")  
 c.drawString(270, y, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
 c.setFont(font\_bold, 11)  
 c.drawString(420, y, "И.И. Иванов")  
  
 y -= 30  
 c.setFont(font\_regular, 11)  
 c.drawString(80, y, "Зам. директора по УР")  
 c.drawString(270, y, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
 c.setFont(font\_bold, 11)  
 c.drawString(420, y, "С.П. Сидоров")  
  
 *# === ПЕЧАТЬ ===* y -= 40  
 c.setFont(font\_regular, 10)  
 c.drawString(80, y, "М.П.")  
  
 *# Добавляем изображение печати* stamp\_path = 'stamp.png'  
 if os.path.exists(stamp\_path):  
 *# Вставляем печать рядом с "М.П."* c.drawImage(stamp\_path, 110, y - 50, width=80, height=80, preserveAspectRatio=True, mask='auto')  
  
 c.save()  
 buffer.seek(0)  
  
 return buffer

Листинг 30 – API для экспорт pdf-файла

@app.get("/api/export/student/{student\_id}/certificate-pdf")  
def export\_student\_certificate\_pdf(student\_id: int, db: Session = Depends(get\_db)):  
 *"""Экспорт справки студента в PDF"""* student = db.query(models.Student).filter(models.Student.id == student\_id).first()  
 if not student:  
 raise HTTPException(status\_code=404, detail="Студент не найден")  
  
 group\_name = None  
 if student.group\_id:  
 group = db.query(models.Group).filter(models.Group.id == student.group\_id).first()  
 group\_name = group.name if group else None  
  
 buffer = create\_student\_certificate\_pdf(student, group\_name)  
  
 *# Безопасное имя файла без русских символов* filename = f"certificate\_{student.surname}\_{student.name}.pdf"  
 safe\_filename = urllib.parse.quote(filename)  
  
 return StreamingResponse(  
 buffer,  
 media\_type="application/pdf",  
 headers={  
 "Content-Disposition": f"attachment; filename={safe\_filename}; filename\*=UTF-8''{safe\_filename}"  
 }  
 )

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