**Содержание**

Введение

1. Разработка технического задания

2. Проектирование интерфейса приложения

Figma — это онлайн-платформа для дизайна и прототипирования интерфейсов, которая позволяет создавать макеты и взаимодействующие прототипы веб-страниц, мобильных приложений и других цифровых продуктов. Это инструмент, в котором дизайнеры, разработчики и другие члены команды могут работать совместно в реальном времени.

+ Интуитивно – понятный интерфейс

+ Возможность генерации интерфейса по запросу с помощью ИИ

+ Автоматическое сохранение проектов и возможность совместной работы

- Без подключения не полностью работает функционал

- Подтормаживает и может зависать с большими проектами

- Ограничение бесплатного функционала

Я выбрал Figma. Причиной стало то, что у меня уже был небольшой опыт работы в этой программе. Мне не пришлось ее устанавливать и изучать первичный функционал, поскольку я уже делал проекты в ней.

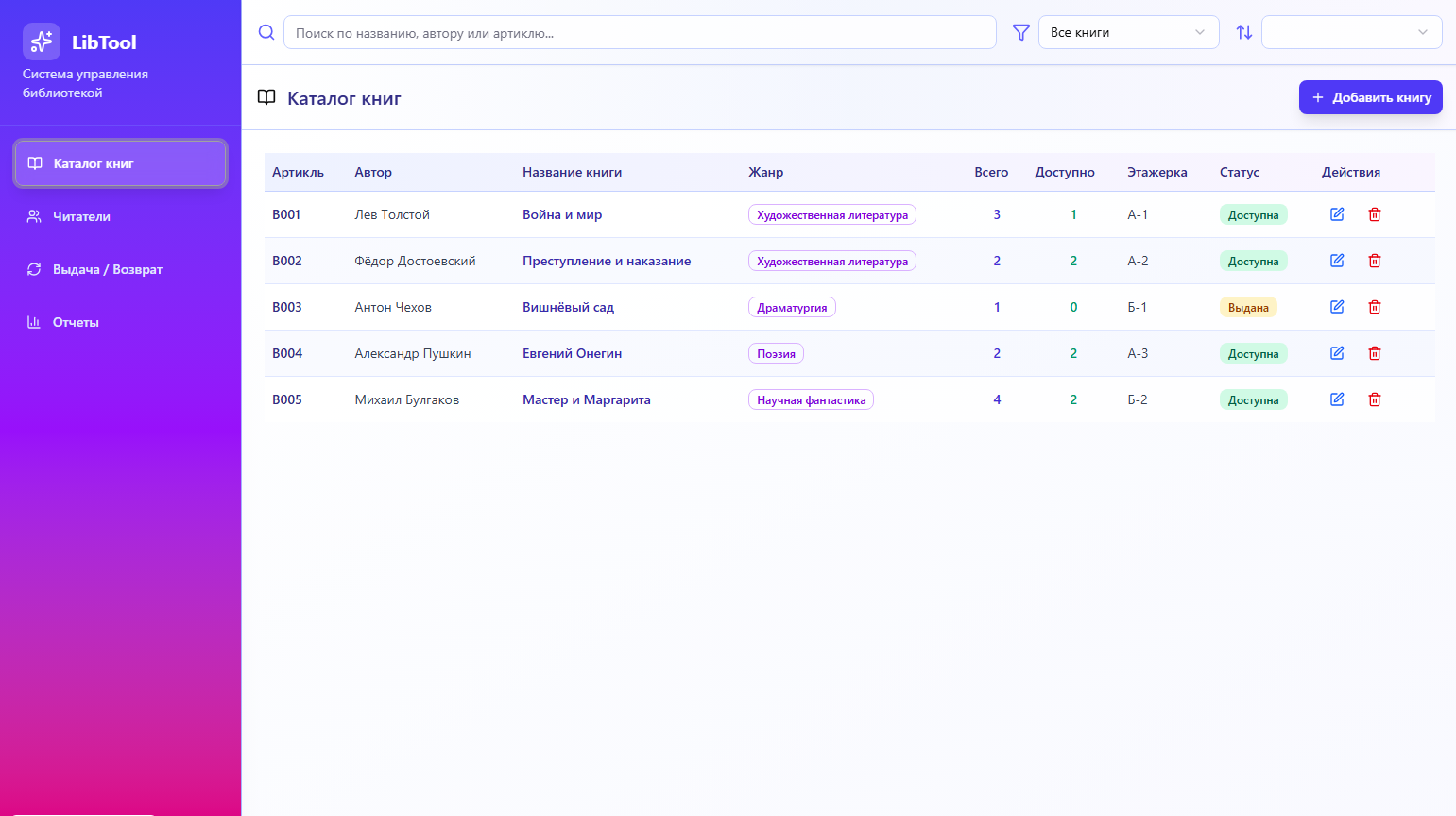


Рис.1 «Каталог книг»

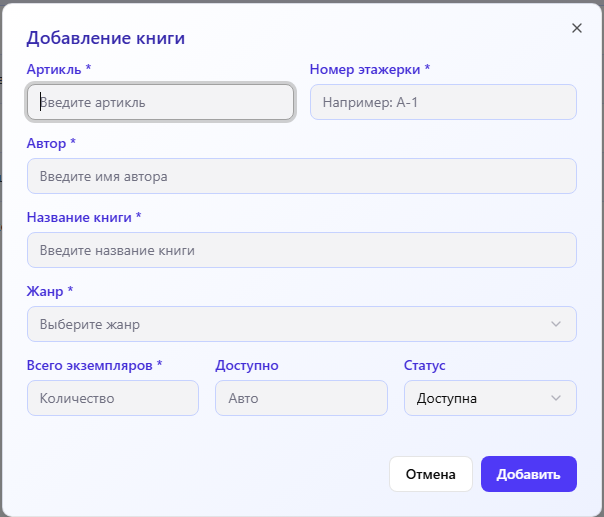


Рис.2 «Окно добавления книги»

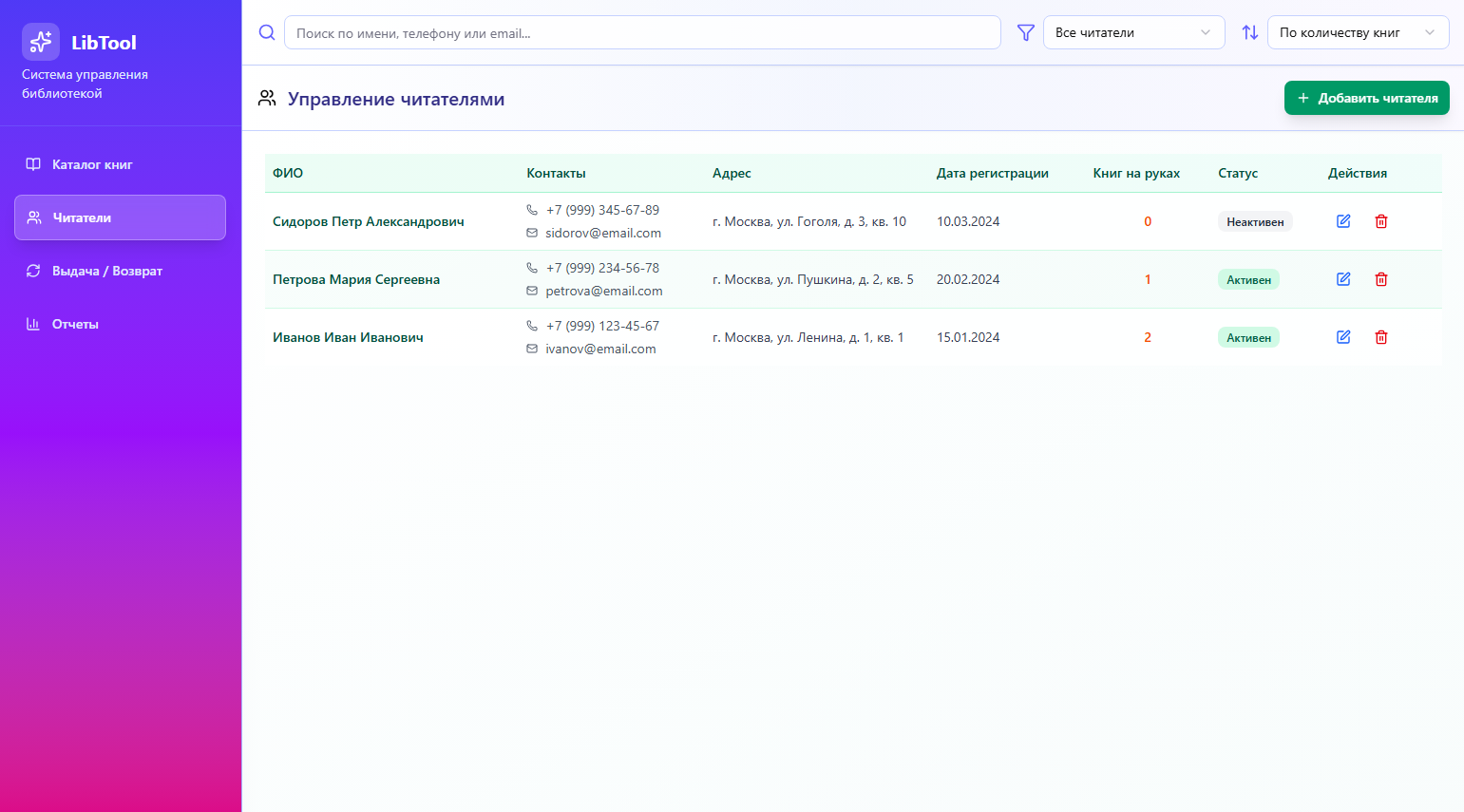


Рис.3 «Читатели»

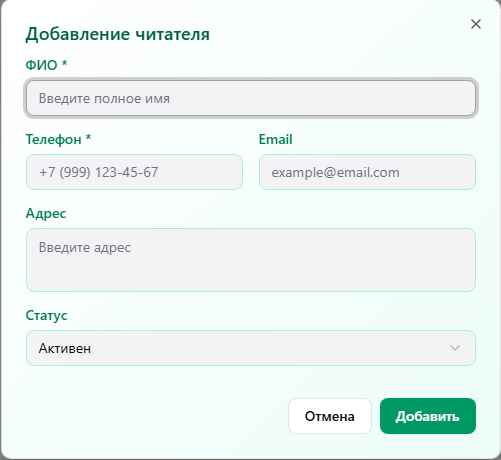


Рис.4 «Окно добавления читателя»

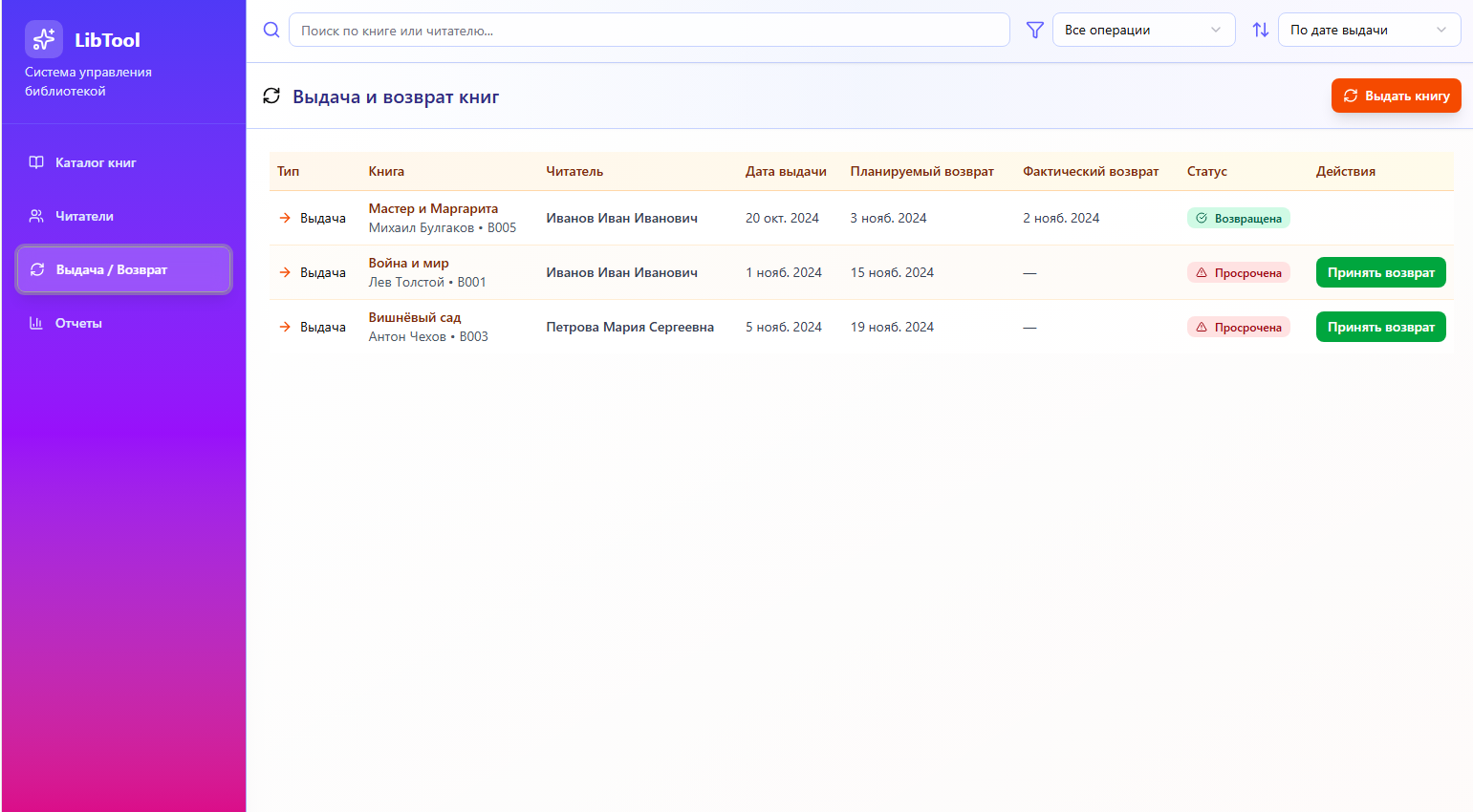


Рис.5 «Выдача/Возврат»

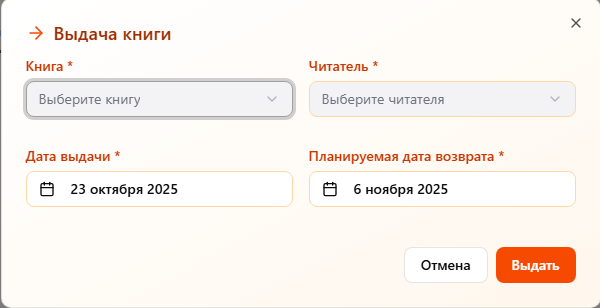


Рис.6 «Окно оформления выдачи книги»

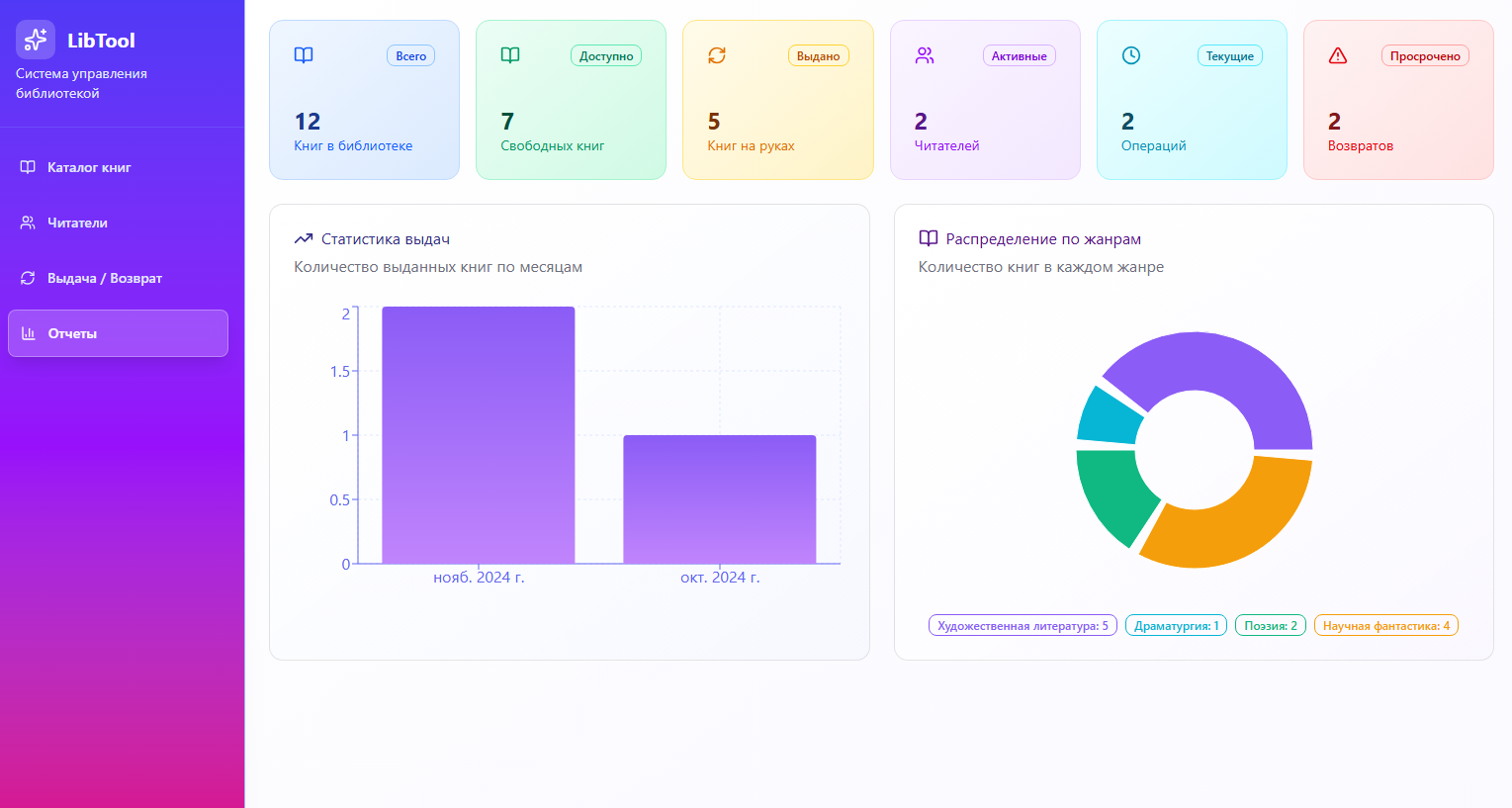


Рис.7 «Отчеты»

3. Проектирование и создание базы данных

3.1. Разработка структуры БД

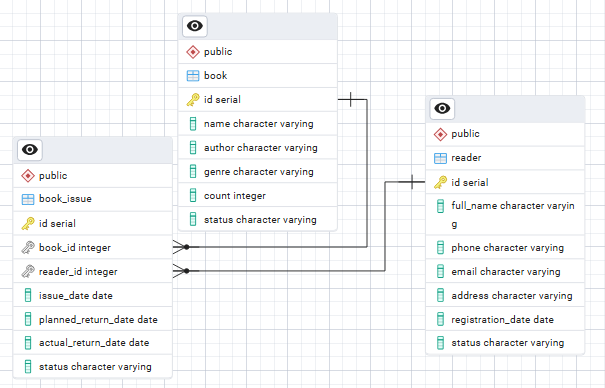


Рис.8 «Схема данных»

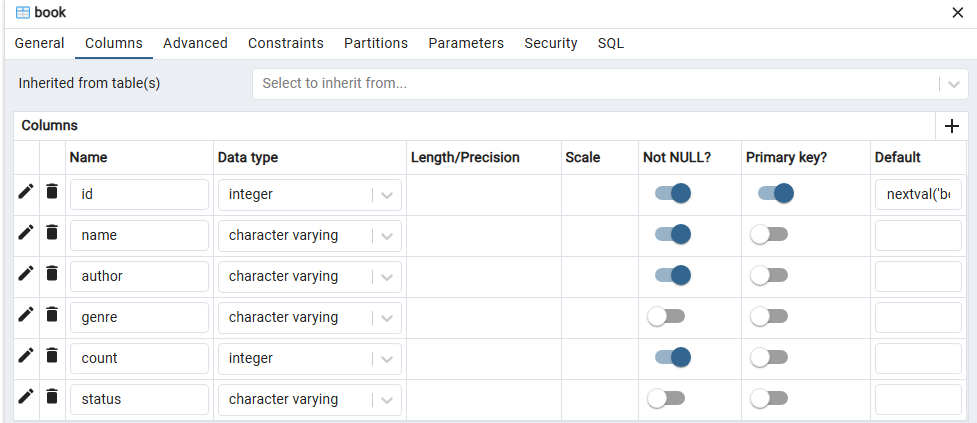


Рис.9 «Таблица book»

*Имя: id; Тип данных: integer*Not NULL  
Primary Key  
Autoincrement

*Имя: name; Тип данных: character varying*  
Not NULL

*Имя: author; Тип данных: character varying*Not NULL

*Имя: genre; Тип данных: character varying*

*Имя: count; Тип данных: integer*  
Not NULL

*Имя: status; Тип данных: character varying*

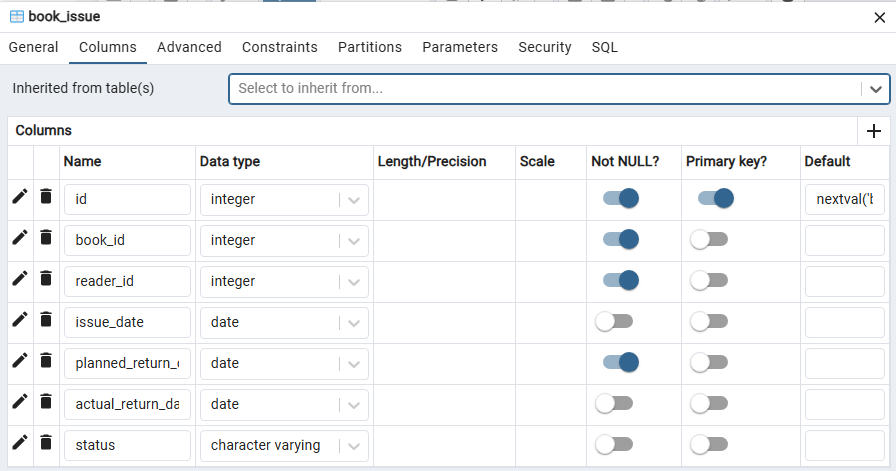


Рис.10 «Таблица book\_issue»

*Имя: id; Тип данных: integer*Not NULL  
Primary Key  
Autoincrement

*Имя: book\_id; Тип данных: integer*Not NULL  
Foreing Key

*Имя: reader\_id; Тип данных: integer*Not NULL  
Foreing Key

*Имя: issue\_date; Тип данных: date*

*Имя: planned\_return\_date; Тип данных: date*Not NULL

*Имя: actual\_return\_date; Тип данных: date*

*Имя: status; Тип данных: character varying*

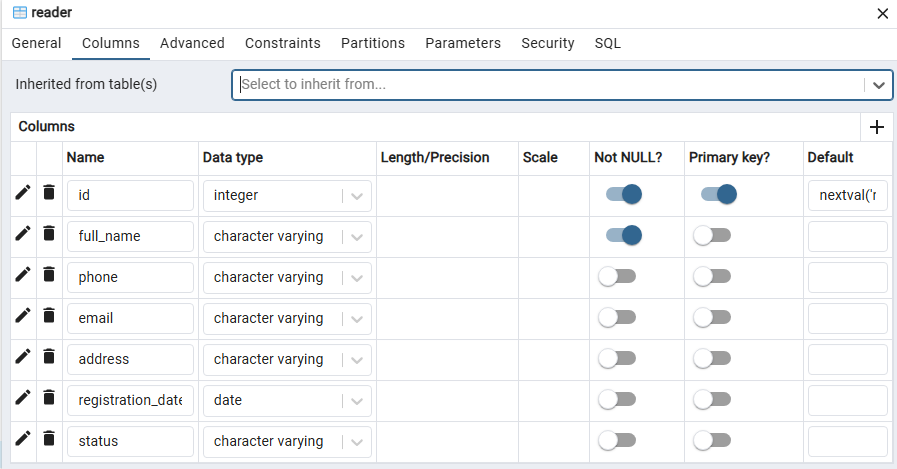


Рис.11 «Таблица reader»

*Имя: id; Тип данных: integer*Not NULL  
Primary Key  
Autoincrement

*Имя: full\_name; Тип данных: character varying*Not NULL

*Имя: phone; Тип данных: character varying*

*Имя: email; Тип данных: character varying*

*Имя: address; Тип данных: character varying*

*Имя: registration\_date; Тип данных: date*

*Имя: status; Тип данных: character varying*

3.2. Создание модели базы данных

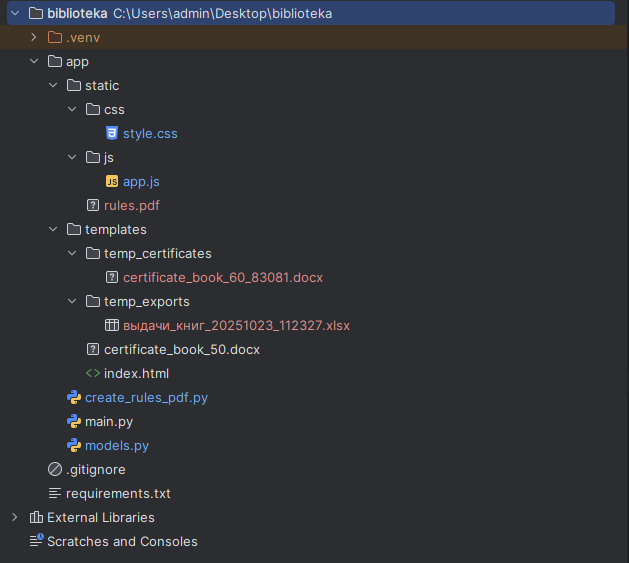
****

Рис.12 «Структура проекта»

- модуль классов

Листинг 1 - Описание моделей БД для хранения информации

class Book(Base):  
 \_\_tablename\_\_ = "book"  
 id = Column(Integer, primary\_key=True, index=True)  
 name = Column(String(200), nullable=False)  
 author = Column(String(100), nullable=False)  
 genre = Column(String(50), nullable=True)  
 count = Column(Integer, default=1, nullable=False)  
 status = Column(String(20), default="available", nullable=False)  
  
 issues = relationship("BookIssue", back\_populates="book")  
  
class Reader(Base):  
 \_\_tablename\_\_ = "reader"  
 id = Column(Integer, primary\_key=True, index=True)  
 full\_name = Column(String(100), nullable=False)  
 phone = Column(String(20), nullable=True)  
 email = Column(String(100), nullable=True)  
 address = Column(Text, nullable=True)  
 registration\_date = Column(Date, default=date.today)  
 status = Column(String(20), default="active")  
  
 issues = relationship("BookIssue", back\_populates="reader")  
  
class BookIssue(Base):  
 \_\_tablename\_\_ = "book\_issue"  
 id = Column(Integer, primary\_key=True, index=True)  
 book\_id = Column(Integer, ForeignKey("book.id"), nullable=False)  
 reader\_id = Column(Integer, ForeignKey("reader.id"), nullable=False)  
 issue\_date = Column(Date, default=date.today)  
 planned\_return\_date = Column(Date, nullable=False)  
 actual\_return\_date = Column(Date, nullable=True)  
 status = Column(String(20), default="issued") *# issued, returned, overdue* book = relationship("Book", back\_populates="issues")  
 reader = relationship("Reader", back\_populates="issues")

4. Разработка программных модулей

4.1. Создание сервера

- модуль подключения к базе данных

Листинг 2 – Подключение к БД Postgres

DATABASE\_URL = "postgresql+psycopg2://postgres:1234@localhost:5432/libTool"

engine = create\_engine(DATABASE\_URL, echo=True)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

def create\_tables():

Base.metadata.create\_all(bind=engine)

Листинг 3 – Основная настройка и запуска API с интерфейсом

BASE\_DIR = Path(\_\_file\_\_).resolve().parent # Текущая директория (app)

static\_dir = BASE\_DIR / 'static'

templates\_dir = BASE\_DIR / 'templates'

# Создаем папки если их нет

static\_dir.mkdir(exist\_ok=True, parents=True)

templates\_dir.mkdir(exist\_ok=True, parents=True)

# Монтируем статические файлы

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

templates = Jinja2Templates(directory=templates\_dir)

Листинг 4 – Создание таблиц при запуске и автоматическая проверка просроченного возврата книг

@app.on\_event("startup")

def startup\_event():

create\_tables()

try:

db = next(get\_db())

updated\_count = book\_issue\_store.check\_overdue\_issues(db)

print(f"🔍 Автоматически проверены просрочки: обновлено {updated\_count} выдач")

except Exception as e:

print(f"⚠️ Ошибка автоматической проверки просрочек: {e}")

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

Листинг 5 – Главная страница

@app.get("/")

async def index(request: Request):

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

Листинг 6 – Вывод списка книг, удаление, поиск и добавление

@app.get("/api/books", response\_model=List[BookOut])

async def get\_books(db: Session = Depends(get\_db)):

try:

books = book\_store.list\_books(db)

return books

except Exception as e:

raise HTTPException(500, f"Ошибка загрузки книг: {str(e)}")

@app.get("/api/books/{book\_id}", response\_model=BookOut)

async def get\_book(book\_id: int, db: Session = Depends(get\_db)):

book = book\_store.get\_book(db, book\_id)

if not book:

raise HTTPException(404, "Книга не найдена")

return book

@app.post("/api/books", response\_model=BookOut)

async def create\_book(book: BookCreate, db: Session = Depends(get\_db)):

try:

return book\_store.create\_book(db, book)

except Exception as e:

raise HTTPException(500, f"Ошибка создания книги: {str(e)}")

@app.put("/api/books/{book\_id}", response\_model=BookOut)

async def update\_book(book\_id: int, book: BookUpdate, db: Session = Depends(get\_db)):

updated\_book = book\_store.update\_book(db, book\_id, book)

if not updated\_book:

raise HTTPException(404, "Книга не найдена")

return updated\_book

@app.delete("/api/books/{book\_id}")

async def delete\_book(book\_id: int, db: Session = Depends(get\_db)):

success = book\_store.delete\_book(db, book\_id)

if not success:

raise HTTPException(404, "Книга не найдена")

return {"ok": True}

Листинг 7 – Вывод списка читателей, удаление, поиск и добавление

@app.get("/api/readers", response\_model=List[ReaderOut])

async def get\_readers(db: Session = Depends(get\_db)):

try:

readers = reader\_store.list\_readers(db)

return readers

except Exception as e:

raise HTTPException(500, f"Ошибка загрузки читателей: {str(e)}")

@app.post("/api/readers", response\_model=ReaderOut)

async def create\_reader(reader: ReaderCreate, db: Session = Depends(get\_db)):

try:

return reader\_store.create\_reader(db, reader)

except Exception as e:

raise HTTPException(500, f"Ошибка создания читателя: {str(e)}")

@app.put("/api/readers/{reader\_id}", response\_model=ReaderOut)

async def update\_reader(reader\_id: int, reader: ReaderUpdate, db: Session = Depends(get\_db)):

updated\_reader = reader\_store.update\_reader(db, reader\_id, reader)

if not updated\_reader:

raise HTTPException(404, "Читатель не найден")

return updated\_reader

@app.delete("/api/readers/{reader\_id}")

async def delete\_reader(reader\_id: int, db: Session = Depends(get\_db)):

success = reader\_store.delete\_reader(db, reader\_id)

if not success:

raise HTTPException(404, "Читатель не найден")

return {"ok": True}

Листинг 8 – Выдача/возврат книг

@app.get("/api/issues", response\_model=List[BookIssueOut])

async def get\_issues(db: Session = Depends(get\_db)):

try:

issues = book\_issue\_store.list\_issues(db)

return issues

except Exception as e:

raise HTTPException(500, f"Ошибка загрузки выдач: {str(e)}")

@app.post("/api/issues", response\_model=BookIssueOut)

async def issue\_book(issue: BookIssueCreate, db: Session = Depends(get\_db)):

try:

return book\_issue\_store.issue\_book(db, issue)

except Exception as e:

raise HTTPException(500, f"Ошибка выдачи книги: {str(e)}")

return {"ok": True, "message": "Выдача отмечена как просроченная"}

except Exception as e:

raise HTTPException(500, f"Ошибка отметки просрочки: {str(e)}")

raise HTTPException(404, "Выдача не найдена или уже возвращена")

return {"ok": True}

Листинг 9 – Проверка на просроченную выдачу и принудительная отметка как просроченную

@app.post("/api/issues/check-overdue")

async def check\_overdue\_issues(db: Session = Depends(get\_db)):

"""Проверить и обновить статусы просроченных выдач"""

try:

updated\_count = book\_issue\_store.check\_overdue\_issues(db)

return {

"updated\_count": updated\_count,

"message": f"Обновлено {updated\_count} просроченных выдач"

}

except Exception as e:

raise HTTPException(500, f"Ошибка проверки просрочек: {str(e)}")

@app.post("/api/issues/{issue\_id}/mark-overdue")

async def mark\_issue\_overdue(issue\_id: int, db: Session = Depends(get\_db)):

"""Принудительно отметить выдачу как просроченную"""

try:

success = book\_issue\_store.mark\_issue\_overdue(db, issue\_id)

if not success:

raise HTTPException(400, "Не удалось отметить выдачу как просроченную")

return {"ok": True, "message": "Выдача отмечена как просроченная"}

except Exception as e:

raise HTTPException(500, f"Ошибка отметки просрочки: {str(e)}")

Листинг 10 – Отчет/статистика

@app.get("/api/reports/stats")

async def get\_stats(db: Session = Depends(get\_db)):

try:

# Статистика по книгам

books = book\_store.list\_books(db)

total\_books = len(books)

available\_books = len([b for b in books if b.status == "available"])

# Статистика по читателям

readers = reader\_store.list\_readers(db)

total\_readers = len(readers)

active\_readers = len([r for r in readers if r.status == "active"])

# Статистика по выдачам

issues = book\_issue\_store.list\_issues(db)

total\_issues = len(issues)

current\_issues = len([i for i in issues if i.status == "issued"])

overdue\_issues = len([i for i in issues if i.status == "overdue"])

returned\_issues = len([i for i in issues if i.status == "returned"])

# Статистика по жанрам

genre\_stats = {}

for book in books:

genre = book.genre or "Без жанра"

genre\_stats[genre] = genre\_stats.get(genre, 0) + 1

return {

"books": {

"total": total\_books,

"available": available\_books

},

"readers": {

"total": total\_readers,

"active": active\_readers

},

"issues": {

"total": total\_issues,

"current": current\_issues,

"overdue": overdue\_issues,

"returned": returned\_issues

},

"genres": genre\_stats

}

except Exception as e:

raise HTTPException(500, f"Ошибка загрузки статистики: {str(e)}")

Листинг 11 – Проверка работоспособности

@app.get("/api/health")

async def health\_check(db: Session = Depends(get\_db)):

try:

books = book\_store.list\_books(db)

return {

"status": "healthy",

"database": "connected",

"books\_count": len(books)

}

except Exception as e:

raise HTTPException(500, f"Database error: {str(e)}")

4.2. Создание клиента

Для реализации клиентской части я использовал HTML, JavaScript, CSS.

Основной модуль index.html, в котором реализован первичный интерфейс (кнопки, подписи, модальные окна). Для более красивой реализации интерфейса у меня есть модуль style.css, в котором прописаны все стили кнопок и остальных элементов. Также у меня есть модуль app.js, в котором прописаны скрипты, благодаря которым фронтэнд и бэкэнд работают вместе, кэшируются данные, обновляется динамически интерфейс, считывается пользовательский ввод. В целом app.js помогает превратить обычное приложение с API в практически полноценное Web - приложение для библиотеки.

Листинг 12 – Боковая панель управления (index.html)

<aside class="sidebar">

<div class="brand">

<div class="logo">📚</div>

<div class="title">LibTool</div>

</div>

<nav class="navigation">

<button class="nav-btn active" data-page="books">

<span>📖</span>

<span>Каталог книг</span>

</button>

<button class="nav-btn" data-page="readers">

<span>👥</span>

<span>Читатели</span>

</button>

<button class="nav-btn" data-page="issues">

<span>🔄</span>

<span>Выдача/Возврат</span>

</button>

<button class="nav-btn" data-page="reports">

<span>📊</span>

<span>Отчеты</span>

</button>

<button id="downloadRules" class="btn btn-info">

<i class="fas fa-download"></i> Скачать правила библиотеки

</button>

</nav>

Листинг 13 – Реализация отчетов и работы диаграммы (app.js)

renderReports(stats) {

const container = document.getElementById('reports-container');

if (!container) return;

const statsItems = [

{ label: 'Всего книг:', value: stats.books.total },

{ label: 'Доступно книг:', value: stats.books.available },

{ label: 'Всего читателей:', value: stats.readers.total },

{ label: 'Активных читателей:', value: stats.readers.active },

{ label: 'Всего выдач:', value: stats.issues.total },

{ label: 'Текущих выдач:', value: stats.issues.current }

];

let html = `

<div class="reports-container">

<div class="report-card">

<h3>📊 Общая статистика</h3>

<div class="stats-grid">

${statsItems.map(item => `

<div class="stat-item">

<span>${item.label}</span>

<span class="stat-value">${item.value}</span>

</div>

`).join('')}

</div>

</div>

<div class="report-card">

<h3>📚 Распределение по жанрам</h3>

<div style="height: 300px; position: relative;">

<canvas id="genreChart"></canvas>

</div>

</div>

</div>

`;

container.innerHTML = html;

this.renderGenreChart(stats.genres);

}

renderGenreChart(genres) {

const ctx = document.getElementById('genreChart').getContext('2d');

if (this.genresChart) {

this.genresChart.destroy();

}

this.genresChart = new Chart(ctx, {

type: 'doughnut',

data: {

labels: Object.keys(genres),

datasets: [{

data: Object.values(genres),

backgroundColor: [

'#3498db', '#e74c3c', '#2ecc71', '#9b59b6',

'#f1c40f', '#1abc9c', '#34495e', '#e67e22',

'#95a5a6', '#d35400'

],

borderWidth: 2,

borderColor: '#fff'

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

plugins: {

legend: {

position: 'right',

labels: {

padding: 15,

usePointStyle: true,

font: { size: 12 }

}

},

tooltip: {

callbacks: {

label: function(context) {

const label = context.label || '';

const value = context.parsed;

const total = context.dataset.data.reduce((a, b) => a + b, 0);

const percentage = Math.round((value / total) \* 100);

return `${label}: ${value} (${percentage}%)`;

}

}

}

}

}

});

}

Листинг 14 – Стиль для диаграммы (style.css)

.chart-container {

position: relative;

height: 300px;

width: 100%;

margin: 0 auto;

}

.report-card canvas {

max-height: 300px;

width: 100% !important;

height: auto !important;

}

.chartjs-legend {

display: flex;

flex-wrap: wrap;

gap: 10px;

margin-top: 15px;

justify-content: center;

}

.chartjs-legend-item {

display: flex;

align-items: center;

gap: 5px;

font-size: 12px;

}

.chartjs-legend-color {

width: 12px;

height: 12px;

border-radius: 3px;

}

4.3. Реализация модулей вывода на печать

- вывод в шаблон .docx

Листинг 15 – API генерации сертификата по шаблону, методом замены строк (main.py)

@app.get("/api/certificate/{book\_id}")

async def generate\_certificate(book\_id: int, db: Session = Depends(get\_db)):

try:

print(f"🔍 Запрос на генерацию сертификата для книги ID: {book\_id}")

# Получаем данные книги

book = book\_store.get\_book(db, book\_id)

if not book:

raise HTTPException(404, "Книга не найдена")

# Путь к шаблону сертификата

template\_path = templates\_dir / "certificate\_book\_50.docx"

if not template\_path.exists():

raise HTTPException(500, "Шаблон сертификата не найден")

# Создаем временный файл

cert\_number = random.randint(10000, 99999)

# Фиксируем текущую дату

current\_date = datetime.now()

day = current\_date.day

month = current\_date.month

year = current\_date.year

# Русские названия месяцев

month\_names = [

"января", "февраля", "марта", "апреля", "мая", "июня",

"июля", "августа", "сентября", "октября", "ноября", "декабря"

]

month\_name = month\_names[month - 1]

# Форматируем дату

full\_date = f"«{day}» {month\_name} {year} г."

temp\_dir = templates\_dir / "temp\_certificates"

temp\_dir.mkdir(exist\_ok=True)

output\_path = temp\_dir / f"certificate\_book\_{book\_id}\_{cert\_number}.docx"

# Копируем шаблон

shutil.copy(template\_path, output\_path)

# Открываем документ

doc = Document(output\_path)

# Простая замена по ключевым фразам

for paragraph in doc.paragraphs:

text = paragraph.text

# Заменяем номер сертификата

if "\_\_\_\_\_\_\_\_\_\_" in text and "№" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_", str(cert\_number))

# Заменяем дату

if "«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г." in text:

text = text.replace("«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.", full\_date)

# Заменяем информацию о книге

if "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Название:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", book.name)

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Автор:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", book.author)

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Жанр:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", book.genre or "Не указан")

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Количество экземпляров:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", str(book.count))

# Если текст изменился, обновляем параграф

if text != paragraph.text:

paragraph.clear()

paragraph.add\_run(text)

# Обрабатываем таблицы

for table in doc.tables:

for row in table.rows:

for cell in row.cells:

for paragraph in cell.paragraphs:

text = paragraph.text

if "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Описание повреждений" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "отсутствуют")

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Комментарии проверяющего" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "книга в отличном состоянии")

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Проверил:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "Иванова А.С.")

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Утвердил:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "Петров И.В.")

elif "«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г." in text:

text = text.replace("«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.", full\_date)

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Организация:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"Центральная городская библиотека")

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Адрес:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"г. Москва, ул. Читательская, д. 1")

elif "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" in text and "Телефон:" in text:

text = text.replace("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"+7 (495) 123-45-67")

if text != paragraph.text:

paragraph.clear()

paragraph.add\_run(text)

# Сохраняем изменения

doc.save(output\_path)

return FileResponse(

path=output\_path,

filename=f"Сертификат\_качества\_{book.name.replace(' ', '\_')}.docx",

media\_type='application/vnd.openxmlformats-officedocument.wordprocessingml.document'

)

except Exception as e:

print(f"❌ Ошибка генерации сертификата: {str(e)}")

raise HTTPException(500, f"Ошибка генерации сертификата: {str(e)}")

Листинг 16 – Скачивание сертификата(app.js)

async downloadCertificate(bookId) {

try {

this.showNotification('Генерация сертификата...', 'info');

const response = await fetch(`/api/certificate/${bookId}`);

if (!response.ok) throw new Error('Ошибка генерации сертификата');

await this.downloadFile(response, `certificate\_book\_${bookId}.docx`);

this.showNotification('Сертификат успешно скачан', 'success');

} catch (error) {

this.showNotification('Ошибка скачивания сертификата: ' + error.message, 'error');

}

}

- создание документа .xlsx

Листинг 17 – Генерация Excel файла(main.py)

@app.get("/api/issues/export-excel")

async def export\_issues\_to\_excel(db: Session = Depends(get\_db)):

"""Экспорт списка выдач в Excel файл"""

try:

print("🔍 Запрос на экспорт выдач в Excel...")

# Получаем все выдачи

issues = book\_issue\_store.list\_issues(db)

print(f"📊 Найдено {len(issues)} выдач для экспорта")

# Создаем Excel workbook

wb = Workbook()

ws = wb.active

ws.title = "Выдачи книг"

# Стили

header\_font = Font(bold=True, size=12)

normal\_font = Font(size=10)

border = Border(

left=Side(style='thin'),

right=Side(style='thin'),

top=Side(style='thin'),

bottom=Side(style='thin')

)

center\_align = Alignment(horizontal='center', vertical='center')

# Заголовки

headers = [

'ID', 'Книга', 'Читатель', 'Дата выдачи',

'Планируемый возврат', 'Фактический возврат', 'Статус'

]

# Записываем заголовки

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

cell = ws.cell(row=1, column=col, value=header)

cell.font = header\_font

cell.alignment = center\_align

cell.border = border

# Записываем данные

for row, issue in enumerate(issues, 2):

# Преобразуем статус в русский текст

status\_text = {

'issued': 'Выдана',

'returned': 'Возвращена',

'overdue': 'Просрочена'

}.get(issue.status, issue.status)

ws.cell(row=row, column=1, value=issue.id).border = border

ws.cell(row=row, column=2, value=issue.book\_name).border = border

ws.cell(row=row, column=3, value=issue.reader\_name).border = border

ws.cell(row=row, column=4, value=issue.issue\_date.strftime('%d.%m.%Y')).border = border

ws.cell(row=row, column=5, value=issue.planned\_return\_date.strftime('%d.%m.%Y')).border = border

actual\_return = issue.actual\_return\_date.strftime('%d.%m.%Y') if issue.actual\_return\_date else '-'

ws.cell(row=row, column=6, value=actual\_return).border = border

ws.cell(row=row, column=7, value=status\_text).border = border

# Настраиваем ширину колонок

column\_widths = [8, 40, 30, 12, 15, 15, 12]

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

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

# Добавляем подпись директора

signature\_row = len(issues) + 4

# Пустая строка для разделения

ws.cell(row=signature\_row, column=1, value="")

# Подпись директора

signature\_row += 1

ws.merge\_cells(f'A{signature\_row}:G{signature\_row}')

ws.cell(row=signature\_row, column=1, value="Директор библиотеки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

ws.cell(row=signature\_row, column=1).font = Font(size=12, bold=True)

ws.cell(row=signature\_row, column=1).alignment = Alignment(horizontal='right')

signature\_row += 1

ws.merge\_cells(f'A{signature\_row}:G{signature\_row}')

ws.cell(row=signature\_row, column=1, value="Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

ws.cell(row=signature\_row, column=1).font = Font(size=12)

ws.cell(row=signature\_row, column=1).alignment = Alignment(horizontal='right')

# Сохраняем временный файл

temp\_dir = templates\_dir / "temp\_exports"

temp\_dir.mkdir(exist\_ok=True)

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

filename = f"выдачи\_книг\_{timestamp}.xlsx"

file\_path = temp\_dir / filename

wb.save(file\_path)

print(f"✅ Excel файл создан: {file\_path}")

print(f"📁 Размер файла: {file\_path.stat().st\_size} байт")

# Проверяем что файл создался

if not file\_path.exists():

raise Exception("Файл не был создан")

# Возвращаем файл для скачивания

return FileResponse(

path=file\_path,

filename=filename,

media\_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'

)

except Exception as e:

print(f"❌ Ошибка экспорта в Excel: {str(e)}")

import traceback

print(f"🔍 Детали ошибки: {traceback.format\_exc()}")

raise HTTPException(500, f"Ошибка экспорта в Excel: {str(e)}")

Листинг 18 – Экспорт Excel файла(app.js)

async exportToExcel() {  
 try {  
 this.showNotification('Подготовка Excel файла...', 'info');  
 const response = await fetch('/api/issues/export-excel');  
  
 if (!response.ok) throw new Error(`Ошибка создания Excel файла: ${response.status}`);  
  
 await this.downloadFile(response, 'выдачи\_книг.xlsx');  
 this.showNotification('Excel файл успешно скачан', 'success');  
 } catch (error) {  
 this.showNotification('Ошибка экспорта в Excel: ' + error.message, 'error');  
 }  
}

- создание документа .pdf

Листинг 19 – Создание PDF в отдельном модуле(create\_rules\_pdf.py)

def create\_rules\_pdf():

# Создаем папку static если её нет

os.makedirs('static', exist\_ok=True)

# Регистрируем шрифт с поддержкой кириллицы

try:

# Для Windows

font\_path = "C:/Windows/Fonts/arial.ttf"

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

pdfmetrics.registerFont(TTFont('Arial', font\_path))

pdfmetrics.registerFont(TTFont('Arial-Bold', "C:/Windows/Fonts/arialbd.ttf"))

pdfmetrics.registerFont(TTFont('Arial-Italic', "C:/Windows/Fonts/ariali.ttf"))

main\_font = 'Arial'

else:

# Для Linux или если Arial не найден

main\_font = 'Helvetica'

except:

main\_font = 'Helvetica'

# Создаем PDF

c = canvas.Canvas('static/rules.pdf', pagesize=A4)

width, height = A4

# Добавляем водяной знак "LibTool" на задний план - менее яркий и на всю страницу

c.setFillColor(lightgrey)

c.setFillAlpha(0.45) # Делаем очень прозрачным

c.setFont("Helvetica", 60)

c.rotate(45)

# Увеличиваем покрытие для всей страницы

for i in range(-3, 5):

for j in range(-3, 5):

c.drawString(i \* 250, j \* 180, "LibTool")

c.rotate(-45)

c.setFillAlpha(1.0) # Возвращаем нормальную прозрачность для основного текста

c.setFillColor(black)

# Основной текст

c.setFillColor(black)

# Заголовок

c.setFont(f"{main\_font}-Bold", 16)

c.drawString(50, height - 80, "ПРАВИЛА БИБЛИОТЕКИ")

# Основной текст

c.setFont(main\_font, 11)

rules = [

"1. ОБЩИЕ ПОЛОЖЕНИЯ",

"1.1. Библиотека осуществляет выдачу книг читателям на условиях настоящих Правил.",

"1.2. Читателем библиотеки может стать любой гражданин, достигший 14 лет.",

"1.3. Регистрация читателей осуществляется при предъявлении документа, удостоверяющего личность.",

"",

"2. ПРАВА И ОБЯЗАННОСТИ ЧИТАТЕЛЕЙ",

"2.1. Читатель имеет право:",

" - Бесплатно пользоваться фондами библиотеки",

" - Получать книги во временное пользование",

" - Получать консультации по работе с каталогами",

" - Продлевать срок пользования книгами",

"2.2. Читатель обязан:",

" - Бережно относиться к книгам и другим материалам",

" - Возвращать книги в установленные сроки",

" - Соблюдать тишину в читальном зале",

" - Соблюдать правила внутреннего распорядка",

"",

"3. ПОРЯДОК ВЫДАЧИ И ВОЗВРАТА КНИГ",

"3.1. Максимальный срок пользования книгой - 14 дней.",

"3.2. Возможно продление срока при отсутствии очереди на данную книгу.",

"3.3. За утерю или порчу книги читатель возмещает ущерб в 3-х кратном размере.",

"3.4. При получении книги читатель обязан проверить её целостность.",

"",

"4. РЕЖИМ РАБОТЫ",

"4.1. Библиотека работает с понедельника по пятницу с 9:00 до 18:00.",

"4.2. Обеденный перерыв: с 13:00 до 14:00.",

"4.3. Выходные дни: суббота, воскресенье.",

"",

"5. САНКЦИИ ЗА НАРУШЕНИЕ ПРАВИЛ",

"5.1. За нарушение сроков возврата - штраф 10 руб./день за каждую книгу.",

"5.2. За утерю читательского билета - штраф 50 руб.",

"5.3. Систематические нарушения могут привести к отлучению от библиотеки.",

"",

"6. ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ",

"6.1. Настоящие Правила вступают в силу с момента их утверждения.",

"6.2. Все спорные вопросы решаются администрацией библиотеки.",

]

y\_position = height - 110

line\_height = 15

for rule in rules:

if y\_position < 50: # Если текст не помещается, создаем новую страницу

c.showPage()

# Добавляем водяной знак на новую страницу

c.setFillColor(lightgrey)

c.setFillAlpha(0.25)

c.setFont("Helvetica", 60)

c.rotate(45)

for i in range(-3, 5):

for j in range(-3, 5):

c.drawString(i \* 250, j \* 180, "LibTool")

c.rotate(-45)

c.setFillAlpha(1.0)

c.setFillColor(black)

c.setFont(main\_font, 11)

y\_position = height - 50

if rule.startswith(('1.', '2.', '3.', '4.', '5.', '6.')) and not rule.startswith(

(' -', '1.1', '2.1', '3.1', '4.1', '5.1', '6.1', '1.2', '2.2', '3.2', '4.2', '5.2', '6.2', '1.3', '3.3', '3.4', '1.1', '4.3', '5.3')):

c.setFont(f"{main\_font}-Bold", 11)

else:

c.setFont(main\_font, 11)

c.drawString(50, y\_position, rule)

y\_position -= line\_height

# Подпись в конце

y\_position -= 20

c.setFont(f"{main\_font}-Bold", 11)

c.drawString(50, y\_position, "Директор библиотеки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

y\_position -= 15

c.drawString(50, y\_position, "Дата введения в действие: «\_\_\_» \_\_\_\_\_\_\_\_ 20\_\_\_ г.")

y\_position -= 30

c.setFont(f"{main\_font}-Italic", 9)

c.drawString(50, y\_position, "Документ создан в системе управления библиотекой LibTool")

c.save()

print("PDF файл с правилами создан: static/rules.pdf")

Листинг 20 – Экспорт PDF файла(main.py)

@app.get("/api/rules/download")

async def download\_rules():

"""Скачать правила библиотеки в формате PDF"""

try:

# Путь к PDF файлу с правилами

rules\_path = static\_dir / "rules.pdf"

if not rules\_path.exists():

rules\_path = templates\_dir / "rules.pdf"

if not rules\_path.exists():

raise HTTPException(404, "Файл правил не найден")

return FileResponse(

path=rules\_path,

filename="Правила\_библиотеки.pdf",

media\_type='application/pdf'

)

except Exception as e:

raise HTTPException(500, f"Ошибка загрузки правил: {str(e)}")

Листинг 21 – Экспорт PDF файла(app.js)

async downloadRules() {  
 try {  
 this.showNotification('Подготовка файла правил...', 'info');  
 const response = await fetch('/api/rules/download');  
  
 if (!response.ok) throw new Error(`Ошибка загрузки правил: ${response.status}`);  
  
 await this.downloadFile(response, 'Правила\_библиотеки.pdf');  
 this.showNotification('Правила библиотеки успешно скачаны', 'success');  
 } catch (error) {  
 this.showNotification('Ошибка скачивания правил: ' + error.message, 'error');  
 }  
}

Листинг 22 – Общий метод скачивания файлов (app.js)

async downloadFile(response, defaultFilename) {  
 const blob = await response.blob();  
 const url = window.URL.createObjectURL(blob);

const a = document.createElement('a');  
 a.href = url;  
  
 *// Получаем имя файла из заголовков ответа* const contentDisposition = response.headers.get('content-disposition');  
 let filename = defaultFilename;  
  
 if (contentDisposition) {  
 const filenameMatch = contentDisposition.match(/filename="(.+)"/);  
 if (filenameMatch) filename = filenameMatch[1];  
 }  
  
 a.download = filename;  
 document.body.appendChild(a);  
 a.click();  
 document.body.removeChild(a);  
 window.URL.revokeObjectURL(url);  
}

4.4. Реализация импорта данных из файла в базу данных

4.5. Загрузка приложения в репозиторий GitHub

Загрузку в репозиторий я реализовывал прямо из PyCharm. Для начала я привязал свой аккаунт GitHub к PyCharm (Settings – Version Control – GitHub). После привязки аккаунта мы открываем в верхней доп. Панели Git – GitHub – Share Project on GitHub. В открытом окне придумываем название для репозитория и при желании добавляем описание. Важно не забыть выбрать аккаунт, с которого мы делимся проектом (тот аккаунт, где создается репозиторий с проектом).

Загрузка через GitBush. Открываем в проводнике папку с проектом и жмем ПКМ – выбираем GitBush.

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