# Содержание

[Содержание 2](#_Toc212119422)

[Введение 3](#_Toc212119423)

[1. Разработка технического задания 3](#_Toc212119424)

[2. Проектирование интерфейса приложения 3](#_Toc212119425)

[2.1 Макет интерфейса в figma 3](#_Toc212119426)

[3. Проектирование и создание базы данных 5](#_Toc212119427)

[3.1. Разработка структуры БД 6](#_Toc212119428)

[3.2. Создание модели базы данных 6](#_Toc212119429)

[4. Разработка программных модулей 7](#_Toc212119430)

[4.1. Создание сервера 7](#_Toc212119431)

[4.2. Создание клиента 10](#_Toc212119432)

[4.3. Реализация модулей вывода на печать 13](#_Toc212119433)

[4.4. Реализация импорта данных из файла в базу данных 21](#_Toc212119434)

[4.5. Загрузка приложения в репозиторий GitHub 21](#_Toc212119435)

[5. Тестирование и отладка 22](#_Toc212119436)

[5.1. Проведение модульного тестирования 22](#_Toc212119437)

[5.2. Отладка программы 22](#_Toc212119438)

[6. Разработка мобильной версии 22](#_Toc212119439)

[7. Разработка технической документации 22](#_Toc212119440)

[7.1. Руководство пользователя 22](#_Toc212119441)

[7.2. Руководство программиста 22](#_Toc212119442)

[7.3. Руководство системного программиста 22](#_Toc212119443)

[7.4. Методика тестирования и испытания программы 22](#_Toc212119444)

[Заключение 22](#_Toc212119445)

[Список литературы 22](#_Toc212119446)

# Введение

Система тестирования знаний на Python с графическим интерфейсом Tkinter для проведения интерактивных тестов и анализа результатов. Программа поддерживает загрузку тестов из текстовых файлов, отслеживание времени прохождения и детальную статистику ответов. Система включает базу данных PostgreSQL для хранения всех результатов тестирования с возможностью просмотра истории.

# 1. Разработка технического задания

# 2. Проектирование интерфейса приложения

## 2.1 Макет интерфейса в figma

Figma — это облачный редактор для создания интерфейсов, прототипов и дизайн-систем. Работает прямо в браузере.

**Плюсы:**

* **Реальная совместная работа** — несколько человек могут работать над макетом одновременно
* **Кроссплатформенность** — работает на Windows, Mac и даже Linux через браузер
* **Авто-сохранение и версионность** — ничего не теряется, можно откатиться к любой предыдущей версии

**Минусы:**

* **Требует интернет**— без сети функциональность ограничена
* **Производительность** — может подтормаживать с очень сложными макетами
* **Подписка** — продвинутые функции (например, Dev Mode) платные

Макеты:



Рис.1 «Макет главного меню»

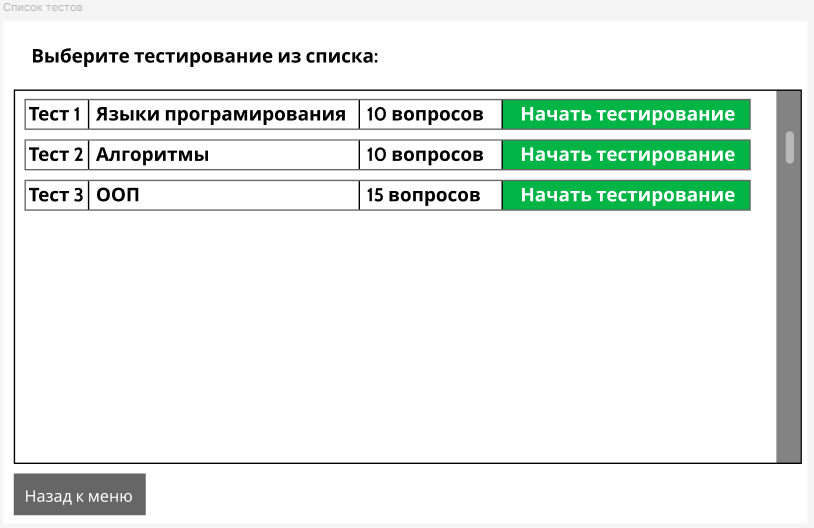


Рис.2 «Макет списка тестов»

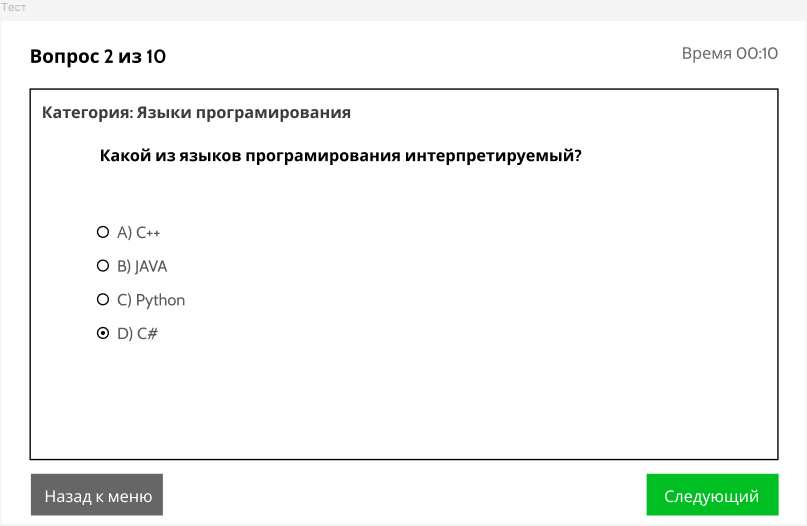


Рис.3 «Макет окна решения теста»

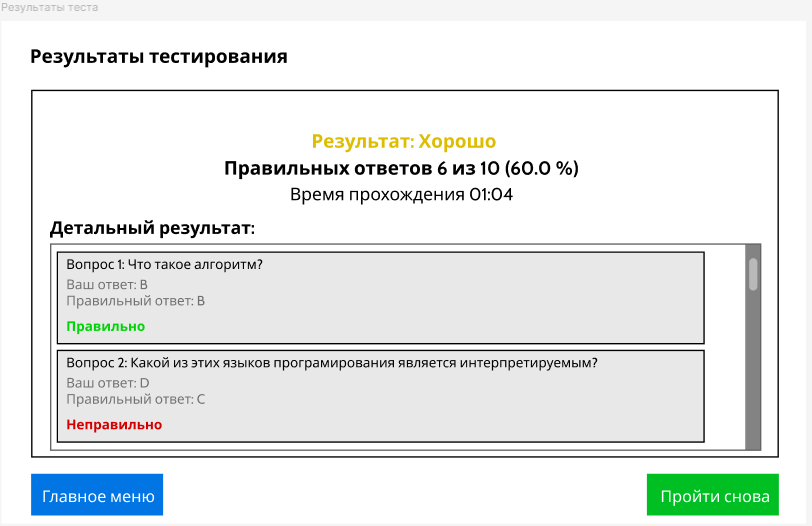


Рис.4 «Макет окна результатов теста»

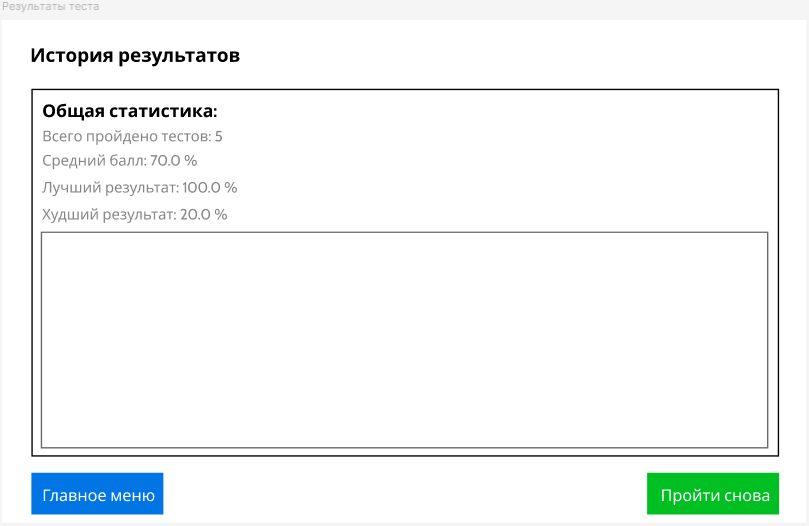


Рис.5 «Макет окна истории результатов»

# 3. Проектирование и создание базы данных

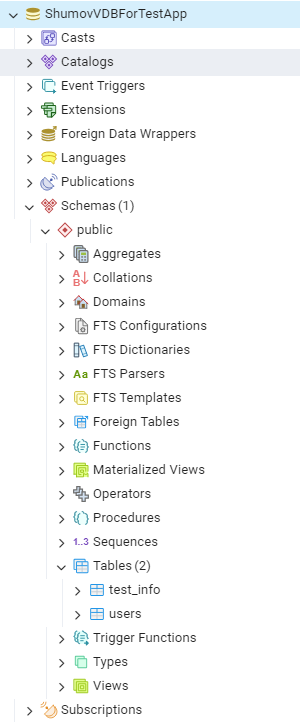


Рис.6 «База данных»

## 3.1. Разработка структуры БД

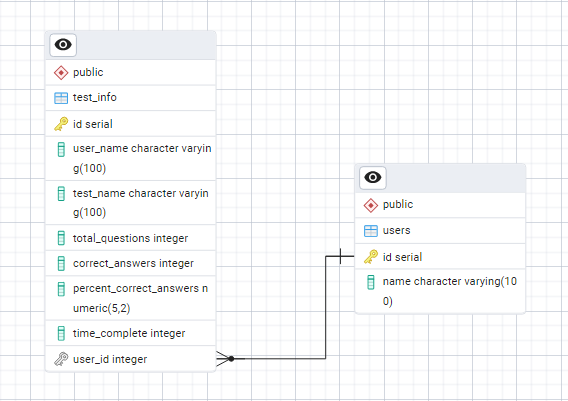


Рис.7 «Структура базы данных»

## 3.2. Создание модели базы данных

- модуль классов

Листинг 1 - модели классов pydantic

class TestResultItem(BaseModel):  
 id: int  
 user\_name: str  
 test\_name: str  
 total\_questions: int  
 correct\_answers: int  
 percent\_correct\_answers: float  
 time\_complete: int  
 user\_id: int

class TestResult(BaseModel):  
 user\_name: str  
 test\_name: str  
 total\_questions: int  
 correct\_answers: int  
 percent\_correct\_answers: float  
 time\_complete: int  
 user\_answers: Optional[List[int]] = None  
  
  
class TestResultResponse(BaseModel):  
 status: str  
 message: str  
 result\_id: Optional[int] = None  
 user\_id: Optional[int] = None  
  
  
class TestResultsShow(BaseModel):  
 status: str  
 message: str  
 results: List[TestResultItem]  
  
  
class DeleteRequest(BaseModel):  
 ids: List[int]  
  
  
class DeleteResponse(BaseModel):  
 status: str  
 message: str  
 deleted\_count: int

# 4. Разработка программных модулей

## 4.1. Создание сервера

- модуль подключения к базе данных

Листинг 2 – Подключение к базе данных

def get\_db\_connection():  
 try:  
 conn = psycopg2.connect(  
 dbname='ShumovVDBForTestApp',  
 user='postgres',  
 password='1234',  
 host='localhost',  
 port='5433'  
 )  
 return conn  
 except psycopg2.Error as e:  
 raise HTTPException(status\_code=500, detail=f'Database connection error: {str(e)}')

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

Листинг 3 – сохраненение результатов теста в базу данных

@app.post("/api/save\_test\_result", response\_model=TestResultResponse)  
async def save\_test\_result(result\_data: TestResult):  
 conn = None  
 try:  
 conn = get\_db\_connection()  
 cursor = conn.cursor()  
  
 *# Находим или создаём пользователя* cursor.execute('SELECT id FROM users WHERE name = %s', (result\_data.user\_name,))  
 user\_result = cursor.fetchone()  
  
 if user\_result:  
 user\_id = user\_result[0]  
 else:  
 *# Создаем нового пользователя* cursor.execute('INSERT INTO users (name) VALUES (%s) RETURNING id', (result\_data.user\_name,))  
 user\_id = cursor.fetchone()[0]  
  
 *# Сохраняем результаты теста* cursor.execute('''INSERT INTO test\_info   
 (user\_name, test\_name, total\_questions, correct\_answers, percent\_correct\_answers, time\_complete, user\_id)   
 VALUES (%s, %s, %s, %s, %s, %s, %s) RETURNING id''',  
 (result\_data.user\_name, result\_data.test\_name, result\_data.total\_questions,  
 result\_data.correct\_answers, result\_data.percent\_correct\_answers,  
 result\_data.time\_complete, user\_id))  
  
 result\_id = cursor.fetchone()[0]  
 conn.commit()  
  
 return TestResultResponse(  
 status='success',  
 message="Results saved successfully",  
 result\_id=result\_id,  
 user\_id=user\_id  
 )  
  
 except psycopg2.Error as e:  
 if conn:  
 conn.rollback()  
 raise HTTPException(status\_code=500, detail=f"Database error: {str(e)}")  
 except Exception as e:  
 if conn:  
 conn.rollback()  
 raise HTTPException(status\_code=500, detail=f"Unexpected error: {str(e)}")  
 finally:  
 if conn:  
 conn.close()

Листинг 4 – вывод результатов из базы данных

@app.get('/api/show\_tests\_results', response\_model=TestResultsShow)  
async def show\_tests\_results():  
 conn = None  
 try:  
 conn = get\_db\_connection()  
 cursor = conn.cursor(cursor\_factory=RealDictCursor)  
  
 cursor.execute('SELECT \* FROM test\_info ORDER BY id ASC')  
 users\_data = cursor.fetchall()  
  
 results = []  
 for row in users\_data:  
 results.append(TestResultItem(  
 id=row['id'],  
 user\_name=row['user\_name'],  
 test\_name=row['test\_name'],  
 total\_questions=row['total\_questions'],  
 correct\_answers=row['correct\_answers'],  
 percent\_correct\_answers=float(row['percent\_correct\_answers']),  
 time\_complete=row['time\_complete'],  
 user\_id=row['user\_id']  
 ))  
  
 return TestResultsShow(  
 status="success",  
 message="Results retrieved successfully",  
 results=results  
 )  
 except psycopg2.Error as e:  
 raise HTTPException(status\_code=500, detail=f"Database error: {str(e)}")  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Unexpected error: {str(e)}")  
 finally:  
 if conn:  
 conn.close()

Листинг 5 – удаление записи из базы данных  
  
@app.delete('/api/delete\_results', response\_model=DeleteResponse)  
async def delete\_results(delete\_request: DeleteRequest):  
 conn = None  
 try:  
 conn = get\_db\_connection()  
 cursor = conn.cursor()  
  
 if not delete\_request.ids:  
 return DeleteResponse(  
 status="error",  
 message="No IDs provided",  
 deleted\_count=0  
 )  
  
 *# Создаем плейсхолдеры для IN условия* placeholders = ','.join(['%s'] \* len(delete\_request.ids))  
  
 *# Удаляем записи* cursor.execute(f'DELETE FROM test\_info WHERE id IN ({placeholders})', delete\_request.ids)  
 deleted\_count = cursor.rowcount  
  
 conn.commit()  
  
 return DeleteResponse(  
 status="success",  
 message=f"Deleted {deleted\_count} records",  
 deleted\_count=deleted\_count  
 )  
  
 except psycopg2.Error as e:  
 if conn:  
 conn.rollback()  
 raise HTTPException(status\_code=500, detail=f"Database error: {str(e)}")  
 except Exception as e:  
 if conn:  
 conn.rollback()  
 raise HTTPException(status\_code=500, detail=f"Unexpected error: {str(e)}")  
 finally:  
 if conn:  
 conn.close()

Листинг 6 – удаление всех записей из базы данных  
  
@app.delete("/api/clear\_all\_results", response\_model=DeleteResponse)  
async def clear\_all\_results():  
 conn = None  
 try:  
 conn = get\_db\_connection()  
 cursor = conn.cursor()  
  
 *# Удаляем все записи* cursor.execute('DELETE FROM test\_info')  
 deleted\_count = cursor.rowcount  
  
 cursor.execute('ALTER SEQUENCE test\_info\_id\_seq RESTART WITH 1')  
  
 conn.commit()  
  
 return DeleteResponse(  
 status="success",  
 message=f"All records deleted ({deleted\_count} total)",  
 deleted\_count=deleted\_count  
 )  
  
 except psycopg2.Error as e:  
 if conn:  
 conn.rollback()  
 raise HTTPException(status\_code=500, detail=f"Database error: {str(e)}")  
 except Exception as e:  
 if conn:  
 conn.rollback()  
 raise HTTPException(status\_code=500, detail=f"Unexpected error: {str(e)}")  
 finally:  
 if conn:  
 conn.close()

## 4.2. Создание клиента

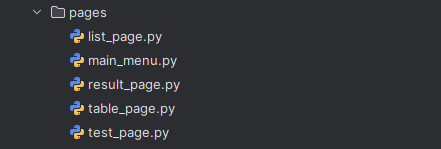


Рис.8 «Структура пользовательского интерфейса»

Для реализации интерфейса использована библиотека tkinter.

При запуске приложения открывается main\_menu. Из main\_menu можно перейти в list\_page, table\_page. List\_page предтавляет собой список тестов. Выбрав тест и нажав кнопку «Начать тест» вы перейдете на test\_page. Когда вы завершите тест, вы перейдете на result\_page.

Каждая страница реализована классом с функцией create\_widgets(), которая заполняет окно виджетами.

Интерфейс программы:

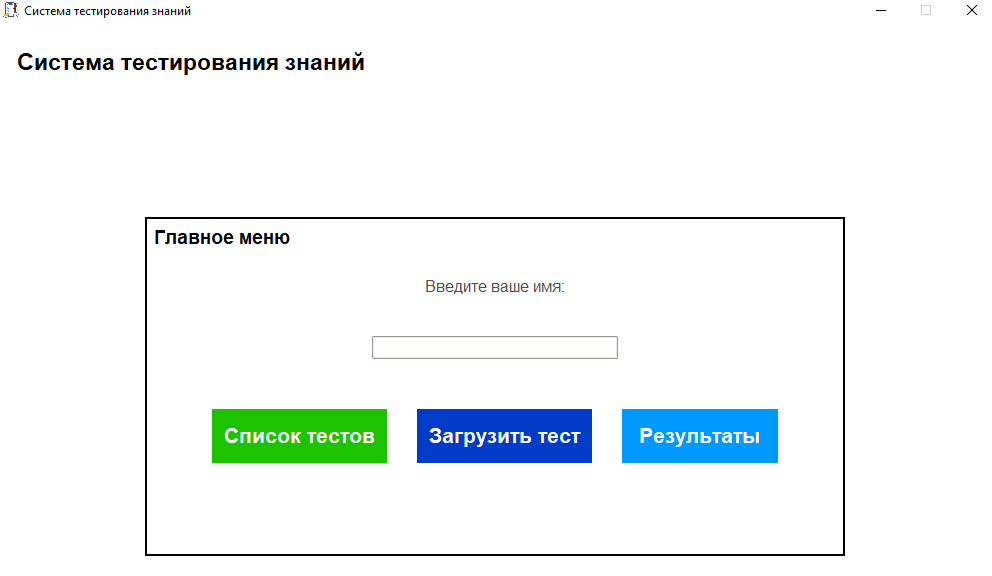


Рис.8 «Главное меню программы»

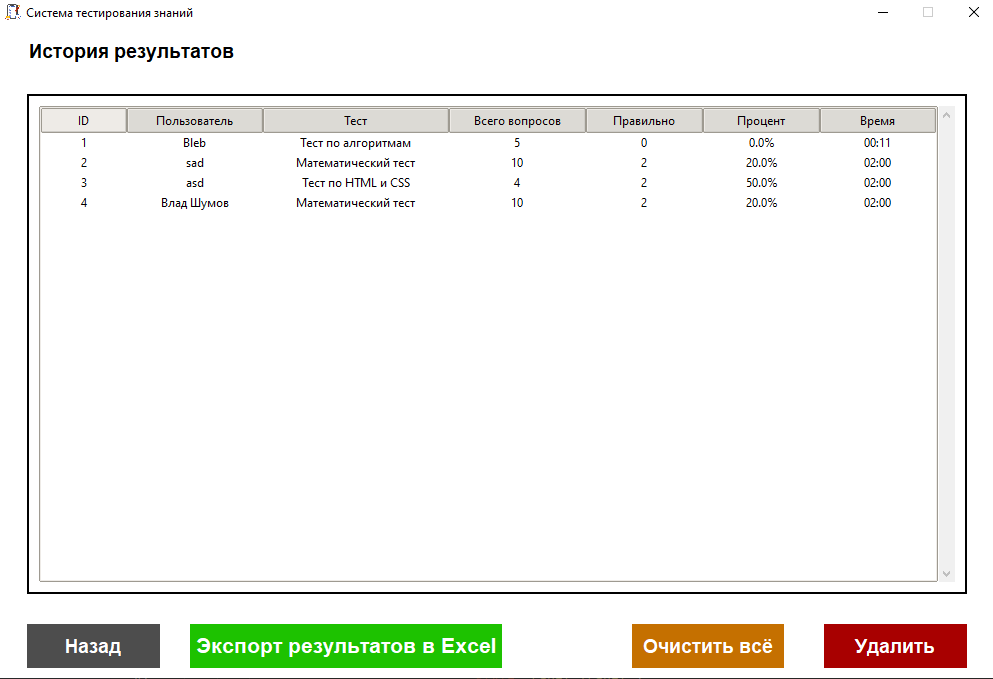


Рис.9 «Страница истории результатов»

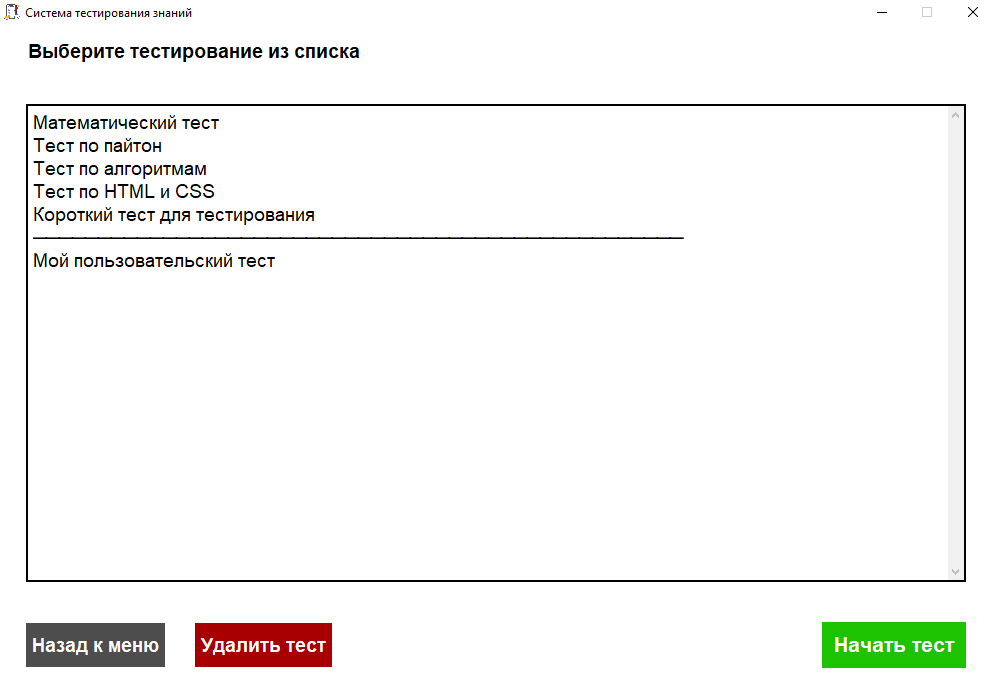


Рис.10 «Страница выбора теста»

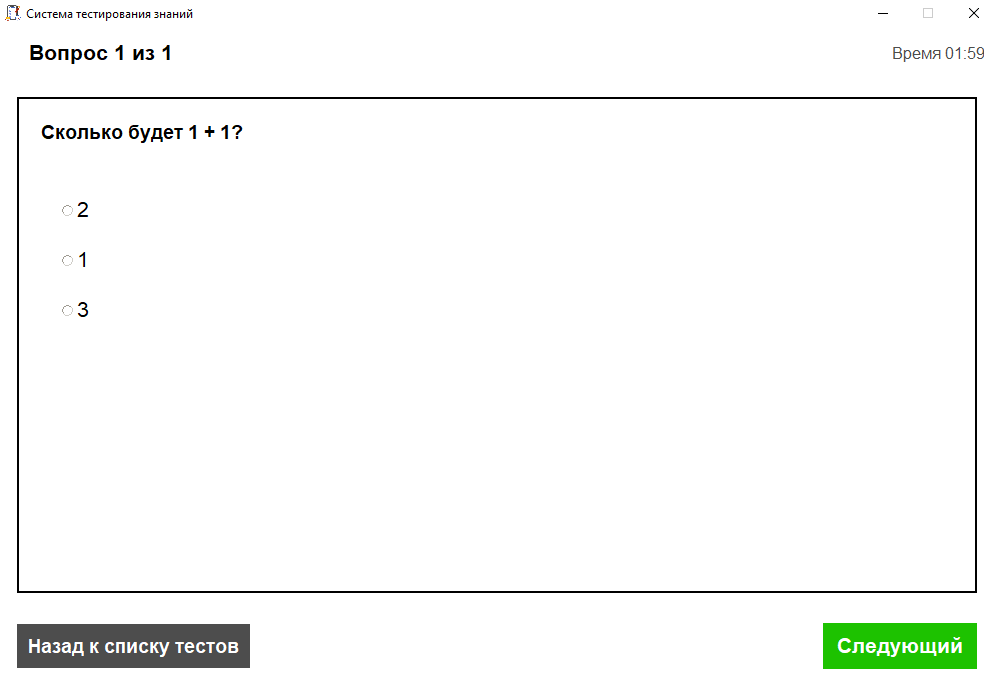


Рис.11 «Страница прохождения теста»

Страница прохождения теста при нажатии кнопки «Следующий» переходит к следующему вопросу, путём обновления данных в рамке.

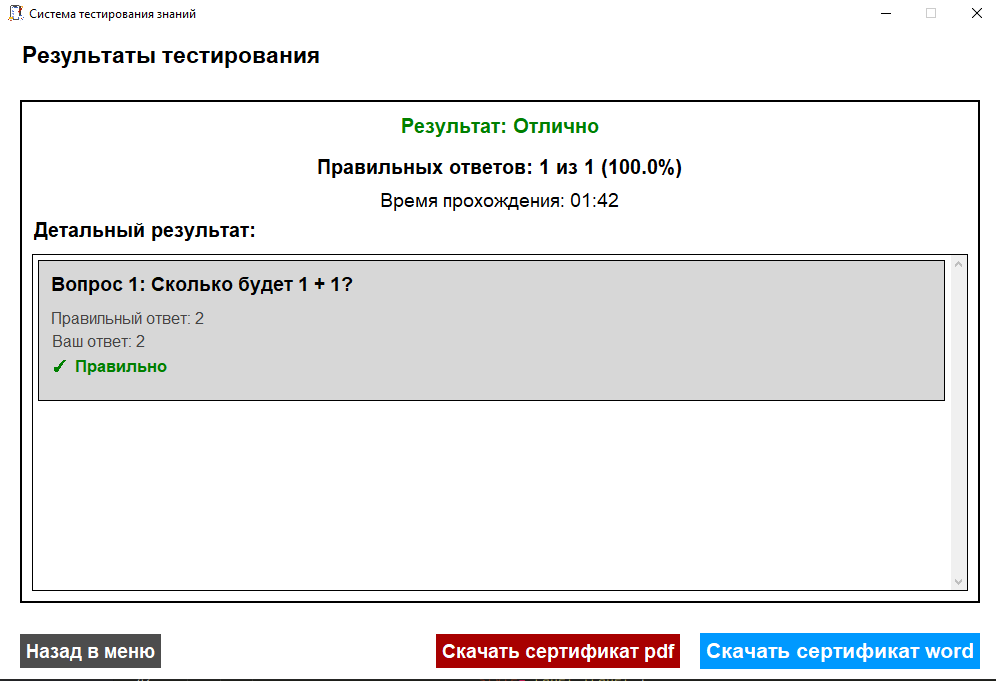


Рис.12 «Страница результатов тестирования»

## 4.3. Реализация модулей вывода на печать

- вывод в шаблон .docx

Листинг 7 – Экспорт сертификата в word

from docx import Document

def download\_certificate\_word(self):  
 try:  
 template\_path = "certificates/sample.docx"  
  
 doc = Document(template\_path)  
  
 certificate\_number = datetime.now().strftime("%Y%m%d%H%M")  
  
 replacements = {  
 "{{CERTIFICATE\_NUMBER}}": certificate\_number,  
 "{{USER\_NAME}}": self.user\_name or "Пользователь",  
 "{{TEST\_NAME}}": self.test\_name or "Тест",  
 "{{CORRECT\_ANSWERS}}": str(self.correct\_answers),  
 "{{TOTAL\_QUESTIONS}}": str(self.total\_questions),  
 "{{PERCENTAGE}}": f"{str(self.correct\_percent)}%",  
 "{{GRADE}}": self.grade,  
 "{{TIME}}": str(self.time\_complete),  
 "{{DATE}}": datetime.now().strftime('%d.%m.%Y %H:%M')  
 }  
  
 *# Замена текста* 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)  
  
 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)  
  
 *# Сохранение результатов* filename = f"certificate\_{self.test\_name}\_{certificate\_number}.docx"  
 filepath = os.path.join(self.certificates\_dir, filename)  
  
 doc.save(filepath)  
  
 messagebox.showinfo("Успех", f"Сертификат успешно сохранён:\n{filename}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить сертификат:\n{e}")

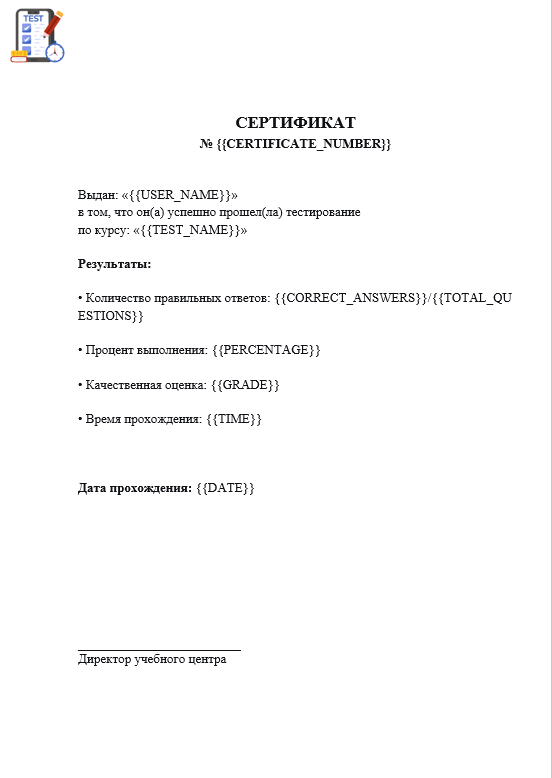


Рис.13 «Шаблон»

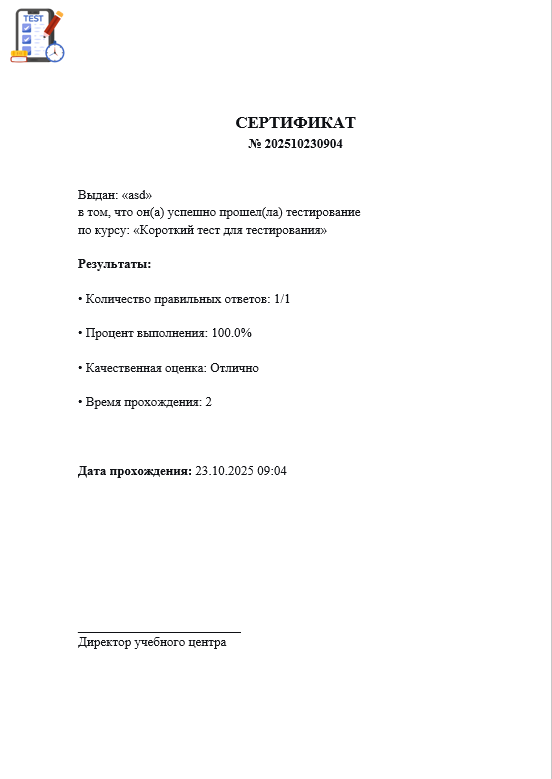


Рис.14 «Пример сертификата word»

- создание документа .xlsx

Листинг 8 – экспорт таблицы результатов в файл .xlsx

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

def export\_excel(self):  
 *"""Экспорт результатов в Excel"""* try:  
 all\_items = self.tree.get\_children()  
 if not all\_items:  
 messagebox.showwarning("Внимание", "Нет данных для экспорта")  
 return  
  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".xlsx",  
 filetypes=[("Excel files", "\*.xlsx"), ("All files", "\*.\*")],  
 title="Сохранить результат как"  
 )  
  
 if not file\_path:  
 return  
  
 wb = Workbook()  
 ws = wb.active  
 ws.title = "Результаты тестирований"  
  
 *# Стили для форматирования* header\_font = Font(bold=True, size=12, color="FFFFFF")  
 header\_fill = PatternFill(start\_color="366092", end\_color="366092", fill\_type="solid")  
 header\_alignment = Alignment(horizontal="center", vertical="center")  
 border = Border(  
 left=Side(style='thin'),  
 right=Side(style='thin'),  
 top=Side(style='thin'),  
 bottom=Side(style='thin')  
 )  
  
 *# Заголовки столбцов* headers = ['ID', 'Пользователь', 'Тест', 'Всего вопросов', 'Правильно', 'Процент', 'Время']  
 for col, header in enumerate(headers, 1):  
 cell = ws.cell(row=1, column=col, value=header)  
 cell.font = header\_font  
 cell.fill = header\_fill  
 cell.alignment = header\_alignment  
 cell.border = border  
  
 *# Заполняем данными* row\_num = 2  
 for item in all\_items:  
 item\_values = self.tree.item(item)['values']  
  
 *# Пропускаем строки с сообщениями об ошибках или отсутствии данных* if not item\_values:  
 continue  
  
 for col, value in enumerate(item\_values, 1):  
 cell = ws.cell(row=row\_num, column=col, value=value)  
 cell.border = border  
 if col in [1, 2]: *# Выравнивание по левому краю для текстовых полей* cell.alignment = Alignment(horizontal="left", vertical="center")  
 else:  
 cell.alignment = Alignment(horizontal="center", vertical="center")  
  
 row\_num += 1  
  
 *# Автоматическая ширина столбцов* for column in ws.columns:  
 max\_length = 0  
 column\_letter = column[0].column\_letter  
 for cell in column:  
 try:  
 if len(str(cell.value)) > max\_length:  
 max\_length = len(str(cell.value))  
 except:  
 pass  
 adjusted\_width = (max\_length + 2)  
 ws.column\_dimensions[column\_letter].width = adjusted\_width  
  
 *# Добавляем информацию о дате экспорта* ws.cell(row=row\_num + 1, column=1, value="Дата экспорта:").font = Font(bold=True)  
 ws.cell(row=row\_num + 1, column=2, value=datetime.now().strftime("%d.%m.%Y %H:%M"))  
  
 ws.cell(row=row\_num + 2, column=1, value="Всего записей:").font = Font(bold=True)  
 ws.cell(row=row\_num + 2, column=2, value=row\_num - 2)  
  
 *# Сохраняем файл* wb.save(file\_path)  
  
 messagebox.showinfo("Успех", f"Данные успешно экспортированы в файл:\n{file\_path}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Произошла ошибка при экспорте в Excel:\n{str(e)}")

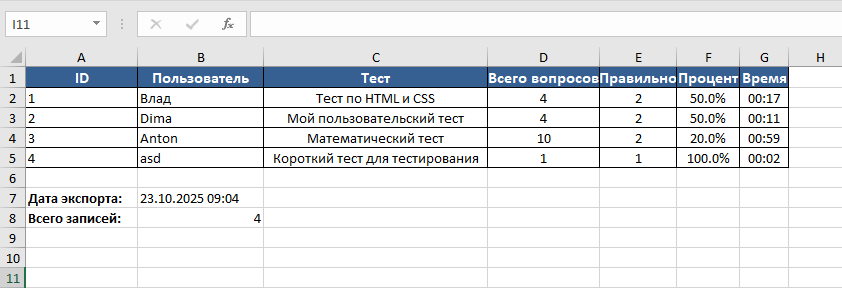


Рис.15 «Пример таблицы excel»

- создание документа .pdf

Листинг 9 – Экспорт сертификата в pdf

from reportlab.lib.pagesizes import A4  
from reportlab.pdfgen import canvas  
from reportlab.lib.units import cm  
from reportlab.pdfbase import pdfmetrics  
from reportlab.pdfbase.ttfonts import TTFont

def download\_certificate\_pdf(self):  
 try:  
 timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")  
 filename = f"certificate\_{self.user\_name}\_{timestamp}.pdf"  
 filepath = os.path.join(self.certificates\_dir, filename)  
  
 *# Стандартные пути к шрифтам в Windows* windows\_fonts\_paths = [  
 "C:/Windows/Fonts/arial.ttf",  
 "C:/Windows/Fonts/ARIAL.TTF",  
 "C:/Windows/Fonts/ariali.ttf", *# Arial Italic* "C:/WinNT/Fonts/arial.ttf", *# Для старых версий Windows* ]  
  
 windows\_bold\_paths = [  
 "C:/Windows/Fonts/arialbd.ttf",  
 "C:/Windows/Fonts/ARIALBD.TTF",  
 "C:/Windows/Fonts/arialbi.ttf", *# Arial Bold Italic* "C:/WinNT/Fonts/arialbd.ttf",  
 ]  
  
 *# Поиск обычного шрифта Arial* font\_path = None  
 for path in windows\_fonts\_paths:  
 if os.path.exists(path):  
 font\_path = path  
 break  
  
 *# Поиск жирного шрифта Arial* font\_bold\_path = None  
 for path in windows\_bold\_paths:  
 if os.path.exists(path):  
 font\_bold\_path = path  
 break  
  
 *# Регистрируем найденные шрифты* font\_name = 'Helvetica' *# значение по умолчанию* font\_bold = 'Helvetica-Bold'  
  
 if font\_path and font\_bold\_path:  
 try:  
 pdfmetrics.registerFont(TTFont('Arial', font\_path))  
 pdfmetrics.registerFont(TTFont('Arial-Bold', font\_bold\_path))  
 font\_name = 'Arial'  
 font\_bold = 'Arial-Bold'  
 print(f"Используются шрифты: {font\_path}, {font\_bold\_path}")  
 except Exception as e:  
 print(f"Ошибка регистрации шрифтов: {e}")  
 *# Пробуем зарегистрировать только обычный шрифт* try:  
 pdfmetrics.registerFont(TTFont('Arial', font\_path))  
 font\_name = 'Arial'  
 font\_bold = 'Helvetica-Bold'  
 except:  
 pass  
 else:  
 print("Шрифты Arial не найдены, используем стандартные Helvetica")  
  
 *# Альтернативные варианты если Arial не найден* if font\_name == 'Helvetica':  
 *# Пробуем найти другие кириллические шрифты* alternative\_fonts = [  
 "C:/Windows/Fonts/tahoma.ttf",  
 "C:/Windows/Fonts/tahomabd.ttf",  
 "C:/Windows/Fonts/verdana.ttf",  
 "C:/Windows/Fonts/verdanab.ttf",  
 "C:/Windows/Fonts/times.ttf",  
 "C:/Windows/Fonts/timesbd.ttf",  
 ]  
  
 for i in range(0, len(alternative\_fonts), 2):  
 if i + 1 < len(alternative\_fonts) and os.path.exists(alternative\_fonts[i]):  
 try:  
 regular\_path = alternative\_fonts[i]  
 bold\_path = alternative\_fonts[i + 1] if os.path.exists(alternative\_fonts[i + 1]) else \  
 alternative\_fonts[i]  
  
 font\_family = os.path.basename(regular\_path).split('.')[0].capitalize()  
 pdfmetrics.registerFont(TTFont(font\_family, regular\_path))  
 pdfmetrics.registerFont(TTFont(font\_family + '-Bold', bold\_path))  
  
 font\_name = font\_family  
 font\_bold = font\_family + '-Bold'  
 print(f"Используются альтернативные шрифты: {regular\_path}, {bold\_path}")  
 break  
 except Exception as e:  
 print(f"Ошибка регистрации альтернативного шрифта: {e}")  
 continue  
  
 c = canvas.Canvas(filepath, pagesize=A4)  
 width, height = A4  
  
 *# Фон* c.setFillColorRGB(0.98, 0.98, 0.98)  
 c.rect(0, 0, width, height, fill=1)  
 c.setFillColorRGB(0, 0, 0)  
  
 *# Рамка* c.setStrokeColorRGB(0.3, 0.5, 0.8)  
 c.setLineWidth(2)  
 c.rect(1.5 \* cm, 1.5 \* cm, width - 3 \* cm, height - 3 \* cm)  
  
 *# Заголовок* c.setFont(font\_bold, 26)  
 c.setFillColorRGB(0.2, 0.4, 0.8)  
 c.drawCentredString(width / 2, height - 4 \* cm, "СЕРТИФИКАТ")  
  
 *# Основной текст* c.setFont(font\_name, 14)  
 c.setFillColorRGB(0, 0, 0)  
 c.drawCentredString(width / 2, height - 7 \* cm, "Настоящим удостоверяется, что")  
  
 *# Имя пользователя* c.setFont(font\_bold, 16)  
 c.setFillColorRGB(0.8, 0.2, 0.2)  
 c.drawCentredString(width / 2, height - 9 \* cm, self.user\_name or "Пользователь")  
  
 c.setFont(font\_name, 14)  
 c.setFillColorRGB(0, 0, 0)  
 c.drawCentredString(width / 2, height - 11 \* cm, "успешно прошел(ла) тестирование")  
 c.drawCentredString(width / 2, height - 12.5 \* cm, "по курсу:")  
  
 *# Название теста* c.setFont(font\_bold, 15)  
 c.setFillColorRGB(0.2, 0.4, 0.8)  
 test\_name\_text = f"«{self.test\_name}»" if self.test\_name else "«Неизвестный тест»"  
 c.drawCentredString(width / 2, height - 14.5 \* cm, test\_name\_text)  
  
 *# Результаты* results\_y = height - 18 \* cm  
  
 c.setFont(font\_bold, 14)  
 c.drawCentredString(width / 2, results\_y, "Результаты тестирования")  
  
 c.setFont(font\_name, 12)  
 c.drawString(4 \* cm, results\_y - 1.5 \* cm,  
 f"Правильных ответов: {self.correct\_answers} из {self.total\_questions}")  
 c.drawString(4 \* cm, results\_y - 2.5 \* cm, f"Процент выполнения: {self.correct\_percent:.1f}%")  
 c.drawString(4 \* cm, results\_y - 3.5 \* cm, f"Оценка: {self.grade}")  
 c.drawString(4 \* cm, results\_y - 4.5 \* cm, f"Время выполнения: {self.format\_time()}")  
  
 *# Дата* c.drawString(4 \* cm, results\_y - 6 \* cm, f"Дата прохождения: {datetime.now().strftime('%d.%m.%Y %H:%M')}")  
  
 *# Подпись* signature\_y = 6 \* cm  
 c.setFont(font\_name, 12)  
 c.drawString(width - 8 \* cm, signature\_y + 1.5 \* cm, "Директор по обучению")  
 c.line(width - 8 \* cm, signature\_y + 1 \* cm, width - 3 \* cm, signature\_y + 1 \* cm)  
  
 c.save()  
  
 messagebox.showinfo("Успех", f"PDF сертификат сохранен:\n{filename}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось создать PDF сертификат:\n{str(e)}")

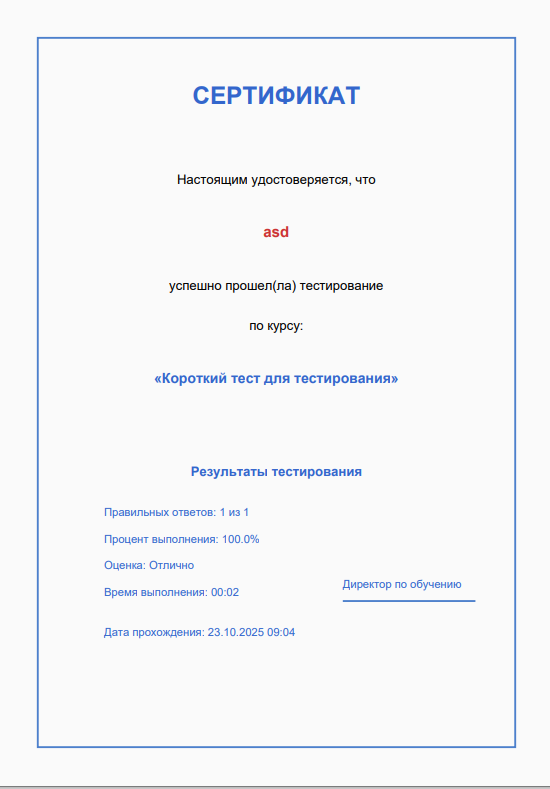


Рис.16 «Пример сертификата PDF»

## 4.4. Реализация импорта данных из файла в базу данных

## 4.5. Загрузка приложения в репозиторий GitHub

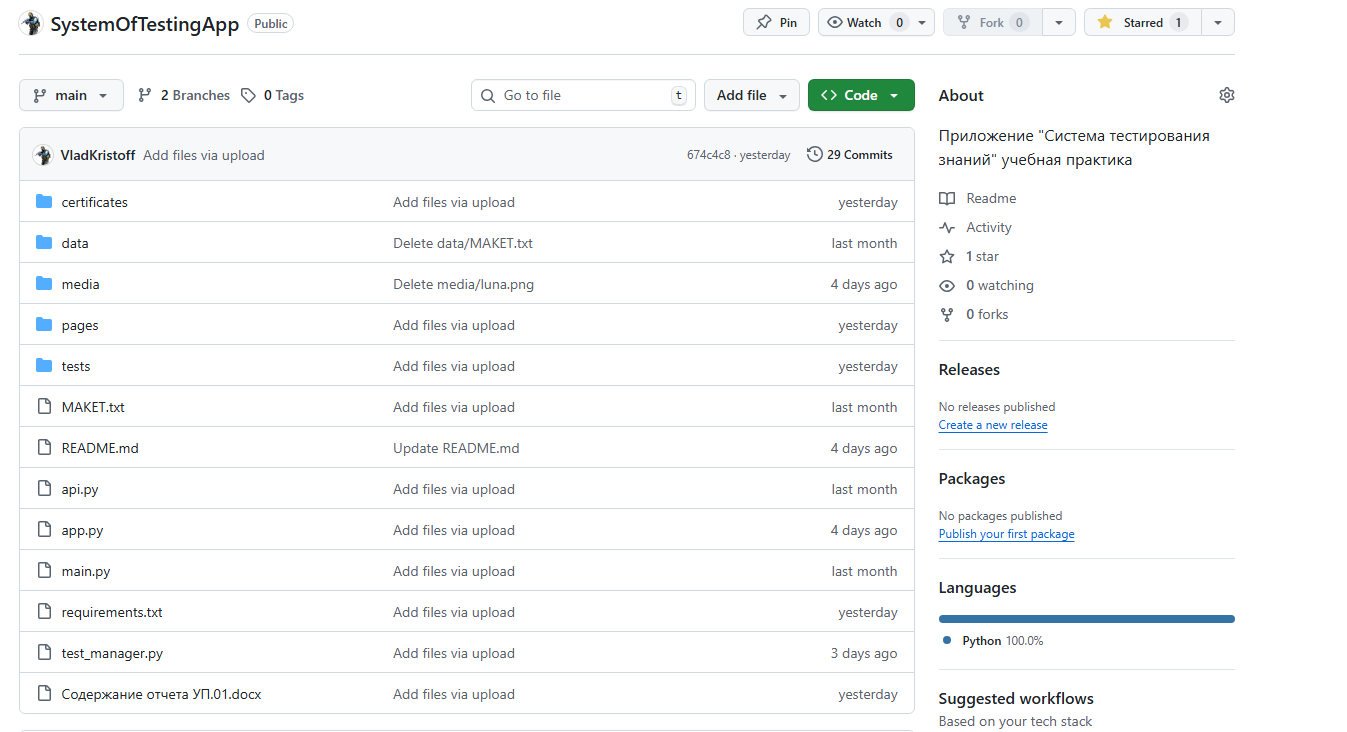


Рис.17 «Репозиторий на github»

# 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