**Отчёт по оценке проектов «Учебный календарь занятий»**

**Оглавление**

[Введение 2](#_Toc200379336)

[Методология и критерии оценки 2](#_Toc200379337)

[Инструкция по проведению оценки 3](#_Toc200379338)

[Детальная оценка проектов 3](#_Toc200379339)

[Как проводить проверку проектов (практически) 3](#_Toc200379340)

[1. Проверка PEP-8 (стиля кода) 3](#_Toc200379341)

[Через консоль: 3](#_Toc200379342)

[В PyCharm: 4](#_Toc200379343)

[2. Проверка pylint 4](#_Toc200379344)

[Через консоль: 4](#_Toc200379345)

[3. Проверка покрытия тестами 4](#_Toc200379346)

[Через pytest и coverage: 4](#_Toc200379347)

[4. Профилирование (измерение скорости) 4](#_Toc200379348)

[Простой способ через cProfile: 4](#_Toc200379349)

[Вывод в файл: 5](#_Toc200379350)

[Вставка прямо в код: 5](#_Toc200379351)

[Проект 1 – «Алябьев» 5](#_Toc200379352)

[Проект 2 – «Zelinskiy\_Dmitriy» 6](#_Toc200379353)

[Проект 3 – «Andrey Martynov» 7](#_Toc200379354)

[Проект 4 – «Kuzin\_Anton» 9](#_Toc200379355)

[Итоговые оценки проектов 11](#_Toc200379356)

[Заключение 11](#_Toc200379357)

[Шаблон для оценок проектов другими участниками: 12](#_Toc200379358)

[Алябьев Григорий Олегович: 12](#_Toc200379359)

[Зелинский Дмитрий Евгеньевич: 12](#_Toc200379360)

[Мартынов Андрей Сергеевич: 13](#_Toc200379361)

[Корнеев Антон Артурович 13](#_Toc200379362)

# Введение

В данной работе участникам было предложено разработать на Python программу «Учебный календарь занятий», которая **выводит расписание** для заданного периода. Базовое требование – при запуске программы в консоли должен отображаться готовый календарь или расписание (возможно с данными, интегрированными в программном коде, либо подгружаемыми из файла или базы данных). Любой работоспособный вариант решения этого минимального требования считается приемлемым. Дополнительные возможности (поиск по дате, экспорт в CSV/JSON, добавление/удаление событий, графический интерфейс, интеграция с LLM и т.д.) приветствуются и дают дополнительные баллы, но не являются обязательными.

Исходный код всех проектов расположен в репозитории <https://github.com/Sarend9/Class-Schedule>. В отчёте представлены методология оценки, инструкция, детальная переоценка четырёх проектов по актуальным файлам и результирующие таблицы.

# Методология и критерии оценки

Каждый проект проверяется по восьми критериям с указанными весами. Результат по каждому критерию оценивается от 0 до 10, а итоговый балл проекта рассчитывается как взвешенная сумма критериев.

**Общее минимальное требование:** при запуске проекта должно печататься расписание (перечень занятий) на заданный период (месяц, неделю и т.д.). Данные о расписании могут быть прописаны в коде или загружаться из текстового файла, CSV/JSON, SQLite БД. Всё, что выходит за рамки вывода базового расписания (например, интерактивность, LLM-интеграция, GUI, экспорт, поиск) считается дополнительным функционалом и учитывается в соответствующих критериях.

**Критерии и веса оценки:**

* **Функциональность (×2):** наличие базового вывода расписания. Бонусные функции (поиск по дате, добавление/удаление событий, экспорт, GUI, интеграция LLM и др.) повышают оценку.
* **Читаемость кода (×1.5):** понятные имена переменных и функций, логичная структура, наличие комментариев.
* **Стиль (×1):** соответствие PEP-8 (flake8) и рекомендации pylint (отсутствие предупреждений).
* **Модульность (×1):** логическое разделение по файлам и модулям, отсутствие чрезмерно длинных «монолитных» скриптов.
* **Документированность (×1.5):** наличие README, подробные docstring и поясняющие комментарии в коде.
* **Покрытие тестами (×0.5):** наличие тестов (pytest) и уровень покрытия кода тестами.
* **Обработка ошибок (×1):** наличие конструкций try/except, информативных сообщений об ошибках и проверок входных данных.
* **Производительность (×1.5):** эффективная реализация без явных узких мест (контроль через профилирование cProfile или тест на больших данных: <1.5 с на 1000 записей, <3 с на 5000 записей).

Каждый критерий оценивается по 10-балльной шкале.

# Инструкция по проведению оценки

1. **Подготовка окружения:** рекомендуется использовать IDE (PyCharm, VSCode) или командную строку Python. Убедитесь, что установлены нужные библиотеки (например, PySide6 для GUI-проектов, SQL-библиотеки и т.д.). Если в проекте есть requirements.txt, выполните pip install -r requirements.txt.
2. **Поиск точки входа:** определите главный скрипт проекта (например, main.py, run.py или аналог).
3. **Запуск проекта:** в терминале перейдите в каталог проекта и запустите основной скрипт (python main.py или название файла). Убедитесь, что программа выполняется без ошибок.
4. **Функциональное тестирование:** проверьте, что при запуске выводится корректное расписание (минимальное требование). Изучите дополнительный функционал: поддерживаются ли поиск по дате, добавление/удаление, графический интерфейс, LLM-интеграция и т.д. Выполняйте указанные в программе команды или запросы, чтобы убедиться в их работе.
5. **Качество кода:** проанализируйте код на читаемость и соответствие стилевым стандартам. Запустите flake8 и pylint (при возможности) и учтите найденные замечания. Опционально в PyCharm можно запустить Code → Analyze with Qodana для дополнительного статического анализа.
6. **Структура проекта:** обратите внимание на разбиение кода по файлам/модулям. Слишком длинные файлы (>200 строк без оправдания) снижают оценку по модульности.
7. **Документация:** найдите README и комментарии. Оцените, насколько легко понять назначение программы и её функций по сопровождающим текстам.
8. **Тесты:** проверьте наличие папки tests или файлов с тестами pytest, запустите pytest для оценки покрытия.
9. **Обработка ошибок:** проверьте, как программа реагирует на некорректный ввод (например, неверный формат даты). Наличие try/except и дружелюбных сообщений улучшают оценку.
10. **Производительность:** при возможности запустите профилировщик (cProfile) или повторите вывод расписания на большем объёме данных (например, несколько лет) и оцените время работы.

Детальные инструкции по установке окружения (виртуального окружения, установка библиотек и т.д.) повторяются аналогично для всех проектов и при необходимости могут быть опущены для краткости. Главное – обеспечить запуск кода и провести тестирование по каждому критерию.

# Детальная оценка проектов

# Как проводить проверку проектов (практически)

## 1. Проверка PEP-8 (стиля кода)

### Через консоль:

1. Установите flake8:
2. pip install flake8
3. Перейдите в папку проекта:
4. cd путь/к/проекту
5. Выполните проверку:
6. flake8 . --max-line-length=100

Вывод покажет строки с нарушениями стиля. Например:

main.py:23:80: E501 line too long (95 > 79 characters)

database.py:10:1: E302 expected 2 blank lines, found 1

### В PyCharm:

1. Откройте проект в PyCharm.
2. Меню: **Code** → **Inspect Code...**
3. Выберите весь проект или конкретную директорию.
4. PyCharm покажет результаты и предложения по стилю, включая ошибки PEP-8.

## 2. Проверка pylint

### Через консоль:

1. Установите pylint:
2. pip install pylint
3. Запустите проверку:
4. pylint имя\_файла.py

Пример вывода:

Your code has been rated at 7.50/10 (previous run: 7.50/10, +0.00)

Чем выше оценка, тем лучше. Цель — выше 8.5–9.

## 3. Проверка покрытия тестами

### Через pytest и coverage:

1. Установите:
2. pip install pytest coverage
3. Запустите:
4. coverage run -m pytest
5. coverage report -m

Вы получите отчёт с % покрытия по каждому модулю. Например:

name Stmts Miss Cover Missing

------------------------------------------------

main.py 40 5 88% 23-26, 39

## 4. Профилирование (измерение скорости)

### Простой способ через cProfile:

1. В консоли:
2. python -m cProfile main.py

Это выведет список всех функций, сколько раз они были вызваны и сколько времени заняли.

### Вывод в файл:

python -m cProfile -o profile.out main.py

Затем установите snakeviz для визуализации:

pip install snakeviz

snakeviz profile.out

Откроется браузер с графиком загруженности по функциям.

### Вставка прямо в код:

import cProfile

cProfile.run('main()')

## Проект 1 – «Алябьев»

**Структура и реализация:** проект состоит из одного скрипта lesson\_schedule.py. В нём определён шаблон дневного расписания (schedule\_template) и словарь предметов subjects для каждого дня недели (понедельник–пятница). Функция print\_schedule\_for\_month(year, month) перебирает дни указанного месяца, печатая дату и список занятий или обозначая выходные. В конце файла сразу вызывается print\_schedule\_for\_month(2025, 5), поэтому при запуске программы в консоли выводится расписание на май 2025. В файле **нет** дополнительных модулей или сложной логики: данные полностью интегрированы в коде, никаких операций ввода/вывода из внешних источников и отсутствует GUI.

**Функциональность (оценка 4 из 10):** базовое требование выполнено: программа при запуске печатает расписание (дату и предметы занятий на каждый будний день). Никаких дополнительных функций нет: поиск по дате, экспорт, интерактивность отсутствуют. Данные о расписании не загружаются извне, а жёстко запрограммированы (это допускается для базовой реализации). Суммарно, функционал минимален – за счёт этого оценка невысока.

**Читаемость кода (7/10):** код сравнительно небольшой и понятный. Имена переменных (например, schedule\_template, subjects, print\_schedule\_for\_month) достаточно описательные. Есть комментарии к шаблону расписания и ключевым частям функции, что упрощает понимание. Однако некоторых комментариев не хватает (например, пояснений параметров функции). В целом структура линейная и прозрачная.

**Стиль (6/10):** стиль в целом приемлемый. Используется snake\_case для имён функций и переменных. Есть несколько мелких замечаний по PEP-8: например, импортируются модули datetime и timedelta, но они не используются (flake8/pylint выдадут предупреждения о неиспользуемом импорте). Некоторая строка может выходить за рекомендуемую длину (но в данном скрипте не критично). Docstring для функции отсутствует. В целом стиль кода не идеален, но критичных нарушений нет.

**Модульность (5/10):** проект не модульный – это один скрипт без деления на компоненты. Для такой задачи один файл можно было бы считать приемлемым, но лучше было бы хотя бы разбить код на функции и/или вынести данные в отдельный модуль или конфиг. Отсутствие дополнительных файлов упрощает запуск, но с точки зрения модульности решение очень простое.

**Документированность (3/10):** читающий проект не найдет никакого README или внешней документации – все сведения о программе есть только в виде комментариев внутри кода. Комментарии есть лишь для описания формата расписания; функция имеет логичное название, но её можно было бы снабдить docstring. Отсутствуют инструкции по запуску или описание возможностей. Таким образом, документированность минимальна.

**Покрытие тестами (0/10):** тестов нет. В проекте отсутствует папка tests или файлы с тестами pytest.

**Обработка ошибок (2/10):** в коде нет никаких проверок пользовательского ввода (он вообще не предусмотрен) и нет конструкций try/except. При формировании расписания не обрабатываются исключения (например, на случай некорректного указания даты). Тем не менее программа написана достаточно просто, так что ошибки маловероятны. Но формально защита от неверных данных и отработки исключений нет.

**Производительность (9/10):** расчёт расписания на один месяц выполняется мгновенно. Алгоритмы примитивны (построение списка и вывод строк) – нет узких мест. Нагрузка очень низкая (несколько десятков записей), поэтому производительность практически не ограничивает возможности приложения. При больших данных (несколько лет) время будет линейно расти, но всё равно оставаться в разумных пределах.

## Проект 2 – «Zelinskiy\_Dmitriy»

**Структура и реализация:** проект – это единственный файл database.py, выполняющий генерацию и вывод расписания. В нём есть список SUBJECTS из 4 предметов. Функция generate\_full\_month\_schedule(year, month) проходит по всем дням месяца и для каждого дня формирует словарь занятия: дата, время и описание. Предмет выбирается циклически из списка SUBJECTS. Функция print\_full\_month\_schedule сортирует полученный список по дате и времени и выводит ежедневное расписание. В if \_\_name\_\_ == "\_\_main\_\_" используются фиксированные параметры (2025, 6). В целом проект динамически генерирует расписание, но никак не загружает/не сохраняет данные в файлах или БД: всё формируется в памяти.

**Функциональность (4/10):** проект выводит расписание в текстовом виде при запуске, поэтому базовая задача решена. Дополнительные функции отсутствуют: нет поиска, добавления/удаления, GUI или интеграции. Зато сам алгоритм генерации чуть более «живой», чем в предыдущем проекте, но это формальный нюанс. Функционально – чисто вывод заранее сгенерированных данных, поэтому балл за функциональность невысокий.

**Читаемость кода (7/10):** имена функций (generate\_full\_month\_schedule, print\_full\_month\_schedule) и переменных (SUBJECTS, days\_in\_month, schedule\_sorted и т.д.) достаточно понятны. Код разделён на две функции с ясной структурой: генерация расписания и печать. Оба блока имеют логическую смысловую завершённость. Присутствуют комментарии к списку предметов и к алгоритму добавления урока. Код можно читать и разбираться с ним без особых трудностей.

**Стиль (6/10):** стиль кода оценивается на 6 из 10. В целом стиль соблюдён: импорты расположены корректно, функции имеют осмысленные имена, форматирование аккуратное. Основные замечания связаны с отсутствием docstring у модуля и функций, что вызывает предупреждения Pylint (C0114, C0116). Рекомендуется добавить описания к модулю и функциям в формате docstring для улучшения читаемости и поддержки кода. Также есть предупреждения о переопределении переменных из внешней области видимости (W0621), что стоит учесть и при необходимости переименовать локальные переменные. В коде импортируется timedelta, который не используется (W0611) — этот импорт можно удалить.

**Модульность (5/10):** проект также монофайловый. Однако код логически разделён на функции, что хоть как-то упрощает чтение и позволяет переиспользовать логику. Но с точки зрения структуры проекта – один файл database.py выполняет всё. Нет отдельных модулей для утилит или моделей, но сама функциональность тоже небольшая. Можно считать, что модульность на среднем уровне: не идеально, но функции разбиты.

**Документированность (3/10):** у проекта тоже нет README и внешней документации. Есть комментарий перед списком SUBJECTS и текстовое описание задачи не предоставлено. Функциям не хватает docstring. Описание того, как запускать программу (что это выводит расписание) отсутствует. Следовательно, документированность слабая.

**Покрытие тестами (0/10):** тесты отсутствуют.

**Обработка ошибок (2/10):** проверок на допустимость входных данных нет (если в main задать, например, month=13, будет ошибка). В print\_full\_month\_schedule ошибки ввода не обрабатываются. Но сам код чтения данных простой, исключений по работе с базой или файлами нет (потому что БД не используется). Можно сказать, что проект почти не защищён от некорректных вводов. Зато он достаточно короткий, так что поведение при ошибке предсказуемо.

**Производительность (9/10):** расчёт расписания прост — он генерирует список из N элементов (N = количество дней) и сортирует его. Для месяца с N≈30 это быстро. Даже если использовать год или несколько лет (≤365 элементов), время выполнения будет пренебрежимо мало. Используемые алгоритмы O(N log N) по дате, что для привычных размеров данных не проблема. Профилировать нечего.

## Проект 3 – «Andrey Martynov»

**Структура и реализация:** проект представляет собой набор скриптов, реализующих интерактивный «планировщик» событий с использованием LLM и базы данных. Файлы:

* AddEventTool.py: содержит класс AddEvent, унаследованный от BaseTool (langchain). Читает конфиг config.ini для получения пути к SQLite-базе events.db, создаёт таблицу events при необходимости. Метод \_run парсит JSON-строку с полями события (event, start\_date, start\_time, end\_date, end\_time), проверяет обязательные поля и записывает событие в БД. В конце есть примерный тестовый вызов.
* FindEventsByDateTool.py: класс FindEventsByDate, также BaseTool. Определены вспомогательные функции merge\_date\_time и next\_day. Метод \_run принимает JSON-запрос с ключами start\_date, end\_date, start\_time, end\_time (в разных комбинациях), рассчитывает интервал поиска и выполняет SQL-запрос к БД на поиск событий, пересекающихся с периодом. Формирует читаемый для пользователя результат. В конце – примеры использования.
* LLM.py: настраивает объект LLM (используется langchain\_openai.ChatOpenAI) через параметры base\_url, api\_key и model из config.ini. Возвращает привязанный к API ChatOpenAI.
* main.py: основной сценарий запуска. Построен на langgraph (состояния чата, узлы инструментов). Определяется функция chatbot, задаётся системное сообщение (что «вы – электронный планировщик» и указаны правила обработки событий). Настроена связка инструментов add\_event и find\_events\_by\_date (импортированы из соответствующих файлов). Затем запускается цикл ввода-команд пользователя (input("User: ")). На каждое сообщение строятся события графа, выполняется LLM-инференс и, при необходимости, срабатывают инструменты (запрос к SQLite). Результаты выводятся в консоль.

Таким образом, проект – это консольное приложение, реализующее чат с ИИ-помощником по планированию событий. При запросах пользователя бот может сохранять события в базу и находить их по дате/интервалу. Прямого «вывода расписания на экран» нет: программа ожидает команд от пользователя.

**Функциональность (8/10):** функционал богатый: проект **не только** выводит фиксированное расписание, он позволяет пользователю вводить команды и взаимодействовать с «планировщиком». Реализованы поиск событий по дате и добавление новых событий, предусмотрена работа с временем и интервалами. Присутствует интеграция с LLM (LangChain), которая позволяет обрабатывать естественный язык пользователя. Есть даже примеры запросов и сценариев. Если рассматривать «вывод расписания» формально – программу нужно заставить (попросить бота) показать мероприятия, а не просто запустить и сразу увидеть готовый календарь. Но за счёт сложной логики и множества возможностей оценка функциональности высокая.

**Читаемость кода (7/10):** код довольно объёмный и использует продвинутые инструменты (langchain, langgraph). Имена функций и переменных в целом осмысленные (add\_event, find\_events\_by\_date, schedule\_manager и т.д.). В AddEventTool и FindEventsByDateTool есть подробные описания (description) и примеры в строках, что помогает понять назначение. Однако используемые библиотеки langchain, langgraph без дополнительного контекста усложняют восприятие для неподготовленного читателя. Код разделён на несколько файлов, что хорошо: каждый выполняет свою задачу. Проблемы: есть длинные строки (например, формирование SQL-запроса), локальные переменные с похожими именами, что затрудняет восприятие в некоторых местах. Но общая структура чётко разделена по файлам, так что знакомиться с ней относительно просто.

**Стиль (6/10):** в большинстве файлы написаны с соблюдением общих рекомендаций: используем двойные и одиночные кавычки по необходимости, методы классов расположены корректно. Префиксы BaseTool используются правильно.

Однако анализ pylint выявил ряд проблем:

1. Слишком длинные строки (превышение лимита в 100 символов)

2. Отсутствуют docstring для модуля и некоторых функций

3. Переопределение переменной из внешней области видимости

4. Неправильный порядок импортов (сторонние библиотеки должны идти перед локальными, стандартные - перед сторонними)

5. Стилистические проблемы с форматированием SQL и аннотациями типов

Вывод: код требует доработки в части соблюдения стилевых норм PEP 8, но не содержит критических синтаксических ошибок. Средний балл по стилю.

**Модульность (8/10):** проект хорошо структурирован: функции и классы разбиты по файлам по функциональным блокам (база данных, инструменты, LLM-конфиг, главный интерфейс). Такой уровень модульности позволяет легко поддерживать каждый компонент отдельно. Объекты AddEvent и FindEventsByDate вынесены в собственные модули. Главный чат-бот (main.py) осуществляет лишь координацию. Недостатки: в корне папки есть ещё events.db (бинарный файл БД), config.ini. Это не проблема само по себе. Нельзя назвать проект слишком «супер-модульным», но в рамках задачи с LLM он очень хорошо разнесён по файлам.

**Документированность (4/10):** документация минимальна. Есть config.ini с подсказками по заполнению (URL сервиса LLM, ключи и т.д.), но нет README или инструкции по запуску и требованиям (какой язык, какие библиотеки). Классы инструментов содержат описание (description для LLM), что технически является частью кода, но как документация это не воспринимается обычным пользователем. Хотя внутри кода есть поясняющие комментарии (и примеры JSON в описаниях), для полноценной документации проекта этого мало. Неясно, как готовить config.ini (нет примера ключа), какой именно сервис LLM необходим. В целом – хорошее кодовое комментирование, но нет внешней документации или инструкций.

**Покрытие тестами (0/10):** тестов нет.

**Обработка ошибок (8/10):** инструменты реализованы аккуратно. В AddEventTool.\_run обрабатывается ошибочный JSON (try/except json.JSONDecodeError), проверяются обязательные поля (event и start\_date) с понятными сообщениями. Работа с SQLite обёрнута в try/except Exception с сообщением об ошибке БД. Аналогично, в FindEventsByDateTool.\_run есть отлов неверного JSON, есть проверка наличия start\_date и информативные ответы, если события не найдены. В главном цикле консоли есть обработка пользовательского ввода (while True, ловится "exit"), но нет явных блоков try/except вокруг основных частей LLM-логики (предполагается, что langchain сам безопасно работает). В целом – проект предусмотрительно проверяет входные данные и пытается не сломаться.

**Производительность (7/10):** здесь смешанные факторы. С одной стороны, генерируется и хранится малое количество событий (в БД обычно пара десятков). Базы данных SQLite справляются с такими объёмами без проблем. С другой стороны, каждый запрос к LLM (по условиям работы чата) может занимать секунды или десятки секунд, что делает общую работу медленной. Однако эти задержки связаны с внешним API, а не с алгоритмической сложностью кода. Среди внутренних алгоритмов – создание SQL-запросов с перебором периодов – всё эффективно, нигде нет квадратичных вложенных циклов, всё построено на SQL-запросах. Если оценивать производительность «в коде», она приемлемая. Но в пересчёте на профилирование большая часть времени уйдёт на API LLM, а не на Python-логику.

## Проект 4 – «Kuzin\_Anton»

**Структура и реализация:** это полноценное GUI-приложение для управления расписанием, написанное с использованием PySide6 (Qt). Структура проекта:

* **config.py:** задаёт пути к БД.
* **core/database.py:** при инициализации создаётся БД с таблицами (subjects, groups, teachers, rooms, schedule) по скрипту SCHEMA\_SQL.
* **core/models.py:** содержит класс ScheduleManager с методами для взаимодействия с БД. Здесь есть методы add\_subject, get\_all\_subjects, add\_group, add\_teacher, add\_room, а также для работы с расписанием: add\_schedule\_entry, get\_full\_schedule, get\_filtered\_schedule, get\_schedule\_for\_date, delete\_schedule\_entry, get\_distinct\_schedule\_dates. Все запросы сформулированы через SQLite, результаты возвращаются в виде словарей.
* **ui/main\_windows.ui и ui/main\_window.py:** сгенерированные (и отредактированные) файлы интерфейса Qt – определяют виджет календаря, таблицы расписания, поля ввода и кнопки.
* **main.py:** основной класс MainWindow(QMainWindow), отвечающий за логику программы. При создании окна происходит инициализация БД и при необходимости заполнение тестовыми данными (30 случайных записей). Настраиваются сигналы/слоты: добавление, удаление записей, переключение дат в календаре, применение фильтра поиска. Метод display\_schedule\_for\_selected\_date выводит записи дня в таблицу. Есть методы для добавления новых занятий (с проверками ввода), удаления выбранных, фильтрации всех записей, и выделения дней календаря, где есть события (подсветка цветом).
* **run.py:** запускает приложение, создаёт QApplication и окно MainWindow.

В результате программа представляет собой полноценную систему: с календарём, возможностью CRUD операций над расписанием, фильтрацией по предметам/группам/преподавателям/дате, а также инициализацией тестовых данных.

**Функциональность (9/10):** реализовано всё необходимое и даже больше. При запуске пользователю доступно графическое окно с календарём и таблицей расписания на выбранную дату. Окно *«Все события»* показывает весь список занятий с фильтром по поисковой строке. Есть кнопки «Добавить запись» (с выбором предмета, группы, преподавателя и времени), «Удалить запись» для текущего дня и «Удалить запись» из таблицы «Все события» (с подтверждением). Программа автоматически подсвечивает даты с событиями и генерирует тестовые данные при пустой БД. Есть также выпадающие списки для выбора предмета, группы и т.д., что упрощает ввод. Компенсируя это, нет LLM, но GUI с полным CRUD – это сильный бонус. Единственное, чего не хватает – экспорта расписания (например, в CSV) или импорта из внешнего источника, но это не критично. Отлично решены поисковые и фильтрующие функции. В целом функциональность очень высокая.

**Читаемость кода (7/10):** код большой, но логически разделён по методам и классам. Имена методов и переменных осмысленны (add\_subject, load\_all\_schedule, apply\_schedule\_filter, schedule\_date и т.д.). Большая часть кода снабжена комментариями или docstring описаниями (несколько методов имеют тройные кавычки с описанием). Разделение core/ui/main упрощает понимание. Некоторая сложность вызвана тем, что это GUI: много строк по настройке интерфейса, работы со слотами и т.д. Это неизбежно делает скрипт длинным, но структура в целом ясна. Единственная значительная проблема читабельности – это наличие длинных SQL-запросов и SQL-скрипта в коде, а также переменные для временных значений времени. Однако общий стиль кода аккуратный, обработка ошибок локальна, и можно разобраться в логике.

**Стиль (7/10):** в целом соблюдены правила PEP-8: имена snake\_case, функциям по возможности даны docstring, используется PyCharm-генерация кода PySide (классы Ui\_MainWindow и т.д.). Длинные SQL-строки и длинные списки параметров (например, создание 30 случайных записей) нарушают ограничение в 80–100 символов, но для читаемости в данном контексте это допустимо. Файл README.md написан, что хорошо. В целом код выглядит аккуратно.

**Модульность (9/10):** прекрасная модульность. Проект чётко разделён на слои: core (бизнес-логика и работа с БД), ui (слои интерфейса), и основной код. Каждый класс в ScheduleManager отвечает за свою сущность (предметы, группы и т.д.). UI-слой отделён от логики. run.py служит точкой входа, main.py – логикой окна, database.py – работой с БД. Такое разделение соответствует «чистой архитектуре» и упрощает поддержку и расширение.

**Документированность (8/10):** в папке проекта есть README.md, подробно описывающий назначение проекта и дающий ключевые особенности. В тексте README приведён UML-диаграмма классов (в PlantUML) и описания сущностей (Database, ScheduleManager, Ui\_MainWindow, MainWindow). Метод \_\_init\_\_ в MainWindow снабжён docstring, а многие методы имеют комментарии или краткие строки-документации. Также есть вывод информационных сообщений через QMessageBox (например, после инициализации или добавления записи).

**Покрытие тестами (0/10):** тестов нет.

**Обработка ошибок (8/10):** ошибок «надёжно» обрабатывается много. При добавлении новой записи программа проверяет заполнение обязательных полей (предмет, группа, дата) и корректность времён (время начала раньше конца), и показывает предупреждения через диалог. При попытке удалить запись появляется диалог подтверждения, после удаления – уведомление об успехе или предупреждение в случае сбоя. Отключения базы данных практически нет (т.к. SQLite обеспечивает откат транзакций сам). Исключения, возникающие при обращении к базе (например, нарушения уникальности при добавлении справочных данных), перехватываются в ScheduleManager и влияют на возврат ID, но не приводят к аварии. Единичные try/except есть в методах удаления для показа критической ошибки. Таким образом обработка ввода и ошибок в UI-приложении продумана и надёжна.

**Производительность (7/10):** всё рассчитано на работу с небольшой базой (сотня-другая записей). Алгоритмы в get\_filtered\_schedule выполняют поиск по LIKE на текстовых полях – для небольших объёмов это быстро. Генерация тестовых данных – O(N) с N=30 – незначительна. GUI сам по себе может «тормознуть» при очень большом количестве строк, но на практике вывод в таблицы в Qt оптимизирован. Можно отметить, что при большем объёме данных (тысячи событий) поиск с LIKE мог бы стать медленнее, но для прикладного использования это приемлемо. С точки зрения условий (время <1.5 с на 1000 записей) – скорее всего, программа потянет и больше. Поэтому оценка по производительности высокая.

# Итоговые оценки проектов

Далее приведены результаты оценки по критериям и итоговый взвешенный балл каждого проекта.

| **Критерий** | **Вес** | **«Алябьев»** | **«Zelinskiy»** | **«Martynov»** | **«Kuzin»** |
| --- | --- | --- | --- | --- | --- |
| **Функциональность** | ×2 | 4 | 4 | 8 | 9 |
| **Читаемость кода** | ×1.5 | 7 | 7 | 7 | 7 |
| **Стиль** | ×1 | 6 | 6 | 6 | 7 |
| **Модульность** | ×1 | 5 | 5 | 8 | 9 |
| **Документация** | ×1.5 | 3 | 3 | 4 | 8 |
| **Покрытие тестами** | ×0.5 | 0 | 0 | 0 | 0 |
| **Обработка ошибок** | ×1 | 2 | 2 | 8 | 8 |
| **Производительность** | ×1.5 | 9 | 9 | 7 | 7 |
| **Итоговый взвеш. балл** |  | *49.5 из 100* | *49.5 из 100* | *65 из 100* | *75 из 100* |

Итоговый балл рассчитан как сумма всех критериев с учётом весов (Функциональность учитывается двойным). Чем больше итоговый балл, тем более полный и качественный проект.

# Заключение

Проекты продемонстрировали разный уровень проработки задачи. Простейшие решения (проекты «Алябьев» и «Zelinskiy») справляются с базовым требованием – выводом расписания – но практически не имеют дополнительных возможностей или документации. Их сильная сторона – компактность и простота (в том числе высокую производительность при любых объёмах они обеспечили), но они не масштабируются и плохо подходят к расширению функционала. Рекомендуется добавить документацию (README), разделять код на модули и предусмотреть хотя бы минимальную обработку ошибок и возможность ввода данных из внешних источников.

Проект «Andrey Martynov» отличается сложной логикой и нацелен на расширенный функционал: он умеет сохранять события и искать их по дате, имеет обработку JSON-запросов и интеграцию с LLM. Это очень мощное решение по функционалу, хотя требующее ручного общения с программой. Стоит отметить, что проект не печатает расписание напрямую, а ждёт от пользователя команд. Рекомендуется добавить базовую документацию по взаимодействию с чат-ботом.

Лидером по качеству является GUI-приложение «Kuzin\_Anton»: оно объединяет почти все возможные функции (просмотр, фильтрацию, добавление, удаление записей; календарь; удобный интерфейс). Код хорошо структурирован, понятен, подробно документирован. Рекомендуется лишь дополнить его системами тестирования (написание unit-тестов) и, возможно, добавить экспорт данных.

**Общие рекомендации:** во всех проектах рекомендуется добавить тесты (например, с помощью pytest) для критических функций (генерация расписания, CRUD-операции). Это повысит надёжность и позволит сразу заметить регрессии при расширении функционала. Также важно привести код в соответствие со стандартами (flake8/pylint), чтобы избежать стиля «лоскутного одеяла» и упростить работу с ним новым разработчикам. Финальные оценки проектов отражают как их текущие сильные стороны (базовый функционал, гибкость, GUI), так и области для улучшения (документирование, тестирование, расширение функциональности).

# Шаблон для оценок проектов другими участниками:

## Алябьев Григорий Олегович:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Критерий** | **Вес** | **«Алябьев»** | **«Zelinskiy»** | **«Martynov»** | **«Kuzin»** | **«Korneev»** |
| **Функциональность** | ×2 |  |  |  |  |  |
| **Читаемость кода** | ×1.5 |  |  |  |  |  |
| **Стиль** | ×1 |  |  |  |  |  |
| **Модульность** | ×1 |  |  |  |  |  |
| **Документация** | ×1.5 |  |  |  |  |  |
| **Покрытие тестами** | ×0.5 |  |  |  |  |  |
| **Обработка ошибок** | ×1 |  |  |  |  |  |
| **Производительность** | ×1.5 |  |  |  |  |  |
| **Итоговый взвеш. балл** |  |  |  |  |  |  |

## Зелинский Дмитрий Евгеньевич:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Критерий** | **Вес** | **«Алябьев»** | **«Zelinskiy»** | **«Martynov»** | **«Kuzin»** | **«Korneev»** |
| **Функциональность** | ×2 |  |  |  |  |  |
| **Читаемость кода** | ×1.5 |  |  |  |  |  |
| **Стиль** | ×1 |  |  |  |  |  |
| **Модульность** | ×1 |  |  |  |  |  |
| **Документация** | ×1.5 |  |  |  |  |  |
| **Покрытие тестами** | ×0.5 |  |  |  |  |  |
| **Обработка ошибок** | ×1 |  |  |  |  |  |
| **Производительность** | ×1.5 |  |  |  |  |  |
| **Итоговый взвеш. балл** |  |  |  |  |  |  |

## Мартынов Андрей Сергеевич:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Критерий** | **Вес** | **«Алябьев»** | **«Zelinskiy»** | **«Martynov»** | **«Kuzin»** | **«Korneev»** |
| **Функциональность** | ×2 |  |  |  |  |  |
| **Читаемость кода** | ×1.5 |  |  |  |  |  |
| **Стиль** | ×1 |  |  |  |  |  |
| **Модульность** | ×1 |  |  |  |  |  |
| **Документация** | ×1.5 |  |  |  |  |  |
| **Покрытие тестами** | ×0.5 |  |  |  |  |  |
| **Обработка ошибок** | ×1 |  |  |  |  |  |
| **Производительность** | ×1.5 |  |  |  |  |  |
| **Итоговый взвеш. балл** |  |  |  |  |  |  |

## Корнеев Антон Артурович

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Критерий** | **Вес** | **«Алябьев»** | **«Zelinskiy»** | **«Martynov»** | **«Kuzin»** | **«Korneev»** |
| **Функциональность** | ×2 |  |  |  |  |  |
| **Читаемость кода** | ×1.5 |  |  |  |  |  |
| **Стиль** | ×1 |  |  |  |  |  |
| **Модульность** | ×1 |  |  |  |  |  |
| **Документация** | ×1.5 |  |  |  |  |  |
| **Покрытие тестами** | ×0.5 |  |  |  |  |  |
| **Обработка ошибок** | ×1 |  |  |  |  |  |
| **Производительность** | ×1.5 |  |  |  |  |  |
| **Итоговый взвеш. балл** |  |  |  |  |  |  |