«Санкт-Петербургский политехнический университет»
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

по дисциплине

“Конструирование программного обеспечения ”

Выполнили студенты гр. 5130904/30101 Карпов Д. О.
Шабала А.Р.

Руководитель Иванов А.С.

Санкт-Петербург
2026

Оглавление

Задание

1. **Пройти основные этапы разработки ПО**:
    - Кратко сформулировать проблему, которую решает проект.
    - Описать пользовательские сценарии, оценить предполагаемое число пользователей (от 10 000 в сутки) и период хранения данных (>5 лет).
    - Разработать архитектуру: описать нагрузку, трафик, дисковую систему, нарисовать 2 диаграммы C4, описать API, нефункциональные требования, схему БД и масштабирование.
    - Реализовать код с коммитами от всех участников, покрыть код unit-тестами и реализовать интеграционный тест для одной пользовательской истории.
    - Описать сборку и запуск проекта через Docker (одна команда для сборки, unit- и интеграционных тестов, запуска приложения).
2. **Оформить отчет** в README.md репозитория: указать участников, группу, описать все этапы

Реализация

Сборка и запуск проекта

Для удобства развертывания и проверки проекта реализована полностью автоматизированная сборка и запуск приложения с помощью Docker. Это гарантирует, что проверяющему не потребуется устанавливать дополнительные зависимости - достаточно только Docker и bash.

Инструкция по сборке и запуску

1. Клонирование репозитория

```
git clone https://github.com/dwdfedf123/KPO
```

2. Запуск сборки, тестов и приложения одной командой

```
bash run_all.sh
```

Скрипт выполняет следующие действия:

- Сборка Docker-образа приложения.
- Запуск unit-тестов (python -m unittest discover tests/).
- Запуск интеграционных тестов (pytest integration_tests/).
- Автоматический запуск приложения (GUI доступен через VNC или локально, подробности - в README.md).

3. Ручной запуск через bash

   Если требуется поэтапный запуск:

```
git clone https://github.com/dwdfedf123/KPO
cd KPO\KPO\program_files
pip install -r requirements.txt
python main.py #запуск исполняемого файла программы

python -m unittest discover -s tests #запуск unit тестов

python -m pytest integration_tests -v #запуск всех и. и н. тестов
```

Минимальные требования

- Docker и bash должны быть установлены на системе проверяющего.

Примечание

Подробная инструкция по запуску, тестированию и работе с приложением находится в файле README.md репозитория (также отдельные README.md файлы есть в папках разработки, unit и интеграционного тестирования).

Определение проблемы

Многим студентам, школьникам и самообучающимся сложно сохранять концентрацию и мотивацию во время учебы.

Также затруднено отслеживание прогресса и времени, проведенного за учебными занятиями и другими видами деятельности, что мешает эффективно планировать и анализировать процесс обучения.

Выработка требований

Функциональные требования (ФТ)

Основные функции:

1. Таймер для учебы (по технике Pomodoro)
   - Возможность выбора интервалов (например, 25/5 минут)
   - Автоматический переход между рабочими и перерывными сессиями
   - Возможность ручной настройки времени таймера
2. Секундомер
   - Возможность запуска/остановки секундомера
   - Возможность сброса секундомер
3. Расписание студента Политеха
   - Подключение актуального расписания студента Политеха
   - Удобное для чтение форматирование
4. Система наград
   - Начисление баллов за учебные сессии
   - Разблокировка уровней и достижений
   - Возможность просмотра наград в отдельном разделе

Детализированные функциональные требования (ДФТ)

1. Регистрация и авторизация
   - Вход через почту/пароль
   - Вход через Google
   - Восстановление пароля
2. Работа с таймером
   - Пользователь может выбрать предустановленный режим Pomodoro или задать индивидуальный
   - Таймер продолжает работу в фоне, если приложение свернуто
   - Возможность остановки, паузы и перезапуска таймера
3. Работа с системой наград
   - За каждые X минут учебы начисляется N очков

- Достижения разблокируются при выполнении определенных условий (например, 5 учебных сессий подряд)
- Визуальное отображение прогресса к следующим наградам

4. Отчеты и статистика
   - Графики (линейный, круговой) с отображением времени учебы
   - Доступ к истории сессий за 5 лет
   - Анализ продуктивности пользователя по дням и неделям

Нефункциональные требования (НФТ)

1. Производительность
   - Время отклика основных функций ≤ 500 мс
   - Время загрузки приложения ≤ 5 секунд
   - Поддержка одновременной работы до 15 000 активных пользователей в сутки
2. Безопасность
   - Шифрование паролей (bcrypt)
   - Использование HTTPS для всех соединений
   - Бэкапы пользовательских данных (раз в неделю)
3. Совместимость
   - Windows
   - Поддержка экранов с разными DPI

4. Нагрузка на сервис

Ожидаемая нагрузка:

- 15 000 активных пользователей в сутки
- Пиковая нагрузка:
  - Утренний пик (08:00 – 12:00): до 1000 одновременных пользователей
  - Вечерний пик (18:00 – 22:00): до 900 одновременных пользователей
  - В ночное время (00:00 – 06:00): не более 200 одновременных пользователей
  - Предполагаемые операции и частота их выполнения:

Операция	Количество запросов в час (среднее)	Количество запросов в секунду (пиковое)
Авторизация / регистрация	~1 000	~0.5
Запуск таймера	~5 000	~2
Пауза / остановка таймера	~3 500	~1.5
Обновление статистики в реальном времени	~60 000	~20
Выдача наград	~8 000	~3
Запросы к базе данных (чтение, анализ статистики)	~90 000	~30

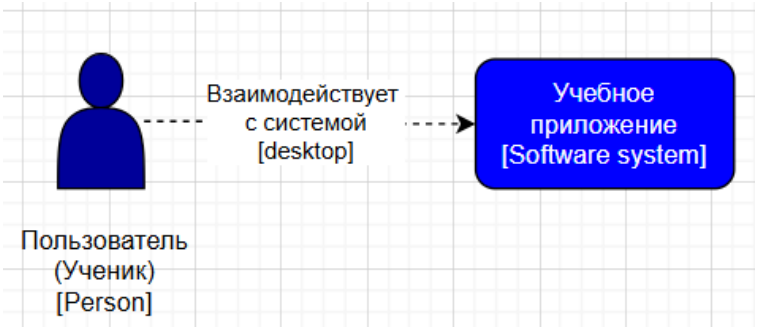
Операция	Количество запросов в час (среднее)	Количество запросов в секунду (пиковое)
Запросы к UI (интерфейс, обновление страниц)	~120 000	~40
Общее число запросов	~287 000 в час	~95 в секунду (в пике)

5. Оценка затрат

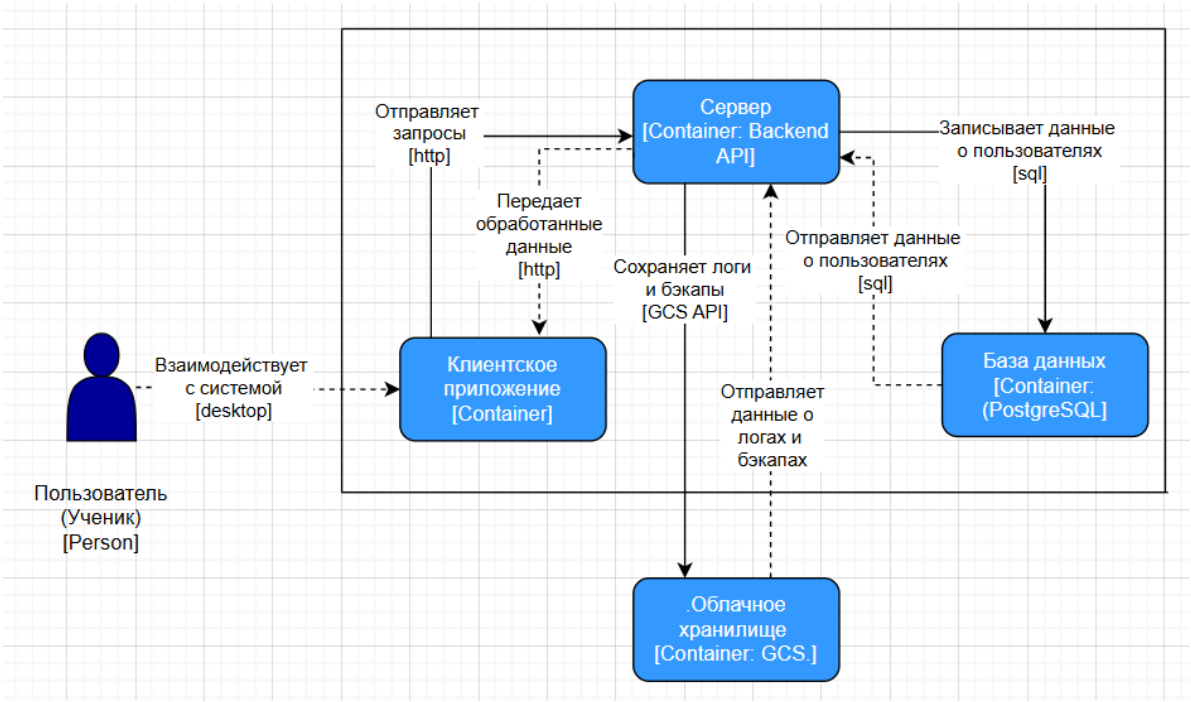
Категория	Описание	Стоимость в месяц
Хостинг	VPS (2 vCPU, 4GB RAM, 80GB SSD)	~$10
База данных	PostgreSQL (на VPS)	~$0 (включено в VPS)
Облачное хранилище	50GB для логов и бэкапов	~$5
Домен + SSL	.com + Let’s Encrypt	~$3
Итого	-	~$18 в месяц

Разработка архитектуры и детальное проектирование

1. Нагрузка и объемы
   - Соотношение R/W:
     - Чтение (70-80%): просмотр статистики (5-10 запросов/день), история занятий (при каждом входе), награды (1-2 раза/день).
     - Запись (20-30%): сохранение учебных сессий (5-10 записей/день), обновление настроек (1 раз/неделю).
   - Трафик:
     - 150–225 тыс. запросов/сутки (15 000 пользователей).
     - Средний ответ: 50–200 КБ → суточный трафик: 10–30 ГБ.
   - Дисковая система:
     - 54 ГБ/год для 15 000 пользователей.
     - Хранилище: 100 ГБ (хватит на 2 года с последующей архивацией).
2. Архитектура
   - C1-диаграмма контекста



   - C2-диаграмма контейнеров (приложены в документации).



3. Масштабирование (×10)

- API-сервер:
  - Горизонтальное масштабирование + балансировка нагрузки.
  - Кэширование частых запросов (Redis).
- Фронтенд:
  - CDN для статики + lazy loading.
- Инфраструктура:
  - Автоматическое масштабирование (Kubernetes/AWS).
  - Мониторинг (Prometheus/Grafana).
- Оптимизации:
  - Микросервисы (таймер, статистика, награды, расписание).
  - Очереди сообщений для асинхронных задач.

Разработка ПО

Технологический стек

Клиентская часть (Desktop приложение):

- Язык программирования: Python 3.8+
- GUI Framework:
  - customtkinter==5.2.2 - современная версия Tkinter с Material Design элементами
  - matplotlib==3.8.4 - визуализация статистики

Работа с данными:

- База данных: SQLite (встроенная, файл study.db)
- ORM: Нативный SQL-запросы
- Кэширование: Локальное хранение настроек в JSON

Внешние сервисы:

- Расписание: API Политеха (ruz.spbstu.ru)
- Время: WorldTimeAPI (worldtimeapi.org)
- Мотивация: Quotable API (api.quotable.io)

Дополнительные компоненты:

- Планировщик: APScheduler для работы с таймерами
- Безопасность:
    - bcrypt==4.0.1 - хеширование паролей
    - python-jose==3.3.0 - JWT-токены

Сборка и зависимости:

- Менеджер пакетов: pip (requirements.txt)
- Верстка: Нативные компоненты CTk с кастомными стилями

Архитектура приложения

Основные модули:

1. main.py - ядро приложения, интерфейс
2. database.py - работа с SQLite
3. auth.py - система аутентификации
4. timer.py - логика таймеров
5. api_client.py - интеграция с внешними API

Ключевые функции:

- 🔐 Авторизация/регистрация пользователей
- ⏱️ Таймер Pomodoro и секундомер
- 🗓️ Интеграция с расписанием Политеха (группа 5130904/30101)
- 🏆 Система достижений

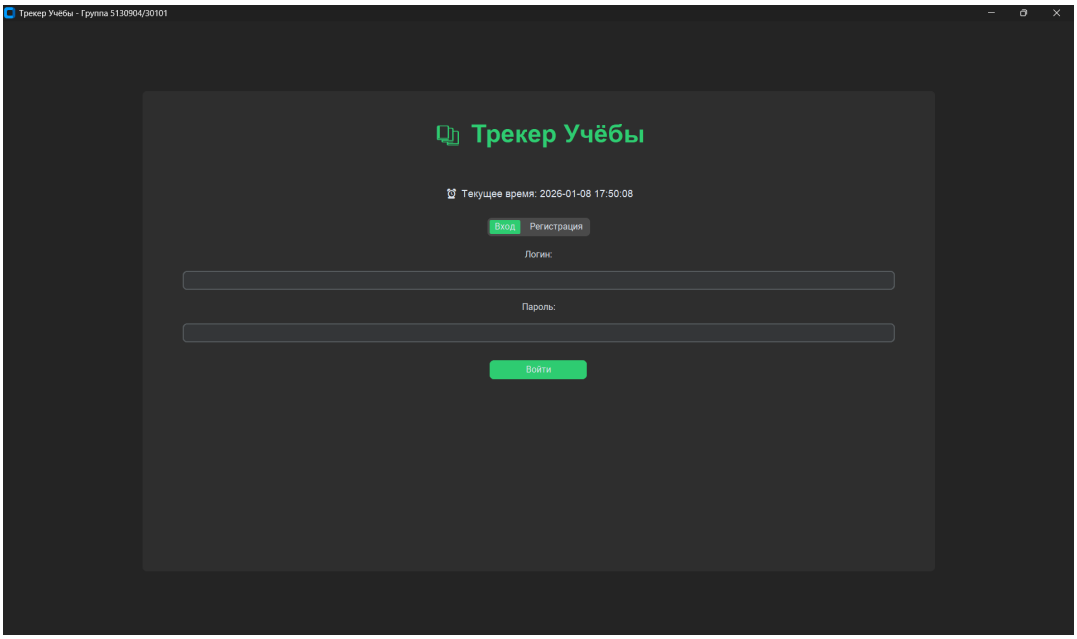
Особенности реализации:

1. Офлайн-работа - локальное хранение всех данных
2. Адаптивный дизайн - поддержка разных разрешений экрана
3. Безопасность - хеширование паролей + JWT
4. Интеграция с API - автоматическое обновление расписания

Реализация

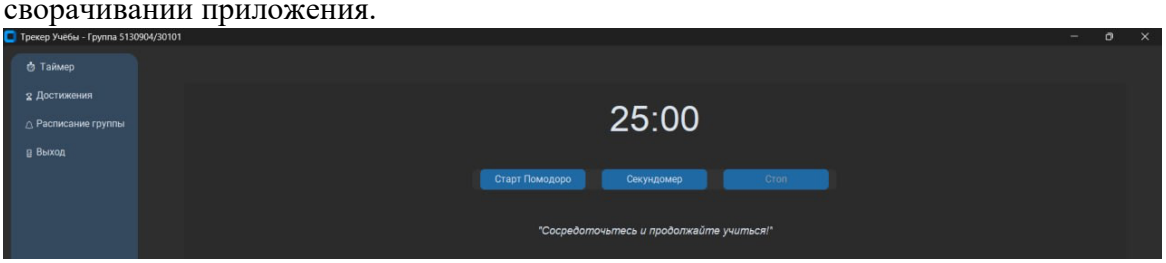
В ходе разработки программного обеспечения были реализованы все ключевые функции, обеспечивающие удобный и функциональный пользовательский опыт.

1. **Авторизация и регистрация пользователей**
   Реализована система входа через email и пароли с безопасным хешированием, а также регистрация новых пользователей. Интерфейс авторизации интуитивно понятен и адаптивен.



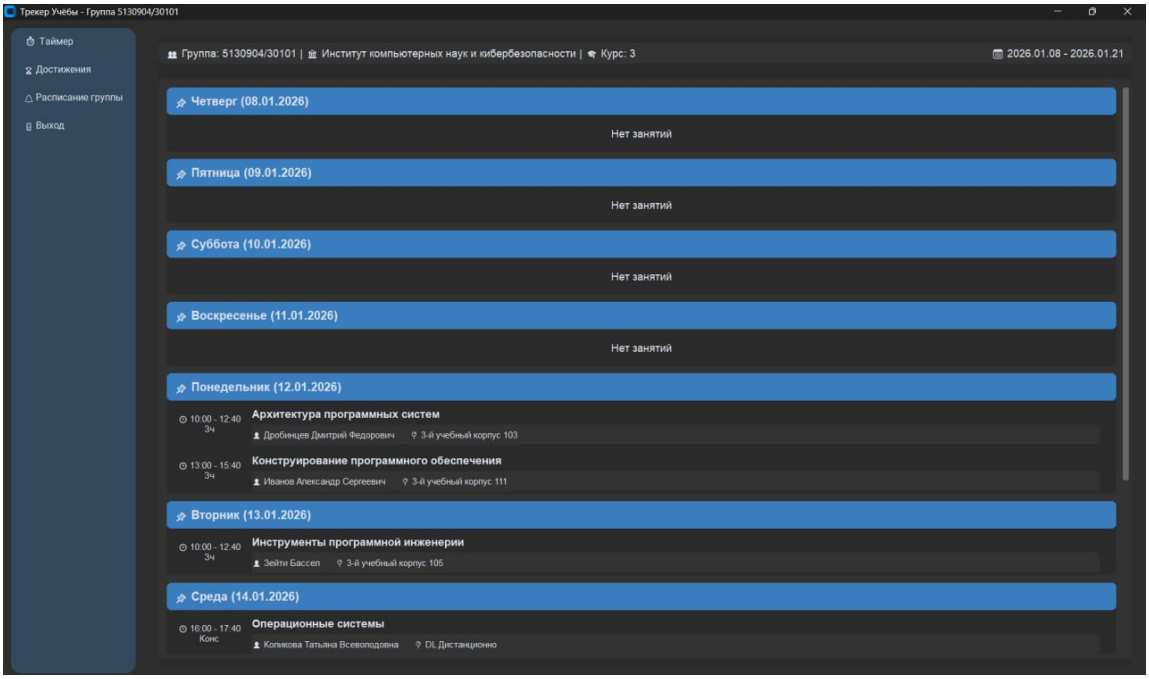
Скриншот 1: Экран входа и регистрации.

2. **Таймер Pomodoro и секундомер**
   Реализован таймер с возможностью выбора интервалов, автоматическим переключением между рабочими и перерывными сессиями, а также секундомер с функциями запуска, паузы и сброса. Таймер продолжает работать в фоне при сворачивании приложения.



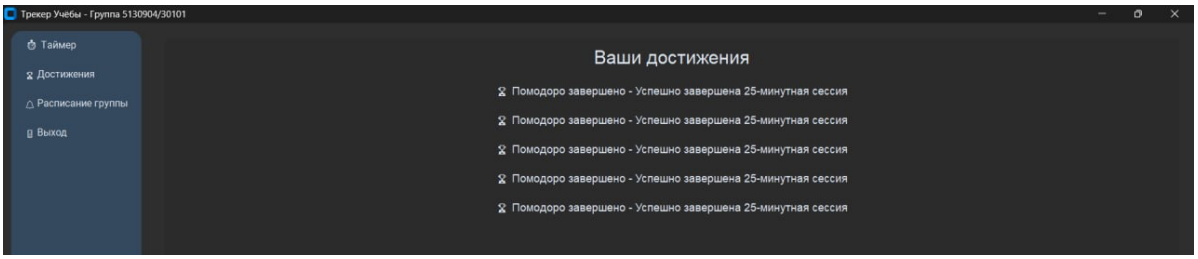
Скриншот 2: Основное окно с таймером Pomodoro. секундомером

3. **Интеграция с расписанием Политеха**
   Подключено актуальное расписание студента через API ruz.spbstu.ru. Расписание отображается в удобном для чтения формате с возможностью быстрого просмотра предстоящих занятий.



Скриншот 3: Экран с расписанием занятий.

4. **Система наград и достижений**
   Реализована система начисления баллов за учебные сессии, разблокировки уровней и достижений. Пользователь может просматривать свои награды в отдельном разделе с визуализацией прогресса.



Скриншот 4: Раздел с достижениями и наградами.

Unit-тестирование

Цель работы

Проверить корректность работы основных компонентов проекта “Трекер Учёбы” с помощью модульного тестирования, выявить возможные ошибки на раннем этапе и убедиться в стабильности ключевых функций.

Проведённые действия

1. Выделены ключевые модули для тестирования:
   - Аутентификация пользователей (auth.py)
   - Работа с базой данных (database.py)
   - Взаимодействие с внешними API (api_client.py)
2. Разработаны юнит-тесты с использованием встроенного модуля unittest.
   - Общая структура тестов сохранена в папке tests/, каждый модуль тестируется в отдельном файле.
3. Использована техника мокирования (mock):
   - Для изоляции от внешних сервисов (requests.get) применён unittest.mock.
   - Это позволило эмулировать поведение API без необходимости подключения к интернету.
4. Автоматически очищается база данных перед каждым тестом — для предотвращения влияния предыдущих данных.

Покрытие тестами

Модуль Проверяемые функции Кол-во тестов
auth.py Регистрация, вход, проверка токена 4

database.py Пользователи, сессии, достижения 4+
api_client.py Мотивация, время, расписание 8

Итого: 16+ тестов, покрывающих критически важные компоненты приложения.

Результат

Все тесты успешно проходят:

```
----------------------------------------------------------------------
Ran 16 tests in 0.12s

OK
```

Тесты подтверждают, что:

• Пользователь может успешно зарегистрироваться и войти;
• База данных корректно сохраняет и извлекает информацию;
• API-интерфейсы обрабатываются корректно, даже при ошибках или недоступности сервиса.

Разработка юнит-тестов позволила убедиться в корректной работе приложения при различных сценариях, включая ошибки внешних API. Использование unittest и mock позволило добиться надёжности, изоляции и повторяемости тестов. Это повышает качество кода и упрощает дальнейшее сопровождение проекта.

Интеграционное тестирование

Цель работы

Проверить корректность взаимодействия модулей системы в реальных сценариях использования, убедиться в целостности рабочего процесса приложения и сохранении данных в БД.

Проведённые действия

1. Выделены ключевые рабочие процессы для тестирования:
   - Полный цикл работы с Pomodoro-таймером
   - Взаимодействие между модулями аутентификации, таймера и базы данных
2. Разработаны интеграционные тесты:
   - Тесты расположены в папке integration_tests/
   - Каждый тест проверяет комплексное взаимодействие компонентов
3. Особенности тестирования:
   - Используется реальная база данных (не моки)
   - Все тестовые данные автоматически удаляются после выполнения
   - Для изоляции тестов применяются уникальные имена пользователей
4. Техническая реализация:
   - Использован фреймворк pytest для удобного написания тестов
   - Применены фикстуры для настройки тестового окружения
   - Реализована автоматическая очистка БД после тестов

Покрытие тестами

Основной тест: test_pomodoro_workflow.py

Проверяет полный цикл работы:
1. Регистрация нового пользователя
2. Аутентификация пользователя
3. Запуск Pomodoro-сессии
4. Корректное сохранение данных сессии в БД
5. Автоматическую очистку тестовых данных

Проверяемые аспекты:
- Взаимодействие модулей Auth ↔ Timer ↔ Database
- Целостность данных в БД
- Корректность временных меток

Результат

Все интеграционные тесты успешно проходят:

```
============== test session starts ==============
collected 1 item

integration_tests/test_pomodoro_workflow.py::test_pomodoro_workflow
PASSED [100%]

============== 1 passed in 1.67s ==============
```

Тесты подтверждают что:

- Пользователь может успешно зарегистрироваться и войти в систему
- Таймер корректно запускается и останавливается
- Данные о сессиях правильно сохраняются в БД
- Временные параметры сессий рассчитываются верно
- Система автоматически очищает тестовые данные
- Подтверждена корректная интеграция модулей и работа с БД
- Pytest обеспечил эффективное тестирование с изоляцией сценариев
- Система готова к расширению тестового покрытия

Нагрузочное тестирование

Цель работы

Проверить устойчивость приложения к высокой нагрузке пользователей, имитируя одновременную работу множества пользователей в рамках функционала:
- Авторизация/регистрация
- Работа Pomodoro-таймера
- Получение достижений
- Запрос расписания занятий
- Получение мотивационных цитат

Для проведения нагрузочного тестирования был выбран Locust — фреймворк для имитации поведения пользователей и анализа производительности систем под нагрузкой. Он был выбран по следующим причинам:
- Поддерживает Python-скрипты , что удобно для GUI-приложений.

- Интегрируется без необходимости HTTP-сервера , так как позволяет запускать произвольные Python-функции.
- Удобный веб-интерфейс дает наглядное представление о результатах тестирования.
- Гибкая настройка сценариев поведения пользователей.

Нагрузочные параметры

Был проведен тест продолжительностью 10 минут с использованием:
- Пиковое количество пользователей : 200
- Скорость создания новых пользователей : 10 в секунду

Почему эти значения?
- 10-минутная симуляция позволяет оценить стабильность работы приложения за достаточно длительный период.
- 200 пользователей выбрано как показатель, который значительно превышает среднюю нагрузку (~7 пользователей в минуту), чтобы протестировать запас прочности системы.
- Скорость создания пользователей — 10 в секунду — обеспечивает плавное увеличение нагрузки, что позволяет наблюдать за реакцией системы постепенно.

Все задачи в тесте имитируют действия реального пользователя:
- Вход в систему
- Попытка авторизации каждого пользователя
- Запуск Pomodoro-сессии
- Имитация запуска и остановки таймера
- Получение достижений
- Чтение данных из БД
- Получение расписания
- Вызов внешнего API
- Получение мотивационной цитаты
- Вызов внешнего API

Эти действия были выбраны потому, что они являются основными рабочими процессами приложения и задействуют все ключевые модули: базу данных, аутентификацию, таймер, графический интерфейс и внешние API.

Результаты тестирования

Все задачи выполнены успешно:
- Приложение выдержало пиковую нагрузку в 200 пользователей .
- Все операции (вход, запуск таймера, запросы к API) отработали корректно.
- Система не столкнулась с ошибками или значительным замедлением.

Полученные метрики:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/achievements	9149	0	0	1	1	0.11	0	1	0	12.7	0
GET	/api/group-schedule	4632	0	3700	17000	29000	5538.53	18	43651	0	12.7	0
GET	/api/quote	4503	0	340	2000	12000	770.65	1	25543	38	5	0
POST	/pomodoro/session	13835	0	2300	3500	14000	2527.49	20	30252	0	16.4	0
	Aggregated	32119	0	1600	5200	17000	1995.5	0	43651	5.33	46.8	0

Нагрузочное тестирование было проведено с помощью Locust , что позволило:
- Оценить поведение приложения при одновременной работе до 200 пользователей.
- Проверить надежность взаимодействия с базой данных, таймером и внешними API.
- Подтвердить, что приложение способно масштабироваться для обслуживания большого числа пользователей.
- Выбор Locust и указанных параметров нагрузки обусловлен необходимостью проверить производительность и стабильность логики приложения в условиях, приближенных к реальным.

Вывод

В ходе выполнения курсового проекта был реализован полнофункциональный трекер учебы с поддержкой Pomodoro-таймера, секундомера, интеграцией с расписанием Политеха и системой наград.

Проведены модульные и интеграционные тесты, подтверждающие корректность работы ключевых компонентов.

Все этапы – от проектирования архитектуры до тестирования и автоматизированной сборки – выполнены с учетом современных требований к качеству и безопасности ПО. Проект готов к дальнейшему развитию и масштабированию.

Приложение

Все исходные файлы, инструкции по запуску, тестам и документация находятся в открытом репозитории:

https://github.com/dwdfedf123/KPO

Литература

1. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
2. Martin R.C. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.
3. Docker Documentation - https://docs.docker.com/
4. Python 3.8 Documentation - https://docs.python.org/3/
5. Tkinter Documentation - https://docs.python.org/3/library/tkinter.html
6. APScheduler Documentation - https://apscheduler.readthedocs.io/
7. SQLite Documentation - https://www.sqlite.org/docs.html
8. pytest Documentation - https://docs.pytest.org/
9. GitHub Guides - https://docs.github.com/en/get-started/quickstart
10. Официальная документация API ruz.spbstu.ru