Санкт-Петербургский политехнический университет
Институт компьютерных наук и кибербезопасности
“Высшая школа программной инженерии”

Курсовая работа

по дисциплине “Конструирование ПО”

Выполнили
студенты гр. 5130904/20104 Чемодуров А.А.

Литвин С.Р.

Руководитель Юркин В.А.

Санкт-Петербург
2025

Оглавление

Формулировка задания

Разработать Telegram-бота, предоставляющего пользователям удобный интерфейс для поиска книг через интеграцию с Google Books API, с возможностью сохранения понравившихся книг в персональное "Избранное" и управления списком сохраненных книг.
Работа выполнена в команде: Литвин София, Чемодуров Артем.

Основные требования

Бэкенд-часть

1. Telegram-бот на Python с функционалом:
   - Поиск книг через Google Books API
   - Сохранение в избранное
   - Управление списком избранного
   - Полнотекстовый поиск по названиям и авторам
2. Архитектурные слои:
   - Слой представления (Telegram handlers)
   - Бизнес-логика (сервисы поиска и работы с избранным)
   - Репозитории (PostgreSQL)
3. Хранение данных:
   - PostgreSQL для персистентного хранения
   - Кеширование популярных запросов

Тестирование

1. Юнит-тесты:
   - Покрытие основных компонентов бота
   - Тесты API клиентов (Google Books)
2. Интеграционные тесты:
   - Взаимодействие с базой данных
   - Тестирование сценариев пользователя

DevOps

1. Контейнеризация:
   - Docker-образы для бота, базы данных
   - Bash скрипты для запуска Docker-образов

Технические требования

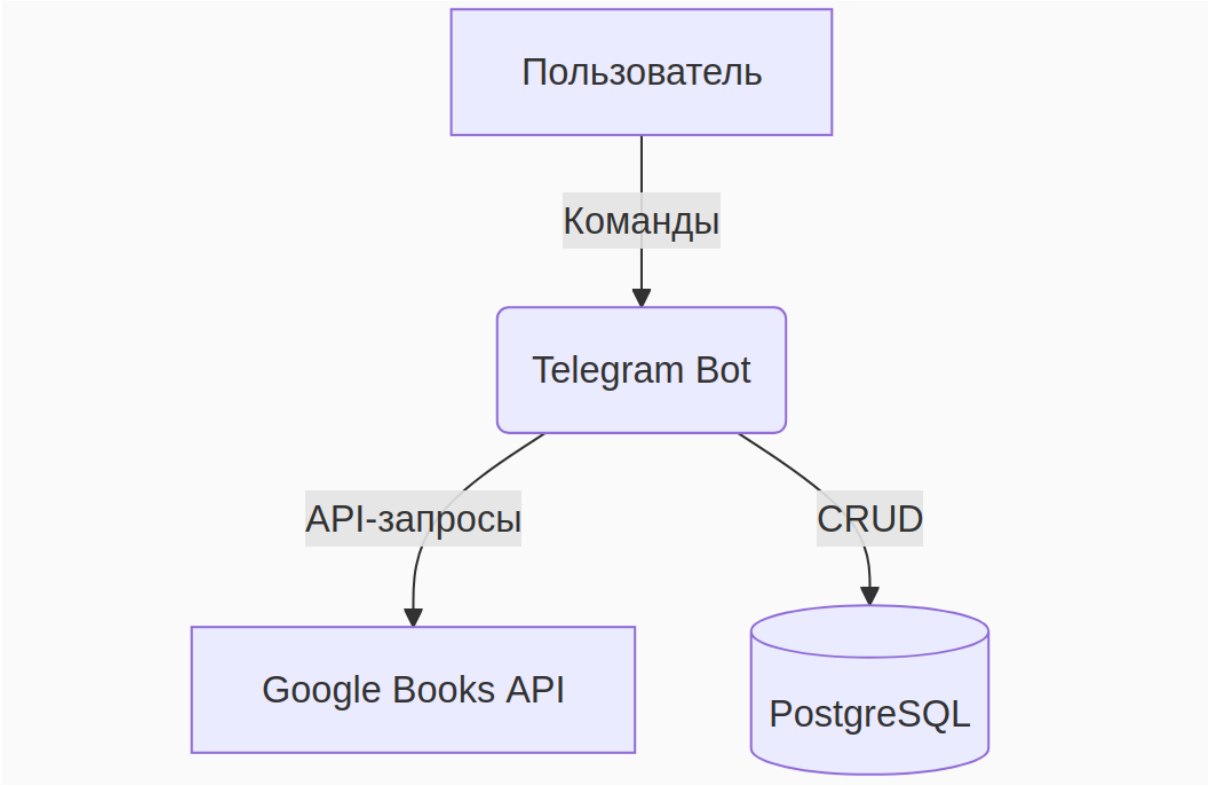
- **Язык**: Python 3.10+
- **Библиотеки**: python-telegram-bot, SQLAlchemy, aiohttp
- **База данных**: PostgreSQL 14+

Характер нагрузки

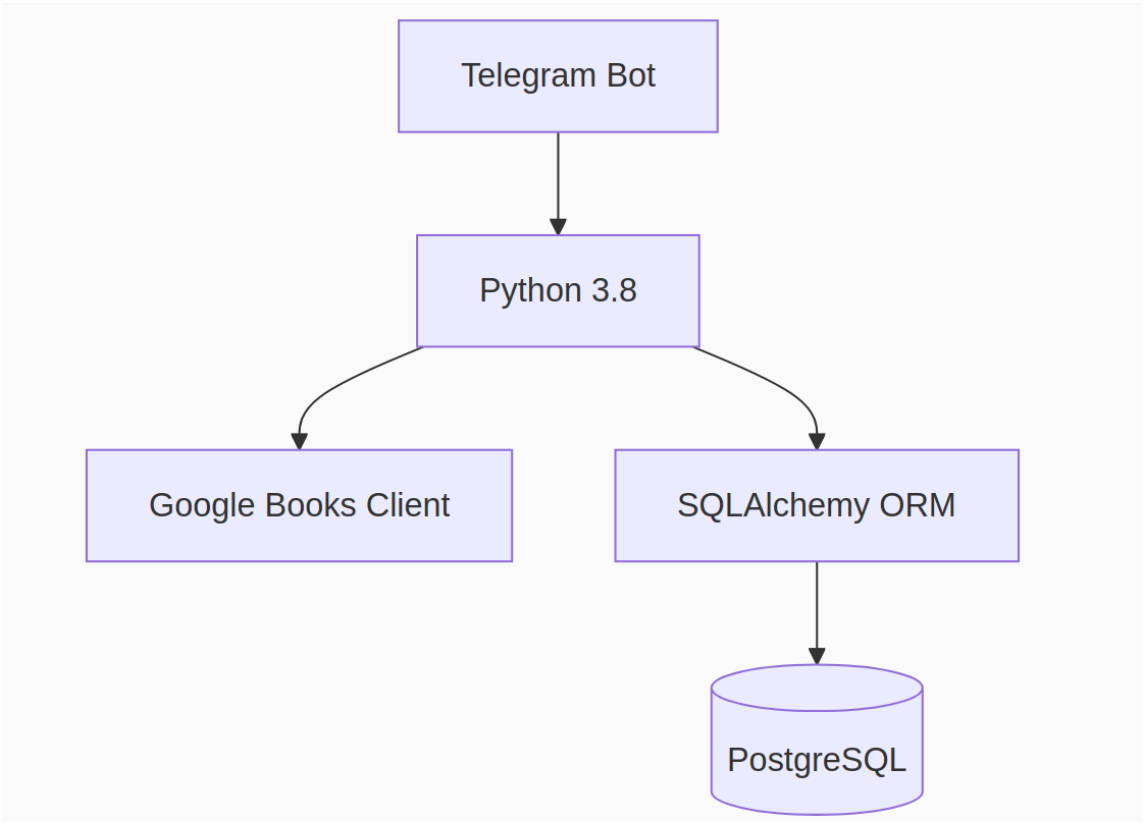
1. **Соотношение R/W**: 3:1
   - Read: Поиск книг, просмотр избранного (75%)
   - Write: Добавление/удаление из избранного (25%)
2. **Объемы трафика**:
   - 10,000 запросов/сутки
   - 700-1500 MB/месяц
3. **Дисковая система**:
   - PostgreSQL: 100-200 GB
   - Логи: 2 GB/месяц

Диаграммы C4 model

Контекстная диаграмма



Контейнерная диаграмма:



API-контракты и нефункциональные требования

Контракт для Google Books API:

```
endpoint: GET https://www.googleapis.com/books/v1/volumes
params:
  q: string  # Поисковый запрос
  maxResults: integer
  key: string  # API-ключ
response:
  items:
- id: string
        volumeInfo:
        title: string
        authors: string[]
        description: string
        imageLinks: {thumbnail: string}
```

Контракт для OpenLibrary API:

Эндпоинт: GET https://openlibrary.org/search.json

Параметры:

```
{
  q: string          // Поисковый запрос
  limit?: number             // Лимит результатов (по умолчанию 10)
  language?: string  // Язык (например "rus" для русского)
  fields?: string   // Дополнительные поля (через запятую)
}
```

Нефункциональные требования

1. Производительность:
   - Время ответа: <1 сек (95% запросов)
   - Доступность: 99%
2. Ограничения:
   - Лимит Google Books API: 1000 запросов/день, после этого используется Open Library
   - Максимальный размер БД: 200GB

Схема базы данных

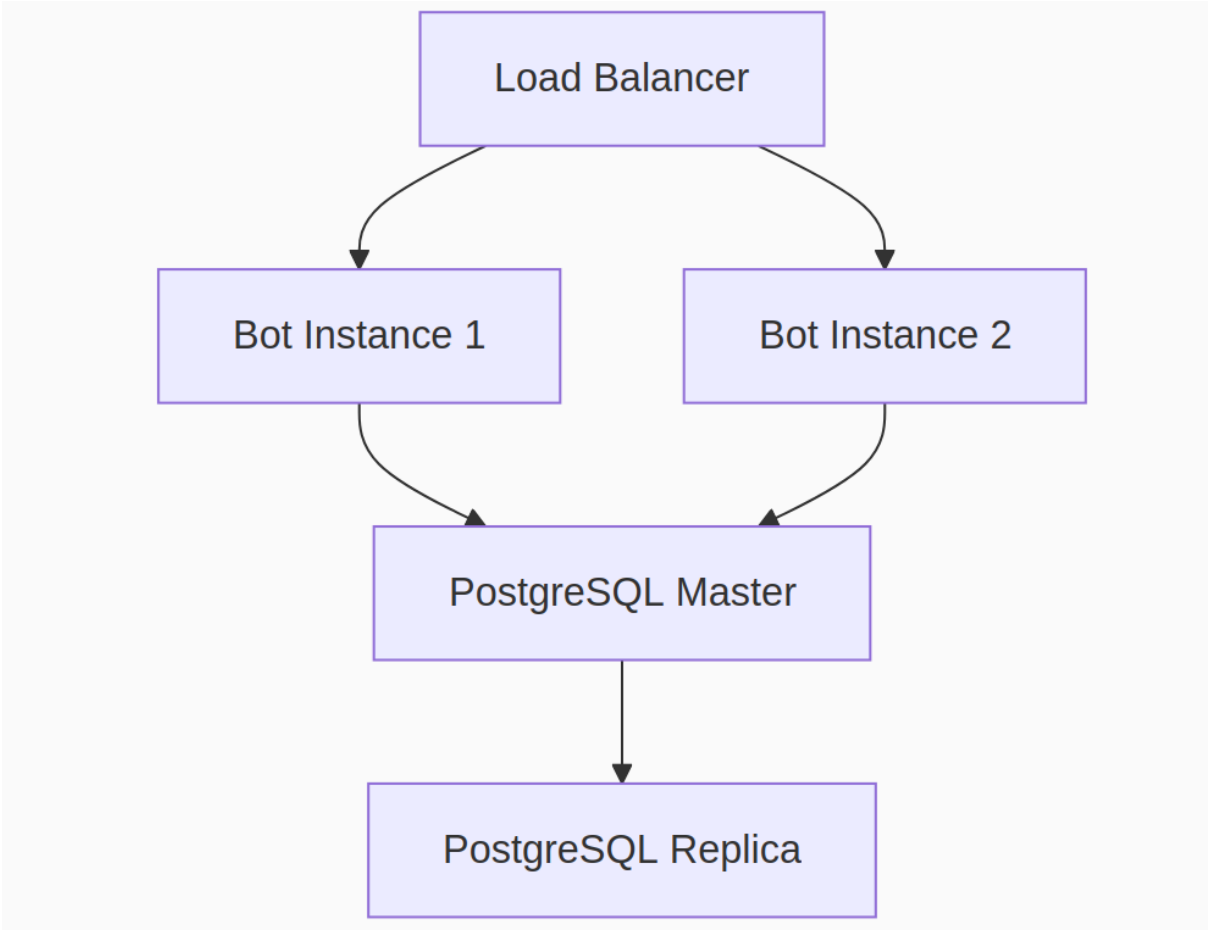
Есть таблица favorite_books(string book_id, string title, string authors, text description, string thumbnail_url, bigint user_id).

База данных должна соответствовать нефункциональным требованиям, потому что мы в ней создали следующие индексы для ускорения:

```
CREATE INDEX idx_user_books ON favorite_book(user_id);

CREATE INDEX idx_book_id ON favorite_book(book_id);
```

Схема масштабирования при росте нагрузки в 10 раз

1. Вертикальное:
   - Увеличение CPU/RAM для PostgreSQL (2 vCPU → 4 vCPU)
   - Кэширование Redis для популярных поисковых запросов
2. Горизонтальное:



3. Оптимизации:
   - Асинхронные запросы: Использование aiohttp вместо requests
   - Батчинг: Группировка запросов к БД
   - CDN: Для кэширования обложек книг

User Story

1. Поиск книг

Как пользователь,
Я хочу искать книги по названию, автору или ключевым словам,
Чтобы быстро находить нужные мне произведения.

Примечание:

- Интеграция с Google Books API для получения актуальных данных.
- Отображение обложки, автора и краткого описания.
- Поддержка пагинации при большом количестве результатов.

2. Сохранение в избранное

Как читатель,
Я хочу сохранять понравившиеся книги в персональный список,
Чтобы легко возвращаться к ним позже.

Примечание:

- Возможность добавлять/удалять книги из избранного.
- Синхронизация между устройствами через учетную запись.
- Категоризация по жанрам или тегам (опционально).

3. Просмотр избранного

Как пользователь,
Я хочу видеть список сохраненных книг с возможностью фильтрации,
Чтобы быстро находить нужные мне произведения.

Примечание:

- Отображение обложек и ключевых метаданных.
- Поиск по названию/автору внутри избранного.
- Экспорт списка в CSV или PDF (опционально).

4. Управление профилем

Как пользователь,
Я хочу регистрироваться и настраивать свой профиль,
Чтобы получить персонализированные рекомендации.

Примечание:

- Привязка к Telegram ID.
- Настройка уведомлений о новых книгах.
- История поисковых запросов.

Результат

Полностью функциональный Telegram-бот с поиском через Google Books API, персональной библиотекой избранного, удобным интерфейсом на основе кнопок и команд, хранением данных в PostgreSQL.

Юнит тесты

Были юнит тесты для db.py, bot.py, google_books.py, open_lib.py.

test_db.py

```
sofia@sofia:~/kpo$ python3 -m pytest ./units/src/test_db.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.0, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/sofia/kpo
plugins: asyncio-1.0.0, anyio-4.9.0
asyncio: mode=strict, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 8 items

units/src/test_db.py::TestDatabase::test_add_favorite_success PASSED     [ 12%]
units/src/test_db.py::TestDatabase::test_add_favorite_duplicate PASSED   [ 25%]
units/src/test_db.py::TestDatabase::test_get_favorites_empty PASSED      [ 37%]
units/src/test_db.py::TestDatabase::test_get_favorites_with_data PASSED  [ 50%]
units/src/test_db.py::TestDatabase::test_get_favorites_filter_by_user PASSED [ 62%]
units/src/test_db.py::TestDatabase::test_remove_favorite_success PASSED  [ 75%]
units/src/test_db.py::TestDatabase::test_remove_favorite_not_found PASSED [ 87%]
units/src/test_db.py::TestDatabase::test_remove_favorite_wrong_user PASSED [100%]

============================== 8 passed in 0.88s ===============================
```

test_google_books.py

```
sofia@sofia:~/kpo$ python3 -m pytest ./units/src/test_google_books.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.0, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/sofia/kpo
plugins: asyncio-1.0.0, anyio-4.9.0
asyncio: mode=strict, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 3 items

units/src/test_google_books.py::TestGoogleBooksAPI::test_search_books_success PASSED [ 33%]
units/src/test_google_books.py::TestGoogleBooksAPI::test_search_books_empty_response PASSED [ 66%]
units/src/test_google_books.py::TestGoogleBooksAPI::test_search_books_error_handling PASSED [100%]

============================== 3 passed in 0.17s ===============================
```

test_open_lib.py

```
sofia@sofia:~/kpo$ python3 -m pytest ./units/src/test_open_lib.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.0, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/sofia/kpo
plugins: asyncio-1.0.0, anyio-4.9.0
asyncio: mode=strict, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 6 items

units/src/test_open_lib.py::TestOpenLibraryAPI::test_search_books_success PASSED [ 16%]
units/src/test_open_lib.py::TestOpenLibraryAPI::test_search_books_empty_result PASSED [ 33%]
units/src/test_open_lib.py::TestOpenLibraryAPI::test_search_books_error_handling PASSED [ 50%]
units/src/test_open_lib.py::TestOpenLibraryAPI::test_parse_results PASSED [ 66%]
units/src/test_open_lib.py::TestOpenLibraryAPI::test_get_cover_url PASSED [ 83%]
units/src/test_open_lib.py::TestOpenLibraryAPI::test_clean_description PASSED [100%]

============================= warnings summary =================================
units/src/test_open_lib.py::TestOpenLibraryAPI::test_parse_results
  units/src/test_open_lib.py:90: PytestWarning: The test <Function test_parse_results> is marked with '@pytest.mark.asyncio' but it is not an async function. Please remove the asyncio
 mark. If the test is not marked explicitly, check for global marks applied via 'pytestmark'.
    def test_parse_results(self, mock_book_data):

units/src/test_open_lib.py::TestOpenLibraryAPI::test_get_cover_url
  units/src/test_open_lib.py:103: PytestWarning: The test <Function test_get_cover_url> is marked with '@pytest.mark.asyncio' but it is not an async function. Please remove the asynci
o mark. If the test is not marked explicitly, check for global marks applied via 'pytestmark'.
    def test_get_cover_url(self):

units/src/test_open_lib.py::TestOpenLibraryAPI::test_clean_description
  units/src/test_open_lib.py:112: PytestWarning: The test <Function test_clean_description> is marked with '@pytest.mark.asyncio' but it is not an async function. Please remove the as
yncio mark. If the test is not marked explicitly, check for global marks applied via 'pytestmark'.
    def test_clean_description(self):

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
================== 6 passed, 3 warnings in 0.88s ===========================
```

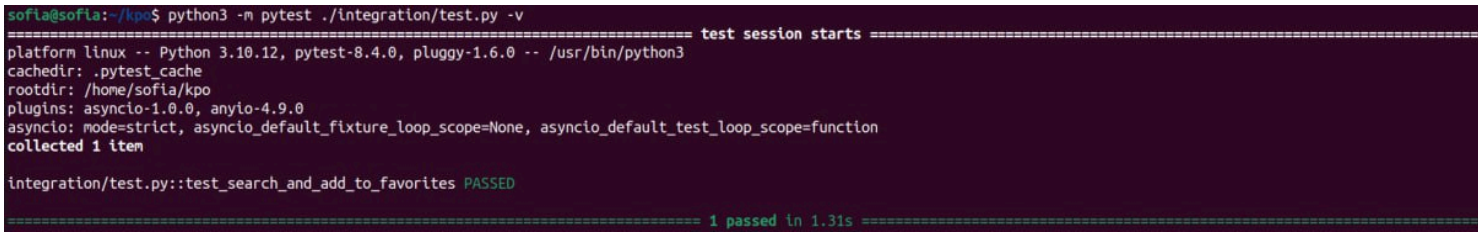
test_bot.py

```
sofia@sofia:~/kpo$ python3 -m pytest ./units/src/test_bot.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.0, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/sofia/kpo
plugins: asyncio-1.0.0, anyio-4.9.0
asyncio: mode=strict, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 14 items

units/src/test_bot.py::test_start PASSED                                 [  7%]
units/src/test_bot.py::test_help PASSED                                  [ 14%]
units/src/test_bot.py::test_search_books_no_query PASSED                 [ 21%]
units/src/test_bot.py::test_search_books_no_results PASSED               [ 28%]
units/src/test_bot.py::test_search_books_with_google_results PASSED      [ 35%]
units/src/test_bot.py::test_search_books_with_openlib_results PASSED     [ 42%]
units/src/test_bot.py::test_show_favorites_empty PASSED                  [ 50%]
units/src/test_bot.py::test_show_favorites_with_books PASSED             [ 57%]
units/src/test_bot.py::test_handle_button_click_add PASSED               [ 64%]
units/src/test_bot.py::test_handle_button_click_remove PASSED            [ 71%]
units/src/test_bot.py::test_handle_button_click_remove_not_found PASSED  [ 78%]
units/src/test_bot.py::test_handle_button_click_invalid_data PASSED      [ 85%]
units/src/test_bot.py::test_get_book_data_google PASSED                  [ 92%]
units/src/test_bot.py::test_get_book_data_openlib PASSED                 [100%]

============================== 14 passed in 3.24s ==============================
```

Все тесты прошли успешно.

Интеграционное тестирование

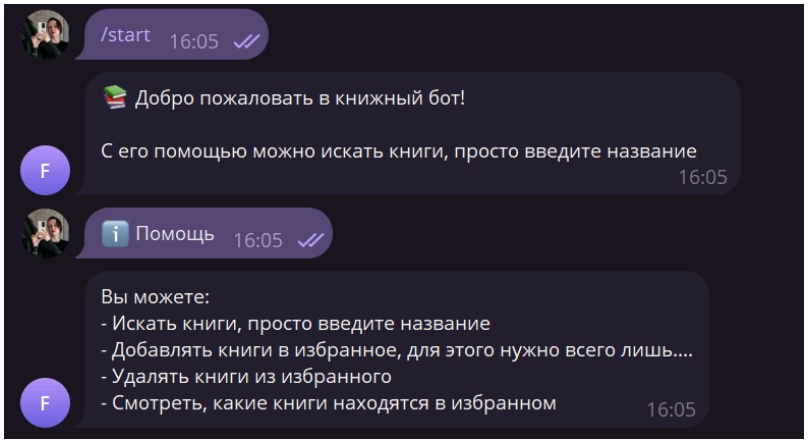
test.py



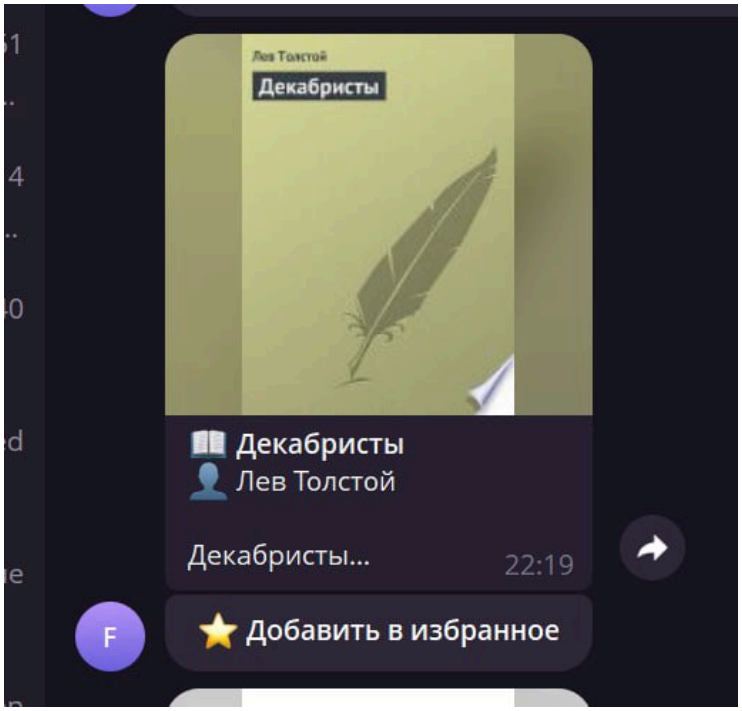
Интеграционный тест тоже прошел успешно.

Демонстрация работы бота

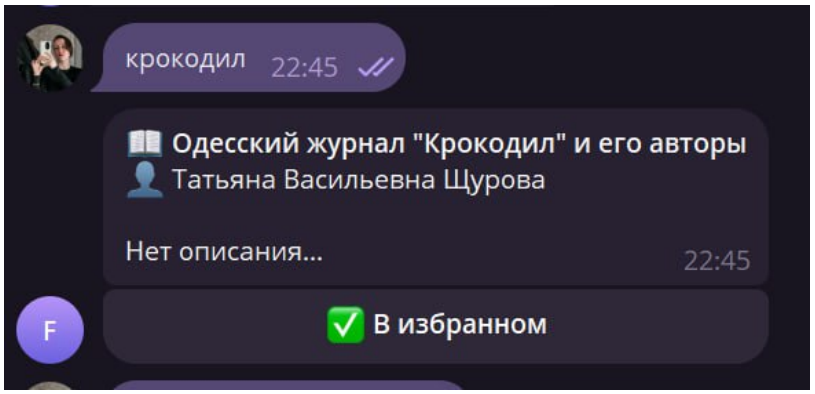
Начало работы с ботом



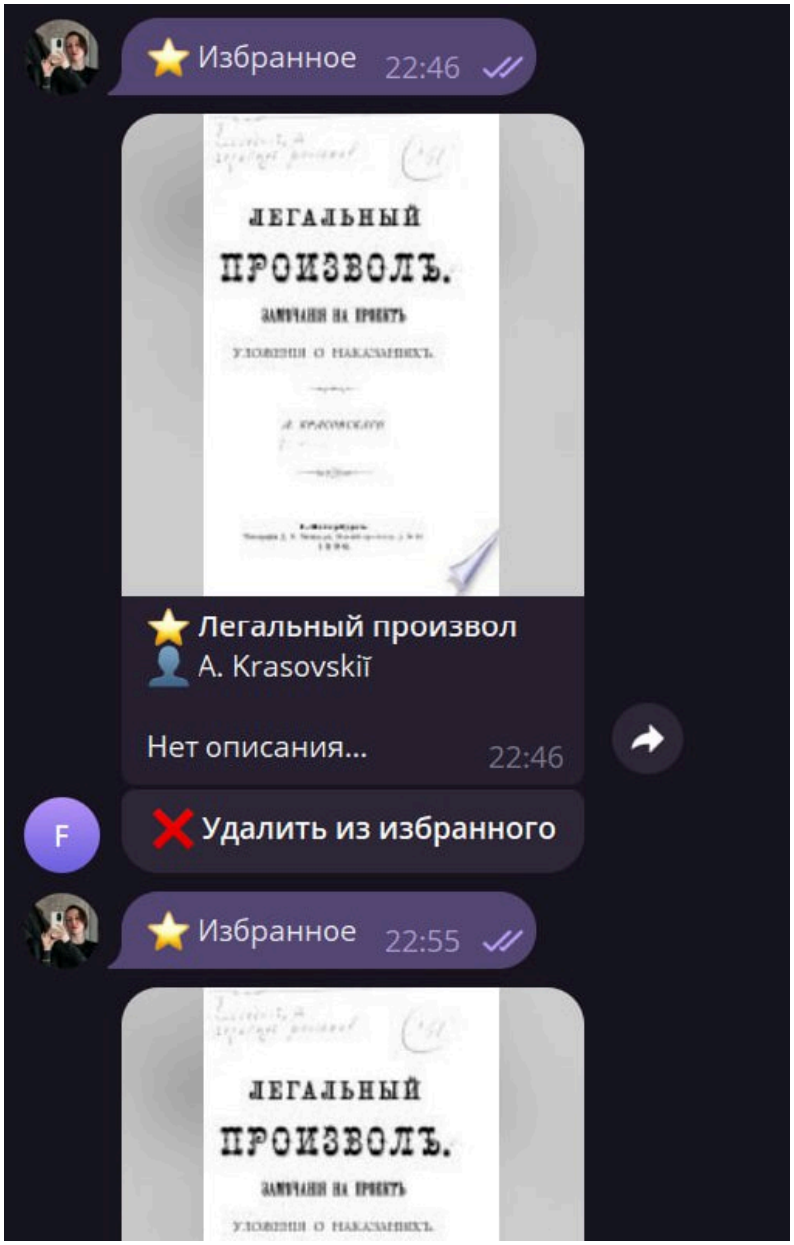
Так отображаются книги, найденные по запросу пользователя и не добавленные в избранное.



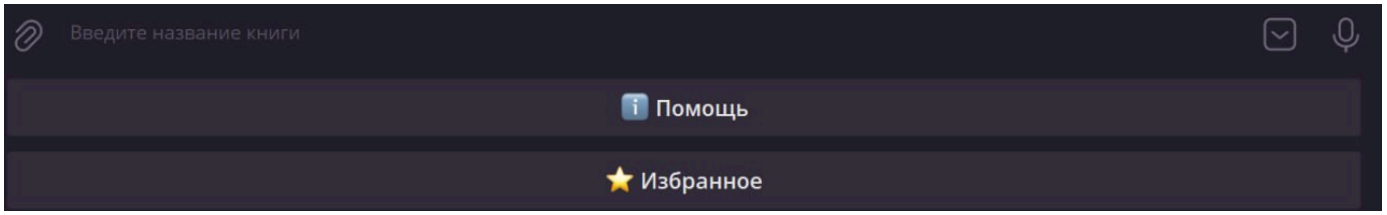
А так выглядят в чате книги по запросу, добавленные в избранное:



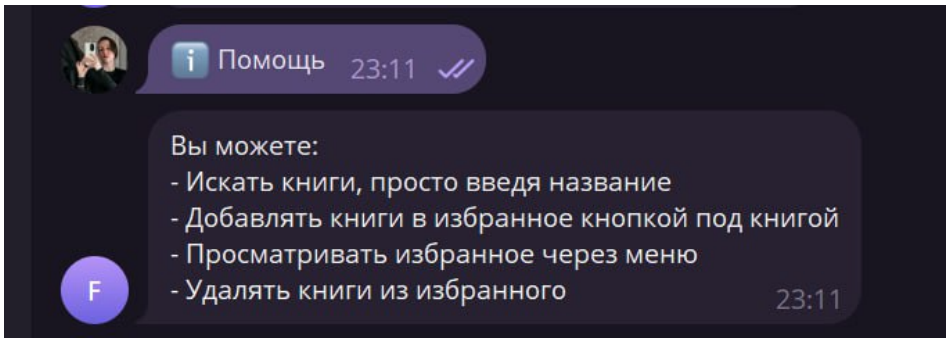
Избранное:



Кнопки для команд Помощь и Избранное:



Помощь:



Инструкция к боту

1. Создать в корневой папке репозитория файл .env и заполнить его по шаблону .env.template
2. Запустить нужные скрипты в зависимости от цели: Для запуска самого бота запустить скрипты scripts/build.sh, затем scripts/run.sh. Для запуска юнит тестов запустить скрипты scripts/build.sh, затем scripts/units.sh. Для запуска интеграционного теста запустить скрипты scripts/build.sh, затем scripts/integration.sh.