

Аудит Telegram-бота BookClub: архитектура и бизнес-процессы

Процесс регистрации пользователя

Описание: Регистрация новых участников клуба происходит с подтверждением секретной фразы. Пользователь стартует бота, бот создаёт профиль со статусом «inactive» и предлагает авторизоваться. Процесс авторизации пошаговый: пользователь вводит секретную фразу, после чего бот активирует аккаунт (статус «active») или отклоняет попытку при ошибке. Механизм включает ограничение попыток ввода и возможность отмены.

Flowchart (регистрация):

- **Старт /start:** Пользователь вызывает команду `/start`.
- *[Новый пользователь]* – Бот **создаёт профиль** (ID, имя, username) со статусом `inactive` и приветствует: «Добро пожаловать... пройдите `/register`».
- *[Существующий пользователь]* – Бот проверяет статус:
- Если `active` – приветствует вернувшегося участника: «Привет... `/help` для команд».
- Если `banned` – сообщает о блокировке: « Аккаунт заблокирован...» и логирует попытку входа заблокированного.
- Иначе (`inactive`) – приветствие как для нового: «С возвращением... `/register` для доступа».
- **Авторизация /register:** Пользователь запускает команду `/register` для авторизации.
- Бот **логирует начало** регистрации и получает профиль пользователя из базы.
- Если профиль найден:
- Статус `active` – регистрация не нужна: бот отвечает «Вы уже авторизованы!» и прекращает процесс.
- Статус `banned` – бот отвечает « Аккаунт заблокирован...», логирует попытку, процесс прерывается.
- Если профиль не найден (теоретически, пользователь не вызывал `/start`) – **! ошибка логики:** бот продолжит как при неактивном (см. оценку).
- Бот вызывает `security_manager.check_login_attempts()`:
- Если число попыток превышено – отвечает « Слишком много попыток. Попробуйте позже.» и отменяет регистрацию.
- Иначе – бот запрашивает « Введите секретную фразу...» и подсказку, переводит FSM в состояние `waiting_for_phrase`.
- **Ввод секретной фразы:** Пользователь отправляет фразу.
- Бот проверяет `security_manager.verify_secret_phrase(ввод, правильная_фраза)`.
- Если фраза неверная – бот отвечает « Неверная фраза, попробуйте ещё... `/cancel` для отмены», регистрирует неудачную попытку и остаётся в состоянии ожидания (можно попробовать снова).
- Если фраза правильная – бот вызывает `user_service.activate_user(user_id)`:
- Если активация успешна – бот сбрасывает счётчик попыток, **присваивает статус** `active`, отвечает « Авторизация успешна! `/help` для списка команд», затем FSM **сбрасывается** (конец процесса).
- Если активация не удалась – бот отвечает « Ошибка авторизации. Попробуйте позже.» (например, при сбое сохранения).
- **Отмена /cancel:** Пользователь может ввести `/cancel` в любой момент.




- Бот проверяет `FSMContext.get_state()`: если есть активный шаг – выполняет `state.clear()`, отправляет « *Операция отменена* » и завершает регистрацию. Если же никакой процесс не шёл – « *Нет активных операций для отмены.* ».

Оценка завершённости: Процесс регистрации в целом **реализован полностью**: предусмотрены создание нового аккаунта при `/start`, многоэтапная проверка при `/register`, защита от перебора фразы (брутфорс) и обработка отмены. Имеется мелкий недочёт: если вызвать `/register` до `/start`, бот не создаст профиль (отсутствует явное создание пользователя) – активация тогда вернёт `False` и пользователь получит « *Ошибка авторизации* » без явного решения. Впрочем, сценарий маловероятен (обычно Telegram первым делом отправляет `/start`). В остальном, логика регистрации завершена и соответствует задумке: новый пользователь не получит доступ без ввода корректной фразы, а заблокированный не сможет повторно активироваться.

Процесс взаимодействия с библиотекой книг

Описание: Данный процесс охватывает просмотр библиотеки, поиск книг и скачивание файлов. Авторизованный пользователь может запросить список всех книг или выполнить поиск по названию/автору. Бот отображает книги с интерактивными кнопками. Пользователь выбирает книгу, бот показывает карточку с описанием и форматами, после чего пользователь загружает файл книги в выбранном формате. Предусмотрен возврат к списку и обработка отсутствующих файлов или форматов. (Примечание: В коде есть два варианта реализации поиска и каталога — “simple” через FSM и улучшенный с хешированием; здесь описывается объединённая логика желаемого поведения).

Flowchart (поиск и просмотр книг):

- **Доступ к библиотеке:** Пользователь вводит команду `/library` (либо `/books`).
- Бот проверяет статус пользователя: если он **не активирован** – ответ « *Необходимо авторизоваться (/register)...* » и остановка.
- Если активен – бот **загружает список книг** (`book_service.get_all_books()` из `data/books.json`).
- Если книг нет – отвечает «  *Библиотека пуста.* ».
- Если книги найдены – бот формирует интерактивную клавиатуру: каждая книга как кнопка. В простом варианте `callback-data` содержит название (`"book:<Название>"`), в улучшенном – безопасный хеш названия (`"book_<hash>"`). Бот отправляет сообщение: « *Всего книг: N. Выберите книгу...* » с этой клавиатурой.
- **Выбор книги:** Пользователь нажимает кнопку книги (`callback "book:..." / "book_<hash>"`).
- Бот получает выбранное название книги (прямо или декодируя хеш через `safe_decode_title`) и загружает её данные.
- Если книга не найдена (например, удалена за время между шагами) – бот посылает ответ « *Книга не найдена!* ».
- Если найдена – бот формирует **карточку книги**:
 - Заголовок: название **жирным**; поля автора, описания, года добавляются, если присутствуют. Длинное описание укорачивается до ~300 символов.
 - Ссылки: если в данных книги есть URL на Яндекс.Книги или ЛитРес, бот добавляет строки «  *Яндекс.Книги: ...* », « *ЛитРес: ...* » и т.д.. Аудиоформат (если текстовое описание или ссылка на аудиокнигу) тоже включается.
 - Бот готовит клавиатуру для скачивания:
 - Собирает доступные форматы файлов из полей `fb2_file`, `epub_file`, `mobi_file` книги. Для каждого найденного формата добавляет кнопку «  *Скачать [FORMAT]* », с `callback "download:<Название>:fmt"` или `"download_<hash>_fmt"`.

- Добавляет кнопку «*Назад*», с соответствующим callback ("back_to_books" или "back_to_library") для возврата к списку.
- Бот обновляет сообщение на карточку книги с этой клавиатурой (использует edit_text, сохраняя историю чата чище).
- **Скачивание книги:** Пользователь нажимает кнопку скачивания (напр., "download:Название:epub").
- Бот обрабатывает колбэк, разделяет данные чтобы получить название и формат.
- Ищет книгу по названию; если не найдена – «*Книга не найдена!*», выход.
- Иначе проверяет, есть ли путь к файлу запрошенного формата в данных книги (book_info[<fmt>_file]).
- Если ключ отсутствует – отвечает «*Этот формат недоступен!*».
- Если путь есть – бот **валидирует и отправляет файл**:
- Проверяет, что путь начинается с допустимой папки (например, books/) и экранирует потенциально опасные символы (security_manager.sanitize_filename).
- Затем вызывает bot.send_document(chat_id, document=FSInputFile(path)) для отправки файла пользователю.
- Если отправка успешна – уведомляет через callback.answer("Файл отправлен!") и логирует операцию.
- Если файл отсутствует физически – «*Файл не найден на сервере!*» и логируется ошибка.
- При исключении сети/Telegram – ловит exception, логирует и сообщает «*Ошибка отправки файла!*» пользователю.
- **Возврат к списку:** Пользователь может вернуться к списку книг, нажав «*Назад*».
- Бот ловит колбэк "back_to_books" / "back_to_library".
- Заново получает актуальный список книг и обновляет сообщение обратно на список с кнопками. (Если книг уже нет – покажет «*Библиотека пуста*» вместо списка.)

- **Поиск книги:** Пользователь вводит команду /search для поиска по библиотеке.
- Бот проверяет авторизацию так же, как для /library: неавторизованным выдаёт отказ «*Для поиска нужно /register*».
- Если пользователь активен – бот запрашивает поисковый запрос, например: «🔍 *Введите название или автора...*» (с подсказками примерных запросов). Бот может либо включить FSM-состояние waiting_for_query, либо (в новой реализации) просто ждать текст без FSM.
- **Ввод поискового запроса:** Пользователь отправляет текст (не команду).
- Бот перехватывает сообщение (либо через StateFilter(BookSearchStates.waiting_for_query), либо любая несервисная текстовая реплика, если используется упрощённый режим).
- Бот нормализует запрос (trim, lower-case). Если длина < 2 символов – «🔍 *Введите минимум 2 символа...*» и ожидает повторного ввода.
- Бот выбирает из всех книг подмножество, где запрос встречается в названии, авторе или описании (регистронезависимо).
 - Если найдено хотя бы 1 книга – бот сообщает «🔍 *Найдено X книг:*» и выводит интерактивный список результатов (формирует клавиатуру так же, как список всей библиотеки, но только с найденными позициями).
 - Если ничего не найдено – ответ «*Книги не найдены. Попробуйте другой запрос.*».
- **Дальнейшие шаги совпадают с «Выбор книги»:** пользователь нажимает на книгу из результатов и просматривает/скачивает, либо может вернуться к общему списку книг.

Оценка завершённости: Бизнес-процесс работы с книгами в основном реализован и покрывает необходимые сценарии: просмотр библиотеки, чтение описания, поиск и скачивание. Навигация удобна (кнопки вперёд-назад). Важные проверки присутствуют: требование

авторизации перед доступом к книгам, обработка ошибок загрузки файлов, фильтрация пользовательского ввода (sanitize, ограничения в `security_manager`) и защита от неизвестных команд. Тем не менее, обнаружены **несоответствия в коде**, которые могут влиять на работу:

- **Дубликаты команд /search и /library:** В кодовой базе существуют две реализации этих команд (старая через FSM в `user_handlers.py` и новая в `library_handlers.py`), подключённые одновременно. Это может привести к непредсказуемому поведению: например, команда `/search` обрабатывается тем обработчиком, чей Router подключён раньше (в данном случае `library_router` имеет приоритет перед `user_router`). В результате FSM-сценарий поиска из `user_handlers` может никогда не выполняться, оставаясь мёртвым кодом. Аналогично, в справке бота указана команда `/library`, хотя в `user_handlers` используется `/books` – обе существуют, что путает пользователя. Эти логические дубли **следует объединить**, иначе возможны конфликты и лишние ответы.
- **Отсутствие некоторых частей диалога:** Пользователь не предупреждается явно, когда поиск активирован. В новой реализации бот после `/search` сразу ждёт текст, но не использует FSM – фактически любое текстовое сообщение (не команда) от активного пользователя будет воспринято как поисковый запрос. Это значит, что пока бот «ждёт» ввод, нет возможности отличить целевой ввод от случайного. Старый же вариант с FSM чётко ограничивал следующий ввод. **Рекомендация:** или вернуть FSM для поиска, или добавлять фильтр (флаг состояния поиска) при не-FSM реализации, чтобы не ловить вообще все сообщения чата.

В остальном, функционал просматривания/скачивания книг завершён: бот корректно реагирует на все ветви (нет книги, нет формата, неавторизован, пустой результат поиска), что подтверждает логическую завершённость процесса.

Процесс обсуждения (расписание встреч клуба)

Описание: Обсуждения книг в рамках клуба реализованы как **события-встречи** с определённой датой, временем и темой. Пользователь может посмотреть расписание предстоящих встреч командой `/schedule`. Бот выводит список ближайших событий, включая название встречи, дату, время и описание, если есть. Возможности записаться на встречу или обсуждать внутри бота не реализованы – предполагается, что встречи происходят вне бота (офлайн или в общем чате), а бот лишь информирует о них.

Flowchart (просмотр расписания):

- Пользователь вызывает команду `/schedule`.
- Бот проверяет статус пользователя: неавторизованным отказ («*Вы не авторизованы... /register*»), поскольку расписание доступно только членам клуба.
- Если пользователь активен – бот запрашивает из базы событий предстоящие встречи: `event_service.get_upcoming_events()`.
- Если встреч нет – бот отвечает: «*Встречи пока не запланированы.*».
- Если найдены – бот формирует сообщение «*Ближайшие встречи:*» и перечисляет каждое событие в формате:
 - **Название встречи** (или «Без названия», если не указано).
 - **Дата:** YYYY-MM-DD (либо «Не указана»).
 - **Время:** HH:MM (либо «Не указано»).
 - **Описание:** краткое описание встречи (если есть).




- После каждого события добавляется пустая строка для визуального разделения.
- Бот отправляет сформированный список пользователю (в режиме HTML для форматирования).

Оценка завершённости: Функциональность просмотра расписания встреч реализована *частично*. Пользователь может **получить информацию** о ближайших событиях, однако сам процесс «обсуждения» ограничен статичным выводом. Нет интерактивности: например, **не предусмотрено** подтверждение участия, вопросы к встрече или уведомления. Кроме того, **создание новых событий через бота не реализовано** – администратор не имеет команды для добавления встречи (разработан метод `create_event` в сервисе, но нет соответствующего обработчика). Таким образом, процесс обсуждений пока **неполноценен**: бот выступает как электронное расписание, но не площадка для коммуникации. С точки зрения логики ошибок в выводе расписания не обнаружено – сценарий прост, и при отсутствии событий ответ информативен. Для повышения полезности следует развить этот модуль (добавить команды для админа: создание/редактирование событий, а также участие пользователей, напоминания и т.п.).

Процессы модерации и администрирования

Описание: Административный функционал предоставляет модераторам и администраторам средства управления клубом: панель админа с разделами, управление пользователями (блокировка, роли, теги), управление библиотекой (добавление книг, редактирование ссылок), просмотр статистики и логов. Логика администрирования распределена по командам `/admin`, `/settag`, `/setrole`, `/ban`, `/users` и др., а также множеству callback-обработчиков для кнопок административного меню. Ниже приведены ключевые потоки административных процессов.

Панель администратора (основное меню)

- **Доступ:** Привилегированный пользователь (ID в списке `ADMIN_IDS .env`) вводит команду `/admin`.
- Декоратор `@admin_required` проверяет права: если вызвал не админ – бот отвечает « *У вас нет прав администратора!* » и игнорирует команду.
- Если права есть – бот логирует действие и отображает **панель**: отправляет сообщение « *Панель администратора – Выберите действие:* » с **inline-клавиатурой**. Клавиатура содержит кнопки:
 -  **Статистика** (callback `admin_stats`) – обзор общей статистики клуба;
 - **Пользователи** (callback `admin_users`) – управление участниками;
 -  **Управление книгами** (callback `admin_books`) – управление библиотекой;
 - **Управление событиями** (callback `admin_events`) – управление встречами клуба.
- **Просмотр статистики:** При нажатии  **Статистика** бот обрабатывает `admin_stats`:
 - Проверяет права (админ?) — если нет, через `callback.answer` уведомляет « *Нет прав!* ».
 - Если да – бот собирает показатели:
 - `user_service.get_user_stats()` – общее число пользователей, активных, новых сегодня и т.д..
 - `book_service.get_all_books()` – количество книг в библиотеке.
 - `event_service.get_event_stats()` – всего событий и предстоящих.

- Формирует текст отчёта, например:

```

III Статистика клуба:
  Пользователи: 123 (активных: 100)
  Новых сегодня: 2
  Книг в библиотеке: 45
  Событий: 5 (предстоящих: 1)
  
```

- Обновляет текущее сообщение панели на этот текст (без клавиатуры). (Замечание: После просмотра статистики у пользователя не остаётся кнопки «Назад», приходится снова вызвать /admin — небольшое упущение UX).

- **Возврат в меню:** В некоторых под-разделах предусмотрена кнопка `Назад` (callback `admin_back`). По её нажатию бот заново выводит главное меню панели (редактирует сообщение на исходный текст « *Панель администратора...*» с начальными кнопками). Это реализовано, например, после списка пользователей или при отмене добавления книги. В статистике и списке событий такой кнопки нет (см. выше).

Управление пользователями (модерация участников)








Администратор может просматривать список пользователей, фильтровать по статусам, просматривать профиль каждого и применять меры: блокировка/разблокировка или изменение роли/тегов.




- **Список пользователей:** При нажатии `Пользователи` (`admin_users`):
 - Бот проверяет права (админ?) – иначе « *Нет прав!*».
 - Логирует действие и получает всех пользователей через `user_service.get_all_users()`.
 - Если база пуста – « *Пользователи не найдены*».
 - Иначе – бот формирует текст: статистику (всего, активных, заблокированных) и запрос «*Выберите фильтр:*».
 - Добавляет клавиатуру фильтров (`create_users_filter_keyboard()`): кнопки «*Все*», «*Активные*», «*Неактивные*», «*Заблокированные*», «*Новые (неделя)*», «*С тегами*» и опции «*Поиск*», «*III Экспорт*», « *Назад*».

- Бот отправляет сообщение со списком (или обновляет панель админа на этот список).

- **Фильтрация списка:** При выборе одного из фильтров (`users_filter_*`):

- Бот снова проверяет права (если вдруг нажал неадмин) – нет прав -> « *Нет прав!*».
- Определяет тип фильтра из callback (`filter_type` после префикса) и отбирает соответствующих пользователей из полного списка:
 - `all` – всех;
 - `active` / `inactive` / `banned` – сравнивает поле `status`;
 - `new` – тех, у кого `first_interaction` в последние 7 дней;
 - `tagged` – у кого непустой список `tags`.
- Если отфильтрованный список пуст – через `callback.answer` уведомляет: «... *не найдены*».

- Иначе – бот собирает пагинированный список: отображает первые 5 пользователей этого списка с кнопками.
 - Формирует клавиатуру (`create_users_list_keyboard`): по каждому пользователю кнопка `[статус_иконка] Имя` (имя обрезается до 20 символов), callback `user_detail_<user_id>` для детализации.
 - Добавляет навигацию страниц: «/» при наличии >5 пользователей, и индикатор страницы `1/total`.
 - Добавляет кнопки управления: « *Фильтры*» (возврат к выбору фильтра, callback `users_filters`) и « *Назад*» (в главное админ-меню, callback `admin_back`).
- Бот **обновляет сообщение** списком с текущим фильтром (заменяет текст на «*Найдено X пользователей... Список:*») и новой клавиатурой.
- **Пагинация списка:** Если есть несколько страниц, админ может нажать «» или «» (`users_page_<filter>_<page>`):
 - Бот проверяет права (админ?) – иначе « *Нет прав!*», выход.
 - Если нажата кнопка с номером (`users_page_info`) – бот просто `answer()` с подсказкой « *Информация о страницах*» (в данный момент без другой логики).
 - Иначе бот парсит `filter_type` и `page` из callback, аналогично фильтру снова отбирает нужных пользователей и формирует клавиатуру для указанной страницы.
- Обновляет сообщение списка на новую страницу (текст + кнопки).
- **Детали пользователя:** Админ выбирает конкретного пользователя (нажимает кнопку списка `user_detail_ID`).
 - Бот проверяет права – при отсутствии « *Нет прав!*».
 - Получает `user_id` и загружает профиль из базы (`user_service.get_user(user_id)`).
 - Если не найден – « *Пользователь не найден*» через `callback.answer`.
 - Если найден – формирует подробную карточку:
 - **ID** (в виде `<code>ID</code>` для копирования).
 - **Имя** (или «Не указано») и **Username** (кликабельный `@username` или «Не указан»).
 - **Дата регистрации, Статус, список Тегов** (или «Не указаны»), а также **Роль** пользователя.
 - Если есть `activated_at` – добавляет строку, когда активирован.
 - Логирует просмотр деталей и выводит сообщение с профилем.
- Прикрепляет клавиатуру управления данным пользователем:
 - Кнопки: « *Заблокировать*» (`user_ban_ID`), « *Разблокировать*» (`user_unban_ID`).
 - « *Роли*» (`user_roles_ID`) – перейти к управлению ролью.
 - « *Добавить тег*» (`user_addtag_ID`) – добавить новый тег.
 - « *Активность*» (`user_activity_ID`) – посмотреть активность пользователя.
 - « *Назад к списку*» (`users_filters`) – вернуться к предыдущему списку (фильтры).
- **Блокировка/разблокировка:** При нажатии « *Заблокировать*» (`user_ban_ID`) или « *Разблокировать*» (`user_unban_ID`):
 - Бот проверяет права (админ?) – иначе « *Нет прав!*».
 - Выделяет `user_id` из callback.

- Вызывает `user_service.ban_user(id)` или `unban_user(id)` – обновляет статус пользователя в базе на `"banned"` или `"active"`.
- Если операция успешна:
 - Логирует действие (кто заблокировал кого).
 - Через `callback.answer` показывает краткое подтверждение: « *Пользователь заблокирован*» или «... *разблокирован*».
 - Обновляет сообщение **деталей пользователя**: переформировывает профиль с новым статусом (иконка меняется на `ban` или `unban`). Клавиатуру тоже обновляет (кнопки остаются те же, т.к. можно повторно нажать заблокировать/разблокировать — хотя повторная блокировка заблокированного по логике не нужна, кнопка не деактивируется явно). (На практике, можно было бы убирать или менять кнопки после изменения статуса, но бот этого не делает — небольшое упущение UI).
- Если операция не удалась (например, проблемы сохранения) – через `callback.answer` сообщает « *Ошибка блокировки/разблокировки пользователя*», сообщение профиля остаётся прежним.
- **Изменение роли:** В детали профиля админ нажимает « Роли» (`user_roles_ID`).
 - Бот проверяет права – если нет, « *Нет прав!*», выход.
 - Загружает профиль пользователя. Если не найден – « *Пользователь не найден*».
 - Определяет текущую роль (`user` / `moderator` / `admin`) и формирует текст меню ролей: «Текущая роль: ... Доступные роли: ... Выберите новую роль:».
 - Формирует клавиатуру с кнопками для каждой роли:
 - « *Пользователь*», « Модератор», « *Администратор*». Текущая роль помечается ☐ на кнопке. Callback имеет вид `user_setrole_ID_user/moderator/admin`.
 - « *Назад к пользователю*» (возврат к деталям профиля).
 - Редактирует сообщение профиля на меню изменения роли с указанными кнопками.
 - При выборе новой роли (`user_setrole_ID_role`):
 - Бот проверяет админ-права – нет -> « *Нет прав!*».
 - Парсит `user_id` и `new_role` из callback, получает текущие данные пользователя.
 - **Защита от ошибок:** если пользователя нет – « *Пользователь не найден*». Если админ пытается снять права с **самого главного администратора** (ID которого в `ADMIN_IDS` .env), и новая роль != `admin` – бот отклоняет: « *Нельзя изменить роль главного администратора!*» (важная защита, чтобы никто из админов не мог случайно разжаловать создателя бота).
- Иначе вызывает `user_service.set_user_role(user_id, role)`:
 - Если успех – бот логирует изменение и формирует уведомление: « *Роль успешно изменена! ... old_role → new_role*».
 - Обновляет клавиатуру: предлагает « *Назад к ролям*» (повторно настроить роль) и « *К пользователю*» (вернуться в профиль).
 - Редактирует сообщение на это уведомление.
 - Если не удалось – через `callback.answer` сообщает « *Ошибка при изменении роли!*». (Например, неверная роль – но такие отфильтрованы заранее, либо ошибка сохранения на диск.)
- **Добавление тега:** В деталях профиля при нажатии « Добавить тег» (`user_addtag_ID`):
 - Бот проверяет права – иначе « *Нет прав!*».

- Ищет пользователя; если не найден – « *Пользователь не найден*».
- Если найден – бот формирует сообщение: «*Добавление тега... Пользователь: Имя, текущие теги: [список]... Отправьте новый тег для добавления. Для отмены – /cancel*».
- Обновляет профиль на этот текст (режим HTML), без специальных кнопок (т.к. ожидает текст).
- ⚠ **Ограничение:** В коде видно, что после этого шага FSM **не устанавливается** (написано `# TODO: Добавить FSM`). То есть бот просто ждёт текстового сообщения с тегом, но не отслеживает явно состояние «добавления тега». Пользователь должен либо отправить тег, либо отменить через общую команду `/cancel`. **Дальнейшая логика не реализована** – нет обработчика, принимающего введенный тег. Поэтому на практике админ увидит сообщение «Отправьте новый тег...», пошлёт текст, и... бот его не обработает (скорее всего перехватит как неизвестную команду или поисковый запрос, в худшем случае). Эта функция задокументирована как «*в разработке*».
- **Просмотр активности пользователя:** Нажатие «*Активность*» (`user_activity_ID`):
 - Бот проверяет права – иначе « *Нет прав!*».
 - Загружает пользователя; если нет – « *Пользователь не найден*».
 - Формирует (пока статичный) отчёт об активности: показывает дату регистрации, статус, первое взаимодействие, дату активации.
 - Добавляет приписку: «*Детальная аналитика будет добавлена в будущих версиях.*» (т.е. пока без реальных метрик сообщений).
 - Клавиатура: « *Назад к пользователю*» (возвращает к деталям профиля).
 - Редактирует сообщение на этот отчёт.
- **Поиск пользователя:** В списке пользователей есть кнопка «*Поиск*» (`users_search`).
 - Бот проверяет права – иначе « *Нет прав!*».
 - Обновляет сообщение на: «*Поиск пользователей... Отправьте имя или @username... Для отмены /cancel*».
 - И сразу через `callback.answer` сообщает «*Функция поиска в разработке*». По сути, этот раздел тоже не реализован: нет FSM или хендлера, который бы ловил введенный после этого текст и фильтровал пользователей. Админу показывается инструкция, но поиск фактически не работает.
- **Экспорт пользователей:** Кнопка «*Экспорт*» (`users_export`):
 - Бот проверяет права.
 - Собирает всех пользователей; если никого нет – « *Нет пользователей для экспорта*» через `callback.answer`.
 - Если есть – формирует CSV-строку: заголовки (ID, Имя, Username, Статус, Дата регистрации, Теги) и строки по каждому пользователю. Особенность – любые запятые в полях имени/username заменяются на `;`, чтобы CSV не поломался.
 - Для корректного отображения русских букв в Excel – добавляет BOM в начало байтовой строки и кодирует как UTF-8 (с сигнатурой).
 - Отправляет файл пользователю методом `answer_document` с параметром `BufferedInputFile(...)` (отправка содержимого как файла с именем `users_export_YYYYMMDD_HHMMSS.csv`).

- Отвечает через `callback.answer(" Экспорт завершен")` и логирует действие.

Управление библиотекой (книги и контент)

Администратор может добавлять новые книги в библиотеку двумя способами, а также редактировать ссылки (метаданные) существующих книг. Эти процессы используют FSM для пошагового ввода данных.

- **Список книг (админ):** При нажатии кнопки «📖 Управление книгами» (`admin_books`):
 - Проверка прав (админ?) – нет -> « *Нет прав!*».
 - Логирует действие и получает все книги (`book_service.get_all_books()`).
 - Если библиотека пуста – выводит «📖 Книг пока нет».
 - Если книги есть – формирует текст: перечень книг (название и автор каждую с новой строки). (Каждая запись разделяется пустой строкой.)
 - Обновляет сообщение панели админа на этот список (без клавиатуры). *(Здесь снова нет кнопки «Назад» – чтобы вернуться, админ должен повторно вызвать /admin или воспользоваться старым сообщением панели, если оно не заменено.)*
- **Примечание:** Список книг в админ-панели **только для чтения** – нажатия по ним не предусмотрены. Добавление/удаление книг осуществляется отдельными командами, а не через эту клавиатуру.

• Добавление новой книги вручную (/addbook):

- Администратор вводит команду `/addbook`. Декоратор `@admin_required` проверяет права (иначе « *Нет прав!*»).
- Бот начинает FSM (шаг 1) для пошагового ввода информации. Логирует событие «начало добавления книги».
- Отправляет сообщение: «📖 Добавление новой книги – Шаг 1/4: введите название книги...
Примеры: ...», прикрепляет клавиатуру с кнопкой « Отмена» (`callback.cancel_addbook`).
- **Шаг 1 – Название:** Пользователь-админ присылает название книги (текстовое сообщение).
 - Бот получает `message.text`, проверяет минимальную длину (≥ 2 символов) и не слишком ли длинное (< 200 символов).
 - Если не проходит – бот просит повторить: « *Название слишком короткое/длинное... попробуйте ещё.*».
 - Проверяет, нет ли уже книги с таким названием (`book_service.get_book(title)`): если уже существует – « *“Название” уже есть... Введите другое название.*».
 - Если всё хорошо – сохраняет `title` во временные данные FSM (`state.update_data(book_title=...)`), переводит FSM в состояние `waiting_for_author` (шаг 2).
 - Отправляет сообщение: « *Название: [title]... Шаг 2/4: Введите автора книги...*
Примеры: ...», с клавиатурой « Отмена».
- **Шаг 2 – Автор:** Админ присылает имя автора.
 - Бот проверяет длину (≥ 2 и ≤ 100 символов).
 - Если невалидно – сообщение « *Имя автора слишком короткое/длинное...*», запрос повторить.
 - Если ок – сохраняет `author` в FSM, ставит состояние `waiting_for_description` (шаг 3).

- Отправляет сообщение: « *Название: X, Автор: Y... Шаг 3/4: Введите описание книги (или пропустите)...* ».
 - Клавиатура: «**»» Пропустить**» (`skip_description`) чтобы не вводить описание, и « *Отмена* » для отмены.
 - Примеры описания приводятся (несколько фраз) для ориентира.
- **Шаг 3 – Описание (опционально):**
- Если админ нажал «**»» Пропустить**» – бот сразу переходит к Шагу 4 (см. далее).
 - Иначе, админ отправляет текст описания. Бот принимает `message.text` .
 - Если описание >1000 символов – « *Описание слишком длинное (макс 1000)...* », просит повторить.
 - Иначе – сохраняет `description` (может быть пустой строкой) и ставит состояние `waiting_for_file_path` (шаг 4).
 - Шлёт сообщение: « *Название: X, Автор: Y, Описание: Z... Шаг 4/4: Введите путь к файлу книги (или пропустите)...* Поддерживаемые форматы: EPUB, FB2, MOBI...
Примеры: books/...», клавиатура: «**»» Пропустить**» (`skip_file`) и « *Отмена* ».
- (Примечание: Путь к файлу – имеется в виду относительный путь на сервере бота, где уже загружены файлы. Это требует, чтобы админ предварительно поместил файл на сервер, либо мог использовать эту команду на компьютере с доступом к файловой системе.)
- **Шаг 4 – Файл книги (опционально):**
- Если админ нажал «**»» Пропустить**» (`skip_file`):
 - Бот загружает из FSM `title` , `author` , `description` , вызывает `book_service.add_book(title, author, description)` **без файлов**.
 - Если добавление успешно – логирует действие и формирует финальное сообщение: « *Книга успешно добавлена! ... Форматы: Не указаны* » (так как файл не прикреплен).
 - Клавиатура: «**📖 Добавить ещё книгу**» (`add_another_book`) и « *К панели администратора* » (`admin_back`).
 - Бот редактирует сообщение шага 4 на это финальное сообщение.
 - Если добавление не удалось – « *Ошибка при добавлении книги! Попробуйте ещё или к администратору.* » (по сути, администратору показывается, что что-то пошло не так при сохранении).
 - FSM состояние очищается (`state.clear()`) в конце.
 - Иначе, если админ отправил путь (например, `books/filename.epub`):
 - Бот принимает строку `file_path` . Если введено 'пропустить' (на русском) – трактует как пропуск (превращает в пустую строку).
 - Если указан путь: бот проверяет расширение:
 - `.epub` / `.fb2` / `.mobi` – определяет формат и сохраняет путь в словарь `files` соответствующего ключа.
 - Если расширение другое – « *Неподдерживаемый формат... (поддерживаются EPUB, FB2, MOBI)* », просит ввести заново либо написать «пропустить».
 - После валидации собирает все данные из FSM: `title` , `author` , `description` и dict `files` (может содержать от 0 до 3 путей).
 - Вызывает `book_service.add_book(title, author, description, files)` :
 - Эта функция добавляет запись в `books.json` со всей информацией. В коде она также выполняет валидацию обязательных полей (название, автор) через `data_manager.validate_book_data` .
 - Если запись успешно сохранена – бот логирует «*добавление книги: [title]*».
 - Формирует сообщение успеха:
 - « *Книга успешно добавлена! Название: ..., Автор: ..., Описание: ...* » (обрезает описание до 200 символов для компактности).

- Если какие-либо файлы были указаны – перечисляет форматы: « *Форматы: EPUB, FB2...*» (список ключей из `files`, переведённых в верхний регистр). Если `files` пуст – пишет «Не указаны».
 - Добавляет кнопки: «📖 *Добавить ещё книгу*» и « *К панели администратора*».
 - Отправляет сообщение (новое, а не edit, т.к. ранее бот ожидал сообщение от пользователя, а не редактировал своё – здесь реализация через `message.answer()`).
 - Если добавление книги вернуло False (ошибка записи файла) – бот сообщает « *Ошибка при добавлении книги! Попробуйте ещё...*».
 - Очищает FSM состояние.
- **Отмена добавления:** Если на любом шаге (1–4) админ нажимает кнопку « *Отмена*» (`cancel_addbook`):
- Бот выполняет `state.clear()` (сбрасывает FSM).
 - Редактирует текущее сообщение на « *Добавление книги отменено*», с клавиатурой « *К панели администратора*» (возвращение к меню).
- **Добавить ещё книгу:** Если после успешного добавления админ нажимает «📖 *Добавить ещё книгу*» (`add_another_book`):
- Колбэк-обработчик просто вызывает `cmd_addbook(callback.message, state)` повторно. То есть, бот повторно запускает FSM добавления книги, начиная с шага 1, в том же чате. Это удобно для пакетного ввода нескольких книг подряд.
- **Добавление книги из ZIP (/uploadbook):**
Администратор может загрузить книгу пакетно, отправив ZIP-архив с файлом FB2 (и бот автоматически извлечёт метаданные и создаст все форматы). Процесс более сложный, включая скачивание файла от Telegram, конвертацию через внешний инструмент (Calibre) и загрузку файлов обратно в облако Telegram.
- Админ вводит команду `/uploadbook`. Требуется админ-права (иначе отказ).
- Бот логирует «*начало загрузки книги из ZIP*», устанавливает FSM состояние `waiting_for_zip_file`.
- Отправляет инструкцию: «📖 *Загрузка книги из ZIP... Шаг 1: отправьте ZIP-архив с FB2-файлом. Требования: ... Максимум 50MB. Что произойдёт: ...*». Прикреплена клавиатура « *Отмена*» (тот же `cancel_addbook`). (То есть используется одна кнопка отмены для обоих сценариев добавления).
- **Ожидание архива:** Админ отправляет файл (Document) в чат.
- Бот **проверяет тип сообщения:** если `message.document` отсутствует (т.е. админ отправил не файл) – « *Пожалуйста, отправьте ZIP-архив...*».
 - Проверяет размер: `message.document.file_size`. Если >50MB – « *Файл слишком большой! (макс 50MB)*».
 - Проверяет имя файла: должно оканчиваться на `.zip` – иначе « *Файл должен быть ZIP!*».
 - Если все проверки пройдены:
 - Бот отвечает «📖 *Скачиваю ZIP-архив...*» (уведомляет, что начал процесс).

- Вызывает `message.bot.get_file(file_id)` для получения пути файла на серверах Telegram, затем создаёт локальное имя `temp/filename.zip` и через `message.bot.download_file(file_path, dest_path)` скачивает архив во временную папку `temp`. (Эти операции выполняются последовательно; в коде они `await` – т.е. асинхронно, но фактически могут занять время. Бот не блокирует другие команды, так как это внутри своего handler. Однако `download_file` может быть ресурсоёмким.)
- После скачивания – бот отвечает «📄 Извлекаю FB2 файл из архива...».
- Вызывает `fb2_parser.extract_fb2_from_zip(zip_path)` – утилита, распаковывающая FB2 из архива.
 - Если не удалось найти FB2 внутри – бот отвечает « Не удалось извлечь FB2... убедитесь, что ZIP содержит FB2 файл.» и на этом заканчивает обработку. (FSM остаётся в ожидании файла? Код при return не делает clear – возможно, стоит вручную ввести /cancel, но бот этого не делает: **упущение**, т.к. state не очищено. Однако, можно повторно прислать правильный файл — FSM всё ещё ждёт `waiting_for_zip_file`.)
- Если FB2 извлечён (получен путь `fb2_path`): бот отвечает « Парсинг метаданных FB2 файла...».
- Вызывает `fb2_parser.parse_fb2_metadata(fb2_path)` – парсер формата FB2: извлекает заголовок, автора, год, жанры, описание и др.
 - Если `metadata` не получилось (например, FB2 без необходимых тегов) – бот « Не удалось извлечь метаданные... Убедитесь, что файл содержит корректные данные.» и завершает обработку.
- Проверяет ключевые поля в метаданных: `title` и `author`. Если их нет – « В FB2 файле отсутствуют обязательные метаданные: название и автор.», процесс прерывается.
- Проверяет, нет ли уже книги с таким названием в библиотеке (`book_service.get_book(title)`) – если есть, бот сообщает « “Название” уже существует в библиотеке!» и прерывает процесс (чтобы не дублировать книги).
- Затем – **конвертация форматов**:
 - Бот убеждается, что установлен конвертер Calibre: `book_converter.check_calibre_installed()`. Если Calibre недоступен:
 - Вызывает `book_converter.get_installation_instructions()` – получает инструкции по установке (в коде вероятно заранее прописанные для Windows/Linux).
 - Отвечает « Calibre не найден! Для конвертации установить Calibre: [инструкции]... После установки перезапустите бота.». Процесс на этом завершается без очистки FSM (admin должен установить софт и повторить /uploadbook заново).
 - Если Calibre есть – бот сообщает « Конвертирую в EPUB и MOBI...».
 - Вызывает `book_converter.convert_book_formats(fb2_path)`, который с помощью CLI утилиты `ebook-convert` генерирует файлы EPUB и MOBI из исходного FB2. Возвращается словарь `converted_files` с путями полученных файлов (например, `{ 'fb2': ..., 'epub': ..., 'mobi': ... }`). (В текущей реализации `convert_book_formats` – синхронная операция с `subprocess`, она не вынесена в отдельный поток. Это может временно подвесить бота, если конвертация долгая, но т.к. ожидается, что команда редко используется, можно считать приемлемым. В идеале – сделать асинхронно.)
 - По завершении конвертации – бот говорит «📁 Загружаю файлы в Telegram...».

- Вызывает `telegram_uploader.upload_book_formats(bot, converted_files, chat_id)`: эта утилита отправляет сгенерированные файлы (EPUB/MOBI) **обратно администратору** через бот, видимо, чтобы Telegram присвоил им `file_id` и их можно было легко расшарить. Функция возвращает `telegram_file_ids` – словарь или список `file_id` загруженных документов. *(Это сделано, вероятно, для кэширования файлов в облаке Telegram, но прямого использования `file_id` в коде не прослеживается, возможно, на будущее.)*
 - Бот сообщает « Сохраняю в базу данных...».
 - **Сохранение книги:** Формирует расширенные метаданные: `description` (из FB2), `year`, `genres`, `created_date` и `telegram_file_ids` и т.п. Собирает итоговые файлы (FB2, EPUB, MOBI) и вызывает `book_service.add_book(title, author, description, files, extended_metadata)`.
 - Если добавление прошло успешно – бот формирует сообщение « Книга успешно добавлена! Название: ..., Автор: ...». Добавляет: год (если есть) и жанры. Перечисляет форматы, которые были сгенерированы: «*Форматы: FB2, EPUB, MOBI*». Дополнительно отмечает: «📁 Файлы загружены в Telegram для шаринга.».
 - Клавиатура: «📁 Добавить ещё книгу» и « К панели администратора» (как и при ручном добавлении).
 - Отправляет сообщение успеха администратору, логирует операцию «добавление книги из ZIP: [title]».
 - Если сохранение книги вернуло False – бот сообщает « Ошибка при сохранении книги в базу данных! Попробуйте ещё...» (маловероятно, но на случай проблем с файлом JSON).
 - **Завершение:** Бот удаляет временные файлы: сам ZIP, FB2 и сгенерированные EPUB/MOBI (через `fb2_parser.clean_temp_files`). Очищает FSM `state.clear()`.
 - В случае любой непоиманной ошибки в процессе (например, исключение) – попадает в `except Exception as e`: бот логирует ошибку и сообщает « Произошла ошибка при обработке файла! Попробуйте ещё...», затем очищает состояние FSM.
- **Редактирование ссылок на книгу (/editlinks):**
 Эта команда позволяет администратору изменить или добавить ссылки в карточке книги (внешние ресурсы: Яндекс.Книги, ЛитРес, аудио-формат). Процесс использует FSM для ввода каждой ссылки.
- Админ вводит `/editlinks`. Права проверяются (декоратор `@admin_required`).
 - Бот логирует «редактирование ссылок на книги», получает все книги как словарь `books`.
 - Если библиотека пуста – отправляет «📁 Нет книг для редактирования. Сначала добавьте книги.» (HTML).
 - Если есть книги – формирует клавиатуру: каждая книга -> кнопка с обрезанным названием (макс 30 символов + "...") и callback `editlinks_<hash>` (название кодируется в 16-символьный хеш, как и в `/library`).
 - Отправляет сообщение: « Редактирование ссылок на книги... Всего книг: N. Выберите книгу:» с этой клавиатурой.

• **Выбор книги:** Админ нажимает на книгу (callback `editlinks_<hash>`).

- Бот декодирует hash в оригинальный title (`safe_decode_title`).
- Если по каким-то причинам книга не найдена – « *Книга не найдена!*» через `callback.answer`.
- Если найдена – бот формирует текст « *Ссылки для книги: [Название]*» со списком текущих ссылок:
- *Яндекс.Книги*: URL или «Не указана»,
- *ЛитРес*: URL или «Не указана»,
- *Аудиоформат*: текстовое описание или «Не указан».
- Клавиатура: кнопки «📖 *Яндекс.Книги*» (`link_yandex_<hash>`), « *ЛитРес*» (`link_litres_<hash>`), «🔊 *Аудиоформат*» (`link_audio_<hash>`), и « *Назад*» (`back_to_editlinks`) для возврата к списку книг.
- Бот редактирует сообщение на список ссылок с этой клавиатурой.

• **Редактирование конкретной ссылки:**

- Если админ нажал «📖 *Яндекс.Книги*» (`link_yandex_<hash>`):
- Бот переходит в FSM состояние `BookLinkStates.waiting_for_yandex_url`.
- Сохраняет `book_title` в данные FSM.
- Редактирует сообщение: «📖 *Редактирование ссылки Яндекс.Книги*\n*Книга: [Название]*\n\n*Отправьте ссылку или 'нет' для удаления.*».
- Аналогично, « *ЛитРес*» (`link_litres_<hash>`) -> состояние `waiting_for_litres_url`, сообщение « *Редактирование ссылки ЛитРес...*».
- «🔊 *Аудиоформат*» (`link_audio_<hash>`) -> состояние `waiting_for_audio_format`, сообщение «🔊 *Редактирование аудиоформата...*\n*отправьте информацию или 'нет' для удаления.*».
- В любом случае, после нажатия одной из этих кнопок бот ждёт сообщения с новым значением ссылки.

• **Ввод нового значения:** Администратор присылает текст (URL или описание, либо слово 'нет'):

- Обработчик `@router.message(StateFilter(BookLinkStates.*))` ловит сообщение в соответствующем состоянии:
- Для `waiting_for_yandex_url`:
 - Бот достаёт из FSM название книги, берёт `message.text` как `url`. Если `url.lower() == 'нет'` – заменяет на `""` (пустую строку).
 - Загружает актуальный `books = book_service.get_all_books()`. Если книга есть в словаре – обновляет поле `yandex_books_url` на новое значение (даже пустое).
 - Вызывает `data_manager.save_json(config.database.books_file, books)` для сохранения базы.
 - Отправляет админу сообщение: « *Ссылка обновлена! Книга: [Название], Яндекс.Книги: [url или 'Удалена']*».
 - Очищает FSM `state.clear()` (конец под-процесса).
- Для `waiting_for_litres_url`: аналогично, обновляется `litres_url` книги, ответ « *Ссылка обновлена! ... ЛитРес: [новое или 'Удалена']*», FSM clear.

- Для `waiting_for_audio_format`: обновляется поле `audio_format` (текстовое описание аудиокниги, или пустое), ответ « *Аудиоформат обновлён! ... Аудио: [новое или 'Удален']*», FSM clear.
- Если по каким-то причинам книга не найдена (например, была удалена параллельно) – бот на этапе сохранения выдаст « *Книга не найдена!*», но это крайне маловероятно при локальной JSON базе без конкуренции.
- На этом редактирование конкретного поля завершено. (*Бот не возвращает автоматически в меню ссылок данной книги – админу придётся вручную снова вызвать /editlinks или нажать «Назад к библиотеке», см. ниже.*)

• **Назад к выбору книг:** Если в меню ссылок книги админ нажимает « *Назад*» (`back_to_editlinks`):

- Бот получает текущий список книг (как при `/editlinks`).
- Если список пуст – редактирует сообщение: «👉 *Нет книг для редактирования...*».
- Иначе – формирует клавиатуру со всеми книгами, аналогично первому шагу, и редактирует сообщение обратно на « *Редактирование ссылок... Всего книг: N...*».
- Админ может выбрать другую книгу для редактирования или отменить команду, нажав `/cancel` (которая просто не найдёт активного FSM и ответит «*Нет активных операций*»).

Защита от спама и прочие админ-инструменты

В боте реализована **middleware от спама**: класс `SpamProtectionMiddleware`, подключённый в диспетчер для всех входящих сообщений и колбэков. Его логика: если пользователь заблокирован (`status == banned`), ограничивать частоту его сообщений по жёстким правилам (не более 3 в минуту, 10 в час, 30 в день). При превышении – дальнейшие сообщения игнорируются и пользователь временно **блокируется на отправку**: на 5 минут (за >3 сообщений/мин), 1 час, либо 6 часов (за >30 сообщений/сутки). Эти временные блокировки хранятся в `security_manager._spam_protection` с меткой `blocked_until`. Таким образом, злонамеренный заблокированный пользователь не сможет зафлудить бота (его сообщения просто не пройдут в основные хендлеры, middleware вернёт False). Статистика по таким попыткам доступна через команду `/spamstats` для админов: она выводит список заблокированных пользователей, сколько сообщений они отправили, и кто сейчас заблокирован на какое время.

Оценка завершённости: Функционал модерации и администрирования **богатый, но не полностью завершён**. С положительной стороны, бот обеспечивает админам широкий контроль: интерактивный список пользователей с фильтрами и пагинацией, изменение ролей и статусов, экспорт данных, добавление книг и правку метаданных – всё это указывает на продуманность архитектуры и попытку следовать принципам *SOLID* (разделение на сервисы, утилиты, роутеры). Реализованы и меры безопасности: проверка прав для каждой админ-команды, защита от ввода скриптов в тексте (`validate_user_input` в `security_manager` отсекает HTML-теги, JavaScript и т.п.), а также цензура логов (например, секретные данные `.env` не логируются напрямую). Graceful shutdown и логирование ошибок тоже учтены (глобальный обработчик ошибок `dp_instance.error` логирует исключения, `on_shutdown` закрывает сессии и storage).

Однако имеются **заметные пробелы и потенциальные ошибки**:

- Некоторые функции помечены как «*в разработке*» и не работают до конца: поиск пользователей по имени, добавление тегов пользователю – бот принимает ввод, но не

обрабатывает результат. Эти команды **присутствуют в интерфейсе**, что может ввести админа в заблуждение.

- Отсутствуют команды управления событиями (встречами) – админ не может через бота добавить или изменить событие, хотя структура для этого заложена (методы `create_event/update_event` есть в сервисе событий). Раздел «Управление событиями» фактически только выводит список, без кнопок добавить/удалить.
- В процессе добавления книги из ZIP есть риск, что при **ошибке распаковки или метаданных FSM не сбрасывается**. Бот сообщает об ошибке и выходит, но состояние `waiting_for_zip_file` остаётся активным – следующий присланный файл может неправильно интерпретироваться. На практике админ может сам ввести `/cancel`, но лучше бы бот делал это автоматически.
- В интерфейсе админ-панели есть мелкие недоработки UX: после некоторых действий (статистика, список событий, детали книги) отсутствует кнопка «Назад», что нарушает целостность навигации (в других местах она есть). Это не критично, но слегка снижает удобство.
- Как уже отмечалось, **дублирование логики команд** (`/books` vs `/library`, два обработчика `/search`) присутствует и в админской части: например, `/stats` (расширенная статистика) дублирует часть данных, которые выдаёт кнопка «*Статистика*», но в другом формате. Сами админ-команды `/ban`, `/unban`, `/userinfo` частично перекрываются функциональностью кнопок в панели (блокировать/разблокировать, просмотр профиля). Наличие двух способов делать одно и то же — не ошибка, но требует синхронного сопровождения (и тестирования обоих путей).

В целом, администрирование работает, **кроме заведомо незавершённых модулей** (поиск, теги, события). Критических багов в реализованных частях (блокировка, смена ролей, добавление книг) не обнаружено: они содержат необходимые проверки и логируют ошибки, что упрощает отладку. Тем не менее, для полноты картины требуется дописать и отполировать упомянутые компоненты.

Обнаруженные проблемы и несоответствия

Проблема / Участок кода	Описание и возможное влияние
Дублирование команд / несинхронность <i>Handlers:</i> user_handlers vs library_handlers для /search и /library	В коде одновременно существуют два набора обработчиков для одних и тех же функций (поиск книг, вывод библиотеки). Например, /search объявлен в user_handlers.py (с FSM) и в library_handlers.py (без FSM), при этом оба Router подключены. Это приводит к неопределённости: какой из них сработает? Вероятно, один перехватит команду раньше, а второй останется невыполненным, что означает «мертвый код». Также, команда /library задокументирована и реализована в library_handlers, но в user_handlers по-прежнему есть /books (и она даже используется в help для активных пользователей). Такое разночтение может сбить с толку пользователей (бот принимает оба варианта команды) и затруднить поддержку кода. Требуется рефакторинг: убрать дубликаты, оставить единый подход к поиску и каталогу.
Не полностью реализованные функции <i>Handlers:</i> admin_handlers.py – поиск пользователей, добавление тега	Некоторые admin-функции только намечены: при нажатии «Поиск» бот просит ввести запрос, но сразу отвечает, что «в разработке» – т.е. поиск не производится. Аналогично с тегами: админ может запросить добавление тега пользователю, но бот никак не обрабатывает отправленный тег, оставляя операцию незавершённой. Эти полу-реализации не сообщают явно о своей незавершённости (кроме единственного callback.answer при поиске), что может привести к путанице или зависанию ожидания. Желательно либо скрыть неработающие элементы интерфейса (до реализации), либо как минимум явно уведомлять админа, что функция недоступна.

Проблема / Участок кода	Описание и возможное влияние
<p>Отсутствие проверки существования пользователя при /register Code: <code>user_handlers.cmd_register</code></p>	<p>В сценарии регистрации предполагается, что профиль пользователя уже создан (командой /start). Если же по какой-то причине новый пользователь начнёт с <code>/register</code>, в коде не происходит <code>create_user</code> – бот попытается активировать несуществующую запись и вернёт «<i>Ошибка авторизации</i>». Хотя ситуация редкая (обычно /start всегда предшествует, а Telegram не показывает бот-команды без старта), это логическая лазейка. Для надёжности стоит в <code>/register</code> вызывать <code>create_user</code> если <code>get_user</code> вернул None (с статусом inactive), либо хотя бы сообщать: «Сначала выполните /start».</p>
<p>Состояние FSM не очищается при прерывании процессов Examples: не сбрасывается при ошибке /uploadbook</p>	<p>В длинных цепочках FSM (например, загрузка книги из ZIP) обнаружено, что при некоторых условиях выход из процесса не сопровождается <code>state.clear()</code>. Пример: если <code>extract_fb2_from_zip</code> вернул None (архив без FB2), бот сообщает об ошибке и просто <code>return</code> из handler – FSM остаётся висеть в состоянии <code>waiting_for_zip_file</code>. Пользователь повторно шлёт файл – бот, возможно, примет (handler сработает), но сам факт несброшенного state может привести к путанице. Аналогично, после показа расписания <code>/schedule</code> бот не очищает никакого состояния, т.к. FSM не используется – это ок. Проблема именно в частичных выходах из FSM без завершения. Следует всегда либо вызывать <code>await state.clear()</code> при отмене/ошибке, либо переводить FSM в новое состояние, иначе возможны непредвиденные переходы или блокирование команд (например, <code>/cancel</code> может не сработать, если FSM уже не активен, а бот его ждёт).</p>

Проблема / Участок кода	Описание и возможное влияние
<p>UI/UX непоследовательность
<i>Examples:</i> разные команды для одного действия, отсутствие “Назад”</p>	<p>Интерфейс бота в целом удобен, но есть мелкие несоответствия: в пользовательской справке указана команда <code>/library</code>, хотя бот принимает и <code>/books</code> – это может запутать. В админ-панели для разделов «Статистика» и «События» при переходе пропадает кнопка возврата, в то время как для «Пользователей» и «Добавления книги» она есть. Также, bot выводит некоторые данные двумя путями: напр. командой <code>/stats</code> и кнопкой «Статистика» – информация похожая, но оформлена по-разному. Эти аспекты не ломают логику, но снижают целостность восприятия. Рекомендуется выровнять пользовательский опыт: унифицировать названия команд, везде добавить кнопку «Назад», убрать дубли.</p>
<p>Ресурсозатратные операции в асинхронном контексте
<i>Examples:</i> <code>book_converter.convert_book_formats</code></p>	<p>При загрузке книги из ZIP бот выполняет конвертацию FB2->EPUB/MOBI вызовом внешней программы (Calibre) через <code>subprocess</code>. Этот вызов делается внутри асупс-функции без выноса в отдельный поток, что блокирует event loop на время конвертации. Если файл большой, бот может приостановить обработку <i>всех</i> событий (включая ответы другим пользователям). Подобное справедливо и для операций чтения/записи больших файлов. Для небольшого числа пользователей это не критично, но при росте нагрузки станет проблемой. Решение – использовать <code>await asyncio.to_thread()</code> для таких функций или планировать задачи в фоне. Без изменений, бот потенциально уязвим к временной недоступности во время тяжёлых задач.</p>

Проблема / Участок кода	Описание и возможное влияние
Логирование и отладка (положительный момент)	<p>Отметим, что бот качественно логирует практически все действия: <code>bot_logger.log_user_action</code> и <code>log_admin_action</code> вызываются при каждой команде и важном шаге. Ошибки ловятся и записываются через <code>log_error</code> с понятным сообщением, события безопасности (неверная фраза, попытка доступа) тоже логируются отдельно. Логи пишутся в файл (указанный в <code>config.logging.file</code>) и на консоль. Кроме того, <code>env_validator</code> гарантирует корректность <code>.env</code> при старте (формат токена, длина фразы, валидность ID) – это предотвращает запуск с неподходящими параметрами. Эти аспекты упрощают отладку и эксплуатацию, повышая надёжность системы.</p>

(Таблица содержит как проблемы, так и некоторые важные наблюдения. Каждой проблеме присвоен уровень критичности: выше описаны влияющие на логику или UX, ниже – на производительность и поддержку.)

Заключение

Общая оценка архитектуры: Проект `bookclub_bot_simple` демонстрирует хорошо структурированный подход с разделением по пакетам: конфигурация, утилиты, сервисы, обработчики команд – всё разложено по своим модулям. Используется современная библиотека `aiogram v3`, Router'ы и FSM, что упрощает сопровождение. Заявленные принципы SOLID во многом соблюдаются: бизнес-логика вынесена в слой `services` (например, `UserService`, `BookService`), а проверки прав и ограничения – в `utils` (`access_control`, `security`). Код достаточно **читаемый**: понятные имена функций и переменных, подробные docstring'и на русском объясняют назначение каждого обработчика. Логика покрывает основные потребности книжного клуба: регистрация по секретному слову, выдача книг, учёт участников, роль модератора и т.д.

Готовность к продакшену: На текущий момент бот ближе к стадии **бета-тестирования**. Базовый функционал работает и защищён (нет явных уязвимостей, есть защита от спама, от несанкционированного доступа, валидация вводов). Однако, некоторые функции не завершены, что в продакшене недопустимо: пользователи столкнутся с неработающими кнопками ("в разработке"), а модераторы – с невозможностью управлять событиями или тегами через бот. Кроме того, выявленные дублирующие или противоречивые обработчики могут привести к непредвиденному поведению при расширенном тестировании. Перед релизом в продакшн рекомендуется:

- **Завершить незаконченные функции:** реализовать поиск пользователей, добавление тегов, создание/редактирование событий, либо скрыть их до реализации, чтобы не вводить в заблуждение.

- **Устранить дублирование команд:** оставить одну команду для открытия библиотеки (например, только `/library` вместо `/books`) и единый механизм поиска. Это уменьшит вероятность ошибок и облегчит поддержку.
- **Провести нагрузочное тестирование конвертации:** убедиться, что при добавлении книг больших размеров бот остается отзывчивым, при необходимости вынести тяжёлые операции в фон.
- **Улучшить UX мелочи:** синхронизировать справку с реальными командами, добавить недостающие кнопки «Назад», возможно, сделать сообщения более консистентными.

Уровень зрелости: Архитектура бота можно оценить как **средне зрелую**. С одной стороны, налицо продвинутые механизмы (FSM, middleware, модульность, логирование) – это отличительные признаки тщательно спроектированной системы. С другой стороны, присутствуют признаки, что проект еще развивается: миграция данных при старте (добавление роли всем старым пользователям), TODO-комментарии, несовпадения между документацией и реализацией. Всё это характерно для версии 0.x, где основная функциональность готова, но требуется шлифовка.

Пригодность к реальному использованию: Бот **может быть запущен** и будет выполнять свои основные задачи в текущем виде, но админам и пользователям придётся мириться с некоторыми недоработками. Для внутреннего пользования клуба (где разработчик может быстро поправить вручную базу или объяснить про “фичи в разработке”) – этот бот уже ценен. Но для широкой публики, вероятно, стоит дождаться реализации всех заявленных возможностей и проведения полной отладки. В целом, проект имеет хорошую основу и при доработке упомянутых проблем сможет претендовать на продакшн-статус с уровнем качества, достаточным для реального книжного сообщества.
