**Часть 1: Проектирование общей архитектуры**

1. **Выбор архитектурного стиля**:  
     
    Для моего приложения я выбрал архитектурный стиль **MVC (Model-View-Controller)**. Это подходящий вариант, потому что:  
   * Простота реализации для настольных приложений.
   * Я могу четко разделить ответственность между слоями:
     + **Model** — отвечает за взаимодействие с базой данных (SQLite) и хранение данных.
     + **View** — пользовательский интерфейс, который предоставляет взаимодействие с пользователем.
     + **Controller** — бизнес-логика, которая связывает **Model** и **View**.
   * Легкость тестирования: каждый слой можно тестировать независимо, что облегчает разработку и поддержку.
2. Если же в будущем расширить приложение до веб или мобильной версии, можно рассмотреть клиент-серверную архитектуру как альтернативный вариант.
3. **Диаграмма архитектуры**:  
     
    В моей архитектуре будут следующие основные компоненты:  
   * **UI (View)**: интерфейс на основе инструментов, таких как PyQt, Tkinter или аналоги.
   * **Backend (Controller)**: обрабатывает запросы от пользователя и взаимодействует с моделью.
   * **Database (Model)**: база данных SQLite для хранения информации о книгах, категориях, аннотациях и прогрессе чтения.
4. Использовал Draw.io для реализации архитектуры

**Часть 2: Определение структуры модулей**

1. **Определение модулей**:  
     
    Модули, которые я выделяю в приложении:  
   * **BookModule**: реализует **CRUD-операции для книг** (добавление, чтение, обновление, удаление).  
     + Добавление книг в базу данных.
     + Генерация краткого содержания книги.
   * **CategoryModule**: управление категориями книг (например, "Прочитанные", "Запланированные").  
     + Перемещение книг между категориями.
   * **ProgressModule**: сохраняет прогресс чтения книги.  
     + Сохранение текущей страницы.
     + Получение прогресса чтения.
   * **ExportModule**: экспорт данных в CSV или JSON формат.
   * **AIIntegrationModule**: взаимодействие с моделью **GPT** для генерации аннотаций к книгам.
2. **Разработка API интерфейсов** (если требуется REST API):  
     
    Основные методы взаимодействия с API:  
   * POST /books — для добавления новой книги.
   * GET /books/{id} — для получения информации о книге.
   * PATCH /books/{id}/progress — для обновления прогресса чтения книги.
   * GET /categories — для получения списка категорий.

**Часть 3: Проектирование компонентов**

**3. Распределение ответственности**:

* + **Контроллеры**: взаимодействуют с пользовательским интерфейсом, принимают запросы.
  + **Сервисы**: содержат бизнес-логику (например, управление категориями, генерация аннотаций).
  + **Репозитории**: взаимодействуют только с базой данных, обеспечивая доступ к данным.

**Часть 4: Документирование архитектуры**

1. **Архитектурный документ**:  
     
    Мой архитектурный документ будет включать:  
   * Общее описание архитектуры приложения.
   * Диаграммы (архитектуры, модулей, компонентов).
   * Таблицу модулей с кратким описанием их функциональности.
2. **Обоснование решений**:  
     
    **MVC**  
   * **Гибкость**: модульный подход позволяет легко изменять или добавлять новые функции.
   * **Простота поддержки**: каждый модуль изолирован, что упрощает тестирование и поддержку.

Диаграмма компонентов (Component Diagram)

