**Тестовое Задание для Стажера Python-Разработчика: API для Библиотеки**

**Описание проекта:**Вам необходимо разработать RESTful API для базового управления библиотечным каталогом. Система должна позволять управлять информацией о книгах и читателях, а также процессом выдачи и возврата книг, с использованием JWT для аутентификации пользователей (библиотекарей).

**Требования к функционалу:**

1. **Аутентификация (Новое):**
   * Реализовать эндпоинт для регистрации пользователей (библиотекарей) с email и password. Пароль должен храниться в хешированном виде.
   * Реализовать эндпоинт для входа пользователей, который по email и password возвращает JWT access token.
   * Все последующие операции по управлению книгами, читателями (кроме регистрации/логина самого читателя, если бы он был отдельной сущностью с логином), выдаче и возврату должны быть защищены JWT.
2. **Управление книгами (Защищено JWT):**
   * CRUD операции для книг (Создание, Чтение, Обновление, Удаление).
   * Поля книги:
     + ID (автоматически генерируемый)
     + Название (строка, обязательное)
     + Автор (строка, обязательное)
     + Год публикации (число, необязательное)
     + ISBN (строка, должен быть уникальным, необязательное)
     + Количество экземпляров (число, по умолчанию 1, не может быть меньше 0)
3. **Управление читателями (Защищено JWT, кроме регистрации/логина, если бы они были):**
   * CRUD операции для читателей.
   * Поля читателя:
     + ID (автоматически генерируемый)
     + Имя (строка, обязательное)
     + Email (строка, должен быть уникальным, обязательное)
     + *Примечание: В рамках этого задания читатели не имеют собственных паролей и не логинятся в систему. Ими управляют аутентифицированные пользователи (библиотекари).*
4. **Выдача и возврат книг (Защищено JWT):**
   * Эндпоинт для выдачи книги читателю:
     + Принимает book\_id и reader\_id.
     + **Бизнес-логика 1:** Книгу можно выдать, только если есть доступные экземпляры (количество экземпляров > 0). При выдаче количество экземпляров уменьшается на 1.
     + Фиксируется факт выдачи (например, в отдельной таблице BorrowedBooks с полями id, book\_id, reader\_id, borrow\_date, return\_date (изначально NULL)).
     + **Бизнес-логика 2:** Один читатель не может взять более **3-х** книг одновременно.
   * Эндпоинт для возврата книги читателем:
     + Принимает book\_id и reader\_id (или borrow\_id если используете отдельную таблицу).
     + При возврате количество экземпляров соответствующей книги увеличивается на 1.
     + В записи о выдаче проставляется return\_date.
     + **Бизнес-логика 3:** Нельзя вернуть книгу, которая не была выдана этому читателю или уже возвращена.
5. **Получение информации:**
   * Эндпоинт для получения списка всех книг (может быть публичным или защищенным JWT – на ваше усмотрение, но объясните свой выбор в README).
   * Эндпоинт для получения списка всех книг, взятых конкретным читателем (и еще не возвращенных) (Защищено JWT).

**Технические требования:**

* Язык программирования: Python 3.8+
* Фреймворк: FastAPI
* База данных: PostgreSQL (или SQLite для упрощения, но PostgreSQL предпочтительнее)
* ORM: SQLAlchemy (Core или ORM – на выбор кандидата, но ORM предпочтительнее)
* **Аутентификация (Новое):** JWT (например, с использованием python-jose для токенов и passlib[bcrypt] для хеширования паролей).
* Валидация данных: Pydantic
* Управление миграциями: Alembic (если используется PostgreSQL)
  + **Задание по Alembic:** Создать начальную миграцию для всех таблиц (включая таблицу пользователей для аутентификации). Затем, после создания таблиц, создать **вторую** миграцию, которая добавляет новое поле (например, description для книг, необязательное) и **обновляет существующие записи** (если это поле имеет значение по умолчанию или требует заполнения для старых данных).
* Тестирование: Pytest. Написать юнит-тесты для проверки бизнес-логики (например, попытка взять 4-ю книгу, попытка взять книгу, которой нет в наличии) и **хотя бы один тест для защищенного эндпоинта** (проверка доступа с токеном и без).

**Дополнительные условия:**

* Репозиторий на GitHub/GitLab с осмысленными коммитами.
* Код оформлен по PEP8.
* README.md с:
  + Инструкцией по запуску (включая информацию о том, как зарегистрировать первого пользователя, если это не очевидно из API).
  + Краткое описание структуры проекта.
  + **Обязательно:** Описание принятых решений по структуре БД.
  + **Обязательно:** Объяснение, как реализована бизнес-логика (особенно пункты 4.1, 4.2, 4.3). С какими сложностями столкнулись и как их решили?
  + **Обязательно (Новое):** Краткое описание реализации аутентификации: как генерируются и проверяются токены, какие эндпоинты защищены и почему. Какие библиотеки использовались и почему.
  + **Творческая часть:** Предложите одну дополнительную фичу, которую можно было бы добавить в этот API, и кратко опишите, как бы вы ее реализовывали (на уровне идеи, без кода).

**Критерии оценки:**

1. **Функциональность:** Корректная реализация всех заявленных эндпоинтов, бизнес-правил и **механизма аутентификации**.
2. **Код:** Читаемость, структура, следование PEP8.
3. **Работа с БД и ORM/Alembic:** Правильное определение моделей, использование миграций. Особенно важна вторая, модифицирующая миграция.
4. **Безопасность (Новое):** Корректная реализация JWT аутентификации, хеширование паролей.
5. **Тесты:** Наличие и адекватность тестов для проверки бизнес-логики и защищенных эндпоинтов.
6. **README и объяснения:** Полнота и ясность документации. **Способность кандидата объяснить свои решения (включая выбор по аутентификации) – ключевой момент для оценки самостоятельности.**
7. **Подход к решению проблем:** Как кандидат справляется с реализацией специфичных бизнес-правил и интеграцией аутентификации.
8. **Творческая часть:** Оригинальность и продуманность предложенной фичи.