**Тестовое Задание**

**Необходимо реализовать backend-приложение – собственную систему аутентификации и авторизации. Приложение не должно быть полностью основано на соответствующих возможностях фреймворков, идущих «из коробки».**

Основной задачей разработки является продумывание и предложение собственной системы доступа к ресурсам (Вы должны продумать схему БД, описать ее составляющие , которые будут отражать правила, к каким ресурсам и какие действия может совершать определенный пользователь).

**Приложение должно реализовывать следующий функционал:**

### **Основные модули системы**

#### **1. Взаимодействие с пользователем**

Позволяет пользователям регистрироваться, входить в систему, выходить из учетной записи, обновлять свои данные и удалять аккаунт.

* Регистрация: Ввод имени (фамилии, отчества), email, пароля, повтор пароля.
* Обновление информации: Пользователь может редактировать свой профиль.
* Удаление пользователя: Удаление аккаунта (мягкое) — пользователь инициирует удаление, происходит logout, пользователь больше не может залогиниться, но при этом в базе учетная запись остается со статусом is\_active=False.
* Login: пользователь входить в систему по email и паролю.
* Logout: пользователь выходит из системы.

\* После login система при последующих обращениях должна идентифицировать пользователя.

#### **2. Система разграничения прав доступа.**

* Вы должны продумать и в текстовом файле или в REAME.md описать схему вашей структуры управления ограничениями доступа.
* Реализованы соответствующие таблицы в БД.
* Таблицы заполнены тестовыми данными для минимальной отработки приложения для демонстрации работающей системы.
* Если пользователь имеет доступ к ресурсу по вышеописанным правилам, ему выдается запрашиваемый ресурс. Если по входящему запросу не удается определить залогиненного пользователя, выдается ошибка 401. Если пользователь определен, но запрашиваемый ресурс ему не доступен 403 ошибка — Forbidden. role\_rule
* Реализовать API с возможностью получения и изменения этих правил пользователю, имеющему роль администратора.

#### **3. Минимальные вымышленные объекты бизнес-приложения, к которым могла бы применяться созданная система.**

Таблицы в БД создавать не требуется. Можно просто написать Mock-View, которые по обращениям будут выдавать список потенциальных объектов или описанные выше ошибки.

**Выбор технологий на Ваше усмотрение. Наш совет — DRF + Postgres.**

**P.S. Даже если не удастся полностью выполнить задание, у обучающегося появится понимание различий между аутентификацией и авторизацией, знания о формировании jwt-токенов, что такое сессии, как они связаны с вопросами и ответами клиента и сервера, как работает login и logout в системах, как можно управлять безопасностью любого приложения.**

**Если вы долистали до этого места и вам кажется, что задание очень сложное и вам пока недостаточно знаний, чтобы придумать такую реализацию, поменяйте цвет текста в строках ниже.**

**Про аутентификацию:**

Перевести пользовательский пароль для хранения в БД поможет библиотека bcrypt.

Для создания токена из id пользователя поможет библиотека jwt.

Можно определять пользователя из header Authorization : Bearer {user\_token}, либо после логина создавать сессию (доп таблица sessions, и в response устанавливать пользователю Cookie с sessionid, expire\_at …

В request сразу присваивать request.user перед обработкой запроса в кастомном Middleware в Django.

**Про авторизацию:**

Можно создать таблицы:

roles для описания пользовательских ролей в проекте (админ, менеджер, пользователь, гость);

business\_elements для описания объектов приложения к которым будет осуществляться доступ (пользователи, товары, магазины, заказы, сами правила доступа);

access\_roles\_rules для хранения правил доступа определенной роли к определенному блоку приложения (столбцы role\_id, element\_id, read\_permission, read\_all\_permission, create\_permission,update\_permission, update\_all\_permission, delete\_permission, delete\_all\_permission).

Все столбцы, оканчивающиеся на \_permission, имеют тип bool и отражают, что пользователь может делать с объектами — со всеми объктами, или может что-то делать с объектами, которые создавал сам (предполагается в табличках были бы поля owner (ссылка на user.id)