Необходимо разработать сервис для учёта личных расходов, в котором пользователь может записывать траты и относить их к определенным категориям.

Технические требования

1 Модели

User - можно использовать стандартную модель пользователя Django

Category

- `id` - идентификатор (uuid)
- `name` - название категории
- `creator` - создатель
- `created_at` - дата и время создания (устанавливается автоматически)
- `updated_at` - дата и время обновления (устанавливается автоматически)

Expense

- `id` - идентификатор (uuid)
- `value` - сумма траты
- `spent_at` - дата и время траты
- `description` - описание (может быть null)
- `creator` - создатель
- `categories` - категории траты (может быть несколько)
- `created_at` - дата и время создания (устанавливается автоматически)
- `updated_at` - дата и время обновления (устанавливается автоматически)

2 Административная панель

- Все модели должны быть зарегистрированы в Django Admin
- В админ-панели модели Expense должны отображаться присвоенные категории с помощью TabularInline

3 API

- REST API должен быть реализован с помощью Django REST Framework
- Необходимо использовать разные сериализаторы для чтения и записи
- Каждый эндпоинт должен аутентифицировать пользователя по jwt-токену

CRUD категорий - пользователь может взаимодействовать только со своими категориями:

- `GET /api/categories` - список категорий пользователя
- `POST /api/categories` - создать категорию
- `GET /api/categories/{id}` - получить категорию
- `PUT /api/categories/{id}` - обновить категорию
- `DELETE /api/categories/{id}` - удалить категорию

CRUD расходов - пользователь может взаимодействовать только со своими расходами:

- `GET /api/expenses` - список расходов пользователя
  - поддерживает фильтрацию по диапазону дат, сумме и категориям (для реализации необходимо воспользоваться библиотекой django-filter)
- `POST /api/expenses` - создать расход
- `GET /api/expenses/{id}` - получить расход
- `PUT /api/expenses/{id}` - обновить расход
- `DELETE /api/expenses/{id}` - удалить расход

Примеры API

1 Создание категории

Запрос:

```
POST /api/categories/
Authorization: Bearer <token>
Content-Type: application/json

{
    "name": "Спорт"
}
```

Ответ:

```
{
     "id": "c8a5b8b6-58a1-4a2b-9b0b-1b34d2b85b01",
     "name": "Спорт",
     "created_at": "2025-09-24T12:00:00Z",
     "updated_at": "2025-09-24T12:00:00Z"
}
```

2 Создание расхода

Запрос:

```
POST /api/expenses/
Authorization: Bearer <token>
Content-Type: application/json

{
    "value": 34999,
    "spent_at": "2025-09-20T14:30:00Z",
    "description": "Абонемент в спортзал",
    "categories": [
        "c8a5b8b6-58a1-4a2b-9b0b-1b34d2b85b01",
        "bd2a47ba-da95-4fdb-a738-5152a1927566"
    ]
}
```

Ответ:

```
{
    "id": "b9c7a4a3-91c8-4f22-9cbe-8b234be731ef",
    "value": 34999,
    "spent_at": "2025-09-20T14:30:00Z",
    "description": "Абонемент в спортзал",
    "categories": [
        {
            "id": "c8a5b8b6-58a1-4a2b-9b0b-1b34d2b85b01",
            "name": "Спорт"
        },
        {
            "id": "bd2a47ba-da95-4fdb-a738-5152a1927566",
            "name": "Здоровье"
        }
    ]
    "created_at": "2025-09-24T12:10:00Z",
    "updated_at": "2025-09-24T12:10:00Z"
}
```

4 База данных

- В качестве СУБД необходимо использовать PostgreSQL 14
- Настройки подключения необходимо вынести из кода в переменные окружения

5 Контейнеризация

Сборка и развертывание сервиса должны осуществляться через Docker + docker-compose:

- `web` - Django-приложение
- `db` - PostgreSQL

6 Зависимости

Необходимо использовать зависимости следующих версий:

- `Django==4.1.7`
- `djangorestframework==3.14.0`
- `django-filter==23.2`

Разрешается добавлять дополнительные библиотеки (например, для JWT).

Критерии оценки задания

- Аккуратность и читаемость кода (структура проекта, PEP8)
- Корректность моделей и связей
- Работоспособный CRUD с фильтрацией и авторизацией
- Наличие README с инструкцией по запуску
- Умение работать с Git (осмысленные коммиты, git-flow)