Друзья,

Рады приветствовать вас на практике в компании Sibdev!

Летняя практика всегда была возможностью прокачать навыки программирования и с пользой провести лето. Мы пошли дальше — и попытались сделать ее увлекательной, добавив элементы игры.

Задание максимально приближено к тому, с чем вы столкнетесь в будущем на реальных проектах. Приступить к его выполнению можно с минимальным уровнем подготовки в технологии. А вот чтобы успешно закончить, потребуется приложить усилия.

Чтобы научиться профессионально разрабатывать на Python, недостаточно лекций и онлайн-курсов. Мы уверены, что реальное обучение — это писать код, ошибаться и пробовать снова. Поэтому предлагаем сделать это вместе на Практике.

Желаем успехов от всей команды Sibdev ❄️

Задания

В ходе практики мы разработаем бэкенд веб-приложения. Проект основывается на идее удобного ресурса для простого учета и планирования расходов.

Вас ждут 10 последовательных этапов разработки. Их сложность будет возрастать. Поэтому, чтобы лучше разобраться, мы приложили ссылки на документацию и гайды в самих заданиях и в конце документа.

Если не получается самостоятельно решить задание, просите помощи у других практикантов в чате и в последнюю очередь — у наставника. Вы также можете помогать коллегам советом.

Проект

Нашем проектом станет приложение для учета средств и получении простой аналитики по категориям расходов и доходов. Приложение будет иметь 5 блоков.
В первый пользователь будет вводить данные о доходах и расходах.
Второй соберет суммы затрат в определенных категориях и сведет их в удобную таблицу.
Третий - блок Виджетов, он позволяет ставить и отслеживать экономические цели.
Четвертый - покажет пользователю общий доход и расход.
А скрепит их вместе пятый - календарь с фильтрами.
Общий макет: Макет

Баллы

За задание можно получить от 1 до 10 коинов.

Если вы получили от 1 до 6 коинов за задание — увы, его придется переделать. При этом максимум вы сможете получить 8 баллов.

Если оценка 7, то можно перейти к следующему заданию. А можно повысить оценку до 8, выполнив правки наставника.

Оценку 8–10 повысить нельзя — и так круто. Но можно доработать задание, чтобы утолить перфекционизм.

Достижения

За определенные действия вы получите достижения. За каждое достижение — коины. Достижения описаны в разделе «Достижения».

Рейтинг

На основе баллов формируется рейтинг практикантов: по каждому направлению и по всей практике в целом.

Рейтинг пересчитывается каждый день в 21:00. Бот отправит уведомление в телеграм.

Финал практики

Конец практики — 19 августа.

В разделе «Маркет» вас ждет стильный мерч. Его можно получить за коины после окончания практики.

По итогам практики мы пригласим лучших практикантов на стажировку.

Вам понадобятся

- Базовые знания Python;
- Базовые представления о работе веба;
- Базовые представления об API веб-сервисов.
- Базовые знания по работе в shell? (zsh, bash)

Задание 1: Рабочее окружение

Начнем с рабочего окружения: репозитория, среды разработки и запуска локального сервера.

Разрабатывать код будем в локальном репозитории, а загружать результаты — в удаленный. Там их будет проверять наставник. Договоримся: при каждой отправке merge request будем указывать номер задания и описывать выполненную работу в описании.

У нас будет стаб — заготовка приложения, с ней и будем работать
https://github.com/SibdevPro/practice2021-django-stub

Ход работы

1. Клонировать репозиторий проекта себе в рабочую папку на компьютер.
2. В репозитории ответвиться от ветки `master (main)` и создать ветку `feature/#1`. Под каждую задачу будем создавать ветку, соответствующую номеру задачи.
3. Добавить себе в проект стаб django проекта. https://github.com/SibdevPro/practice2021-django-stub
4. Скачать и настроить PyCharm по инструкции. Или можно использовать любую другую IDE. https://www.jetbrains.com/pycharm/ https://docs.google.com/document/d/13G_jjdX4nLDUtsrl7vB2l6YQUNFWFhW0Oi_2yanylXY/edit
5. Запустить локальный сервер. Как это сделать — описано в Readme стаба.
6. Установить rest client для работы с API. Подойдет Postman, Insomnia или Advanced Rest Client.
7. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Задание 2: Авторизация и регистрация

Мы настроили окружение, самое время переходить к разработке приложения.

Пользователи вносят свои данные и получают аналитику. Чтобы пользователь мог хранить свои данные, ему нужно завести аккаунт. С регистрации и авторизации мы и начнем.

В ходе задания мы настроим авторизацию по токенам. Создадим приложение users, настроим сериалайзеры и контроллеры (viewset/view) для модели пользователя. Настроим роутинг.

Получение данных о юзере будет на условный эндпоинт /users/me/

Для регистрации необходимы следующие поля:

- Username
- Email
- Password

Для авторизации необходимы следующие поля:

- Email
- Password

Ход работы

1. Подключить к django проекту библиотеку Simple JWT. https://django-rest-framework-simplejwt.readthedocs.io/en/latest/
2. Создать и подключить приложение `users` в директории `apps`. Внутри него создать пакеты для моделей, контроллеров и сериалайзеров.
3. Переопределить модель django-пользователя: сделать поле email уникальным и назначить его полем юзернейма для авторизации. Указать новый класс пользователя в настройках в качестве пользователя по умолчанию (AUTH_USER_MODEL). https://docs.djangoproject.com/en/3.2/topics/auth/customizing/#specifying-a-custom-user-model
4. Добавить новый контроллер регистрации, унаследовав его от `CreateAPIView`. Контроллер должен обрабатывать POST-запрос на создание пользователя.
5. Добавить контроллеру регистрации сериалайзер, который сможет провалидировать необходимые для регистрации поля и обработать вызов `create` от контроллера для создания нового пользователя.
6. Зарегистрировать новый контроллер в urls, чтобы он был доступен для вызова.
7. Добавить эндпоинты авторизации и обновления `access token'a`. Simple JWT уже имеет такую функциональность, остается подключить.
8. Добавить эндпоинт для GET запроса, который вернет клиенту информацию об Авторизованном пользователе - id и username.
9. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Готовая логика регистрации
- Готовая логика авторизации
- Готовая логика рефреша токена
- Готовая логика получения информации о юзере

Задание 3: Категории и расходы

Теперь у нас есть пользователь. Пора создать основной функционал приложения. Давайте разработаем логику, модели данных и API для категорий, расходов и доходов.

Здесь создадим объекты в базе данных, описывающие категории (таблица Summary на макете) и факт расхода и дохода (Таблица Транзакций). Расход и доход - это одна модель.

Также добавим функциональность удаления и редактирования транзакции, чтобы приложение позволяло гибко работать с внесенными данными.

Данные для категорий:

- Тип категории (расход/доход)
- Название категории
- Владелец

Данные для транзакции:

- Владелец
- Категория
- Сумма
- Дата операции

Ход работы

1. Добавить в проект новое приложение для работы с расходами и доходами.
2. Добавить модели категории транзакций и транзакции.
3. Добавить контроллер типа viewset и сериалайзеры для работы с категориями. Подробнее о вьюсетах - https://www.django-rest-framework.org/api-guide/viewsets/
   Полученный контроллер должен обрабатывать следующие запросы:
   a. Добавление категории
   b. Удаление категории
   c. Получение списка категорий
4. Для отображения расходов / доходов по категориям добавить в контроллер категорий новый экшен, который будет возвращать список категорий и сумму всех транзакций по категории. Для этого использовать `rest_framework.decorators.action`.
5. Добавить контроллер типа viewset и сериалайзеры для работы с транзакциями. Полученный контроллер должен обрабатывать следующие запросы:
   a. Добавление транзакции
   b. Удаление транзакции
   c. Редактирование транзакции
   d. Получение списка транзакций. Для объектов добавить тип их категории, чтобы фронтенд мог определить расход это или доход.
6. Для списка транзакций добавить пагинацию.
7. Зарегистрировать вьюсеты в routers.py https://www.django-rest-framework.org/api-guide/routers/
8. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Готовые модели данных для категорий и транзакций
- Готовые эндпоинты для создания / удаления / получения списка категорий
- Готовый эндпоинт для расходов по категориям (блок справа от списка расходов / доходов, который отображает сумму расходов по категориям расходов)
- Готовые эндпоинты для создания / редактирования / удаления / списка транзакций

Задание 4: Блок Global

Необходимо разработать логику и эндпоинт для блока Global. В данном блоке отображаются суммарные данные по расходам/доходам за выбранный период в календаре.

Ход работы

1. В контроллер транзакций добавить новый экшен, который будет возвращать суммарные данные по расходам и доходам.
2. Построить запрос на агрегацию значений по расходам и доходам. Если транзакций нужного типа нет, то вернуть 0.
3. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Блок Global отображает суммарную информацию по расходам
- Блок Global отображает суммарную информацию по доходам

Задание 5: Календарь

Мы уже создали аккаунт и возможность проводить транзакции. Самое время разработать логику несложного Календаря, чтобы получать данные по фильтру. Мы хотим, чтобы пользователь смог выбрать произвольный период и при выборе периода данные для блоков Таблица Транзакций, Summary и Global обновлялись, фильтруясь по выбранному периоду.

Здесь поработаем с новой библиотекой Django-filter и разберем подключение фильтров.

Ход работы

1. Добавить в проект новую библиотеку django-filter. https://django-filter.readthedocs.io/en/stable/
2. Создать FilterSet для фильтрации временного промежутка. Временной промежуток указывается двумя датами: start_date и end_date.
3. Подключить фильтр для следующих запросов:
    a. Список транзакций.
    b. Список категорий (блок Summary) с суммарным расходом. В расчете расходов должны участвовать транзакции из временного промежутка.
    c. Блок Global. При подсчете расходов и доходов нужно учитывать транзакци из временного промежутка.
4. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Список транзакций учитывает транзакции только из временного промежутка
- Блок Summary учитывает транзакции только из временного промежутка
- Блок Global учитывает транзакции только из временного промежутка

Задание 6: Виджеты

А вот и финальный функциональный блок приложения - Виджеты. Разработаем логику и модель данных для виджетов пользователя.

Каждый виджет - это небольшое напоминание-трекер. По заданной в нем цели он собирает в себя данные и следит, сравнивая суммы пользователя в категории с заданным лимитом.

Логика блока:

- Пользователь создает блок для контроля своих трат по различным категориям. Задача блока - отображать ему данные по тратам по выбранной категории в сравнении с лимитом
- Срок начинается с момента создания виджета (день - 24 часа, неделя - 7 дней, месяц - 30 дней)
- Пользователь может выбрать критерий "Меньше" или "Больше". Критерий отвечает за цели, например, тратить Больше 3500 на Игры или тратить Меньше 1000 на кино

Данные для виджета:

- Владелец
- Категория
- Лимит суммы, которую можно потратить
- Срок действия (день - 1д (текущий день), неделя - 7д, месяц - 30д)
- Критерий (больше, меньше)
- Цвет (hex)
- Дата создания

Ход работы

1. Добавить новую модель - Widget.
2. Добавить контроллер и сериалайзеры для работы с виджетами. Контроллер должен обрабатывать следующие действия:
    a. Добавление виджета
    b. Удаление виджета
    c. Получение списка виджетов
3. При получении списка виджетов дополнительно для каждого объекта возвращать текущую сумму цели и дату окончания цели.
4. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Готовые эндпоинты для создания / удаления / получения списка виджетов
- В списке виджетов для каждого объекта дополнительно возвращается текущая сумма цели и дата окончания цели.

Задание 7: Права доступа

Необходимо разработать логику по ограничению прав доступа к информации пользователей другими пользователями. Займемся безопасностью приложения.

Ход работы

1. Ознакомиться с DRF permissions: https://www.django-rest-framework.org/api-guide/permissions/
2. Добавить следующие ограничения:
    a. Взаимодействовать с транзакциями, категориями, виджетами могут только авторизованные пользователи.
    b. Только авторизованный пользователь может обратиться на эндпоинт получения информации о себе.
    c. Редактировать и удалять транзакции могут только владельцы.
    d. Удалять категории могут только владельцы.
    e. Удалять виджеты могут только владельцы.
3. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Готовая логика ограничения доступа к данным другим пользователям

Задание 8: Покрытие тестами

Кроме документации нам понадобятся тесты. Они позволяют убедиться, что все части веб-приложения ведут себя так, как ожидалось. Это обеспечивает безопасность при доработках, рефакторинге и командной работе.

Есть 3 типа тестов. Юнит-тесты проверяют отдельный модуль приложения. Интеграционные — модули в связке. E2E-тесты — всё приложение с точки зрения пользователя.

Мы покроем юнит-тестами несколько запросов.

Ход работы

1. Ознакомиться с тем, как разработать тест в DRF. https://www.django-rest-framework.org/api-guide/testing/
2. Разработать тест: только владельцы категорий и виджетов могут выполнять DELETE-запрос.
3. Разработать тест: только владельцы транзакций могут выполнять PUT-, PATCH- и DELETE-запросы.
4. Разработать тест: по GET-запросам на категории, транзакции и виджеты учитываются данные только авторизованного пользователя.
5. Разработать тест: по POST-запросу на категории, транзакции и виджеты возвращается 201, только если указан корректный заголовок авторизации.
6. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат

- Основной функционал приложения покрыт тестами

Задание 9: Документирование кода

Итак, вы разработали бэкенд. Ура! 🎉

Представим: мы отложили проект в папку «Важные достижения в моей жизни». Но спустя год решили его передать коллеге или доработать сами. Придется потратить время, чтобы разобраться в коде. Причем не только коллеге, но и нам — чтобы вспомнить всё спустя год. Можно облегчить эту боль: заранее сделать документацию.

Есть сервисы, которые автоматически документируют код.
Например, Swagger. Его и интегрируем в проект. Он сгенерирует документацию, а мы ее доработаем.

Настроим автогенерацию API (или можешь подключить сваггер ручками и заточить его вручную, пропустив 2 шаг), подключим к проекту drf_yasg.

Ход работы

1. Ознакомиться со Swagger.
   https://swagger.io/docs/specification/about/
   https://drf-yasg.readthedocs.io/en/stable/
2. Интегрировать Swagger в проект и сгенерировать документацию. Можно воспользоваться автогенерацией при помощи `drf-yasg` или написать всю документацию самому.
3. Внести недостающую информацию в документацию вручную.
4. Проверить функциональность, загрузить документацию в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Готовая swagger документация

Задание 10: Настройка веб-сервера

Последнее задание! 🏁

Сейчас API возвращает текстовые данные, а также статические и медиа-объекты. В реальных проектах, возвращать статику и медиа — работа веб-сервера. Веб-сервер справляется с этим быстрее.

Воспользуемся веб-сервером Nginx. Установим и настроим его так, чтобы он возвращал статику и медиа. Остальные запросы пусть обрабатывает Django.

В директории nginx уже есть файл `nginx.conf`, он имеет минимальную настройку, которая позволяет получать коды состояния от сервера.

Ход работы:

1. Связать Nginx с веб-приложением. http://nginx.org/ru/docs/http/ngx_http_proxy_module.html
2. Настроить веб-сервер для работы со статикой. Во время сборки контейнера была выполнена команда `python manage.py collectstatic`, и теперь все статичные файлы находятся в `/nginx/static/`. http://nginx.org/ru/docs/http/ngx_http_core_module.html#alias
3. Проверить работу. Перейти на http://localhost/admin и убедиться, что панель администратора отображается корректно.
4. Проверить функциональность, загрузить код в удаленный репозиторий и отправить merge request наставнику.

Ожидаемый результат:

- Статика и медиа обрабатываются на веб-сервере

Приложение

Python и фреймворки

- Learn Python 3 | Codecademy
- Getting started with Django
- Django Rest Framework

API

- Learn REST: A RESTful Tutorial
- RESTful Web Services Tutorial
- REST Resource Naming Guide

Linux и утилиты

- Learn the Command Line
- Interactive vim tutorial
- Learn Git
- Learn Git branching
- Try Git
- Deploy a Website

Docker

- Работа с Docker
- Хранилище Docker образов

Swagger

- Swagger Documentation