Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

Рефакторинг Баз Данных и Приложений

Лабораторная работа № 1

Выполнил студент группы

БПИ2303

Кузнецов Игорь Вячеславович

**Цель работы:**

1. Ознакомиться с основными принципами рефакторинга и обратного проектирования.

2. Научиться анализировать существующий код, выявлять его слабые стороны и предлагать улучшения.

3. Применить на практике методы рефакторинга для улучшения читаемости, структуры и производительности кода.

4. Развить навыки документирования изменений в коде.

**Выполнение лабораторной работы**

**Задание №1:**

Ознакомьтесь с основными принципами рефакторинга, такими как принцип разделения ответственности, уровни абстракции и устранение дублирования. Выберите небольшой проект или модуль, который вы ранее разработали.

**Выполнение задания:**

**Принцип разделения ответственности** утверждает, что класс или модуль должен иметь только одну причину для изменения, то есть выполнять только одну задачу или иметь одну ответственность. Это помогает сделать код более понятным, поддерживаемым и тестируемым. Когда классы имеют четко определенные обязанности, изменения в одной части системы не влияют на другие части, что снижает вероятность возникновения ошибок.

**Уровни абстракции** относятся к тому, как информация и функциональность представляются в коде. Высокий уровень абстракции скрывает детали реализации и предоставляет более общий интерфейс, тогда как низкий уровень абстракции предоставляет более детализированное представление. Использование разных уровней абстракции позволяет разработчикам сосредоточиться на более высоких концепциях, не отвлекаясь на детали реализации. Это улучшает читаемость кода и упрощает его поддержку.

Принцип **устранения дублирования** гласит, что информация и логика должны быть представлены в коде только один раз. Дублирование может привести к проблемам с поддержкой, так как изменения в одном месте могут не быть отражены в других местах, что приводит к несоответствиям и ошибкам. Устранение дублирования достигается через использование функций, классов и модулей, что позволяет заново использовать код и уменьшать вероятность ошибок

В качестве кода для рефакторинга я выбрал репозиторий своего проекта учебной практики: <https://github.com/kivthe/Academical-Practice>

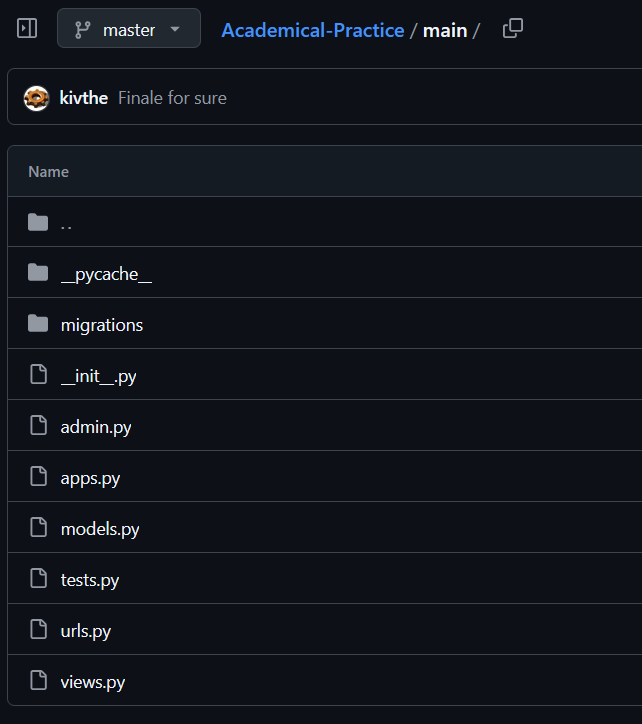
**Задание №2:**

Проведите обратное проектирование выбранного кода. Для этого: постройте диаграмму классов или модулей, чтобы визуализировать структуру программы, определите основные зависимости между компонентами, найдите участки кода, которые сложно читать, поддерживать или тестировать

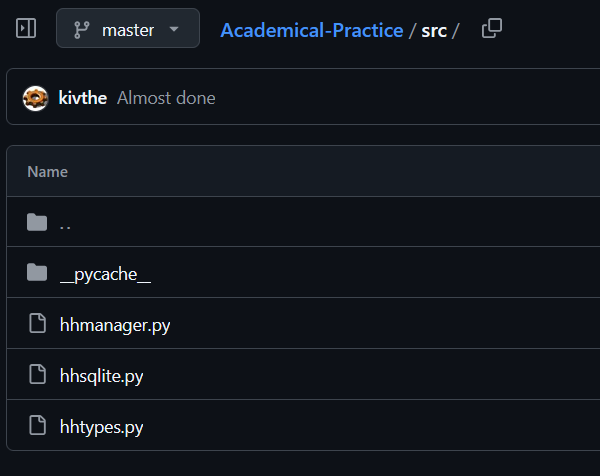
**Выполнение задания:**

В выбранном мной примере не получается построить диаграмму классов, так как весь функционал был реализован в виде отдельных функций, а не классов, так что одной из главных задач рефакторинга будет применения классов и механизмов ООП там, где это необходимо, если это вообще будет нужно. Остальные классы, содержащиеся в проекте, являются классами фреймворка Django, диаграмма для которых уже построена в онлайн-документации, так что создавать её нет никакого смысла.

Основные файлы программы:

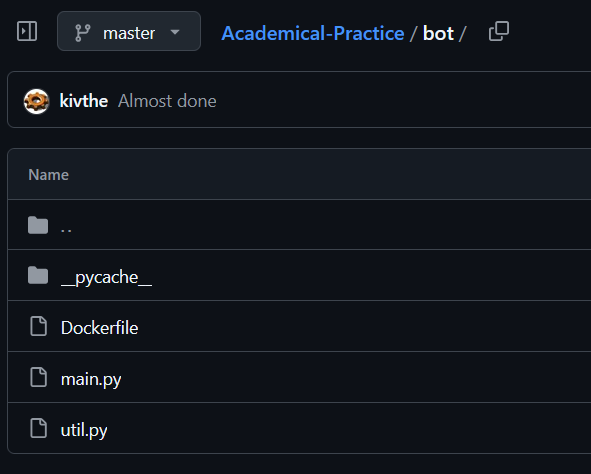


Файл ‘views.py’ зависит от файлов:



Эти файлы реализуют один из основных функционалов.

Оставшиеся файлы являются файлами telegram-бота и являются независимыми:



**Задание №3:**

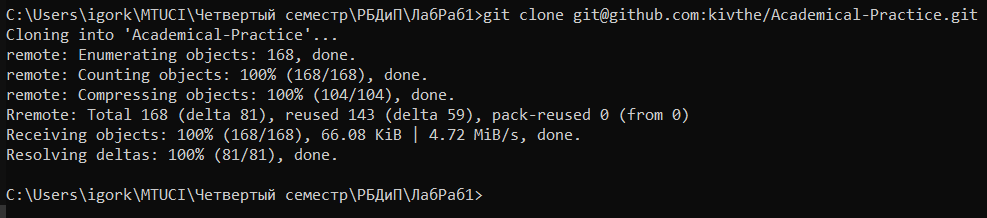
Примените следующие техники рефакторинга: разделение больших функций на более мелкие, устранение дублирующегося кода, улучшение именования переменных, функций и классов, введение уровней абстракции для упрощения сложных участков кода, убедитесь, что после каждого изменения программа сохраняет своё поведение

**Задание №4:**

Опишите все изменения, которые вы внесли в код, и обоснуйте их необходимость. Подготовьте отчет, включающий исходное состояние кода (с примерами проблемных участков), внесенные изменения (с пояснениями), итоговое состояние кода.

**Выполнение заданий:**

Сначала я клонирую репозиторий:



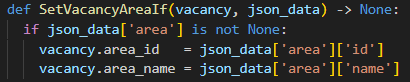
Теперь я начинаю процесс рефакторинга

Начинаю я функции GetVacancies, определенной в ‘hhmanager.py’



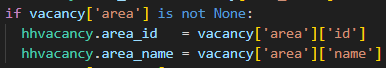
Функция на столько большая, что не помещается целиком на мой экран. Такой большой размер обусловлен множественными if-конструкциями. Такой подход сильно ухудшает читаемость кода, поэтому if-конструкции я заменю на отдельные вызовы функций, проверяющие json-данные.

Пример функции:



Эта функция вставляет в объект вакансии данные из полученного json’a, если они там есть.

Благодаря этому этот код:



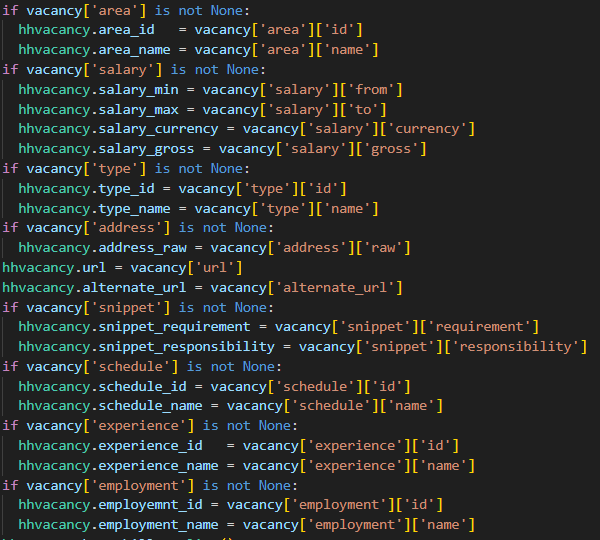
Становится этим кодом:



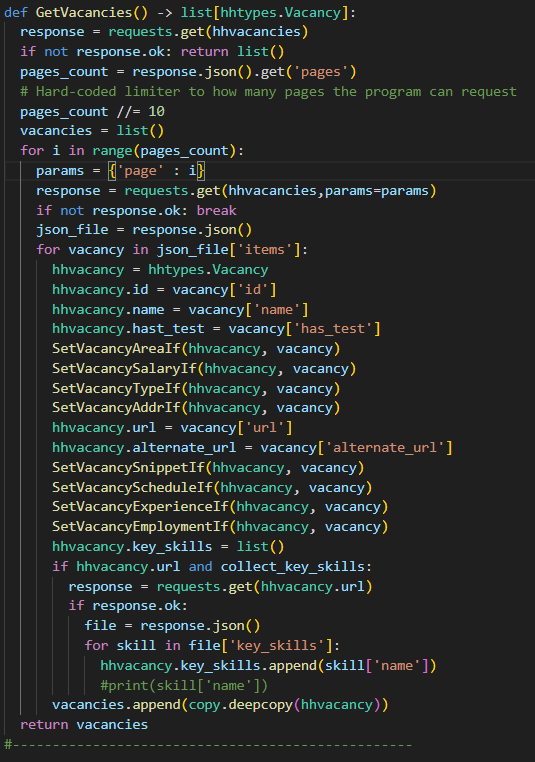
И превращается в простой вызов функции.

Осталось лишь создать отдельные функции для каждого поля данных.

Итого вот эта часть кода:



Превращается в эту:



Благодаря выведению условных установлений значений в отдельные функции, я облегчил функцию GetVacancies, сделав её исходный код меньше и облегчив понимание последовательности её исполнения.

После этого я изменил имена несколько имен локальных переменных, чтобы они лучше отражали свой смысл и хранящиеся в них данные:



hhvacancy -> vacancy

vacancy -> vacancy\_data

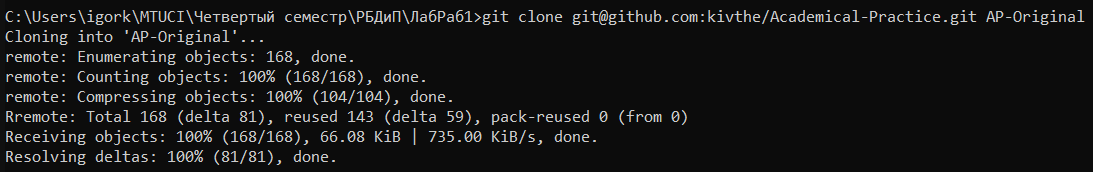
С таким наименованием будет понятнее, что из себя представляет каждая из переменных.

**Задание №5:**

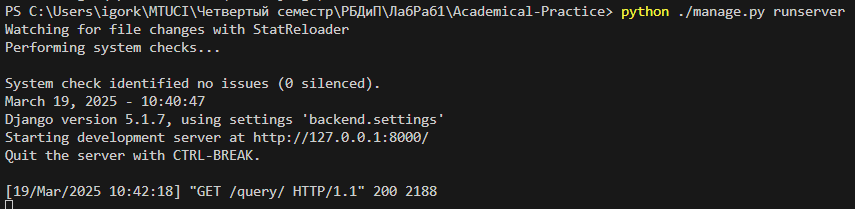
Проверьте, что после рефакторинга программа работает корректно.

**Выполнение задания:**

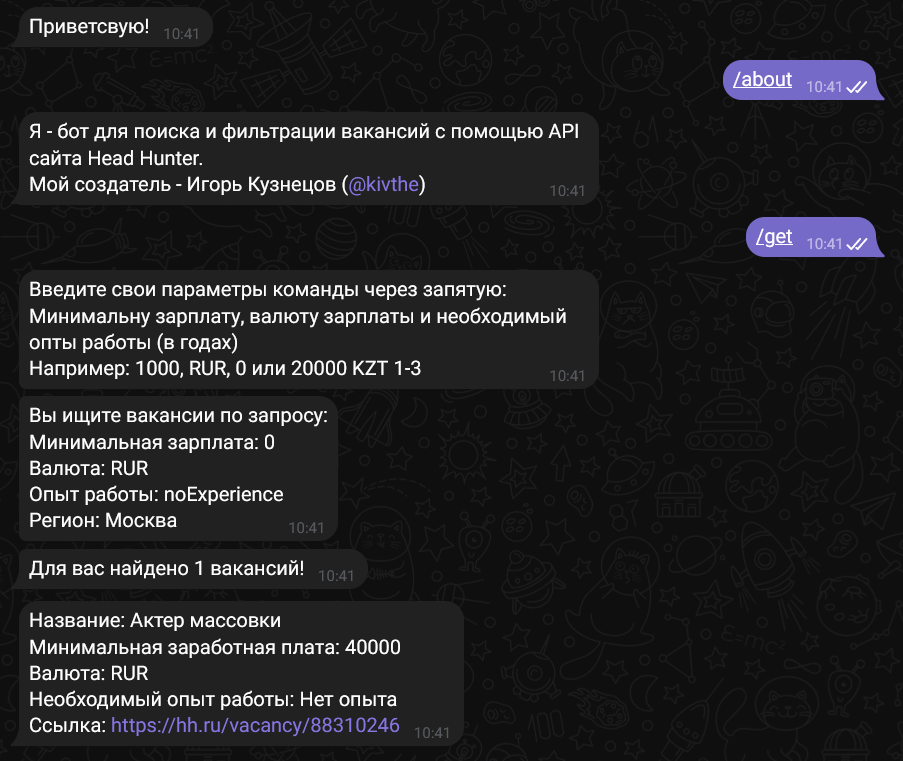
Снова клонирую репозиторий для получения оригинального кода



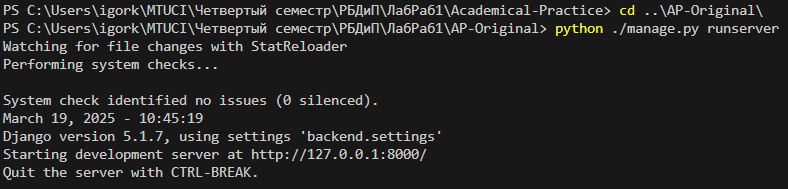
Дальше по очереди запускаю приложения и проверяю работоспособность

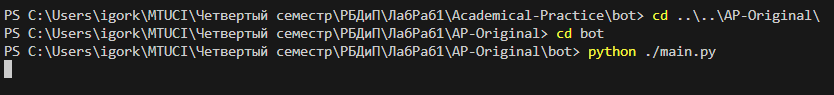


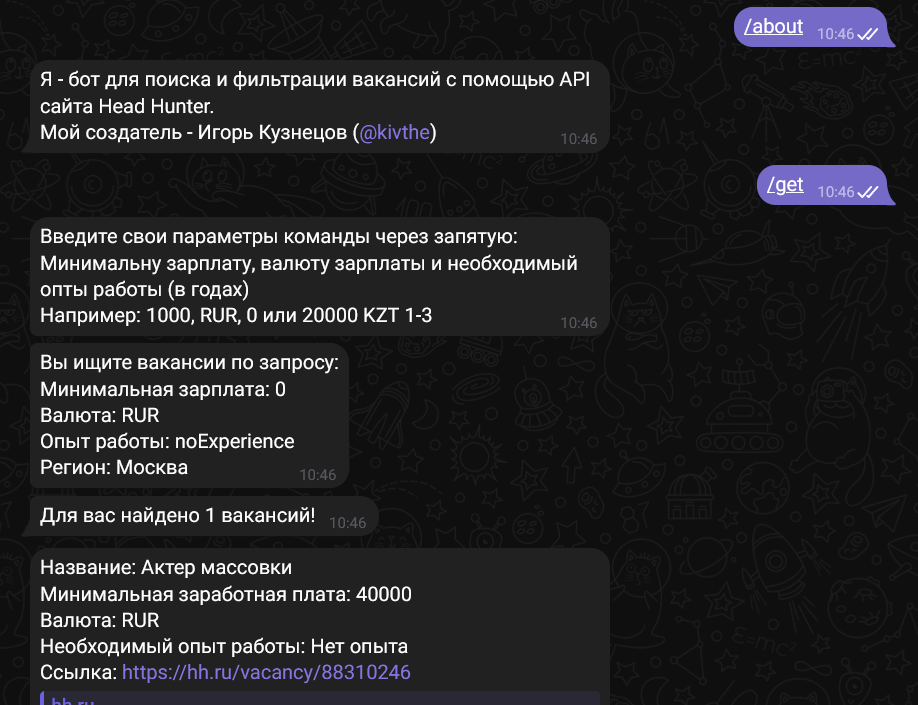




Теперь запускаю оригинальную версию:







У обеих программ идентичный вывод, соответственно, рефакторинг был удачным: код стал более понятен для других программистов, и это никак не сказалось на работоспособности программы