МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Технологии разработки программно-информационных систем

Отчет по практическому заданию № 3

Руководство по стилю и code review

Вариант 1

Выполнил студент группы М3О-108СВ-25

Давыдов А.П.

Принял:

Титов Ю.П.

Москва, 2025

Задание

При выполнении практической работы студенты разрабатывают руководство по стилю для реализации командного взаимодействия и проводят code review. Руководство по стилю может быть основано на известных сторонних руководствах, так и на принятых в определенном языке программирования нормах и парадигмах.

В рамках первого этапа практического задания необходимо совместными усилиями разработать руководство по стилю для бригады программистов.

Пункты руководства по стилю определяются в соответствии с вариантом, при этом необходимо сделать все пункты/разделы из варианта.

Вариант 7

Избегание излишней вложенности и длинных цепочек вызовов методов.

Обеспечение безопасности кода и защиты от уязвимостей.

Разработка документации для кода, проекта и пользователей.

Использование контейнеризации и оркестрации для управления развертыванием приложений.

На втором этапе студенты должны провести code review. Выбирается небольшой участок кода. Рекомендуется, чтобы данный участок кода исправлял автор, но если студент не писал код проекта, то можно выбрать несложный небольшой (150–200 строк кода), обязательно разобравшись в нем. Выбранный участок кода отдается на code review другому студенту, который проверяет написанный код и предлагает исправления и улучшения. Все предложенные исправления и взаимодействие между студентами должно быть оформлены в отчете. Например, если общение происходит средствами почты или системы управления проектами (что является наиболее распространенной практикой), то необходимо предоставить тексты писем, если устный code review, то приводится стенограмма разговоров. Минимум необходимо провести 2 раунда code review с выявлением исправлений и рекомендаций в каждом раунде.

Выполнение

Первая версия

Общий проект пишется на фреймворке Django на языке питон. В команде было решено использовать стандарт «PEP 8».

В соответствии с вариантом, следует оценить следующие рекомендации из PEP 8:

- Избегание излишней вложенности и длинных цепочек вызовов методов.
  - Длинные цепочки вида `obj.method1().method2().method3()` следует разбивать на промежуточные переменные
  - Писать отдельные функции, инкапсулирующие такие цепочки
  - Пример из кода

```
feedback_edit(request: HttpRequest, feedback_id: int):
if request.method == "POST":
    form = FeedbackForm(request.POST)
    instance = Feedback.objects.filter(id=feedback_id)[0]
    if instance.author == request.user:
        if form.is_valid():
            instance.text = form.cleaned_data["feedback_text"]
            instance.score = form.cleaned_data["feedback_score"]
            instance.save()
            # form.save()
            return redirect(
                "client:feedback_list", profile_id=instance.recipient.id
            )  # profile_id - id НА КОГО оставлен отзыв
```

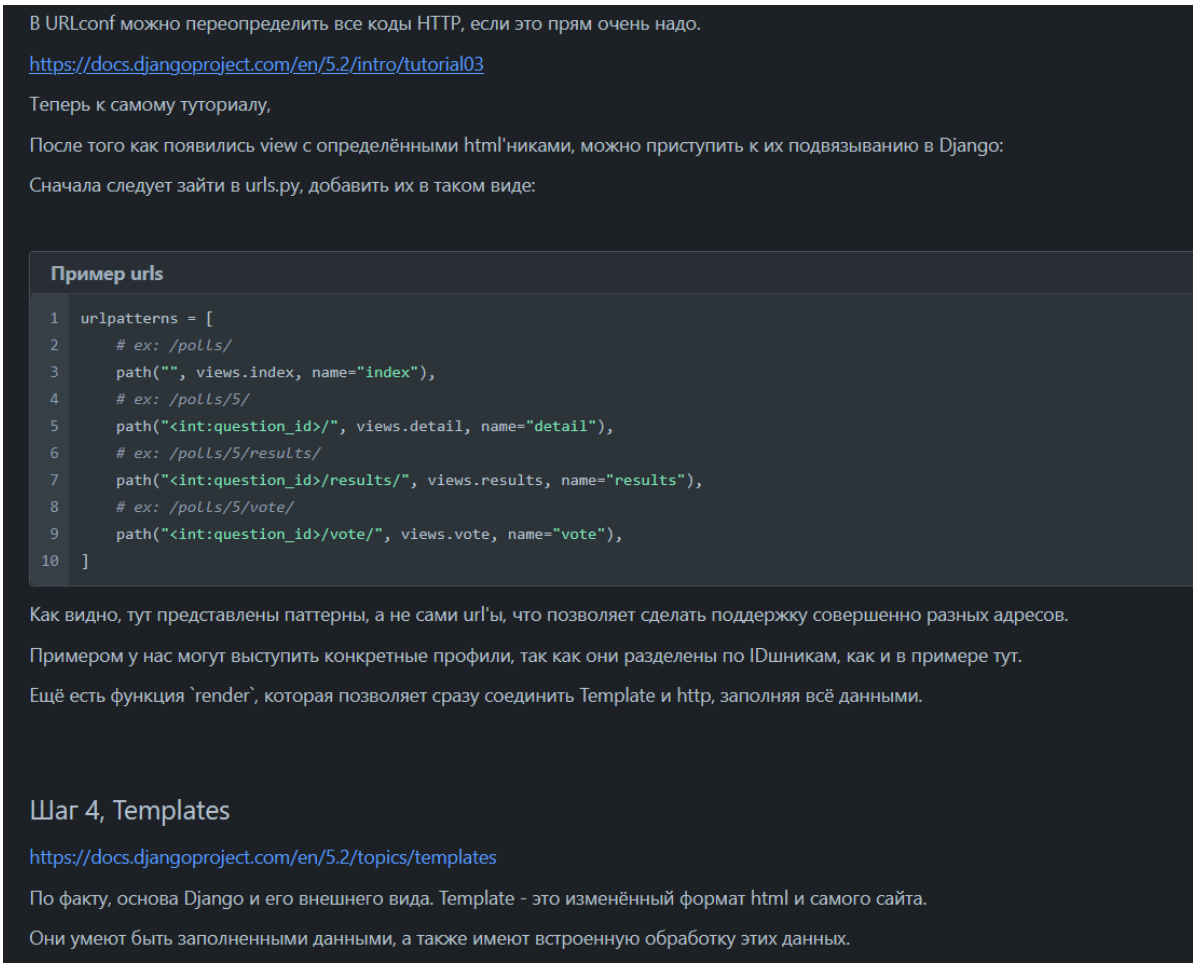
- Обеспечение безопасности кода и защиты от уязвимостей.
  - В контексте Django с этим помогают декораторы, такие как @login_required

```
@login_required(login_url="client:login")
def feedback_edit(request: HttpRequest, feedback_id: int):
    if request.method == "POST":
        form = FeedbackForm(request.POST)
        instance = Feedback.objects.filter(id=feedback_id)[0]
```

    Это обеспечивает доступ к более глубоким системам только для авторизованных пользователей. В данном случае очень важно, чтобы анонимные пользователи не могли никак взаимодействовать с формой оценки, ведь тогда вся система будет уязвима к бот-нет атакам из-за прямого доступа к базам данных. Благодаря этому злоумышленник сможет изменять характеристики работодателя, возведя их в абсолют или обрушив.

- Разработка документации для кода, проекта и пользователей.

- В ходе проектной работы использовался Confluence, в котором были расписаны ключевые фрагменты системы



В URLconf можно переопределить все коды HTTP, если это прям очень надо.

https://docs.djangoproject.com/en/5.2/intro/tutorial03

Теперь к самому туториалу,

После того как появились view с определёнными html'никами, можно приступить к их подвязыванию в Django:

Сначала следует зайти в urls.py, добавить их в таком виде:

Пример urls

```
urlpatterns = [
    # ex: /polls/
    path("", views.index, name="index"),
    # ex: /polls/5/
    path("<int:question_id>/", views.detail, name="detail"),
    # ex: /polls/5/results/
    path("<int:question_id>/results/", views.results, name="results"),
    # ex: /polls/5/vote/
    path("<int:question_id>/vote/", views.vote, name="vote"),
]
```

Как видно, тут представлены паттерны, а не сами url'ы, что позволяет сделать поддержку совершенно разных адресов.

Примером у нас могут выступить конкретные профили, так как они разделены по IDшникам, как и в примере тут.

Ещё есть функция `render`, которая позволяет сразу соединить Template и http, заполняя всё данными.

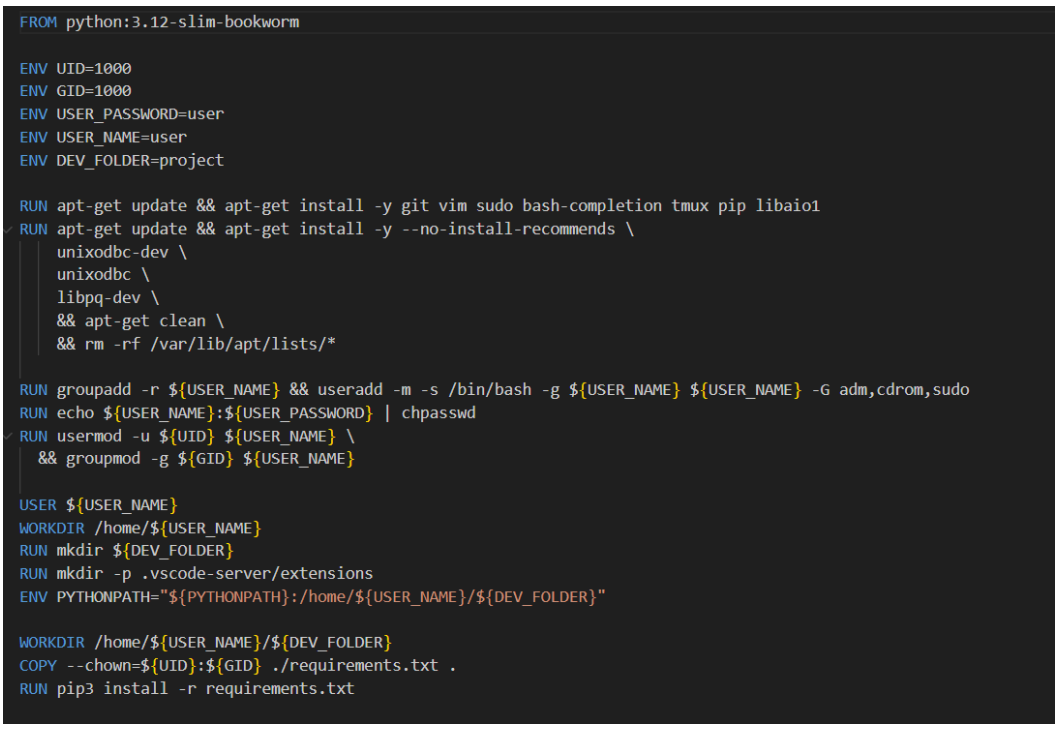
Шаг 4, Templates

https://docs.djangoproject.com/en/5.2/topics/templates

По факту, основа Django и его внешнего вида. Template - это изменённый формат html и самого сайта.

Они умеют быть заполненными данными, а также имеют встроенную обработку этих данных.

- Использование контейнеризации и оркестрации для управления развертыванием приложений.
  - Проект целиком помещён в контейнер и запускается с помощью Docker. Для этого используется автоматизация запуска проекта с помощью Dockerfile (И расширения VSCode)



```
FROM python:3.12-slim-bookworm

ENV UID=1000
ENV GID=1000
ENV USER_PASSWORD=user
ENV USER_NAME=user
ENV DEV_FOLDER=project

RUN apt-get update && apt-get install -y git vim sudo bash-completion tmux pip libaio1
RUN apt-get update && apt-get install -y --no-install-recommends \
    unixodbc-dev \
    unixodbc \
    libpq-dev \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

RUN groupadd -r ${USER_NAME} && useradd -m -s /bin/bash -g ${USER_NAME} ${USER_NAME} -G adm,cdrom,sudo
RUN echo ${USER_NAME}:${USER_PASSWORD} | chpasswd
RUN usermod -u ${UID} ${USER_NAME} \
  && groupmod -g ${GID} ${USER_NAME}

USER ${USER_NAME}
WORKDIR /home/${USER_NAME}
RUN mkdir ${DEV_FOLDER}
RUN mkdir -p .vscode-server/extensions
ENV PYTHONPATH="${PYTHONPATH}:/home/${USER_NAME}/${DEV_FOLDER}"

WORKDIR /home/${USER_NAME}/${DEV_FOLDER}
COPY --chown=${UID}:${GID} ./requirements.txt .
RUN pip3 install -r requirements.txt
```

А также развёртка проекта в виртуальном окружении (venv) для Python, организованная с помощью WSL (Windows Subsystem for Linux). Это обеспечивает гибкую поддержку скачиваемых пакетов, больше конфигурации и настройки проекта, а также возможности для работы той же БД. В случае чистого Windows для каждой из этих систем пришлось бы создавать собственное отдельное окружение.

Второй этап:

Вторая версия

(Сделано для варианта 1 из-за неожиданной коллизии)

Определение стандартов и принципов стиля кодирования.

Эти стандарты актуальны для языка C# и были написаны автором работы исходя из опыта.

1) Любые константы следует писать КАПСОМ, разделяя_слова_подчёркиваниями.
2) Любые переменные следует называть с маленькой буквы. Составные слова писать с большой, вотТакимОбразом.
3) В названии переменной не должно быть больше 4-х слов, либо переменная не должна занимать треть вашего экрана по длине. При этом она должна максимально отражать свой смысл.
4) Закрытые (Со словом private) переменные, использующиеся в пределах класса следует отмечать подчёркиванием
5) Функции, названия классов и делегаты пишутся с большой буквы

6) Переменные в пределах цикла перебора foreach можно называть краткой аббревиатурой, но только если весь такой цикл помещается в экран (Чтобы подсветка работала)

Применение логических операторов и условий для улучшения читаемости кода.

Правило Nesting'a, когда чтобы не делать избыточной табуляции, проверки и условия выносятся в отдельные условия с выходом из функции или цикла, и, соответственно, инверсией проверки.

Плохая практика:

```
public void Foo()
{
    if(check1 < 0)
    {
        if (check2 != 167)
        {
            DoSomething()
        }
    }
}
```

Рефактор такого кода:

```
public void FooRight()
{
    if (check1 >= 0)
        return;

    if (check2 == 167)
        return;

    DoSomething();
}
```

Это позволяет уменьшить количество фигурных скобок, а также повышает читаемость самих условий.

Кроме того, такой подход вынуждает декомпозировать функции по принципу выхода из них, своим дизайном создавая необходимость разделения функций.

Иными словами, если требуется сделать много проверок — создавай функцию.

Соблюдение принципов DRY (Don't Repeat Yourself) и KISS (Keep It Simple, Stupid).

DRY:

Означает, что функционал должен повторяться только в вызове функций. Если какой-то фрагмент кода повторяется больше уже хоть раз — это серьёзный повод его вынести в отдельную сущность.

Три — это приговор для декомпозиции.

Тем не менее, осмысленно декомпозировать только достаточно большие функциональные компоненты, в которых делается больше 3-х действий, либо используется хотя бы один цикл.

KISS:

Код должен выглядеть и читаться не как набор чрезвычайно сложной, закрученной логики, а как чёткая последовательность действий.

Чёткие правила в данном случае извлечь практически невозможно, но можно придерживаться некоторых из следующих принципов:

1) Составные функции, выполняющие последовательные шаги, следует разбить по шагам и писать следующим обазом:

```
public IEnumerator GenerateTerrain()
{
    yield return GenerateFloor();
    yield return GenerateWalls();
    yield return PlaceStructures();
    yield return DisplaceEnemies();
}
```

2) Придерживаться правила «Не больше 7 действий». То-есть если функция оказывается влияние на 7 каких-то систем или сущностей, её лучше разбить, упростив таким образом

```
public IEnumerator CreateInstance()
{
    Singleton<ObjectManager>.Instance.AddNew(this);
    DoCalculations(); //Здесь внутри ещё 2 вызова менеджеров
    NotifyNeighbors(); //Здесь вызов 4-х соседей
    Singleton<ObjectSaver>.Instance.SaveData(this);

    //Итого 4 действия
}
```

Работа с базами данных и обработка запросов к ним.

Здесь будет описана информация об актуальном для проекта SQL.

- Рекомендуется использовать следующие правила при работе с SQL:
  Используйте форматирование SQL-запросов для улучшения читаемости: ключевые слова пишите заглавными буквами, каждый основной оператор (SELECT, FROM, WHERE и т.д.) — с новой строки.
- Избегайте вложенных запросов, заменяйте их JOIN-ами, где это возможно, для повышения производительности и понятности.
- Обязательно используйте алиасы для таблиц и колонок, чтобы код был лаконичным и понятным.
- Используйте параметры запроса (prepared statements) для защиты от SQL-инъекций.
- Избегайте динамической конкатенации строк для формирования запросов.
- При выполнении запросов обрабатывайте возможные ошибки, логируйте их в течение выполнения.