## *План выступление на защите проекта по Python Basic*

1. Вступление. Рассказ о себе

Добрый вечер. Меня зовут Екатерина, по образованию я инженер в области гидравлики. Опыт работы в автоматических системах управления. Пришла в Skillbox для смены сферы деятельности и углубленного изучения программирования. В своей учебной и профессиональной деятельности часто сталкивалась с задачами, где имелась возможность автоматизировать некоторые процессы. Неоднократно мне помогали мои базовые знания Pascal и Matlab для решения задач.

1. Команды телеграмм бота

Основной целью являлось итогового проекта создание работоспособного и комфортного для использования телеграмм-бота поиска отелей.

На слайде представлен список команд телеграмм бота. Сейчас подробно останавливаться не буду. В дальнейшем, при демонстрации работы бота, расскажу подробнее.

1. Основные учебный цели

В пределах данной задачи я также поставила перед собой цели: закрепление и расширение знаний, полученных на курсе Python Basic и углубленное изучение парадигмы ООП и паттернов проектирования для построения гибкой структуры приложения. Кроме того, необходимо было изучить и применить основы БД, а также ознакомится с ORM pewee.

1. Архитектура. Слои приложения

Для реализации вышеуказанной задачи, я разделила бота на 3 слоя:

* Слой bot. Реализует взаимодействие пользователя в телеграмм-чате.
* Слой rapid. Делает запросы и обрабатывает данные, полученные с API hotels.
* Слой db. Построение моделей БД и создание объектов для взаимодействия с БД.

1. Архитектура. Паттерны проектирования и реализация

В ходе реализации телеграмм-бота были применены паттерны проектирования, указанные на слайде. Это позволило создать гибкую структуру готовую к изменениям. В дальнейшем, при демонстрации кода, я остановлюсь подробнее.

1. Архитектура. Дерево классов handler\_message.

На данном слайде представлен пример дерева классов для обработчиков сообщений, необходимых для команд поиска отелей. В данном случае, хорошо просматривается примененный паттерн «Фабричный метод».

1. Трудности. Улучшение бота.

В ходе проектирования бота, я столкнулась с определенными трудностями, такими как изучение и выбор библиотеки для телеграмма, самостоятельное изучение основ БД, логирование и дебагинг приложения, работа с исключениями и их «отлов» и т.д. Однако, основной трудностью было применение парадигмы ООП после долгого применения структурного и функционального стиля программирования.

На данный момент бот работает корректно, однако появилось несколько идей для улучшения программы и кода. Такие как:

* Создание файла с шаблонами сообщений, отправляемых пользователю.

Так называемых «констант» сообщений;

* Выбор команд «на лету»

В случае, если пользователь решит заменить команду на любом этапе выполнения уже выбранной команды;

* Добавления cancel в виде команды/горячей клавиши

Для удобства пользователя, чтобы тот не вводил команду отмены вручную.

Сейчас я продемонстрирую работу бота в телеграмм.

**Демонстрация бота**

Запустим телеграмм бота. Файл Main.py содержит 2 строки. Создание телеграмм бота и его запуск.

При открытии телеграмм с чатом бота, мы видим меню, где указаны команды. При нажатии команды start, бот отправляет приветственное сообщение с предложением новой команды – help. Нажмем на нее. В ответ мы получаем сообщение с перечнем команд и их описанием.

Предлагаю выбрать команду bestdeal, как наиболее высоконагруженную. Данная команда подбирает отели по удаленности от центра и диапазону цен. При ее нажатии пользователь получает информационное сообщение о выбранной команде и ключевым словом для отмены действия.

Введем город **Берлин.** Далее предлагается ввести год начала поездки, месяц и дату, причем мы не можем выбрать дату до сегодняшнего дня. Введем: **15.05.2022.** Также введем дату окончания поездки **20.05.2022.** (здесь тоже нет возможности выбрать дату раньше даты заезда).

Далее бот нам предлагает выбрать или ввести количество проживающих в номере. Стоит отметить, что, если мы попробуем ввести текст или неверное число, бот «отловит» ошибку и предложит ввести заново. Нажмем кнопку **2**.

Здесь бот предлагает выбрать или ввести число отелей. Работает аналогично предыдущему. Нажмем кнопку **5**.

Аналогично, выберем количество фото для каждого отеля. Например, **3 фото.**

Бот предлагает ввести диапазон цен. Например, введем **5000-30000**. Причем, здесь можно вводить через любой разделитель.

Введем удаленность от центра, пример 2 км.

Сейчас нам предлагается проверить введенные данные и начать поиск. Нажмем **Да**.

Начался поиск. Сейчас приложение запрашивает данные о городе. Получен список отелей. И получены фото для отелей.

Мы видим результаты поиска. Первым идет карточка отеля, где можно увидеть название, адрес, звездность, рейтинг, а также цену за сутки и за все время проживания. Далее идет фотоальбом с фотографиями отеля.

Предлагаю перейти к другому классу команд. Lowprice и highprice. Данные команды выводят список отелей, сортированных по минимальной или максимальной стоимости. Например, выберем **lowprice.** Действия аналогичные предыдущему. Выбираем **Лондон** - **15.05.2022 - 20.05.2022 – 2 – 3 – Без фото.**

Перейдем к команде history. Данная команда отправляет сообщения о 5 последних действиях. Причем в каждом сообщении есть дата и время ввода команды, какая команда была введена и список найденных отелей с их названиями и ссылками.

Предлагаю перейти к коду, я остановлюсь на более интересных, по моему мнению, моментах.

**Демонстрация кода.**

Для начала предлагаю посмотреть слой rapid.

1. Hotel\_hadnler.py

Для создания единого интерфейса взаимодействия со слоем rapid был применен паттерн Фасад, который объединяет последовательность 3х запросов: локации, отелей и фото.

Здесь создаются экземпляры для вышеуказанных запросов, а также для обработки локации, отелей и фото.

Далее перейдем в метод handler. Мы видим поочередный запуск запросов и обработки. Получение id города, формирование списка отелей и, в случае если, пользователь запросил фотографии, формирование списка с url-адресами фото.

1. Hotel.py

В данном файле реализованы запросы, а также обработка данных о полученных отелях для дальнейшего их использования телеграмм-ботом. Например, рассмотрим объект **LocationRequest**, где происходит запрос локации исходя из данных, полученных от пользователя. С помощью миксина **RequestMixin**, удалось объединить все запросы в 1 объект, который отвечает за request. Перейдем далее, в объект LocationHandler, который непосредственно получает результат запроса (JSON-файл) и обрабатывает его (получает id города).

Аналогичным образом реализованы остальные объекты для отелей и фото.

**Перейдем к bot**

1. Bot\_main.py

Здесь происходит создание телеграмм бота и добавление обработчиков сообщений и команд.

1. Command\_handler.py

В данном файле описаны обработчики «простых» команд (т.е. не используемые диалог с пользователем), такие как start, history, help. Рассмотрим, например, объект Welcome, который отвечает за команду start. Необходимо уточнить, что приветственное сообщение, как и многие другие оставлены для читаемости кода, но в дальнейшем такого типа сообщения необходимо перенести в файл setting, что я указывала в призентации.

Кроме того, при выполнении команды start пользователь добавляется в таблицу БД, а при выполнении history происходит запрос в БД и ответом является список 5 последних действий.

1. Handler\_message.py

Здесь обработчики сообщений, приходящих от пользователя в ходе «диалога». В данном случае применен Фабричный метод, предлагаю остановится на объекте PhotoCountHandler. Это обработчик числа фотографий, запрашиваемых пользователем. В данном случае обрабатываются нажатие кнопок на inline клавиатуре и текстовые сообщения.

1. Registry.py

Для реализации возможности работы с несколькими пользователями и сохранении в памяти всех введённых данных был применен паттерн Реестр или Регистр, где для каждого пользователя создается свой словарь с данными. Реестр является единственным и используется паттерн Одиночка. Объект Data отвечает за сбор данных о запросе пользователя.

Стоит отметить, что все данные хранятся в оперативной памяти. Далее при нажатии кнопки поиска, формируется запрос отелей исходя из данных из реестра. В случае успешного поиска, данные об отелях и запросе отправляются в БД для хранения и удаляются из реестра.

1. Conversation\_handler.py

В данном файле происходит создание обработчика диалога с пользователем. Здесь применен паттерн Строитель, поскольку типов обработчиков сообщений, необходимых для работы бота, достаточно много.

Рассмотрим подробнее методы:

* create\_attr – создаются атрибуты и присваиваются им экземпляры классов обработчиков сообщений;
* command\_handler – добавление первого и последнего обработчика диалога;
* msg\_handler – добавление обработчиков только для текстовых сообщений;
* query\_handler – только для кнопок;
* msg\_query\_handler – для кнопок и текстовых сообщений.

Далее идет класс директор, который последовательно вызывает методы и создает conversation handler. И классы SortPriceConversationHandler и BestdealConversationHandler, которые идут в регистратор обработчиков.

**Перейдем к БД**

1. DBeaver

База данных состоит из двух таблиц: users и user\_requests. Первая таблица создана для «регистрации» пользователей, в ней хранятся Имя пользователя, его id и время начала работы с ботом (первый ввод команды start).

Вторая таблица создана для работы команды history, которая выводит 5 последних действий пользователя. Здесь основными полями являются id пользователя, дата и время запроса, город, введенная команда и результат выполненной команды. Причем, данные об отелях хранятся в JSON.

1. Models.py

Здесь описаны модели базы данных. В таблице users есть поле user\_id с уникальным значение. Кроме того, таблицы связаны между собой этим полем. В таблицу user\_requests не может быть добавлена запись с user\_id, которого нет в таблице users

1. Db\_handler.py

В данном файле происходит взаимодействие с БД. Основным объектом является DatabaseHandler, который объединяет 3 стратегии (создание записи в таблицу, чтение данных о пользователе и чтение данных о запросе). Подробнее рассмотрим стратегию ReadRequest, здесь виден запрос на данные из базы, причем для этого используется контекст менеджер, и производится обработка данных.

**Логирование**

Стоит отметить логирование ошибок и работы приложений. Логер реализован с помощью библиотеки logging, «вешается» декоратором на необходимые классы для логирования всех методов класса. Например, в файле command\_handler класс Welcome. Отдельно производится логирование ошибок и предупреждений.

**Settings и README.md**

Также, хочу показать файл settings.py, где собраны все константы, необходимые для работы бота. Мною также был создан файл README.md с кратким описанием работы телеграмм-бота.