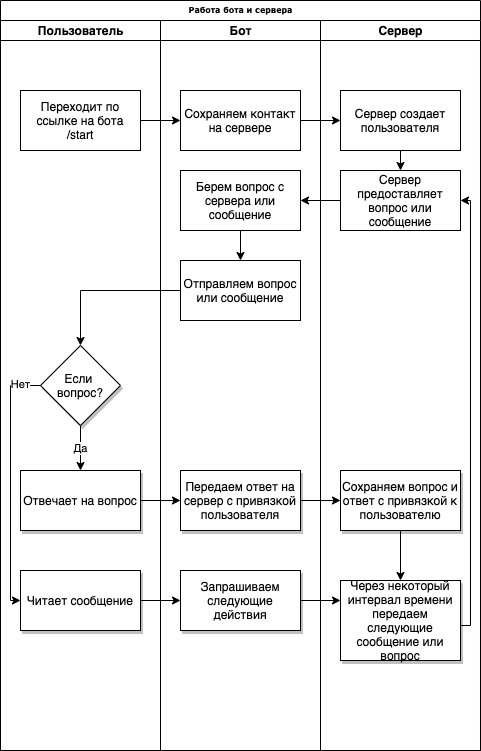
Задание на разработку Telegram бота

Задача: Разработать связку Telegram bot + backend для предоставления контента пользователю.

# Основная логика работы



Представляет собой диалог между пользователем и ботом где информацию которую нужно получить от пользователя сохраняем на сервере с привязкой к пользователю.

Вопросы и сообщения берем с сервера.

# Что должен делать telegram bot.

Задача бота демонстрировать сообщения и вопросы создание в бизнес логике сервера пользователя.

1. В начале отправляем запрос на создание пользователя на сервер
2. Далее получаем инструкции с сервера и транслируем их пользователю
3. Если это вопрос то передаем ответ на сервер с привязкой к пользователю

Общение с сервером ведем через API Json

# Что должен делать backend

Задача серверного приложения обеспечить работоспособность бота, сохранение и отображение полученной информации.

1. Должен предоставить API для работы с ботом (используем Django Rest Framework)
2. Предоставить интерфейс для создания и редактирования “логики общения” с пользователем
3. Предоставить доступ к информации от пользователей
4. Определять тайминг отправки сообщений пользователям

# Пример “логики общения”

**Пользователь**: /start

**Бот**: На связи Бот кейса курсов компании фогстрим

Мне будет приятно обращаться к Вам по имени. Напишите, как Вас зовут?

**Пользователь**: Петр

**Бот**: Рад знакомству 😉

Пока готовлю информацию для Вас, попрошу ответить еще на пару вопросов. Это поможет в будущем подобрать полезную информацию для Вас.

Согласны?

⚠️ Напишите "Да" или "Нет"

**Пользователь**: Да

**Бот**: Отлично!

Сейчас Вам придут 3 вопроса, один за одним. Пожалуйста, отвечайте на каждый вопрос одним сообщением

Поехали

“Через 30 секунд” **Бот**: Сколько Вам лет?

**Пользователь**: 20

“Через 10 секунд” **Бот**: В каком городе проживаете?

**Пользователь**: Тверь

“Через 10 секунд” **Бот**: Что вы хотите узнать?

**Пользователь**: Все

**Бот**: Спасибо!

Сейчас запишу Ваши ответы и вышлю вам интересную информацию. Одну минутку...

“Через 1 минуту” **Бот**:

Вот ссылка на интересный материал

• http://python.fogstream.ru/

“Через 1 час” **Бот**: Получилось ознакомится с материалом ?

Разработан практический сценарий дерева диалога с ботом, на который можно ориентироваться.

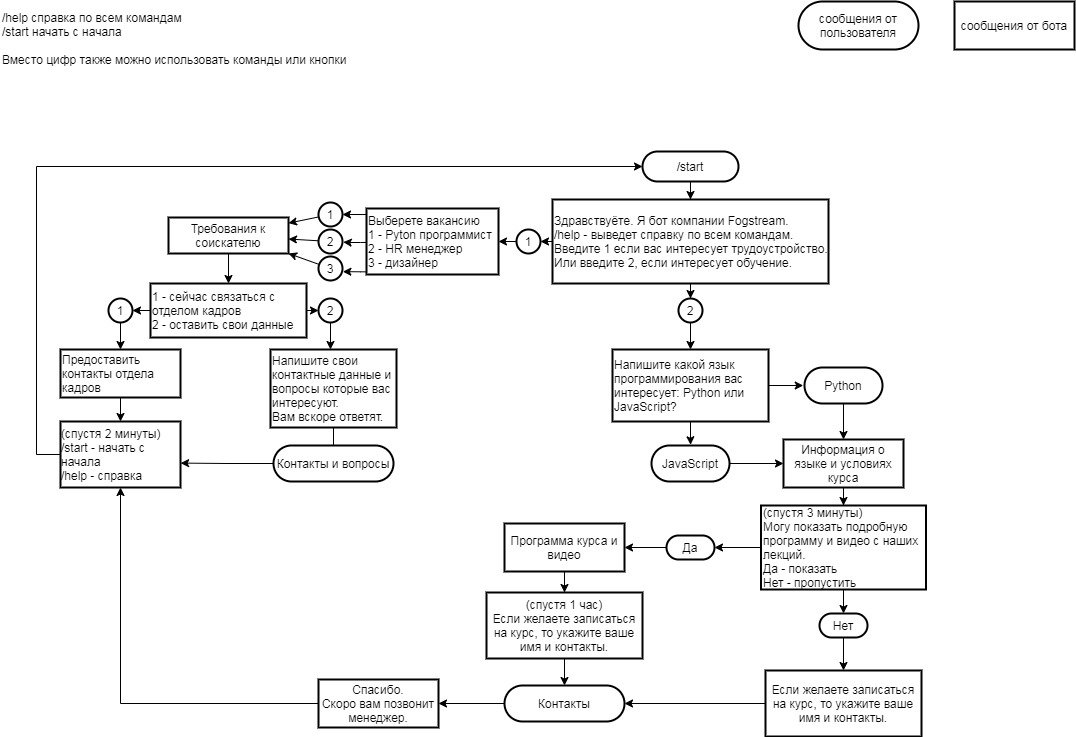


Рисунок 1

Данный бот предназначен для вывода заранее заготовленных сообщений пользователю.

С ботом работают три категории людей: Программист, Администратор, Пользователь.

Программист разрабатывает архитектуру проекта и вносит изменения в код проекта. Проект можно разбить на три части: телеграм бот, API для связки базы данных и бота, а так же веб приложение для создания и редактирования с сообщений.

Диалог выглядит как дерево с узлами из сообщений, которые имеют разные свойства.

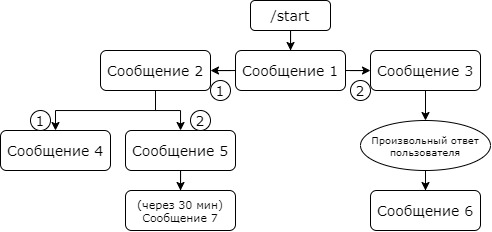


Рисунок 2

Анализ модели диалога (Рисунок 2) показывает, что сообщения бывают вопросительные и повествовательные, а так же могут имеють потомков и не иметь потомков.

Таблица 1 показывает, какие данные (поля) понадобятся вместе с сообщением.

1. pk\_message – номер сообщения, для упрощения совпадает с id сообщения.
2. text\_message – текст сообщения
3. parent\_id – номер предыдущего, родительского, сообщения
4. display\_condition – если родитель это вопросительное сообщение, то данное поле содержит ответ по которому происходит вывод данного сообщения
5. write\_answer – записывать ли ответ на данное сообщение. Если оно содержит True, это автоматически означает, что это вопросительное сообщение
6. delay – задержка которая может быть после данного сообщения

Таблица 1 (message\_table) содержит сообщения в БД

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pk\_message | text\_message | parent\_id | display\_condition | write\_answer | delay |
| 1 | Сообщение 1 | None | None | True | None |
| 2 | Сообщение 2 | 1 | 1 | True | None |
| 3 | Сообщение 3 | 1 | 2 | True | None |
| 4 | Сообщение 4 | 2 | 1 | False | None |
| 5 | Сообщение 5 | 2 | 2 | False | 30 |
| 6 | Сообщение 6 | 3 | None | False | None |
| 7 | Сообщение 7 | 5 | None | False | None |

Также нам понадобятся таблицы, которые будут содержать данные связанные с пользователем и его ответами.

Таблица 2 (answer\_table) содержит ответы пользователя

|  |  |  |
| --- | --- | --- |
| ***answer\_table***. pk\_message | text\_answer | user\_table .user\_id |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

Таблица 3 (user\_table) содержит информацию о пользователе и какое сообщение для него текущее

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| user\_id | user\_name | first\_name | last\_name | message\_table.pk\_message |
| 1 | Жора | Евгений | Петров | 6 |
| 2 | Хацкер | Алексей | Иванов | 3 |

Так же необходимо разработать интерфейс веб приложения, на базе Django, через который Администратор сможет работать с сообщениями.

Администратор, может добавлять, редактировать и удалять сообщения, выстраивая логику **о**бщения бота с пользователем. А также работать с ответами пользователя. В Таблице 4 и Таблице 5 содержатся те поля, которые минимально необходимы.

Таблица 4. Поля формы ввода вопросов в админке.

|  |  |  |  |
| --- | --- | --- | --- |
| Поле ввода текста сообщения (text\_message).  Текстовое поле. | Номер предыдущего сообщения (parent\_id). Текстовое поле. | Ответ на предыдущее сообщение (display\_condition).  Текстовое поле. | Указываем нужно ли сохранить ответ (write\_answer).  CheckBox |

Таблица 5. Поля формы для работы с ответами пользователя

|  |  |  |
| --- | --- | --- |
| Имя Пользователя | Сообщение (вопросительное) | Ответ на сообщение |
| Евгений Петров | Вопрос 1 | Ответ на вопрос 1 |
| Евгений петров | Вопрос 2 | Ответ на вопрос 2 |

Пользователь заходит в Telegram, запускает диалог командой /start, читает сообщения от бота, а после отвечает на них.

Описание принципов работы Telegram бота

Бот должен решать следующие вопросы:

1. Вывод сообщений Пользователю в заданной последовательности,
2. Приём ответа от Пользователя на данное сообщение и передача его на сервер для обработки,
3. Если сообщение было вопросительным с несколькими вариантами ответов, то вывод следующего сообщения с учётом выбранного Пользователем варианта ответа,
4. Поддержка одновременной работы нескольких пользователей.

Для начала нужно определить, какие варианты сообщений могут быть. Тут видим два параметра.

Первый параметр определяет: это сообщение вопрос или нет? Следовательно, два состояния True и False. Если данное сообщение это вопрос, то ответ на него должен быть записан. Тут нам поможет Таблица 1. message\_table.write\_answer принимает True для вопросительных сообщений, а для повествовательных False.

Количество потомков, count\_child, для каждого сообщения можем узнать посчитав в Таблице 1 parent\_id сколько раз встречается id данного сообщения. Это даёт три принципиальных варианта для количества потомков: ноль, один и больше одного.

Перебор всех сочетаний write\_answer и count\_child даёт понимание, как действовать боту:

write\_answer == True and count\_child == 0

это сообщение вопрос и он последний

write\_answer == True and count\_child == 1

это сообщение вопрос, ответ на него произвольный, сообщение не последнее

write\_answer == True and count\_child > 1

это сообщение вопрос с заданными вариантами выбора

write\_answer == False and count\_child == 0

это просто сообщение и оно последнее

write\_answer == False and count\_child == 1

это просто сообщение и оно не последнее

write\_answer == False and count\_child > 1

ошибочное состояние, потомков больше одного, это означает выбор и сообщение является вопросом

Если сообщение не вопрос, то просто выводится следующее по порядку. Обработка вопросительного сообщения зависит от количества потомков для данного сообщения. Количество потомков имеет три варианта и поэтому будет три обработчика, которые срабатывают в зависимости от значения bot\_flag.

Последующее сообщение определяется родительским сообщением и вариантом ответа, если родительское является вопросом. Таблица 1 содержит необходимые поля. Это проще понять, взглянув на Рисунок 2. Сообщение 4 будет выведено пользователю, если текущим сообщением является Сообщение 2 и ответ на Сообщение 2 был дан «1». Сообщение 7 будет выведено без вариантов, если текущим является Сообщение 5. Технически это выглядит так: если текущим сообщением является Сообщение 2 и получен ответ «2». Тогда обратившись к полям parent\_id == 2 и display\_condition == 2, Таблицы 1, однозначно получим для вывода Сообщение 5.

Для работы бота в многопользовательском режиме необходимо для каждого пользователя в БД хранить номер его текущего сообщения. В Таблице 3 для этого есть поле, связанное с message\_table.pk\_message.

Так же для каждого пользователя нужно хранить значение bot\_flag.

Для практической реализации бота была применена асинхронная библиотека aiogram.

Для отправки запросов к серверу используется класс BotServer. Реализованные в нём функции позволяют отправлять на эндпоинты сервера запросы для работы с БД.

Пояснения к коду программы бота.

Пользователь вводит /start

Бот обрабатывает эту команду обработчиком

@dp.message\_handler(commands=['start'])

Из сообщения от пользователя берётся

from\_user.id (он же message.chat.id)

from\_user.username

from\_user.last\_name

from\_user.first\_name

и делает запрос серверу.

Сервер регистрирует пользователя в БД, если пользователя с таким from\_user.id ещё нет.

Если есть такой from\_user.id есть, то пользователь не создается. Например, пользователь запустил /start и решил пройти по другой ветке диалога. Тогда новые ответы перезапишут старые или добавятся те, которых не было. Так же для пользователя устанавливается первое сообщение в качестве текущего.

Вызывается функция checking\_message(message.chat.id) , которой передаётся id пользователя.

Получаем текущее сообщение с базы данных для данного пользователя

cur\_mes = bot\_server.get\_current\_message(ms\_chat\_id)

Сообщение является словарём.

{'id': 61,

'text\_message': Сообщение 1',

'id\_parent': 58,

'display\_condition': 'Да',

'write\_answer': True,

'delay': 900,

'options\_answer': ['2', '1', '3']}

delay - задержка этого сообщения

options\_answer - варианты ответа на сообщение

(если вариантов нет, то None)

id - id\_сообщения,

text\_message - текст сообщения

write\_answer - Вопрос ли это (надо ли отвечать на этот вопрос)

display\_condition - Условие отображения (ответ пользователя на предыдущее сообщение)

Получаем количество потомков для данного сообщения

count\_child = bot\_server.get\_count\_child(cur\_mes['id'])

Далее проверяем, как обработать это сообщение. Если оно не является вопросом, то просто выводим следующее или заканчиваем диалог. Если это вопрос, тогда смотрим на количество потомков, обрабатываем подходящим условным переходом и выставляем флаг для обработчика ответа- bot\_flag.set\_flag\_user(ms\_chat\_id, 3) и ждём ответа пользователя.

При ответе пользователя обработчик срабатывает по определённому флагу.

@dp.message\_handler(lambda message: bot\_flag.get\_flag\_user(message.chat.id) == 3)

Вызывается текущее сообщение для данного пользователя.

cur\_mes = bot\_server.get\_current\_message(message.chat.id)

Для данного пользователя и на текущее сообщения записывается ответ.

bot\_server.set\_answer\_user(message.chat.id, cur\_mes['id'], message.text)

Для данного пользователя устанавливается текущим следующее сообщение.

bot\_server.get\_next\_fullmessage(message.chat.id, cur\_mes['id'], message.text)

Вызывается функция checking\_message(message.chat.id), которая проверит следующее сообщение.