**Аннотация**

*Мы начинаем блок по работе с голосовым помощником от Яндекса — Алисой.*

**Алиса**

Алиса — это голосовой помощник от Яндекса. Фактически, это робот, который умеет общаться с нами посредством компьютера или любого другого вычислительного устройства: мобильного телефона, часов, «умной колонки» и даже системы управления автомобилем. В процессе общения Алиса может отвечать на вопросы, искать информацию в Интернете, играть с нами в различные игры, торговать на бирже, информировать о погоде, распознавать образы, управлять компонентами «умного дома» и делать множество других вещей.

**Навыками** принято называть отдельные задачи, которые Алиса умеет решать. Например, программист может научить Алису заказывать пиццу. Это умение и будет навыком. Сейчас различными разработчиками создано [большое количество навыков](https://dialogs.yandex.ru/store/), мы же с вами сегодня сделаем простой навык под названием «Купи слона», в котором Алиса будет с нами играть в простую игру:

* Купи слона
* Нет
* Все говорят «Нет». А ты купи слона
* Хорошо
* Слона можно найти на Яндекс.Маркете!

И на этом все. Эта игра — пример, который приведен на официальном сайте платформы Яндекс.Диалоги — места, где любой человек может публиковать свои навыки для Алисы.

Но для начала работы узнаем несколько вещей о разработке навыков.

**Интеграция с Алисой**

Первое, в чем нужно разобраться — как интегрироваться с Алисой. Алиса не имеет API в классическом понимании — как, например, у Яндекс.Карт. Интеграция происходит по технологии WebHook (технология — это громкое слово, скорее, идея).

Почему Алиса просто не могла предоставить нам API для работы? Как вы думаете, в чем сложность?

Алиса может обратиться к нашему навыку в любое время, поскольку никто не знает, когда один из миллионов пользователей Алисы решит им воспользоваться. Если бы Алиса предоставляла свое API, мы должны были бы регулярно ее спрашивать: «Эй, Алиса, никто не запрашивал наш навык?», затем при положительном ответе: «Запрашивали? Вот мой ответ. А что там ответил пользователь?» и так далее.

Как же быть?

Оказывается, лучшим решением является такое, в котором Алиса сама сообщит нам обо всех событиях, которые нас касаются. Идея WebHook’ов состоит в том, что мы не обращаемся к API, а реализуем свое, но по правилам, описанным в документации. Даем доступ к созданному API Алисе, и уже она начинает общаться с ним самостоятельно. Получается как бы «интеграция наоборот». В документации Алисы есть строгие требования к нашему API. После того, как API реализован, мы «говорим» Алисе, куда обращаться: сообщаем наш адрес. Если в интерфейсе Алисы кто-то вызвал наш навык, то Алиса сама нам об этом сообщит. Очень удобно! Не надо постоянно «пинать» Алису.

**Логирование (журналирование)**

Представьте ситуацию: вы не можете видеть на мониторе работу вашей программы. У вас просто может не быть монитора. Или программа размещена не на вашем компьютере, а где-то в облаке. И вы понимаете, что программа работает некорректно. Как же узнать, как она работает, и где вкралась ошибка? В таком случае помогает логирование.

**Логирование**

Логирование — это запись и вывод информации о выполняемых операциях, ошибках и других событиях в коде. Это могут быть значения переменных или любой текст.

Логирование может происходить в консоль или файл. Логи выводятся в консоль, например, в процессе отладки приложения. Если вы уже написали программу, которая выполняется на сервере, то записывать логи желательно в файл, чтобы потом их можно было изучить в случае остановки, падения или перезагрузки сервера.

Для логирования в Python используется библиотека logging.

Пример:

import logging

def log():

i = 0

while i < 10:

logging.warning(i)

i += 1

if \_\_name\_\_ == '\_\_main\_\_':

log()

Выводит в консоль:

WARNING:root:0

WARNING:root:1

WARNING:root:2

WARNING:root:3

WARNING:root:4

WARNING:root:5

WARNING:root:6

WARNING:root:7

WARNING:root:8

WARNING:root:9

В этом примере мы используем библиотеку logging с параметрами по умолчанию, поэтому вся информация выводится в консоль. Мы использовали функцию warning(), которая предназначена для привлечения внимания к проблеме, которая еще не считается ошибкой. root — это имя журнала, его можно менять.

Давайте научимся сохранять данные журнала в файл. Еще раз напомним, что это — правильная практика, когда программа используется в боевом режиме (production), так как файл сохранится в случае остановки, падения или перезагрузки сервера.

Для логирования в файл нужно добавить в код следующую строку:

logging.basicConfig(filename='example.log')

Теперь код целиком выглядит так:

import logging

logging.basicConfig(filename='example.log')

def log\_to\_file():

i = 0

while i < 10:

logging.warning(i)

i += 1

if \_\_name\_\_ == '\_\_main\_\_':

log\_to\_file()

После того, как мы запустим программу, в папке программы создастся файл example.log, и в нем окажется та же информация, что ранее выводилась в консоль:

WARNING:root:0

WARNING:root:1

WARNING:root:2

WARNING:root:3

WARNING:root:4

WARNING:root:5

WARNING:root:6

WARNING:root:7

WARNING:root:8

WARNING:root:9

Если мы запустим программу повторно, то увидим, что старые данные из файла не удаляются. Таким образом, информация будет записана дважды.

Чего же не хватает в файле? Что еще нам важно знать? Правильно — время события, ведь без него никакого расследования не получится.

Для того чтобы вывести временную метку, надо в функцию logging.basicConfig() передать еще один параметр — format. Как несложно догадаться, это формат сообщения (как выглядит и что содержит), которое записывается в лог. Сообщение в логе может содержать много полезной информации (подробно можно изучить [тут](https://docs.python.org/3/library/logging.html#logrecord-attributes)). Мы рассмотрим только четыре атрибута, которые можно вывести, и подробно расскажем о них.

С добавлением параметра format наш код будет выглядеть так:

import logging

logging.basicConfig(

filename='example.log',

format='%(asctime)s %(levelname)s %(name)s %(message)s'

)

def log\_to\_file():

i = 0

while i < 10:

logging.warning(i)

i += 1

if \_\_name\_\_ == '\_\_main\_\_':

log\_to\_file()

Рассмотрим параметр format='%(asctime)s %(levelname)s %(name)s %(message)s'.

Здесь:

|  |  |
| --- | --- |
| **asctime** | Время события |
| **levelname** | Уровень логирования (подробно об этом расскажем ниже) |
| **name** | Имя логера (журнала). По умолчанию — root, но вы можете задавать разные имена, чтобы сделать свои логи более информативными. Например, считается хорошей практикой назначать имя файла с кодом |
| **message** | Сообщение, которое вы отправили в logging.warning() |

После запуска мы увидим в файле:

2018-12-23 21:08:58,805 WARNING root 0

2018-12-23 21:08:58,805 WARNING root 1

2018-12-23 21:08:58,805 WARNING root 2

2018-12-23 21:08:58,805 WARNING root 3

2018-12-23 21:08:58,805 WARNING root 4

2018-12-23 21:08:58,806 WARNING root 5

2018-12-23 21:08:58,806 WARNING root 6

2018-12-23 21:08:58,806 WARNING root 7

2018-12-23 21:08:58,806 WARNING root 8

2018-12-23 21:08:58,806 WARNING root 9

**Уровни логирования**

Запись в логи принято разбирать на типы — уровни. Любое сообщение несет в себе информацию определенной важности, и время реакции на сообщения отличается. Приведем примеры:

|  |  |
| --- | --- |
| **Уровень** | **Пример события** |
| **Debug** | Отправлен запрос в базу данных на сохранение |
| **Debug** | Завершен запрос в базу данных на сохранение |
| **Debug** | Запрос в базу занял 0.02 секунды, извлечено 1000 записей |
| **Info** | Зарегистрирован новый пользователь (user\_id = 123123) |
| **Warning** | Отклонена транзакция с суммой платежа 0 |
| **Error** | Ошибка при сохранении данных пользователя (user\_id = 123124, operation\_id = 12312313) |
| **Critical (Fatal)** | Ошибка ответа API Яндекс Карт, код: 404. Маршруты не рассчитываются |

Начинающему программисту бывает сложно определить, к какому уровню следует отнести то или иное событие, возникающее при работе программы. Мы приведем базовые принципы, однако надо иметь в виду, что правила всегда определяет руководитель команды, которая работает над проектом.

|  |  |
| --- | --- |
| **Уровень** | **Относящиеся события** |
| **Debug** | Сообщения отладки. В боевом режиме (production) сообщения этого уровня обычно отключены, чтобы не засорять файлы журналов. Но они могут включаться для поиска багов, которые не удалось воспроизвести |
| **Info** | Обычные сообщения, информирующие о действиях программы. Реагировать на такие сообщения вообще не надо, но они могут помочь, например, при поиске багов, расследовании интересных ситуаций и т. д. |
| **Warning** | Записывая такое сообщение, программа обычно пытается привлечь внимание. Произошло что-то странное. Возможно, это новый тип ситуации, еще не известный на текущий момент. Следует разобраться в том, что произошло, что это означает, и отнести ситуацию либо к инфо-сообщению, либо к ошибке. Соответственно, придется доработать код обработки таких ситуаций |
| **Error** | Ошибка в работе программы, требующая вмешательства. Что-то не сохранилось, что-то отвалилось. Необходимо принимать меры довольно быстро! Ошибки этого уровня и выше требуют немедленной записи в лог, чтобы ускорить реакцию на них |
| **Critical (Fatal)** | Это — особый класс ошибок, приводящих к неработоспособности программы в целом или неработоспособности одного из ее модулей. Чаще всего случаются фатальные ошибки из-за неверной конфигурации или отказов оборудования. Требуют срочной, немедленной реакции |

Имейте в виду, что это — рекомендации. Вы как архитектор программы вправе определять критичность события или ошибки в своей программе, а если работа происходит в команде, то решение о критичности той или иной ситуации принимается руководителем команды или коллегиально.

Разработчик может настраивать уровень логирования. Логируются сообщения установленного уровня и уровней более высокой критичности. Например, если установлен уровень Info, то в консоль или в файл будут выводиться сообщения уровней: Info, Warning, Error и Fatal.

Уровень логирования по умолчанию — WARNING.

Уровень логирования в Python настраивается все в той же функции logging.basicConfig() следующим образом:

import logging

logging.basicConfig(level=logging.DEBUG)

def log():

logging.debug('Debug')

logging.info('Info')

logging.warning('Warning')

logging.error('Error')

logging.critical('Critical or Fatal')

if \_\_name\_\_ == '\_\_main\_\_':

log()

Видим в консоли:

DEBUG:root:Debug

INFO:root:Info

WARNING:root:Warning

ERROR:root:Error

CRITICAL:root:Critical or Fatal

Если вызвать logging.basicConfig(level=logging.ERROR), в консоли мы увидим:

ERROR:root:Error

CRITICAL:root:Critical or Fatal

Используйте логирование в процессе разработки. Это поможет вам правильно и своевременно реагировать на ошибки, искать баги и понимать, что происходит в вашей программе в любой момент времени.

**Пишем код**

Теперь переходим к самому интересному. Начнем! Мы разработаем навык, который описан в разделе [«Быстрый старт»](https://tech.yandex.ru/dialogs/alice/doc/quickstart-python-docpage/) документации Алисы.

Вы уже знакомы с библиотекой Flask. С ее помощью мы напишем небольшой веб-сервер, который будет обрабатывать запросы от Алисы.

Алиса будет передавать нам JSON, содержащий данные о пользователе, и информацию, которую пользователь ввел. Мы же должны будем вернуть в ответ свой JSON (соответственно документации). Алиса его обработает и покажет пользователю.

Рассмотрим JSON запроса, который отправит нам Алиса. На самом деле JSON выглядит сложнее и имеет больше полей, но мы рассмотрим только те из них, которые будем использовать (внимательно изучить содержание JSON’ов можно в [документации](https://yandex.ru/dev/dialogs/alice/doc/protocol-docpage/)):

{

"request": {

"command": "закажи пиццу на улицу льва толстого, 16 на завтра",

"original\_utterance": "закажи пиццу на улицу льва толстого, 16 на завтра"

},

"session": {

"new": true,

"message\_id": 4,

"session\_id": "2eac4854-fce721f3-b845abba-20d60",

"skill\_id": "3ad36498-f5rd-4079-a14b-788652932056",

"user\_id": "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"

},

"version": "1.0"

}

* request — данные, полученные от пользователя
  + command — запрос, который был передан вместе с командой активации навыка. Например, если пользователь активирует навык словами «спроси у Сбербанка, где ближайшее отделение», в этом поле будет передана строка «где ближайшее отделение»
  + original\_utterance — полный текст пользовательского запроса, максимум 1024 символа
* session — данные о сессии (разговоре с Алисой)
  + new — признак новой сессии. Возможные значения: true — пользователь начинает новый разговор с навыком, false — запрос отправлен в рамках уже начатого разговора
  + message\_id — идентификатор сообщения в рамках сессии, максимум 8 символов. Инкрементируется (увеличивается на единицу) с каждым следующим запросом
  + session\_id — уникальный идентификатор сессии, максимум 64 символа
  + skill\_id — идентификатор вызываемого навыка, присвоенный при создании
  + user\_id — идентификатор экземпляра приложения, в котором пользователь общается с Алисой, максимум 64 символа. Даже если пользователь авторизован с одним и тем же аккаунтом в приложении Яндекс для Android и iOS, Яндекс.Диалоги (так называется сервис Яндекса, управляющий навыками для Алисы) присвоят отдельный user\_id каждому из этих приложений
* version — версия протокола. Текущая версия — 1.0

JSON ответа Алисы (то есть тот JSON, что сформирует наша программа) будет выглядеть так (конечно же, полное описание можно посмотреть в [документации](https://tech.yandex.ru/dialogs/alice/doc/protocol-docpage/)):

{

"response": {

"text": "Здравствуйте! Это мы, хороводоведы.",

"tts": "Здравствуйте! Это мы, хоров+одо в+еды.",

"buttons": [

{

"title": "Надпись на кнопке",

"payload": {},

"url": "https://example.com/",

"hide": true

}

],

"end\_session": false

},

"session": {

"session\_id": "2eac4854-fce721f3-b845abba-20d60",

"message\_id": 4,

"user\_id": "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"

},

"version": "1.0"

}

* response — данные для ответа пользователю
  + text — текст, который следует показать. Максимум 1024 символа. Не должен быть пустым
  + tts — текст, который следует сказать пользователю. Максимум 1024 символа. Не должен быть пустым
  + buttons — массив кнопок, которые следует показать пользователю. Все указанные кнопки выводятся после основного ответа Алисы, описанного в свойстве text. Кнопки можно использовать как релевантные ответу ссылки или подсказки для продолжения разговора
  + title — текст кнопки, обязателен для каждой кнопки. Максимум 64 символа
  + url — URL, который должна открывать кнопка, максимум 1024 байта
  + payload — значения, которые передадутся в программу после нажатия этой кнопки
  + hide — признак того, что кнопку нужно убрать после следующей реплики пользователя. Допустимые значения: false — кнопка должна оставаться активной (значение по умолчанию), true — кнопку нужно скрывать после нажатия
  + end\_session — признак конца разговора. Допустимые значения: false — сессию следует продолжить, true — сессию следует завершить
* session — данные о сессии (аналогичные запросу)
* version — версия протокола. Текущая версия — 1.0

Вы можете скачать [файл с шаблоном кода](https://yastatic.net/s3/lyceum/content/resources/flask_app.py) или скопировать текст программы ниже.

# импортируем библиотеки

from flask import Flask, request

import logging

# библиотека, которая нам понадобится для работы с JSON

import json

# создаём приложение

# мы передаём \_\_name\_\_, в нем содержится информация,

# в каком модуле мы находимся.

# В данном случае там содержится '\_\_main\_\_',

# так как мы обращаемся к переменной из запущенного модуля.

# если бы такое обращение, например,

# произошло внутри модуля logging, то мы бы получили 'logging'

app = Flask(\_\_name\_\_)

# Устанавливаем уровень логирования

logging.basicConfig(level=logging.INFO)

# Создадим словарь, чтобы для каждой сессии общения

# с навыком хранились подсказки, которые видел пользователь.

# Это поможет нам немного разнообразить подсказки ответов

# (buttons в JSON ответа).

# Когда новый пользователь напишет нашему навыку,

# то мы сохраним в этот словарь запись формата

# sessionStorage[user\_id] = {'suggests': ["Не хочу.", "Не буду.", "Отстань!" ]}

# Такая запись говорит, что мы показали пользователю эти три подсказки.

# Когда он откажется купить слона,

# то мы уберем одну подсказку. Как будто что-то меняется :)

sessionStorage = {}

@app.route('/post', methods=['POST'])

# Функция получает тело запроса и возвращает ответ.

# Внутри функции доступен request.json - это JSON,

# который отправила нам Алиса в запросе POST

def main():

logging.info(f'Request: {request.json!r}')

# Начинаем формировать ответ, согласно документации

# мы собираем словарь, который потом при помощи

# библиотеки json преобразуем в JSON и отдадим Алисе

response = {

'session': request.json['session'],

'version': request.json['version'],

'response': {

'end\_session': False

}

}

# Отправляем request.json и response в функцию handle\_dialog.

# Она сформирует оставшиеся поля JSON, которые отвечают

# непосредственно за ведение диалога

handle\_dialog(request.json, response)

logging.info(f'Response: {response!r}')

# Преобразовываем в JSON и возвращаем

return json.dumps(response)

def handle\_dialog(req, res):

user\_id = req['session']['user\_id']

if req['session']['new']:

# Это новый пользователь.

# Инициализируем сессию и поприветствуем его.

# Запишем подсказки, которые мы ему покажем в первый раз

sessionStorage[user\_id] = {

'suggests': [

"Не хочу.",

"Не буду.",

"Отстань!",

]

}

# Заполняем текст ответа

res['response']['text'] = 'Привет! Купи слона!'

# Получим подсказки

res['response']['buttons'] = get\_suggests(user\_id)

return

# Сюда дойдем только, если пользователь не новый,

# и разговор с Алисой уже был начат

# Обрабатываем ответ пользователя.

# В req['request']['original\_utterance'] лежит весь текст,

# что нам прислал пользователь

# Если он написал 'ладно', 'куплю', 'покупаю', 'хорошо',

# то мы считаем, что пользователь согласился.

# Подумайте, всё ли в этом фрагменте написано "красиво"?

if req['request']['original\_utterance'].lower() in [

'ладно',

'куплю',

'покупаю',

'хорошо'

]:

# Пользователь согласился, прощаемся.

res['response']['text'] = 'Слона можно найти на Яндекс.Маркете!'

res['response']['end\_session'] = True

return

# Если нет, то убеждаем его купить слона!

res['response']['text'] = \

f"Все говорят '{req['request']['original\_utterance']}', а ты купи слона!"

res['response']['buttons'] = get\_suggests(user\_id)

# Функция возвращает две подсказки для ответа.

def get\_suggests(user\_id):

session = sessionStorage[user\_id]

# Выбираем две первые подсказки из массива.

suggests = [

{'title': suggest, 'hide': True}

for suggest in session['suggests'][:2]

]

# Убираем первую подсказку, чтобы подсказки менялись каждый раз.

session['suggests'] = session['suggests'][1:]

sessionStorage[user\_id] = session

# Если осталась только одна подсказка, предлагаем подсказку

# со ссылкой на Яндекс.Маркет.

if len(suggests) < 2:

suggests.append({

"title": "Ладно",

"url": "https://market.yandex.ru/search?text=слон",

"hide": True

})

return suggests

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Сохраните ваш файл под именем flask\_app.py.

**Как работает данный код**

Разберем тезисно, как работает приведенная программа.

1. Когда мы запускаем программу, точкой входа в нее со стороны Алисы является функция main(), которая «обернута» декоратором app.route.
2. Внутри main() мы сначала логируем полученный запрос, затем начинаем формировать ответ, потом вызываем функцию handle\_dialog() для обработки диалога с пользователем, в результате же логируем и возвращаем ответ.
3. Функция handle\_dialog() формирует ответ, исходя из данных запроса и состояния сессии (новый разговор или продолжение старого).
4. Вспомогательная функция get\_suggests() формирует подсказки в ответе.
5. В глобальном словаре sessionStorage мы будем хранить информацию о подсказках для каждого пользователя.

**Отладка с Postman**

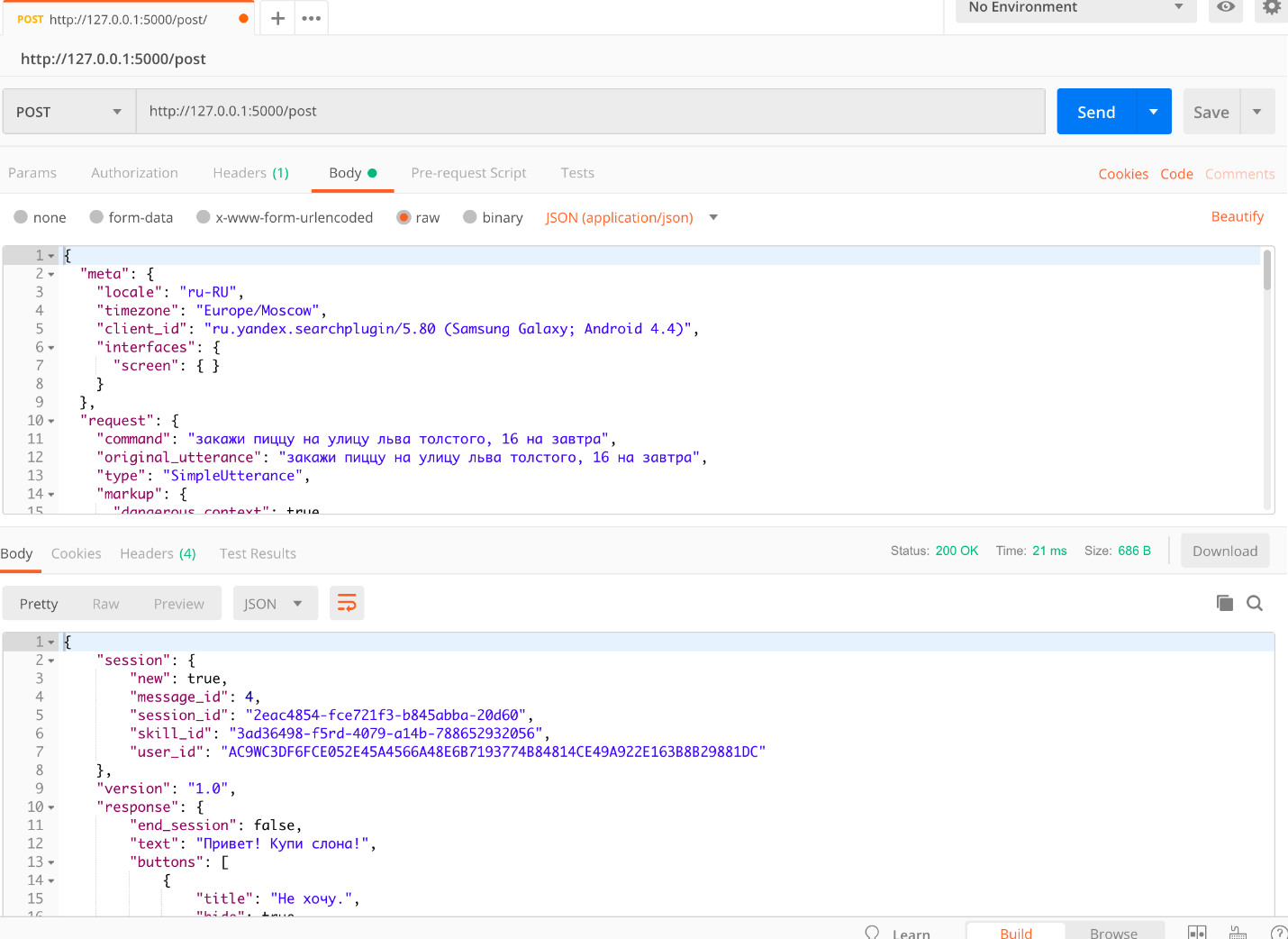
Перед тем, как продолжить, нам надо убедиться, что созданный нами веб-сервер возвращает правильный ответ (или похожий на правильный). Воспользуемся для этого программой [Postman](https://www.getpostman.com/).

Postman — это приложение, которое позволяет взаимодействовать с API без написания кода. Это бывает удобно, чтобы находить ошибки и «пробовать» новое API перед разработкой. Postman не единственное подобное приложение, можете попробовать еще, например, [Insomnia](https://insomnia.rest/).

Сейчас мы проверим ответ от нашего приложения.

Запустите созданное ранее приложение в среде разработки и скопируйте адрес вида http://127.0.0.1:5000/post в строку запроса программы Postman.

JSON запроса можно взять из [документации](https://tech.yandex.ru/dialogs/alice/doc/protocol-docpage/) Алисы или, на крайний случай, из рассмотренного нами примера.



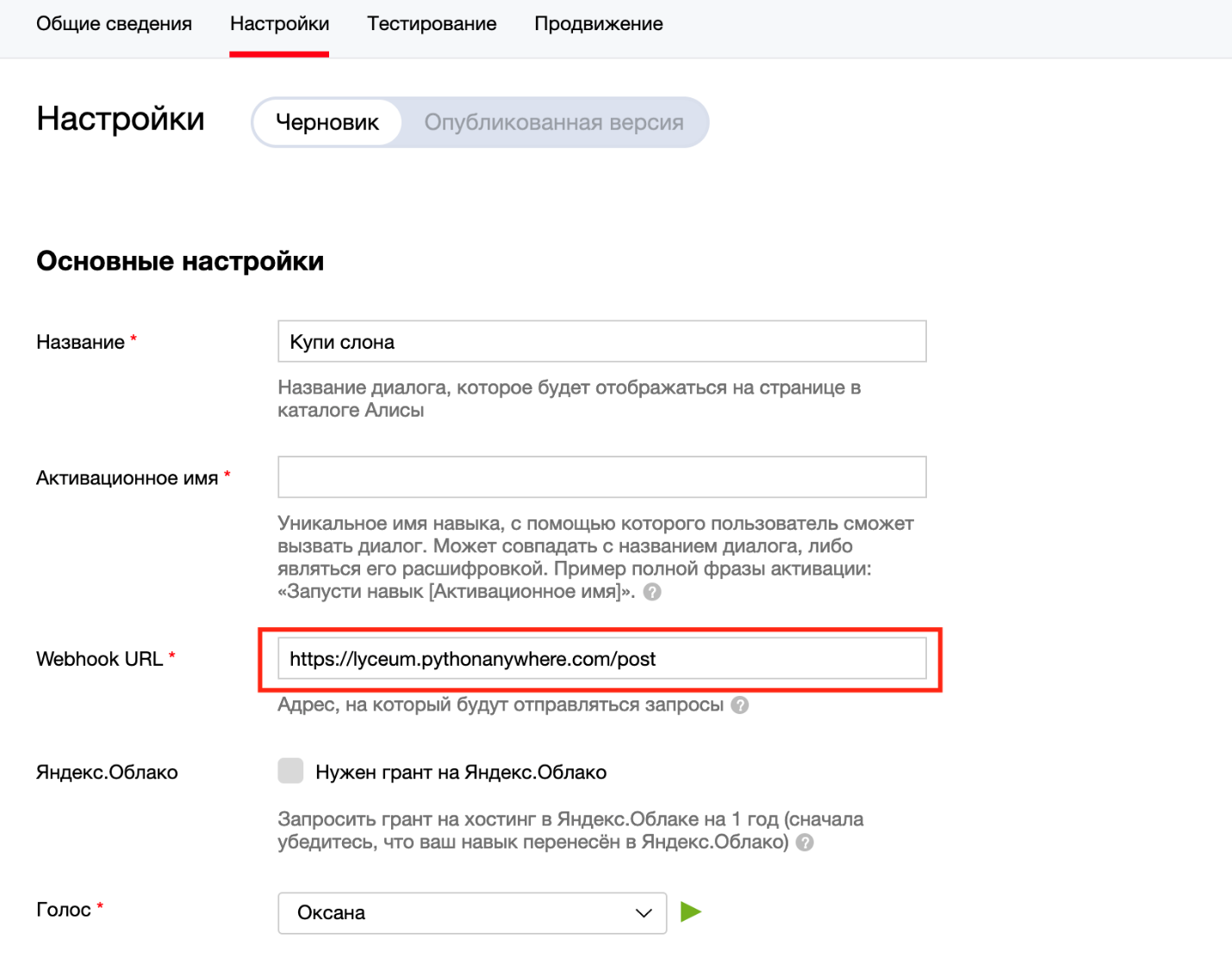
Проверьте, что ваш ответ похож на тот, что требует документация Алисы. Если это так, то поздравляем, мы добились результата :)

Для следующего шага нам понадобится, чтобы наш навык был доступен из Интернета, поэтому давайте либо задеплоим (разместим) его на Heroku, либо просто сделаем туннель с помощью ngrok.

**Регистрация и тестирование навыка**

Перейдем к завершающему и самому простому пункту. Это регистрация навыка в Алисе.

1. Зарегистрируйтесь в Яндексе или залогиньтесь, если вы уже зарегистрированы.
2. Перейдите по ссылке <https://dialogs.yandex.ru/developer/>.
3. Нажмите **создать диалог**.
4. Выберите вариант **Навык в Алисе**.
5. Заполните «активационное имя». По этой фразе Алиса будет вызывать ваш навык.
6. Заполните поля, указав ссылку в поле Webhook URL.



Перейдите на вкладку **Тестирование** и проверьте работу своего навыка.

**Именованные сущности**

Часто при создании навыка из ответа пользователя необходимо извлечь некоторую информацию, и не всегда это делается просто. Но разработчики Алисы уже позаботились о нас и сами выделяют некоторые такие сущности, которые в терминах Алисы называются **именованными**.

Под именованными сущностями подразумеваются:

* Имена
* Фамилии
* Названия городов и т. д.

Когда пользователь в своем сообщении использует имя, фамилию или город, эти данные попадают в специальный раздел JSON’а, который отправляет нам Алиса.

Рассмотрим пример:

{

"meta": {

"locale": "ru-RU",

"timezone": "Europe/Moscow",

"client\_id": "ru.yandex.searchplugin/5.80 (Samsung Galaxy; Android 4.4)",

"interfaces": {

"screen": { }

}

},

"request": {

"command": "закажи пиццу на улицу льва толстого, 16 на завтра",

"original\_utterance": "закажи пиццу на улицу льва толстого, 16 на завтра",

"type": "SimpleUtterance",

"markup": {

"dangerous\_context": true

},

"payload": {},

"nlu": {

"tokens": [

"закажи",

"пиццу",

"на",

"льва",

"толстого",

"16",

"на",

"завтра"

],

"entities": [

{

"tokens": {

"start": 2,

"end": 6

},

"type": "YANDEX.GEO",

"value": {

"house\_number": "16",

"street": "льва толстого"

}

},

{

"tokens": {

"start": 3,

"end": 5

},

"type": "YANDEX.FIO",

"value": {

"first\_name": "лев",

"last\_name": "толстой"

}

},

{

"tokens": {

"start": 5,

"end": 6

},

"type": "YANDEX.NUMBER",

"value": 16

},

{

"tokens": {

"start": 6,

"end": 8

},

"type": "YANDEX.DATETIME",

"value": {

"day": 1,

"day\_is\_relative": true

}

}

]

}

},

"session": {

"new": true,

"message\_id": 4,

"session\_id": "2eac4854-fce721f3-b845abba-20d60",

"skill\_id": "3ad36498-f5rd-4079-a14b-788652932056",

"user\_id": "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"

},

"version": "1.0"

}

Обратите внимание на раздел request → nlu → tokens:

"tokens": [

"закажи",

"пиццу",

"на",

"льва",

"толстого",

"16",

"на",

"завтра"

]

Можно заметить, что это полученный от пользователя текст, разобранный на отдельные слова. Каждое слово при этом приводится к нижнему регистру. Это очень удобно, если надо искать вхождения каких-то определенных слов. Например, «да» или «нет».

Второй раздел, на который стоит обратить внимание — это entities (то есть **сущности**):

"entities": [

{

"tokens": {

"start": 2,

"end": 6

},

"type": "YANDEX.GEO",

"value": {

"house\_number": "16",

"street": "льва толстого"

}

},

{

"tokens": {

"start": 3,

"end": 5

},

"type": "YANDEX.FIO",

"value": {

"first\_name": "лев",

"last\_name": "толстой"

}

},

{

"tokens": {

"start": 5,

"end": 6

},

"type": "YANDEX.NUMBER",

"value": 16

},

{

"tokens": {

"start": 6,

"end": 8

},

"type": "YANDEX.DATETIME",

"value": {

"day": 1,

"day\_is\_relative": true

}

}

]

Мы видим, что текст от пользователя подробно разобран и из него выделены все сущности от имени до даты.

entities — это массив таких сущностей. Рассмотрим одну сущность и подробно распишем, из чего же она состоит:

{

"tokens": {

"start": 3,

"end": 5

},

"type": "YANDEX.FIO",

"value": {

"first\_name": "лев",

"last\_name": "толстой"

}

}

* tokens — обозначение начала и конца именованной сущности в массиве слов (tokens, что мы рассматривали выше). Нумерация слов в массиве начинается с 0
  + start — первое слово именованной сущности
  + end — последнее слово после именованной сущности
* type — тип именованной сущности. Возможные значения:
  + YANDEX.DATETIME — дата и время
  + YANDEX.FIO — фамилия, имя и отчество
  + YANDEX.GEO — местоположение (адрес или аэропорт)
  + YANDEX.NUMBER — число, целое или с плавающей точкой
* value — формальное описание именованной сущности (мы приведем ниже пример для типа YANDEX.FIO. Описание остальных типов ищите в [документации](https://yandex.ru/dev/dialogs/alice/doc/naming-entities-docpage/))
  + first\_name — имя
  + last\_name — фамилия

Пример кода, получающий имя человека из JSON:

def get\_first\_name(req):

# перебираем сущности

for entity in req['request']['nlu']['entities']:

# находим сущность с типом 'YANDEX.FIO'

if entity['type'] == 'YANDEX.FIO':

# Если есть сущность с ключом 'first\_name',

# возвращаем ее значение.

# Во всех остальных случаях возвращаем None.

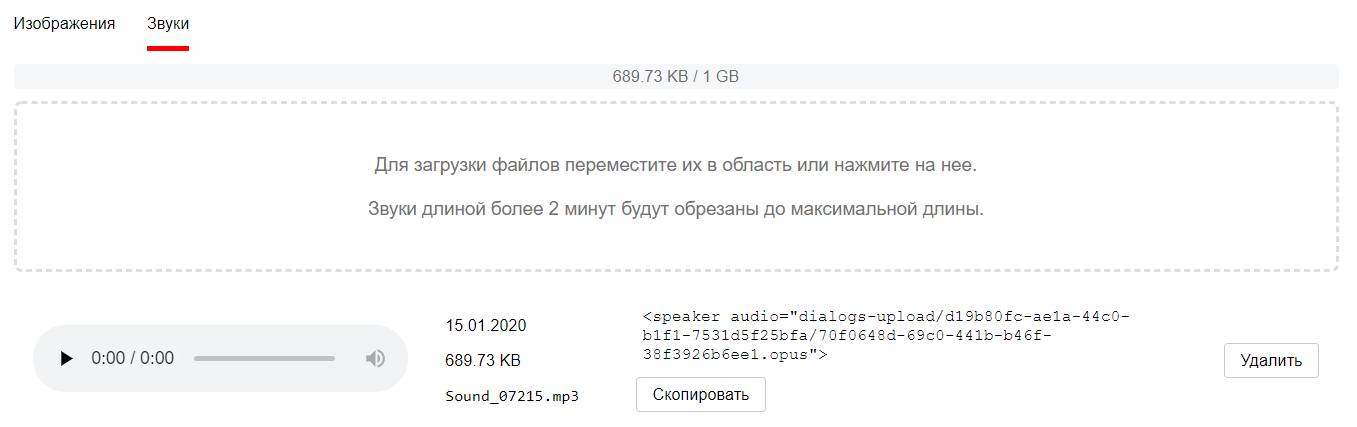
return entity['value'].get('first\_name', None)

**Ресурсы**

Для каждого навыка Алиса позволяет размещать до 100 MB дополнительных медиафайлов (изображения и звуки), но с некоторыми ограничениями:

* Изображение должно быть размером от 1 KB до 1 MB
* Звуковой файл должен быть не длиннее 2-х минут

Ресурсы можно добавить на вкладке «Ресурсы» в разделе управления навыком. Как работать с изображениями мы рассмотрим на следующем уроке, а пока давайте добавим какой-нибудь звук.

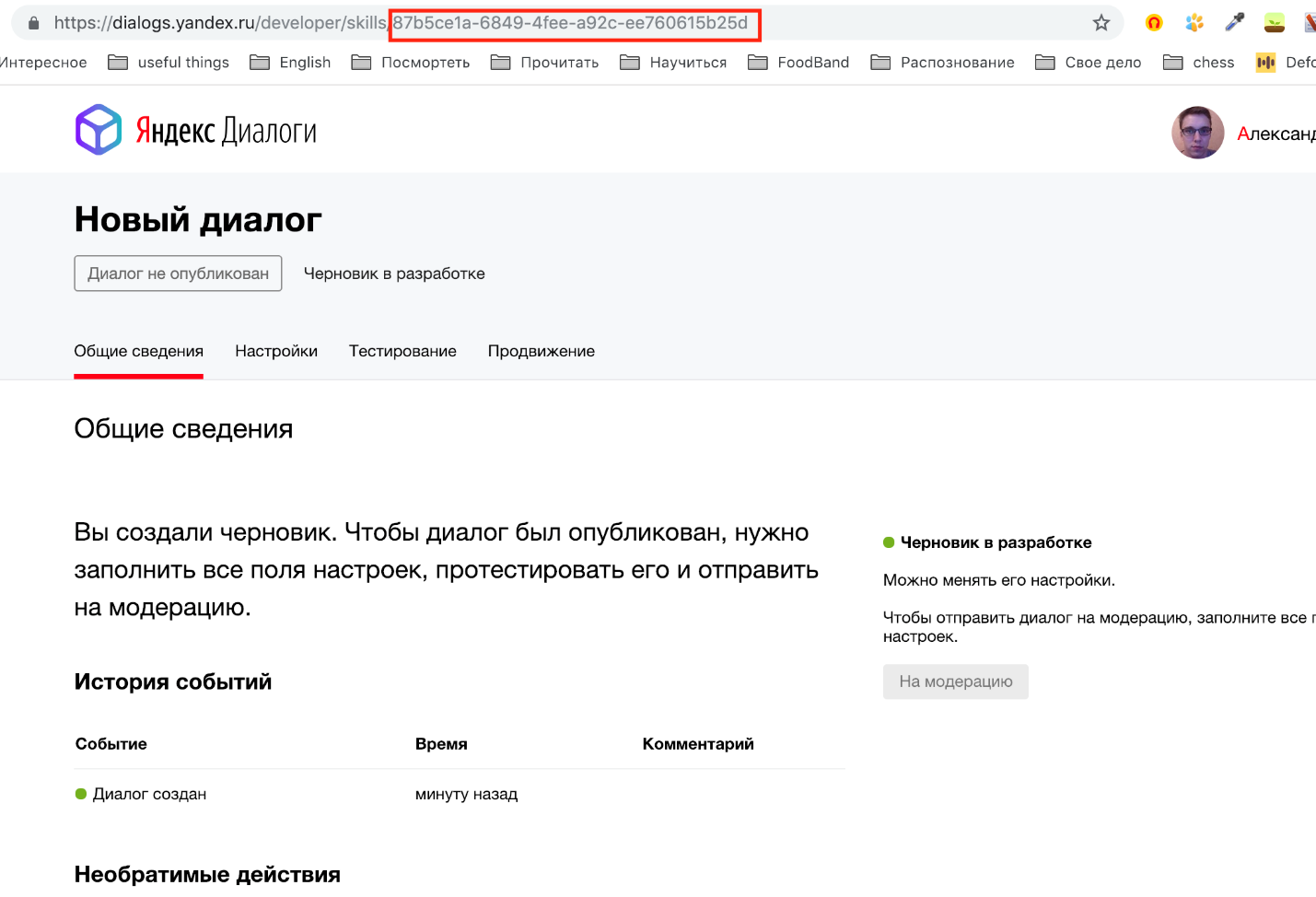


После загрузки звукового файла добавится код его вставки в ответ. Чтобы добавить проигрывание звукового файла, надо добавить этот код в поле tts ответа.

Загружать ресурсы к навыку можно не только через веб-интерфейс, но и с помощью [HTTP API Алисы](https://tech.yandex.ru/dialogs/alice/doc/resource-upload-docpage/). Можно написать код с использованием библиотеки requests или воспользоваться Postman.

Идем по пунктам:

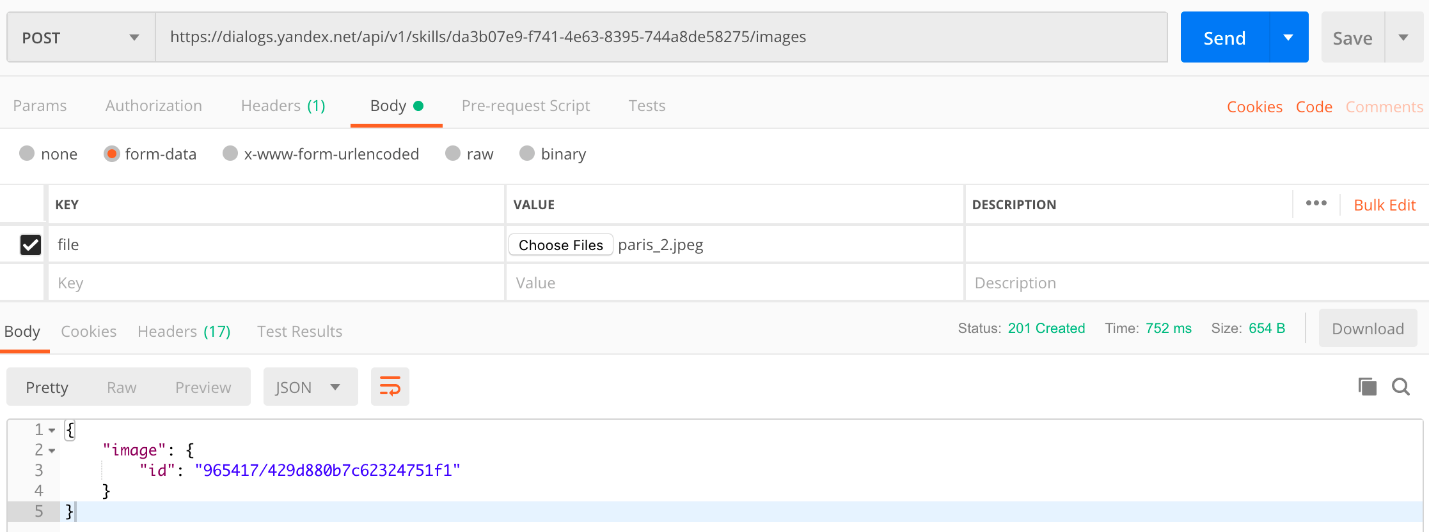
1. Создадим новый навык. Зайдем на [платформу диалогов](https://dialogs.yandex.ru/developer/) и нажмем **создать новый навык**
2. Скопируем из URL и сохраним его идентификатор (ID):



1. Получим token по [ссылке](https://oauth.yandex.ru/authorize?response_type=token&client_id=c473ca268cd749d3a8371351a8f2bcbd). Сохраните его тоже. Он вам пригодится
2. Теперь запускаем Postman
   * Заполняем поля на вкладке Headers
   * Добавьте параметр с названием Authorization. Для этого в колонке KEY пишем Authorization, а в колонке VALUE — «OAuth Token» (OAuth пробел token, который вы получили)
   * В поле ссылки добавляем ссылку https://dialogs.yandex.net/api/v1/skills/id\_навыка/images (заменить id\_навыка на id из пункта 2) для изображения или https://dialogs.yandex.net/api/v1/skills/id\_навыка/sounds для звукового файла



1. Теперь на вкладке Body выбираем радио-кнопку form-data и добавляем файл. В колонке key напишите «file», в колонке value нажмите «choose file» и выберите файл с вашего компьютера. Далее нажимаем Send.



Если все сделано правильно, в ответе вы получите идентификатор картинки или звукового файла, который вы сможете использовать в дальнейшем.