

Разработка навыка для Маруси (на JAICP)



План занятия

Знакомство с CAILA

- что такое Caila
- интенты
- сущности
- slot-filling

Практика: slot-filling в чат-боте

HTTP-запросы

Как выполнить запрос

Как распарсить ответ

Практика: учим чат-бота сообщать прогноз погоды

<https://app.jaicp.com/>



Что такое CAILA?

CAILA (Conversational AI Linguistic Assistant) — NLU-ядро для разработки на базе платформы JAICP, выполняет анализ запросов пользователя на естественном языке и обнаружение сущностей и интенгов.

CAILA позволяет:

- использовать **активационное правило intent**.
- использовать в сценариях **системные и кастомные сущности**.



Подключение и настройка CAILA

Создать новый проект JAISР



Создать с нуля



Использовать шаблон

Выберите из нашей библиотеки



Подключиться к Git-проекту

К существующему проекту из Git-репозитория



Загрузить архив

Загрузите проект с вашего устройства

Расскажите о своем проекте

Название проекта

Например, Бот для прогноза погоды

Язык NLU

 Русский

Шаблон

Проект для CAILA

Проект для CAILA

Шаблон проекта с конфигурацией для CAILA

Пустой проект

Пустой проект. На каждую фразу отвечает "Вы сказали: \$Text"

Группы примеров





Проект классификатором, который отвечает на самые частые вопросы.



Подключение и настройка CAILA

Доступные алгоритмы:

- STS (Semantic Text Similarity)
- Classic ML
- Deep Learning

Наименование	Размещение	Классификатор	Настройки NLU	API-токены	Справка ?
Алгоритм классификатора 		STS			
Исправление орфографии		<input type="checkbox"/>			
Часовой пояс 		UTC			
API-ключ CAILA 		Ключ		Сгенерировать	
Импорт проекта CAILA		Прикрепите файл или перетащите его сюда			



Подключение и настройка CAILA

language — язык классификатора

noMatchThreshold — параметр, задающий минимально необходимую похожесть фразы на один из классов (от 0 до 1)

```
botEngine: v2
language: en

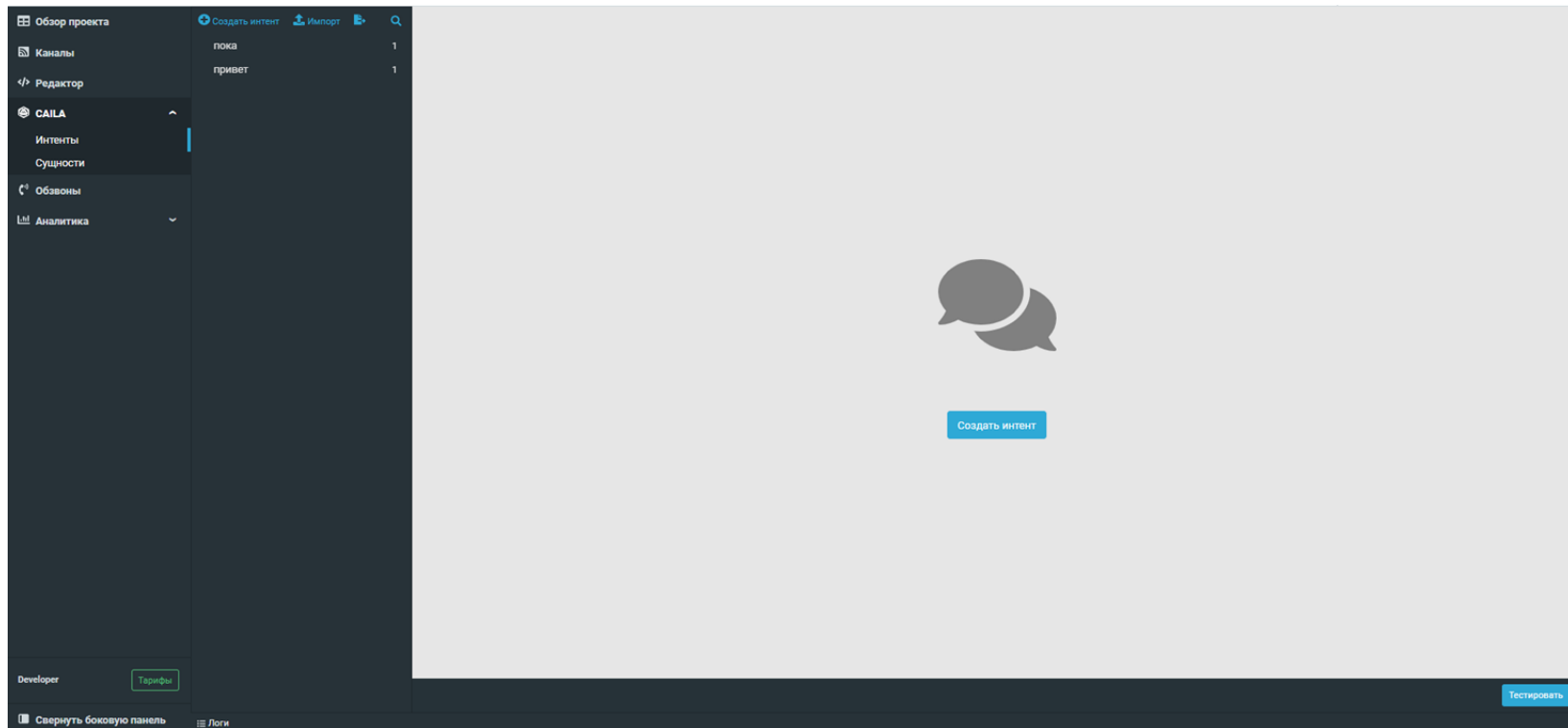
sts:
  noMatchThreshold: 0.2

caila:
  noMatchThreshold: 0.1
```



Создание интента

Интент — ключевая единица NLU-сервиса, объединяющая в себе набор фраз, намерение пользователя и другую метainформацию.





Создание интента

Название

Новый интент

/Новый интент

Комментарий

Ответ

Слоты

Добавить слот

Список слотов пуст

Тренировочные фразы

”

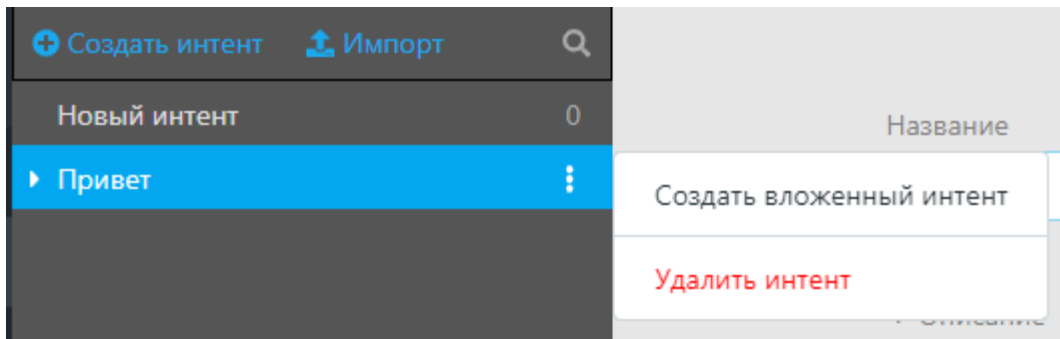
Enter

”

Список фраз пуст



Создание вложенного интента



- Интенты отсортированы в лексикографическом порядке
- Для каждого узла указывается число примеров фраз в данном интенте и через символ / число всех дочерних фраз.



Наполнение интентов

- текстовая фраза



хочу заказать еду

- Паттерны



[хоч*] * (заказать/заказ) * еду

Рекомендации:

- От 10 тренировочных фраз на интент
- В примерах не должно быть фраз приветствия, прощания, показателей вежливости, частотных слов, мата
- Фразы не должны быть слишком длинными или короткими
- Максимальное разнообразие формулировок



Интененты в сценарии

- Теги **intent!** и **intent**:

```
theme: /


state: Hello
  intent!: /hi
  a: Здравствуйте.

state: Goodbye
  intent: /bye
  a: До свидания.

state: CatchAll
  event: noMatch
  a: Вы сказали: {{ $request.query }}
```



Экспорт и импорт интентов

- **Экспорт:** список проектов >  Скачать >
caila_import.json
- **Импорт:** *Интененты* > *Импорт* >
загрузить файл

```
[
  {
    "id": 38886,
    "path": "/City",
    "description": "Название города",
    "answer": "Подождите, загружаю погоду в этом городе.",
    "enabled": true,
    "phrases": [
      {
        "text": "@mystem.geo"
      }
    ],
    "patterns": [],
    "slots": [
      {
        "name": "geo",
        "entity": "mystem.geo",
        "required": true,
        "prompts": [
          "Введите ваш город.",
          "В каком городе вы хотели бы узнать прогноз?"
        ],
        "array": false
      }
    ]
  }
]
```



Пополнение интенгов из диалогов

Добавьте фразу в нужный интенг или создайте новый интенг.

Не забудьте прописать новый интенг в сценарии!

Клиент: Не помню

Бот: Пожалуйста, отправьте ваш номер телефона для проверки статуса заказа 

[Назад](#)

Метки

Оценка

[Неполный ответ](#)

[Новая тема](#)

[Тема перепутана](#)

[Офф-топик](#)

[Полный ответ](#)

[Подсказка помогла](#)

[Подсказка не помогла](#)

[Сложная формулировка](#)

[Мусор](#)

[Затрудняюсь](#)

[Баг в сценарии](#)

[+ Добавить](#)

Прочее

[Есть ошибки](#)

[+ Добавить](#)

Тональность

[положительная](#)

[отрицательная](#)

[нейтральная](#)

[+ Добавить](#)

[Добавить комментарий](#)

[Добавить задачу](#)

[Добавить фразу в интенг](#)

ПРАКТИКА



Механизм активации правил

При использовании в одном сценарии паттернов, интенгов и групп примеров, обработка правил активаций происходит с приоритетом по мере убывания:

1. Паттерны.
2. Группа примеров классификатора STS.
3. Интенды.

Для непредусмотренных сценарием запросов пользователя используйте **event: noMatch**:

```
state: CatchAll
  event: noMatch
  a: Вы сказали: {{ $request.query }}
```



Сущности

Сущность — единица NLU-ядра CAILA. Представляет собой последовательность слов, объединенных некоторым смыслом или правилом. Например: имена, дата и время, местоположение и пр.

CAILA предоставляет для работы:

- *Системные сущности*
- *Пользовательские сущности*



Системные сущности

Системные сущности — встроенные сущности, которые разработчик может активировать в редакторе сущностей.

Системные сущности

mystem.geo
Географическое название

Примеры

Исходный текст	Значение
в москве	москве
Китай - великая страна	Китай

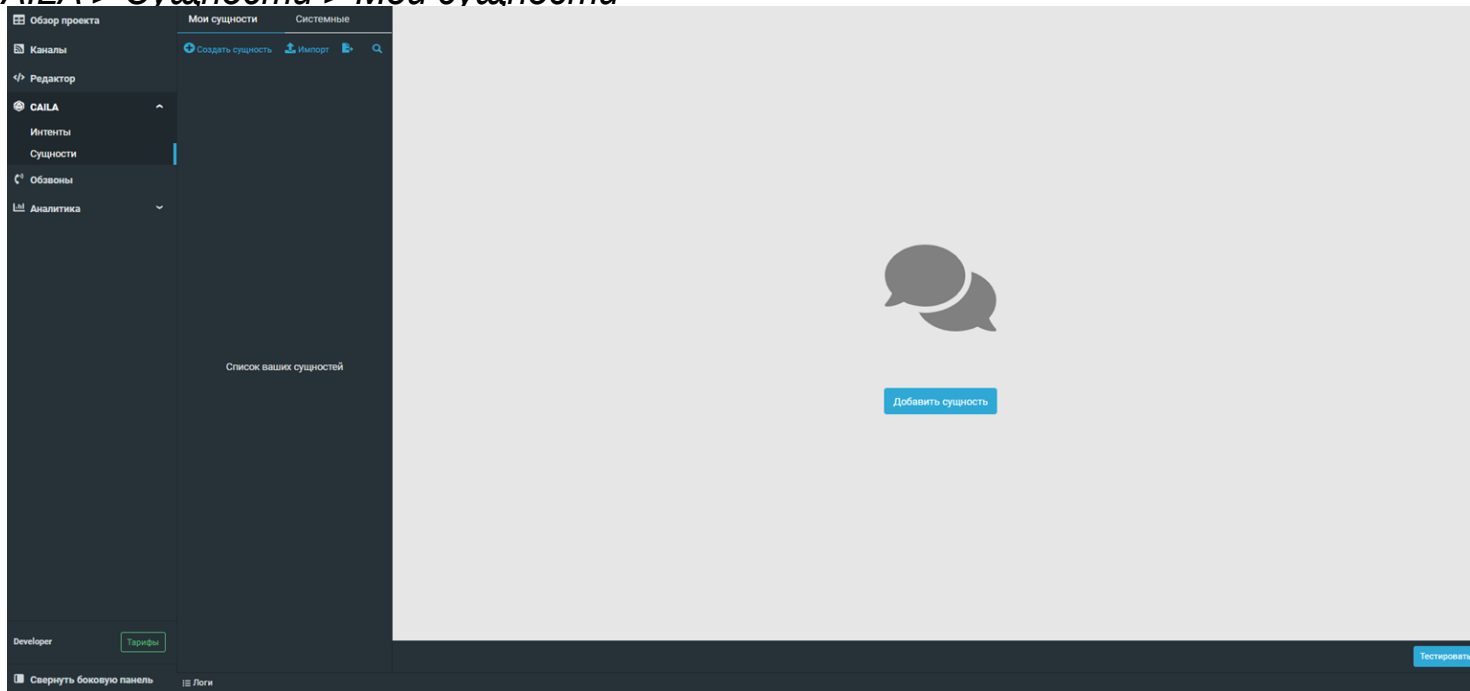
Тестировать



Пользовательские сущности

Пользовательские сущности — сущности, которые разработчик заполняет и настраивает в редакторе сущностей самостоятельно.

CAILA > Сущности > Мои сущности





Пользовательские сущности

- *Название* — укажите имя сущности.
- *Включить распознавание сущности* - включенная сущность будет распознаваться в диалоге
- *Сделать сущность клиентской* - клиент может персонализировать сущность в течение диалога
- *Приводить слова к нормальной форме* - происходит “мягкое” сравнение слова с сущностью.
- *Использовать нечёткий поиск/поиск по подстроке* - при поиске сущности в запросе допускается вариативность.
- *Автоматически расширять интенты* - в обучающих фразах синонимы сущности автоматически расширяются до всех слов, которые распознает данная сущность.

Название

Новая_сущность

распознается | клиентская

Настройки

Прочитайте [о настройках сущности](#)

☒ Включить распознавание сущности

☐ Сделать сущность клиентской
Разрешить пользователю бота добавлять данные в эту сущность

Исправление орфографии

Не исправлять опечатки

Открыть свойства проекта?

Дополнительные настройки:

☒ Приводить слова к нормальной форме
Перед началом поиска все слова введенной пользователем фразы будут приведены к нормальной форме

☐ Использовать нечеткий поиск / поиск по подстроке

☐ Автоматически расширять интенты



Пользовательские сущности

- *Синонимы* — укажите набор синонимов: все варианты написания, которые считаются эквивалентными данному значению.
- *Паттерны* — укажите паттерн: формальное правило, описывающее ключевые слова и выражения.

Справочник

[+ Добавить запись](#)

[Импорт справочника](#)

☐ Выбрано: 0

☐

Синонимы

Синонимы

Pattern

[+ DATA](#)

Свернуть



Пользовательские сущности

Дополнительно вы можете указать **DATA** — значение сущности, в формате string или JSON.

JSON



```
{  
  "name": "Анастасия",  
  "full": "Анастасия",  
  "diminutive": "Настенька",  
  "sex": "ж"
```

Отмена

Сохранить

ПРАКТИКА



Сущности в сценарии

1. Использование сущностей напрямую

```
state:
  q!: Информация по продукту *
  a: Информация по продукту: {{ $entities[0].value }}
```

2. Использование сущностей в паттернах

```
state:
  q!: * @Product::p1 *
  a: Информация по продукту: {{ $parseTree._p1 }}
```

3. Создание и заполнение слотов в интентах



Заполнение слотов

Слоты — данные, которые клиент передает с запросом или в процессе дозапроса. У каждого слота есть обязательные атрибуты: Имя, Тип.

Слоты [+ Добавить слот](#)

<input type="checkbox"/>	Название	Сущность	Обязательно	Массив	Вопросы
<input type="checkbox"/>	departure	@departure	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	Из какого города отправляетесь?
<input type="checkbox"/>	destination	@destination	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	В какой город вы летите?
<input type="checkbox"/>	date	@duckling.time	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	Назовите, пожалуйста, дату отправления.



Заполнение слотов

Интен > Добавить слот

- *Название* — название слота.
- *Сущность* — выберите из списка сущность, определяющую тип данных, которые попадут в слот.
- *Обязательно* — переведите переключатель в активное положение, если слот является обязательным для заполнения.
- *Массив* — если надо отловить несколько повторяющихся сущностей.
- *Вопросы* — укажите вопросы, которые будут использованы при процессе

Слоты [+ Добавить слот](#)

<input type="checkbox"/>	Название	Сущность	Обязательно	Массив	Вопросы
<input type="checkbox"/>	departure	@departure	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	Из какого города отправляетесь?
<input type="checkbox"/>	destination	@destination	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	В какой город вы летите?
<input type="checkbox"/>	date	@duckling.time	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	Назовите, пожалуйста, дату отправления.



Заполнение слотов

- Для подключения модуля слот-филлинга укажите в сценарии в файле main.sc:

```
require: slotfilling/slotFilling.sc  
module = sys.zb-common
```

- Использование заполненных слотов в сценарии:

```
state:  
  intent!: /Погода  
  a: Погода в {{ $parseTree._City }} на {{ $parseTree._Date.value }}
```



Заполнение слотов

Слоты [+ Добавить слот](#)

<input type="checkbox"/>	Название	Сущность	Обязательно	Массив	Вопросы
<input type="checkbox"/>	City	@City	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	В каком городе? Где?
<input type="checkbox"/>	Date	@duckling.time	<input checked="" type="checkbox"/> Да	<input type="checkbox"/> Нет	Когда? На какую дату? Скажите мне дату.

1. **Погода на завтра в Москве**: слоты заполнены, бот сразу предоставит ответ.
2. **Погода на завтра**: слот **City** не заполнен, будут заданы уточняющие вопросы. Как только будет дан корректный ответ, управление перейдет в сценарий, бот даст ответ.
3. **Погода в Москве**: слот **Date** не заполнен, будут заданы уточняющие вопросы. Как только будет дан корректный ответ, управление перейдет в основной, бот даст ответ.

ПРАКТИКА

СОЗДАНИЕ ДИАЛОГОВОЙ СИСТЕМЫ С ОБРАЩЕНИЕМ К API



Что такое API

Программный интерфейс приложения – это набор готовых классов, функций, процедур, структур и констант, который используется при взаимодействии с внешними программами.



Что такое API

Программный интерфейс приложения – это набор готовых классов, функций, процедур, структур и констант, который используется при взаимодействии с внешними программами.

- Обеспечивает максимально простое соединение двух независимых систем.



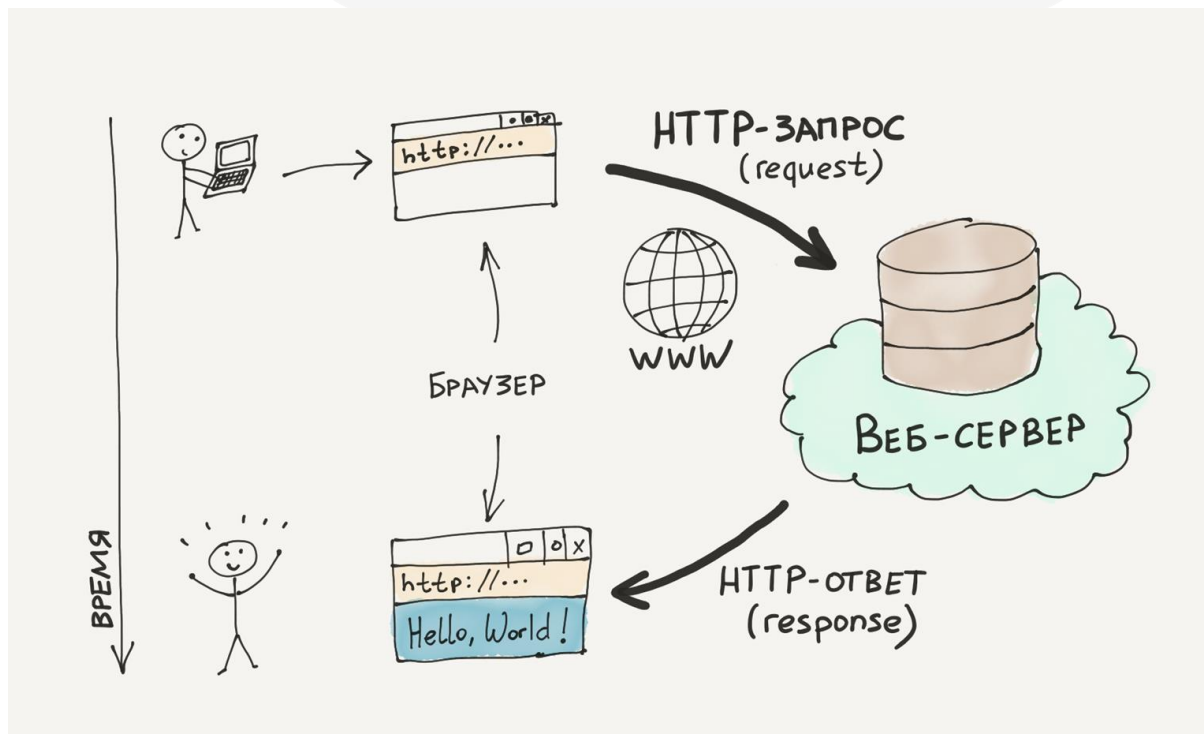
Что такое API

Программный интерфейс приложения – это набор готовых классов, функций, процедур, структур и констант, который используется при взаимодействии с внешними программами.

- Обеспечивает максимально простое соединение двух независимых систем.
- Позволяет обращаться к приложению как к «черному ящику» и использовать его функционал. При этом внутренний механизм работы приложения не имеет значения.



http-запросы





Структура запроса

```
$http.query(url, [settings]);
```

На вход:

- url – адрес в виде строки;
- объект settings – параметры запроса.



Структура запроса

`$http.query(url, [settings]);`

На вход:

- url – адрес в виде строки;
- объект settings – параметры запроса.

Простейший запрос по указанному url с параметрами по умолчанию:

```
state:
  q!: test 1
  script:
    $reactions.answer($http.query('http://localhost:9001/method1').data.text);
```



Параметры запроса

```
var result = $http.query('http://localhost:9000/get-with-  
params?resultPart1=${resultPart1}&resultPart2=${resultPart2}', {  
  method: "GET",  
  query: {  
    resultPart1: 'query',  
    resultPart2: 'with parameters'  
  },  
  body: { payload: "text"},  
  form: { formField1: '1', formField2: '2'},  
  headers: [ ["Authorization", "Basic xxxxx"] ]  
  dataType: "json",  
  timeout: 10000,  
  cachingRequired: true  
});
```



Параметры запроса

```
var result = $http.query('http://localhost:9000/get-with-  
params?resultPart1=${resultPart1}&resultPart2=${resultPart2}', {  
  method: "GET",  
  query: {  
    resultPart1: 'query',  
    resultPart2: 'with parameters'  
  },  
  body: { payload: "text"},  
  form: { formField1: '1', formField2: '2'},  
  headers: [ ["Authorization", "Basic xxxxx"] ]  
  dataType: "json",  
  timeout: 10000,  
  cachingRequired: true  
});
```



http-методы

Метод HTTP-запроса указывает серверу на то, какое действие мы хотим произвести с ресурсом (URL).

- GET — получение ресурса. Сокращенно*: `$http.get(url, [settings])`
- POST — создание ресурса. Сокращенно: `$http.post(url, [settings])`
- PUT — обновление ресурса. Сокращенно: `$http.put(url, [settings])`
- DELETE — удаление ресурса. Сокращенно: `$http.delete(url, [settings])`

*Поле `method` заполнять не нужно



Настройки http-клиента

```
$http.config(settings);
```

```
init:
```

```
  $http.config({  
    url: {  
      protocol: 'http',  
      host: 'example.com',  
      port: '80'  
    },  
    timeout: 1000,  
    authHeader: 'Basic XXXXXXXXXXXXX',  
    cacheTimeToLiveInSeconds: 1 * 60 * 30,  
  });
```



Структура ответа

```
{  
  "data" : {...},  
  "status" : 200,  
  "response" : {  
    ... },  
  "isOk" : true  
}
```

```
{  
  "error" : {...},  
  "data" : undefined,  
  "status" : 400,  
  "response" : {  
    ... },  
  "isOk" : false,  
}
```




Как правильно парсить ответ

- Записать результат http-вызова в переменную
- Проверить успешность выполнения запроса, обработать ошибки
- Продолжать работу с полями тела ответа (response.data)

```
function getApiResponse(param) {  
  ... var response = $http.get("http://localhost:9000/get-with-params?result=${param}", {  
    ... query: {  
      ... param: param  
    ... }  
  ... });  
  
  ... if(response.isOk) {  
    ... return response.data;  
  ... }  
  
  ... $reactions.transition("/OnError");  
}
```



Как правильно парсить ответ

```
{  
  "data" : {  
    "field1" : "value1"  
  },  
  "status" : 200,  
  "response" : {  
    ...  
  },  
  "isOk" : true  
}
```

```
state: ApiResponse  
script:  
  var result = getApiResponse("parametr");  
  $session.value1 = result.field1;  
a: Ответ гласит: value1 = {{$session.value1}}.
```

ПРАКТИКА

ДОМАШНЕЕ ЗАДАНИЕ



Домашнее задание

1. Покрыть тестами новый функционал учебного чат-бота: написать тест-кейс, используя тег <mockData>:

https://help.just-ai.com/docs/ru/Content_testing/tests_xml/tags/mockData

Для проверки прислать данные своего аккаунта на
<https://app.jaicp.com/>

2. Создать канал Маруся и зарегистрировать свой скилл в ассистенте.

Важно! Публиковать навык в прод не нужно!

<https://help.just-ai.com/docs/ru/channels/marusya/marusya>

Для проверки выдать права администрирования навыка пользователю
<https://vk.com/riya404> и прислать активационную фразу навыка.



Критерии проверки

Всего за задание вы можете получить 10 баллов

- написан тест с использованием `<mockData>`, работает корректно – 5 баллов;
- скилл развернут в канале Маруся, в него можно зайти из приложения или капсулы по активационной фразе (пользователям с доступом к скиллу) – 5 баллов.