СОЗДАНИЕ ЧАТ-БОТОВ С DIALOGFLOW, WATSON, CHATTERBOT И RASA

ЧАТБОТЫ С ИСКУССТВЕННЫМ ИНТЕЛЛЕКТОМ БЕЗ ПРОГРАММИРОВАНИЯ И С **PYTHON**

ТИМУР МАШНИН

Тимур Машнин

Создание чат-ботов с Dialogflow, Watson, ChatterBot и Rasa

«Автор»
2022

Машнин Т.

Создание чат-ботов с Dialogflow, Watson, ChatterBot и Rasa / Т. Машнин — «Автор», 2022

С этой книгой Вы познакомитесь с чат-ботами и поймете как создавать чат-ботов без программирования с использованием таких облачных служб как Google Dialogflow и IBM Watson.Также Вы узнаете как реализовать для чат-бота Webhook - механизм получения уведомлений об определённых событиях, чтобы выполнять внешнюю бизнес-логику.Вы узнаете как можно интегрировать вашего чат-бота с другими платформами.Познакомитесь с библиотеками ChatterBot и Rasa и узнаете как создавать чат-ботов на языке Python.

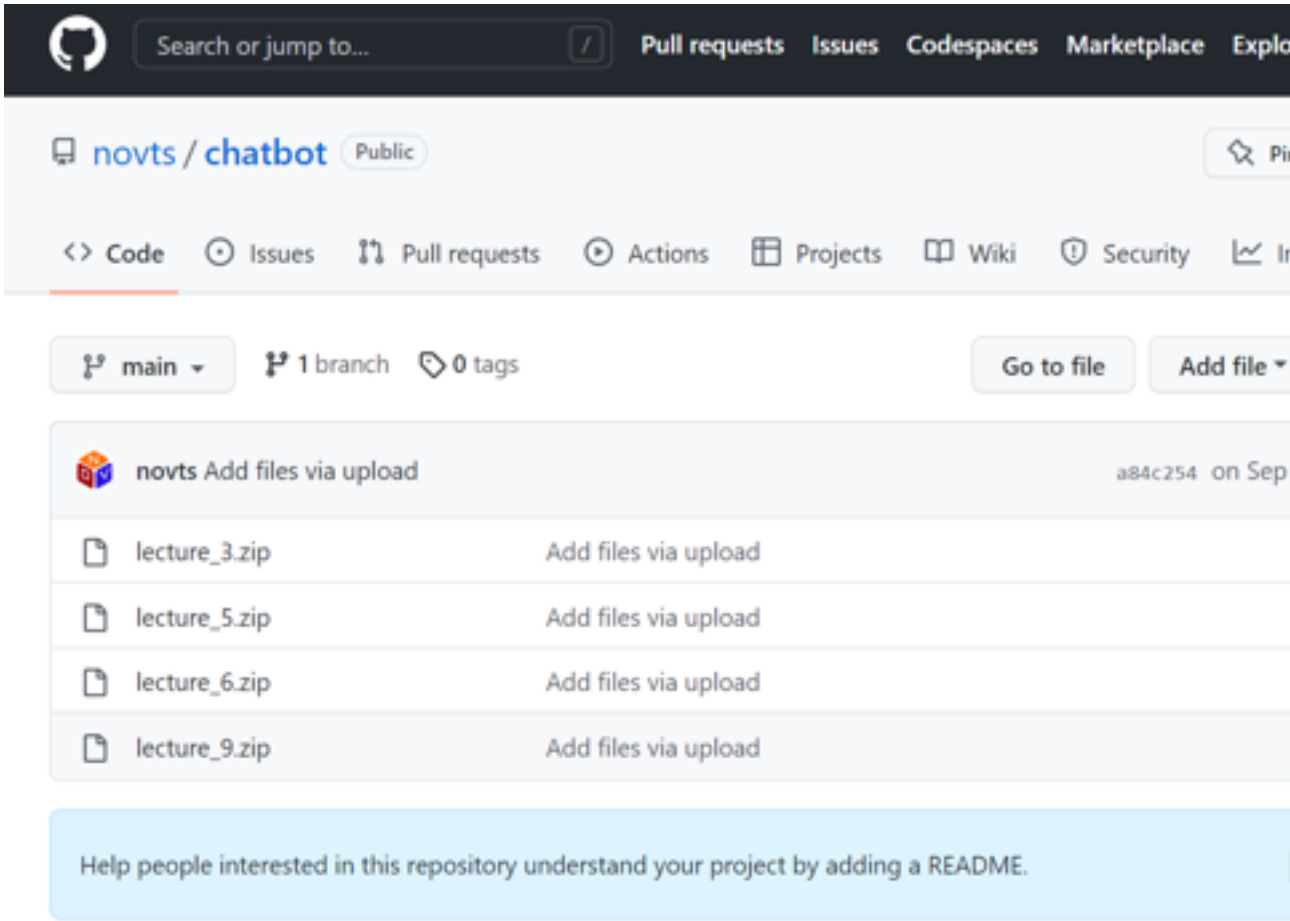
Содержание

Исходный код 5
Введение 6
Введение в 8
Google 32
Google Dialogflow. Интеграция с Telegram 54
ChatterBot 59
Конец ознакомительного фрагмента. 63

Тимур Машнин
Создание чат-ботов с Dialogflow, Watson, ChatterBot и Rasa

Исходный код

Исходный код к примерам можно скачать с сайта GitHub (https://github.com/novts/chatbot).

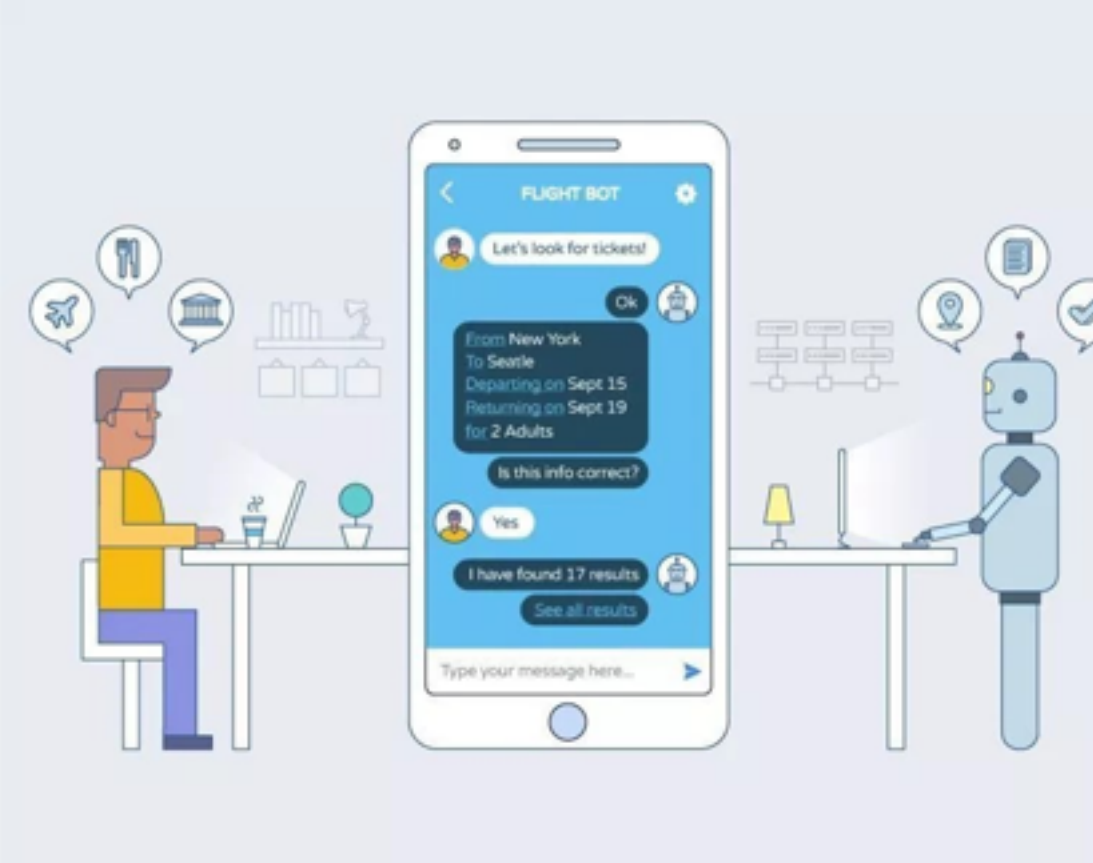


Введение

Что такое бот?

Бот – это программное обеспечение для искусственного интеллекта, предназначенное для выполнения ряда задач самостоятельно и без помощи человека.

Задачи, которые может выполнять бот, могут варьироваться от таких вещей, как бронирование в ресторане, отметка даты в календаре или сбор и отображение информации для своих пользователей, а также информирование пользователя о погоде и т. д.



Наиболее часто встречающийся вид ботов – это чат-боты.

Чат-боты могут симулировать разговор с человеком и, часто встречаются в приложениях обмена сообщениями.

Чат-боты универсальны, они способны адаптироваться и помогают решать различные бизнес-задачи.

чат-бот – это программная система, которая может взаимодействовать или общаться в чате с пользователем на естественном языке (таком как английский или любом другом языке). чат-боты могут дать различного рода информацию пользователю или помочь ему в выполнении задачи.

Как на самом деле работают чат-боты?

Есть два типа чат-ботов, – это чат-боты основанные на правилах, и чат-боты с ИИ.

Чат боты, основанные на правилах, отвечают на вопросы, основываясь на некоторых правилах, на которых они обучаются.

Такие чат-боты предоставляют ответы только на основе комбинации предопределенных сценариев.

Определенные правила, на которых обучен такой чат-бот, могут быть очень простыми или очень сложными.

И создание этих ботов относительно просто, но эти боты неэффективны в ответах на вопросы, чей шаблон не соответствует правилам, по которым был обучен бот.

Поэтому, чтобы чат-бот мог делать больше, чем отвечать на предопределенные вопросы, он должен быть подключен к искусственному интеллекту.

ИИ – это технология, которая позволяет боту учиться на взаимодействиях с конечными пользователями.

И вам не нужно быть экспертом ИИ или техническим экспертом, чтобы создать чат-бота.

Разработка чат-бота не более сложная, чем разработка простого веб-приложения.

Что в действительности могут делать боты и ИИ?

Боты могут быть виртуальными помощниками.

Предприятия используют чат-ботов для различных случаев, таких как обслуживание клиентов.

Проще говоря, сервис искусственного интеллекта может использоваться для ответа на простые вопросы, помогать пользователям бронировать услуги, получать информацию по определенной теме, покупать товары и т. д.

Наличие чат-бота помогает ускорить выполнение задач этого типа, что позволяет сосредоточиться персоналу. на более актуальных проблемах.

В то же время чат-бот позволяет компании иметь круглосуточный сервис для удовлетворения потребностей своих клиентов.

Боты помогают генерировать идеи.

Данные являются товаром, который питает цифровую экономику.

Однако нужно иметь необходимые ресурсы, чтобы превратить их в нечто ценное.

В идеале компании должны иметь ИИ, который автоматически учится на всех данных, которые компании собирают.

Это позволит компаниям адаптироваться при изменении поведения рынка, а также постоянно повышать производительность по мере поступления новых данных.

Боты автоматизируют ручные процессы.

Искусственный интеллект быстро автоматизирует рутинные и механические когнитивные процессы.

Оставляя больше времени для инноваций.

Например, использование ИИ может автоматизировать процесс сбора данных из различных отчетов и выполнять анализ для определения прибыльности конкретного бизнес-процесса.

ИИ может анализировать неструктурированные данные.

Предполагается, что 80% цифровых данных не структурированы.

Организация и отслеживание этих данных может привести к лучшему пониманию пользователей и прогнозированию рынка на основе тенденций.

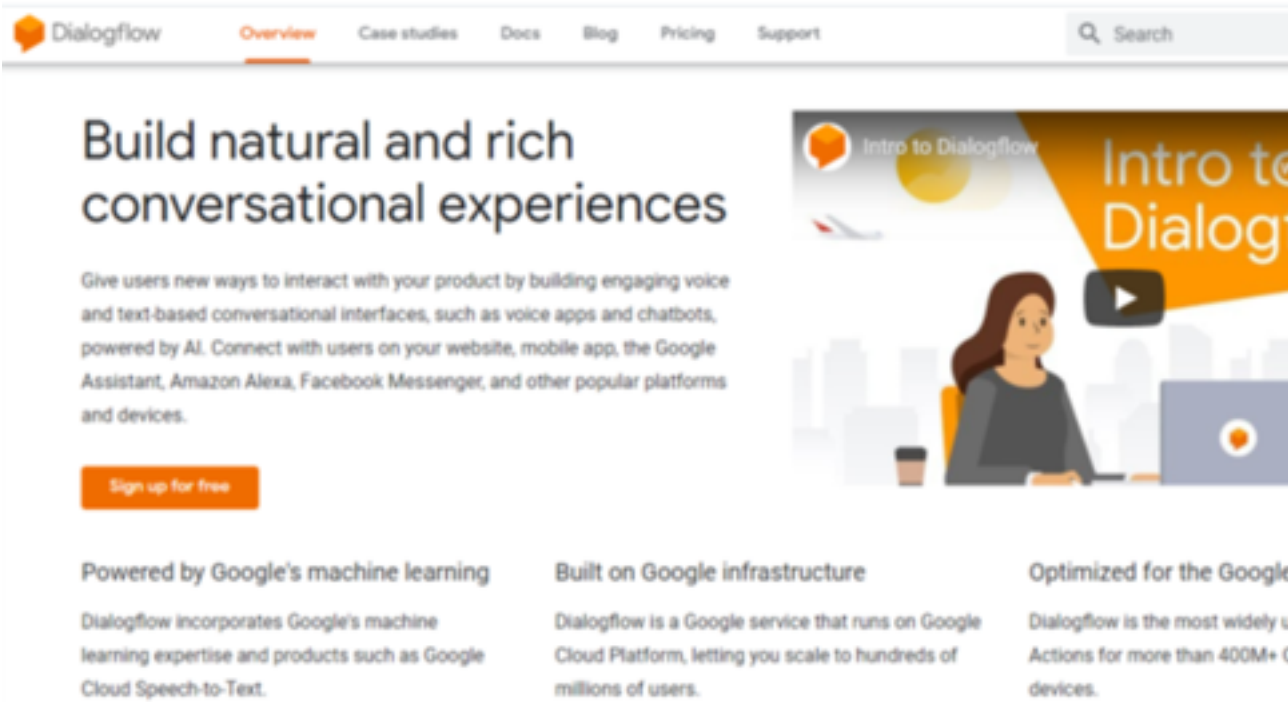
Обучение чат-бота с ИИ происходит значительно быстрее, чем обучение персонала.

В то время как обычным представителям службы поддержки клиентов даются инструкции, с которыми они должны тщательно разбираться, чат-бот поддержки клиентов снабжается большим количеством журналов разговоров, и из этих журналов чат-робот может понять, на какой тип вопроса нужен какой ответ.

Введение в Google Dialogflow

Dialogflow – это инструмент, который может помочь создать умного чат-бота.

https://dialogflow.com



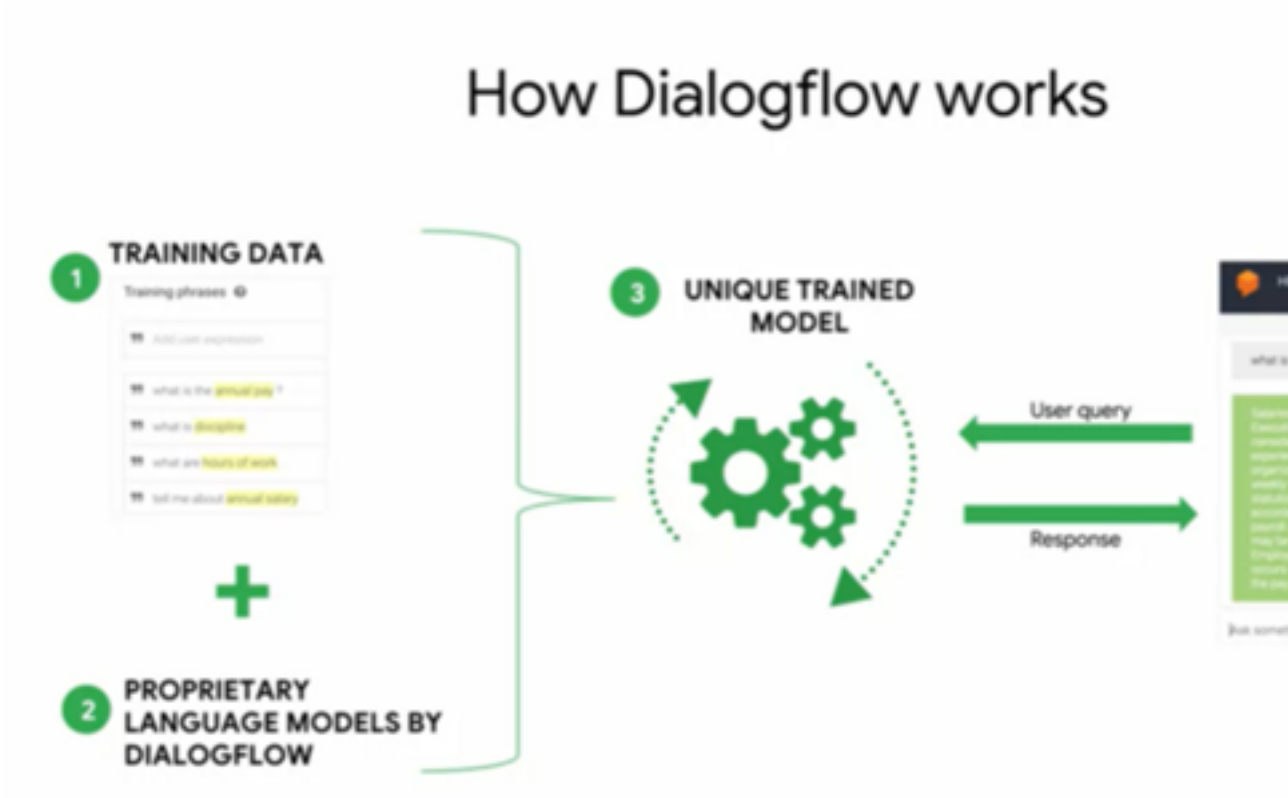
Dialogflow – это платформа для создания естественных и богатых диалогов.

По своей сути Dialogflow – это мощный механизм понимания естественного языка для обработки и понимания ввода на естественном языке.

Другими словами, он позволяет вам легко общаться с пользователем, понимая естественный язык.

Dialogflow построен на ресурсах и возможностях ИИ мирового класса, которые были изначально разработаны для таких продуктов, как Gmail и Google Search.

И Dialogflow включает в себя постоянно растущий опыт Google в области искусственного интеллекта, включая опыт машинного обучения, возможности поиска, распознавание речи и, конечно, понимание естественного языка.



И возможности обработки естественного языка Google включают в себя синтаксический анализ, который позволяет извлекать токены и предложения.

Определение частей речи и создание деревьев анализа зависимостей для каждого предложения.

Распознавание сущностей в пользовательском вводе позволяет идентифицировать такие типы, как человек, организация, местоположение, события, продукты и так далее.

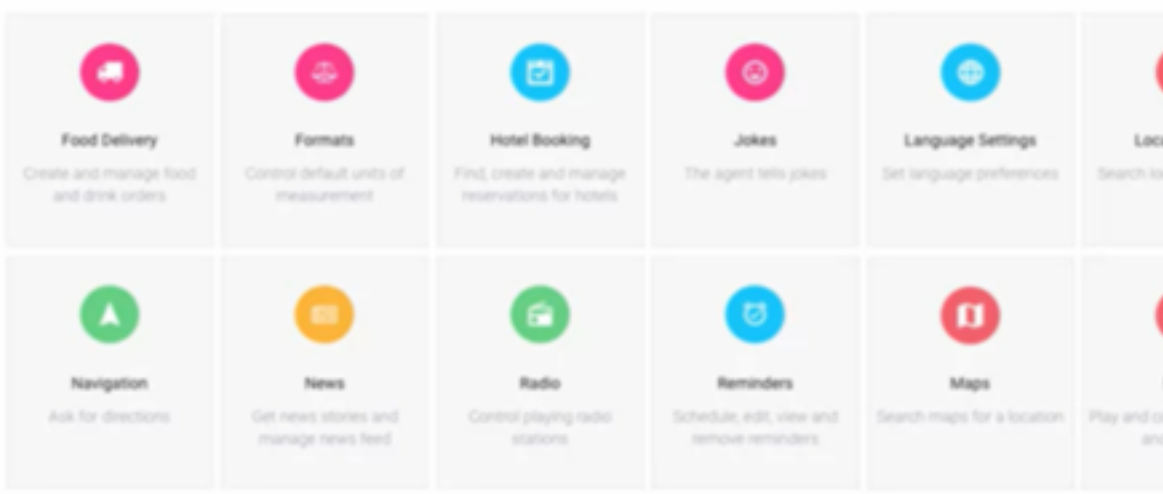
Анализ настроений дает понимание общего настроения, выраженного в блоке текста.

Классификация контента позволяет классифицировать документы по более чем 700 предварительно определенным категориям.

Многоязычная поддержка включает в себя возможность легко анализировать текст на нескольких языках.

Используя эти возможности и то, что разработчик предоставляет в качестве входных данных для обучения, Dialogflow создает уникальные алгоритмы для каждого конкретного собеседника, при этом постоянно обучаясь и настраиваясь, по мере того как все больше и больше пользователей взаимодействуют с чат-ботом.

С Dialogflow вы можете быстро создать своего агента, начав с нескольких обучающих фраз или используя один из более чем 40 предварительно созданных агентов.

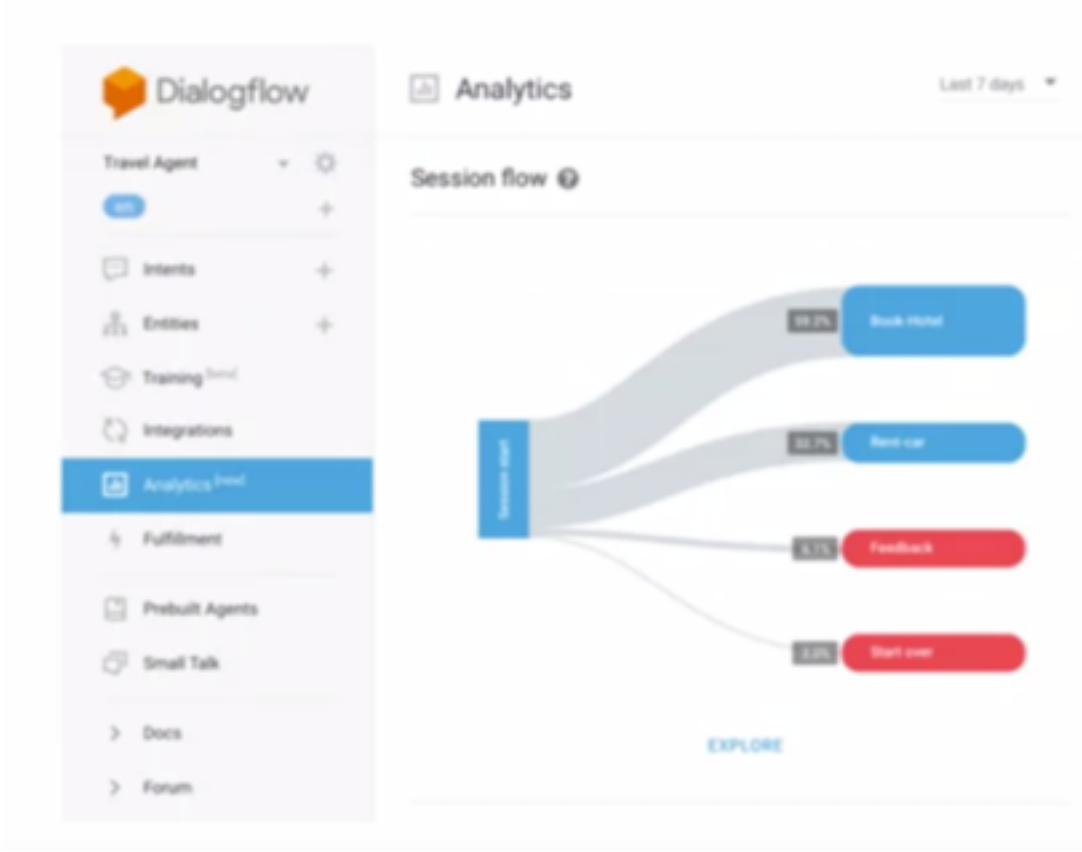


Эти предварительно созданные агенты могут использоваться непосредственно из коробки или импортироваться в ваш агент для создания и настройки вашего собственного варианта использования.

Они включают в себя все, от доставки еды до бронирования отелей, новостей и напоминаний.

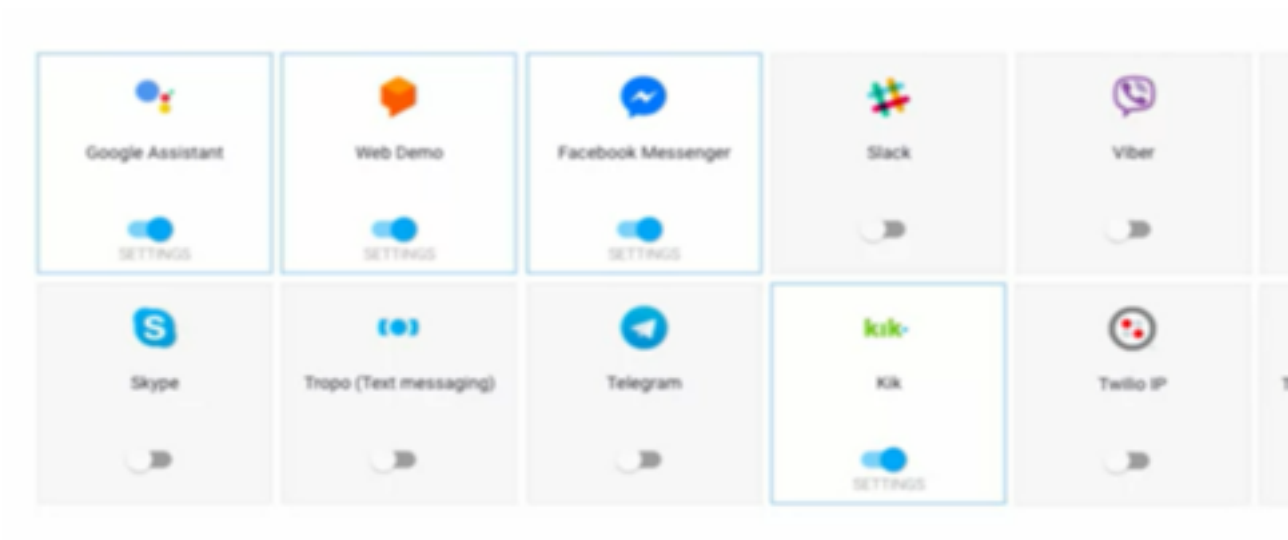
И вы можете легко импортировать эти предварительно созданные агенты из консоли Dialogflow.

Встроенная аналитика Dialogueflow может многое рассказать вам о взаимодействии пользователей с вашим чат-ботом.

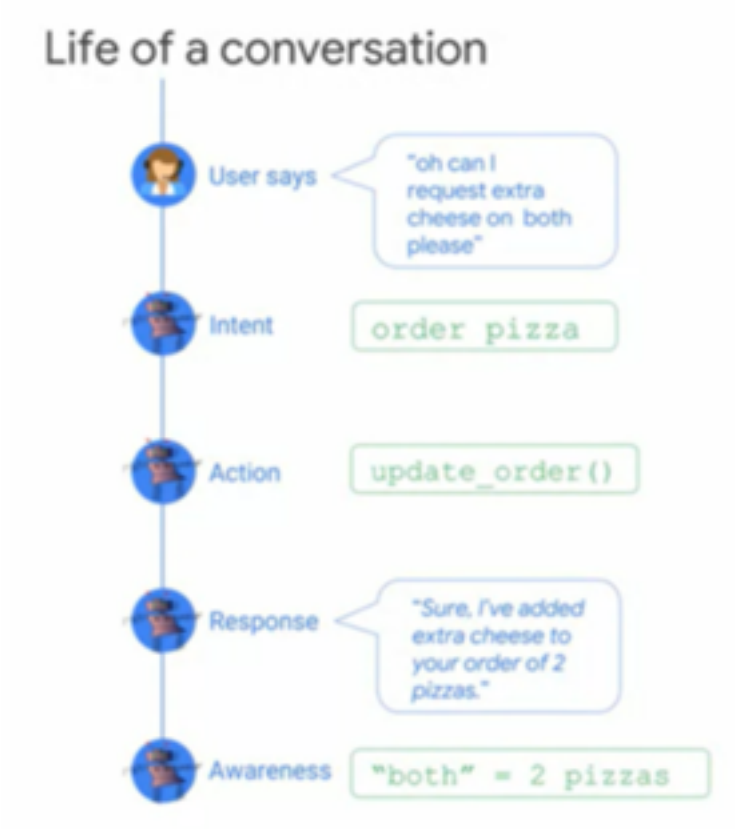


Например, она может показать вам, как часто срабатывают различные намерения.

Вы можете легко развернуть свой чат-бот на нескольких платформах, таких как Facebook Messenger, Twitter, и другие.



Давайте внимательнее посмотрим, как происходит диалог, чтобы понять, какие элементы понадобятся вашему чат-боту.



Естественно, диалог начинается с пользователя, которому что-то нужно от чат-бота, и он начинает разговор, чтобы сказать, что ему нужно.

Чат-бот должен сопоставить это с намерением, запрограммированным для обработки запроса.

Например, когда пользователь заказывает пиццу, распознается подходящее намерение для заказа пиццы.

И это намерение подразумевает наличие нескольких компонентов.

Что на самом деле говорит пользователь, какое действие предпринять, ответ чат-бота и понимание контекста.

И это намерение запускает действие по размещению заказа.

Это может быть похоже на функциональность сервера, который обрабатывает заказ.

Затем чат-бот может дать соответствующий ответ, например, подтверждение того, что заказ пользователя был размещен.

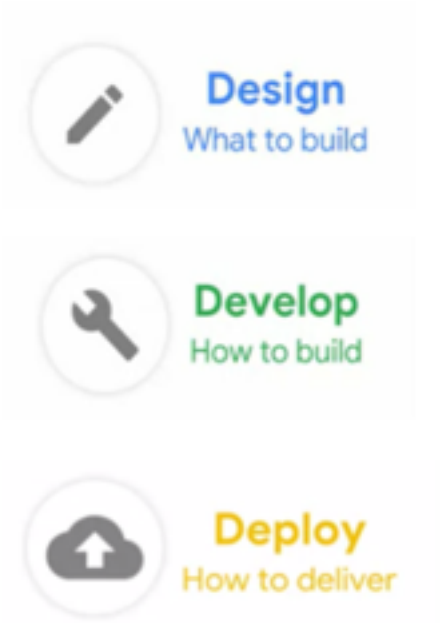
И чат-бот также должен иметь возможность обрабатывать ветвление диалога, которое не всегда следует именно этому потоку.

Например, что, если пользователь, заказавший пиццу, сделает дополнительный запрос на заказ?

Чат-бот должен поддерживать естественный разговор, который учится на прошлых диалогах.

Он может вернуться к тому же самому намерению и добавить дополнительный уровень контекста или осведомленности, чтобы понять, что слово «оба» в запросе пользователя относится к двум пиццам, которые он заказывает.

Ваш чат-бот может скорректировать заказ и удовлетворить дополнительный запрос пользователя.



Как правило, рабочий процесс создания чат-бота состоит из трех этапов.

На этапе дизайна вы определяете индивидуальность вашего чат-бота.

Будет ли он упреждающим, например, делать предложения пользователям, или реагировать, просто отвечая на запросы пользователей.

Определите атрибуты, которые вы хотите добавить в диалог, стиль письма и индивидуальность диалога.

Подумайте о том, как ваш чат-бот будет приветствовать пользователя и как завершит разговор.

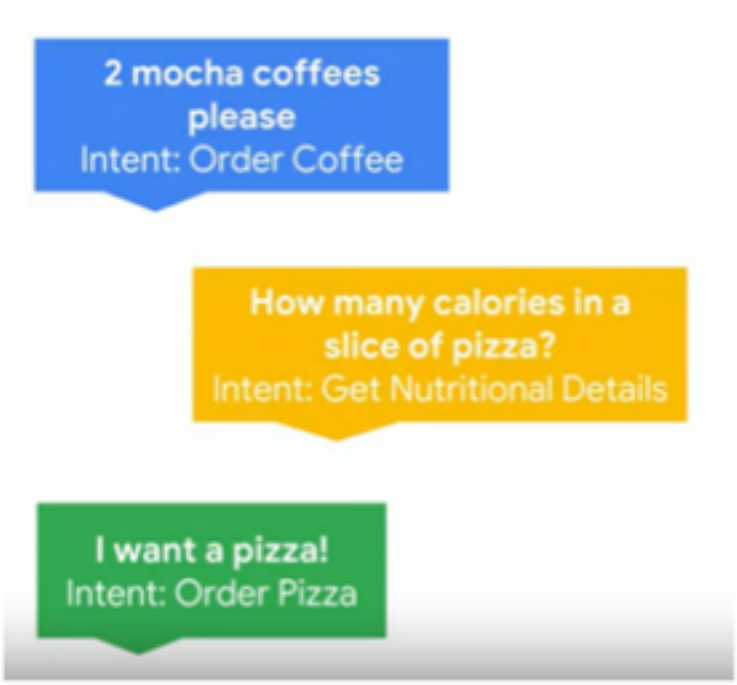
Как разговор должен проходить для нового пользователя по сравнению с вернувшимся пользователем.

На этапе разработки вы используете поток диалога для создания своего чат-бота с комбинацией прямого добавления намерений и ответов в консоли и написания кода для подключения к внутренним службам.

Этап развертывания в основном зависит от того, какие компоненты нужны вашему чат-боту, и каких приложений он будет касаться.

Здесь подумайте о безопасности, интеграции и масштабировании.

И здесь нужно определить, для каких платформ нужен ваш чат-бот.



Работа чат-бота всегда начинается с намерений.

Намерения – это соединительные линии дерева диалога.

Они соединяют все ветви.

Намерения определяют, в какую сторону пойдет разговор и что должен делать чат-бот.

В общении намерения можно рассматривать как корневые глаголы в диалоге, например, хочу кофе транслируется в приобретение напитка.

Иногда намерения не являются явными и выводятся из всей фразы.

И нужно сопоставить намерения с какими-то действиями.

Если у вас приложение службы поддержки, тогда намерения могут инициировать открытие заявки, обновление заявки, закрытие заявки на поддержку.

Также вашему приложению может потребоваться получить доступ и обновить информацию об учетной записи пользователя, обратиться к специалисту и провести опрос по обеспечению качества.

Даже утверждение, да или нет, может являться намерением.

И намерения развиваются по мере того, как развивается ваше понимание потребностей пользователей.

Чтобы упростить задачу определения намерений, можно применить некоторые практические правила.

Сначала определите глаголы в диалоге.

Это позволит вашему чат-боту сопоставить свои действия с потребностями пользователя.

Также нужно определить, где диалог должен ветвиться согласно логике.

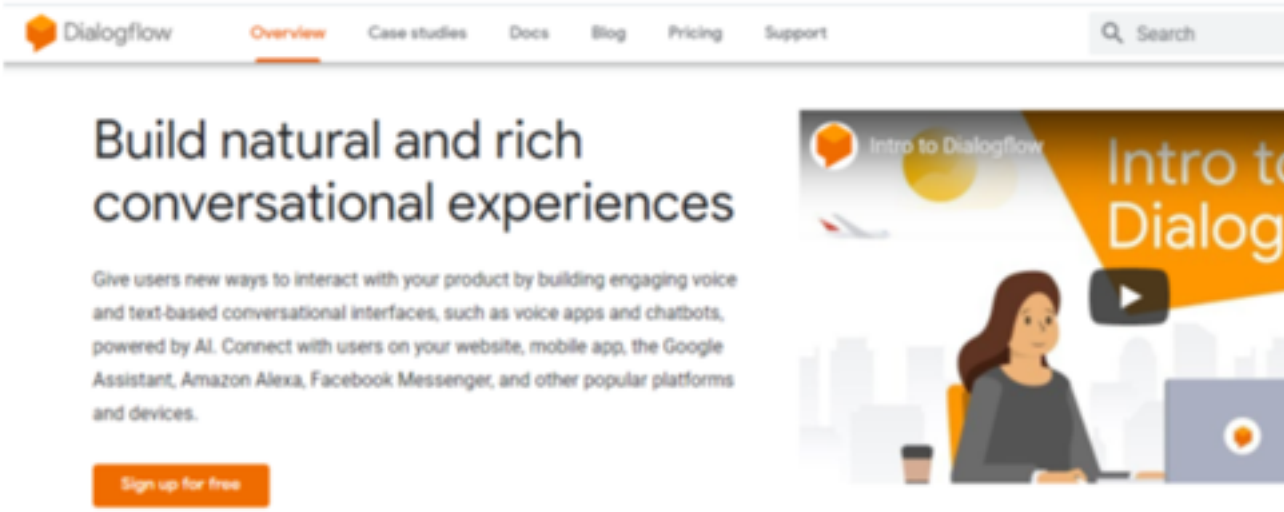
После того, как вы определили намерения, вам нужно обучить своего чат-бота распознавать их.

Это можно сделать с использованием обучающих фраз.

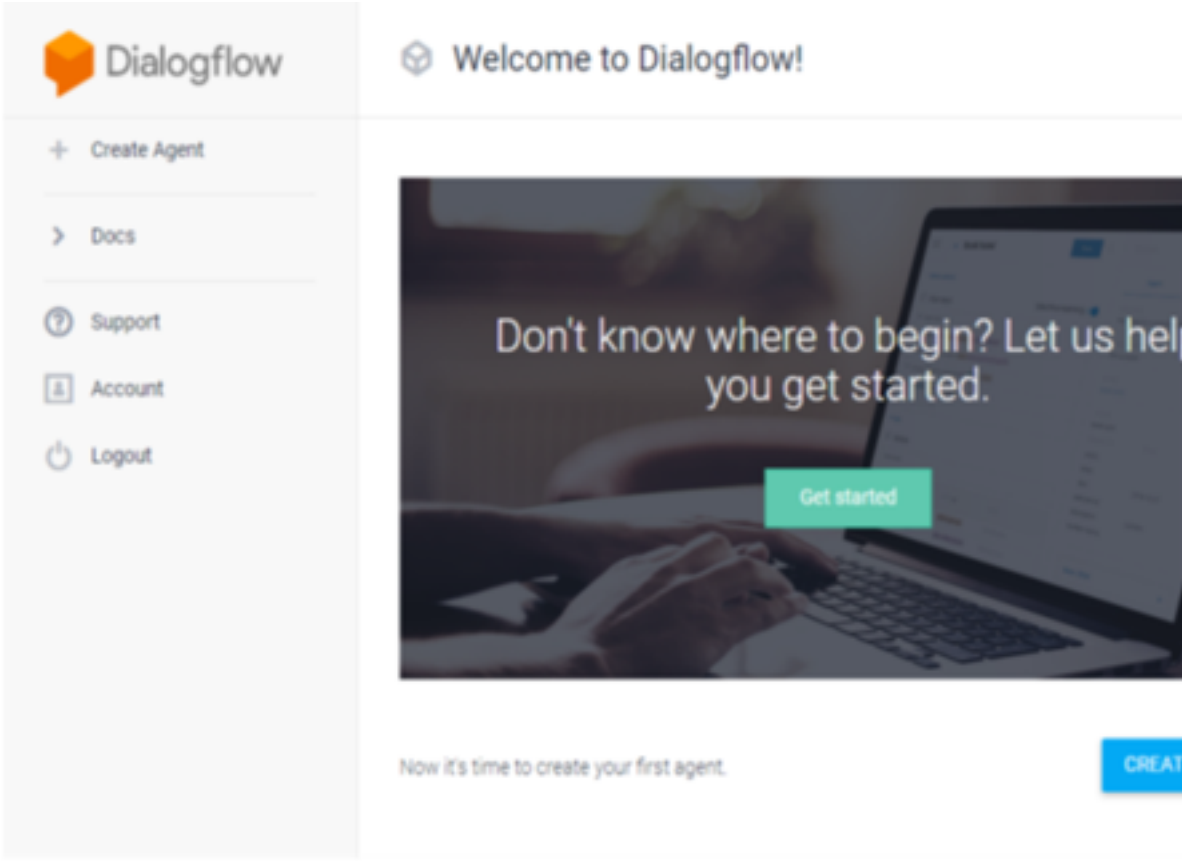
Обучающие фразы для каждого намерения должны отражать то, как пользователи проявляют такое намерение.

Всегда полезно добавлять варианты грамматической конструкции запроса, используя пассивные и активные глаголы, вопросы и т. д.

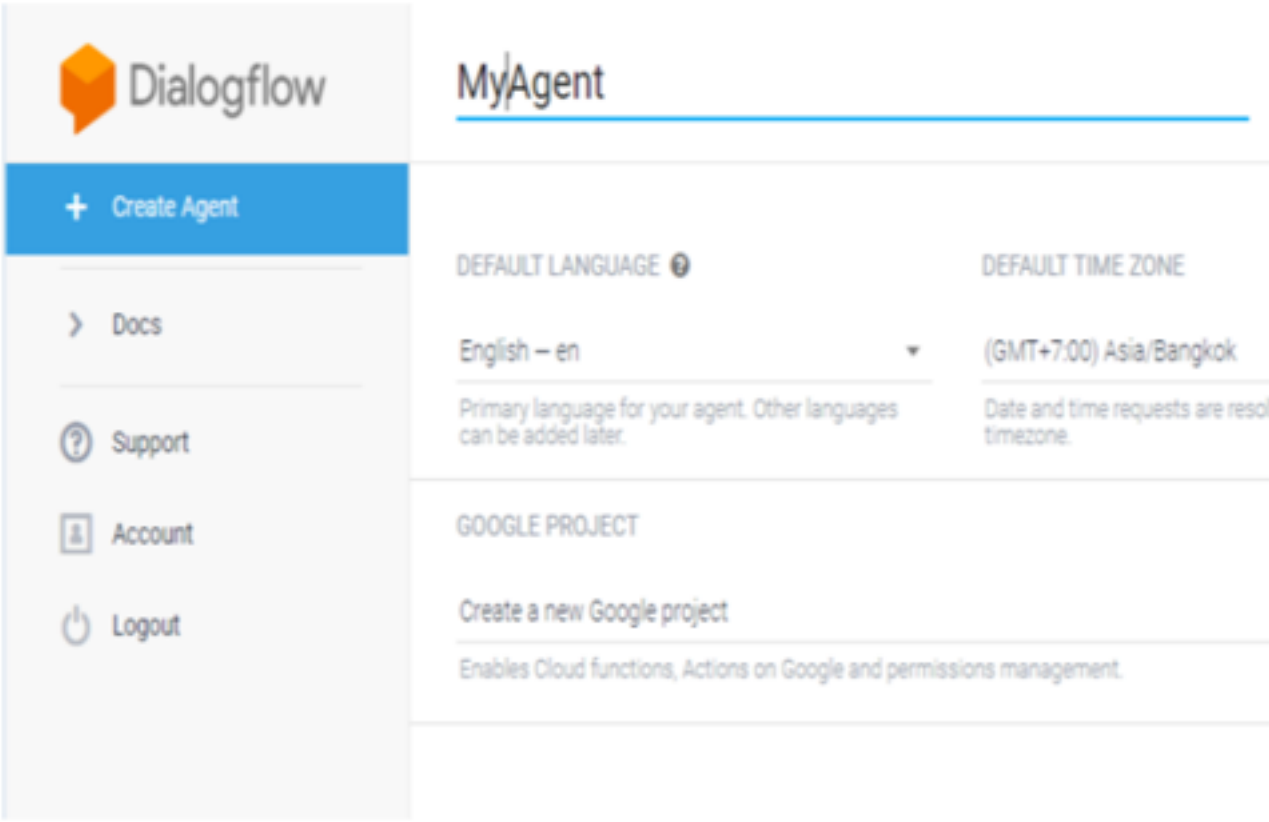
При создании намерения, чем больше учебных фраз вы можете придумать, тем лучше.



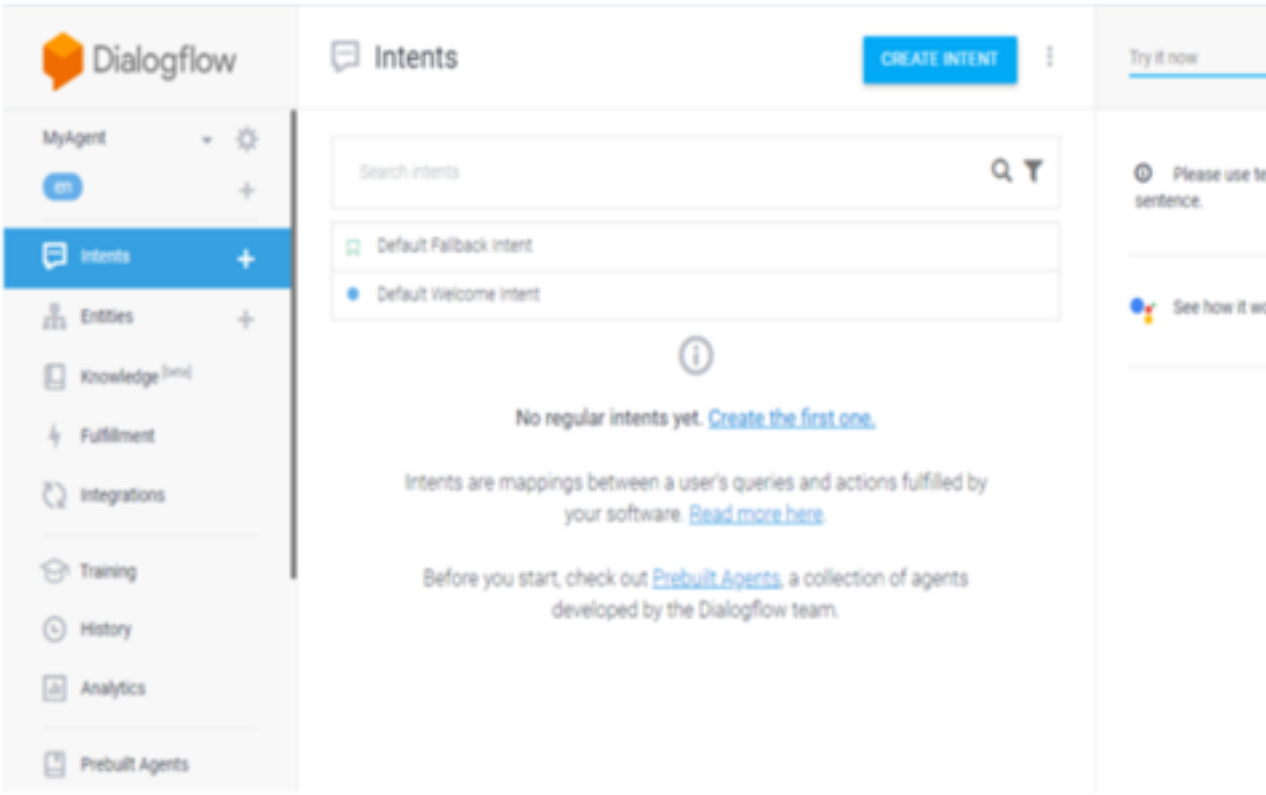
Откроем консоль Dialogflow.



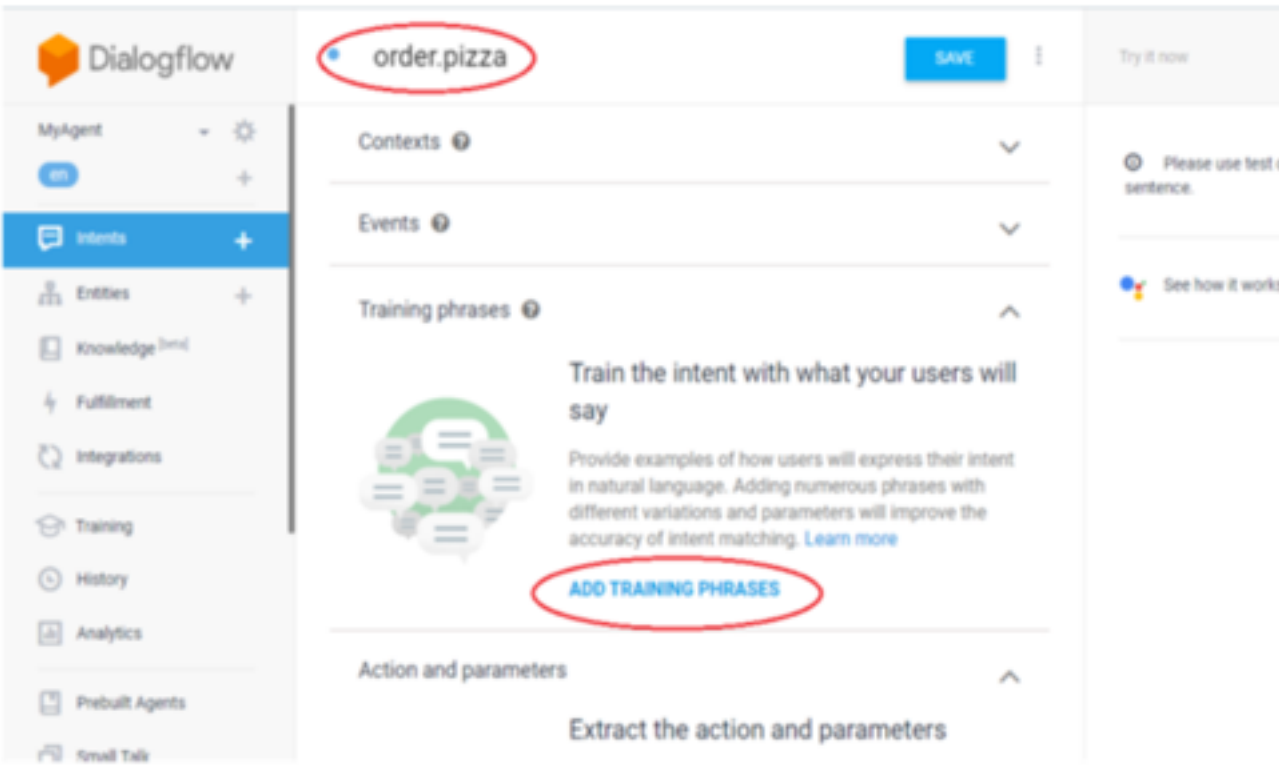
И создадим агента – чат-бот с помощью кнопки Create agent.



Введем имя агента и нажмем кнопку Create.



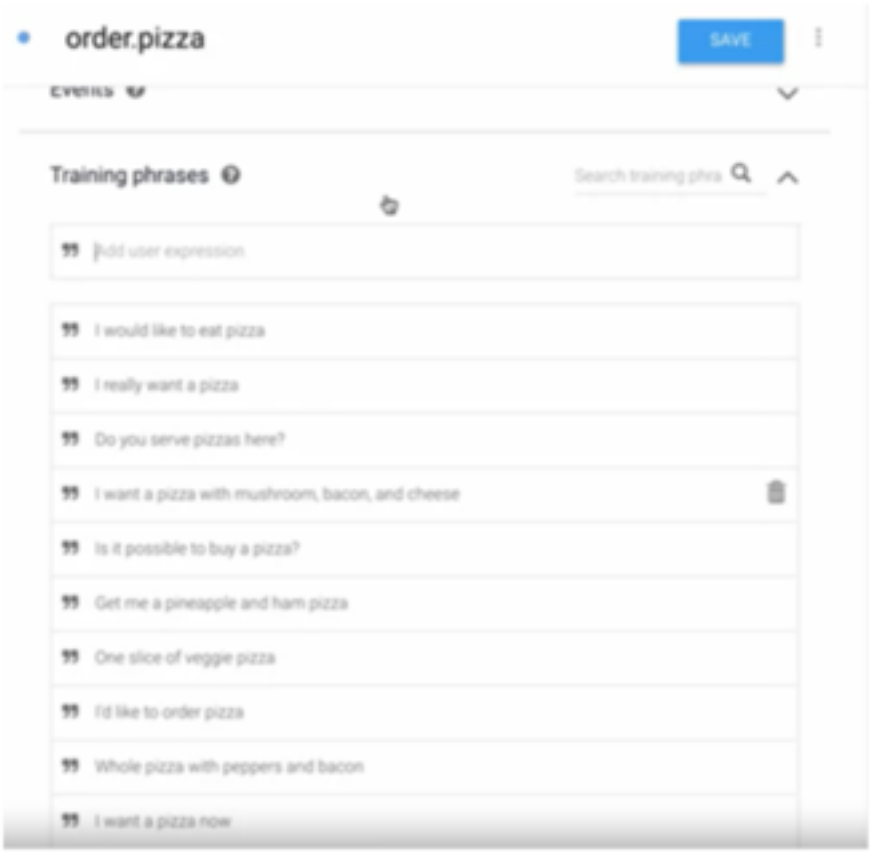
И теперь, здесь мы можем добавлять намерения.
Нажмем кнопку Create Intent.



Введем имя намерения order.pizza.

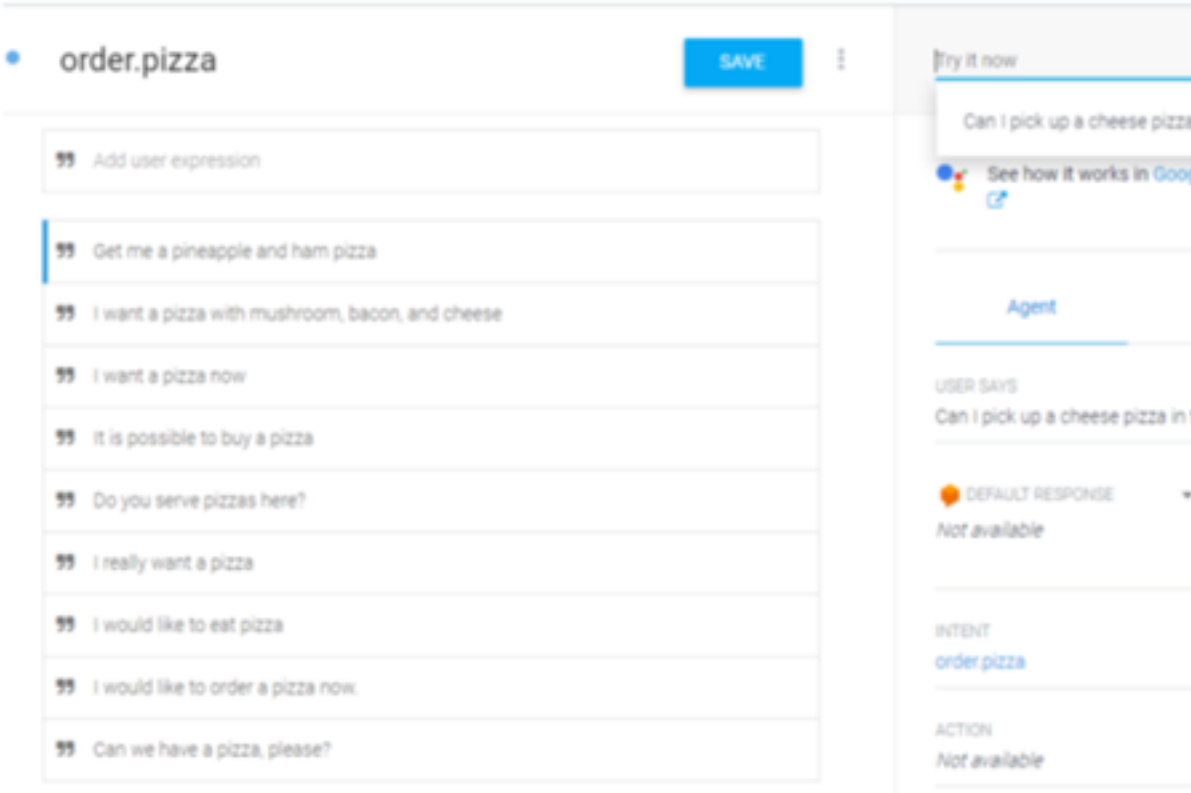
И теперь, ниже мы можем добавлять фразы для обучения этому намерению, используя кнопку ADD TRAINING PHRASES.

После ввода не забудьте нажимать кнопку Save сохранения.



Теперь, когда мы ввели фразы для обучения, мы можем протестировать агента.

И чтобы проверить, правильно ли было обучено намерение, мы можем использовать правую боковую панель со строкой «Попробуй сейчас» Try it now.



Здесь мы можем ввести фразу и посмотреть, сможет ли агент определить намерение.

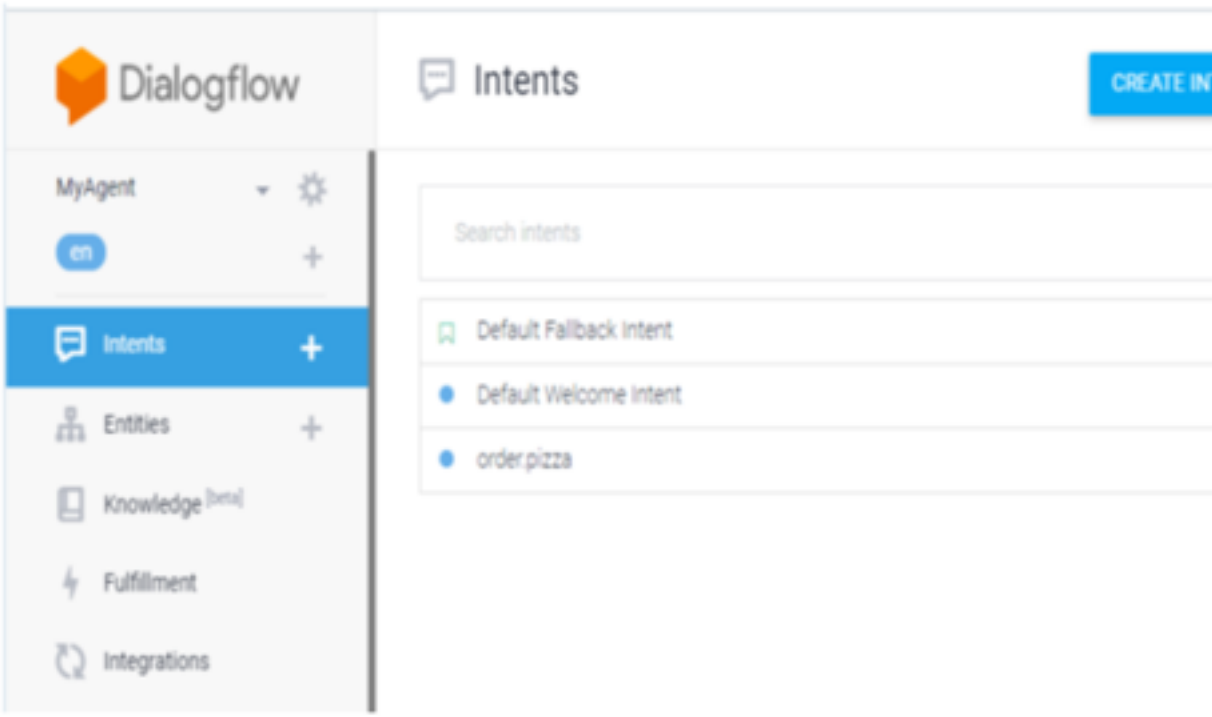
В строке Try it now введем «Могу ли я забрать сырную пиццу за два часа?».

И здесь мы видим, что намерение определено верно – order.pizza.

И обратите внимание, что ответ по умолчанию недоступен, потому что мы не определили никаких ответов, которые агент должен был предоставить после того, как он определил намерение.

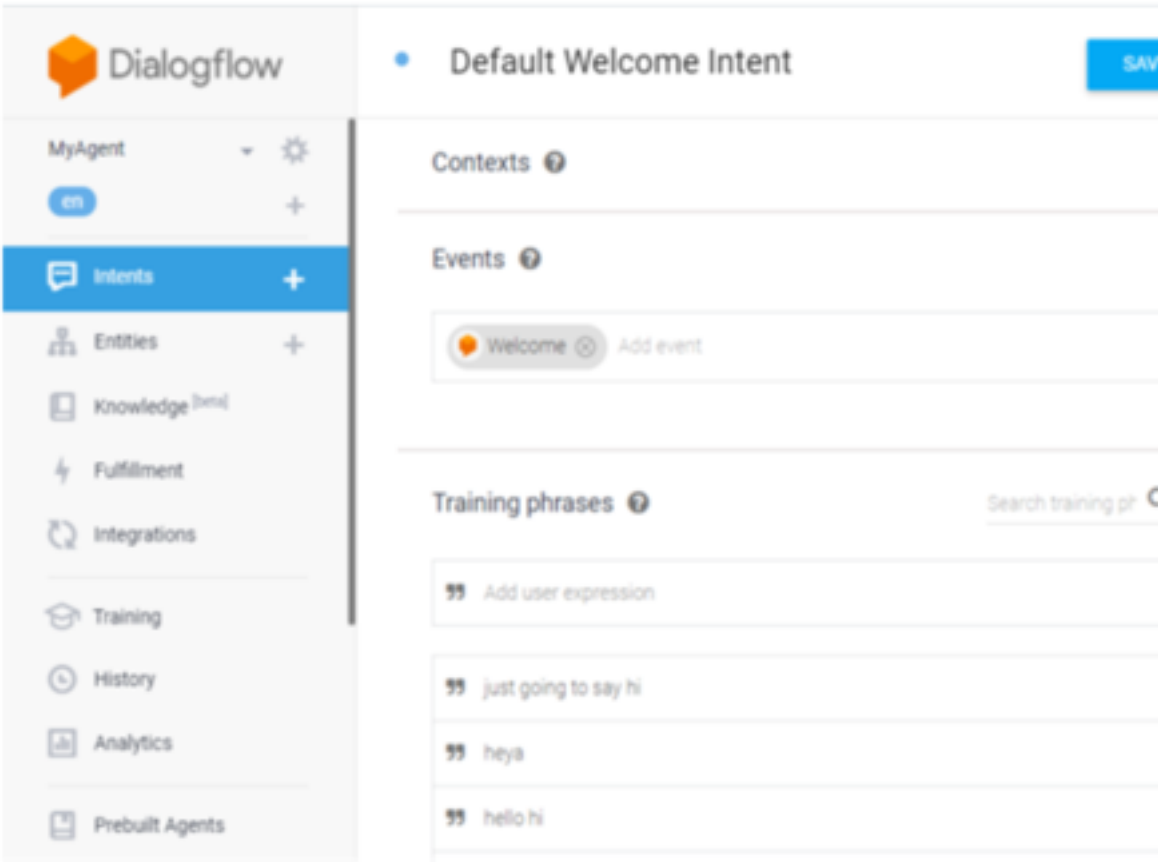
Также обратите внимание, что хотя введенная фраза не является частью обучающих фраз, агент верно определил намерение, потому что Dialogflow использует ИИ.

Агент в состоянии определить правильное намерение, потому что он определяет семантическое сходство между обучающими фразами и вводом пользователя.



Теперь, когда вы вернетесь на страницу намерений, вы увидите, что кроме намерения, которое мы только что создали, здесь уже есть два намерения, и они оба являются намерениями по умолчанию.

Откроем намерение Welcome.



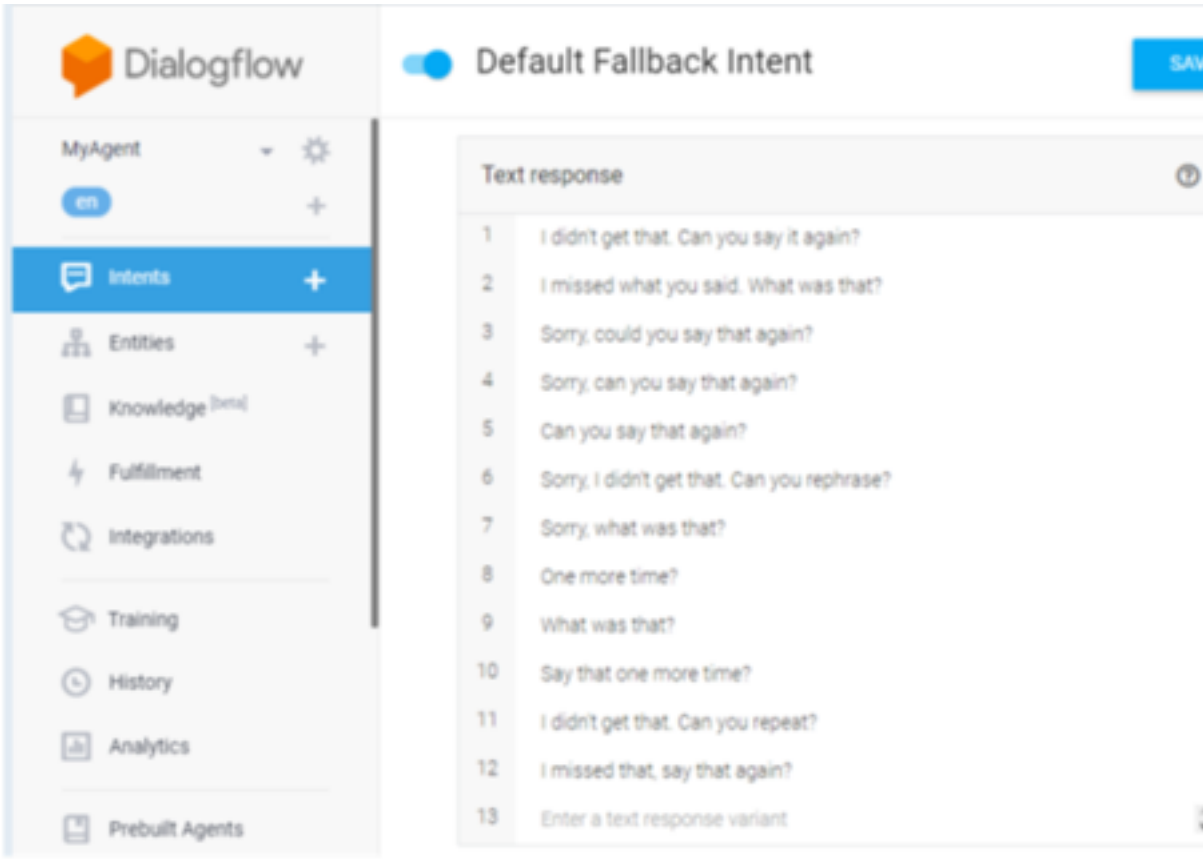
Это намерение приветствия по умолчанию.

И это намерение позволяет агенту распознавать приветствия от пользователя.

Поэтому, когда пользователь говорит «Привет», «Привет», «Как дела?», агент сможет ответить приветствием и спросить, как он может помочь пользователю.

Нам не нужно определять это намерение.

Эти намерения по умолчанию создаются автоматически вместе с агентом.



Fallback намерение, как следует из названия, является запасным вариантом для агента, который не понимает, о чем просит пользователь.

Вы можете попробовать задать вопрос о погоде агенту заказа пиццы и посмотреть, что произойдет.

Вот несколько рекомендаций, которые следует соблюдать при определении намерений чат-бота.

При выборе обучающих фраз для тренировки намерения обязательно учитывайте, каким образом пользователи могут выразить это намерение.

Это может варьироваться от синонимов до различных грамматических конструкций фраз.

Другим важным аспектом является определение намерений не двусмысленным.

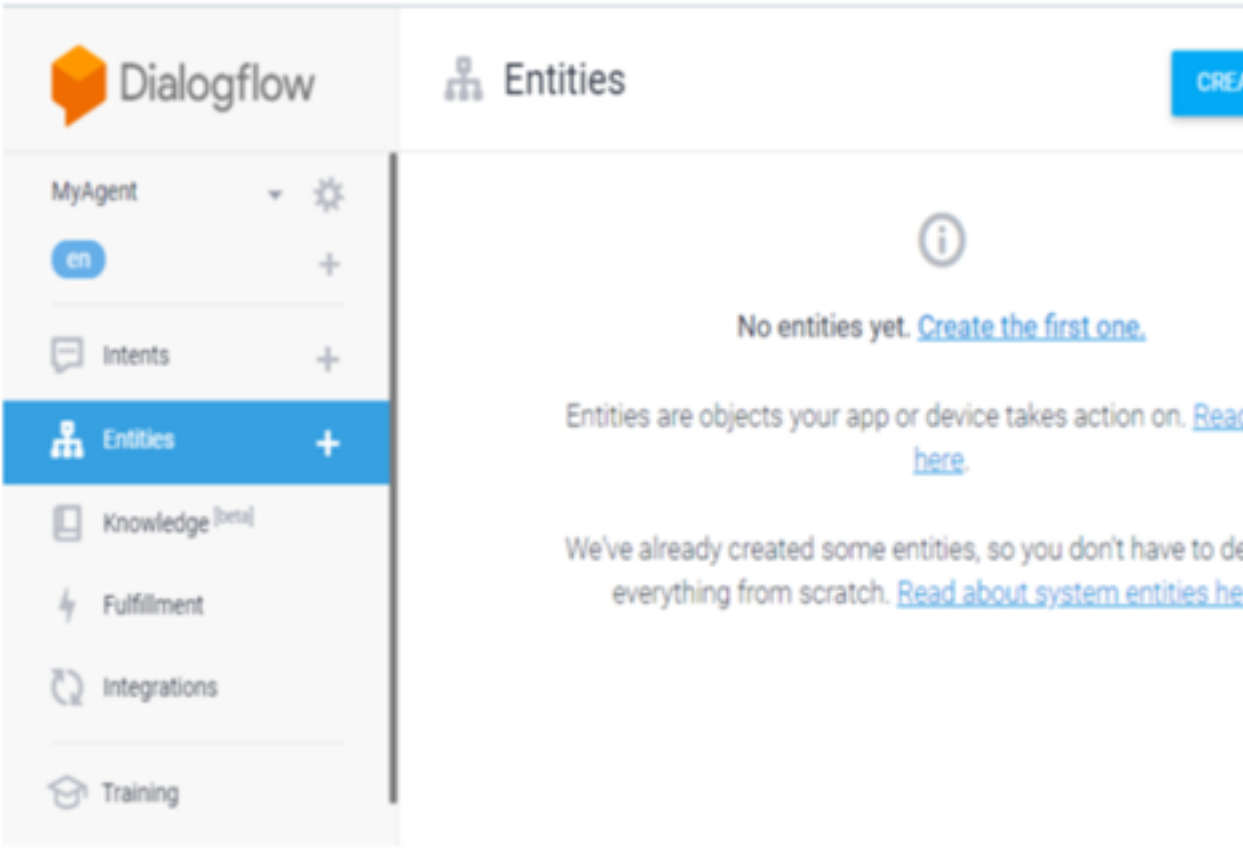
Это означает, что нужно избегать ситуаций, когда похожие запросы сопоставляются с разными намерениями в агенте.

Таким образом, мы узнали о создании намерений и использовании обучающих фраз, чтобы научить агента распознавать эти намерения.

Но допустим, что вы хотите, чтобы ваш агент извлек конкретную информацию, предоставленную пользователем.

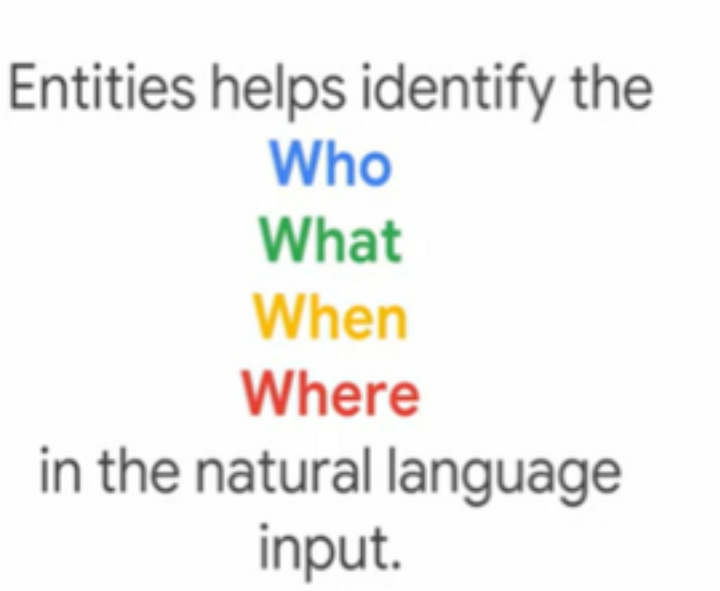
Например, начинку, которую пользователь хочет для пиццы, при ее заказе, или количество ломтиков пиццы.

И вы можете сделать это с помощью сущностей.



Сущности помогают вам разобраться в особенностях взаимодействия с пользователем.

В диалоге сущности – это существительные, найденные в ходе разговора, такие как имя человека, конкретные цифры, даты, и так далее.



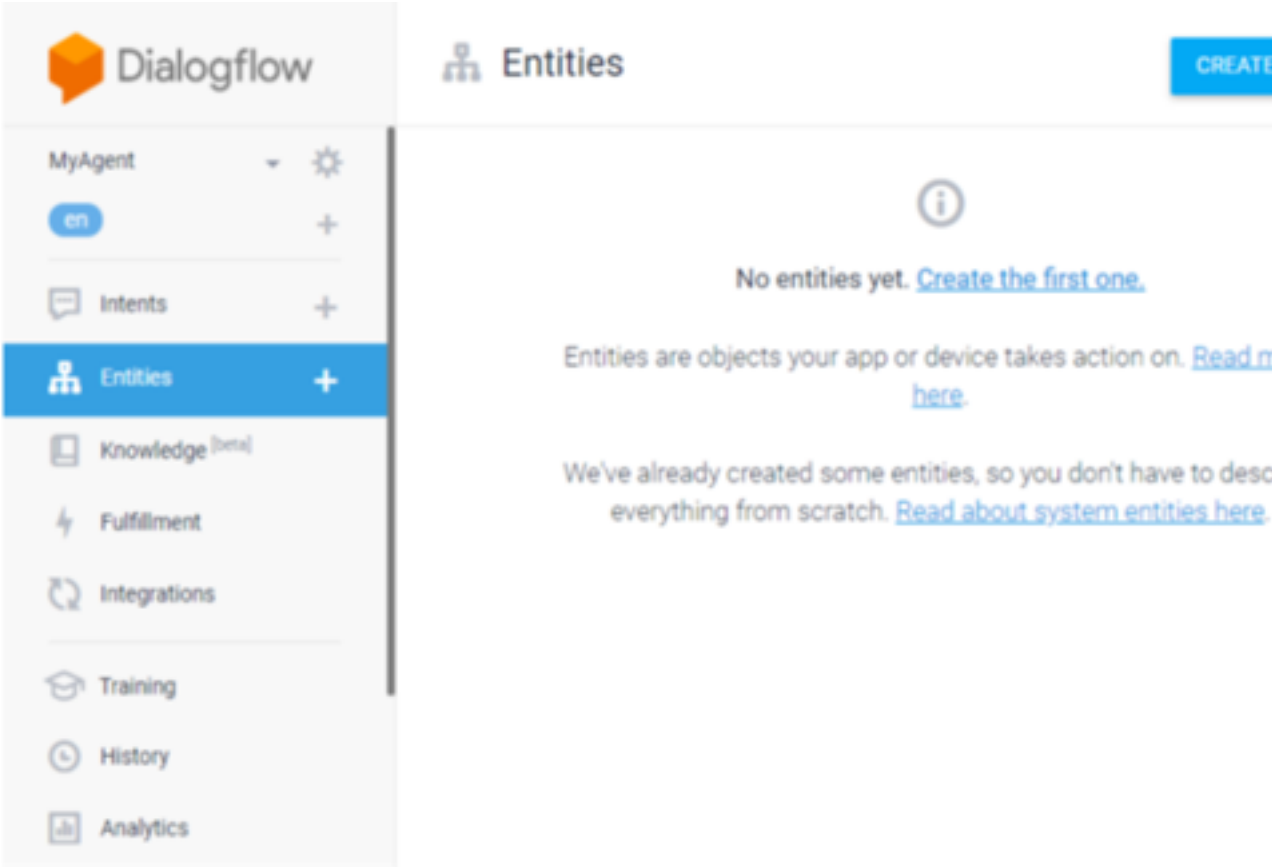
В случае заказа пиццы, пицца будет нести с собой группу атрибутов, которые можно рассматривать как сущности.

При заказе пиццы могут указываться такие атрибуты, как количество пицц, начинка, тип корочки, и время доставки.

Сущности помогают вашему агенту детализировать намерение и решить, как он должен действовать, основываясь на этих деталях.

Сущности также являются отличным способом добавления персонализации.

Вы можете использовать сущности, хранящиеся в базе данных, для запоминания подробностей о пользователе, таких как его имя или предпочтения, затем вы можете отобразить эти детали обратно пользователю, превратив простой диалог в дружескую беседу.



Теперь, давайте посмотрим, как мы можем создавать сущности в Dialogflow.

Чтобы создать новую сущность в Dialogflow, нажмите в правой части Entities, и на этой странице нажмите «Create Entity».

Введите имя сущности.

И здесь вы увидите две опции: одна – определить синонимы, по умолчанию, а другая – автоматическое расширение.

Давайте оставим синонимы и определим термины, чтобы описать начинку пиццы.

Давайте введем сыр.

Когда вы нажмете ввод, вы увидите, что сыр уже добавлен в качестве синонима.

Введем другие начинки, по возможности добавляя синонимы.

И нажмем сохранить Save.

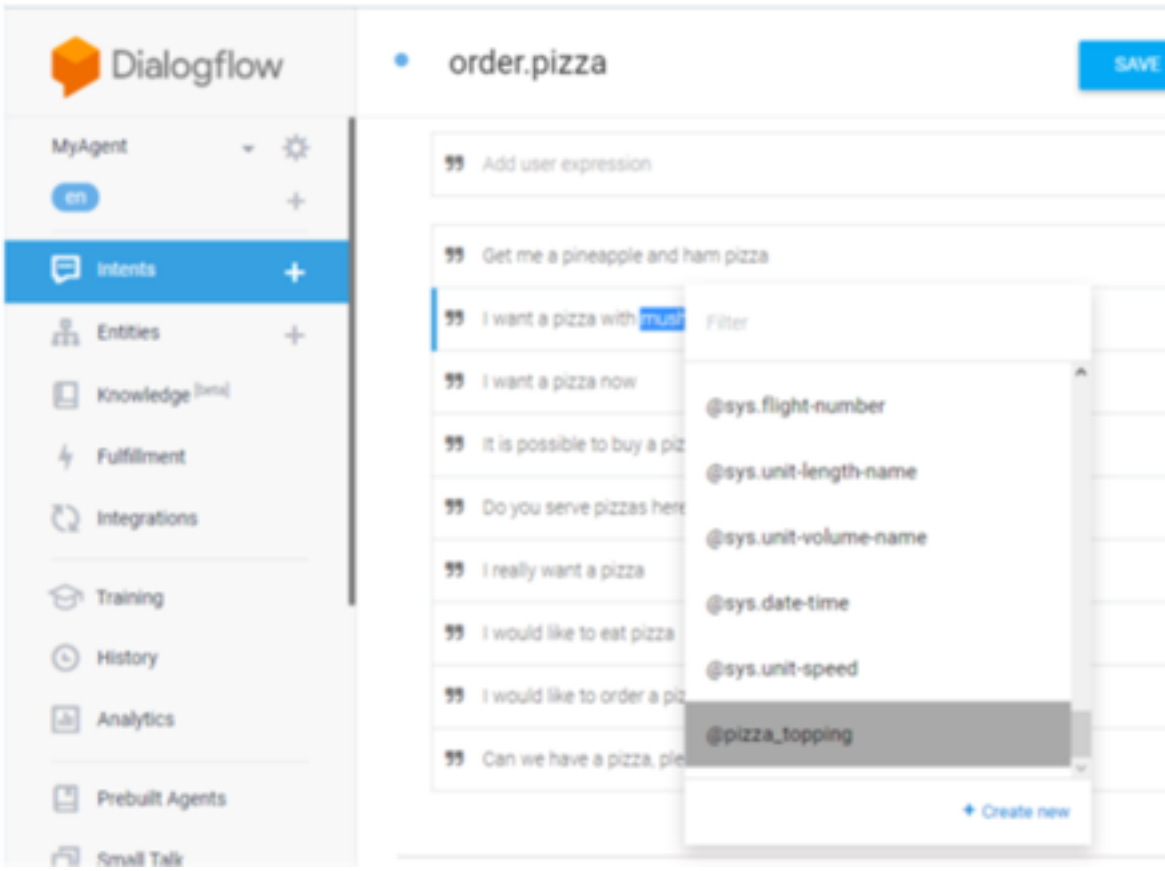
Теперь, что насчет опции автоматического расширения.

Разрешить автоматическое расширение – это означает, что мы хотим разрешить агенту принимать термины, которые могут быть сказаны пользователем и изначально не добавлены в список.

Допустим, пользователь хочет помидоры в пиццу.

Если установлен флажок «Разрешить автоматическое расширение», то, когда пользователь заказывает пиццу и упоминает помидоры, и хотя этой начинки здесь нет, помидоры будут добавлен в список.

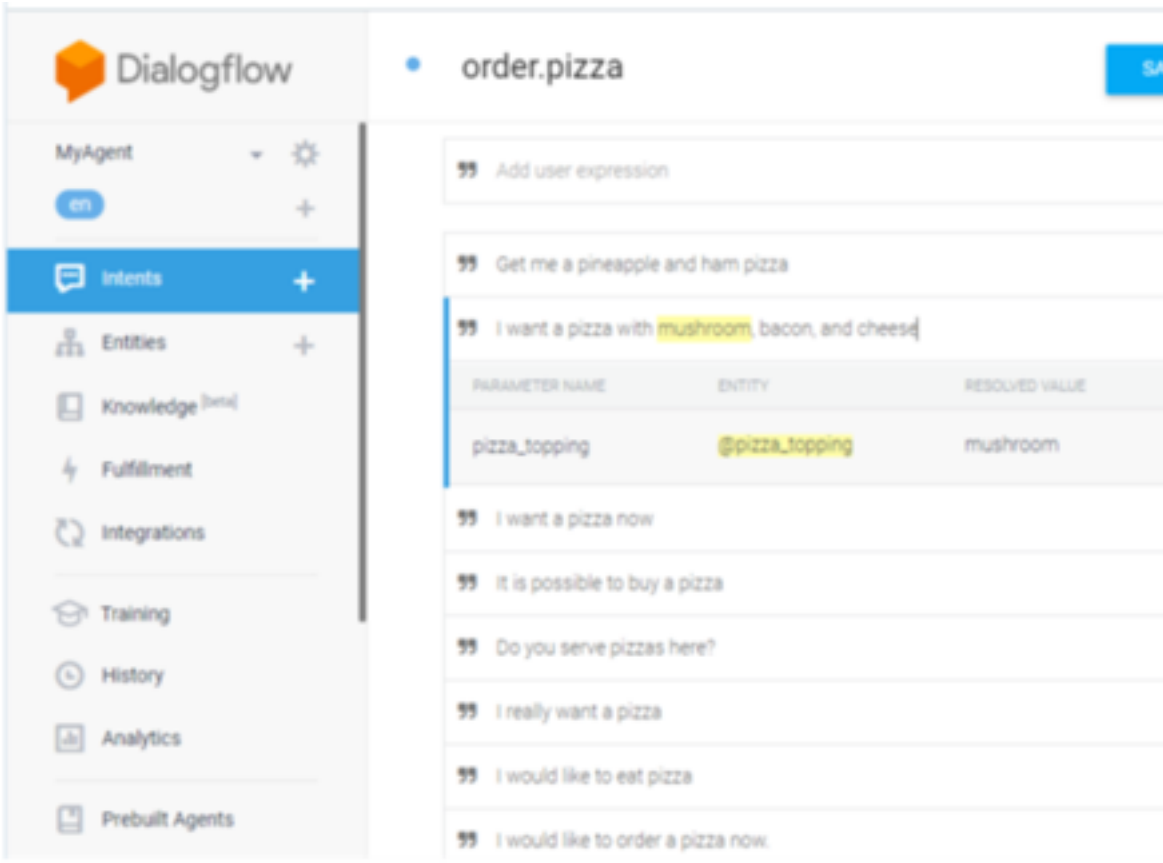
Но если вы хотите зафиксировать этот список начинок, и не хотите, чтобы новые начинки добавлялись в ваш список, вам не нужно включать эту опцию.



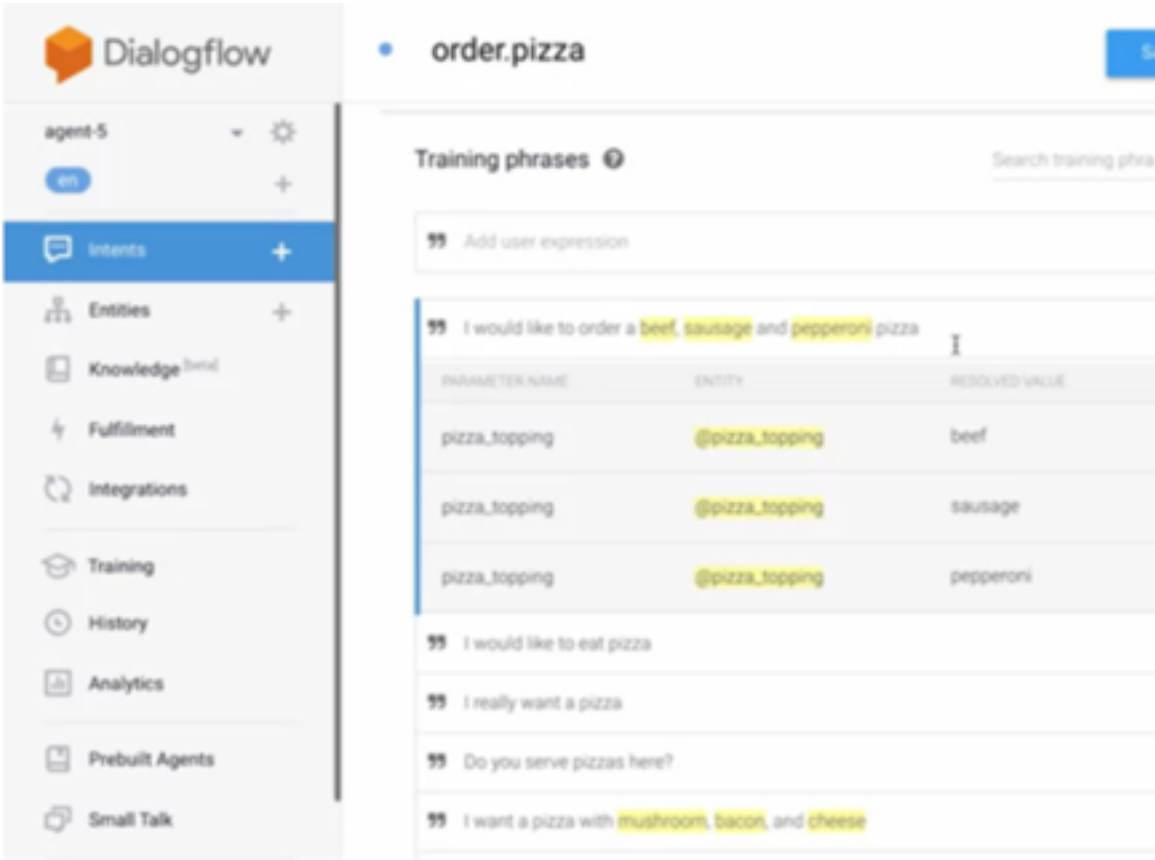
Теперь, мы можем промаркировать наши обучающие фразы намерения сущностью, которую мы только что создали.

Для этого откроем намерение, и в обучающей фразе дважды щелкнем слово, которое мы хотим промаркировать.

И в списке выберем нашу сущность.



Теперь термин грибы помечен сущностью pizza_topping.
Сделаем то же самое с беконом и сыром и с другими начинками.



И в конце не забудем нажать кнопку Сохранить.

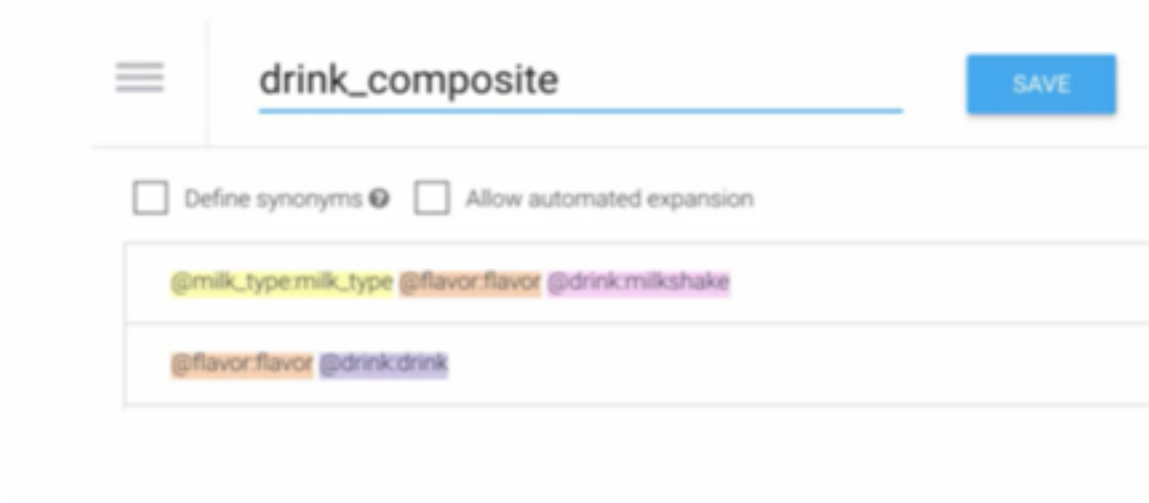
И если мы теперь добавим еще одну обучающую фразу в намерение «Я хотел бы заказать пиццу с говядиной, колбасой и пепперони».

Как только мы нажмем Enter, вы увидите, что все три сущности, которые присутствуют в обучающей фразе, будут промаркированы как pizza_topping автоматически.

И так как агент автоматически маркирует соответствующие значения сущностями, имеет смысл создавать сущности перед добавлением обучающих фраз.

Теперь, когда вы создаете сущность, она может сама содержать атрибуты.

И одним из способов является использование составных сущностей.



Предположим, мы хотим создать намерение для заказа напитка.

Напиток может быть типа молочный коктейль или смузи.

Молочный коктейль и смузи являются записями в «сущности», которая называется напитком.

Кроме того, скажем, у нас есть разные вкусы и типы молока, которые можно выбрать для напитка.

И здесь мы можем использовать составную сущность, чтобы позволить агенту идентифицировать эти атрибуты, когда пользователь заказывает напиток.

Например, можно мне обезжиренный клубничный молочный коктейль?

Для этого случая создайте отдельную сущность тип молока, перечислив все виды молока, сущность вкусы.

А затем объедините эти сущности в составной сущности напитка.

Entity Name	Description	Examples	Returned Data Type	Returned Object Example
@sys.date-time	Matches date, time, intervals or date and time together	2:30 pm 13 July April morning tomorrow at 4:30 pm tomorrow afternoon	String in ISO-8601 format or object: Strings in ISO-8601 format	"2018-04-05T14:30:00-06:00" "2018-07-13T18:00:00-06:00" {"startDate":"2018-04-01T12:00:00-06:00","endDate":"2018-04-30T12:00:0 {"endTime":"2018-04-06T12:00:00-06:00","startTime":"2018-04-06T08:00 {"date_time":"2018-04-06T16:30:00-0 {"startDateTime":"2018-04-06T12:00:0 06:00","endDateTime":"2018-04-06T1
@sys.date	Matches a date	tomorrow	String in ISO-8601 format	"2018-04-06T12:00:00-06:00"
@sys.date-period	Matches a date interval	April	"object: Strings in ISO-8601 format"	{"startDate":"2018-04-01T12:00:00-06:00","endDate":"2018-04-30T12:00:0
@sys.time	Matches a time	4:30 pm	String in ISO-8601 format	"2018-04-05T16:30:00-06:00"

Теперь, скажем, пользователь хочет указать время, когда он хочет забрать свой заказ.

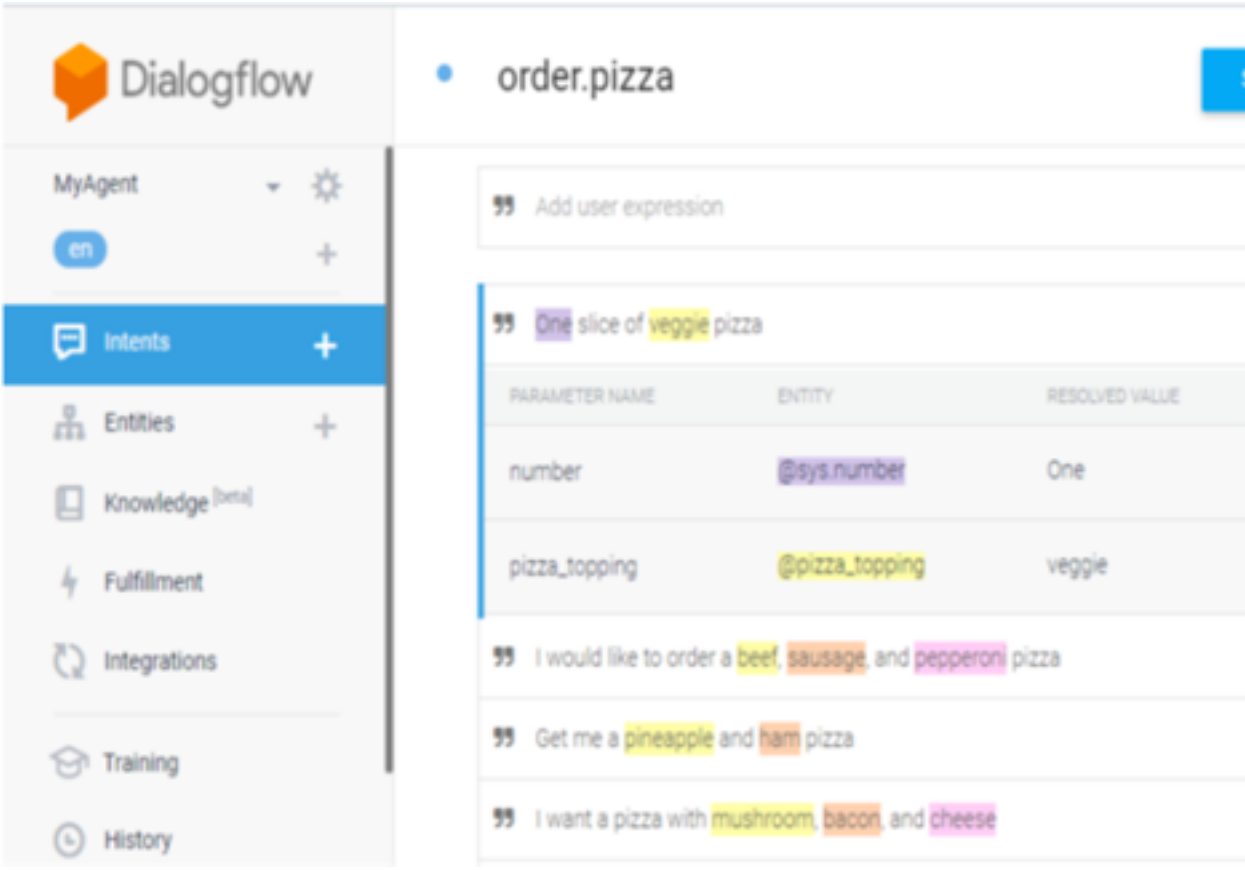
В том случае, агент должен иметь возможность идентифицировать и извлечь время, в стандартном формате.

И агент сможет сообщить это время внутренней системе, ответственной за заказы.

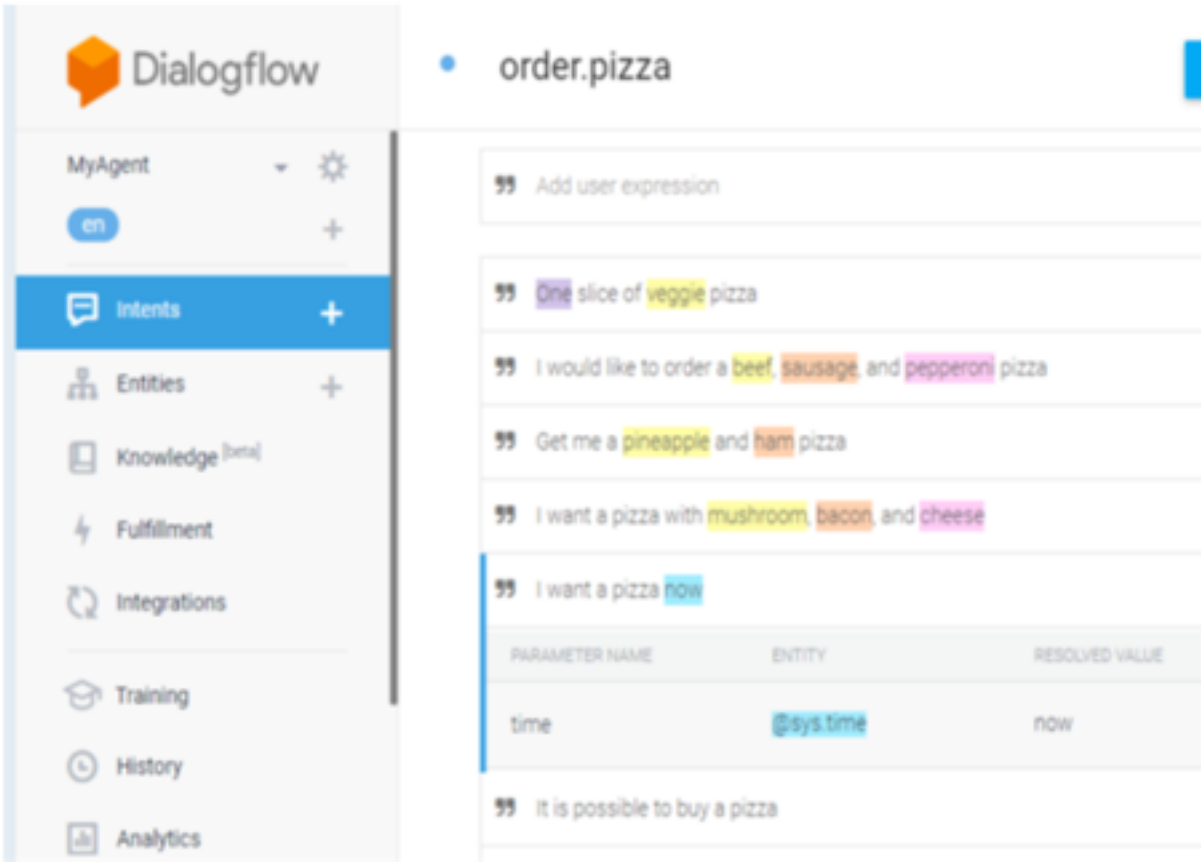
Точно так же иногда нам нужно определить такие общие понятия, как даты, адреса, номера телефонов, имена и так далее.

Для этого можно использовать одну из системных сущностей, например, представляющую дату и время.

Системные сущности – это предварительно созданные сущности в Dialogflow, чтобы упростить обработку наиболее популярных понятий, таких как адреса, валюта, дата, время и многие другие.



Например, в обучающей фразе мы может промаркировать системной сущностью число пицц или время.



При создании сущностей необходимо помнить несколько вещей, чтобы хорошо обучить агента.

Во-первых, важно быть последовательным при маркировке сущностей в обучающих фразах.

Это поможет агенту не запутаться в том, что следует опознать в качестве данной сущности.

Например, не нужно включать предлоги в маркировку в обучающей фразе.

И нужно указать разнообразие примеров конкретной сущности в обучающих фразах.

Это позволит агенту правильно научиться распознавать эту сущность.

Google Dialogflow . Контекст и выполнение

Вы когда-нибудь сталкивались с ситуацией, когда вы подходите к группе людей, и вы ловите себя на том, что пытаетесь понять, о чем они говорят?

Или если к вам приходит друг и говорит: «А как насчет завтра?»

Вы, вероятно, спросите: «Что ты имеешь в виду?»

И в этих случаях вы пытаетесь понять контекст.

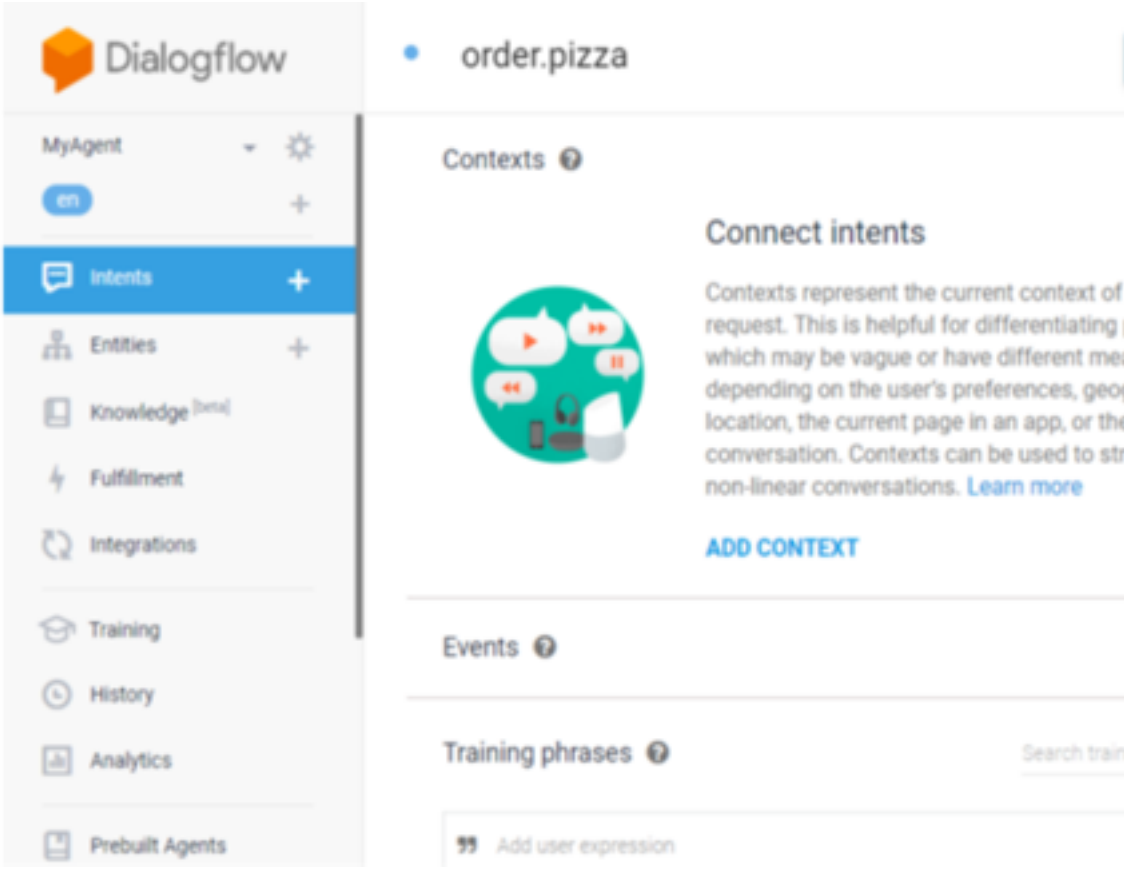
То же самое происходит с чат-ботами, которым нужно знать в каком контексте пользователь общается с чат-ботом.

Например, я спрашиваю: «Что там сегодня на обед?»

И получаю в ответ: «Сэндвич».

Тогда, если я спрошу: «А как насчет ужина?», я ожидаю, что другой человек знает, что я имею в виду то, что мы собираемся съесть, а не то, во сколько мы должны отправиться на обед.

Эти сведения могут быть предоставлены агенту через контекст.

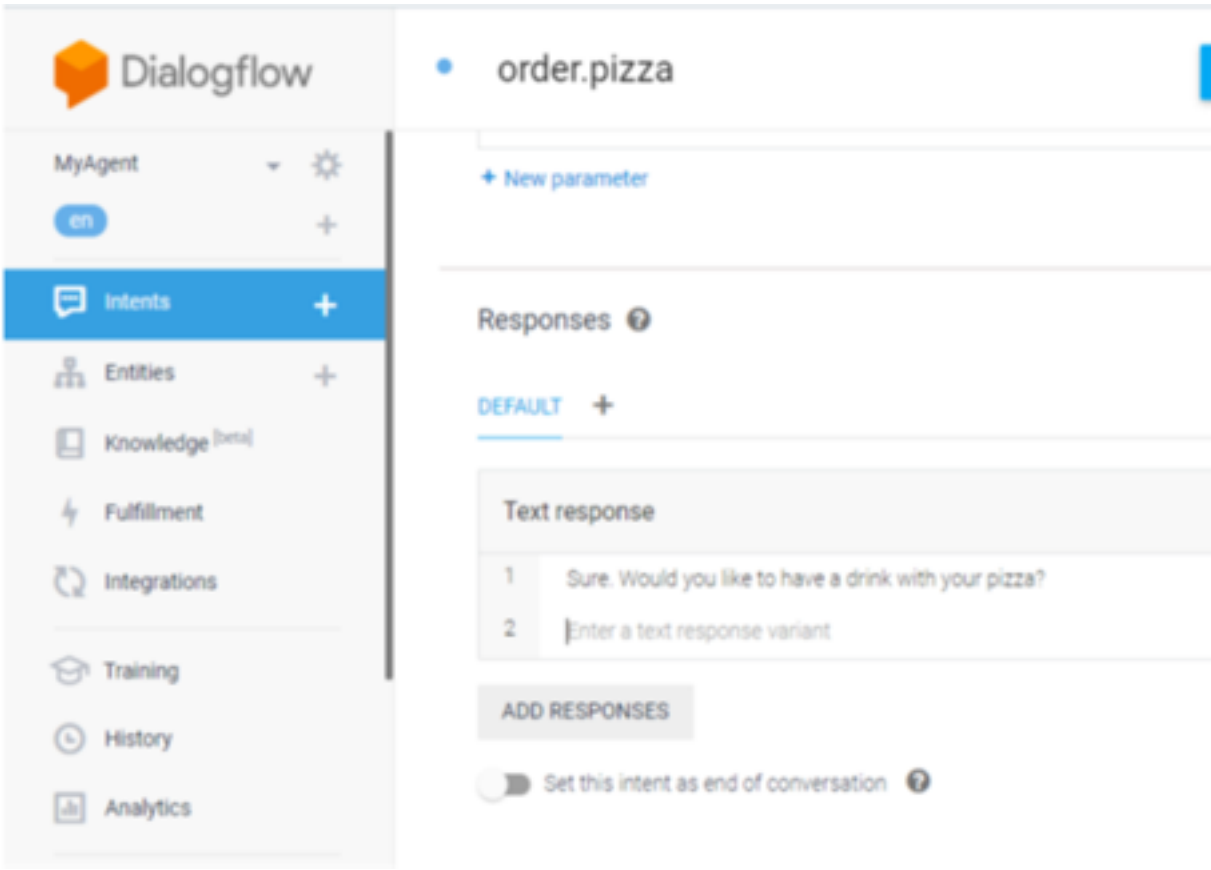


Контекст позволяет агенту отслеживать, где находится пользователь в диалоге.

В Dialogflow, контекст – это средство для приложения восстановить значения переменных, которые были упомянуты в диалоге.

И контекст позволяет агенту контролировать потоки диалога.

Это можно сделать, определив конкретные состояния, в которые диалог должен находиться в случае совпадения с конкретным намерением.

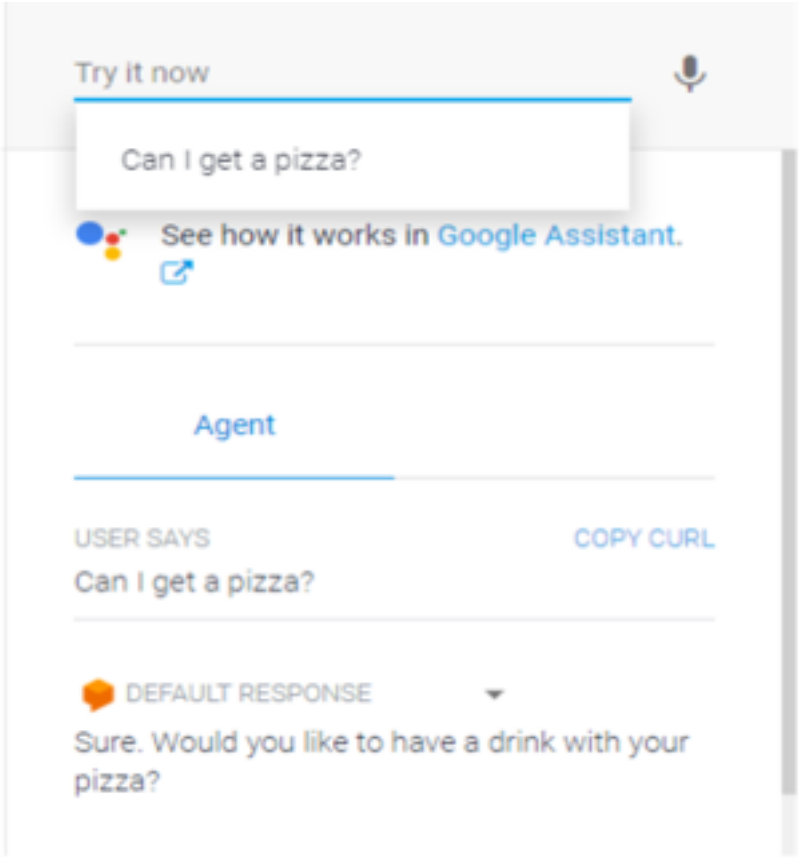


Давайте посмотрим пример того, как добавить контекст к намерению.

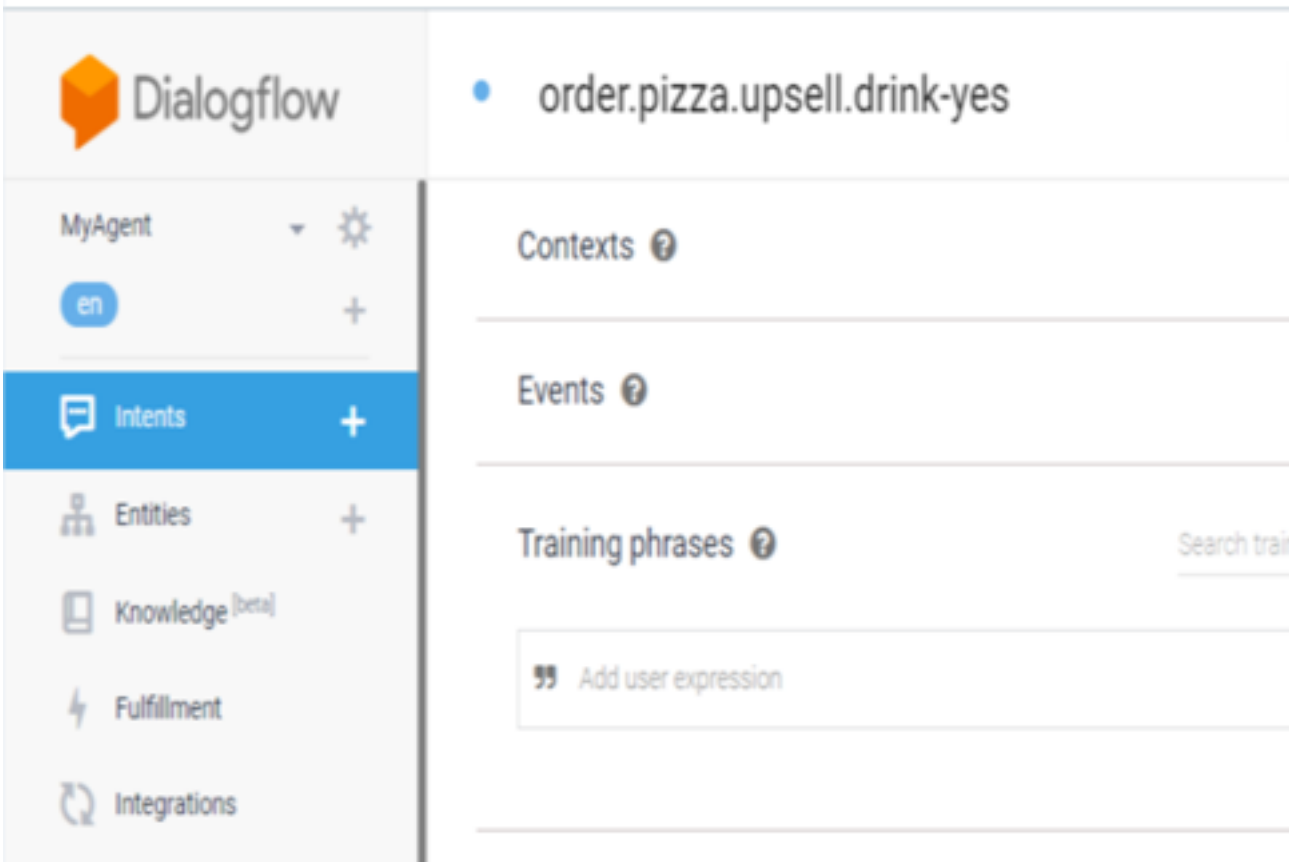
Здесь мы создадим два новых намерения для отрицательных и положительных ответов и добавим к ним контекст.

Но для начала, добавим ответ в намерение order.pizza.

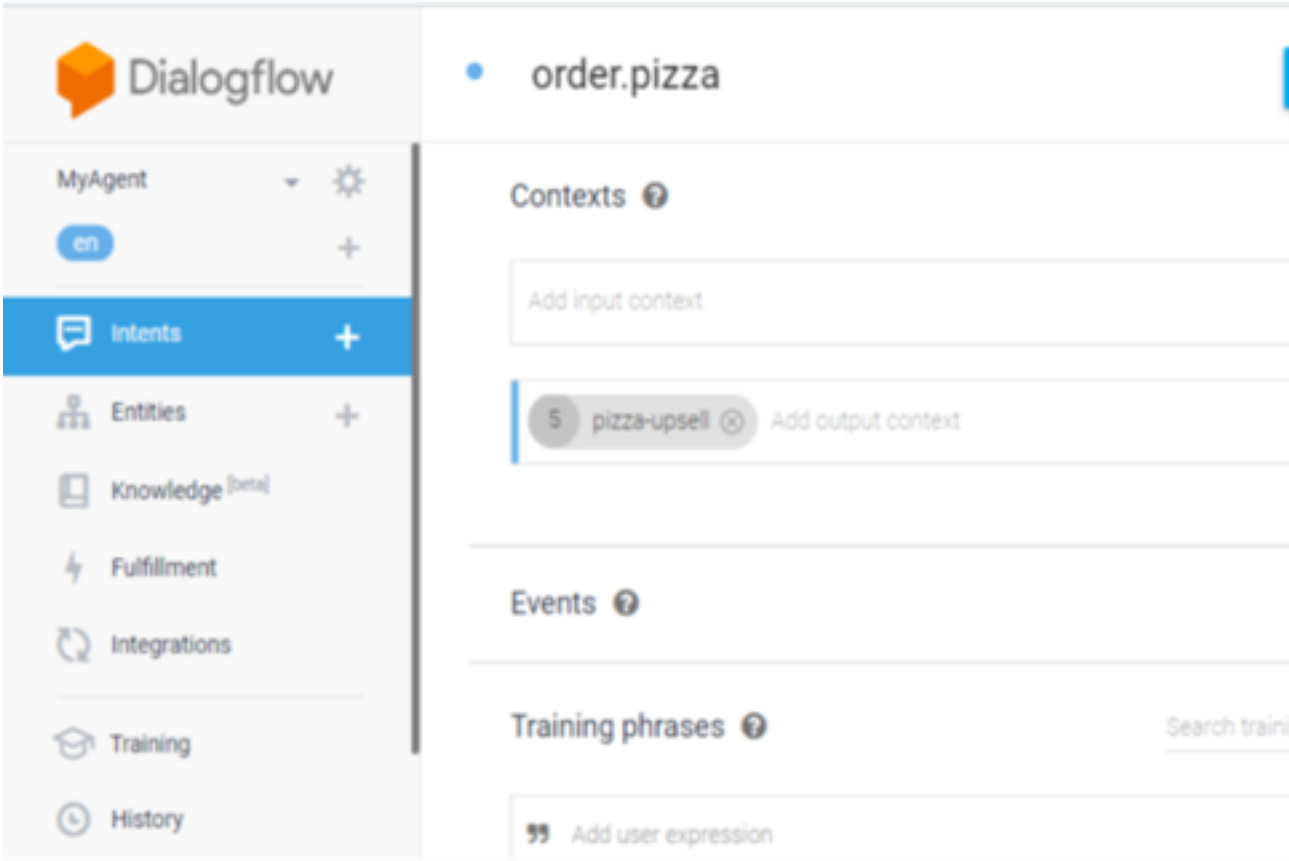
И не забудем нажать кнопку сохранения.



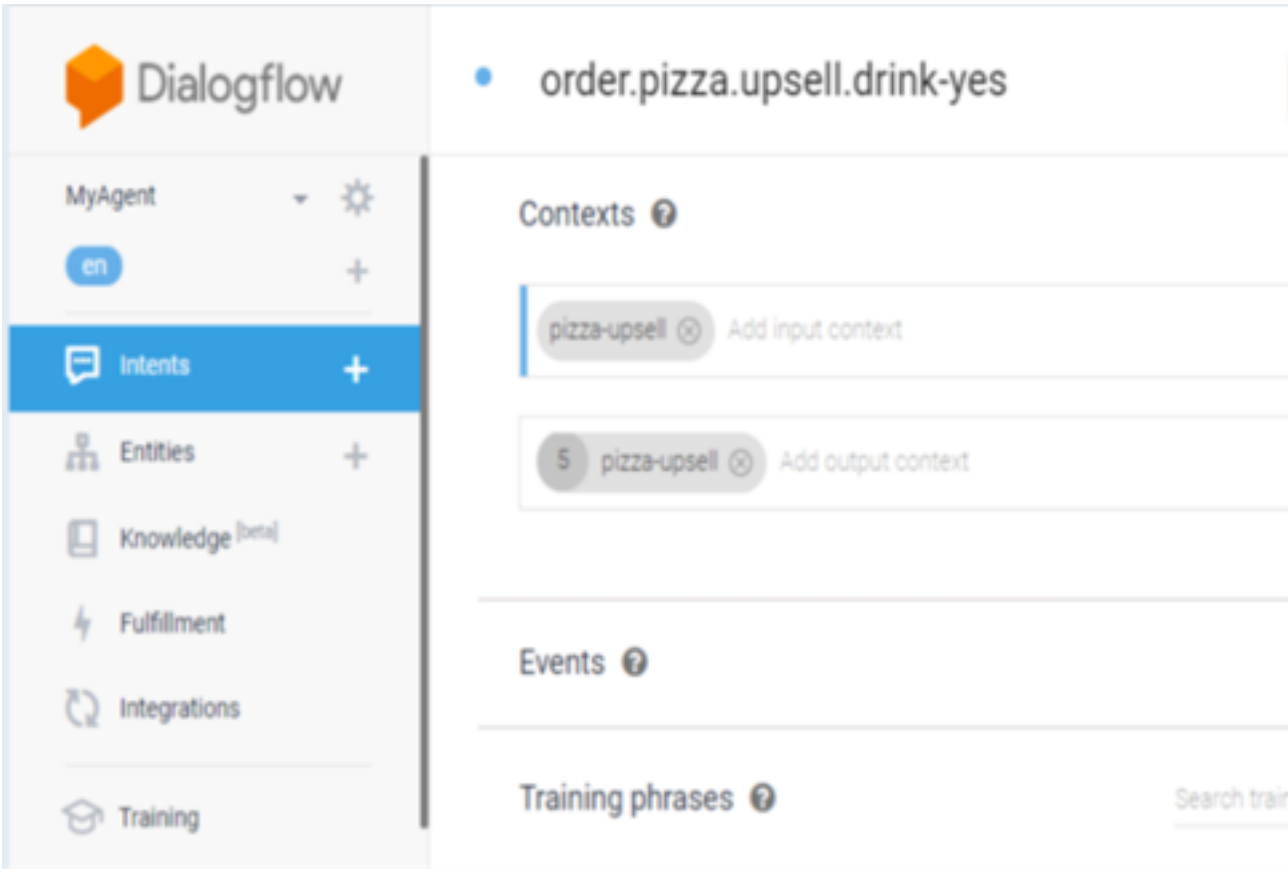
Теперь, когда мы зададим вопрос, «Могу ли я получить пиццу?»
Агент ответит «Конечно. Хотели бы вы получить напиток с вашим заказом?».
И если я просто наберу ответ «Да», агент на самом деле не будет знать, что делать.



Вернемся на страницу «намерения» и создадим новое намерение.
Назовем это новое намерение «Заказать пиццу и дополнительно напиток – да».
Нажмем кнопку сохранения и вернемся в намерение order.pizza.



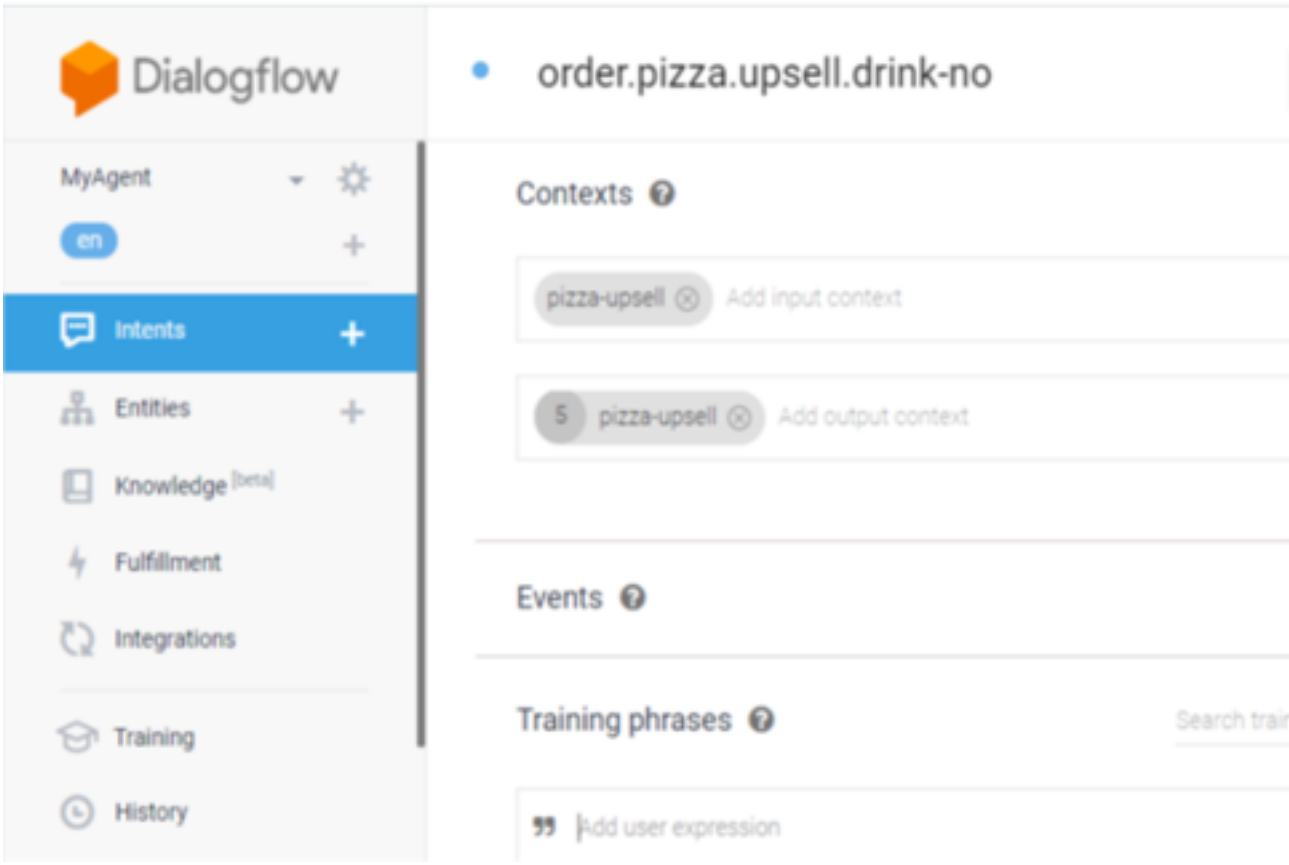
И здесь создадим выходной контекст pizza-upsell и сохраним намерение.
И когда мы это сделаем, вы можете заметить, что к контексту добавилось число 5, и это означает продолжительность жизни контекста.
Таким образом, этот контекст будет активным для пяти взаимодействий.
Теперь, мы можем предоставить этот же контекст, как входной контекст для нашего нового намерения.



Добавим контекст pizza upsell в качестве входного контекста в это намерение.

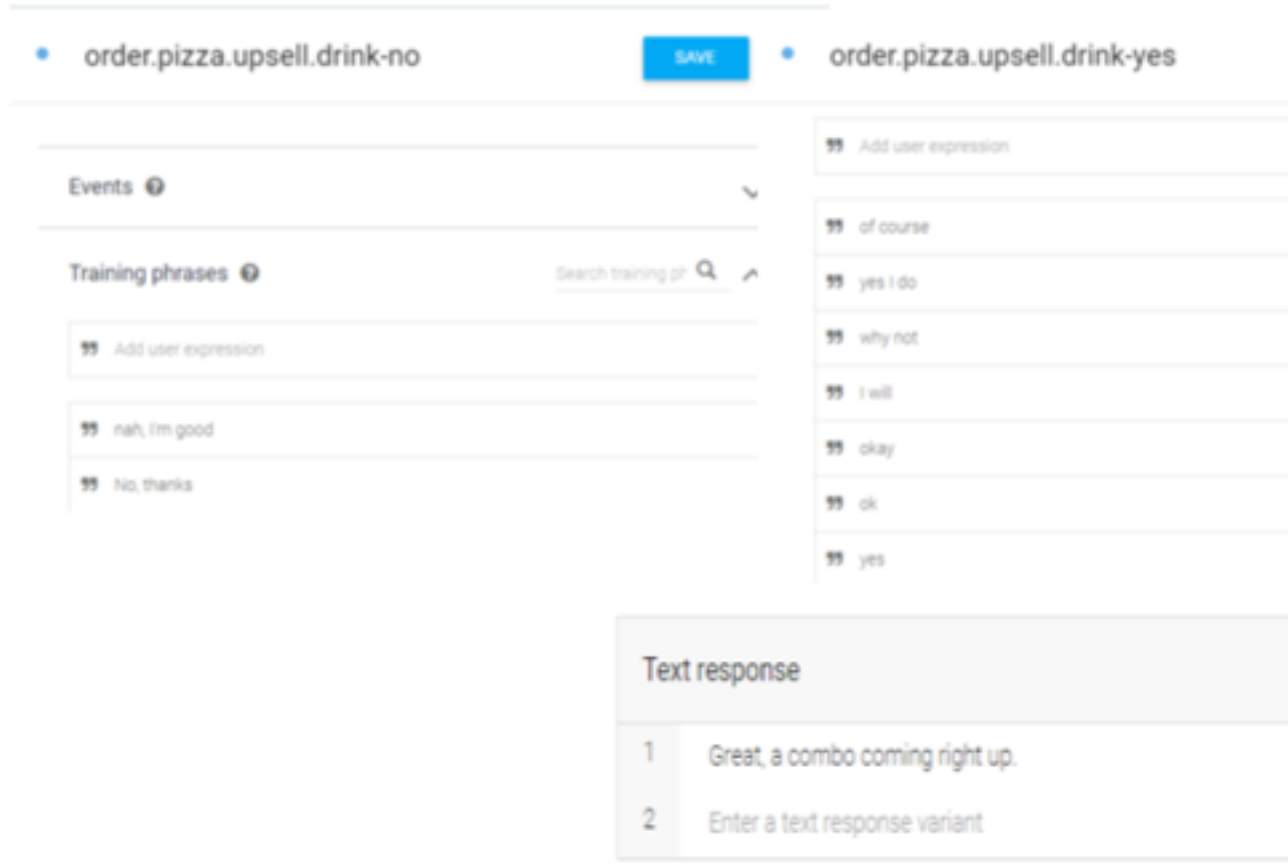
Таким образом, при повторном заказе, когда пользователь закажет пиццу, агент распознает намерение, и активирует этот контекст.

А затем агент прослушает ответ и попытается определить, это да или нет.



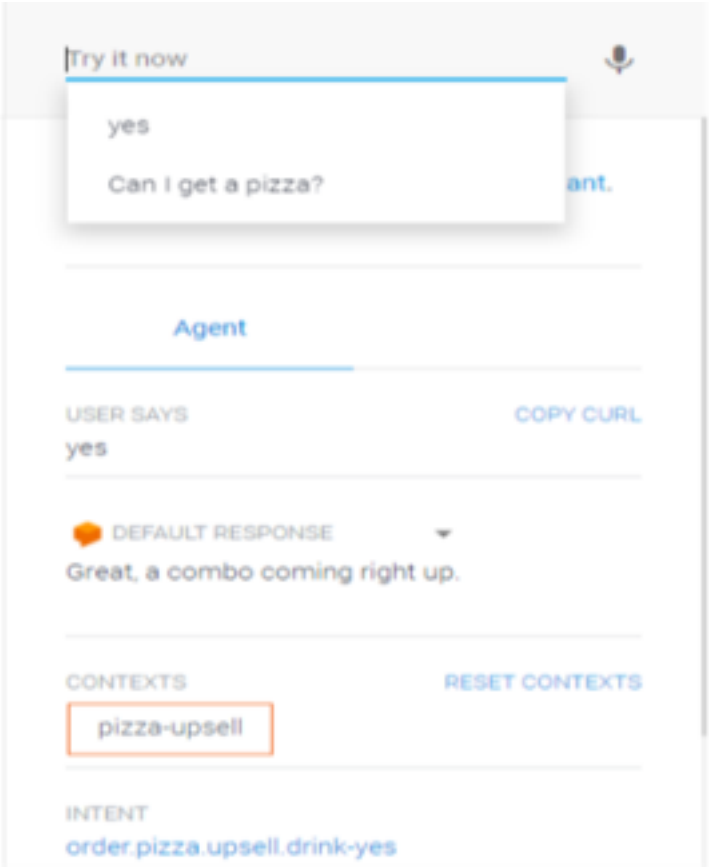
И мы создадим такое же намерение для отрицательного ответа, и этим же контекстом в качестве входного контекста.

Теперь у нас есть два намерения, но нам нужно добавить для них обучающие фразы.



Для намерения нет, мы добавим фразы с отказом, а для намерения да, мы добавим подтверждающие фразы.

И добавим ответ в это намерение.



Теперь давайте проверим.

Давайте зададим вопрос: «Могу ли я получить пиццу?»

Агент скажет: «Конечно, вы хотели бы получить напиток с пиццей?»

И если я скажу «да», тогда ответ будет: «Отлично, скоро будет».

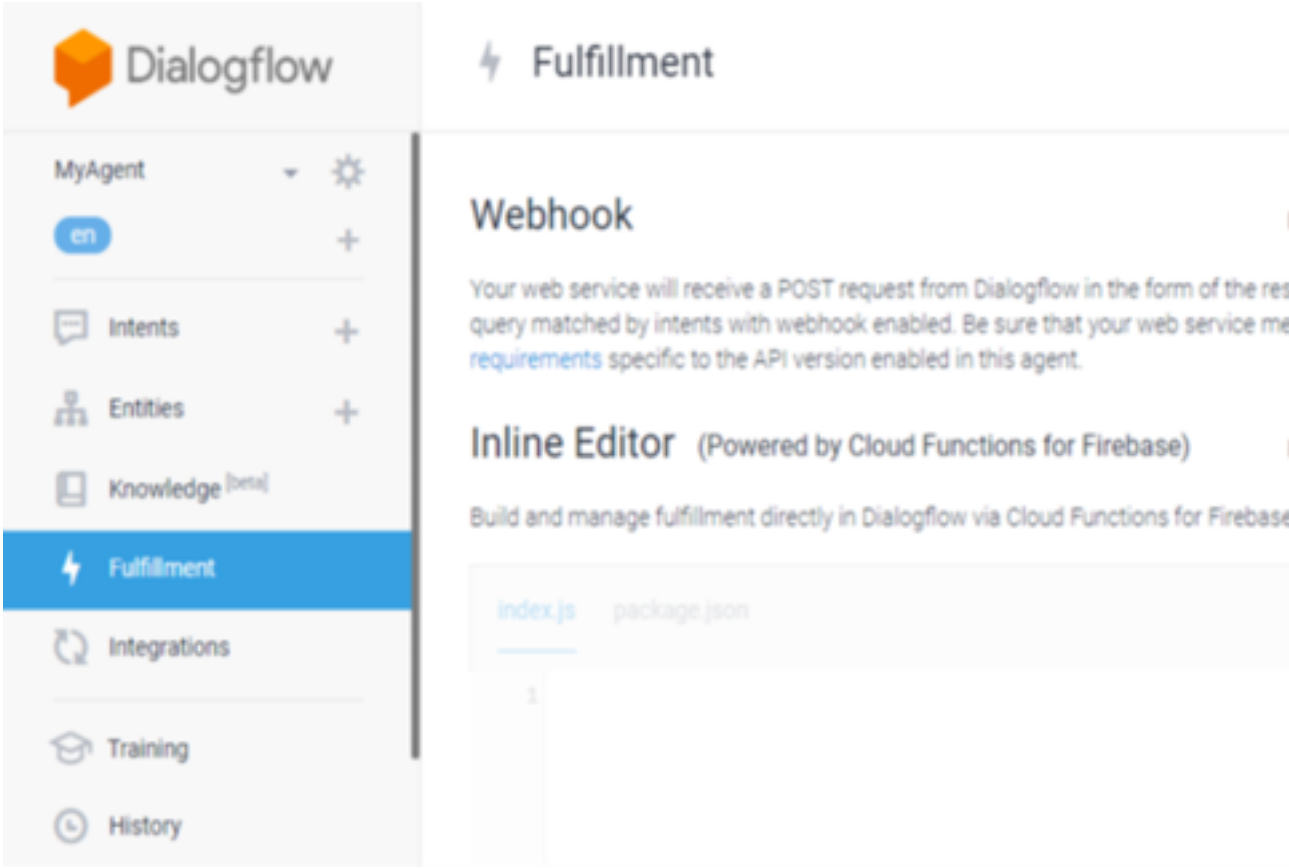
Теперь мы видим, что «да» связано с заказом пиццы с напитком.

В случае нет, мы должны просто разместить заказ на пиццу.

Теперь, что, если вы хотите, чтобы агент сделал больше, чем просто давал ответы пользователю?

Что если вы решите сохранить заказ пиццы в базе данных?

Вы можете достичь этого с выполнением fulfillment.



Выполнение – это действие с использованием кода, развернутого вне диалога.

Это позволяет чат-боту выполнять внешнюю бизнес-логику на основе намерения.

После обнаружения намерения, которое соответствует действию, агент должен иметь возможность обратиться к внешней системе для выполнения действия.

И мы можем написать код для этого взаимодействия с внешней системой.

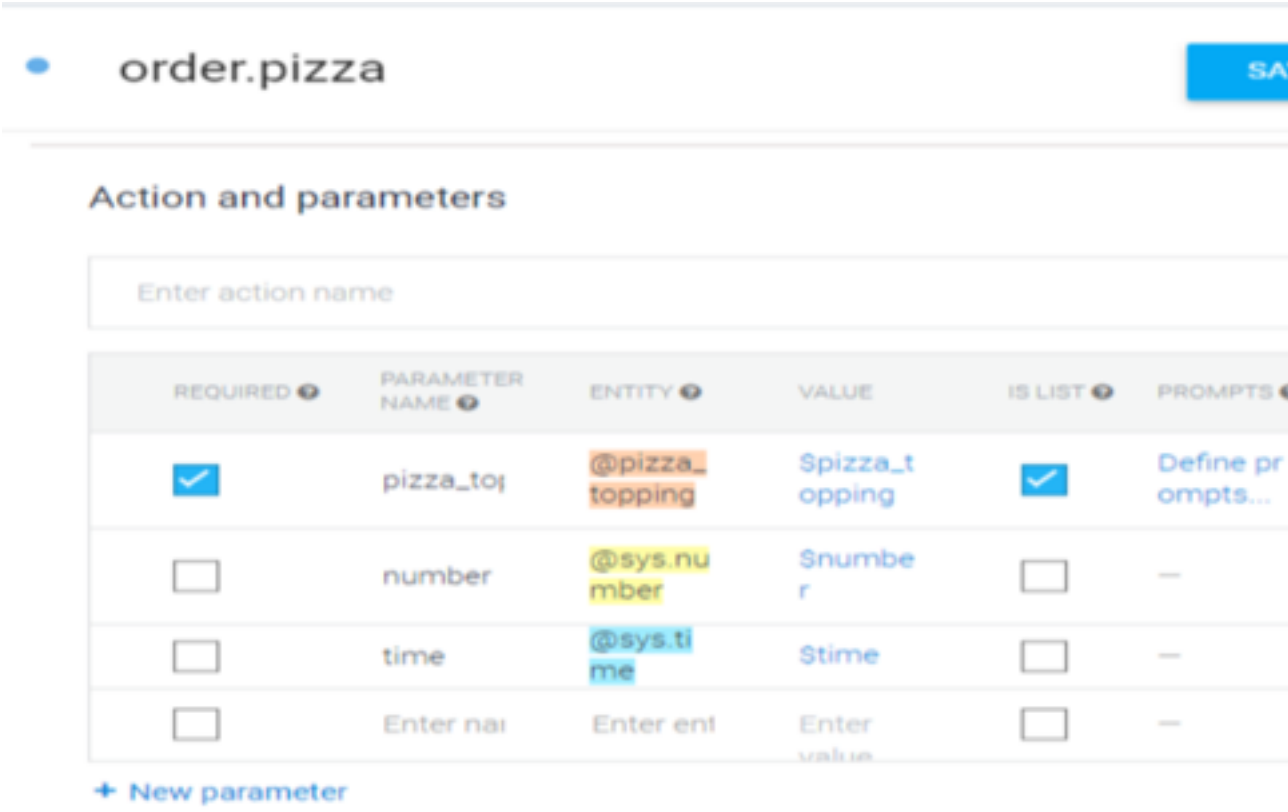
Здесь мы будем использовать встроенный редактор DialogFlow для написания кода.

Для размещения заказа пиццы, серверная сторона должна знать как минимум три фрагмента информации; размер пиццы, начинку и время получения заказа.

Это будут три разных сущности, которые нам необходимо идентифицировать и извлечь из запроса клиента.

Если клиент говорит: «Можно мне пиццу?», нам нужно настроить агента запросить дополнительную информацию, необходимую для отправки заказа в бэкэнд-систему, ответственную за размещение заказов.

Как мы можем собрать эти недостающие фрагменты информации?

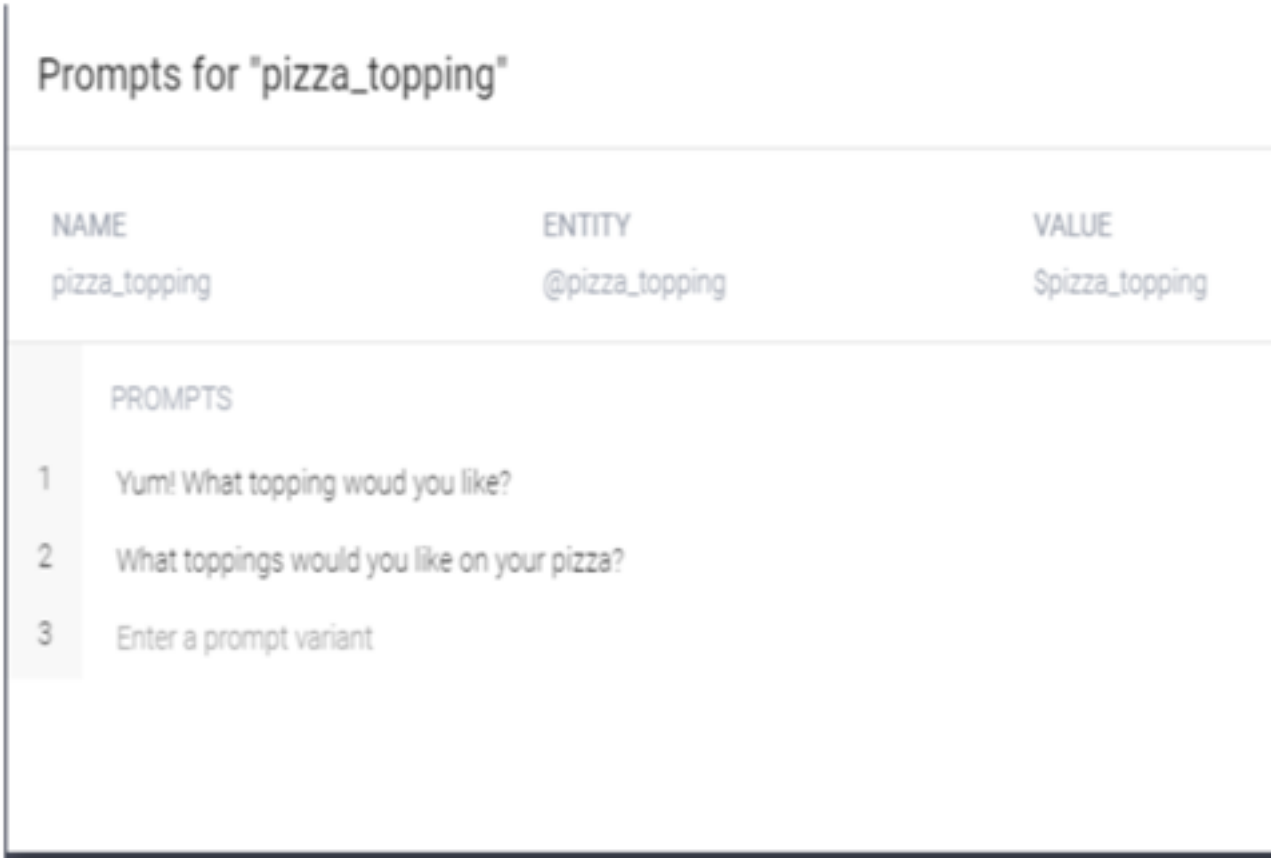


Для этого мы можем использовать раздел действия и параметры намерения.

Здесь вы можете установить необходимые значения параметров, соответствующие сущностям в запросе.

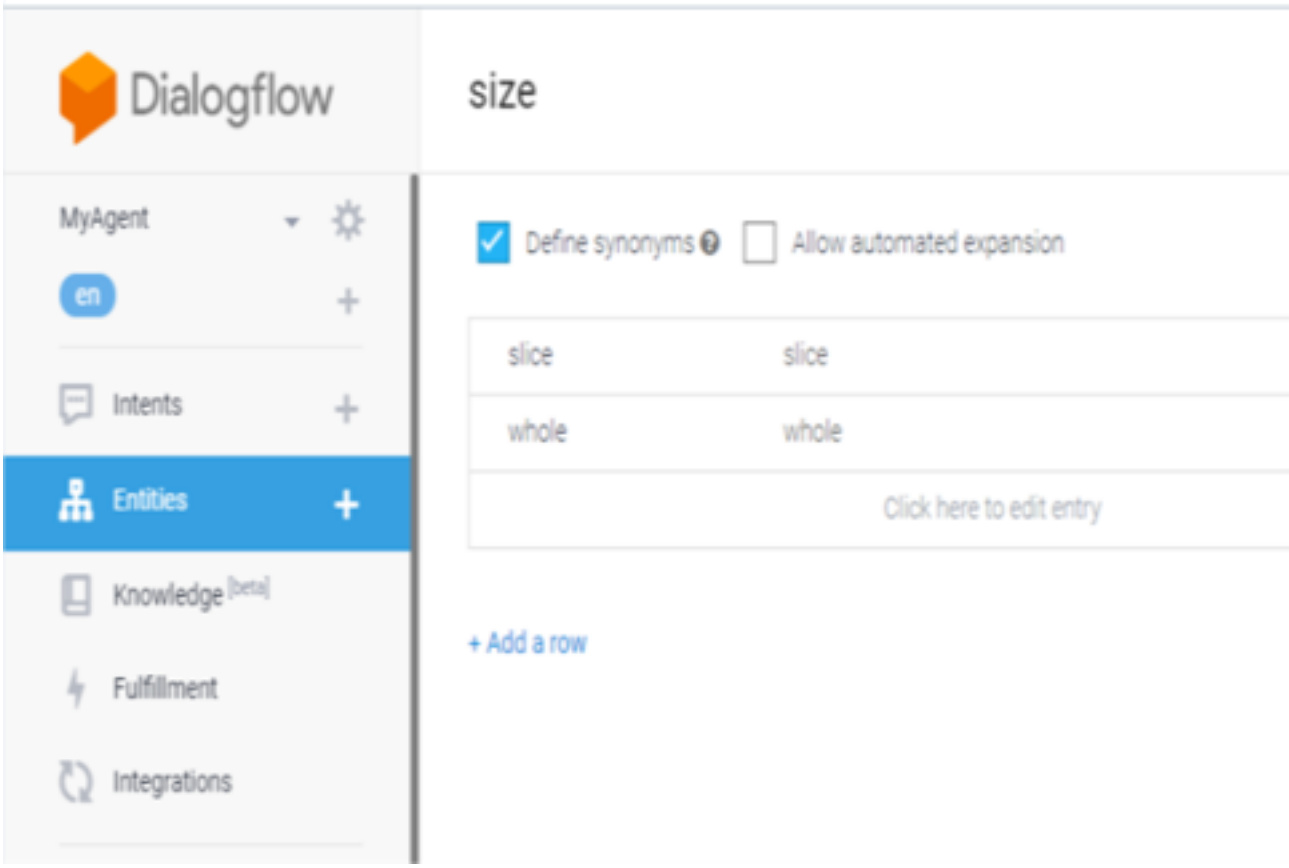
Если пользователи опустят один или несколько параметров в своем ответе, ваш агент попросит их указать значения для каждого пропущенного параметра.

Поэтому в разделе действия и параметры отметим параметр pizza_topping и нажмем Define prompts.

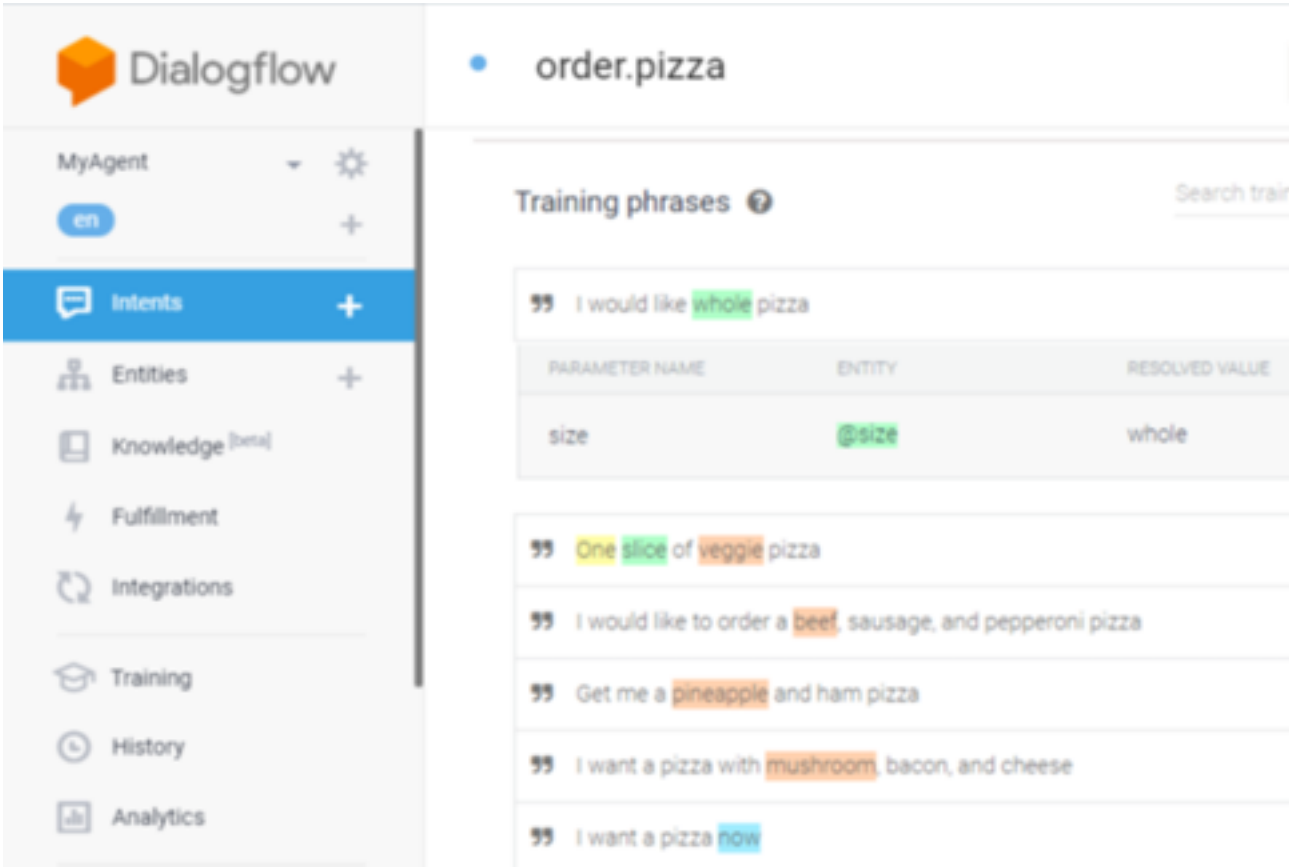


И здесь мы введем вопросы, которые чат-бот задаст, если не обнаружит в намерении пользователя сущность pizza_topping.

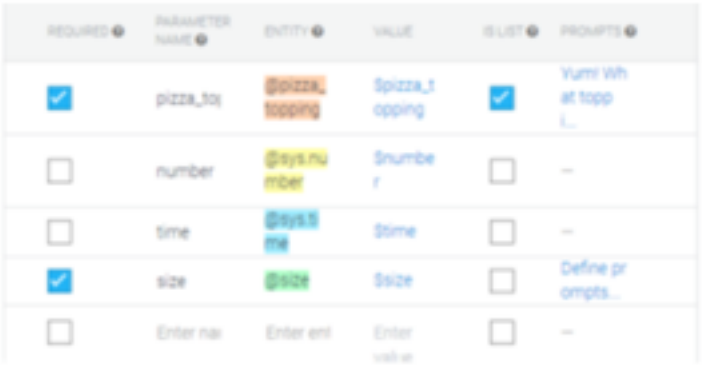
И здесь вы также можете заметить, что отмечена опция «Список» для начинки, чтобы агент распознавал несколько начинок в запросе.



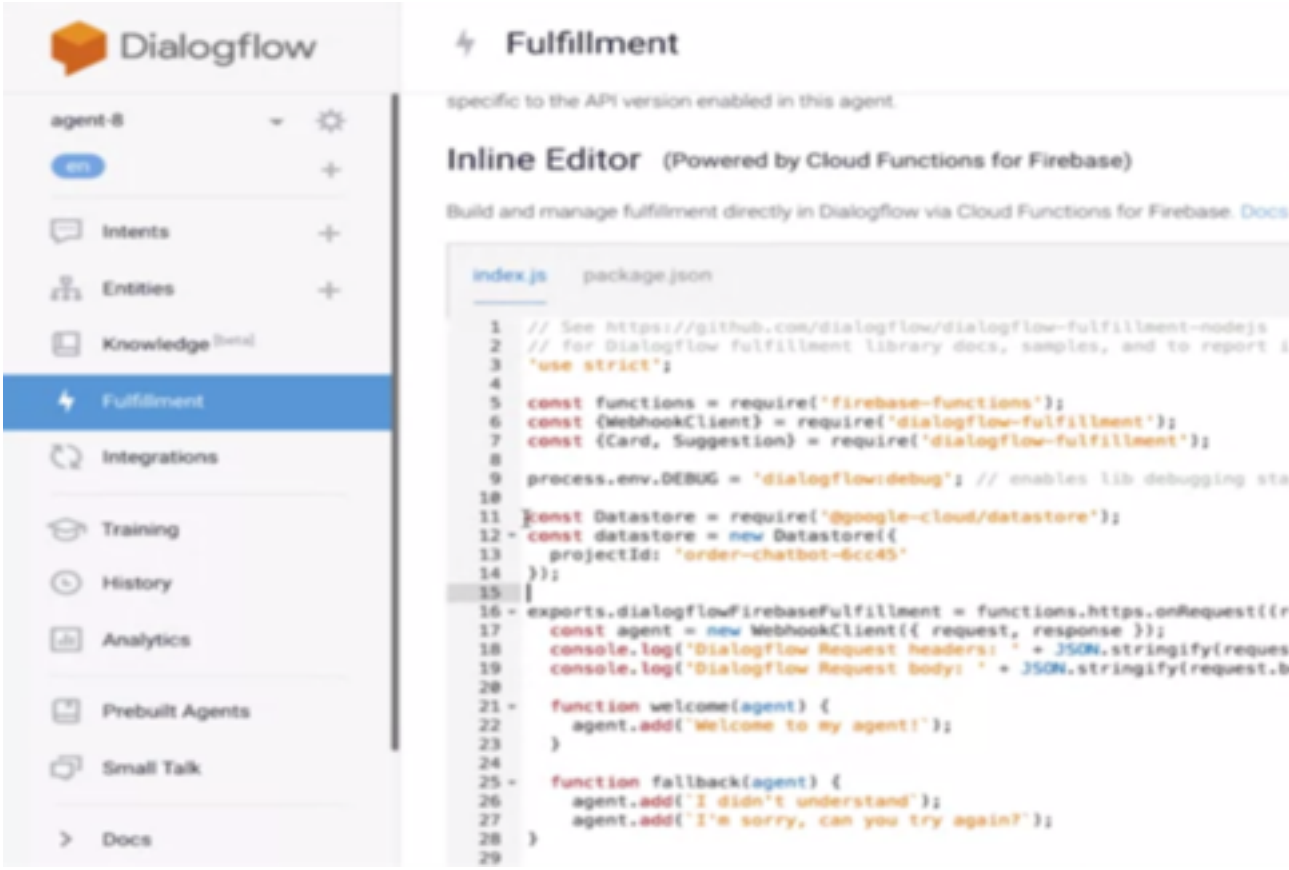
И мы создадим сущность размер size.
Далее вернемся в намерение и разметим его обучающие фразы этой сущностью.



Далее перейдем в раздел действия и параметры.



И здесь отметим параметр size и нажмем Define prompts.
И здесь введем уточняющий вопрос.
Таким образом, здесь мы добавим: «Хотите кусок или целый пирог?»
Это позволит агенту запросить информацию, если она не была захвачена.



Теперь, переключимся на выполнение.

И здесь мы видим встроенный редактор, который мы активируем.

И вы увидите, что здесь уже есть шаблон с некоторым кодом, написанным на nodeJS.

Этот код представляет собой веб-приложение nodeJS webhook, которое будет развернуто в Google сервисе Firebase.

Webhook – это механизм получения уведомлений об определённых событиях.

В нашем случае – это механизм уведомления об обнаружении определенного намерения чат-ботом.

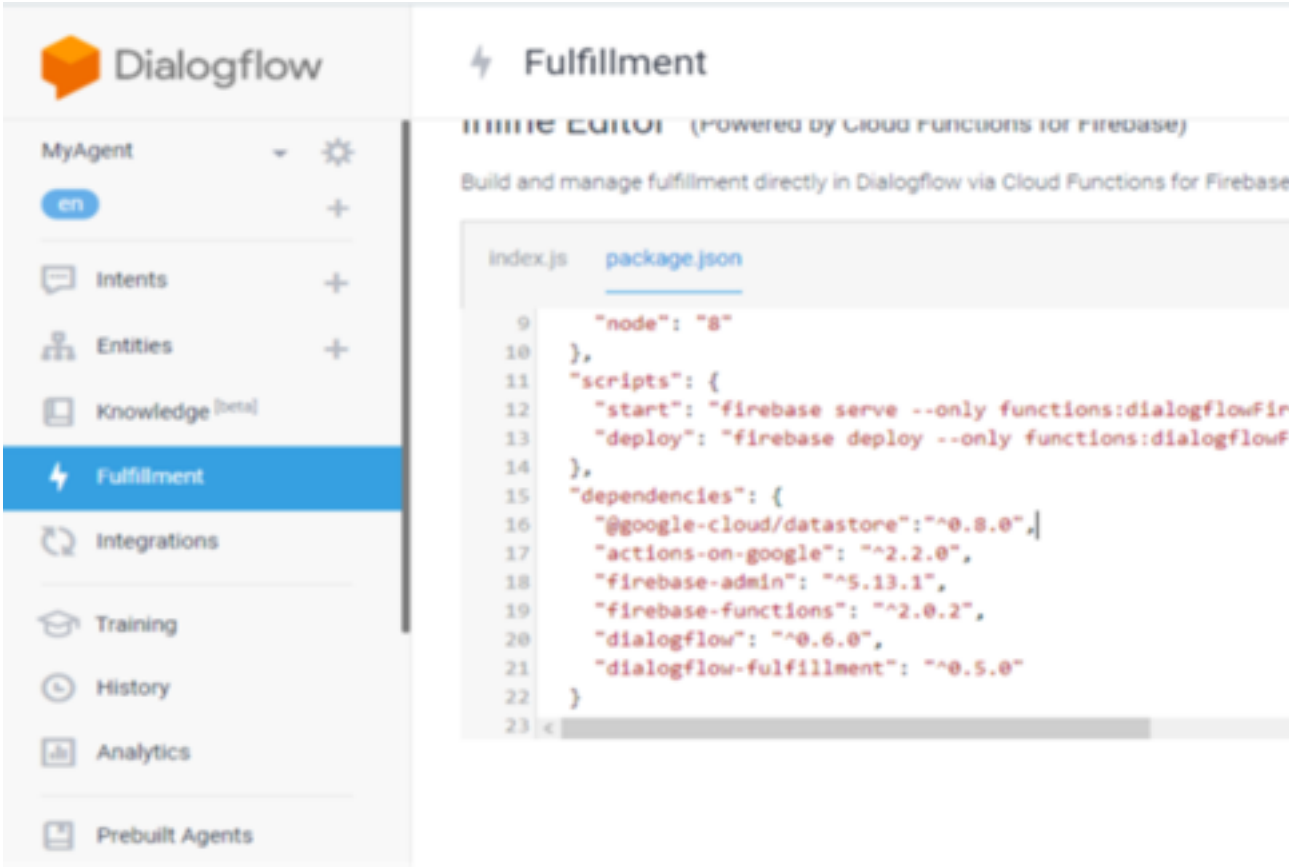
И webhook в нашем примере будет развернут с помощью облачной функциональности Cloud Functions for Firebase, которая позволяет автоматически запускать код в ответ на события, вызванные HTTP-запросами.

Ваш код хранится в облаке Google и работает в управляемой среде.

После того, как вы напишите и развернете код, серверы Google сразу же начнут управлять этой облачной функцией.

И для нашего чат-бота бесплатного плана Spark Firebase будет достаточно.

И здесь во встроенном редакторе, у нас также есть файл package.json, и нам нужно изменить его.

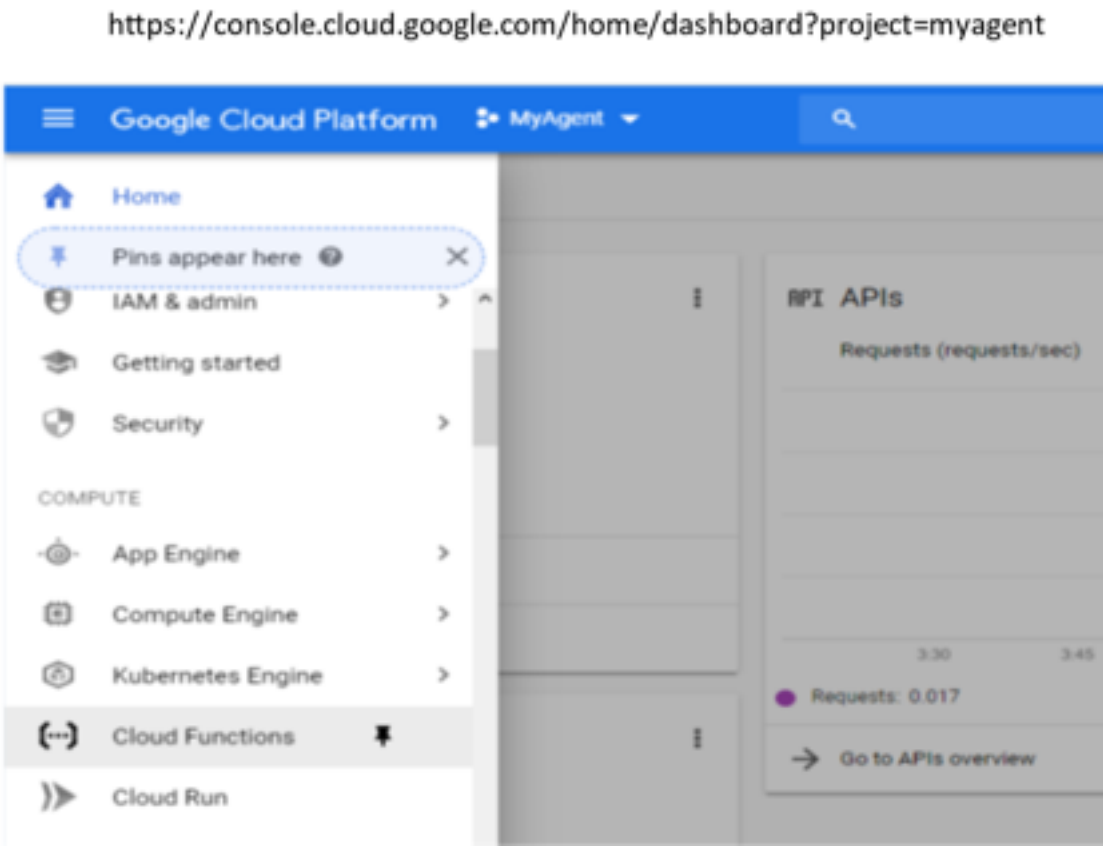


Нам нужно добавить зависимость от Google базы данных Datastore, которую мы будем использовать для хранения заказа пиццы.

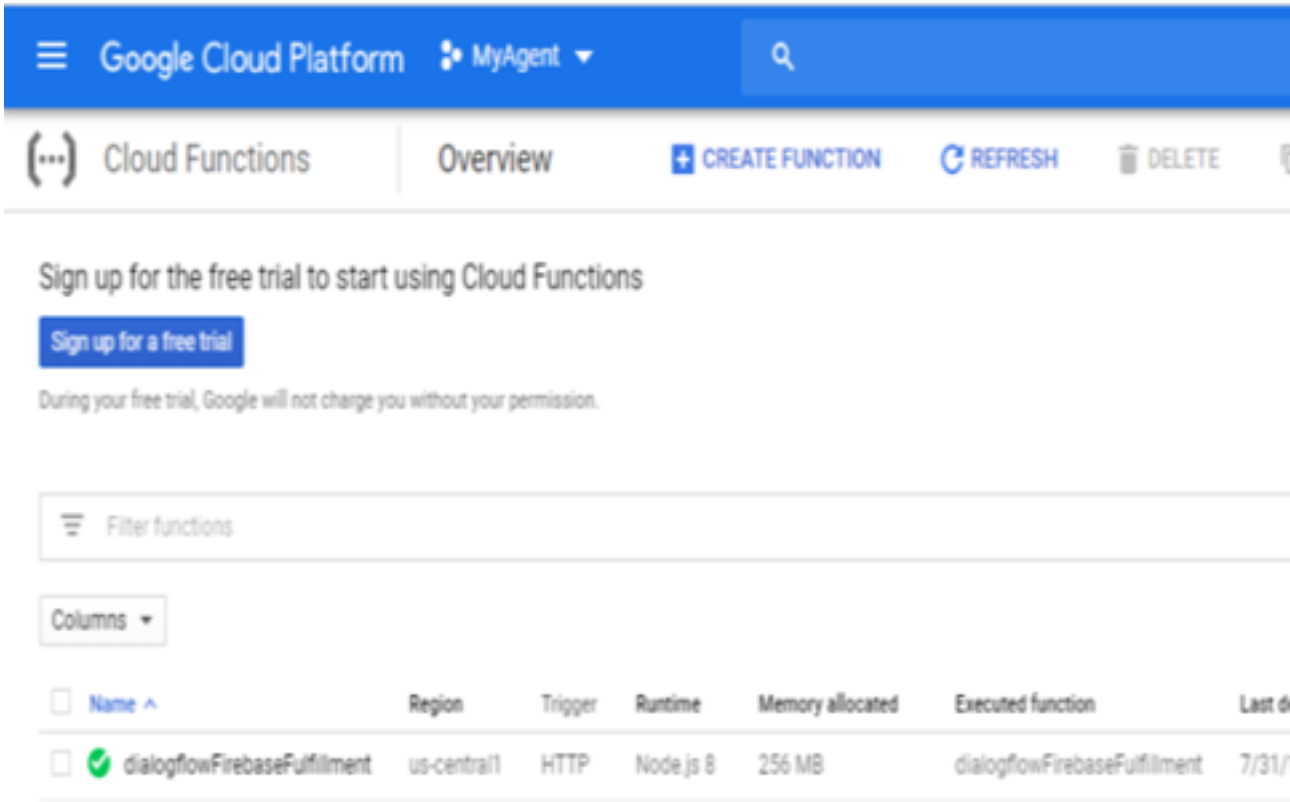
Поэтому мы добавим @google-cloud/datastore.

После этого нажмем кнопку Deploy развернуть.

В результате в наш проект будет добавлен облачный сервис Cloud Functions for Firebase, где будет развернут наш webhook.



Чтобы проверить развернут ли наш webhook, откроем страницу нашего Google проекта Dialogflow и нажмем Cloud Functions.



И здесь мы увидим нашу развернутую облачную функцию.

```
// See https://github.com/dialogflow/dialogflow-fulfillment-nodejs
// for Dialogflow fulfillment library docs, samples, and to report issues
'use strict';

const functions = require('firebase-functions');
const {WebhookClient} = require('dialogflow-fulfillment');
const {Card, Suggestion} = require('dialogflow-fulfillment');

process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging statements

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request,
  const agent = new WebhookClient({ request, response });
  console.log('Dialogflow Request headers: ' + JSON.stringify(request.header
  console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

  function welcome(agent) {
    agent.add(`Welcome to my agent!`);
  }

  function fallback(agent) {
    agent.add(`I didn't understand`);
    agent.add(`I'm sorry, can you try again?`);
  }

```

Теперь более подробно рассмотрим код webhook.

Здесь, у нас есть объявление о некоторых необходимых пакетах, которые нам нужно импортировать, и нам также необходимо импортировать пакет хранилища данных Datastore.

Поэтому здесь мы импортируем пакет Datastore.

И в строке 12 мы создадим новый экземпляр хранилища данных, привязав его к идентификатору нашего Google проекта.

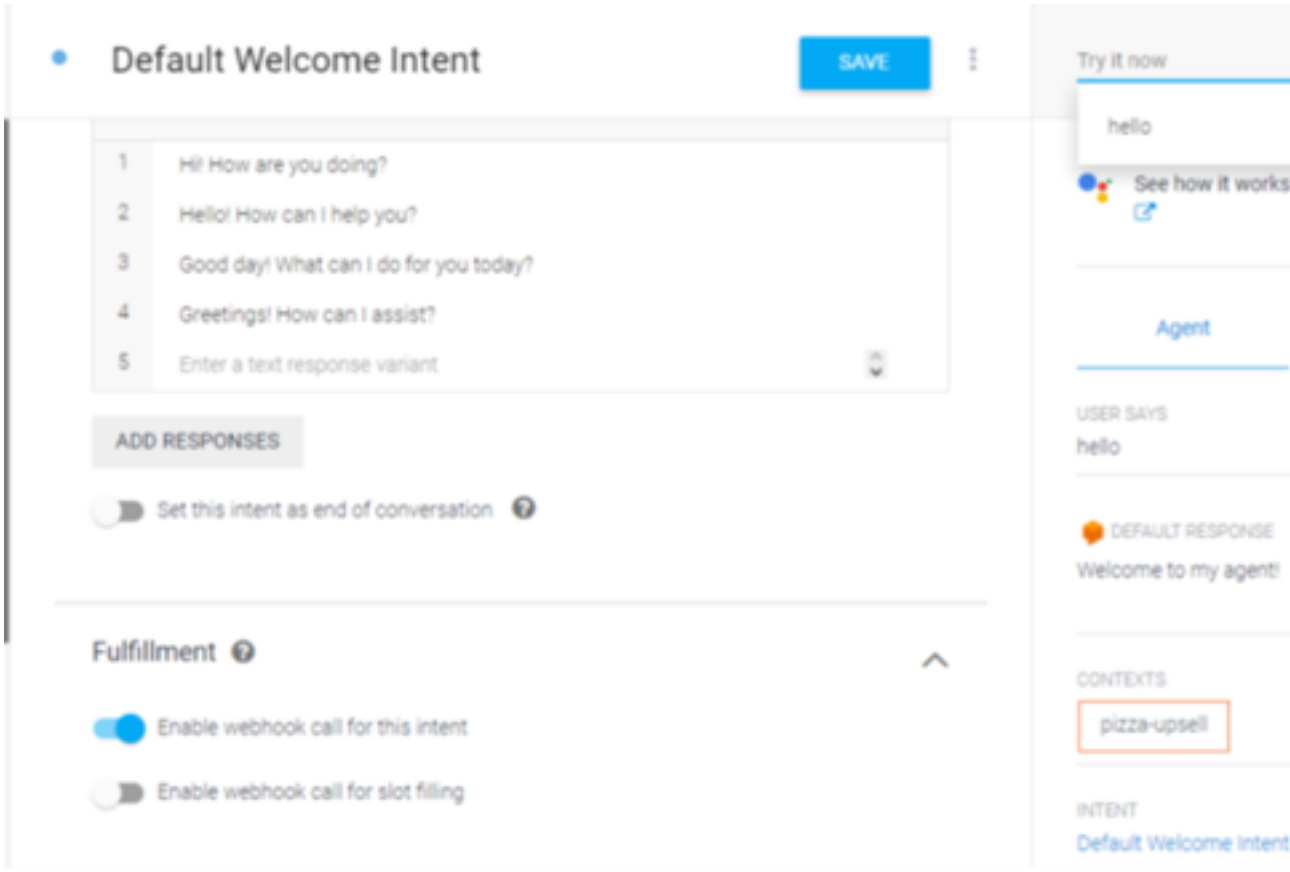
Идентификатор проекта можно посмотреть в консоли проектов по адресу, указанному на слайде.

```
exports.dialogflowFirebaseFulfillment= functions.https.onRequest((request, response) => {
const agent = new WebhookClient({ request, response });
console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

function welcome(agent) {
 agent.add(`Welcome to my agent!`);
}

function fallback(agent) {
 agent.add(`I didn't understand`);
 agent.add(`I'm sorry, can you try again?`);
}

                                        let intentMap = new Map();
                                        intentMap.set('Default Welcome Intent', welcome);
                                        intentMap.set('Default Fallback Intent', fallback);
                                        // intentMap.set('your intent name here', yourFunction
                                        // intentMap.set('your intent name here', googleAssista
                                        agent.handleRequest(intentMap);
                                       });
```

И здесь у нас есть основная функция dialogflowFirebaseFulfillment, где у нас есть функция для приветствия агента, и у нас есть функция для агента, который ничего не понимает.

Но у нас нет функции для заказа пиццы, и это то, что мы собираемся сюда добавить.

И далее, как только вы создали функцию, вам нужно сопоставить намерение с выполнением этой функции, с помощью добавления записи в карту намерений Map.



И если мы включим, например, Fulfillment в намерении приветствия, тогда если мы наберем в Try it – hello, чат-бот ответит не фразой намерения, а функцией приветствия агента, которая определена в вебхук.

Теперь, давайте создадим функцию для заказа пиццы.

```
function order_pizza(agent){
  var pizza_size = agent.contexts[0].parameters.size;
  var pizza_topping = agent.contexts[0].parameters.pizza_topping;
  var time = agent.contexts[0].parameters.time;

  const taskKey = datastore.key('order_item');
  const entity = {
   key:taskKey,
    data:{
     item_name:'pizza',
     topping:pizza_topping,
     time:time,
     order_time:new Date().toLocaleString(),
     size:pizza_size
    };

    return datastore.save(entity).then(()=>{
     agent.add(`Your order for ${entity.data.topping} pizza has been placed!`);
    });

}                                                    intentMap.set ('order.pizza.upsell.drink-no
```

Здесь у нас есть функция order pizza, которая определяет переменные для извлечения параметров из пользовательского запроса.

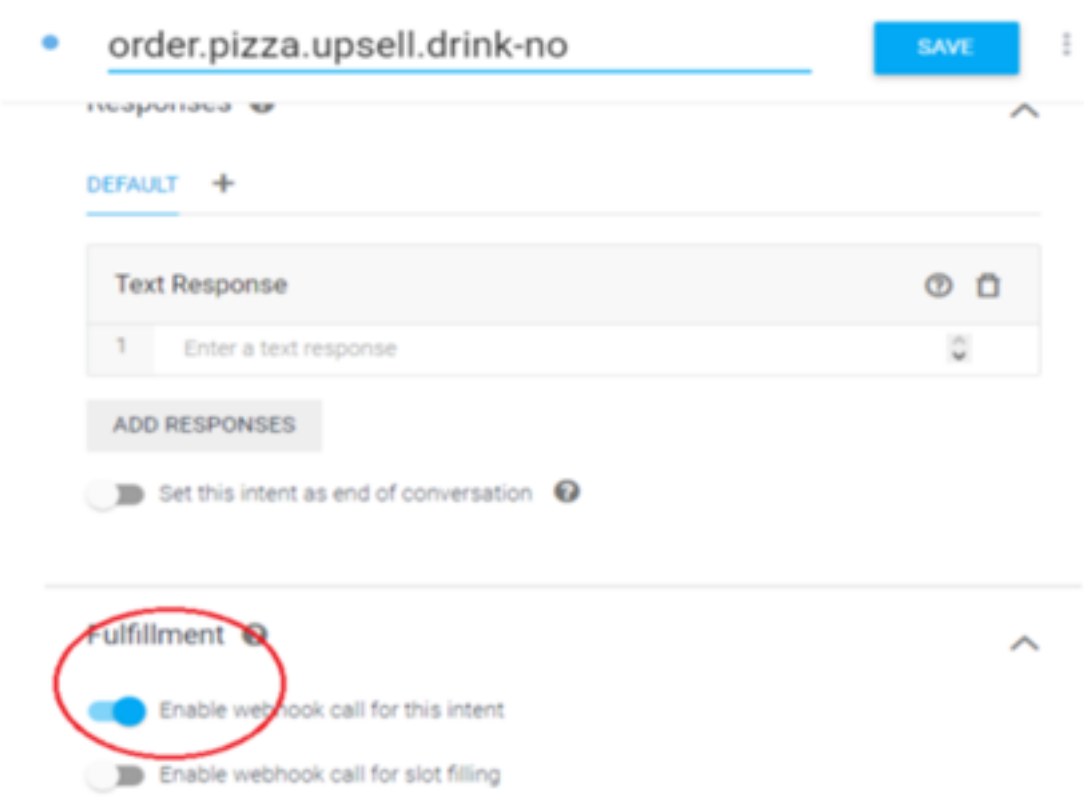
Она создает ключ для хранения в базе данных, а затем создает новую сущность.

Эта сущность будет содержать значения переменных.

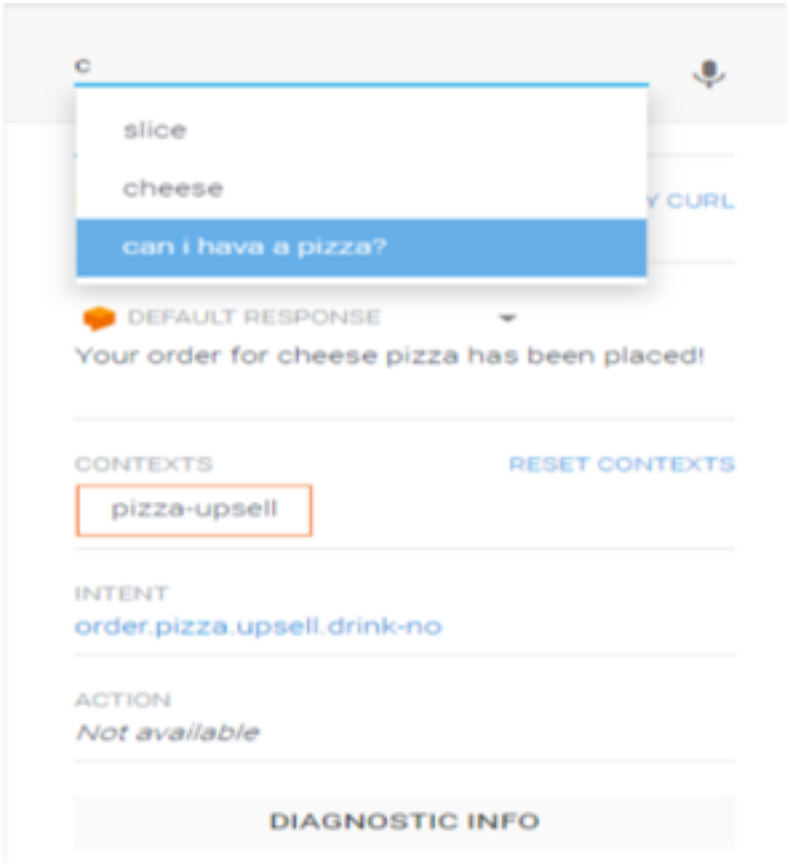
Возврат этой функции – это действие сохранения содержимого созданной нами сущности в Datastore.

И в конце, мы должны добавить запись в карту намерений.

После этого развернем заново наш вебхук.



И теперь нужно включить Fulfillment для намерения order.pizza.upsell.drink-no, чтобы после того, как клиент отказался от напитка, мы сохранили наш заказ в базе данных.



Теперь все готово к работе и в панели Try it наберем

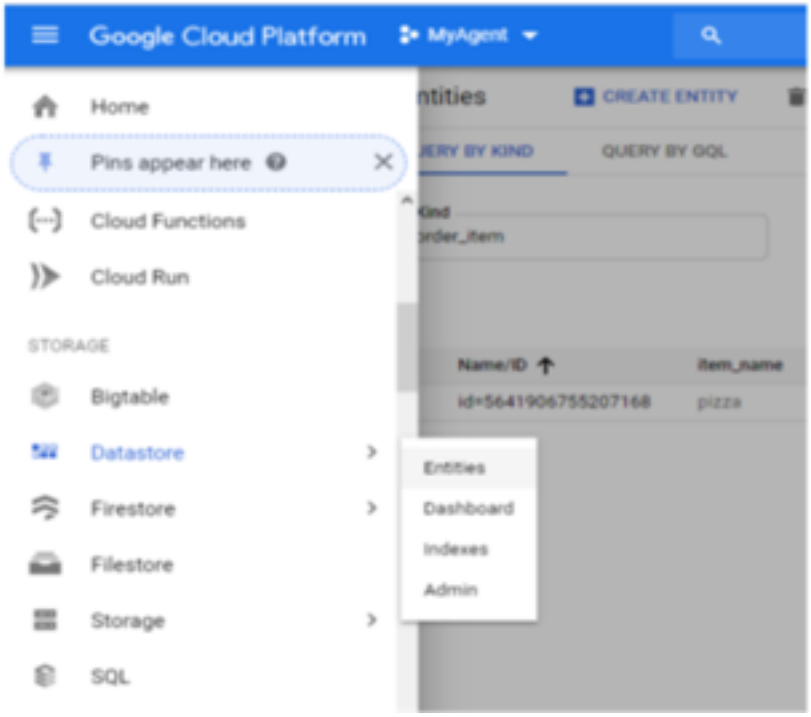
Могу ли я получить пиццу.

Затем ответим на вопрос о начинке и на вопрос о размере.

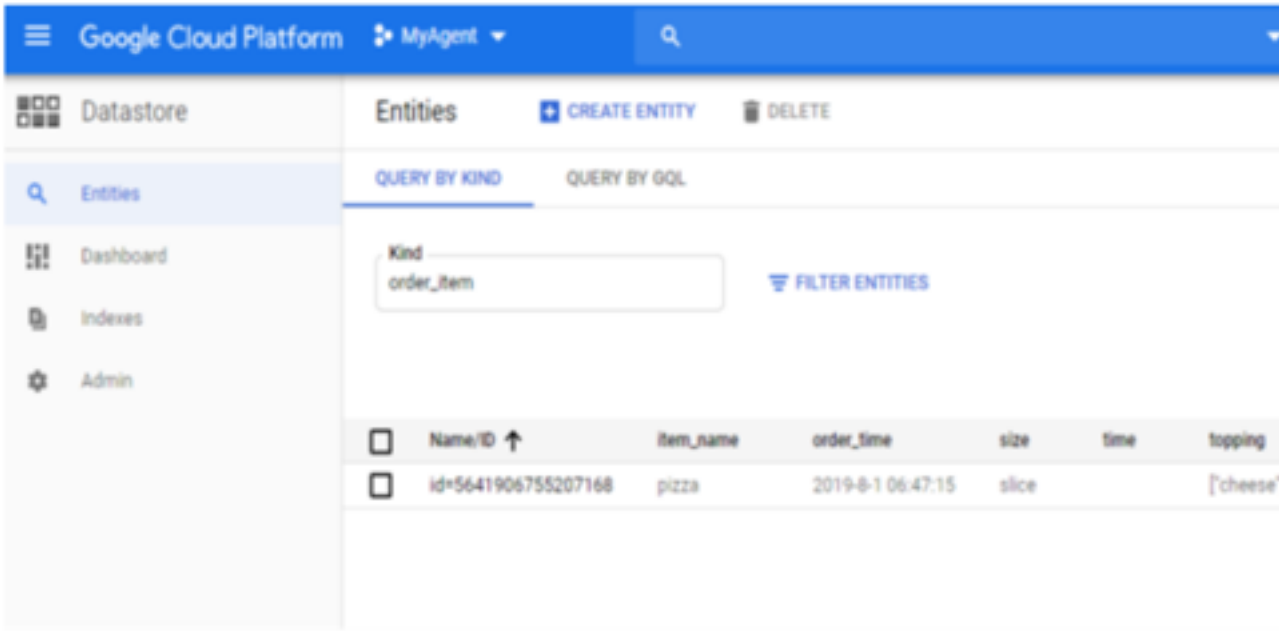
В результате получим ответ от агента, что наш заказ размещен.

Нажав на кнопку Diagnostic info можно посмотреть запросы и ответы вебхука в формате Json.

https://console.cloud.google.com/datastore/entities



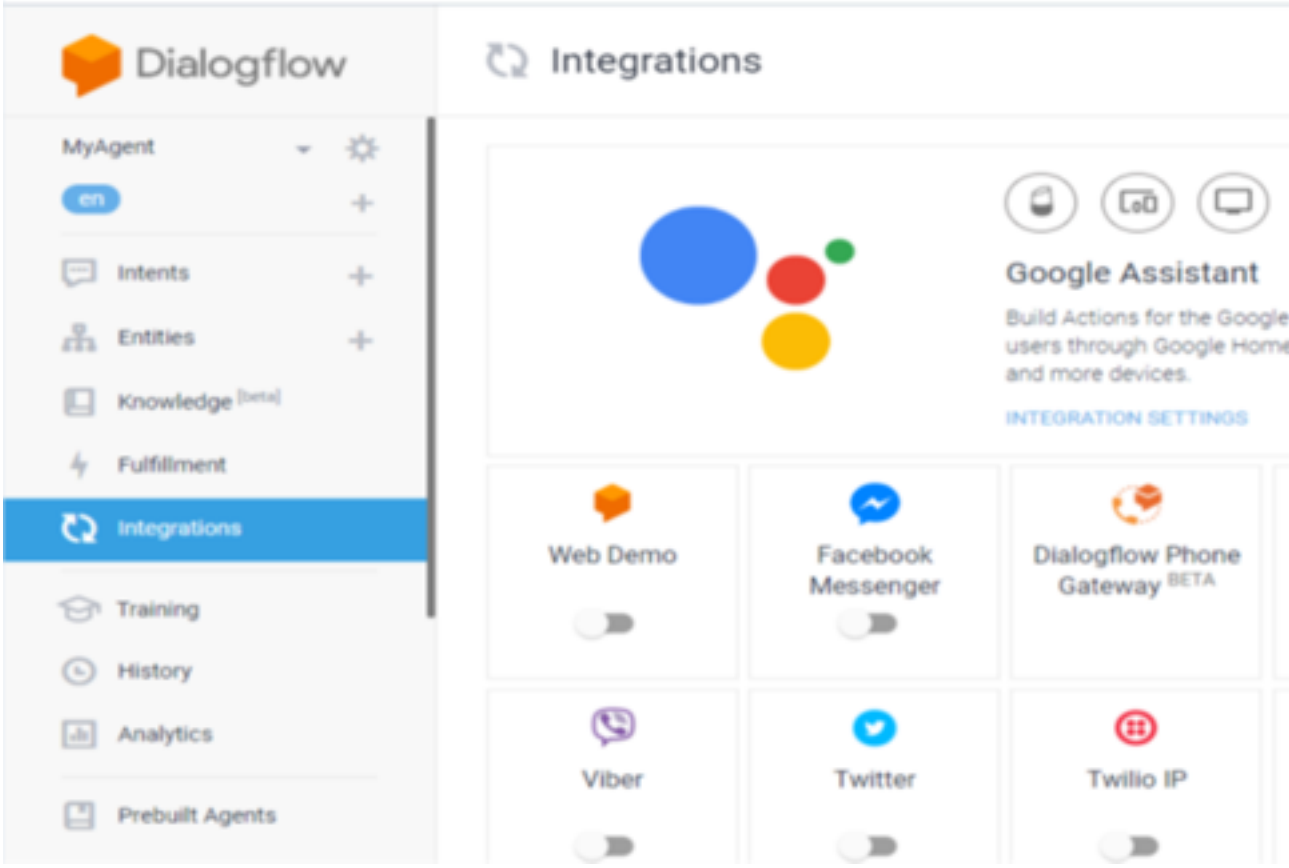
Чтобы проверить, сохранился ли заказ, откроем Google проект и в боковой панели выберем Datastore – Entites.



И здесь мы увидим, что наш заказ успешно сохранился в облаке Google.

Google Dialogflow. Интеграция с Telegram

Dialogflow позволяет интегрировать вашего чат-бота с различными платформами.

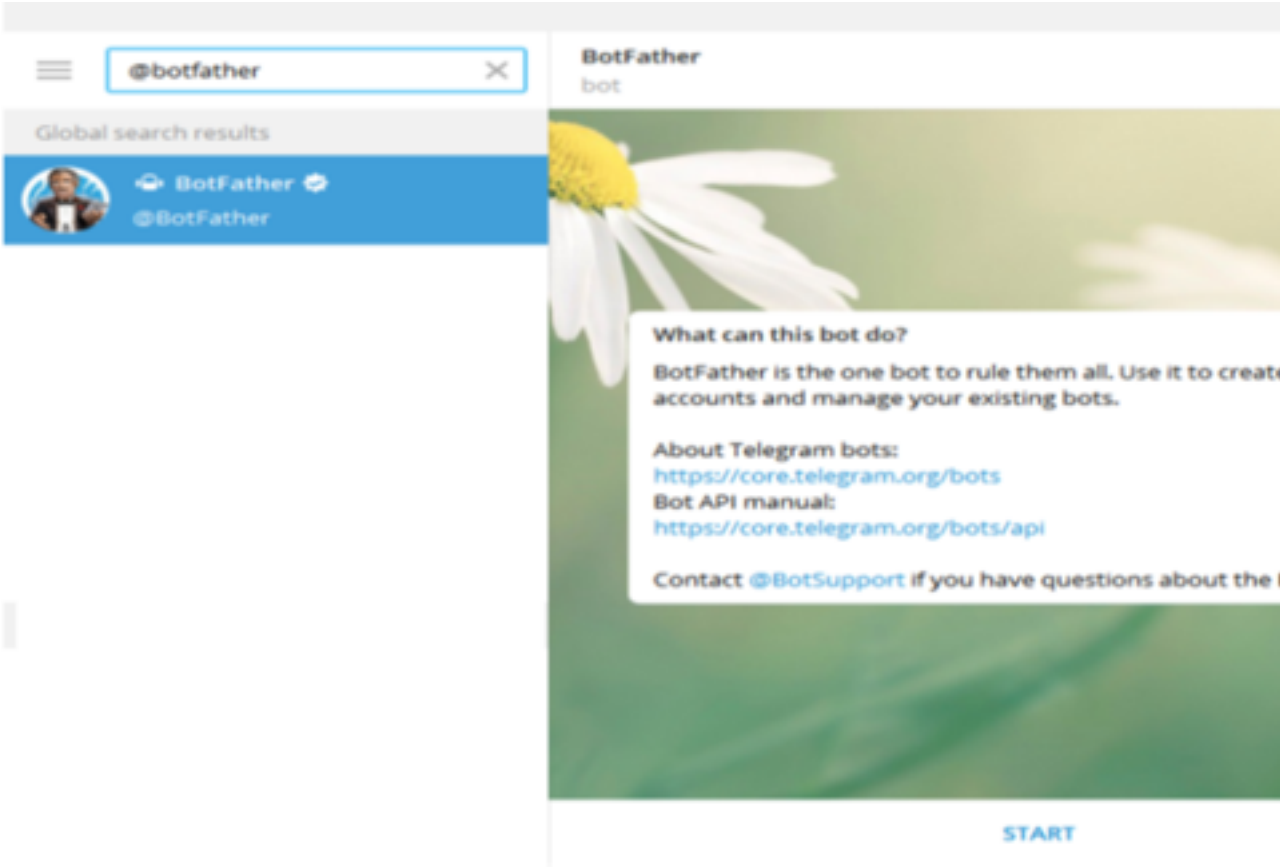


Это такие популярные приложения как Google Assistant, Slack и Facebook Messenger и другие.

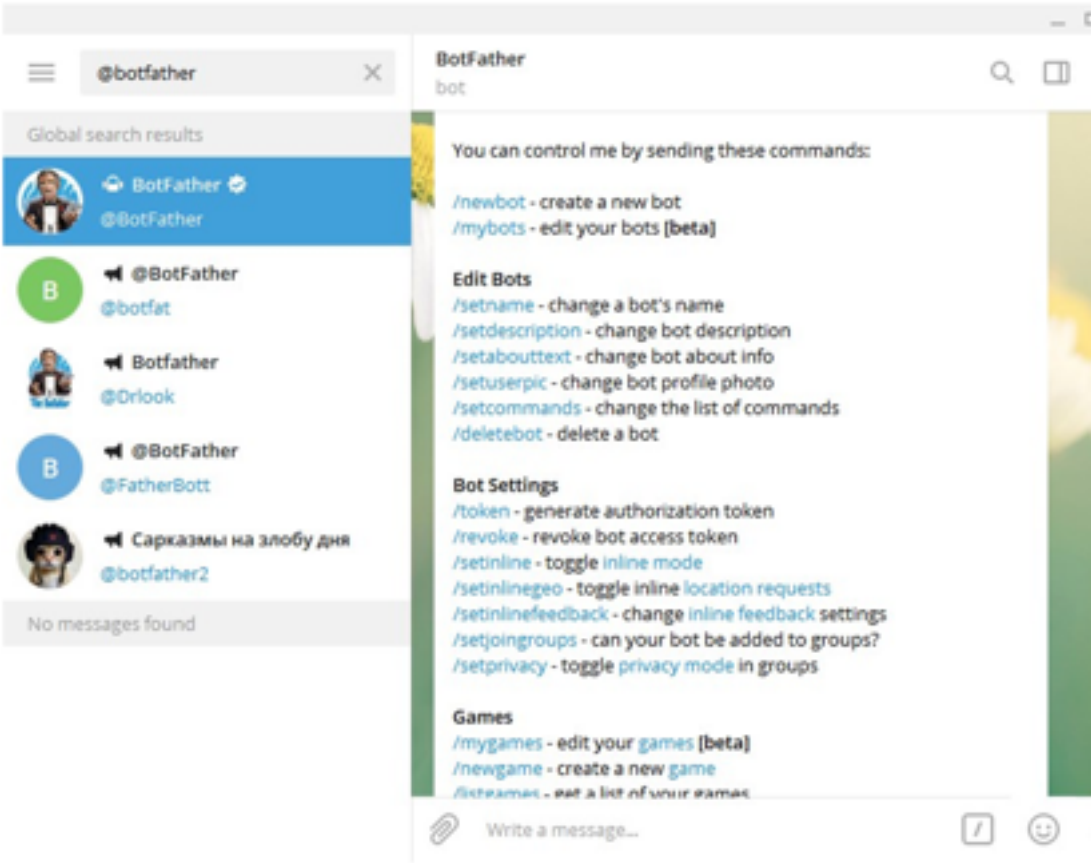
В качестве примера мы рассмотрим интеграцию нашего чат-бота с мессенджером Telegram.

Опция интеграция Telegram позволяет легко создавать ботов Telegram с пониманием естественного языка на основе технологии Dialogflow.

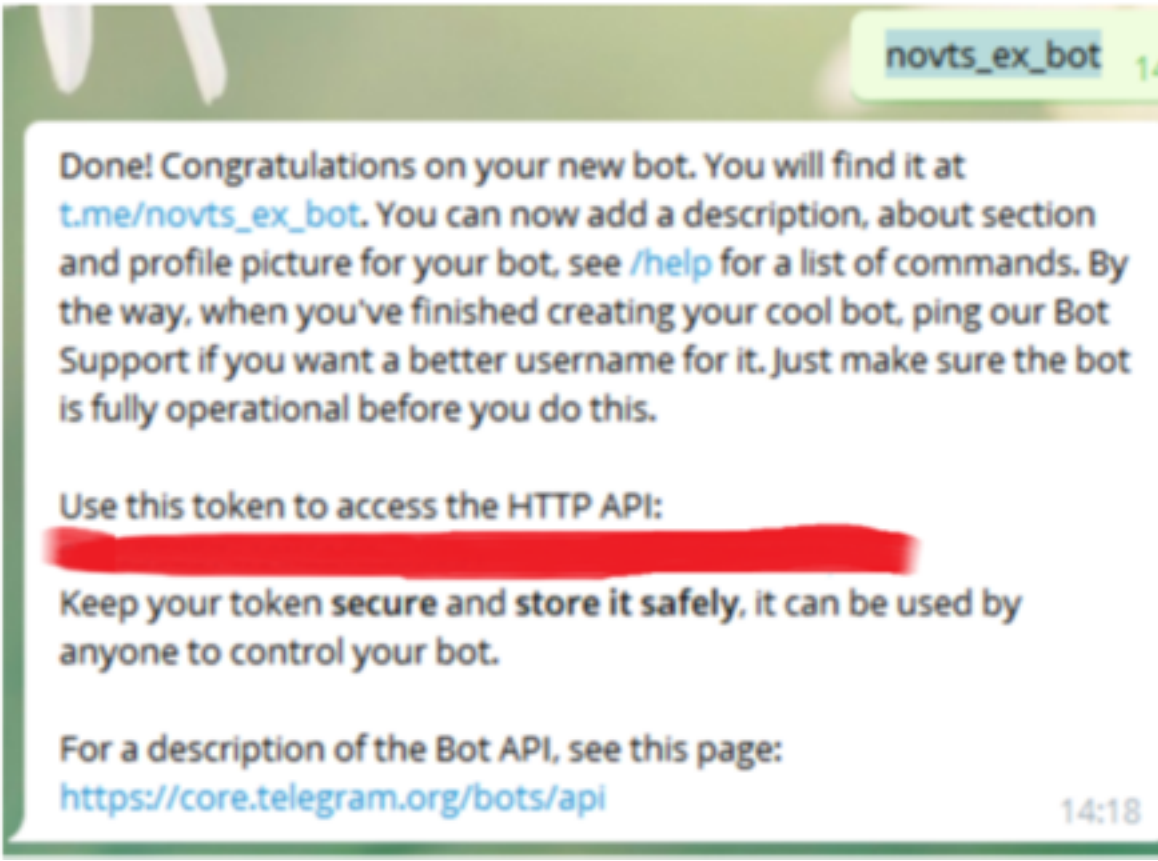
И для начала работы, откроем Telegram.



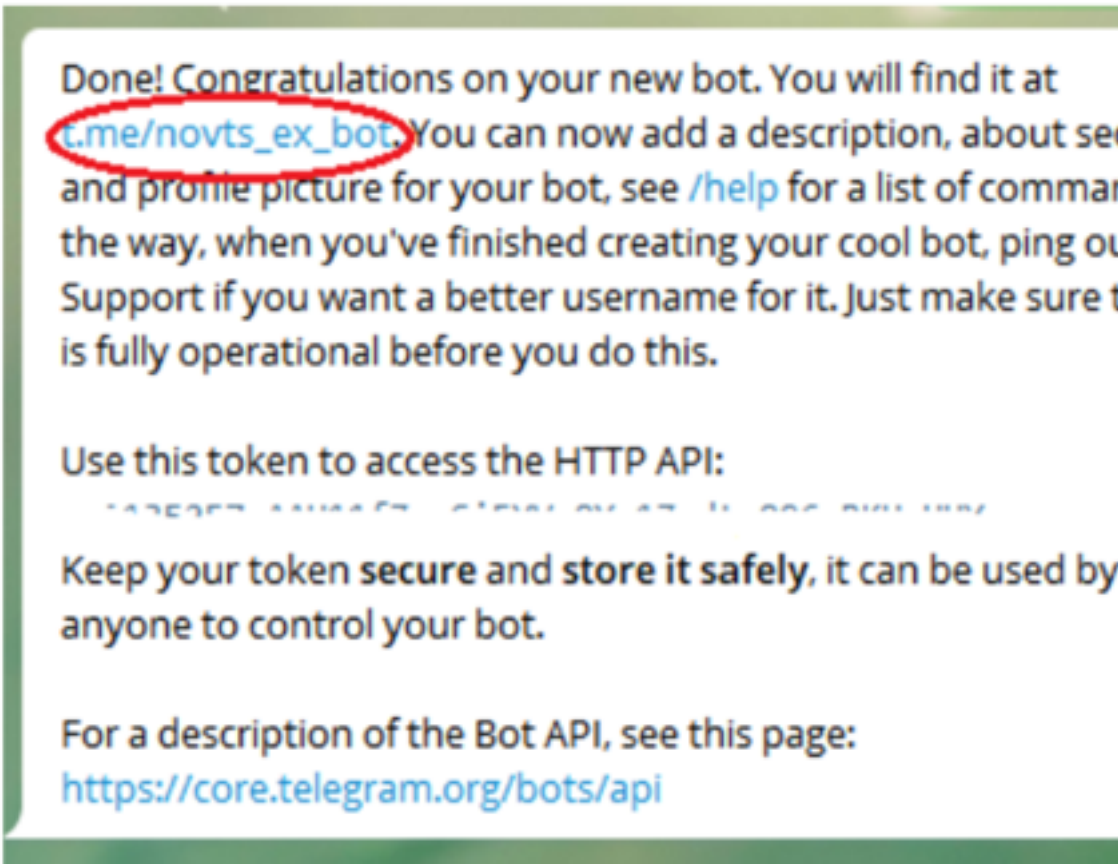
И здесь наберем @BotFather.



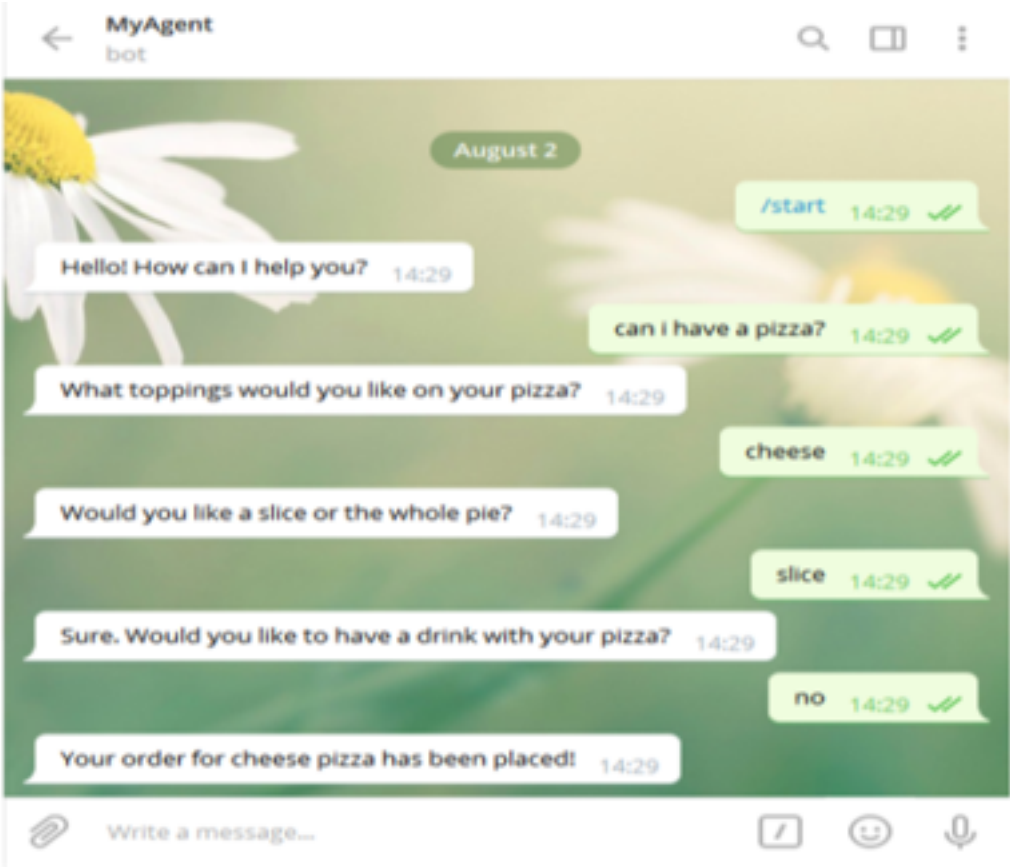
Далее нажмем кнопку Start.



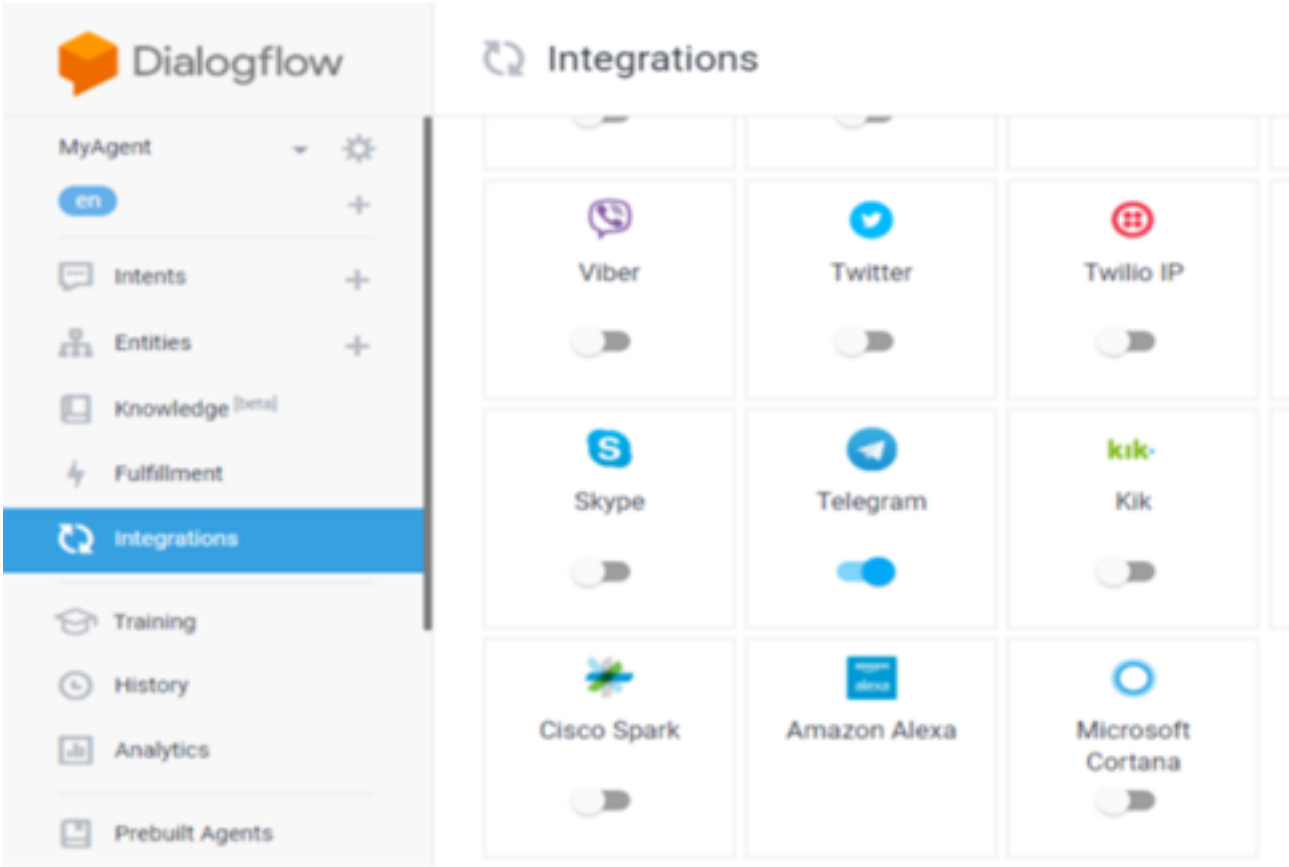
Здесь нажмем ссылку /newbot и введем имя бота ex_bot.



И здесь мы должны скопировать сгенерированный токен доступа.

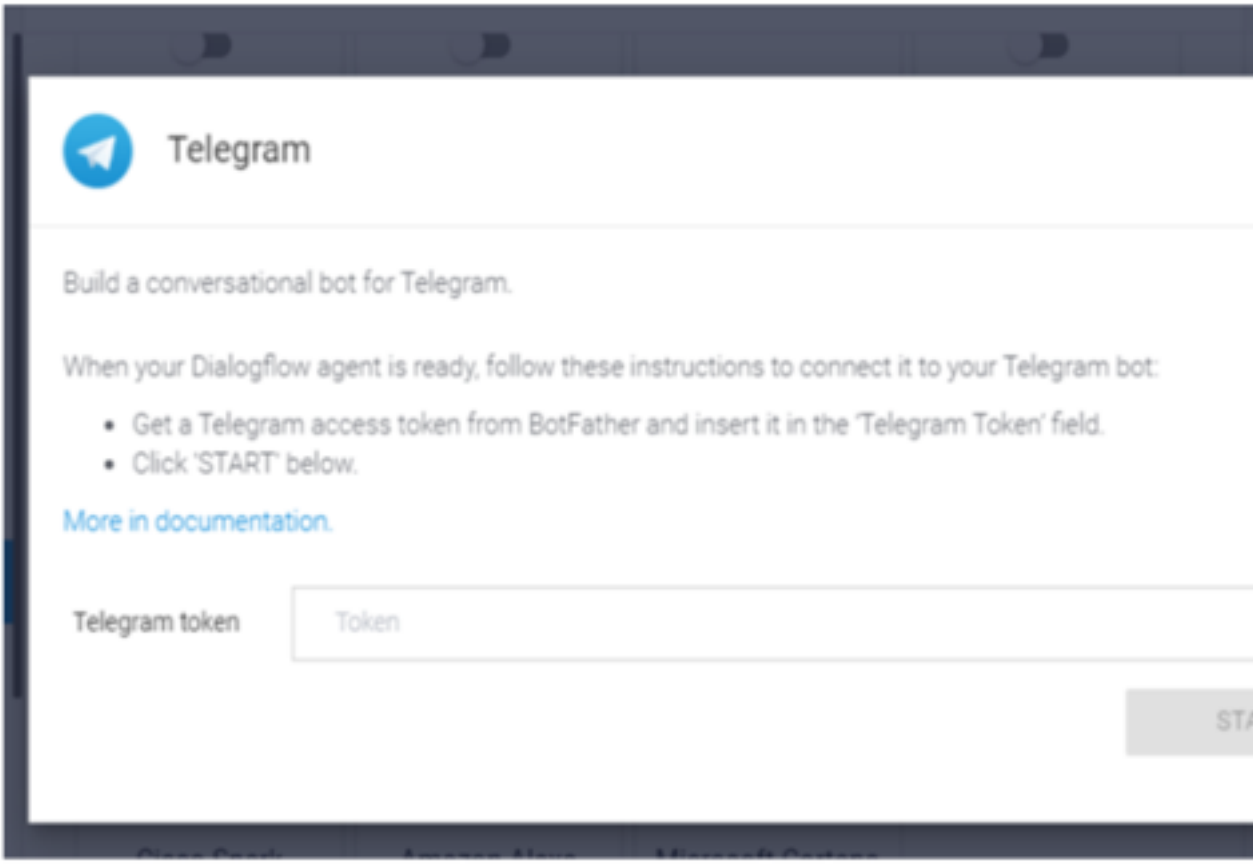


Вернемся в Dialogflow и включим интеграцию с Telegram.
В результате откроется диалоговое окно.



И здесь мы должны ввести сгенерированный токен доступа.

И нажать кнопку Start.



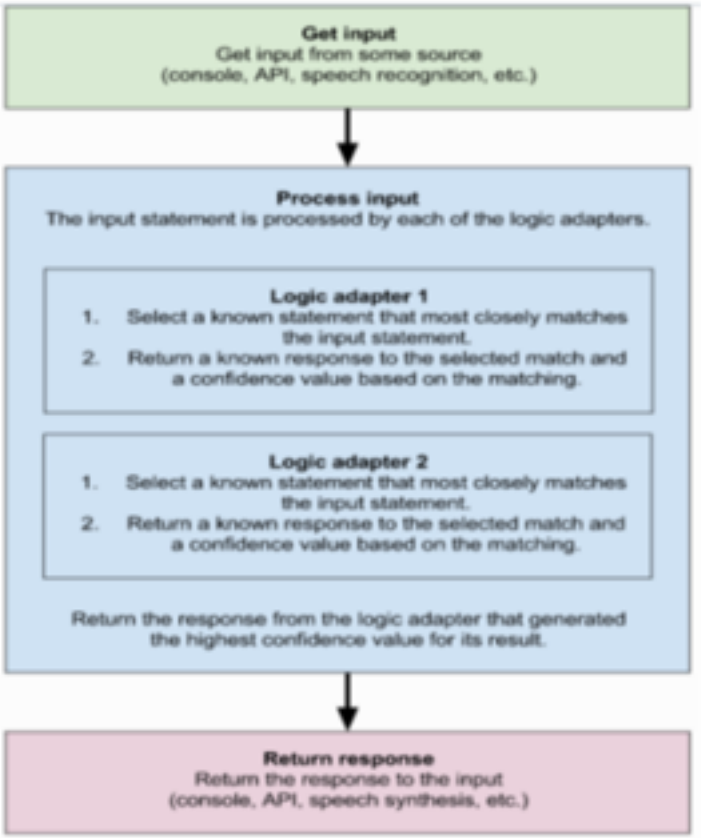
немся в Telegram и откроем бота по его ссылке.

И здесь, нажав кнопку Start мы можем разговаривать с нашим чат-ботом.

ChatterBot

ChatterBot – это библиотека Python, которая позволяет легко генерировать автоматические ответы на вводимые пользователем данные.

И ChatterBot использует набор алгоритмов машинного обучения для получения различных типов ответов.



И ChatterBot является независимой от языка библиотекой, что позволяет обучать чат-бота говорить на любом языке.

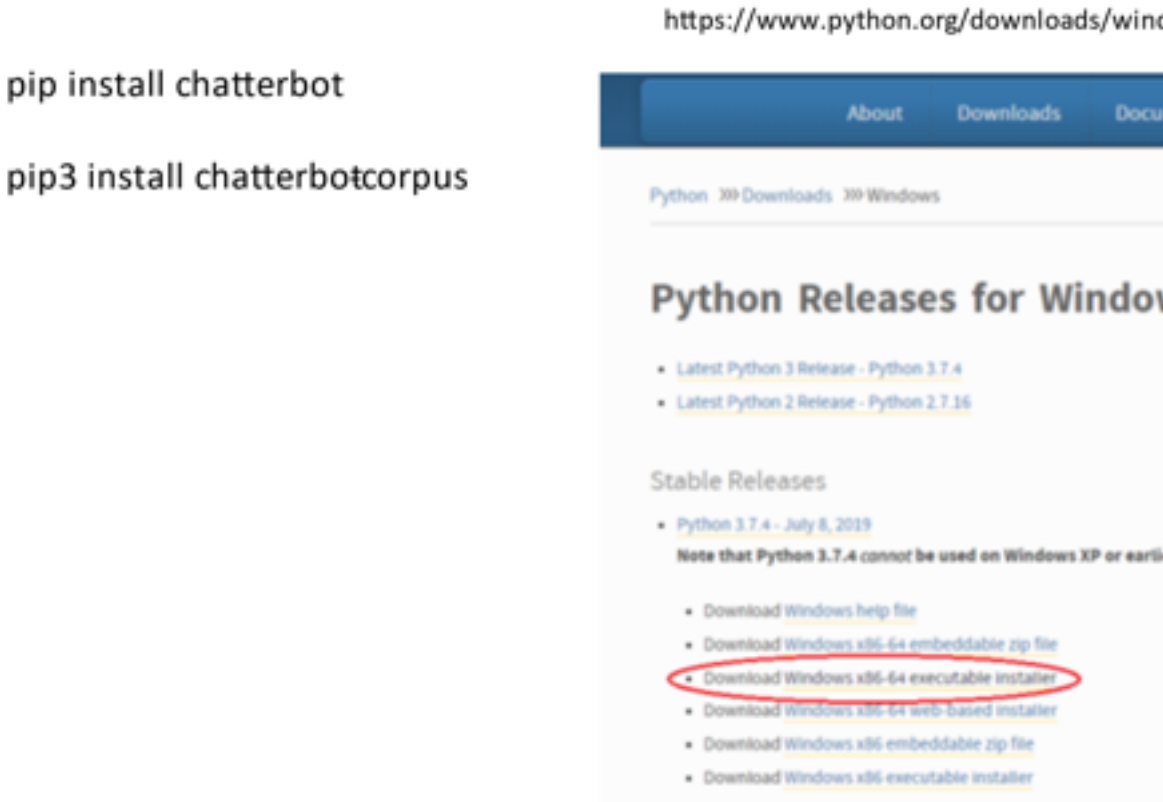
Кроме того, машинное обучение ChatterBot позволяет экземпляру агента улучшить свои знания о возможных ответах при дальнейшем взаимодействии с людьми и другими источниками данных.

Изначально, необученный экземпляр ChatterBot запускается без знания того, как общаться.

Каждый раз, когда пользователь вводит фразу, библиотека сохраняет введенный текст и текст ответа.

По мере того, как ChatterBot получает больше входных данных, количество ответов, которыми он может ответить, и точность каждого ответа по отношению к вводу пользователя увеличивается.

Программа выбирает наиболее подходящий ответ, выполняя поиск наиболее подходящего ответа, который соответствует вводу.



Для начала работы с ChatterBot, необходимо установить библиотеку с помощью инструмента pip.

И у вас должен быть установлен питон 64 битный, а не 32 битный.

New Slide

Прежде всего, ChatterBot должен быть импортирован.

И здесь мы импортируем класс ChatBot из библиотеки chatterbot.

И мы создаем новый экземпляр класса ChatBot.

Библиотека ChatterBot поставляется со встроенными классами адаптеров, которые позволяют подключаться к различным типам баз данных.

И класс адаптера, и путь к базе данных указываются как параметры конструктора класса ChatBot.

Класс SQLStorageAdapter является адаптером ChatterBot по умолчанию.

Если вы не укажете адаптер в конструкторе, адаптер SQLStorageAdapter будет использоваться автоматически.

И класс SQLStorageAdapter позволяет чат-боту подключаться к базам данных SQL.

По умолчанию этот адаптер создает базу данных SQLite.

Библиотека ChatterBot включает в себя инструменты, которые помогают упростить процесс обучения экземпляра чат-бота.

Обучение ChatterBot включает загрузку примера диалога в базу данных чат-бота.

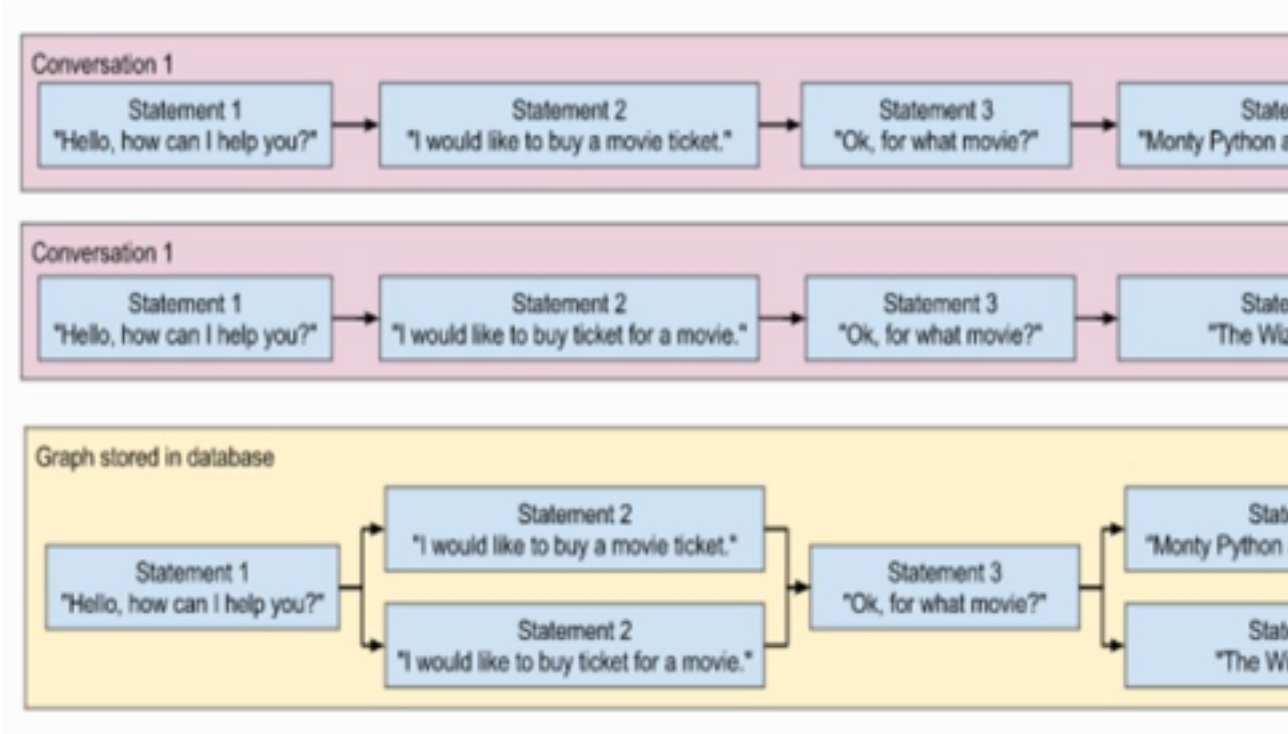
```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer

chatbot = ChatBot("MyAgent")

conversation = [
"Hello! How can I help you?",
 "hello"
]

trainer = ListTrainer(chatbot)
trainer.train(conversation)

trainer.train([
   "Hi there!",
   "Good day! What can I do for you today?",
])

trainer.train([
   "hey",
   "Greetings! How can I assist?",
])
```

```
while True:
   try:
      response = chatbot.get_response(input
      print(response)

   except(KeyboardInterrupt, EOFError, Syst
      break
```

При этом строится граф, который представляет наборы известных вводов и ответов.

Когда тренеру чат-бота предоставляется набор данных, он создает необходимые записи в графе знаний чат-бота.

И библиотека ChatterBot поставляется со встроенными классами тренеров, или вы можете создать свой собственный класс тренера, если это необходимо.



Чтобы использовать класс тренера, вы вызываете метод train для экземпляра тренера, который был инициализирован вашим чат-ботом.

Класс тренера ListTrainer позволяет обучить чат-бота, используя список строк, где список представляет собой разговор пользователя с чат-ботом.

И для процесса обучения вам нужно передать список фраз этого разговора.

Каждый такой список будут представлять отдельный разговор.

После обучения мы создаем цикл while для чат-бота.

Конец ознакомительного фрагмента.