**ChatBot Complete Market Survey**

With the Advancements in technology, many platforms for ChatBot came into picture based on the need in a particular domain. To create a chatbot, there is currently an incredible amount of **platforms** and **tools**, with different complexity levels, expressive powers and integration capabilities. Let’s suppose you want to develop a chatbot. The million-dollar question is: **among all the existing platforms, which one fits my needs the best?**

**The answer is simple; Depending on the use case and domain required to be implemented some platforms are more appropriate than others.**

**General ChatBot Architecture:**

* Understanding what user says:

First, chatbot needs to understand the input from the user. This involves two techniques:

1. Pattern Matching
2. Intent Classification

**Pattern Matching** : Giving input patterns manually , easily readable by humans. Drawback is that this cannot be applied to many real time use cases.

**Intent Classification :** An intent classification approach relies upon machine learning techniques. You need a set of examples to train a **classifier** that will choose, given a user input, among all the possible intents (e.g. buy a ticket, check flight status, get specific information, etc.).

Classifying the input and extracting information from it are two key concepts that you have to keep in mind.

* Responding to the user

Once the chatbot understands what the user says, it can choose or generate a response, based on the current input and the context of the conversation. There are different type of responses like,

1. **Static Responses:**

The simplest way is to have a static response, with eventually a list of variants, for each user input. These static responses could be templates, such as The flight time is <ft> hours, where <ft> is a variable computed on the fly by the chatbot.

1. **Dynamic Responses:**

A different approach would be to use resources, such as a knowledge base, to get a list of potential responses, and then score them to choose the better response. This is particularly appropriate if you chatbot acts mainly like a question-answering system.

1. **Generated Responses:**

If you have a huge corpus of examples of conversations, you could use a [**deep learning technique**](https://arxiv.org/abs/1409.3215)to train a generative model that, given an input, will generate the answer. You will need millions of examples to reach a decent quality and sometimes the results are going to be unexpected, but it could be interesting and fun to test the approach and see what happens.

The important thing to be noted is that **Do not forget the context of the application** i.e some questions require the knowledge of previously asked questions to answer. In that case, each chatbot platform should have its own notion of context to analyze and store the necessary information.

**Existing Platforms:**

Before you can choose a platform, you must know what kind of chatbot you are trying to build. Is it a **goal-oriented**, **conversational or goal-oriented** with strong conversational abilities chatbot?

A **goal-oriented** or **transactional** chatbot is the most frequent kind of chatbot for business. It helps users achieve tasks such as buying a ticket, ordering food or getting specific information.

A **conversational** chatbot is focused on having a conversation with the user. It does not need to deeply understand what the user says and does not have to remember all the context of the conversation, it just need to emulate a conversation.

We can distinguish three families among the existing platforms:

* No programming platforms.
* Conversation-oriented platforms.
* Platforms backed by tech giants.

**No Programming Platforms:**

They are non-technical user oriented platforms. It is usually easy to code a chatbot without having programming skills and without having machine learning or natural language processing expertise. The key idea is that the user does not have to worry about the technical details.

Some of the platforms are **Chatfuel, Motion.ai, Manychat, Octane.ai, Massively.** They are all task-oriented.

**Pros:**

* You can develop chatbot very quickly.
* Very low learning curve.
* Ideal for simple bots.

**Cons:**

* There are a lot of platforms, with different levels of maturity and stability.
* Sometimes the GUIs are not so easy to understand and when the chatbot logic gets more complex, it becomes hard to handle.
* They have little or no natural language processing capabilities. For example, some platforms cannot perform information extraction. Therefore, given a phrase such as “I’m in Boston” they cannot extract the fact that the city of Boston (location entity) occurs.
* They do not seem appropriate for complex bots.

#### Conclusion:

From our point of view, the no programming platforms lack of power for large scale commercial projects. The conversations cannot be very complex and usually it is not possible to integrate external resources, such as NLP and ML specific components.

However, they are really good platforms for small scale projects, typically to quickly add a chatbot functionality to a Facebook page, for example.

**Conversational Platforms:**

The main goal is to have conversation with the bot without task-oriented scenario. These platforms typically use AIML to model the interactions.



The best example for this kind of platforms is [pandorabots.](https://playground.pandorabots.com/en/quickstart/)

**Pros:**

* AIML is a standard.
* It is very flexible to create conversations.

**Cons:**

* It could be difficult to scale if patterns are manually built.
* The information extraction capabilities are limited.
* They are not really appropriate for task oriented bots.

#### Conclusion

You would not use these platforms to build a chatbot for ordering food or buying tickets, but you could find that they are very interesting to quickly model an entertainment chatbot or, for example, a chatbot that replaces a FAQ and gives a better user experience.

### Platforms backed by tech giants

These platforms are developed by tech giants companies and, somehow, they represent already a standard or at least they are on its way to become one:

* [**Api.ai**](https://api.ai/) (Google)
* [**Wit.ai**](https://wit.ai/) (Facebook)
* [**LUIS**](https://www.luis.ai/) (Microsoft)
* [**Watson**](https://www.ibm.com/watson/) (IBM)
* [**Lex**](https://aws.amazon.com/lex/) (Amazon)

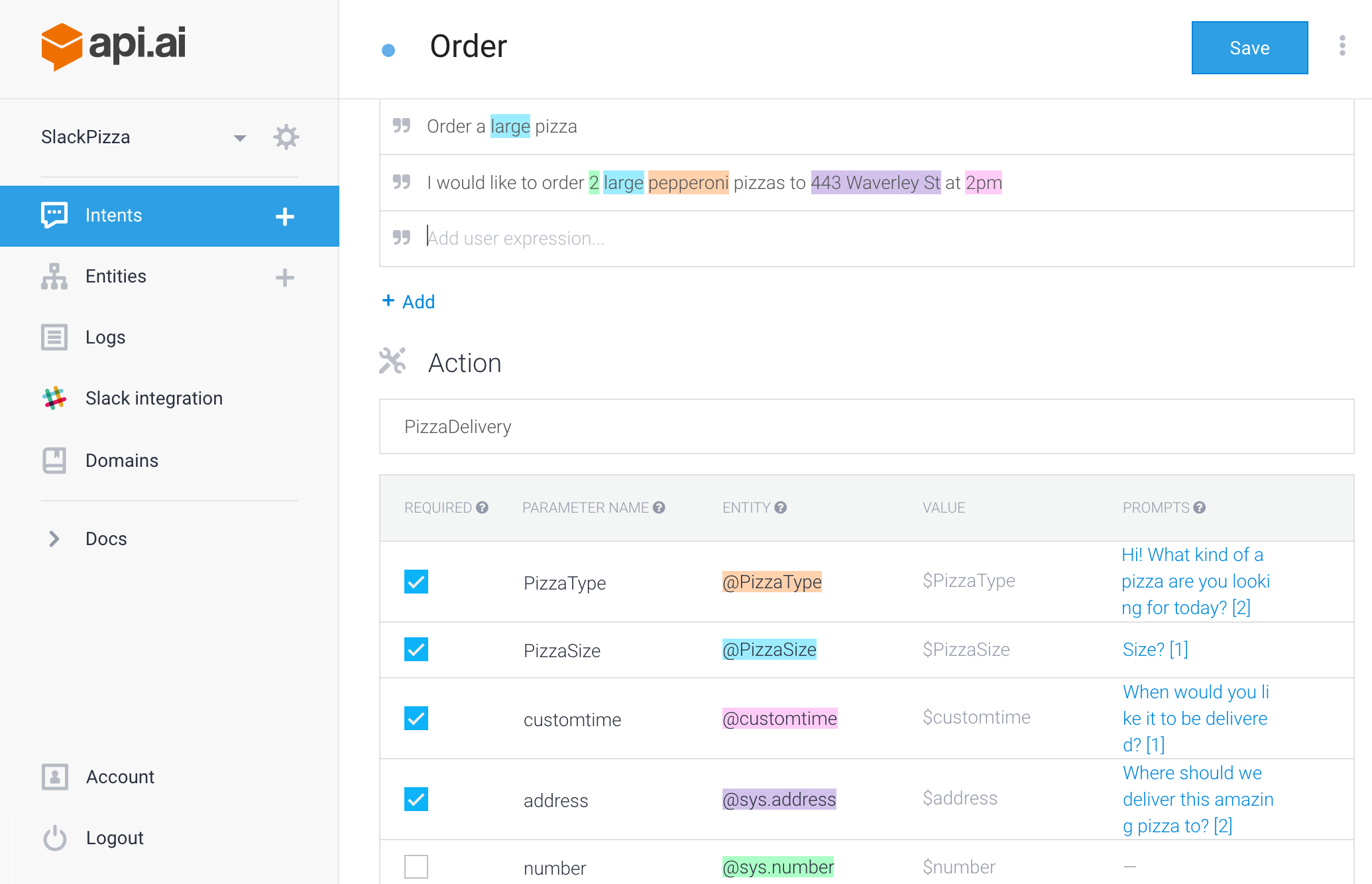
They try to have a low learning curve and, at the same time, a strong expressive power. When you model a chatbot, one of the hardest thing is to model the conversational flow.

**Api.ai:**

**Chatbot Behavior:**

Intents and Contexts are the key concepts to model the behavior of a chatbot with Api.ai. Intents creates links between what a user says and what action should be taken by the bot. Contexts are string values, useful for differentiating requests which might have different meaning depending on previous requests.

Basically, when Api.ai receives a user request, it is first classified to determine if it matches a known intent. Api.ai proposes a “Default Fallback intent” to deal with requests that do not match any user intent.



You can restrict the matching of an intent by specifying a list of contexts that mustbe active. At the same time, the matching of an intent can create and delete contexts. However, you cannot model that intent can be matched only if a certain context is not present. This is the limitation with api.ai.

#### Slot-filling capabilities

This is a key point of Api.ai, that brings at the same time flexibility and power. Slot-filling allows you to indicate, for a given intent, what are the fields that play a role and if they are mandatory or not.

This is great since you do not have to deal with missing information since it is done on the Api.ai side. In the example above, Api.ai will ask for each mandatory field until they are filled in by the user: pizza type and size, addres and time of delivery. As you can see, the field “number” may be part of the intent but it is not mandatory.

#### Pros

* Api.ai proposes a powerful way of modeling large and complex flows using Intents and Contexts.
* Slot-filling is an integrated feature. Consequently, a good part of the logic can be solved by the chatbot, which decreases the server side coding.
* Domains are available, that is specifications that can deal with several common use cases and applications (e.g. small talk, wisdom, flight schedules, reminders…).
* A section “Training” (in beta) is proposed to train the chatbot with examples.
* One-click integration with several platforms: Facebook Messenger, Slack, Twitter, Telegram…

#### Cons

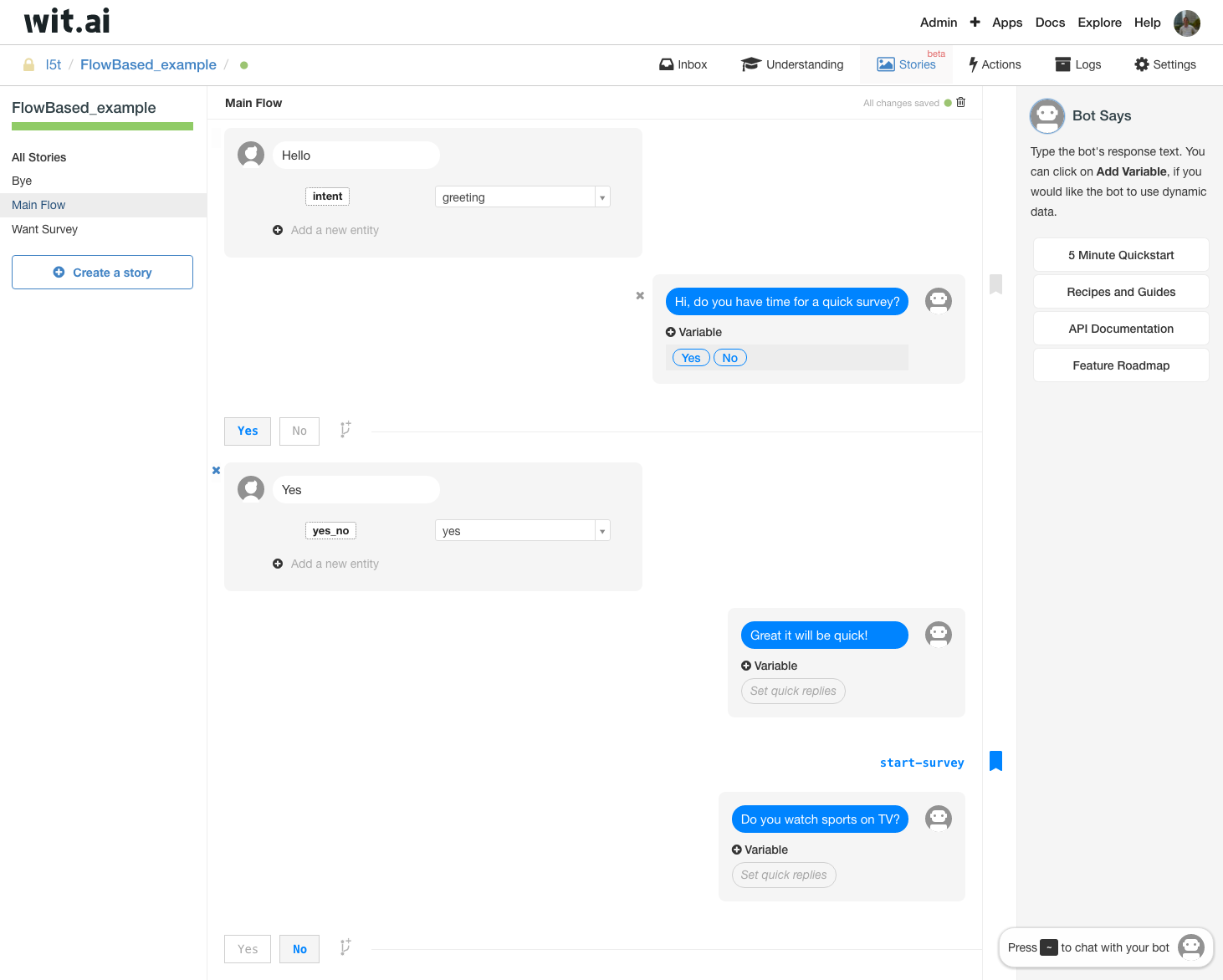
* It is impossible to block the matching of an intent if a context is present.
* The training section is still in beta.

**Wit.ai:**

**Chatbot Behavior:**

Stories are the key concept to model the behavior of a chatbot with **Wit.ai**. Each story represents an example of a possible conversation. It should be noted that “intent” is no longer a concept but a user entity, non mandatory. This was a change of greater impact in Wit.ai, motivated by the fact that a complex chatbot needs a lot of intents that can, in some way, be grouped in stories.

Bot developers basically teach Wit.ai by example. The subjacent idea is that when a user writes “similar” requests, Wit.ai will process the request, extract the entities and apply the logic defined by the developer.



A story can be seen as a graph of user intents. You can add branches that are triggered on conditions such as the existence or not of specific variable values, that are extracted from the user input. This allows you to define a conversation flow. Moreover, you have a *bookmark* mechanism, used to jump between intents and also between stories.

To interact with the server side, you have “Bot sends” commands, that are basically calls to functions. A very interesting point is that you can set the role of the entities in a phrase. For example, in “I want to fly to Venice, Italy from Paris, France, on January 31”, you can state that the first city is the departure and the second one the destination.

### Server size coding

Wit.ai proposes a webhook integration: it passes information for each “Bot sends” command into a web service and gets a result from it. On the server side you are typically going to create or expand the context of the conversation. The result sent to Wit.ai can add, modify and delete context variables used on the chatbot side.

**Pros**

* The concept of story is powerful.
* Wit.ai allows controlling the conversation flow using branches and also conditions on actions (e.g. show this message only if some specific variables are defined).
* Assigning roles to entities helps server side processing.
* A section “Understanding” is proposed to train the chatbot with examples.
* An “Inbox” exists, where the requests that could not be processed by the chatbot are listed, so the developers can teach the bot.

**Cons**

* Stories are in beta.
* Even if stories are a powerful concept, there are cases where it is difficult to control the flow of the conversation and the bot tends to misunderstand the user requests.

**Current Limitations using NLP and ML**

As we have seen, to model a chatbot we need to provide the logic and the linguistic resources, mainly the input and output phrases and the entities. This is particularly true for Api.ai and Wit.ai. For small chatbots this should not be a problem, but if you are planning to deal with a big terminology and a lot of variants for phrases, you should consider using NLP and ML. We mention a few examples where they could be useful.

### Singular and plural forms

If you want your chatbot to extract “pizzas” as an entity, it is not enough to define “pizza”, you need to provide “pizzas” as well.

Api.ai has a feature called “automatic expansion” and Wit.ai has “free-text” entities. They are mechanisms that will try to catch new items, based on word context. So if you have trained your chatbot with phrases such as “I’d like to order a pizza”, it is likely that it will understand that in “I’d like to order 3 pizzas” the word “pizzas” is an entity. But the accuracy of this feature will depend on the training and you cannot be sure about how much noise it is going to bring.

A sure alternative is to provide, for each concept, singular and plural forms. You can generate them using NLP tools called inflectors.

### Synonyms, hypernyms and hyponyms

### NLP has a lot of resources to handle all these called thesaurus and ontologies. But it doesn’t contain words like coco-cola or pepsi which is a specific domain term. The problem arises when user asks for soda and bot couldn’t get related with pepsi or coco-cola.

### You could try to find an existing thesaurus that fits your problem or build it by your own. Resources built by domain experts are expensive but highly accurate. With [Machine Learning](https://arxiv.org/pdf/1411.4166.pdf) you can create linguistic resources, particularly with Deep Learning techniques, that could be good enough to your use case.

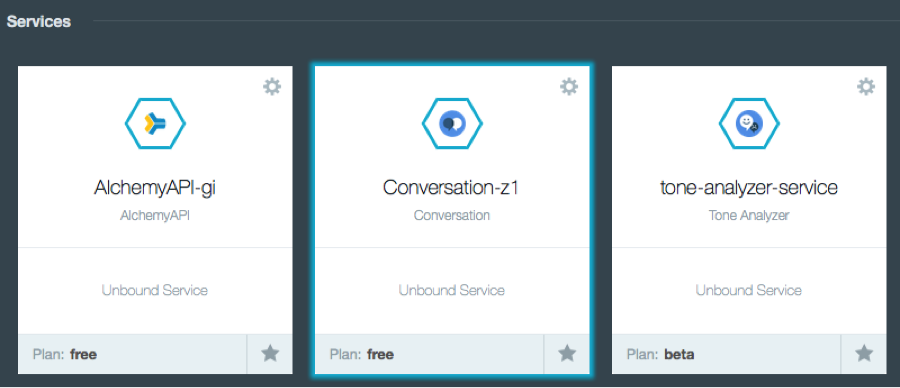
### Sentiment analysis

You could try to perform sentiment analysis on the server side to adapt the responses consequently which adds emotional reactivity to the bot.

However, it might not be an easy task if you are using Api.ai or Wit.ai. If you want a very flexible and rich chatbot, you should probably consider developing the chatbot from the scratch.

**IBM Watson Chatbot:**

To get started with IBM Watson services provided, sign in to [Bluemix](https://console.ng.bluemix.net/registration/?cm_mc_uid=12797211719714931151805&cm_mc_sid_50200000=1493291528&cm_mc_sid_52640000=1493291528); which is a free trail for 30 days. This gives access to the Watson api services.

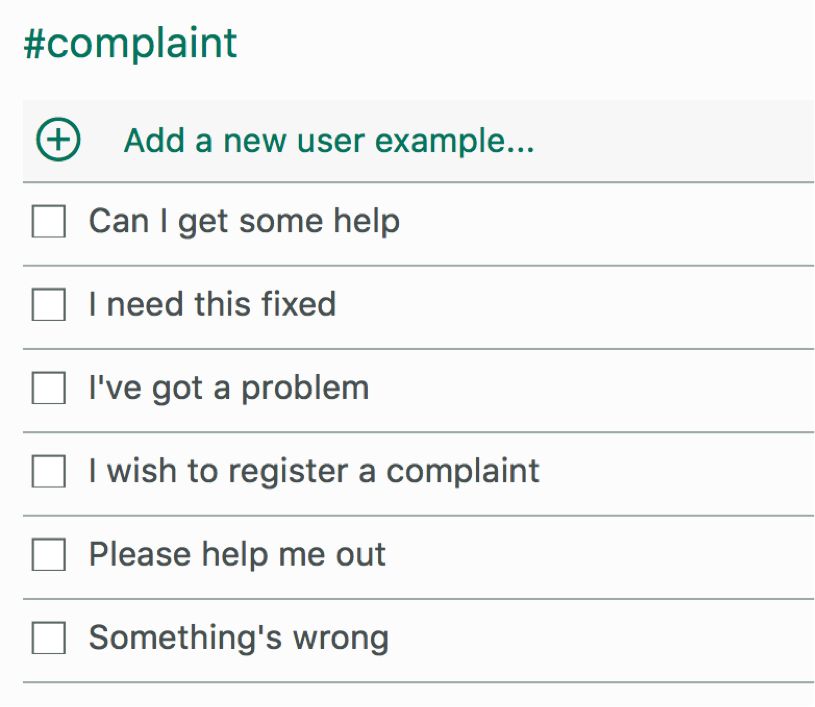


**Create a workspace and configure intents:**

Intents are things that the chat users are looking to do: change passwords, get status updates, make complaints, etc. What’s really neat about managing intents is that you are actually training the system to understand what is needed without depending on the exact words. You give multiple training examples that correspond to specific actions or requests.

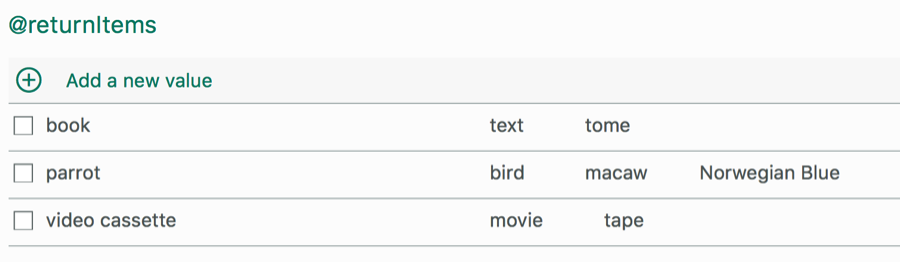
Creating new intents is easy. All you have to do to finish is add examples of user requests that fit the category.

For this bot, start with the basics: hello and goodbye. You’re going to train some intents to recognize opening and closing statements in a conversation. Here’s an example on how to recognize a complaint without actually providing the exact keyword.



**Preparing entities**

Once you’ve finished training your intents, move to the *Entities*. Entities are the specific pieces of information we want to extract from the user response. You’ll want to group entities that might trigger a similar response in your dialog. In this customer service example, we want to put together a list of potential items that the customer may wish to return. Here are my **@returnItems**:



#### Dialog flow

When your intents and entities are specified, you can move on to constructing the dialog flow. This is the most complicated piece of your bot, because conversations can go anywhere! You want to focus on providing answers for the tasks you can help with, and providing feedback when the conversation gets out of scope. The super simple customer service chat bot will be able to respond to greetings and goodbyes, as well as give direction when a customer has a complaint, including specific feedback when the customer is looking to return an item.

Once you are in the Dialog tab, click Start and you will see the first node. Type “#greetings” into the Enter a condition field, and “Hello!” into the Watson says field.

Listing Brief, IBM Watson has taken over other chatbot platforms efficiently using Artificial Intelligence:

* Breaks down the pros and cons of chatbots.
* Explains the different ways businesses can access, utilize, and distribute content via chatbots.
* Forecasts the potential impact chatbots could have for businesses.
* Looks at the potential barriers that could limit the growth, adoption, and use of chatbots.

**Conclusion:**

Currently, there are a plethora of platforms that can assist you when creating a chatbot. Some of these platforms have been built with different use cases in mind, so it is clear that depending on the business case addressed by your chatbot, some platforms may be more appropriate than others.

If you are planning to build a complex chatbot, you should seriously consider stability, scalability and flexibility aspects. If you don’t pay enough attention the intricacies of human language, a conversation can quickly go off the rails. You may be either required to build your own solution from scratch or use a combination of a tool for solving general NLP problems (i.e Api.ai) plus custom server side logic for more powerful features.

As of today, it is clear that when trying to build an ambitious chatbot, which is able to handle complex conversations and take actions (i.e payments), one cannot rely 100% on the platforms and custom NLP development is needed. [**Recent advancements**](https://tryolabs.com/blog/2016/12/06/major-advancements-deep-learning-2016/) in Deep Learning techniques may come to be of great help in the near future.