CARE

COLLEGE OF ENGINEERING

Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai
Accredited by NAAC with 'A' Grade
#27, Thayanur, Tiruchirappalli - 620009

COLLEGE CODE: 8107

COURSE: Cloud Application Development - Group 5

PHASE V: PROJECT SUBMISSION

PROJECT TITLE: Chatbot Deployment with IBM Cloud Watson Assistant

Team Members:

ABINAYA S – 810721243002

abinaya.s1@care.ac.in

DHARSHINI K – 810721243017

dharshini.k@care.ac.in

HELAN SUJA R – 810721243021

helansuja.r@care.ac.in

MANIDEEPA – 810721243030
manideepa.m@care.ac.in

MONICA PUSHPA X – 810721243031
monicapushpa.x@care.ac.in

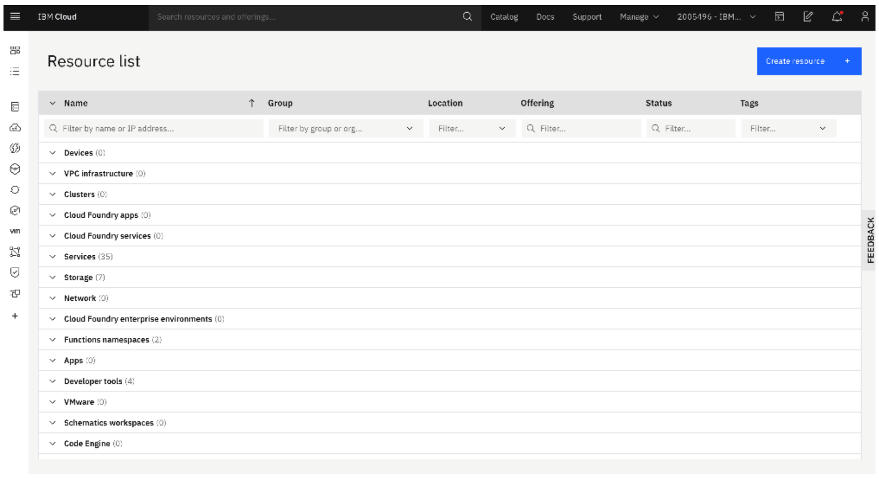
DEVELOPING WATSON ASSISTANT SERVICE

The first task is to create an instance of Watson Assistant, which is the service that allows the creation of your Watson Assistant, on IBM Cloud. Along with your Watson Assistant instance, you create your first assistant and skill.

An *assistant* helps your customers complete tasks and get information faster. It can clarify requests, search for answers from a knowledge base, and direct your customer to a human if needed. An assistant can hold a dialog skill, a search skill, or both. Think of an assistant as the end product.

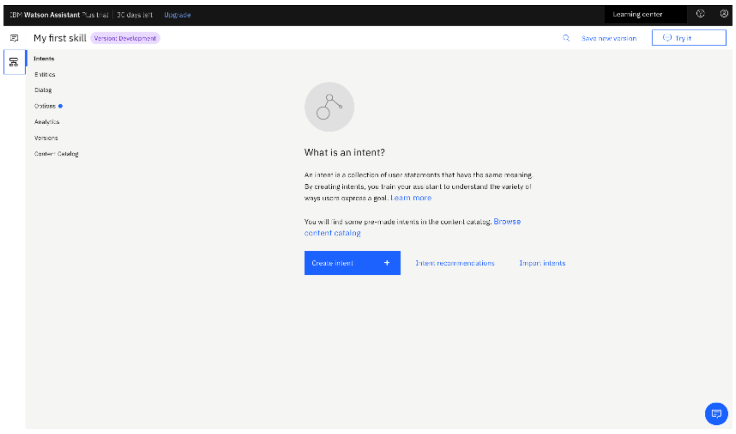
Skills contain the training to respond to your customer queries. Add skills to your assistant and then deploy to your channels. A skill can be one of two types: dialog or search. For your assistant, you create and use a dialog skill only. Think of a skill as the means to an end.

1. Logging into IBM Cloud account.



Name	Group	Location	Offering	Status	Tags
Filter by name or IP address...	Filter by group or org...	Filter...	Filter...	Filter...	Filter...
Devices (0)					
VPC infrastructure (0)					
Clusters (0)					
Cloud Foundry apps (0)					
Cloud Foundry services (0)					
Services (35)					
Storage (7)					
Network (0)					
Cloud Foundry enterprise environments (0)					
Functions namespaces (2)					
Apps (0)					
Developer tools (4)					
VMware (0)					
Schematics workspaces (0)					
Code Engine (0)					

2. Creating an assistant service.The "My first skill" page is displayed.

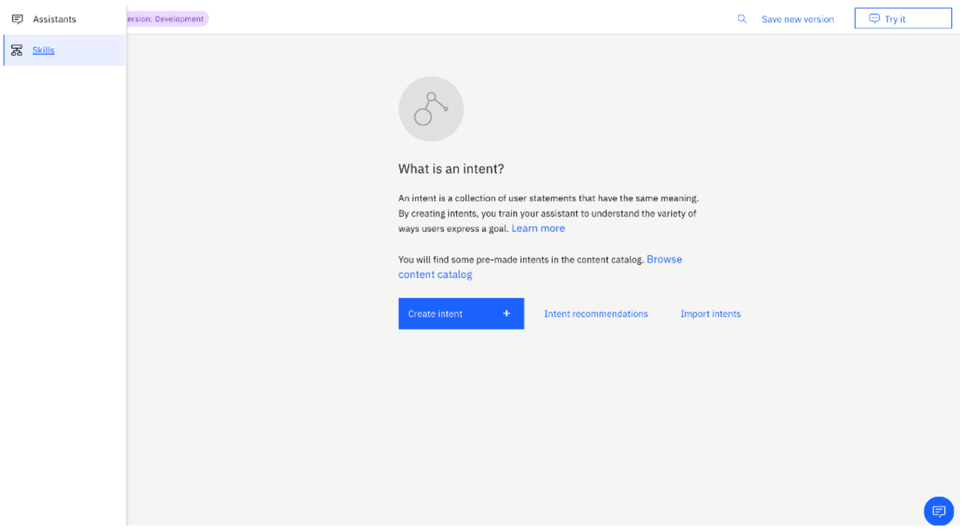


What is an intent?

An intent is a collection of user statements that have the same meaning. By creating intents, you train your assistant to understand the variety of ways users express a goal. Learn more

You will find some pre-made intents in the content catalog. Browse content catalog

3. For the service name, type `ITSupportConversation`. Click **Create**. Click **Skills**.

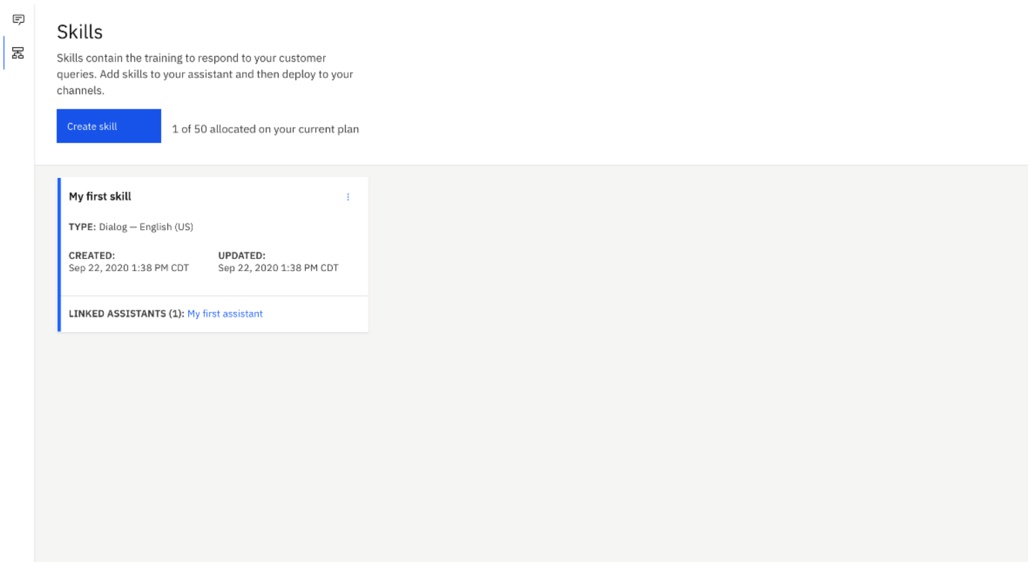


What is an intent?

An intent is a collection of user statements that have the same meaning. By creating intents, you train your assistant to understand the variety of ways users express a goal. Learn more

You will find some pre-made intents in the content catalog. Browse content catalog

4. Click **Create Skill**.



Skills

Skills contain the training to respond to your customer queries. Add skills to your assistant and then deploy to your channels.

1 of 50 allocated on your current plan

My first skill

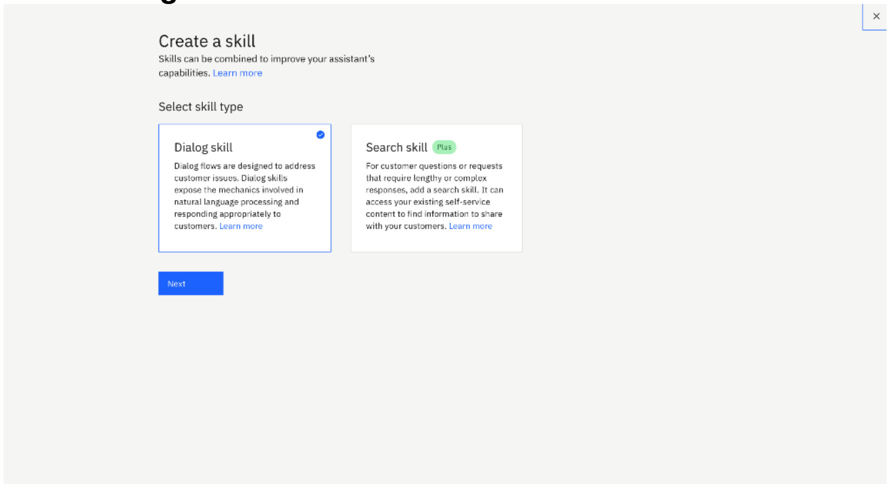
TYPE: Dialog — English (US)

CREATED: Sep 22, 2020 1:38 PM CDT

UPDATED: Sep 22, 2020 1:38 PM CDT

LINKED ASSISTANTS (1): My first assistant

5. Click **Dialog Skill** and click **Next**.



Create a skill

Skills can be combined to improve your assistant's capabilities. Learn more

Select skill type

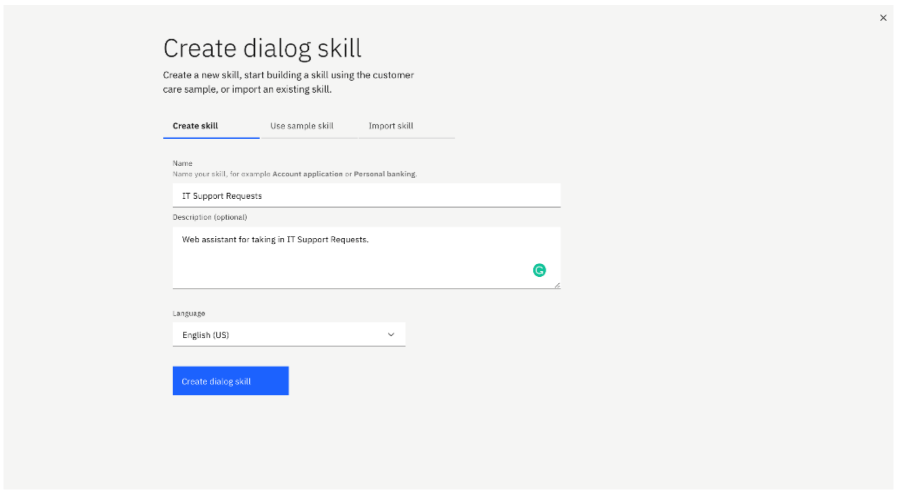
Dialog skill

Dialog flows are designed to address customer issues. Dialog skills expose the mechanics involved in natural language processing and responding appropriately to customers. Learn more

Search skill Plus

For customer questions or requests that require lengthy or complex responses, add a search skill. It can access your existing self-service content to find information to share with your customers. Learn more

6. Naming the skill `IT Support Requests` and adding a description of your own.



Create dialog skill

Create a new skill, start building a skill using the customer care sample, or import an existing skill.

Create skill | Use sample skill | Import skill

Name
Name your skill, for example **Account application** or **Personal banking**.

IT Support Requests

Description (optional)

Web assistant for taking in IT Support Requests.

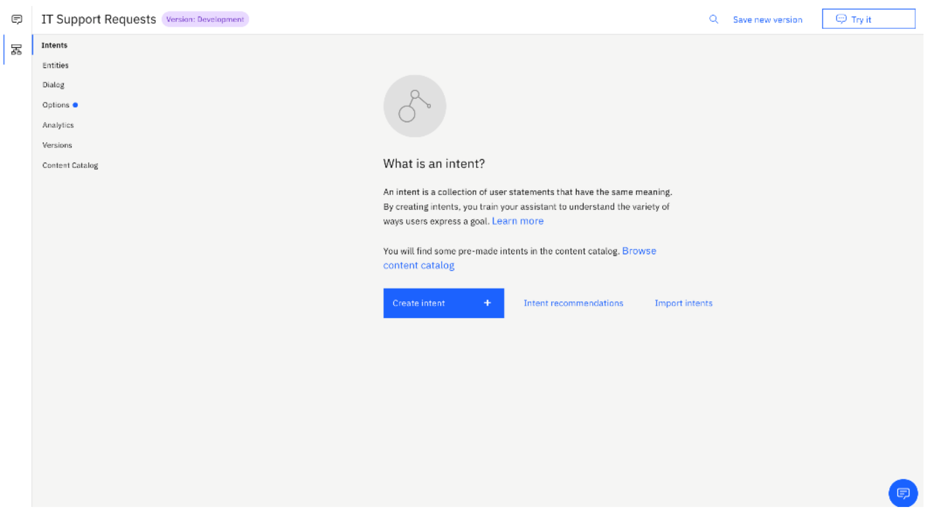
Language

English (US)

Create dialog skill

CREATING AN WORKSPACE WITH IT SUPPORT:

An intent is a collection of user statements that have the same meaning. By creating intents, you train your assistant to understand the many ways that users express a goal.
On the Intents page, you can create an intent, import intents, and search through intent recommendations.



IT Support Requests Version: Development

Save new version | Try it

Intents
Entities
Dialog
Options
Analytics
Versions
Content Catalog

What is an intent?

An intent is a collection of user statements that have the same meaning. By creating intents, you train your assistant to understand the variety of ways users express a goal. Learn more

You will find some pre-made intents in the content catalog. Browse content catalog

Create intent + | Intent recommendations | Import intents

To identify intents, start with something that a user might want and then list the ways that the user might describe it. For each intent, think of the various ways that users might express their desires—those are the examples. You can develop examples by using a crowdsourcing approach. For instance, in a discussion with the support team, you might gather this set of standard questions that support received from users:

- What is the status of the business application? I can't access it.
- How do I get access to a business application?
- How do I reset my password for a specific application?

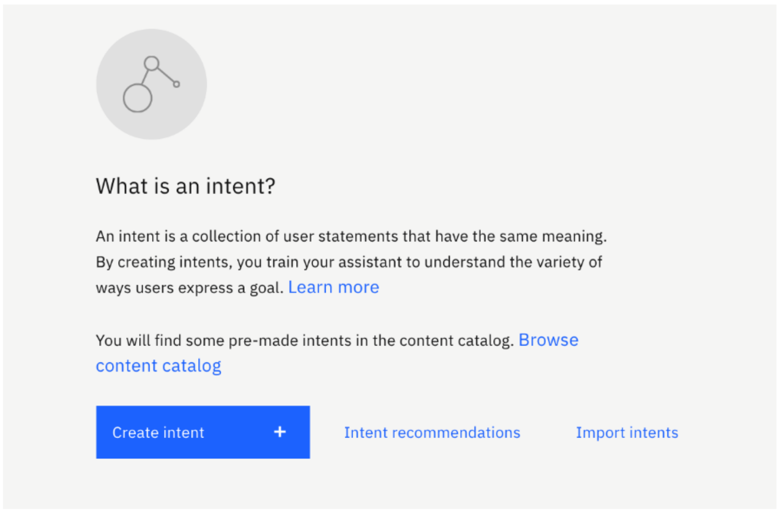
- When should I renew my workstation?
- How do I bring my own device and connect it to an enterprise network?

Each of those questions is documented as a frequently asked question in the support team's document repository. Some solutions persist in a relational database in the form of `application > problem > solution`.

Based on the questions, you can extract these intents:

- Access to a business application such as an expense report
- Reset password
- Access to supplier onboarding business process
- Bring your own device

1. Add intents to the skill.

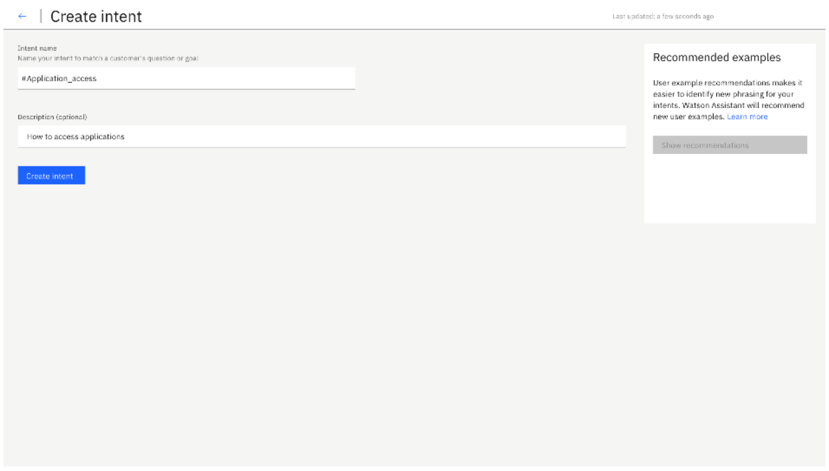


What is an intent?

An intent is a collection of user statements that have the same meaning. By creating intents, you train your assistant to understand the variety of ways users express a goal. Learn more

You will find some pre-made intents in the content catalog. Browse content catalog

2. For the intent name, type `Application_access` after the number sign (#). Add a brief description and click **Create intent**. Within Watson Assistant, intents are denoted and prefixed with a # sign.



Create intent

Last updated: a few seconds ago

Intent name
Name your intent to match a customer's question or goal

#Application_access

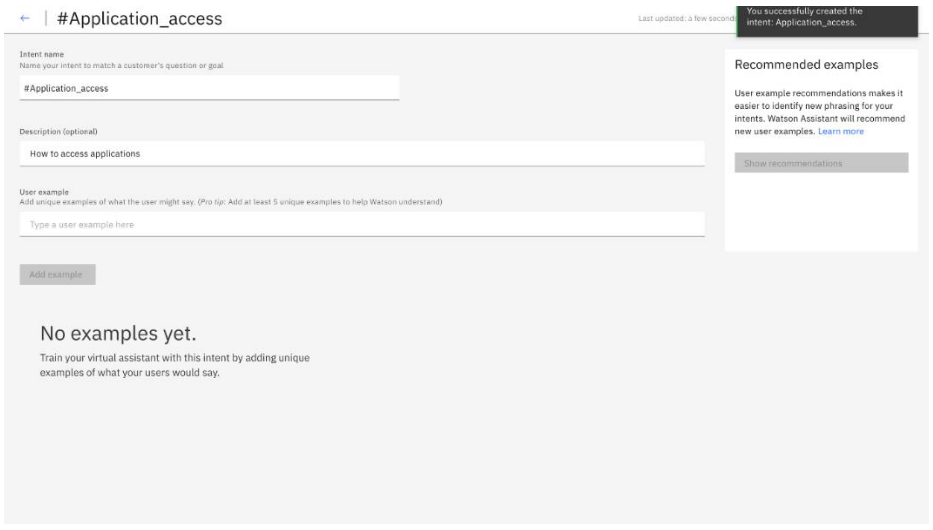
Description (optional)

How to access applications

Recommended examples

User example recommendations makes it easier to identify new phrasing for your intents. Watson Assistant will recommend new user examples. Learn more

3. For each intent, add examples to train the conversation for intent recognition. You can enter these examples in the **User example** field one by one and press Return after each example.


← | #Application_access

Last updated: a few seconds

You successfully created the intent: Application_access.

Intent name
Name your intent to match a customer's question or goal

#Application_access

Description (optional)

How to access applications

User example
Add unique examples of what the user might say. (*Pro tip*: Add at least 5 unique examples to help Watson understand)

Type a user example here

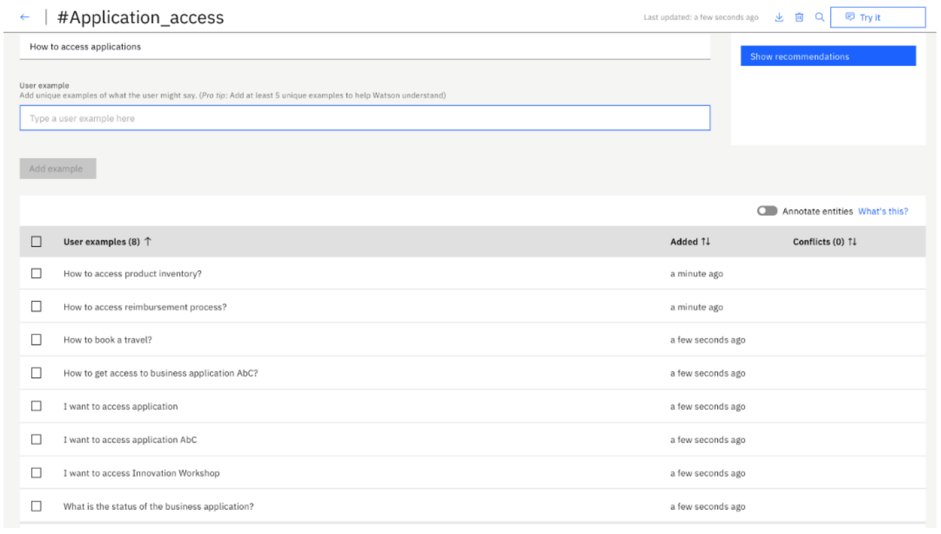
Recommended examples

User example recommendations makes it easier to identify new phrasing for your intents. Watson Assistant will recommend new user examples. Learn more

No examples yet.

Train your virtual assistant with this intent by adding unique examples of what your users would say.

This example shows your #Application_access intent and your user examples:


← | #Application_access

Last updated: a few seconds ago

How to access applications

User example
Add unique examples of what the user might say. (*Pro tip*: Add at least 5 unique examples to help Watson understand)

Type a user example here

☐	User examples (8) ↑	Added ↑↓	Conflicts (0) ↑↓
☐	How to access product inventory?	a minute ago	
☐	How to access reimbursement process?	a minute ago	
☐	How to book a travel?	a few seconds ago	
☐	How to get access to business application AbC?	a few seconds ago	
☐	I want to access application	a few seconds ago	
☐	I want to access application AbC	a few seconds ago	
☐	I want to access Innovation Workshop	a few seconds ago	
☐	What is the status of the business application?	a few seconds ago	

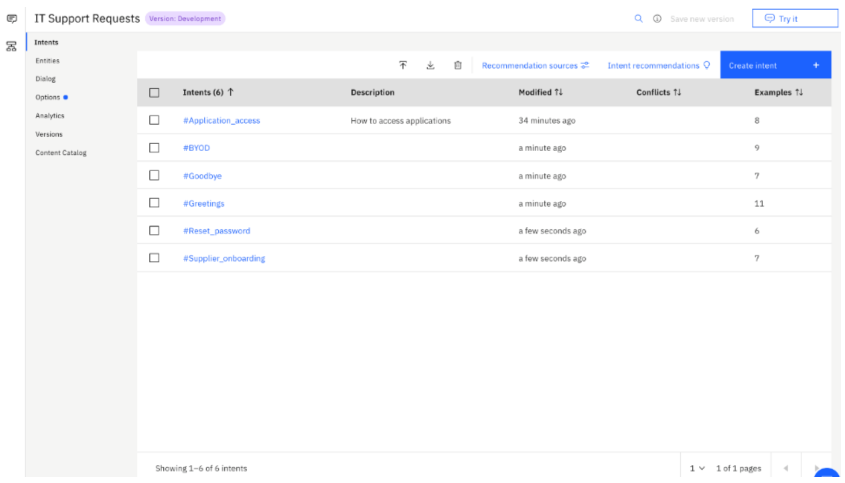
4. Create five more intents for your IT Service Request Assistant and add user examples to them. The intents and their corresponding user examples are listed in the following table.

Intent	**User examples**
Greetings	Hello
	Hi
	How are you?
	Bonjour
	Howdy
	What's up?

	Good Morning?
	Namaste
	Hola
	What is up?
	Hey
Goodbye	Bye
	Bye-bye
	Ciao
	Have a good day
	Take care
	See you later
	TTYL
Reset password	How can I reset my password?
	I could not login in this morning. Is my password expired?
	How can I change my password?
	How can I reset the access password for the AbC application?
	What are the different options to reset my password?
	I want to reset my intranet password
Supplier on boarding	I want access to the supplier due diligence business process
	I want to onboard a new supplier
	Onboarding a new supplier
	Supplier due diligence
	What is the process to onboard a new supplier?
	What is the URL of the business process for onboarding a supplier?
	I have a new supplier. What is the procedure to access it?

BYOD	Bring my laptop
	Bring my phone
	BYOD
	Bring my tablet
	Use my iPhone for work
	Use my phone for business
	Use my own smartphone for work
	Use my iPad for business
	I want to bring my own device

After updating the intents, your Intents page looks like this :

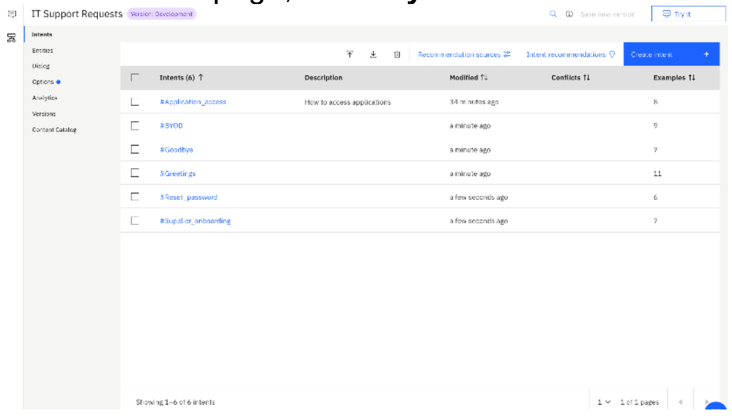


Intents (6)	Description	Modified	Conflicts	Examples
#Application_access	How to access applications	34 minutes ago		8
#BYOD		a minute ago		9
#Goodbye		a minute ago		7
#Greetings		a minute ago		11
#Reset_password		a few seconds ago		6
#Supplier_onboarding		a few seconds ago		7

Showing 1–6 of 6 intents 1 1 of 1 pages

TESTING THE CONVERSATIONS:

1. On the Intents page, click **Try it.**

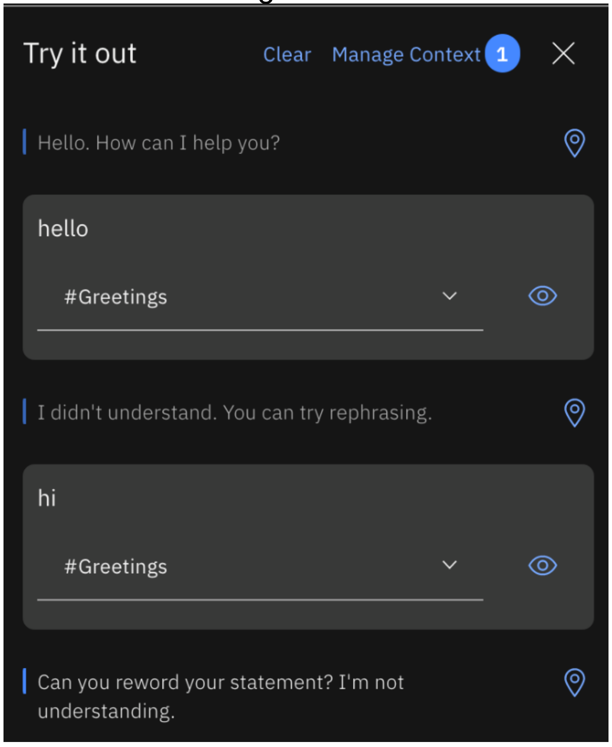

IT Support Requests Version: Development

Intents (6)	Description	Modified	Conflicts	Examples
#Application_access	How to access applications	34 minutes ago		8
#BYOD		a minute ago		9
#Goodbye		a minute ago		7
#Greetings		a minute ago		11
#Reset_password		a few seconds ago		6
#Supplier_onboarding		a few seconds ago		7

Showing 1–6 of 6 intents 1 1 of 1 pages

You can test a couple of your intents with a few sample utterances in the **Enter something to test your assistant** field.

2. #Greetings intent. Watson identifies the #Greetings intent. Enter other greetings to test the Greetings intent.



Try it out

Hello. How can I help you?

hello

#Greetings

I didn't understand. You can try rephrasing.

hi

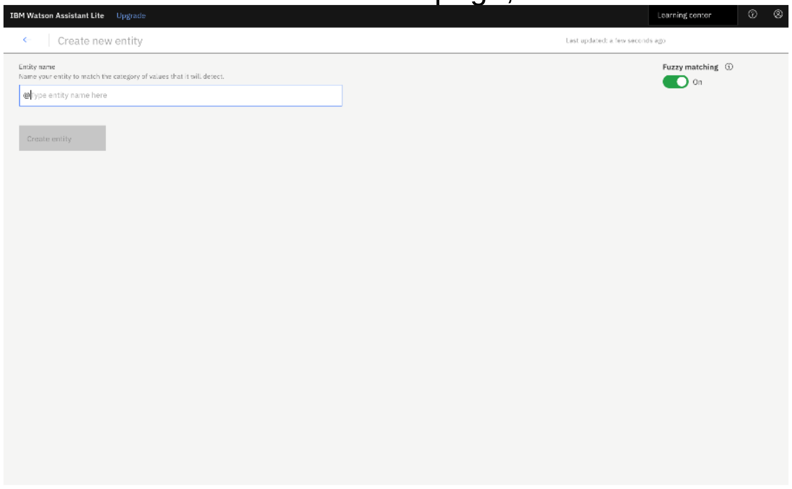
#Greetings

Can you reword your statement? I'm not understanding.

ADDING ENTITIES:

An *entity* is a portion of the user's input that you can use to provide a different response to a particular intent. Entities are like nouns or keywords.
You can also enable pre-built system entities, dates, times, and numbers. You don't use the pre-built entities yet.

1. Click **Entities**. On the Entities page, click **Create new**.



Create new entity

Last updated: a few seconds ago

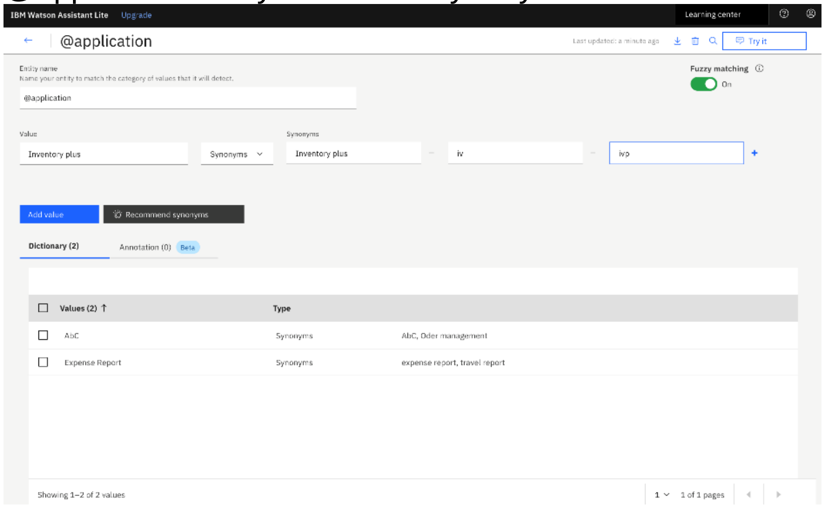
Entity name
Name your entity to match the category of values that it will detect.

@Type entity name here

Fuzzy matching
On

Adding values and synonyms to entities helps your chatbot to learn important details that your users might mention. Each entity definition includes a set of specific entity values.

2. Create entities to represent to the application that the user wants to access. Entities are represented in Watson Assistant with the at (@) sign. Enter the @application entity with a few synonyms.



@application

Entity name
Name your entity to match the category of values that it will detect.
@application

Fuzzy matching: On

Value		Synonyms		
Inventory plus	Synonyms	Inventory plus	iv	ivp

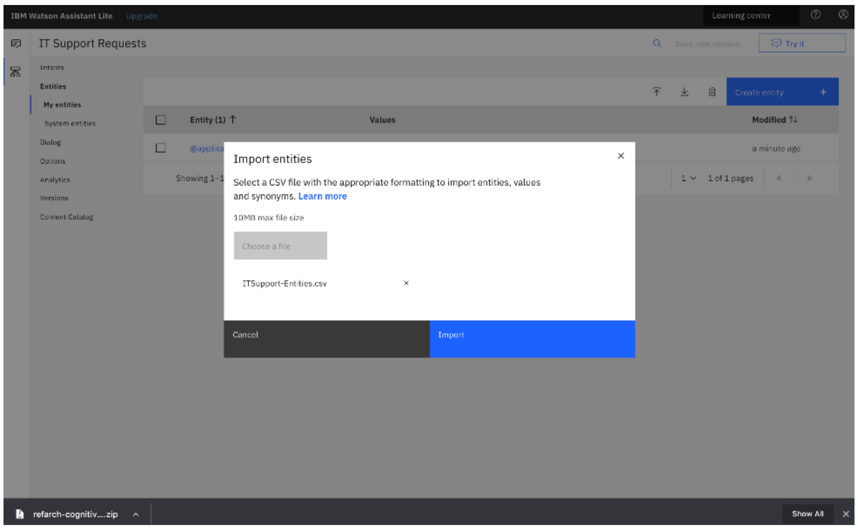
Dictionary (2)

Values (2)	Type	
AbC	Synonyms	AbC, Oder management
Expense Report	Synonyms	expense report, travel report

Showing 1–2 of 2 values 1 1 of 1 pages

you can reuse entities' definitions through the export and import capabilities. Import the `ITSupport-Entities.csv` file.

3. On the "My entities" page, click the **Upload** icon. Click **Choose a file** and select **ITSupport-Entities.csv**. Click **Import**.



Import entities

Select a CSV file with the appropriate formatting to import entities, values and synonyms. Learn more

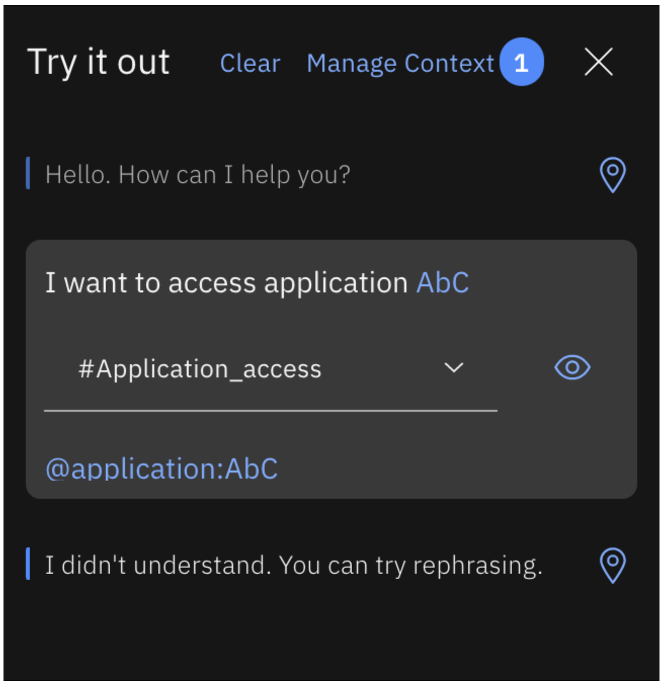
10MB max file size

Choose a file

ITSupport-Entities.csv

Cancel   Import

4. Test your work so far. If you click the **Try it** icon immediately after you import the entities, the `Watson is training` message is displayed while Watson Assistant class. The following image shows both the intent and entity (@application:AbC) that is extracted by Watson Assistant:


Try it out Clear Manage Context 1

Hello. How can I help you?

I want to access application AbC

#Application_access

@application:AbC

I didn't understand. You can try rephrasing.

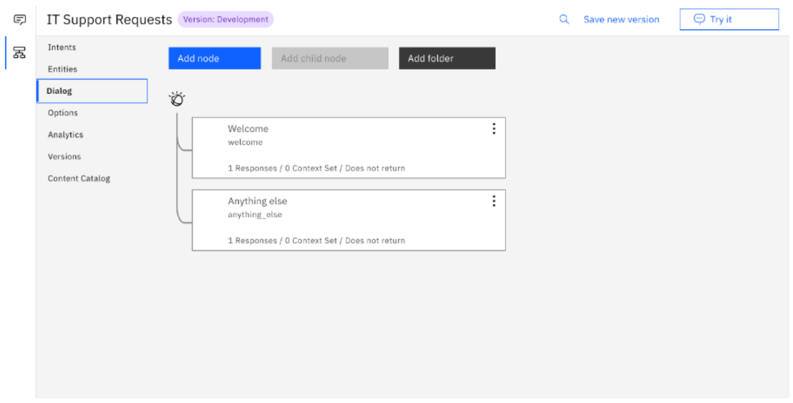
Now that you entered your intents and entities, you're ready to create the dialog flow.

BUILDING THE DIALOGUE:

After you specify your intents and entities, you can construct the dialog flow. A dialog is made up of nodes that define steps in the conversation.

Get familiar with dialogs

1. Click **Dialog**.


IT Support Requests Version: Development Save new version Try it

Intents
Entities
Dialog
Options
Analytics
Versions
Content Catalog

Add node Add child node Add folder

Welcome
welcome
1 Responses / 0 Context Set / Does not return

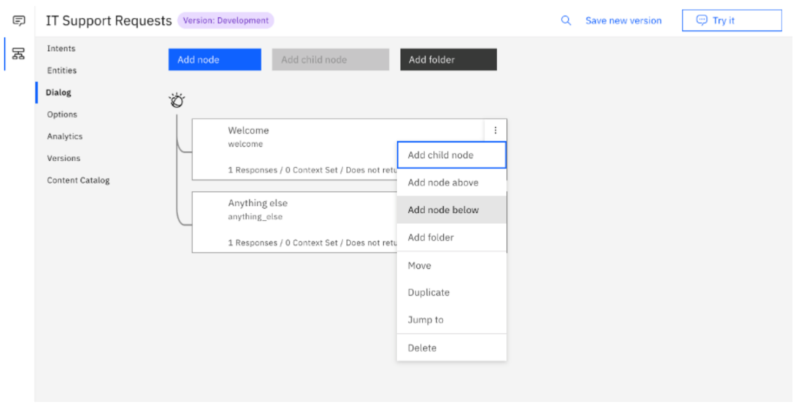
Anything else
anything_else
1 Responses / 0 Context Set / Does not return

Click to expand the image

   Two dialog nodes are shown. The first node is the standard welcome message. The other node is a catch-all node that is named "Anything else." Dialog

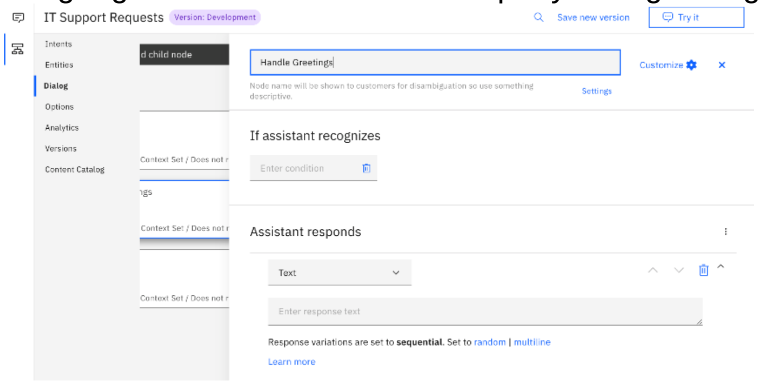
   If you click the welcome node, the standard Watson response is "Hello. How can I help you?"

2. The first node addresses greetings in a response to a query such as "hello." On the welcome node, click the menu icon, which looks like three dots, and click **Add node below**.



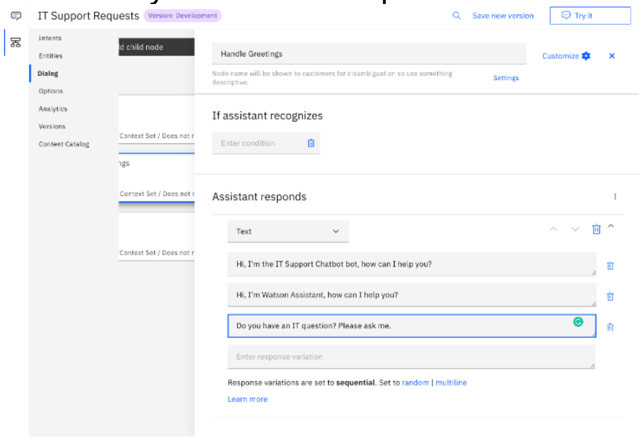
A new node is added between the welcome and "Anything else" nodes. At each node level, you can expand the conversation.

1. Name the new node `Handle Greetings`. In the **If bot recognizes** field, change the value to `#Greetings`.The condition is triggered when the Watson Natural Language Classifier classifies the query as a greeting intent.



Handle Greetings

2. In the "Assistant responds" section, add these responses:
   - Hi, I'm the IT Support Chatbot bot, how can I help you?
   - Hi, I'm Watson Assistant, how can I help you?
   - Do you have an IT question? Please ask me.



Handle Greetings

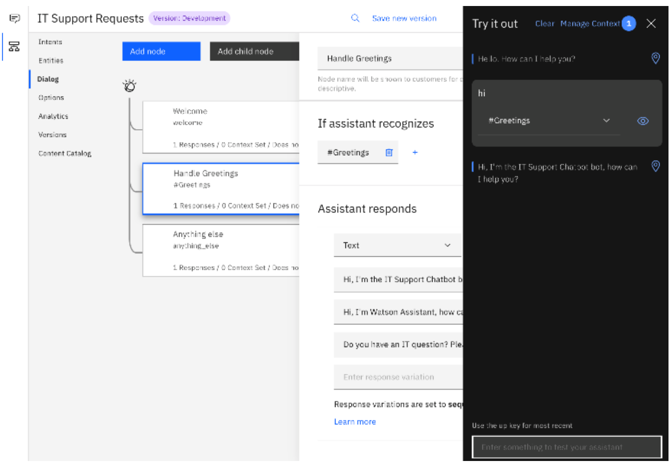
Hi, I'm the IT Support Chatbot bot, how can I help you?

Hi, I'm Watson Assistant, how can I help you?

Do you have an IT question? Please ask me.

As shown in the image, you can use the multiple responses pattern to avoid being repetitive. The bot can present different answers to the same query.

3. Unit-test your dialog by clicking **Try it**.



Welcome
welcome
1 Responses / 0 Context Set / Does no

Handle Greetings
#Greetings
1 Responses / 0 Context Set / Does no

Anything else
anything_else
1 Responses / 0 Context Set / Does no

Handle Greetings

Node name will be shown to customers for
descriptive.

If assistant recognizes

#Greetings

Assistant responds

Hi, I'm the IT Support Chatbot b

Hi, I'm Watson Assistant, how ca

Do you have an IT question? Ple

Enter response variation

Response variations are set to seq

Hello. How can I help you?

hi

#Greetings

Hi, I'm the IT Support Chatbot bot, how can I help you?

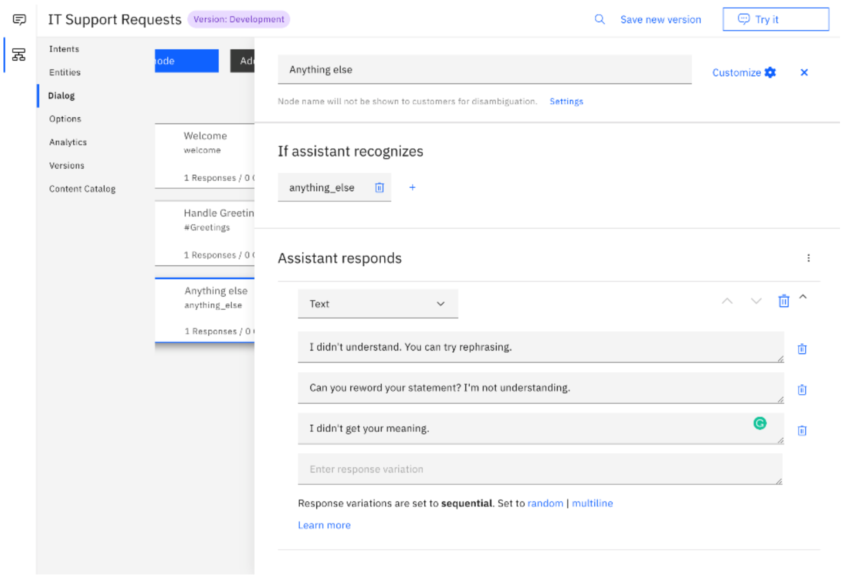
Use the up key for most recent

Enter something to test your assistant

At the beginning of each conversation, the evaluation starts at the first level of dialog nodes.

Manage the "Anything else" use case

The last node is used when none of the defined intents are matched.



Welcome
welcome
1 Responses / 0

Handle Greetin
#Greetings
1 Responses / 0

Anything else
anything_else
1 Responses / 0

Anything else

Node name will not be shown to customers for disambiguation.

If assistant recognizes

anything_else

Assistant responds

I didn't understand. You can try rephrasing.

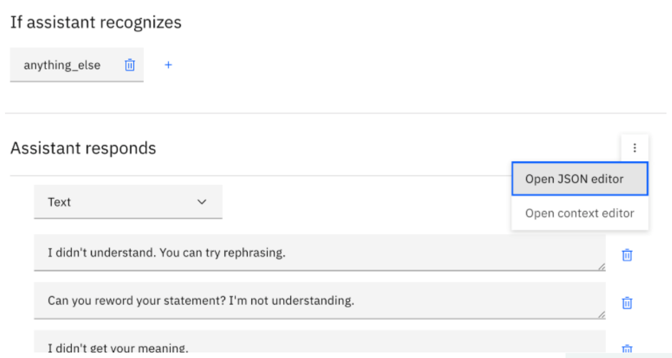
Can you reword your statement? I'm not understanding.

I didn't get your meaning.

Enter response variation

Response variations are set to **sequential**. Set to random | multiline

1. From the menu in the "Assistant response" area, open the JSON editor to assess the data that is returned as part of the conversation interaction.



If assistant recognizes

anything_else

Assistant responds

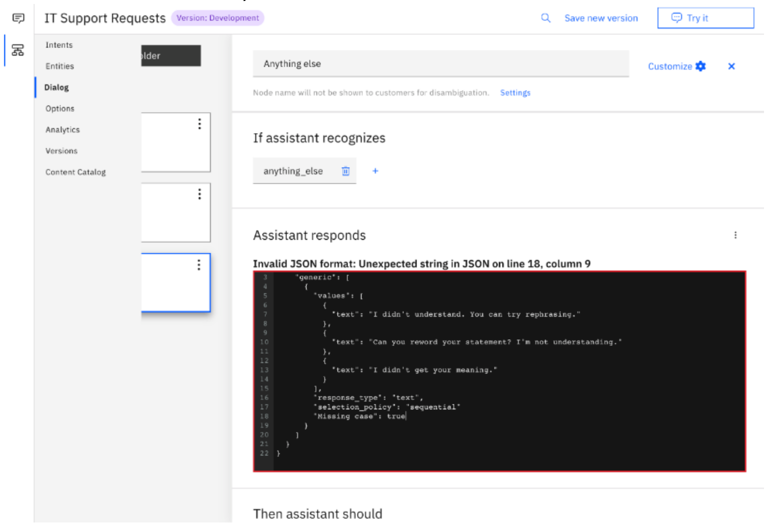
I didn't understand. You can try rephrasing.

Can you reword your statement? I'm not understanding.

I didn't get your meaning.

2. To the output, add an attribute, name it `Missing case`, and set it to true. When you persist the conversation flow into a document-oriented database, you can

search the queries that the dialog nodes didn't address so that you can add more cases later, if needed.



Anything else

Node name will not be shown to customers for disambiguation.

If assistant recognizes

anything_else

Assistant responds

Invalid JSON format: Unexpected string in JSON on line 18, column 9

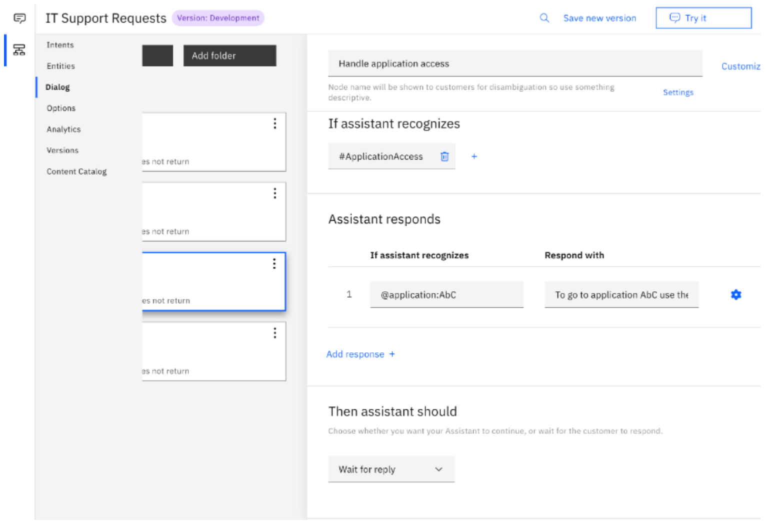
```
    "generic": [
      {
        "values": [
          {
            "text": "I didn't understand. You can try rephrasing."
          },
          {
            "text": "Can you reword your statement? I'm not understanding."
          },
          {
            "text": "I didn't get your meaning."
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
        "Missing case": true
      }
    ]
  }
}
```

Then assistant should

Define the application access dialog flow

Create a dialog branch to handle the #ApplicationAccess intent.

1. Click the **Handle Greetings** node and click **Add node** and select **Add below**. Name the new node `Handle application access`. In the **If bot recognizes** field, type `#ApplicationAccess`.



Handle application access

Node name will be shown to customers for disambiguation so use something descriptive.

If assistant recognizes

#ApplicationAccess

Assistant responds

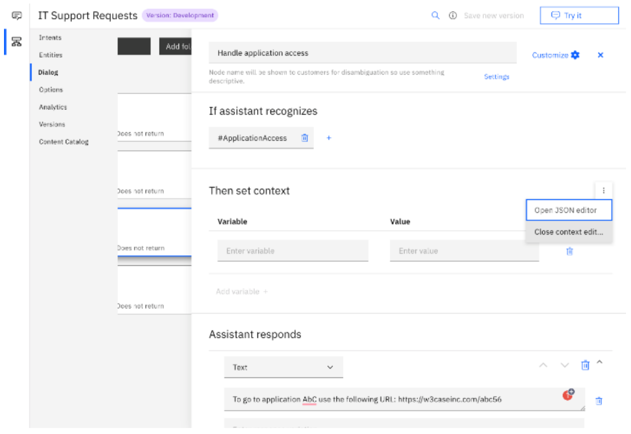
	If assistant recognizes	Respond with
1	@application:AbC	To go to application AbC use the

Then assistant should

Choose whether you want your Assistant to continue, or wait for the customer to respond.

Wait for reply

2. Click **Customize** next to the node name.



Handle application access

Node name will be shown to customers for disambiguation so use something descriptive.

If assistant recognizes

#ApplicationAccess

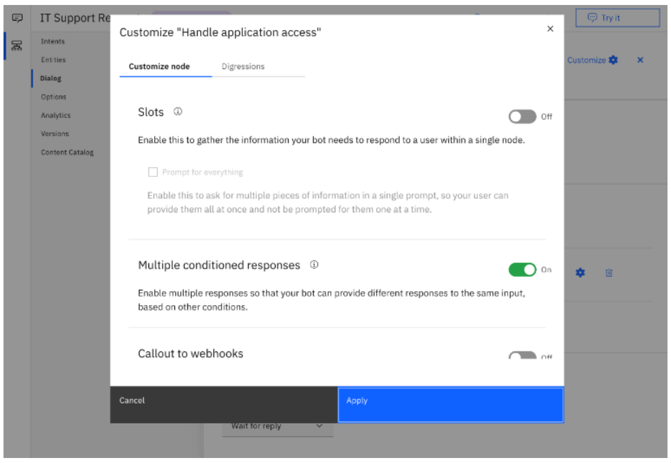
Then set context

Variable	Value
Enter variable	Enter value

Assistant responds

To go to application AbC use the following URL: https://w3caseinc.com/abc56

3. In the dialog that is displayed, set **Multiple conditioned responses** to **On**.


Customize "Handle application access"

Customize node | Digressions

Slots — Off

Enable this to gather the information your bot needs to respond to a user within a single node.

Prompt for everything

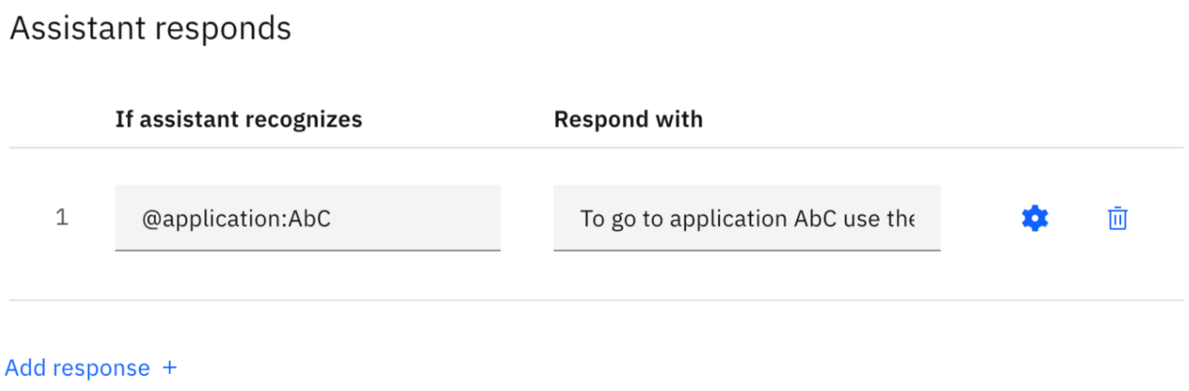
Enable this to ask for multiple pieces of information in a single prompt, so your user can provide them all at once and not be prompted for them one at a time.

Multiple conditioned responses — On

Enable multiple responses so that your bot can provide different responses to the same input, based on other conditions.

Callout to webhooks — Off

4. To add more conditions in the same node, click **Add response**.

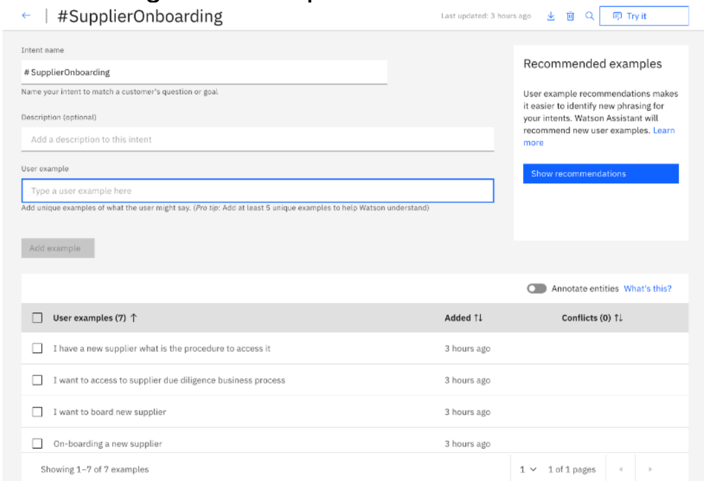

Assistant responds

	If assistant recognizes	Respond with
1	@application:AbC	To go to application AbC use the

ADDING HTML TO THE RESPONSE:

Suppose that you want to return an actionable URL, meaning that the response includes a `URL` variable that the user can click to go to a new page. To try this scenario, you present the URL of a business process that is deployed on IBM Business Process Manager on Cloud.

1. Add an intent to support the user's query about accessing the "Supplier onboarding business process.


#SupplierOnboarding

Last updated: 3 hours ago

Intent name: #SupplierOnboarding

Name your intent to match a customer's question or goal

Description (optional): Add a description to this intent

User example: Type a user example here

Add unique examples of what the user might say. (*Pro tip*: Add at least 5 unique examples to help Watson understand)

Recommended examples

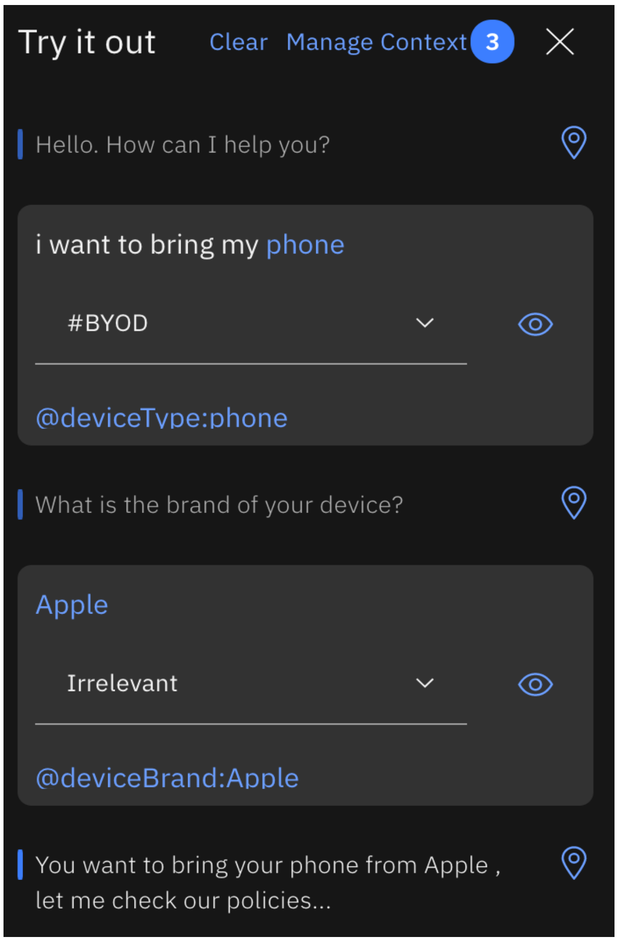
User example recommendations makes it easier to identify new phrasing for your intents. Watson Assistant will recommend new user examples. Learn more

User examples (7)	Added	Conflicts (0)
I have a new supplier what is the procedure to access it	3 hours ago	
I want to access to supplier due diligence business process	3 hours ago	
I want to board new supplier	3 hours ago	
On-boarding a new supplier	3 hours ago	

Showing 1–7 of 7 examples — 1 of 1 pages

Using slots and context variables

Add a dialog flow to address when users wants to bring their own device. Only certain brands and devices are supported, so you must determine the brand and type of device. The goal is illustrated by this dialog:



Hello. How can I help you?

i want to bring my phone

#BYOD

@deviceType:phone

What is the brand of your device?

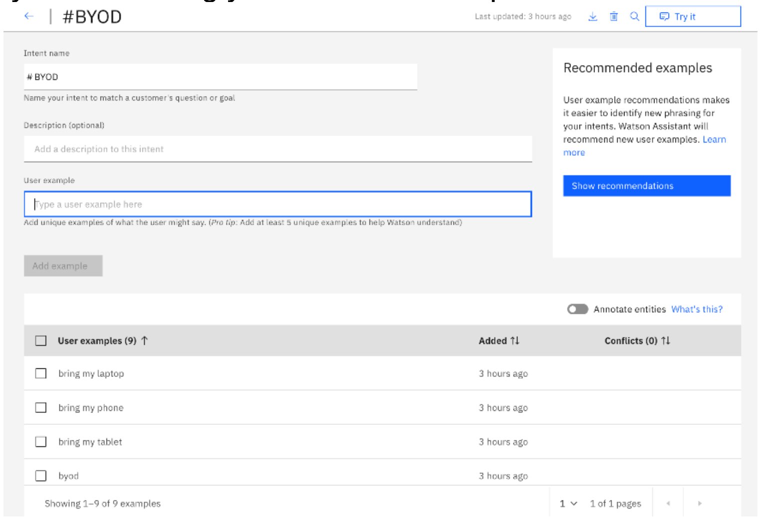
Apple

Irrelevant

@deviceBrand:Apple

You want to bring your phone from Apple , let me check our policies...

From the first query, "I want to bring my phone", Watson Assistant can get the #BYOD intent and the @deviceType:phone entity.
 The dialog flow asks for the brand If you didn't import the intent and entities, create an entity for the "bring your own device" question.



Intent name

# BYOD

Name your intent to match a customer's question or goal

Description (optional)

Add a description to this intent

User example

Type a user example here

Add unique examples of what the user might say. (*Pro tip*: Add at least 5 unique examples to help Watson understand)

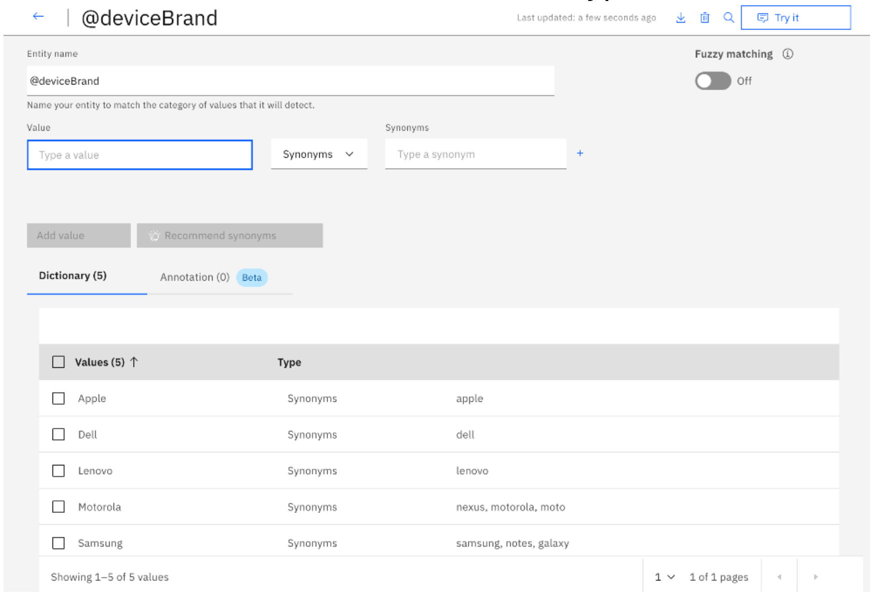
Recommended examples

User example recommendations makes it easier to identify new phrasing for your intents. Watson Assistant will recommend new user examples. Learn more

User examples (9) ↑	Added ↑↓	Conflicts (0) ↑↓
bring my laptop	3 hours ago	
bring my phone	3 hours ago	
bring my tablet	3 hours ago	
byod	3 hours ago	

Showing 1–9 of 9 examples 1 1 of 1 pages

1. Create the @deviceBrand and @deviceType entities.



← | @deviceBrand

Last updated: a few seconds ago

Try it

Entity name

@deviceBrand

Name your entity to match the category of values that it will detect.

Fuzzy matching

Off

Value

Type a value

Synonyms

Synonyms

Type a synonym

Add value

Recommend synonyms

Dictionary (5)

Annotation (0) Beta

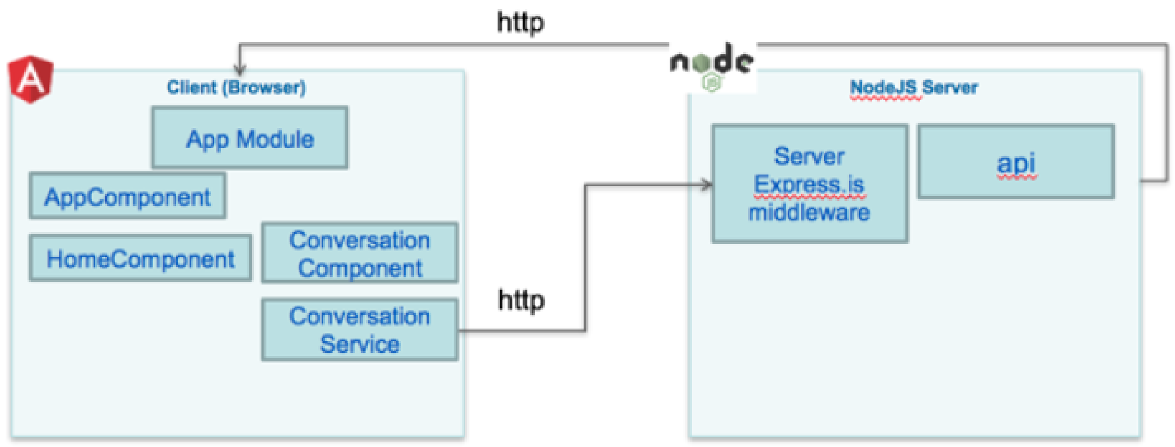
Values (5) ↑	Type	
Apple	Synonyms	apple
Dell	Synonyms	dell
Lenovo	Synonyms	lenovo
Motorola	Synonyms	nexus, motorola, moto
Samsung	Synonyms	samsung, notes, galaxy

Showing 1–5 of 5 values

1 1 of 1 pages

CODING EXPLANATION:

The project is split into two parts: the client side that is an Angular 2 single page application (code under client folder) and the server which is an express app with code under folder:



Server side

The code is under the *server/* folder. The server.js is the main javascript entry point code, started when the *npm start* command is executed. The server uses *expressjs*, serves a index.html page for the angular front end, and delegates to another javascript module (routes/api.js) any HTTP calls to url starting with **/api/***.
Expressjs is a routing and middleware web framework used to simplify web server implementation in nodejs. An app is a series of middleware function calls. See expressjs.com for more details. The cfenv is used to deploy the application in Bluemix as a cloud foundry application.

```
const express = require('express');
const app = express();
```

```
var config = require('./config/config.json');
require('./routes/api')(app,config)

/ Catch all other routes and return the index file
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../dist/index.html'));
});

// get the app environment from Cloud Foundry
var appEnv = cfenv.getAppEnv();
/**
 * Get port from environment or local config parameters.
 */
const port = process.env.PORT || config.port;
..
```
When a user enters the hostname url without any web context, the index.html page will be presented. Any URL with the pattern http://hostname/api will be supported by api.js script.

dist is a folder for scripts built with @angular/cli **ng build** command, so it contains the user interface generated code.

This code needs to be improved with authentication and authorization controls.

The package.json file specifies the minimum dependencies for the server and client codes. The interesting dependencies is the watson-developer-cloud module needed to interact with any on cloude Watson service.

```
"dependencies": {
  ...
  "body-parser": "^1.15.0",
  "cookie-parser": "^1.4.1",
  "express": "^4.13.x",
  "express-session": "1.13.0",
  "request": "^2.72.0",
  "watson-developer-cloud": "^2.x",
```
The api.js defines the URLs to be used by *angular 2* AJAX calls. The user interactions in the Browser are supported by Angular 2, with its own Router mechanism and with its DOM rendering capabilities via directives and components. When there is a need to send data to the server for calling one of the Cognitive Service, an AJAX calls is done and the server will respond asynchronously later.

api.js uses the express.js middleware router to handle URL mapping.

```
module.exports = function(app,config) {

// Support REST call
app.post('/api/conversation',function(req,res){
    if(!req.body){
      res.status(400).send({error:'no post body'});
    } else {
      if (req.body.context.type !== undefined && req.body.context.type ==
"sodb") {
        conversation.sobdConversation(config,req,res);
      } else {
        conversation.itSupportConversation(config,req,res);
      }
    }
});
}
```
On the HTTP POST to /api/conversation the text is in the request body, and can be sent to Watson conversation. The code here is illustrating how to support different conversations from the user interface: for demonstration purpose the base interface is using the IT support conversation, but there is a second conversation for the Supplier on boarding business process (sobd) to use, so a second interface is used, and the context.type variable will help to understand what conversation is called.

The second piece of interesting code is the Watson Conversation Broker under routes/features/conversation.js

This code is straight forward, it uses the configuration given as parameter then interacts with Watson cloud developer javascript APIs.

```
module.exports = {
  /**
  Specific logic for the conversation related to IT support. From the
response the
  code could dispatch to BPM.
  It persists the conversation to remote cloudant DB
  */
   itSupportConversation : function(config,req,res) {
        // this logic applies when the response is expected to be a value
to be added to a context variable
        // the context variable name was set by the conversation dialog
        if (req.body.context.action === "getVar") {
            req.body.context[req.body.context.varname] = req.body.text;
        }

sendMessage(config,req,config.conversation.workspace1,res,processITSupport
Response);
  }, // itSupportConversation
}
var sendMessage = function(config,req,wkid,res,next){
  conversation = watson.conversation({
          username: config.conversation.username,
          password: config.conversation.password,
          version: config.conversation.version,
          version_date: config.conversation.versionDate});

  conversation.message(...);
}
```

The two exposed functions are used to separate the call to the different Watson Conversation workspace. Remember a Watson Conversation service can have one to many workspaces. The settings are externalized in the config/config.json file.

```
"conversation" :{
  "version":"2017-02-03",
  "username":"291d93   ae533",
  "password":"aDF QlD",
  "workspace1":"1a3b0abfc1",
  "conversationId":"ITSupportConversation",
  "workspace2":"80b459cd2405",
  "usePersistence": true
},
```

Finally the last method is to send the text and conversation context to Watson Conversation.

As the conversation holds a context object to keep information between different interactions, the code specifies a set of needed attributes: input, context and workspace ID which can be found in the Watson Conversation Service. If the context is empty from the first query, the conversationId is added. See Watson Conversation API for information about the context.

Service orchestration

A broker code is doing service orchestration. There are two examples illustrated in the `conversation.js` code via the **next** function given as parameter to sendMessage. For example if the dialog flow adds an **action** variable in the context then the code can test on it and call a remote service.

```
...
      if (rep.context.action === "trigger" && rep.context.actionName ===
"supplierOnBoardingProcess") {

bpmoc.callBPMSupplierProcess(rep.context.customerName,rep.context.productN
ame);
      }
   }
```

See this note for BPM integration details.

We can also persist the conversation flow inside a document oriented database, for detail consultthis note

User interface controls

A classical usage of the conversation is to propose a predefined set of answers the end user can select from. The user interface can propose a HTML button for each option. The 'Advisor' page is defining such UI controls:

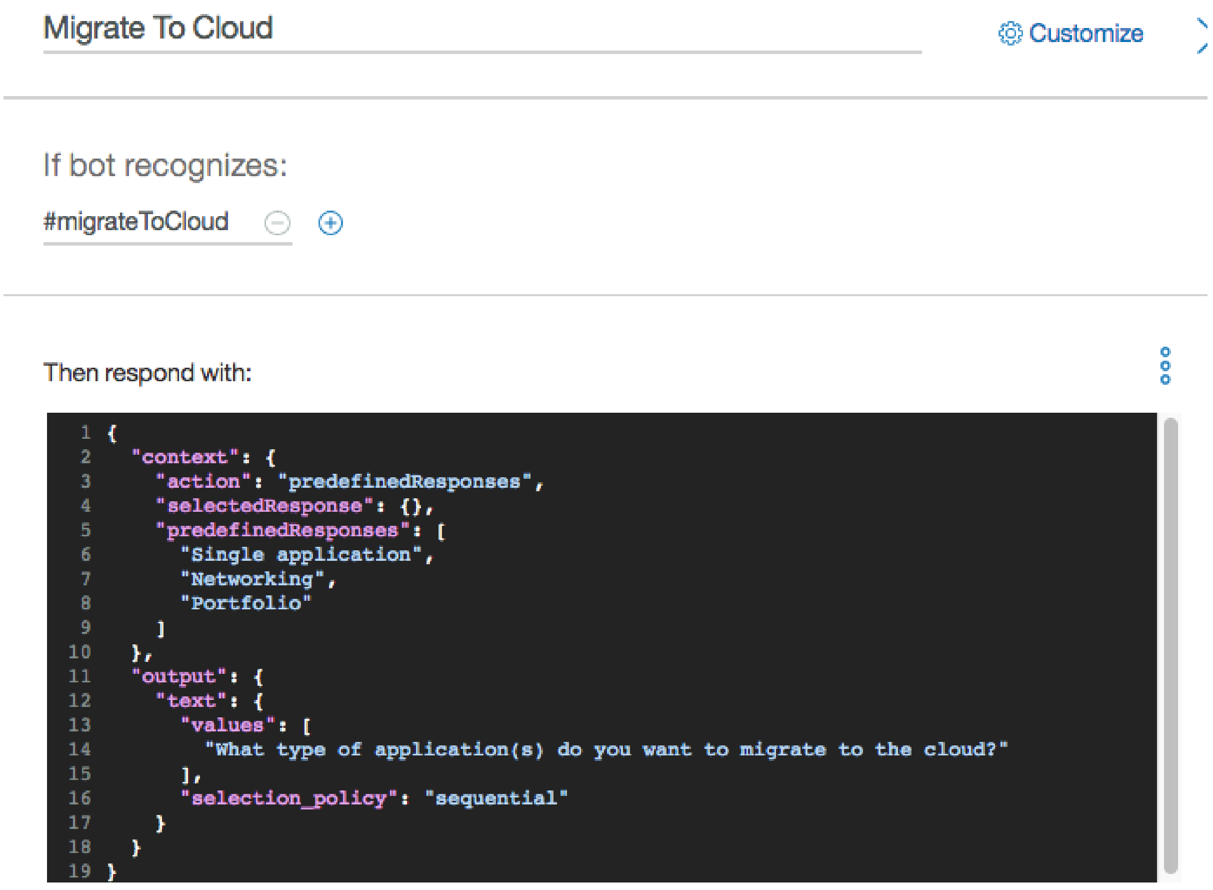
```
<div class="{{p.direction+'-text'}}">
  <div [innerHTML]="p.text"></div>
  <div *ngIf="p.options">
    <div *ngFor="let c of p.options">
```

```
        <br/>
      <button type="button" (click)="advisorResponse(c)" class="btn btn-
primary">{{c}}</button>
    </div>
  </div>
</div>
```

The options are built in the controller `advisor.component.ts` using the response coming from the server:

```
  s.options=data.context.predefinedResponses;
```

The context is the Watson Conversation context object and the `predefinedResponses` is an attribute added by the Watson conversation dialog. The figure below illustrates this settings in one of the response:



```
{
  "context": {
    "action": "predefinedResponses",
    "selectedResponse": {},
    "predefinedResponses": [
      "Single application",
      "Networking",
      "Portfolio"
    ]
  },
  "output": {
    "text": {
      "values": [
        "What type of application(s) do you want to migrate to the cloud?"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

Angular 4 client app

The code is under *client* folder. It was built using the Angular command line interface (ng new <aname>``). The ngtool with thenew` command creates the foundation for a simple Angular web app with the tooling to build and run a light server so the UI developer can work on the layout and screen flow without any backend. It is possible to use the angular server and be able to develop and test the user interface using the command:

```
$ ng serve
or
$ ng build
```

And use the URL http://localhost:4200.

In the current project this type of testing is no more necessary as the server code exists and supports the REST api needed by the user interface. So to run the server use the npm command:

```
$ npm run dev
```

And then use the URL with the port number reported by the trace:

```
[1] [nodemon] starting `node server/server server/server`
[1] info: ** No persistent storage method specified! Data may be lost when
process shuts down.
[1] info: ** Setting up custom handlers for processing Slack messages
[1] info: ** API CALL: https://slack.com/api/rtm.start
[1] Server v0.0.1 starting on http://localhost:3001
```

Client code organization

Under the client folder the first important file to consider is the `index.html` which loads the angular 2 app.
The following code illustrates the most important parts:

- including bootstrap css
- loading the angular *app-root* directive
- and as we use client side url routing the base href="/" is added to the HTML header section to give *Angular* the url base context.

```
<head>
    <base href="/">
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.cs
s" rel="stylesheet">
</head>
<body>
  <app-root>Welcome to IT Support App... Loading AppComponent content here
..</app-root>
</body>
```

The main.ts script is a standard **Angular 2** typescript file to load the main module. Nothing to add.

The *app-root* tag is defined in the `app/app.components.ts`. This file uses a base html, used as template for header, body and footer parts. The body part will be injected using the Angular 2 routing mechanism.

```
  <router-outlet></router-outlet>
```

Most of the user interactions on the Browser are supported by Angular 2, with its Router mechanism and its DOM rendering capabilities via directives and components. When there is a need to send data to the server for persistence or calling one of the Cognitive Service, an AJAX calls is done and the server will respond asynchronously later.

The application uses a unique route, as of now, but the approach should help you to add more elements as user interface components.

The application routing is defined in the app.module.ts as

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'conversation/:type', component: ConversationComponent },
  // otherwise redirect to home
  { path: '**', redirectTo: '' }
]
```

This module defines the different components of the application, the imports and routes. The routes json object declares the URLs internal to the angular app and which components is supporting it. So when /conversation will be reached the ConversationComponent will be loaded. The `:type` is used to control the different user interface for the chatbot.
A home component serves the body content of the main page. It, simply, displays a link to the conversation user interface. The conversation could have been integrated directly into the home page.

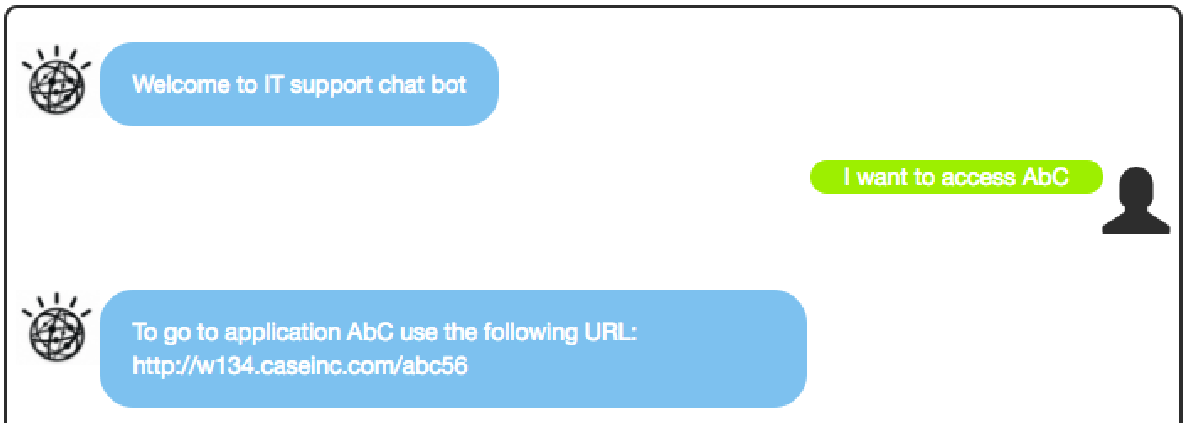
```
<div class="col-md-4 roundRect" style="box-shadow: 10px 10px 5px #10d1d1;
border-color: #10d1d1;">
      <h2>Conversation</h2>
      <p>Start a simple IT support problem shooting conversation</p>
      <p><button (click)="conversation()" class="btn">Ask
Support</button></p>
</div>
```

So the last important component is app/conv/conversation which defines a simple view which combines a scrolling area where the dialog will be displayed and an input field to let the user entering his request or answer.

The presented *div* part lists the text from an array of existing sentences, as we want to present the complete conversation history. The syntax is using Angular 2 `for loop` to get each element of the array `currentDialog` and then add the text of this object in a div.

```
<div *ngFor="let p of currentDialog">
   <div class="message-box">
     <div class="{{p.direction}}">
        <div class="{{p.direction+'-icon'}}" >
          <span *ngIf="p.direction === 'to-watson'" class="glyphicon
glyphicon-user" aria-hidden="true"></span>
          <div *ngIf="p.direction === 'from-watson'" class="from-watson-
icon">
            <img src='assets/images/watson-globe.png' style='width:50px'>
          </div>
        </div>
        <div class="{{p.direction+'-text'}}" [innerHTML]="p.text">
        </div>
      </div>
  </div>
```

Th `*ngIf` is an Angular 2 construct to apply condition on the html element it controls. Here the approach is to present different css class depending of the interaction direction:



Welcome to IT support chat bot

I want to access AbC

To go to application AbC use the following URL:
http://w134.caseinc.com/abc56

The conversation component uses the constructor to do a first call to Watson Conversation so the greetings intent is processed and Watson starts the dialog. If we do not use this technic, the user has to start the conversation, and most likely the greeting will not happen as a user will not start by a "hello" query but directly by his/her request.

```
export class ConversationComponent {


  constructor(private convService : Conversation Service, private route: ActivatedRoute){
    // depending of the url parameters the layout can be simple or more demo oriented with instruction in html
    this.type=this.route.snapshot.params['type'];
    // Uncomment this line if you do not have a conversation_start trigger in a node of your dialog
    this.callConversationBFF("Hello");
  }
}
```

When the user click to the **Send** button the following code is executed to create a Sentence and then delegates to a second method

```
// method called from html button
submit(){
  let obj:Sentence = new Sentence();
  obj.direction="to-watson";
  obj.text=this.queryString;
  this.currentDialog.push(obj);
  this.callConversationBFF(this.queryString);
  this.queryString="";
}
```

The second function uses the conversation service to send the user input, and waits (via subscribe to promise) to get Watson response as calls are asynchronous.

```
callConversationBFF(msg:string) {
  this.convService.submitMessage(msg,this.context).subscribe(
    data => {console.log(data)
```

```
      this.context=data.context;
      let s:Sentence = new Sentence();
      s.direction="from-watson";
      s.text=data.text;
      this.currentDialog.push(s)
    },
    error => {
      return "Error occurs in conversation processing"
      }
  )
}
```
The conversation component needs to have the currentDialog, and as it is a sentence we need to add a Sentence.ts under the conv folder. Sentence is a basic class with a text and a direction. The direction will be used to present who is speaking: to-watson when the user chat, from-watson when Watson answer.

The Context variable is used to keep conversation context between each interaction, this is an important mechanism in Watson conversation to exchange data between application and dialog flow. It is detailed in *Using the context* section of the tutorial

The conversation.service.ts defines the method to do the HTTP request to the backend for frontend server running in nodejs as presented in previous section.

```
submitMessage(msg:string,ctx:any): Observable<any>{
  let bodyString = JSON.stringify(  { text:msg,context:ctx });
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers })
  return this.http.post(this.convUrl,bodyString,options)
       .map((res:Response) => res.json())
}
```
The HTTP module is injected via the constructor and the submitMessage use the HTTP module with the post operation. As the call is asynchronous, we want the function to return a promise. To do so we need to import `rxjs` module and use its `map` function. The method declares the message and the context parameters. The context variable is here to keep the Watson conversation context so it can be sent back to the service so dialog context is kept. We need to propagate to the client as the conversation is with a unique client, where the server is serving multiple web browser.

Link to your Watson Conversation service

You need to create a Watson Conversation Service in IBM Bluemix, get the credential and update the file config/config.json with your own credential:

```
{
    "conversation" :{
      "version":"2017-02-03",
      "username":"",
      "password":"",
      "workspace1":"",
      "conversationId":"",
      "usePersistence": false
    }
}
```
If you want to use persistence, you need to create a Cloudant Service and define the credential in the same file.

Exposed REST APIs

The exposed API is:

```
title: conversation-broker
case:
 - operations:
     - verb: post
       path: /api/conversation/base
       data: {context:{}, text: "a question"}
```
The body should content at least the {text: message} json object. The context object is optional, it will be added with the Watson Conversation ID reference in the code on the first call to the service.

Build and Deploy

You can clone the repository, and uses the following commands:

```
# Install all the dependencies defined in the package.json
$ npm install
# Install angular Command Line Interface for compiling code
$ sudo npm install -g @angular/cli
# the previous commands are to prepare the environment

# to compile the angular code
$ ng build
# to test the server with mocha
$ npm test
```
Execute locally

```
# execute with a build of angular code and start the server in monitoring
mode,
# so change to server restarts the server.
```

To use the API, you need the service credentials and the tool to complete an HTTP request. For instructions, see [Use Watson Assistant API](#).