Advanced Practical Course – Developing innovative services at the example of SAP technologies  
(IN2128, IN2106)  
  
  
  
  
Documentation  
  
Winter Semester 2021/2022  
  
  
  
Ibsolution: SAP Conversational AI  
  
SAP Chatbot Team  
  
  
  
  
  
Marek Szakacs   
Sükrü Can Gültop  
Hongkuan Zhou  
Eric Nguyen

# Table of Contents

[Table of Contents I](#_Toc95404380)

[List of Figures II](#_Toc95404381)

[List of Tables III](#_Toc95404382)

[List of Abbreviations IV](#_Toc95404383)

[1 Environment Setup and Deployment 5](#_Toc95404384)

[1.1 GIT installation 5](#_Toc95404385)

[1.2 Node.js installation 5](#_Toc95404386)

[1.3 Cloud Foundry command line interface installation 5](#_Toc95404387)

[1.4 Creating a CAP application 5](#_Toc95404388)

[1.5 Add a service model to the application 7](#_Toc95404389)

[1.6 Prepare the project for build and deployment 8](#_Toc95404390)

[1.7 Build and deploy CAP application to SAP BTP 9](#_Toc95404391)

[2 Connecting to the external SAP BTP Destination 12](#_Toc95404392)

[2.1 Importing SalesOrder Odata service 12](#_Toc95404393)

[2.2 Modifiying our example service to use newly imported API specification 13](#_Toc95404394)

[2.3 Define Destination service 14](#_Toc95404395)

[2.4 Test the destination by deplyoing the application to Cloud Foundry 15](#_Toc95404396)

[3 Binding APIs to CAI Chatbot 17](#_Toc95404397)

[3.1 Log in to the platform 17](#_Toc95404398)

[3.2 Create your Chatbot 17](#_Toc95404399)

[3.3 Creating Sales Order Intent 19](#_Toc95404400)

[3.4 Create Sales Order ID Entity 20](#_Toc95404401)

[3.5 Create New Skill 22](#_Toc95404402)

[3.6 Test your Skill 26](#_Toc95404403)

[4 Outline level no. 1 29](#_Toc95404404)

[4.1 Outline level no. 2 29](#_Toc95404405)

[4.1.1 Outline level no. 3 29](#_Toc95404406)

[4.1.2 Outline level no. 3 29](#_Toc95404407)

[5 Bibliography 31](#_Toc95404408)

[6 Appendix 32](#_Toc95404409)

# List of Figures

[Figure 1: Exemplary figure 1](#_Toc425252393)

[Figure 2: Integrierte Informationssystem-Architektur (ISA) 3](#_Toc425252394)

# List of Tables

[Table 1: Exemplary table 2](#_Toc425252385)

# List of Abbreviations

AK1 abbreviation 1

AK2 abbreviation 2

AK3 abbreviation 3

… …

# Environment Setup and Deployment

## GIT installation

This project uses Git as a source control system.

1. Check whether git is already installed.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | git version | |  |

1. The output should look like:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | git version 2.x.x | |  |

If not, navigate to the [Git webpage](https://git-scm.com/downloads) and follow installation instructions for your specific system.

## Node.js installation

Backend application is based on CAP (SAP Cloud Application Programming Model). There are two runtimes a developer can choose, Node.js and Java. We will be using Node.js throughout our project.

1. Check whether Node.js version >= 16 is already installed.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | node -v | |  |

1. The output should look like:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | v16.x.x | |  |

If not or the version number <= 16, navigate to the [Node.js webpage](https://nodejs.org/en/download/) and follow installation instructions for your specific system.

## Cloud Foundry command line interface installation

CF CLI is used to deploy and manage your apps.

1. On macOS:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | brew install cloudfoundry/tap/cf-cli | |  |

For Windows, download a binary installer as described [here](https://github.com/cloudfoundry/cli#installers-and-compressed-binaries).

For Linux, follow the steps described [here](https://docs.cloudfoundry.org/cf-cli/install-go-cli.html#pkg-linux).

1. Check if installation was successful. Based on your operating system the command might be different. For example, in Windows with Cloud Foundry CLI version 8, use cf8 --version.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cf --version | |  |

## Creating a CAP application

If the application already exists, clone the repository into your chosen folder and continue with [step 7](#step7). Otherwise follow the tutorial. For overview, you can preview also the intermediate steps.

1. Navigate to your chosen directory and create an initial CAP project. The directory name cannot contain invalid characters such as spaces.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds init | |  |

You should see something like this.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | [init] – creating new project in current folder | |  |
| 2 | | [init] – adding feature ‘nodejs’ … | |  |
| 3 | | [init] – done. | |  |
| 4 | | . . . | |  |

1. Open the generated project in your preferred editor. We will be using Microsoft Visual Studio Code since it provides useful SAP extensions and syntax highlighting. For installation, visit the webpage [here](https://code.visualstudio.com/Download). The generated code structure in VS Code looks like this:

Graphical user interface, application

Description automatically generated

1. When developing with VS Code, go to extensions and search for SAP CDS Language Support. Install the extension to enable syntax highlighting, code completion and more.

A screenshot of a computer

Description automatically generated with medium confidence

1. Run the following command in the terminal from the project directory to install necessary packages.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | npm install | |  |

1. To verify that everything went correctly, start a CAP server from the project directory using following command in the terminal:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds watch | |  |

You should see the following:

Text

Description automatically generated

There are for now no defined models that the server could serve. You can shut down the server by pressing Command+C/Ctrl+C based on your OS.

## Add a service model to the application

We are going to define an example service model which will serve as a basis for our CAP application that we will deploy in the next step.

1. Create a file named *example-model.cds* and place it into *db* folder. Paste the following content into the file.

namespace my.sales;

entity SaleOrders {

key ID : Integer;

title : localized String;

productManager : Association to ProductManagers;

stock : Integer;

shippingDate : Date;

}

entity ProductManagers {

key ID : Integer;

name : String;

saleOrders : Association to many SaleOrders on saleOrders.productManager = $self;

}

1. Create a file named *example-service.cds* and place it into *srv* folder. Paste the following content into the file.

using my.sales as my from '../db/example-model';

service ExampleService {

entity SaleOrders @readonly as projection on my.SaleOrders;

entity ProductManagers @readonly as projection on my.ProductManagers;

}

1. Run the server with:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds watch | |  |

1. Navigate *to* [*http://localhost:4004/example/SaleOrders*](http://localhost:4004/example/SaleOrders). You should see following screen. There is no data yet, but the GET request was executed successfully.

Graphical user interface, text, application

Description automatically generated

## Prepare the project for build and deployment

We need to specify deployment options for an MTA application.

1. Declare which Node.js version should be used in the Cloud Foundry in the *package.json*.

|  |  |  |  |
| --- | --- | --- | --- |
| package.json | |  |  |
| 1 | | { | |  |
| 2 | | ... | |  |
| 3 | | “devDependencies”: { | |  |
| 4 | | ... | |  |
| 5 | | }, | |  |
| 6 | | “engines”: { | |  |
| 7 | | “node”: “>=14” | |  |
| 8 | | }, | |  |

1. Generate MTA deployment descriptor (*mta.yaml*) file. Run the following command from the directory of your project.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds add mta | |  |

1. Specify disk and memory quota in the *mta.yaml* file based on your available resources in the Cloud Foundry space.

|  |  |  |  |
| --- | --- | --- | --- |
| mta.yaml | |  |  |
| 1 | | modules: | |  |
| 2 | | - name: sap\_chatbot-srv | |  |
| 3 | | type: nodejs | |  |
| 4 | | path: gen/srv | |  |
| 5 | | parameters: | |  |
| 6 | | buildpack: nodejs\_buildpack | |  |
| 7 | | disk-quota: 512M | |  |
| 8 | | memory: 512M | |  |
| 9 | | provides: | |  |
| 10 | | . . . | |  |

## Build and deploy CAP application to SAP BTP

To deploy a CAP application, you will need a SAP BTP global account, a subaccount, and a Cloud Foundry space with the required entitlements. Follow this [tutorial](https://developers.sap.com/tutorials/btp-app-prepare-btp.html) if you don’t have these accounts yet.

1. Find the API Endpoint of your Cloud Foundry subaccount environment.

Graphical user interface, application

Description automatically generated

1. Set the Cloud Foundry API endpoint.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cf api <API endpoint> | |  |

1. Log into the Cloud Foundry account using SAP BTP credentials.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cf login | |  |

If you are getting an Unauthorized error and you typed your credentials correctly, log in via SSO.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cf login --sso | |  |

1. Check if Cloud MTA Build Tool is already installed.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | mbt --version | |  |

If not, install it with the following command:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | npm install --global mbt | |  |

1. On Windows, install *make* from <http://gnuwin32.sourceforge.net/packages/make.htm>. Choose the download **Complete package, except sources**. Run the installer, find the location where it was installed and add the folder *GnuWin32\bin* to the *Path* environment user variable.
2. Install *multiapps* plugin that is necessary for deploying a CAP application.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cf install-plugin multiapps | |  |

1. Build the MTA module from the project root folder.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | mbt build -t ./ | |  |

1. This generated *.mtar* file in the project root folder. Deploy the module to the Cloud Foundry space using:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cf deploy <mtar file> | |  |

If the deployment was successful, you should get a link which points to your CAP application. After opening the link, you should see the following screen:



You can see two service endpoints. The service that we defined as *ExampleService* and *$metadata* service that gets added by default.

If you face an issue regarding deployment saying “Service operation failed: Controller operation failed: 502 Updating service” it may mean you have too many service instances. You can delete the unused ones following [these steps](https://help.sap.com/viewer/09cc82baadc542a688176dce601398de/Cloud/en-US/753463e1542f445895b420cd7957811c.html).

# Connecting to the external SAP BTP Destination

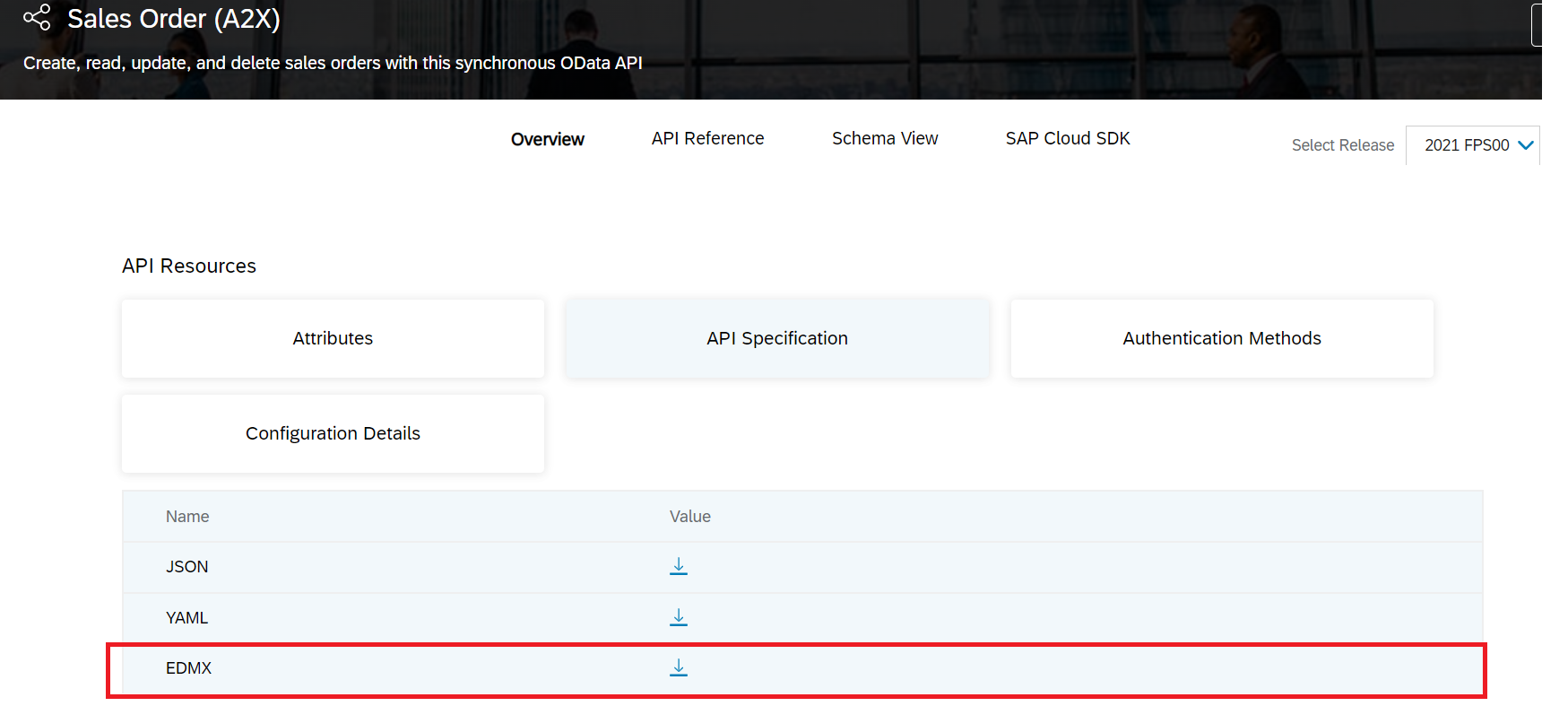
**Prerequisites**: completed Environment Setup tutorial

**Result**: Configured destination service that consumes external service and returns SalesOrder data.

## Importing SalesOrder Odata service

To define a SalesOrder model, we can download and import API reference from SAP API Business Hub to our CAP application.

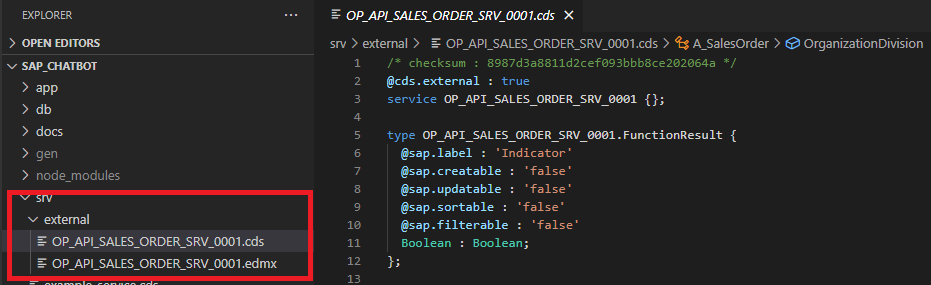
1. Download API specification of SalesOrder OData in EDMX format from *https://api.sap.com/api/OP\_API\_SALES\_ORDER\_SRV\_0001/overview*



1. Move the downloaded file to the project root folder and run the following command to import API specification.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds import <filename>.edmx | |  |

This should have created a new folder called *external* with two files as seen in the screenshot.



## Modifiying our example service to use newly imported API specification

We are going to use the newly imported API to define a projection on the SalesOrder data.

1. Replace the content of the example-service.cds file to define a service endpoint/entity *TestSaleOrders* with the projection of the imported SalesOrder data.

using OP\_API\_SALES\_ORDER\_SRV\_0001.A\_SalesOrder as salesorder

from './external/OP\_API\_SALES\_ORDER\_SRV\_0001';

service ExampleService {

@readonly

entity TestSaleOrders as projection on salesorder {

key SalesOrder, LastChangeDate, CreationDate, TotalNetAmount

};

}

1. Create a new *example-service.js* Javascript handler. The name must match the name of the cds service so that CAP knows that it should invoke methods/routines when the *TestSaleOrders* entity is being read/written/updated.

const cds = require('@sap/cds');

module.exports = cds.service.impl(async function() {

const service = await cds.connect.to('OP\_API\_SALES\_ORDER\_SRV\_0001'); // connect to the external destination

const { TestSaleOrders } = this.entities;

this.before('\*', (req) => {

console.debug('>>>', req.method, req.target.name)

});

this.on('READ', TestSaleOrders, request => {

return service.tx(request).run(request.query);

});

});

The code gets called once the TestSaleOrders entity is being requested using a GET request. It connects to the BTP destination and returns all results in *this.on()* part.

1. Since CAP does not support ODataV2 by default, we need to use middleware to enable it.

Add the middleware package to the project by running:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | npm add @sap/cds-odata-v2-adapter-proxy | |  |

Create a new file *server.js* under *srv* folder with:

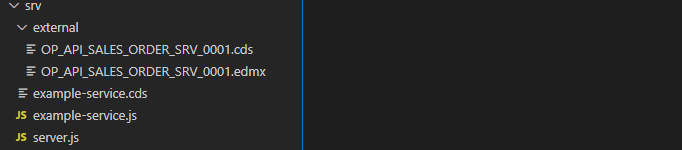
const proxy = require('@sap/cds-odata-v2-adapter-proxy')

const cds = require('@sap/cds')

cds.on('bootstrap', app => app.use(proxy()))

module.exports = cds.server

The *srv* folder structure after step 2 should look like this:



## Define Destination service

The destination service can be created either from the BTP cockpit or in the *mta.yaml* file. We are going to use the *mta.yaml* approach based on which CAP will create the services upon deployment of the application.

1. Generate security descriptor file *xs-security.json* using the command below:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds compile srv/ --to xsuaa > xs.security.json | |  |

1. Define the destination service with provided credentials and properties. Since we are connecting to an OnPremise destination, we need to additionally define a connectivity service. In the end, add the destination service, connectivity service and authentication service to the *“requires”* part of the *mta.yaml* file. The file after the *“requires”* part should look something like this:

...

requires:

- name: my-destination-service

- name: sales-xsuaa

- name: my-connectivity-service

resources:

- name: my-destination-service

type: org.cloudfoundry.managed-service

parameters:

config:

HTML5Runtime\_enabled: true

version: 1.0.0

init\_data:

instance:

destinations:

- Authentication: BasicAuthentication

Name: ERP\_IDA\_SO\_SRV

ProxyType: OnPremise

CloudConnectorLocationId: SCC01

Type: HTTP

URL: <your URL>

User: <your User>

Password: <your Password>

HTML5.ForwardAuthToken: true

service: destination

service-plan: lite

- name: sales-xsuaa

type: org.cloudfoundry.managed-service

parameters:

path: ./xs-security.json

service: xsuaa

service-plan: application

- name: my-connectivity-service

type: org.cloudfoundry.managed-service

parameters:

service-plan: lite

service: connectivity

1. Adjust *package.json* file to make a correspondence between the model specification and the destination.

. . .

},

"rules": {

"no-console": "off",

"require-atomic-updates": "off"

}

},

"cds": {

"requires": {

"OP\_API\_SALES\_ORDER\_SRV\_0001": {

"kind": "odata-v2",

"model": "srv\\external\\OP\_API\_SALES\_ORDER\_SRV\_0001",

"credentials": {

"destination": "ERP\_IDA\_SO\_SRV"

}

}

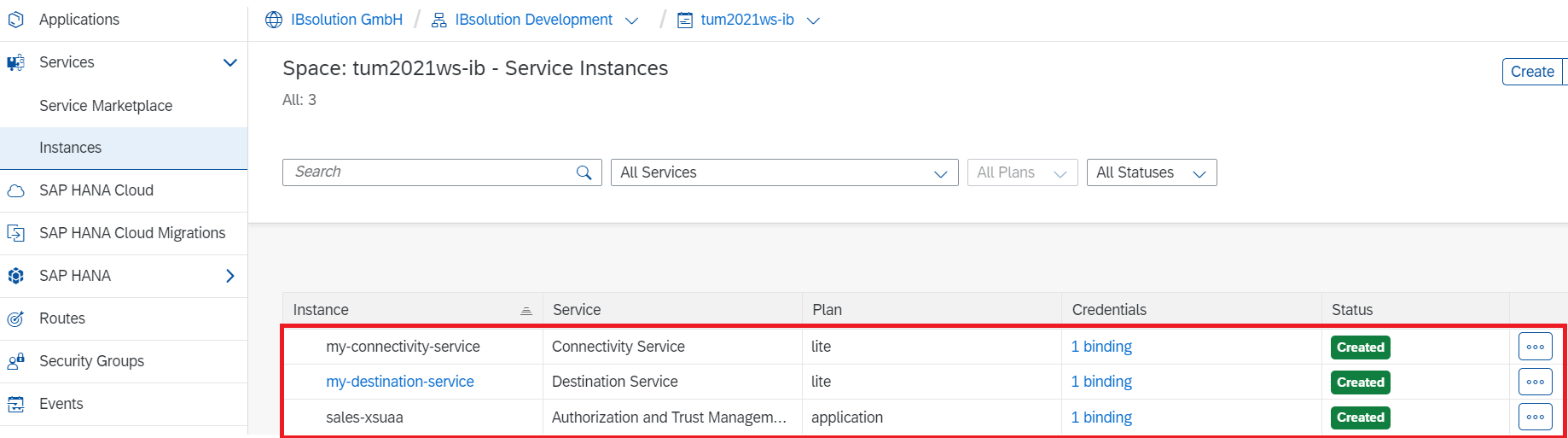
}

}

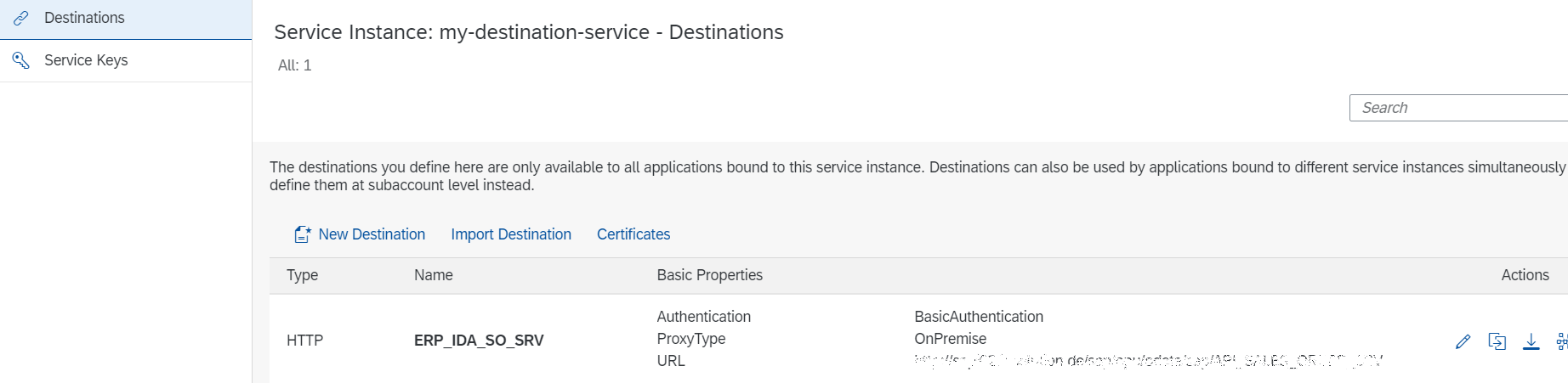
## Test the destination by deplyoing the application to Cloud Foundry

To test the destination service, we need to deploy the application to the Cloud Foundry and check if the GET request for the defined entity endpoint executes successfully.

1. Build and deploy the application to the Cloud Foundry as illustrated in the Environment Setup tutorial. Once the application is running, navigate to your application service instances. You should see following services.



Click on *my-destination-service* and navigate to Destinations, you should see your destination.



1. Navigate to the <url>/example/TestSaleOrders and check if the request executes successfully. If the request returns 401 Forbidden or 502 Bad Gateway code, make sure the credentials are filled correctly. You should get a response in JSON format with TestSaleOrders entities.



# Binding APIs to CAI Chatbot

**Prerequisites**: Running API endpoints, completed previous 2 tutorials

**Result**: Running CAI chatbot with single action connected to API endpoint

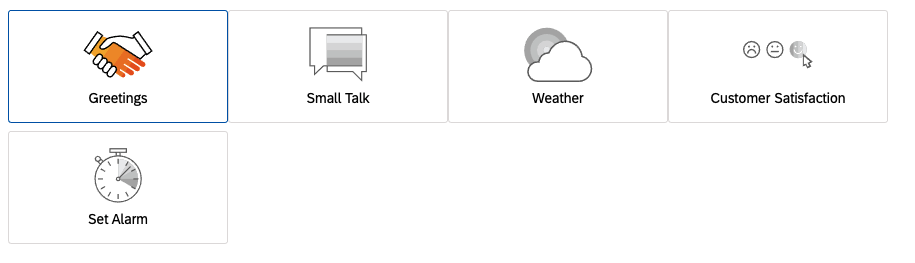
## Log in to the platform

Go to <https://cai.tools.sap/> . Before you use the platform for the first time, you need to sign up and create your user account.

## Create your Chatbot

Click *+ New* Bot at the top right of the page in SAP Conversational AI and create your first chatbot.

1. Select Perform Actions option for chatbotEin Bild, das Text enthält.

   Automatisch generierte Beschreibung
2. Select greetings predefined skill for your bot
3. Create your bot:
   1. Enter a name and, if desired, a description for your bot.
   2. (Optional) Add up to six topics to your bot. By categorizing bot in this way, CAI can suggest more appropriate training data to improve it later on.
   3. Set default language as English

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Select your data policy according to [Bot Data Policy settings.](https://help.sap.com/viewer/3aab817a03be4432abbfd00c42b23cb8/latest/en-US/65a2096b1eea4c48ae3e609d049707a3.html)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Set your bot visibility. And create your bot.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Creating Sales Order Intent

An intent is a collection of sentences that all have the same meaning, even though they can be very different from one another. When a user sends a message to your bot, CAI algorithm predicts the intents to which it’s close enough and decides what the intention of the message is.

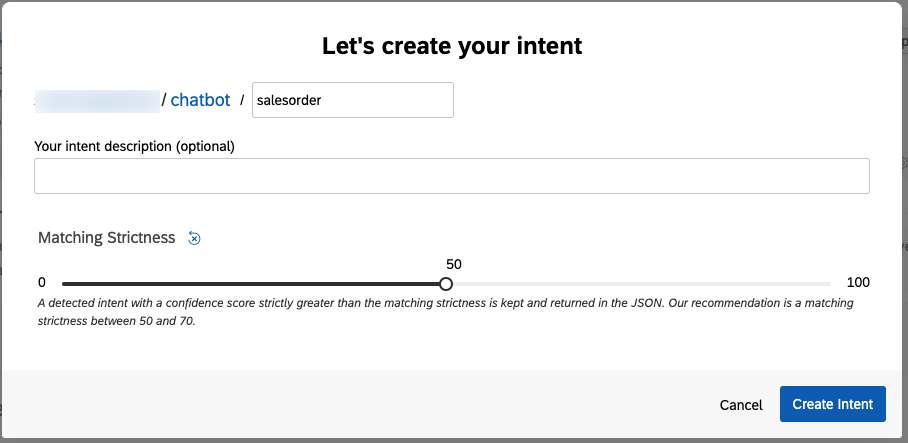
Everything your chatbot understands is in the intents. Each intent corresponds to an action that your user wants to perform. For example, the intent greetings enables your bot to understand when a user says Hello. Explore each intent by clicking the name of the intent (for example, greetings), and you’ll see the expressions inside that train your bot to understand the user’s intent.

1. Click + New Intent button to create a new intent.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Name Your Intent and decide set your matching strictness to 50



1. Click the salesorder intent and add your expressions to train your intent. Try to add more than 50 expressions per intent at least for better training of chatbot.

## Create Sales Order ID Entity

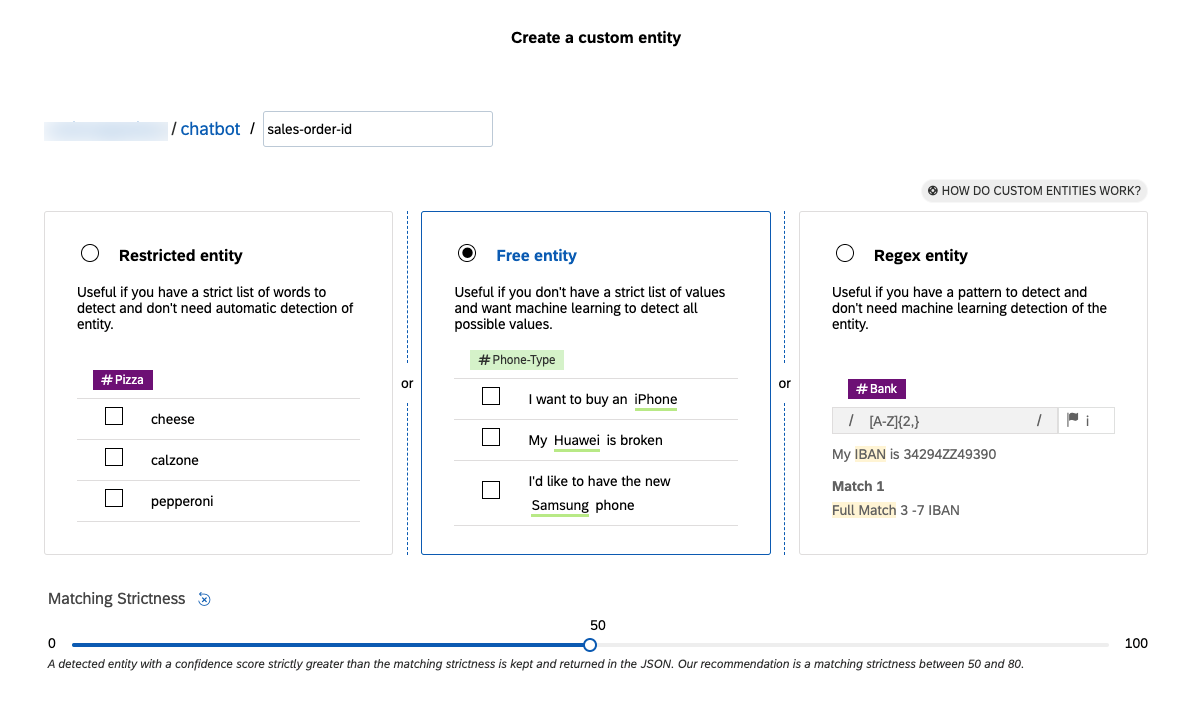
An entity is a keyword that is extracted from an expression. CAI automatically detects [28 different entities](https://help.sap.com/doc/9b639cad3d734675971ab22ed10bbf28/latest/en-US/UserGuideToConceptsOfSAPConversationalAI.pdf#page=55&zoom=100,0,541), such as Datetime, Location, Person, and so on. However, we’re not limited to these gold entities. We can create a custom entity like Sales Order ID.

1. Navigate to entity tab in Train tab.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Create your free entity with sales-order-id name and 50 matching strictness

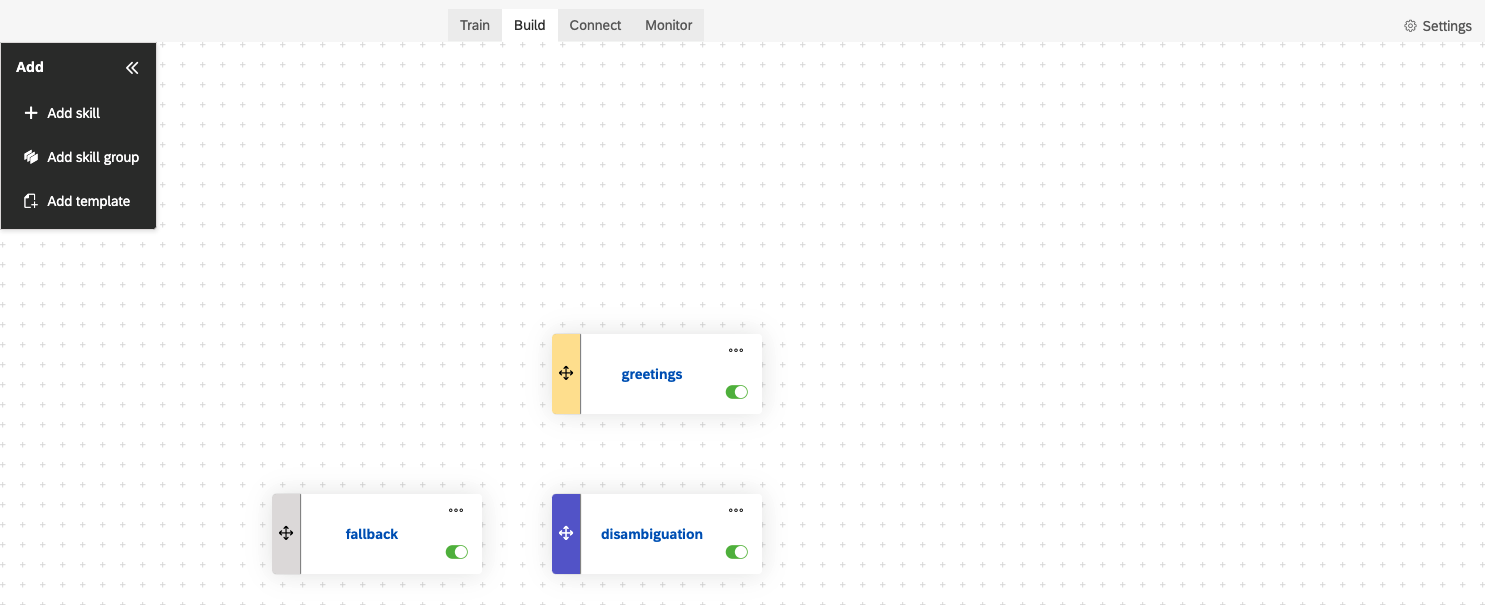


1. Now go back to your salesorder intent and expand your expressions and tag your entities like below.Ein Bild, das Text enthält.

   Automatisch generierte Beschreibung

## Create New Skill

A skill is a block of conversation that has a clear purpose and that your bot can execute to achieve a goal. It can be as simple as the ability to greet someone, but it can also be more complex, like getting details of an order according to information given by user. You can add a skill to your bot on the Build tab by clicking Add skill in the command panel on the left. You can add as many skills to your bot as you wish.

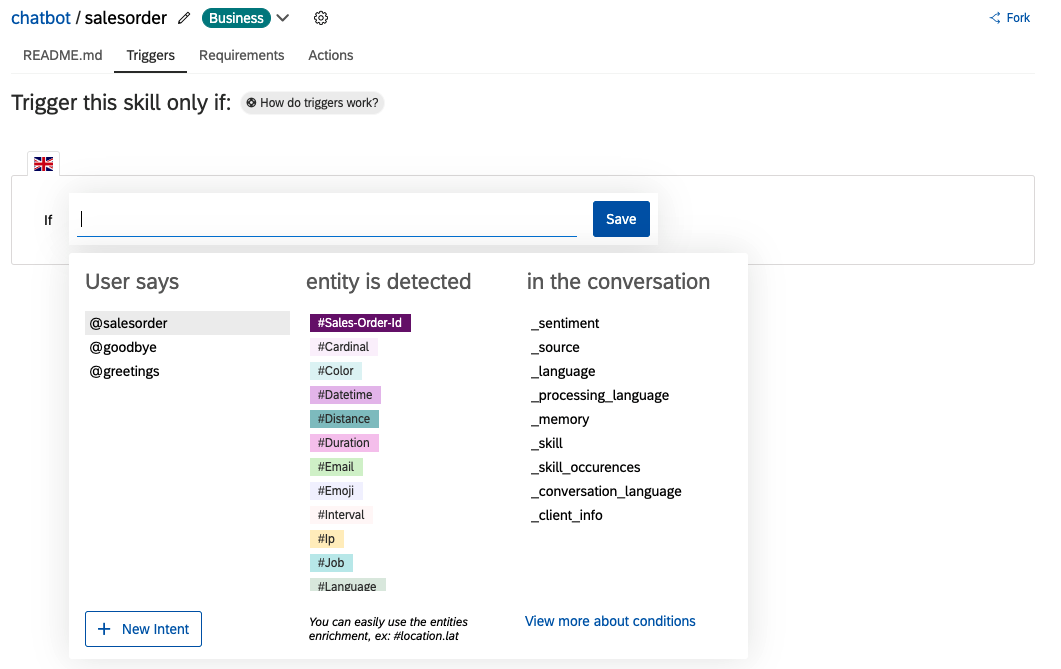


1. Create your business skill named salesorder and open your skill by clicking on it

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Set trigger for your action



Ein Bild, das Text enthält.

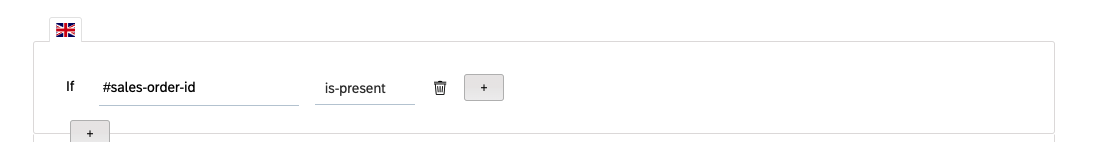
Automatisch generierte Beschreibung

1. Set your requirements for action as below

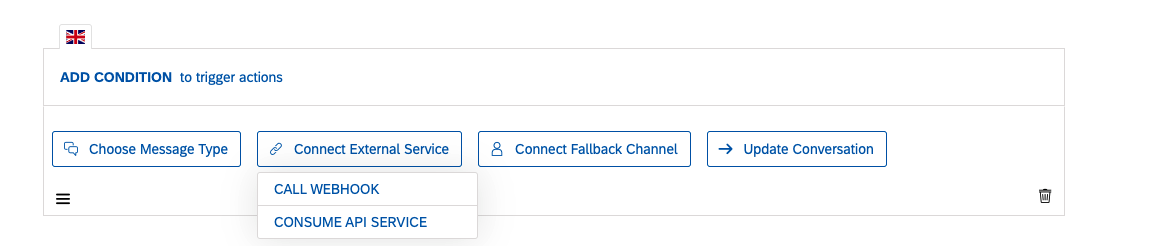
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Add condition for your action



1. Create your action in Actions tab
   1. Select connect external service. We can select either call webhook or consume API service. We will use consume API service since our TestSalesOrder service is only OData endpoint. To use webhooks, we need [special response formats.](https://help.sap.com/viewer/a4522a393d2b4643812b7caadfe90c18/latest/en-US/7f06277acb1d46529daa9c5828a8a045.html)



* 1. Define your API request

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* 1. Select choose message type -> custom and in the message type select card and replace script with below

{

"type": "card",

"delay": "",

"content": {

"title": "SALES ORDER {{memory.salesOrderID.raw}}",

"subtitle": "Type: {{api\_service\_response.default.body.SalesOrderType}} Date: {{api\_service\_response.default.body.CreationDate}}",

"description": "Total Net Amount: {{api\_service\_response.default.body.TotalNetAmount}} {{api\_service\_response.default.body.TransactionCurrency}}",

"status": "{{api\_service\_response.default.body.OverallSDProcessStatus}}",

"statusState": "information"

}

}

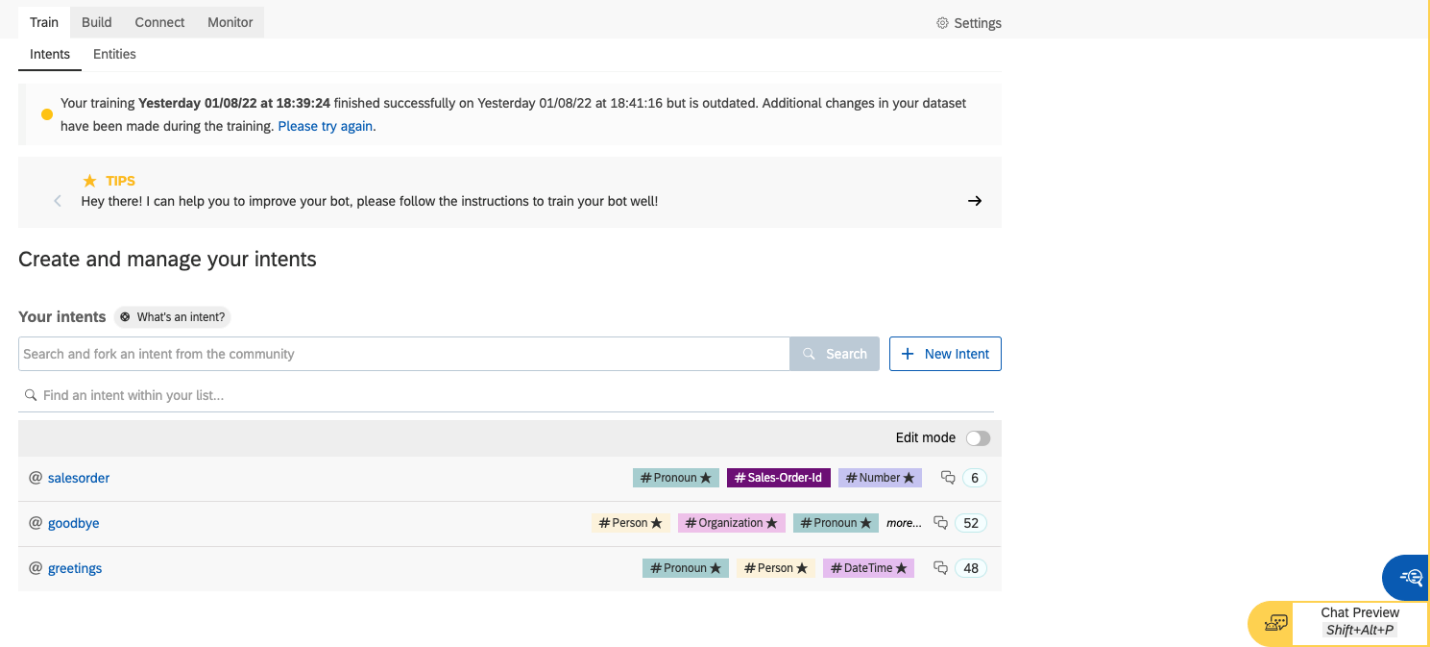
* 1. For the final step select update conversation -> edit memory and delete the salesOrderID.

Ein Bild, das Text enthält.

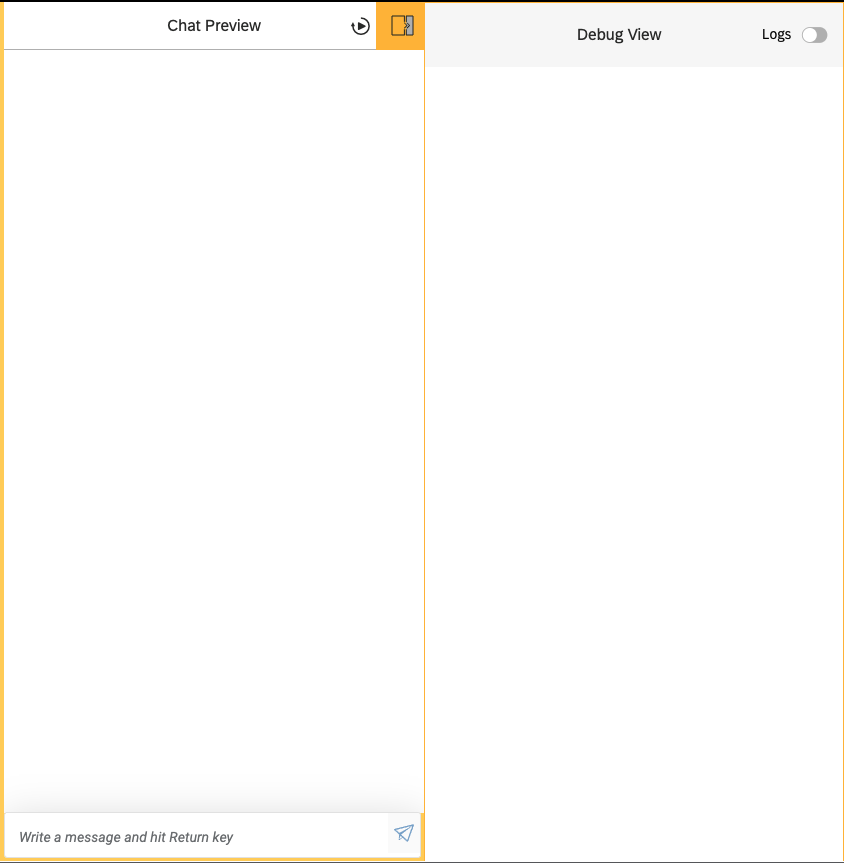
Automatisch generierte Beschreibung

## Test your Skill

You can test your chatbot by clicking chat preview at the bottom right corner of the page.



After opening chat preview you can also open the debug area for details of your chatbot events.



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure

Figure 1: Exemplary figure

Source: own representation

# Outline level no. 1

## Outline level no. 2

A fitting citation to introduce a particular section, which is not part of the following text block.

(source)

### Outline level no. 3

Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. „usual citation within a text block” ([Krcmar, 2005, p. x](#_ENREF_2)).

„indented citation over multiple lines …………………………………………… ………………………………………………………………………………………… ………………………………………………………………………………………… …………………………………………………………“ (source).

Text text text text[[1]](#footnote-1)

Text text text text[[2]](#footnote-2)

* Text\_enumeration;
* Text\_enumeration;
* Text\_enumeration.

Enumerations have to be embedded correctly within a sentence

### Outline level no. 3

Text text text text. Table 1 refers to an exemplary table. Die formatation template for the descriptions are analogous to the desctription of figures „Beschriftung“ and „Beschriftung\_Quelle“.

|  |  |
| --- | --- |
| Table\_Headline | Table\_Headline |
| Table\_Cell | Table\_Cell |
| Table\_Cell | Table\_Cell |
| Table\_Cell | Table\_Cell |

Table 1: Exemplary table

source: …

Text text text text

Text text text text

Headline for usual text block

Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

As an example for a formula Formel 1 is illustrated beneath. Even here the description and ist source is handled the same way as it was done with figures.



Formula 1: Exemplary formula

source: own representation

Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Headline for usual text block

Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

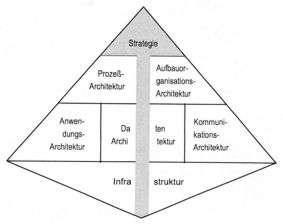


Figure 2: Integrierte Informationssystem-Architektur (ISA)

source: ([Krcmar, 1990, p. 399](#_ENREF_1))

# Bibliography

**Krcmar, H. (1990)**. Bedeutung und Ziele von Informationssystem-Architekturen. *Wirtschaftsinformatik, 32*(5), 395-402.

**Krcmar, H. (2005)**. *Informationsmanagement* (4 ed.). Berlin; Heidelberg; New York: Springer Verlag.

# Appendix

1. Appendix outline level no. 1
   1. Appendix outline level no. 2

Plaease be aware that only the first outline level will be added to the table of contents. The formatting templates for the structure of the appendix are named „Ü1Anhang“ and „Ü2Anhang“.

1. footer line 1 [↑](#footnote-ref-1)
2. footer line 2 [↑](#footnote-ref-2)