**Detailed explanation of the architecture, source code in the ppt link (from github) -** [**https://github.com/DivyaSundar-89/CFC\_2020\_SlotBooking/blob/master/Submission\_IBM\_Call\_for\_Code.pptx**](https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/Submission_IBM_Call_for_Code.pptx)

**Finals video recording link -** [**https://drive.google.com/file/d/1T1rAtFNGvT1gLZBYQPeHlInue7XZi6mB/view?usp=sharing**](https://drive.google.com/file/d/1T1rAtFNGvT1gLZBYQPeHlInue7XZi6mB/view?usp=sharing)

**Use case 1 - iTRAM**

**Web application for vendor and customer registration for inventories**

**Use Case 1 - iTRAM**

**\*\* Pre-requisites for running the project**

1. Download and install an IDE such as Eclipse or Netbeans or IntelliJ.
2. Download and install web container such as tomcat 7.0 for Windows.
3. MYSQL database setup required in windows.
4. Watson assistant to be created and dialog skill to be imported from this project.
5. Watson discovery to be provisioned with the default pre-enriched data set and create a cloud function for the node js script to access the external REST API.

**\*\* Steps to execute the webapp project**

1. Import the project comm in an IDE and execute it in tomcat or, deploy the war file comm.war available in the same path in tomcat webapps.
2. Run the SQL statements from the file Sql\_Statements in MYSQL DB to setup and create the tables and DB required.
3. Launch the application in http://localhost:8080/comm/index.jsp. Please note that vendor registration and customer registration are mandatory before launching the dashboard and order details view.

**\*\* Launch the Spring boot**

1. Launch the jar file cfcrest-0.0.1-SNAPSHOT.jar (java -jar cfcrest-0.0.1-SNAPSHOT.jar) or build the project "demo" through maven (mvn clean install which generates the jar file cfcrest-0.0.1-SNAPSHOT.jar in target), Link from github for the jar file - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/cfcrest-0.0.1-SNAPSHOT.jar>
2. Running the jar will launch the REST API in localhost:8083. The REST API is http://localhost:8083/getslotdetail/${customer\_name} where customer\_name is the path parameter based on which the slot and the vendor id is retrieved and response is sent. Backend project link from github - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/tree/master/demo>

**\*\* Create the required IBM functions**

1. Create a watson assistant chatbot using the procedures given in the link <https://developer.ibm.com/tutorials/create-a-covid-19-chatbot-embedded-on-a-website/>
2. Import the dialog skill given in this project skill-CDC-COVID-FAQ.json (which has custom intents and dialogs to get the slot details booked for the customer) and associate it with the watson assistant. Link of created custom dialg skill from github - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/skill-CDC-COVID-FAQ.json>
3. Create a watson discovery with the pre-enriched data set using the instructions given in the link <https://developer.ibm.com/tutorials/create-a-covid-19-chatbot-embedded-on-a-website/>
4. Create a cloud function and action with runtime as Node 10. Use the script as action code - discovery-covid-async.js in this project which makes the external REST call to fetch the vendor slot id. Github URL - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/discovery-covid-async.js>
5. Enable the cloud function as a web action and make the public URL as a webhook.
6. Customize the dialog "GetSlotBooked" (under CDC-COVID-FAQ) and enable the webhook. The parameter name "cust\_name" is not dynamic from the chat-bot, so add the customer\_name as a parameter in the dialog customization, example: cust\_name key has value Alex (which will be passed as Path parameter to the external REST call).
7. IBM cloudant database is used which stores the details of the record for vendor id, customer id and the slot booked with the vendor. The cloudant database is accessed by the node10 runtime cloud function which gives the response to the watson assistant chat-bot.

**\*\* Testing the chat-bot**

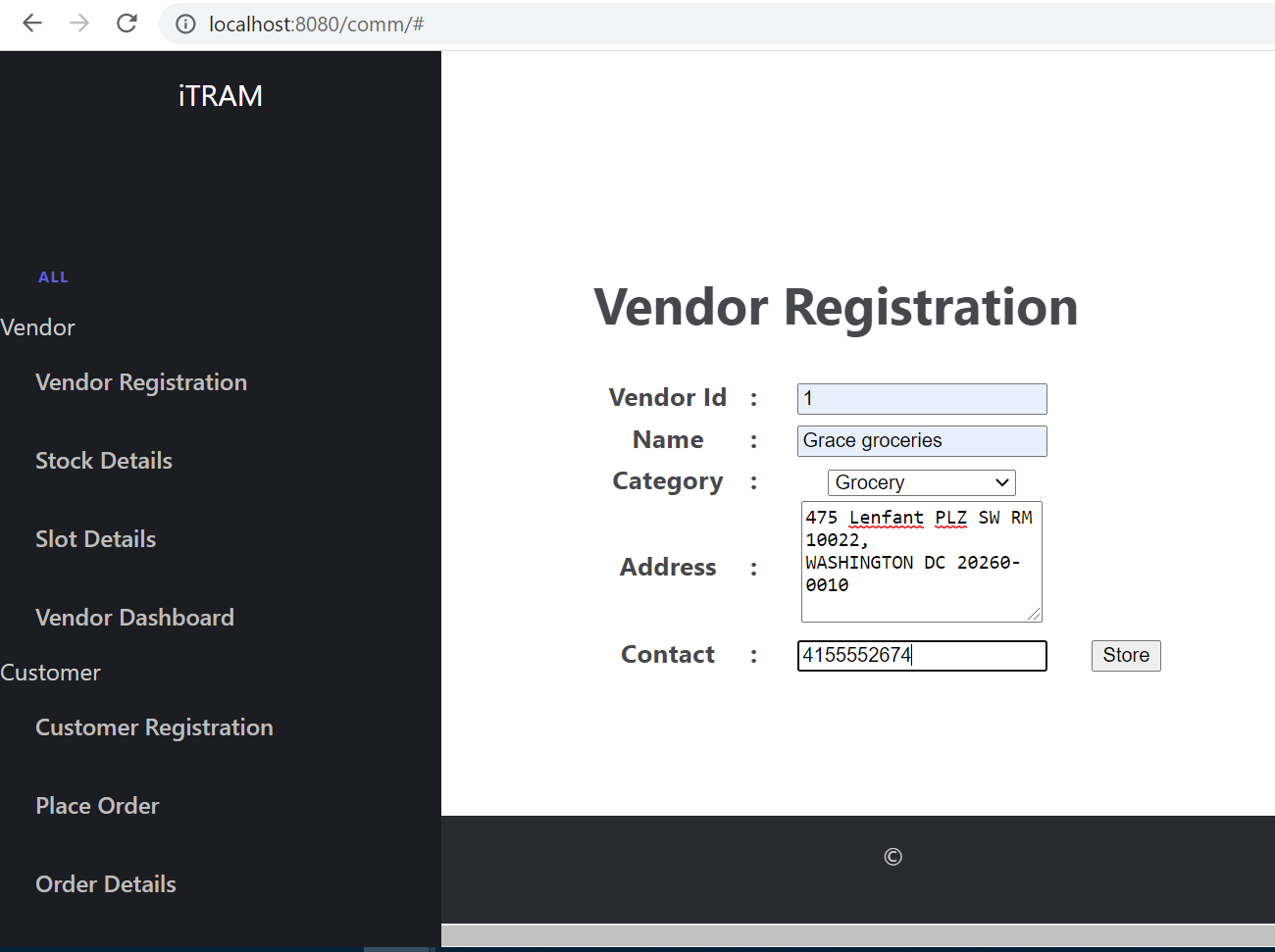
1. Preview the assistant by launching it and type the sample question assuming Alex has logged on in the chat-window - "What is the slot booked with the vendor ?". This would invoke the web-hook and issue REST call externally to get the vendor slot id which is linked to Alex.
2. Previous to this, it is expected that the vendor has registered the stock and inventory details and customer (Alex) registers in the app and books a slot for the necessary inventory. A slot id from the vendor is associated to Alex ensuring no two customers get the same slot id when slot booking is done.

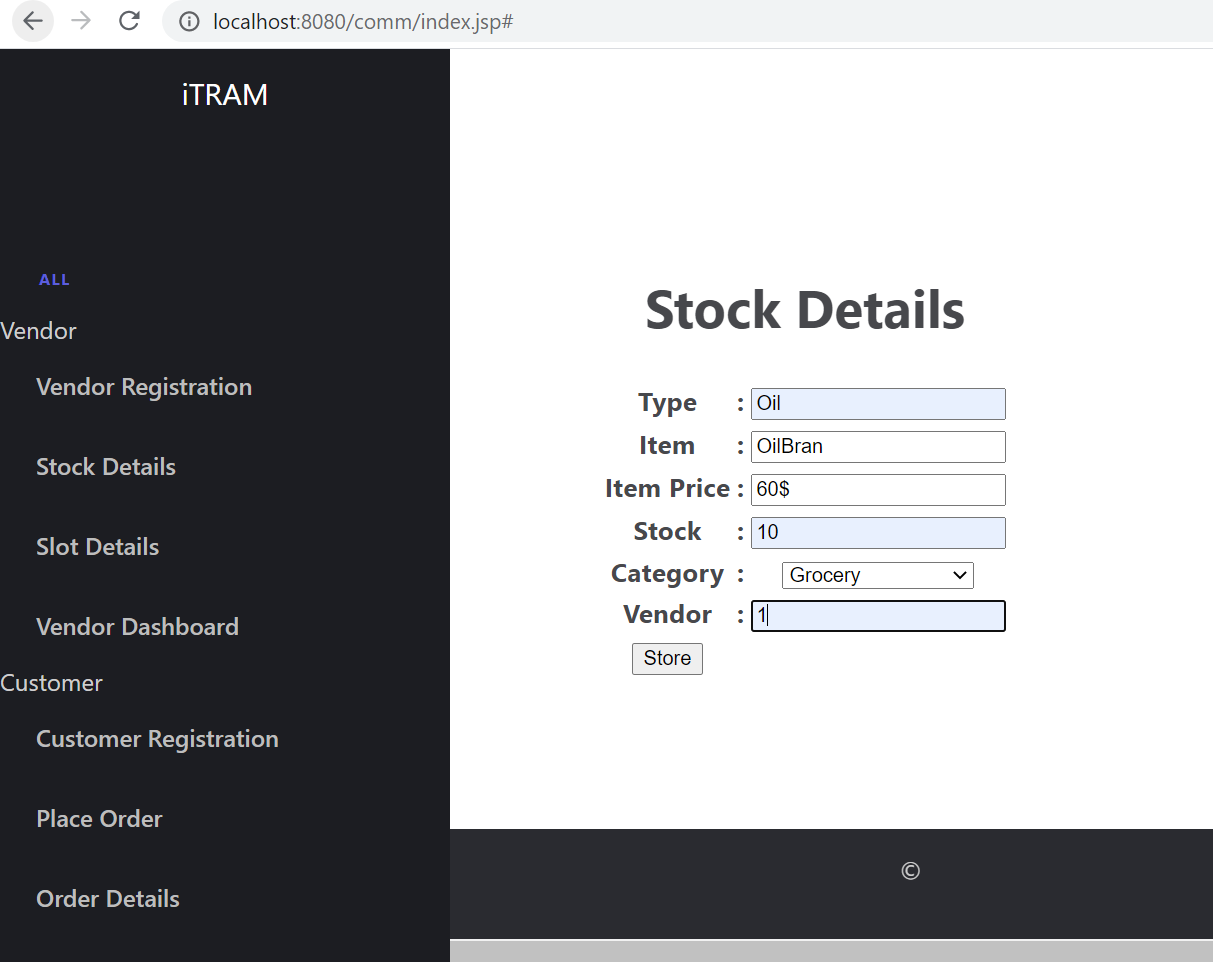
**\*\* Demo recording of use case 1**

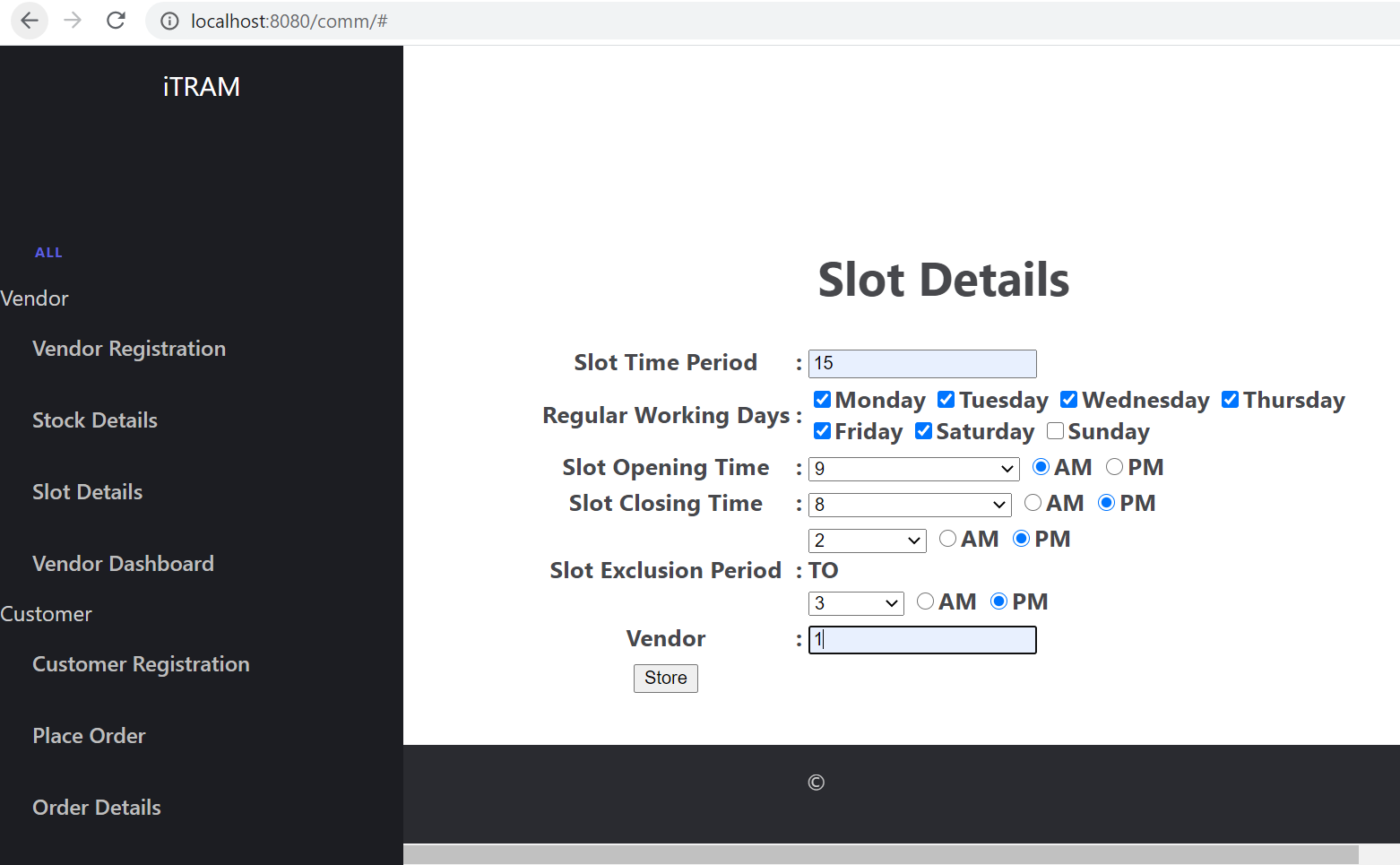
Demo recording video link - <https://www.youtube.com/watch?v=PdwA0tA-LeI&feature=youtu.be>

**\*\* Snapshots of use case 1**

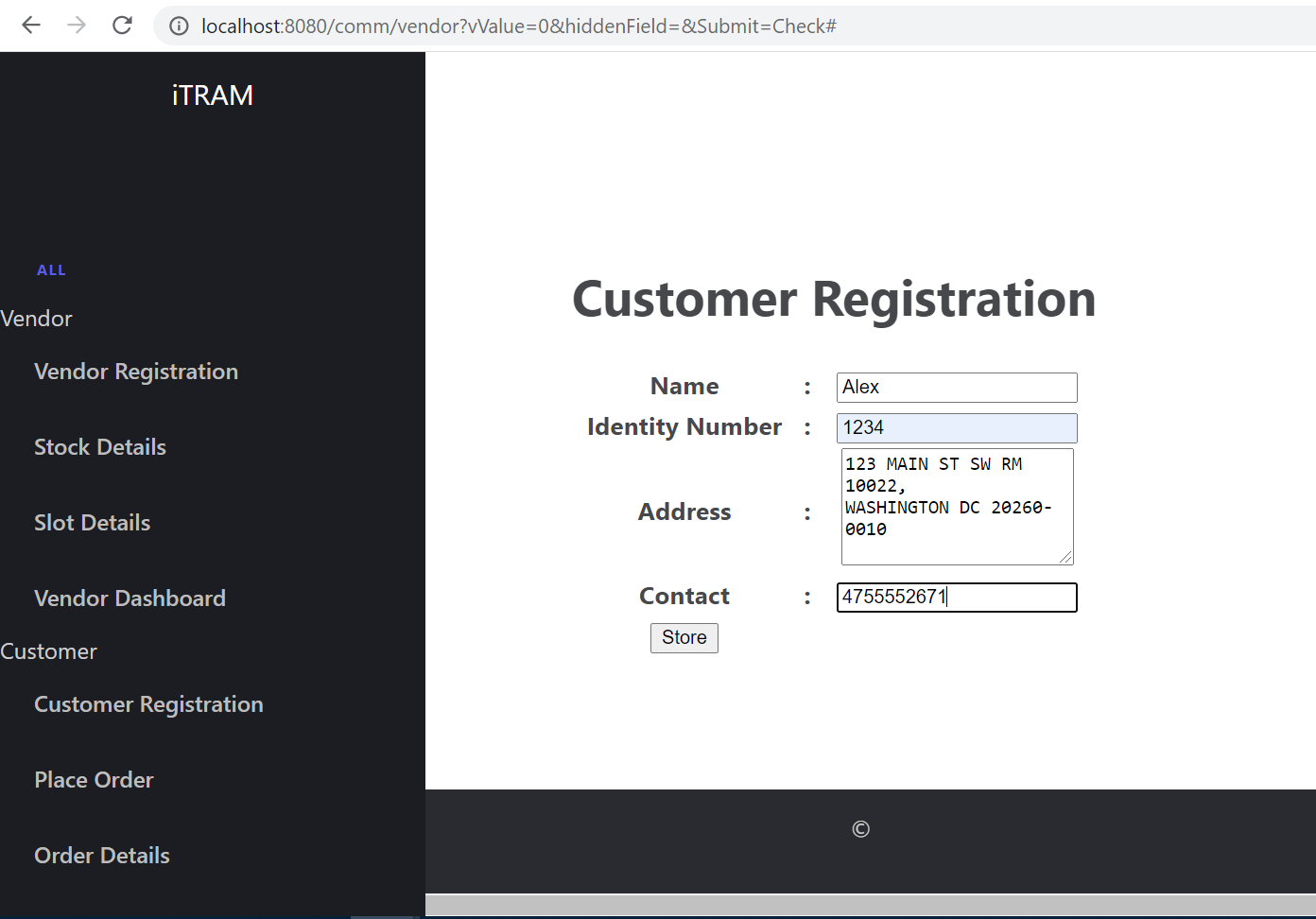
Vendor registers the stock and slot availabilities in the app along-with the location.

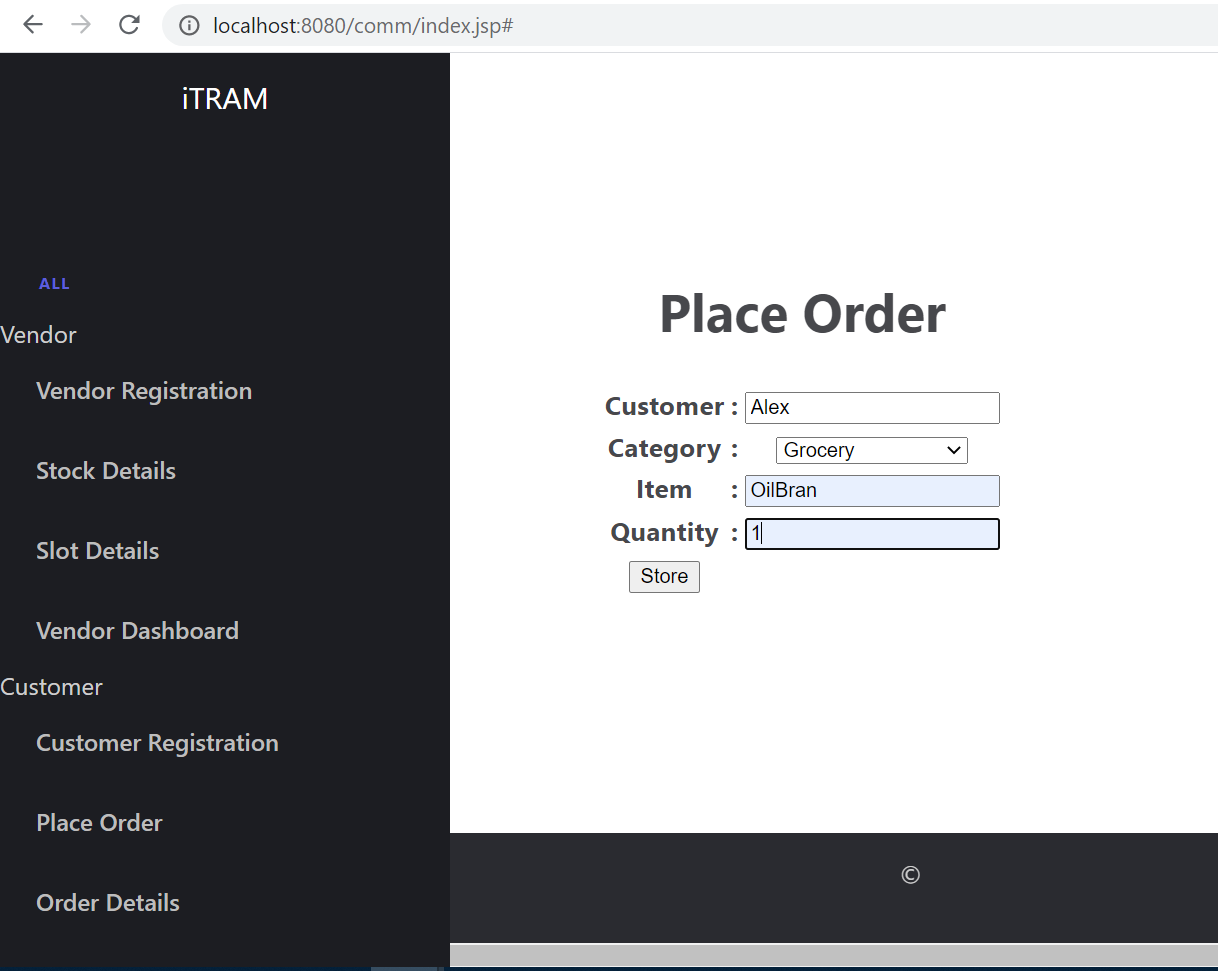




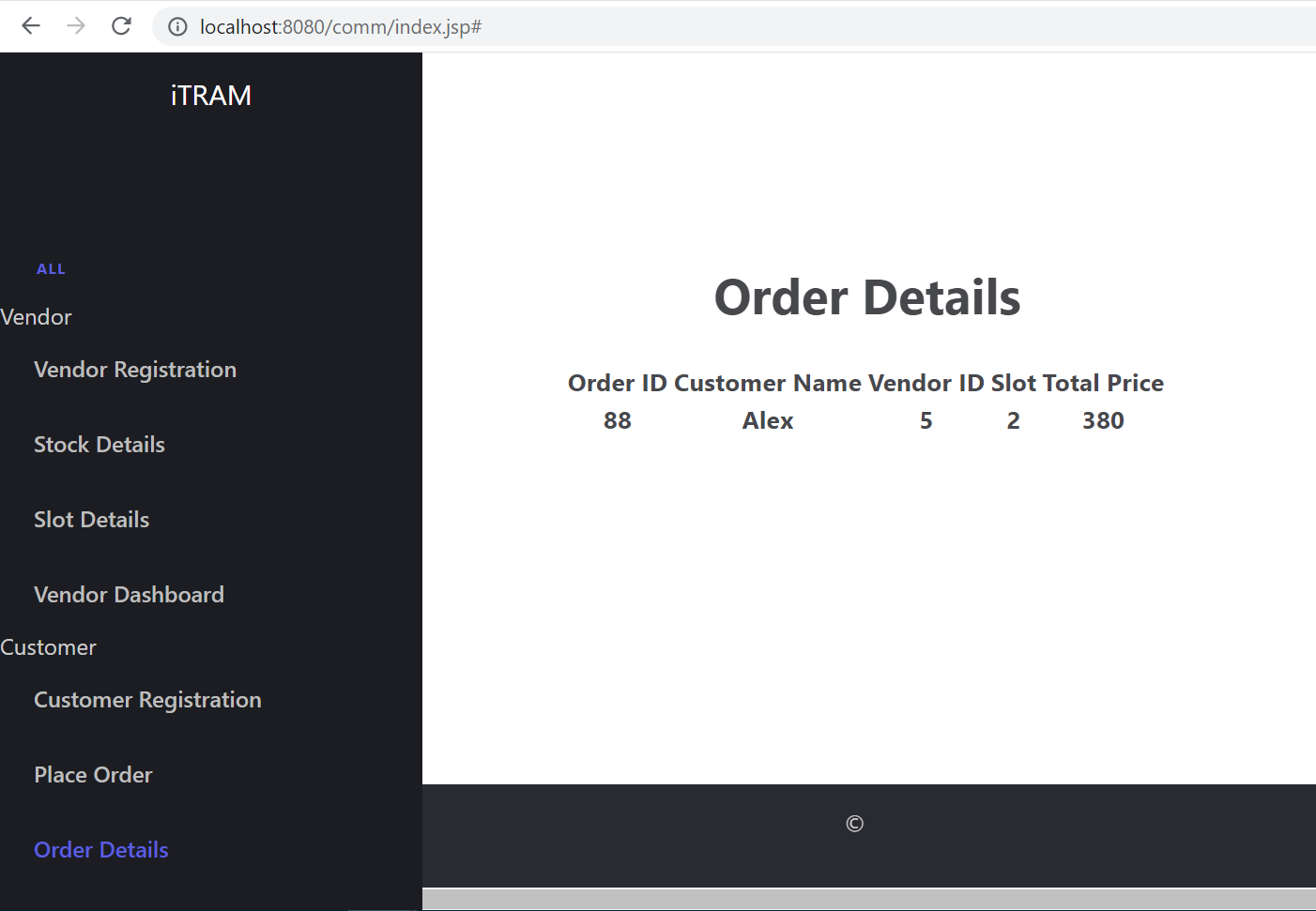


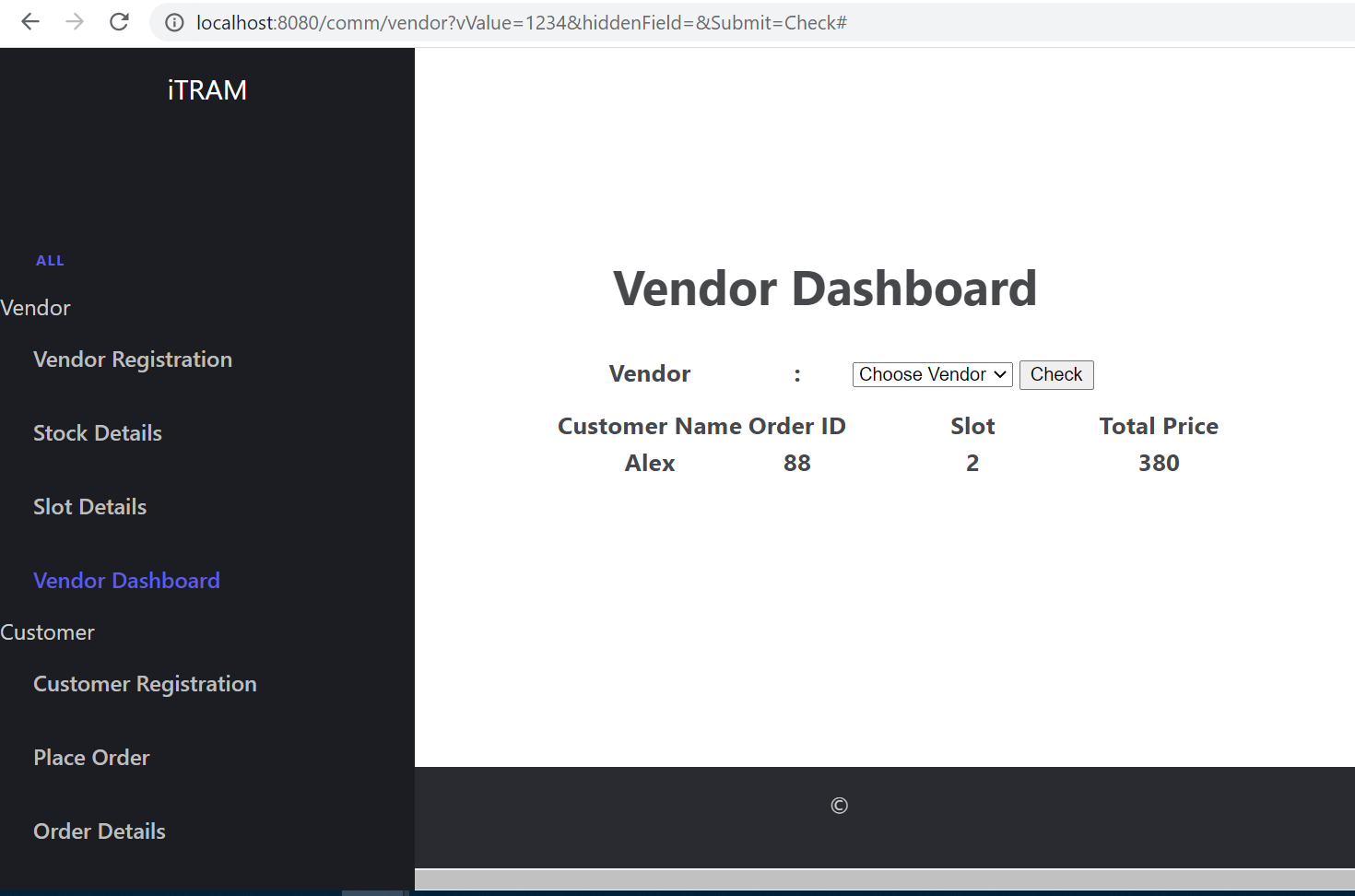
Customer regsiters and places an order in the app for the inventory needed.



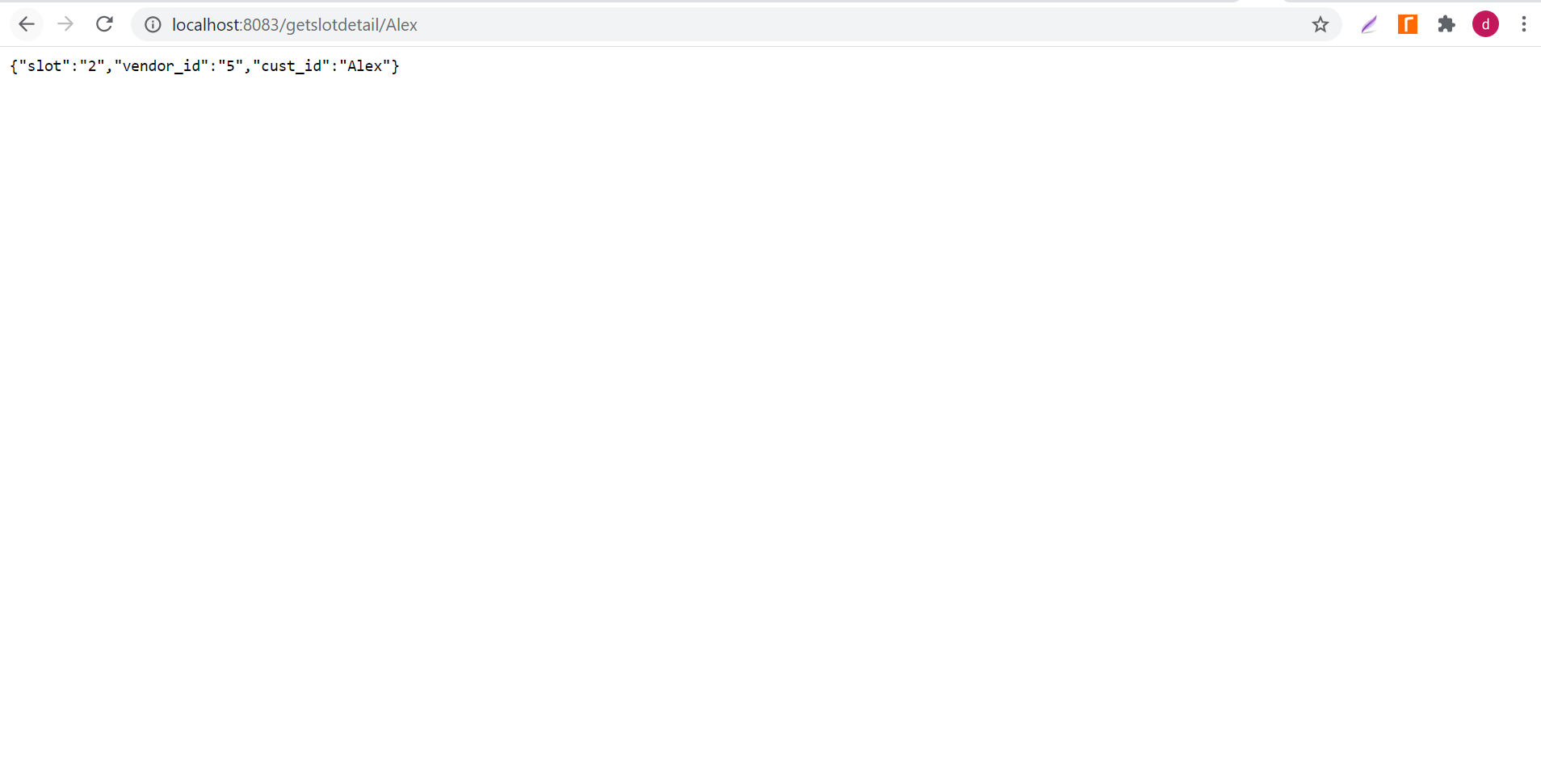


Customer order details are populated by choosing the location of the vendor closer to the customer location and vendor is also able to see the dashboard of the slot availabilities.



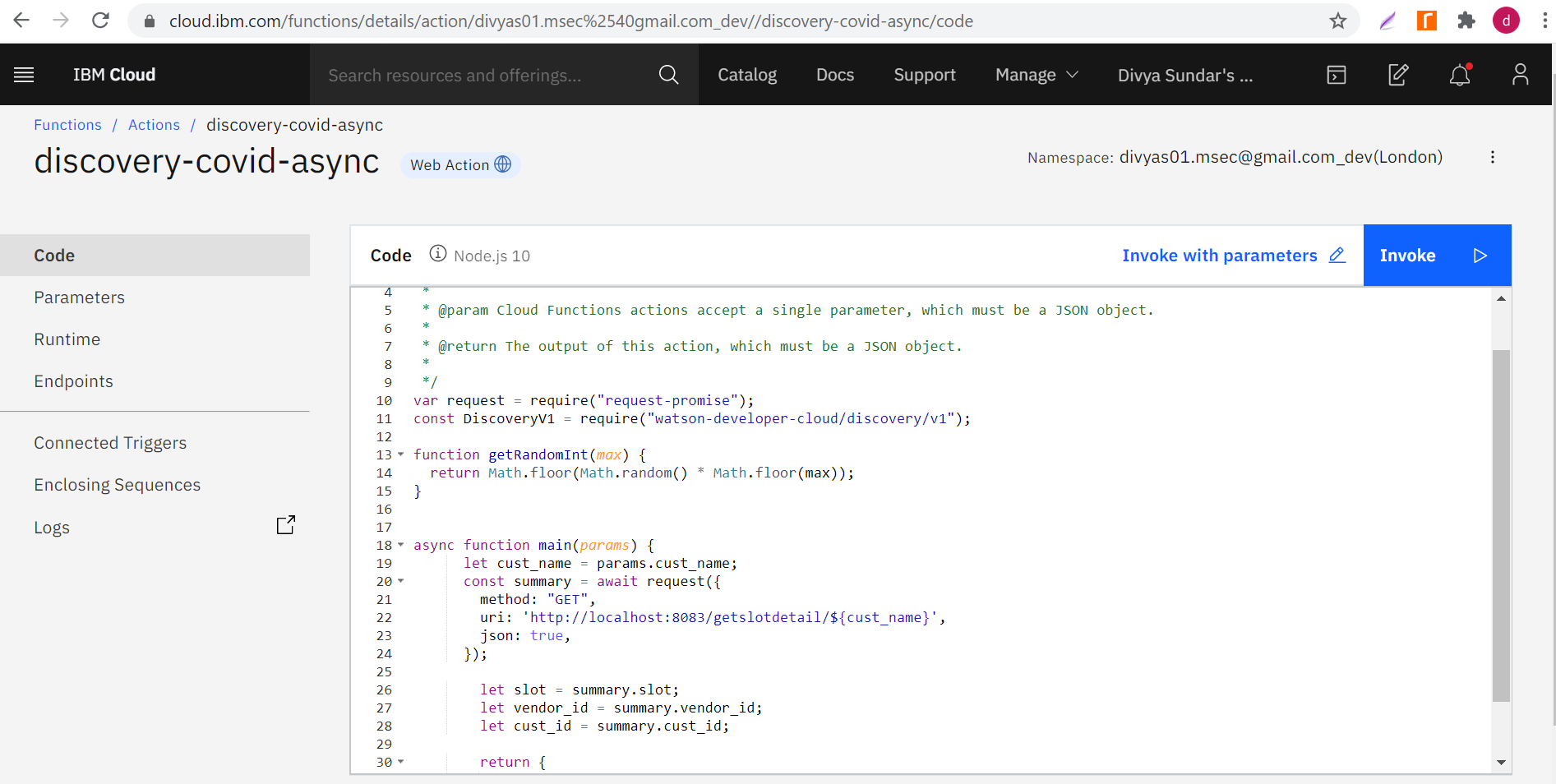


**Rest Response for the spring-boot application to give the customer and slot booked**

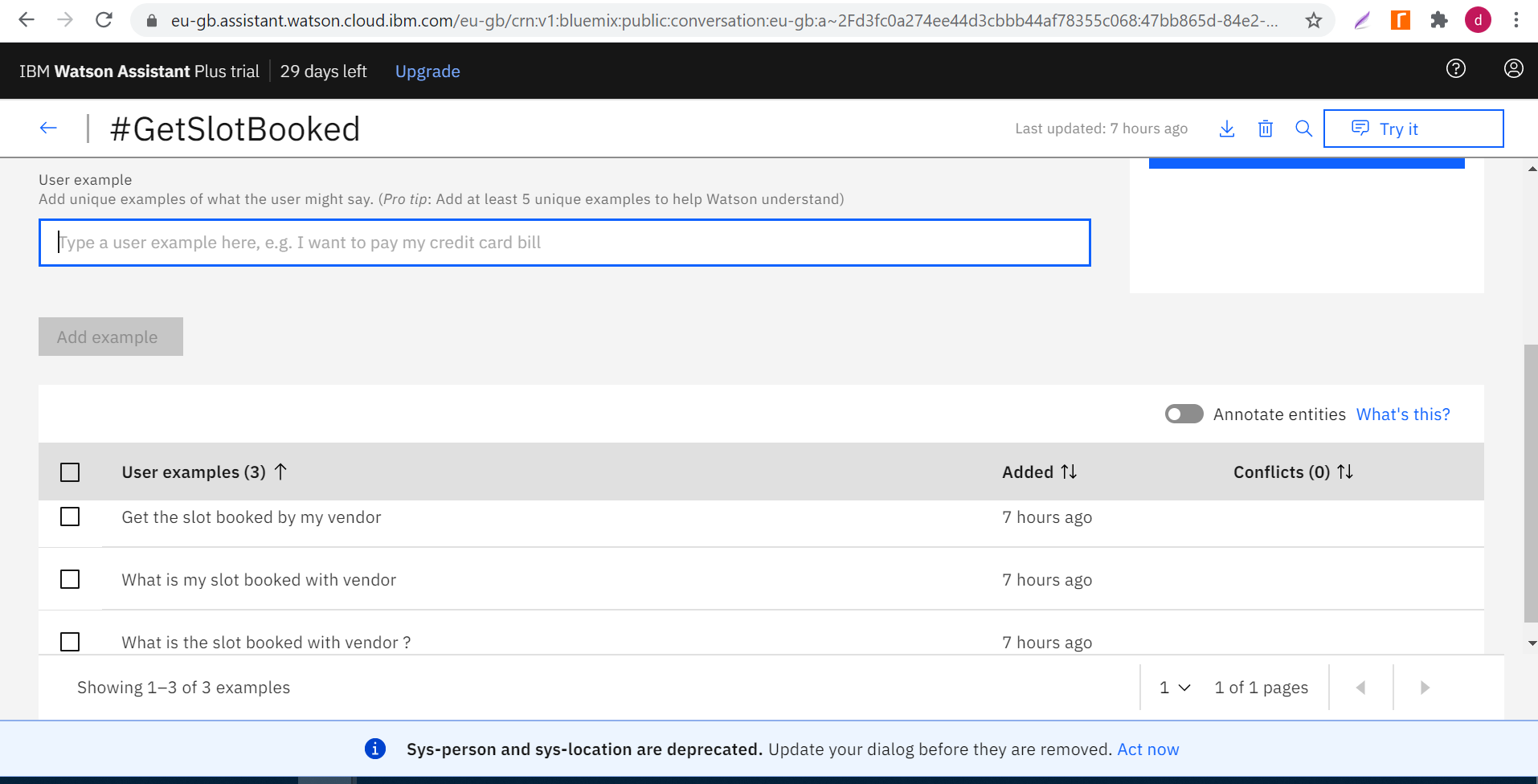


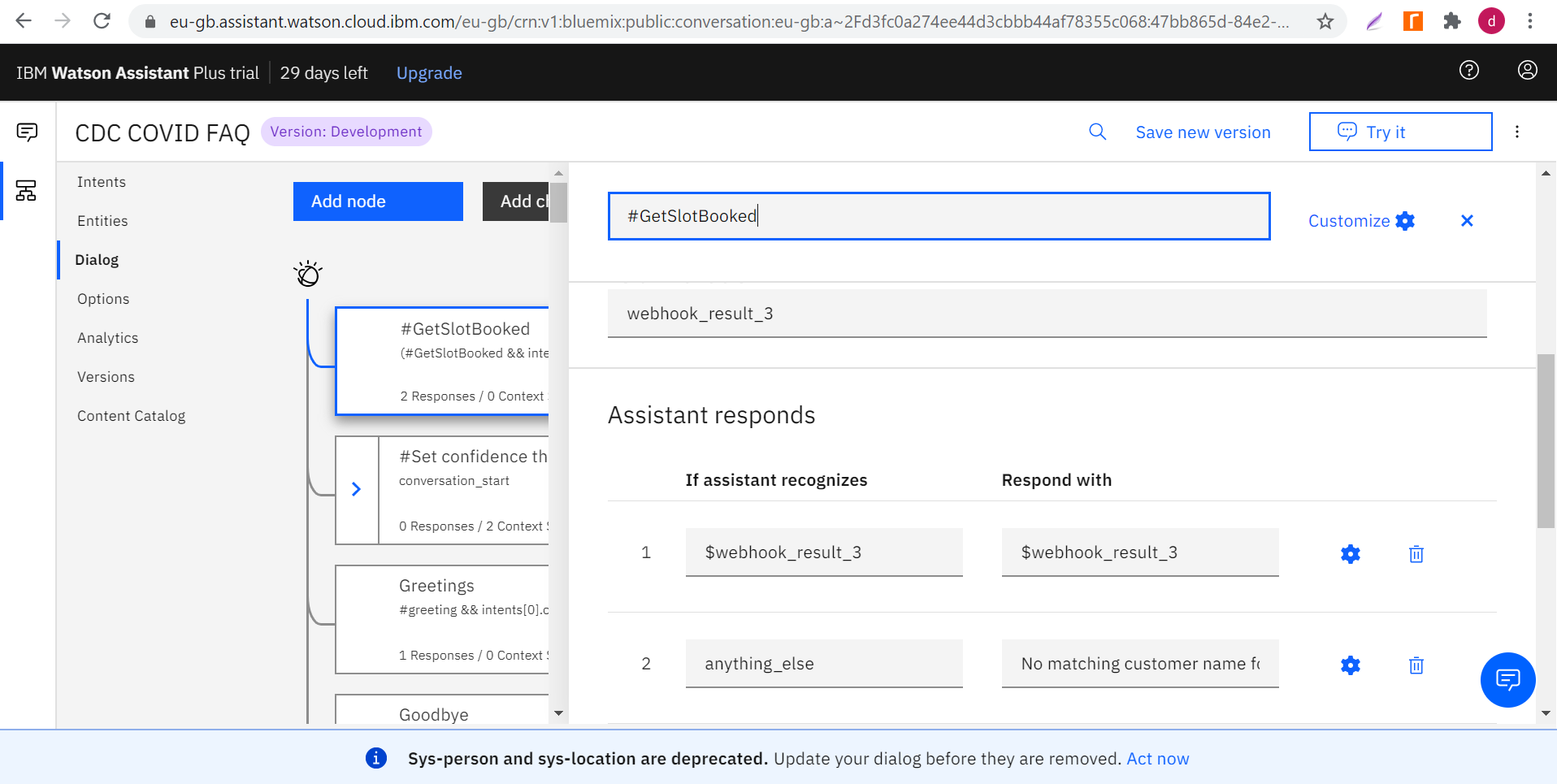
**IBM tools used**

Cloud function having the node 10 code to invoke the external REST call

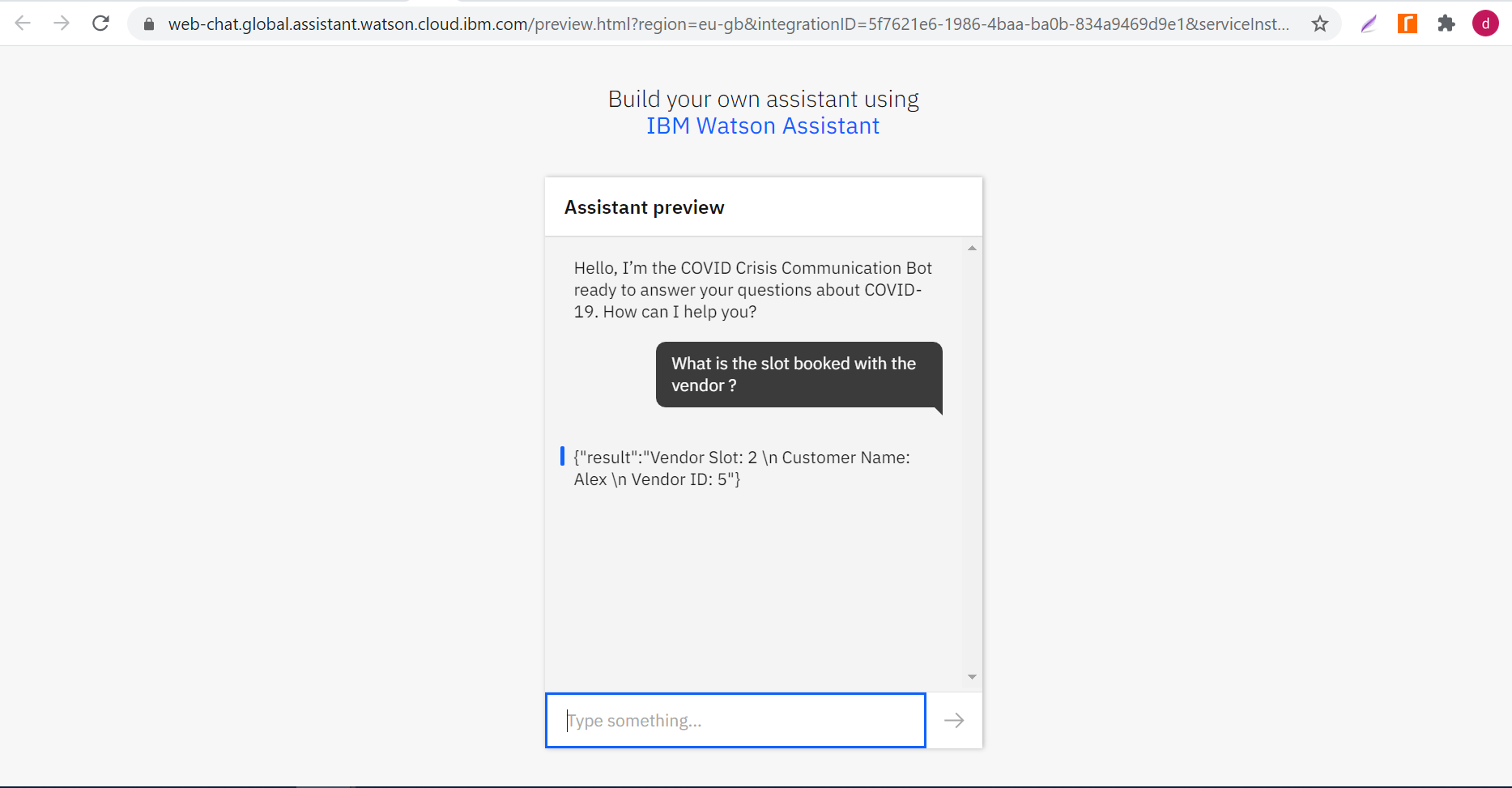


Custom intent (#GetSlotBooked) and dialog skill created to invoke the webhook

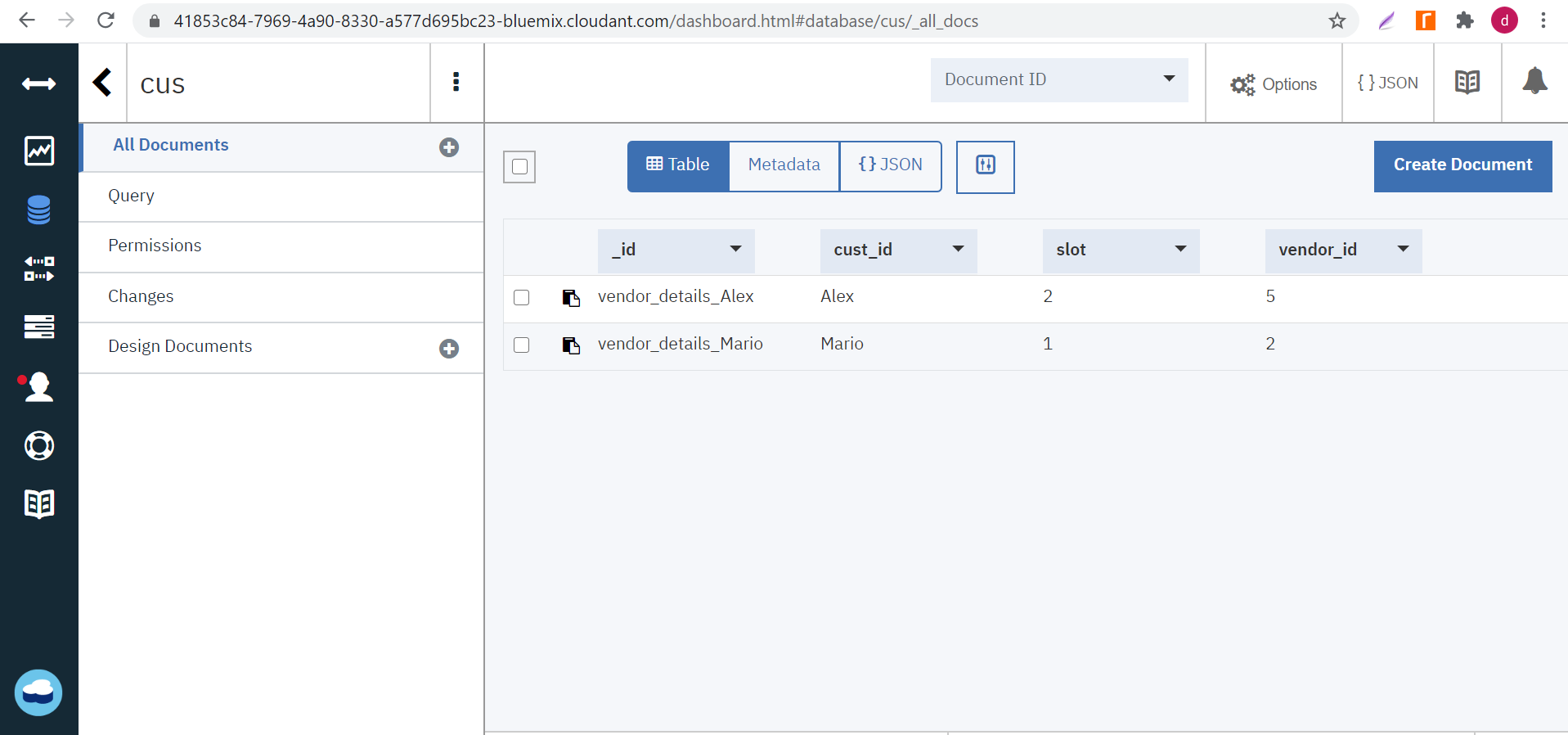




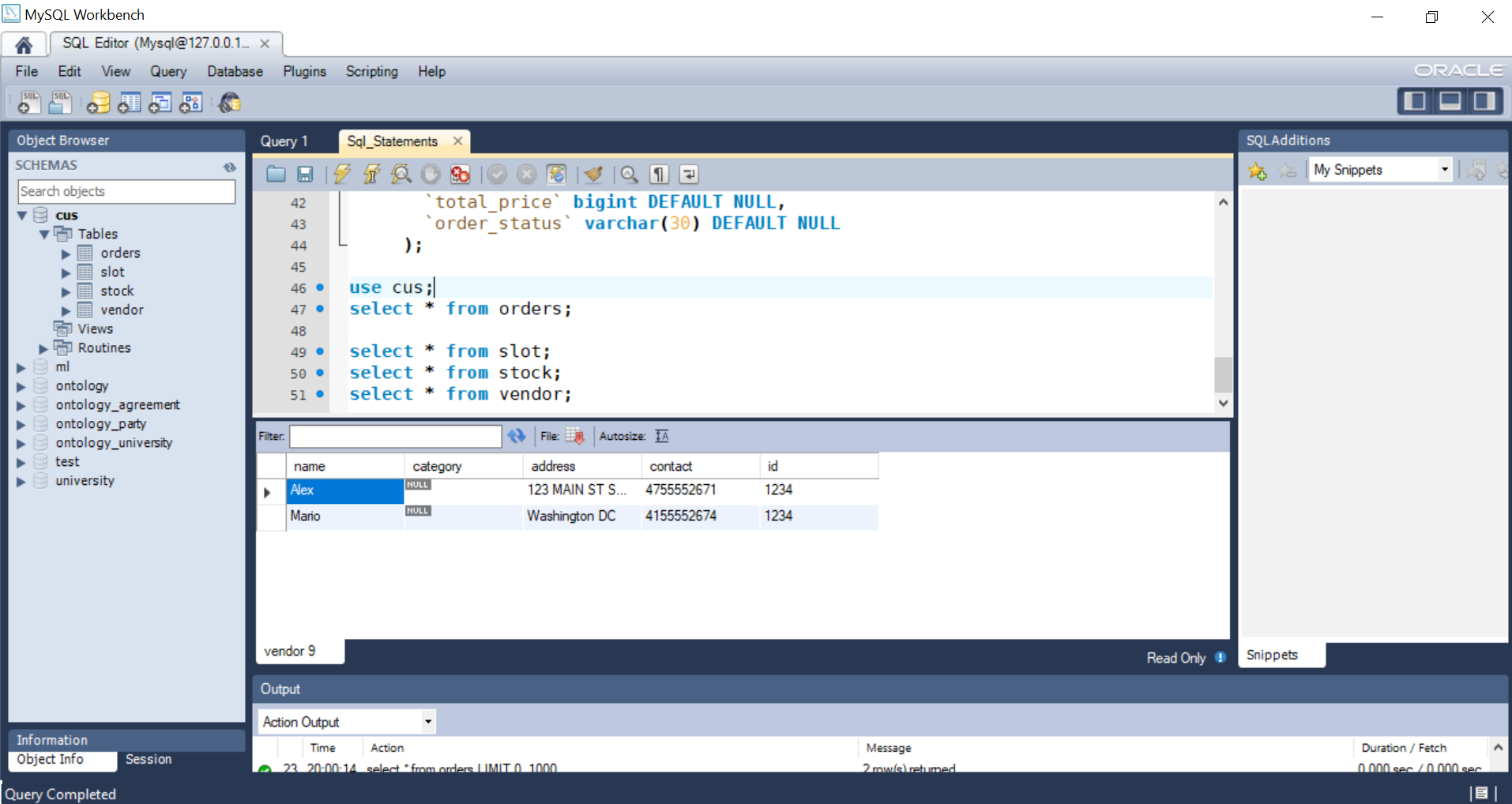
Watson chatbot launched which responds with the customer id and the slot booking id based on the query asked, assuming Alex logs in the chatbot.



Snapshot of cloudant database where the records for vendor slot id, vendor id and the customer id is stored (when the springboot project is started to access the REST calls), which is accessed by the nodejs cloud function that serves a response to the assistant



Snapshots from the local MYSQL database where data is stored in tables for each of the vendor, customer registration and details.



**Use case 2**

**Help giver application benefitting good will donors and receivers**

**\*\* Steps to run the project (locally)**

1. Install angular cli 10 and NPM in the enviroment. Create two angular projects called helper-application and receiver-application using ng new helper-application, ng new receiver-application
2. Copy the src/app folder and src/assets/img from the github link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/tree/master/helper_application/src> and paste it in the newly created angular project for helper-application.
3. Copy the src/app folder and src/assets/img from the github link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/tree/master/recipient_application/src> and paste it in the newly created angular project for receiever-application.
4. In both angular projects, make sure to have only the following lines of code in src/index.html to load the UI properly

<my-app>loading</my-app>

1. Launch both the ngular projects using ng serve
2. Use the Nodejs discovery action discvery-covid-new (link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/discovery-covid-new.js> ) as a discovery action running as a webhook for the chatbot assistant.
3. Link the dialog skill (link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/skill-CDC-COVID-FAQ.json> ) with the chat bot assistant.
4. Pass a parameter name in the chat-bot assistant, key - recipient\_name and value is the name of the receiver.
5. Register the donor and receiver in the UI and use the watson chatbot assistant to get the necessary details.

**\*\* Steps to run the project (hosted in IBM cloud services)**

1. Access the donor and receiver UI applications hosted as the IBM cloud foundry applications.

<http://helper-app.eu-gb.cf.appdomain.cloud> – Helper donor registration

<http://recipient-app.eu-gb.cf.appdomain.cloud/> - Helper receiver registration

1. Install docker locally to access the backend micro-service hosted in the IBM cloud registry service.

Steps to install docker engine, docker ce and docker compose can be found at <https://docs.docker.com/get-docker/>

1. Use docker pull to pull the image hosted in the IBM cloud registry service. The following command pulls the image with the latest tag from the repository us.icr.io

docker pull us.icr.io/divyasundar/cfc2020:cfc-spring-boot-docker

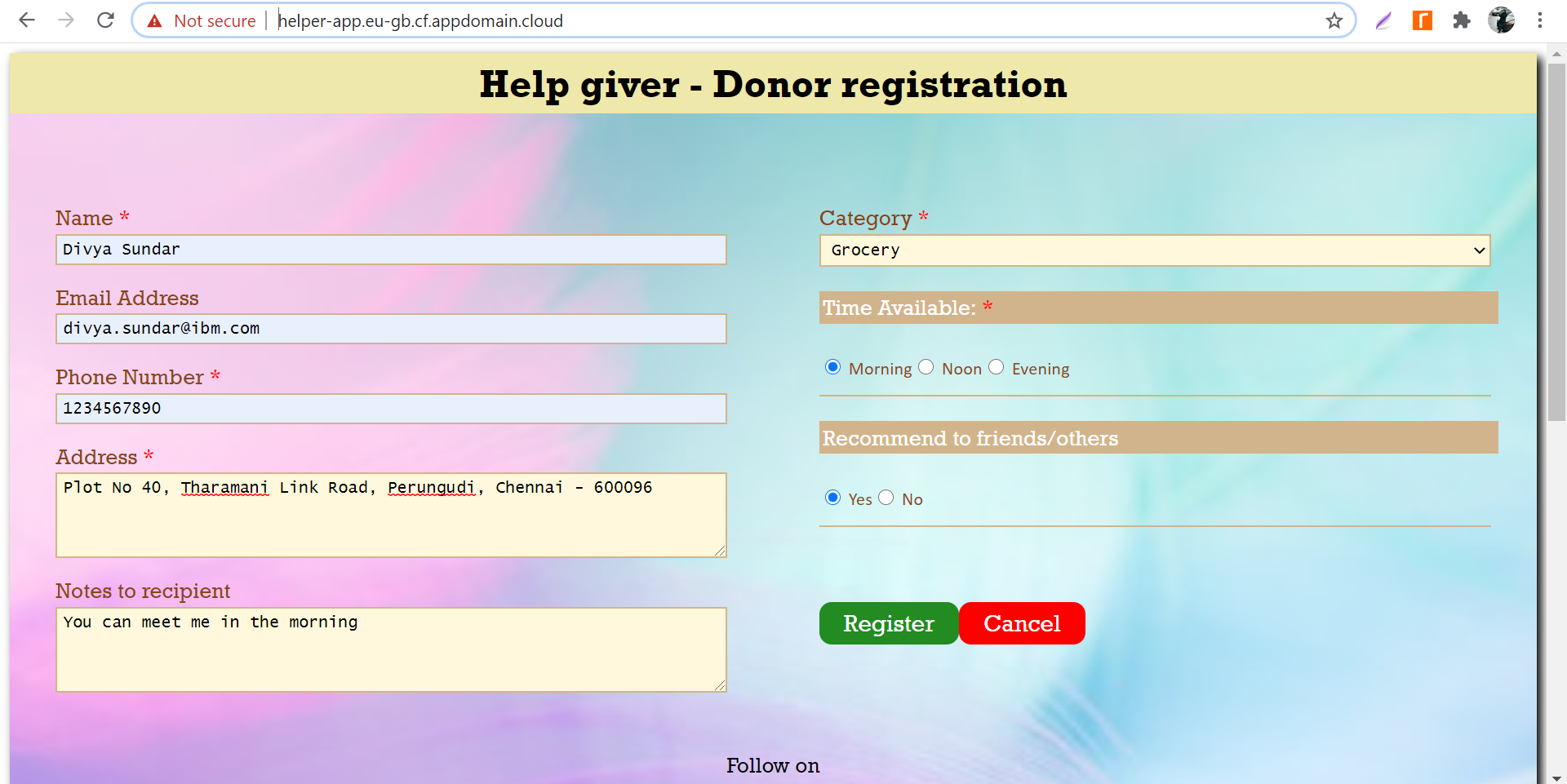
The namespace created in the registry service is divyasundar and the repository tag is cfc2020.

1. Install the IBM cloud service/cf cli utility to connect with the central hub where docker images are pushed in IBM repositories. Ibmcloud command line utility offers a wide variety of docker commands and cf push commands.
2. Use the Nodejs discovery action discvery-covid-new (link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/discovery-covid-new.js> ) as a discovery action running as a webhook for the chatbot assistant to get the receiver details.
3. Use the Nodejs discovery action discovery-covid-donor (link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/discovery-covid-donor.js> ) as as a discovery action running as a webhook for the chatbot assistant to get the details of the receiver slots and receiver names given a donor.
4. Link the dialog skill (link - <https://github.com/DivyaSundar-89/CFC_2020_SlotBooking/blob/master/skill-CDC-COVID-FAQ.json> ) with the chat bot assistant.
5. Pass a parameter name in the chat-bot assistant, key - recipient\_name and value is the name of the receiver.
6. Register the donor and receiver in the UI and use the watson chatbot assistant to get the necessary details.
7. Use the webhook function of discovery-covid-donor to get the details of the receiver slot id and the receiver names given a donor queries through the chatbot assistant.

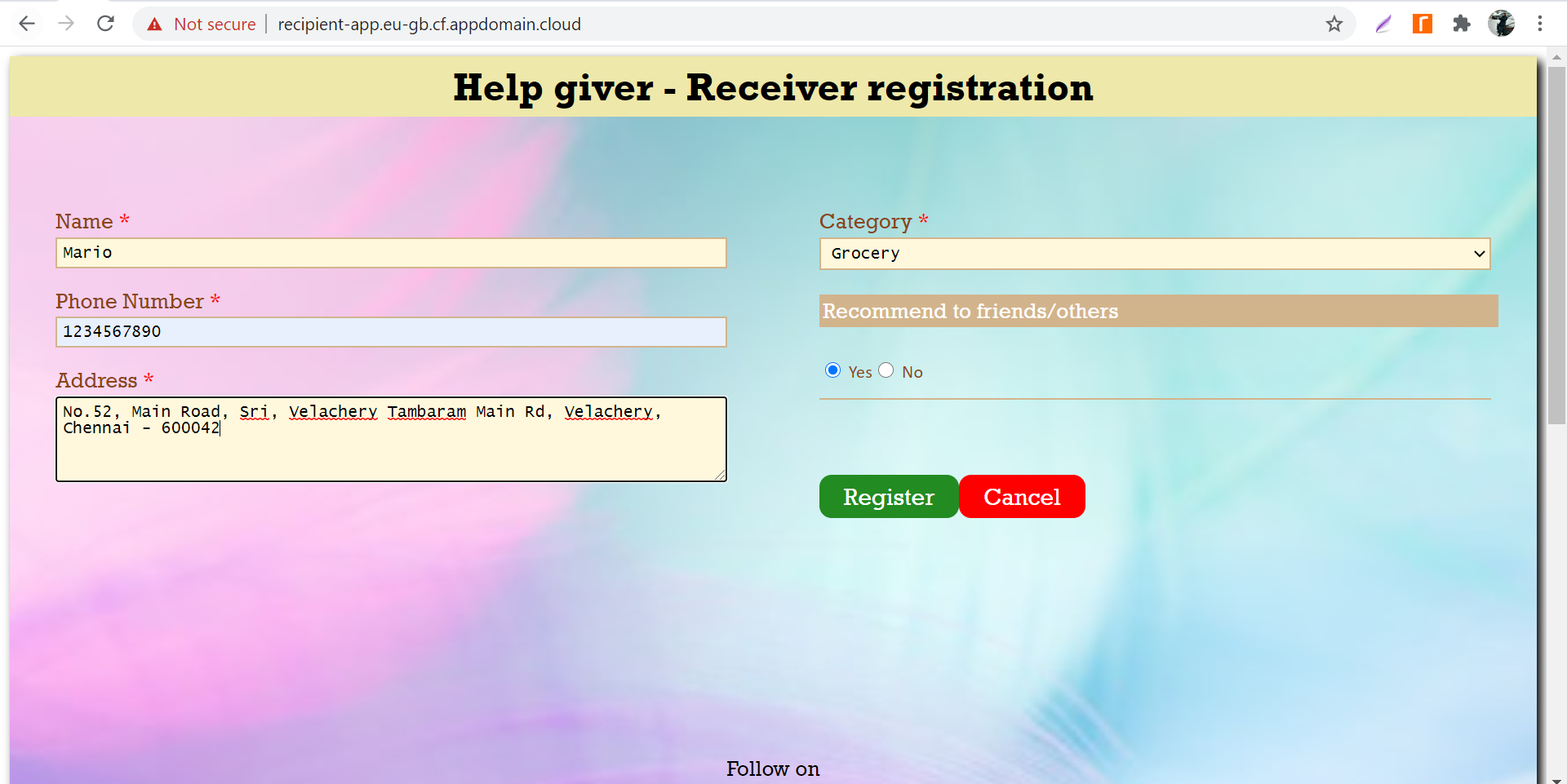
**\*\* Demo recording of use case 2**

Demo recording video link - <https://www.youtube.com/watch?v=sCK1K-pKS7Q>

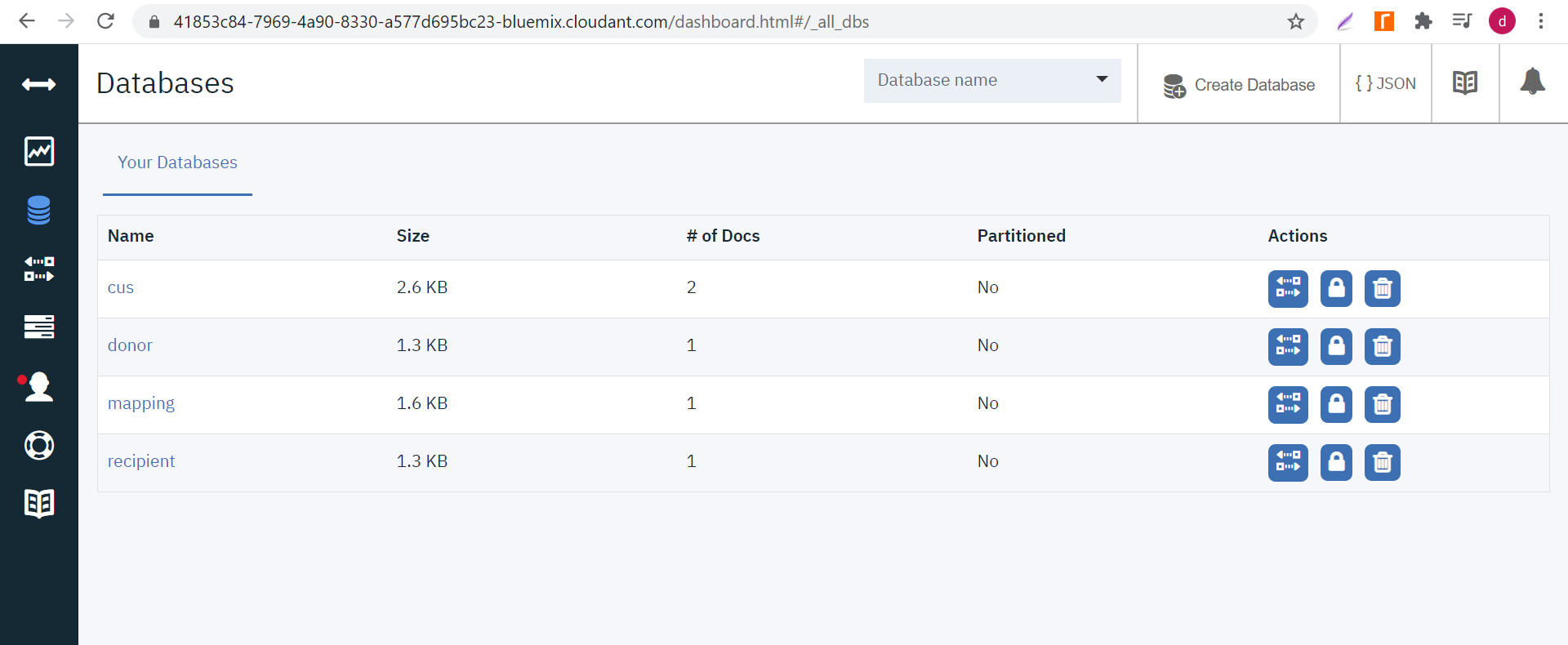
Donor registration in the helper application hosted in IBM cloud foundry bluemix application



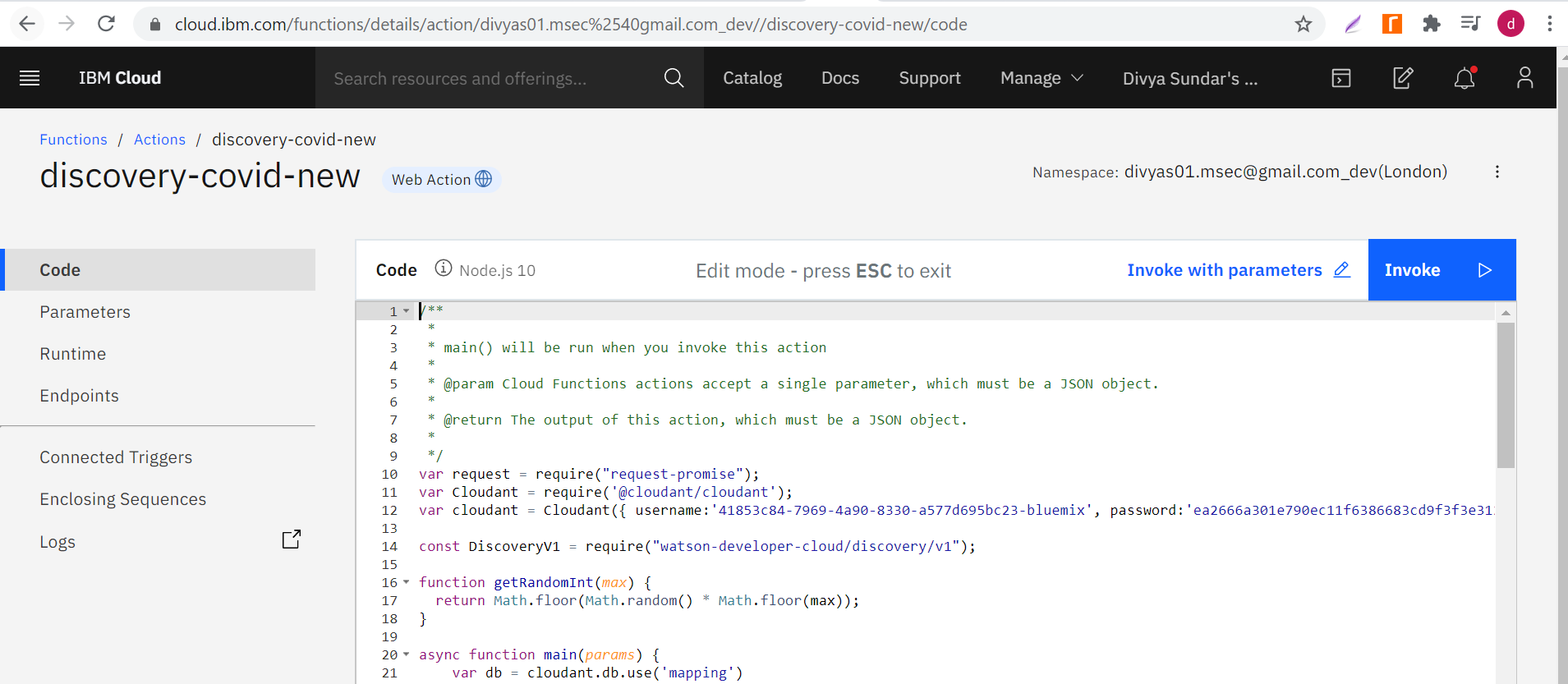
Receiver registration in the receiver application hosted in the IBM cloud foundry bluemix application. Once receiver registers, the normatim openAPI is used to determine the donor willing to share the category and close to the location of the receiver and mapped with the receiver.

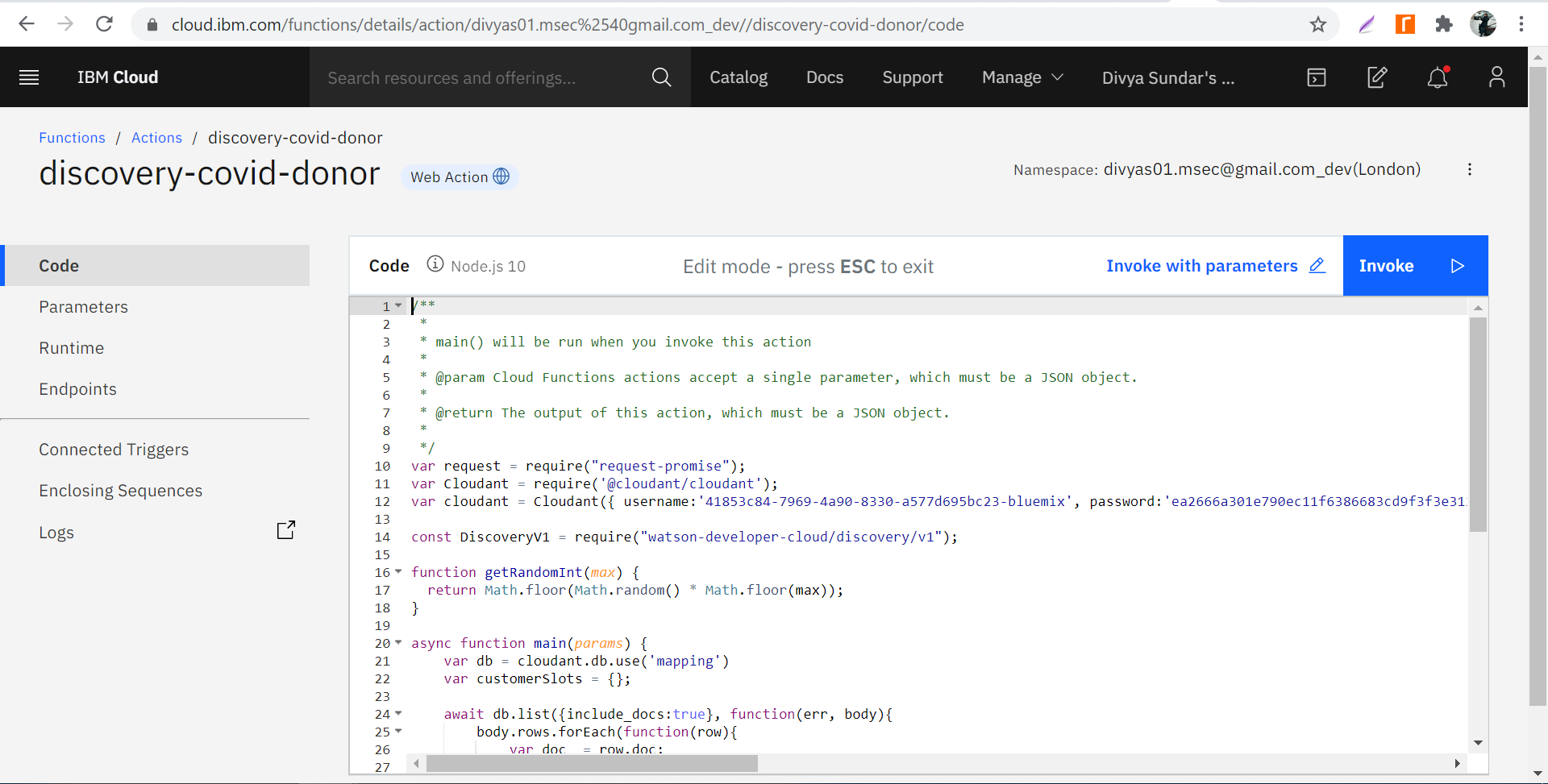


Snapshot from IBM cloudant DB based on the donor and receiver details

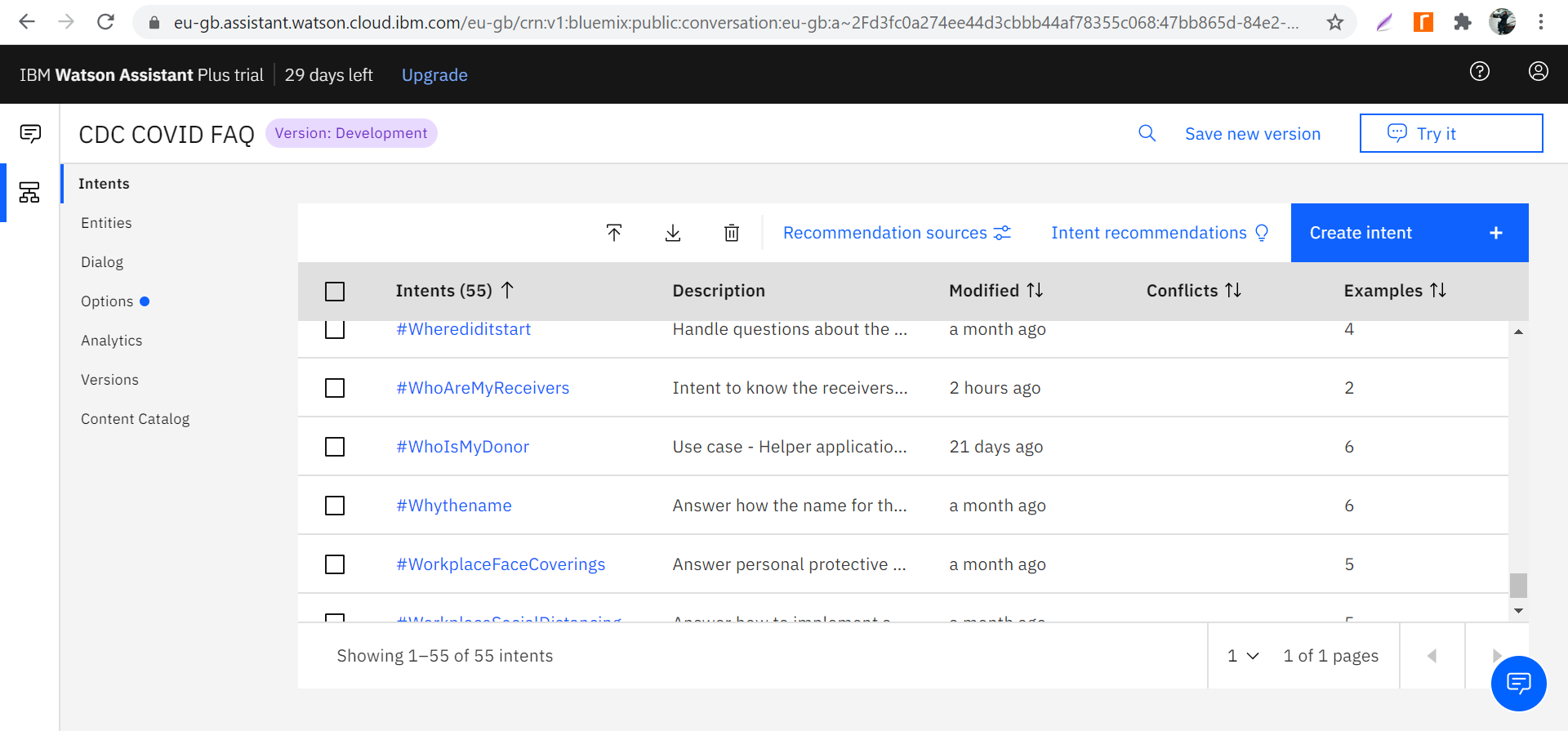


Discovery function as a web-hook action

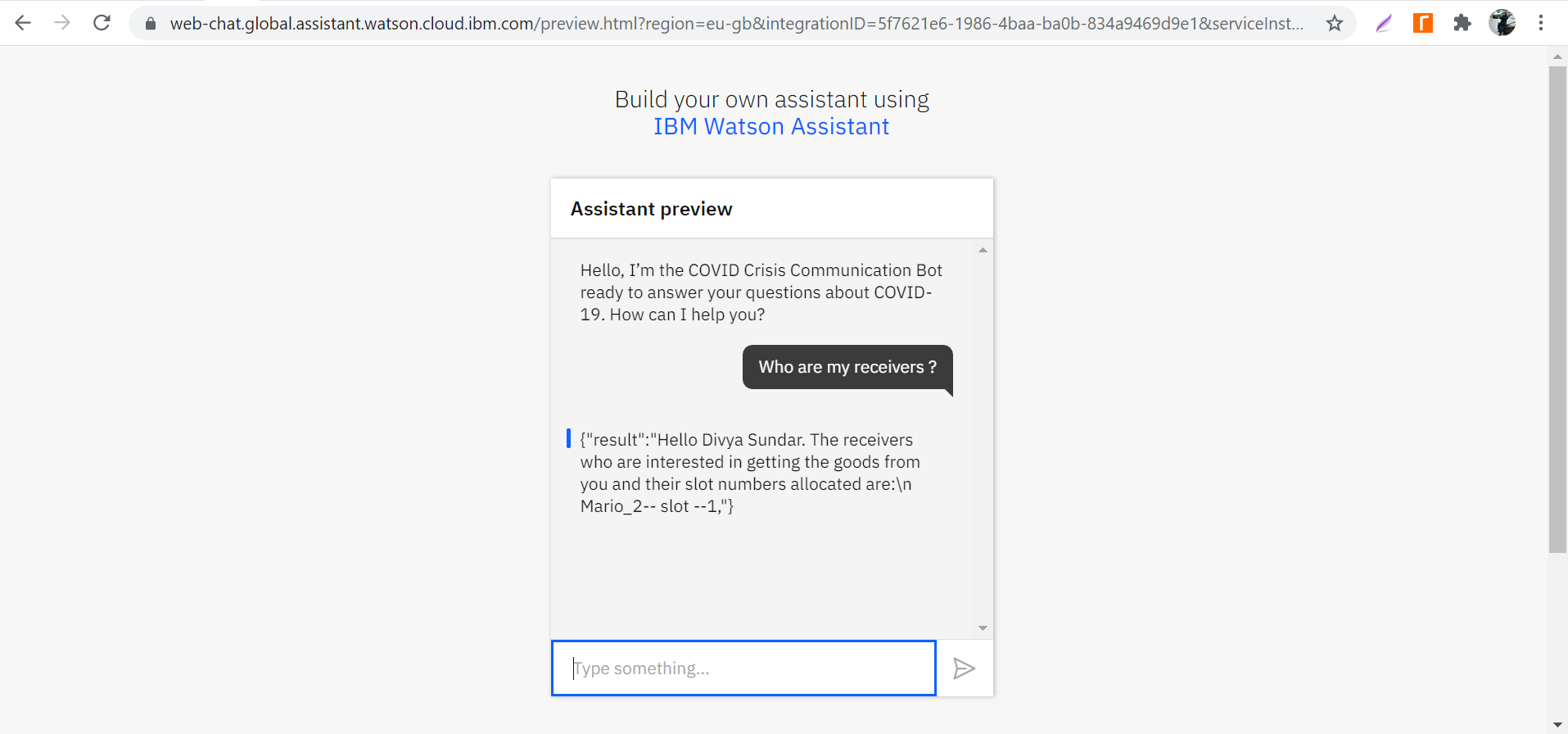




Custom dialog skill to respond for the effective help giver application



Snapshot for the IBM Watson chatbot assistant to get the list of receivers and the slot id’s booked by the donor who queries through the chatbot.



Snapshot for the IBM Watson chatbot assistant to get the details by the receiver (including the slot id booked by the donor) when receiver queries through the chatbot.

