

## Project Development Phase

### Sprint 4

Date	18 November 2022
Team ID	PNT2022TMID35942
Project Name	Project – AI based discourse for Banking Industry

In sprint 4, we have focused on developing a database to store user data and improving the features of our chatbot by adding new actions to it.

We have used **IBM cloudant** to create the required databases for our model.

Offers, current account balance, feedback data storage and net banking registration are the new features added in this sprint. Links for current account balance, feedback form and net banking registration form is provided by the chatbot as response and they are also provided in the webpage created.

### IMPORTING REQUIRED LIBRARIES:

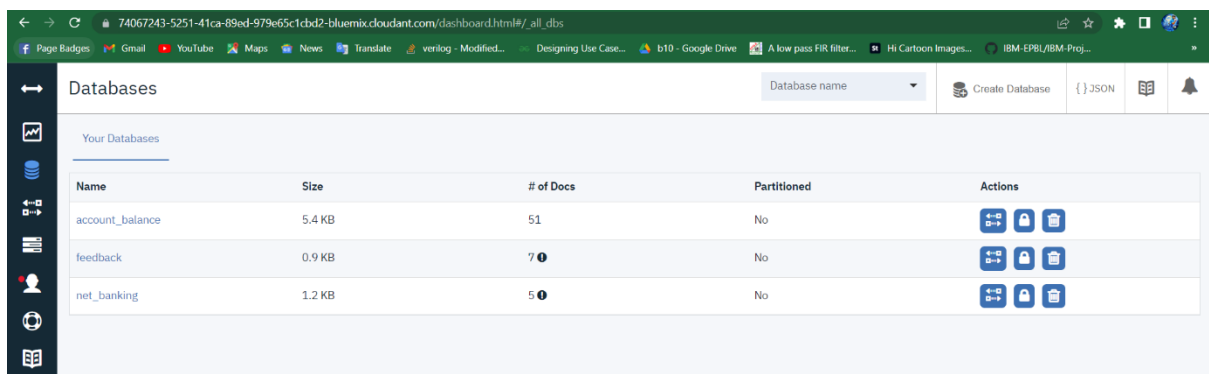
```
from flask import Flask,render_template
from flask import *
from cloudant.client import Cloudant
from cloudant.result import Result
```

### INTEGRATING IBM CLOUDANT DATABASE:










```
ACCOUNT_NAME = "74067243-5251-41ca-89ed-979e65c1cbd2-bluemix"
API_KEY = "IqlWgY3pe1n9DGN4pN_9uSXzmlCs27ML4DkcTAePwkFbv"

client = Cloudant.iam(ACCOUNT_NAME,API_KEY,connect=True)
```

### DIFFERENT DATABASES CREATED:

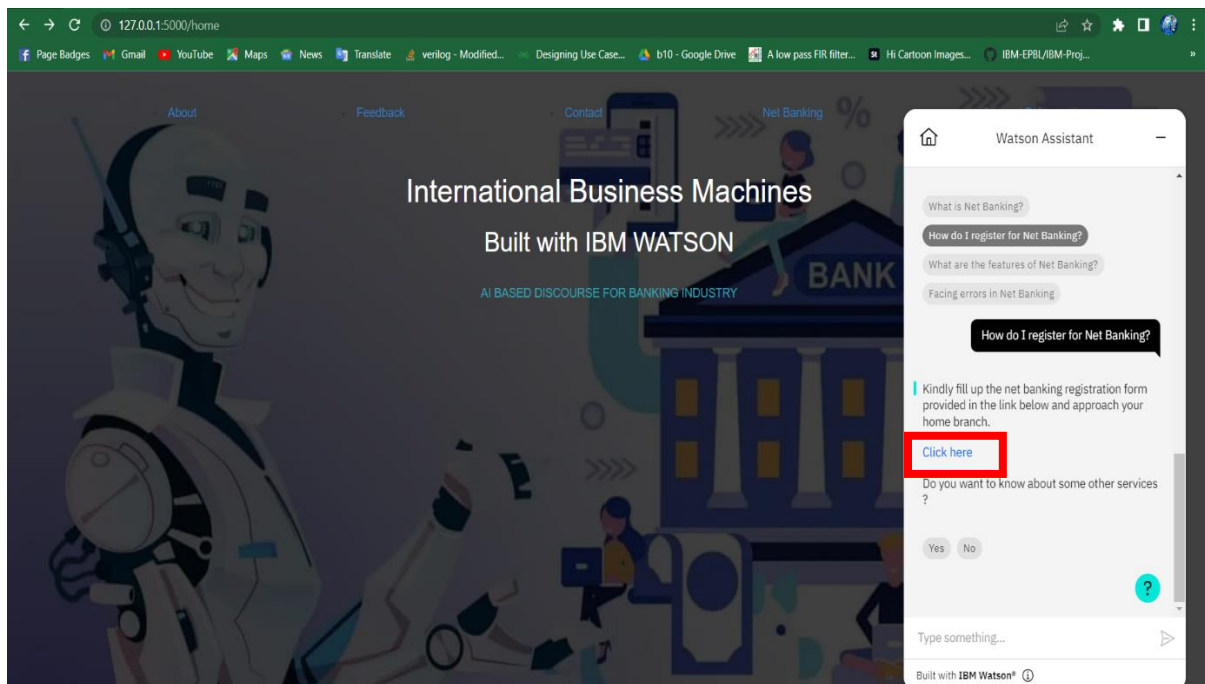


The screenshot shows the IBM Cloudant dashboard with a table of databases. The table has columns for Name, Size, # of Docs, Partitioned, and Actions. Three databases are listed: account\_balance (5.4 KB, 51 docs), feedback (0.9 KB, 7 docs), and net\_banking (1.2 KB, 5 docs). Each database has icons for adding, deleting, and locking documents.

Name	Size	# of Docs	Partitioned	Actions
account_balance	5.4 KB	51	No	  
feedback	0.9 KB	7	No	  
net_banking	1.2 KB	5	No	  

## NET BANKING:

We have added a link in the chatbot which is given as a response when the user asks about net banking registration.



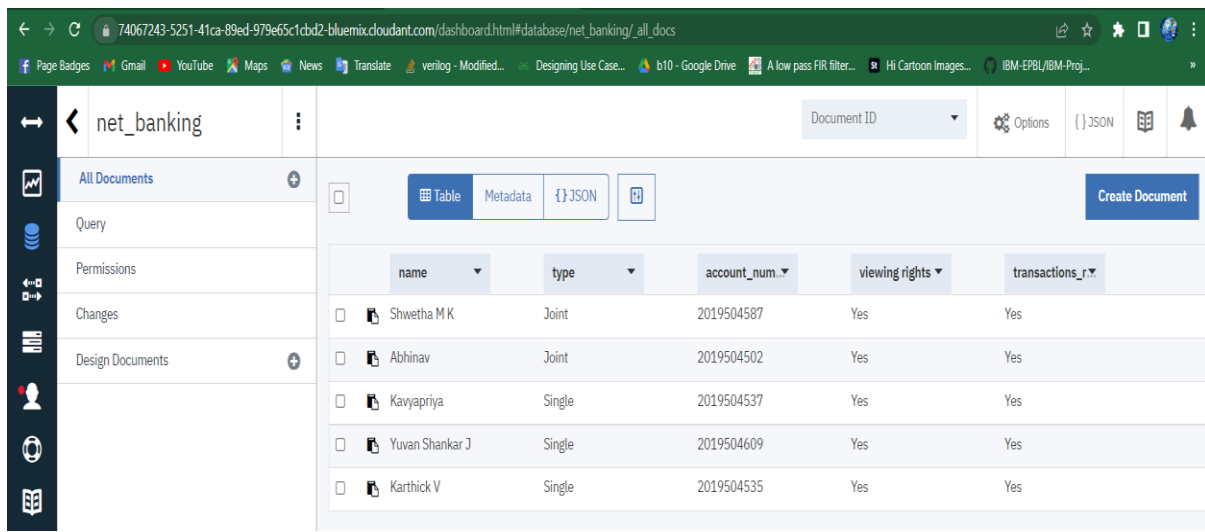
The link “click here” will direct the user to net banking registration form.

## NET BANKING REGISTRATION FORM:

A screenshot of the "Net Banking Registration" form. The form is titled "Net Banking Registration" and is set against a red background. It contains several input fields and checkboxes. The fields are: "Name of the customer" (with a placeholder "Enter your name"), "E-mail Address" (with a placeholder "Enter your e-mail"), "Contact Number" (with a placeholder "Enter your contact number"), "Account Number" (with a placeholder "Enter your account number"), "Date of Birth" (with a dropdown menu showing "dd-mm-yyyy"), "Viewing Rights (Yes/No)" (with a placeholder "Enter Yes/No"), "Transaction Rights (Yes/No)" (with a placeholder "Enter Yes/No"), "Single/Joint Account" (with a placeholder "Enter your Account Type"), and "Date of Application" (with a dropdown menu showing "dd-mm-yyyy"). There is a "Submit" button at the bottom left of the form.

## STORAGE OF USER DATA PROVIDED IN IBM CLOUDANT DATABASE:

The data provided by the users in the registration form is stored in the net\_banking database as shown below.



	name	type	account_num	viewing rights	transactions_r
<input type="checkbox"/>	Shwetha M K	Joint	2019504587	Yes	Yes
<input type="checkbox"/>	Abhinav	Joint	2019504502	Yes	Yes
<input type="checkbox"/>	Kavyapriya	Single	2019504537	Yes	Yes
<input type="checkbox"/>	Yuvan Shankar J	Single	2019504609	Yes	Yes
<input type="checkbox"/>	Karthick V	Single	2019504535	Yes	Yes

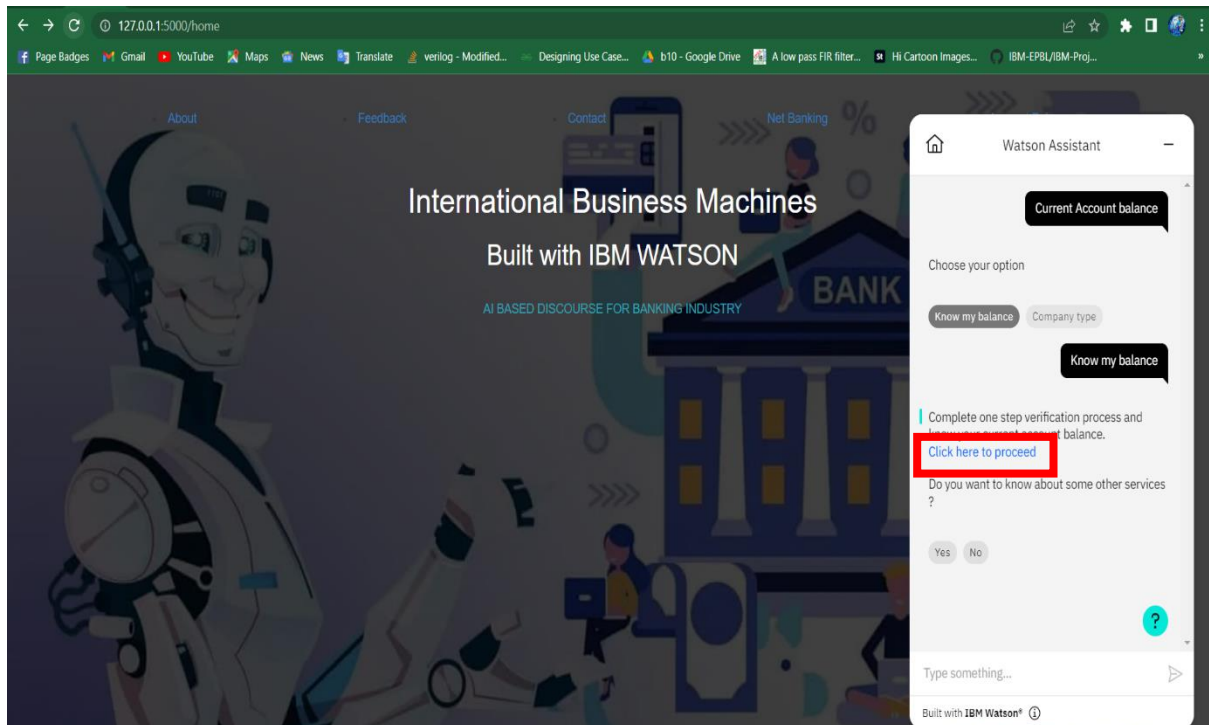
## PYTHON CODE FOR USER DATA STORAGE:

```
@app.route('/netbanking', methods = ['GET', 'POST'])
def netbanking():
    print("*****")
    if request.method == "POST":
        email = request.form['mail']
        name = request.form['name']
        acc_num = request.form['num']
        contact_number=request.form['number']
        date_of_birth=request.form['dob']
        date_of_application=request.form['doa']
        viewing_rights=request.form['viewing']
        transaction_rights=request.form['transaction']
        type_of_account=request.form['type']
        jsonDocument= {
            'email':email,
            'name':name,
            'account_number':acc_num,
            'contact_number':contact_number,
            'viewing_rights':viewing_rights,
            'transactions_rights':transaction_rights,
            'date_of_birth':date_of_birth,
            'date_of_application':date_of_application,
            'type':type_of_account
        }
        newDocument = mydatabase1.create_document(jsonDocument)
        result = Result(mydatabase1.all_docs,include_docs=True)
        print(result[0])
        return redirect(url_for('home'))
        print('#####')
    return render_template('netbanking.html')
```

## ACCOUNT BALANCE:

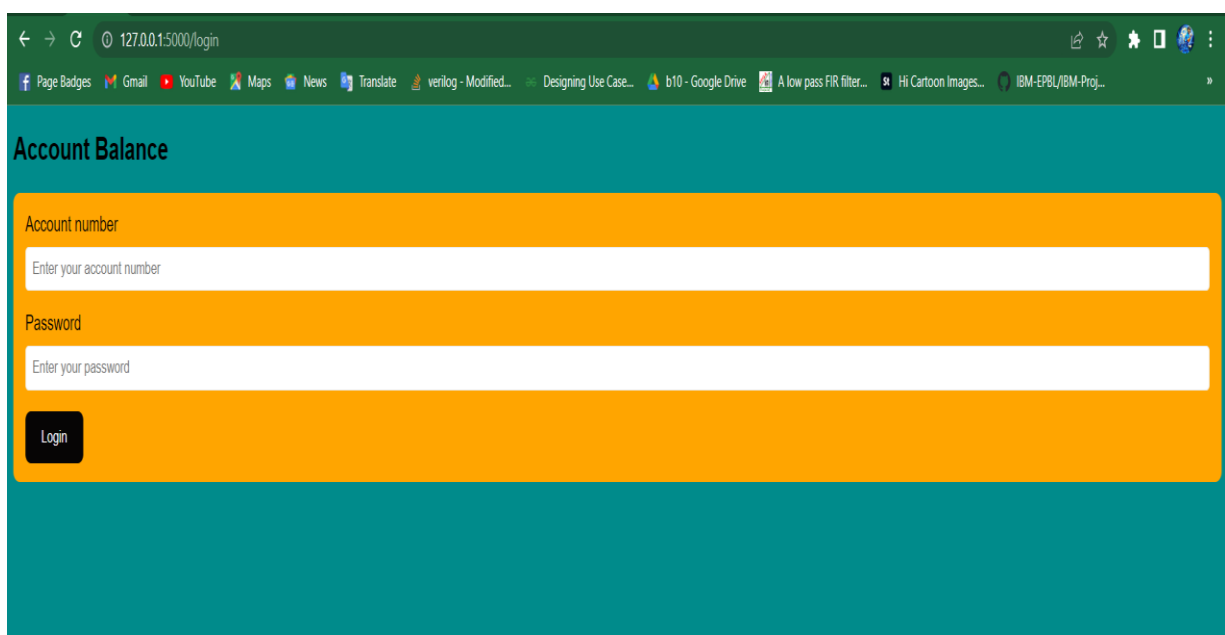
We have added a new feature for the users to find their account balance with one step authentication.

When the user asks for his account balance, a link appears as a response which will direct the user to the login page.



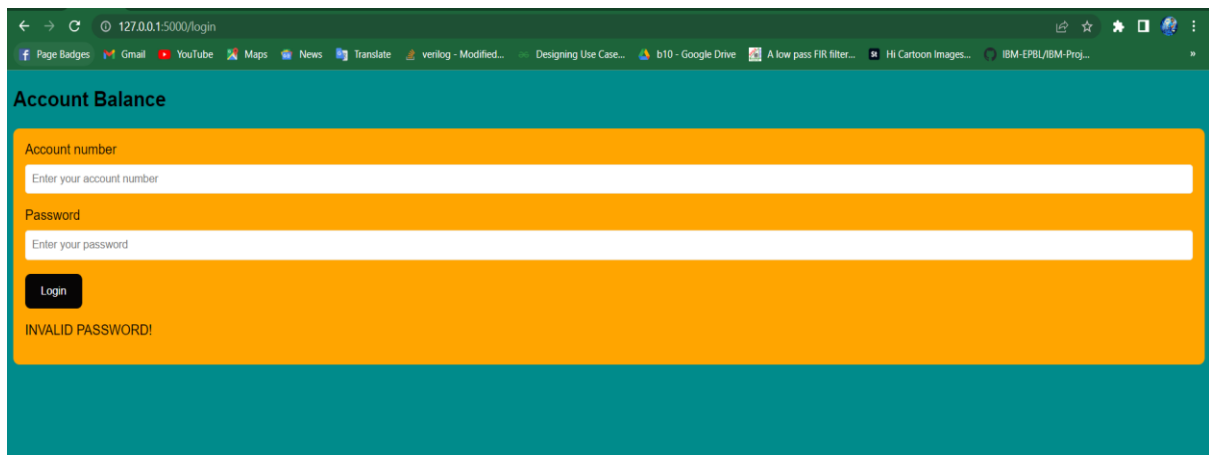
## ACCOUNT BALANCE WEBPAGE:

The 'click here' link shown in the above image directs the user to the login page.



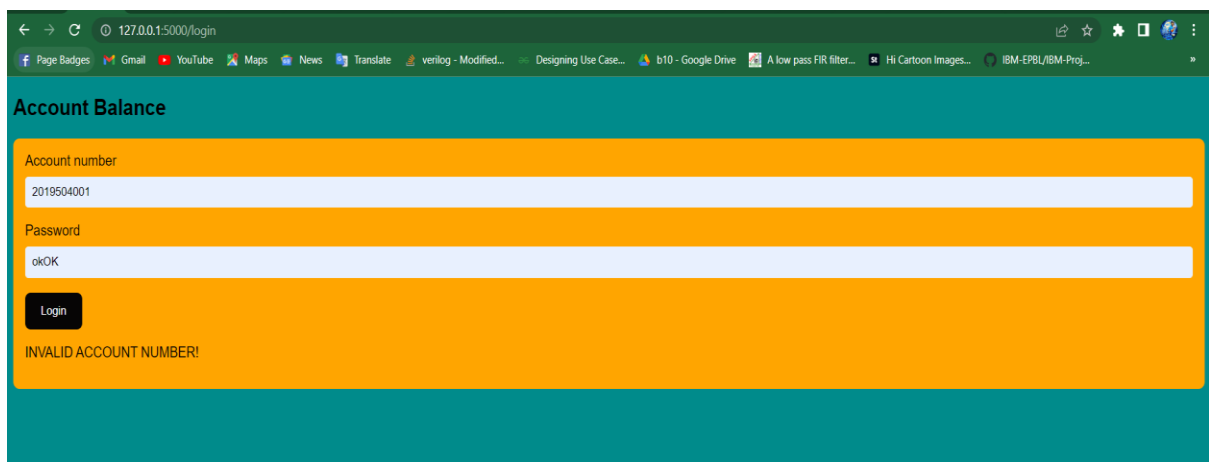
The information given by the user is validated and the following messages are shown based on the result of validation.

### **CASE 1: ACCOUNT NUMBER AND PASSWORD MISMATCH:**



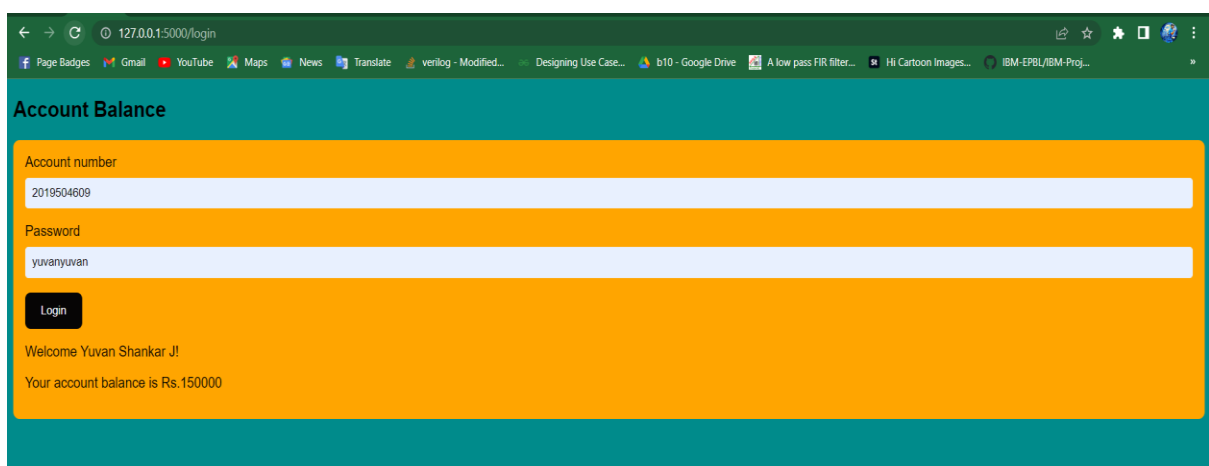
A screenshot of a web browser window showing a login page. The browser's address bar displays '127.0.0.1:5000/login'. The page has a teal header with the text 'Account Balance'. Below the header is a large orange rectangular area containing the login form. The form has two input fields: 'Account number' with the placeholder text 'Enter your account number' and 'Password' with the placeholder text 'Enter your password'. Below these fields is a black 'Login' button. At the bottom of the orange area, the text 'INVALID PASSWORD!' is displayed in black. The browser's tab bar shows several open tabs, including 'Page Badges', 'Gmail', 'YouTube', 'Maps', 'News', 'Translate', 'verilog - Modified...', 'Designing Use Case...', 'b10 - Google Drive', 'A low pass FIR filter...', 'Hi Cartoon Images...', and 'IBM-EPBL/IBM-Proj...'.

### **CASE 2: ACCOUNT NUMBER IS INVALID**



A screenshot of a web browser window showing a login page. The browser's address bar displays '127.0.0.1:5000/login'. The page has a teal header with the text 'Account Balance'. Below the header is a large orange rectangular area containing the login form. The form has two input fields: 'Account number' with the value '2019504001' and 'Password' with the value 'okOK'. Below these fields is a black 'Login' button. At the bottom of the orange area, the text 'INVALID ACCOUNT NUMBER!' is displayed in black. The browser's tab bar shows several open tabs, including 'Page Badges', 'Gmail', 'YouTube', 'Maps', 'News', 'Translate', 'verilog - Modified...', 'Designing Use Case...', 'b10 - Google Drive', 'A low pass FIR filter...', 'Hi Cartoon Images...', and 'IBM-EPBL/IBM-Proj...'.

### **CASE 3: ACCOUNT NUMBER AND PASSWORD MATCH**



A screenshot of a web browser window showing a login page. The browser's address bar displays '127.0.0.1:5000/login'. The page has a teal header with the text 'Account Balance'. Below the header is a large orange rectangular area containing the login form. The form has two input fields: 'Account number' with the value '2019504609' and 'Password' with the value 'yuvanyuvan'. Below these fields is a black 'Login' button. At the bottom of the orange area, the text 'Welcome Yuvan Shankar J!' and 'Your account balance is Rs.150000' is displayed in black. The browser's tab bar shows several open tabs, including 'Page Badges', 'Gmail', 'YouTube', 'Maps', 'News', 'Translate', 'verilog - Modified...', 'Designing Use Case...', 'b10 - Google Drive', 'A low pass FIR filter...', 'Hi Cartoon Images...', and 'IBM-EPBL/IBM-Proj...'.

## PYTHON CODE FOR USER DATA VALIDATION:

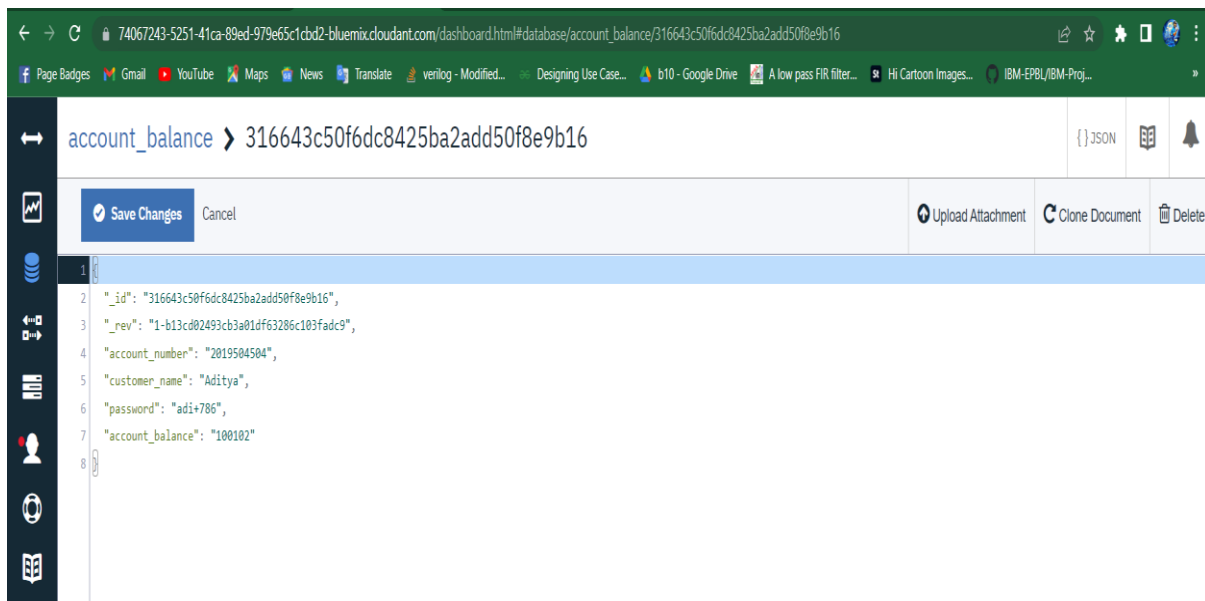
```
mydatabase2=client.create_database('account_balance')
@app.route('/Login',methods=['GET','POST'])
def login():
    if request.method=='POST':
        account_number=request.form['accnum']
        password=request.form['pass']
        account_found=False
        for documents in mydatabase2:
            if documents['account_number']==account_number:
                if documents['password']==password:
                    flash_name='Welcome'+ ' '+documents['customer_name']+'!'
                    flash(flash_name)
                    flash_text='Your account balance is'+ ' '+Rs."+documents['account_balance']
                    flash(flash_text)
                else:
                    flash('INVALID PASSWORD!')
                    account_found=True
                    break
        if not account_found:
            flash('INVALID ACCOUNT NUMBER!')
        return render_template('Login.html')
```

## ACCOUNT BALANCE DATABASE:

The input data provided by the user is validated using the python code above. Validation is done by comparing the inputs given and the data available in the database. If the account number and password provided matches with account number and password of any document in the database, the corresponding account balance is given as output.

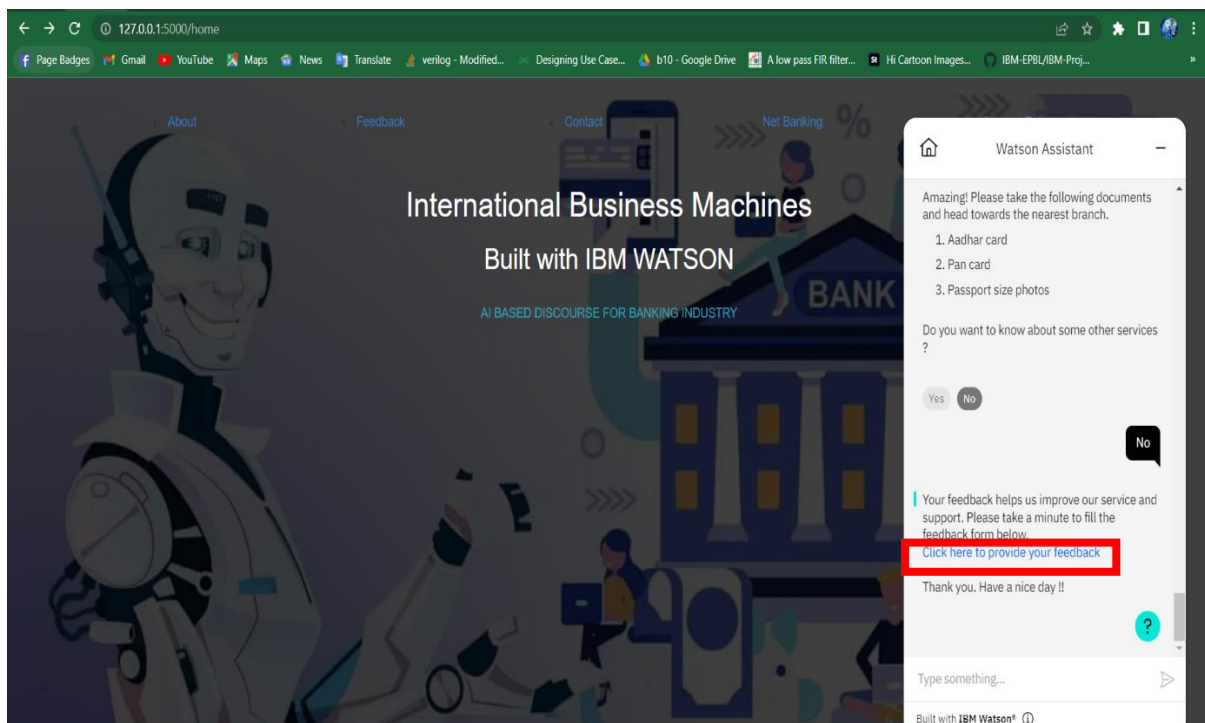
	customer_name	password	account_num	account_bala	_id
<input type="checkbox"/>	Dhanesh K	dhanu_456	2019504514	132013	0a02d03cb7c8ca0d8a86...
<input type="checkbox"/>	Shakeel Mohamed G	shakeel	2019504584	230000	159bc16b139816a51980...
<input type="checkbox"/>	Akshay K P	akshay	2019504506	235070	16a075cb5b257b8e45ce...
<input type="checkbox"/>	Goutham	gou_23	2019504523	720200	1704561c00e8f8897ef0f...
<input type="checkbox"/>	Srivatsan	sri_7618	2019504591	198201	1704561c00e8f8897ef0f...
<input type="checkbox"/>	Shwetha M K	shwe_178	2019504587	831024	1704561c00e8f8897ef0f...
<input type="checkbox"/>	Naveen Kumar	nac_754	2019504554	238500	316643c50f6dc8425ba2...
<input type="checkbox"/>	Aditya	adi+786	2019504504	100102	316643c50f6dc8425ba2...
<input type="checkbox"/>	Shakeel Magdum	shak_5	2019504574	234260	316643c50f6dc8425ba2...
<input type="checkbox"/>	Sanjay Krushnan	sanjay	2019504577	431231	4686590059665de9f5ffe...
<input type="checkbox"/>	Sudharsan A R	sudha_345	2019504594	122013	4686590059665de9f5ffe...
<input type="checkbox"/>	Priyadharshini B	priyu_225	2019504565	231600	4686590059665de9f5ffe...

## A SAMPLE DOCUMENT STORED IN THE ACCOUNT BALANCE DATABASE:



## FEEDBACK:

A feedback form has been provided for the users at the end of a conversation to share their experience and mention any difficulties faced by them while they use the chatbot.



The link 'Click here to provide your feedback' will direct the user to feedback form webpage.

## FEEDBACK FORM WEBPAGE:

**Feedback Form**

Name  
Enter your name

E-mail Address  
Enter your e-mail

Contact Number  
Enter your contact number

Place  
Enter your place

Feedback  
Provide your feedback

Submit

## STORAGE OF USER FEEDBACKS IN IBM CLOUDANT DATABASE:

Data provided by the users is stored in the feedback database as shown below.

feedback

Document ID Options {} JSON

Create Document

	name	email	contact_num...	place	feedback
<input type="checkbox"/>	Ashwin	ash211234@gmail.com	9102492019	Anna Nagar	Nice
<input type="checkbox"/>	Sam	sam78123@gmail.com	7635790912	Adyar	Satisfied with the service
<input type="checkbox"/>	Abhinav	abhxyz@gmail.com	9103458910	Chennai	Thanks for the chatbot!
<input type="checkbox"/>	John	john174@gmail.com	8567156702	Kancheepuram	Good

## PYTHON CODE FOR FEEDBACK DATA COLLECTION:

```
@app.route('/feedback', methods = ['GET', 'POST'])
def feedback():
    print("*****")
    if request.method == "POST":
        email = request.form['mail']
        name = request.form['name']
        num = request.form['num']
        place = request.form['place']
        msg = request.form['message']
        #print(email, name, num, place, msg)
        jsonDocument = {
            'email': email,
            'name': name,
            'contact_number': num,
            'place': place,
            'feedback': msg
        }
        newDocument = mydatabase.create_document(jsonDocument)
        result = Result(mydatabase.all_docs, include_docs=True)
        print(result[0])
        return redirect(url_for('home'))
        print('####')
    return render_template('feedback.html')
```



## OFFERS:



