

## IMPLEMENTING WEB APPLICATION

Team ID	PNT2022TMID35639
Project Name	Personal Expense Tracker Application

### SCREENSHOT:

The screenshot shows the IBM Db2 on Cloud web interface. The top navigation bar includes 'Load Data', 'Load History', 'Tables', 'Views', 'Indexes', 'Aliases', 'MQTs', 'Sequences', and 'Application objects'. The 'Tables' tab is selected. On the left, a search bar says 'Find schemas or tables'. Below it, the 'Schemas' section shows a table with columns 'Name', 'Type', and 'Tables'. One schema is listed: 'RFH77431' with Type 'User' and 3 tables. At the bottom, it says 'Total: 1, selected: 1'. On the right, the 'Tables' section shows a table with columns 'Name', 'Schema', and 'Properties'. Three tables are listed: 'EXPENSES', 'LIMITS', and 'REGISTER', all under schema 'RFH77431'. At the bottom, it says 'Total: 3, selected: 0'.

The screenshot shows the IBM Db2 on Cloud web interface with the 'Table definition' for the 'EXPENSES' table. The top navigation bar is the same. The 'Tables' tab is selected. On the left, the 'Tables' section shows the same list of tables. On the right, the 'Table definition' section shows the structure of the 'EXPENSES' table. It includes a header row with columns: 'Name', 'Data type', 'Nullable', 'Length', and 'Scale'. Below the header, the table structure is defined:

Name	Data type	Nullable	Length	Scale
EXPENSE_ID	INTEGER	N		0
USER_ID	INTEGER	N		0
DATE	TIMESTAMP	N	10	6
EXPENSE_NAME	VARCHAR	N	30	0
AMOUNT	INTEGER	N		0
PAYMODE	VARCHAR	N	30	0
CATEGORY	VARCHAR	N	30	0

At the bottom of the 'Table definition' section, there is a 'View data' button. The status at the bottom of the 'Tables' section says 'Total: 3, selected: 0'.

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

Name	Schema	Properties
EXPENSES	RFH77431	...
LIMITS	RFH77431	...
REGISTER	RFH77431	...

Total: 3, selected: 0

Table definition

LIMITS

Approximate 5 rows (32.0 KB)  
Updated on 2022-11-20 13:17:39

Name	Data type	Nullable	Length	Scale
LIMIT_ID	INTEGER	N		0
USER_ID	INTEGER	N		0
LIMITSS	INTEGER	N		0

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

Name	Schema	Properties
EXPENSES	RFH77431	...
LIMITS	RFH77431	...
REGISTER	RFH77431	...

Total: 3, selected: 0

Table definition

REGISTER

Approximate 16 rows (32.0 KB)  
Updated on 2022-11-14 10:27:39

Name	Data type	Nullable	Length	Scale
USER_ID	INTEGER	N		0
USERNAME	VARCHAR	N	255	0
EMAIL	VARCHAR	Y	255	0
PASSWORD	VARCHAR	Y	255	0

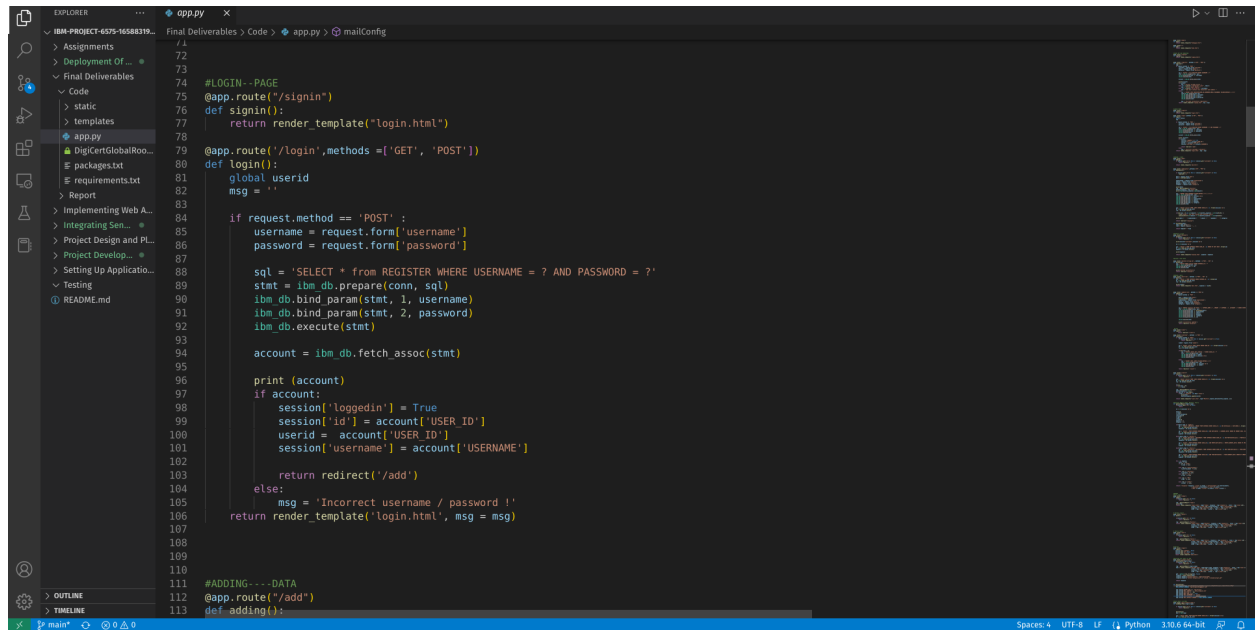
View data

```
EXPLORER
IBM-PROJECT-6575-16588319
> Assignments
> Deployment Of...
> Final Deliverables
  > Code
    > static
    > templates
  app.py
  DigICertGlobalRoo...
  packages.txt
  requirements.txt
  Report
> Implementing Web A...
> Integrating Sen...
> Project Design and PL...
> Project Develop...
> Setting Up Applicatio...
> Testing
  README.md
  OUTLINE
  TIMELINE

app.py
1 from flask import Flask, render_template, request, redirect, session, make_response, url_for, jsonify, flash
2 import re, ibm_db
3 import ibm_db_dbi, pandas as pd
4 from flask_mail import Mail, Message
5 import os, datetime
6 from pandas import Timestamp
7 from pretty_html_table import build_table
8 import pdfkit
9
10 app = Flask(__name__)
11
12 app.secret_key = 'SECRET_KEY'
13
14 conn = ibm_db.connect("DATABASE=bludb; HOSTNAME=fbd88901-ebdb-4a4f-a32e-9822b9fb237b.clogj3sd0tgtu0lqde00.databases.appdomain.cloud; PORT=32731; SECURITY=SSL;");
15 pd_conn = ibm_db_dbi.Connection(conn)
16
17 #HOME--PAGE
18 @app.route("/")
19 def home():
20     return render_template("homepage.html")
21
22 @app.route("/")
23 def add():
24     return render_template("home.html")
25
26 #SIGN-UP--OR--REGISTER
27 @app.route("/signup")
28 def signup():
29     return render_template("signup.html")
30
31 @app.route('/register', methods=['GET', 'POST'])
32 def register():
33     msg = ''
34     if request.method == 'POST':
35         username = request.form['username']
36         email = request.form['email']
37
38         sql = 'SELECT * FROM REGISTER WHERE USERNAME = ?'
39         stmt = ibm_db.prepare(conn, sql)
40         ibm_db.bind_param(stmt, 1, username)
41         ibm_db.execute(stmt)
42
43         account = ibm_db.fetch_assoc(stmt)
44
45         if account:
46             msg = 'Account already exists !'
47         elif not re.match(r'[^@]+\@[^@]+\.[^@]+', email):
48             msg = 'Invalid email address !'
49         elif not re.match(r'[A-Za-z0-9]+', username):
50             msg = 'name must contain only characters and numbers !'
51         else:
52             sql = 'INSERT INTO REGISTER(USER_ID,USERNAME,EMAIL,PASSWORD) VALUES(DEFAULT,?,?,?)'
53             stmt=ibm_db.prepare(conn,sql)
54             ibm_db.bind_param(stmt,1,username)
55             ibm_db.bind_param(stmt,2,email)
56             ibm_db.bind_param(stmt,3,password)
57             ibm_db.execute(stmt)
58
59             msg = 'You have successfully registered !'
60             return render_template('signup.html', msg = msg)
61
62 #LOGIN--PAGE
63 @app.route("/signin")
64 def signin():
65     return render_template("login.html")
66
67 @app.route('/login', methods=['GET', 'POST'])
```

```
EXPLORER
IBM-PROJECT-6575-16588319
> Assignments
> Deployment Of...
> Final Deliverables
  > Code
    > static
    > templates
  app.py
  DigICertGlobalRoo...
  packages.txt
  requirements.txt
  Report
> Implementing Web A...
> Integrating Sen...
> Project Design and PL...
> Project Develop...
> Setting Up Applicatio...
> Testing
  README.md
  OUTLINE
  TIMELINE

app.py
38 @app.route('/register', methods=['GET', 'POST'])
39 def register():
40     msg = ''
41     if request.method == 'POST':
42         username = request.form['username']
43         email = request.form['email']
44         password = request.form['password']
45
46         sql = 'SELECT * FROM REGISTER WHERE USERNAME = ?'
47         stmt = ibm_db.prepare(conn, sql)
48         ibm_db.bind_param(stmt, 1, username)
49         ibm_db.execute(stmt)
50
51         account = ibm_db.fetch_assoc(stmt)
52
53         print(account)
54         if account:
55             msg = 'Account already exists !'
56         elif not re.match(r'[^@]+\@[^@]+\.[^@]+', email):
57             msg = 'Invalid email address !'
58         elif not re.match(r'[A-Za-z0-9]+', username):
59             msg = 'name must contain only characters and numbers !'
60         else:
61             sql = 'INSERT INTO REGISTER(USER_ID,USERNAME,EMAIL,PASSWORD) VALUES(DEFAULT,?,?,?)'
62             stmt=ibm_db.prepare(conn,sql)
63             ibm_db.bind_param(stmt,1,username)
64             ibm_db.bind_param(stmt,2,email)
65             ibm_db.bind_param(stmt,3,password)
66             ibm_db.execute(stmt)
67
68             msg = 'You have successfully registered !'
69             return render_template('signup.html', msg = msg)
70
71
72
73 #LOGIN--PAGE
74 @app.route("/signin")
75 def signin():
76     return render_template("login.html")
77
78 @app.route('/login', methods=['GET', 'POST'])
```



The image shows a Visual Studio Code editor window with a Python file named `app.py` open. The file contains a Flask application with a login route. The code is as follows:

```
72 #!/usr/bin/env python
73 # coding: utf-8
74 # LOGIN - PAGE
75 @app.route("/signin")
76 def signin():
77     return render_template("login.html")
78
79 @app.route('/login', methods=['GET', 'POST'])
80 def login():
81     global userid
82     msg = ''
83
84     if request.method == 'POST':
85         username = request.form['username']
86         password = request.form['password']
87
88         sql = 'SELECT * from REGISTER WHERE USERNAME = ? AND PASSWORD = ?'
89         stmt = ibm_db.prepare(conn, sql)
90         ibm_db.bind_param(stmt, 1, username)
91         ibm_db.bind_param(stmt, 2, password)
92         ibm_db.execute(stmt)
93
94         account = ibm_db.fetch_assoc(stmt)
95
96         print (account)
97         if account:
98             session['logged_in'] = True
99             session['id'] = account['USER ID']
100             userid = account['USER ID']
101             session['username'] = account['USERNAME']
102
103             return redirect('/add')
104         else:
105             msg = 'Incorrect username / password !'
106     return render_template("login.html", msg = msg)
107
108
109
110
111 #ADDING---DATA
112 @app.route("/add")
113 def adding():
```

The code defines a `login` function that handles both GET and POST requests. For POST requests, it extracts the username and password from the request form, constructs an SQL query to check the credentials in a database named `REGISTER`, and uses `ibm_db` to execute the query. If the credentials are correct, it sets session variables and redirects to the `/add` route. Otherwise, it displays an error message. The code also includes a commented-out `adding` function.