PHASE 4: Development Part 2

To build a mobile app that displays real-time parking availability data received from a Raspberry Pi using HTML, CSS, JavaScript, Flask, and IBM Cloud DB2 follow the below steps.

STEPS:

Set Up the Development Environment:

- Install the necessary tools and frameworks for mobile app development. For this project, you will need Python, Flask, and a code editor of your choice.

Design the User Interface:

- Create a user-friendly design for your mobile app. Use HTML for structuring the app, CSS for styling, and JavaScript for interactivity. Make sure to have a section where parking availability data will be displayed.

Client-Side App Development:

- Write HTML to create the structure of your app. Design a user interface that includes buttons, labels, and any other necessary elements.
- Use CSS to style your app, making it visually appealing and easy to use.
- Write JavaScript to handle user interactions and make the app dynamic. Create functions that will request parking availability data from the server (Flask) and update the UI with real-time information when the data is received.

Server-Side App Development with Flask:

- Set up a Flask application to serve as your server. Flask is a Python web framework that can handle HTTP requests and responses.
- Create API endpoints in Flask that can receive and process data from the Raspberry Pi. This data should include real-time parking availability information.
- Establish a connection to the IBM Cloud DB2 database to store and retrieve parking availability data.

Database Integration with IBM Cloud DB2:

- Create a DB2 service instance on the IBM Cloud platform.

- Configure the Flask application to connect to the IBM Cloud DB2 database to store and retrieve parking availability data. You will need to provide credentials and connection details to establish this connection.

Real-time Data Update:

- Implement a mechanism in your Flask application to continuously update the parking availability data from the Raspberry Pi. This can be done using web sockets or other real-time data transfer methods.
- When new data is available, update the database and make it accessible to the mobile app through the API endpoints.

Data Display and Interaction:

- Use JavaScript in your mobile app to make periodic requests to the Flask server to retrieve parking availability data.
- Update the user interface with the latest parking availability information, ensuring that the app displays real-time data.
- Consider adding features such as notifications or alerts when
- parking spaces become available or limited.

Testing and Debugging:

Thoroughly test your mobile app to ensure that it works as expected. Check for any bugs or issues in both the client-side and server-side components.

- Make sure the app can handle real-time data updates and display them accurately.

Deployment:

- Deploy your Flask application to a web server or cloud platform, making it accessible to mobile devices.
- Make the mobile app available for download or access through a URL.

User Documentation:

- Create user documentation or guides to help users understand how to use the app effectively. Explain how to access real-time parking availability data and any other features.

Maintenance and Updates:

- Regularly monitor the app's performance and data accuracy.
- Consider adding additional features or making improvements based on user feedback and changing requirements.

This overview provides a high-level explanation of the steps involved in building a mobile app that displays real-time parking availability data from a Raspberry Pi using HTML, CSS, JavaScript, Flask, and IBM Cloud DB2. The actual implementation will require writing code, configuring the server, and handling specific technical details, which may vary depending on your specific project requirements and technology stack.

“Below are the sample application code and screenshot of the application”.

CODE:

SAMPLE HTML FILE:

#index.html

```
{% extends "base.html" %}

{% block content %}
   <h2>Welcome to Your Web App</h2>
   <p>This is the home page.</p>
   <a href="{{ url_for('login') }}">Login</a> or <a href="{{
url_for('register') }}">Register</a>
{% endblock %}
```

#login.html

```
{% extends "base.html" %}

{% block content %}
   <h2>Login</h2>
   <form method="post">
       <label for="email">Email:</label>
       <input type="text" id="email" name="email" required>
       <label for="password">Password:</label>
       <input type="password" id="password" name="password"
required>
       <button type="submit">Login</button>
   </form>
   <p>Don't have an account? <a href="{{ url_for('register')
}}">Register</a></p>
{% endblock %}
```

#register.html

```
{% extends "base.html" %}

{% block content %}
   <h2>Register</h2>
   <form method="post">
       <label for="email">Email:</label>
       <input type="text" id="email" name="email" required>
       <label for="password">Password:</label>
       <input type="password" id="password" name="password"
required>
       <button type="submit">Register</button>
   </form>
   <p>Already have an account? <a href="{{ url_for('login')
}}">Login</a></p>
{% endblock %}
```

APP.PY FILE:

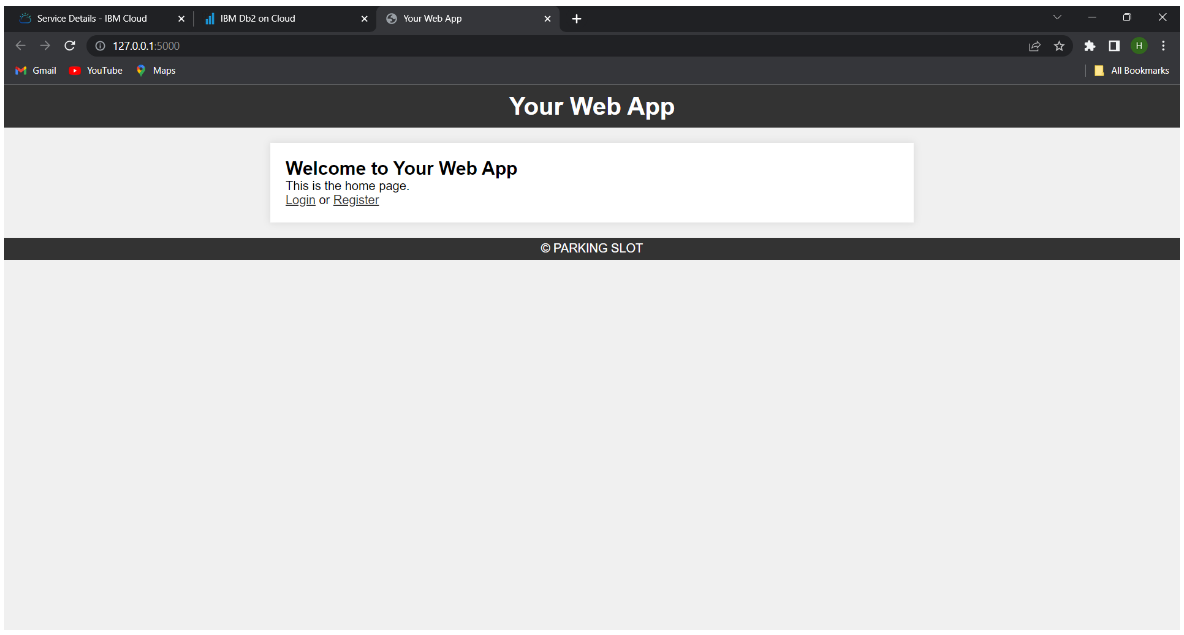
“The below code contains framework of flask in python and the connection of IBM Cloud DB2 database”.

```
from flask import *
import ibm_db as ibm

app = Flask(__name__)
app.secret_key = "this is very confidential"

connection = ibm.connect(

"DATABASE=bludb;HOSTNAME=55fbc997-9266-4331-afd3-888b05e734c0.bs2i
o90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31929;SECURITY=SSL
;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=qhp77202;PWD=xK
Dt1Nphp7i1BGAl;",
   "", "")

print("CONNECTED!")


@app.route('/')
def index():
   return render_template('index.html')
```

```
@app.route('/register', methods=['GET', 'POST'])
def register():
   if request.method == 'POST':
       username = request.form['email']
       password = request.form['password']

       if connection:
           query = "INSERT INTO register (email, password) VALUES
(?, ?)"
           stmt = ibm.prepare(connection, query)
           ibm.bind_param(stmt, 1, username)
           ibm.bind_param(stmt, 2, password)
           result = ibm.fetch_assoc(stmt)
           ibm_db.close(connection)

           if result:
               flash('Registration successful!', 'success')
               return render_template('login.html')
           else:
               flash('Registration failed. Please try again.',
'error')
       else:
           flash('Failed to connect to the IBM Db2 Cloud
database.', 'error')

   return render_template('register.html')


@app.route('/index', methods=['GET', 'POST'])
def login():
   if request.method == 'POST':
       username = request.form['email']
       password = request.form['password']

       if connection:
           query = "SELECT * FROM REGISTER WHERE EMAIL = ? AND
PASSWORD = ?"
           stmt = ibm.prepare(connection, query)
           result = ibm.execute(stmt, (username, password))
           user = ibm.fetch_assoc(stmt)

           if user:
               session['user_id'] = user['EMAIL']
               flash('Login successful!', 'success')
               return render_template('dashboard.html')
           else:
               flash('Login failed. Please check your
credentials.', 'error')
       else:
           flash('Failed to connect to the IBM Db2 Cloud
database.', 'error')

   return render_template('login.html')


@app.route('/parking_availability', methods=['GET'])
def parking_availability():
   if connection:
       try:
           query = "SELECT slot_1, slot_2, slot_3, slot_4 FROM
availability"
           stmt = ibm.exec_immediate(connection, query)
           result = ibm.fetch_assoc(stmt)
           if result:
               slot_1 = result['SLOT_1']
               slot_2 = result['SLOT_2']
               slot_3 = result['SLOT_3']
               slot_4 = result['SLOT_4']

               parking_status = {
                   "Slot 1": "Occupied" if slot_1 == 1 else
"Vacant",
                   "Slot 2": "Occupied" if slot_2 == 1 else
"Vacant",
                   "Slot 3": "Occupied" if slot_3 == 1 else
"Vacant",
                   "Slot 4": "Occupied" if slot_4 == 1 else
"Vacant",
               }

               flash('Parking availability:', 'success')
               return render_template('dashboard.html',
parking_status=parking_status)
               print("Availability fetched")
           else:
               flash('No data found in the availability table.',
'error')
       except Exception as e:
           flash(f'Error: {str(e)}', 'error')
   else:
       flash('Failed to connect to the IBM Db2 Cloud database.',
'error')
```

```
    return render_template('dashboard.html')


if __name__ == "__main__":
    app.run(debug=True)
```

HOME PAGE:

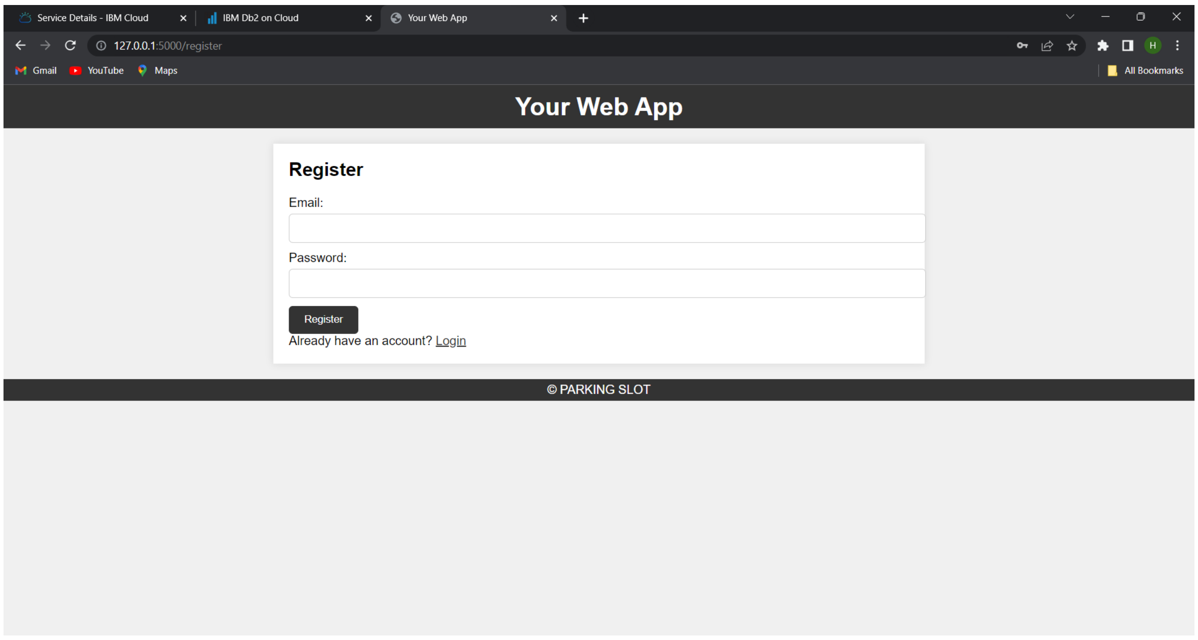


Your Web App

Welcome to Your Web App
This is the home page.
Login or Register

© PARKING SLOT

Registration page:

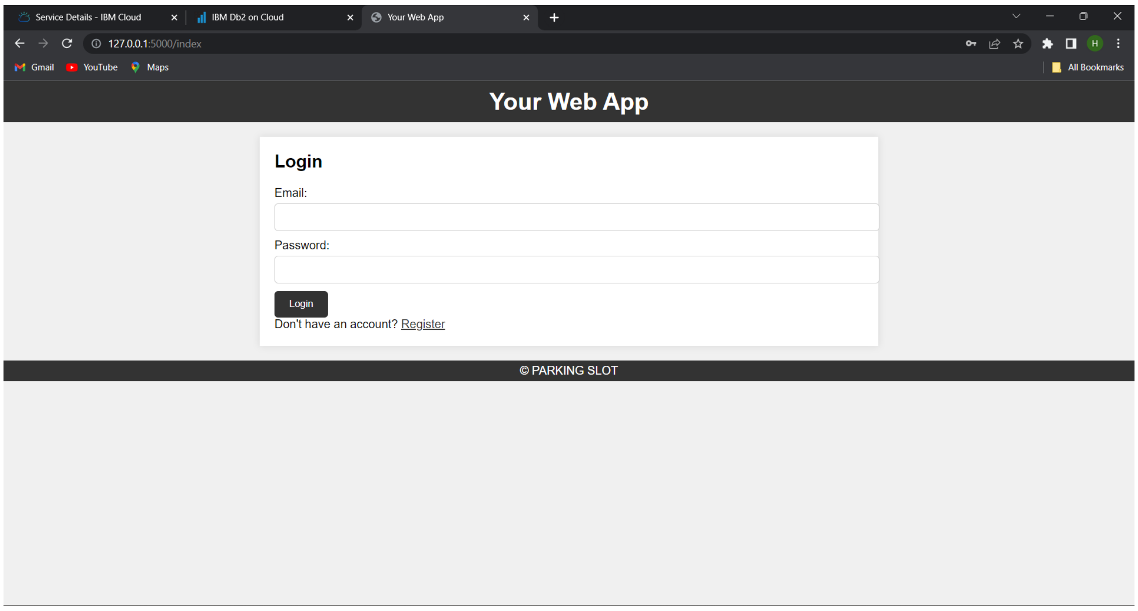


Your Web App

Register
Email:
Password:
Register
Already have an account? Login

© PARKING SLOT

Login page:



Your Web App

Login

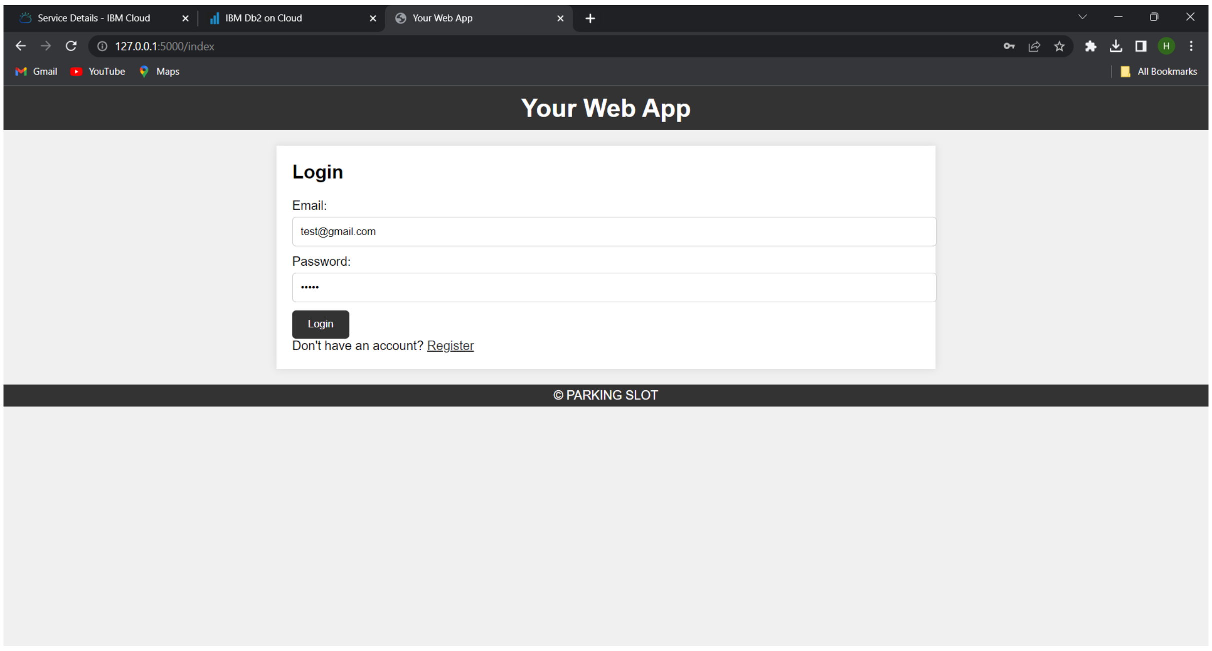
Email:

Password:

Login

Don't have an account? Register

© PARKING SLOT



Your Web App

Login

Email: test@gmail.com

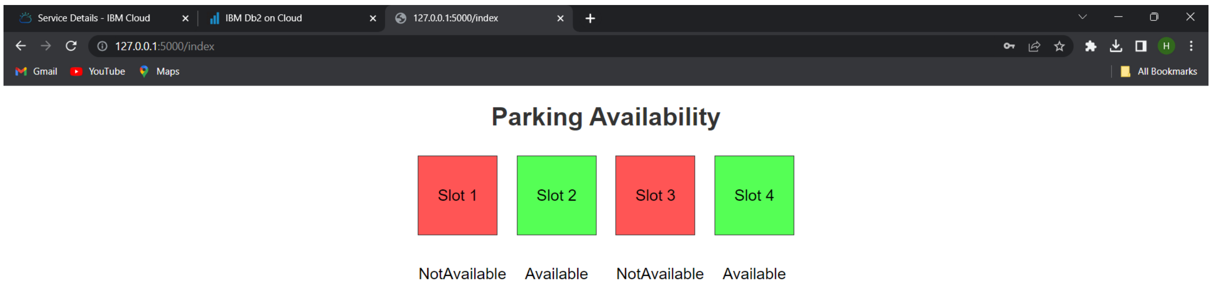
Password: •••••

Login

Don't have an account? Register

© PARKING SLOT

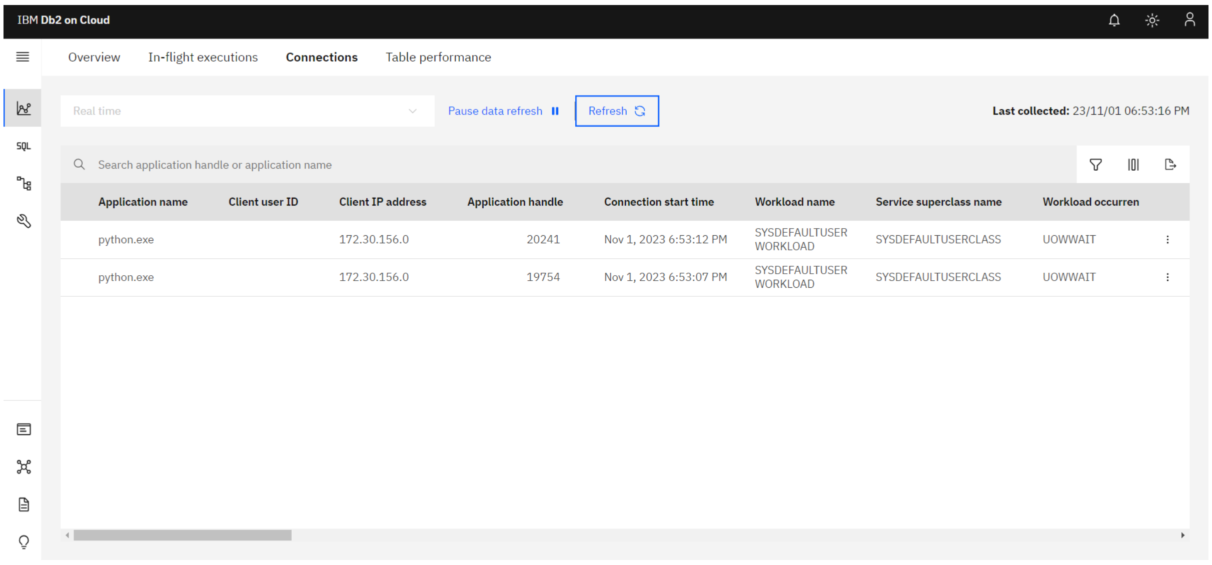
Parking Availability:



Parking Availability

Slot 1	Slot 2	Slot 3	Slot 4
NotAvailable	Available	NotAvailable	Available

CLOUD CONNECTION:



Last collected: 23/11/01 06:53:16 PM

Application name	Client user ID	Client IP address	Application handle	Connection start time	Workload name	Service superclass name	Workload occurren
python.exe		172.30.156.0	20241	Nov 1, 2023 6:53:12 PM	SYSDEFAULTUSER WORKLOAD	SYSDEFAULTUSERCLASS	UOWWAIT
python.exe		172.30.156.0	19754	Nov 1, 2023 6:53:07 PM	SYSDEFAULTUSER WORKLOAD	SYSDEFAULTUSERCLASS	UOWWAIT

“The application contains the space availability to park vehicles which minimize the time for finding the space to park”

CONCLUSION:

The web/app application for smart parking has been successfully implemented with the help of cloud connection.