API REST con Flask

Samuel Martín Morales alu0101359526@ull.edu.es
Jorge Domínguez González alu0101330600@ull.edu.es

10 de diciembre de 2023

Índice general

Capítulo 1 Actividad 1

1.1. Instalación de Flask

Para la implementación de la práctica número 6 de la asignatura *Administración y Diseño de Bases de Datos*, se ha decidido utilizar el lenguaje de programación *Python* y el *framework Flask* para la creación de una API REST que permita realizar operaciones sobre una serie de bases de datos que podrán ser observadas durante el desarrollo de dicha práctica.

Antes de comenzar con el desarrollo de la API REST, es necesario instalar el *framework Flask* en el sistema. Para ello, se ha utilizado el gestor de paquetes *pip* que viene instalado por defecto en las versiones más recientes de *Python*.

Para poder instalar *Flask* utilizando *pip*, es necesario ejecutar los siguientes comandos en la terminal:

```
$ python3 -m pip install flask
$ python3 -m pip install psycopg2-binary
```

Nota: Cabe destacar que los comandos anteriores se pueden instalar en la máquina local o en un entorno virtual de *Python*. En este caso, se ha decidido instalarlos en la máquina local.

1.2. Despliegue de aplicación Web

Para el despliegue de la aplicación denominada como 'app.py' (`se encuentra desarrollada de manera previa`) , la cual hace uso de flask para implementar una API REST que permite en esta ocasión realizar operaciones **CRUD** sobre una base de datos de PostgreSQL, se hace uso del siguiente comando:

```
$ flask - -app app.py run - -host 0.0.0.0 - -port 8080
```

Una vez se tiene el despliegue de una aplicación web con flask, se realiza la implementación de la base de datos de PostgreSQL.

1.3. Creación de la base de datos

Para la implementación de la base de datos de PostgreSQL, en primer lugar se debe de crear la base de datos dentro del propio gestor de base de datos, haciendo uso de lo siguientes comandos:

```
$ sudo -i -u postgres
$ psql
```

Una vez nos encontramos dentro de la interfaz interactiva de *PostgreSQL*, se procede a la creación de la base de datos denominada como `flask_db` haciendo uso de la sentencia:

```
CREATE DATABASE flask_db;
```

Una vez se tiene creada la base de datos dentro de PostgreSQL, se procede a la inserción de al menos 9 registros dentro esta, es por ello que, se hace uso del script denominado como *init_db*, el cual se encuentra dentro de la carpeta de recursos de la práctica 7. Una vez se tiene dicho script, se realiza la modificación de este de la siguiente manera para que cumpla los requisitos de la insersión de los comentados 9 registros y el control de excepciones en el código.

```
import os
import psycopg2

conn = psycopg2.connect(
        host="127.0.0.1",
        database="flask_db",
        #user=os.environ['DB_USERNAME'],
        #password=os.environ['DB_PASSWORD']
    user='postgres',
        password='tibYDKQ8')

# Open a cursor to perform database operations
cur = conn.cursor()

# Execute a command: this creates a new table
cur.execute('DROP TABLE IF EXISTS books;')
cur.execute('CREATE TABLE books (id serial PRIMARY KEY,'
                                 'title varchar (150) NOT NULL,'
                                 'author varchar (50) NOT NULL,'
                                 'pages_num integer NOT NULL,'
                                 'review text,'
                                 'date_added date
                                   DEFAULT CURRENT_TIMESTAMP);'
                                 )

# Insert data into the table

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('A Tale of Two Cities',
            'Charles Dickens',
            489,
            'A great classic!')
            )


cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('Anna Karenina',
            'Leo Tolstoy',
            864,
            'Another great classic!')
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('The Great Gatsby',
            'F. Scott Fitzgerald',
            218,
            'Another great classic!')
```

```
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('One Hundred Years of Solitude',
            'Gabriel García Márquez',
            417,
            'Another great classic!')
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('A Passage to India',
            'E. M. Forster',
            378,
            'A great classic!')
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('The Adventures of Huckleberry Finn',
            'Mark Twain',
            366,
            'A great classic!')
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('The Catcher in the Rye',
            'J. D. Salinger',
            234,
            'Another great classic!')
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('The Grapes of Wrath',
            'John Steinbeck',
            464,
            'A great classic!')
            )

cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('The Great Gatsby',
            'F. Scott Fitzgerald',
            218,
            'Another great classic!')
            )
```

```
cur.execute('INSERT INTO books (title, author, pages_num, review)'
            'VALUES (%s, %s, %s, %s)',
            ('The Odyssey',
            'Homer',
            213,
            'A great classic!')
            )

# Exception handling example
try:
    cur.execute('SELECT * FROM books;')
    books = cur.fetchall()
    print(books)
except psycopg2.Error as e:
    print(f'ERROR: {e}')
    conn.rollback()

conn.commit()

cur.close()
conn.close()
```

Una vez se tiene el script modificado, se procede a ejecutarlo de la siguiente manera:

```
$ python3 init_db.py
```

Tras estos pasos, se tiene la base de datos implementada y con registros dentro de esta.

1.4. Creación de la página de *about*

Con todo esto anteriormente realizado, se procede a la creación de la página de *about* haciendo uso de *Flask*. Para ello, se crea una nueva ruta dentro del fichero `app.py` de la siguiente manera:

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

Una vez se tiene la ruta creada, se procede a la creación de la plantilla `about.html` dentro de la carpeta `templates` de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} About us {% endblock %}</h1>
    <p> This is the about page </p>
    <h2> {% block subtitle %} Project Components {% endblock %}</h2>
    <p> Samuel Martín Morales --> alu0101359526@ull.edu.es </p>
    <p> Jorge Domínguez  González --> alu0101330600@ull.edu.es </p>
{% endblock %}
```

Una vez se tiene la plantilla creada, se procede a la modificación de la plantilla `base.html` dentro de la carpeta `templates` de la siguiente manera:

```
<a href="{{ url_for('about') }}">About</a>
```

1.5. Chequeo de excepciones de la página de *Visualización de registros*

En cuanto al funcionamiento de la operación de visualización de los registros de la base de datos es correcta, es decir, se muestran todos los registros de la base de datos de manera correcta. Por tanto, se realiza la implementación del control de excepciones en el código de la siguiente manera:

```
def index():
  try:
      conn = get_db_connection()
      cur = conn.cursor()
      cur.execute('SELECT * FROM books;')
      books = cur.fetchall()
      cur.close()
      conn.close()
      return render_template('index.html', books=books)
  except Exception as e:
      current_app.logger.error
        (f"Error en la consulta a la base de datos: {e}")
      return render_template('error.html', error_type=type(e).__name__,
        error_message=str(e)), 500
```

Por otra parte, se puede observar a continuación la implementación de una página de error en el caso de que se produzca un error en la base de datos:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Error en la aplicación</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }

        .error-container {
            text-align: center;
        }

        h1 {
            color: #d64161;
        }

        p {
            color: #333;
        }
    </style>
</head>
```

```
<body>
    <div class="error-container">
        <h1>Error en la aplicación</h1>
        <p>Ocurrió un error de tipo <strong>{{ error_type }}</strong>.</p>
        <p>Detalle del error: <em>{{ error_message }}</em></p>
        <p>Por favor, inténtalo de nuevo más tarde.</p>
    </div>
</body>
</html>
```

1.6. Chequeo de excepciones de la página de *Inserción de registros*

Tras la revisión del funcionamiento de la página denominada como *create*, es decir, la página de inserción de nuevos registros dentro de la base de datos, se puede observar que el funcionamiento de esta es correcto, es decir, se insertan nuevos registros dentro de la base de datos de manera apropiada. Por tanto, se realiza la implementación del control de excepciones en el código de la siguiente manera:

```
def create():
if request.method == 'POST':
    try:
        title = request.form['title']
        author = request.form['author']
        pages_num = int(request.form['pages_num'])
        review = request.form['review']

        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('INSERT INTO books (title, author, pages_num, review)'
                    'VALUES (%s, %s, %s, %s)',
                    (title, author, pages_num, review))
        conn.commit()
        cur.close()
        conn.close()
        return redirect(url_for('index'))
    except Exception as e:
        # Manejar la excepción de manera adecuada
        current_app.logger.error
          (f"Error al insertar datos en la base de datos: {e}")
        return render_template('error.html', error_type=type(e).__name__,
          error_message=str(e)), 500

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

1.7. Creación de la operación de borrado de registros

Para la construcción de la nueva operación de *borrado* dentro de la API REST, en primer lugar se debe de implementar la siguiente función dentro del archivo ‘app.py‘:

```
@app.route('/delete/', methods=('GET', 'POST'))
def delete():
    if request.method == 'POST':
        try:
            id = request.form['id']

            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute('DELETE FROM books WHERE id = %s', (id,))
            conn.commit()
            cur.close()
            conn.close()
            return redirect(url_for('index'))
        except Exception as e:
            current_app.logger.error
              (f"Error al eliminar datos de la base de datos: {e}")
            return render_template('error.html', error_type=type(e).__name__,
              error_message=str(e)), 500

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

Para continuar y de manera final, se realiza la creación de un nuevo fichero denominado como *delete.html*, de tal manera que permite realizar la operación de delete dentro del frontend que se está usando para poder realizar las distintas operaciones de la API REST:

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

{% block content %}
    <h1>{% block title %} Delete a Book {% endblock %}</h1>
    <form method="post">
        <p>
          <label for="id">ID</label>
          <input type="text" name="id"
                  placeholder="ID of the book">
            </input>
        </p>
        <p>
          <button type="submit">Submit</button>
      </p>
    </form>
{% endblock %}
```

1.8. Creación de la operación de actualización de registros

Para la construcción de la nueva operación de *actualización* dentro de la API REST, en primer lugar se debe de implementar la siguiente función dentro del archivo 'app.py':

```
@app.route('/update/', methods=('GET', 'POST'))
def update():
```

```
    if request.method == 'POST':
        try:
            id = request.form['id']
            title = request.form['title']
            author = request.form['author']
            pages_num = int(request.form['pages_num'])
            review = request.form['review']

            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute
              ('UPDATE books SET title = %s, author = %s, pages_num = %s, review = %s WHERE id
                        (title, author, pages_num, review, id))
            conn.commit()
            cur.close()
            conn.close()
            return redirect(url_for('index'))
        except Exception as e:
            current_app.logger.error
              (f"Error al actualizar datos en la base de datos: {e}")
            return render_template('error.html', error_type=type(e).__name__,
              error_message=str(e)), 500

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

Para finalizar, se implementa un nuevo fichero denominado como *update.html* de tal manera que permite realizar la operación de update dentro del frontend que se está usando para poder realizar las distintas operaciones de la API REST:

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

{% block content %}
    <h1>{% block title %} Update a Book {% endblock %}</h1>
    <form method="post">
        <p>
          <label for="id">ID</label>
          <input type="text" name="id"
                 placeholder="ID of the book">
            </input>
        </p>
        <p>
          <label for="title">Title</label>
          <input type="text" name="title"
                 placeholder="Title of the book">
            </input>
        </p>
        <p>
          <label for="author">Author</label>
          <input type="text" name="author"
                 placeholder="Author of the book">
            </input>
```

```
        </p>
        <p>
          <label for="pages_num">Pages Number</label>
          <input type="text" name="pages_num"
                  placeholder="Pages number of the book">
            </input>
        </p>
        <p>
          <label for="review">Review</label>
          <input type="text" name="review"
                  placeholder="Review of the book">
            </input>
        </p>
        <p>
          <button type="submit">Submit</button>
      </p>
    </form>
{% endblock %}
```

Capítulo 2 Actividad 2

2.1. Despliegue de la base de datos *myhome*

Para el despligue de dicho script se hace uso del siguiente comando:

```
[language=bash]
  $ psql
  $ \i MyHome.sql
```

De esta forma se crea la base de datos denominada *myhome*, junto con sus respectivas tablas y sus respectivos registros.

2.2. Creación de una API REST haciendo uso de Flask

2.2.1. Creación de la operación de *listado de registros*

Para comenzar con la construcción de una REST API para la base de datos 'myhome', se realiza la implementación de una página principal, que permita visualizar todos los registros de las habitaciones de la base de datos, de la siguiente manera:

```
@app.route('/')
def index():
    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT * FROM temperatures;')
        temperature = cur.fetchall()
        cur.close()
        conn.close()
        return render_template('index.html', rooms=temperature)
    except Exception as e:
        current_app.logger.error
          (f"Error en la consulta a la base de datos: {e}")
        return render_template('error.html', error_type=type(e).__name__,
          error_message=str(e)), 500
```

Teniendo en cuenta esto anterior, el archivo *index.html* se desarrolla de la siguiente manera para poder visualizar dichos registros:

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

{% block content %}
    <h1>{% block title %} Rooms {% endblock %}</h1>
    {% for room in rooms %}
        <div class='room'>
            <h3>#{{ room[0] }} - Room {{ room[1] }}
```

```
            temperature: {{ room[2] }}</h3>
          <i><p> On ({{ room[3] }})</p></i>
        </div>
    {% endfor %}
{% endblock %}
```

2.2.2. Creación de la operación de *cálculo de la media de temperatura*

Una vez se tiene la página principal, se procede a realizar la implementación de la página que se encarga de mostrar la temperatura media de todas las habitaciones que se encuentran dentro de la base de datos, para ello:

```
@app.route('/average/')
def average():
    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT AVG(temperature) FROM temperatures;')
        average = cur.fetchall()
        cur.close()
        conn.close()
        average = average[0][0]
        return render_template('average.html', average=average)
    except Exception as e:
        current_app.logger.error
          (f"Error en la consulta a la base de datos: {e}")
        return render_template
          ('error.html', error_type=type(e).__name__,
            error_message=str(e)), 500
```

Una vez se realiza la consulta obteniendo la temperatura media de todas las habitaciones que se encuentran dentro de la base de datos, se realiza la implementación del archivo *average.html* de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Average {% endblock %}</h1>
    <p> The average of the temperature of all the rooms is:
      {{ average }} </p>
{% endblock %}
```

2.2.3. Creación de la operación de *cálculo del máximo de temperatura*

Para la implementación de la operación de cálculo del máximo de temperatura de todas las habitaciones que se encuentran dentro de la base de datos, se realiza la implementación de la siguiente función dentro del archivo *app.py*:

```
@app.route('/max/')
def max():
    try:
        conn = get_db_connection()
        cur = conn.cursor()
```

```
        cur.execute('SELECT MAX(temperature) FROM temperatures;')
        max = cur.fetchall()
        cur.close()
        conn.close()
        max = max[0][0]
        return render_template('max.html', max=max)
    except Exception as e:
        current_app.logger.error
          (f"Error at the data base consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
        error_message=str(e)), 500
```

Una vez se realiza la consulta obteniendo la temperatura máxima de todas las habitaciones que se encuentran dentro de la base de datos, se realiza la implementación del archivo *max.html* de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Maximum {% endblock %}</h1>
    <p>The maximum temperute on the rooms is: {{ max }}</p>
{% endblock %}
```

2.2.4. Dado un identificador retornar el nombre de la habitación

Para poder implementar la página que se encarga de mostrar el nombre de la habitación de partir del ID de esta que para el caso particular que se decide implementar, se obtiene a partir de un desplegable que te permite seleccionar el identificador de todos los existentes dentro de la base de datos, para ello se realiza la siguiente implementación:

```
@app.route('/room/', methods=['GET'])
def room_id():
    try:
        if request.method == 'GET':
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute('SELECT id FROM rooms;')
            room_id = cur.fetchall()
            cur.close()
            conn.close()
            return render_template('room_id.html', room_id=room_id)
    except Exception as e:
        current_app.logger.error(f"Error at the data base consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
        error_message=str(e)), 500
```

Dentro del HTML se hace uso del siguiente desarrollo:

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

{% block content %}
    <h1>{% block title %} Select a Room ID {% endblock %}</h1>
```

```
    <form action="/room/result/" method="post">
        <label for="room_id">Select an ID:</label>
        <select name="room_id" id="room_id">
            {% for id in room_id %}
                <option value="{{ id[0] }}">{{ id[0] }}</option>
            {% endfor %}
        </select>
        <br>
        <input type="submit" value="Select">
    </form>
{% endblock %}
```

Una vez el usuario realiza la selección de uno de los identificadores de la lista, se procede a obtener el nombre de la habitación a partir del ID seleccionado, para ello se realiza la siguiente implementación:

```
@app.route('/room/result/', methods=['POST'])
def room_result():
    try:
        room_id = request.form['room_id']
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT name FROM rooms WHERE id=%s;', (room_id,))
        roomname = cur.fetchone()[0]
        cur.close()
        conn.close()
        return render_template('room_name.html', room_name=roomname)
    except Exception as e:
        current_app.logger.error(f"Error at the database consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
          error_message=str(e)), 500
```

En cuanto al frontend, se realiza la implementación de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Room Name {% endblock %}</h1>
    <p>The name of the room that you selected is: {{ room_name }}</p>
{% endblock %}
```

2.2.5. Dado un identificador retornar la temperatura media de la habitación

Para la realización del cuarto punto a implantar dentro de la REST API, se debe de implementar de manera similar a la anterior, pero para este caso, se debe de devolver la media histórica de la temperatura de una habitación a partir del ID de esta, para ello, en primer lugar se obtiene una lista con cada uno de los distintos identificadores de la habitaciones que existen dentro de la base de datos:

```
@app.route('/room_temperature_id/', methods=['GET'])
def room_temperature_id():
    try:
        if request.method == 'GET':
            conn = get_db_connection()
```

```
            cur = conn.cursor()
            cur.execute('SELECT id FROM rooms;')
            room_id = cur.fetchall()
            cur.close()
            conn.close()
            return render_template('room_temperature_id.html',
              room_id=room_id)
    except Exception as e:
        current_app.logger.error
          (f"Error at the data base consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
          error_message=str(e)), 500
```

Una vez se tiene la lista de identificadores, se procede a realizar la implementación del HTML de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Select a Room ID to calculate the average
      temperature {% endblock %}</h1>
    <form action="/room_temperature_id/result/" method="post">
        <label for="room_id">Select an ID:</label>
        <select name="room_id" id="room_id">
            {% for id in room_id %}
                <option value="{{ id[0] }}">{{ id[0] }}</option>
            {% endfor %}
        </select>
        <br>
        <input type="submit" value="Select">
    </form>
{% endblock %}
```

Para finalizar, se realiza la petición a la base de datos para obtener la media histórica de la temperatura de la habitación seleccionada por el usuario:

```
@app.route('/room_temperature_id/result/', methods=['POST'])
def room_temperature_id_result():
    try:
        roomID = request.form['room_id']
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT AVG(temperature)
          FROM temperatures WHERE room_id=%s;', (roomID,))
        temperature = cur.fetchall()
        cur.close()
        conn.close()
        temperature = temperature[0][0]
        return render_template('room_temperature.html',
          room_id=roomID,room_temperature=temperature)
    except Exception as e:
        current_app.logger.error
```

```
            (f"Error at the data base consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
            error_message=str(e)), 500
```

Una vez se tiene la temperatura media de la habitación seleccionada por el usuario, se realiza la implementación del HTML de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Room average temperature {% endblock %}</h1>
    <p>The room number {{ room_id }} has the historic
      average temperature of: {{ room_temperature }}</p>
{% endblock %}
```

2.2.6. Dado un identificador retornar la temperatura mínima de la habitación en formato JSON

Para el quinto y último punto de la segunda actividad, se debe de implementar una página que devuelva la temperatura mínima de la habitación y su nombre en formato JSON, de tal manera que a partir del ID de la habitación se pueda obtener dicha información, para ello se realiza la siguiente implementación:

```
@app.route('/room_temperature_id_min/', methods=['GET'])
def room_temperature_id_min():
    try:
        if request.method == 'GET':
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute('SELECT id FROM rooms;')
            room_id = cur.fetchall()
            cur.close()
            conn.close()
            return render_template
              ('room_temperature_id_min.html', room_id=room_id)
    except Exception as e:
        current_app.logger.error
          (f"Error at the data base consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
          error_message=str(e)), 500
```

Una vez se tiene la lista de identificadores, se procede a realizar la implementación del HTML de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Select a Room ID to calculate the
      Minimum temperature {% endblock %}</h1>
    <form action="/room_temperature_id_min/result/" method="post">
        <label for="room_id">Select an ID:</label>
        <select name="room_id" id="room_id">
            {% for id in room_id %}
```

```
                <option value="{{ id[0] }}">{{ id[0] }}</option>
            {% endfor %}
        </select>
        <br>
        <input type="submit" value="Select">
    </form>
{% endblock %}
```

Con esto anterior, se realiza un desplegable que permite seleccionar el identificador de la habitación la cual se quiere mostrar la temperatura mínima de esta. Una vez se selecciona el identificador de la habitación, se realiza la petición a la base de datos para obtener la temperatura mínima de la habitación seleccionada por el usuario:

```
@app.route('/room_temperature_id_min/result/', methods=['POST'])
def room_temperature_id_min_result():
    try:
        roomID = request.form['room_id']
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT MIN(temperature)
          FROM temperatures WHERE room_id=%s;', (roomID,))
        temperature = cur.fetchall()
        cur.execute('SELECT name FROM rooms WHERE id=%s;', (roomID,))
        roomname = cur.fetchone()[0]
        cur.close()
        conn.close()
        temperature = temperature[0][0]
        # json format
        data = {
            "id": roomID,
            "name": roomname,
            "min_temperature": temperature
        }
        return render_template('room_temperature_id_min_result.html',
          json_data=json.dumps(data))
    except Exception as e:
        current_app.logger.error
        (f"Error at the data base consultation: {e}")
        return render_template('error.html', error_type=type(e).__name__,
          error_message=str(e)), 500
```

Una vez se tiene la temperatura mínima de la habitación seleccionada por el usuario, se realiza la implementación del HTML de la siguiente manera:

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

{% block content %}
    <h1>{% block title %} Room Minimum temperature {% endblock %}</h1>
    <p>{{ json_data }}</p>
{% endblock %}
```