

### Exercício 1:

Inserir uma reserva e o respetivo pagamento código python:

```
@app.route('/reserva/register', methods=['POST'])
def register_reserva():
    data = request.get_json()
    id_utilizador = data.get("id_utilizador")
    id_quarto = data.get("id_quarto")
    data_inicio = data.get("data_inicio")
    data_fim = data.get("data_fim")
    pagamento = data.get("pagamento")
    # Validação básica
    if not all([id_utilizador, id_quarto, data_inicio, data_fim, pagamento]):
        return jsonify({"erro": "Todos os campos são obrigatórios."}), 400
    try:
        conn = get_connection()
        cur = conn.cursor()
        cur.execute("CALL inserir_reserva(%s, %s, %s, %s, %s)", (id_utilizador,
id_quarto, data_inicio, data_fim, None))
        reserva_id = cur.fetchone()[0]
        cur.execute("CALL inserir_pagamento(%s, %s)", (reserva_id, pagamento))
        conn.commit()
        cur.close()
        conn.close()
        return jsonify({"mensagem": "Reserva registada com sucesso."}), 201
    except psycopg.Error as e:
        conn.rollback()
        return jsonify({"erro": str(e)}), 400
```

Para visualizar o resultado utilize o seguinte link “<https://bdproject-nu.vercel.app/reserva/register>”:

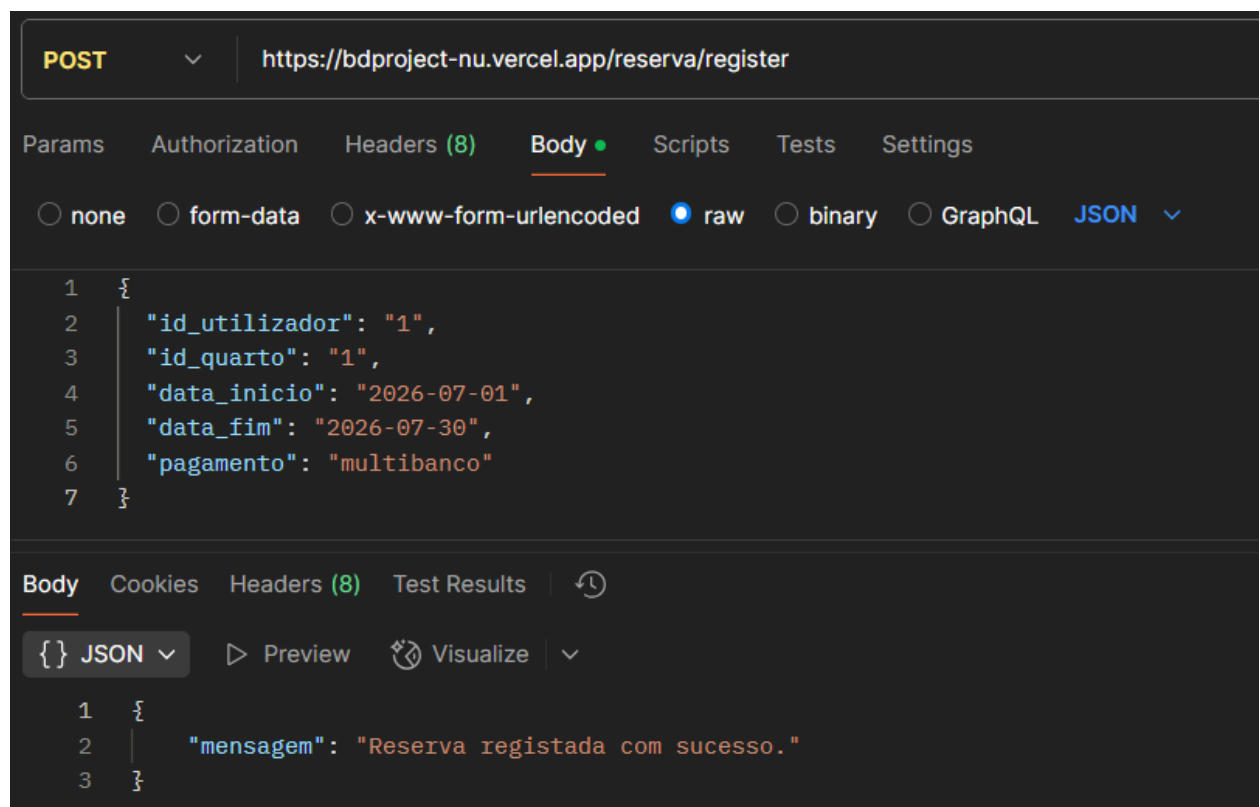


Figura 1 - resultado inserir reserva/pagamentos

## Exercício 2:

Verificar se um quarto está disponível para reserva numa determinada data, código em python:

```
@app.route('/room/<room_num>/<date>', methods=['GET'])
def get_room_availability(room_num, date):
    try:
        conn = get_connection()
        cur = conn.cursor()

        cur.execute("SELECT is_available(%s, %s)", (room_num, date))
        result = cur.fetchone()[0]

        conn.commit()
        cur.close()
        conn.close()

        if result is True:
            return jsonify({"disponibilidade": "Quarto disponível."}), 200
        elif result is False:
            return jsonify({"disponibilidade": "Quarto indisponível."}), 200

    except psycopg.Error as e:
        return jsonify({"erro": str(e)}), 400
```

Resultado para quarto indisponível com o link “bdproject-nu.vercel.app/room/123/2025-07-29”:

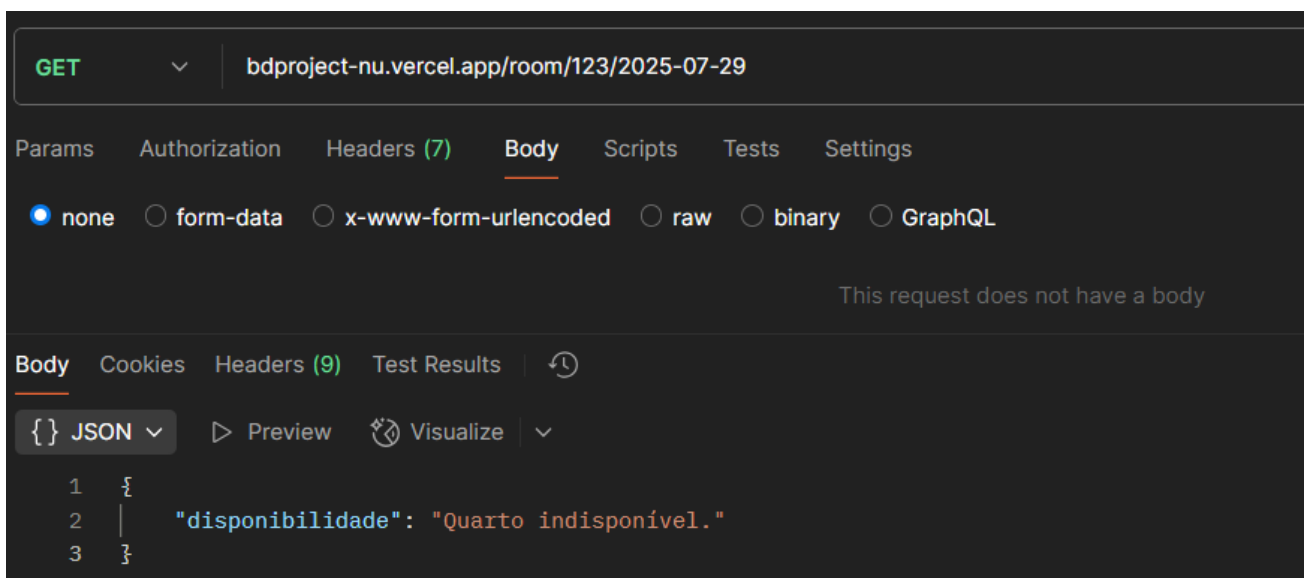


Figura 2 - quarto indisponível

Resultado para quarto disponível, link “bdproject-nu.vercel.app/room/123/2025-07-31”

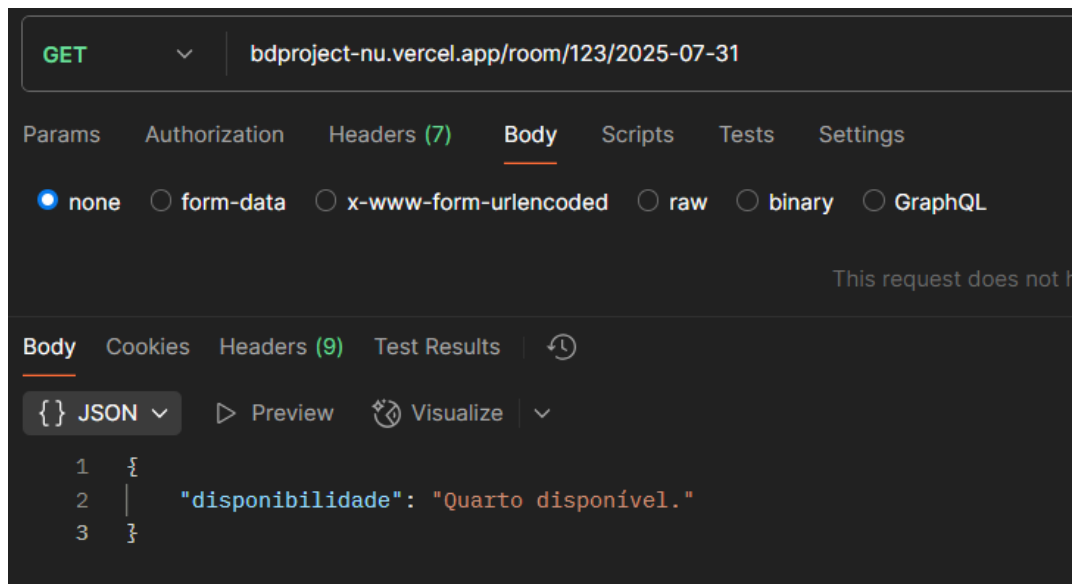


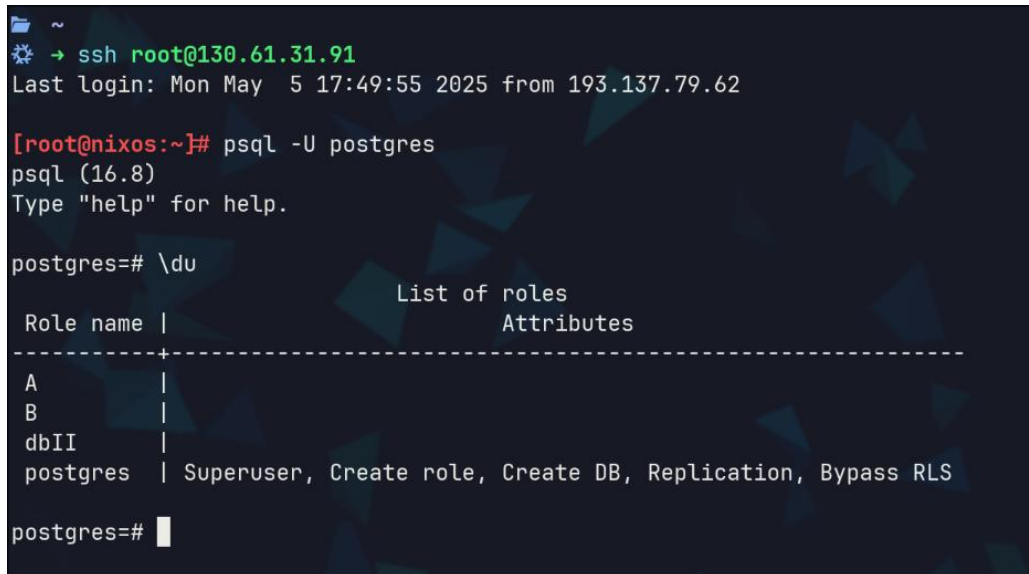
Figura 3 - quarto disponivel

Devido a alguns problemas com a base de dados alocada no servidor da escola decidimos usar uma base de dados que esta alocada numa vps, hospedada na oracle cloud que roda um sistema operacional nixos que possui uma base de dados postgresql. Por esse motivo a criação de utilizadores da base de dados pode ser feita de maneira diferente.

Para criar um utilizador basta no pedaço de código da imagem adicionar “{ name = “A”; } { name = “B”; }” .

```
# Postgres.
services.postgresql = {
  enable = true;
  enableTCPIP = true;
  enableJIT = true;
  ensureUsers = [
    { name = pgConf.userName; }
    { name = "A"; }
    { name = "B"; }
  ];
  ensureDatabases = [ pgConf.dbName ];
  authentication = pkgs.lib.mkOverride 10 ''
    #type database DBuser auth-method
    local all all trust
    host all all 127.0.0.1/32 trust
    host all all ::1/128 trust
    host all all 0.0.0.0/0 scram-sha-256
  '';
};
```

Feito isto podemos verificar que os utilizadores foram criados (figura 4) mas ainda sem permissões.

A terminal window showing a SSH session to root@130.61.31.91. The user runs 'psql -U postgres' and then '\du' to list roles. The output shows roles A, B, dbII, and postgres with their attributes. The postgres role is the superuser.

```
→ ssh root@130.61.31.91
Last login: Mon May 5 17:49:55 2025 from 193.137.79.62

[root@nixos:~]# psql -U postgres
psql (16.8)
Type "help" for help.

postgres=# \du
                                List of roles
Role name | Attributes
-----+-----
A         |
B         |
dbII      |
postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS

postgres=#
```

Figura 4 - confirmação da criação de utilizadores

Para adicionar as permissões e uma password, no mesmo ficheiro do código para criar adicionamos o seguinte:

```
# Systemd post start script.
systemd.services.postgresql.postStart = let
  psql = "${pkgs.postgresql}/bin/psql";
in ''
  ${psql} -U postgres -c "ALTER USER \"A\" WITH PASSWORD 'a';"
  ${psql} -U postgres -c "ALTER USER \"B\" WITH PASSWORD 'b';"

  ${psql} -U postgres -c "GRANT USAGE ON SCHEMA public TO \"A\";"
  ${psql} -U postgres -c "GRANT USAGE ON SCHEMA public TO \"B\";"

  ${psql} -U postgres -c "GRANT SELECT, UPDATE, INSERT, DELETE ON ALL
TABLES IN SCHEMA public TO \"A\";"
  ${psql} -U postgres -c "GRANT SELECT, UPDATE, INSERT, DELETE ON ALL
TABLES IN SCHEMA public TO \"B\";"
'';
```

No final fazemos o rebuild do sistema para aplicar as novas configurações.

Não foi possível implementar a ultima parte de um sistema de auditoria que registre cada vez que os endpoints (a) e (b) eram executados.