

Relevem os erros de português. São 03:39 e foi feito às pressas. 🍷

ETAPA 1 — Criação do Banco de Dados: Sistema de Reserva de Salas de Laboratório

1. Criamos o banco de dados:

```
CREATE DATABASE IF NOT EXISTS reserva_laboratorio;  
USE reserva_laboratorio;
```

2. Criamos as tabelas principais sendo:

Tabela de usuários (alunos, professores, admins)

```
CREATE TABLE usuarios (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    nome VARCHAR(100) NOT NULL,  
    email VARCHAR(100) UNIQUE NOT NULL,  
    senha VARCHAR(255) NOT NULL,  
    tipo ENUM('aluno', 'professor', 'admin') DEFAULT 'aluno'  
);
```

Tabela de salas

```
CREATE TABLE salas (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    nome_sala VARCHAR(50) NOT NULL,  
    capacidade INT NOT NULL,  
    equipamentos TEXT  
);
```

Tabela de reservas

```
CREATE TABLE reservas (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    id_usuario INT NOT NULL,  
    id_sala INT NOT NULL,  
    data_reserva DATE NOT NULL,  
    hora_inicio TIME NOT NULL,  
    hora_fim TIME NOT NULL,
```

```
        status ENUM('pendente', 'aprovada', 'rejeitada') DEFAULT
'pendente',
        FOREIGN KEY (id_usuario) REFERENCES usuarios(id),
        FOREIGN KEY (id_sala) REFERENCES salas(id)
    );
```

3. Inserimos dados de exemplo para fazer testes (Essa parte é mais opcional, vai de cada um, mas é sempre importante verificar se está tudo correto para não ter erros futuros por causas de problemas nas tabelas):

Inserindo usuários

```
INSERT INTO usuarios (nome, email, senha, tipo) VALUES
('João da Silva', 'joao@email.com', '123456', 'aluno'),
('Maria Souza', 'maria@email.com', '123456', 'professor'),
('Admin Sistema', 'admin@email.com', 'admin123', 'admin');
```

Inserindo salas

```
INSERT INTO salas (nome_sala, capacidade, equipamentos) VALUES
('Lab 01', 25, 'Projektor, Computadores'),
('Lab 02', 20, 'Quadro Interativo, Computadores');
```

Inserindo reservas

```
INSERT INTO reservas (id_usuario, id_sala, data_reserva,
hora_inicio, hora_fim, status) VALUES
(1, 1, '2025-06-15', '08:00:00', '10:00:00', 'pendente'),
(2, 2, '2025-06-16', '14:00:00', '16:00:00', 'aprovada');
```

4. Criamos um usuário com acesso limitado assim como foi pedido pelo professor

Criar o usuário

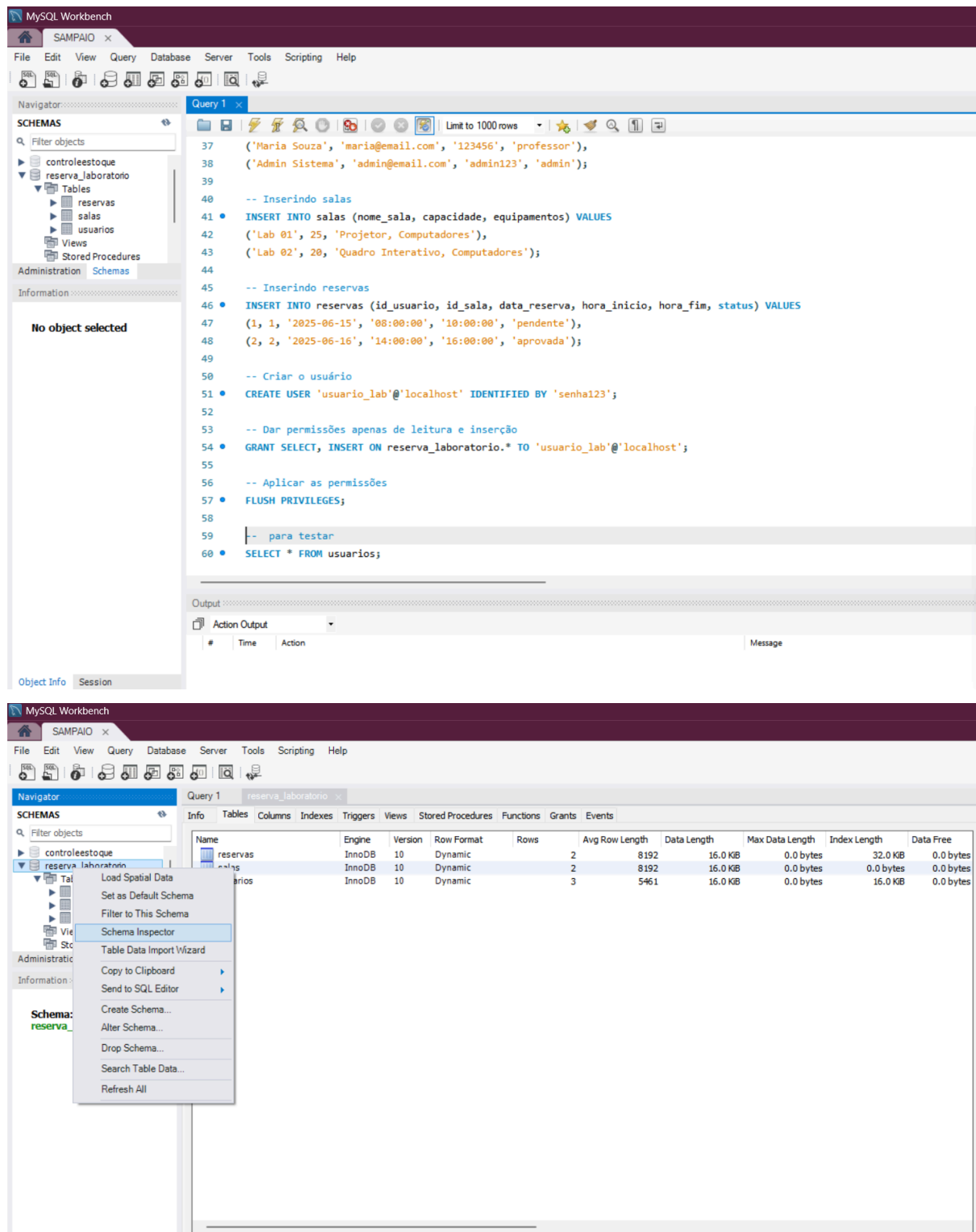
```
CREATE USER 'usuario_lab'@'localhost' IDENTIFIED BY 'senha123';
```

Dar permissões apenas de leitura e inserção

```
GRANT SELECT, INSERT ON reserva_laboratorio.* TO
'usuario_lab'@'localhost';
```

Aplicar as permissões

```
FLUSH PRIVILEGES;
```

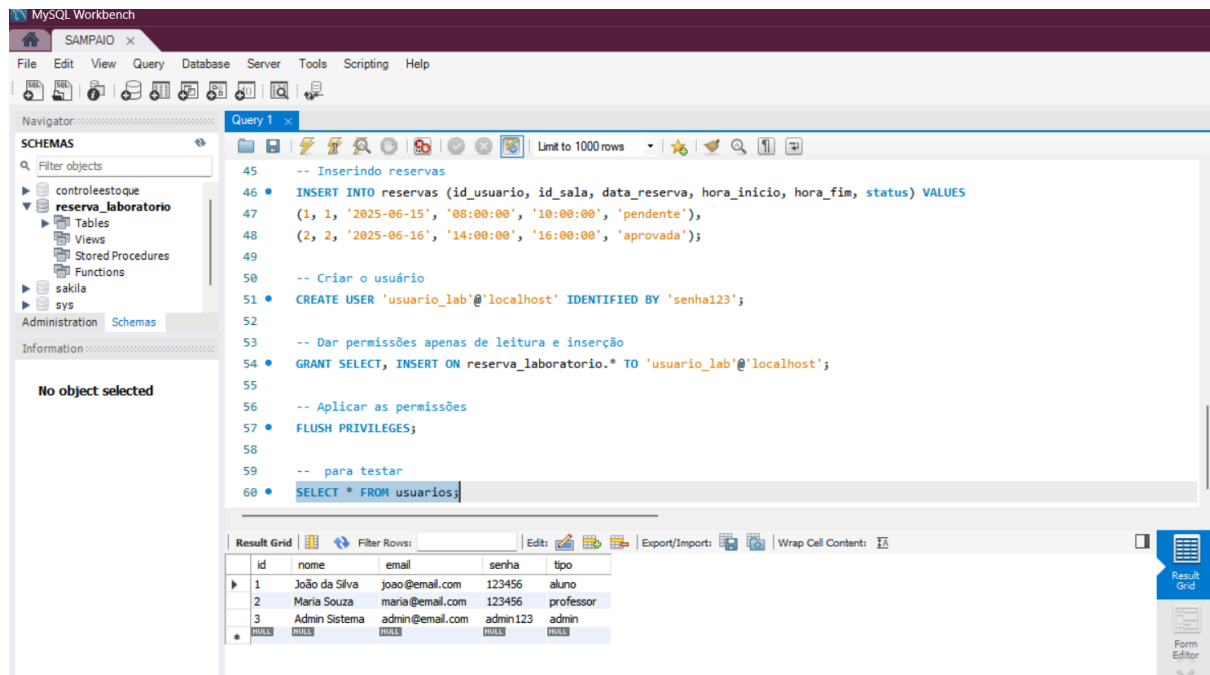


PRONTO, Agora o nosso banco já está criado, com tabelas, dados e controle de acesso

- Consultar as tabelas clicando em **Schemas > reserva_laboratorio > Tables** ou apenas olhando na lateral logo abaixo do no reserva_laboratorio

para testar se tudo está correndo bem e visualizar usamos o :

```
SELECT * FROM usuarios;
```



- Rodar `SELECT * FROM usuarios;` para testar e assim por diante (poderia ser salas, reservas e o que mais a gente quisesse colocar nas tabelas)

ETAPA 2 — Foi o Backup Automatizado do MySQL com Python e Upload para Google Drive

O Objetivo foi:

Fazer um script em **Python** que:

1. Faz backup do banco de dados.
2. Salva o `.sql` do computador.
3. Envia automaticamente o arquivo para o diretório escolhido no **Google Drive** via **API**.

O Primeiro Passo Foi Verificar se o Python está Instalado Junto com algumas bibliotecas necessarias (se ainda não tivesse teriamos que baixar)

bibliotecas necessárias

No terminal (cmd) do vs code instalamos o:

```
1° mysql-connector-python
2° google-api-python-client google-auth google-auth-oauthlib
google-auth-httpplib2
```

Passo 2 — foi Criar backup local com Python

então Criamos o arquivo chamado `backup.py` com o código abaixo:

```
import os
import datetime
```

Configurações do MySQL

```
usuario = "root"
senha = "sua_senha_do_mysql"
banco = "reserva_laboratorio"
```

Lembrando de **Substituir** `sua_senha_do_mysql` e o `sua_Usuario_do_mysql` (que por coincidência é `root` tbm) pela senha e usuario que você usa no Workbench.

Nome do arquivo de backup com data e hora

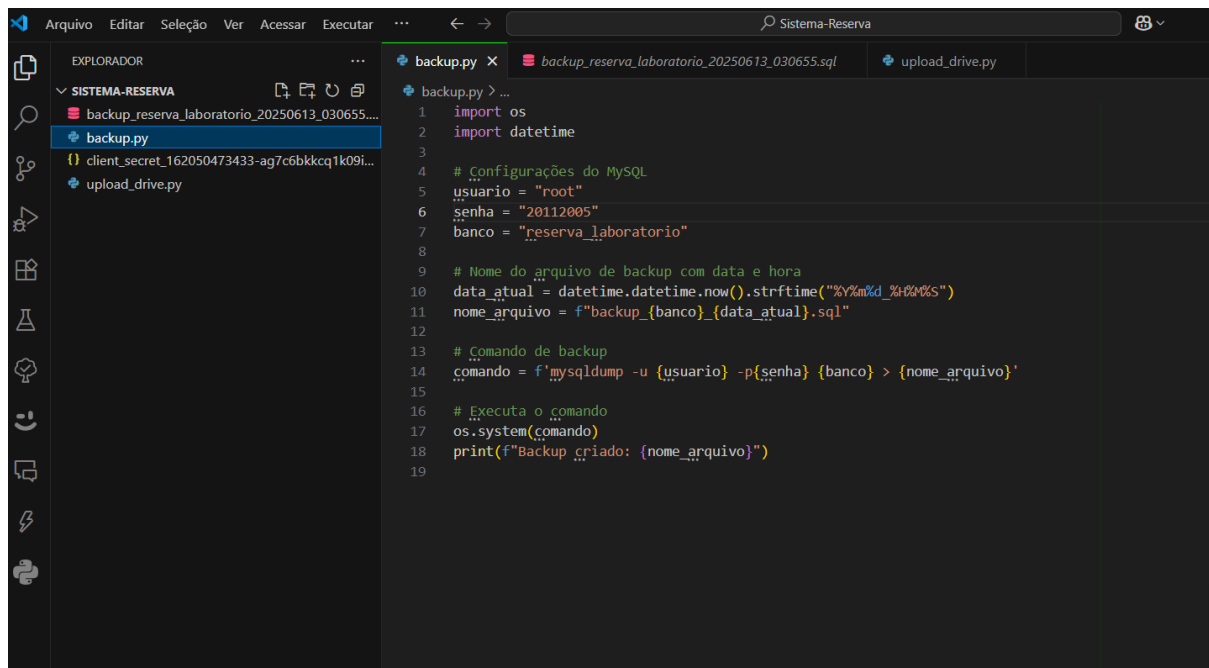
```
data_atual = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
nome_arquivo = f"backup_{banco}_{data_atual}.sql"
```

Comando de backup

```
comando = f'mysqldump -u {usuario} -p{senha} {banco} > {nome_arquivo}'
```

Executa o comando

```
os.system(comando)
print(f"Backup criado: {nome_arquivo}")
```



The image shows a code editor interface with a dark theme. On the left, there is a file explorer pane titled 'EXPLORADOR' showing a project named 'SISTEMA-RESERVA'. Inside this project, there are three files: 'backup_reserva_laboratorio_20250613_030655...', 'backup.py', and 'upload_drive.py'. The 'backup.py' file is selected and its contents are displayed in the main editor area. The code is a Python script that performs a MySQL backup. It imports 'os' and 'datetime' modules. It defines MySQL configuration variables: 'usuario' (root), 'senha' (20112005), and 'banco' (reserva_laboratorio). It calculates the current date and time to create a unique filename for the backup. It constructs a MySQL command using 'mysqldump' and executes it using 'os.system()'. Finally, it prints a message indicating the backup was created.

```
1 import os
2 import datetime
3
4 # Configurações do MySQL
5 usuario = "root"
6 senha = "20112005"
7 banco = "reserva_laboratorio"
8
9 # Nome do arquivo de backup com data e hora
10 data_atual = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
11 nome_arquivo = f"backup_{banco}_{data_atual}.sql"
12
13 # Comando de backup
14 comando = f'mysqldump -u {usuario} -p{senha} {banco} > {nome_arquivo}'
15
16 # Executa o comando
17 os.system(comando)
18 print(f"Backup criado: {nome_arquivo}")
19
```

Passo 3 — Foi criar um projeto no Google Cloud (uma vez só)

1. Fomos em: console.cloud.google.com
2. Depois em **APIs e serviços > Tela de consentimento OAuth > colocamos o nome de Reservas-de-Lab, colocamos o e-mail de suporte e tudo mais** → marcamos a opção “Externo” pois no Externo fica disponível para qualquer usuário de teste que tenha uma Conta do Google. O aplicativo ficará no modo de testes e só será disponibilizado aos usuários que você adicionou à lista de usuários de teste. Quando o aplicativo estiver pronto para ser enviado à produção, talvez seja necessário. verificá-lo.
3. depois em **Credenciais > Criamos uma credencial > com o ID do cliente OAuth** com o Tipo: app para computador
4. Baixamos o arquivo **credentials.json**

[←](#) Criar ID do cliente do OAuth

Um ID do cliente é usado para identificar um único app nos servidores OAuth do Google. Se o app for executado em várias plataformas, cada uma precisará de um ID do cliente. Para acessar mais informações, consulte [Como configurar o OAuth 2.0](#). [Saiba mais](#) sobre tipos de clientes OAuth.

Tipo de aplicativo *
App para computador

Nome *
Reservas-Lab

O nome do cliente OAuth 2.0. Esse nome é usado apenas para identificar o cliente no console e não será mostrado aos usuários finais.

Observação: pode levar de cinco minutos a algumas horas para que as configurações entrem em vigor

Criar

Cancelar

The screenshot shows the Google Cloud console interface for creating an OAuth 2.0 client. The browser address bar shows 'Google Cloud' and 'Reservas-Lab'. The page title is 'Google Auth Platform / Visão geral / Criar branding'. The left sidebar contains a menu with 'Visão geral' (selected), 'Branding', 'Público-alvo', 'Clientes', 'Acesso a dados', and 'Central de verificação'. The main content area is titled 'Configuração do projeto' and contains a progress indicator with four steps: 1. Informações do app, 2. Público, 3. Dados de contato, and 4. Concluir. Step 1 is currently active. It includes a form with 'Nome do app' (filled with 'Reservas-Lab') and 'E-mail para suporte do usuário' (filled with 'samuelcampio476@gmail.com'). Below the form is a 'Próxima' button. At the bottom of the main content area are 'Criar' and 'Cancelar' buttons.

Google Cloud Reservas-Lab Pesquise (/) recursos, documentos, produtos e muito mais

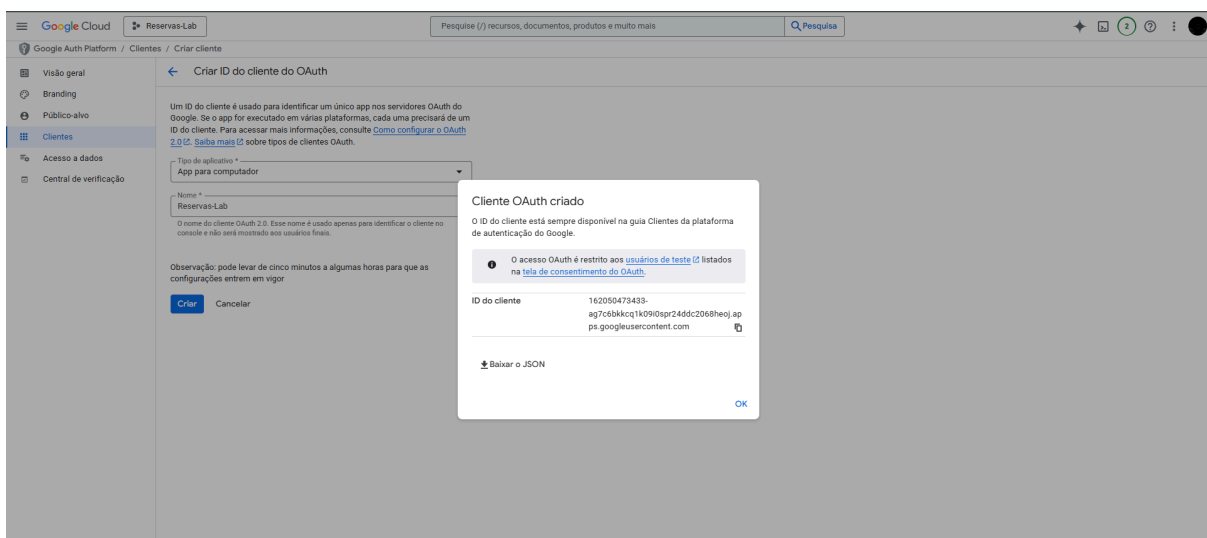
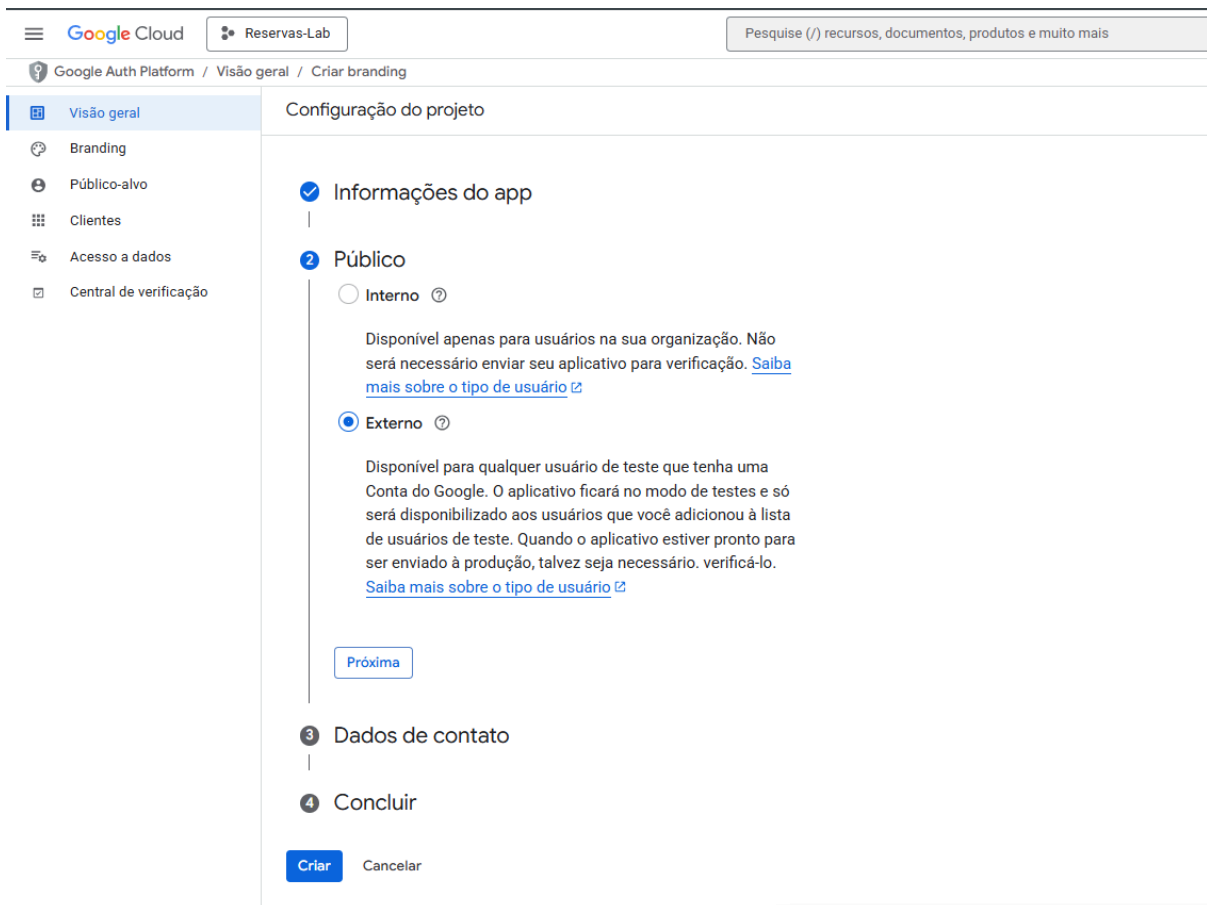
Google Auth Platform / Visão geral / Criar branding

Visão geral Branding Público-alvo Clientes Acesso a dados Central de verificação

Configuração do projeto

- 1 Informações do app
Nome do app *
Reservas-Lab
O nome do aplicativo que precisa da permissão
E-mail para suporte do usuário *
samuelcampio476@gmail.com
Para que os usuários entrem em contato com você a respeito do consentimento.
[Saiba mais](#)
Próxima
- 2 Público
- 3 Dados de contato
- 4 Concluir

Criar Cancelar



✓ Passo 4 — Criamos códigos Python para fazer o upload para o Google Drive

chamado `upload_drive.py`:

```
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload
```



```
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
import pickle
import os
```

Escopos

```
SCOPES = ['https://www.googleapis.com/auth/drive.file']
```

Autenticação

```
def autenticar():
    creds = None
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)

    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow =
InstalledAppFlow.from_client_secrets_file('credentials.json',
SCOPES)
            creds = flow.run_local_server(port=0)

        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)

    return build('drive', 'v3', credentials=creds)
```

Upload para o Drive

```
def upload_arquivo(nome_arquivo):
    service = autenticar()
    arquivo = MediaFileUpload(nome_arquivo,
mimetype='application/sql')
    response = service.files().create(
        body={'name': nome_arquivo},
        media_body=arquivo
    ).execute()
    print(f'Arquivo enviado para o Drive: {response["id"]}')

```

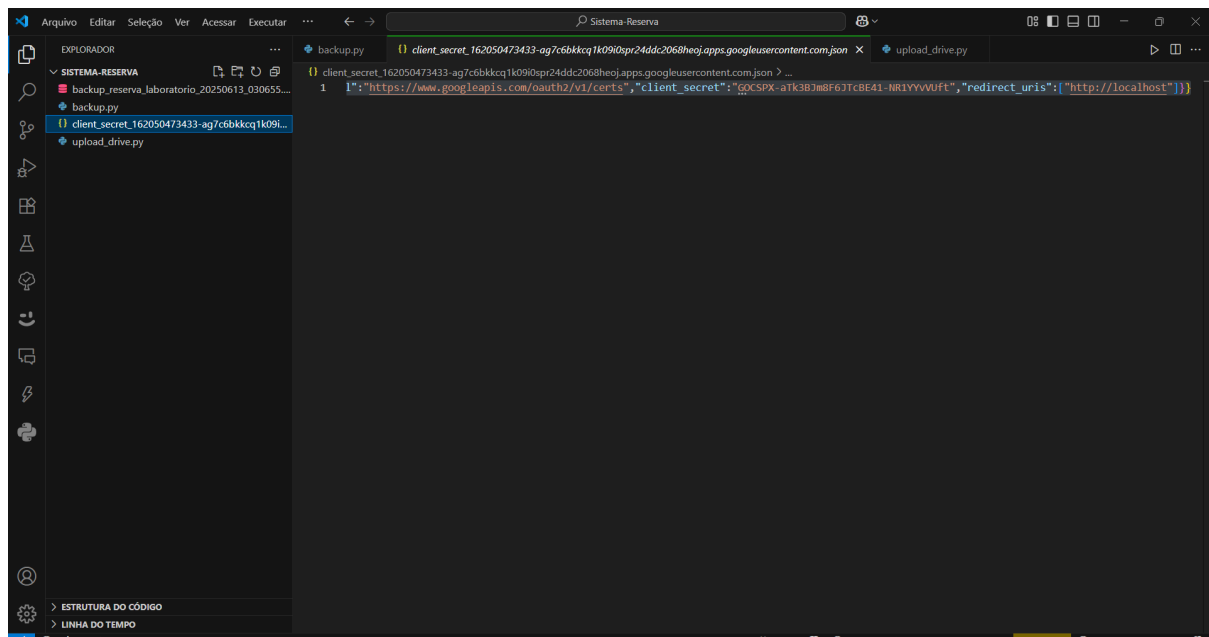
```
20 --
21
22 DROP TABLE IF EXISTS `reservas`;
23 /*!40101 SET @saved_cs_client = @@character_set_client */;
24 /*!50503 SET character_set_client = utf8mb4 */;
25 CREATE TABLE `reservas` (
26   `id` int NOT NULL AUTO INCREMENT,
27   `id_usuario` int NOT NULL,
28   `id_sala` int NOT NULL,
29   `data_reserva` date NOT NULL,
30   `hora_inicio` time NOT NULL,
31   `hora_fim` time NOT NULL,
32   `status` enum('pendente','aprovada','rejeitada') DEFAULT 'pendente',
33   PRIMARY KEY (`id`),
34   KEY `id_usuario` (`id_usuario`),
35   KEY `id_sala` (`id_sala`),
36   CONSTRAINT `reservas_ibfk_1` FOREIGN KEY (`id_usuario`) REFERENCES `usuarios` (`id`),
37   CONSTRAINT `reservas_ibfk_2` FOREIGN KEY (`id_sala`) REFERENCES `salas` (`id`),
38   ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
39 /*!40101 SET character_set_client = @saved_cs_client */;
40
41 --
42 -- Dumping data for table `reservas`
43 --
44
45 LOCK TABLES `reservas` WRITE;
46 /*!40000 ALTER TABLE `reservas` DISABLE KEYS */;
47 INSERT INTO `reservas` VALUES (1,1,1,'2025-06-15','08:00:00','10:00:00','pendente'),(2,2,2,'2025-06-16','14:00:00','16:00:00','aprovada');
48 /*!40000 ALTER TABLE `reservas` ENABLE KEYS */;
49 UNLOCK TABLES;
50
51 --
52 -- Table structure for table `salas`
53 --
54
55 DROP TABLE IF EXISTS `salas`;
56 /*!40101 SET @saved_cs_client = @@character_set_client */;
```

```
1 from googleapiclient.discovery import build
2 from googleapiclient.http import MediaFileUpload
3 from google_auth_oauthlib.flow import InstalledAppFlow
4 from google.auth.transport.requests import Request
5 import pickle
6 import os
7
8 # Escopos
9 SCOPES = ['https://www.googleapis.com/auth/drive.file']
10
11 # Autenticação
12 def autenticar():
13     creds = None
14     if os.path.exists('token.pickle'):
15         with open('token.pickle', 'rb') as token:
16             creds = pickle.load(token)
17
18     if not creds or not creds.valid:
19         if creds and creds.expired and creds.refresh_token:
20             creds.refresh(Request())
21         else:
22             flow = InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)
23             creds = flow.run_local_server(port=0)
24
25         with open('token.pickle', 'wb') as token:
26             pickle.dump(creds, token)
27
28     return build('drive', 'v3', credentials=creds)
29
30 # Upload para o Drive
31 def upload_arquivo(nome_arquivo):
32     service = autenticar()
33     arquivo = MediaFileUpload(nome_arquivo, mimetype='application/sql')
34     response = service.files().create(
35         body={'name': nome_arquivo},
36         media_body=arquivo
37     ).execute()
```

Exemplo de uso

`upload_arquivo('backup_reserva_laboratorio_20250613_203000.sql')`

Mude o nome o backup conforme foi gerado no backup.py



Passo 5 —foi testar Executando tudo

1. Rode primeiro o backup:

[backup.py](#) Depois o upload: [upload_drive.py](#)

Se tudo tiver certo o backup será criado e enviado automaticamente para o **Google Drive!**

(PAREI NESSA PARTE, TENHO QUE ARRUMAR A CREDENCIAL PARA DAR PERMISSÃO)