

Relatório - CheckIn

1) Rotas existentes — input / output (JSON modelo)

- **POST /usuarios/register**
 - **Input**
{ "nome": "string", "email": "string", "senha": "string" }
 - **Output**
{ "message": "Usuário criado com sucesso!", "usuario": { "id": 1, "nome": "string", "email": "string" } }
- **POST /usuarios/login**
 - **Input**
{ "email": "string", "senha": "string" }
 - **Output**
{ "access_token": "jwt", "user": { "id": X, "nome": "string", "email": "string" } }
- **GET /hospede/hoespedes**
 - **Input:** query status
 - **Output**
[{ "id": 1, "nome": "", "cpf": "", "telefone": "", "email": "", "endereço": "" }]
- **POST /hospede/**
 - **Input**
{ "nome": "", "cpf": "", "telefone": "", "email": "", "endereço": "" }
 - **Output:** objeto hóspede criado.
- **PUT /hospede/{hospede_id}** — atualiza hóspede.
 - **Input**
{ "nome": "string", "email": "string", "senha": "string" }
 - **Output**
{ "message": "Usuário atualizado com sucesso!", "usuario": { "id": X, "nome": "string", "email": "string" } }
 -
- **DELETE /hospede/{hospede_id}**
 - **Output:** { "message": "Hóspede removido com sucesso!" }
- **GET /hospede/{hospede_id}** — retorna hóspede.

- **GET /quarto/quartos**

- **Input:** query status
- **Output**

```
[ { "id":1, "numero":101, "capacidade":2, "tipo":"Standard", "preco_diaria":120.5, "status":"disponível", "recursos":"string" } ]
```

- **POST /quartos/**

- **Input**
{ "número":101, "capacidade":2, "tipo":"Standard", "preco_diaria":120.5, "status":"disponível", "recursos":"string" }
- **Output:** Quarto Cadastrado com sucesso.

- **PUT /quartos/{quarto_id}**

- **Input**
{ "número":101, "capacidade":2, "tipo":"Standard", "preco_diaria":120.5, "status":"disponível", "recursos":"string" }
- **Output:** Quarto atualizado com sucesso.

- **DELETE /quartos/{quarto_id}**

- **Output:** { "message": "Quarto removido com sucesso!" }

- **GET /reserva/reservas**

- **Input:** ()
- **Output**
[{ "id":1, "hospede_id":1, "quarto_id":1, "data_entrada":"ISO8601", "data_saida":"ISO8601", "status":"", "observacoes":"" }]

2) Dicionários de dados

Usuario

```
{ "id": "int", "nome": "string", "email": "string", "senha": "string" }
```

Hospede

```
{ "id": "int", "nome": "string", "cpf": "string", "telefone": "string", "email": "string", "endereco": "string" }
```

Quarto

```
{ "id": "int", "numero": "int", "capacidade": "int", "tipo": "string", "preco_diaria": "float", "status":  
"string", "recursos": "string"}
```

Reserva

```
{ "id": "int", "hospede_id": "int", "quarto_id": "int", "data_entrada": "datetime", "data_saida":  
"datetime", "status": "string", "observacoes": "string"}
```

CheckIn

```
{ "id": "int", "reserva_id": "int", "data_hora": "datetime"}
```

CheckOut

```
{ "id": "int", "reserva_id": "int", "data_hora": "datetime", "valor_total": "float",  
"forma_pagamento": "string"}
```

3) Rotas que faltam implementar / corrigir

Faltam implementar (ou estão com bugs):

1. Corrigir decorator e endpoints em reserva.py (ex.: GET /reserva/, POST /reserva/, GET /reserva/{id}, PUT, DELETE).
2. Endpoints de **check-in** e **check-out**: POST /checkin, POST /checkout (validar reserva, atualizar status do quarto, registrar valor).
3. Endpoints de **relatórios** (ocupação, receita por período) — GET /relatorios/ocupacao, GET /relatorios/receita.
4. GET /usuarios/ (retornar usuário atual a partir do token).
5. Padronizar inputs (usar JSON), padronizar retornos (não retornar hash da senha).

4) Prazo estimado

- Correções críticas (typos, auth e não expor a senha): 15 dias
- Implementar CRUD de Reservas + Check-in/Check-out + testes: 15 dias

Tempo previsto: 15 dias

5) Fluxo de Login / Logout

- **Registro:** POST /usuarios/register → senha é hashada (bcrypt) e usuário salvo.
- **Login:** POST /usuarios/login → valida credenciais, gera JWT (access_token) com sub=email e expiração (30 min). Resposta contém o token. Frontend envia Authorization: Bearer <token> nas requisições.
- **Proteção:** rotas usam Depends(oauth2_scheme) e get_current_user que decodifica JWT e retorna usuário; rotas com user = Depends(get_current_user) exigem token válido.
- **Logout: não implementado no servidor** (JWT sem blacklist). Logout atual é cliente removendo token localmente; para logout server-side é necessário implementar blacklist/armazenamento dos tokens inválidos ou usar refresh tokens + revogação.

6) Equipe

- Davi Guedes Noberto;
- Estela Áurea da Nóbrega Calixto;
- João Marcos de Azevedo Dantas;
- Raun Allyson de Araújo Felix.

7) Link GitHub

- https://github.com/joao-marcos-azvd/Gestao_Hoteis