**Fatec Ogari de Castro Pacheco**

Luan Garcia, Igor Teodoro

**Reserva de Mesas**

“Mesas e comandas”

Itapira/SP

27/11/2024

Sumário

[Resumo Executivo: 3](#_Toc183556472)

[Descrição da Aplicação 3](#_Toc183556473)

[Serviços AWS Utilizados 3](#_Toc183556474)

[Proposta 3](#_Toc183556475)

[Introdução 4](#_Toc183556476)

[Descrição Técnica 5](#_Toc183556477)

[Serviços AWS: 5](#_Toc183556478)

[API Gateway 5](#_Toc183556479)

[Lambda 7](#_Toc183556480)

[DynamoDB 8](#_Toc183556481)

[AWS Amplify: 9](#_Toc183556482)

[Desenvolvimento 12](#_Toc183556483)

[Desafio encontrados: 17](#_Toc183556484)

[Teste da aplicação 18](#_Toc183556485)

[Conclusão e trabalhos futuros: 19](#_Toc183556486)

[Referências 20](#_Toc183556487)

# Resumo Executivo:

## Descrição da Aplicação

É uma aplicação para gerência reserva de mesas e os produtos em comandas.

## Serviços AWS Utilizados

Lambda, DynamoDB, AWS Amplify, API Gateway.

## Proposta

É promover uma aplicação flexível para gerência as comandas com os itens comprados e as reservas de mesa.

# Introdução

O presente trabalho apresenta o desenvolvimento de uma aplicação inovadora voltada para a gestão de reservas de mesas e controle de comandas em restaurantes, utilizando recursos avançados da plataforma Amazon Web Services (AWS). O projeto tem como principal objetivo proporcionar uma experiência otimizada para restaurantes e clientes, permitindo maior agilidade no gerenciamento de mesas, controle de pedidos e redução de erros operacionais.

Para alcançar esses objetivos, foram utilizados quatro serviços principais da AWS: o Amazon DynamoDB, para armazenamento e gerenciamento eficiente de dados em um banco de dados NoSQL; o AWS Lambda, para execução de funções serverless que garantem alta escalabilidade e baixo custo operacional; o Amazon API Gateway, para gerenciamento de APIs que conectam o frontend ao backend de forma segura e eficiente; e o AWS Amplify, que fornece integração contínua, hosting e suporte para o desenvolvimento frontend.

A escolha dos serviços AWS justifica-se pela necessidade de uma infraestrutura que combine desempenho, escalabilidade e segurança. A aplicação foi desenvolvida considerando-se as melhores práticas de arquitetura em nuvem, com ênfase na modularidade e integração entre os serviços. Com isso, busca-se atender às demandas específicas do setor de restaurantes, garantindo um sistema confiável, flexível e preparado para futuros aprimoramentos.

# Descrição Técnica

## Serviços AWS:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

## API Gateway

Foi criado a API com o nome API Gerenciar Reservas.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Foi criado os recursos “adicionarItem”, “consultarComanda”, “ConsultarReserva”, “finalizarComanda”, “liberarMesa”, “reservarMesa”, e todos com method POST, com os CORS ativado e no estágio PROD, para a requisição do Front-end e todos as integração com as função criadas no lambda.  
  
  
Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

## [Lambda](https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1)

Foi utilizado lambda para execução do back-end com as funções que recebe as requisições da API e esse mesmo serviço faz as interações com o banco de dados DynamoDB como create, insert, delete, update.

Foi criando a função lambda GerenciarReservasRestaurante:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Nesse próximo print é possível ver a ligação da função lambda ligada ao gatilho do API Gateway:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

O código da função no lambda está disponível no GITHUB.

## DynamoDB

Foi utiliza o serviço dynamoDB para o banco de dados não relacional que interage com o lambda.

Print das tabelas:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Print dos atributos:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente  
Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

## AWS Amplify:

Foi utilizado o serviço aws amplify para roda o front-end, e para fazer a ligação com o github para as implementações por commit, onde a cada commit realizado, ele fazia uma implantação automática a cada commit, e o projeto todo está rodando em react.

Foi criado uma aplicação chamada AWSreservarMesa:  
Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente  
  
Interface gráfica do usuário, Texto, Aplicativo, Teams

Descrição gerada automaticamente

Na próxima tela é a implementação e o histórico de implementação.  
Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Como podemos ver ali possui o link do domínio da aplicação <https://main.dp4fqk71bihlo.amplifyapp.com/>, como o total de implementação, também o histórico de implementação.

# Desenvolvimento

O front-end foi desenvolvido em react o back-end desenvolvido em python no lambda, todos os códigos estão disponíveis no github:

https://github.com/IgorTeodoroDEV/AWSreservaMesa

Código lambda:

import json

import boto3

from botocore.exceptions import ClientError

# Inicializando o recurso DynamoDB

dynamodb = boto3.resource('dynamodb')

reservas\_table = dynamodb.Table('Reservadas') # Referência à tabela de reservas de mesas

comandas\_table = dynamodb.Table('Comandas') # Referência à tabela de comandas de pedidos

def lambda\_handler(event, context):

"""

Função principal Lambda que é chamada quando o evento é acionado.

Processa a entrada e direciona para a função correspondente baseada na ação especificada.

"""

action = event.get('action') # Obtém o tipo de ação a ser realizada

mesa\_id = event.get('mesaID') # ID da mesa (se aplicável)

comanda\_id = event.get('comandaID') # ID da comanda (se aplicável)

# Verifica se a ação foi fornecida

if not action:

return {

'statusCode': 400,

'body': json.dumps("acao nao fornecida.") # Retorna erro se a ação não for especificada

}

# Redireciona para a função correspondente com base na ação

if action == 'reservarMesa':

return reservar\_mesa(mesa\_id) # Chama a função para reservar uma mesa

elif action == 'liberarMesa':

return liberar\_mesa(mesa\_id) # Chama a função para liberar uma mesa

elif action == 'adicionarItem':

produto = event.get('produto') # Produto a ser adicionado à comanda

quantidade = event.get('quantidade') # Quantidade do produto

# Verifica se produto e quantidade foram fornecidos

if not produto or not quantidade:

return {

'statusCode': 400,

'body': json.dumps("Produto ou quantidade não fornecidos.") # Erro por falta de informações

}

return adicionar\_item\_comanda(comanda\_id, produto, quantidade) # Adiciona item à comanda

elif action == 'finalizarComanda':

return finalizar\_comanda(comanda\_id) # Finaliza a comanda

elif action == 'consultarReserva':

return consultar\_reserva() # Consulta todas as reservas

elif action == 'consultarComanda':

return consultar\_comanda(comanda\_id) # Consulta uma comanda específica

elif action == 'consultarItensComanda':

return consultar\_itens\_comanda(comanda\_id) # Consulta itens de uma comanda específica

else:

# Retorna erro se a ação não for reconhecida

return {

'statusCode': 400,

'body': json.dumps('acao invalida fornecida.')

}

def reservar\_mesa(mesa\_id):

"""

Reserva uma mesa com base no ID fornecido.

"""

if not mesa\_id:

return {'statusCode': 400, 'body': json.dumps('Mesa ID não fornecido.')} # Erro se o ID da mesa não for informado

print(f"Tentando reservar a mesa {mesa\_id}")

try:

# Insere um item na tabela de reservas, garantindo que a mesa ainda não esteja reservada

reservas\_table.put\_item(

Item={

'mesaID': str(mesa\_id), # ID da mesa

'reservada': True # Status de reserva

},

ConditionExpression="attribute\_not\_exists(mesaID)" # Condição: só insere se não existir o ID

)

return {'statusCode': 200, 'body': json.dumps(f"Mesa {mesa\_id} reservada com sucesso.")}

except ClientError as e:

# Tratamento de erros relacionados ao DynamoDB

error\_message = e.response['Error']['Message']

error\_code = e.response['Error']['Code']

print(f"Erro ao tentar reservar a mesa {mesa\_id}: {error\_message} (Code: {error\_code})")

if error\_code == 'ConditionalCheckFailedException':

# Erro específico: a mesa já está reservada

return {'statusCode': 400, 'body': json.dumps(f"Mesa {mesa\_id} já está reservada.")}

return {'statusCode': 500, 'body': json.dumps("Erro ao reservar mesa")}

def liberar\_mesa(mesa\_id):

"""

Libera uma mesa, alterando seu status de reservada para disponível.

"""

if not mesa\_id:

return {'statusCode': 400, 'body': json.dumps('Mesa ID não fornecido.')} # Erro se o ID não for informado

try:

# Atualiza o status da mesa na tabela de reservas

reservas\_table.update\_item(

Key={'mesaID': mesa\_id}, # Localiza a mesa pelo ID

UpdateExpression="set reservada = :r", # Atualiza o campo 'reservada'

ConditionExpression="reservada = :true", # Só atualiza se a mesa estiver reservada

ExpressionAttributeValues={':r': False, ':true': True} # Valores para a atualização

)

return {'statusCode': 200, 'body': json.dumps(f"Mesa {mesa\_id} liberada.")}

except ClientError as e:

if e.response['Error']['Code'] == 'ConditionalCheckFailedException':

# Erro específico: a mesa já não estava reservada

return {'statusCode': 400, 'body': json.dumps(f"Mesa {mesa\_id} não está reservada.")}

return {'statusCode': 500, 'body': json.dumps("Erro ao liberar mesa.")}

def adicionar\_item\_comanda(comanda\_id, produto, quantidade):

"""

Adiciona um item (produto e quantidade) a uma comanda específica.

"""

if not comanda\_id or not produto or not quantidade:

return {'statusCode': 400, 'body': json.dumps('Comanda ID, produto ou quantidade não fornecidos.')} # Validação de entrada

try:

# Adiciona o item à lista de itens da comanda no banco de dados

comandas\_table.update\_item(

Key={'comandaID': str(comanda\_id)}, # Localiza a comanda pelo ID

UpdateExpression="SET itens = list\_append(if\_not\_exists(itens, :empty\_list), :new\_item)", # Adiciona item à lista

ExpressionAttributeValues={

':new\_item': [{'produto': produto, 'quantidade': quantidade}], # Novo item a ser adicionado

':empty\_list': [] # Cria uma lista vazia se não houver itens

}

)

return {'statusCode': 200, 'body': json.dumps(f"Adicionado {quantidade}x {produto} na comanda {comanda\_id}.")}

except ClientError:

return {'statusCode': 500, 'body': json.dumps("Erro ao adicionar item na comanda.")}

def finalizar\_comanda(comanda\_id):

"""

Finaliza uma comanda, calculando o total e removendo-a do banco de dados.

"""

if not comanda\_id:

return {'statusCode': 400, 'body': json.dumps("Comanda ID não fornecido.")}

try:

# Obtém os detalhes da comanda

response = comandas\_table.get\_item(Key={'comandaID': str(comanda\_id)})

if 'Item' not in response:

return {'statusCode': 400, 'body': json.dumps("Comanda não encontrada.")}

itens = response['Item'].get('itens', [])

if not itens:

return {'statusCode': 400, 'body': json.dumps("Comanda não contém itens.")}

# Calcula o total

total = sum(

float(item['quantidade']) \* float(get\_preco\_produto(item['produto']))

for item in itens

)

comandas\_table.delete\_item(Key={'comandaID': comanda\_id}) # Remove a comanda do banco

return {'statusCode': 200, 'body': json.dumps(f"Comanda {comanda\_id} finalizada. Total: {total}")}

except ClientError:

return {'statusCode': 500, 'body': json.dumps("Erro ao finalizar comanda.")}

except ValueError as e:

return {'statusCode': 400, 'body': json.dumps(f"Erro de valor inválido: {str(e)}")}

def get\_preco\_produto(produto):

"""

Retorna o preço de um produto com base em uma tabela fixa.

"""

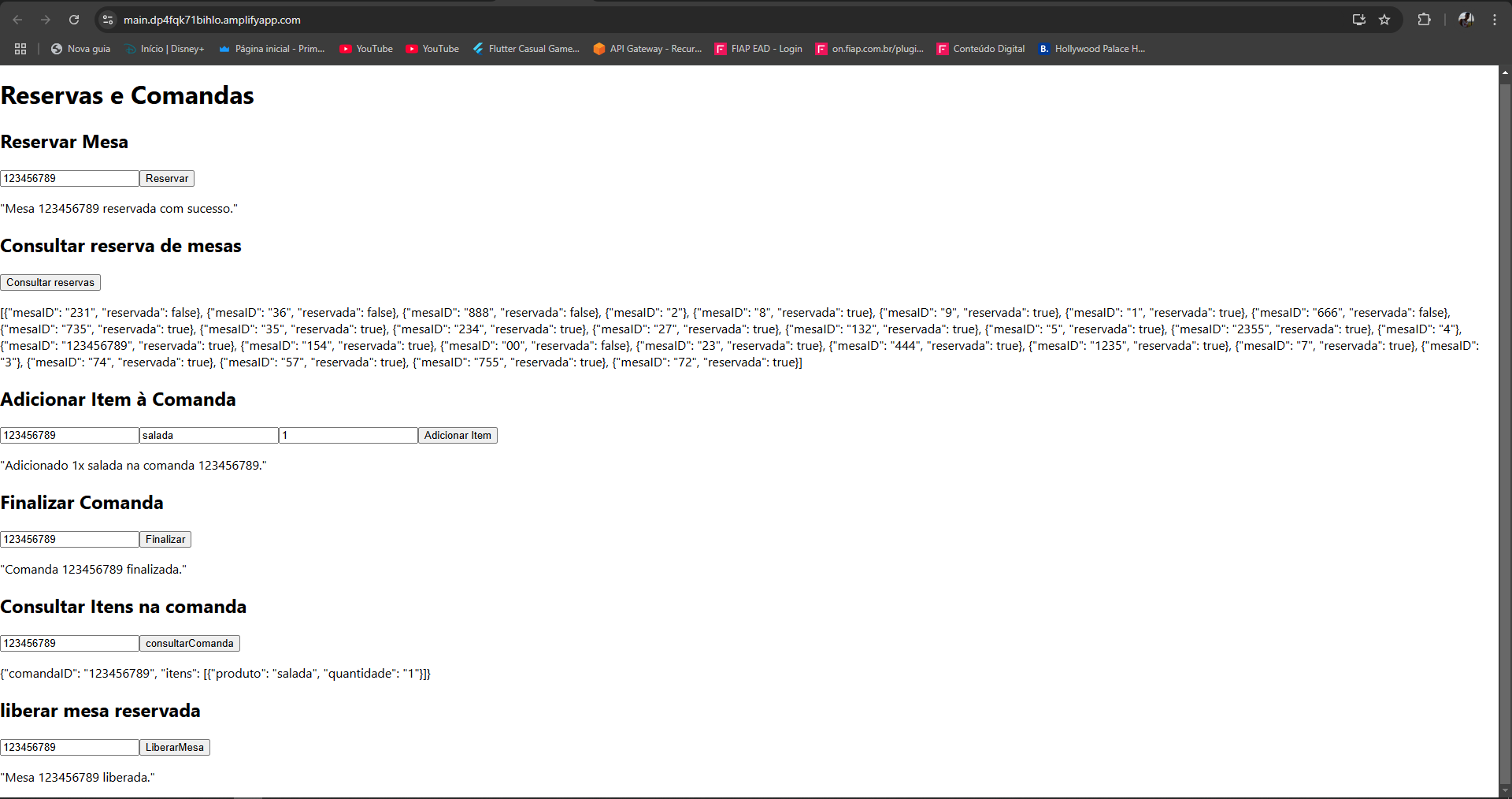
precos = {'Hamburguer': 15.0, 'Refrigerante': 5.0, 'Pizza': 20.0} # Preços fixos

return precos.get(produto, 0.0) # Retorna 0.0 se o produto não estiver listado

## Desafio encontrados:

De entender a forma que funciona os serviços utilizados, mas foram solucionados através da documentação da própria AWS.

# Teste da aplicação

Todos os campos foram testados, e os resultados correspondentes estão apresentados logo abaixo de cada um.  


# Conclusão e trabalhos futuros:

O desenvolvimento da aplicação de reserva de mesas e controle de comandas proporcionou um aprendizado valioso na integração de serviços AWS, como DynamoDB, Lambda, API Gateway e Amplify, para criar uma solução escalável, eficiente e segura. A aplicação alcançou seus objetivos principais, oferecendo uma experiência intuitiva para o gerenciamento de mesas e comandas, com funcionalidades robustas e processamento ágil.

No entanto, durante o processo, identificamos áreas que podem ser aprimoradas no futuro. Possíveis melhorias e expansões incluem:

* **Otimização do desempenho**: Reduzir latências em operações de leitura e escrita no DynamoDB, especialmente em cenários de alta concorrência.
* **Relatórios analíticos**: Implementar dashboards que apresentem relatórios detalhados sobre o uso de mesas e consumo de itens, utilizando serviços como AWS QuickSight.
* **Notificações em tempo real**: Adicionar notificações para clientes ou gerentes sobre atualizações de reservas ou mudanças no status das comandas, por meio de AWS SNS.
* **Integração com pagamentos**: Incorporar funcionalidades para pagamento direto na aplicação, utilizando serviços como AWS Payment Gateway ou integração com APIs externas.
* **Suporte multilíngue**: Adaptar a interface da aplicação para suportar diferentes idiomas, melhorando a experiência do usuário.
* **Segurança avançada**: Implementar autenticação multifator e logs de auditoria para reforçar a segurança do sistema.

Essas melhorias visam não apenas ampliar a funcionalidade da aplicação, mas também torná-la ainda mais útil e adaptável às necessidades dos usuários, garantindo escalabilidade e relevância no longo prazo.

# Referências

* **Amazon Web Services Documentation**. DynamoDB, Lambda, API Gateway e Amplify. Disponível em: <https://aws.amazon.com/documentation/>
* **Tutoriais de Desenvolvimento AWS Amplify**. Exemplos práticos para configuração de aplicações web. Disponível em: https://docs.amplify.aws/.
* **Documentação Python**. Referência oficial para a linguagem Python. Disponível em: <https://www.python.org/doc/>
* **GitHub**: repositório onde esta todo o projeto desenvolvido. Disponível em: https://github.com/IgorTeodoroDEV/AWSreservaMesa