



# **Atividade Microserviços**

**Grupo:**  
**Matheus Lemes Tassara**  
**Vitória Barbosa da Silva**

**São Paulo, 15 de fevereiro de 2023**

## Etapa 1:

### Implementar algumas telas usando o framework AngularJS. Avaliar se vale a pena iniciar a montagem ou uso de um container.

---

As atividades foram baseadas num aplicativo como autosserviço de compra de bilhetes de embarque de trens do metrô.

Embora os membros do time não estejam acostumados com AngularJS, um frontend simples pode ser encontrado aqui: <https://github.com/mathtass/atv4-fiap>

A página inicial se resume em uma lista de usuários, pensando em um controle gerencial de todos os usuários da plataforma. Além disto, para implementar o uso de botões, foi implementado os botões “Comprar Bilhete” e “Usar Bilhete”, de modo que “Usar Bilhete” só aparece para aqueles que possuem um saldo disponível.



Abaixo, temos a visão do usuário *Vitória* com informações de acordo com o seu perfil, como Detalhes do usuário com login, saldo e número de bilhetes disponíveis:



Além disto, podemos ver outra tela com um formulário para compra de bilhetes e função para o carrinho de compras:

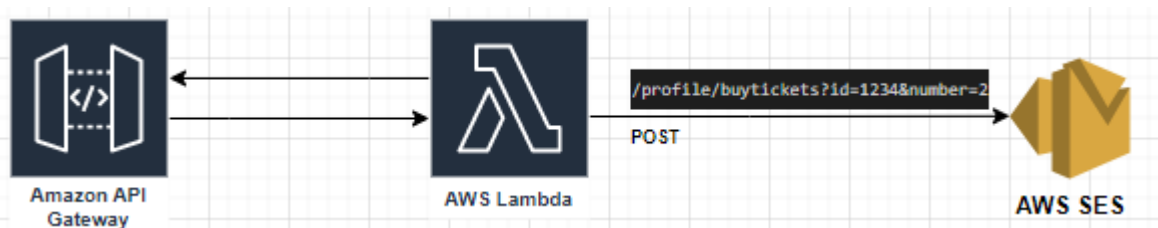
The screenshot shows a web browser at localhost:4200/cart. The header is dark red with 'Saldo Usuários' on the left and a shopping cart icon labeled 'Carrinho' on the right. Below the header, there's a light gray box containing 'Vitória Barbosa' and '\$2.00'. Underneath is a 'RECARGA' label and an empty input field. At the bottom is a red 'Comprar' button.

## Etapa 2:

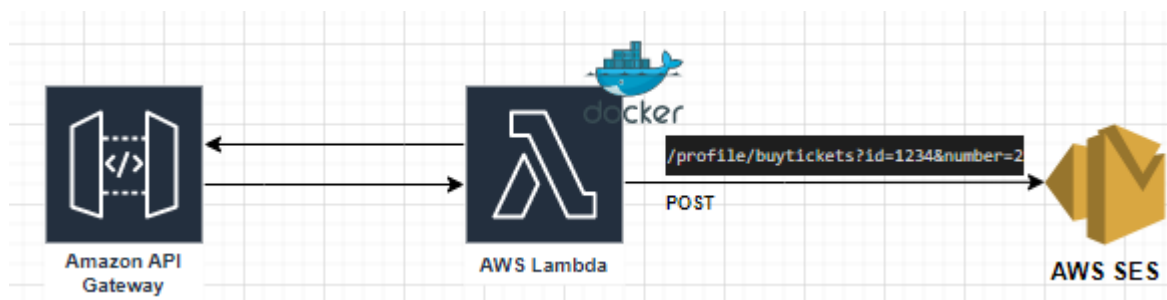
**Antes de implementar, discuta com o grupo a real necessidade, considerando as vantagens e pontos de atenção apresentados no curso!**

De acordo com o cenário, o uso de microsserviços não é aplicado a este projeto, em específico, pois sua arquitetura tem sido voltada para serverless com a integração AWS API Gateway x AWS Lambda, na qual fora desenvolvida em Python tendo em vista a familiaridade entre os membros do grupo.

Para que possa ser visualizado, o desenho da arquitetura idealizada está abaixo:



No entanto, para desenvolvimento da atividade, adaptamos a Lambda para uma Lambda Container com uso de Docker.



### Etapa 3:

A terceira etapa consiste em implementar o Container.

---

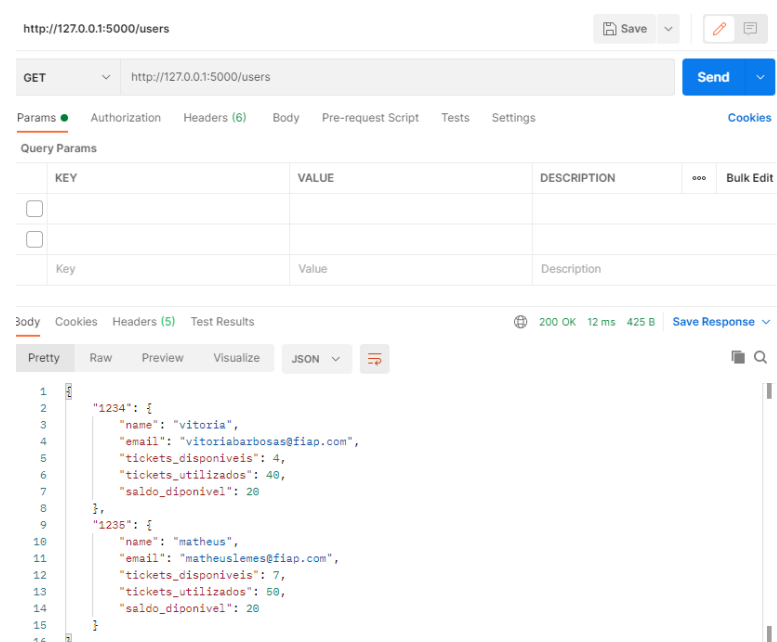
Esta atividade foi realizada em Python com Flask, e pode ser encontrado no repositório <https://github.com/vitoriabarbosas/pos/tree/main/Microservicos>

Seguindo a mesma ideia de um autosserviço que possa realizar a compra de tickets, foi dado o nome de RecargaTOP para esta aplicação.

Basicamente a API possui 4 rotas principais:

Rota	Método	Descrição
/users	GET	Acesso a todos os usuários
/profile	GET	Acesso a um perfil específico
/createuser	POST	Criação de novos usuários
/buytickets	POST	Compra de novos tickets

Na imagem abaixo, é possível ter um controle gerencial de todos os usuários da plataforma a partir da rota /users:



Pensando em criação de novos usuários, foi criada a rota /createuser, onde requisita os dados do novo usuário a ser criado:

http://127.0.0.1:5000/createuser

The screenshot displays a REST client interface with the following components:

- Request Section:**
  - Method: **POST**
  - URL: `http://127.0.0.1:5000/createuser`
  - Body Type: **JSON** (selected from a dropdown menu that also includes none, form-data, x-www-form-urlencoded, raw, binary, and GraphQL).
  - Body Content (lines 1-7):

```
1 {  
2   "name": "alberto",  
3   "email": "alberto@fiap.com",  
4   "tickets_disponiveis": 5,  
5   "tickets_utilizados": 50,  
6   "saldo_diponivel": 100  
7 }
```
- Response Section:**
  - Tab: **Body** (selected from Cookies, Headers (5), and Test Results).
  - Status: **200 OK** (indicated by a globe icon and green text).
  - Format: **JSON** (selected from Pretty, Raw, Preview, and Visualize).
  - Body Content (lines 1-4):

```
1 {  
2   "status": 200,  
3   "message": "Usuário Cadastrado com Sucesso!"  
4 }
```

Para este novo usuário, será gerado um ID randômico para funcionamento do sistema.

http://127.0.0.1:5000/users Save

GET ▼ http://127.0.0.1:5000/users Send ▼

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Key	Value	Description
-----	-------	-------------

Body Cookies Headers (5) Test Results 200 OK 9 ms 545 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼

```
1  {
2    "1234": {
3      "name": "vitoria",
4      "email": "vitoriabarbosas@fiap.com",
5      "tickets_disponiveis": 4,
6      "tickets_utilizados": 40,
7      "saldo_diponivel": 20
8    },
9    "1235": {
10     "name": "matheus",
11     "email": "matheuslemes@fiap.com",
12     "tickets_disponiveis": 7,
13     "tickets_utilizados": 50,
14     "saldo_diponivel": 20
15   },
16   "1": {
17     "name": "alberto",
18     "email": "alberto@fiap.com",
19     "tickets_disponiveis": 5,
20     "tickets_utilizados": 50,
21     "saldo_diponivel": 100
22   }
23 }
```

Para obter informações do perfil do usuário é possível utilizar a rota /profile, onde é requisitado o query parameter de acordo com o ID deste usuário. Em uma aplicação real, ao realizar o login na plataforma, este ID é passado por default entre as rotas.

http://127.0.0.1:5000/profile?id=1

GET http://127.0.0.1:5000/profile?id=1... Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	id	1			
	Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK 15 ms 282 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "alberto",
3   "email": "alberto@fiap.com",
4   "tickets_disponiveis": 5,
5   "tickets_utilizados": 50,
6   "saldo_diponivel": 100
7 }
```

Caso o usuário não exista:

http://127.0.0.1:5000/profile?id=167567867867

GET http://127.0.0.1:5000/profile?id=167567867867 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	id	167567867867			
	Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK 408 ms 226 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 400,
3   "message": "Usuário não encontrado!"
4 }
```

Além disto, temos a rota /buytickets para compra de bilhetes.

http://127.0.0.1:5000/profile/buytickets?id=1234&number=2

POST http://127.0.0.1:5000/profile/buytickets?id=1234&number=2

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	id	1234			
<input checked="" type="checkbox"/>	number	2			
	Key	Value	Description		

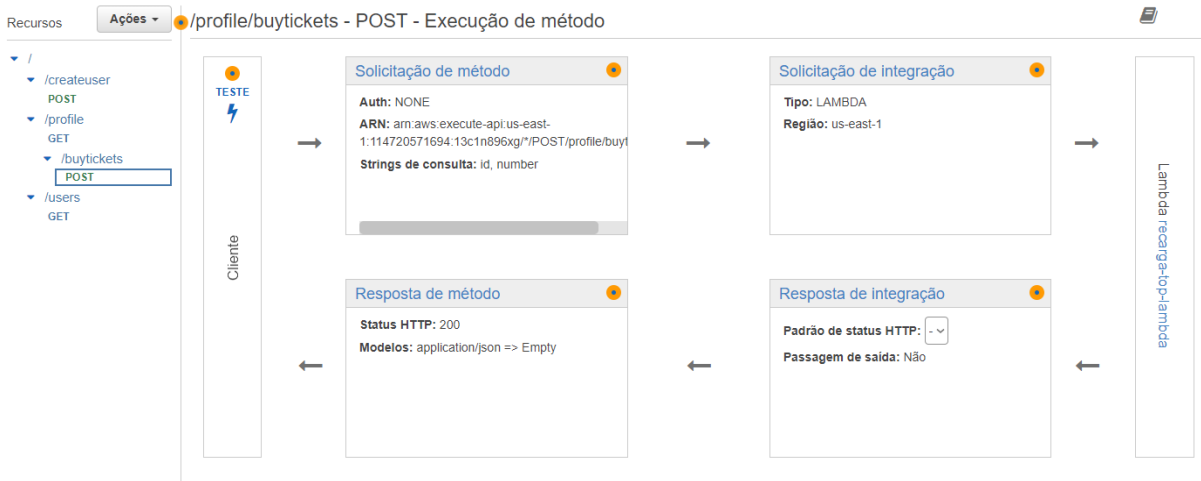
Body Cookies Headers (5) Test Results 200 OK 10 ms 224 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 200,
3   "message": "Obrigado por fazer esta compra!"
4 }
```

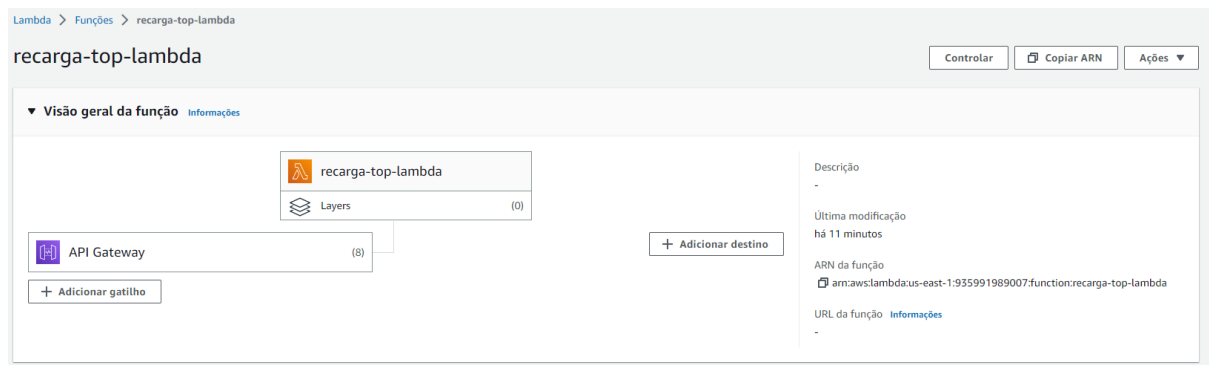
Componentes de infraestrutura na AWS:

## API Gateway



## Lambda





## Etapa 4:

### Implementação de um serviço de mensageria para que seja realizado o envio de mensagens como SMS e e-mail.

Por estarmos trabalhando na AWS podemos utilizar o SES (Simple Email Service), um serviço dedicado para envio de e-mails.

Para isto, quando uma requisição é feita na rota de compra de tickets, um email é enviado ao usuário solicitante:

Recarga Top: Você tem novos tickets disponíveis! ➡ Caixa de entrada x



vitoriabarbosas12@gmail.com por amazoneses.com  
para mim ▼

Olá, Vitória :)  
Muito obrigadx por utilizar o RecargaTop!  
Você tem 14 tickets disponíveis  
e um saldo de R\$ 14,00, que pode ser utilizado  
na compra de novos tickets!

Na aplicação, foi adotado um email noreply, no entanto, este email precisa ser criado e confirmado no SES para que o envio seja possível. Para execução dos testes, um email já existente foi adotado.

## Implementação da Infraestrutura na AWS SES:

Amazon SES > Configuração: Identidades verificadas

### Identidades verificadas

A identidade verificada é um domínio, subdomínio ou endereço de e-mail que você usa para enviar e-mails por meio do Amazon SES. [Saiba mais](#)

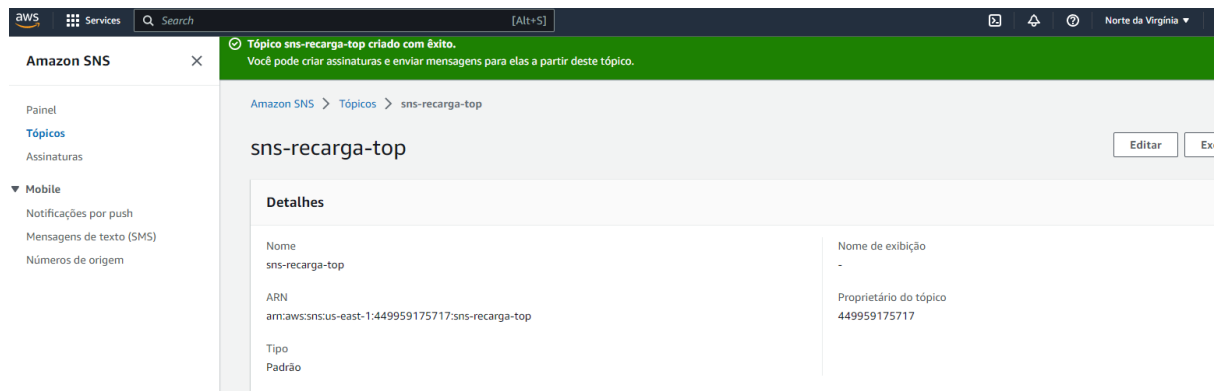
**Nova atualização de status de identidade**  
A **Status da identidade** agora representa a verificação explícita da identidade em si. Para as identidades de domínio, isso significa verificar a propriedade por meio de atualizações nos registros DNS, e para as identidades de endereços de e-mail, isso significa abrir o e-mail de verificação de `no-reply-aws@amazon.com` e selecionar o link para concluir o processo de verificação. [Saiba mais](#)

**Identidades (1)** [Informações](#)

Enviar e-mail de teste Excluir **Criar identidade**

<input type="checkbox"/>	Identidade	Tipo de identidade	Status da identidade
<input type="checkbox"/>	vitoriabarbosas12@gmail.com	Endereço de e-mail	Verificado

Por outro lado, também poderia ser utilizado o AWS SNS, no entanto, como a ideia do grupo foi realizar um email customizado, o AWS SES foi adotado.



## Referências:

[1]AWS Lambda Container -

<https://docs.aws.amazon.com/lambda/latest/dg/images-create.html>