

# Documentação

## Sistema de Semáforos Inteligente

Este projeto simula o funcionamento de um sistema de semáforos inteligente em um cruzamento. O sistema gerencia quatro semáforos (Norte, Sul, Leste e Oeste) e toma decisões automáticas para abrir ou fechar semáforos com base no número de veículos esperando e no tempo em que o semáforo permanece fechado.

### Arquitetura do Sistema:

O sistema é dividido em três partes principais:

1. **Back-end:** Implementado em Python, onde está toda a lógica do funcionamento dos semáforos.
2. **Front-end:** Responsável por exibir o estado dos semáforos em tempo real, utilizando HTML, CSS e JavaScript.
3. **Servidor Web:** Utilizando o Flask, para integrar o back-end com o front-end e permitir a visualização e interação com o sistema via navegador.

## 1. Backend

### 1.1. semaforos.py

Este arquivo contém a implementação da classe **Semaforo**, que é a base para o controle de cada semáforo individual. Abaixo está a documentação das principais funcionalidades:

#### Classe Semaforo:

##### Atributos:

- **direcao:** Define a direção que o semáforo controla (ex: Norte, Sul, etc.).
- **estado:** Define o estado atual do semáforo, sendo "verde" ou "vermelho".
- **veiculos:** Uma lista de veículos que estão esperando no semáforo.
- **tempo\_fechado:** Conta o número de ciclos em que o semáforo está fechado.
- **tempo\_aberto:** Conta o número de ciclos em que o semáforo está aberto.

### Métodos:

- **adicionar\_veiculo(veiculo):** Adiciona um novo veículo à lista de veículos esperando.
- **atualizar\_estado(estado\_novo):** Atualiza o estado do semáforo, mudando para verde ou vermelho e reseta o tempo aberto ou fechado conforme necessário.
- **liberar\_veiculos():** Libera todos os veículos se o semáforo estiver verde.
- **deve\_abrir():** Retorna **True** se o semáforo deve abrir, com base na quantidade de veículos ou no tempo em que está fechado.

### 1.2. veiculos.py

Define a classe **Veiculo**, que é usada para simular veículos chegando ao cruzamento. Cada veículo tem:

#### Atributos:

- **Id:** Um identificador único.
- **tempo\_chegada:** O tempo em que o veículo chegou ao semáforo.

### 1.3. sistema\_semaforo.py

O arquivo **sistema\_semaforo.py** contém a lógica de controle do sistema de semáforos, gerenciando os semáforos e determinando qual deles deve abrir. A classe principal é a **SistemaSemaforo**.

Classe **SistemaSemaforo**:

#### Atributos:

- **semaforos:** Um dicionário com quatro semáforos (Norte, Sul, Leste, Oeste).
- **tempo\_atual:** Um contador que representa o tempo total que o sistema está rodando.

### Métodos:

- **adicionar\_veiculos():** Simula a chegada de novos veículos aleatoriamente a cada direção.
- **atualizar():** Atualiza o estado dos semáforos, priorizando abrir o semáforo com mais veículos esperando.

- **exibir\_estados():** Exibe no console o estado atual dos semáforos e o número de veículos esperando.
- **executar(ciclos):** Executa o sistema por um número determinado de ciclos, atualizando a cada segundo.

## 2. Frontend

### Estrutura HTML e CSS:

O front-end consiste em um arquivo **index.html**, que renderiza o cruzamento e exibe os quatro semáforos. Cada semáforo é representado por uma imagem, que muda dinamicamente com base nos estados dos semáforos recebidos do back-end.

### HTML:

- Estrutura o layout do cruzamento com os quatro semáforos (Norte, Sul, Leste, Oeste).
- Usa fetch em JavaScript para obter o estado dos semáforos da rota ``/status`` e atualizar dinamicamente as imagens e contadores de veículos.

### CSS:

- Usa grid para posicionar os semáforos em uma estrutura de cruzamento.
- Desenha as ruas em um fundo preto que fica por trás dos semáforos.

### JavaScript:

O front-end faz requisições periódicas ao servidor Flask para obter o estado dos semáforos. A função **atualizarSemaforos** busca as informações da rota ``/status`` e atualiza o layout conforme necessário, mudando a cor dos semáforos e exibindo o número de veículos esperando.

## 3. Servidor Web (app.py)

O arquivo **app.py** gerencia a comunicação entre o front-end e o back-end usando o framework Flask. Ele cria o servidor, fornece as rotas para exibição do front-end e retorna o estado dos semáforos em formato JSON para atualização em tempo real.

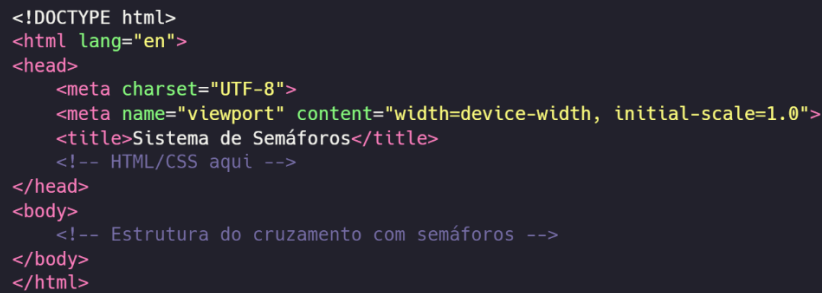
### 1.Rota / (GET):

- **Descrição:** Serve o arquivo index.html, que contém a interface gráfica do sistema de semáforos.
- **Método HTTP:** GET

- **Resposta:**

1. **HTML:** O conteúdo da página com o layout do cruzamento, onde os semáforos são exibidos.

2. **Exemplo de Resposta:**



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sistema de Semáforos</title>
  <!-- HTML/CSS aqui -->
</head>
<body>
  <!-- Estrutura do cruzamento com semáforos -->
</body>
</html>
```

## 2.Rota /status (GET):

- **Descrição:** Retorna o estado atual de todos os semáforos e o número de veículos esperando em cada um. Essa rota é chamada periodicamente pelo front-end para atualizar a interface gráfica.

- **Método HTTP:** GET

- **Resposta:** JSON com o estado dos semáforos (verde ou vermelho) e o número de veículos aguardando em cada direção.

- **Exemplo de Resposta:**



```
{
  "Norte": {
    "estado": "vermelho",
    "veiculos": 3
  },
  "Sul": {
    "estado": "verde",
    "veiculos": 0
  },
  "Leste": {
    "estado": "vermelho",
    "veiculos": 5
  },
  "Oeste": {
    "estado": "vermelho",
    "veiculos": 0
  }
}
```

## Explicação das Funcionalidades:

- A **rota /** serve o arquivo HTML com o layout do cruzamento, enquanto a **rota /status** é responsável por fornecer as informações dinâmicas do sistema, como o número de veículos e o estado dos semáforos.
- O front-end faz chamadas periódicas para a rota /status usando JavaScript (via fetch), recebendo dados no formato JSON e atualizando as imagens dos semáforos e o contador de veículos na interface do usuário.

## Exemplo de Como o Front-end Consome a Rota /status:

No arquivo index.html, o JavaScript utiliza a função fetch para obter o estado dos semáforos e atualizar a interface:



```
function atualizarSemaforos() {  
  fetch('/status')  
    .then(response => response.json())  
    .then(data => {  
      // Atualiza os semáforos e o número de veículos  
      for (const direcao in data) {  
        const semaforo = data[direcao];  
        const imgElement = document.getElementById(`img-${direcao.toLowerCase()}`);  
        const veiculosElement = document.getElementById(`veiculos-${direcao.toLowerCase()}`);  
  
        // Muda a imagem do semáforo e exibe o número de veículos  
        imgElement.src = semaforo.estado === "verde" ? "static/images/semaforo_verde.png" :  
        "static/images/semaforo_vermelho.png";  
        veiculosElement.textContent = `Veículos esperando: ${semaforo.veiculos}`;  
      }  
    });  
}  
  
// Atualiza os semáforos a cada 2 segundos  
setInterval(atualizarSemaforos, 2000);  
veiculos":  
}  
}
```

## Como o Sistema Funciona:

- O servidor Flask inicializa e serve a página `index.html`, que exibe a interface gráfica dos semáforos.
- O sistema de semáforos é executado em segundo plano (usando threading), simulando a chegada de veículos e decidindo qual semáforo abrir com base nas regras de priorização.
- O front-end faz requisições periódicas ao servidor, obtendo o estado atualizado dos semáforos e exibindo as informações ao usuário.
- As regras de prioridade garantem que o semáforo com mais veículos esperando ou que está fechado por mais tempo seja aberto.

## Ferramentas Utilizadas:

- **Flask:** Framework web usado para criar o servidor e gerenciar as rotas.
- **Threading:** Para executar o sistema de semáforos em paralelo ao servidor Flask.
- **HTML, CSS e JavaScript:** Para construir e atualizar a interface gráfica do cruzamento de semáforos em tempo real.