Apresentação da UC - PWBE

Prof. Israel Gomes da Silva
Prof. Fernanda Fretes

Fundamentos Técnicos e Científicos ou Capacidades Técnicas

1. Identificar as características de programação back-end em ambiente web

2. Preparar o ambiente necessário ao desenvolvimento back-end para a plataforma web

3. Definir os elementos de entrada, processamento e saída para a programação da aplicação web

4. Utilizar design patterns no desenvolvimento da aplicação web

5. Definir os frameworks a serem utilizados no desenvolvimento da aplicação web

6. Utilizar interações com base de dados para desenvolvimento de sistemas web

7. Transferir arquivos entre cliente e servidor por meio da aplicação web

Fundamentos Técnicos e Científicos ou Capacidades Técnicas

8. Estabelecer envio de notificações entre cliente e servidor por meio de aplicação web

9. Desenvolver API (web services) para integração de dados entre plataformas

10. Desenvolver sistemas web de acordo com as regras de negócio estabelecidas

11. Publicar a aplicação web

Apresentação da UC - PWBE

Prof. Israel Gomes da Silva
Prof. Fernanda Fretes

Capacidades Socias, Organizativas e Metodológicas

1. Demonstra atenção a detalhes;

2. Demonstrar capacidade de comunicação com profissionais de diferentes áreas e especialidades;

3. Demonstrar capacidade de organização;

4. Demonstrar raciocínio lógico na organização das informações;

5. Seguir método de trabalho;

6. Trabalhar em equipe.

Apresentação da UC - PWBE

Prof. Israel Gomes da Silva
Profa. Fernanda Fretes

Conhecimentos

1. Ambiente de desenvolvimento web

- Definição
- Histórico
- Características
- Ambiente de desenvolvimento
  - Instalação e configuração
  - Recursos e interfaces
  - Gerenciamento de dependências

2. Padrão de desenvolvimento MVC

- Definição
- Aplicabilidade
- Design patterns

Apresentação da UC - Projetos

Prof. Israel Gomes da Silva
Prof. Fernanda Fretes

Conhecimentos

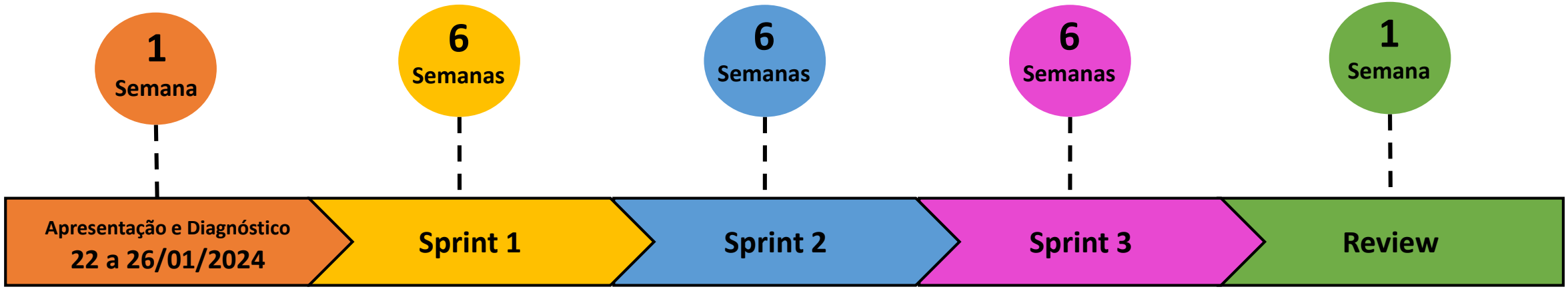
3. Frameworks
- Definição
- Modelos e tipos
- Instalação e configuração
- Criação de projetos utilizando framework

4. Persistência de dados
- Conexão com base de dados
- CRUD
- Transferência de arquivos locais para ambiente servidor
- Geração de relatórios
- Manipulação de dados utilizando XML
- Manipulação de dados utilizando JSON

5. Web Services
- Definição
- Integração de plataformas utilizando API
- Notificação entre cliente servidor utilizando Web Services
- publicação do web site

Escola e Faculdade de Tecnologia
SENAI "Roberto Mange"

PWBE – 2 DES-MB 1º S 2024



- Conhecendo a Turma
- Apresentação do Plano de Ensino

- **Atividade Formativa I**
- **Desenvolvimento de sistema web sem utilização de FrameWork**

- **Atividade Formativa II**
- **Desenvolvimento de sistema web com utilização de FrameWork**

- **Atividade Somativa**
- **Desenvolvimento do Projeto Integrador**

Smart City Roberto Mange

- Revisão final do curso

Escola e Faculdade de Tecnologia
SENAI "Roberto Mange"

Trabalho de PWBE – Cidades Inteligentes

COMUM A TODOS OS GRUPOS

- **Pesquisar sobre as tecnologias de comunicação, sensores e redes que formam a base da infra estrutura de uma cidade inteligente.**
- **Como a Internet das Coias (IoT) é utilizada para coletar dados em tempo real para melhorar a segurança, eficiência e qualidade de vida das pessoas.**
- **Experiências de implantação de Cidades Inteligentes pelo Brasil e pelo mundo.**

1

Mobilidade Urbana Inteligente

- Soluções para o transporte público, compartilhamento de veículos, gestão de tráfego, estacionamento inteligente.

2

Governança Digital e Participação Cidadã

- Como a tecnologia é empregada na governança da cidade incluindo a participação cidadã, transparência dos dados e tomada de decisões em tempo real.

3

Sustentabilidade Ambiental e Eficiência Energética

- Práticas sustentáveis em Cidades Inteligentes, gestão de resíduos, eficiência energética, qualidade do ar e iniciativas para redução da pegada de carbono

4

Qualidade de Vida e Bem Estar

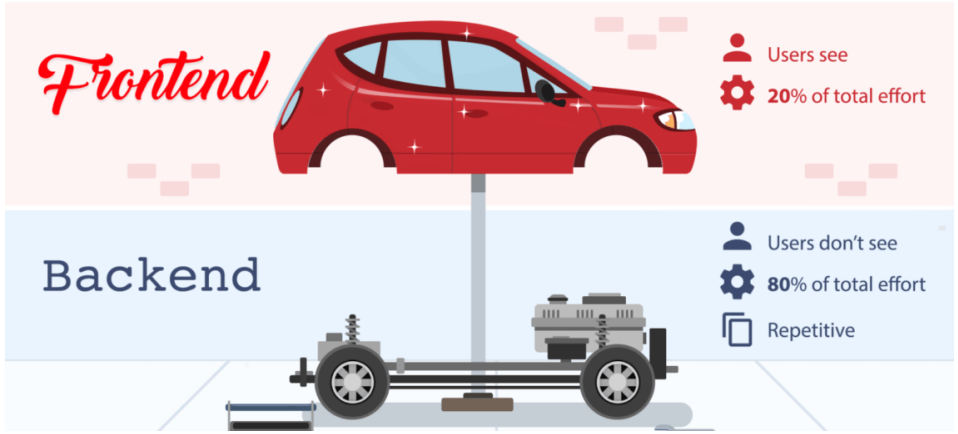
- Como a tecnologia melhora a qualidade de vida dos cidadãos com saúde, educação, segurança pública e iniciativas para promover uma cidade mais inclusiva e acessível

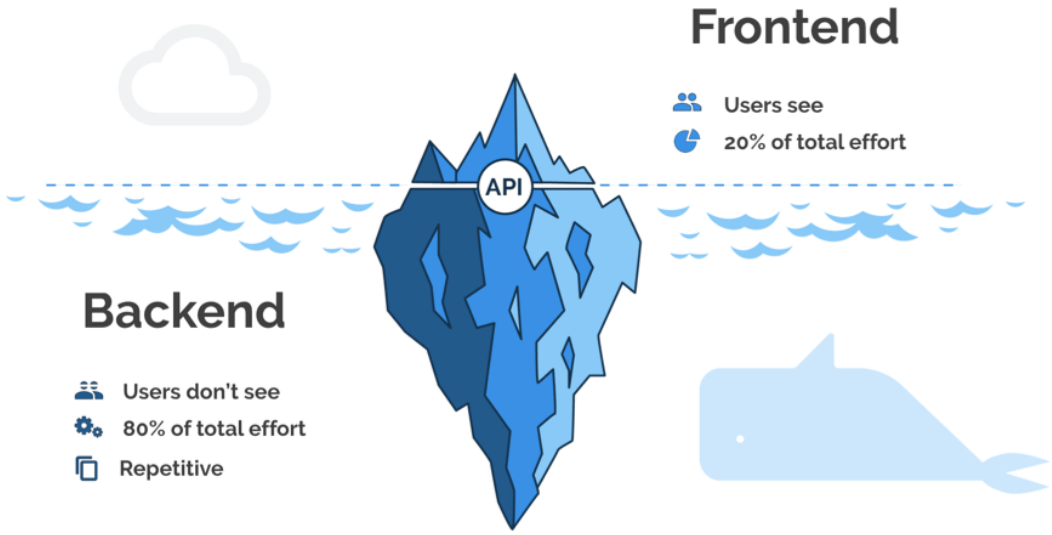
5

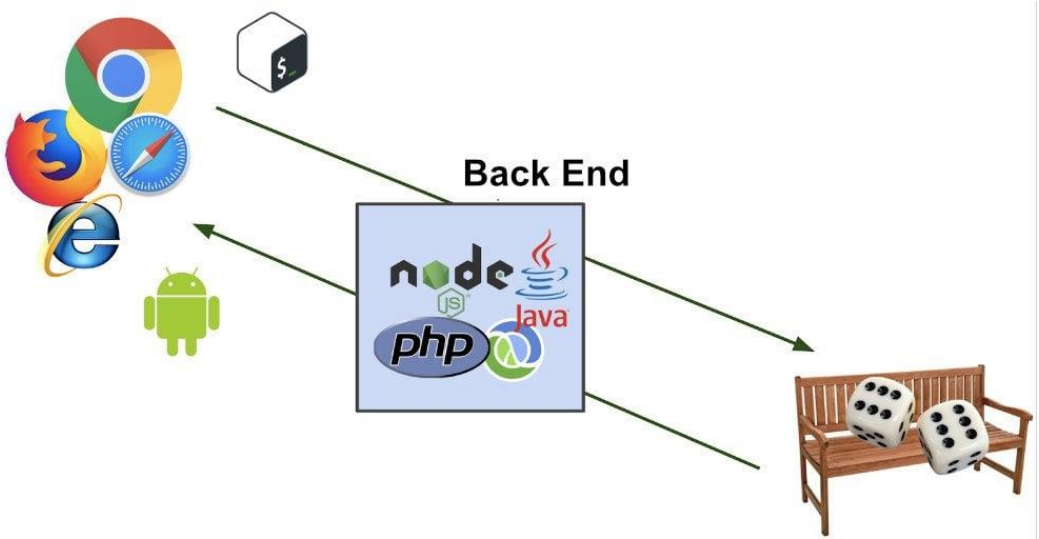
Inovação em Serviços Públicos

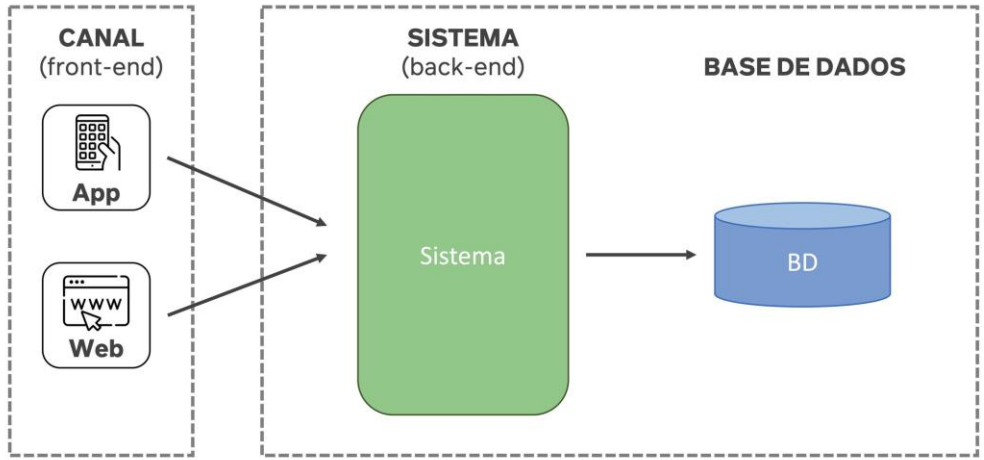
- Iniciativas inovadoras em serviços públicos que melhoram a administração municipal, trazem transparência e simplificam a interação dos cidadãos com os serviços governamentais.

O que é Programação Back End









Algumas considerações para alocação de programas/processos no Back End

Qual a responsabilidade principal desse processo ou programa ?

- Se envolve principalmente a interação com o usuário e a interface gráfica, provavelmente é FrontEnd.
- Se lida principalmente com a lógica de negócios e processamento de dados, pode ser BackEnd.
- Se gerencia o armazenamento e a recuperação de dados, é mais provável que seja Banco de Dados.

Qual é a interação direta com o usuário ?

- Se o processo ou programa está diretamente envolvido na interface do usuário, é mais provável ser FrontEnd.
- Se o usuário não interage diretamente com ele, pode ser BackEnd ou Banco de Dados.

Como esse processo ou programa se comunica com outros componentes do sistema ?

- Se está envolvido na comunicação com FrontEnd, pode ser FrontEnd ou BackEnd.
- Se está integrado com o armazenamento e recuperação de dados, pode ser BackEnd ou Banco de Dados.
- Se gerencia o armazenamento e a recuperação de dados, é mais provável que seja Banco de Dados.

Desafio

Desenvolver um sistema web do pátio de automóveis de uma fábrica, onde os funcionários poderão clicar em uma área e visualizar os automóveis que estão atualmente alocados para aquela área. Também deve ser possível, através desse sistema, vender automóveis de uma área qualquer do pátio. O funcionário clica sobre a área e o sistema exibe uma lista dos veículos que estão disponíveis naquela área podendo escolher reservar ou comprar o veículo desejado. Este benefício só é disponibilizado para funcionários com mais de 3 anos de empresa.

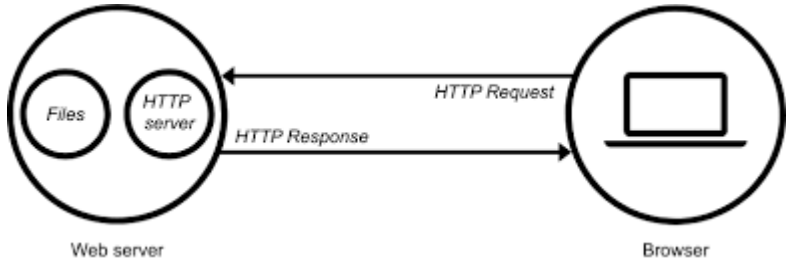
- A - Informar Login e Senha
- B - Criptografar Senha
- C - Comparar se senha informada é correta
- D - Permitir acesso ao sistema
- E - Buscar automóveis disponíveis por área do pátio
- F - Exibir automóveis por área do pátio
- G - Verificar disponibilidade em estoque
- H - Escolher veículo para compra
- I - Fazer reserva do veículo escolhido
- J - Efetuar o pagamento da compra de veículo
- K - Baixar veículo em estoque
- L - Emitir recibo de compra e venda
- M - Carregar estoque de veículos
- N - Exibir em azul áreas que têm veículos disponíveis
- O - Exibir em vermelho áreas que não têm veículos disponíveis
- P - Validar se funcionário tem mais de 3 anos de empresa

O que faz um Servidor Web

Receber Requisições HTTP:

O servidor web aceita solicitações HTTP (Hypertext Transfer Protocol) de clientes, que podem incluir solicitações para recuperar páginas da web, arquivos, ou executar outras operações.

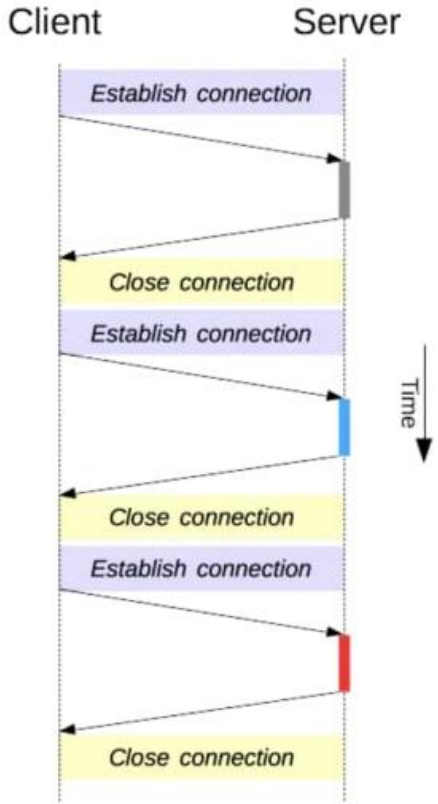
Processar Requisições: O servidor web interpreta e processa as solicitações recebidas, determinando qual conteúdo ou recurso deve ser enviado de volta ao cliente. Isso pode envolver a execução de scripts, a recuperação de dados de um banco de dados, ou outras operações específicas.



O que faz um Servidor Web

Gestão de Conexões:
Gerencia e mantém conexões com clientes. Isso inclui a abertura e o encerramento de conexões, mantendo o controle do estado das conexões e gerenciando a concorrência para manipular múltiplas solicitações simultâneas.

Envio de Respostas HTTP: Após processar a solicitação, o servidor web envia uma resposta HTTP de volta ao cliente. Isso pode incluir o envio de páginas HTML, arquivos de imagem, dados JSON, ou qualquer outro tipo de conteúdo solicitado.

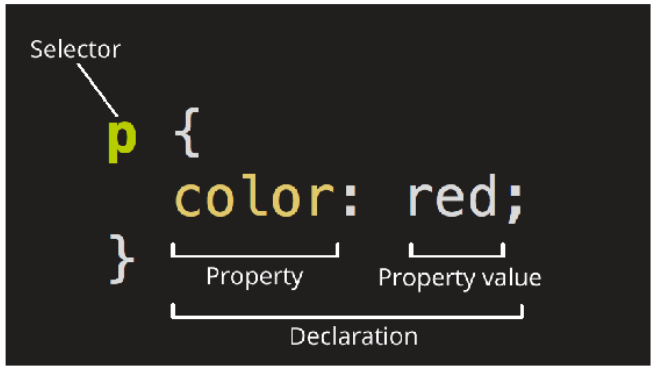


Short-lived connections

O que faz um Servidor Web

Hospedagem de Arquivos Estáticos: :
Servidores web muitas vezes são responsáveis por servir arquivos estáticos, como imagens, folhas de estilo CSS e scripts JavaScript, diretamente para os clientes.

Execução de Scripts: Quando necessário, o servidor web pode executar scripts do lado do servidor para gerar dinamicamente o conteúdo da página antes de enviá-lo ao cliente.



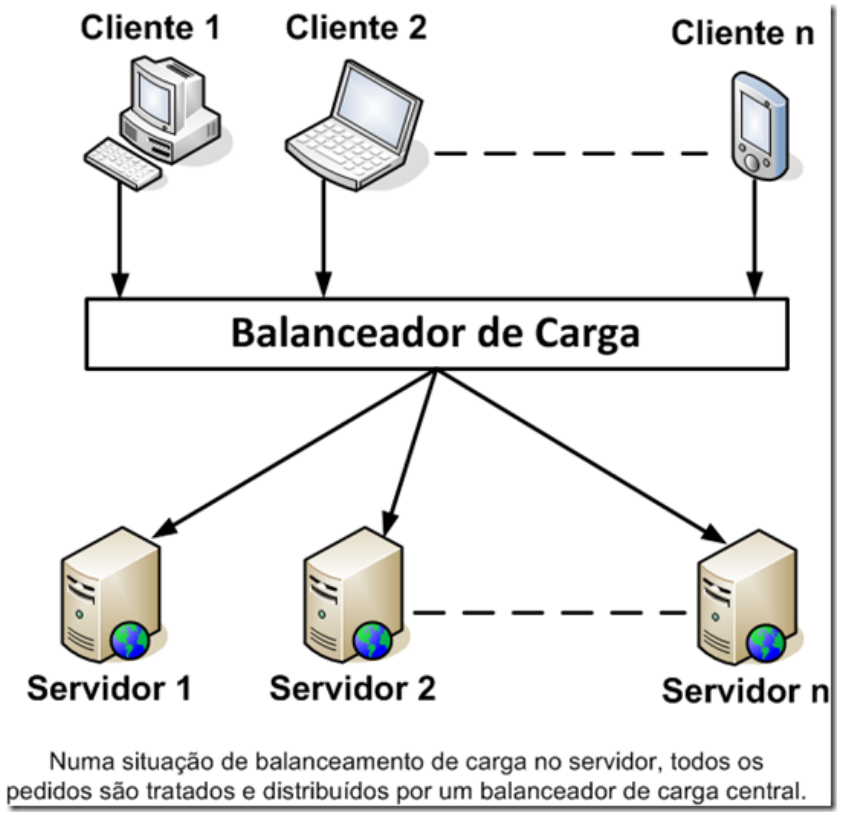
O que faz um Servidor Web

Segurança :
Implementa medidas de segurança para proteger contra ameaças.

Log de Acessos: Registra informações sobre as solicitações e acessos ao servidor em arquivos de log. Esses registros podem ser usados para monitoramento, análise de desempenho e resolução de problemas.

Controle de Acesso:
Gerencia o acesso aos recursos do servidor, determinando quais usuários ou clientes têm permissão para acessar determinados arquivos ou serviços.

Balanceamento de Carga: Em ambientes com alto tráfego, o servidor web pode distribuir as solicitações entre vários servidores para melhorar o desempenho e a disponibilidade, uma técnica conhecida como balanceamento de carga.



Numa situação de balanceamento de carga no servidor, todos os pedidos são tratados e distribuídos por um balanceador de carga central.

Criando um Web Server em Python

main.py

```python
# Importa o módulo http.server
import http.server
import socketserver

# Define a porta a ser utilizada
porta = 8000

# Configura o manipulador (handler) para o servidor
handler = http.server.SimpleHTTPRequestHandler

# Cria um servidor na porta especificada
with socketserver.TCPServer( server_address: ("", porta), handler) as httpd:
    print(f"Servidor iniciado na porta {porta}")
    # Mantém o servidor em execução
    httpd.serve_forever()

```

Criando um Web Server em Python

```
with socketserver.TCPServer(("", porta), handler) as httpd:
```

cria uma instância do servidor TCP utilizando o módulo socketserver em Python.

```
("", porta)
```

("", porta): Define a tupla ("", porta) como o endereço no qual o servidor vai escutar por conexões. "" significa que o servidor aceitará conexões em todos os IPs disponíveis na máquina, e porta é o número da porta na qual o servidor estará escutando.

```
5   # Define a porta a ser utilizada
6   endereco_ip = "0.0.0.0" # Escuta qualquer endereço IP
7   porta = 8000
```

# Define o IP e a porta a serem utilizados
endereco_ip = "0.0.0.0" # Escuta solicitações de qualquer IP
porta = 8000

```
with socketserver.TCPServer( server_address: (endereco_ip, porta), handler) as httpd:
```

Criando um Web Server em Python

```
with socketserver.TCPServer(("", porta), handler) as httpd:
```

cria uma instância do servidor TCP utilizando o módulo socketserver em Python.

```
("", porta)
```

("", porta): Define a tupla ("", porta) como o endereço no qual o servidor vai escutar por conexões. "" significa que o servidor aceitará conexões em todos os IPs disponíveis na máquina, e porta é o número da porta na qual o servidor estará escutando.

```
5   # Define a porta a ser utilizada
6   endereco_ip = "0.0.0.0" # Escuta qualquer endereço IP
7   porta = 8000
```

# Define o IP e a porta a serem utilizados
endereco_ip = "0.0.0.0" # Escuta solicitações de qualquer IP
porta = 8000

```
with socketserver.TCPServer( server_address: (endereco_ip, porta), handler) as httpd:
```

Criando um Web Server em Python

Fazendo com que a página "**index.html**" seja a primeira página a ser exibida (parte 1 de 2)

```python
import os
from http.server import SimpleHTTPRequestHandler
import socketserver

class MyHandler(SimpleHTTPRequestHandler):
    def list_directory(self, path):
        try:
            # Tenta abrir o arquivo index.html
            f = open(os.path.join(path, 'index.html'), 'r')
            # Se existir, envia o conteúdo do arquivo
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write(f.read().encode('utf-8'))
            f.close()
            return None
        except FileNotFoundError:
            pass

        return super().list_directory(path)
```

Criando um Web Server em Python

Fazendo com que a página "**index**.**html**" seja a primeira página a ser exibida (parte 2 de 2)

```
# Define o IP e a porta a serem utilizados
endereco_ip = "0.0.0.0"
porta = 8000

# Cria um servidor na porta e IP especificados
with socketserver.TCPServer((endereco_ip, porta), MyHandler) as httpd:
    print(f"Servidor iniciado em {endereco_ip}:{porta}")
    # Mantém o servidor em execução
    httpd.serve_forever()
```

Criando um Web Server em Python

Explicando o código

```
class MyHandler(SimpleHTTPRequestHandler):
```

Criamos uma nova classe chamada MyHandler que herda da classe SimpleHTTPRequestHandler.
Essa nova classe personalizada permite modificar o comportamento padrão do manipulador de solicitações HTTP.

```
def list_directory(self, path):
```

Esta é uma sobrescrita do método list_directory da classe pai SimpleHTTPRequestHandler.
Este método é chamado quando o servidor precisa listar o conteúdo de um diretório.
Aqui, iremos sobrescrever este método, para personalizar o comportamento padrão.

```
try:
    Fazer alguma coisa
except  FileNotFoundError:
```

Try tenta fazer alguma coisa que são as linhas que seguem entre "try" e "except". O "Fazer alguma coisa" será uma sequência de comandos que comentaremos em seguida com a finalidade de fazer exibir a página index.html. Caso encontre erro nesta tentativa ele executará a função que estiver no "except"

Criando um Web Server em Python

Explicando o código

```
f = open(os.path.join(path, 'index.html'), 'r')
```

Tenta abrir o arquivo index.html no diretório especificado (path) para leitura.

```
# Se existir, envia o conteúdo do arquivo
self.send_response(200)
```

Envia uma resposta HTTP 200 (OK) para indicar que a solicitação foi bem-sucedida.

```
self.send_header("Content-type", "text/html")
self.end_headers()
```

Adiciona um cabeçalho à resposta indicando que o tipo de conteúdo é HTML e finaliza a parte de cabeçalho da resposta.

```
self.wfile.write(f.read().encode('utf-8'))
f.close()
```

Escreve o conteúdo do arquivo index.html no objeto wfile, que representa o fluxo de saída para o cliente.
O conteúdo é lido do arquivo e codificado para UTF-8 antes de ser enviado.
Em seguida fecha o arquivo index.html após a leitura.

```
return None
```

Retorna None para indicar que o método foi tratado e não precisa responder mais nada.

Criando um Web Server em Python

Explicando o código

```
except FileNotFoundError:
    pass
```

"FileNotFoundError" é uma classe em Python que faz parte das exceções padrão do Python.
Essa classe é geralmente levantada quando um arquivo ou diretório não pode ser encontrado durante uma operação que envolve manipulação de arquivos.

"pass" Neste caso, não faz nada se o arquivo não for encontrado.
Isso permite que o código continue sendo executado.

```
return super().list_directory(path)
```

Se o arquivo index.html não for encontrado, chama o método list_directory da classe pai (SimpleHTTPRequestHandler) para continuar com o comportamento padrão de listar o conteúdo do diretório.

Criando um Web Server em Python

Criando a rota "login". Para isso acrescente esta alteração na classe MyHandler

```
def do_GET(self):
    if self.path == '/login':
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write(b"<html><body><h1>Minha pagina de login</h1></body></html>")
    else:
        # Se não for a rota "/login", continua com o comportamento padrão
        super().do_GET()
```

Explicando o código

O método "do_GET" é parte da classe SimpleHTTPRequestHandler em Python dentre outros métodos.
Ao estender a classe SimpleHTTPRequestHandler como fizemos aqui com a classe MyHandler,
estamos alterando método do_GET para personalizar o comportamento ao manipular solicitações GET específicas.
Isso permite que você adicione lógica personalizada para rotas específicas, como aqui que adicionamos a rota "/login".

Construindo uma Aplicação

Fazendo exibir uma página específica na rota "login".

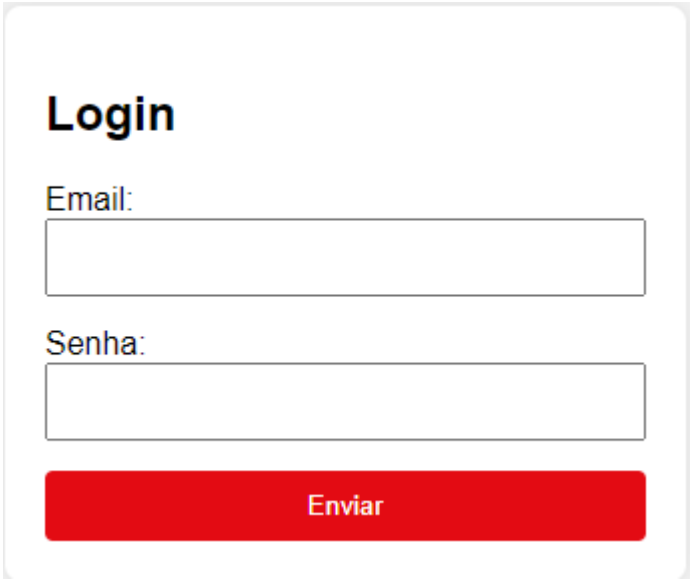
```python
def do_GET(self):
    if self.path == '/login':
        # Tenta abrir o arquivo login.html
        try:
            with open(os.path.join(os.getcwd(), 'login.html'), 'r') as login_file:
                content = login_file.read()
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write(content.encode('utf-8'))
        except FileNotFoundError:
            self.send_error(404, "File not found")
    else:
        # Se não for a rota "/login", continua com o comportamento padrão
        super().do_GET()
```

Explicando o código

Agora observe que alteramos o método "do_GET" para que responda a página login.html.
A rotina é a mesma já utilizada quando fizemos aparecer uma outra página em substituição a index.html .
Abre o arquivo na linha `with open(os.path.join(os.getcwd(), 'login.html'), 'r') as login_file: ,` e na sequência envia resposta 200 dizendo que está OK a solicitação, envia o cabeçalho da resposta, escreve o arquivo da resposta na linha `self.wfile.write(content.encode('utf-8'))`

Construindo uma Aplicação

Exibindo a página login.html




```
<body>

  <div class="login-container">
    <div class="error-message">
      <!-- Mensagem de erro será inserida aqui -->
    </div>
    <h2>Login</h2>
    <form action="/enviar_login" method="post">
      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required>

      <label for="senha">Senha:</label>
      <input type="password" id="senha" name="senha" required>

      <button type="submit">Enviar</button>
    </form>
  </div>

</body>
```

Explicando o código

Ponto chave do seu código html para a aplicação que estaremos construindo é a tag "<form> </form>". Nela constarão os campos de dados que serão enviados para a aplicação. É nela também que definimos o endereço para onde enviaremos esses dados e o método se GET ou POST e isto está na linha `<form action="/enviar_login" method="post">`

Construindo uma Aplicação

Criando a rota "enviar_login".

```
from urllib.parse import parse_qs
```

```
def do_POST(self):
    # Verifica se a rota é "/enviar_login"
    if self.path == '/enviar_login':
        # Obtém o comprimento do corpo da requisição
        content_length = int(self.headers['Content-Length'])
        # Lê o corpo da requisição
        body = self.rfile.read(content_length).decode('utf-8')
        # Parseia os dados do formulário
        form_data = parse_qs(body)

        # Exibe os dados no terminal
        print("Dados do formulário:")
        print("Email:", form_data.get('email', [''])[0])
        print("Senha:", form_data.get('senha', [''])[0])

        # Responde ao cliente
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write("Dados recebidos com sucesso!".encode('utf-8'))
    else:
        # Se não for a rota "/submit_login", continua com o comportamento padrão
        super(MyHandler, self).do_POST()
```

Construindo uma Aplicação

Criando a rota “enviar_login”.

Explicando o código

```
if self.path == '/enviar_login':
```

Verifica se o caminho chamado é /enviar_login

```
# Obtém o comprimento do corpo da requisição
content_length = int(self.headers['Content-Length'])
```

é necessário para saber a quantidade de dados que está sendo enviada no corpo da solicitação POST.
Sem isso o servidor não saberia quando parar de ler e poderia ficar aguardando indefinidamente.

```
# Lê o corpo da requisição
body = self.rfile.read(content_length).decode('utf-8')
```

```
# Parseia os dados do formulário
form_data = parse_qs(body)
```

A função “parse_qs” transforma os dados da string no formato de uma query string em um dicionário Python.
Exemplo, o corpo da requisição POST manda assim “login=BarackObama&senha=123456", a chamada parse_qs resultará em um dicionário assim: {‘login': [' BarackObama '], ' senha ': [' 123456 ']}
Esses dados podem então ser utilizados pelo servidor para processar as informações enviadas pelo cliente, neste caso, para obter o valor do campo “login" e “senha" do formulário.

Construindo uma Aplicação

Criando a rota "enviar_login".

Explicando o código

```
if self.path == '/enviar_login':
```

Verifica se o caminho chamado é /enviar_login

```
# Obtém o comprimento do corpo da requisição
content_length = int(self.headers['Content-Length'])
```

é necessário para saber a quantidade de dados que está sendo enviada no corpo da solicitação POST.
Sem isso o servidor não saberia quando parar de ler e poderia ficar aguardando indefinidamente.

```
# Lê o corpo da requisição
body = self.rfile.read(content_length).decode('utf-8')
```

```
# Parseia os dados do formulário
form_data = parse_qs(body)
```

A função "parse_qs" transforma os dados da string no formato de uma query string em um dicionário Python.
Exemplo, o corpo da requisição POST manda assim "**login=BarackObama&senha=123456**", a chamada **parse_qs** resultará em um dicionário assim: **{'login': [' BarackObama '], ' senha ': [' 123456 ']}**
Esses dados podem então ser utilizados pelo servidor para processar as informações enviadas pelo cliente, neste caso, para obter o valor do campo "login" e "senha" do formulário.

Construindo uma Aplicação

Gravando os dados recebidos da página de login

```
# Exibe os dados no terminal
print("Dados do formulário:")
print("Email:", form_data.get('email', [''])[0])
print("Senha:", form_data.get('senha', [''])[0])

# Armazena os dados em um arquivo TXT
with open('dados_login.txt', 'a') as file:
    login = form_data.get('email', [''])[0]
    senha = form_data.get('senha', [''])[0]
    file.write(f"{login};{senha}\n")

# Responde ao cliente
self.send_response(200)
self.send_header("Content-type", "text/html")
self.end_headers()
self.wfile.write("Dados recebidos e armazenados com sucesso no arquivo dados_login.txt!".encode('utf-8'))
```