EXPRESS

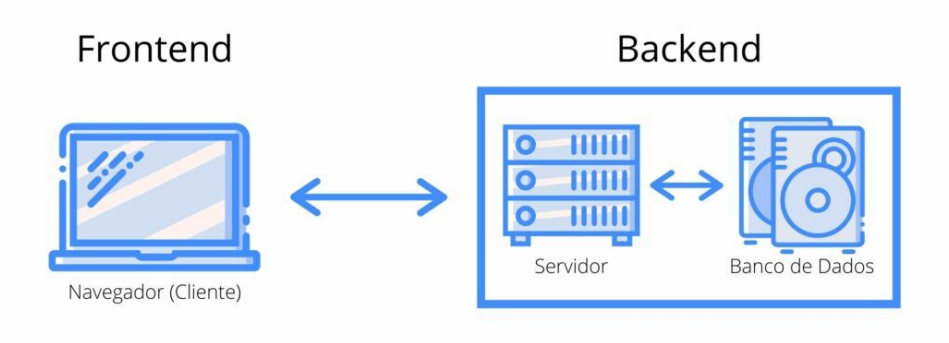
Framework para back-end

SAE COMP

FUNCIONAMENTO DE UM BACK-END

Fluxo do back-end

O back-end é um programa que roda em um servidor remoto e que clientes (navegadores) podem fazer requisições de conteúdo. Em um fluxo básico, o back-end recebe o request, faz alguma operação do banco de dados e retorna uma resposta para o cliente.



Request HTTP

O HTTP é o protocolo de comunicação usado nos navegadores. Ele funciona enviando uma mensagem que tem uma URI, um tipo de requisição, uma header e um body de requisição.

Request

```
POST / HTTP/1.1
Host: example.com/listener
User-Agent: curl/8.6.0
Accept: */*
Content-Type: application/json
Content-Length: 345

{
  "data": "ABC123"
}
```

← Request headers

← Representation headers

Response HTTP

Depois de receber um request, o servidor tem que necessariamente enviar uma resposta. Essa resposta vai ter uma header, um status e um body que contém a informação retornada.

Response

```
HTTP/1.1 200 OK
Server: Apache
Date: Fri, 21 Jun 2024 12:52:39 GMT
Cache-Control: public, max-age=3600
Content-Type: text/html
ETag: "abc123"
Last-Modified: Thu, 20 Jun 2024 11:30:00 GMT

<!DOCTYPE html>
<html lang="en"
(more data)
```

← Response headers

← Representation headers

HTTP STATUS CODES

2xx Success

- 200 Success / OK

3xx Redirection

- 301 Permanent Redirect
- 302 Temporary Redirect
- 304 Not Modified

4xx Client Error

- 401 Unauthorized Error
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed

5xx Server Error

- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout

API EXPRESS SIMPLES

Instalando

O express usa o npm para ser instalado junto com suas dependências.

```
# cria uam pasta para o projeto
mkdir api-ts
# entra na pasta
cd api-ts
# inicializa o projeto com o npm
npm init -y
# instala o express
npm install express
# instala o tsx
npm install -D typescript ts-node-dev @types/node @types/express
# inicializa o typescript
npx tsc --init

# cria o arquivo principal
mkdir -p src && touch src/index.ts
```

```
# adicionar dentro de scripts no package.json
"scripts": {
  "dev": "ts-node-dev src/index.ts"
}

# roda o projeto no modo dev
npm run dev
```

Estrutura básica

Para o express funcionar, deve-se definir o seu objeto com app e fazer com que ele escute em uma porta específica.

```
1  import express, { Request, Response } from 'express';
2
3  const app = express();
4  const port = 3000;
5
6  app.use(express.json());
7
8  app.listen(port, () => {
9    console.log(`serving on http://localhost:${port}`);
10 });
```

Rotas

Cada rota do express deve ter uma função associada. Elas podem ser de qualquer método do HTTP, sendo que deve haver uma função para cada um.

```
const produtos: Produto[] = [
  { id: 1, nome: 'Camisa', preco: 49.9 },
  { id: 2, nome: 'Calça', preco: 89.9 },
  { id: 3, nome: 'Tênis', preco: 199.9 }
];

app.get('/produtos', (req: Request, res: Response) => {
  res.json(produtos);
});
```

```
app.post('/produtos', (req: Request, res: Response) => {
  const novoProduto: Produto = req.body;
  novoProduto.id = produtos.length + 1;
  produtos.push(novoProduto);
  res.status(201).json(novoProduto);
});
```

Rotas dinâmicas

Uma rota pode ter uma parte dela que não tem um valor definido e pode ser utilizado para realizar queries dentro da função.

```
app.get('/produtos/:id', (req: Request, res: Response) => {
  const id = parseInt(req.params.id);

  const produto = produtos.find((p) => p.id === id);
  if (!produto) {
    res.status(404).json({ message: 'Produto não encontrado' });
    return;
  }
  res.json(produto);
});
```

Testando a API

Para testar a API criado, podemos utilizar várias opções como postman e Insomnia. No exemplo, será utilizado o curl que é uma ferramenta de terminal para fazer requests HTTP.

```
curl -X GET localhost:3000/produtos

curl -X POST http://localhost:3000/produtos \
  -H "Content-Type: application/json" \
  -d '{"nome": "Boné", "preco": 29.9}'
```

ATIVIDADES

Atividade 1

Fazer um sistema de cadastro de tarefas

Crie uma aplicação em TypeScript que permita:

- Adicionar uma tarefa
- Listar todas as tarefas
- Marcar uma tarefa como concluída
- Filtrar tarefas por status (pendente, concluída)

Requisitos:

- Usar uma interface Tarefa com id, titulo, status
- Usar um enum Status com Pendente e Concluida
- Criar funções separadas: adicionarTarefa, listarTarefas, concluirTarefa, filtrarPorStatus

Implementar uma API no express que tenha as rotas:

- GET /tarefas -> Listar todas as tarefas
- POST /tarefas -> Adicionar uma nova tarefa
- PATCH /tarefas/:id -> Marcar tarefa como concluída
- GET /tarefas/status/:status -> Listar tarefas por status (pendente/concluída)

MUITO OBRIGADO


@saecomp.ec


saecomp@usp.br


saecomp.github.io


Prédio da Engenharia de Computação,
Campus 2, USP São Carlos

SAECOMP