JAVASCRIPT

DEUS ME LIVRE
MAS QUEM ME DERA

CAMILLA MARTINS E THAYSE ONOFRIO

CAMILLA MARTINS

Punk paulista

Software Consultant

Santista fanática

Zagueira no PampaCats

Universo Marvel

Foragida há 3 anos

Node.Js e NodeGirls

Professora de alemão

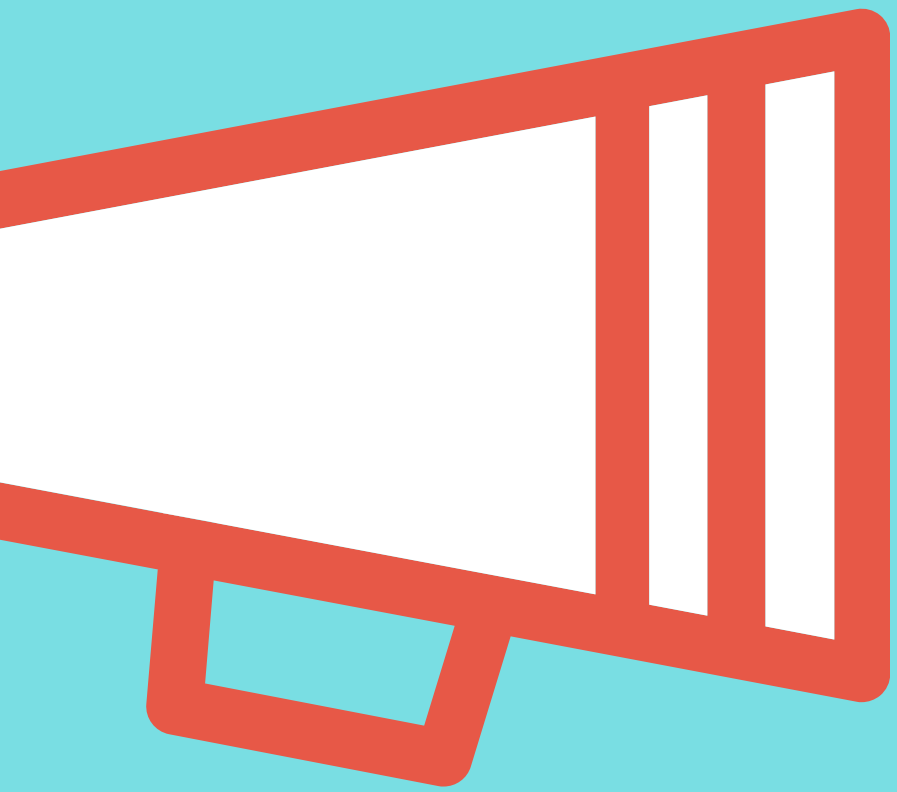
JAVASCRIPT: DEUS ME LIVRE MAS QUEM ME DERA

THAYSE ONOFRIO

25 anos
Publicitária
(quase) Tecnóloga em SI
Consultora de Desenvolvimento
AfroPython
Diversidade

JAVASCRIPT: DEUS ME LIVRE MAS QUEM ME DERA


JAVASCRIPT



o estilo do código importa

Consistência no estilo do código torna ele mais fácil de ler, alterar e integrar

o que são styles guides?

Style guides são conjuntos de padrões que definem como o código deve ser escrito e organizado.

Style Guides

Airbnb

ponto e vírgula: sim
indentação: 2 espaços
aspas: simples
espaço depois do nome da função: não

Google

ponto e vírgula: sim
indentação: 2 espaços
aspas: simples
espaço depois do nome da função: não

Standard

ponto e vírgula: não
indentação: 2 espaços
aspas: simples
espaço depois do nome da função: sim

MANTENDO O ESTILO COM

FERRAMENTAS

prettier

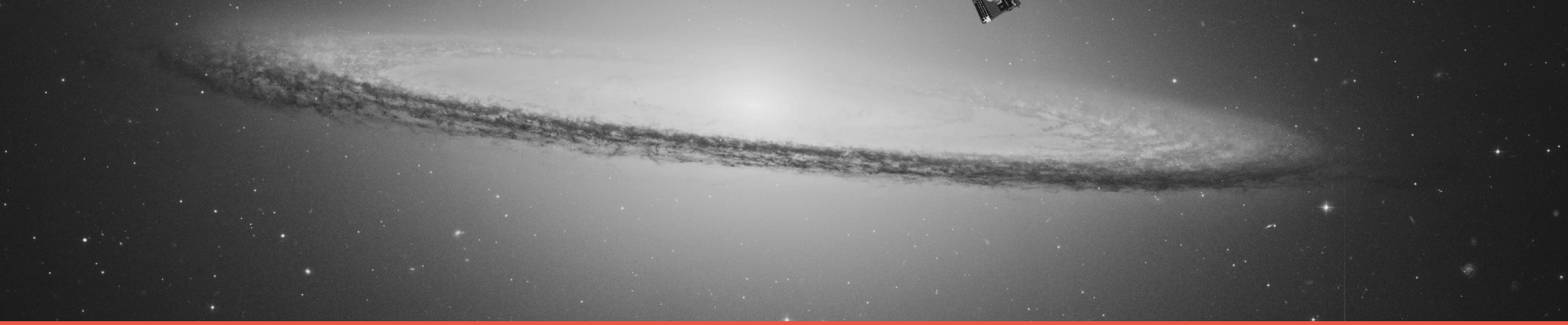
é um formatador de código que força um estilo consistente analisando o seu código e reescrevendo com suas próprias regras

ESlint

é um linter - analisa o código procurando erros, desde o estilo até erros no código.

"Qualquer um consegue escrever código que um computador consegue entender. Boas programadoras escrevem código que humanos conseguem entender."

—MARTIN FOWLER



o que é código limpo?

Código limpo é escrever código para pessoas, não máquinas. O código limpo é fácil de entender e de alterar.

Use nomes de variáveis que realmente querem dizer alguma coisa

```
const l = []
const lista1 = []
const dds = []
```

VS

```
const diasDaSemana =
['segunda', 'terça', 'quarta'];
```

Não adicione informações desnecessárias

```
const Pessoa = {
  nomeDaPessoa: 'Maria',
  idadeDaPessoa: '20'
}
```

VS

```
const Pessoa = {
  nome: 'Maria',
  idade: '20'
}
```

Funções devem fazer só uma coisa

```
comprarProdutos = (produtos) => {
  produtos.forEach((produto) => {
    const dadosDoProduto =
bancoDeDados.buscar(produto);
    if
(dadosDoProduto.estaDisponivel()) {
      comprar(produto);
    }
  });
 }
```

VS

```
comprarProdutosDisponiveis =
(produtos) => {
  produtos
    .filter(verificarDisponibilidade)
    .forEach(comprar);
}
 verificarDisponibilidade =
(produto) => {
  const dadosDoProduto =
bancoDeDados.buscar(produto);
  return
dadosDoProduto.estaDisponivel();
 }
```

Remova código duplicado

```
buscarDesenvolvedoras = () => {
   return funcionarios.filter(funcionario
=> funcionario.tipo ===
"desenvolvedora");
}

buscarAnalistas = () => {
   return funcionarios.filter(funcionario
=> funcionario.tipo === "analista");
}

buscarAnalistas()
buscarDesenvolvedoras()
```

VS

```
buscarFuncionariosPorTipo = (tipo) => {
   return funcionarios.filter(funcionario
=> funcionario.tipo === tipo);
}

buscarFuncionariosPorTipo("analista");
buscarFuncionariosPorTipo("desenvolved
ora");
```

Menos Callbacks, mais Promises

```
get('uri', (primeiroErro,
primeiraResposta) => {
  get(`uri/${primeiraResposta.valor}`,
(segundoErro, segundaResposta) => {
    console.log(segundaResposta);
  });
});
```

VS

```
return new Promise((resolver, rejeitar) => {
  get('uri', (primeiroErro, primeiraResposta) => {
    resolver(primeiraResposta);
  });
})
.then((primeiraResposta) => {
  return new Promise((resolver, rejeitar) => {
    get(`uri/${primeiraResposta.valor}`,
(segundoErro, segundaResposta) => {
      resolver(segundaResposta);
    });
  })
})
.then((segundaResposta) => {
  console.log(segundaResposta);
});
```

Async/Await é mais lindo ainda

```
return new Promise((resolver, rejeitar) => {
  get('uri', (primeiroErro, primeiraResposta) => {
    resolver(primeiraResposta);
  });
})
  .then((primeiraResposta) => {
    return new Promise((resolver, rejeitar) => {
      get(`uri/${primeiraResposta.valor}`,
(segundoErro, segundaResposta) => {
        resolver(segundaResposta);
      });
    })
  })
  .then((segundaResposta) => {
    console.log(segundaResposta);
  });
```

VS

```
funcaoAssincrona = async () => {
  const primeiraResposta = await get('uri');
  const segundaResposta = await
get(`uri/${primeiraResposta.valor}`);
  console.log(segundaResposta);
}

funcaoAssincrona();
```

Faça alguma coisa com os erros

```
try {
  algoQuePodeDarErro();
} catch (erro) {
   console.log(erro);
}
```

VS

```
try {
  algoQuePodeDarErro();
} catch (erro) {
   notificarUsuarios(erro);
}
```

Evite Efeitos Colaterais

```
let numero = 1
incrementarNumero = () => numero
+= 1;
incrementarNumero();
```

VS

```
incrementarNumero =
numero => numero + 1;
const numero = 1;
const novoNumero =
incrementarNumero(numero);
```

LEMBRE-SE DE

TESTAR

Frameworks

Jest

rápido em projetos grandes
recomendado pelo Facebook
pronto para usar

Jasmine

sugerido pelo Angular
muitas referências disponíveis
pronto para usar

Mocha

biblioteca mais usada
mais flexível
difícil de configurar
precisa de outras bibliotecas

REFATORE

Obrigada!

DÚVIDAS?

camartins@thoughtworks.com
tonofrio@thoughtworks.com

REFERÊNCIAS

. Clean Code - Robert Martin
. Refactoring - Martin Fowler com Kent Beck
. Clean Code Javascript - https://github.com/ryanmcdermott/clean-code-javascript
. JavaScript Style Guides - https://codeburst.io/5-javascript-style-guides-including-airbnb-github-google-88cbc6b2b7aa
. An Overview of JavaScript Testing in 2018 - https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2018-f68950900bc3