JAVASCRIPT

FrontEnd

BackEnd

OS FRAMEWORKS QUE CONECTA FRONTEND E BACKEND

Laryssa Santos

Frameworks e Ferramentas JavaScript

Desenvolvimento Web

JavaScript é uma das linguagens mais populares para criação de aplicações web completas, seja no frontend, backend ou até mesmo para manipulação de bancos de dados. Este eBook apresenta os principais frameworks de frontend, backend e bancos de dados com exemplos práticos para você começar.

Frameworks
FrontEnd

REACT

React é uma biblioteca (não exatamente um framework) focada em criar interfaces de usuário baseadas em componentes.

Por que usar?

- Modularidade: Crie e reutilize componentes.
- Comunidade enorme e vasto ecossistema.

Exemplo:

Um contador simples:

Laryssa.js

```jsx
import React, { useState } from 'react';

function Contador() {
  const [contador, setContador] = useState(0);

  return (
    <div>
      <h1>Contador: {contador}</h1>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}

export default Contador;
```

Vue.js

Vue é um framework progressivo, ideal para projetos pequenos e escalável para projetos maiores.

Por que usar?

- Curva de aprendizado baixa.
- Flexibilidade para integrar em projetos existentes.

Exemplo:

Um componente Vue com dois-way data binding:

Laryssa.js

```vue
<template>
  <div>
    <input v-model="mensagem" placeholder="Escreva algo" />
    <p>Você digitou: {{ mensagem }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return { mensagem: '' };
  },
};
</script>
```

Svelte

Svelte é um framework revolucionário que compila seu código em JavaScript puro.

Por que usar?

- Rápido e eficiente, sem necessidade de um runtime.
- Código mais enxuto e legível.

Exemplo:

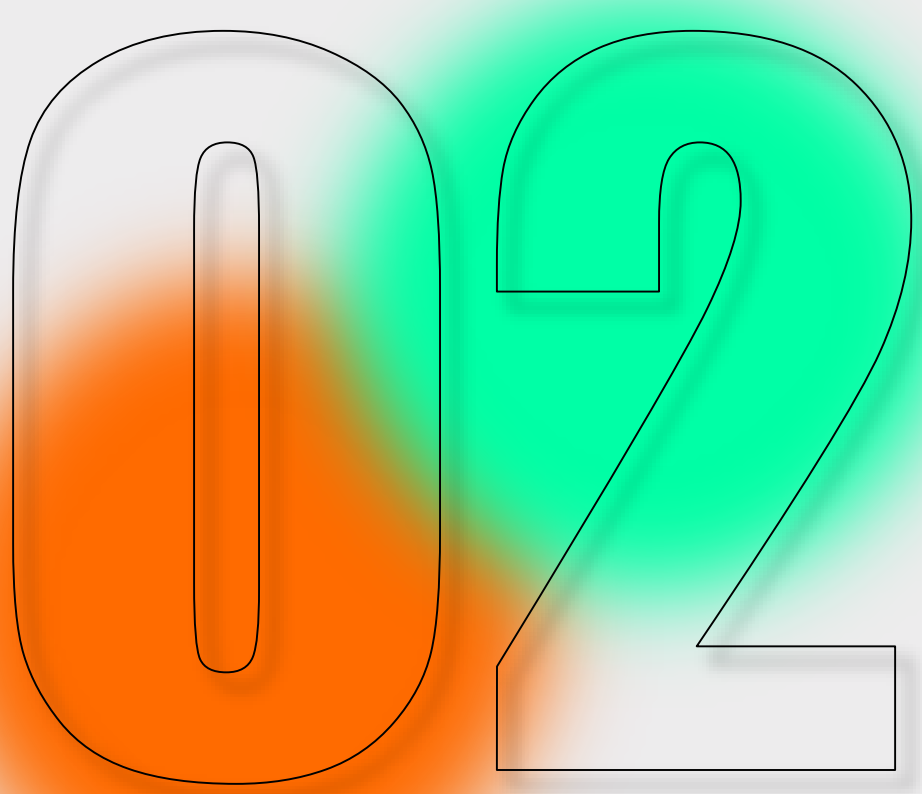
Um contador com Svelte:

Laryssa.js

```
<script>
  let contador = 0;
</script>

<h1>Contador: {contador}</h1>
<button on:click={() => contador++}>Incrementar</button>
```

Frameworks
BackEnd



Express.js

Express é um framework minimalista para Node.js, amplamente usado para criar APIs.

Por que usar?

- Simples, leve e flexível.
- Ótimo para criar APIs RESTful.

Exemplo:

Servidor simples com Express:

Laryssa.js

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Olá, mundo!');
});

app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

NestJS

Nest é um framework completo baseado em TypeScript, projetado para aplicações escaláveis.

Por que usar?

- Suporte nativo para microsserviços.
- Arquitetura modular.

Exemplo:

Controlador simples com NestJS:

Laryssa.js

```ts
import { Controller, Get } from '@nestjs/common';

@Controller('saudacao')
export class SaudacaoController {
  @Get()
  getSaudacao(): string {
    return 'Olá, mundo!';
  }
}
```

Koa.js

Koa é um framework moderno que facilita o uso de middlewares com async/await.

Por que usar?

- Flexibilidade e controle.
- Ideal para projetos customizados.

Exemplo:

Servidor básico com Koa:

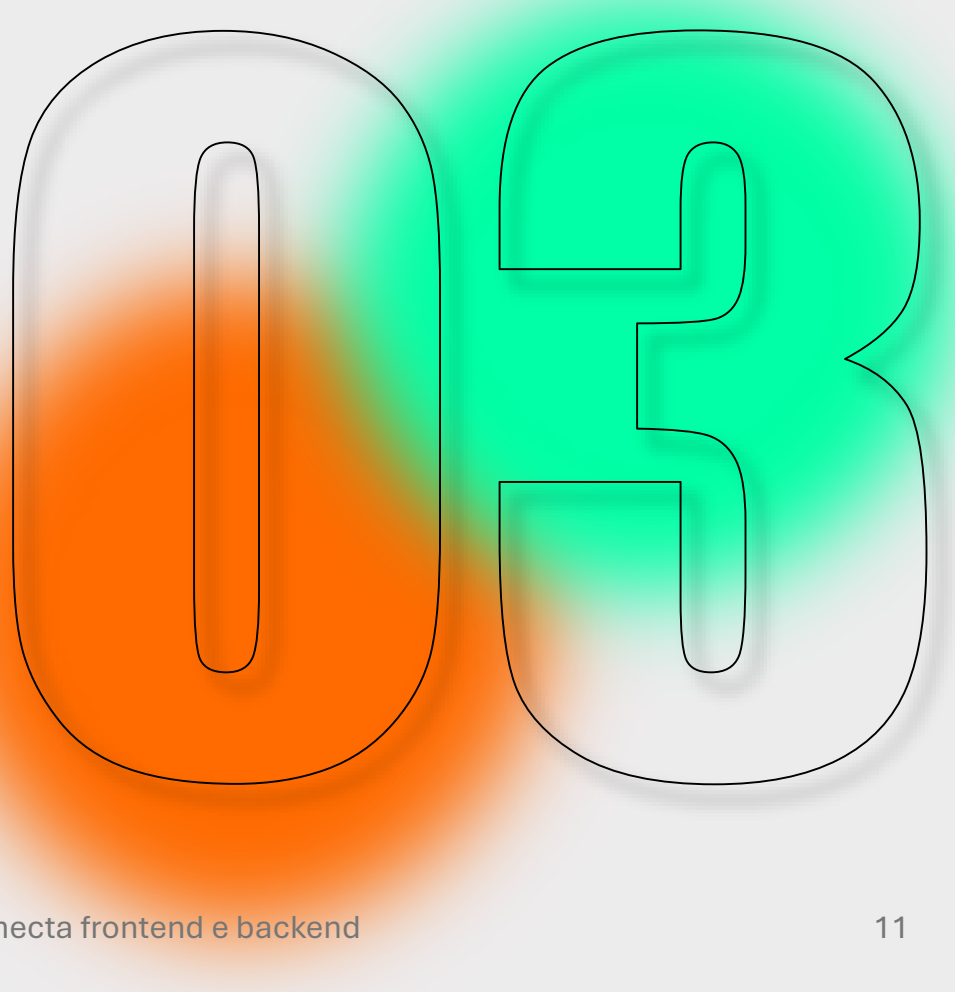
Laryssa.js

```
const Koa = require('koa');
const app = new Koa();

app.use(async ctx => {
  ctx.body = 'Olá, mundo!';
});

app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

Banco de dados



MongoDB

Banco de dados NoSQL orientado a documentos.

Por que usar?

- Flexível para dados não estruturados.
- Integrado facilmente com Node.js via Mongoose.

Exemplo:

Conectando ao MongoDB com Mongoose:

Laryssa.js

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/teste', { useNewUrlParser: true });
const Usuario = mongoose.model('Usuario', { nome: String });
const novoUsuario = new Usuario({ nome: 'Danilo' });
novoUsuario.save().then(() ⇒ console.log('Usuário salvo!'));
```

PostgreSQL

Banco de dados relacional robusto e escalável.

Por que usar?

- Suporte a transações complexas.
- Integrado com bibliotecas como pg.

Exemplo:

Consultando o PostgreSQL:

Laryssa.js

```
const { Client } = require('pg');

const client = new Client({ connectionString: 'postgresql://localhost:5432/teste' });
client.connect();

client.query('SELECT * FROM usuarios', (err, res) => {
  console.log(res.rows);
  client.end();
});
```

Redis

Banco de dados em memória usado para cache e sessões.

Por que usar?

- Rápido e eficiente.
- Ideal para dados temporários.

Exemplo:

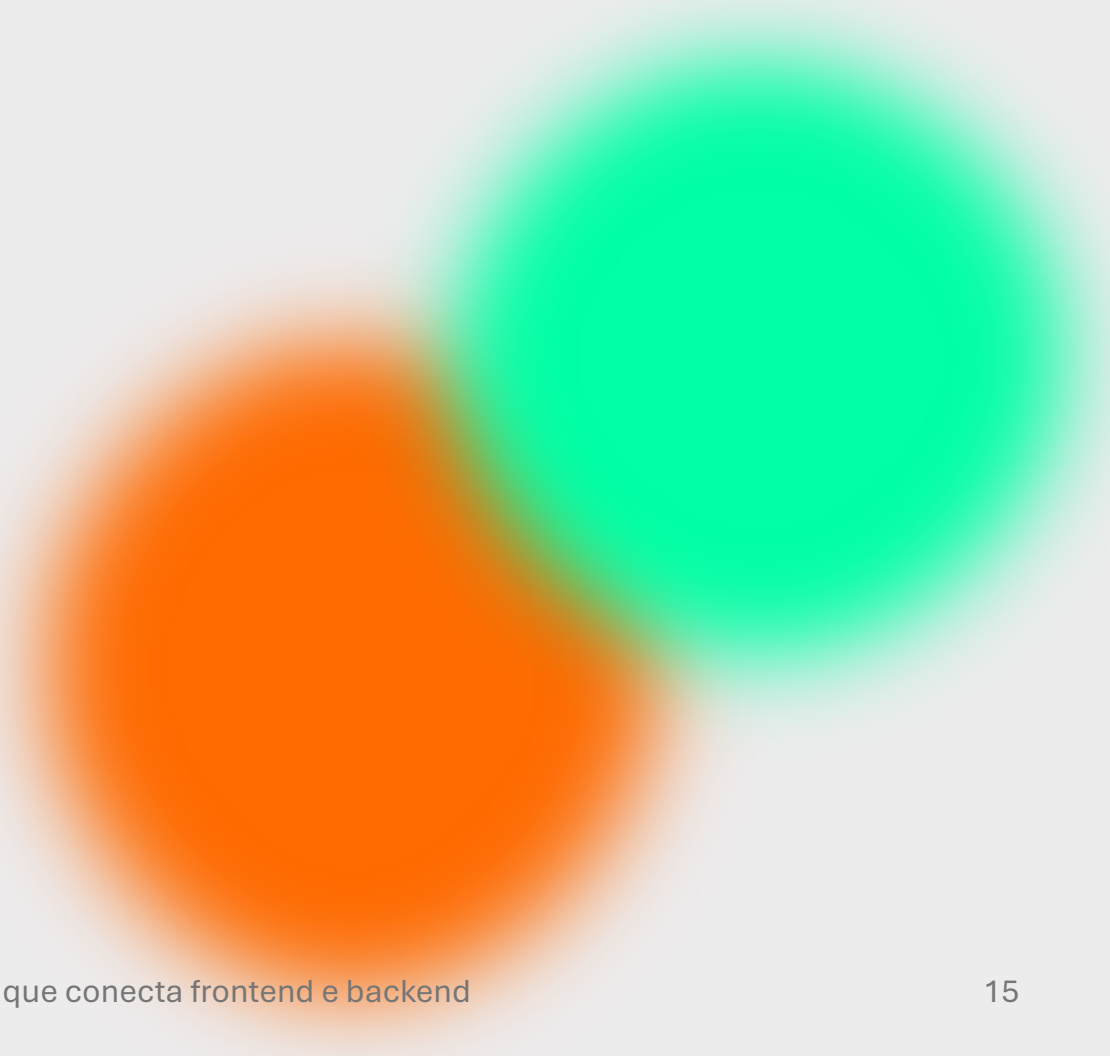
Salvando e recuperando dados no Redis:

Laryssa.js

```
const redis = require('redis');
const client = redis.createClient();

client.set('chave', 'valor', redis.print);
client.get('chave', (err, reply) => {
  console.log(reply);
});
```

CONCLUSÃO



Obrigada pela sua atenção e ter chegado até aqui

Esse Ebook foi gerado por IA, e diagramado por humano.
O passo a passo se encontra no meu Github

•

Esse conteúdo foi gerado com fins didáticos de aprendizado e para por em pratica o que eu aprendi durante o curso.

https://github.com/Laryssaavlis/ebook-create-from-ia