Estudos Node.js

▼ O que é?

- Node é uma plataforma para execução de JS no lado do servidor
- Usa o motor V8 de browser
- Pega o código JS e compila ele em C++, aumentando a rapidez do código
- ▼ Baseado no contexto
  - Lado usuário (utiliza DOM): eventos de tempo, eventos de mouse, tudo que afeta o usuário diretamente
  - Lado servidor: manipulação de arquivos, manipulação de tempo
- npm: gerenciador de pacotes do node

▼ Conceitos básicos HTTP

- Request: quando solicita uma informação
- Response: devolve uma resposta para a solicitação (não necessariamente a informação)

▼ NPM X Yarn

- ▼ NPM

- Node Package Manager
- Gerenciador de pacotes
- Biblioteca ou repositório público onde módulos de terceiros são publicados e reutilizados

▼ Yarn

- “Concorrente” do NPM
- Mais rápido e instala pacotes na ordem correta
- Deve ser utilizado usando o padrão:

```
yarn add <nome do módulo>
```

▼ Conexão bloqueante X Conexão não bloqueante

- Blocking X Non Blocking
- No JS há o single thread: o código roda dentro de um único processo
- Outras linguagens utilizam mais de uma thread, podem acontecer várias requisições ao mesmo tempo, com processos multithread
- **JS por padrão é single thread**
- O node possui o **Event Loop,** que **permite fazer várias coisas ao mesmo tempo, mas ainda usando uma única thread**
- Conexão bloqueante: faz uma ação, e enquanto ela não termina não dá pra fazer a próxima
- Conexão não bloqueante: utiliza a mesma trilha para executar várias coisas ao mesmo tempo de forma concorrente (Node permite)

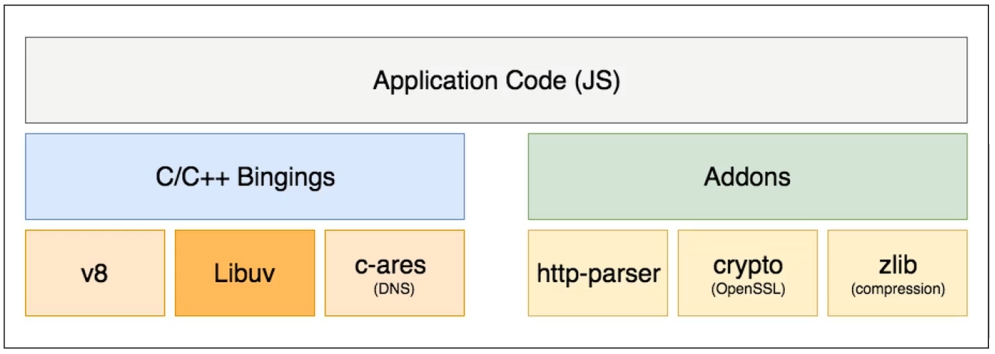
▼ Concorrência

- É usar o **mesmo tempo para fazer várias coisas**
- Event Loop: com uma única thread consegue fazer várias coisas
- Usando Event Loop podemos fazer várias coisas ao mesmo tempo se estivermos usando métodos não bloqueantes
- **Concorrência no JS se refere à capacidade que o Event Loop possui de executar funções em paralelo**

▼ Como Node.js funciona internamente?

- Ambiente de execução **sob o V8 JS Engine**
- Tiraram o V8 do Chrome e o Node roda dentro do V8
- O node.js utiliza programação dirigida ao evento: os eventos acontecem e o **Node tá sempre de olho, e vai reagindo disparando callbacks;**
- Node utiliza **métodos não bloqueantes de I/O**

▼ Estrutura interna do Node



▼ I/O: Entrada e saída

- Input: informações de entrada, o usuário envia as informações
- Output: informações de saída, com base no input devolve uma resposta
- Exemplos de I/O: File System, Network, DNS;
- **LIBUV: traduz uma linguagem de alto nível para uma linguagem de máquina**
- APIs do Node são assíncronas **(SEMPRE USAR APIS ASSÍNCRONAS)**

▼ NodeJS Single Thread

- O que é processo? É um programa sendo executado
- Processo pode executar múltiplas threads
- Exemplo: usar o photoshop para renderizar uma imagem pesada. O photoshop pode rodar em uma thread e usar outra para renderizar a imagem
- **Uma thread sempre procura um processador para executar**

- Quando temos uma única thread, ela utiliza o mesmo espaço de memória, porém várias threads podem alocar vários espaços diferentes de memória, consumindo processamento
- Usando uma única thread, o processamento é otimizado, já que **apenas um espaço de memória é utilizado por todos os processos**

▼ Multiprocessamento

- Processadores com mais de um núcleo: octacore, quadcore;
- **Núcleos são miniprocessadores dentro de um processador físico** (vários computadores dentro de um só)
- Permite executar vários processos ao mesmo tempo

▼ Multithreading

- Um único processo que dispara várias threads ao mesmo tempo
- Permite executar operações pesadas mais rapidamente

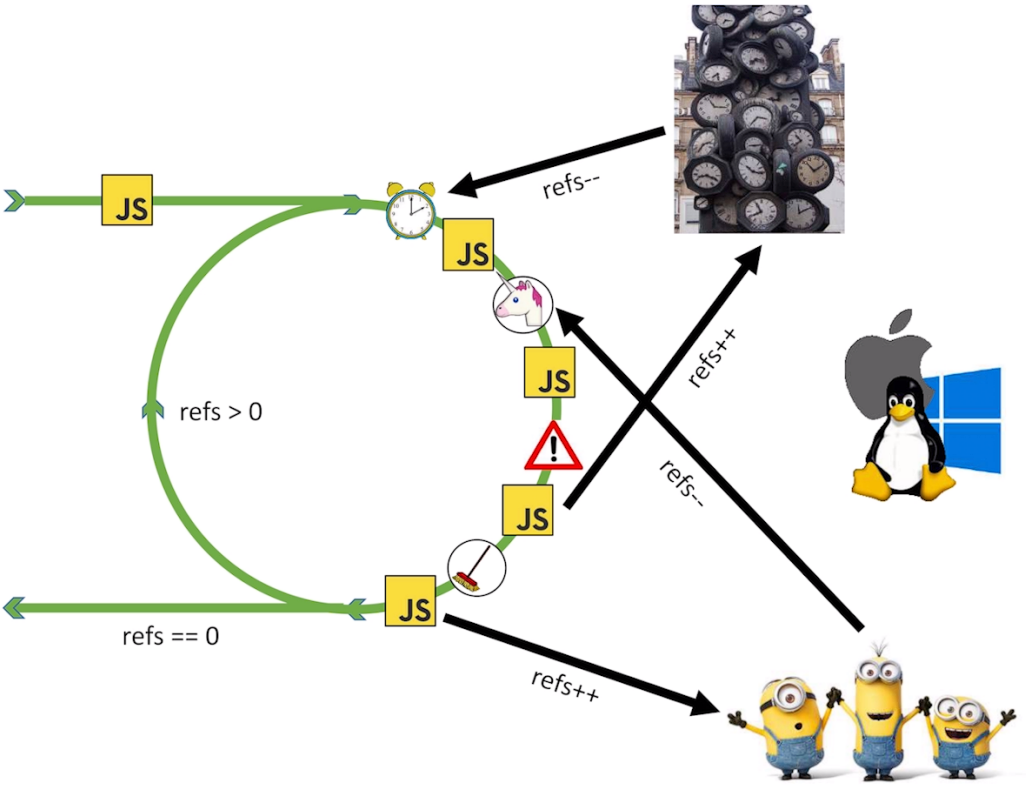
▼ Threading pool

- São 4 threads, cada uma utilizando um processador ou núcleo de processamento para processar tarefas parrudas
- Node utiliza uma única thread para processar, melhor utilizando a memória, porém se tiver um processamento muito pesado, o **Node usa a libuv que pega as informações por meio do EventLoop, e classifica se são muito pesadas para ser processada, caso sim, pega a informação e usa 4 núcleos de processamento** para processar ao mesmo tempo, possibilitando um ganho de performance
- **Quem faz a conexão entre a libuv com o processamento é o SO**
- Cada sistema tem seu agendador de tarefas chamado scheduler, para enviar a notificação que precisa ser processado alguma coisa

Resumo: utiliza as 4 threads para aumentar o desempenho e acontecer ao mesmo tempo, rodando o script em uma thread e processando informações mais pesadas em outras threads que assim que terminarem irão ser notificadas ao script principal

▼ Libuv - Event Loop (importante)

https://b2ml.udemy.com/course/nodejs-curso-completo/learn/lecture/16742864#questions/15484856




```
function 🦄() {

    // Put the main thread to sleep.
    // Wake up when:
    //
    //  * there are events from the kernel to process
    //  * a thread pool thread has completed an operation
    //  * the next timer goes off

    epoll_wait()        kevent()        GetQueuedCompletionStatusEx()

    // Return the collected events.

}
```

Palestra de Event Loop: https://www.youtube.com/watch?v=PNa9OMajw9w

▼ Ciclos de vida de uma aplicação NodeJS

▼ Aplicações Web

- Express: módulo externo que permite fazer o Node ouvir rotas, e esse caminho vai processar o código JS e devolve uma resposta no browser. Exemplo: aplicação front-end que consome uma rota
- Micro serviços: rotinas que fornecem informações pela web
- REST API: Onde se concentra a maior parte de aplicações Node atualmente. Consultamos alguma coisa e fornecemos isso para terceiros;

▼ Interface de Linha de Comando

- CLI: comand line interface (terminal)
- Tools (NPM, WEBPACK):
- Exemplos: Backup, Sync: software de automação linkado na nuvem onde toda vez que alguém joga algum arquivo compartilhado lá, quando vc baixar quer que pegue o arquivo e faça alguma manipulação com ele

▼ Iot

- Conexão com dispositivos via porta UDP;
- Bibliotecas para Hardware como Johnny Five;
- Sensores;

▼ Aplicação Desktop

▼ Chamadas assíncronas

- Em Node as APIs são assíncronas, sendo necessário ter certeza de que foram executadas;
- Callbacks Functions (funções de retorno, que falam quando a função terminou);
- Promises: permite que trabalhe com processamento assíncrono;
- Async/Await: forma elegante de escrever promisses;

▼ Express

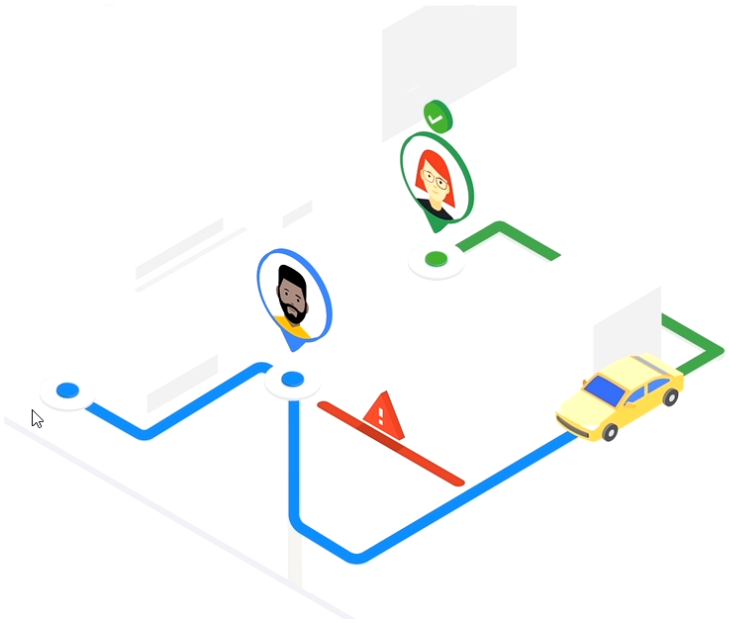
▼ Infos básicas

- Framework **rápido, flexível e minimalista**
- Rápido: muito utilizado em aplicações escaláveis, por isso tem que ser rápido
- Flexível: pode utilizar express para apps que trabalham com layouts ou rotas (infos que são consumidas por outra app)
- Minimalista: não é difícil de aprender
- Dentro do express: rotas, middleware, templates, MVC
- MVC: model, view (templates), controller
- templates: interaje o html com a programação

▼ Rotas

- São caminhos que utilizamos para chegar a um ponto final ou **EndPoint**
- Method Get e Post
  - Get: **dados aparecem na URL**
  - Post: informações aparecem junto ao corpo da mensagem no Header, **não tão fácil de acessar quanto o Get**
- Toda vez que você acessa uma url você ta acessando uma rota

▼ Imagem



▼ Middleware

▼ Tipos

▼ Funções em nível de aplicação

- Acontece enquanto uma aplicação está rodando
- Carrega uma função de middleware como app.use(express.json())

▼ Funções em nível de rota

- para manipular rotas

▼ Funções para manipular erros

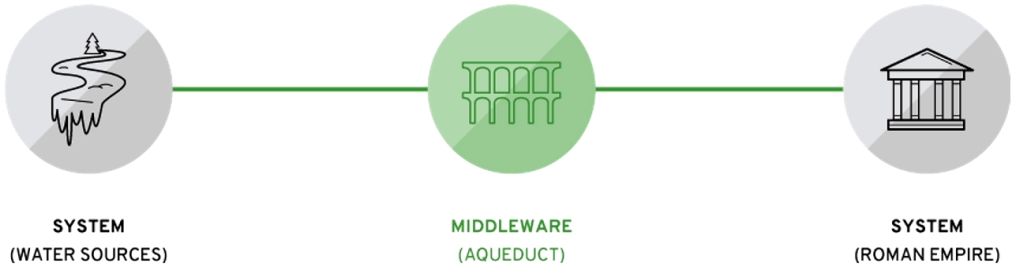
-

▼ Funções de terceiros

- Funções criadas por outras pessoas
- Exemplo: body parser

- é uma camada que será aplicada entre a requisição do usuário e a aplicação
- App.use: “utilize essa função de middleware nessa requisição do usuário”
- funções middleware são usadas para interferir na requisição do usuário, seja para autenticar, validar, etc

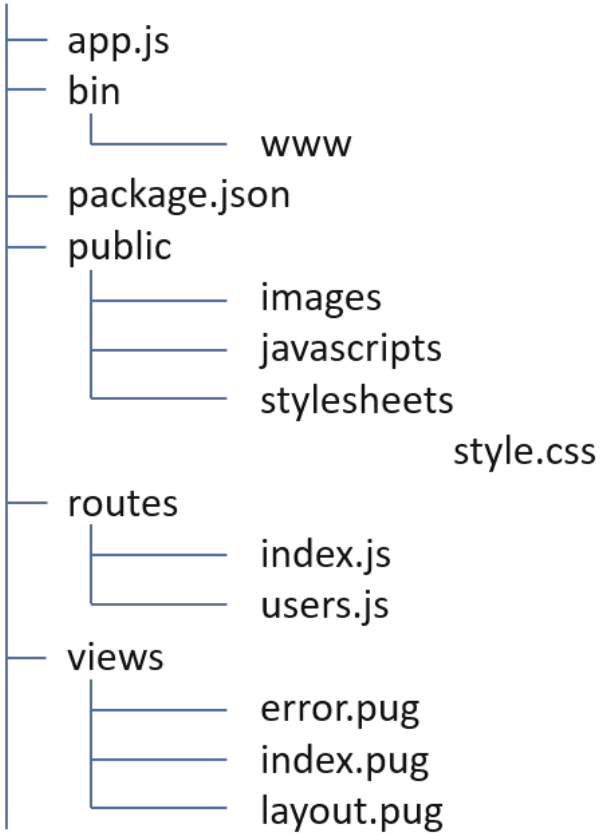
▼ Imagem



▼ Templates

▼ Templates pug: mais recomendado

▼ Express gera uma estrutura de pastas



▼ MongoDB

- Usar $gt para “maior que” e $lt para “!menor que”, durante as pesquisas

▼ Para criar um data base

```
use <nome do DB>
```

▼ Para inserir um dado dentro de uma collection

```
db.<nomeDaCollection>.insertOne({<objetoDoItem>})
```

- Para inserir vários dados dentro de uma collection de uma só vez

```
db.<nomeDaCollection>.insertMany([{<infosItem1>},{<infos
```

- Para fazer pesquisas dentro do db
  - Procurar dados gerais

```
db.clientes.find(*)
```

  - Procurar por id ou outra informação

```
db.clientes.find({<nomeDaInfo>:<valorDaInfo>})
```

  - Procurar usando cláusula OR

```
db.produtos.find({$or: [{<nomeDodado>: {<valorDoDado>
```

  - Procurar usando claúsula AND

```
db.produtos.find({v{<nomeDodado>: {<valorDoDado>}, {<
```

- Para fazer atualizações
  - Update

```
db.produtos.updateOne({<qualProduto>, {$set: <oQueMud
```

```
db.produtos.updateMany({value: {$lt: 5000}}, {$set:{v
```

  - Replace

```
db.produtos.replaceOne({_id:4}, {name:"LG Curve Monit
```

  - Delete

```
db.produtos.replaceOne({_id:4}, {name:"LG Curve Monit
```

```
//TODOS OS DADOS: ATENÇÃO
db.produtos.deleteMany{}
```

```
//APAGAR COM BASE NUMA CONDIÇÃO
db.produtos.deleteMany({value: 999})
```

▼ REST API

- Representation State Transfer: transferência representacional de estado
- Controla as rotas e os métodos para determinar o que o usuário deseja fazer

▼ Exemplo:

  - Um aeroporto possui diversas zonas de chegada: para taxis, aplicativo(uber, 99), onibus, etc. Cada tipo de veículo possui sua zona e podem haver benefícios, por exemplo: ”ir de uber até o aeroporto dá direito a um voucher para um café”

- Eu tenho uma informação, e quero transferir ela para você
- Utiliza conceitos de rotas e métodos HTTP
- Ao trabalhar com formulários em HTML, há uso do Get e Post que são formas de chegar ao destino
- Na REST API, há outro métodos, um para cada tipo de ação como consulta, exclusão, etc.

▼ Sistema CRUD e link com métodos REST

  - CREATE: POST
  - READ: GET
  - UPDATE: PUT/PATCH
  - DELETE: DELETE

▼ Métodos

  - GET: fazer uma consulta
  - POST: fazer um cadastro

- PUT: alteração de dados de todos os registros
- PATCH: alteração de dados de um único registro
- DELETE: exclusão de dados
- entre outros menos utilizados

▼ Imagem