



UNIVERSIDADE  
FEDERAL DO CEARÁ  
CAMPUS QUIXADÁ

# Fundamentos de NodeJS

QXD0020 - Desenvolvimento de Software para Web

Prof. Bruno Góis Mateus ([brunomateus@ufc.br](mailto:brunomateus@ufc.br))

# Agenda

- Introdução
- Criando um projeto Node
- Criando um servidor web com Node
- Arquitetura MVC

# Introdução



# Introdução

## Node

- É uma cross-platform runtime de código aberto que permite que desenvolvedores criem aplicações server-side em JS
- Executada “diretamente” no sistema operacional, fora do contexto do navegador
- Prover suporte a API mais tradicionais dos sistemas operacionais
  - Ex: HTTP, FileSystem

# Introdução

## História

- Enquanto a web tem 30 anos, JavaScript 26, Node tem apenas 12 anos
- Antes do sucesso do Node, a Netscape havia investido no LiveWire, o ambiente capaz de criar páginas web dinâmicas usando JavaScript no server-side, no entanto não obteve sucesso
- Aplicações server-side com JavaScript se popularizam a partir da introdução do Node.js
  - Fator decisivo: Timing
  - JavaScript passou ser utilizado em aplicações de maior porte graças a Web 2.0. Ex: Flickr, Gmail, etc.
  - Engine JavaScript melhoraram consideravelmente devido a competição entre navegadores
- Node usa a V8 ou Chrome V8, uma engine open-source JavaScript do Projeto Chromium que evoluiu bastante devido a essa competição

# Introdução

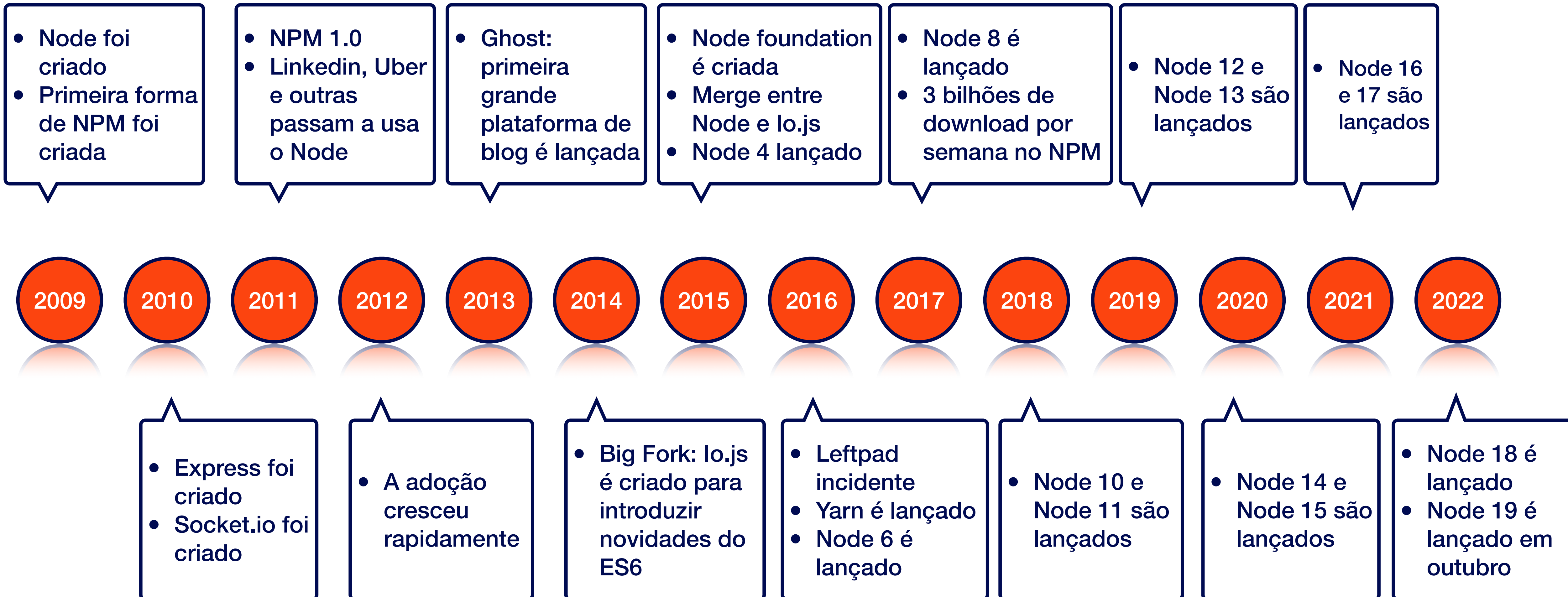
## História - v8

- Engine de alto desempenho JavaScript e WebAssembly
- Escrita em C++
- Usada no Chrome e no Node entre outros projetos
- Compila e executa código JS, gerencia a alocação de memória e realiza a desalocação de objetos não necessário (*garbage collector*)



# Introdução

## História - Timeline



# Introdução

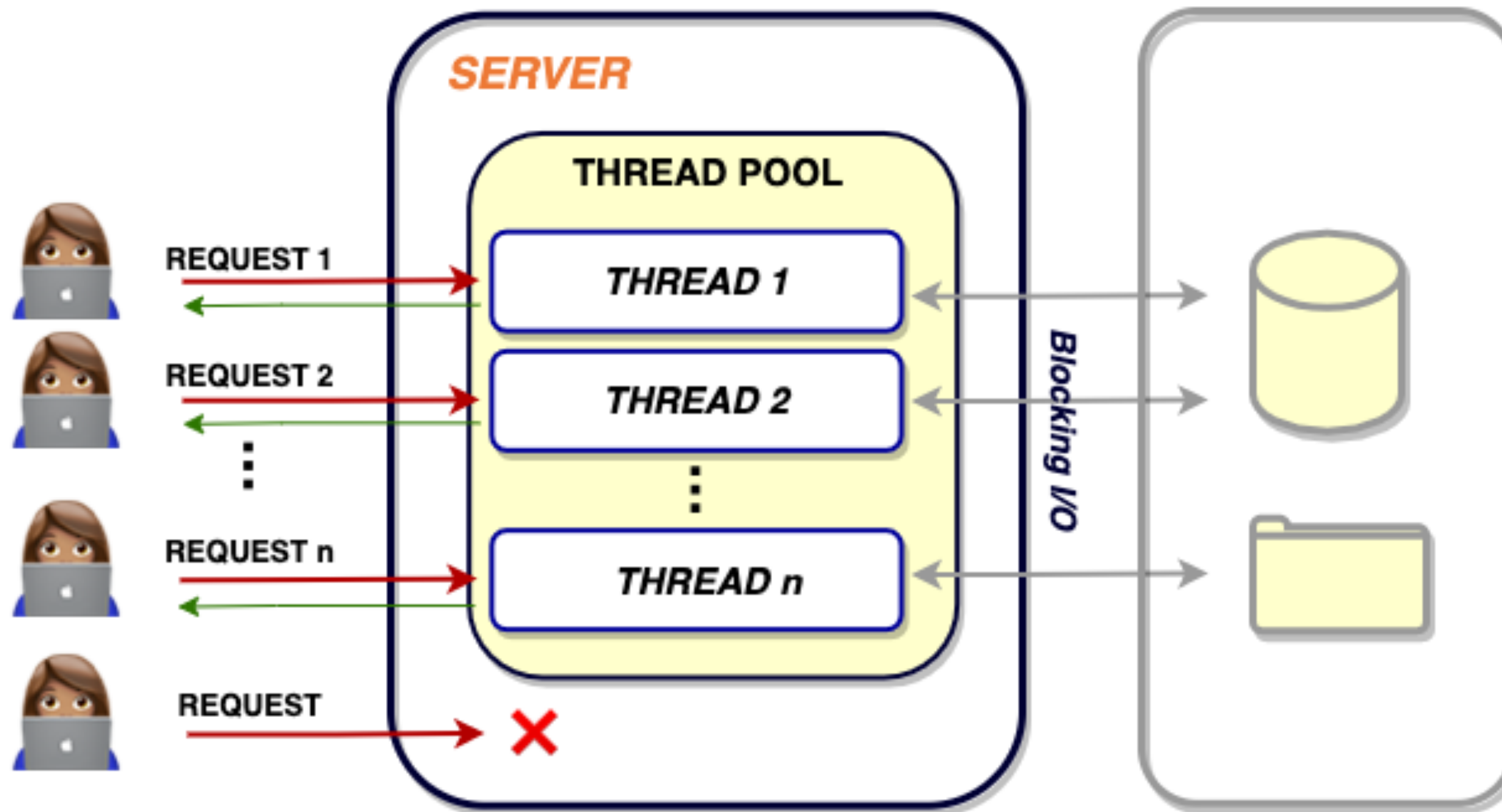
## Características

- Node.js app são executadas em um único processo
  - Não é necessária a criação de uma thread para cada requisição
- Fornece um conjunto de operações primitivas de I/O assíncronas
  - Evita que códigos de maneira geral sejam “bloqueantes”
- Escalável e mais simples de debugar, não há concorrência entre threads
- Novidades do ESMA Script podem ser usadas sem problemas já que o usuário possui o controle do ambiente de execução
  - No front-end dependemos dos navegadores



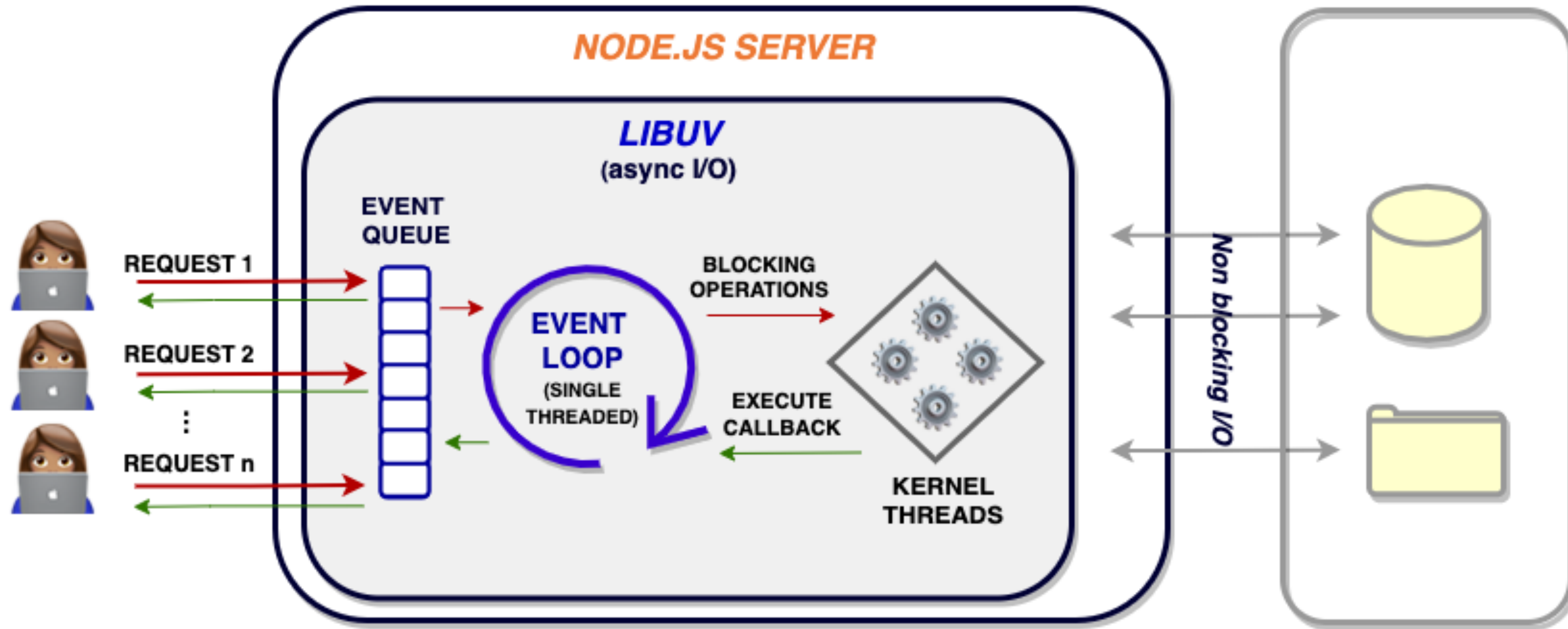
# Introdução

## Características



# Introdução

## Características



# Introdução

## Vantagens

- Excelente desempenho e escalável
- Escrito em JS, familiar para desenvolvedores Web
- Grande comunidade de usuários e desenvolvedores
- O gerenciador de pacote do Node, NPM, prover acesso a diversas bibliotecas reusáveis
  - Gerenciamento de dependências
- Portável, disponível para Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS, and NonStop OS

# Criando um projeto Node



# Criando um projeto Node

## Hello World

- O seu primeiro programa em Node

```
console.log("Olá mundo");
```

```
node app.js
```

# Criando um projeto Node

## Módulos nativos

- [assert](#)
- [buffer](#)
- [child\\_process](#)
- [console](#)
- [cluster](#)
- [crypto](#)
- [dgram](#)
- [dns](#)
- [events](#)
- [fs](#)
- [http](#)
- [http2](#)
- [https](#)
- [net](#)
- [os](#)
- [path](#)
- [perf\\_hooks](#)
- [process](#)
- [querystring](#)
- [readline](#)
- [repl](#)
- [stream](#)
- [string\\_decoder](#)
- [timers](#)
- [tls](#)
- [tty](#)
- [url](#)
- [util](#)
- [v8](#)
- [vm](#)
- [wasi](#)
- [worker](#)
- [zlib](#)

# Criando um projeto Node

## Módulos nativos

Nome	Descrição
<a href="#"><u>console</u></a>	Prover um console para debug
<a href="#"><u>events</u></a>	Prover uma API para o gerenciamento de eventos
<a href="#"><u>fs</u></a>	Prover uma API para interagir com o sistema de arquivos
<a href="#"><u>http</u></a>	Prover uma implementação HTTP cliente/servidor
<a href="#"><u>os</u></a>	Prover propriedades e métodos utilitários relacionados ao sistema operacional
<a href="#"><u>path</u></a>	Prover utilitários para trabalhar com path e diretórios
<a href="#"><u>querystring</u></a>	Prover utilitários para " <i>parsear</i> " e formatar URL de string de consulta (querystring)

# Criando um projeto Node

## Módulos nativos

Nome	Descrição
<a href="#"><u>repl</u></a>	Prover um implementação Read-Eval-Print-Loop (REPL) disponível como um versão standalone, mas que também pode ser adicionada a outras aplicações
<a href="#"><u>timers</u></a>	Prover funções para agendar execuções de funções em um período futuro
<a href="#"><u>url</u></a>	Prover utilitários para resolução e “parseamento” de URL



# Criando um projeto Node

## NPM

- Node Package Manager - Gerenciador de pacotes do Node
- Inicialmente era uma maneira de fazer download e gerenciar as dependências
- Atualmente é também utilizado em projetos front-end
- Possui mais de 1.3 milhões de pacotes disponíveis
  - Maior repositório de software do mundo



# Criando um projeto Node

- Para iniciar um projeto **node**, é necessário criar um arquivo chamado **package.json**
  - Lista todas as dependências do projeto e suas versões
  - Torna o processo de build reproduzível e portanto mais fácil de compartilhar com outros desenvolvedores
  - Deve conter pelo menos o atributo **name** e **version**
- A maneira mais simples de criar esse arquivo é usando o comando:
  - `$ npm init --yes`

# Criando um projeto Node

```
{
  "name": "my package",
  "description": "",
  "version": "1.0.0",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/monatheoctocat/my_package.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/monatheoctocat/my_package/issues"
  },
  "homepage": "https://github.com/monatheoctocat/my_package"
}
```

Nome do diretório

Informação contida no README ou string vazia

Sempre 1.0.0

Script de test vazio

No caso de se um repositório git

Sempre vazio

Sempre vazio

Por padrão ISC

Caso hospedado no GitHub

Caso hospedado no GitHub

# Criando um projeto Node

## NPM e suas funções

- Instalar e atualizar dependências
  - `$ npm install` ou `$ npm install <package-name>`
  - `$ npm update` ou `$ npm update <package-name>`
- Versionamento
- Execução de tarefas
  - Ex: Executar em produção, testar ...

# Criando um projeto Node

## Pacotes populares



Express



Socket.io



Fastify



Rxjs



Lodash



Ramda



Jest



ESLint



Nodemon



Dotenv



Yargs



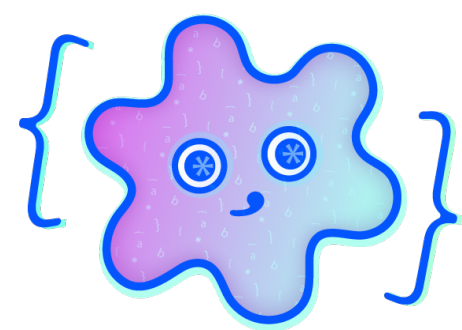
Passport



Nodemailer



Mongoose

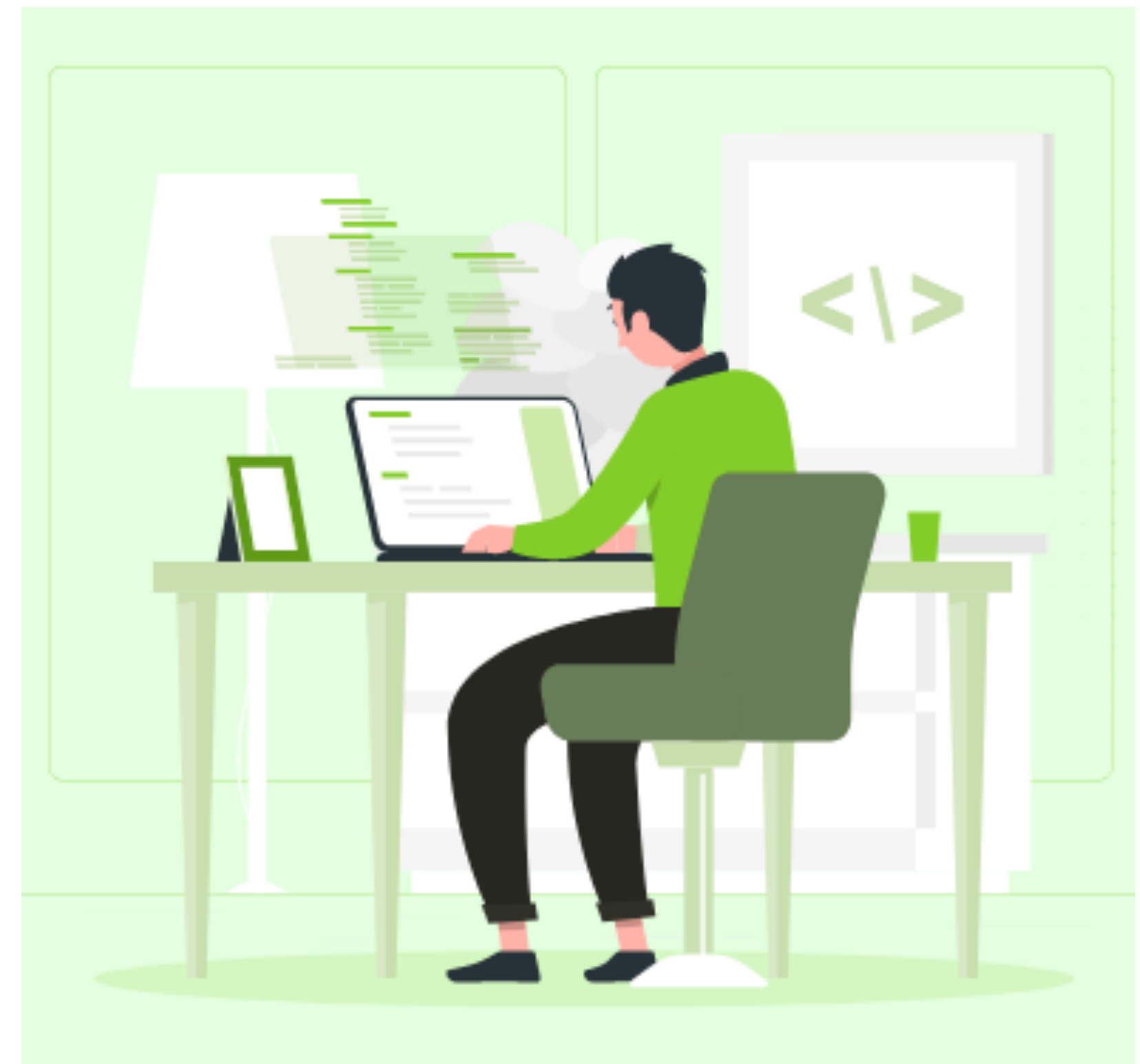


glob



[Lista completa contendo 40 pacotes](#)

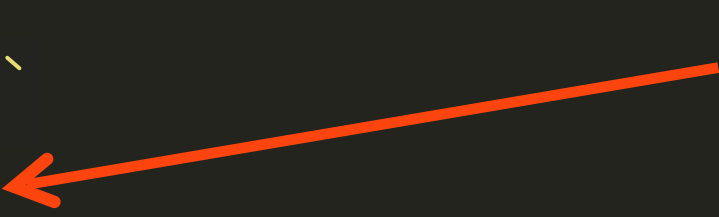
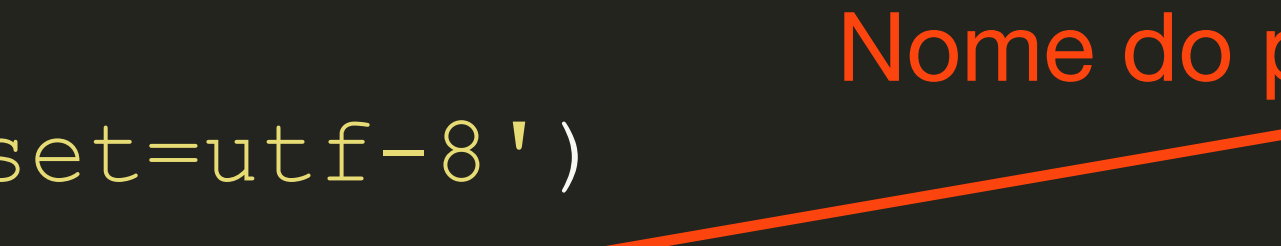
# Criando um servidor web com Node



# Enviando parâmetros na requisição

## Obtendo parâmetros de uma requisição GET

- Precisamos utilizar acessar o objeto a URL dentro do objeto que encapsula a requisição HTTP

```
const server = http.createServer(req: IncomingMessage, res: ServerResponse) => {  
  
  const baseUrl = `http://${req.headers.host}/`  
  const parsedUrl = new URL(req.url, baseUrl)   
  
  res.statusCode = 200  
  res.setHeader('Content-Type', 'text/html; charset=utf-8')  
  
  const name = parsedUrl.searchParams.get("name")   
  res.end(`<html><head></head><body> Contéudo </body></html>`)  
});
```

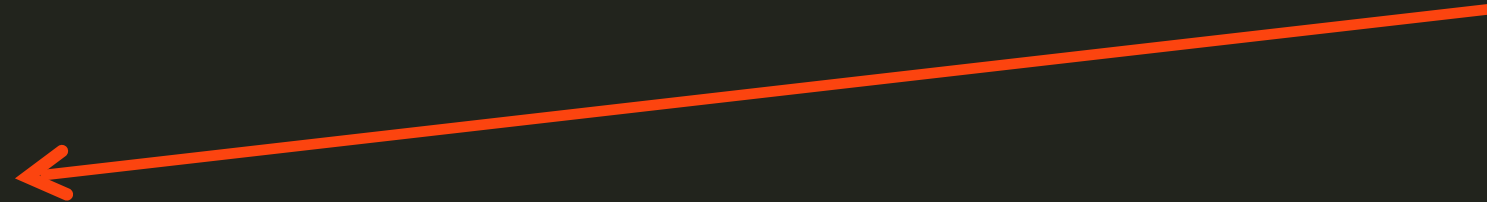
# Enviando parâmetros na requisição

## Obtendo parâmetros de uma requisição POST

- Precisamos utilizar acessar o objeto a URL dentro do objeto que encapsula a requisição HTTP

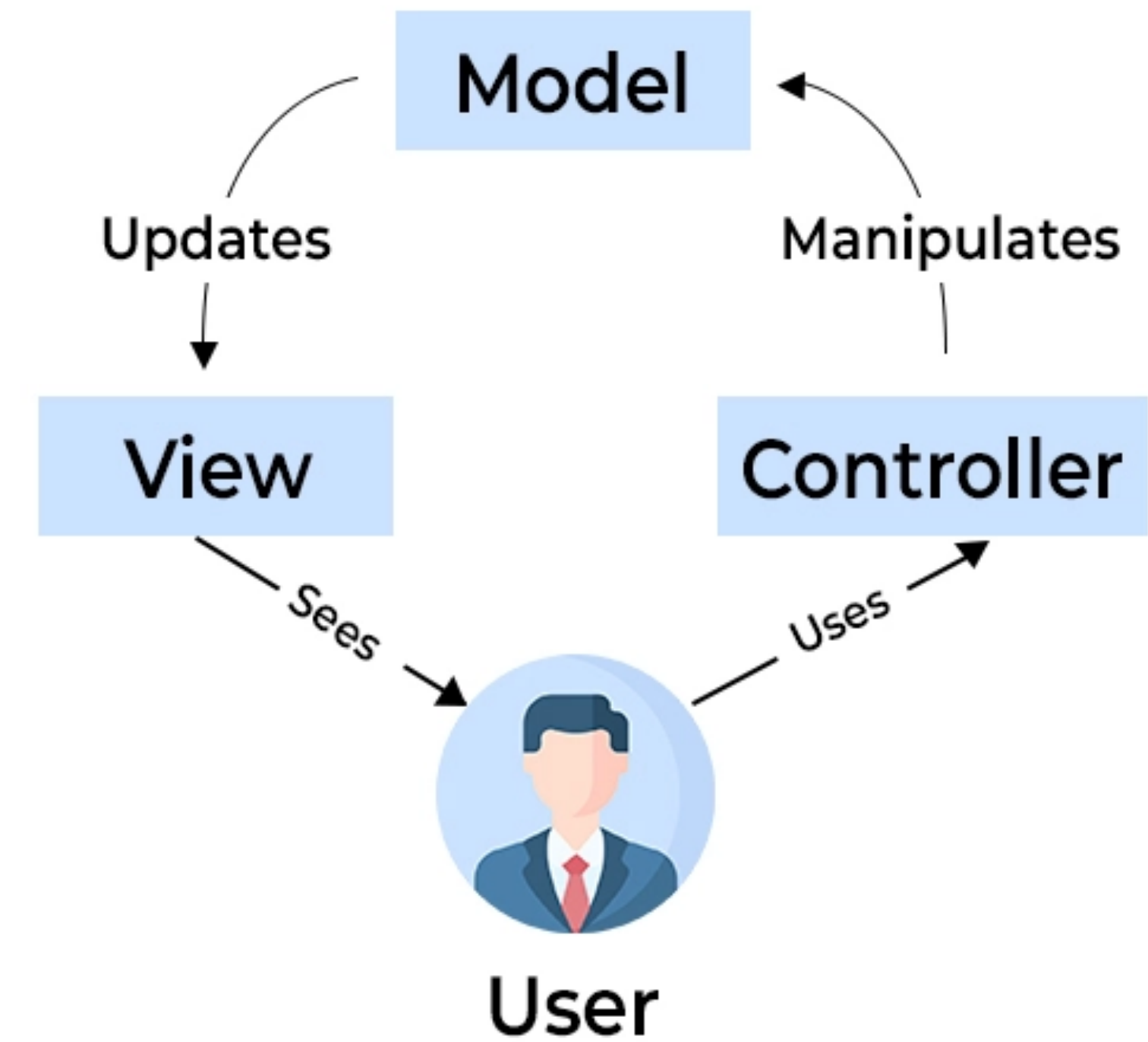
```
const server = http.createServer(async (req: IncomingMessage, res: ServerResponse) => {  
  
  const baseUrl = `http://${req.headers.host}/`  
  const parsedUrl = new URL(req.url, baseUrl)  
  
  const chunks = [];  
  for await (const chunk of req) {  
    chunks.push(chunk);  
  }  
  const data = Buffer.concat(chunks);  
  res.end(data.toString())  
});
```

Navegando na stream de dados





# Arquitetura MVC



[2]

# Arquitetura MVC

## Introdução

- A possibilidade de misturar código de uma **linguagem qualquer** com **HTML** resultava em código de **baixa manutenabilidade**
  - Muitas vezes as regras de negócio estavam juntas da lógica de visualização
- A medida que as aplicações web tornavam-se mais complexas, foi surgindo a necessidade de melhorar a organização do código dessas aplicações via **separação de conceitos**
- Um **padrão de software arquitetural** se tornou muito popular nesse contexto, o **MVC**

# Arquitetura MVC

## PHP: Exemplo de uso de estruturas de repetição

```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array (
    array ("Volvo", 22, 18),
    array ("BMW", 15, 13),
    array ("Saab", 5, 2),
    array ("Land Rover", 17, 15)
);

for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Row number $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
        echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
}
?>
</body>
</html>
```

### Row number 0

- Volvo
- 22
- 18

### Row number 1

- BMW
- 15
- 13

### Row number 2

- Saab
- 5
- 2

### Row number 3

- Land Rover
- 17
- 15

# Arquitetura MVC

## PHP: Mostrando informações armazenadas em um banco de dados

```
<!DOCTYPE html>
<html>
<body>
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<tr><td>" . $row["id"] . "</td><td>" . $row["firstname"] . " " . $row["lastname"] . "</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}
$conn->close();
?>
</body>
</html>
```

# Arquitetura MVC

## JSP: Mostrando informações armazenadas em um banco de dados

```
<!DOCTYPE html>
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
    <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
    <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
  <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
</body>
</html>
```

# Arquitetura MVC

## JSP: Mostrando informações armazenadas em um banco de dados

```
<!DOCTYPE html>
<html>
<body>
  <%
    String[] authors = request.getParameterValues("author");
    if (authors != null) {
  %>
  <%% page import = "java.sql.*" %>
  <%
    Connection conn = DriverManager.getConnection(
      "jdbc:mysql://localhost:3306/ebookshop", "myuser", "xxxx"); // <== Check!
    // Connection conn =
    // DriverManager.getConnection("jdbc:odbc:eshopODBC"); // Microsoft Access
    Statement stmt = conn.createStatement();

    String sqlStr = "SELECT * FROM books WHERE author IN (";
    sqlStr += "'" + authors[0] + "'"; // First author
    for (int i = 1; i < authors.length; ++i) {
      sqlStr += ", '" + authors[i] + "'"; // Subsequent authors need a leading commas
    }
    sqlStr += ") AND qty > 0 ORDER BY author ASC, title ASC";

    System.out.println("Query statement is " + sqlStr);
    ResultSet rset = stmt.executeQuery(sqlStr);
  %>
  <hr>
  <form method="get" action="order.jsp">
    <table border=1 cellpadding=5>
      <tr>
        <th>Order</th>
        <th>Author</th>
        <th>Title</th>
        <th>Price</th>
        <th>Qty</th>
      </tr>
  <%
    while (rset.next()) {
      int id = rset.getInt("id");
  %>
  <%
    <tr>
      <td><input type="checkbox" name="id" value="<%= id %>"></td>
      <td><%= rset.getString("author") %></td>
      <td><%= rset.getString("title") %></td>
      <td><%= rset.getInt("price") %></td>
      <td><%= rset.getInt("qty") %></td>
    </tr>
  <%
    }
  %>
  </table> ...
```

# Arquitetura MVC

## Model - View - Controller

- Padrão arquitetural que se tornou popular em meados de 1970
- Separa a representação da informação da visualização da mesma
- Divide o sistema em três partes interconectadas
  - Model
  - View
  - Controller

# Arquitetura MVC

## Controller

- Realizam a ligação entre o usuário e o sistema
- Devem aguarda por requisições HTTP
  - Aceita entradas e converte para comandos para view ou model
  - Delega as regras de negócio para modelos e serviços
- Retorna com uma resposta significativa



# Arquitetura MVC

## View

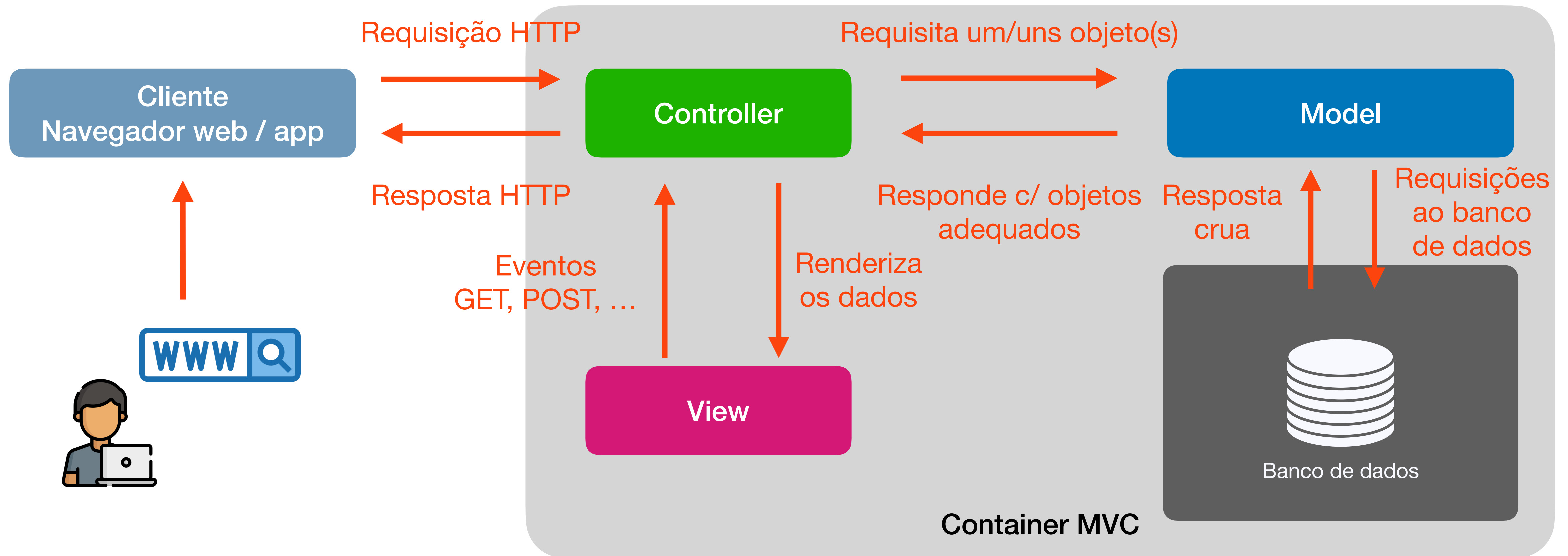
- Representação visual da nossa aplicação (GUI - Graphical User Interface)
- Mostram os dados ao usuário em forma fácil de entender baseado nas suas ações
  - Camada de interação com o usuário
- Deve refletir mudanças ocorridas nos modelos

# Arquitetura MVC

## Model

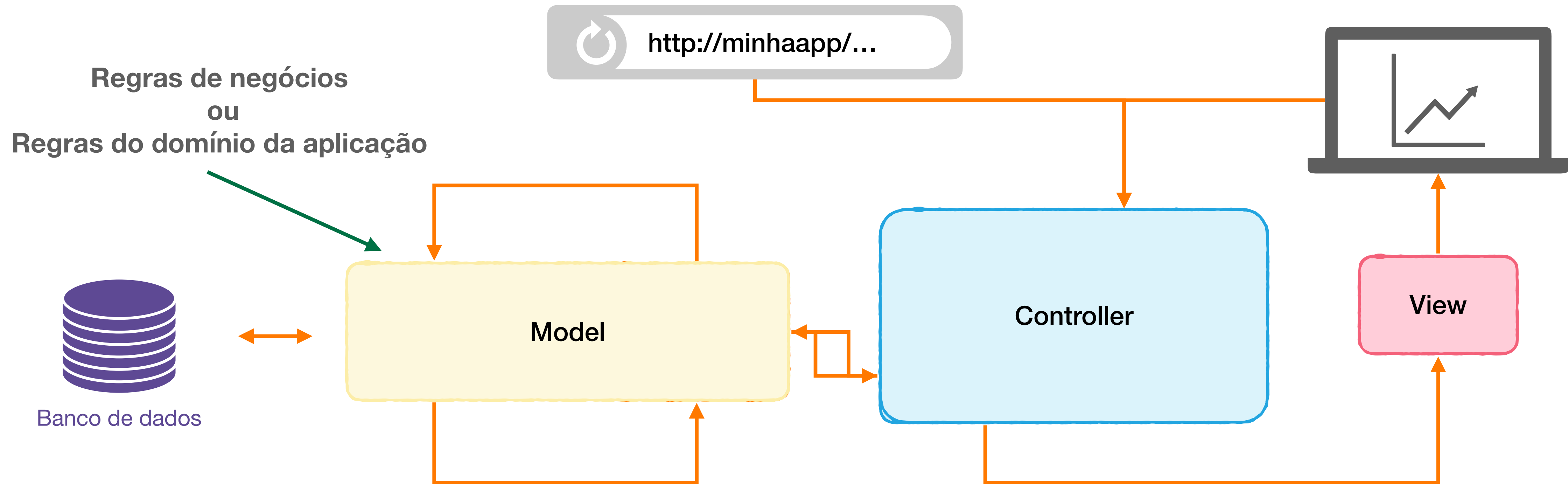
- Modelos representam o conhecimento do domínio da aplicação
- Gerencia os dados, a lógica e as regras da aplicação
- Independente da interface com o usuário
- Encapsulam os dados do banco de dados
  - Tabelas

# Arquitetura MVC



# Arquitetura MVC

## Quebrando em mais pedaços



# Referências

- <https://nodejs.dev/learn>
- <https://leanylabs.com/blog/npm-packages-for-nodejs/>
- <https://flaviocopes.com/node-core-modules/>
- <https://docs.npmjs.com/>
- <https://scoutapm.com/blog/nodejs-architecture-and-12-best-practices-for-nodejs-development>
- <https://leanylabs.com/blog/npm-packages-for-nodejs/>
- <https://ctrly.blog/nodejs-layered-architecture/>

Por hoje é só