



UNIVERSIDADE
FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Criando páginas dinâmicas

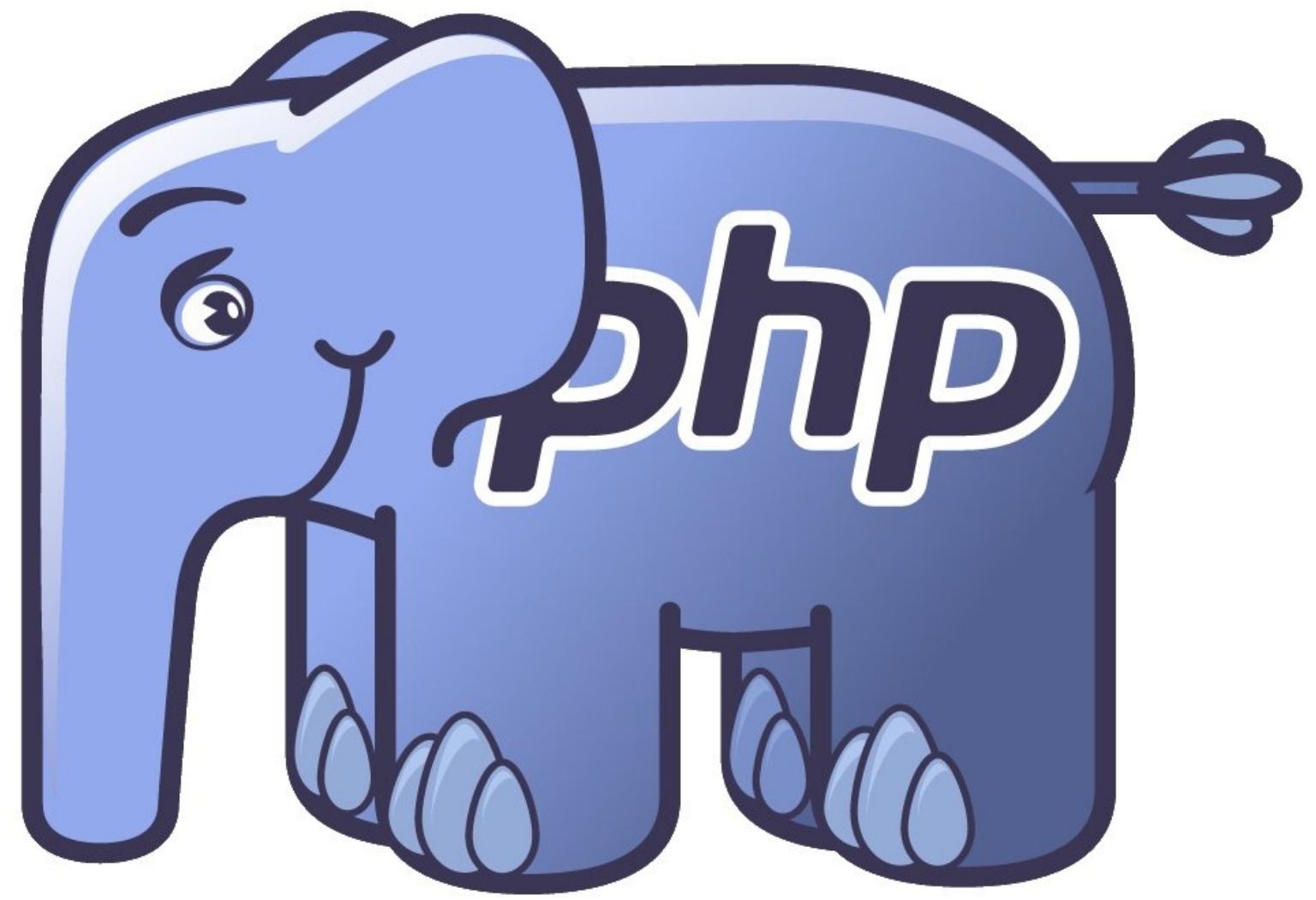
QXD0193 - Projetos de Interfaces Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Arquitetura MVC
- Fundamentos de Express
- Template Engine
- Middleware

Introdução



[1]

Introdução

- O script CGI deram início a interatividade na web
- Um mês após o lançamento do JavaScript em 1995, um canadense chamado Rasmus Lerdof liberou aquilo que ele chamou de *Personal Home Page Tools (PHP Tools)*
 - Nesta data ainda não se tratava de uma linguagem de script
 - Um conjunto de utilitários para facilitar o uso de scripts CGI
 - Todos escritos em C

“I wrote the same code over and over—basically, CGI [Common Gateway Interface] scripts written in C. I wrote code to handle forms, POST data, filtering, and other common Web things that you have to write in C when you’re writing CGI programs. It was kind of tedious and boring, so if I could reduce the amount of time I had to spend programming, maximize the output, and get to the solution quicker, then that was my goal with PHP. I put all my common stuff into a C library, hacked it into the NCSA [National Center for Computing Applications] webserver, and then added a templating system on top of it to let me easily call into it.”

Fonte: [Inventing PHP: Rasmus lerdorf](#)

Introdução

- A primeira versão consistia de 30 scripts CGI combinados em uma biblioteca escrito em C
- Um parser analisava o código HTML e substituía tag específicas por resultados da execução de funções escritas em C
- No entanto, inicialmente houve pouca adoção do PHP
- A segunda versão foi lançada em 1996
 - A essa altura o conjunto de ferramentas já estava evoluindo para se tornar uma linguagem de programação

Introdução

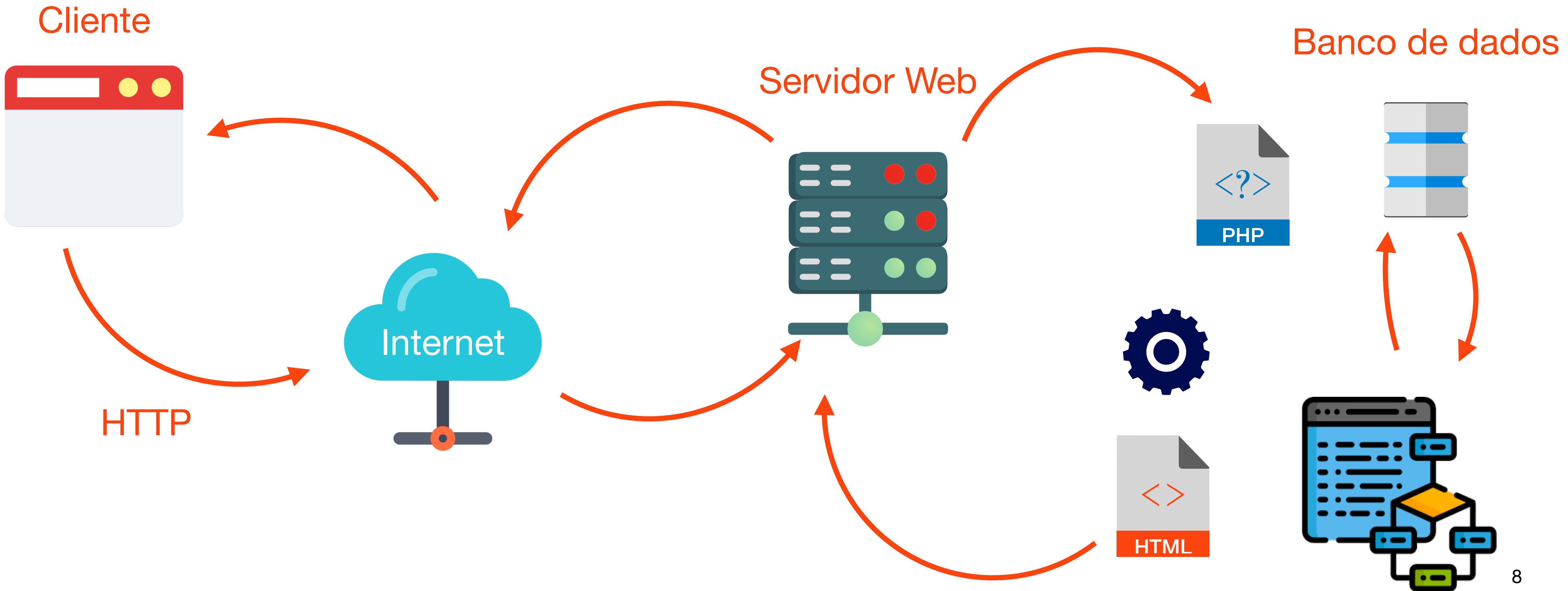
- Em 1997, Zeev Surasaki e Andi Gutmans rescreveram o parser que se tornou a base do PHP 3
 - Agora chamado de PHP: Hypertext Preprocessor
 - Foi lançada oficialmente em 1998



PHP logo [2]

Introdução

Funcionamento básico



Introdução

Exemplo de código PHP

```
<html>
  <body>
    <form action="welcome_get.php" method="GET">
      Name: <input type="text" name="name"><br>
      E-mail: <input type="text" name="email"><br>
      <input type="submit">
    </form>
  </body>
</html>
```

```
<html>
  <body>
    Welcome <?php echo $_GET[ "name" ] ; ?><br>
    Your email address is: <?php echo $_GET[ "email" ] ; ?>
  </body>
</html>
```

Introdução

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

Introdução

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>
```

Introdução

Problemas

- A possibilidade de misturar código PHP (ou outra 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

Introdução

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>
```

Introdução

JSP: Mostrando informações armazenadas em um banco de dados

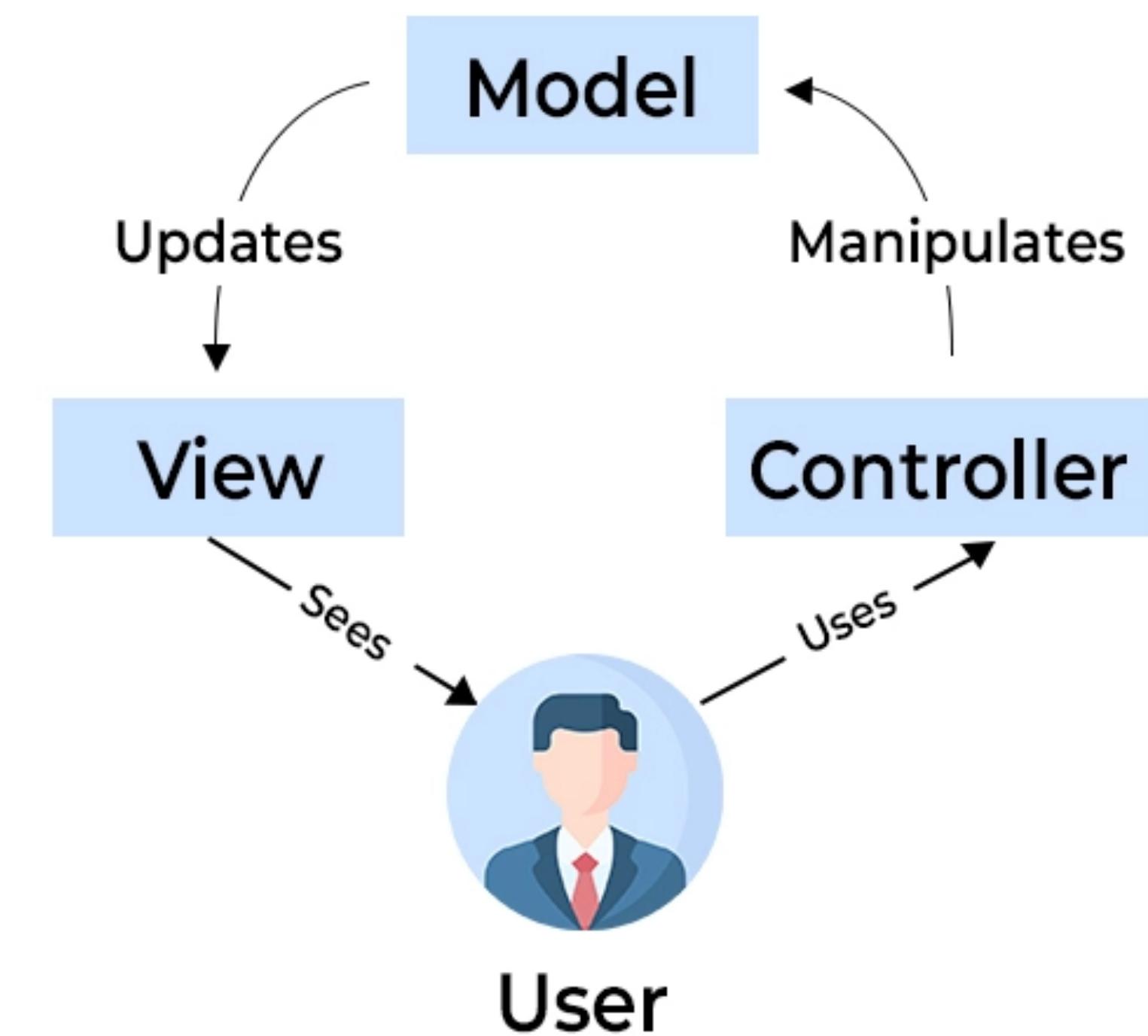
```
<!DOCTYPE html>
<html>
<body>
<%
    String[] authors = request.getParameterValues("author");
    if (authors != null) {
%>
<%@ page import = "java.sql.*" %>
<%
    Connection conn = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/ebookshop", "myuser", "xxxx"); // <== Check!
  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



[2]

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 aguardar 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

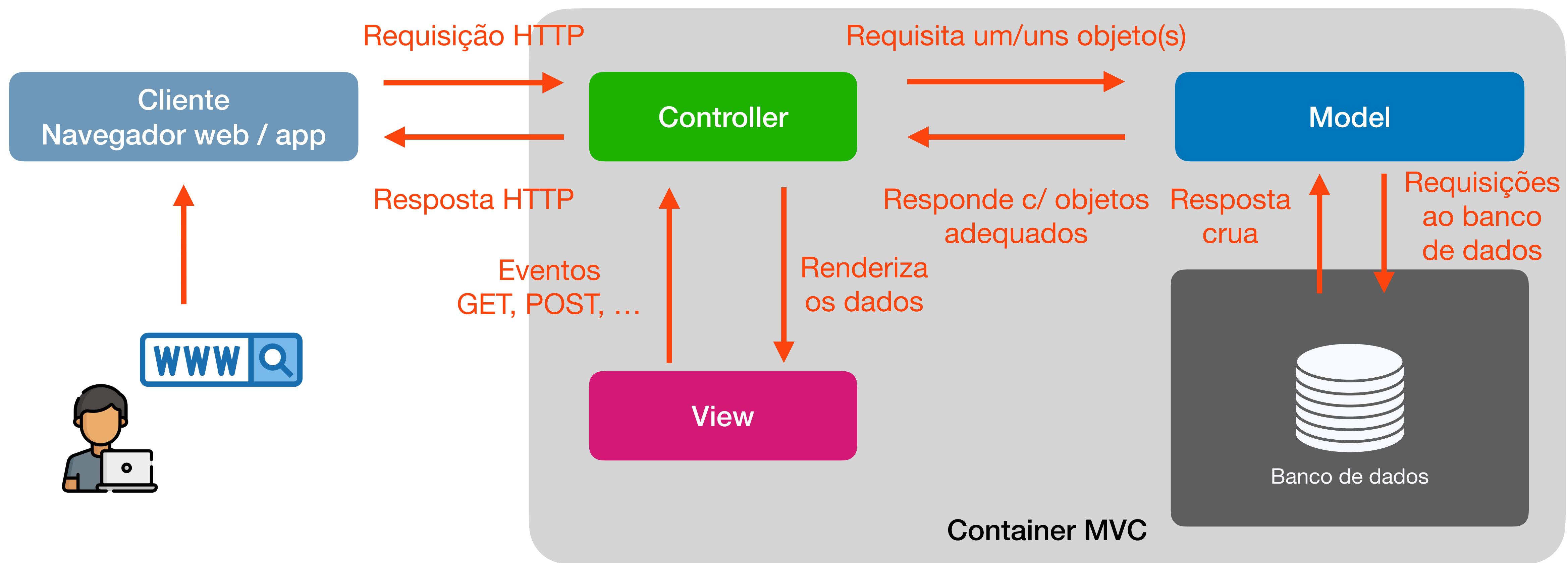
- Representação visual da nossa aplicação (GUI - Graphical User Interface)
- Mostram os dados ao usuário em forma fácil de entender baseado nas suas ações
 - Camada de interação com o usuário
- Deve refletir mudanças ocorridas nos modelos

Arquitetura MVC

Model

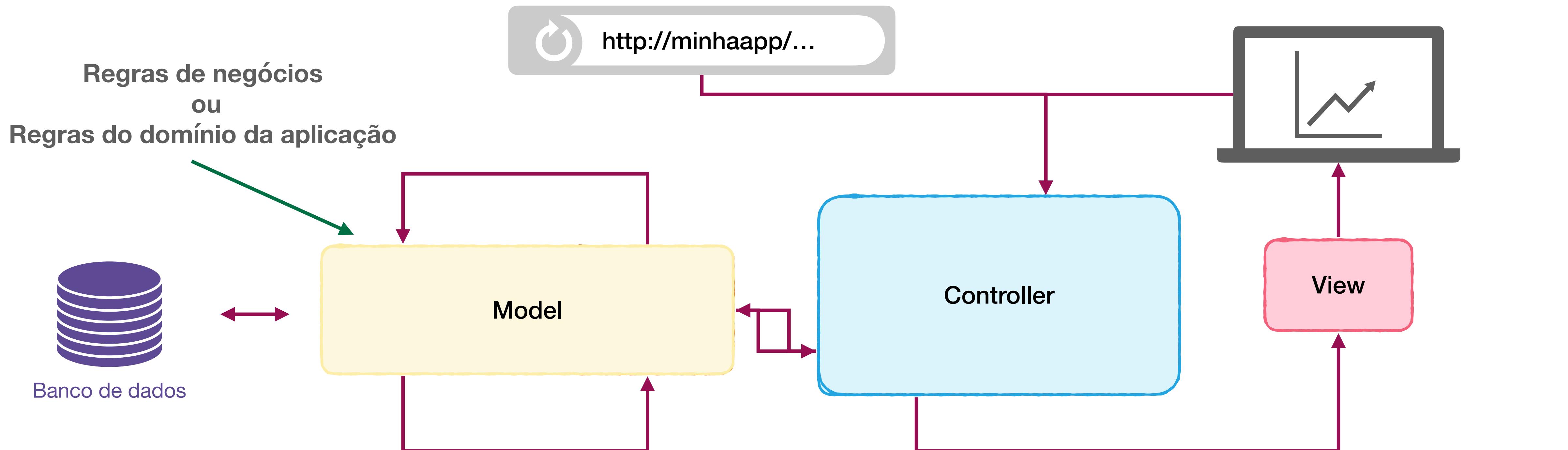
- Modelo representam o conhecimento do domínio da aplicação
- Gerencia os dados, a lógica e as regras da aplicação
- Independente da interface com o usuário
- Encapsulam os dados do banco de dados
 - Tabelas

Arquitetura MVC



Arquitetura MVC

Quebrando em mais pedaços



Fundamentos de Express

eX

Fundamentos de Express

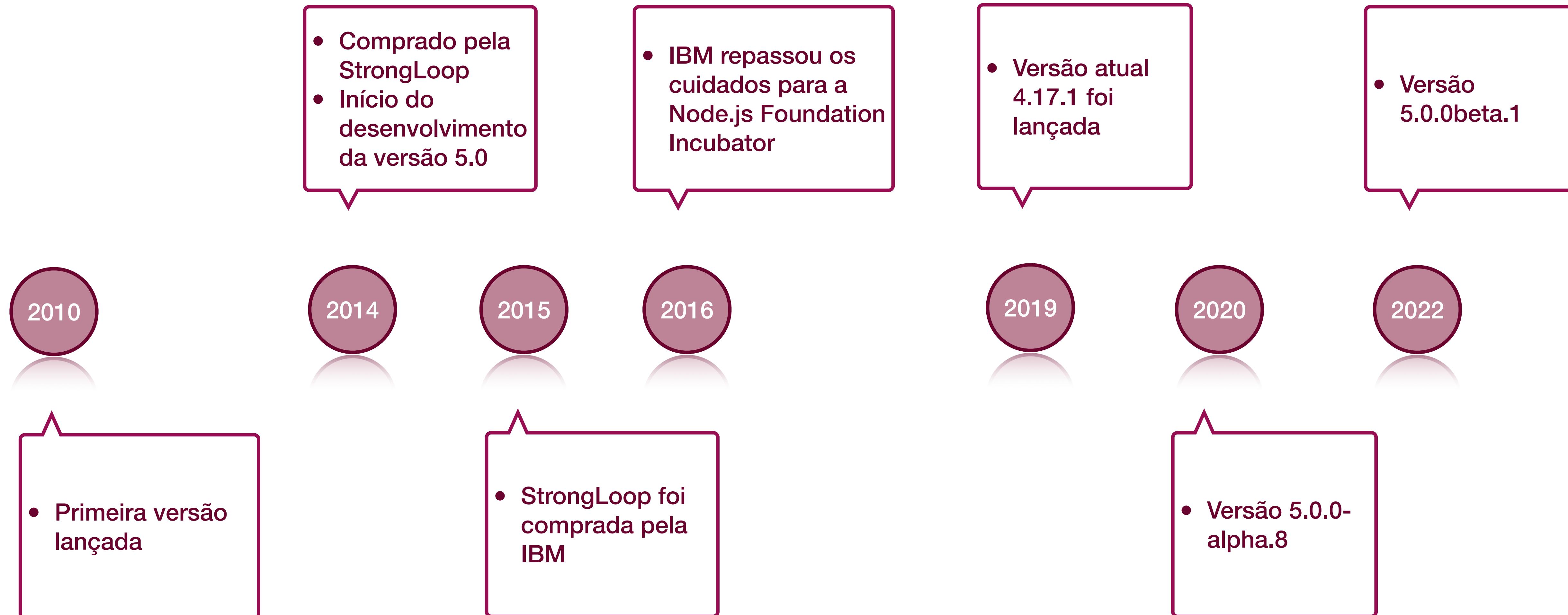
Introdução

- “Um framework web para Node, rápido, minimalista e não opinativo”
- Projetado para a criação de aplicações web e APIs utilizando o Node.js
 - Inspirado no Sinatra (Ruby)
 - Padrão de facto entre as opções de servidor web em Node



Fundamentos de Express

História - Timeline



Fundamentos de Express

Motivação

- Algumas tarefas comuns no desenvolvimento web não são suportadas diretamente pelo Node
 - Gerenciamento de recursos estáticos
 - Template engines
 - Suporte ao diversos métodos HTTP
 - Gerenciamento de Rotas

Fundamentos de Express

Características

- Open-source
- Light-weight server-side (minimalista)
- Sistema completo de rotas
- Tratamento de exceções dentro da aplicação
- Gerencia os diversos tipos (method) de requisições HTTP
- Da suporte a diversas “view engines”

Fundamentos de Express

Vantagens

- Menor tempo de desenvolvimento
- Alta escalabilidade
- Flexibilidade
- Consistência entre as linguagens de backend e frontend
- Gerenciamento de requisições concorrentes
- Grande comunidade de desenvolvedores

Fundamentos de Express

Desvantagens

- Muito trabalho manual
- Falta de padronização
 - A flexibilidade do Express é uma espada de dois gumes
 - Há pacotes de middleware para resolver quase qualquer problema
 - Utilizar os pacotes corretos para cada situação às vezes se torna um grande desafio
 - Não há "caminho certo" para estruturar um aplicativo

Fundamentos de Express

Criando um projeto

```
npm init --yes  
npm install express  
npm install -D typescript  
npm install -D @types/node @types/express
```

```
npx tsc --init  
npm install -D nodemon ts-node
```

Fundamentos de Express

Configurando os scripts

```
{  
  "scripts": {  
    "build": "npx tsc",  
    "start": "node dist/index.js",  
    "dev": "nodemon src/index.ts"  
  }  
}
```

Fundamentos de Express

Hello World

```
import express from 'express';           → Importa o módulo do Express
const app = express();                 → Cria uma aplicação Express
const PORT = 8000;
app.get('/', (req, res) => res.send('Express + TypeScript Server'));
app.listen(PORT, () => {
  console.log(`⚡ [server]: Server is running at https://localhost:${PORT}`);
});                                     → Cria uma rota
                                            → Inicia o servidor
```

```
npm run dev
```

Fundamentos de Express

O objeto app

Propriedade/Método	Descrição
<code>app.set(name, value)</code>	Define propriedades específicas da aplicação
<code>app.get(name)</code>	Recupera os valores definidos por meio da chamada <code>app.set()</code>
<code>app.enable(name)</code>	Habilitar um configuração na aplicação
<code>app.disable(name)</code>	Desabilita uma configuração na aplicação
<code>app.enabled(name)</code>	Verifica se uma configuração está habilitada
<code>app.disabled(name)</code>	Verifica se uma configuração está desabilitada
<code>app.configure([env], callback)</code>	Configura a aplicação condicionalmente de acordo com ambiente de desenvolvimento
<code>app.use([path])</code>	Carrega um middleware na aplicação
<code>app.engine(ext, callback)</code>	Regista um engine template na aplicação

Fundamentos de Express

O objeto app

Propriedade/Método	Descrição
<code>app.VERB(path, [callback...], callback)</code>	Define uma rota de acordo com o método HTTP e como tratá-la
<code>app.all(path, [callback...], callback)</code>	Define uma rota para todos método HTTP e como tratá-la
<code>app.locals</code>	Armazena todas as variáveis visíveis em views
<code>app.render(view, [options], callback)</code>	Renderiza um view da aplicação
<code>app.routes</code>	A lista de todas as rotas da aplicação
<code>app.listen</code>	Realiza a ligação e passa a esperar por conexões

Fundamentos de Express

O objeto request

Propriedade/Método Descrição

<code>req.params</code>	Armazena os valores do parâmetros nomeados na rotaparameters
<code>req.params(name)</code>	Retorna o valor de parameters nomeados em rotas de GET ou POST
<code>req.query</code>	Armazena os valores enviados via GET
<code>req.body</code>	Armazena os valores enviados via POST
<code>req.files</code>	Armazena arquivos enviados via formulário de upload
<code>req.route</code>	Prover detalhes da rota atual
<code>req.cookies</code>	Armazena os valores dos cookies

Fundamentos de Express

O objeto request

Propriedade/Método	Descrição
req.ip	O endereço IP do cliente
req.path	O path requisitado
req.host	O hostname contido no cabeçalho HTTP
req.protocol	O protocolo utilizado para realizar a requisição
req.secure	Verifica se a conexão é segura
req.url	A url requisitada junto com os parâmetros enviados na query

Fundamentos de Express

O objeto response

Propriedade/Método	Descrição
<code>res.status(code)</code>	Define o código HTTP da resposta
<code>res.set(field, [value])</code>	Define campos no cabeçalho HTTP
<code>res.get(header)</code>	Recupera informação do cabeçalho HTTP
<code>res.cookie(name, value, [options])</code>	Define um cookie no cliente
<code>res.clearCookie(name,</code>	Deleta um cookie no cliente
<code>res.redirect([status], url)</code>	Redireciona o cliente para uma URL
<code>res.location</code>	O valor da localização presente no cabeçalho HTTP

Fundamentos de Express

O objeto response

Propriedade/Método	Descrição
<code>res.send([body status], [body])</code>	Envia uma resposta HTTP com um código de resposta opcional
<code>res.json([status body], [body])</code>	Envia um JSON como resposta HTTP com um código de resposta opcional
<code>res.type(type)</code>	Define o tipo da media da resposta HTTP
<code>res.attachment([filename])</code>	Informa presença de um anexo no cabeçalho HTTP Content-Disposition
<code>res.sendFile(path, [options], [callback])</code>	Envia um arquivo para o cliente
<code>res.download(path, [filename], [callback])</code>	Solicita que o cliente baixe um arquivo
<code>res.render(view, [locals], callback)</code>	Renderiza uma view

Fundamentos de Express

Exemplo do uso de rotas

- API de usuários
 - Listar todos os usuários
 - Adicionar novos usuários
 - Mostrar os detalhes de um usuário
 - Editar/Atualizar um usuário
 - Remover um usuário

Fundamentos de Express

Exemplo do uso de rotas

Tarefa/Funcionalidade	HTTP Method	URL
Listar usuários	GET	/users
Formulário p/ adicionar um usuário	GET	/users/add
Adicionar um usuário	POST	/users
Formulário p/ editar um usuário	GET	/users/:id
Atualizar um usuário	POST	/users/:id
Remover um usuário	GET	/users/remove/:id

Fundamentos de Express

Rotas

```
import express from 'express';
const app = express();
const PORT = 8000;
app.get('/', (req, res) => res.send('Express + TypeScript Server'));
app.get('/usuarios', (req, res) => ???);
app.get('/usuarios/novo', (req, res) => ???);
app.post('/usuarios', (req, res) => ???);
app.get('/usuarios/:id', (req, res) => ???);
app.get('/usuarios:id/editar', (req, res) => ???);
app.put('/usuarios/:id', (req, res) => ???);
app.delete('/usuarios/:id', (req, res) => ???);
app.listen(PORT, () => {
  console.log(`⚡ [server]: Server is running at https://localhost:${PORT}`)
}) ;
```

Fundamentos de Express

Rotas e parâmetros

- Inevitavelmente será preciso enviar informações via url
 - Id de uma entidade no banco de dados
 - Informações para filtrar os dados do banco de dados
 - Informação para realizar a paginação do resultado de uma consulta

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

```
Route path: /flights/:from-:to
Request URL: http://localhost:3000/flights/LAX-SFO
req.params: { "from": "LAX", "to": "SFO" }
```

Fundamentos de Express

Roteadores

- Frequentemente chamados de mini-app
- Utilizados para lidar com rotas de maneira modular

```
import express from 'express';
const router = Router()
router.get('/', (req, res) => ???);
router.get('/novo', (req, res) => ???);
router.post('/', (req, res) => ???);
router.get('/:id', (req, res) => ???);
router.get('/:id/editar', (req, res) => ???);
router.put('/:id', (req, res) => ???);
router.delete('/:id', (req, res) => ???);

module.exports = router
```

Fundamentos de Express

Roteadores

```
import express from 'express';
import userRouter from './routes/userRoutes'

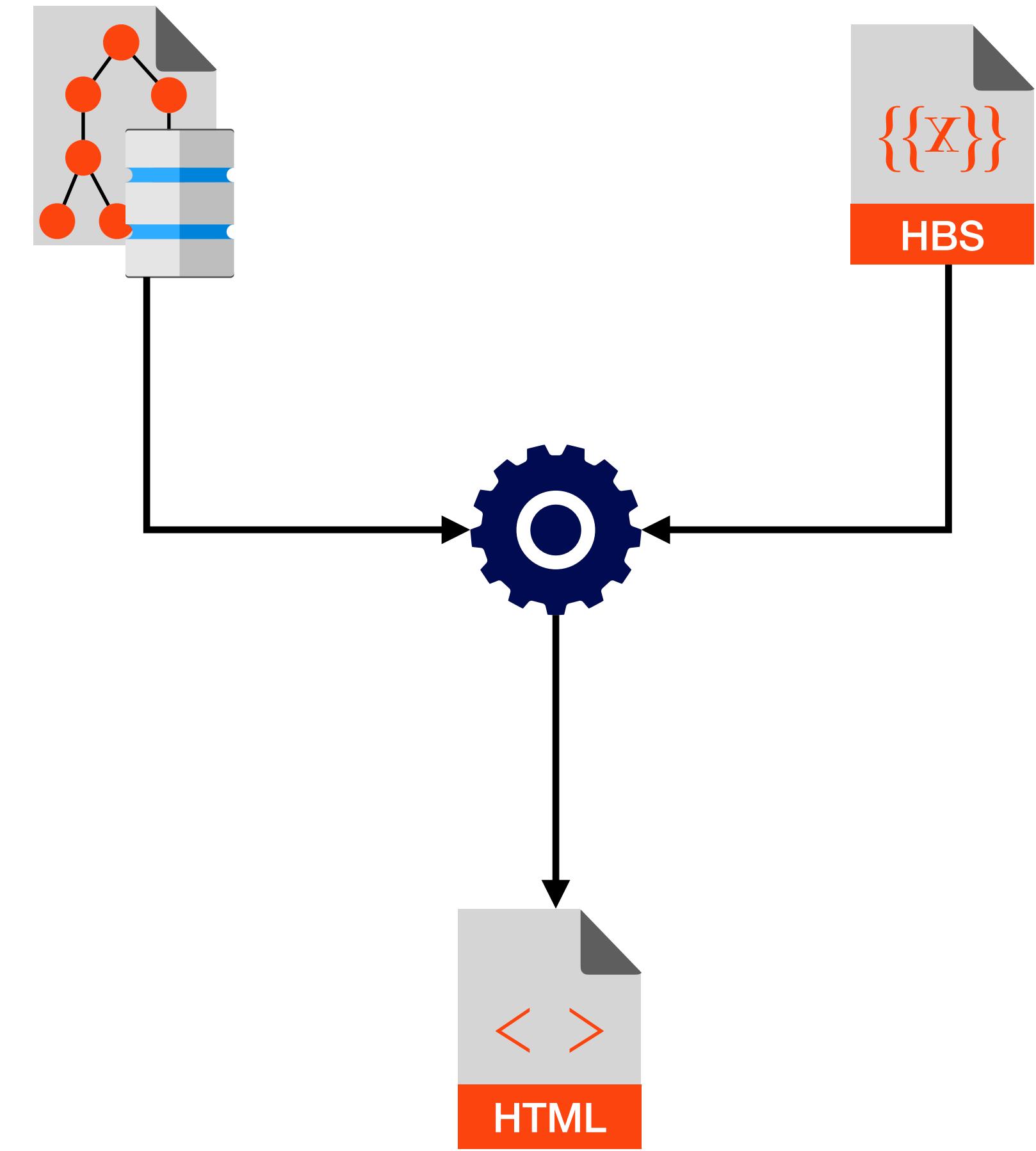
const app = express();
const PORT = 8000;

app.get('/', (req, res) => res.send('Express + TypeScript Server'));

app.use('/usuarios', userRouter)

app.listen(PORT, () => {
  console.log(`⚡ [server]: Server is running at https://localhost:${PORT}`)
}) ;
}) ;
```

Template Engine



Template Engine

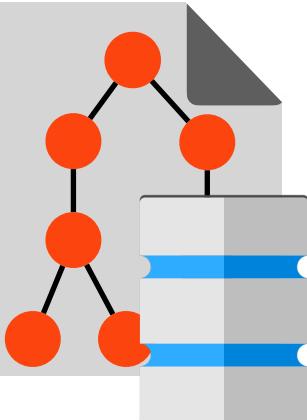
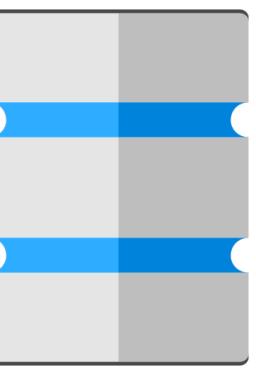
Introdução

- Também chamados de *template processor* ou *template parser*
- Combinam um *template* e dados para produzirem documentos ou até mesmo programas
- No contexto do desenvolvimento web, servem para facilitar a criação de páginas HTML de forma mais simples e organizada
 - Majoritariamente usadas em aplicação que não são construídas como APIs
- A linguagem de escrita dos template é chamada de *template language*

Template Engine

Funcionamento

- É composto dos seguintes elementos:
 - Modelo de dados
 - Banco de dados relacional, arquivo XML, planilhas
 - Arquivo de template
 - Template engine
 - Responsável por:
 - Conectar-se ao modelo de dados
 - Processar os arquivos de template



Banco de dados

Template Engine

Uso na web

- Com o passar do tempo, a comunidade de desenvolvedores web percebeu a necessidade de cada vez mais garantir o uso correto da arquitetura MVC
- *Template Engines* foram adotados para separar de forma eficaz as camadas de view e controllers
- Vale ressaltar que uma *template language* é uma linguagem insuficiente para lidar com regras de negócio
 - É impossível realizar uma consulta a um banco de dados no meio do código HTML

Template Engine

Uso na web

- Em um determinado momento vários template engines surgiram, alguns dos mais populares são/foram:

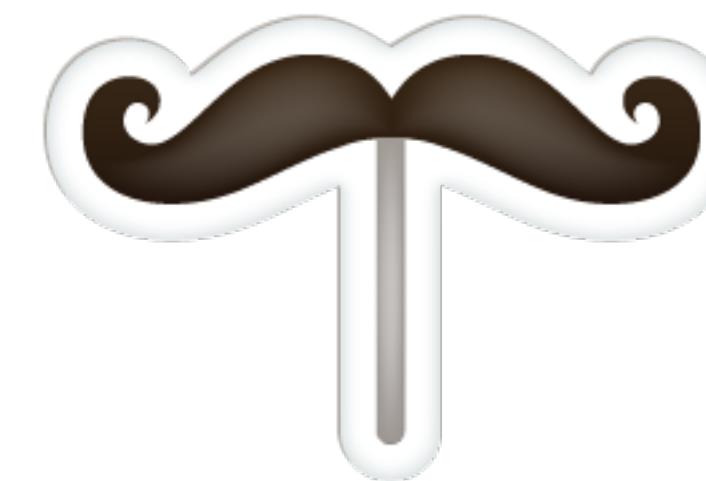


[4]



Pug [5]

handlebars



[6]



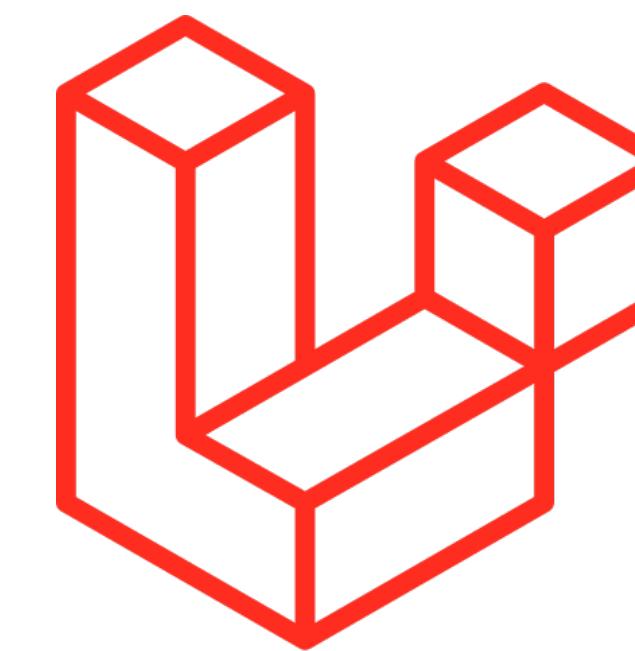
[7]



Thymeleaf [8]



Liquid [9]



Blade [10]

Template Engine

Exemplo - GitPages = Jekyll + Liquid

```
---
title: Notas de aula
---

## Notas de aula

{%
  for file in site.static_files %
    {% if file.extname == '.pdf' %}
- [Aula {{ file.basename | replace: "-", " " }}] ( {{ file.path }} )
    {% endif %}
  {% endfor %}
```

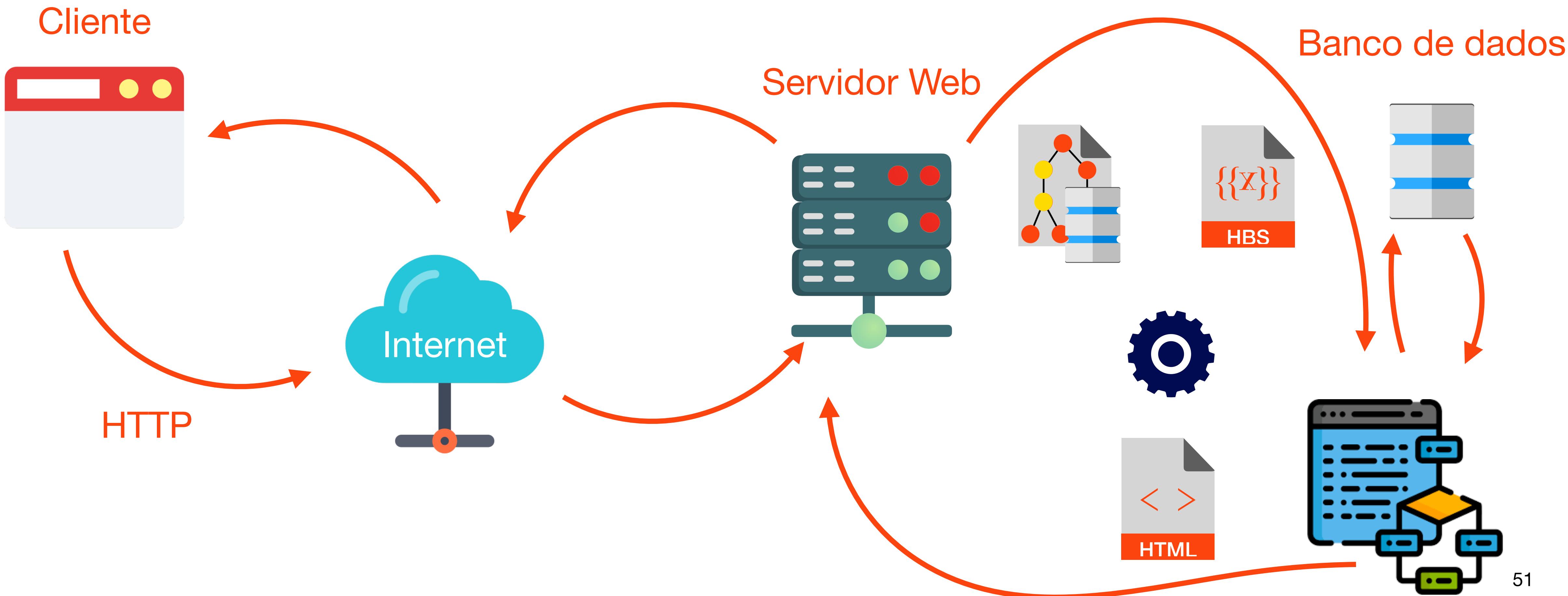
Template Engine

Exemplo - Handlebars ou hbs

```
{ {#each pokemons} }
  <div class="col-3">
    <div class="card">
      
      <div class="card-body">
        <h5 class="card-title">{{nome}}
        { {#each tipos} }
          <span class="badge bg-secondary">{{this}}
        { {/each} }
        <a href="/pokemons/{{id}}" class="btn btn-danger">Mais detalhes</a>
      </div>
    </div>
  </div>
{ {/each} }
```

Template Engine

Funcionamento



Template Engine

Configurando o Handlebars como engine no Express

```
npm install express-handlebars  
mkdir src/views  
mkdir src/views/layouts
```

```
import { engine } from 'express-handlebars'  
import path from 'path'  
  
const app = express()  
  
app.set('view engine', 'hbs');  
app.engine('hbs', engine({  
  extname: '.hbs'  
}));
```

Determina engine que será utilizada

Determina a extensão dos arquivos que serão utilizados

Indica onde estão localizados os arquivos de view

Template Engine

Configurando o layout principal

- Salvar conteúdo abaixo na pasta `src/views/layouts` como `main.hbs`

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Example App</title>
</head>
<body>
  {{body}}
</body>
</html>
```

Place holder que indica onde
o conteúdo de um view será inserido



Template Engine

Renderizando uma view

```
app.get('/', function (req, res, next) {  
  res.render('home', {  
    showTitle: true,  
  }) ;  
}) ;
```

Indica que o arquivo home.hbs localizado na pasta /src/views deve ser renderizado

Variável enviada para dentro do arquivo de view

Referências

- 1995: PHP Quietly Launches as a CGI Scripts Toolset
- PHP Examples
- Java Server-side Programming Getting started with JSP by Examples
- GUI Architectures
- Mastering MVC Architecture: A Comprehensive Guide for Web Developers
- MVC Pattern in NodeJS and express, old but gold
- MVC (Laravel) where to add logic

Referências

- [How to set up TypeScript with Node.js and Express](#)
- [O que é Template Engine?](#)
- [What are template engines?](#)
- [Wikipedia: Template processor](#)
- [Web template system](#)
- [PHP is A-OK for Templating](#)
- [A Step By Step Guide To Using Handlebars With Your Node.js App](#)

Por hoje é só