



**Prof. David Fernandes de Oliveira
Instituto de Computação
UFAM**

O que é um Framework Web

- Um **framework Web** é um conjunto de bibliotecas que visam facilitar o desenvolvimento de sistemas Web completos
- Desenvolver uma aplicação Web do zero é muitas vezes inviável e bastante trabalhoso
 - O uso de bibliotecas e frameworks de código aberto torna tudo mais fácil, rápido, confiável, e provavelmente mais seguro

METER

express

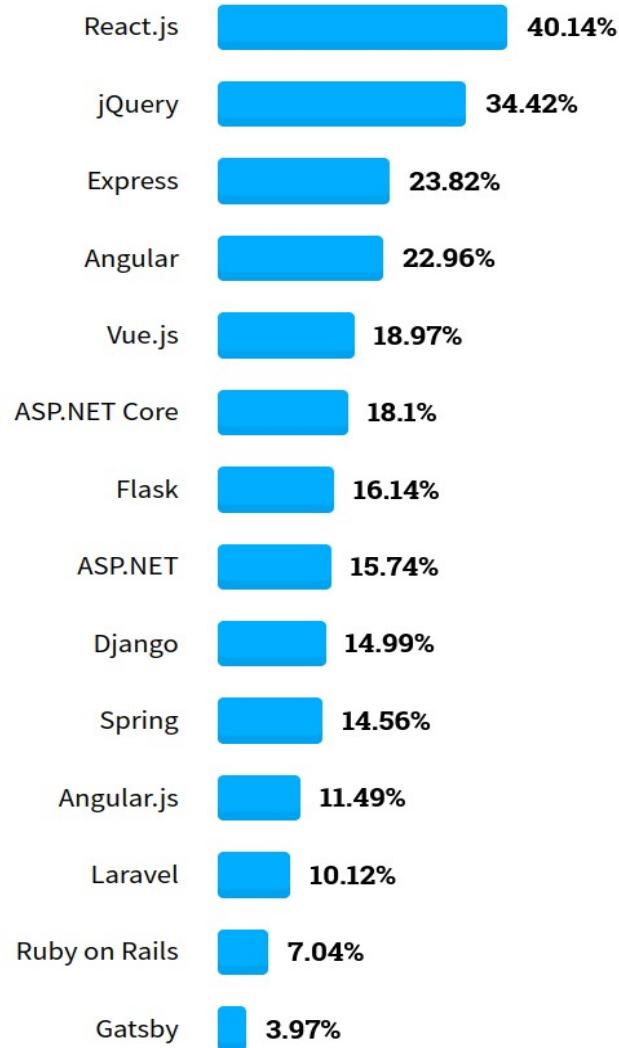
sails



django

laravel

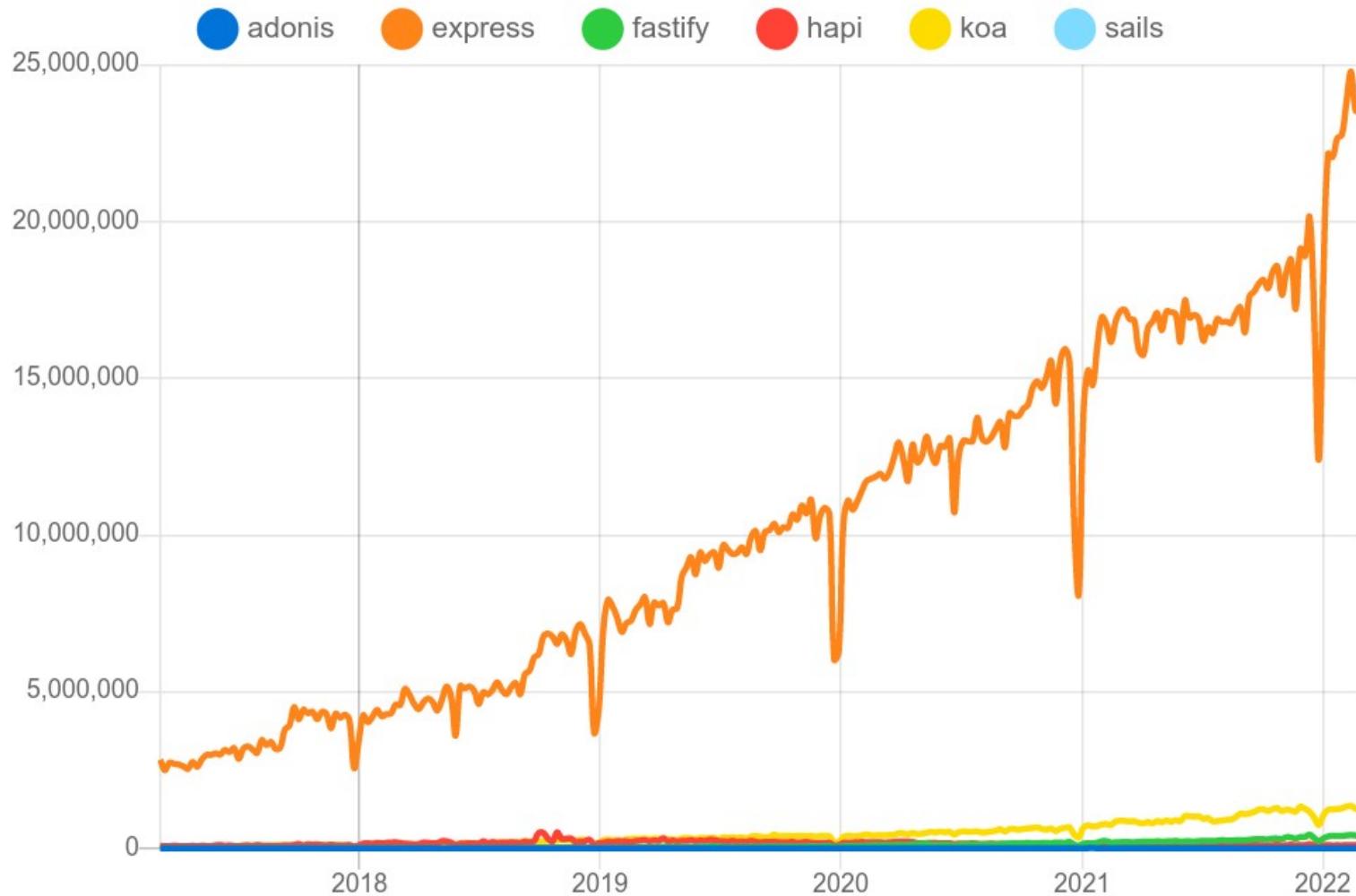
StackOverflow Surveys



Web Frameworks 2021

express

NPM Trends



express **JS**

Um Framework não opinativo

- O Express é um framework Web baseado no Node.js cujo objetivo é simplificar sua API e adicionar novos recursos
- O Express segue uma filosofia **não opinativa** e **minimalista**
 - **Não opinativa** significa que você precisará tomar muitas decisões sobre como organizar seu código dentro de sua aplicação
 - **Minimalista** significa que ele te dá total liberdade de escolher outros módulos para completar as necessidades de sua aplicação
- Essas características nos permitem usar o Express para desenvolver qualquer tipo de aplicação, de um hub de vídeos à um chat

Express - Node.js web application

expressjs.com

Black Lives Matter.
Support the Equal Justice Initiative.

Express  search Home Getting started Guide API reference Advanced topics Resources

Express 4.17.3

Fast, unopinionated, minimalist
web framework for **Node.js**

```
$ npm install express --save
```

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

express **JS**

O Arquivo package.json

- O primeiro passo no desenvolvimento de uma aplicação node.js é a criação de um arquivo chamado **package.json**
 - A função desse arquivo é armazenar vários metadados sobre a aplicação, incluindo suas dependências

```
{  
  "name": "myapp",  
  "author": "David Fernandes",  
  "private": true,  
  "version": "0.0.1",  
  "dependencies": {}  
}
```

O Arquivo package.json

- O primeiro passo no desenvolvimento de uma aplicação node.js é a criação de um arquivo chamado **package.json**
 - A função desse arquivo é armazenar vários metadados sobre a aplicação, incluindo suas dependências

```
{  
  "name": "myapp",  
  "author": "David Fernandes",  
  "private": true,  
  "version": "0.0.1",  
}
```

Para criar um novo arquivo **package.json** com os valores desejados, use o comando **npm init**. Esse comando fará algumas perguntas para o programador, e criará um arquivo **package.json** baseado nas suas respostas.

Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta executarmos o comando **npm install express**

```
myapp: fish — Konsole
david@coyote ~/dev/myapp $ npm install express
added 50 packages, and audited 51 packages in 3s
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
david@coyote ~/dev/myapp $ █
```

Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta

```
myapp : fish — Konsole
david@david-OptiPlex-5090: ~ /dev/myapp $ cat package.json
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.3"
  }
}
david@david-OptiPlex-5090: ~ /dev/myapp $
```

Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta

```
david@coyote ~/dev/myapp $ cat package.json
```

```
{  
  "name": "myapp",  
  "version": "1.0.0",
```

Observe que a definição de dependência do Express se refere à versão 4.17.3. A versão de um dado módulo é representada por três valores:

Major version (4), Minor version (17) e Patch version (3).

```
  "dependencies": {  
    "express": "^4.17.3"  
  }  
}
```

```
david@coyote ~/dev/myapp $
```

Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta

```
david@coyote ~/dev/myapp $ cat package.json
{
  "name": "myapp",
  "version": "1.0.0",
```

Observe que a definição de dependência do Express se refere à versão

Note também que a versão do Express é prefixada por um ^, indicando que ele pode ser atualizado caso seja lançado uma nova **minor version** do framework. Isto é, sua aplicação é dependente do Express versão 4.*.*

Alguns módulos podem ser prefixados com um ~, indicando eles só podem ser atualizados caso seja lançado um novo patch desses módulos.

```
david@coyote ~/dev/myapp $ cat package.json
{
  "dependencies": {
    "express": "^4.17.3"
  }
}
```

```
david@coyote ~/dev/myapp $
```

Hello World em Express

- Para fazer um primeiro teste com o Express, podemos criar um arquivo **app.js** com o seguinte conteúdo:

```
const express = require("express");
const app = express();
app.get("/", function(req, res) {
    res.send("Hello world!");
});
app.listen(3000, function() {
    console.log("Express app iniciada na porta 3000.");
});
```

Envia a string
Hello World
para o browser

Inicializa o
servidor HTTP
na porta 3000

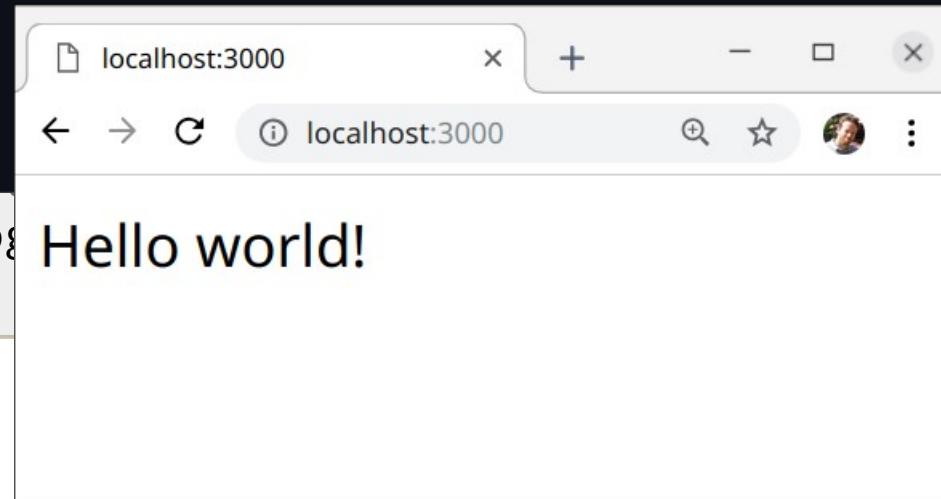
Hello World em Express

- Para fazer um primeiro teste com o Express, podemos criar um arquivo **app.js** com o seguinte conteúdo:

```
const express = require("express"); // Envia a string
david@coyote:~/dev/myapp$ node app
Express app iniciada na porta 3000.

const app = express();
app.get('/', (req, res) => {
  res.send('Hello world!');
});

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



Inicializa o
servidor HTTP
na porta 3000

Tipos de Request Handles

- As requisições GET podem ser tratadas através de **app.get**

```
app.get("/", function(req, res) {  
    res.send("Hello world!");  
});
```

- As requisições POST podem ser tratadas através de **app.post**

```
app.post("/", function (req, res) {  
    console.log("Requisição POST no /");  
})
```

- A função **app.use** é executada em toda requisição

```
app.use(function (req, res) {  
    console.log("Executado por toda requisição");  
})
```

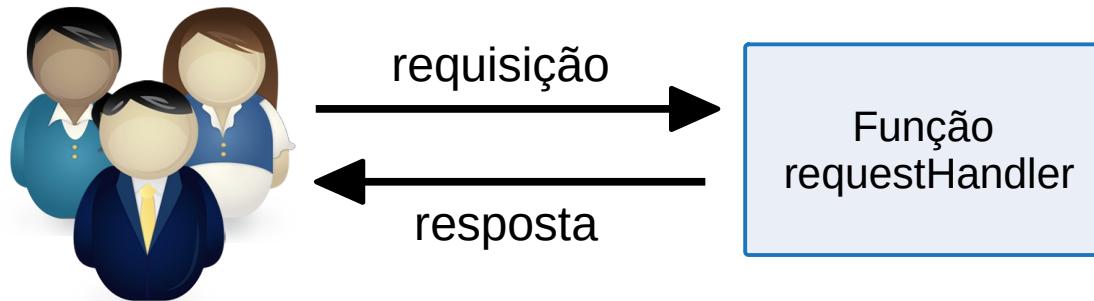
Os Fundamentos do Express

- O framework express provê 4 funcionalidades principais, que serão vistas ao longo dos próximos slides:
 - **Middleware** – São um conjunto de funções que tem acesso ao objeto de solicitação (req) e ao o objeto de resposta (res)
 - **Routing** – As rotas definem como o aplicativo irá responder às solicitações (URI + HTTP Method) dos clientes
 - **Extensões aos objetos request e response** – Express estende os objetos request e response com novos métodos e propriedades
 - **Views** – As views permitem criar conteúdo HTML de forma dinâmica

Middleware

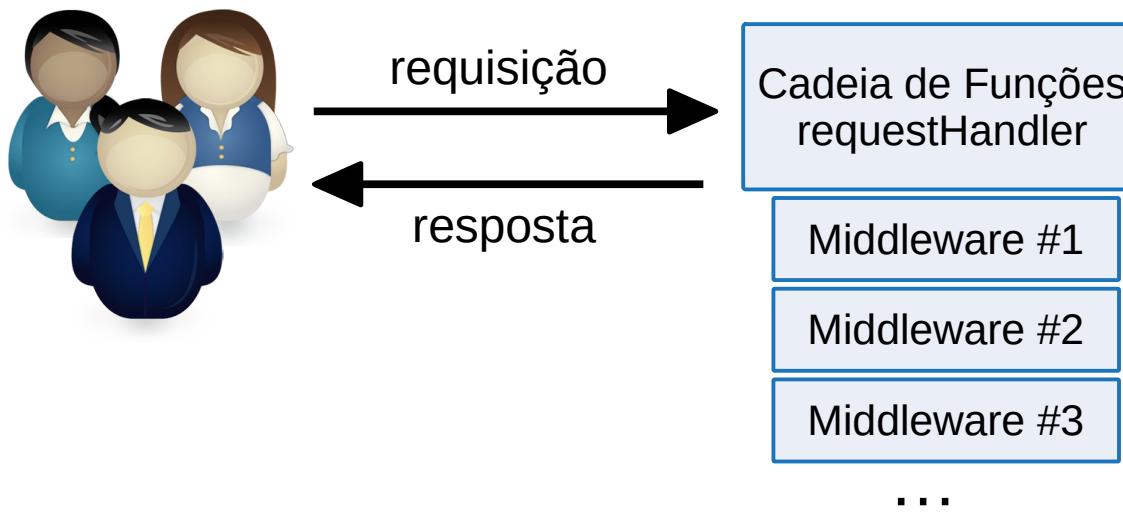
- Usando Node.js puro, usamos o callback de **http.createServer** para responder as requisições dos usuários

```
http.createServer(function(req, res) {  
  console.log("Requição do usuário: " + req.url);  
  res.end("Instituto de Computação!");  
});
```



Middleware

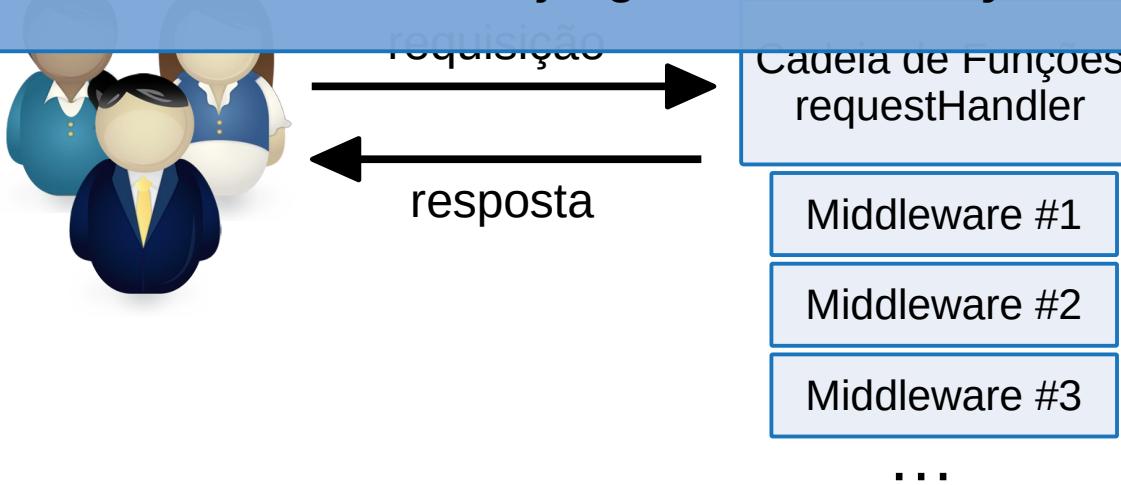
- Os **middlewares** são funções similares ao callback de **http.createServer**, mas possuem uma diferença fundamental: ao invés de um callback, podemos criar vários em sequência



Middleware

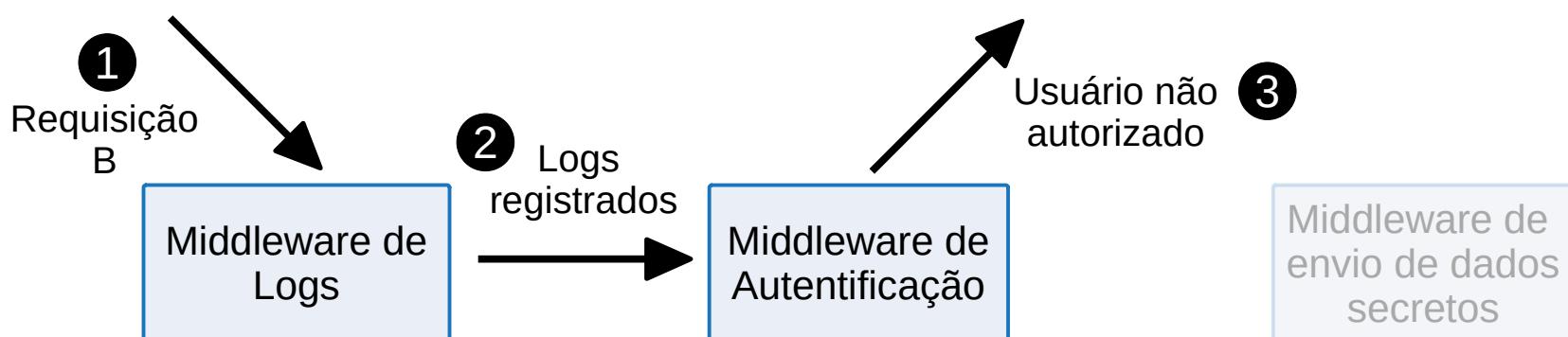
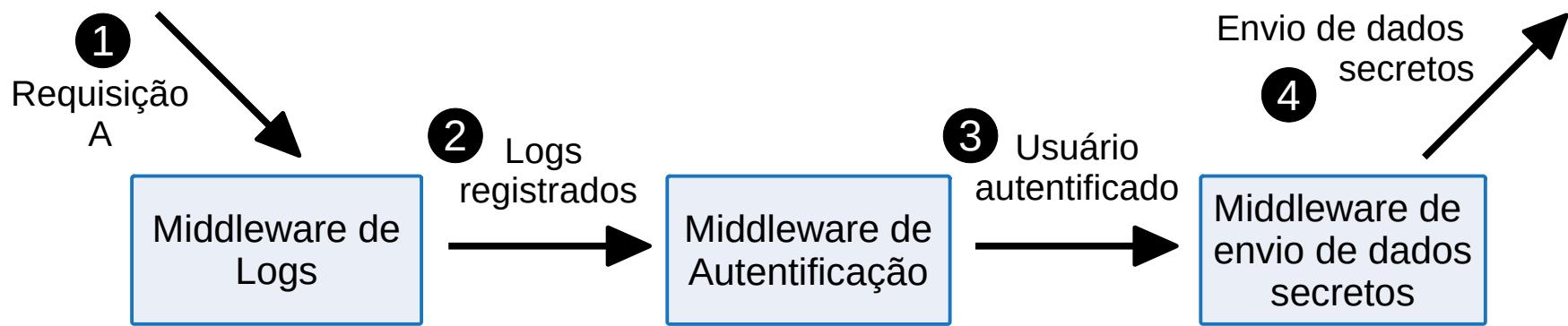
- Os **middlewares** são funções similares ao callback de **http.createServer**, mas possuem uma diferença fundamental:
ao invés de um callback, podemos criar vários em sequência

Os middlewares não são uma invenção do Express, e estão presentes em vários outros frameworks: **Django**, **Laravel**, **Ruby on Rails**, etc.



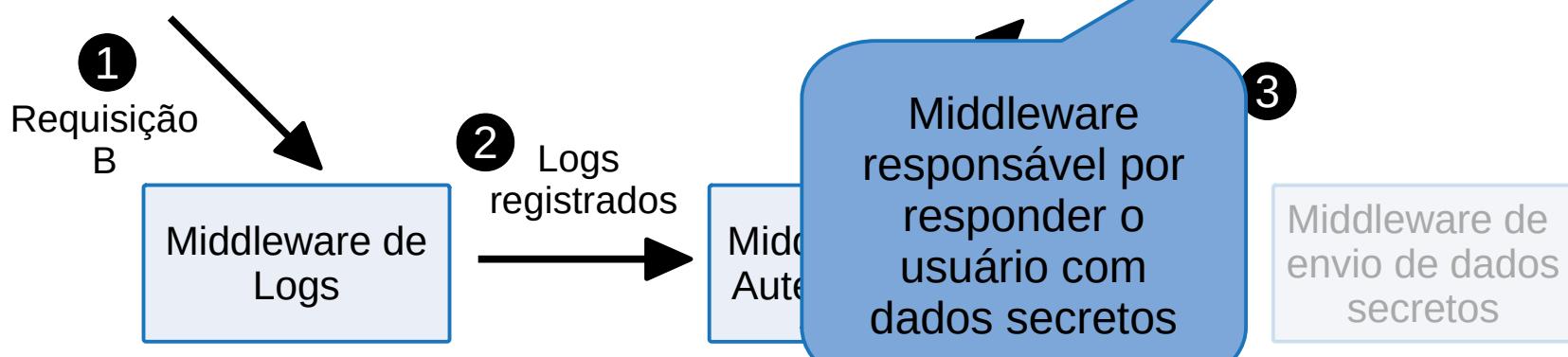
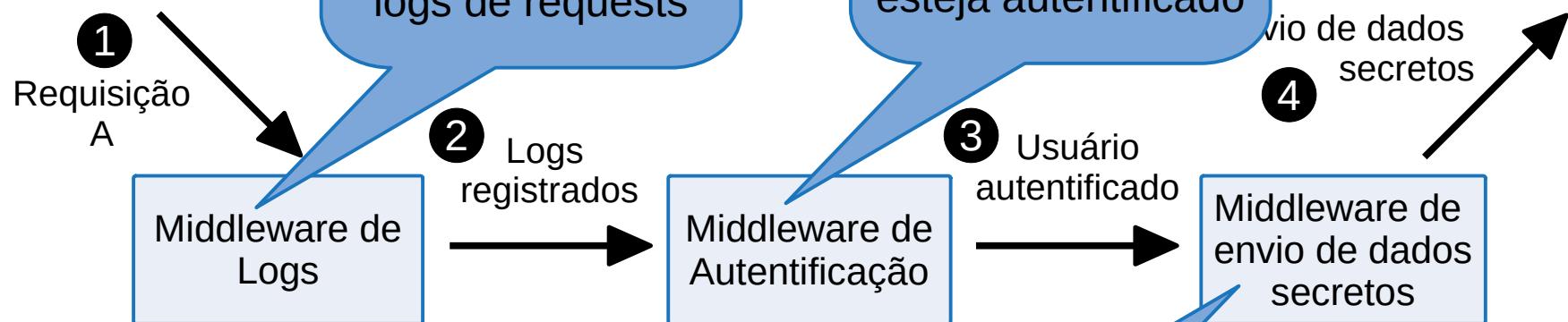
Middleware

- A Figura abaixo mostra um exemplo de app que autentifica os usuários e retorna dados sigilosos para usuários autorizados



Middleware

- A Figura mostra que os usuários devem ser autenticados para enviar dados secretos.



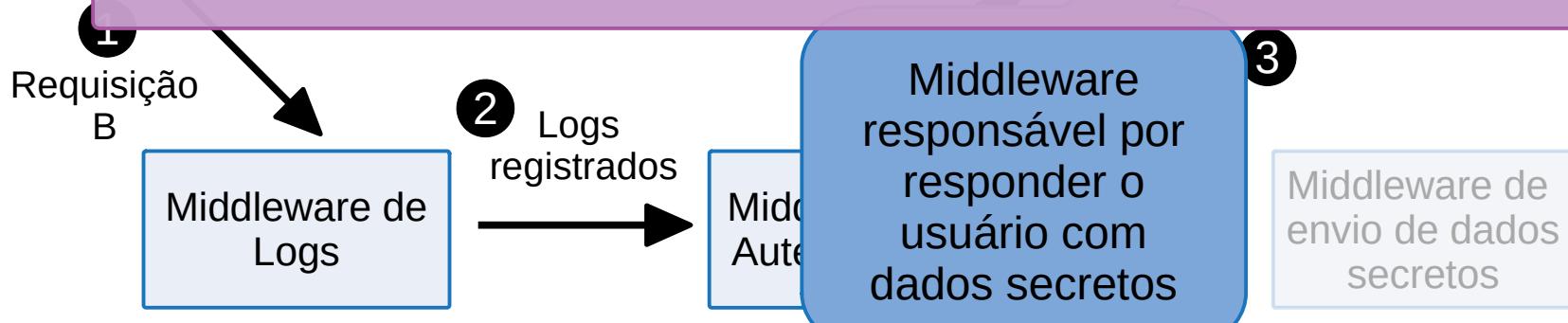
Middleware

- A Figura mostra que os usuários devem ser autenticados para enviar dados secretos



Os middlewares são muito úteis porque permitem dividir a lógica da aplicação em várias partes menores

Além disso, muitos módulos de terceiros são projetados para se encaixarem perfeitamente no formato dos middlewares



Middleware

- Além dos parâmetros **request (req)** e **resposta (res)**, os middlewares também recebem o parâmetro **next**, que pode ser usado para chamar o próximo middleware da cadeia

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

app.use(function(req, res, next) {
  console.log("Requisição " + req.method + " " + req.url);
  next();
});

app.use(function(req, res) {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello world!");
});

app.listen(3000, function() {
  console.log("Express app iniciada na porta 3000.");
});
```

Middleware

- Além de middleware ser usado

```
david@coyote:~/dev/myapp$ npm start
> myapp@0.0.1 start /home/david/dev/myapp
> node app.js
```

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

Express app iniciada na porta 3000.
```

```
Requisição GET /
```

```
Requisição GET /about
```

```
console.log(`Requisição para a url`);
```

```
});
```

```
localhost:3000
```

```
David
```

```
localhost:3000
```

```
David
```

```
x
```

```
app.get('/', (req, res) => {
  res.send('Hello world!');
});
```

```
localhost:3000/about
```

```
David
```

```
x
```

```
Hello world!
```

```
});
```

```
});
```

```
});
```

```
});
```

Middleware

- Dentro de cada middleware, também é possível alterar propriedades dos objetos **request (req)** e **response (res)**

```
app.use(function(req, res, next) {  
  console.log('Requisição à URL ' + request.url);  
  next();  
});  
  
app.use(function(req, res, next) {  
  if (user.checkAuth(req)) {  
    next();  
  } else {  
    res.statusCode = 403;  
    res.end("Não autorizado");  
  }  
});  
  
app.use(function(req, res) {  
  res.end('Dados secretos: {código,156234}');  
});
```

Código HTTP
para
Forbidden

Bibliotecas Middleware

- Ao invés de programar todos os middlewares de sua aplicação, podemos usar middlewares disponíveis no repositório do NPM
- Existem milhares de middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs
- O primeiro passo para usar o Morgan é instalá-lo

```
npm install morgan
```

Bibliotecas Middleware

- Ao invés de programar todos os middlewares de sua aplicação, podemos usar middlewares disponíveis no repositório do NPM
- Existem milhares de middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs
- O primeiro passo para usar o Morgan é instalá-lo

```
n const express = require("express");
const logger = require("morgan");
const app = express();

app.use(logger("short"));

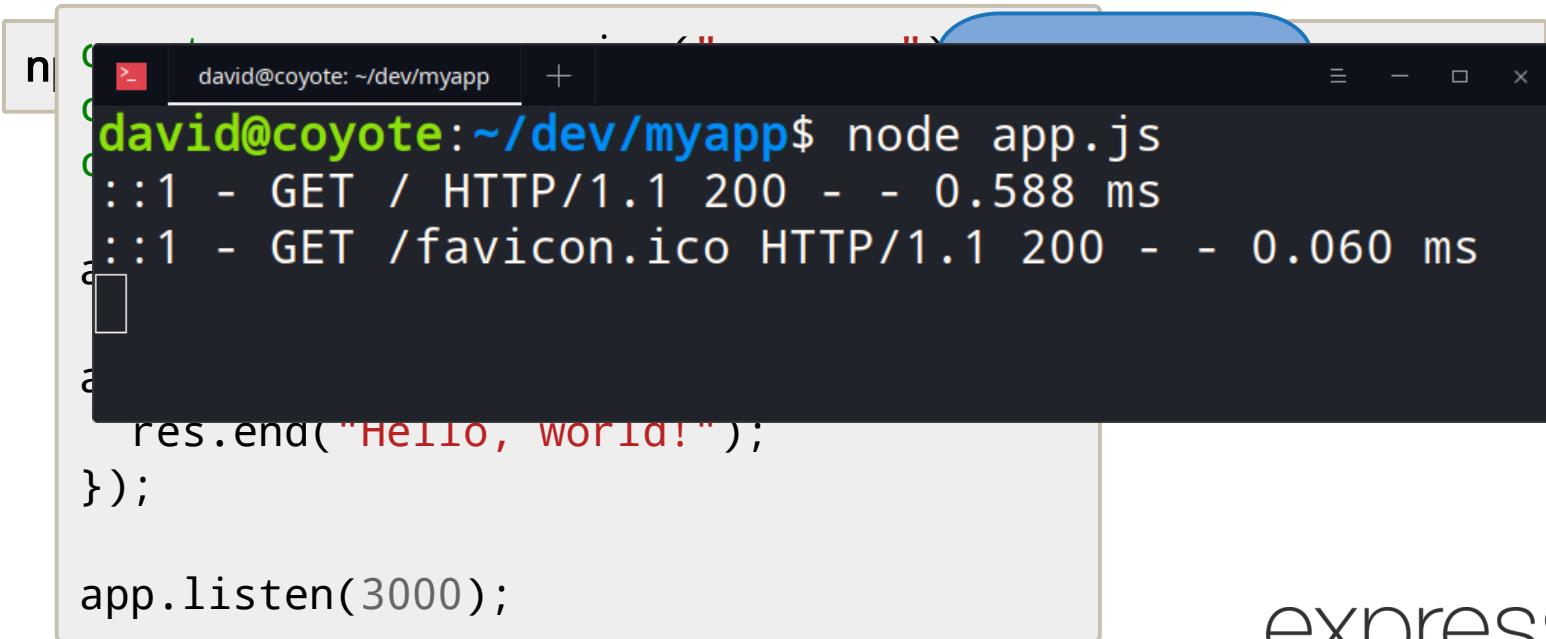
app.use(function(req, res) {
  res.end("Hello, world!");
});

app.listen(3000);
```

logger("short")
retorna uma
função

Bibliotecas Middleware

- Ao invés de programar todos os middlewares de sua aplicação, podemos usar middlewares disponíveis no repositório do NPM
- Existem milhares de middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs
- O primeiro passo para usar o Morgan é instalá-lo



The image shows a terminal window with a dark theme. The command `node app.js` is run, and the output shows two log entries from the Morgan middleware:

```
david@coyote:~/dev/myapp$ node app.js
::1 - GET / HTTP/1.1 200 - - 0.588 ms
::1 - GET /favicon.ico HTTP/1.1 200 - - 0.060 ms
```

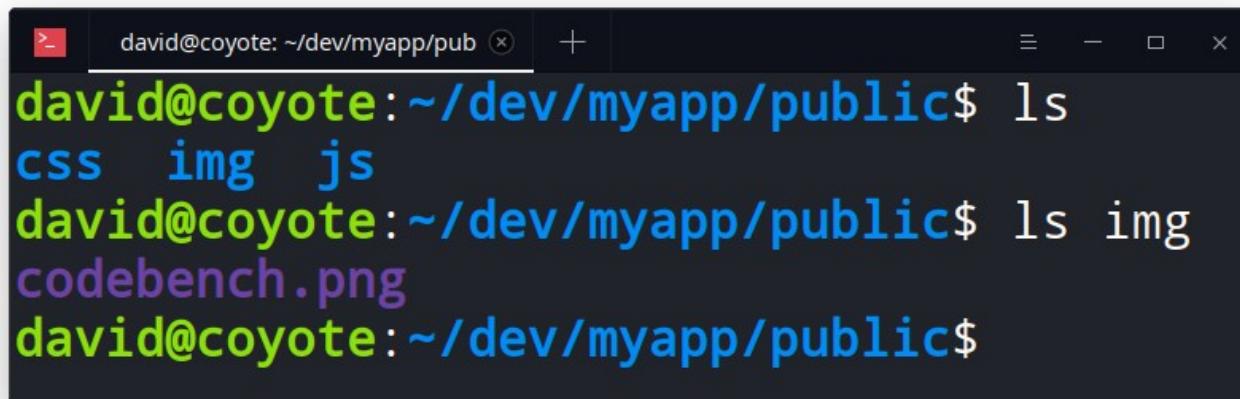
Below the terminal window, parts of the `app.js` file are visible, showing the Morgan configuration and the main application logic:

```
res.end("Hello, world!");
});

app.listen(3000);
```

Bibliotecas Middleware

- Outro exemplo de middleware amplamente utilizado é o **middleware de arquivos estáticos** do Express
- Esse middleware é usado para envio de arquivos de imagens, de estilo CSS e JavaScript para o browser
- A maioria das aplicações possui um diretório específico para esse tipo de arquivo, normalmente chamado de **public**



A screenshot of a terminal window titled "david@coyote: ~/dev/myapp/pub". The window shows a command-line interface with the following text:

```
david@coyote:~/dev/myapp/public$ ls
css  img  js
david@coyote:~/dev/myapp/public$ ls img
codebench.png
david@coyote:~/dev/myapp/public$
```

Bibliotecas Middleware

- Através do código abaixo, qualquer imagem (assets) disponível em **/public/img** pode ser acessada pela aplicação

```
var express = require("express");
var app = express();

app.use('/img', [
  express.static(`__dirname/public/img`)
]);

app.use(function(req, res) {
  res.end("Acesse o arquivo /img/codebench.png");
});

app.listen(3000, function() {
  console.log("Express app iniciada na porta 3000.");
});
```

Executa **next**
caso o arquivo
não seja
encontrado

Executado em
toda requisição
que possua a
substring **/img**

Bibliotecas Middleware

- Através do código abaixo, qualquer imagem (assets) disponível em **/public/img** pode ser acessada pela aplicação

```
var express = require("express");
var app = express();
app.use(express.static('public'));
app.get('/', function(req, res) {
  res.sendFile(__dirname + '/index.html');
});
app.get('/img/:name', function(req, res) {
  res.sendFile(__dirname + '/img/' + req.params.name);
});
app.listen(3000, function() {
  console.log("Express app iniciada na porta 3000.");
});
```



CODEBENCH

Executa **next**
caso o arquivo

0

o em
sição
ua a
/img

Bibliotecas Middleware

- Através do código abaixo, qualquer imagem (assets) disponível em **/public/img** pode ser acessada pela aplicação

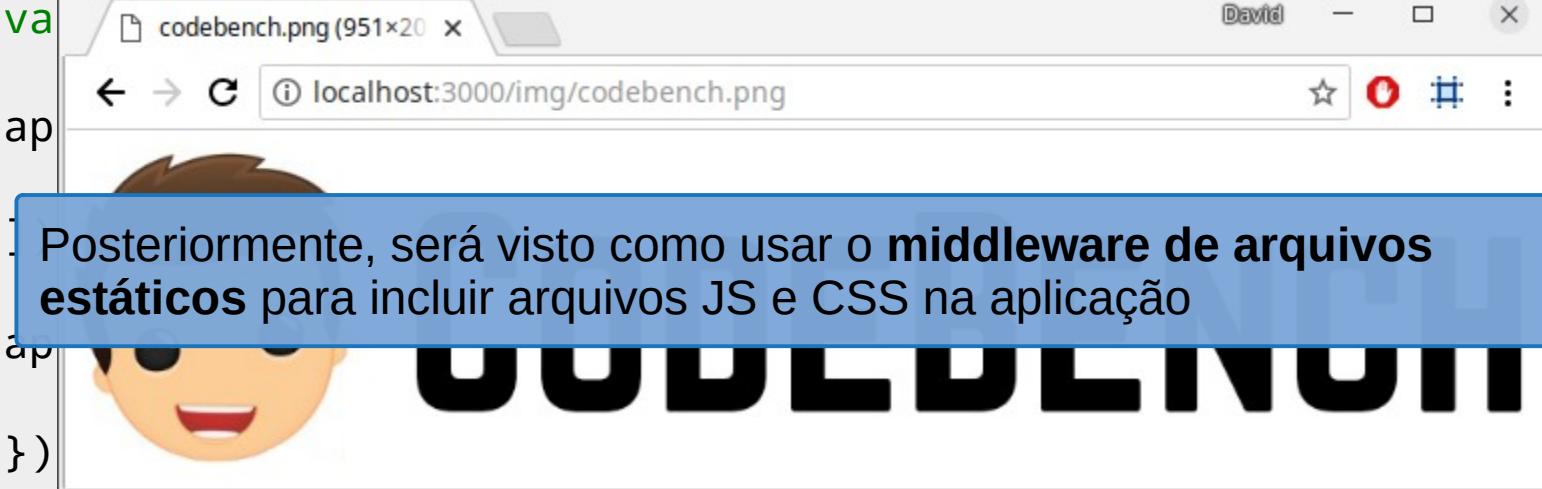
```
var express = require("express");
var app = express();
app.use(express.static("public"));

app.get("/", function(req, res) {
  res.sendFile(__dirname + "/index.html");
});

app.get("/img/:name", function(req, res) {
  res.sendFile(__dirname + "/img/" + req.params.name);
});

app.listen(3000, function() {
  console.log("Express app iniciada na porta 3000.");
});
```

Executa **next**
caso o arquivo



Bibliotecas Middleware

- Existem várias outras bibliotecas middleware muito usadas, dentre as quais podemos destacar:
 - **Connect-ratelimit** – Permite limitar o número de requisições por hora de alguém que esteja tentando derrubar seu servidor
 - **Compression** – comprime os dados enviados para o cliente (browser) usando gzip
 - **Body-parser** – converte a propriedade body da requisição do usuário (req.body) para vários formatos, incluindo JSON
 - **Response-time** – registra o tempo de resposta das requisições dos usuários, sendo útil para debugar a performance da aplicação

Roteamento

- Roteamento é usado para definir quais middlewares serão usados para responder quais tipos de URLs

```
app.get("/", function(req, res) {  
  res.end("Bem-vindo ao meu site!");  
});
```

Executado só
na requisição
/

```
app.get("/sobre", function(req, res) {  
  res.end("Bem-vindo à página sobre!");  
});
```

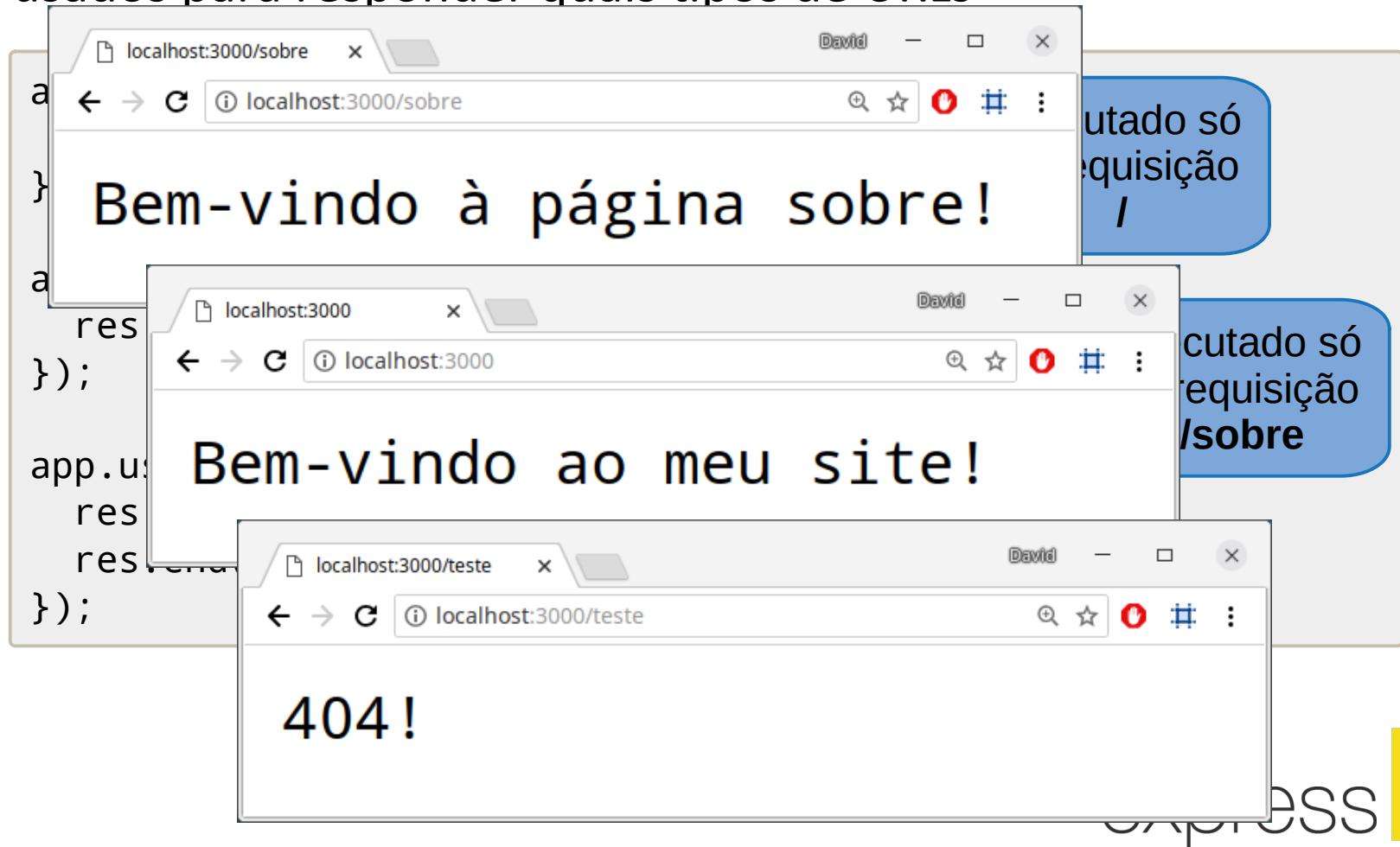
Executado só
na requisição
/sobre

```
app.use(function(req, res) {  
  res.statusCode = 404;  
  res.end("404!");  
});
```

Executado
na falha das
outras rotas

Roteamento

- Roteamento é usado para definir quais middlewares serão usados para responder quais tipos de URLs

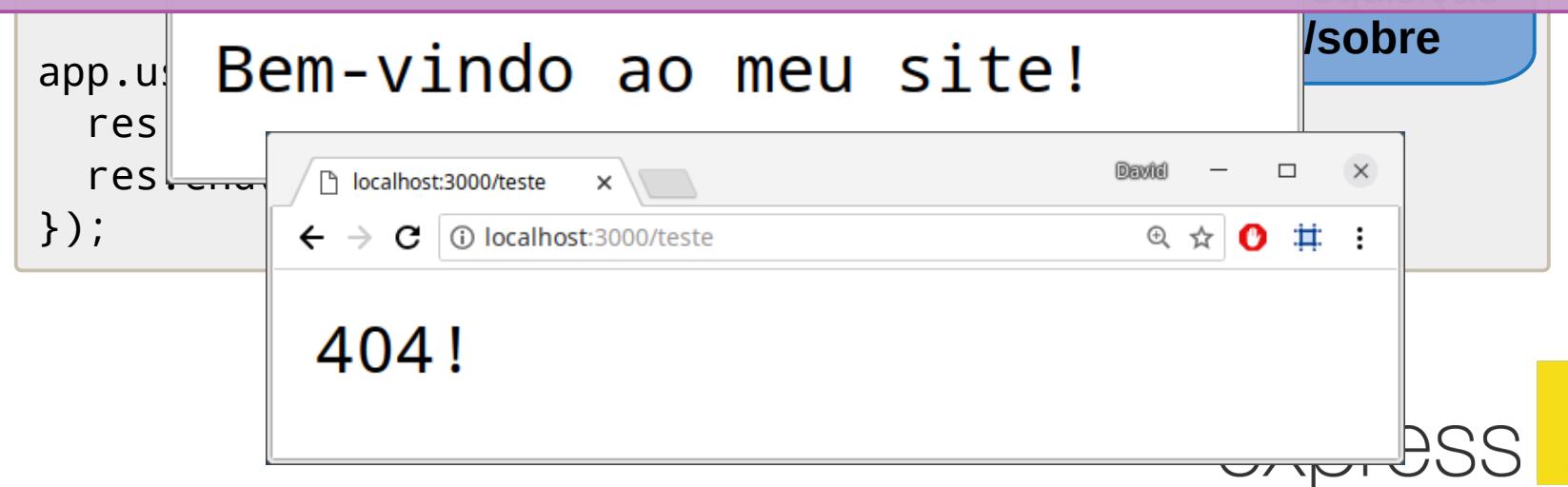


Roteamento

- Roteamento é usado para definir quais middlewares serão usados para responder quais tipos de URLs



Além de `app.get()`, também existe o método `app.post()`, que é usado para responder às requisições do tipo POST



Roteamento

- Além dos nomes de rotas fixos, o Express aceita o uso de rotas mais complexas, incluindo **expressões regulares**

```
app.get( /^\/(api|rest)\/.+$/, function(req, res) {  
    res.send("Envio de dados da API!");  
});
```

- Outra possibilidade é a **passagem de parâmetros** através da URL, tal como demonstrado no exemplo a seguir

```
app.get("/bemvindo/:nome", function(req, res) {  
    res.end(`Seja bem vindo ${req.params.username}`);  
});
```

Roteamento

- Além dos nomes de rotas fixos, o Express aceita o uso de rotas mais complexas, incluindo **expressões regulares**

```
app.get( /^\/(api|rest)\/.+$/, function(req, res) {  
    res.send("Envio de dados da API!");  
});
```



- Outra possibilidade é usar URLs com parâmetros, através da

```
app.get("/bemvindo/:nome", function(req, res) {  
    res.end(`Seja bem vindo ${req.params.username}`);  
});
```



Arquivo Separado de Rotas

- Ao invés de definir todas as rotas no arquivo principal da aplicação, é possível criar um arquivo separado para elas
- Para isso, podemos usar a classe **express.Router**, que retorna um middleware com um sistema de roteamento completo

```
// Módulo de rotas, arquivo /config/routes.js
const express = require('express');
const router = express.Router();

router.get('/', function(req, res) {
  res.send('Página principal do site');
});

router.get('/sobre', function(req, res) {
  res.send('Página sobre');
});

module.exports = router;
```

Arquivo Separado de Rotas

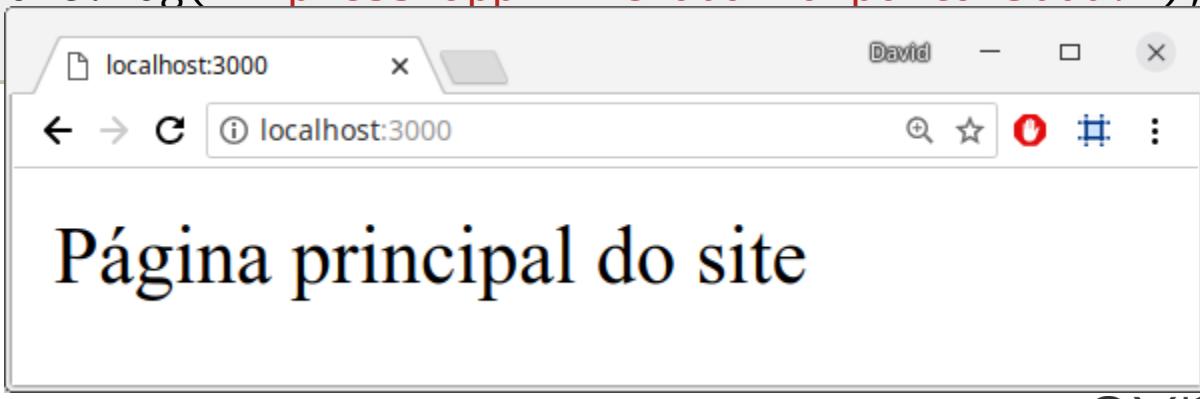
- Em seguida, podemos carregar o módulo de roteamento no arquivo principal da app

```
// Módulo principal, arquivo /app.js

const express = require("express");
const router = require("./config/routes");

const app = express();
app.use(router);

app.listen(3000, function() {
  console.log("Express app iniciada na porta 3000.");
});
```



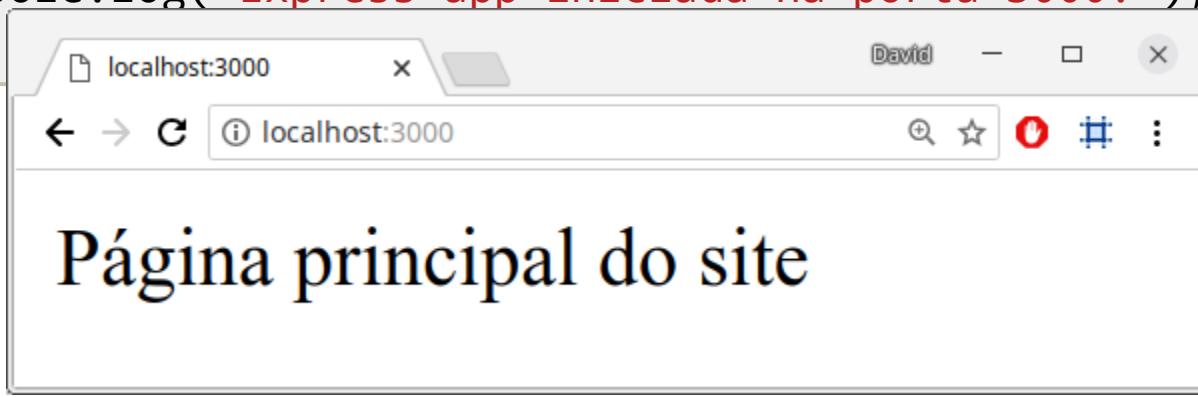
Arquivo Separado de Rotas

- Em seguida, podemos carregar o módulo de roteamento no arquivo principal da app

```
// Módulo principal, arquivo /app.js  
  
const express = require("express");  
const router = require("./config/routes");
```

O **Router** surgiu na versão 4.0 do **Express**, e é uma alternativa ao método tradicional de criação das rotas. A função do Router é permitir a definição das rotas em um módulo separado.

```
});
```



Extensões de Req/Res

- **Request** (req) e **response** (res) são objetos passados para todas as funcões que tratam as requisições de usuários
- Eles já existiam no Node.js puro, mas o Express adicionou novas funcionalidades para esses dois objetos
- Por exemplo, o método **redirect()** foi adicionado pelo Express

```
res.redirect("/hello/world");
res.redirect("http://expressjs.com");
```

Redireciona
o usuário para
a nova URL

- Outro exemplo é o método **sendFile()**

```
res.sendFile("/path/to/song.mp3");
```

Envia o
arquivo para
o browser

Views

- As **views** são responsáveis por gerar automaticamente o código HTML que é enviado pelo usuário a cada requisição
 - Fazem parte do modelo **MVC – Model, View, Controller**
- Existem muitas engines de views disponíveis para o Express, dentre as quais destaca-se: EJS, Handlebars, Pug e Mustache



Views

- As **views** são responsáveis por gerar automaticamente o código HTML que é enviado pelo usuário a cada requisição
 - Fazem parte do modelo **MVC – Model, View, Controller**
- Existem muitas engines de views disponíveis para o Express,

Conforme já dito, o Express é um **framework não opinativo** e requer uma série de acessórios de terceiros para construir uma aplicação completa

Desta forma, como o Express não possui uma **engine de views** própria, é necessário escolher uma dentre as engines disponíveis para node.js



Handlebars provides the power necessary to let you build **semantic templates** effectively with no frustration.

Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates.

Views

- As **views** são responsáveis por gerar automaticamente o código HTML que é enviado pelo usuário a cada requisição
 - Fazem parte do modelo **MVC – Model, View, Controller**
- Existem muitas engines de views disponíveis para o Express,

Conforme já dito, o Express é um **framework não opinativo** e requer uma série de acessórios de terceiros para construir uma aplicação completa

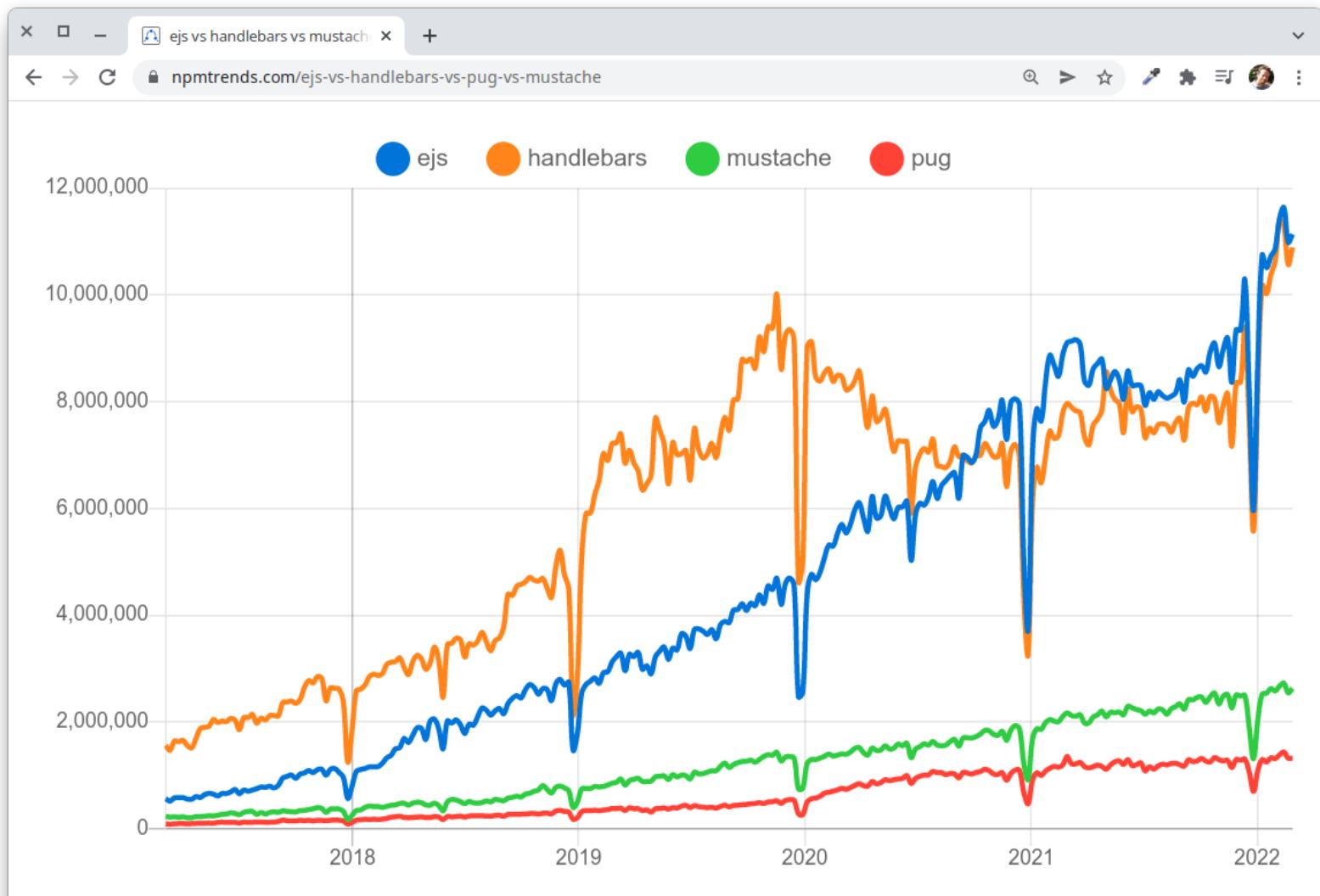
O que parece ser uma limitação é na realidade uma grande vantagem, pois sempre que formos desenvolver uma nova aplicação, podemos escolher as melhores bibliotecas disponíveis para node.js naquele momento



Handlebars provides the power necessary to let you build **semantic templates** effectively with no frustration.

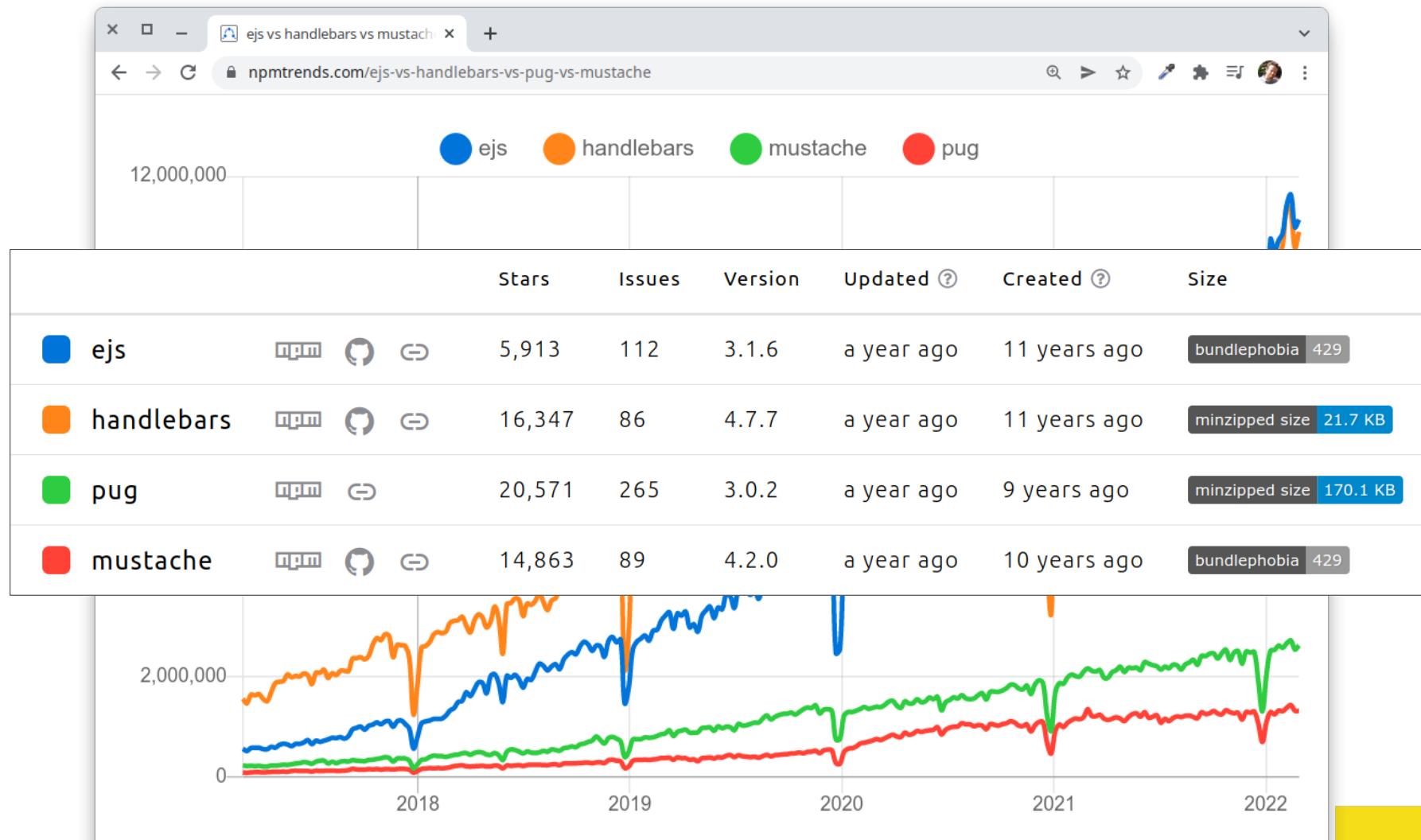
Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates.

Views



express **JS**

Views



express **JS**

Views



express **JS**

Engine Handlebars

- Para usar o **Handlebars**, é necessário instalá-lo através do NPM

```
$ npm install express-handlebars
```

- Após isso, é preciso informar o Express sobre a escolha do **Handlebars** como engine de views

```
const handlebars = require('express-handlebars');

app.engine('handlebars', handlebars());
app.set('view engine', 'handlebars');
app.set('views', __dirname + '/views');
```

Informa a localização do diretório de views

Engine Handlebars

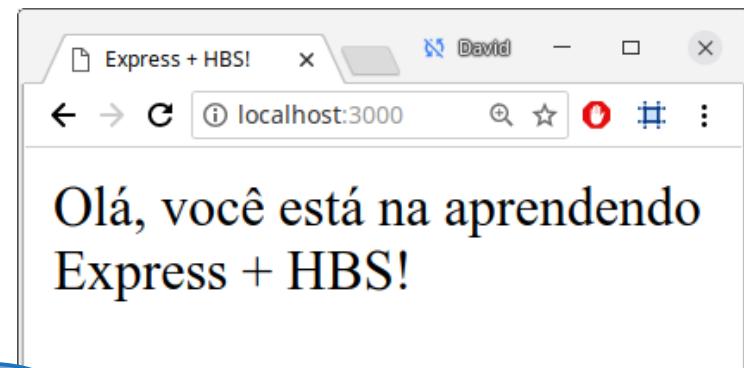
- O Express adiciona o método **render()** ao objeto **response (res)**, usado para renderizar o conteúdo de uma view

```
app.get('/', function(req, res) {  
  res.render('index', {  
    mensagem: 'Olá, você está aprendendo Express + HBS!',  
    layout: false  
  });  
});
```

Objeto com dados para a view

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Express + HBS!</title>  
  </head>  
  <body>  
    {{{message}}}  
  </body>  
</html>
```

Arquivo **views/index.handlebars**



Engine Handlebars

- Também é possível usar estruturas **if** com o **Handlebars**

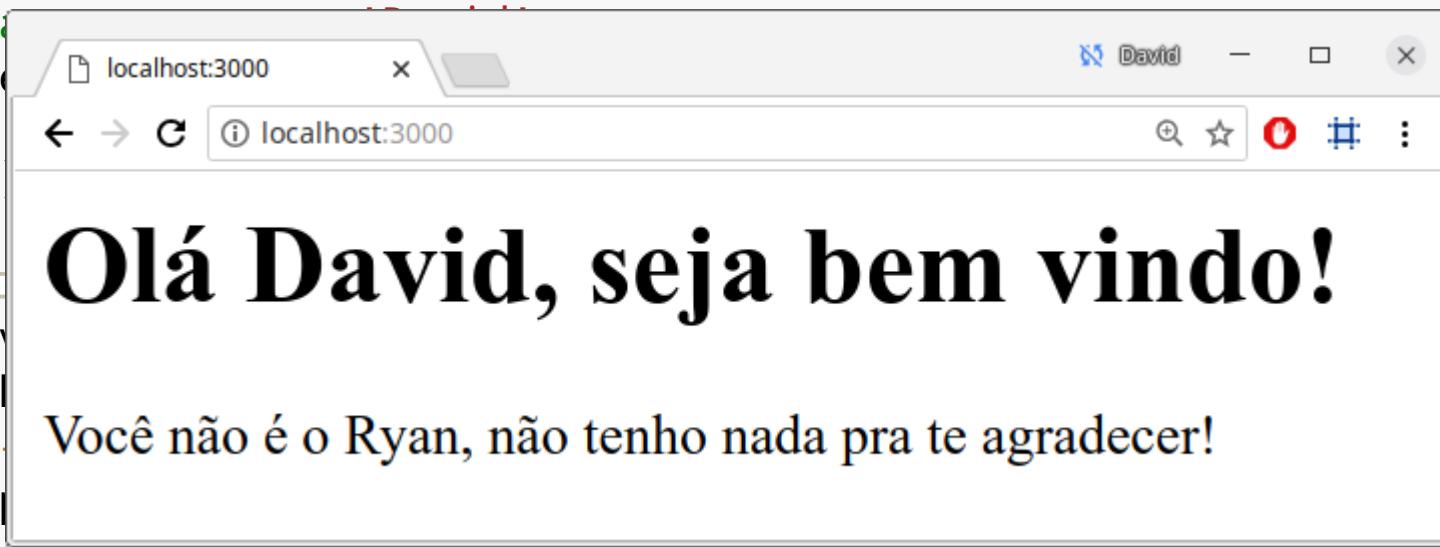
```
app.get('/', function(req, res) {  
  var username = 'David';  
  res.render('index', {  
    username:username, isRyan:(username=='Ryan'), layout:false  
  });  
});
```

```
<div class="container">  
  <h1>Olá {{username}}, seja bem vindo!</h1>  
  {{#if isRyan}}  
    <p>Obrigado Ryan, pelo Node.js!</p>  
  {{else}}  
    <p>Você não é o Ryan, não tenho nada pra te agradecer!</p>  
  {{/if}}  
</div>
```

Engine Handlebars

- Também é possível usar estruturas **if** com o **Handlebars**

```
app.get('/', function(req, res) {  
  res.render('index', {  
    nome: 'David'  
  });  
  
  <div>  
    <h1>Olá David, seja bem vindo!</h1>  
    <p>Você não é o Ryan, não tenho nada pra te agradecer!</p>  
    <{{else}}>  
      <p>Você não é o Ryan, não tenho nada pra te agradecer!</p>  
    <{{/if}}>  
  </div>
```



Engine Handlebars

- Também é possível usar percorrer um vetor com **#each**

```
app.get('/', function(req, res) {  
  var profes = [  
    { nome: 'David Fernandes', sala: 1238 },  
    { nome: 'Horácio Fernandes', sala: 1233 },  
    { nome: 'Edleno Moura', sala: 1236 },  
    { nome: 'Elaine Harada', sala: 1231 }  
  ];  
  res.render('index', { professores: profes, layout: false });  
});
```

```
<div class="container">  
  <h1>Alguns professores do IComp/UFAM</h1>  
  {{#each professores}}  
    <li>{{nome}} - sala {{sala}}</li>  
  {{/each}}  
</div>
```

Engine Handlebars

- Também é possível usar percorrer um vetor com **#each**

```
app.get('/', function(req, res) {  
  var profes = [  
    { nome: 'David Fernandes', sala: 1238 },  
    { nome: 'Horácio Fernandes', sala: 1233 },  
    { nome: 'Edleno Moura', sala: 1236 },  
    { nome: 'Elaine Harada', sala: 1231 }  
  ];  
  
  res.render('index', { profes: profes });  
});  
  
<div>  
  <ul>  
    <li>{{nome}} - sala {{sala}}</li>  
  </ul>  
<{/each}>  
</div>
```



A screenshot of a web browser window titled "David" showing the rendered output of the Handlebars template. The page title is "Alguns professores do IComp/UFAM". The content is a bulleted list of professor names and their respective room numbers:

- David Fernandes - sala 1238
- Horácio Fernandes - sala 1233
- Edleno Moura - sala 1236
- Elaine Harada - sala 1231

Helpers Handlebars

- Um dos recursos mais importantes do Handlebars é a possibilidade de criar **helpers customizados**

```
app.engine('handlebars', handlebars({
  helpers: require(__dirname + '/views/helpers/helpers.js')
}));
```

```
// Arquivo /views/helpers/helpers.js

const toLower = function(value) {
  return value.toLowerCase();
}

const toUpper = function(value) {
  return value.toUpperCase();
}

module.exports = { toLower, toUpper };
```

Helpers Handlebars

- A partir deste momento, as funções **toLower** e **toUpper** podem ser usadas nas views

```
app.get('/', function(req, res) {  
  var username = 'David';  
  res.render('index', {  
    username:username, isRyan:(username=='Ryan'), layout:false  
  });  
});
```

```
<div class="container">  
  <h1>Olá {{toUpper username }}, seja bem vindo!</h1>  
  {{#if isRyan}}  
    <p>Obrigado Ryan, pelo Node.js!</p>  
  {{else}}  
    <p>Você não é o Ryan, não tenho nada pra te agradecer!</p>  
  {{/if}}  
</div>
```

Helpers Handlebars

- A partir deste momento, as funções **toLower** e **toUpper** podem ser usadas nas views

```
app.get('/', function(req, res) {
```

```
    res.send('Olá DAVID, seja bem vindo!')
```

```
})
```

```
<
```

```
Você não é o Ryan, não tenho nada pra te agradecer!
```

```
    {{else}}
```

```
    <p>Você não é o Ryan, não tenho nada pra te agradecer!</p>
```

```
    {{/if}}
```

```
</div>
```

Helpers Handlebars

- A partir deste momento, as funções **toLower** e **toUpper** podem ser usadas nas views

```
app.get('/', function(req, res) {
```

O Handlebars possui uma série de pacotes de helpers que podem ser instalados via NPM, tal como o **handlebars-helpers**

```
}
```

Olá DAVID, seja bem vindo!

```
<
```

Você não é o Ryan, não tenho nada pra te agradecer!

```
  {{else}}
  <p>Você não é o Ryan, não tenho nada pra te agradecer!</p>
  {{/if}}
</div>
```

My Chess App David

localhost:3000/sobre

O Jogo de Xadrez



O xadrez é um jogo de tabuleiro, de caráter competitivo, disputado entre dois participantes. Cada um é representado por peças de cores opostas, geralmente são utilizadas pretas e brancas. O objetivo do jogo é conquistar o "rei" de seu adversário.

Para jogar é necessário um tabuleiro composto por oito colunas e oito linhas, o que resulta em 64 casas possíveis para a mobilidade das peças. As peças são compostas de oito peões, duas torres, dois cavalos,

Exercício 1: Crie uma app em Express contendo uma única página **about**, que deverá conter informações sobre o jogo que está sendo implementado na disciplina.

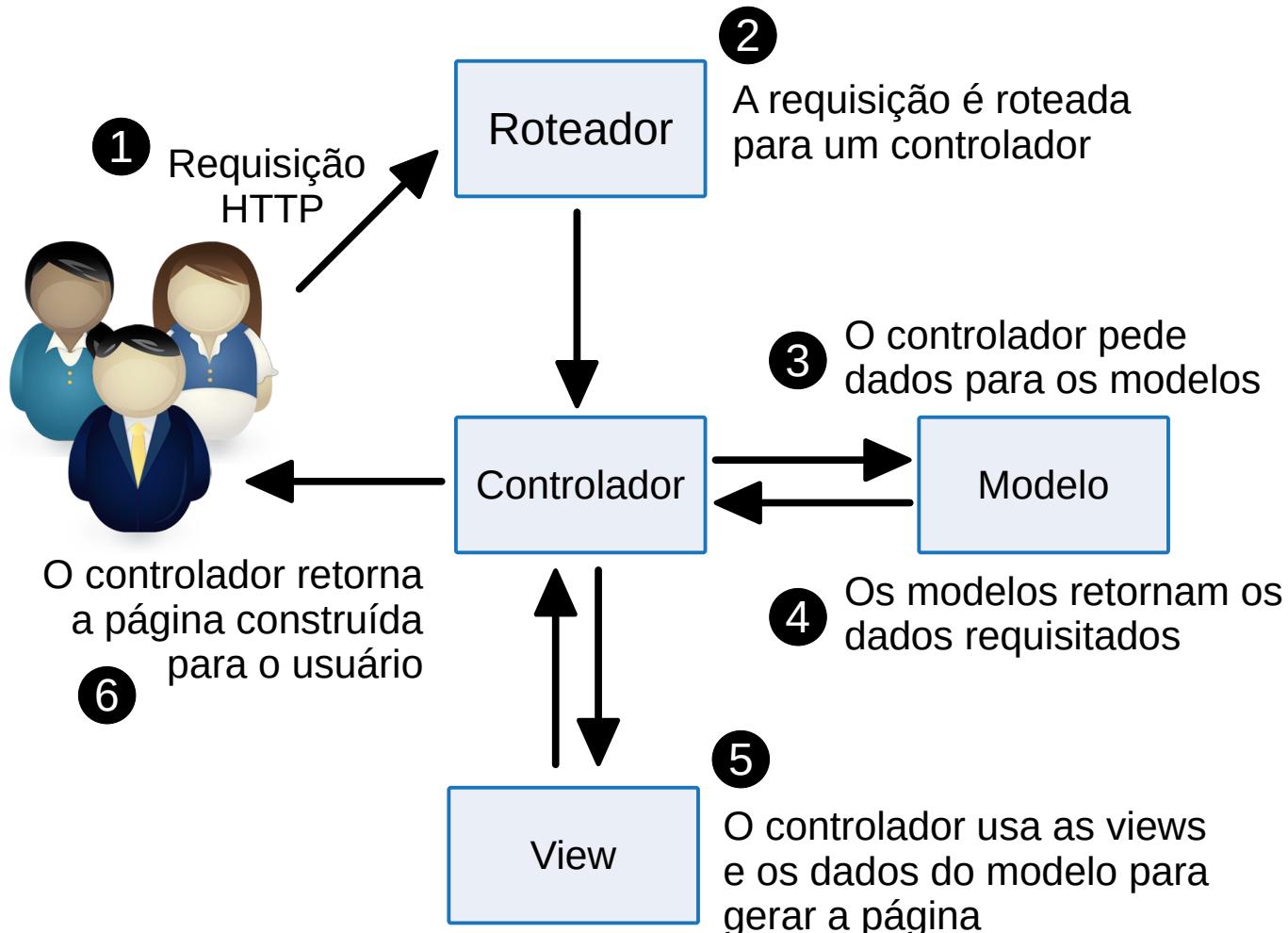
A página deverá gerar logs com o middleware **Morgan**, deverá ter um módulo de rotas separado e usar a engine **HandleBars**.



Movendo o Express para o MVC

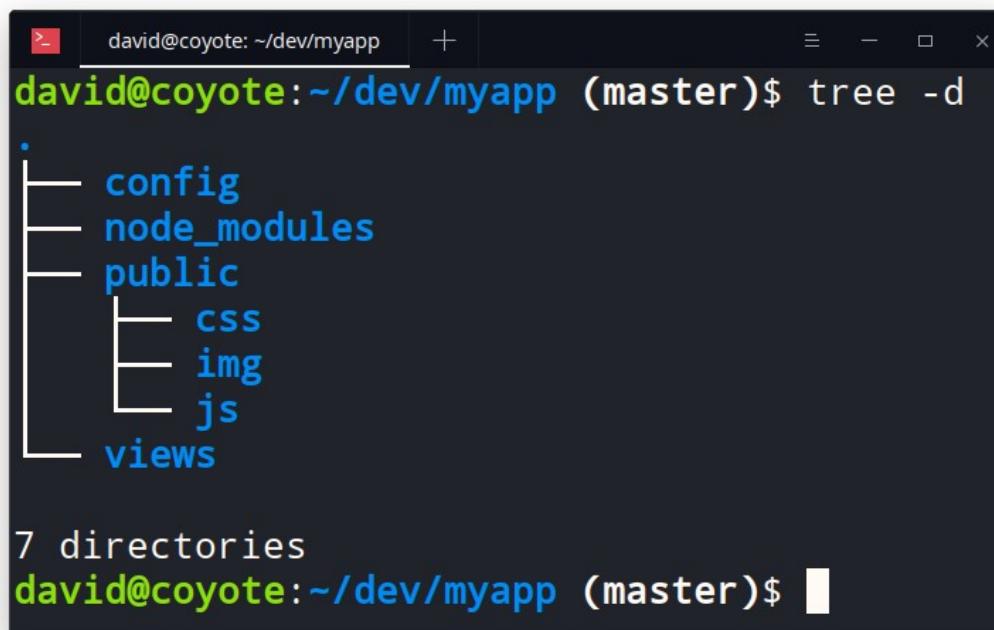
- O MVC é um padrão de arquitetura de software que separa a aplicação em 3 camadas distintas:
 - **Modelos**, responsáveis pela leitura e escrita dos dados provenientes do SGBD utilizado pela aplicação
 - **Visões**, responsáveis por gerar o conteúdo HTML que será enviado para o usuário para que esse possa interagir com a aplicação
 - **Controladores**, responsáveis por responder as requisições dos usuários, fazendo uso dos modelos e apresentando os resultados através das views

Movendo o Express para o MVC



Movendo o Express para o MVC

- Até este momento, nossa aplicação possui os diretórios **config** (com as rotas), **views** e **public** (para os assets)
- No entanto, ela ainda não possui nenhum diretório específico para **controladores** e **modelos**



```
david@coyote: ~/dev/myapp (master)$ tree -d
.
├── config
├── node_modules
├── public
│   ├── css
│   ├── img
│   └── js
└── views

7 directories
david@coyote:~/dev/myapp (master)$
```

Movendo o Express para o MVC

- Desta forma, os diretórios **models** e **controllers** precisam ser criados para completar a arquitetura MVC
- No entanto, ao invés de criar novos diretórios na raiz da aplicação, podemos seguir os seguintes passos:
 - Criamos um novo diretório chamado **app** no diretório raiz
 - Dentro de **app**, criamos os diretórios **models** e **controllers**
 - Movemos o diretório **views** para o diretório **app**

Movendo o Express para o MVC

- Desta forma, os diretórios **models** e **controllers** precisarão ser criados para completar a arquitetura MVC



```
david@coyote:~/dev/myapp (master)$ tree -d
.
├── config
├── node_modules
└── public
    ├── css
    ├── img
    └── js
└── views

7 directories
david@coyote:~/dev/myapp (master)$
```

```
david@coyote:~/dev/myapp (master)$ tree -d
.
├── app
│   ├── controllers
│   ├── models
│   └── views
├── config
├── node_modules
└── public
    ├── css
    ├── img
    └── js

10 directories
david@coyote:~/dev/myapp (master)$
```

Movendo o Express para o MVC

- Como a localização do diretório **views** foi alterada, será preciso avisar o Express dessa mudança
- Para isso, basta encontrar a seguinte linha no arquivo **app.js**:

```
app.set('views', __dirname + '/views');
```

- E então mudá-la para:

```
app.set('views', __dirname + '/app/views');
```

Movendo o Express para o MVC

- Uma aplicação pode conter vários controladores, mas por enquanto vamos criar apenas um controlador chamado **main**, que irá responder pelas rotas / e /sobre
- Primando pela organização do código, vamos criar um diretório **/views/main** que irá conter as views do controlador main



```
david@coyote: ~/dev/myapp/api/views/main$ ls
index.handlebars  sobre.handlebars
david@coyote:~/dev/myapp/api/views/main$
```

A screenshot of a terminal window titled "david@coyote: ~/dev/myapp/api/views/main". The window shows the command "ls" being run, which lists two files: "index.handlebars" and "sobre.handlebars". The terminal has a dark background with light-colored text and standard window controls at the top.

Movendo o Express para o MVC

- Uma aplicação pode conter vários controladores, mas por enquanto Em um primeiro momento, vamos adotar que irá gerenciar o main, o código abaixo para as views **main/index.handlebars** e **main/sobre.handlebars**
- Primeiro /views/main/index.handlebars

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Chess App</title>
</head>
<body>
  <div>{{conteudo}}</div>
</body>
</html>
```

Movendo o Express para o MVC

- O **controlador main** será construído no formato de módulo, e irá conter as funções **index** e **sobre**

```
// Arquivo api/controllers/main.js
const index = (req, res) => {
  const conteudo = 'Página principal da aplicação';
  res.render('main/index', {
    conteudo: conteudo,
    layout: false
  });
};

const sobre = (req, res) => {
  const conteudo = 'Página sobre a aplicação';
  res.render('main/sobre', {
    conteudo: conteudo,
    layout: false
  });
};

module.exports = { index, sobre }
```

Movendo o Express para o MVC

- O **controlador main** será construído no formato de módulo, e irá conter as funções **index** e **sobre**

```
// Arquivo api/controllers/main.js
const index = (req, res) => {
  const conteudo = 'Página principal da aplicação';
```

Note que um controlador é um conjunto de funções, também conhecidas como **actions**. Cada action é responsável por atender uma dada requisição dos usuários. No nosso exemplo, **Index** e **sobre** são **actions** do **controlador main**.

```
const sobre = (req, res) => {
  const conteudo = 'Página sobre a aplicação';
  res.render('main/sobre', {
    conteudo: conteudo,
    layout: false
  });
};

module.exports = { index, sobre }
```

Movendo o Express para o MVC

- O próximo passo é criar o arquivo de **rotas**, que deverá mapear as **actions** dos controladores com as requisições de usuários

```
const express = require('express');
const router = express.Router();
const mainController = require('../app/controllers/main');

router.get('/', mainController.index);
router.get('/sobre', mainController.sobre);

module.exports = router;
```

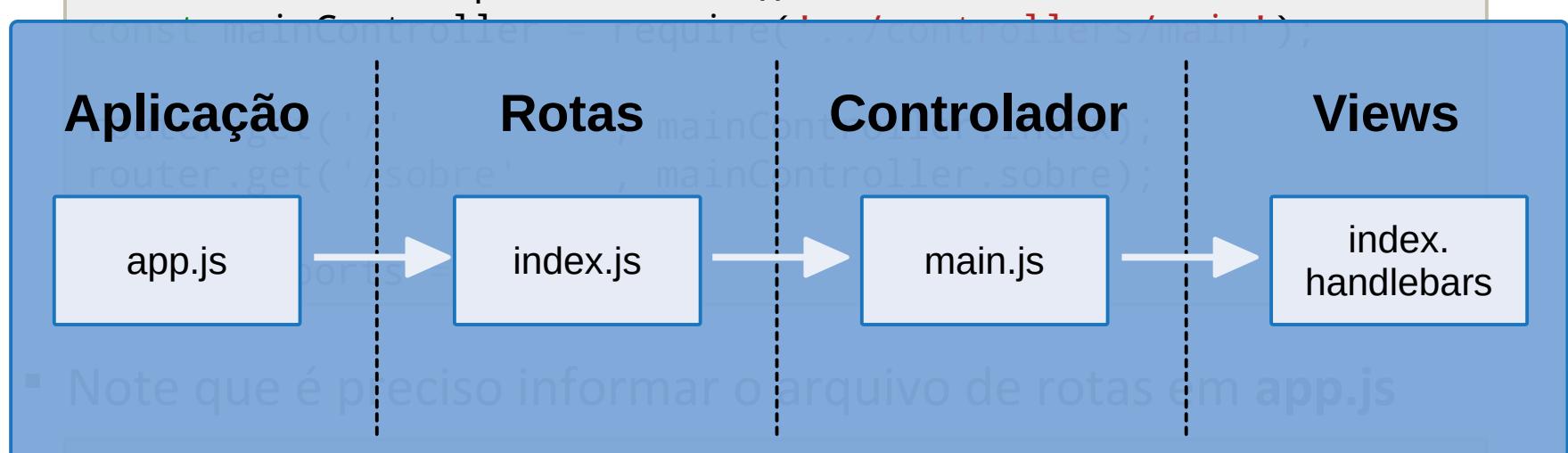
- Note que é preciso informar o arquivo de rotas em **app.js**

```
const router = require("./config/routes");
app.use(router);
```

Movendo o Express para o MVC

- O próximo passo é criar o arquivo de **rotas**, que deverá mapear as **actions** dos controladores com as requisições de usuários

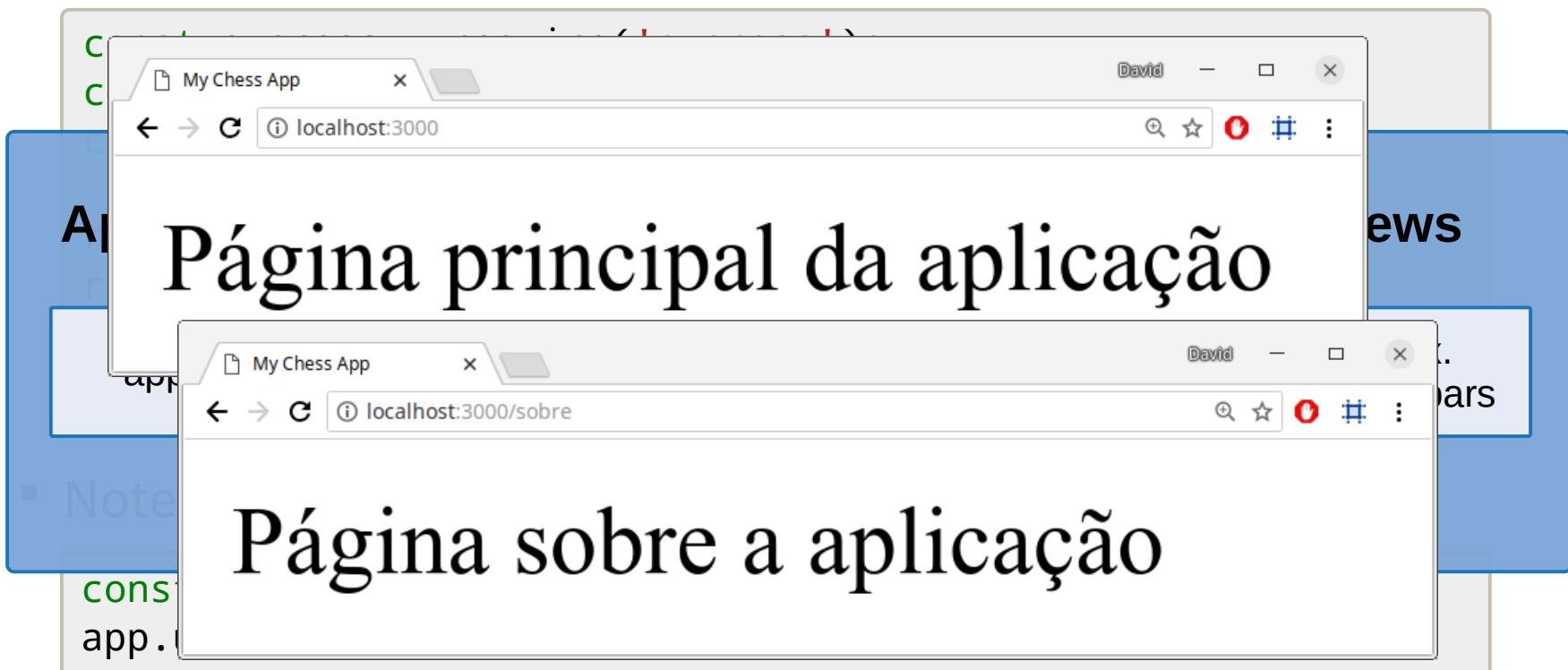
```
const express = require('express');
const router = express.Router();
```



```
const router = require("./config/routes");
app.use(router);
```

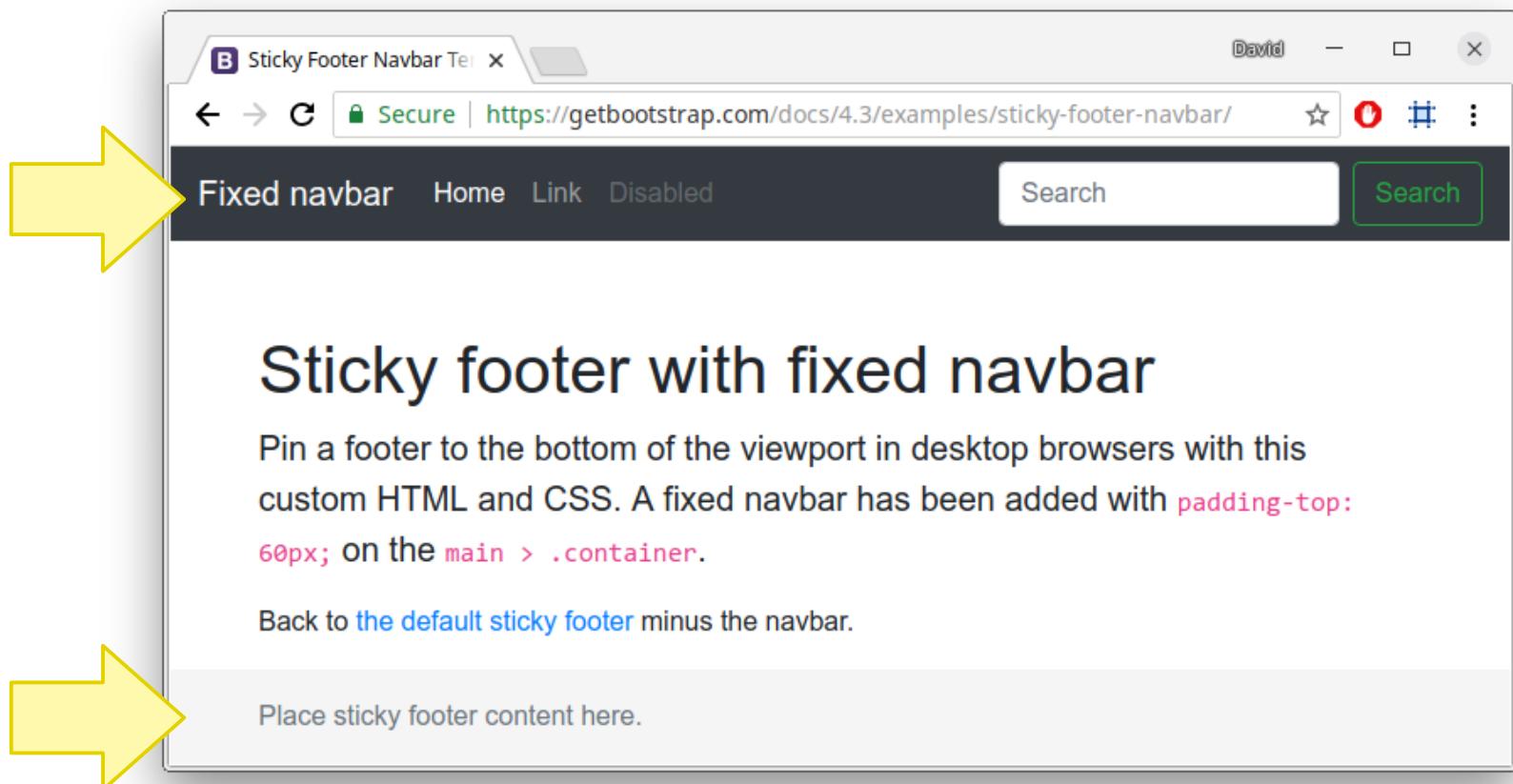
Movendo o Express para o MVC

- O próximo passo é criar o arquivo de **rotas**, que deverá mapear as **actions** dos controladores com as requisições de usuários



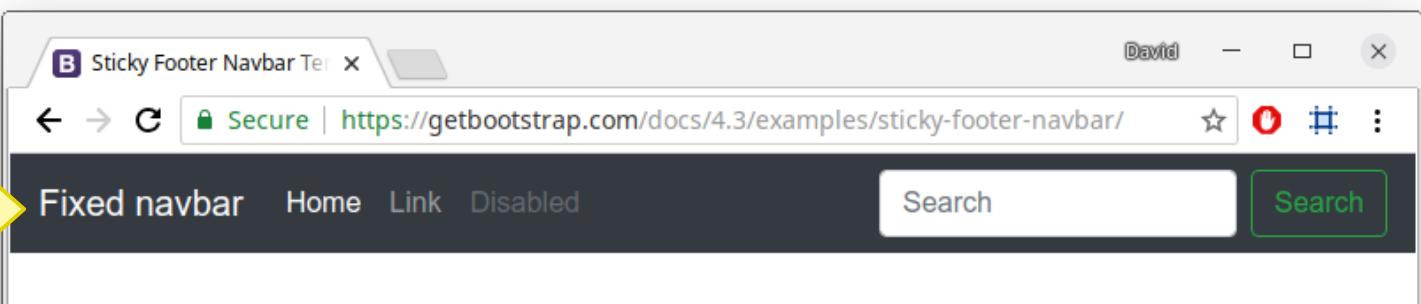
Layout X Views

- Normalmente, as páginas de uma mesma app possuem muito código HTML em comum, como **menus, headers e footers**



Layout X Views

- Normalmente, as páginas de uma mesma app possuem muito código HTML em comum, como **menus, headers e footers**



A screenshot of a web browser window titled "Sticky Footer Navbar Test". The address bar shows a secure connection to "https://getbootstrap.com/docs/4.3/examples/sticky-footer-navbar/". The page content includes a dark header with "Fixed navbar" and "Home Link Disabled" buttons, a search bar, and a large blue callout box containing text about layout. A yellow arrow points from the left towards the browser window. Another yellow arrow points from the bottom left towards the blue callout box.

Esse conteúdo comum, que está presente na maioria das páginas da aplicação, é o que chamamos de **layout**.

custom HTML and CSS. A fixed navbar has been added with `padding-top: 60px;` on the `main > .container`.

Back to [the default sticky footer](#) minus the navbar.

Place sticky footer content here.

Layout X Views

- Por padrão, o **layout** da aplicação deverá ser definido no arquivo **<views>/layouts/main.handlebars**
- No entanto, é possível mudar o arquivo de layout padrão através da função `app.engine`

```
app.engine('handlebars', handlebars({
  layoutsDir: __dirname + '/app/views/diretorio_layouts',
  defaultLayout: 'arquivo_layout',
}))
```

- Também é possível definir layouts específicos para determinadas actions

```
const index = (req, res) => {
  res.render('main/index', { layout: 'main2' });
};
```

Layout X Views

- Os layouts geralmente contém as tags `<html>`, `<head>`, `<body>`, `<style>` e `<meta>` da aplicacão
- Para informar o local onde o código das views será inserido dentro do layout, usa-se o código `{{{body}}}`
- Em outras palavras, o Express irá substituir o código acima pelo conteúdo da view no momento da geracão da página

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Chess App</title>
</head>
<body>
  {{{body}}}
</body>
</html>
```

Local onde as views serão carregadas

Layout X Views

- Os layouts geralmente contém as tags `<html>`, `<head>`,

```
<!DOCTYPE html>
<html lang="en">
<head>
```

A partir deste momento, as views não precisarão mais ter o código HTML já incluído no layout

- Para informar o local onde o código das views será carregado, adicione o diretório de layout na view
- No nosso exemplo, o código das views passa a ser somente

```
<div>{{conteúdo}}</div>
```

código `{{{body}}}`

```
</body>
</html>
```

- Em outras palavras, o Express irá substituir o código acima pelo conteúdo da view no momento da geração da página

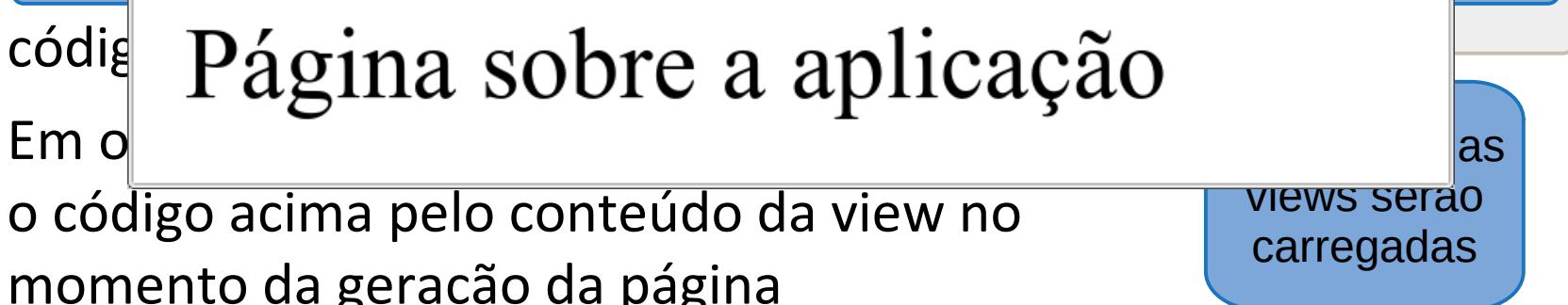
Local onde as views serão carregadas

Layout X Views

- Os layouts geralmente contêm



- as views



- Em outras palavras, o código acima pelo conteúdo da view no momento da geração da página

views serão carregadas

Assets

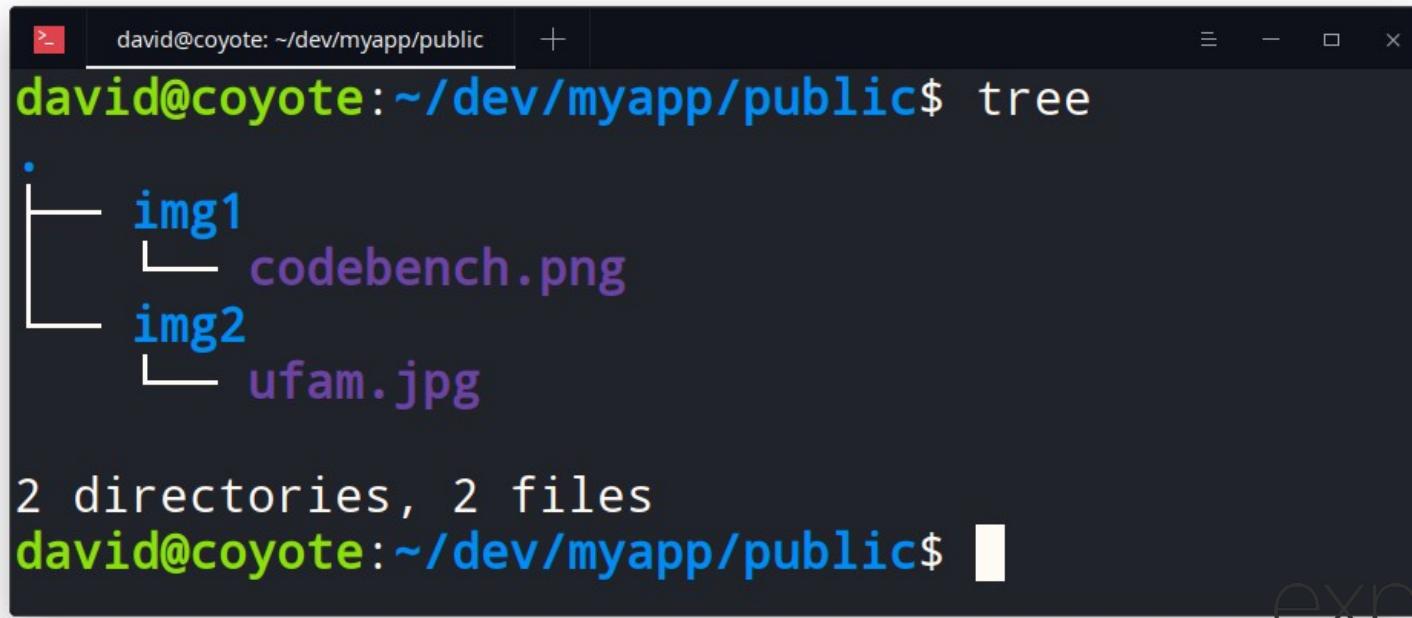
- Conforme já vimos, os assets são **arquivos estáticos** (js, css, img, etc) que são acessíveis pelos usuários da aplicação
- Para disponibilizar esses arquivos na aplicação, usamos o **middleware de arquivos estáticos** do Express
- Por exemplo, para disponibilizar todas as imagens do diretório **/public/img**, usamos o código abaixo:

```
app.use('/img', [
  express.static(__dirname + '/public/img')
]);
```

Assets

- Também é possível disponibilizar imagens de 2 ou mais diretórios através de um único path na URL

```
app.use('/img', [
  express.static(__dirname + '/public/img1'),
  express.static(__dirname + '/public/img2')
]);
```



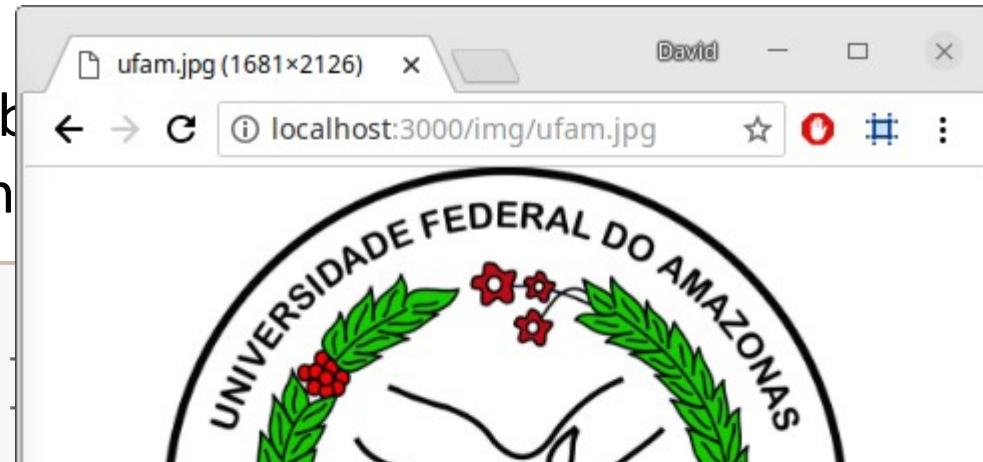
```
david@coyote:~/dev/myapp/public$ tree
.
├── img1
│   └── codebench.png
└── img2
    └── ufam.jpg

2 directories, 2 files
david@coyote:~/dev/myapp/public$
```

Assets

- Também é possível disponibilizar arquivos de diretórios através de um único endpoint

```
app.use('/img', [
  express.static(__dirname + '/img'),
  express.static(__dirname + '/public')
]);
```



2 directories, 2 files

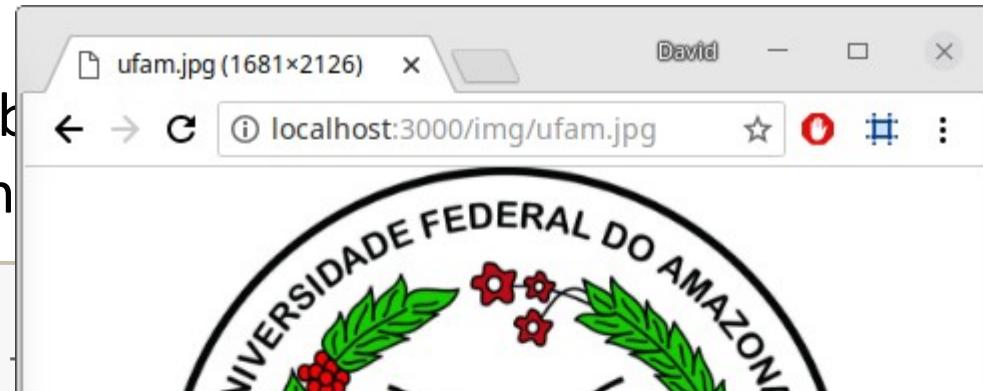
david@coyote:~/dev/myapp/public\$

express JS

Assets

- Também é possível disponibilizar arquivos de diretórios através de um único endpoint

```
app.use('/img', [  
  express.static(__dirname + '/img')  
]);
```



Se houver duas imagens com o mesmo nome nos dois diretórios informados, o Express irá retornar a imagem que encontrar primeiro, seguindo a sequência de diretórios informada

dav]\$



CODEBENCH

2 directories, 2 files

david@coyote:~/dev/myapp/public\$

express JS

Assets

- Também é possível disponibilizar arquivos de diretórios através de um único endpoint

```
app.use('/img', [  
  express.static(__dirname + '/img')])
```



Se houver duas imagens com o mesmo nome nos dois diretórios

Conforme veremos mais tarde, a inclusão de **arquivos JS** segue a mesma sequência de passos



CODEBENCH

2 directories, 2 files

david@coyote:~/dev/myapp/public\$

express **JS**

Assets

- A inclusão de **arquivos CSS** pode ser feita através do middleware de arquivos estáticos do Express
- No entanto, em se tratando de arquivos de estilos, uma opção melhor seria o uso de compiladores de estilos, tal como o **SASS**





- O SASS – Syntactically Awesome Stylesheets – é uma linguagem de **folha de estilos dinâmica** usada para simplificar a criação de código CSS
- Ela permite a criação de código CSS a partir de variáveis, operações, funções e mixins
- Instalação do SASS no sistema Linux:

```
$ sudo npm install sass -g
```

- Para compilar códigos SASS em CSS, usamos:

```
$ sass estilos.scss estilos.css
```

Variáveis SASS

- SASS permite a definição de **variáveis** que podem ser utilizadas por toda a folha de estilos

SCSS

```
$color: #4D926F;  
  
#header {  
  color: $color;  
}  
  
h2 {  
  color: $color;  
}
```

CSS Compilado

```
#header {  
  color: #4D926F;  
}  
  
h2 {  
  color: #4D926F;  
}
```

Variáveis SASS

- SASS permite a definição de **variáveis** que podem ser utilizadas por todo o SCSS

SCSS

```
david@coyote:~/dev/myapp/public/scss$ cat variaveis.scss
$color: #4D926F;

$header {
  color: $color;
}
#header h2 {
  color: $color;
}
h2 {
  #header {
    color: $color;
  }
  h2 {
    color: #4D926F;
  }
}

david@coyote:~/dev/myapp/public/scss$ sass variaveis.scss
david@coyote:~/dev/myapp/public/scss$
```

Mixins *Sass*

- Os Mixins permitem a definição de estilos que podem ser reutilizados em toda a folha de estilo

SCSS

```
@mixin bordered ($wth) {  
    border-top: dotted $wth black;  
    border-bottom: solid $wth black;  
}  
  
#menu a {  
    color: #111;  
    @include bordered(2px);  
}  
  
.post a {  
    @include bordered(1px);  
}
```

CSS Compilado

```
#menu a {  
    color: #111;  
    border-top: dotted 2px black;  
    border-bottom: solid 2px black;  
}  
  
.post a {  
    border-top: dotted 1px black;  
    border-bottom: solid 1px black;  
}
```

Regras Aninhadas



- Com o SCSS é possível aninhar seletores dentro de outros seletores para especificar a ancestralidade dos seletores

SCSS

```
#header {  
  color: black;  
  .navigation {  
    font-size: 12px;  
  }  
  .logo {  
    width: 300px;  
  }  
}
```

CSS Compilado

```
#header {  
  color: black;  
}  
#header .navigation {  
  font-size: 12px;  
}  
#header .logo {  
  width: 300px;  
}
```

Operações



- O SCSS também conta com operações para somar, subtrair, dividir e multiplicar valores de propriedades

SCSS

```
$the-border: 1px;  
  
#header {  
    border-left: $the-border;  
    border-right: $the-border * 2;  
}
```

CSS Compilado

```
#header {  
    border-left: 1px;  
    border-right: 2px;  
}
```

Outras Funções



- Além das funcionalidades mostradas, o SASS possui muitas outras vantagens sobre o CSS

Comentários

```
/* Isto é um bloco  
de comentários! */  
$var: red;  
  
// Comentário em uma linha  
$var: white;
```

Escopo

```
$var: red;  
#page {  
    $var: white;  
    #header {  
        color: $var; // white  
    }  
}
```

Importação de estilos

```
@import "library"; // library.scss  
@import "typo.css";
```

Funções de manipulação de cores

```
lighten($color, 10%);  
darken($color, 10%);  
  
saturate($color, 10%);  
desaturate($color, 10%);  
  
opacity($color, 0.5);  
transparentize($color, 0.5);
```

Adicionando SASS no Express

- Para usarmos o compilador SASS no Express, podemos recorrer ao pacote **node-sass-middleware**

```
$ npm install node-sass-middleware
```

- A configuração desse middleware é simples

```
const sass = require('node-sass-middleware');

app.use(sass({
  src: __dirname + '/public/scss',
  dest: __dirname + '/public/css',
  outputStyle: 'compressed',
  prefix: '/css',
}));

app.use('/css', express.static(__dirname + '/public/css'));
```

Adicionando SASS no Express

- Para usarmos o compilador SASS no Express, podemos

```
david@coyote: ~/dev/myapp$ cat public/scss/styles.scss
$color: red;

div {
  color: $color;
}

david@coyote:~/dev/myapp$ cat public/scss/main.scss
@import "styles.scss";
david@coyote:~/dev/myapp$ 

  dest: __dirname + '/public/css',
  outputStyle: 'compressed',
  prefix: '/css',
});

app.use('/css', express.static(__dirname + '/public/css'));
```

Adicionando SASS no Express

```
david@coyote: ~/dev/myapp$ cat app/views/layouts/main.handlebars
<!DOCTYPE html>
<html lang="en">
$ <head>
  <meta charset="UTF-8">
d  <link rel="stylesheet" href="/css/main.css">
  <title>My Chess App</title>
} </head>
d <body>
@  {{body}}
d </body>
david@coyote:~/dev/myapp$ cat app/views/main/index.handlebars
<div>{{content}}</div>
david@coyote:~/dev/myapp$
```

});

```
app.use('/css', express.static(__dirname + '/public/css'));
```

Adicionando SASS no Express

The image shows a terminal session and a browser window side-by-side.

Terminal:

```
david@coyote:~/dev/myapp$ cat app/views/layouts/main.handlebars
<!DOCTYPE html>
<html lang="en">
$ <head>
```

```
david@coyote:~/dev/myapp$ 
```

```
});
```

```
app.use('/css', express.static(__dirname + '/public/css'));
```

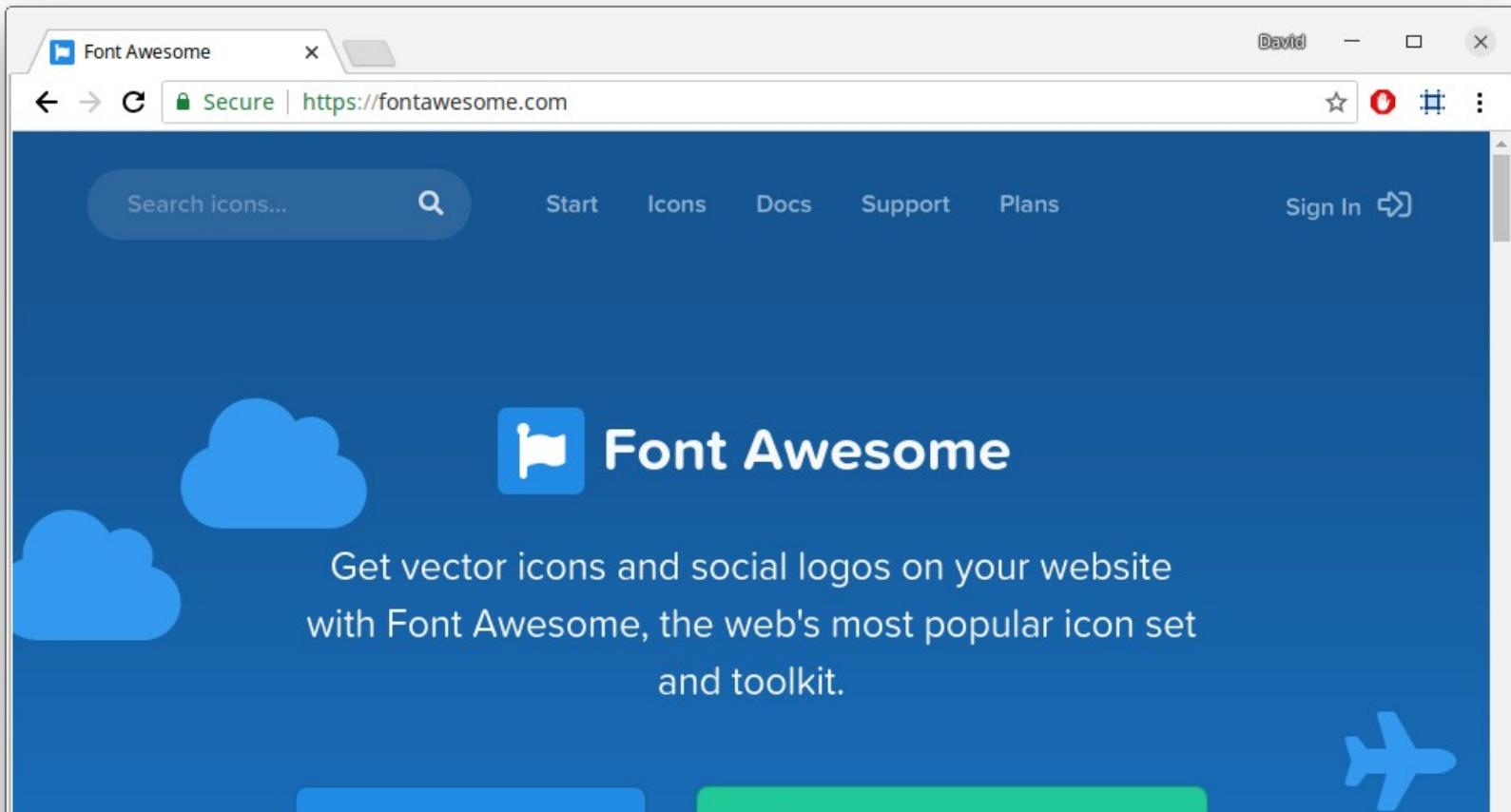
Browser:

My Chess App

localhost:3000

Página principal da aplicação

The browser window displays the main handlebars template. The title bar says "My Chess App" and the address bar says "localhost:3000". The page content is a large red header: "Página principal da aplicação".



Exercício 2: Instalar o pacote do projeto **Font Awesome**, e disponibilizar os ícones para uso em sua aplicação. O nome do pacote NPM é **@fortawesome/fontawesome-free**. Siga as orientações disponíveis na [página oficial](#).

github
Game

Adicionando o Bootstrap



- O próximo passo é adicionar o **Bootstrap** na sua aplicação

The screenshot shows the official Bootstrap website at getbootstrap.com. The page features a large hero section with the text "Build fast, responsive sites with Bootstrap". Below this, a sub-section explains that Bootstrap is a mobile-first front-end framework. A command-line instruction "\$ npm install bootstrap" is highlighted in a box. The footer includes links for "Get started" and "Download", along with version information ("Currently v5.1.3").

Bootstrap · The most popular front-end framework for developing responsive, mobile-first projects.

Build fast, responsive sites with Bootstrap

Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end framework.

\$ `npm install bootstrap`

and powerful JavaScript plugins.

Get started Download

Currently v5.1.3 · [v4.6.x docs](#) · [All releases](#)

Adicionando o Bootstrap



- Após a instalação, é necessário incluir os paths dos arquivos JS dos três pacotes no **middleware de arquivos estáticos**

```
app.use('/js', [
  express.static(__dirname + '/node_modules/bootstrap/dist/js/'),
  express.static(__dirname + '/public/js')
]);
```

- Além disso, é preciso incluir o arquivo **bootstrap.scss** no arquivo **/public/scss/main.scss**

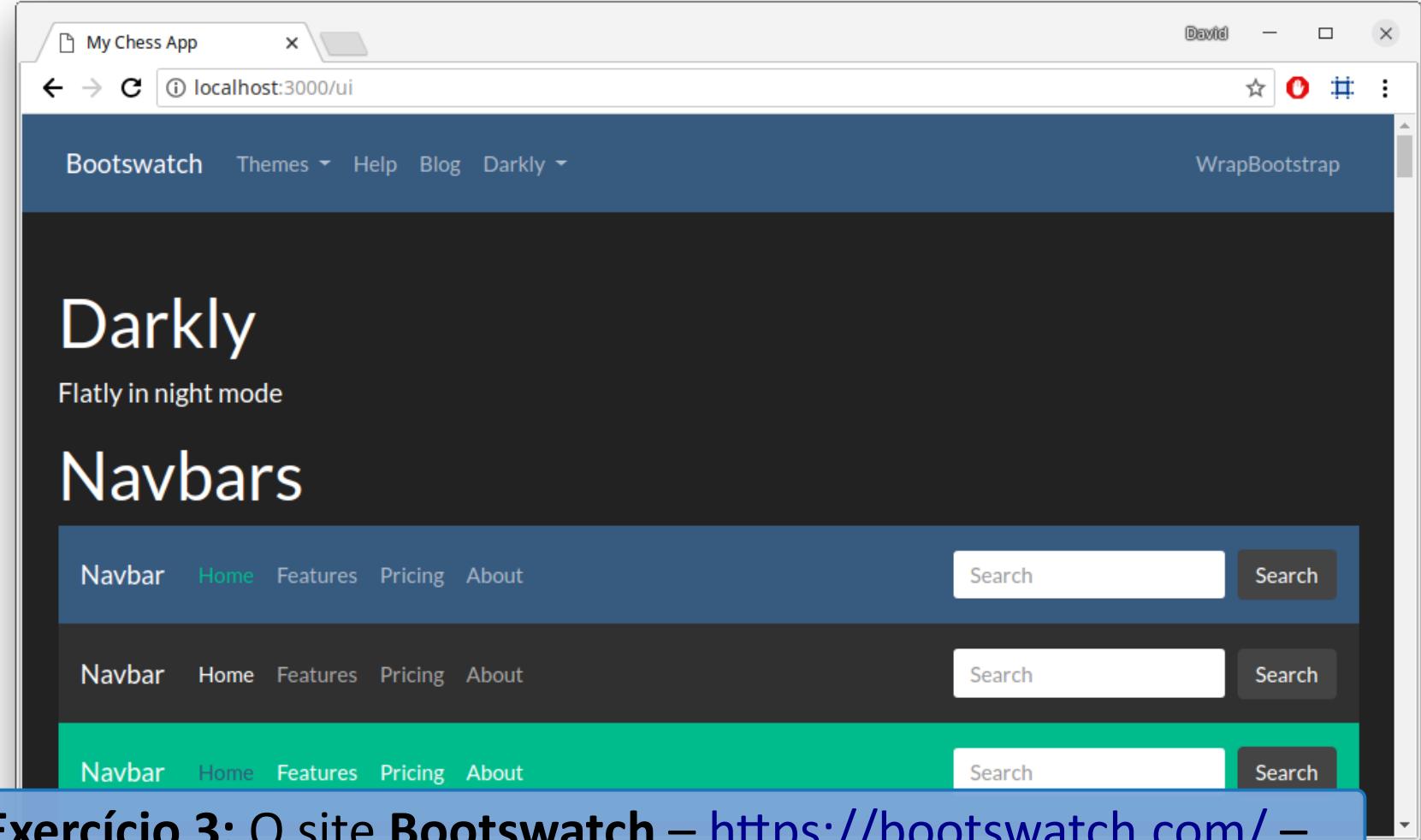
```
@import ".../node_modules/bootstrap/scss/bootstrap.scss";
```

Adicionando o Bootstrap



- Após isso, basta incluir o arquivo **bootstrap.bundle.js** no layout principal da aplicação

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Chess App</title>
  <link rel="stylesheet" href="/css/main.css">
</head>
<body>
  {{>body}}
  <script src="/js/bootstrap.bundle.js"></script>
</body>
</html>
```

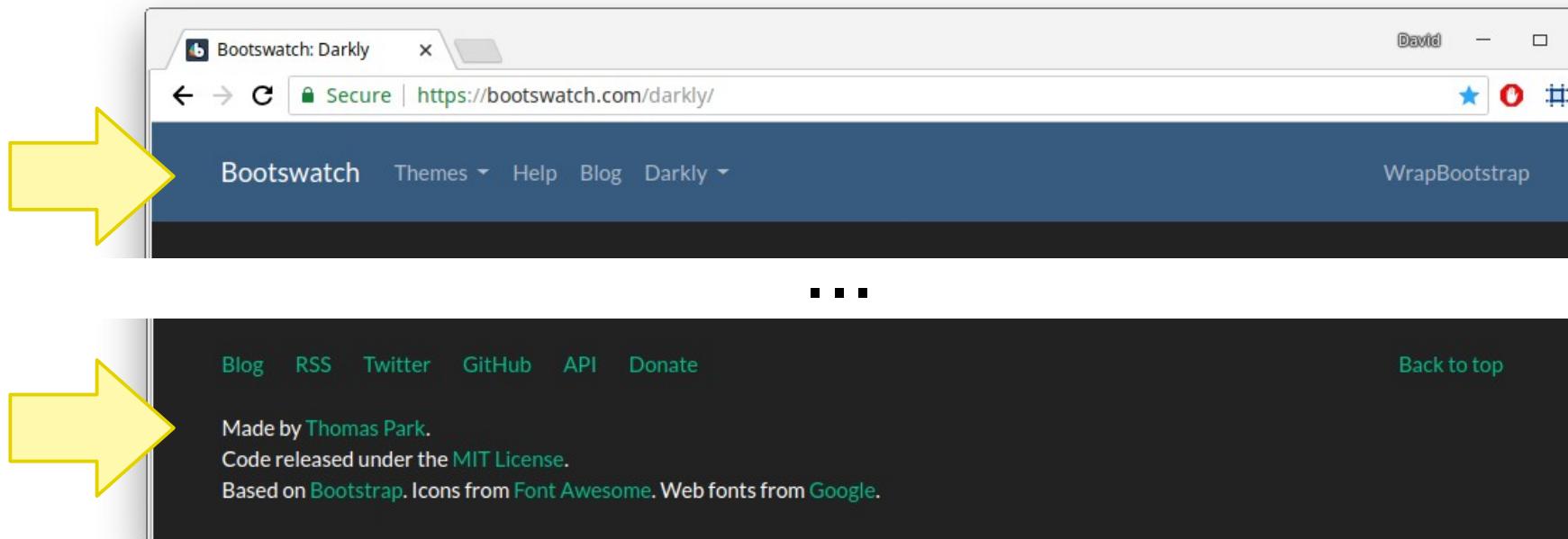


Exercício 3: O site **Bootswatch** – <https://bootswatch.com/> – possui um conjunto de temas baseados no **Bootstrap**. Vá até esse site, escolha um dos temas disponíveis e instale ele em sua aplicação.

github
Game

Dicas para o Exercício 3

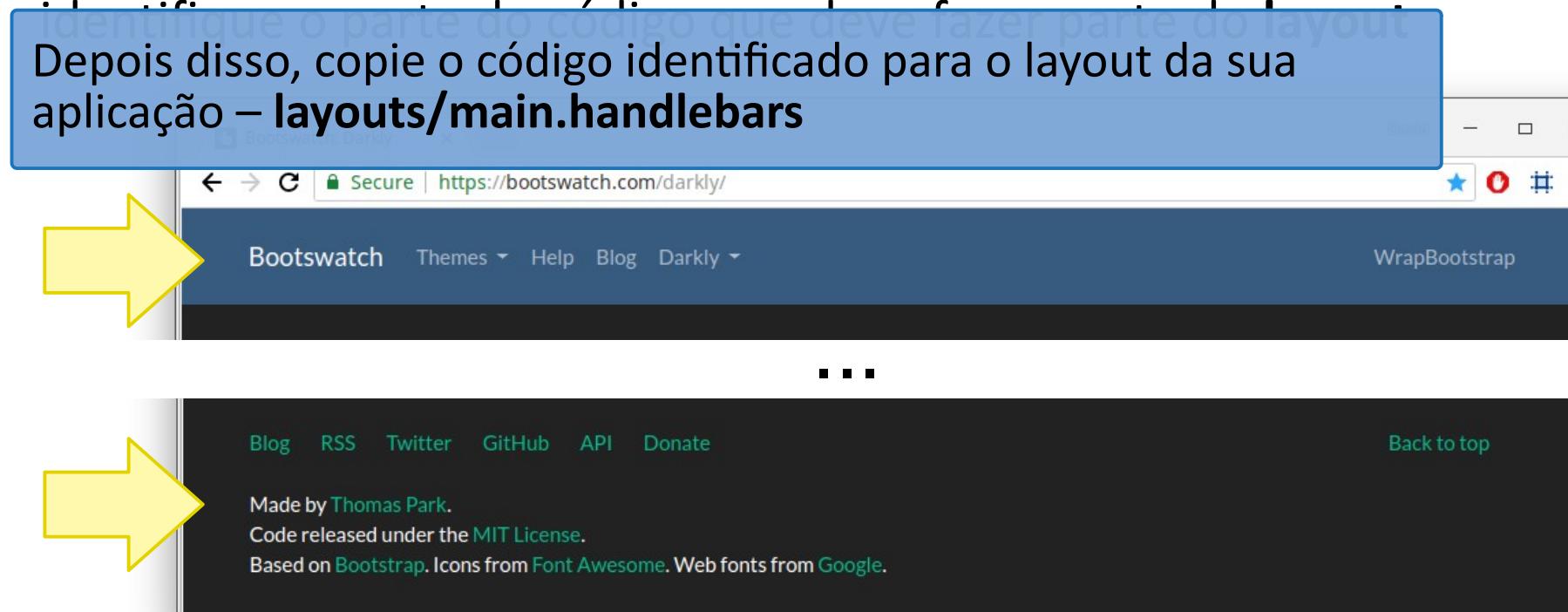
- Copie os arquivos `_bootswatch.scss` e `_variables.scss` para `/public/scss` e faça seus **imports** no arquivo `main.scss`
- Acesse o código-fonte do preview do tema escolhido, e identifique o parte do código que deve fazer parte do **layout**



Dicas para o Exercício 3

- Copie os arquivos `_bootswatch.scss` e `_variables.scss` para `/public/scss` e faça seus **imports** no arquivo `main.scss`
- Acesse o código-fonte do preview do tema escolhido, e

Depois disso, copie o código identificado para o layout da sua aplicação – `layouts/main.handlebars`



Dicas para o Exercício 3

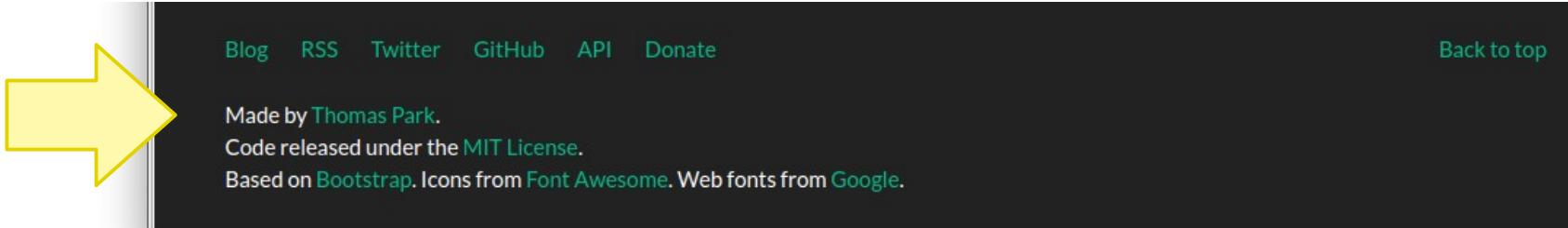
- Copie os arquivos `_bootswatch.scss` e `_variables.scss` para `/public/scss` e faça seus **imports** no arquivo `main.scss`
- Acesse o código-fonte do preview do tema escolhido, e identifique a parte do código que deve fazer parte do layout.

Depois disso, copie o código identificado para o layout da sua aplicação.

a) Adicionalmente, crie uma **action** chamada **ui** e copie em sua **view** todo o código fonte do preview que não faça parte do layout.

O objetivo dessa view é conter um conjunto de **elementos de interface** que você poderá usar na sua aplicação (Ctrl+c Ctrl+v)

...



Adicione no menu
as rotas
disponíveis em
sua aplicação

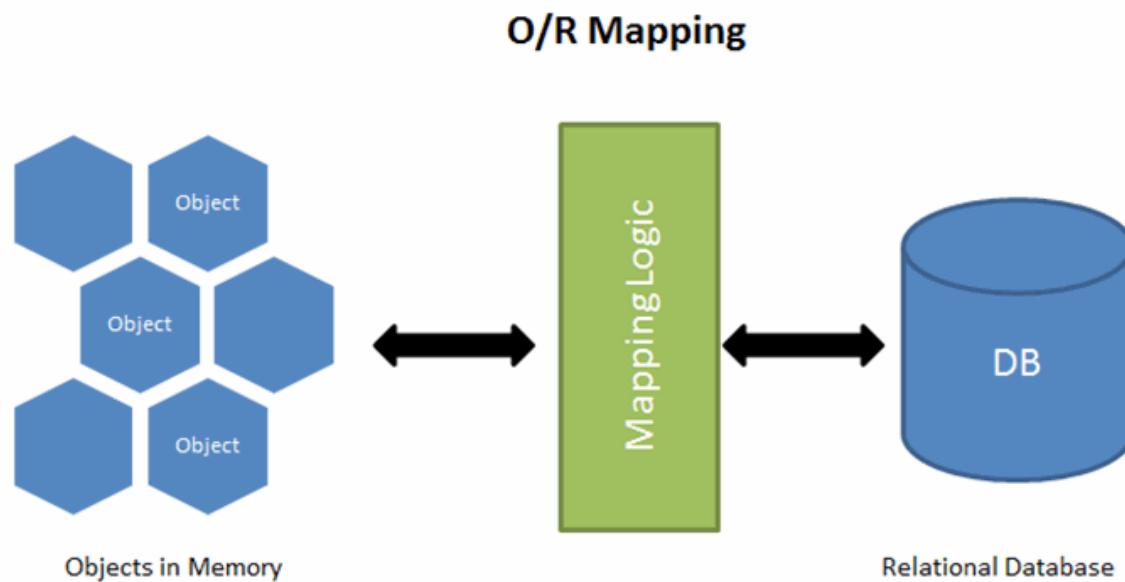


Exercício 4: Crie a rota `/game` e disponibilize o jogo Skifree
nesta rota. Note que para isso você precisará criar
uma nova action no controlador main, uma nova view, e
definir a rota no arquivo config/router.js

github
Game

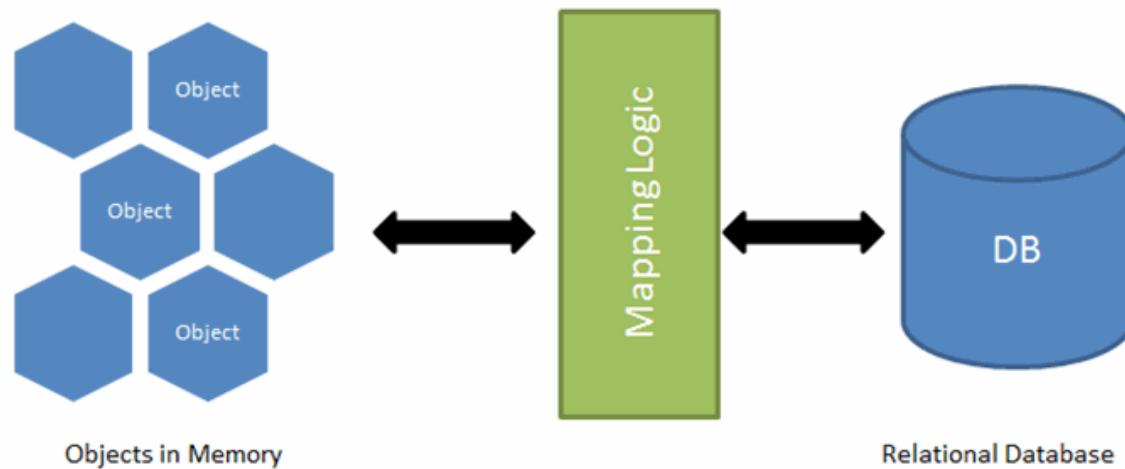
Object Relational Mapper - ORM

- ORM é uma técnica que permite **consultar e manipular dados** de um database usando o paradigma de orientação a objetos
- Desta forma, o acesso aos dados não é feito através da linguagem SQL, e sim através de objetos

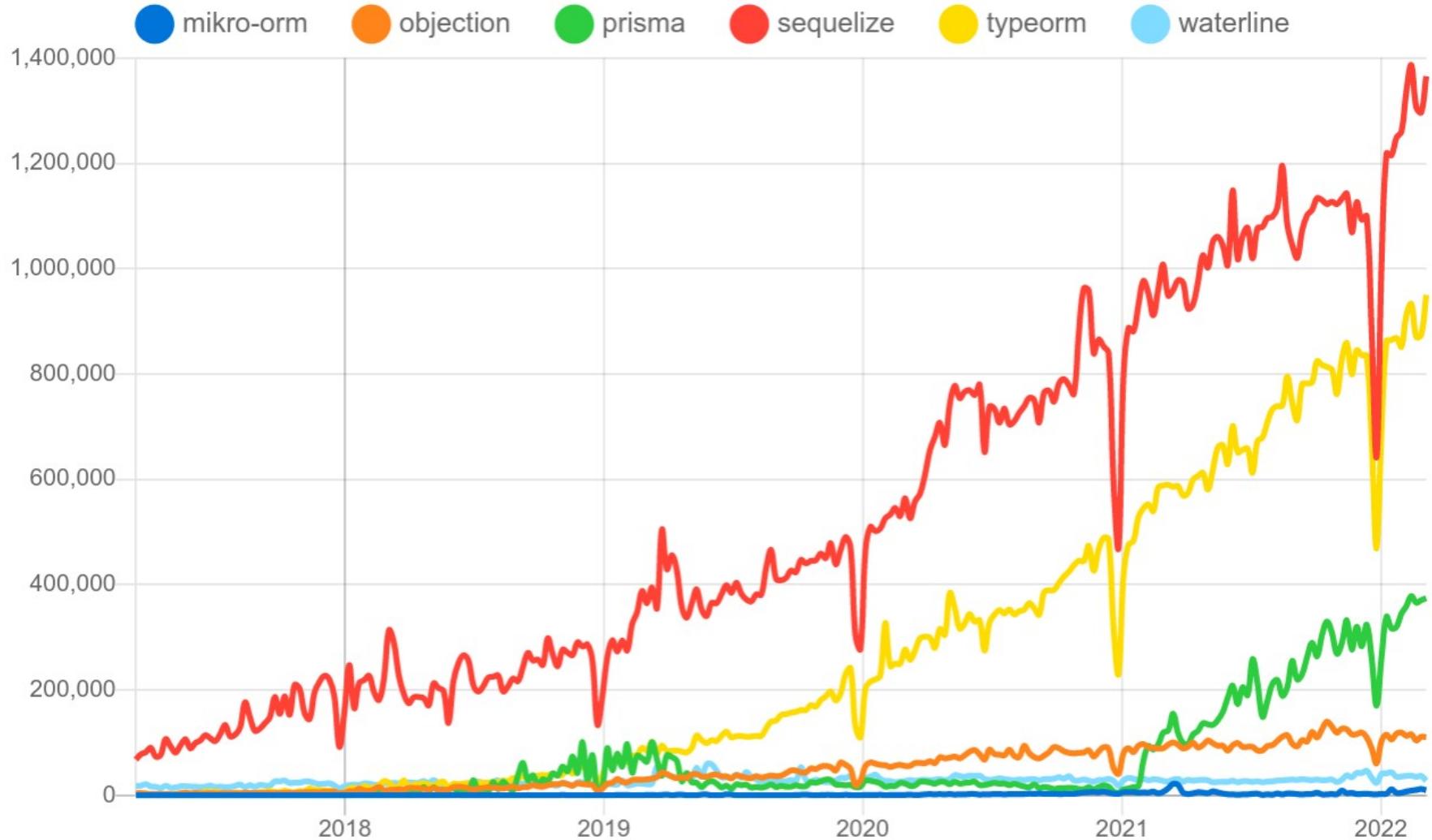


Object Relational Mapper - ORM

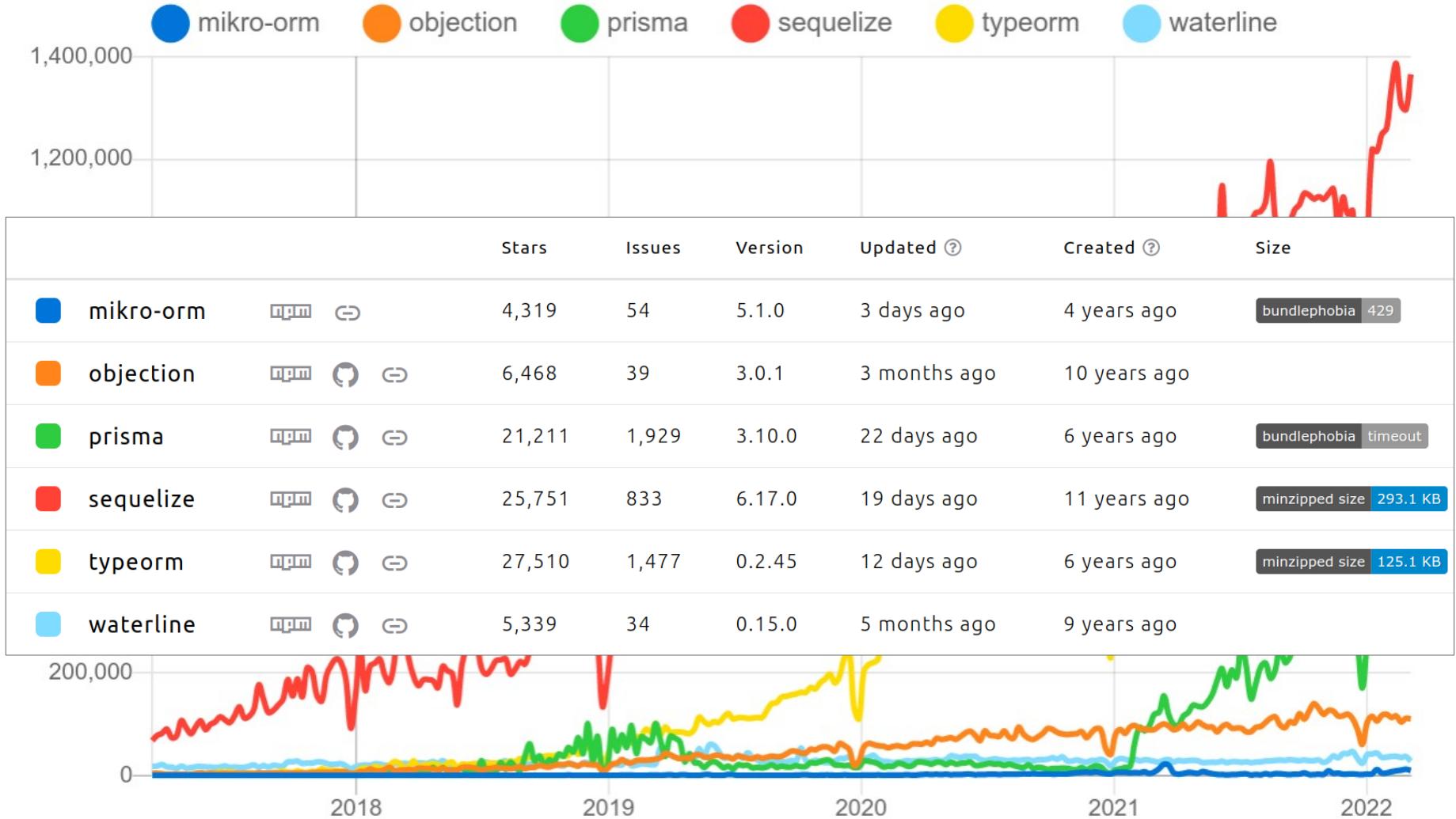
- ORM é uma técnica que permite **consultar e manipular dados** de um database usando o paradigma de orientação a objetos
- Desta forma, o acesso aos dados não é feito através da
| O ORM é responsável pela camada M do padrão MVC. Existem vários pacotes de ORM para Express, e será preciso escolher um antes de prosseguirmos...
| Q/R Mapping



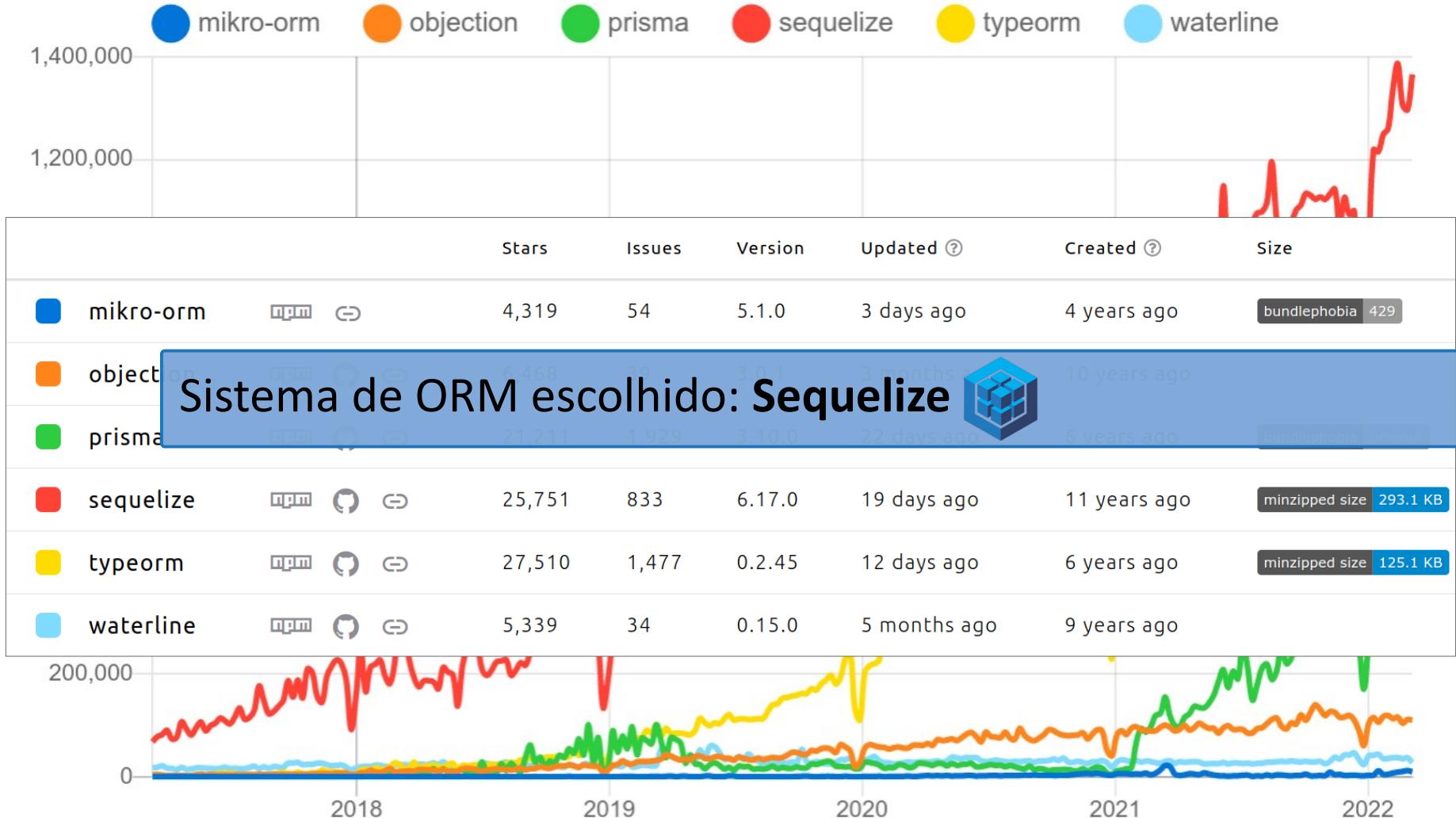
Escolhendo o ORM



Escolhendo o ORM



Escolhendo o ORM



Sequelize



- Use os comandos abaixo para instalar o **Sequelize** com suporte ao banco de dados MySQL

```
$ npm install sequelize sequelize-cli mysql2
```

- Após isso, crie um arquivo **/.sequelizerc** para indicar a localização dos arquivos e diretórios do **Sequelize**

```
module.exports = {  
  'config': __dirname + '/app/config/database.json',  
  'models-path': __dirname + '/app/models',  
  'seeders-path': __dirname + '/app/database/seeders',  
  'migrations-path': __dirname + '/app/database/migrations'  
};
```

Sequelize



- Use os comandos abaixo para instalar o **Sequelize** com suporte ao banco de dados MySQL

```
$ npm install sequelize sequelize-cli mysql2
```

- Após isso, crie um arquivo **/.sequelizerc** para indicar a localização dos arquivos e diretórios do **Sequelize**

O **sufixo rc** significa **runtime configuration**, e é usado em arquivos com configurações para execução de comandos

- Ex: .bashrc, .vimrc, /etc/rc.local, etc.

```
'migrations-path': __dirname + '/app/database/migrations'  
};
```

Sequelize



- Com o arquivo `./sequelizerc` criado, podemos usar o comando abaixo para gerar os arquivos de configuração básicos

```
$ npx sequelize init
```

```
david@coyote ~/dev/myapp $ npx sequelize init
myapp : fish — Konsole
Sequelize CLI [Node: 14.15.0, CLI: 6.2.0, ORM: 6.3.5]

Created "config/database.json"
models folder at "/home/david/dev/myapp/app/models" already exists.
Successfully created migrations folder at "/home/david/dev/myapp/database/migrations".
Successfully created seeders folder at "/home/david/dev/myapp/database/seeders".
david@coyote ~/dev/myapp $ █
```

Sequelize



- Com o arquivo **/.sequelizerc** criado, podemos usar o comando abaixo para gerar os arquivos de configuração básicos

```
$ npx sequelize init
```

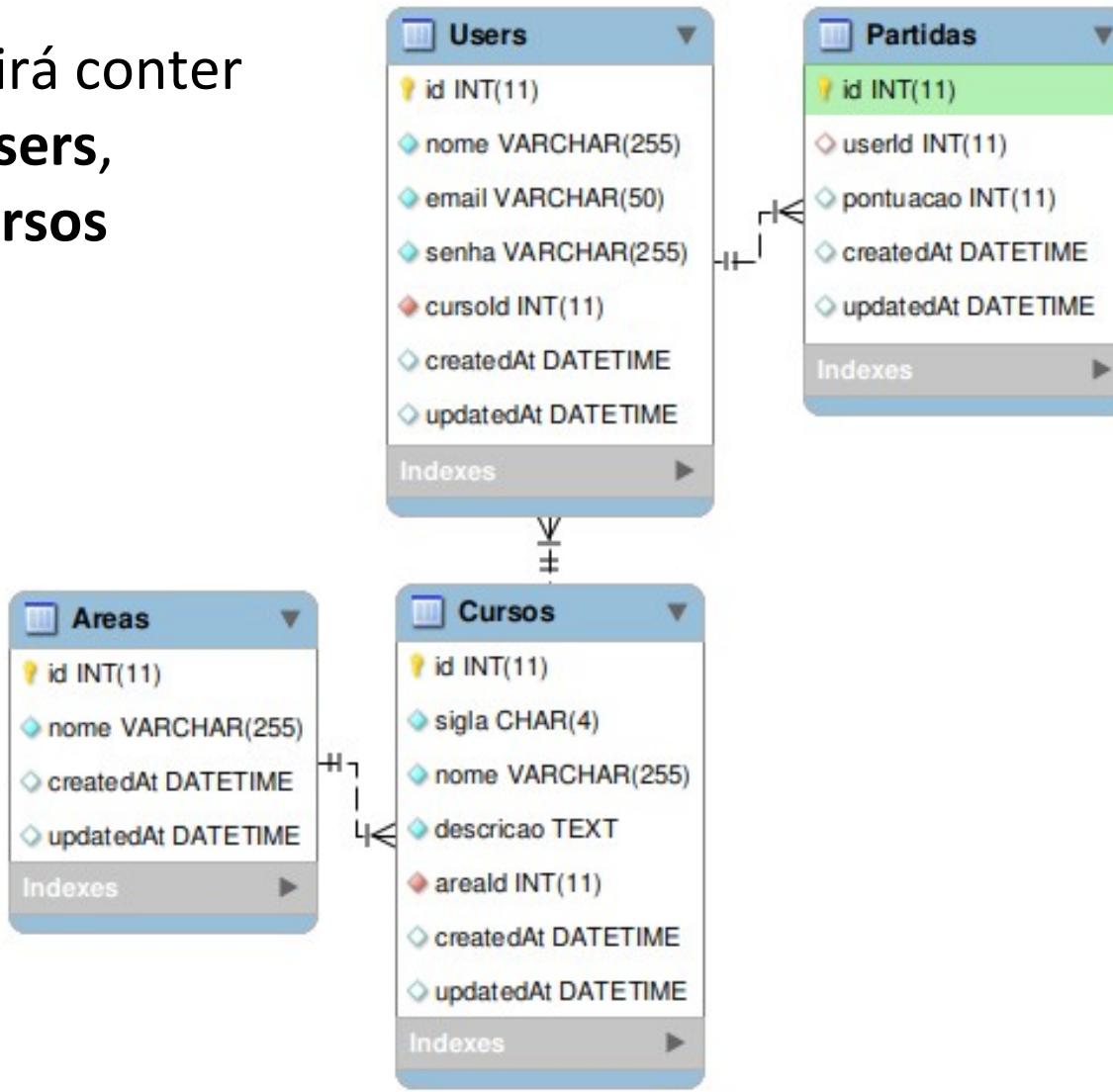
```
david@coyote ~/dev/myapp $ npx sequelize init
```

Muitos pacotes **npm** possuem binários que são copiados para o diretório **node_modules/.bin** após sua instalação

O **npx** é um **package runner** do **npm**, e pode ser usado para executar esses binários, como por exemplo o comando **sequelize** acima

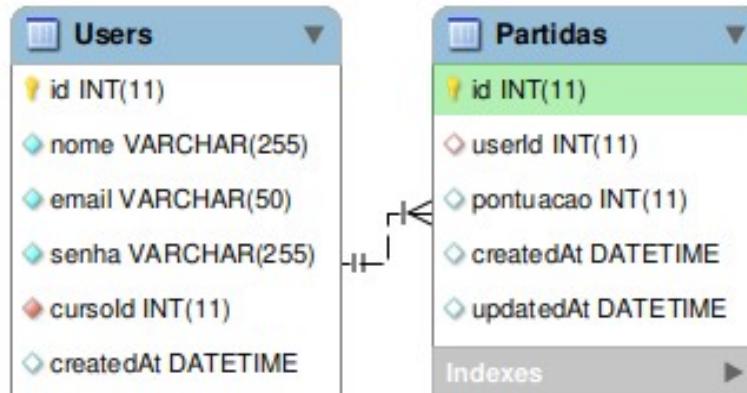
Modelando a Aplicação

- A aplicação irá conter 4 tabelas: **Users**, **Partidas**, **Cursos** e **Areas**



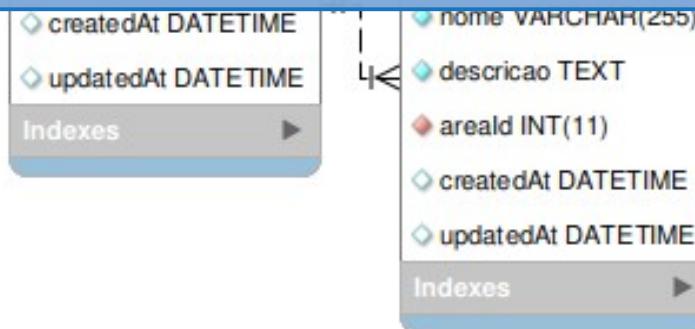
Modelando a Aplicação

- A aplicação irá conter 4 tabelas: **Users**, **Partidas**, **Cursos** e **Areas**



Descrição das Tabelas:

- **Tabela Users:** dados dos usuários da aplicação
- **Tabela Partidas:** dados de cada partida do jogo
- **Tabela Curso:** cursos (cc, es, ec, etc) dos jogadores
- **Tabela Areas:** áreas dos cursos (ciências exatas, biológicas, etc)



Criando o Banco de Dados

- Neste ponto, é importante que o banco de dados e o usuário MySQL já estejam criados no SGBD local
- Caso não estejam, você pode usar comandos abaixo para criá-los, mantendo o usuário e senha conforme informado

```
david@coyote: ~/dev/myapp +  
david@coyote:~/dev/myapp (master)$ mysql -uroot -p --silent  
Enter password:  
mysql> CREATE DATABASE myapp;  
mysql> CREATE USER 'myapp'@'localhost' IDENTIFIED BY 'senha123';  
mysql> GRANT ALL PRIVILEGES ON myapp.* TO 'myapp'@'localhost';  
mysql>  
mysql> █
```

Criando o Banco de Dados

- Neste ponto, é importante que o banco de dados e o usuário MySQL já estejam criados no SGBD local
- Caso não estejam, você pode usar comandos abaixo para criá-

Além do cliente de linha de comando do MySQL, também é possível criar o banco e o usuário através de aplicações como **phpMyAdmin** e **MySQL Workbench**

Enter password:

```
mysql> CREATE DATABASE myapp;
mysql> CREATE USER 'myapp'@'localhost' IDENTIFIED BY 'senha123';
mysql> GRANT ALL PRIVILEGES ON myapp.* TO 'myapp'@'localhost';
mysql>
mysql> █
```

Criando o Banco de Dados

- Neste ponto, é importante que o banco de dados e o usuário MySQL já estejam criados no SGBD local
- Caso não estejam, você pode usar comandos abaixo para criá-

Além do cliente de linha de comando do MySQL, também é possível utilizar o MySQL Workbench para gerenciar suas estruturas de dados.

Vale salientar que uma senha simples como a de nosso exemplo ****não pode**** ser usada em produção

```
mysql> CREATE DATABASE myapp;
mysql> CREATE USER 'myapp'@'localhost' IDENTIFIED BY 'senha123';
mysql> GRANT ALL PRIVILEGES ON myapp.* TO 'myapp'@'localhost';
mysql>
mysql> █
```

Criando o Banco de Dados

- O próximo passo é configurar corretamente os dados de acesso ao banco no arquivo **config/database.json**
- Como o sistema está em desenvolvimento, podemos configurar apenas a seção **development** do arquivo

```
"development": {  
    "username": "myapp",  
    "password": "senha123",  
    "database": "myapp",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
},
```

Criando o Banco de Dados

- O próximo passo é configurar corretamente os dados de acesso ao banco no arquivo **config/database.json**
- Como o sistema está em desenvolvimento, podemos

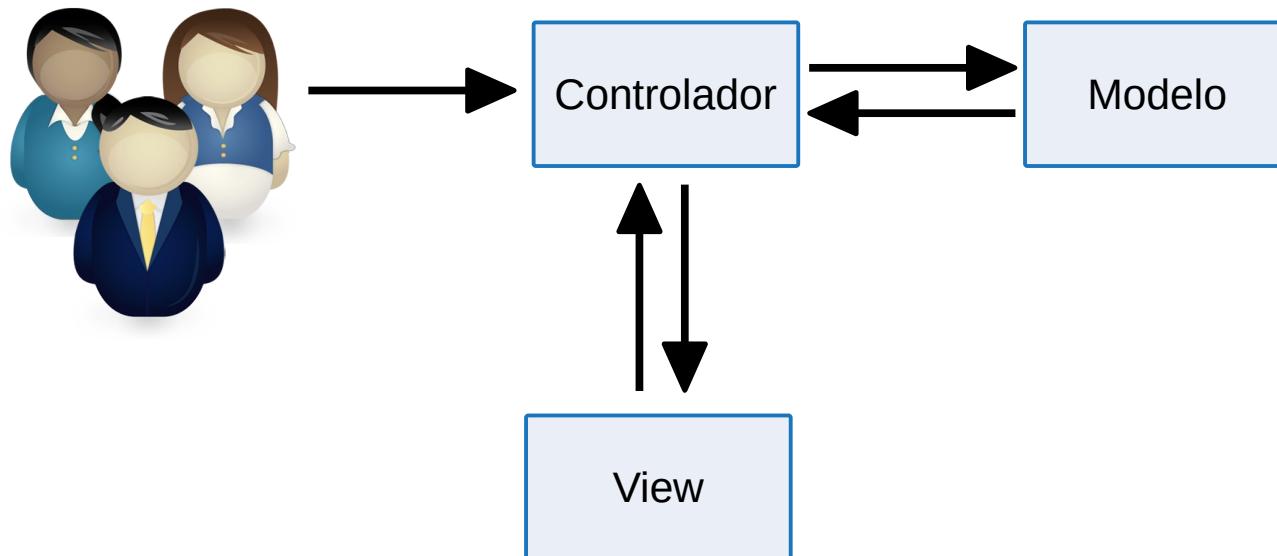
O tipo de servidor – **development**, **test** ou **production** – é definido no arquivo **app/models/index.js** na seguinte linha:

```
const env = process.env.NODE_ENV || 'development';
```

```
"password": "senha123",  
"database": "myapp",  
"host": "127.0.0.1",  
"dialect": "mysql"  
,
```

Gerando modelos e migrações

- Agora que o `sequelize` está devidamente configurado, podemos gerar os **modelos** e **migrações** que serão usados em nossa aplicação
- Modelos e migrações representam a camada M do padrão MVC, e será criado um modelo e uma migração para cada tabela do banco



Gerando modelos e migrações

- Para gerar uma versão inicial dos modelos (e das migrações), podemos usar o comando **sequelize** do pacote **sequelize-cli**

```
$ npx sequelize model:create --name User --attributes \
  "nome:string, email:string, senha:string, cursoId:integer"
```

```
$ npx sequelize model:create --name Partida --attributes \
  "userId:integer, pontuacao:integer"
```

```
$ npx sequelize model:create --name Curso --attributes \
  "sigla:string, nome:string, descricao:text, areaId:integer"
```

```
$ npx sequelize model:create --name Area --attributes \
  "nome:string"
```

Gerando modelos e migrações

- Para gerar uma versão inicial dos modelos (e das migrações), podemos usar o comando **sequelize** do pacote **sequelize-cli**

```
$ npx sequelize model:create --name User --attributes \  
"nome:string, email:string, senha:string, cursoid:integer"
```

Para obter uma listagem completa dos tipos de dados aceitos,

```
$ npx sequelize model:create --name User --attributes \  
"user_id:integer, pontuacao:integer"
```

```
$ npx sequelize model:create --name Curso --attributes \  
"sigla:string, nome:string, descricao:text, areaId:integer"
```

```
$ npx sequelize model:create --name Area --attributes \  
"nome:string"
```

Gerando modelos e migrações

- Para gerar uma versão inicial dos modelos (e das migrações), podemos usar o comando **sequelize** do pacote **sequelize-cli**

```
$ npx sequelize model:create --name User --attributes \
  "nome:string, email:string, senha:string, cursoId:integer"
```

Para obter uma lista com os tipos de dados disponíveis:

```
$ npx sequelize typescript
```

Ainda será preciso editar as migrações e os modelos criados pelo **sequelize-cli**, de forma a adaptá-los às necessidades da aplicação

```
$ npx sequelize model:create --name Curso --attributes \
  "sigla:string, nome:string, descricao:text, areaId:integer"
```

```
$ npx sequelize model:create --name Area --attributes \
  "nome:string"
```

Migrações

- As **migrações** são arquivos que guardam as mudanças que o banco de dados da aplicação sofre ao longo do tempo
 - Elas funcionam como um sistema de controle de versão do esquema do banco, e são usadas para fazer edições no esquema (criar/editar tabelas) ou para reverter edições anteriores
- Os arquivos de migrações ficam em **database/migrations**

```
migrations : fish — Konsole  
david@coyote ~/dev/myapp/database/migrations $ ls  
20201125080657-create-user.js      20201125080756-create-curso.js  
20201125080739-create-partida.js   20201125080811-create-area.js  
david@coyote ~/dev/myapp/database/migrations $ █
```

Migrações

- Uma nova migração deve ser gerada sempre que o esquema do banco de dados precisar sofrer alguma alteração
 - Por exemplo, as migrações geradas pelo sequelize-cli representam a criação das tabelas da aplicação
- Cada migração possui duas funções:
 - Uma **função up**, que contém as edições que precisam ser efetivadas no banco pela migração
 - Uma **função down**, que é usada para reverter as edições no banco geradas pela função **up**

- Uma classe do banco de dados:
 - Por exemplo, criamos a classe
- Cada classe efetivamente gera

```
up: async (queryInterface, Sequelize) => {
  await queryInterface.createTable('Areas', {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: Sequelize.INTEGER
    },
    nome: {
      type: Sequelize.STRING
    },
    createdAt: {
      allowNull: false,
      type: Sequelize.DATE
    },
    updatedAt: {
      allowNull: false,
      type: Sequelize.DATE
    }
  });
}
```

Modificador
de Atributo

quema
resentam
no banco

- Uma
do ba

- Por
a cri

- Cada

- Um

```
up: async (queryInterface, Sequelize) => {
  await queryInterface.createTable('Areas', {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: Sequelize.INTEGER
    },
    nome: {
      type: Sequelize.STRING
    },
    createdAt: {
      type: Sequelize.DATE
    }
  });
}
```

Modificador
de Atributo

quema
resentam

o banco

- Uma das primeiras etapas é definir o banco de dados no Sequelize. Por exemplo:

```
up: async (queryInterface, Sequelize) => {
  await queryInterface.createTable('Areas', {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: Sequelize.INTEGER
    },
    nome: {
      type: Sequelize.STRING
    }
  });
}
```
- Uma vez que o banco de dados é configurado, as migrações são criadas. As migrações geradas pelos comandos **npx sequelize** serão usadas para criar as tabelas iniciais da aplicação. No entanto, antes de criar as tabelas, é preciso adequar as migrações geradas conforme as necessidades do banco, através dos **modificadores** que serão vistos a seguir.

Modificadores de Atributos

- Os **modificadores** são informações adicionais sobre cada atributo que podem ser definidas nos modelos e migrações

Modificador	Descrição	Exemplo
primaryKey	usado para definir a chave primária do modelo	primaryKey: true
defaultValue	usado para definir um atributo padrão para o atributo	defaultsTo:10
allowNull	usado para dizer que o atributo é opcional ou obrigatório	allowNull: true
autoIncrement	usado para dizer que os valores da coluna são auto incrementados	autoIncrement: true
unique	usado para dizer que a coluna não pode ter elementos repetidos	unique: true

Modificadores de Atributos

- Os **modificadores** são informações adicionais sobre cada atributo que podem ser definidas nos modelos e migrações

Modificador	Descrição	Exemplo
primaryKey	usado para definir a chave primária do modelo	primaryKey: true
defaultValue	email: { allowNull: false , unique: true , type: Sequelize.STRING }	defaultsTo:10
allowNull	usado para dizer se o atributo é opcional ou obrigatório	allowNull: true
autoIncrement	usado para dizer que os valores da coluna são auto incrementados	autoIncrement: true
unique	usado para dizer que a coluna não pode ter elementos repetidos	unique: true

Gerando as Tabelas

- Uma vez que as migrações estejam prontas, podemos usar o comando abaixo para gerar as tabelas do banco de dados

```
$ npx sequelize db:migrate
```

- O **sequelize cli** ainda possui outras opções para gerenciamento das migrações

```
sequelize db:migrate:status  
sequelize db:migrate:undo  
sequelize db:migrate:undo:all  
sequelize migration:generate
```

Criando as Tabelas

myapp : fish — Konsole

```
david@coyote ~/dev/myapp $ npx sequelize db:migrate
```

- Usar o comando `npx sequelize db:migrate` para rodar o comando abaixo para gerar as tabelas do banco de dados
- Loaded configuration file "config/database.json".
Using environment "development".
== 20201125080657-create-user: migrating =====
== 20201125080657-create-user: migrated (0.014s)
- O `sequelize cli` ainda possui outras opções para gerenciamento das migrations

```
sequelize db:migrate:status  
== 20201125080756-create-curso: migrating =====  
== 20201125080756-create-curso: migrated (0.012s)  
sequelize migration:generate  
== 20201125080811-create-area: migrating =====  
== 20201125080811-create-area: migrated (0.010s)
```

```
david@coyote ~/dev/myapp $ █
```

Criando as Tabelas

myapp : fish — Konsole

```
david@coyote ~/dev/myapp $ npx sequelize db:migrate
```

- Use o comando `npx sequelize db:migrate` ou o comando abaixo para gerar as tabelas do banco de dados

Loaded configuration file "config/database.json".

Using environment "development".

```
$ npx sequelize db:migrate
```

Table	Action	Rows	Type
Areas		0	InnoDB
Cursos		0	InnoDB
Partidas		0	InnoDB
SequelizeMeta		4	InnoDB
Users		0	InnoDB
5 tables	Sum	4	InnoDB

```
sequelize migration:generate
== 20201125080811-create-area: migrating =====
== 20201125080811-create-area: migrated (0.010s)
```

```
david@coyote ~/dev/myapp $
```

Criando as Tabelas

```
myapp : fish — Konsole
david@coyote ~/dev/myapp $ npx sequelize db:migrate
  Using environment "development".
$ npx sequelize db:migrate
  == 20201125080657-create-user: migrating =====
  -- create table `User`(`id` int NOT NULL AUTO_INCREMENT, `name` varchar(255) NOT NULL, `email` varchar(255) NOT NULL, `password` varchar(255) NOT NULL, `created_at` timestamp NOT NULL, `updated_at` timestamp NOT NULL, PRIMARY KEY(`id`))
  Empty Drop 0 InnoDB
  == 20201125080811-create-area: migrating =====
  -- create table `Area`(`id` int NOT NULL AUTO_INCREMENT, `name` varchar(255) NOT NULL, `description` text, `created_at` timestamp NOT NULL, `updated_at` timestamp NOT NULL, PRIMARY KEY(`id`))
  Empty Drop 0 InnoDB
  == 20201125080811-create-courses: migrating =====
  -- create table `Courses`(`id` int NOT NULL AUTO_INCREMENT, `name` varchar(255) NOT NULL, `description` text, `area_id` int NOT NULL, `created_at` timestamp NOT NULL, `updated_at` timestamp NOT NULL, PRIMARY KEY(`id`), FOREIGN KEY(`area_id`) REFERENCES `Area`(`id`))
  Empty Drop 0 InnoDB
  == 20201125080811-create-partidas: migrating =====
  -- create table `Partidas`(`id` int NOT NULL AUTO_INCREMENT, `name` varchar(255) NOT NULL, `description` text, `area_id` int NOT NULL, `created_at` timestamp NOT NULL, `updated_at` timestamp NOT NULL, PRIMARY KEY(`id`), FOREIGN KEY(`area_id`) REFERENCES `Area`(`id`))
  Empty Drop 0 InnoDB
  == 20201125080811-create-sequelizemeta: migrating =====
  -- create table `SequelizeMeta`(`name` varchar(255) NOT NULL, `version` int NOT NULL, `created_at` timestamp NOT NULL, `updated_at` timestamp NOT NULL, PRIMARY KEY(`name`))
  Empty Drop 4 InnoDB
  == 20201125080811-create-area: migrated (0.010s)
  == 20201125080811-create-area: migrated (0.010s)

david@coyote ~/dev/myapp $
```

Note que as tabelas estão vazias. No entanto, o Sequelize dispõe de um recurso chamado **seeder** para inserir alguns registros iniciais nas tabelas.

Seeders

- Algumas tabelas podem precisar de dados iniciais padrão para que a aplicação funcione adequadamente
 - Por exemplo, nossa aplicação não terá nenhum CRUD para a tabela **area**, e a inserção de dados nessa tabela deverá ser manual
- A inserção de dados iniciais pode ser feita através dos Seeders, que também são criados e gerenciados pelo **sequelize-cli**
- Podemos usar o comando abaixo para gerar um seed para a tabela Areas

```
$ npx sequelize seed:generate --name Areas
```

Seeders

- Algumas tabelas podem precisar de dados iniciais padrão para que a aplicação funcione adequadamente
 - Por exemplo, nossa aplicação não terá nenhum CRUD para a tabela `area`, e a inserção de dados nessa tabela deverá ser manual
- A inserção de dados iniciais pode ser feita através dos Seeders, que também são criados e gerenciados pelo `sequelize-cli`

```
myapp : fish — Konsole
david@coyote ~/dev/myapp $ npx sequelize seed:generate --name Areas
Sequelize CLI [Node: 14.15.0, CLI: 6.2.0, ORM: 6.3.5]

seeders/npx folder was created at "/home/david/dev/myapp/database/seeders" already exists.
New seed was created at /home/david/dev/myapp/database/seeders/2020125094909-Areas.js .
david@coyote ~/dev/myapp $
```

```
module.exports = {
  up: (queryInterface, Sequelize) => {
    /**
     * Add seed commands here.
     *
     * Example:
     * await queryInterface.bulkInsert('People', [{
     *   name: 'John Doe',
     *   isBetaMember: false
     * }], {});
    */
  },
  down: (queryInterface, Sequelize) => {
    /**
     * Add commands to revert seed here.
     *
     * Example:
     * await queryInterface.bulkDelete('People', null, {});
    */
  }
};
```

Um seed possui uma função **up**, para inserção de registros, e uma função **down**, para remoção dos registros inseridos por **up**.

ara

20

```
modul up: async (queryInterface, Sequelize) => {
  up await queryInterface.bulkInsert('Areas', [
    {
      id: 1,
      nome: 'Ciências Exatas',
      createdAt: new Date(),
      updatedAt: new Date()
    },
    {
      id: 2,
      nome: 'Ciências Humanas',
      createdAt: new Date(),
      updatedAt: new Date()
    },
    {
      id: 3,
      nome: 'Ciências Biológicas',
      createdAt: new Date(),
      updatedAt: new Date()
    }
  ], {});
};

down queryInterface.bulkDelete('Areas', null, {});
```

nção
registros,
ra remoção
os por up.

```
modul up: async (queryInterface, Sequelize) => {
  up  await queryInterface.bulkInsert('Areas', [
    {
      id: 1,
      nome: 'Ciências Exatas',
      createdAt: new Date(),
      updatedAt: new Date()
    },
    {
      id: 2,
      nome: 'Ciências Biológicas',
      createdAt: new Date(),
      updatedAt: new Date()
    }
  ], {});
  down: async (queryInterface, Sequelize) => {
    await queryInterface.bulkDelete('Areas', null, {});
  }
};
```

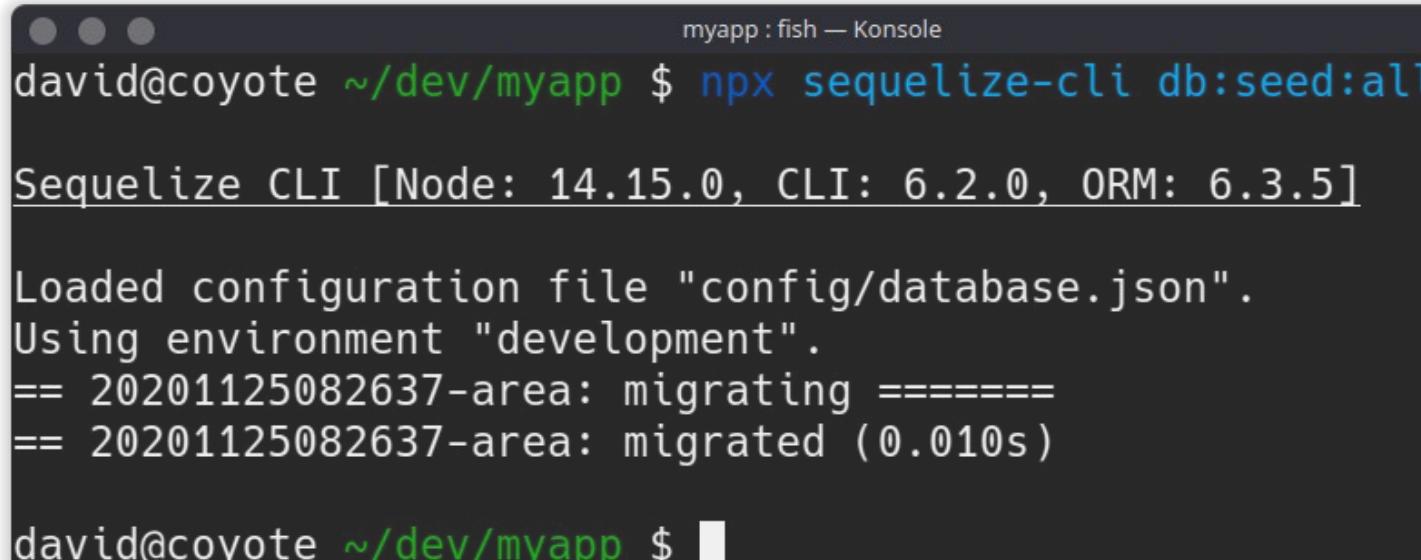
nção
registros,

1, {});

Seeders

- Após a criação dos seeders da aplicação, usamos o comando abaixo para inserir os novos registros nas tabelas desejadas

```
$ npx sequelize-cli db:seed:all
```



```
myapp : fish — Konsole
david@coyote ~/dev/myapp $ npx sequelize-cli db:seed:all
Sequelize CLI [Node: 14.15.0, CLI: 6.2.0, ORM: 6.3.5]

Loaded configuration file "config/database.json".
Using environment "development".
== 20201125082637-area: migrating =====
== 20201125082637-area: migrated (0.010s)

david@coyote ~/dev/myapp $ █
```

Seeders

- Após a criação dos seeders da aplicação, usamos o comando abaixo para inserir os novos registros nas tabelas desejadas

```
$ npx sequelize-cli db:seed:all
```

```
myapp : fish — Konsole
david@coyote ~/dev/myapp $ npx sequelize-cli db:seed:all
```

	id	nome	createdAt	updatedAt
<input type="checkbox"/>	1	Ciências Exatas	2020-11-25 08:31:13	2020-11-25 08:31:13
<input type="checkbox"/>	2	Ciências Humanas	2020-11-25 08:31:13	2020-11-25 08:31:13
<input type="checkbox"/>	3	Ciências Biológicas	2020-11-25 08:31:13	2020-11-25 08:31:13

```
david@coyote ~/dev/myapp $
```

Seeders

- Após a criação dos seeders da aplicação, usamos o comando abaixo para inserir os novos registros nas tabelas desejadas

```
$ npx sequelize-cli db:seed:all
```

Outros comandos do sequelize-cli relacionados com os seeders:

- **sequelize db:seed** – Roda a função **up** de um seeder específico
- **sequelize db:seed:undo** – Roda a função **down** de um seeder específico
- **sequelize db:seed:all** – Roda a função **up** de todos os seeders
- **sequelize db:seed:undo:all** – Roda a função **down** de todos os seeders

 Edit  Copy  Delete 3 Ciências Biológicas 2020-11-25 08:31:13 2020-11-25 08:31:13

david@coyote ~ /dev/myapp \$

Modelos

- Conforme vimos, as migrações são responsáveis por gerir o esquema do banco de dados (estrutura das tabelas)
- Os **modelos**, por outro lado, são responsáveis pela leitura e escrita dos dados presentes nas tabelas
- Cada modelo representa uma tabela do banco de dados
 - Por exemplo, nossa aplicação já possui quatro modelos distintos: **Curso, User, Partida e Area**
 - O modelo **Curso** conterá códigos para acesso e manipulação de todas as linhas da tabela **curso**

Modelos

- Conforme vimos, as migrações são responsáveis por gerir o esquema do banco de dados (estrutura das tabelas)
- Os **modelos**, por outro lado, são responsáveis pela leitura e escrita dos dados presentes nas tabelas

```
models : fish — Konsole
david@coyote ~/dev/myapp/app/models$ ls dados
area.js curso.js index.js partida.js user.js
Por exemplo, nossa aplicação já possui cinco modelos distintos.
david@coyote ~/dev/myapp/app/models$
```

- O modelo **Curso** conterá códigos para acesso e manipulação de todas as linhas da tabela **curso**

Modelos

- Conforme o esquema
- Os modelos são escritos
- **david@cursoemvideo.com.br**

```
// Modelo Area
'use strict';
const { Model } = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class Area extends Model {
    static associate(models) {
      // define association here
    }
  };
  Area.init({
    nome: DataTypes.STRING
  }, {
    sequelize,
    modelName: 'Area',
  });
  return Area;
};
```

gerir o
tura e
los
er.js
ntos.
o de todas

Modelos

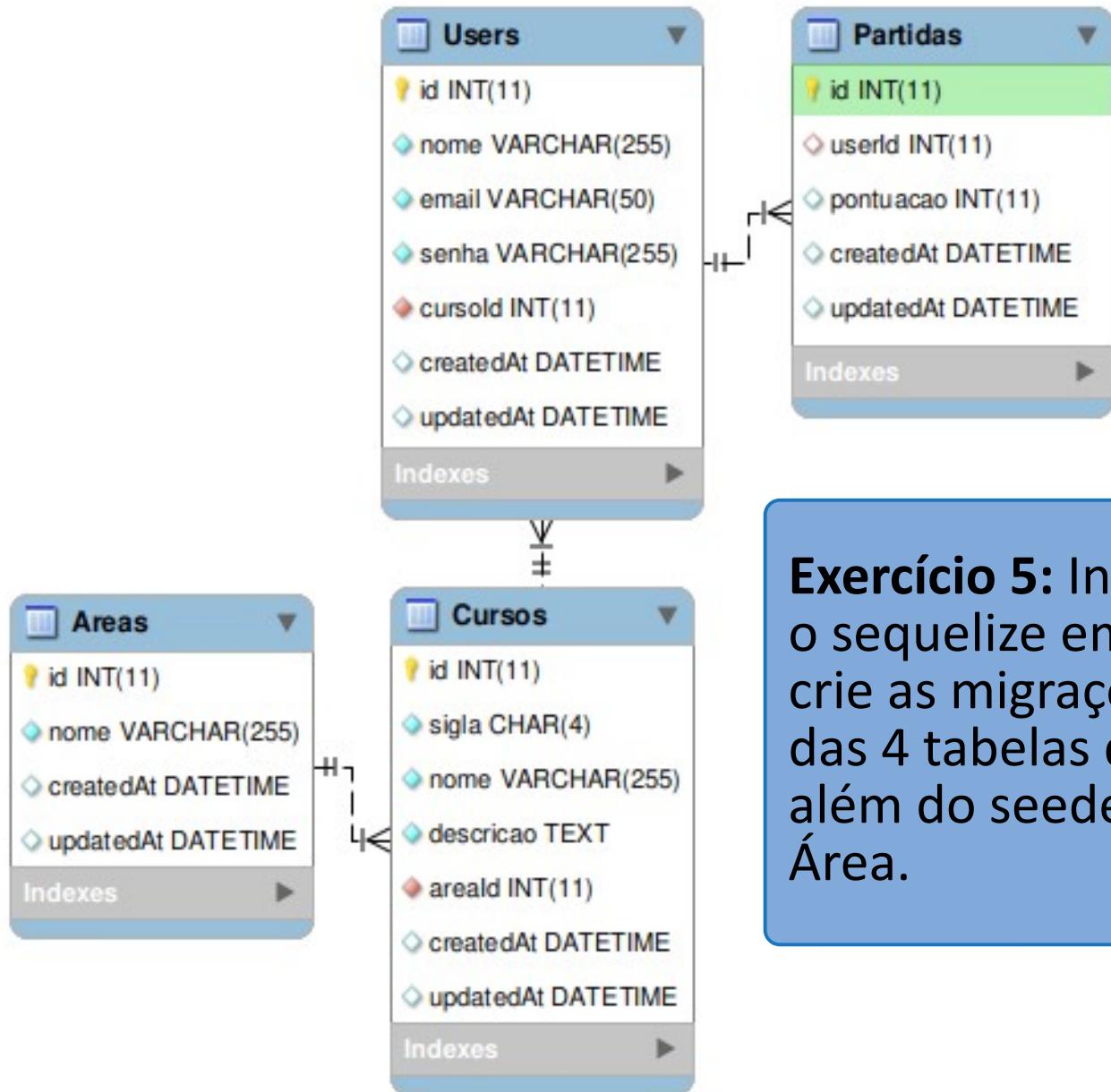
- Conforme o esquema de banco de dados, é necessário gerir o esquema dos modelos.
- Os modelos são classes que herdam da classe Model.

```
// Modelo Area
'use strict';
const { Model } = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class Area extends Model {
```

- Da mesma forma que as migrações, os modelos também precisam ser editados conforme desejado, através dos **modificadores de atributos** apresentados anteriormente.

- O modelo Área terá as seguintes configurações:

```
  nome: DataTypes.STRING
}, {
  allowNull: false,
  sequelize,
  modelName: 'Area',
});
return Area;
};
```



Exercício 5: Instale e configure o sequelize em sua aplicação, crie as migrações e modelos das 4 tabelas da aplicação, além do seeder para a tabela Área.

github
Game

Usando os modelos

- O **controlador** abaixo mostra como usar o **modelo Area** para listar as áreas existentes no banco de dados

```
// Arquivo app/controllers/area.js
const models = require('../models/index');
const Area = models.area;

const index = async (req, res) => {
  const areas = await Area.findAll();
  res.render('area/index', {
    areas: areas.map(area => area.toJSON())
  });
};

module.exports = { index }
```

A função **findAll()** é uma dentre as várias funções do Sequelize para acesso e manipulação de dados

Usando os modelos

- Após a criação do controlador, é preciso definir a rota que será usada para acessar a função **index**

```
const express = require('express');
const router = express.Router();
const MainController = require('../api/controllers/main');
const AreaController = require('../api/controllers/area');

// MainController
router.get('/', MainController.index);
router.get('/sobre', MainController.sobre);
router.get('/ui', MainController.ui);

// AreaController
router.get('/area', AreaController.index);

module.exports = router;
```

Usando os modelos

- O último passo é definir a view **area/index** para impressão dos cursos passados pelo controlador

```
<div class="container">
  <h1>Áreas da Ciência</h1>
  {{#each areas}}
    <li>{{nome}}</li>
  {{/each}}
</div>
```

Usando os modelos

- O
do

The screenshot shows a web browser window titled "My Chess App" with the URL "localhost:3000/area". The page itself has a dark blue header with a white knight icon and the text "Chess App". On the right side of the header is a "Jogar" button with a dropdown arrow. The main content area features a large title "Áreas da Ciência" and a bulleted list: "• Ciências Exatas", "• Ciências Humanas", and "• Ciências Biológicas". At the bottom of the page are links for "Blog", "RSS", "Twitter", "GitHub", "API", and "Donate". Below these links is a note: "Made by Thomas Park. Code released under the MIT License. Based on Bootstrap. Icons from Font Awesome. Web fonts from Google." A yellow "JS" logo is visible in the bottom right corner.

My Chess App

localhost:3000/area

Chess App

Jogar ▾

Áreas da Ciência

- Ciências Exatas
- Ciências Humanas
- Ciências Biológicas

Blog RSS Twitter GitHub API Donate

Made by [Thomas Park](#).
Code released under the [MIT License](#).
Based on [Bootstrap](#). Icons from [Font Awesome](#). Web fonts from [Google](#).

express JS

Usando os modelos

- O
do

The screenshot shows a web browser window titled "My Chess App" at the URL "localhost:3000/area". The page has a dark blue header with a white knight icon and the text "Chess App". On the right side of the header is a "Jogar" button with a dropdown arrow. Below the header, there's a large title "Áreas da Ciência" and a bulleted list: "• Ciências Exatas", "• Ciências Humanas", and "• Ciências Biológicas". At the bottom of the page, there are links for "Blog", "RSS", "Twitter", "GitHub", "API", and "Donate". A blue callout box contains the text: "Exercício 6: Crie a rota area para apresentar os registros na tabela Areas." To the right of the page, there's a red oval containing the text "github Game". At the bottom right, there's a yellow square with the letters "JS" and a black bar with the word "express".

My Chess App

localhost:3000/area

Chess App

Jogar ▾

Áreas da Ciência

- Ciências Exatas
- Ciências Humanas
- Ciências Biológicas

Blog RSS Twitter GitHub API Donate

Made by [Thomas Park](#).
Code released under the [MIT License](#).
Based on [Bootstrap](#). Icons from [Font Awesome](#). Web fonts from [Google](#).

Exercício 6: Crie a rota area para apresentar os registros na tabela Areas.

github Game

express JS

Encontrando registros

- Para recuperar registros no banco de dados:

```
var records = await Something.findAll(criteria);
```

- Exemplos:

```
SELECT * FROM user WHERE nome = 'Carlos':
```

```
User.findAll({ where: { nome: 'Carlos' }});
```

```
SELECT nome,idade FROM user WHERE nome = 'Carlos':
```

```
User.findAll({  
  where: { nome: 'Carlos' },  
  Attributes: [ 'nome', 'idade' ]  
});
```

Encontrando registros

- O objeto **Sequelize.Op** pode ser usado para criar comparações mais complexas durante as consultas

```
const Op = Sequelize.Op

[Op.and]: {a: 5}           // AND (a = 5)
[Op.or]: [{a: 5}, {a: 6}]  // (a = 5 OR a = 6)
[Op.gt]: 6,                 // > 6
[Op.gte]: 6,                // >= 6
[Op.lt]: 10,                // < 10
[Op.lte]: 10,                // <= 10
[Op.ne]: 20,                // != 20
[Op.eq]: 3,                  // = 3
[Op.is]: null,               // IS NULL
[Op.not]: true,               // IS NOT TRUE
[Op.between]: [6, 10],        // BETWEEN 6 AND 10
[Op.notBetween]: [11, 15],     // NOT BETWEEN 11 AND 15
[Op.in]: [1, 2],               // IN [1, 2]
[Op.notIn]: [1, 2],             // NOT IN [1, 2]
```

Encontrando registros

- O objeto **Sequelize.Op** pode ser usado para criar comparações mais complexas durante as consultas

```
const Op = Sequelize.Op

[Op.and]: {a: 5}          // AND (a = 5)
[Op.or]: [{a: 5}, {a: 6}] // (a = 5 OR a = 6)

[Op.like]: '%hat',        // LIKE '%hat'
[Op.notLike]: '%hat'      // NOT LIKE '%hat'
[Op.startsWith]: 'hat'    // LIKE 'hat%'
[Op.endsWith]: 'hat'      // LIKE '%hat'
[Op.substring]: 'hat'     // LIKE '%hat%'
[Op.regexp]: '^h|a|t'     // REGEXP/~/^h|a|t/
[Op.notRegexp]: '^h|a|t'   // NOT REGEXP/!~/^h|a|t/'

[Op.not]: true,           // IS NOT TRUE
[Op.between]: [6, 10],     // BETWEEN 6 AND 10
[Op.notBetween]: [11, 15], // NOT BETWEEN 11 AND 15
[Op.in]: [1, 2],          // IN [1, 2]
[Op.notIn]: [1, 2],        // NOT IN [1, 2]
```

Encontrando registros

- Exemplos de consultas com o objeto **Sequelize.Op**

```
SELECT * FROM user WHERE idade < 30:
```

```
User.findAll({ where: { idade: { [Op.lt]: 30 } }});
```

```
SELECT * FROM user WHERE idade >= 21:
```

```
User.findAll({ where: { idade: { [Op.gte]: 21 } }});
```

```
SELECT * FROM user WHERE nome IN ('Carlos', 'Maria'):
```

```
User.findAll({ where: {  
  nome: { [Op.in]: ['Carlos', 'Maria'] }  
}});
```

```
SELECT * FROM user WHERE nome NOT IN ('Carlos', 'Maria'):
```

```
User.findAll({ where: {  
  nome: { [Op.notIn]: ['Carlos', 'Maria'] }  
}});
```

Encontrando registros

- Selecionando usuários cujo nome possuem a string Carlos

```
SELECT * FROM user WHERE nome like '%Carlos':
```

```
User.findAll({ where: {  
  nome: { [Op.like]: '%Carlos%' }  
}});
```

- Selecionando usuários cujo nome começam com Carlos

```
SELECT * FROM user WHERE nome like 'Carlos%':
```

```
User.findAll({ where: {  
  nome: { [Op.like]: 'Carlos%' }  
}});
```

- Selecionando usuários cujo nome terminam com Oliveira

```
SELECT * FROM user WHERE nome like '%Oliveira':
```

```
User.findAll({ where: {  
  nome: { [Op.like]: '%Oliveira%' }  
}});
```

Paginação

- As cláusulas **skip** e **limit** podem ser usadas em conjunto para construir um sistema de paginação
- Selecionando os usuários 1-10 com idade maior que 20

```
SELECT * FROM user WHERE idade > 20 LIMIT 10 OFFSET 0;
```

```
User.findAll({ where:  
  { idade: { [Op.gt]: 20 }}, limit: 10, offset: 0  
});
```

- Selecionando os usuários 11-20 com idade maior que 20

```
SELECT * FROM user WHERE idade > 20 LIMIT 10 OFFSET 10;
```

```
User.findAll({{ where:  
  { idade: { [Op.gt]: 20 }}, limit: 10, offset: 10  
}});
```

Ordenando Registros

- Os resultados recuperados podem ser ordenados de forma crescente (ASC) ou descrescente (DESC) por qualquer atributo
- Usuários ≥ 18 ordenados pelo nome em ordem crescente

```
SELECT * FROM user WHERE idade >= 18 ORDER BY nome ASC;
```

```
User.findAll({ where:  
    { idade: { [Op.gte]: 18 }}, order: [['nome', 'ASC']]  
});
```

- Usuários ≥ 18 ordenados pelo nome em ordem decrescente

```
SELECT * FROM user WHERE idade >= 18 ORDER BY nome DESC;
```

```
User.findAll({ where:  
    { idade: { [Op.gte]: 18 }}, order: [['nome', 'DESC']]  
});
```

Selecionando apenas um registro

- O comando **findAll()** retorna um array com todos os registros que atenderem os critérios da cláusula **where**
 - Note que, se apenas um registro atender aos critérios da cláusula **where**, será retornado um array com uma única posição
- Uma alternativa é o uso de **findOne()**, que ao invés de recuperar um array, retorna apenas um objeto

```
SELECT * FROM user WHERE id = 1;
```

```
const user = await User.findOne({ where: {id:1}});
```

Criando novos registros

- Criação de novos registros no banco de dados:

```
await Something.create(initialValues);
```

- Exemplo:

```
INSERT INTO user (nome, idade) VALUES ('Carlos', 26);
```

```
await User.create({nome: 'Carlos', idade: 26});
```

Atualizando registros existentes

- Atualização de registros já existentes no banco de dados:

```
await Something.update({values},{criteria});
```

- Exemplo:

```
UPDATE user SET idade=30 WHERE nome = 'Carlos';
```

```
await User.update({ idade: 30 }, {  
  where: { nome: 'Carlos' }  
});
```

Apagando registros existentes

- Apagando registros no banco de dados:

```
await Something.destroy(criteria);
```

- Exemplos:

```
DELETE FROM user WHERE nome = 'Carlos';
```

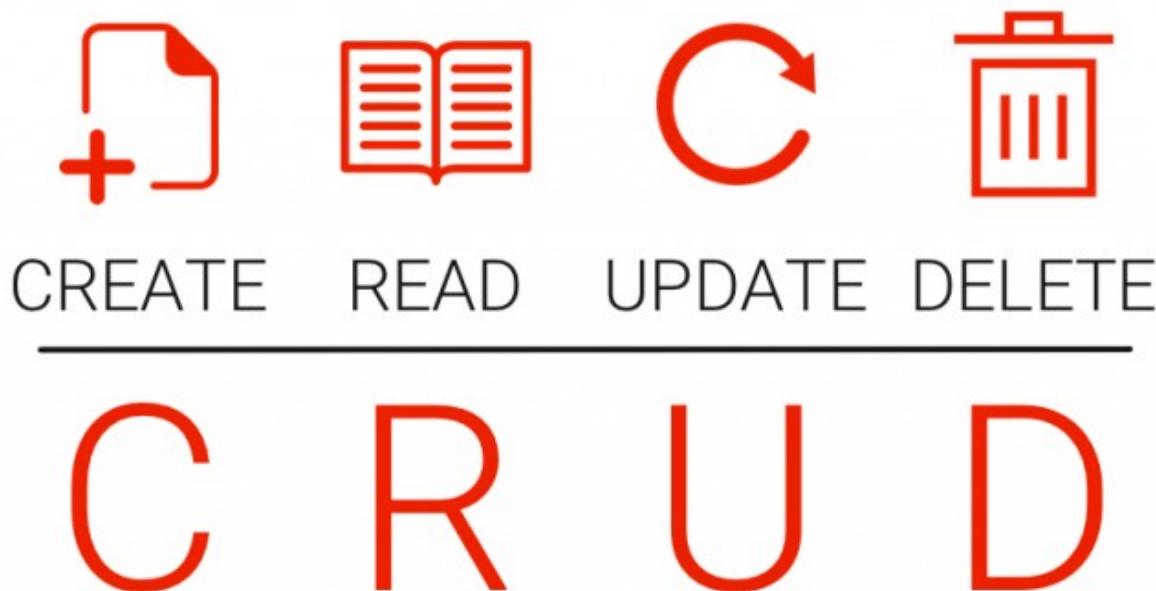
```
await User.destroy({  
  where: { nome: 'Carlos' }  
});
```

```
DELETE FROM user WHERE id IN (3, 97);
```

```
await User.destroy({  
  where: { [Op.in]: [3, 97] }  
});
```

Desenvolvendo CRUDs

- Uma parte importante do desenvolvimento de uma aplicação é a criação do **CRUD** a alguns modelos
 - O CRUD de um modelo é um conjunto de páginas responsáveis por quatro operações sobre esse esse modelo:



Desenvolvendo CRUDs

- Para exemplificar a criação de novos CRUDs, vamos desenvolver um para o modelo Curso
- O primeiro passo é criar um controlador vazio para o CRUD de Curso dentro do diretório **/app/controllers**

```
// Arquivo app/controllers/curso.js
const models = require('../models/index');
const Curso = models.curso;

async function index (req, res) {};
async function read (req, res) {};
async function create (req, res) {};
async function update (req, res) {};
async function remove (req, res) {};

module.exports = { index, read, create, update, remove }
```

Operações
do CRUD

Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
const express = require('express');
const router = express.Router();
const cursoController = require('../api/controllers/curso');

// CursoController
router.get('/curso' , cursoController.index);
router.get('/curso/read/:id' , cursoController.read);
router.get('/curso/create' , cursoController.create);
router.post('/curso/create' , cursoController.create);
router.get('/curso/update/:id' , cursoController.update);
router.post('/curso/update/:id' , cursoController.update);
router.post('/curso/remove/:id' , cursoController.remove);

module.exports = router;
```

Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
const express = require('express');
const router = express.Router();
const cursoController = require('../api/controllers/curso');

// CursoController
router.get('/curso' , cursoController.index);
router.get('/curso/read/:id' , cursoController.read);
router.get('/curso/create' , cursoController.create);
router.get('/curso/update/:id' , cursoController.update);

router.post('/curso/update/:id' , cursoController.update);
router.post('/curso/remove/:id' , cursoController.remove);

module.exports = router;
```

Embora não faça parte do CRUD, o objetivo da rota `/curso`
é listar os cursos existentes

Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
const express = require('express');
const router = express.Router();
const cursoController = require('../api/controllers/curso');

// CursoController
router.get('/curso' , cursoController.index);
router.get('/curso/read/:id' , cursoController.read);
router.get('/curso/read/:id' , cursoController.read);
router.get('/curso/read/:id' , cursoController.read);
router.get('/curso/read/:id' , cursoController.read);

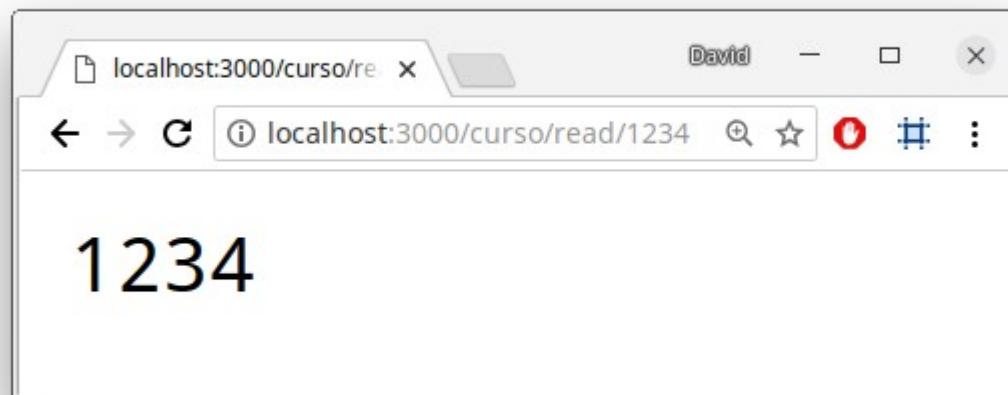
Note que as rotas para read, update e delete terminam com
uma string :id, que representa um parâmetro utilizado para
informar que curso se de deseja ler, atualizar ou apagar
router.post('/curso/remove/:id' , cursoController.remove);

module.exports = router;
```

Definindo as Rotas do CRUD

- Por exemplo, na url `http://localhost:3000/curso/read/1234`, o valor do parâmetro **id** é 1234
- Para ler o valor de **id** dentro da action, podemos usar o atributo **param** de **req** (objeto da requisição do usuário):

```
async function read (req, res) {  
  const cursoId = req.params.id;  
  res.end(cursoId);  
},
```



Requisições POST

- Quando o usuário preenche e submete um formulário POST, os dados informados pelo usuário são enviados para o servidor

```
<form action="/curso/create" method="post">
  <input type="text" name="sigla">
  <input type="text" name="nome">
  <input type="submit" value="Enviar">
</form>
```

- Após a submissão, os dados são enviados através do corpo da requisição HTTP (request body)
- Esses dados são enviados no formato **application/x-www-form-urlencoded**, que também é usado em requisições GET:

```
sigla=IE08&nome=Ciencia+da+Computacao
```

Requisições POST

- O Express possui um middleware nativo chamado **urlencoded**, que pode ser usado para extrair os dados de **request body**
- Para usá-lo, basta inserir a linha abaixo no arquivo **app.js**, em alguma lugar antes da chamada ao middleware router:

```
// Arquivo app.js
app.use(express.urlencoded({extended: false}));
...
app.use(router);
```

Requisições POST

- Após isso, o **urlencoded** irá extrair os dados do **request body** de todas as requisições POST e copiá-los no objeto **req.body**

```
// Curso Controller
const create = async function (req, res) {
  if (req.route.methods.get) {
    res.render('curso/create');
  } else {
    curso = await Curso.create({
      sigla: req.body.sigla,
      nome: req.body.nome,
      descricao: req.body.descricao,
      id_area: req.body.area,
    });
    res.redirect('/curso');
  }
}
```

Regras de Validação

- Os **validadores** são regras usadas nos modelos (não nas migrações) para garantir a consistência dos dados durante a inserção e atualização dos registros

Validador	Descrição	Exemplo
isInt	Verifica se um dado valor é um inteiro	isInt: true
max	Verifica se um dado número é menor que um dado valor	max:10000
min	Verifica se um dado número é maior que um dado valor	min: 0
isEmail	Verifica se um dado valor é um email válido ou não	isEmail: true
isIn	Verifica se um dado valor pertence ou não a uma lista pré-definida	isIn: [['foo', 'bar']]

Regras de Validação

- Os **validadores** são regras usadas nos modelos (não nas migrações) para garantir a consistência dos dados durante a inserção e atualização dos registros

Validador	Descrição	Exemplo
isInt	Outros exemplos de validadores: <code>isUrl</code> , <code>isIP</code> , <code>isAlpha</code> , <code>isAlphanumeric</code> , <code>isNumeric</code> , <code>isFloat</code> , <code>isDecimal</code> , <code>isLowercase</code> , <code>isUppercase</code> , <code>notNull</code> , <code>isNull</code> , <code>notEmpty</code> , <code>equals</code> , <code>contains</code> , <code>notIn</code> , <code>notContains</code> , <code>len</code> , <code>isDate</code> , <code>isAfter</code> , <code>isBefore</code> , <code>isArray</code> , <code>isCreditCard</code> , dentre outros.	
max		
min		
isEmail	A documentação completa dos validadores pode ser encontrada na documentação oficial do sequelize	
isIn	Verifica se um dado valor pertence ou não a uma lista pré-definida	<code>isIn: [['foo', 'bar']]</code>

Regras de Validação

- Os **validadores** são regras usadas nos modelos (não nas migrações) para garantir a consistência dos dados durante a inserção e atualização dos registros

Validador	Descrição	Exemplo
isInt	Outros exemplos de validadores: <code>isUrl</code> , <code>isIP</code> , <code>isAlpha</code> , <code>isAlphanumeric</code> , <code>isNumeric</code> , <code>isFloat</code> , <code>isDecimal</code> , <code>isLowercase</code> , <code>name: { equals: 'foo' }</code> , <code>notNull</code> , <code>notEmpty</code> , <code>isAfter</code> , <code>isBefore</code> , <code>contains</code> , <code>type: DataTypes.STRING</code> , <code>allowNull: false</code> , <code>validate: { len: [3, 50] }</code> , entre outros.	
max		
min		
isEmail	A documentação completa dos validadores pode ser encontrada na documentação oficial do Sequelize .	
isIn	Verifica se um dado valor pertence ou não a uma lista pré-definida	<code>isIn: [['foo', 'bar']]</code>

Regras de Validação

- O módulo de validação usado pelo Sequelize possui mensagens de erro padronizadas
- No entanto, é possível definir mensagens personalizadas para cada um dos validadores apresentados

```
isInt: {  
  msg: "Precisa ser um número inteiro"  
}
```

- Caso o validador necessite de um argumento não booleano, esse argumento pode ser passado através do atributo **args**

```
len: {  
  args: [3, 40],  
  msg: "Precisa conter entre 3 e 40 caracteres"  
}
```

Regras de Validação

- Para exemplificar o uso dos validadores, considere que precisamos limitar o nome dos cursos entre 5 e 40 caracteres

```
nome: {  
  type: DataTypes.STRING,  
  allowNull: false,  
  validate: {  
    len: {  
      args: [5,40],  
      msg: 'O nome precisa ter entre 5 e 40 caracteres.'  
    }  
  },  
},
```

Regras de Validação

- Para exemplificar o uso dos validadores, considere que preciso:

nom
t
a
v
},
},
},

```
// Action create de CursoController
const create = async (req, res) => {
  if (req.route.methods.get) {
    res.render('curso/create');
  } else {
    try {
      await Curso.create(req.body);
    } catch (e) {
      console.log(e);
    }
  }
};
```

Caso o nome do curso tenha um número inválido de caracteres, o **catch** será acionado

caracteres.'



Adicionar Curso

[← Volta](#)

Sigla

IE07

Em caso de dúvida, verifique no site da UFAM.

Nome

ABC

Descrição do curso

Alguma descrição

Área de Conhecimento

- Ciências Exatas – Computação, engenharias, matemática, estatística, etc.
- Ciências Humanas – Letras, psicologia, história, geografia, etc.
- Ciências Biológicas – medicina, odontologia, biologia, agrária, etc.

My Chess App x

David

localhost:3000/curso/create

Chess App

Jogar ▾ Curso Sobre Logo

```
david@coyote: ~/dev/myapp.ok +  
name: 'SequelizeValidationError',  
errors:  
[ ValidationErrorItem {  
  message: 'O nome precisa ter entre 5 e 40 caracteres.',  
  type: 'Validation error',  
  path: 'nome',  
  value: 'ABC',  
  origin: 'FUNCTION',  
  instance: [curso],  
  validatorKey: 'len',  
  validatorName: 'len',  
  validatorArgs: [Array],  
  original: [Error] } ] }
```

Note que o array **errors** é retornado, contendo um erro para cada campo com valor inválido

Área de Conhecimento

- Ciências Exatas – Computação, engenharias, matemática, estatística, etc.
- Ciências Humanas – Letras, psicologia, história, geografia, etc.
- Ciências Biológicas – medicina, odontologia, biologia, agrária, etc.

Regras de Validação

- Para mostrar os erros na view **curso/create**, teremos que usar a função **render** passando o vetor de erros ocorridos

```
// Action create de CursoController
const create = async (req, res) => {
  if (req.route.methods.get) {
    res.render('curso/create');
  } else {
    try {
      await Curso.create(req.body);
    } catch (e) {
      res.render('curso/create', {
        curso: req.body,
        errors: error.errors
      });
    }
  }
};
```

Regras de Validação

- Podemos usar **helpers handlebars** para mostrar os erros para os usuários

```
<div class="form-group">
  <label for="nome">Nome</label>
  <input type="text" name="nome" value="{{curso.nome}}">
  <div class="invalid-feedback">{{showError errors 'nome'}}</div>
</div>
```

```
const showError = function (errors, field) {
  let mensagem;
  if (typeof errors != 'undefined') {
    errors.forEach(function (error) {
      if (error.path == field) {
        mensagem = error.message;
        return;
      }
    });
    return mensagem;
}
```



Adicionar Curso

[← Voltar](#)

Sigla

IE07

Em caso de dúvida, verifique no site da UFAM.

Nome

ABC



O nome precisa ter entre 5 e 40 caracteres.

Descrição do curso

Alguma descrição

Área de Conhecimento

- Ciências Exatas – Computação, engenharias, matemática, estatística, etc.
- Ciências Humanas – Letras, psicologia, história, geografia, etc.

My Chess App X David

localhost:3000/curso/create

Chess App

Jogar ▾ Curso Sobre Log

Adicionar Curso

Sigla

IE07

Em caso de dúvida, verifique no site da UFAM.

Nome

ABC

O nome precisa ter entre 5 e 40 caracteres.

Descrição do curso

Alguma descrição

Exercício 7: Crie o CRUD para o modelo Curso. Nos formulários, a Sigla precisa conter exatamente 4 caracteres, e o nome entre 5 e 40 caracteres.

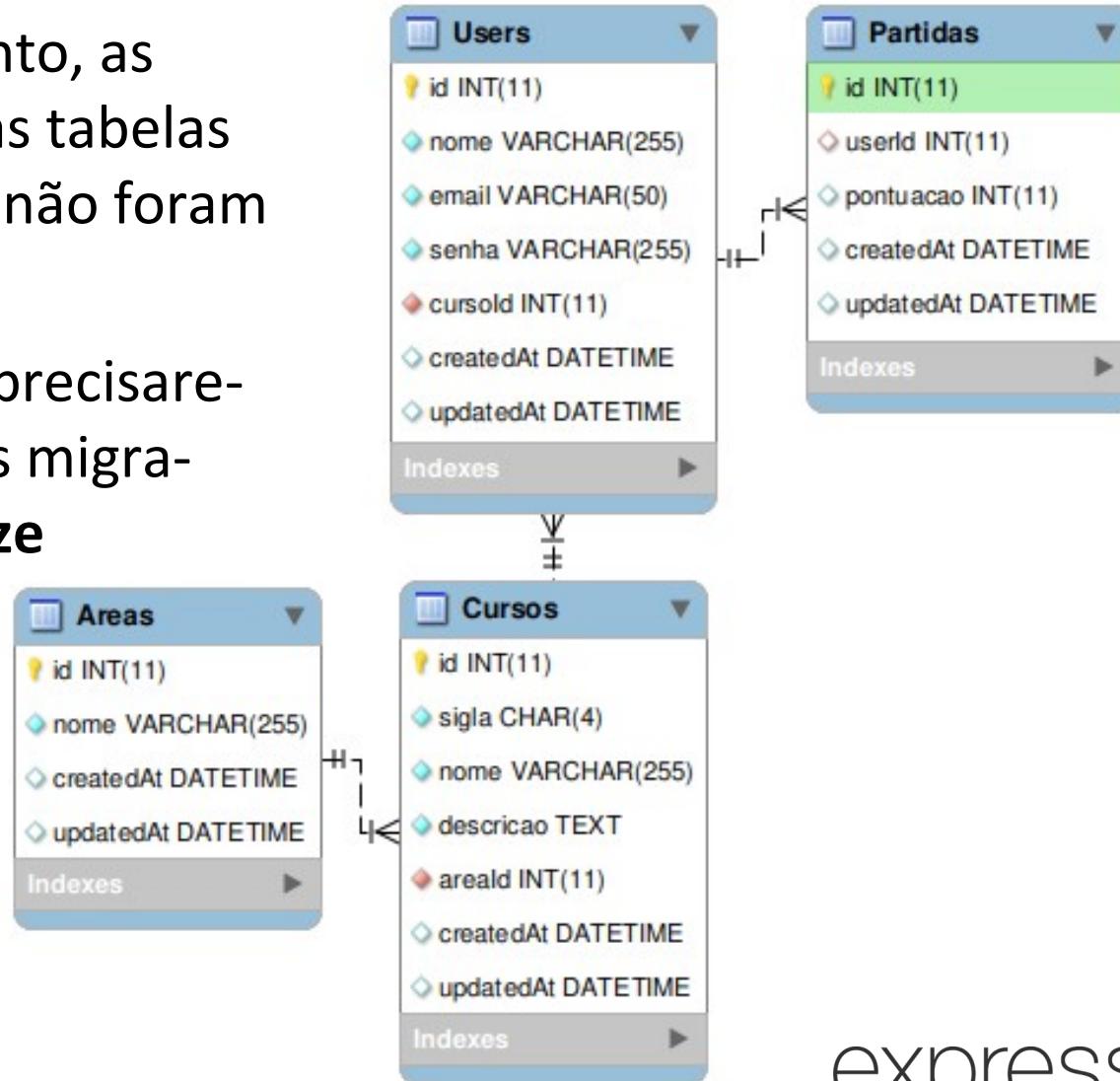
github
Game

Área de Conhecimento

- Ciências Exatas – Computação, engenharias, matemática, estatística, etc.
- Ciências Humanas – Letras, psicologia, história, geografia, etc.

Adicionando Relações entre Tabelas

- Até este momento, as **relações** entre as tabelas de nosso banco não foram configuradas
- Para fazer isso, precisaremos gerar novas migrações no **sequelize**



Adicionando Relações entre Tabelas

- Para gerar os **boilerplates** das migrações, podemos usar novamente o sequelize-cli

```
$ npx sequelize migration:generate --name relacoes-curso  
$ npx sequelize migration:generate --name relacoes-partida
```

- Os comandos acima geram migrações vazias, mas já contendo as funções **up** e **down** com exemplos de uso
- As funções up e down aceitam vários tipos de comandos de edições de tabelas
 - Ex: createTable, dropTable, addColumn, removeColumn, addConstraint, removeConstraint, addIndex, removeIndex, etc

Adicionando Relações entre Tabelas

- A relação entre duas tabelas é definida através de regras de restrição referencial, e por isso podemos usar o comando **addConstraint** para definir essas relações

```
up: async (queryInterface, Sequelize) => {
  await queryInterface.addConstraint('Cursos', {
    type: 'foreign key',
    fields: ['areaId'],
    name: 'curso_area_fk',
    references: {
      table: 'Areas',
      field: 'id'
    },
    onDelete: 'restrict',
    onUpdate: 'restrict'
  })
},
```

Adicionando Relações entre Tabelas

- A relação entre duas tabelas é definida através de regras de restrição referencial, e por isso podemos usar o comando **addConstraint** para definir essas relações

```
up: async (queryInterface, Sequelize) => {
  await queryInterface.addConstraint('Cursos', {
    down: async (queryInterface, Sequelize) => {
      await queryInterface.removeConstraint(
        'Cursos',
        'curso_area_fk'
      )
    }
  },
  onDelete: 'restrict',
  onUpdate: 'restrict'
})
},
```

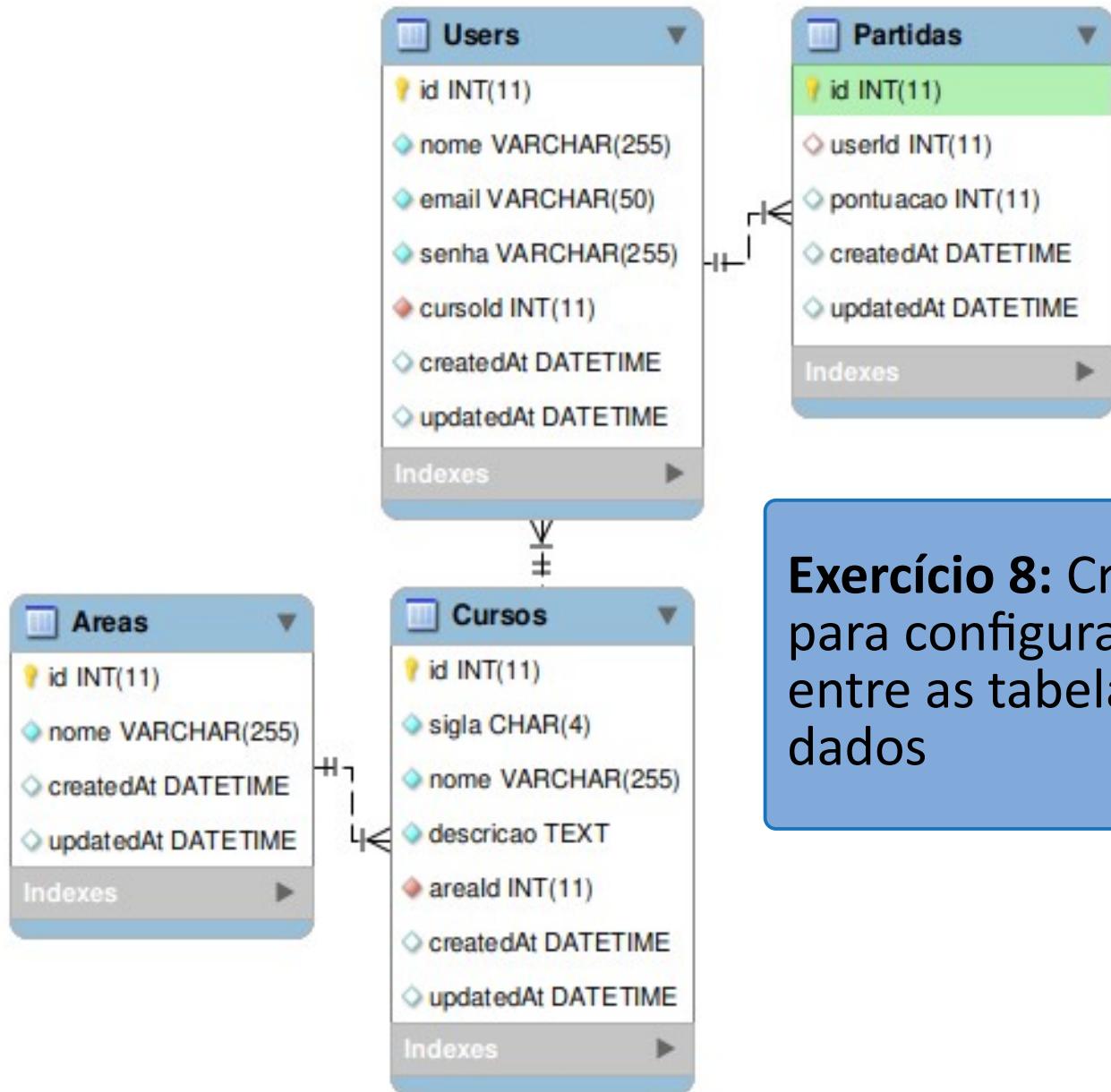
Adicionando Relações entre Tabelas

- A relação entre duas tabelas é definida através de regras de restrição referencial, e por isso podemos usar o comando **addConstraint** para definir essas relações

```
up: async (queryInterface, Sequelize) => {
  await queryInterface.addConstraint('Curros', {
    type: 'FOREIGN KEY',
    name: 'FK_Curros_Professores',
    column: 'professor_id',
    onDelete: 'CASCADE',
    onUpdate: 'restrict'
  })
},
```

Após configurar todas as novas migrações basta rodar o comando:

```
$ npx sequelize db:migrate
```

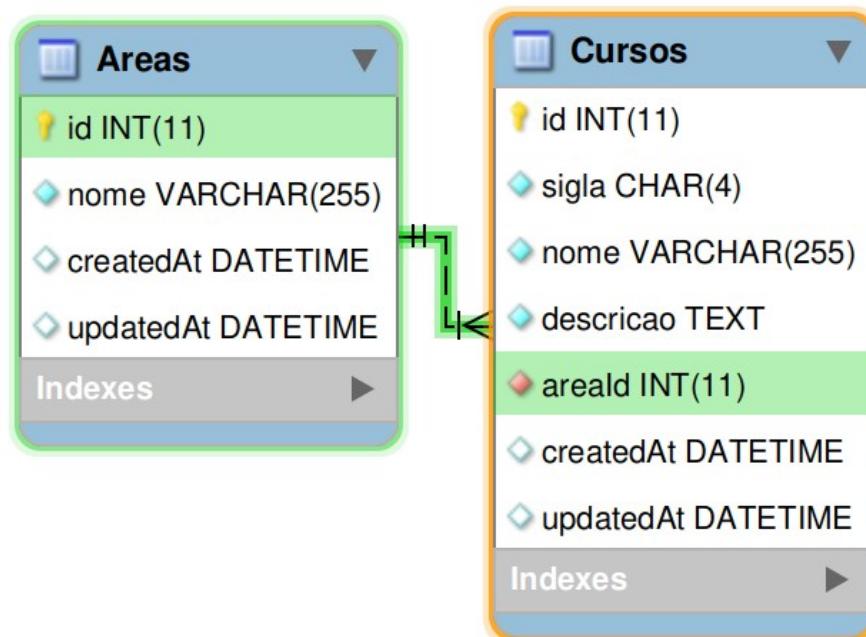


Exercício 8: Crie as migrações para configurar as relações entre as tabelas no banco de dados

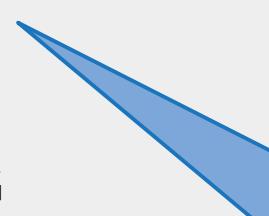
github
Game

Associações

- Além dos atributos (do tipo **string**, **number**, etc), os modelos também podem possuir **conexões** com outros modelos
 - Essas conexões com outros modelos são chamados de **associações**
- Por exemplo, no esquema de banco de dados de nossa aplicação, cada curso está sempre associado a uma área



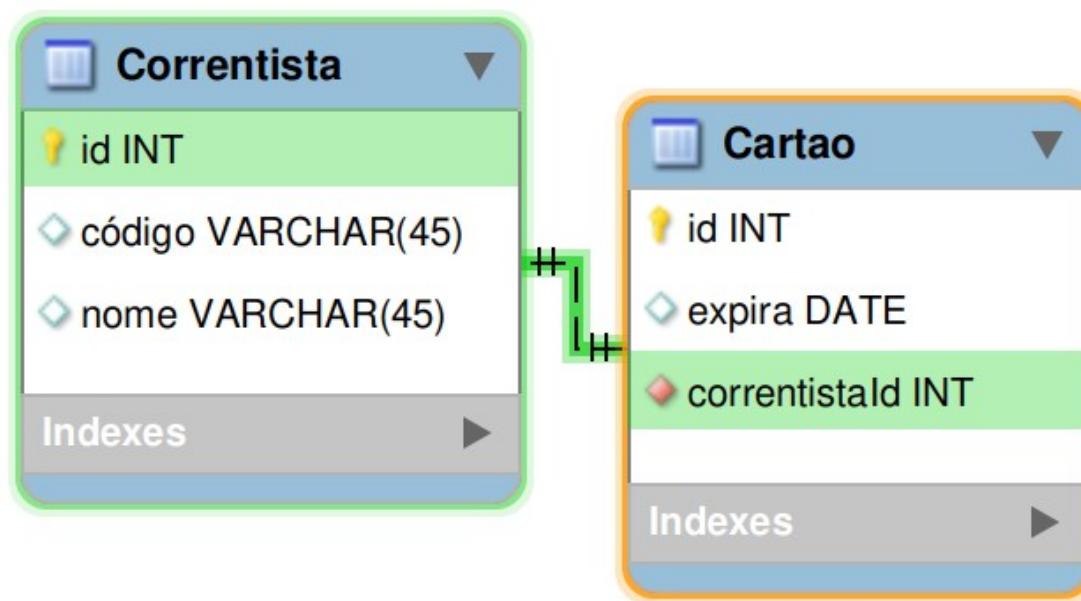
Associações

- Além de mapear tabelas para models, também:
 - Essas associações são definidas nos próprios **modelos**, no método **associate**
 - Por exemplo, em uma aplicação com Sequelize:
- ```
// arquivo app/models/area.js
'use strict';
const { Model } = require('sequelize');
module.exports = (sequelize, DataTypes) => {
 class Area extends Model {
 static associate(models) {
 // define association here
 }
 };
 Area.init({
 nome: DataTypes.STRING
 }, {
 sequelize,
 modelName: 'Area',
 });
 return Area;
};
```
- 
- As associações são definidas nos próprios **modelos**, no método **associate**

# Associações Um para Um



- Em um associação um para um, uma instância de um modelo está associada com **apenas uma** instância de outro modelo
- Por exemplo, no esquema abaixo, um correntista tem apenas um cartão, e um cartão está associado a apenas um correntista



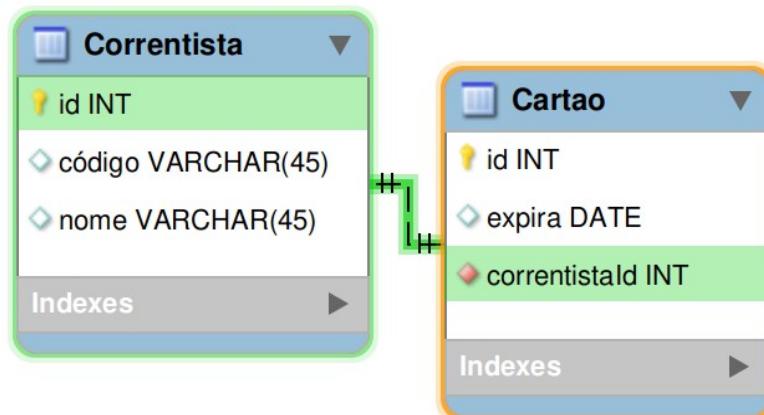
# Associações Um para Um



- Para definirmos uma **associação um para um** no Sequelize, usamos os métodos **belongsTo** e **hasOne**

```
// app/models/cartao.js
static associate(models) {
 this.belongsTo(models.Correntista);
}
```

```
// app/models/correntista.js
static associate(models) {
 this.hasOne(models.Cartao);
}
```



# Associações Um para Um

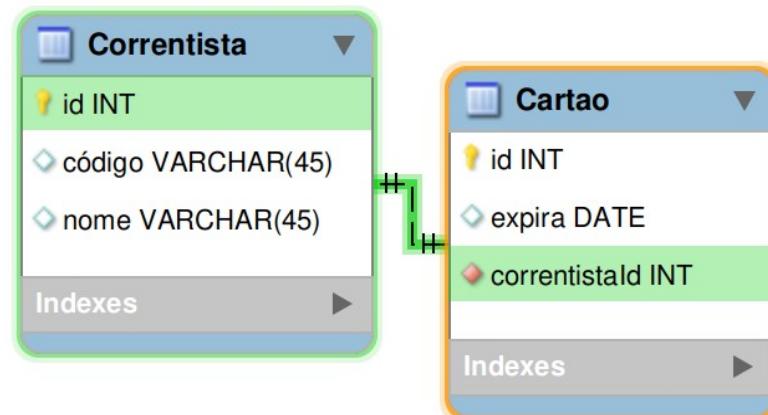


- Para definirmos uma **associação um para um** no Sequelize, usamo

```
// app
static
 this
};

// app
static
 this
}
```

Caso a chave estrangeira não siga o padrão do **sequelize (modelId)**, precisaremos informar seu nome nos métodos **hasOne** e **belongsTo**



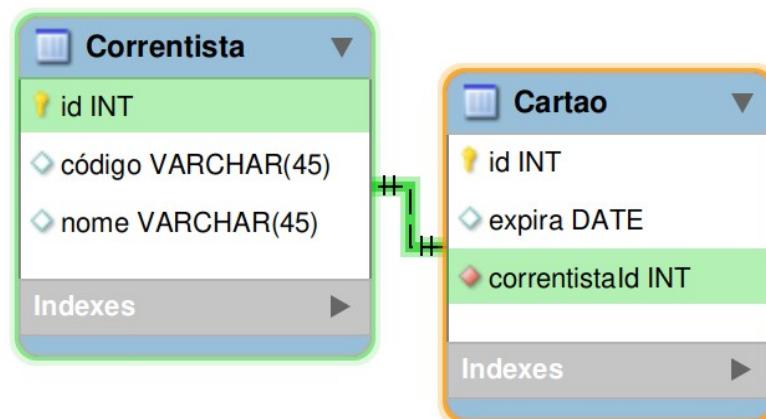
# Associações Um para Um



- Para carregar os dados do cartão de um correntista, podemos usar a função **findAll** (ou **findOne**, etc) com a opção **include**:

```
const correntista = await Correntista.findByPk(123, {
 include: models.Cartao,
});
```

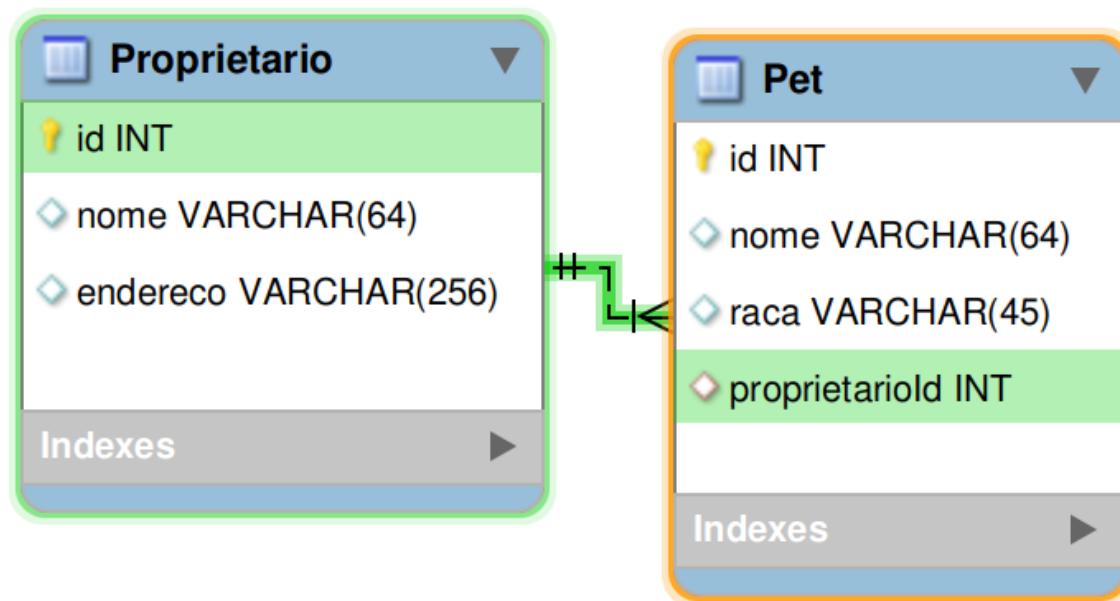
- No exemplo acima, **correntista.Cartao** será um objeto contendo todos os dados do cartão do correntista **id = 123**



# Associações Um para Muitos



- Em uma associação um para muitos, uma instância de um modelo pode estar associada à várias instâncias do outro modelo
- Por exemplo, no esquema abaixo, um proprietário pode ter vários pets (animais de estimação)



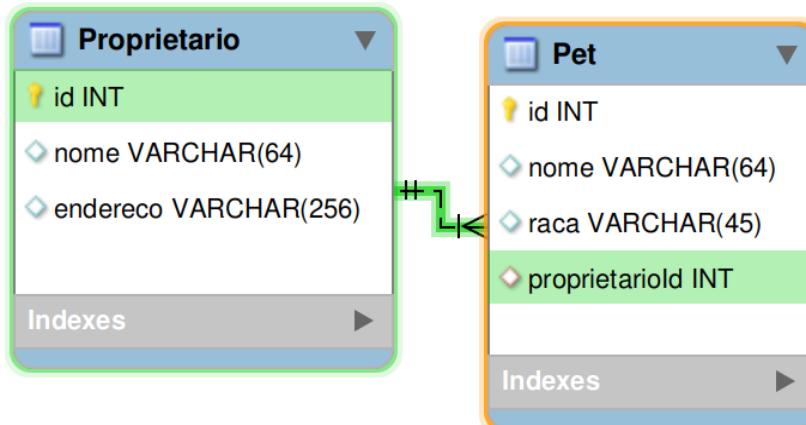
# Associações Um para Muitos



- Para definirmos uma associação um para muitos no sequelize, usamos os métodos **belongsTo** e **hasMany**

```
// app/models/proprietario.js
static associate(models) {
 thishasMany(models.Pet);
}
```

```
// app/models/pet.js
static associate(models) {
 this.belongsTo(models.Proprietario);
}
```



# Associações Um para Muitos

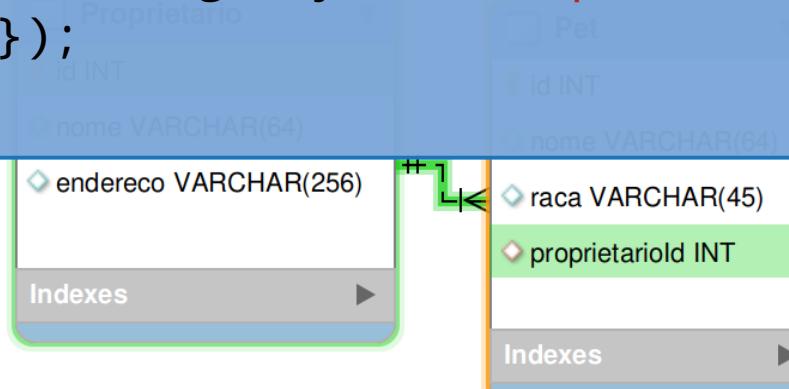


- Para definirmos uma associação um para muitos no sequelize, usamos os métodos **belongsTo** e **hasMany**

```
// app/models/proprietario.js
static associate(models) {
 this.belongsTo(models.Pet);
}
```

Caso a chave estrangeira não siga o padrão do **sequelize**, precisaremos informar seu nome nos métodos **hasMany** e **belongsTo**

```
// app/models/pet.js
static associate(models) {
 static associate(models) {
 this.belongsTo(models.Proprietario{
 foreignKey: 'idProprietario'
 });
 }
}
```



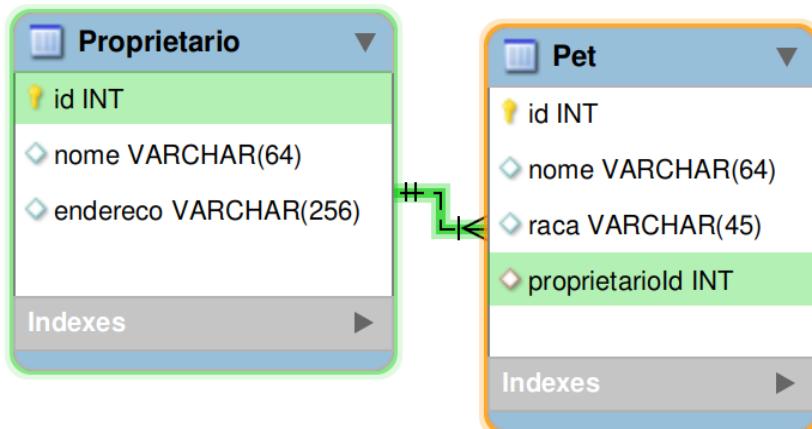
# Associações Um para Muitos



- Podemos carregar os pets de um dado proprietário através da função **findAll** (ou **findOne**, **findByIdk**, etc) com a opção **include**

```
const proprietario = await Proprietario.findByIdk(123, {
 include: models.Pet
});
```

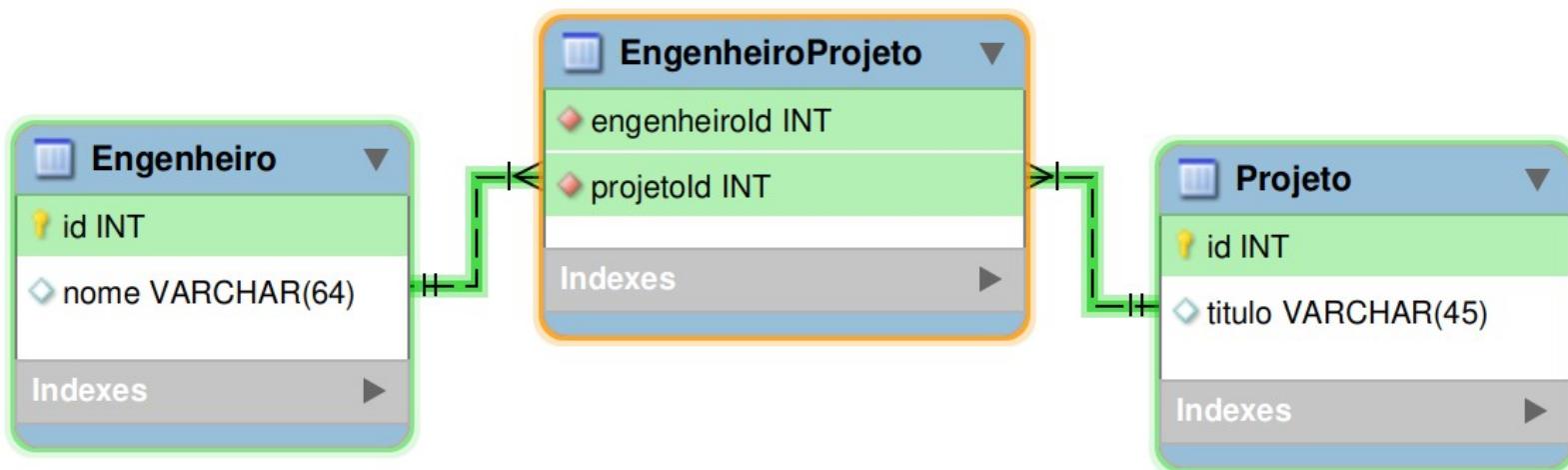
- No exemplo acima, **proprietario.Pet** é um array de objetos contendo todos os pets do proprietário com **id = 123**



# Associações Muitos para Muitos



- Em uma associação muitos para muitos, um registro pode ser associado com **muitos outros e vice-versa**
- Por exemplo, no esquema abaixo, um engenheiro pode ter muitos projetos e um projeto pode ter vários engenheiros



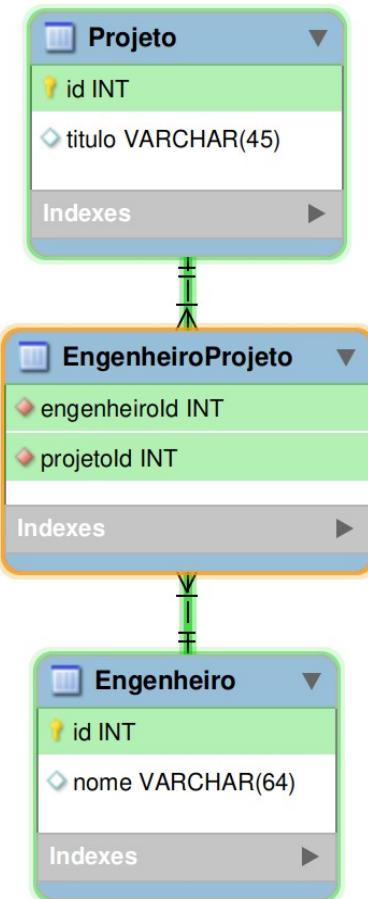
# Associações Muitos para Muitos



- Para definirmos uma associação muitos para muitos, usamos os o método **belongsToMany** nos dois lados da associação

```
// app/models/engenheiro.js
static associate(models) {
 this.belongsToMany(models.Projeto, {
 through: 'EngenheiroProjeto'
 });
}
```

```
// app/models/projeto.js
static associate(models) {
 this.belongsToMany(models.Engenheiro, {
 through: 'EngenheiroProjeto'
 });
}
```



# Associações Muitos para Muitos



- Para definirmos uma associação muitos para muitos, usamos os o método **belongsToMany** nos dois lados da associação

```
// app/models/engenheiro.js
static associate(models) {
 this.belongsToMany(models.projeto, {
 through: 'EngenheiroProjeto',
 foreignKey: 'idEngenheiro',
 otherKey: 'idProjeto'
 });
}
```

Caso as chaves estrangeiras não sigam o padrão do **sequelize**, precisaremos informar as chaves nos métodos **belongsToMany** dos dois modelos



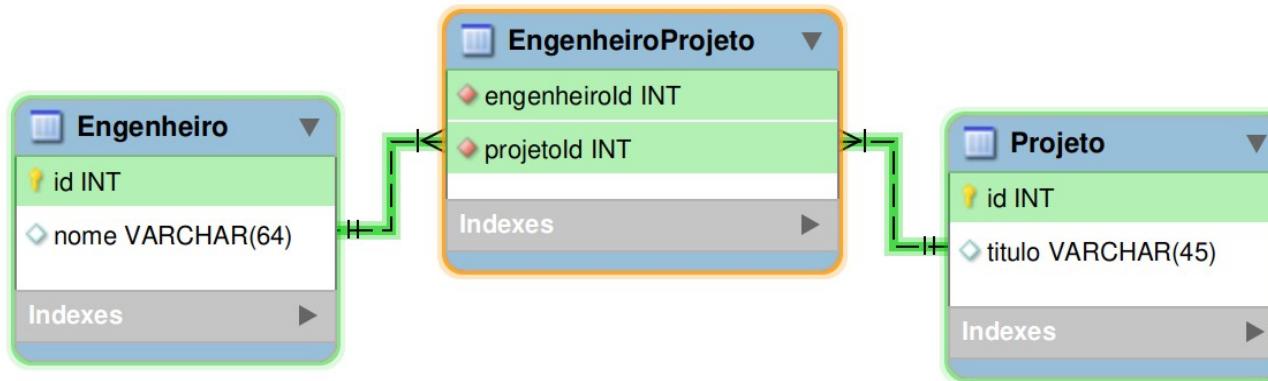
# Associações Muitos para Muitos



- Podemos carregar os projetos de um engenheiro através da função **findAll** (ou **findOne**, **findById**, etc) com a opção **include**

```
const engenheiro = await Engenheiro.findById(123, {
 include: models.Projeto
});
```

No exemplo acima, **engenheiro.Projeto** é um array de objetos contendo todos os projetos do engenheiro com **id = 123**



x □ - ⌂ Bootswatch: Darkly x +

localhost:3000/curso/1

## Vigilantes da Floresta

Ciência da Computação  
editar 

Sigla: IE10

Nome: Ciência da Computação

Descrição: Curso dos alunos top!!

Área: Ciências Exatas

**Exercício 9:** Usando as associações entre as tabelas, mudar a rota **/curso/:id** para que ela mostre o nome da área do curso que está sendo mostrado.

  
**Game**

Back to top

Blog

express JS

# Cookies e Sessões

- HTTP é um protocolo que não mantém estado, isto é **não mantém uma conexão**
- Cada pedido que um browser faz ao servidor Web é independente do pedido anterior
- No entanto, muitas aplicações necessitam manter o estado do usuário enquanto ele navega de uma página para outra
  - Ex: carrinho de compras em sites de comércio eletrônico
- As aplicações possuem duas opções para manter as informações de estado dos clientes:
  - **Cookies**: mantém informações de estado no cliente (browser)
  - **Sessões**: matém informações de estado no lado servidor

# Cookies



- **Cookies** são variáveis enviadas pelo servidor Web para o browser através do **protocolo HTTP**
  - Ficam armazenados no lado cliente
  - São enviados para o servidor em futuros acessos do browser

## Requisições do Browser

1 Get index.html HTTP1.1  
Host: www.icomp.ufam.edu.br

3 Get index.html HTTP1.1  
Host: www.icomp.ufam.edu.br  
**Cookie: usuario=3452**

## Requisições de icomp.ufam.edu.br

2 HTTP1.1 200 OK  
**Set-Cookie: usuario=3452**  
... Conteúdo da página...

4 HTTP1.1 200 OK  
... Conteúdo da página...

# Cookies



- Para habilitar o uso de cookies por sua aplicação, é necessário instalar o middleware **cookie-parser**

```
$ npm install cookie-parser
```

- Para usar o middleware, precisamos dar um **require** no módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo app.js
const cookieParser = require('cookie-parser');
app.use(cookieParser());
```

# Cookies



- A partir deste momento, podemos i) criar novos cookies e ii) identificar os cookies enviados pelo browser para o servidor
  - O array **req.cookies** armazena os cookies enviados pelo cliente
  - O método **res.cookie** é usado para enviar um pedido de criação de um novo cookie no lado cliente (browser)

```
app.get('/', function (req, res) {
 if (!('nome' in req.cookies)) {
 res.cookie('nome', 'valor');
 res.send('Você NUNCA passou por aqui!');
 } else {
 res.send('Você JÁ passou por aqui');
 }
});
```

# Cookies



- Aprendendo
- Você NUNCA passou por aqui!
- Crie um cookie

Ao acessar a página pela primeira vez, aparece a mensagem: **Você nunca passou por aqui.**

Requisição para criação do novo cookie

The screenshot shows a browser window with the URL `localhost:3456`. The Network tab of the developer tools is selected, showing a single request for the page. The Headers section of the response shows the following:

| Header          | Value                              |
|-----------------|------------------------------------|
| HTTP/1.1 200 OK |                                    |
| X-Powered-By    | Express                            |
| Set-Cookie      | nome=valor; Path=/                 |
| Content-Type    | text/html; charset=utf-8           |
| Content-Length  | 28                                 |
| ETag            | W/"1c-CrmtjUEnTwERpXnZP/QQReCSL8s" |
| Date            | Thu, 10 Oct 2019 12:42:42 GMT      |

# Cookies



- Aprendendo sobre cookies
- Crie uma aplicação que imprime na tela: "Você JÁ passou por aqui"
- Crie um cookie que imprime a mesma mensagem

Ao acessar a página pela segunda vez, aparece a mensagem: **Você já passou por aqui.**

})

The screenshot shows a browser window with the URL `localhost:3456`. The developer tools Network tab is open, showing a single request to `localhost`. The response headers include:

| Name            | Value                                                    |
|-----------------|----------------------------------------------------------|
| Set-Cookie      | seen=true; expires=Thu, 10-Oct-2019 12:45:22 GMT; path=/ |
| HTTP/1.1 200 OK |                                                          |
| X-Powered-By    | Express                                                  |
| Content-Type    | text/html; charset=utf-8                                 |
| Content-Length  | 25                                                       |
| ETag            | W/"19-rtc4rRrLzAls30v/0he4PTSBsm8"                       |
| Date            | Thu, 10 Oct 2019 12:45:22 GMT                            |
| Connection      | keep-alive                                               |

# Cookies



- Também podemos criar cookies com data de expiração

```
// Expira 360000 ms (6 minutos) após ser criado
res.cookie(name, 'value', { expire: 360000 + Date.now() });
```

- Usamos a função **clearCookie** para apagar um cookie já criado

```
const express = require('express')
const cookieParser = require('cookie-parser')
const app = express()

app.use(cookieParser())

app.get('/apaga_cookie', function(req, res){
 res.clearCookie('nome');
 res.send('cookie apagado');
});

app.listen(3000);
```

# Cookies



- Também podemos criar cookies com data de expiração

```
// Expira 360000 ms (6 minutos) após ser criado
res.cookie(name, 'value', { expire: 360000 + Date.now() });
```

- Usamos a função **clearCookie** para apagar um cookie já criado

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

Se o cookie for criado sem data de expiração, ele será
apagado após o fechamento da janela do browser
```

```
app.use(cookieParser())

app.get('/apaga_cookie', function(req, res){
 res.clearCookie('nome');
 res.send('cookie apagado');
});

app.listen(3000);
```

# Cross-Site Request Forgery

- Um ataque clássico a aplicações Web é o **Cross Site Request Forgery**, também conhecido pelas siglas **CSRF** ou **XSRF**

[www.bancoficticio.com](http://www.bancoficticio.com)



João faz o login  
em sua conta  
do e-banking



João recebe em um e-mail de spam e clica em um link que redireciona para o site Attacker

[www.attacker.com](http://www.attacker.com)



Attacker aproveita que João está logado no banco e faz um solicitação de transferênci

# Cross-Site Request Forgery

- Um ataque clássico a aplicações Web é o **Cross Site Request Forgery**, também conhecido pelas siglas **CSRF** ou **XSRF**

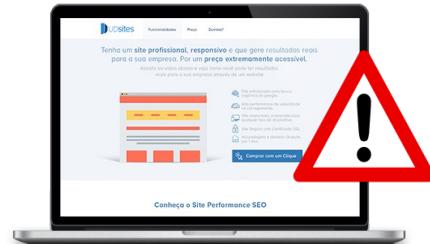
www.bancoficticio.com



João faz o login  
em sua conta  
do e-banking



www.attacker.com



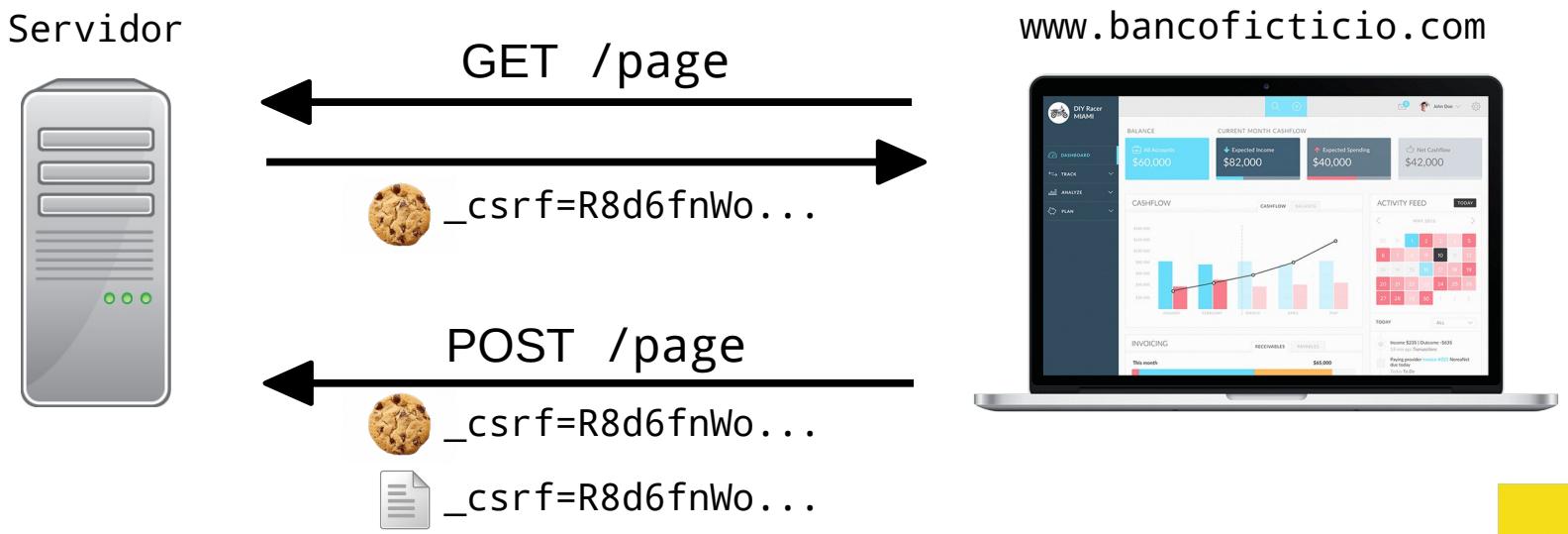
Attacker aproveita que João  
está logado no banco e faz  
um solicitação de  
transferênci

```
<form action="https://www.bancoficticio.com" method="POST">
 <input name="destinatario" value="attacker">
 <input name="quantia" value="10.000,00">
</form>
<script>
 document.forms[0].submit()
</script>
```



# Cross-Site Request Forgery

- Uma forma de proteger os usuários de sua aplicação Web desse tipo de ataque é através dos **tokens CSRF**
- Um token CSRF é um conjunto de 24 caracteres gerado aleatoriamente – Ex: Jq6nXu-4oqaEDluG7XOCCWO
- Um novo token CSRF é enviado como cookie para o browser sempre que o usuário acessar uma página do servidor Web



# Cross-Site Request Forgery

- Além do cookie, o token CSRF também é colocado como um **input hidden** de todo formulário disponível no servidor
- ```
<input type="hidden" name="_csrf" value="{{csrf}}>
```
- Portanto, quando o usuário preenche o formulário, o CSRF é enviado para o server como cookie e como input do formulário



Cross-Site Request Forgery

- Após a submissão de um formulário, o servidor irá comparar o CSRF vindo do formulário com o CSRF vindo do cookie
- Os dados do formulário só serão processados caso os dois CSRFs sejam exatamente iguais

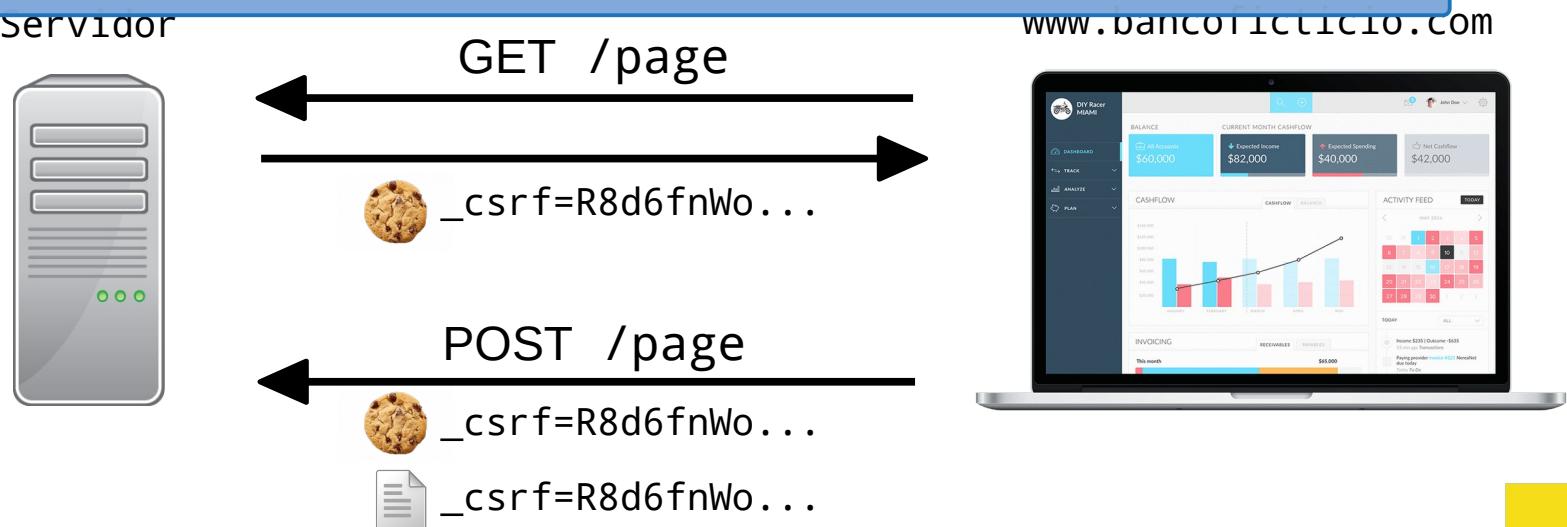


express **JS**

Cross-Site Request Forgery

- Após a submissão de um formulário, o servidor irá comparar o CSRF vindo do formulário com o CSRF vindo do cookie
- Os dados do formulário só serão processados caso os dois CSRFs sejam exatamente iguais

O attacker não será capaz de reproduzir o mesmo comportamento através de formulários ocultos porque ele não é capaz de acessar os cookies dos usuários



Cross-Site Request Forgery

- Para usar o CSRF, precisamos instalar o pacote **csurf**

```
$ npm install csurf
```

- Para usar o middleware, precisamos dar um **require** no módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo app.js
const csurf = require('csurf');
app.use(csurf({ cookie: true }));
```

Cross-Site Request Forgery

My Chess App

localhost:5555

Chess App

App de Xadrez

Back to top Blog RSS Twitter GitHub API

Donate

Made by [Thomas Park](#).
Code released under [MIT](#).
Based on [Bootstrap](#) from [Google](#).

Requisição para criação do cookie _csrf

| Name | Headers | Preview | Response | Cookies | Timing |
|-------------------|---------|---------|----------|---------|--------|
| localhost | | | | | 50 ms |
| main.css | | | | | 100 ms |
| jquery.min.js | | | | | 150 ms |
| popper.min.js | | | | | |
| bootstrap.... | | | | | |
| all.min.js | | | | | |
| logo.png | | | | | |
| css?family... | | | | | |
| S6uyw4BM... | | | | | |
| 10 requests ... | | | | | |

Response Headers

- Connection: keep-alive
- Date: Thu, 10 Oct 2019 13:31:16 GMT
- ETag: W/"9f5-LwrNEdvE8RjpkPCsVoKitQ6dj1w"
- set-cookie: _csrf=jsHBmeJDEYw7_doNMfUZ1PDq; Path=/**
- X-Powered-By: Express

Request Headers

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

express JS

Cross-Site Request Forgery

```
ForbiddenError: invalid csrf token
  at csrf (/home/david/dev/myapp/node_modules/csrf/index.js:14:1)
  at Layer.handle [as handle_request] (/home/david/dev/myapp/node_modules/express/lib/router/layer.js:95:5)
  at trim_prefix (/home/david/dev/myapp/node_modules/express/lib/router/route.js:134:5)
  at /home/david/dev/myapp/node_modules/express/lib/router/route.js:101:15
  at Function.process_params (/home/david/dev/myapp/node_modules/express/lib/router/route.js:123:14)
  at next (/home/david/dev/myapp/node_modules/express/lib/router/route.js:136:14)
  at /home/david/dev/myapp/node_modules/body-parser/lib/read.js:116:5
  at invokeCallback (/home/david/dev/myapp/node_modules/raw-body/index.js:164:16)
  at done (/home/david/dev/myapp/node_modules/raw-body/index.js:144:7)
  at IncomingMessage.onEnd (/home/david/dev/myapp/node_modules/raw-body/index.js:96:7)
```

A partir deste momento, todo formulário submetido sem o token csrf dará o erro **ForbiddenError: invalid csrf token**

Cross-Site Request Forgery

- Para tirar o erro, precisamos devolver o **token csrf** para o servidor após o preenchimento de cada formulário

```
// Arquivo app/controllers/curso.js - Será preciso
// enviar o csrf para a view de todos os formulários
res.render('curso/create', {
  csrf: req.csrfToken()
});
```

```
<!-- Arquivo app/views/curso/create.handlebars -->
...
<input type="hidden" name="_csrf" value="{{csrf}}">
<button type="submit">Adicionar Curso</button>
...
```

Sessões

- Através de sessões, podemos armazenar informações de estado (variáveis) no lado servidor
- Em vez do browser guardar um cookie por dado, ele guarda apenas um cookie contendo um **id de sessão (connect.sid)**

Requisições do Browser

1 Get index.html HTTP1.1
Host: server.com

3 Get index.html HTTP1.1
Host: server.com
Cookie: connect.sid=6Bh

Requisições de server.com

2 HTTP1.1 200 OK
Set-Cookie: connect.sid=6Bh
... Conteúdo da página...

4 HTTP1.1 200 OK
... Conteúdo da página...

Sessão 6Bh
Dados da sessão no servidor
user: Bruna
itens-carrinho: 3
start: 15:30



Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](#)

```
$ npm install uuid
```

- Os UUIDs são valores de 128 bits que podem ser usados como ID únicos de qualquer coisa em sistemas computacionais
 - Ex: f0221c72-ac30-4796-83f5-fd7a8a4f6b15
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](#)

```
$ npm install uuid
```

- Os UUIDs são valores de 128 bits que podem ser usados como Alguns desenvolvedores preferem usar o uuid como **chave primária** de tabelas, ao invés de um ID auto incrementado.
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

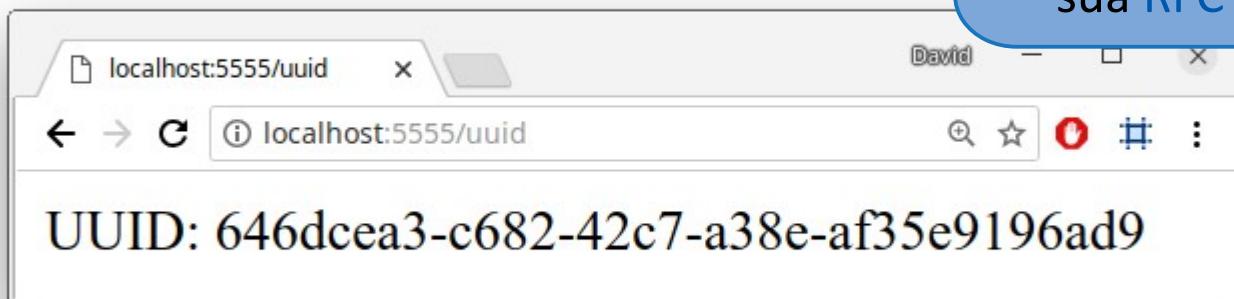
Sessões

- O código abaixo mostra como os UUIDs podem ser gerados a partir do módulo **uuid**

```
const uuid = require('uuid');

app.get('/uuid', function (req, res) {
  const uniqueId = uuid.v4()
  res.send(`UUID: ${uniqueId}`)
})
```

Existem 5 versões de UUID, conforme descrito em sua RFC



Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```

- Para usar o middleware, precisamos dar um **require** no módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo app.js
const session = require('express-session');

app.use(session({
  genid: (req) => {
    return uuid.v4() // usamos UUIDs para gerar os SESSID
  },
  secret: 'Hi9Cf#mK98',
  resave: false,
  saveUninitialized: true
}));
```

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```

- Para usar o middleware, precisamos importá-lo no módulo e adicioná-lo em nossa aplicação:

```
// Arquivo app.js
const session = require('express-session');

app.use(session({
  genid: (req) => {
    return uuid() // Usamos o ID da sessão gerado pelo middleware
  },
  secret: 'Hi9Cf#mK98',
  resave: false,
  saveUninitialized: true
}));
```

Usado para adicionar uma assinatura (similar ao checksum) ao **session.id** enviado para o usuário. Quando o usuário devolve o **session.id**, a assinatura é usada para checar se o **session.id** é válido. Usa uma técnica chamada HMAC.

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```

- Para usar o middleware no módulo e adicioná-lo

```
// Arquivo app.js
const session = require('express-session');

app.use(session({
  genid: (req) =>
    return uuid(),
  secret: 'Hi9cf#mK98',
  resave: false,
  saveUninitialized: true
}));
```

Quando **true**, a sessão do usuário é salva a cada requisição, mesmo que os dados da sessão não tenham sido modificados durante a requisição. Isso mantém a sessão ativa, visto que ela pode ser deletada após algum tempo de desuso.

no modo **use**

SID

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```

- Para usar o middleware no seu código, é necessário importá-lo e adicioná-lo ao seu app.js

```
// Arquivo app.js
const session = require('express-session');

app.use(session({
  genid: (req) =>
    return uuid(),
  secret: 'Hi9cf#mK8',
  resave: false,
  saveUninitialized: true
}));
```

Usado para adicionar uma sessão ao usuário. É necessário que o usuário esteja logado ao sistema.

Quando **true**, a sessão do usuário é criada automaticamente quando o usuário faz login. Quando **false**, a sessão só é criada quando o usuário faz logout.

Quando **true**, a sessão é salva no store. Quando **false**, a sessão não é salva no store.

My Chess App x

localhost:5555 David

Chess App

App de Xadrez

Back to top Blog RSS Twitter GitHub API Donate

Made by Thomas Park.
Code released under the [MIT License](#).
Based on [Bootstrap](#). Icons from [Font Awesome](#). Web fonts from [Google](#).

Elements Console Sources Network »

Filter Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other

100 ms 200 ms 300 ms 400 ms 500 ms

| Name | Headers | Preview | Response | Cookies | Timing |
|----------------|---------|---------|----------|---------|--------|
| all.min.js | | | | | |
| bootstrap.... | | | | | |
| css?family=... | | | | | |
| jquery.min.js | | | | | |
| localhost | | | | | |
| logo.png | | | | | |
| main.css | | | | | |
| popper.min.js | | | | | |
| S6uyw4BM... | | | | | |

▼ Response Headers view source

Connection: keep-alive
Date: Wed, 16 Oct 2019 18:58:41 GMT
ETag: W/"a6b-StlpLtllyzJ6ZgamZZq8waJhhI"
Set-Cookie: _csrf=_b20NIx5X2PmjT0kvQch5CP8; Path=/
Set-Cookie: connect.sid=s%3Ad29aa900-fbb5-49e8-8cee-554c31e34109.9l9VxDsrqjttzu%2FCweCngMudRpY%2B1m3wn0T31dCZj2o; Path=/; HttpOnly
X-Powered-By: Express

9 / 10 requests...

Cadastro de Usuários

- Antes de falar de sessões, que serão usadas nas actions de login e logout, vamos abordar sobre o cadastro de usuários
- O cadastro de usuário envolve, dentre outras coisas, a geração de um senha criptografada e seu registro no banco de dados
- O formulário de cadastro de usuário possui os seguintes campos:
 - **Nome completo** – input text
 - **Endereço de e-mail** – input e-mail
 - **Seu curso na UFAM** – select
 - **Escolha uma senha de acesso** – input password
 - **Confirme sua senha** – input password
 - **Eu li e concordo com os termos de serviço** – radio box

- Antes de falar sobre o login e logon
- O cadastro de usuários é feito com a criação de um senha
- O formulário de criação de usuário tem os seguintes campos:
 - **Nome completo**
 - **Endereço de E-mail**
 - **Seu curso na UFAM**
 - **Escolha uma senha de acesso**
 - **Confirme sua senha**
 - **Eu li e concordo com os termos de serviço.**
- **Eu li e concordo com os termos de serviço.**

The screenshot shows a dark-themed web application window titled 'Chess App'. At the top left is a knight chess piece icon. The main title 'Criar conta' is centered above a sub-instruction: 'Deseja criar uma conta? Basta preencher o formulário abaixo.' Below this are several input fields: 'Nome Completo' with placeholder 'Jorge A. Menezes'; 'Endereço de E-mail' with placeholder 'jorge@example.com'; 'Seu curso na UFAM' with a dropdown placeholder 'Selecione seu curso'; 'Escolha uma senha de acesso' with a redacted password field; 'Confirme sua senha' with another redacted password field; and a checkbox for accepting terms with the label 'Eu li e concordo com os termos de serviço.' at the bottom. A large blue 'Criar conta' button is at the bottom center, and a link 'Já possui uma conta? Faça o login.' is at the bottom right.

Criar conta

Deseja criar uma conta? Basta preencher o formulário abaixo.

Nome Completo

Jorge A. Menezes

Endereço de E-mail

jorge@example.com

Seu curso na UFAM

Selecione seu curso

Escolha uma senha de acesso

Confirme sua senha

Eu li e concordo com os [termos de serviço](#).

Criar conta

Já possui uma conta? [Faça o login](#).

- Antes de falar sobre login e logout
- O cadastro de um usuário
- O usuário
- Endereço de e-mail
- Seu curso na faculdade
- Escolha uma senha
- Confirme sua senha
- Eu li e concordo com os termos de serviço

Criar conta

Deseja criar uma conta? Basta preencher o formulário abaixo.

Nome Completo

Escolha uma senha de acesso

Confirme sua senha

Eu li e concordo com os [termos de serviço](#).

Criar conta

Já possui uma conta? [Faça o login](#).

- Antes de falar sobre login e logout
- O cadastro de um usuário
- O que é necessário para validar os campos
- Seu nome
- Escolha uma senha
- Confirme sua senha
- Eu li e concordo com os termos de serviço.

Criar conta

Deseja criar uma conta? Basta preencher o formulário abaixo.

Nome Completo

Jorge A. Menezes

Confirme sua senha

.....

Eu li e concordo com os [termos de serviço](#).

Criar conta

Já possui uma conta? [Faça o login](#).

Criptografando as senhas

- Após a submissão do formulário, é necessário criptografar a senha antes de salvá-la no banco de dados
- Mas por quê? Por que não guardar a senha crua?
 - Todas as pessoas com acesso ao banco poderiam ver a senha
 - Os usuários frequentemente **usam a mesma senha** em vários sites
 - A senha iria aparecer nos **backups** do banco
 - Se o banco estiver na **cloud**, as senhas ficariam expostas na web
 - As senhas ficariam expostas a **ataques de SQL-injection**

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

| estado | capital |
|----------------|----------------|
| Rio de Janeiro | Rio de Janeiro |
| Amazonas | Manaus |
| Minas Gerais | Belo Horizonte |
| Ceará | Fortaleza |

Tabela Usuario

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Busca da capital pelo estado

Informe o estado

Submit

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

| estado | capital |
|----------------|----------------|
| Rio de Janeiro | Rio de Janeiro |
| Amazonas | Manaus |
| Minas Gerais | Belo Horizonte |
| Ceará | Fortaleza |

Tabela Usuario

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Busca da capital pelo estado

Submit

```
SELECT estado, capital FROM estado  
WHERE estado = 'Amazonas'
```

| estado | capital |
|----------|---------|
| Amazonas | Manaus |

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

| estado | capital |
|----------------|----------------|
| Rio de Janeiro | Rio de Janeiro |
| Amazonas | Manaus |
| Minas Gerais | Belo Horizonte |
| Ceará | Fortaleza |

Tabela Usuario

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit 

SELECT estado, capital FROM Estado
WHERE estado = " UNION SELECT login,
senha FROM Usuario WHERE login != "

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

| estado | capital |
|----------------|----------------|
| Rio de Janeiro | Rio de Janeiro |
| Mato Grosso | Brasília |
| Pará | Belém |
| Maranhão | Fortaleza |
| Ceará | Fortaleza |

Tabela Usuario

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Os ataques de SQL-Injection só são possíveis se o invasor conseguir incluir caracteres como ' (aspas simples), " (aspas duplas) e \b (caracter de backspace) nos inputs dos formulários.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit 

SELECT estado, capital FROM estado
WHERE estado = " UNION SELECT login,
senha FROM Usuario WHERE login != "

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

| estado | capital |
|----------------|----------------|
| Rio de Janeiro | Rio de Janeiro |
| Mato Grosso | Cuiabá |
| Paraná | Foz do Iguaçu |

Tabela Usuario

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Ao inserir o valor Rio de Janeiro no campo 'Estado', o resultado é o login 'alberto' e a senha 'flamengo'. O Sequelize resolve este problema adicionando um caractere de **escape** a tais elementos. Por exemplo, o caractere ' vira \', " vira \" e \b vira \\b. Vide [função de escape de Sequelize](#). Mesmo assim, é sempre saudável fazer validação dos inputs.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit 

SELECT estado, capital FROM estado
WHERE estado = " UNION SELECT login,
senha FROM Usuario WHERE login != "

| login | senha |
|----------|-----------|
| alberto | flamengo |
| maria | teste123 |
| fernanda | RmJ&AnhK@ |
| matheus | pokemon |

Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
 - Ex: md5, sha1, sha256, etc
- São funções de uma **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado

```
david@coyote: ~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote: ~$ echo -n "ufam" | md5sum  
8996fe161805927c27a92fae2ca238d2 -  
david@coyote: ~$ echo -n "computacao" | md5sum  
d0cc5ed8aecb1898032c48af57057b9f -  
david@coyote: ~$ □
```

Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
 - Ex: md5, sha1, sha256, etc
- São funções de uma **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado

The screenshot shows a terminal window with two tabs. The top tab is active and displays the command: `david@coyote:~$ echo -n "minhasenha" | md5sum`. The output is `7c6`. The bottom tab is inactive and displays the command: `david@coyote:~$ echo -n "minhasenha" | sha1sum`. The output is `93d51f52fbfe1e944f084727df24993e88caee7`. Below these, two more commands are shown: `david@coyote:~$ echo -n "ufam" | sha1sum` and `david@coyote:~$ echo -n "computacao" | sha1sum`, both resulting in long hash strings.

```
david@coyote:~$ echo -n "minhasenha" | md5sum
7c6
david@coyote:~$ echo -n "minhasenha" | sha1sum
93d51f52fbfe1e944f084727df24993e88caee7
david@coyote:~$ echo -n "ufam" | sha1sum
4ae250e25cb5e0bdb06cfcc2513cb451f4a610d0e
david@coyote:~$ echo -n "computacao" | sha1sum
915769113d302e7893cc3019c5fc5dcb36682bca
david@coyote:~$
```

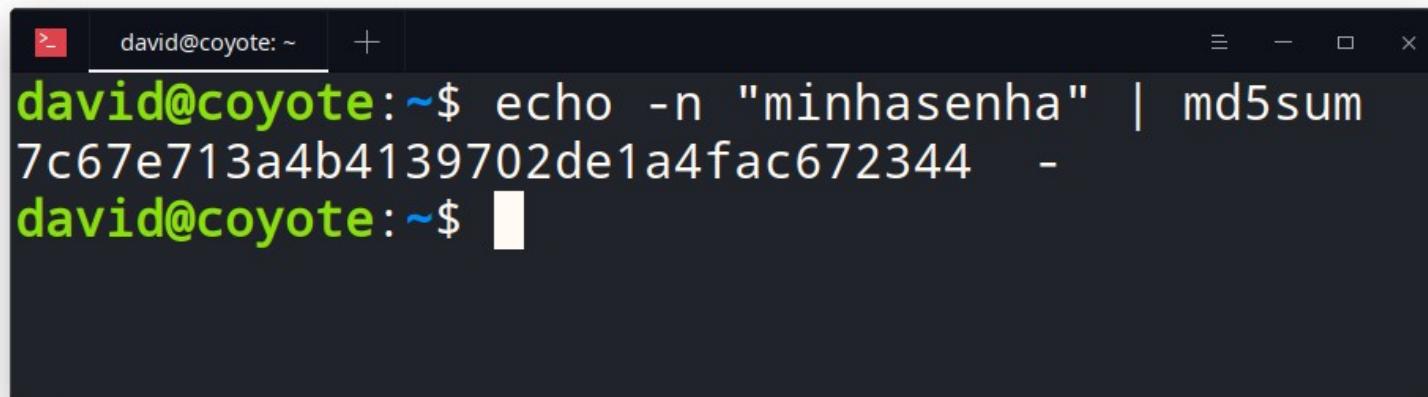
Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
 - Os comandos **md5sum** e **sha1sum** são instalados por padrão em sistemas UNIX , GNU/Linux e BSD. Eles são usados para verificar
- S a integridade de dados transmitidos através da Web. Para r maiores informações, vide [Soma de Verificação \(Checksum\)](#)

The screenshot shows a terminal window with two tabs. The top tab is active and displays the command: `david@coyote:~$ echo -n "minhasenha" | md5sum`. The output is `7c6`. The bottom tab is inactive and displays the command: `david@coyote:~$ echo -n "minhasenha" | sha1sum`. The output is `93d51f52fbfe1e944f084727df24993e88caee7`. Below these, two more commands are shown: `david@coyote:~$ echo -n "ufam" | sha1sum` (output: `4ae250e25cb5e0bdb06cfcc2513cb451f4a610d0e`) and `david@coyote:~$ echo -n "computacao" | sha1sum` (output: `915769113d302e7893cc3019c5fc5dcb36682bca`). Both tabs have the prompt `david@coyote:~$`.

Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
 - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
 - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco



```
david@coyote:~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ █
```

Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
 - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
 - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que **ninguém** poderá descobrir a senha do usuário caso acesso a tabela Usuário do banco de dados

```
david@coyote:~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ █
```

Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
 - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
 - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que ninguém poderá descobrir a senha do usuário caso a Será mesmo? Então vamos no Google e inserir a string hash **7c67e713a4b4139702de1a4fac672344** no campo de busca

```
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ █
```

Funções HASH

- Considerando criptografia:
 - Ex: se a senha é davi... no banco de dados
 - Quando eu digitar "davi..." no meu formulário de login, o sistema vai gerar um hash, que é a senha digitada pelo usuário. Note que a senha digitada é "david", mas o resultado da função hash é "7c67e713a4b4139702de1a4fac672344".

7c67e713a4b4139702de1a4fac672344

Todas Maps Vídeos Imagens Shopping Mais Configurações

Aproximadamente 43 resultados (0,27 segundos)

7c67e713a4b4139702de1a4fac672344 - Hash Toolkit
<https://hashtoolkit.com/reverse-md5-hash> › 7c67e71... - Traduzir esta página
Decrypt md5 Hash Results for: 7c67e713a4b4139702de1a4fac672344 ... md5,
7c67e713a4b4139702de1a4fac672344, minhasenha ...

Hash Md5: 7c67e713a4b4139702de1a4fac672344 - MD5Hashing.net
<https://md5hashing.net/hash> › 7c67e713a4b4139702de1a4fac672344
Decoded hash Md5: 7c67e713a4b4139702de1a4fac672344: minhasenha.

Usando o md5 - Recursos do PHP
[recursosdophp.blogspot.com](http://recursosdophp.blogspot.com/2011/02/normal-0-21-false-false-false-pt-br-x.html) › normal-0-21-false-false-false-pt-br-x
md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você mudar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .

Google

Funções HASH

A screenshot of a web browser window titled "Best MD5 Password Decoder". The URL in the address bar is <https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344>. The page title is "Hash Toolkit". Below the title, it says "Search in 15,786,077,212 decrypted md5 / sha1 hashes.". A search bar contains the hash value "7c67e713a4b4139702de1a4fac672344". To the right of the search bar is a magnifying glass icon. Below the search bar, the text "Decrypt md5 Hash Results for: 7c67e713a4b4139702de1a4fac672344" is displayed. A table follows, with columns "Algorithm", "Hash", and "Decrypted". The first row shows "md5" in the "Algorithm" column, "7c67e713a4b4139702de1a4fac672344" in the "Hash" column, and "minhasenha" in the "Decrypted" column. To the right of the "Decrypted" column is a yellow arrow pointing left.

| Algorithm | Hash | Decrypted |
|-----------|----------------------------------|------------|
| md5 | 7c67e713a4b4139702de1a4fac672344 | minhasenha |

recursosdophp.blogspot.com › normal-0-21-false-false-false-pt-br-x ▾

md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você... dar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



Funções HASH

Best MD5 Password Decoder

Secure | https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344

Hash Toolkit

Search in 15,786,077,212 decrypted md5 / sha1 hashes.

Sistemas como o **Hash Toolkit** usam tabelas contendo bilhões de valores de hash pré-calculados. Essas tabelas normalmente são chamadas de **rainbow tables**.

Decrypt md5 Hash Results for: **7c67e713a4b4139702de1a4fac672344**

| Algorithm | Hash | Decrypted |
|-----------|----------------------------------|----------------|
| md5 | 7c67e713a4b4139702de1a4fac672344 | Q minhasenha Q |



recursosdophp.blogspot.com › normal-0-21-false-false-false-pt-br-x

md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você... dar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma sequência adicional de caracteres aleatórios, chamada salt
- Para exemplificar, podemos aplicar o salt **Us8#upK12MjsM** à senha original do usuário, **minhasenha**

```
david@coyote: ~$ echo -n "Us8#upK12MjsMminhasenha" | md5sum  
d3eeb71a83744a4bbaa3ce41be87a292 -  
david@coyote: ~$ █
```

Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma sequência aleatória de caracteres.

A screenshot of a Google search results page. The search bar at the top contains the text "d3eeb71a83744a4bbbaa3ce41be87a292". Below the search bar, the navigation menu includes "Todas", "Maps", "Vídeos", "Imagens", "Shopping", "Mais", and "Configurações". The main search results area displays the message: "Sua pesquisa - d3eeb71a83744a4bbbaa3ce41be87a292 - não encontrou nenhum documento correspondente." Below this, under "Sugestões:", there is a bulleted list:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

The Google logo is visible at the bottom right of the search results page.

Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma string aleatória.
- De uma forma geral, o salt é gerado aleatoriamente para cada usuário e armazenado no banco junto com o hash MD5 resultante:
Us8#upK12MjsMd3eeb71a83744a4bbaa3ce41be87a292

Sua pesquisa - **d3eeb71a83744a4bbaa3ce41be87a292** - não encontrou nenhum documento correspondente.

Sugestões:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

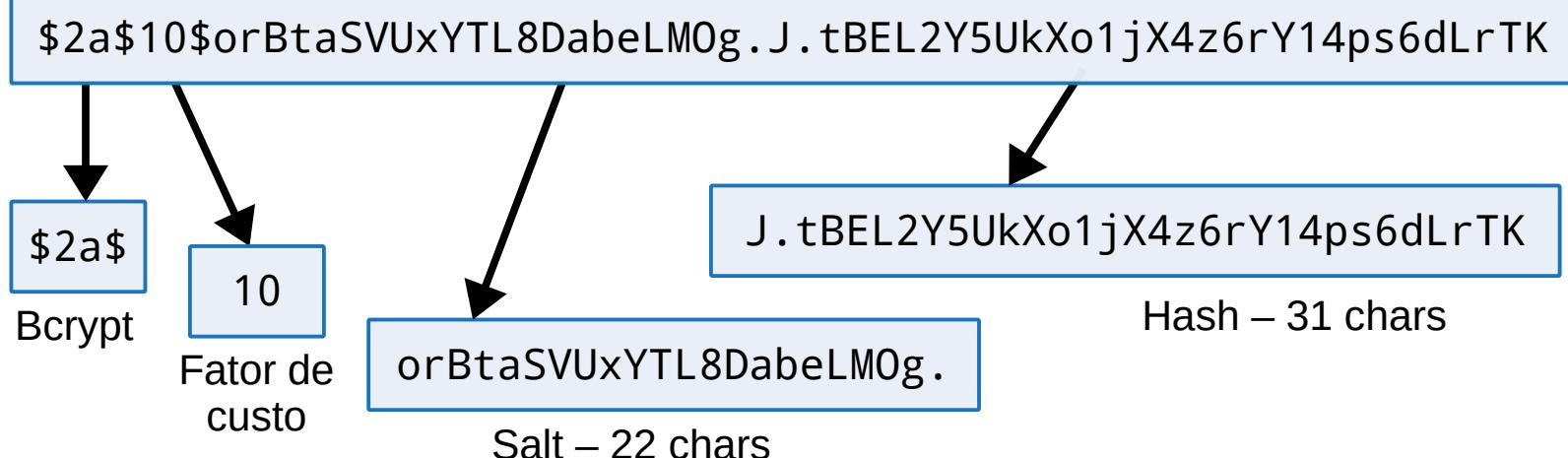


O módulo bcrypt

- O módulo **bcrypt** é uma boa opção para geração de senhas, pois incorpora uma função hash à uma estratégia de salt

```
$ npm install bcryptjs
```

- O bcrypt é o **algoritmo de hash** usado para geração de senhas em sistemas como OpenBSD e algumas distribuições linux



O módulo bcrypt

- Para gerar uma senha com salt podemos usar o código abaixo

```
// Arquivo api/controllers/main.js
const bcrypt = require('bcryptjs');

// Dentro da função signup
bcrypt.genSalt(rounds, function(err, salt) {
  bcrypt.hash(req.body.senha, salt, async(err, hash) => {
    await User.create({
      nome: req.body.nome,
      email: req.body.email,
      senha: hash,
      id_curso: req.body.curso
    });
  });
});
```

O módulo bcrypt

- Para gerar uma senha com sal, é só usar o código abaixo

```
// Arquivo api/controllers.js
const bcrypt = require('bcryptjs');

// Dentro da função signup
bcrypt.genSalt(rounds, function(err, salt) {
  bcrypt.hash(req.body.senha, salt, async(err, hash) => {
    await User.create({
      nome: req.body.nome,
      email: req.body.email,
      senha: hash,
      id_curso: req.body.curso
    });
  });
});
```

Número de rounds para geração do hash

Senha informada pelo usuário

O módulo bcrypt

- Para gerar uma senha com sal, é necessário usar o código abaixo

```
// Arquivo api/controllers.js
const bcrypt = require('bcryptjs');

// Dentro da função signup
bcrypt.genSalt(rounds, function(err, salt) {
  bcrypt.hash(req.body.senha, salt, async(err, hash) => {
    await User.create({
      nome: req.body.nome,
      email: req.body.email,
      senha: hash,
      id_curso: req.body.curso
    });
  });
});
```

Número de rounds para geração do hash

informada pelo usuário

| nome | email | senha | id_curso |
|-----------------|-------------------------|---|----------|
| David Fernandes | david@icomp.ufam.edu.br | \$2a\$10\$LDsc/xMa91HiUY03KiQxVuZUJCbN0CNcA6F/k9vPH9H/NII/MTNqO | 2 |

Login de Usuários

- Da mesma forma que o **signup**, a action responsável pelo **login** pode ser colocada no **controlador main**
- Na action login, usamos o método **bcrypt.compare()** para verificar se a senha digitada pelo usuário está correta ou não

```
var user= await User.findOne({where:{email:req.body.email}});  
  
if (user) {  
  bcrypt.compare(req.body.senha, user.senha, (err, ok) => {  
    if (ok) {  
      req.session.uid = user.id;  
      res.redirect('/');  
    } else {  
      res.render('main/login', {  
        csrf: req.csrfToken()  
      });  
    }  
  });  
}
```

Note que, caso o e-mail e senha do usuário estejam corretos, a **variável de sessão uid** é criada

Logout de Usuários

- O controlador main também deverá conter uma action **logout**, que será usada para **encerrar a sessão** do usuário

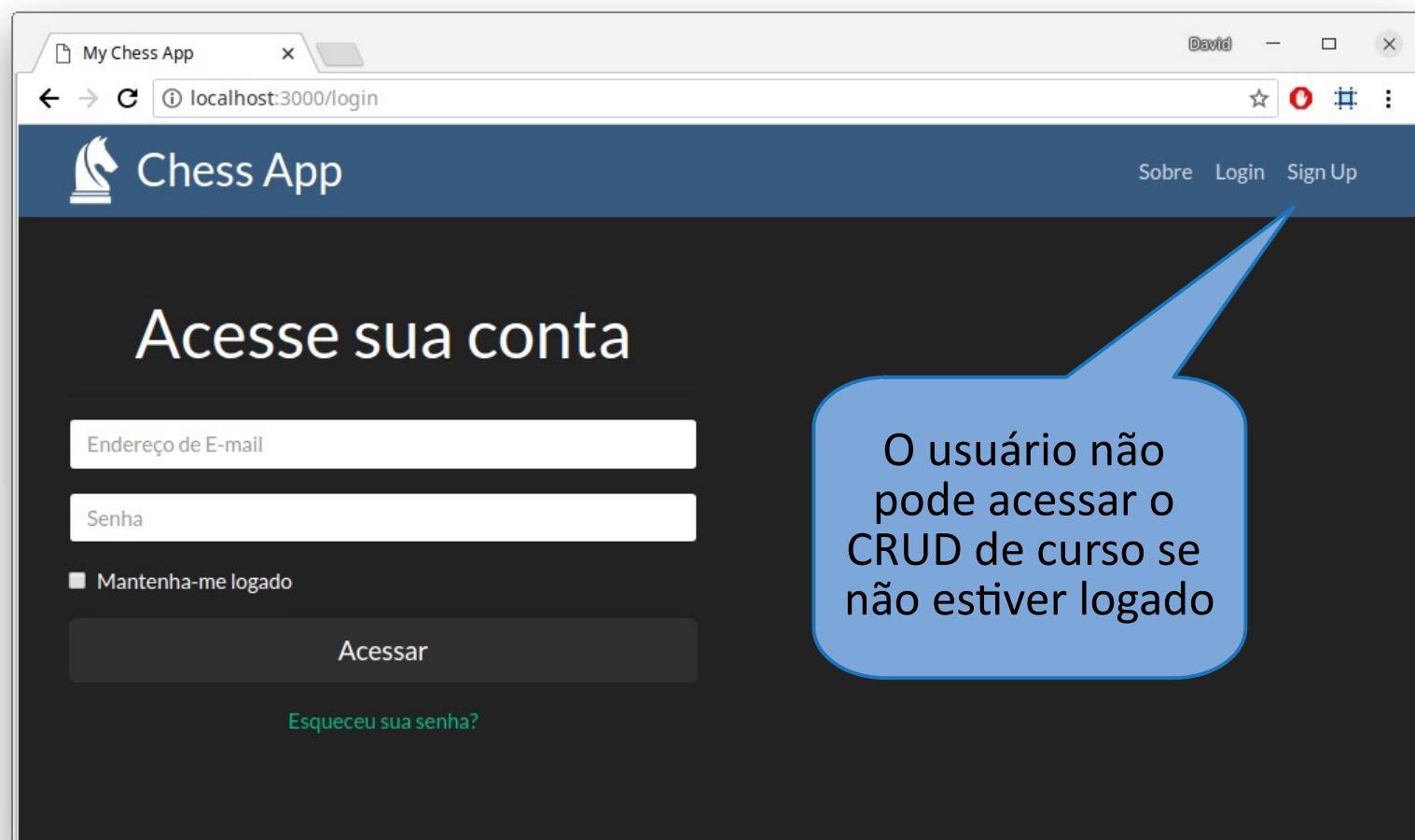


- Para encerrar a sessão, destruindo todas as suas varáveis, usamos o método **req.session.destroy()**

```
const logout = (req, res) => {
  req.session.destroy(function (err) {
    if (err) {
      return console.log(err);
    }
    res.redirect('/');
  });
}
```

Adaptação do menu superior

- Note que o **menu superior** da aplicação é alterado a partir do momento em que o usuário efetua o login no sistema



Adaptação do menu superior

- Note que o **menu superior** da aplicação é alterado a partir do momento em que o usuário efetua o login no sistema

The image shows a screenshot of a chess application interface. At the top, there is a browser window titled "My Chess App" with the URL "localhost:3000/login". The page itself has a dark blue header with a white knight logo and the text "Chess App". On the right side of the header are links for "Sobre", "Login", and "Sign Up". Below the header is a large black area containing a login form. The form includes fields for "Endereço de E-mail" and "Senha", a checkbox for "Mantenha-me logado", and a "Acessar" button. A small link "Esqueceu sua senha?" is located below the button. In the bottom right corner of the main content area, there is a yellow speech bubble containing the text "Após o login o menu é alterado". The overall theme of the application is dark with light-colored text and UI elements.

My Chess App

localhost:3000/login

Chess App

Sobre Login Sign Up

Endereço de E-mail

Senha

Mantenha-me logado

Acessar

Esqueceu sua senha?

Após o login
o menu é
alterado

express JS