CRUD JavaScript

JavaScript Simples
CRUD

JAVASCRIPT PURO

MENU

POR, WANDERSON JONER.

CAPITULO - 1

1 - INSTALANDO NODE.JS

1.1 - Primeiramente é necessário ter instalado o **Node.js**, e já ter feito a modelagem da tabela pessoas com nome, nascimento e cargo no banco de dados mysql 'cadastro' mas o que é "Node.js": Node.js não é um framework Javascript. Ele está mais para uma plataforma de aplicação, na qual você escreve seus programas com Javascript que serão compilados, otimizados e interpretados pela máquina virtual V8. Essa VM é a mesma que o Google utiliza para executar Javascript no browser Chrome, e foi a partir dela que o criador do Node.js, **Ryan Dahl**, criou o projeto. O resultado desse processo híbrido é entregue como código de máquina server-side, tornando o Node.js muito eficiente na sua execução e consumo de recursos suporta 1 milhão de requisições por segundo.
Node.js é uma tecnologia assíncrona que trabalha em uma única thread de execução. Por assíncrona entenda que cada requisição ao Node.js não bloqueia o processo do mesmo, atendendo a um volume absurdamente grande de requisições ao mesmo tempo mesmo sendo single thread.
Você pode baixar ele neste link: https://nodejs.org/en/. Depois de instala-

lo iremos instalar as extensões.

CAPITULO 2

2 - INICIANDO A APLICAÇÃO

2.1 - Instalando Extensões: Entre no CDM ou Terminal, e digite:
2.2 - cd documents, para entrar na pasta documents
2.3 – mkdir projet1;
2.4 – cd projet1
2.5 – npm init

CAPITULO 3

3 - CONHECENDO AS EXTENSÕES

Tenha o YARN instalado em seu terminal. Digite no terminal o comando: **yarn add express ejs mysql body-parser**. Mas antes vamos conhecer um pouco sobre cada extensão que será instalada.

Express: O Express é um framework JavaScript que facilita a criação de aplicativos web utilizando o Node.js.

EJS: O EJS é uma linguagem de templates simples que permite gerar marcação HTML com JavaScript simples. Nenhuma religiosidade sobre como organizar as coisas. Nenhuma reinvenção de iteração e fluxo de controle. É apenas JavaScript.

MYSQL: Para acessar um banco de dados MySQL com o Node.js, você precisa de um driver MySQL. Este tutorial usará o módulo "mysql", baixado do NPM.

Body-Parser: O bodyParser objeto expõe várias fábricas para criar middlewares. Todos os middlewares preencherão a **req.body** propriedade com o corpo analisado quando o Content-Type cabeçalho da solicitação corresponder à type opção ou um objeto vazio ({}) se não houver corpo para analisar, Content-Type se não tiver

correspondência ou se tiver ocorrido um erro.

JAVASCRIPT NA PRÁTICA...

CAPITULO 3

3.1 - INSTALAR EXTENSÕES

Agora que já vimos um pouco sobre
cada extensão, vamos instalar
as extensões como no exemplo logo abaixo:

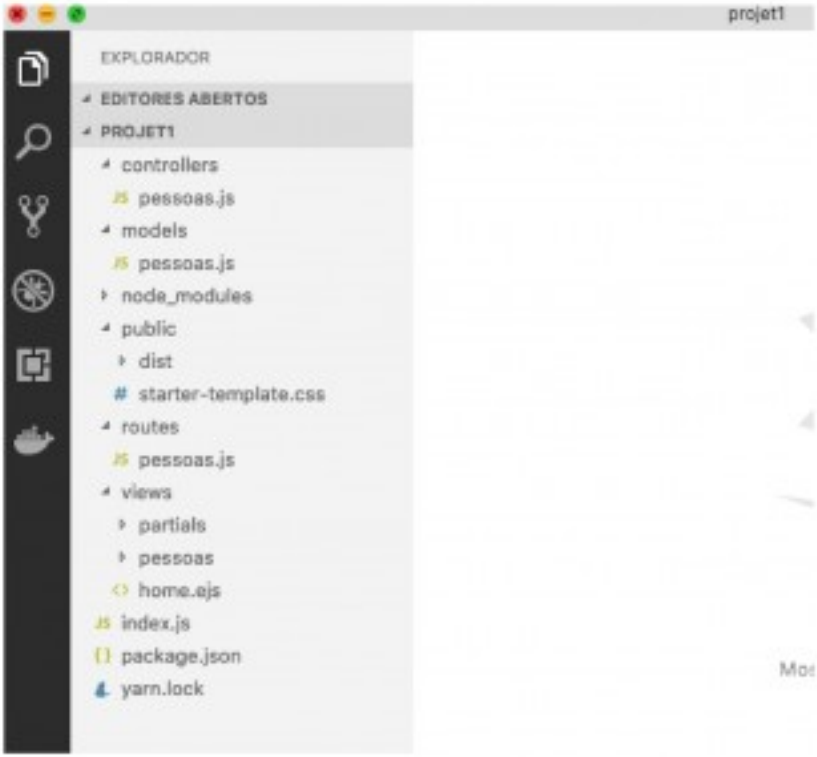
Agora abra o VS Code com o comando code .

```
✨  Done in 4.33s.
MacBook-Pro-de-Wanderson:projet1 wandersonjonersilvacruz$ code .
```

CAPITULO 4

4 - CRIANDO ARQUIVOS

Agora abra o VS Code com o comando code .



CAPITULO 4

4.1 - PACKAGE.JSON

Abra o pacote package.json e insira "start": "node index.js".

{} package.json

```
{
  "name": "teste5",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start":"node index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.3",
    "ejs": "^2.6.1",
    "express": "^4.16.4",
    "mysql": "^2.16.0"
  }
}
```

CAPITULO 4

4.2 - INDEX.JS

Index.js:

```
// criando o servidor
const express = require('express')
const path = require('path')
const app = express()
const port = process.env.PORT || 3000
// importando midlewares
const bodyParser = require('body-parser')
const pessoas = require('./routes/pessoas')
// conexao mysql
//importanto o mysql
const mysql = require('mysql')
// criando a conexao com o Mysql
const connection = mysql.createConnection({
    host: '127.0.0.1', user: 'root',
    password: 'admin', database: 'cadastro'
})
// injecao de dependencia
const dependencies = { connection }
// declarando midlewares
// declarando body-parser para conveter os dados do back-end para o front-end
app.use(bodyParser.urlencoded( { extended : false } ))
// declarando o caminho estatico que tem o template-html e o CSS
app.use(express.static('public'))
//midleware views
app.set('views', path.join(__dirname, 'views'))
//midleware view engine
app.set('view engine', 'ejs')
// declarando o caminho para renderizar o pagina
app.get('/', (req, res)=> res.render('home'))
// declarando o caminho para pessoas e injecao de depedencias
app.use('/pessoas', pessoas(dependencies))
//  conexao integrada entre o servido e a conexao com o BD
connection.connect(()=>{
    // o app.listen tem a funcao de escutar a conexao com o BD e conecta na porta
3000
    app.listen(port, () => console.log('Listering on port:'+port))
})
```

CAPITULO 4

4.3 - VIEWS/HOME.EJS

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <link rel="icon" href="../../../../favicon.ico">
  <title>Script</title>
  <!-- Bootstrap core CSS -->
  <link href="../../../../dist/css/bootstrap.min.css" rel="stylesheet">
  <!-- Custom styles for this template -->
  <link href="starter-template.css" rel="stylesheet">
 </head>
 <body>
  <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
   <a class="navbar-brand" href="/cad_projet">Cadastro Projeto</a>
   <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarsExampleDefault" aria-controls="navbarsExampleDefault" aria-
expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
   </button>
   <div class="collapse navbar-collapse" id="navbarsExampleDefault">
    <ul class="navbar-nav mr-auto">
     <li class="nav-item active">
      <a class="nav-link" href="/">Home <span class="sr-only">(current)</span></a>
     </li>
     <li class="nav-item">
      <a class="nav-link" href="/pessoas">Pessoas</a>
     </li>
     <li class="nav-item">
      <a class="nav-link disabled" href="/projeto">Projeto</a>
     </li>
    </div>
  </nav>
  <main role="main" class="container">
   <div class="starter-template">
    <h1>DevPleno</h1>
    <p class="lead">FULLSTACK MASTER<br> JS CODE PURE</p>
   </div>
  </main><!-- /.container -->
  <!-- Bootstrap core JavaScript
  ================================================== -->
  <!-- Placed at the end of the document so the pages load faster -->
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
  <script>window.jQuery || document.write('<script src="../../../../assets/js/vendor/jquery-
slim.min.js"><\/script>')</script>
  <script src="../../../../assets/js/vendor/popper.min.js"></script>
  <script src="../../../../dist/js/bootstrap.min.js"></script>
 </body>
```

```
</html>
```


CAPITULO 4

4.4 - VIEWS/PESSOAS/INDEX.EJS

```
<% include ../partials/header.ejs %>
<h1>Pessoas</h1>
<p><a href="pessoas/create">Nova Pessoas</a></p>
<table class="table table-striped table-hover table-bordered">
  <tr>
    <th>Nome</th>
    <th>Nascimento</th>
    <th>Cargo</th>
    <th>Acoes</th>
  </tr>
  <% pessoas.map( pessoa => { %>
  <tr>
    <td><%= pessoa.nome %></td>
    <td><%= pessoa.nascimento.toLocaleDateString() %></td>
    <td><%= pessoa.cargo %></td>
    <td><div class="btn-group" role="group" aria-label="Basic example">
      <a type="button" class="btn btn-secondary" href="/pessoas/update/<%= pessoa.id
%>">Editar</a>
      <a type="button" class="btn btn-secondary" href="/pessoas/delete/<%= pessoa.id %>"
onclick="return confirm('Desejas realmente excluir')">Excluir</a>
</div></td>
  </tr>
  <% }) %>
</table>
<% include ../partials/footer.ejs %>
```

CAPITULO 4

4.5 - MODELS/PESSOAS.JS

```
// função ou método para selecionar todas as pessoas
const findAll = (connection)=>{
   // retorna uma promessa com duas possibilidades (resolver ou rejeitar)
   return new Promise((resolve, reject)=>{
      // query para selecionar pessoas no BD
      connection.query('select * from pessoas',(err, results)=>{
         //teste de validação com um resultado entre dois possíveis(erro e resultados)
         if(err){
            reject(err)
         }else{
            resolve(results)
         }
      })
   })
}

// importando modulos
module.exports = {
   findAll,

}
```

Uma Promise representa um proxy para um valor que não é necessariamente conhecido quando a promessa é criado. Isso permite a associação de métodos de tratamento para eventos da ação assíncrona num caso eventual de sucesso ou de falha. Isto permite que métodos assíncronos retornem valores como métodos síncronos: ao invés do valor final, o método assíncrono retorna uma promessa ao valor em algum momento no futuro.

CAPITULO 4

4.6 - CONTROLLERS/PESSOAS.JS

```
// importando o pessoas.js da pasta models
const pessoas = require('../models/pessoas')
// abrindo conexão com mysql e renderizando o dados na página index.ejs
const index = async(connection, req, res)=>{
    const results = await pessoas.findAll(connection)
    res.render('pessoas/index', { pessoas : results })
}
//  importando modulos
module.exports = {
    index,

}
```

As funcionalidades **async / await** não conseguiram chegar para o ES6, mas isso não significa que elas não irão chegar ao JavaScript. Enquanto escrevo esse post, ela é uma proposta na fase 3 e está sendo trabalhada ativamente. As funcionalidades já estão no Edge e devem chegar a outros browsers assim que chegar na fase 4 – pavimentando seu caminho para inclusão na próxima edição da linguagem (veja também: Processo TC39).
Utilizando Promises
Vamos supor que tenhamos o código abaixo. Aqui eu estou encapsulando uma chamada HTTP em uma Promise. A promise executa o body caso haja sucesso e é rejeitada com um err caso contrário. Ela puxa o HTML de um artigo aleatório desse blog toda vez que é executada.

CAPITULO 4

4.7 - ROUTES/PESSOAS.JS

```
// importando o express
const express = require('express')
// importando o codigo que esta na pasta controller com o arquivo pessoas.js
const pessoasController = require('../controllers/pessoas')
// criando conexao entre a rota e a conexao mysql
const pessoasRouter = ({ connection })=>{
   const router = express()
      // rotas para o CRUD
      router.get('/', pessoasController.index.bind(null, connection))
      // rota para o select
      router.get('/delete/:id', pessoasController.deleteOne.bind(null, connection))
      // rota para o delete
      router.get('/create', pessoasController.createForm)
      // rota para o create le os dados para o Form
      router.post('/create', pessoasController.createProcess.bind(null, connection))
      // rota para o create pegar os dados do Form e levar para o processo de
insercao de dados
      router.get('/update/:id', pessoasController.updateForm.bind(null, connection))
      // rota que faz o select pelo id e trazer o dados para o Form para ser atualizados
      router.post('/update/:id', pessoasController.updateProcess.bind(null,
connection))
      // rota para o update pegar os dados do Form e levar para o processo de
atualizacao
   return router
}
// importando o modulo pessoas router
module.exports = pessoasRouter
```

Agora rode o comando npm start para subir nossa aplicação.