

Desenvolvedor Web Back-end: Node.js

Sumário

SUMÁRIO

Sumário	2
Apresentação do curso	5
Introdução	6
CAPÍTULO 1 – Ambiente e fundamentos do Node.js.....	8
Atividade 1 – Download e instalação do editor de código fonte	8
Atividade 2 – Escrever o primeiro script e executá-lo no Node.js	9
Atividade 3 – Incrementar o código do primeiro script	11
Exercícios extras	12
CAPÍTULO 2 – Módulos e NPM	13
Atividade 1 – Tornando o script síncrono.....	13
Atividade 2 – Conhecer o módulo “os” e variáveis de ambiente	14
Atividade 3 – Conhecer o módulo “path”	15
Atividade 4 – Conhecer o módulo “url”	16
Atividade 5 – NPM: Iniciar projetos, instalar e remover módulos.....	17
Atividade 6 – Implementar e utilizar módulos internos próprios.	19
Atividade 7 – Configurar o ambiente para reiniciar os scripts automaticamente a cada modificação	21
Exercícios extras	22
CAPÍTULO 3 – Serviço HTTP e renderização de HTML.....	23
Atividade 1 – Conhecer o módulo “fs”	23
Atividade 2 – Implementação de um servidor HTTP simples.....	25
Atividade 3 – Passagem de parâmetros na url da requisição	27
Atividade 4 – Renderizar arquivos HTML utilizando o módulo “fs”	28
Exercícios extras	30

CAPÍTULO 4 – Express – rotas, requisições e HTML render	31
Atividade 1 – Express e app “Olá Mundo!”	31
Atividade 2 – Parâmetros obrigatórios e opcionais	32
Atividade 3 – Receber parâmetros por meio de query parameters	33
Atividade 4 – Renderizar arquivos HTML utilizando o Express.....	34
Atividade 5 – Formato JSON e persistência de dados em arquivos	37
Exercícios extras	40
CAPÍTULO 5 – Acesso e manipulação de dados com Express e MySQL	41
Atividade 1 – Criação de banco de dados e tabela.....	41
Atividade 2 – Acesso e transação com o banco de dados.....	43
Atividade 3 – API-Rest com CRUD completo	46
Atividade 4 – Testes de funcionalidades da API-Rest	50
Exercícios extras	52
CAPÍTULO 6 – Manipulação de dados utilizando ORM	53
Atividade 1 – Introdução e instalação do ORM - Sequelize	53
Atividade 2 – Desenvolvimento de API – REST.....	57
Atividade 3 – Autenticação e criptografia	65
Exercícios extras	70
CAPÍTULO 7 – View Engine EJS e integração front-end com API.....	71
Atividade 1 – Introdução ao EJS e estrutura condicional IF	71
Atividade 2 – Implementando a estrutura de repetição no EJS	75
Atividade 3 – Utilização de Partial no EJS	80
Atividade 4 – Integração entre o front-end e a API utilizando Axios	87
Exercícios extras	90
CAPÍTULO 8 – Autenticação, Área Exclusiva, Cadastro e Transações	91
Atividade 1 – Autenticação utilizando bearer Token JWT	91

Atividade 2 – Aperfeiçoamento da API para proteger rotas e atender as demandas de cadastro e transações	100
Atividade 3 – Aperfeiçoamento do módulo requests.js	105
Atividade 4 – Cadastrar, editar e exibir registros de usuários	108
Atividade 5 – Realizar transações de registro interesse nos Pets	117
Atividade 6 – Cadastrar, exibir pets e lista de interessados.....	121
Exercícios extras	127
APENDICE	128
Instruções para o download e instalação do Node.js	128
Instruções para o download e instalação do MySQL	130

Apresentação do curso

O Node.js juntamente com a linguagem JavaScript formam uma das stacks mais poderosas do mercado para desenvolvimento web back-end. O Node.js é um ambiente de execução (runtime) de JavaScript, criado por Ryan Dahl em 2009 e tem como principais funções processar e executar códigos JavaScript no lado do servidor, o que eleva a linguagem para criar regras de negócio, acesso a dados e gerenciar recursos do sistema. Sendo a linguagem mais popular do mundo, o JavaScript conta com uma enorme comunidade que coopera no desenvolvimento e compartilhamento de poderosas soluções para problemas e necessidades que se fazem muito presentes na vida dos desenvolvedores. Desta forma, existe uma vasta disponibilidade de bibliotecas e frameworks que podem facilitar o desenvolvimento de aplicações complexas. Essas bibliotecas e frameworks podem facilmente serem integrados a um projeto Node.js utilizando o NPM (gerenciador de pacotes do Node.js), que na maioria das vezes, necessita de apenas uma linha de comando. Estes recursos ajudam a resolver facilmente problemas difíceis, dentre muitos exemplos pode-se citar:

- ✓ *Acesso a recursos do sistema (ex.: sistema de arquivos e dispositivos)*
- ✓ *Configuração de CORS;*
- ✓ *Criação de serviço HTTP;*
- ✓ *Autenticação de usuário e controle de acesso;*
- ✓ *Gerenciamento de Banco de Dados.*

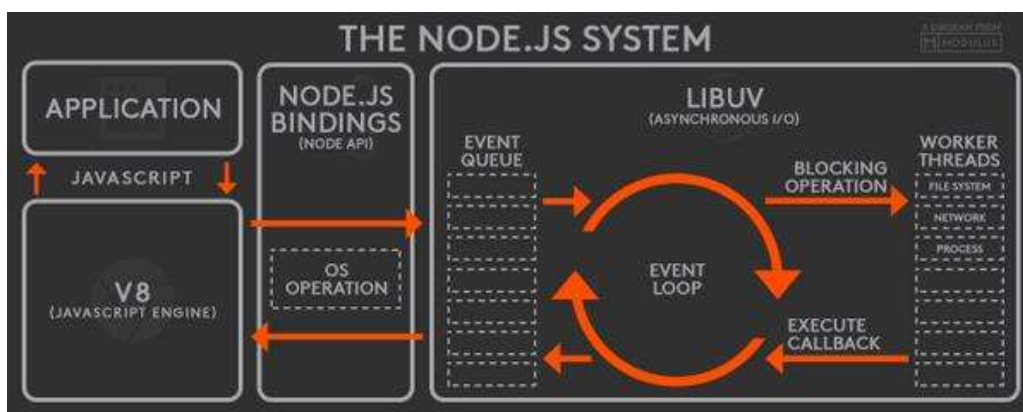
O objetivo deste curso é abordar fundamentos do Node.js, o framework Express, o Sequelize biblioteca ORM, o HTML engine EJS e outras bibliotecas que podem ser bastante úteis do dia a dia de um desenvolvedor. Para isso, a proposta é fazer uma sensibilização que apresenta uma necessidade de um cliente e colocar os alunos no lugar de um desenvolver. Durante o curso os alunos projetam e desenvolvem uma aplicação que visa a solução para essa necessidade.

Introdução

O Node.js já é uma tecnologia adotada por grandes empresas no mercado e alguns dos principais motivos são: sua alta capacidade de escala, sua arquitetura flexível e de baixo custo.

O Node.js foi projetado para criar aplicativos de rede escaláveis, tempo de execução assíncrono (não bloqueante) single-thread, orientado a eventos. O que o torna diferente de outras tecnologias como PHP e Java que têm sua execução orientadas a múltiplas threads e são bloqueantes. Na prática isso quer dizer que aplicações Node.js na maioria das vezes, consomem menos recursos computacionais do servidor como memória RAM e é possível uma melhor gestão de uso de CPU.

A arquitetura do Node.js é constituída principalmente pelas bibliotecas **V8**(interpretador de JavaScript) a biblioteca **LIBUV**(organiza os eventos e a execução de requisições).



Fonte: <https://stackoverflow.com/questions/42957176/nodejs-compilation-flow>

V8 – Biblioteca dos navegadores do Google, fornece ao Node.js um mecanismo JavaScript, que o Node.js controla por meio da API **V8 C++**.

LIBUV - biblioteca C usada para abstrair operações de E/S sem bloqueio. Também inclui um pool de threads para descarregar o trabalho para algumas coisas que não podem ser feitas de forma assíncrona no nível do sistema operacional. A **LIBUV** é composta pelas seguintes partes:

EVENT QUEUE - Escuta os eventos e cria uma ordem para as operações serem colocadas no EVENT LOOP para a execução.

EVENT LOOP - Fluxo de controle definidos por eventos que executa as operações no core central. Quando existem, lança as call-backs ao WORKER THREADS para serem processadas de forma assíncrona.

WORKER THREADS – Mecanismo que permite a execução de operações bloqueantes (principalmente operações de I/O) sem que ocorra bloqueio do thread principal. Ao concluir o processamento da chamada ela é devolvida ao EVENT LOOP para seguir o seu fluxo de processamento.

Como referência para mais informações, os seguintes links da documentação oficial do Node.js podem ser consultados:

<https://nodejs.org/en/docs/meta/topics/dependencies/>

<https://nodejs.dev/en/learn/the-nodejs-event-loop/>

CAPÍTULO 1 – Ambiente e fundamentos do Node.js

Neste capítulo você vai ver:

- Como configurar o ambiente de desenvolvimento fazendo a instalação dos softwares Visual Studio Code e o Node.js.
- Irá escrever seus primeiros scripts e ver como executá-lo no node.js

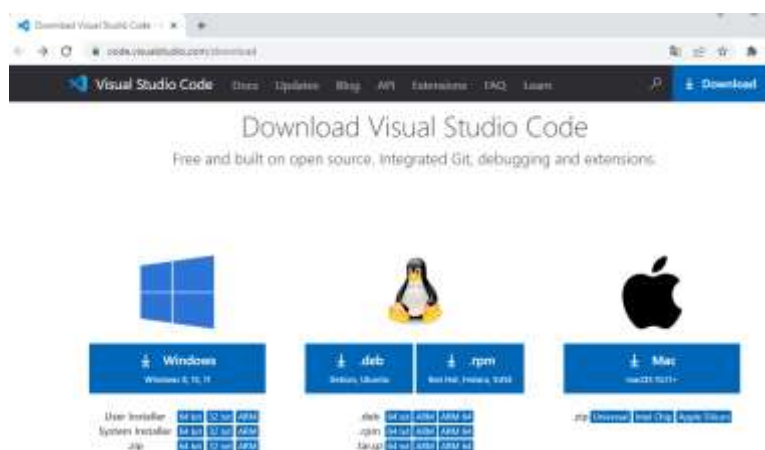
Atividade 1 – Download e instalação do editor de código fonte

Essa atividade tem como objetivo realizar o download e instalação do VS Code.

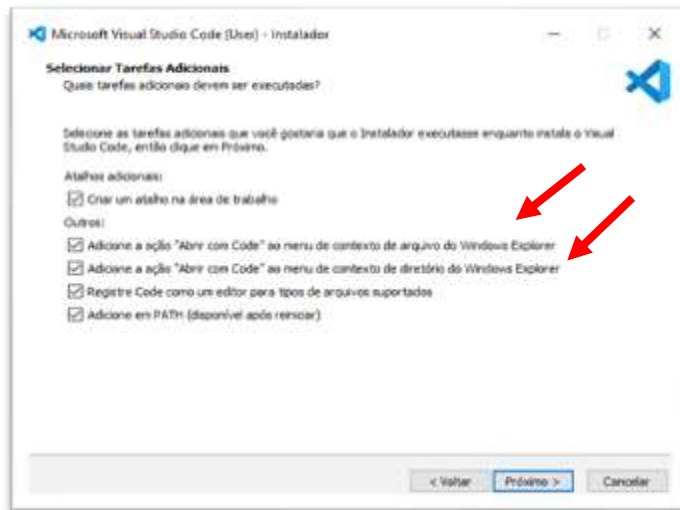
Obs.: Qualquer outro editor de código fonte JavaScript pode ser utilizado no desenvolvimento de aplicações Node.js, porém, o VS Code é o escolhido para o desenvolvimento deste curso.

Realizar o download e instalar o Visual Studio Code

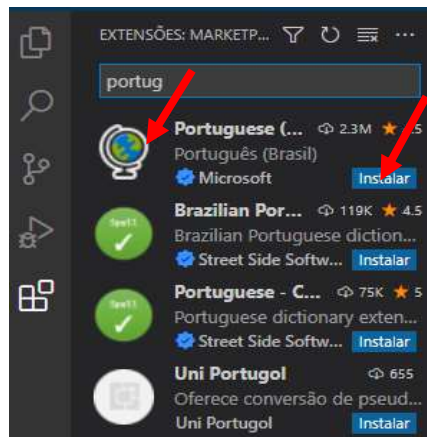
1. Acesse a url <https://code.visualstudio.com/download> e para realizar o download da versão mais atual, clique na opção compatível com o seu sistema operacional, por exemplo: Windows;



2. Após a conclusão do download, execute o instalador e siga os passos do assistente aceitando o contrato de licença, e clicando no botão que indique o passo seguinte. Para um manejo mais fácil dos projetos é indicado habilitar as duas opções indicadas na ilustração abaixo que por padrão aparecem desativadas.



3. A instalação do VS Code é feita nativamente em inglês, mas se caso desejar pode-se instalar uma extensão par ativar o idioma português brasil, conforme indicado abaixo:



Atividade 2 – Escrever o primeiro script e executá-lo no Node.js

Essa atividade tem como objetivo:

- Escrever o primeiro script e executá-lo no Node.js;
- Importar e utilizar um modulo do Node.js;
- Observar na prática o comportamento da programação assíncrona.

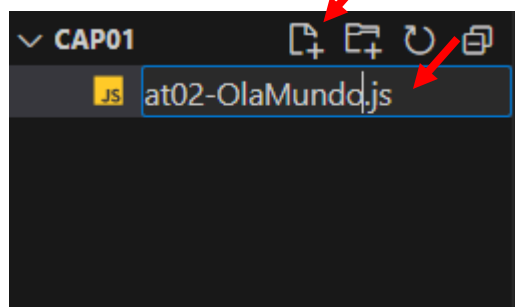
Escrever o primeiro script e executar no Node.js

1. Crie na área de trabalho uma pasta (ou diretório) chamada "NodeJs_Exercicios" e crie dentro dela uma subpasta chamada "Cap01";
2. Abra a pasta e em um lugar em branco clique com o botão direito e escolha a opção "**abrir com Code**" para carregar a pasta no VS Code como diretório de um projeto.

Obs: Outras formas também podem ser utilizadas, tais como abrir o VS Code e arrastar a pasta, além de utilizar o menu do VS Code nas opções “File/Open Folder...” e escolher a pasta.



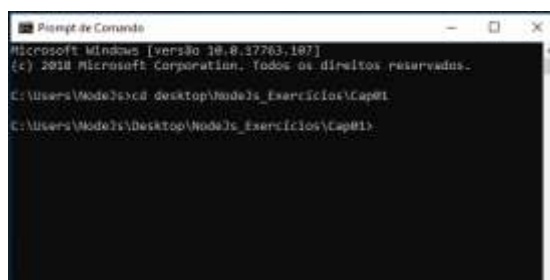
3. Crie um arquivo com o nome “**at02-OlaMundo.js**” para escrever o primeiro script;



4. Insira o seguinte código JavaScript no arquivo:

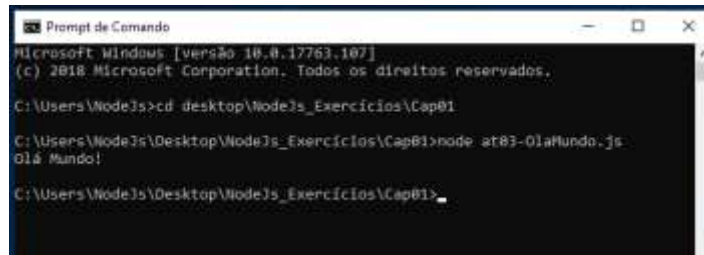
```
console.log('Olá Mundo!')
```

5. Existem algumas formas diferentes de executar um script no node.js, uma delas é utilizar o CMD. Digite CMD na pesquisa do Windows e abra o prompt de comandos;
6. Acesse a pasta do projeto utilizando o comando “**cd desktop\NodeJs_Exercicios\Cap01**”, conforme exemplo abaixo:



Obs.: Caso tenha utilizado outra pasta, adapte o comando “**cd <caminho_da_pasta>**”.

7. Para executar um script no node.js basta digitar o comando “**node <nome_do_arquivo>**”. No caso do nosso exemplo o comando apropriado é “**node at03-OlaMundo.js**” e o script será executado, conforme pode ser visto na imagem abaixo:



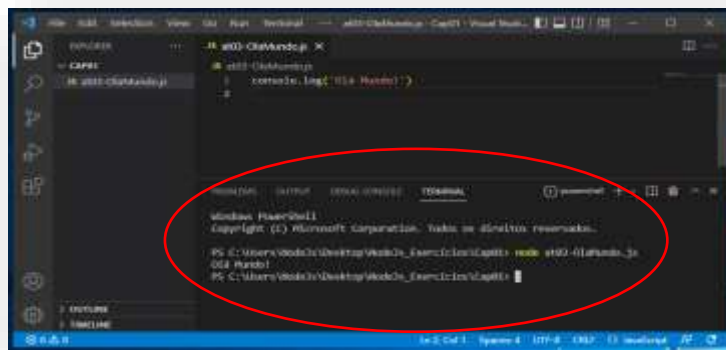
```
Prompt de Comando
Microsoft Windows [versão 10.0.17763.107]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\NodeJs>cd desktop\NodeJs_Exercicios\Cap01

C:\Users\NodeJs\Desktop\NodeJs_Exercicios\Cap01>node at03-OlaMundo.js
Olá Mundo!

C:\Users\NodeJs\Desktop\NodeJs_Exercicios\Cap01>
```

8. A execução do script também pode acontecer à partir do próprio VS Code. Voltando ao VS Code, clique no meu Terminal/New Terminal.
9. Note que o terminal já está apontando para o diretório raiz do projeto, de forma que só dar o comando “node at03-OlaMundo”.



Atividade 3 – Incrementar o código do primeiro script

Essa atividade tem como objetivo:

- Elevar a complexidade do primeiro script;
- Ler entradas a partir do console;
- Importar um modulo interno do Node.js;
- Observar na prática o comportamento da programação assíncrona.

Comandos:

- Declaração de constantes - const
- Importação de um modulo – require('nome_modulo')
- Uso do modulo readline

Incrementando o código

1. Crie novo arquivo no diretório do projeto com o nome de “at03-OláMundo2.js”;
2. Insira o seguinte código no arquivo. Cada linha contém um comentário explicativo.

```
//require tem como função realizar a importação de módulos.
const readline = require('readline')
//readline é um modulo capaz de receber entradas do console de forma
assíncrona
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
})
//console.log é utilizado para exibir mensagens no console
console.log('Olá Mundo!')
//A função question mostra uma mensagem e captura a Sting digitada.
rl.question('Qual é seu nome?\n', nome=>{
  console.log(`Olá ${nome}!`)
  rl.close()
})
console.log('Olá node!')
```

3. Pressione Ctrl+C para finalizar a execução.
4. Note a ordem de execução do código observando as saídas do console:



5. Perceba que a saída “Olá node!” veio antes da saída “Olá João”, isto porque como tratava-se de um código assíncrono e dependia de uma entrada do usuário.

Exercícios extras

Estas atividades extras têm como objetivo revisar e aprimorar fundamentos da linguagem JavaScript que serão utilizados nas atividades deste curso.

1. **Pesquise** quais são as **estruturas de repetição** que podem ser utilizadas no JavaScript;
2. **Pesquise e implemente** pelo menos um exemplo das estruturas: **for** - **while** - **do while**
3. Pesquise a diferença entre **break** e **continue**;

CAPÍTULO 2 – Módulos e NPM

Neste capítulo você vai ver:

- Os recursos dos principais módulos nativos (core modules) do Node.js
- O gerenciador de pacotes NPM
- Como criar, exportar e importar módulos (criar códigos reutilizáveis)
- Construir um serviço HTTP apenas utilizando core modules (sem bibliotecas)

Atividade 1 – Tornando o script síncrono

Essa atividade tem como objetivo:

- Ler entradas a partir do console de forma assíncrona;
- Diminuir a complexidade utilizando módulos e bibliotecas;
- Instalar globalmente um modulo do Node.js por meio do NPM;

Comandos:

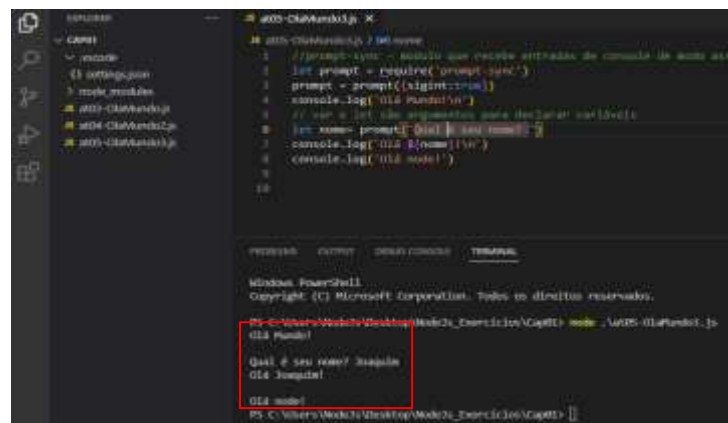
- var ou let - Declaração de variáveis
- npm install prompt-sync -g – Instalação de módulos
- require prompt-sync – Importação de módulos

Instalar o primeiro modulo global e executar o script de forma síncrona

1. Dentro do diretório “NODEJS_EXERCÍCIOS”, crie um diretório chamado “Cap02” e acesse o diretório via terminal com o comando “cd cap02”.
2. Vamos agora **instalar de forma global o modulo** externo **prompt-sync** para receber entradas de console de forma síncrona. Os módulos dos Node.js podem ser instalados por meio do seu gerenciador de pacotes o NPM. Para instalar um pacote utilizando o NPM utiliza-se o comando com a sintaxe “npm install <nome_do_pacote> -parametro”. Neste exemplo, utilize o comando “**npm install prompt-sync -g**” o parâmetro -g significa que a instalação é em modo global.
3. Como trata-se de uma instalação global, o modulo prompt-sync precisa ser “lincado” ao nosso script. O comando tem a sintaxe “npm link <nome_do_pacote>”, neste exemplo utilize o comando “**npm link prompt-sync**”;
4. Após a conclusão, no diretório “Cap02” - crie um arquivo com o nome “at01-OláMundo3.js”;
5. Insira o seguinte código no arquivo:

```
//prompt-sync - modulo que recebe entradas de console de modo síncrono
let prompt = require('prompt-sync')
prompt = prompt({sigint:true})
console.log('Olá Mundo!\r\n')
// var e let são argumentos para declarar variáveis
let nome= prompt('Qual é seu nome? ')
console.log(`Olá ${nome}!\r\n`)
console.log('Olá node!')
```

6. Rode o script e note que retornou um erro indicando que o modulo 'prompt-sync' não foi encontrado. Isto é devido o modulo ter sido instalado no escopo global, por isso é necessário ser linkado ao projeto. Utilize o comando “**npm link prompt-sync**”.
7. Rode o script novamente e ele será executado normalmente. Repare também que desta vez foi executado de maneira síncrona, ou veja, esperou a entrada do dado.



```
PS C:\Users\Wesley\Desktop\Modulo_Exercicios\Cap01> node .\005-01aMundo.js
Olá Mundo!
Qual é seu nome? Seu nome
Olá Seu nome!
Olá node!
PS C:\Users\Wesley\Desktop\Modulo_Exercicios\Cap01>
```

Obs.: O modulo prompt-sync não suporta o charset UTF-8, devido este exemplo ser apenas para fins didáticos iremos lidar com esta questão apenas em atividades

Atividade 2 – Conhecer o modulo “os” e variáveis de ambiente

Essa atividade tem como objetivo:

- Conhecer os recursos do core module “os” que tem acesso direto a informações e recursos do sistema operacional.
- Conhecer as variáveis globais de ambiente do Node.js

Comandos: `os.cpus()` | `os.freemem()` | `os.homedir()` | `os.type()` | `process.env`

Importar o core module “os” e exibir variáveis globais de ambiente do Node.js

Primeiramente vamos importar e utilizar alguns recursos do modulo “os”. Ele é um modulo interno do Node.js e por isso não precisa ser instalado pelo NPM, basta apenas importar com o require.

Depois vamos escrever no script uma que exibe as variáveis de ambiente do Node.js no console e conversar sobre as suas utilidades.

Siga os passos seguintes:

1. No diretório “Cap02” **crie um arquivo** com o nome “**at02-OsProcessEnv.js**”.
2. Implemente o seguinte código no arquivo:

```
//Importação do modulo "os"
const os = require('os')
console.log('Processador(es): ', os.cpus())
console.log('Qtde memória livre: ', os.freemem())
console.log('Diretório do usuário: ', os.homedir())
console.log('Familia de S.O.: ', os.type())
```

3. Execute o script com o comando “**node at02-osProcessEnv.js**” e observe as saídas.
4. Agora o primeiro teste, adicione a seguinte linha no fim do script:

```
//Lista todas as variáveis de ambiente
console.log(process.env)
```

5. Observe as saídas e veja que retornou muitas informações do Sistema Operacional que podem ser uteis no desenvolvimento e na “automação” de várias necessidades que dependa do sistema operacional;

Atividade 3 – Conhecer o modulo “path”

Essa atividade tem como objetivo:

- Conhecer os recursos do core module “path” que tem como principal função fornecer informações de caminho, nome e extensão de arquivos;

Comandos: path.extname() | path.basename() | path.dirname() | path.resolve()

Importar o modulo “path” e exibir informações de caminho e nome de arquivos

Vamos importar o modulo “path” e utilizar alguns de seus recursos. O “path” também é um modulo interno do Node.js, necessitando apenas a importação com o require. Siga os passos seguintes:

1. No diretório “Cap02” **crie um arquivo** com o nome “**at03-Path.js**”.
2. Implemente o seguinte código:

```
const path =require('path')
let arquivo='./at03-Path.js'

console.log('Extensão: ', path.extname(arquivo))
console.log('nome completo: ', path.basename(arquivo))
console.log('Unidade Base: ', path.dirname(arquivo))
console.log('Caminho Absoluto: ', path.resolve(arquivo))
```

3. Execute o script e observe as saídas do console.

Atividade 4 – Conhecer o modulo “url”

Essa atividade tem como objetivo:

- Apresentar alguns recursos do core module “url” que tem como principal função extrair informações das strings no formato de url. Será posteriormente utilizada em conjunto com o serviço HTTP para definir rotas;

Comandos:

```
partUrl.host | partUrl.pathname | partUrl.search |
partUrl.searchParams | partUrl.searchParams.get
```

Importar o modulo “url” e exibir informações de uma string no formato de url

Vamos importar o modulo “url” e utilizar alguns de seus recursos. O “url” também é um modulo interno do Node.js, necessitando apenas a importação com o require. Siga os passos seguintes:

1. No diretório “Cap02” **crie um arquivo** com o nome “**at04-Url.js**”.
2. Implemente o seguinte código:

```
const url = require('url')
let uri = 'https://www.google.com/search?q=node+js&rlz=1C1BNSD_pt-
BRBR946BR946'
let partUrl = new url.URL(uri)
console.log('Domínio: ', partUrl.host)
console.log('Caminho ou Rota: ', partUrl.pathname)
console.log('Query String: ', partUrl.search)
console.log('Parâmetros: ', partUrl.searchParams)
console.log('Valor do parâmetro q: ', partUrl.searchParams.get('q'))
console.log('Valor do parâmetro rlz: ', partUrl.searchParams.get('rlz'))
```

3. Execute o script e observe as saídas do console.

Atividade 5 – NPM: Iniciar projetos, instalar e remover módulos.

Essa atividade tem como objetivo:

- Apresentar as funcionalidades do NPM;
- Iniciar um projeto;
- Conhecer o arquivo package.json;
- Entender como o node organiza seus arquivos;
- Instalar, atualizar e remover módulos;
- Utilizar um modulo instalado pelo NPM;

Comandos: npm init ou npm init -y | npm install <pacote> | npm uninstall <pacote>

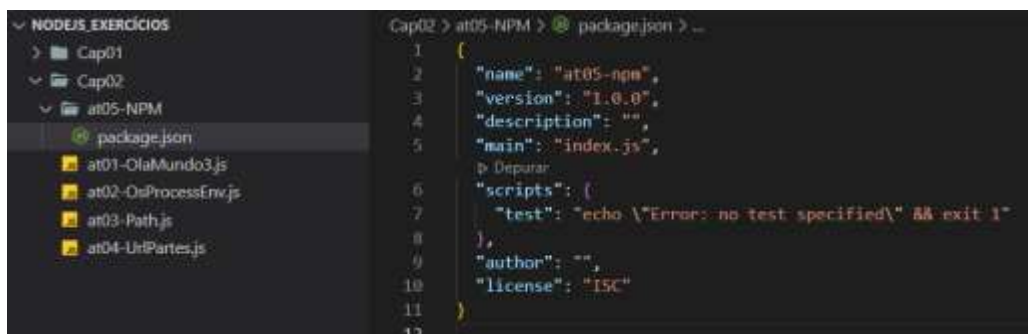
Instalar, importar e remover módulos externos com o NPM

Já foi observado que o NPM é o gerenciador de pacotes do Node.js, então agora vamos ver como adicionar e remover módulos externos. Por meio do arquivo package.json (contém todas as informações sobre o projeto), o npm também pode gerenciar as dependências do projeto.

Vamos experimentar algumas funcionalidades do NPM.

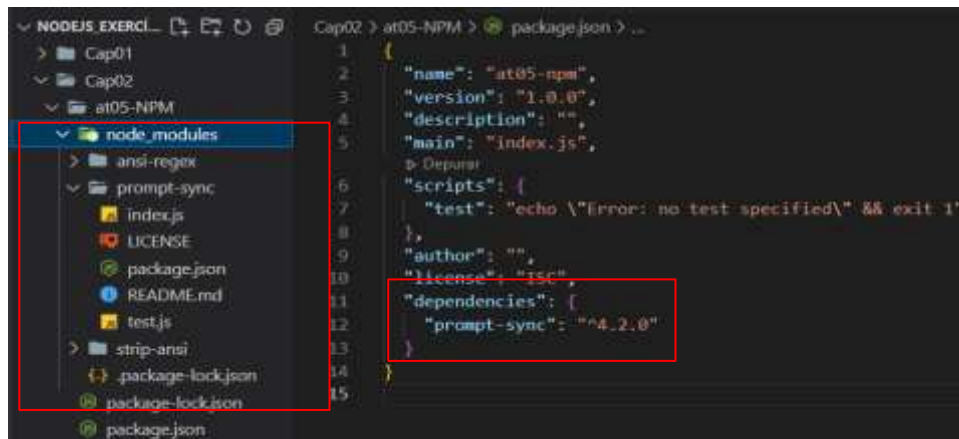
Siga os seguintes passos:

1. Dentro de Cap02 crie um subdiretório “at05-NPM” e acesse-o via terminal com o comando “**cd at05-NPM**”.
2. No diretório “at05-NPM” emita o comando “**npm init**” ou “**npm init -y**”(forma rápida) para iniciar um projeto Node.js;
3. Perceba que o arquivo **package.json** foi criado. Abra o arquivo e perceba seus atributos:



```
Cap02 > at05-NPM > package.json > ...
1  {
2    "name": "at05-npm",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC"
11  }
12
```

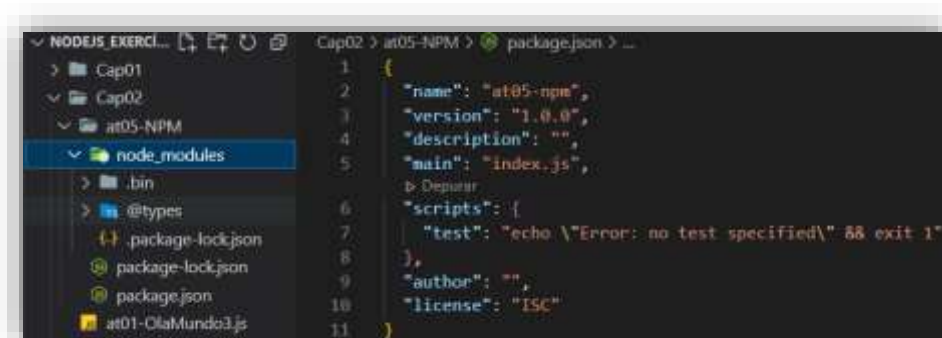
4. Agora, um modulo externo utilizando o NPM. A sintaxe do comando para a instalação de módulos é “**npm install <nome_do_modulo>**”. Para o nosso exemplo, vamos instalar o modulo prompt-sync, desta vez de forma local, para isso emita o comando “**npm install prompt-sync**”.
5. Note que um subdiretório chamado node_modules foi criado. Duas coisas são importantes serem observadas: 1) dentro de node_modules temos uma pasta contendo os módulos com o código do prompt.sync; e 2) um atributo “dependencies” foi adicionado no package.json e contém a dependência do modulo instalado bem como a indicação de sua versão.



6. Agora vamos realizar a instalação de diversos pacotes simultaneamente, utilize o comando **"npm i express mysql2 consign cors sequelize"**(o argumento install pode ser utilizado da forma abreviada **"i"**). Após a conclusão da instalação, perceba que a lista de dependências no package.json cresceu:



7. Com o objetivo de exercitar as funcionalidades do NPM, agora vamos remover os módulos. A sintaxe do comando para remover um pacote é **"npm uninstall <nome_do_pacote>"**. Como exemplo vamos utilizar o comando **"npm uninstall prompt-sync"**.
8. Os arquivos e a dependência do prompt-sync foram removidos. Para exercitar, remova os demais módulos instalados com o comando **"npm uninstall express mysql2 consign cors sequelize"**.
9. Verifique que todos as pastas do node_modules e as dependências do package.json foram removidos:



10. Agora vamos instalar e utilizar um modulo. Para o exemplo vamos utilizar o modulo lodash. Utilize o comando “**npm i lodash**”.
11. Após a instalação podemos importar o modulo normalmente com o comando require. Crie um arquivo com o nome at05-lodash.js e insira código a seguir:

```
const _ = require('lodash'); //Importação da biblioteca lodash
//Função que gera números aleatórios
const randomNum = () => Math.trunc(Math.random()*100);

console.log('----Usando o lodash---')
let numRandoms = _.times(10, randomNum)//Executa uma função X vezes
//Exibição do array - jeito convencional
console.log(typeof(numRandoms)) //Typeof exibe o tipo da variável
console.log(numRandoms)
console.log('Soma dos elementos: ', numRandoms.reduce((s, acc)=> acc+=s))
console.log(`Primeiro Elemento: ${numRandoms[0]}`)
console.log(`Último Elemento: ${numRandoms[numRandoms.length-1]}\r\n`)
// Exibição do array - utilizando o lodash
console.log('----Usando o lodash---')
console.log('Soma dos elementos: ', _.sumBy(numRandoms))
console.log(`Primeiro c/ lodash: ${_.first(numRandoms)}`)
console.log(`Último c/ lodash: ${_.last(numRandoms)}\r\n`)
```

12. Execute o código e observe as saídas no console.

Atividade 6 – Implementar e utilizar módulos internos próprios.

Essa atividade tem como objetivo:

- Implementar, Importar, exportar e reutilizar funções em módulos próprios;
- Transformar functions em arrow functions.

Comandos: exports | module.exports

Vamos aprender a criar nossos próprios módulos, exportá-los para serem reutilizados por outros códigos. Vamos também aproveitar e revisar a sintaxe de functions(funções em javascript) e arrow functions (funções de seta).

Siga os passos seguintes:

1. Dentro do diretório Cap02 crie um arquivo chamado “cd at06-ModulosProprios” e o acesse via terminal com o comando “**cd at06-ModulosProprios**”.
2. Crie um arquivo com o nome “at06-Calc.js” e insira o seguinte código:

```
function soma (a, b){
    return parseInt(a)+parseInt(b)
}
function sub (a, b){
    return parseInt(a)-parseInt(b)
}
function mult (a, b){
    return parseInt(a)*parseInt(b)
}
function div (a, b){
    if (parseInt(b)>0){
        return parseInt(a)/parseInt(b)
    }
    return "Não é possível dividir por zero."
}
console.log(soma(10,10))
console.log(sub(50,10))
console.log(mult(9,8))
console.log(div(100,4))
console.log(div(10,0))
```

3. Execute o código e observe a saída.
4. Note as funções foram escritas na sua forma mais verbosa. Vamos reescrever as utilizando um módulo externo e simplificá-las utilizando arrow function. Para isso, crie um diretório chamado "at06-Calculadora" e acesse via terminal com o comando "cd at06-Calculadora".
5. Crie um arquivo no chamado "calculadora.js" e insira o seguinte código:

```
soma=(a, b)=>{return parseInt(a)+parseInt(b)}
exports.soma = soma

sub=(a, b)=>{return parseInt(a)-parseInt(b)}
exports.sub = sub

mult=(a, b)=>{return parseInt(a)*parseInt(b)}
exports.mult = mult

div=(a, b)=>{return parseInt(b)>0? parseInt(a)/parseInt(b) : "Não é possível dividir por zero."}
exports.div = div
```

6. Perceba que na escrita das funções foi utilizada a sintaxe de arrow function e a escrita das funções ficaram menos verbosas. Note também que foi utilizado argumento "exports" que tem como funcionalidade tornar uma função ou variável disponível para serem reutilizadas em outros módulos.

7. Para utilizarmos as funções do modulo “calculadora.js” crie um novo arquivo no diretório “at06-CalcMod1” com o nome “at06-Calc.js” e insira o seguinte código:

```
const calculadora = require('./calculadora')
console.log(calculadora.soma(10,10))
console.log(calculadora.sub(50,10))
console.log(calculadora.mult(9,8))
console.log(calculadora.div(100,4))
console.log(calculadora.div(10,0))
```

8. Execute o código e analise as saídas. Perceba que a funcionalidade do programa é a mesma do código anterior.
9. Agora vamos ver uma forma mais simplificada de exportar as funções de um modulo e simplificar ainda mais a escrita das funções utilizando a sintaxe de arrow function. Para isso **duplique** o diretório “at06-CalcMod1” e **renomeie a cópia** para “at06-CalcMod2”.
10. Substitua o código do modulo “calculadora.js” pelo seguinte código:

```
const calculadora = {
  soma:(a, b)=> parseInt(a)+parseInt(b),
  sub: (a, b)=>parseInt(a)-parseInt(b),
  mult:(a, b)=>parseInt(a)*parseInt(b),
  div:(a, b)=> parseInt(b)>0? parseInt(a)/parseInt(b) : "Não é possível dividir por zero."
}
module.exports=calculadora
```

11. Note que as funções foram simplificadas ainda mais com a sintaxe de arrow function. Mas o mais importante é perceber que agora as funções foram encapsuladas em um objeto chamado calculadora e foram exportadas todas de uma vez com o argumento “module.exports”.
12. Execute o script do arquivo “at06-Calc.js”, observe a saída e perceba que a funcionalidade dos três códigos são as mesmas.

Atividade 7 – Configurar o ambiente para reiniciar os scripts automaticamente a cada modificação

Essa atividade tem como objetivo:

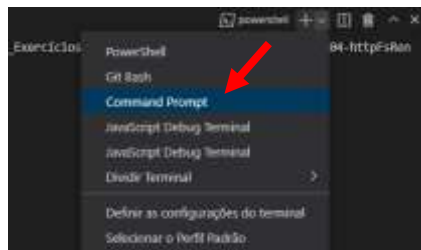
- Instalar o módulo nodemon;
- Configurar o ambiente para reiniciar a execução de scripts automaticamente;

Comando: npm install nodemon -g

Instalar o módulo nodemon

Até o momento, para as modificações nos scripts terem efeito é necessário que a execução seja encerrada e executada novamente. Em um ambiente de desenvolvimento, isto prejudica muito a produtividade. O módulo “**nodemon**” resolve este problema, por isso, vamos utilizar este módulo para executar os scripts para as atividades do restante do curso. Siga os passos a seguir:

1. Execute o comando “`npm install npm -g`” para instalar o módulo nodemon de forma global.
2. Tente executar o script do exercício anterior, mas desta vez utilize o comando “**nodemon** at04-httpFsRenderizarHTML.js”.
3. Um erro será retornado dizendo que a utilização do nodemon foi desativada para este sistema. Podemos resolver este problema de algumas maneiras, a mais simples é executar o terminal por meio do CMD ao invés de utilizar o powershell, conforme abaixo:



4. Acesse o diretório `.\Cap03\At04-httpRenderHTML` e execute novamente o comando “**nodemon** at06-Calc.js”. Perceba que agora o script foi executado por meio do nodemon.
5. Realize alterações no código e perceba que ao salvar alterações, a execução do script é automaticamente reiniciada. Isso trará um grande ganho de produtividade, realize testes no navegador.

Obs.: Existem outras formas para executar o nodemon a partir do VS Code, uma delas é instalar o plugin do VS develop Code PowerShell, conforme ilustrado abaixo. Nos próximos capítulos iremos executar o serviço http por meio de um script no package.json.



Exercícios extras

Estas atividades extras têm como objetivo revisar e aprimorar fundamentos da linguagem JavaScript que serão utilizados nas atividades deste curso.

1. **Pesquise e implemente** exemplos de utilização de **arrays** no JavaScript;
2. **Pesquise e implemente** exemplos das seguintes funções de **manipulação de arrays**:
`Array.push()`; `Array.pop()`; `Array.unshift()`; `Array.shift()`

CAPÍTULO 3 – Serviço HTTP e renderização de HTML

Neste capítulo você vai ver:

- Como criar um servidor HTTP utilizando o módulo nativo “http”;
- Como capturar parâmetros na requisição utilizando o módulo “url”;
- Como ler e gravar arquivos com o modulo nativo “fs”; e
- Como renderizar arquivos HTML combinando os módulos “http”, “url” e “fs” .

Atividade 1 – Conhecer o módulo “fs”

Essa atividade tem como objetivo:

- Ler e gravar arquivos utilizando o core module “fs” (file system) que tem como função acessar os recursos do sistema de arquivos;
- Recapitular as funções de manipulação de dados no formato JSON.

Comandos: fs.readFileSync() | fs.writeFileSync() | JSON.parse() | JSON.stringify()

Ler inserir e gravar em arquivos JSON.

Vamos aprender a ler arquivos gravar arquivos utilizando o módulo “fs”. Vamos também aproveitar e revisar a sintaxe e as funções de manipulação de dados no formato JSON.

1. Para iniciar as atividades do capítulo 03, crie um diretório na pasta “NODEJS_EXERCÍCIOS” com o nome “Cap03”
2. Para este exercício iremos utilizar um arquivo JSON para ser lido e gravado e dois módulos (script principal e modulo com funções exportadas). Por isso, crie um subdiretório em “Cap03” chamado “at01-fsLerGravarJSON”.
3. Dentro do subdiretório “at01-fsLerGravarJSON” crie um arquivo chamado “alunos.json” e insira o seguinte código utilizando a sintaxe JSON:

```
[
  {
    "nome": "Carlos",
    "nota1": 5.5,
    "nota2": 6,
    "nota3": 7,
    "nota4": 8
  }
]
```

4. Agora vamos criar o módulo que abrigará as funções que implementam as necessidades do programa. Para isso crie outro arquivo chamado “fsJSON.js” e insira o seguinte código:

```

const fs = require('fs')
module.exports = {
  lerJSON: (arquivo)=>{
    //Lê arquivos de texto de forma síncrona
    let json = fs.readFileSync(arquivo, 'utf8')
    return json
  },
  converterJSON_Obj: (json)=>{
    //Converte string JSON para um objeto javascript
    let dados = JSON.parse(json)
    return dados
  },
  converterObj_JSON: (dados)=>{
    //Converte objeto JavaScript para uma string JSON
    let json = JSON.stringify(dados)
    return json
  },
  salvarJSON: (json, arquivo)=>{
    //Grava arquivos de texto de forma síncrona
    fs.writeFileSync(arquivo, json)
  }
}

```

5. Agora crie um módulo chamado “**at01-LerGravarJSON.js**” e insira o código a seguir:

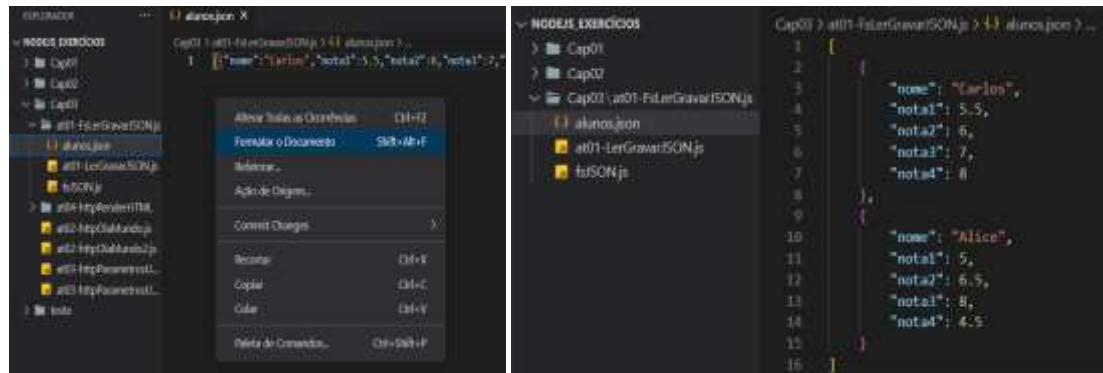
```

const rwJSON = require('./fsJSON') //Importação do módulo criado fsJSON
let arquivo = './alunos.json' //Dados a serem gravados
let obj = {
  nome: 'Alice',
  nota1:5,
  nota2:6.5,
  nota3:8,
  nota4:4.5
}
var json = rwJSON.lerJSON(arquivo) //Leitura e carga de um arquivo JSON
console.log(json)
//Conversão de string JSON em um objeto javascript
var dados = rwJSON.converterJSON_Obj(json)
console.log(dados)
//Inserção de novo registro
dados.push(obj)
console.log(obj)
//Conversão do obj atualizado para JSON
json = rwJSON.converterObj_JSON(dados)
console.log(json)
//Regravação do arquivo com dados atualizados
rwJSON.salvarJSON(json, arquivo)

```

6. Execute o script do módulo agora usando o comando “**nodemon at01-LerGravarJSON.js**” e verifique no arquivo “**alunos.json**” que um novo registro foi adicionado, clique com o botão

direito no arquivo e escolha a opção Formatar documento ou o atalho Shift+Alt+F, conforme observado abaixo:



Atividade 2 – Implementação de um servidor HTTP simples

Essa atividade tem como objetivo:

- Se familiarizar com o core module “http”;
- Compreender funcionamento de requisições e respostas HTTP.

Comandos: http.createServer() | res.setHeader() | res.end() | server.listen()

. “Olá mundo!” no navegador, utilizando um serviço “http”

Vamos aprender a ler arquivos gravar arquivos utilizando o módulo “fs”. Vamos também aproveitar e revisar a sintaxe e as funções de manipulação de dados no formato JSON.

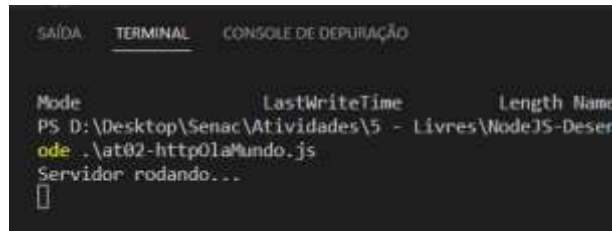
1. No subdiretório “Cap03”, crie um arquivo “at02-OlaMundo.js” e insira o seguinte código:

```
const http = require('http')

//Cria o serviço HTTP, e passa callback que processa as requisições.
const server = http.createServer((req, res)=>{

  //Configura o cabeçalho da resposta
  res.setHeader('Content-Type', 'text/html')
  //Responde a requisição
  res.end(`Olá mundo!!!`)
})
//Ativa o servidor para escutar as requisições
server.listen('3200', ()=>{console.log('Servidor rodando...')})
```

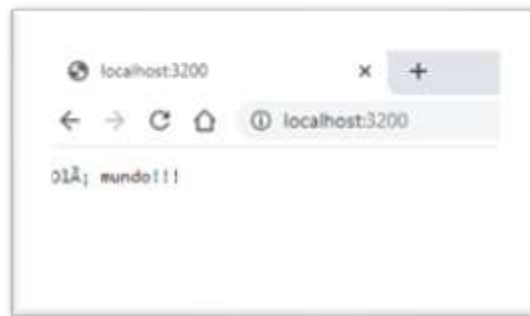
2. Execute o script e observe que ao executar o script não é finalizado, ao invés disso, ele permanece em execução e mantém o serviço ativo:



```
SAÍDA  TERMINAL  CONSOLE DE DEPUÇÃO

Mode                               LastWriteTime           Length Name
PS D:\Desktop\Senac\Atividades\5 - Livres\NodeJS-Desenvolvimento\at02-http01aMundo.js
ode .\at02-http01aMundo.js
Servidor rodando...
█
```

3. Para realizar uma requisição, abra o navegador e **acesse a url “localhost:3200”** e perceba a resposta do servidor:



4. Volte o código e altere a frase de “Olá mundo!!!” para “Olá Node.js!!!”. Atualize a página no navegador e perceba que a resposta continua sendo “Olá Mundo!!!”;
5. Isto acontece devido a necessidade de reiniciar o serviço para que as alterações sejam aplicadas. Para isso, vá para o **terminal do VS Code** e pressione o **atalho Ctrl+C** para encerrar a execução do serviço.
6. Execute o código novamente e perceba que agora a resposta foi atualizada.
7. Vamos agora aperfeiçoar o código respondendo agora um código HTML. Substitua a string no método na chamada da função `res.end()` conforme código HTML abaixo, note que para:

```
const http = require('http')

//Cria o serviço HTTP, com callback que processa e responde as requisições.
const server = http.createServer((req, res)=>{
  //Configura o cabeçalho da resposta
  res.setHeader('Contenty-Type', 'text/html')
  //Responde a requisição
  res.end(`
    <head> <meta charset="UTF-8"></head>
    <body>
      <h1>Olá Mundo!!!</h1>
    </body>
  `)
})

//Ativa o servidor para escutar as requisições
server.listen('3200', ()=>{console.log('Servidor rodando...')})
```

Obs.: Para a manter a formatação de quebra de linhas e tabulação na string de resposta deve-se utilizar ``crase`` e não `'aspas'`.

8. Pare o servidor e execute novamente o script. Acesse novamente o localhost:3200 no navegador e observe o resultado:



Atividade 3 – Passagem de parâmetros na url da requisição

Essa atividade tem como objetivo:

- Passar parâmetros para o servidor HTTP por meio da url da requisição;

Comandos: url.parse()

Passagem de parâmetros na url da requisição

Passar e ler parâmetros para o servidor utilizando a url da requisição utilizando o módulo “url”.

1. No subdiretório “Cap03”, crie um arquivo “at03-httpParametrosUrl.js” e insira o código:

```
const http = require('http')
const url = require('url')
const server = http.createServer((req, res)=>{
  //Converte parâmetros passados na url
  let parametro = url.parse(req.url, true)
  let nome = parametro.query.nome
  res.statusCode=200
  res.setHeader('Contenty-Type', 'text/html')
  if (nome){
    res.end(`<head> <meta charset="UTF-8"></head>
      <body>
        <h1>Olá ${nome}!!!</h1>
      </body>`)
  } else {
    res.end(`<head> <meta charset="UTF-8"></head>
      <body>
        <h1>Olá Node.js!!!</h1>
      <body>`)
  }
})
server.listen('3200', ()=>{console.log('Servidor rodando...')})
```

2. Realize testes executando o script e inicie o servidor. Abra o navegador e acesse com a seguinte url: “localhost:3200” e com “localhost:3200/?nome=João” e veja os resultados:



Atividade 4 – Renderizar arquivos HTML utilizando o módulo “fs”

Essa atividade tem como objetivo:

- Renderizar arquivos html utilizando o core module “fs”;

Comandos: `pathname.substring()` | `fs.existsSync()` | `fs.readFile()`

Renderizar arquivos HTML

Nesta atividade vamos fazer um serviço HTTP renderizar arquivos HTML. Para isso iremos utilizar um módulo JavaScript e três arquivos HTML como exemplo, por isso iremos criar um subdiretório na pasta do capítulo 3.

Siga os passos a seguir:

1. No diretório “Cap03”, crie um subdiretório com o nome “at04-HTTPFsRenderizarHTML”.
2. Crie dentro do subdiretório os arquivos HTML que serão renderizados, conforme abaixo:

a) index.html

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Index</title>
  </head>
  <body>
    <h1>Página Inicial</h1>
  </body>
</html>
```

b) usuario.html

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Usuário</title>
</head>
<body>
  <h1>A rota usuário foi acessada!</h1>
</body>
</html>

```

c) 404.html

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Página não encontrada 404</title>
</head>
<body>
  <h1>Página não encontrada ERRO:404</h1>
</body>
</html>

```

3. Agora faremos o script do serviço HTTP que receberá a requisição e renderizará as páginas.

Crie um arquivo com o nome “**at04-httpFsRenderizarHTML.js**” e insira o seguinte código:

```

const http = require('http')
const url = require('url')
const fs = require('fs')
const porta = 3200

const server = http.createServer((req, res)=>{
  const q= url.parse(req.url, true)
  //Obtem a substring após a "/"
  let pagina = q.pathname.substring(1)
  //Aponta url inicial para o index.html
  pagina = pagina==''? 'index.html':pagina
  //Insere a extensão HTML caso não for especificado
  pagina = !pagina.includes('html')? pagina+'.html': pagina
  console.log(pagina)

  if (fs.existsSync(pagina)){ //Verifica o arquivo existe
    fs.readFile(pagina, function(err, data){

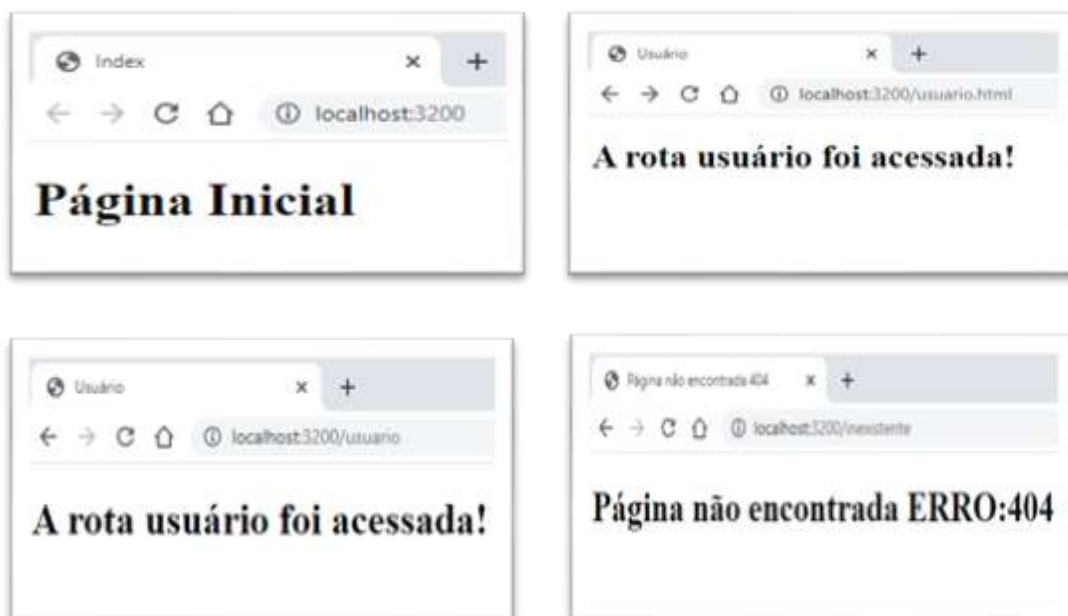
```

```

        res.writeHead(200, {'Content-Type': 'text/html'})
        res.write(data)
        return res.end()
    })
} else {
    fs.readFile('404.html', function(err, data){
        res.writeHead(404, {'Content-Type': 'text/html'})
        res.write(data)
        return res.end()
    })
}
})
server.listen(porta, ()=>{
    console.log('Servidor rodando em: http://localhost:'+porta)
})

```

4. Execute o servidor e realize testes no navegador conforme imagens a seguir:



Exercícios extras

Estas atividades têm como objetivo aprofundar-se em conceitos importantes para a programação back-end.

Pesquisar e ler sobre:

1. Serviço HTTP e quais são as principais partes de uma requisição HTTP.
2. Diferença entre as estruturas de dados JSON e XML.
3. Sintaxe da estrutura de dados JSON.

Após as atividades de leitura, pode-se realizar:

4. Roda de conversa¹ para refletir e compartilhar conhecimento sobre a leitura.

¹ Ou outra ferramenta pedagógica proposta pelo docente.

CAPÍTULO 4 – Express – rotas, requisições e HTML render

Neste capítulo você vai ver:

- Introdução ao express;
- Criação de um servidor HTTP com Express;
- Criação de rotas;
- Gerenciamento de requisições;
- Renderização de arquivos HTML

Atividade 1 – Express e app “Olá Mundo!”

Essa atividade tem como objetivo:

- Instalar o módulo do Express;
- Criar um servidor e desenvolver uma aplicação simples;

Comandos: app.get | res.send | app.listen

Primeira aplicação com Express

Vamos aprender a instalar e utilizar o modulo do Express criando um servidor HTTP simples.

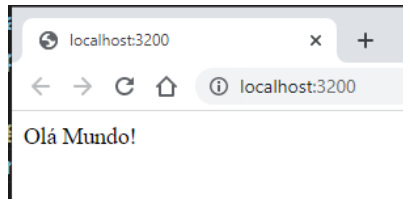
1. Para iniciar as atividades do capítulo 04, crie um diretório na pasta “NODEJS_EXERCÍCIOS” com o nome “Cap04” e acesse o novo diretório através do comando “cd Cap04”.
2. Com o diretório “Cap04” ativo emita o comando “npm i express”.
3. Agora crie um módulo com o nome “at01-ExpressHTTP.js” e insira o seguinte código:

```
const express = require('express')
var app = express() // Instância do módulo Express
var porta = '3200' // Variável com o número da porta usada pelo app

// Recebe uma requisição HTTP do tipo get na rota raiz "/"
app.get('/', function (req, res){
    res.send('Olá Mundo!') // Emite a resposta para a requisição
})

// Coloca o servidor em execução para escutar requisições HTTP
app.listen(porta, function(){
    console.log(`Servidor rodando em: http://localhost:${porta}`)
})
```

4. Para realizar o teste execute o script e acesse no navegador a url <http://localhost:3200/>. Deverá retornar o seguinte resultado:



5. Vamos agora refatorar o código criando novas rotas, passando tags HTML como resposta e utilizando arrow functions. Para isso atualize o escript com o seguinte código:

```
const express = require('express')
var app = express()
var porta = '3200'
app.get('/', (req, res)=>{res.send('<h1>Página Inicial</h1>')})
app.get('/cadastro', (req, res)=>{res.send('<h1>Tela de cadastro</h1>')})
app.get('/usuario', (req, res)=>{res.send('<h1>Tela de usuário</h1>')})
app.get('/consulta', (req, res)=>{res.send('<h1>Tela de consulta</h1>')})
app.listen(porta, function(){console.log(`Servidor rodando em:
http://localhost:${porta}`)})
```

6. Para realizar o teste execute o script e acesse no navegador as seguintes url:

<http://localhost:3200/> | <http://localhost:3200/cadastro> | <http://localhost:3200/usuario> | <http://localhost:3200/consulta>

Atividade 2 – Parâmetros obrigatórios e opcionais

Essa atividade tem como objetivo:

- Receber parâmetros obrigatórios e opcionais por meio da url da requisição;

Comandos: req.params

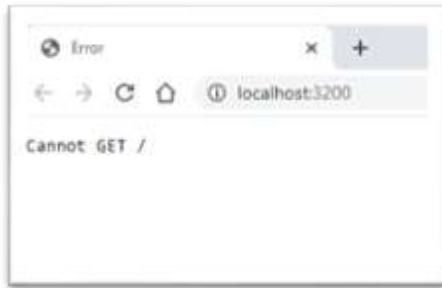
Parâmetros na url da requisição

Vamos ver como receber parâmetros em uma aplicação utilizando o Express.

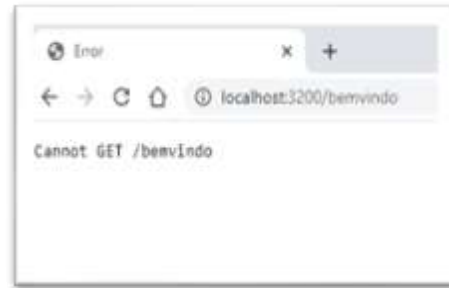
1. Para iniciar a atividade, crie um módulo no diretório Cap04 com o nome “at02-ParametroExpress.js” e insira o seguinte código:

```
const app = require('express')()
var porta = '3200'
//Parâmetros obrigatórios('/:nome')
app.get('/bemvindo/:nome', (req, res)=>{
  let nome = req.params.nome //Acesso ao parâmetro "nome"
  res.send('<h1>Olá ${nome}, seja bem-vindo!</h1>`)
})
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

2. execute o script e realize testes no navegador conforme o seguinte:



Neste caso a rota raiz “/” não existe



Neste caso o parâmetro é obrigatório



Neste caso o parâmetro “nome” recebeu o valor “João”

3. Vamos agora refatorar o código e tornar o parâmetro opcional, para isso atualize o código conforme o seguinte:

```
const app = require('express')()
var porta = '3200'
//Parâmetros opcionais acrescentar '?', exemplo '.../:nome?'
app.get('/bemvindo/:nome?', (req, res) => {
  let nome = req.params.nome? req.params.nome : ''
  res.send(`<h1>Olá ${nome}, seja bem-vindo!</h1>`)
})
app.listen(porta, ()=>{console.log(`Servidor rodando em:
http://localhost:${porta}`)})
```

4. Refaça os testes acessando as mesmas URLs e observe os resultados.

Atividade 3 – Receber parâmetros por meio de query parameters

Essa atividade tem como objetivo:

- Aprender como receber parâmetros na URL da requisição com query parameters

Comandos: req.query

Parâmetros na url da requisição por meio de query parameters

Vamos ver como receber parâmetros na URL por meio de query parameters em uma aplicação do Express.

1. Para iniciar a atividade, crie um módulo no diretório Cap04 com o nome “at03-QueryParams.js” e insira o seguinte código:

```
var app = require('express')()
var porta = '3200'
app.get('/',(req, res)=>{
  //Acessa o Query Parameter na URL da requisição, caso exista
  if (req.query['nome']){
    let nome =req.query['nome']
    res.send(`<h1>Olá ${nome}, seja bem vindo!</h1>`)
  }
  else{
    res.send(`<h1>Informe seu nome</h1>
    <form method="GET">
      <label for="nome">Nome:</label>
      <input type="text" name="nome" placeholder="Informe seu nome" />
      <input type="submit" value="Enviar" />
    </form>`)
  }
})
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

2. Realize o teste conforme acessando o navegador conforme imagens abaixo:



3. Observe no destaque da imagem que os query parameters são definidos na URL pelo caractere “?” seguido do “nome_do_parâmetro” sinal de “=” e o “valor_do_parâmetro”. No exemplo “.../?nome=João”.

Atividade 4 – Renderizar arquivos HTML utilizando o Express

Essa atividade tem como objetivo:

- Aprender como renderizar arquivos HTML como resposta da requisição;
- Utilizar o modulo “path” para configurar o caminho do diretório templates.

Comandos: path.join | res.sendFile | app.use

Renderizar templates HTMLs em aplicações Express

Vamos ver como configurar o caminho de um diretório de templates HTML e renderizá-los em aplicações do Express. Vamos aprender também como estabelecer uma rota de erro 404 para quando o usuário digitar uma rota que não existe.

1. Para iniciar a atividade, crie um subdiretório no diretório Cap04 chamado "at04-RenderizandoHTML". Dentro do subdiretório "at04-RenderizandoHTML" crie outro subdiretório chamado "templates".
2. Copie do material do curso ou crie-os na pasta "templates" os seguintes arquivos HTML com seus respectivos códigos:

a) index.html

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Home</title>
</head>

<body>
  <h1>Página Inicial</h1>
</body>

</html>
```

b) cadastrar.htm

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Cadastrar</title>
</head>

<body>
  <h1>A rota cadastrar foi acessada!</h1>
</body>

</html>
```

c) 404.html

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Erro 404</title>
</head>

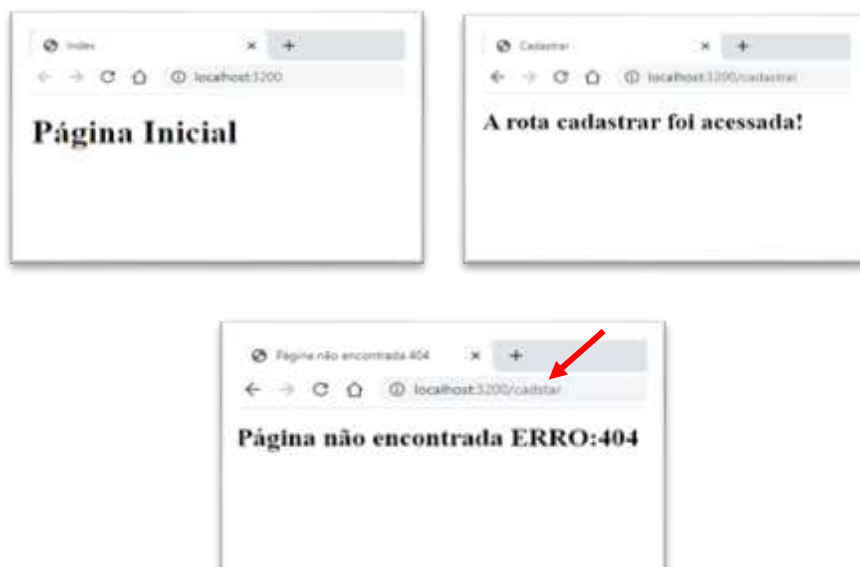
<body>
  <h1>Página não encontrada - ERRO:404</h1>
</body>

</html>
```

3. No diretório “at04-RenderizandoHTML” crie um arquivo “index.js” e insira o seguinte código:

```
const app = require('express')()
const path = require('path')
const porta = '3200'
//String com o caminho do diretório base + /templates
const baseDir = path.join(__dirname, 'templates')
app.get('/', (req, res)=>res.sendFile(`${baseDir}/index.html`))
app.get('/cadastrar', (req, res)=>res.sendFile(`${baseDir}/cadastrar.html`))
app.use((req, res)=>res.sendFile(`${baseDir}/404.html`))//Rota padrão -
Importante ficar por último
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

4. Agora execute o script index.js e teste as rotas:



Atividade 5 – Formato JSON e persistência de dados em arquivos

Essa atividade tem como objetivo:

- Familiarizar-se com respostas no formato dados JSON.
- Introdução a permanência de dados utilizando o NeDB que é um banco de dados baseado em arquivo e que trabalha com o formato JSON.

Comandos:

- `npm i nedb`
- `app.use(express.urlencoded({extended:true}))`
- `app.use(express.json())`
- `app.post()`

Respostas no formato JSON e persistência de dados em arquivos

Nesta atividade vamos ver como podemos responder as requisições com dados no formato JSON, também vamos gravar registros com rotas POST. Para persistir os dados, a título de exemplo utilizaremos o NeDB que é um banco de dados não relacional baseado em arquivos que utiliza o formato JSON.

Para iniciar duplicaremos o código exercício anterior e faremos a instalação do módulo NeDB:

1. A partir do diretório Cap04 emita o comando de instalação do NeDB – **“npm i nedb”**
2. Ainda dentro de Cap04 duplique a pasta “at04-RenderizandoHTML” e renomeie para “at05-NeDB” e depois acesse-o no console com o comando **“cd at05-NeDB”**.
3. Vamos alterar o `<body>` dos arquivos “index.html” e “cadastrar.html” que estão na pasta “templates” com os respectivos códigos:
 - a) index.html

```
<body>
  <nav>
    <ul>
      <li><a href="/cadastrar">Cadastrar</a></li>
      <li><a href="/registros">Ver Registros</a></li>
    </ul>
  </nav>
</body>
```

b) cadastrar.html

```
<body>
  <h1>Cadastrar Pets</h1>
  <form method="POST" action="/cadastrar">
    <div><label for="nome">Nome:</label>
    <input type="text" name="nome" autofocus placeholder="Nome do pet" />
    </div>
    <div><label for="especie">Especie:</label>
    <input type="text" name="especie" autofocus placeholder="Ex: cachorro" />
    </div>
    <div><label for="idade_aproximada">Idade Aproximada:</label>
    <input type="number" name="idade_aproximada" placeholder="Ex: 2.5" />
    </div>
    <div> <label for="porte">Porte:</label>
    <input type="text" name="porte" placeholder="Ex.: Pequeno" />
    </div>
    <div><label for="cor_predominante">Cor Predominante:</label>
    <input type="text" name="cor_predominante" placeholder="Ex.: cinza" />
    </div>
    <div><label for="cor_secundaria">Cor Secundaria:</label>
    <input type="text" name="cor_secundaria" placeholder="Ex.: preto" />
    </div>
    <div><input type="submit" value="Enviar" /></div>
  </form>
</body>
```

4. O próximo passo é criar um arquivo de conexão ao banco de dados. Para isso crie um modulo no diretório do exercício com o nome “db.js” e insira o seguinte código:

```
const NeDB = require('nedb')
module.exports = new NeDB({ //Cria uma instância da classe NeDB
  filename: 'pets.db', //Nome do arquivo do banco
  autoload:true //Biblioteca gerencia o arquivo de forma autônoma
})
```

5. Por último, vamos criar o script principal no arquivo “index.js”, que irá conter as rotas GET e POST da aplicação. Atualize o código do “index.js” da seguinte forma:

```
const express = require('express')
const app = express()
var db = require('./db')
const path = require('path')
```

```

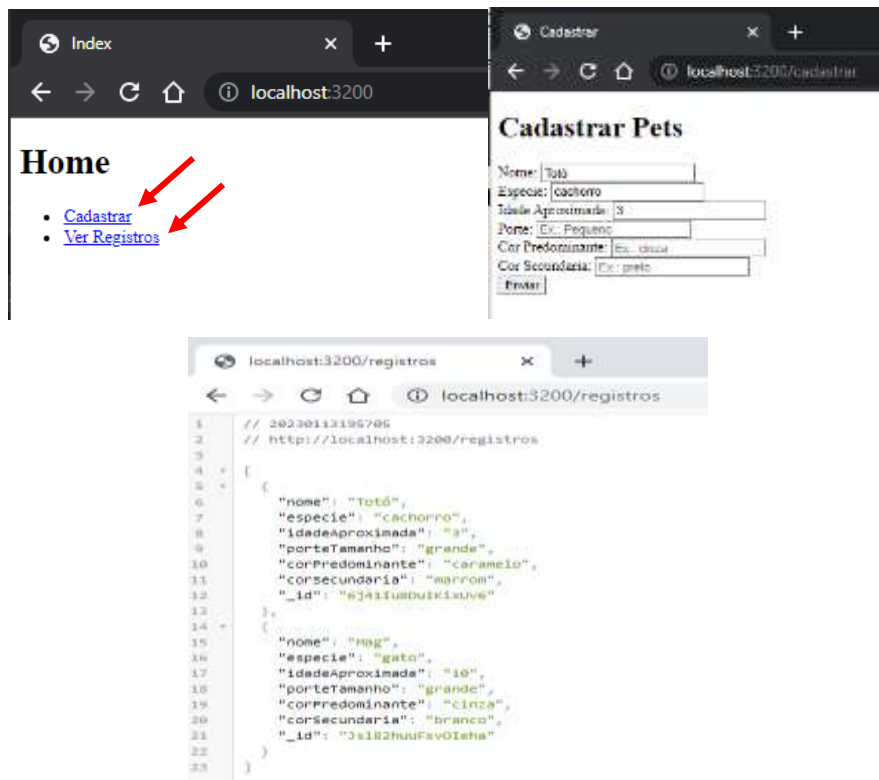
const baseDir=path.join(__dirname, 'templates')
var porta = '3200'
app.use(express.urlencoded({extended:true}))
app.use(express.json())
app.get('/', (req, res)=>res.sendFile(`${baseDir}/index.html`))
app.get('/cadastrar', (req, res)=>res.sendFile(`${baseDir}/cadastrar.html`))

//rota do tipo GET para obter os dados
app.get('/registros', (req, res)=>{
  let dados;
  db.find({}).exec((err, dados)=>{ //ler dados
    if (err) {
      res
        .status(400) //Resposta com status de erro 400
        .json(err) //Resposta no formato JSON
    } else {
      res
        .status(200) //Resposta com status de sucesso 200
        .json(dados) //Resposta no formato JSON
    }
  })
})

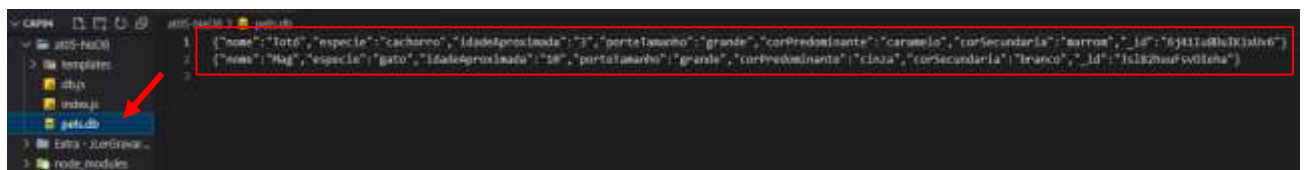
//rota do tipo POST - recebe dados no corpo da requisição
app.post('/cadastrar', (req, res)=>{
  //Salvar dados
  db.insert(req.body, (err, dados)=>{
    if (err){
      res
        .status(400)
        .json(err)
    } else{
      res
        .status(200)
        .json(dados)
    }
  })
})
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))

```

6. Execute o módulo index.js e realize os testes. Acesse a rota raiz “/” no navegador e clique no menu “Cadastrar” e realize alguns cadastros. Em seguida clique no menu “Ver Registros” e veja todos os registros listados no formato JSON.



7. Perceba que após a execução do script um arquivo chamado “pets.db” foi criado automaticamente. Após a realização dos cadastros ele ficou com o seguinte conteúdo:



Exercícios extras

Estas atividades extras têm como objetivo expandir o conhecimento e a prática do express:

1. Acesse a documentação do **Express**: <https://expressjs.com/pt-br/> e pesquisar para que serve, como instalar e como implementar uma aplicação simples.
2. Reescreva o exercício da **Atividade 1 do Capítulo 3 - Ler inserir e gravar em arquivos JSON** utilizando o Express, renderizando páginas HTML e recebendo requisições GET e POST.

CAPÍTULO 5 – Acesso e manipulação de dados com Express e MySQL

Neste capítulo você vai ver:

- Criação de banco de dados e tabelas utilizando o MySQL;
- Desenvolvimento de API-Rest com acesso ao banco de dados usando Express;
- Rotas HTTP dos tipos GET, POST, UPDATE e DELETE;
- Operações CRUD utilizando a linguagem SQL;
- Utilização do software Postman para testar a API;

Atividade 1 – Criação de banco de dados e tabela

Essa atividade tem como objetivo:

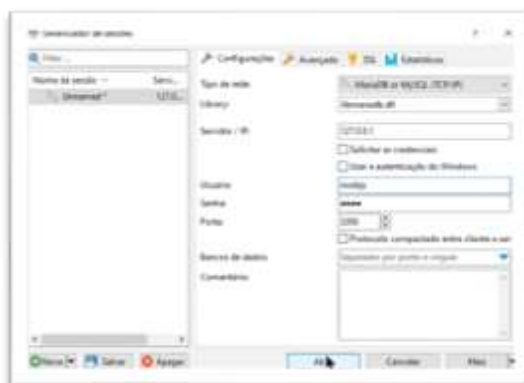
- Criar um banco de dados e uma tabela no MySQL;

Exemplo de criação de tabelas no MySQL

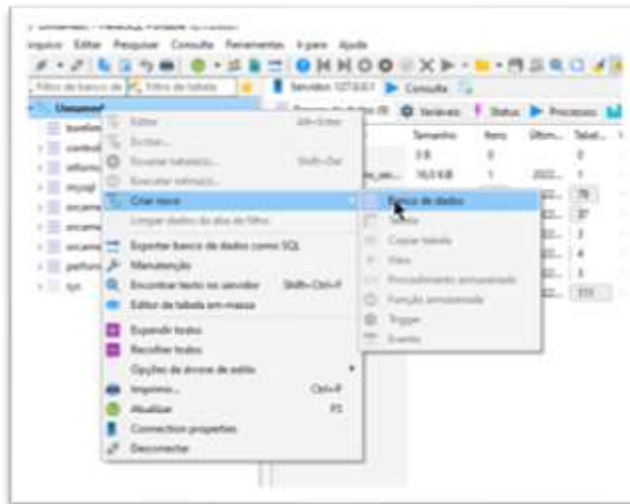
Para esta atividade utilizaremos o SGBD MySQL e um aplicativo cliente chamado HeidiSQL para realizar as configurações, detalhes de instalação destes softwares podem ser vistos no apêndice deste material. Vale ressaltar que outros softwares podem ser utilizados para o mesmo fim tais como Xampp e Mysql Workbench.

Para iniciar a atividade vamos começar do pressuposto de que o MySQL e o HeidiSQL estão devidamente instalados e configurados.

1. Abra o HeidiSQL e realize o login utilizando o usuário e senha cadastrado no PC local.



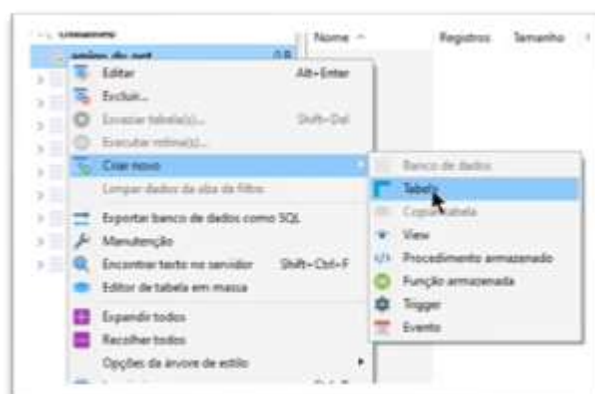
2. Na próxima tela clique com o botão direito na raiz da estrutura hierárquica do MySQL e acesse o menu “Criar novo” e depois “Banco de dados”. Conforme pode ser observado na imagem abaixo:



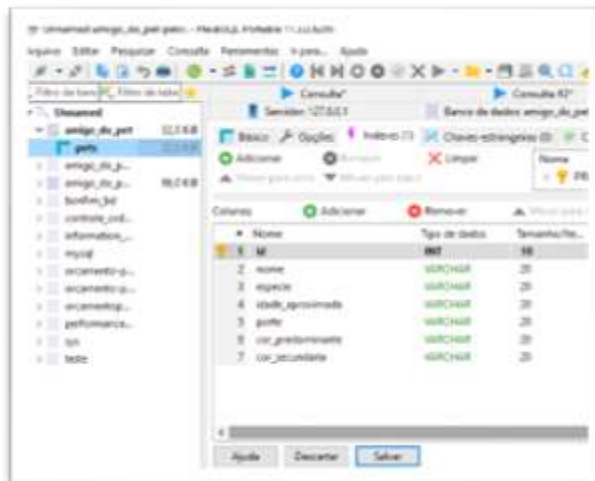
3. Crie o banco com o nome “amigo_do_pet” para utilizarmos como exemplo.



4. Agora clique com o botão direito no banco de dados criado acesse o menu “Criar novo” e depois “Tabela” e defina o nome da tabela como “pets”.



5. Adicione os seguintes atributos com as respectivas configurações e clique em salvar:



Atividade 2 – Acesso e transação com o banco de dados

Essa atividade tem como objetivo:

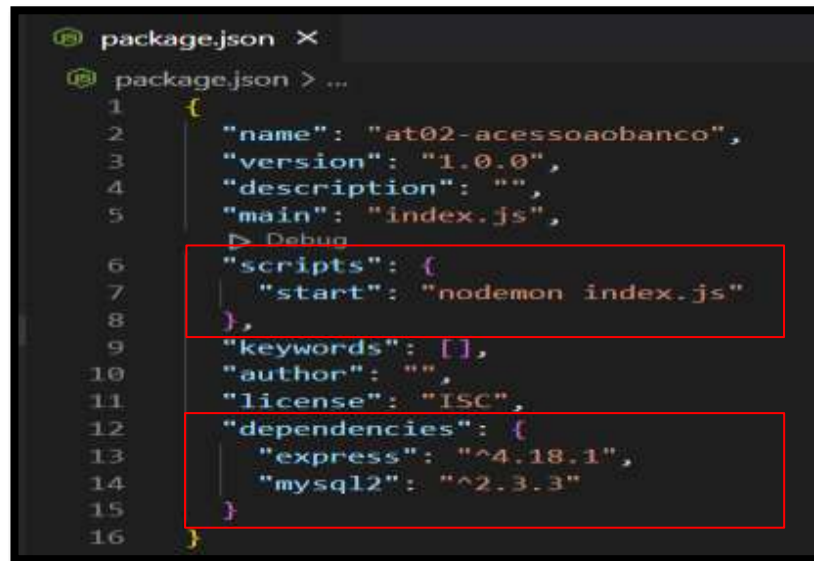
- Desenvolver o script de conexão com o banco de dados;
- Desenvolver uma rota GET e uma rota POST que transacionam com o banco de dados,

Comandos: `npm init -y | npm i express mysql2 | mysql.createConnection`

Parâmetros na url da requisição

Vamos ver como criar uma conexão com o banco de dados e realizar leitura e gravação de dados

1. Para iniciar a atividade, crie um diretório Cap05, dentro dele um subdiretório com o nome “at02-AcessoAoBanco” e depois acesse-o via console;
2. Vamos agora dar o comando para fazer inicialização de um projeto node.js. Acessando o diretório “at02-AcessoAoBanco” emita o comando “**npm init -y**” e aguarde os arquivos “package.json” ser criado.
3. Depois vamos fazer a instalação dos módulos necessários para o nosso projeto. Emita o comando “**npm i express mysql2**”.
4. No arquivo “package.json” insira na chave o script para inicialização via nodemon inserindo a **chave/valor** “start”:“nodemon index.js”, isso permitirá iniciar a aplicação apenas com o comando “**npm start**”. O conteúdo do package.json deve ficar similar ao da imagem a seguir, note também a lista de dependências:



```
package.json X
package.json > ...
1  {
2    "name": "at02-acessoaoabanco",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "nodemon index.js"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "express": "^4.18.1",
14     "mysql2": "^2.3.3"
15   }
16 }
```

5. Após concluir a inicialização do projeto, podemos reaproveitar os templates HTMLs do último exercício do capítulo 4. Por isso vá até a pasta do exercício “Cap04\at05-NeDB” copie a pasta “templates” e cole na pasta do nosso projeto “at02-AcessoAoBanco”.
6. O próximo passo será criar o módulo de conexão com o banco, para isso crie um arquivo no diretório do projeto com o nome “bd.js” e insira o código para a conexão. Para o nosso exemplo ficará com o seguinte código:

```
const mysql = require('mysql2') //Importação do driver do MySQL
//Conexão com o banco de dados
module.exports = ()=> {
  const connection = mysql.createConnection({
    host: 'localhost', //Nome DNS ou IP da hospedagem do banco
    user: 'nodejs', //Usuário do MySQL
    password: '1234', //Senha do MySQL
    database: 'amigo_do_pet' //Nome do Banco da aplicação
  });
  return connection
}
```

7. Agora implemente o script principal criando o arquivo index.js com o seguinte código:

```

const express = require('express')
const app = express()
const con = require('./bd')();
const path = require('path')
const baseDir=path.join(__dirname, 'templates')
var porta = '3200'
app.use(express.urlencoded({extended:true}))
app.use(express.json())

app.get('/', (req, res)=>res.sendFile(`${baseDir}/index.html`))
app.get('/cadastrar', (req, res)=>res.sendFile(`${baseDir}/cadastrar.html`))

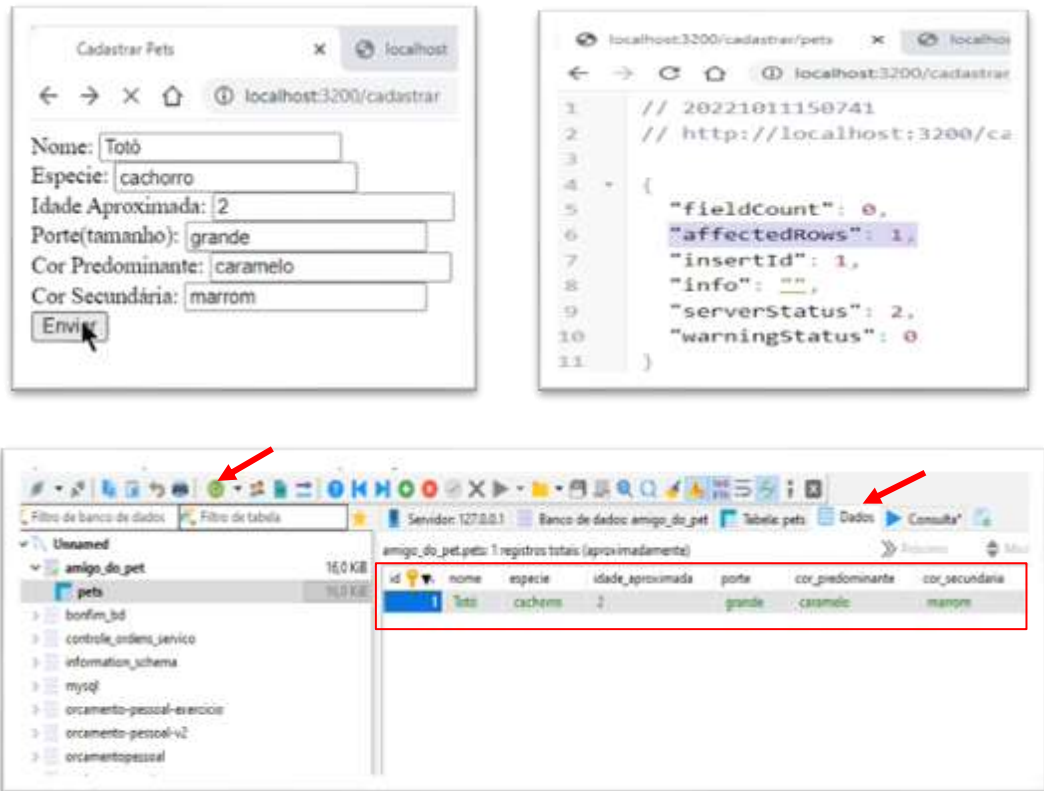
app.post('/cadastrar', async (req, res)=>{
  let {nome, especie, idadeAproximada, porteTamanho, corPredominante,
corSecundaria} = req.body
  let dados = [nome, especie, parseFloat(idadeAproximada), porteTamanho,
corPredominante, corSecundaria]
  const sql = "INSERT INTO pets (nome, especie, idade_aproximada, porte,
cor_predominante, cor_secundaria) VALUES (?, ?, ?, ?, ?, ?)";
  try {
    con.query(sql, dados, (erro, resp)=>{
      let resposta
      if(erro) resposta = {...erro, status:400, message: `Os dados não
foram gravados`}
      else resposta = {...resp, status:200, message: `Sucesso:
${dados.affectedRows} linha(s) alterada(s)`}
      res.json(resposta).status(200)
    })
  } catch (erro) {
    res.send('Erro ao acessar o banco: '+erro).status(400);
  }
})

app.get('/registros', (req, res)=>{
  //ler dados
  const sql = 'SELECT * FROM pets';
  try {
    con.query(sql, (erro, dados)=>{
      if(erro) resposta = {...erro, status:400, message: `Os dados não
foram gravados`}
      else resposta = {...dados, status:200, message: `Sucesso!`}
      res.json(resposta).status(200)
    });
  } catch (erro) {
    res.send('Erro ao acessar o banco: '+erro).status(400);
  }
})

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))

```

8. Por fim, podemos iniciar o serviço da API com o comando “npm start” e realizar os testes de inserção de registros e leitura dos dados no banco através das rotas POST e GET que foram criadas.



Atividade 3 – API-Rest com CRUD completo

Essa atividade tem como objetivo:

- Desenvolver uma API Rest com rotas utilizando os verbos HTTP - GET, POST, PUT e DELETE capaz de executar as transações CRUD o MySQL.

Comandos: mysql2/promise, INSERT, SELECT, UPDATE e DELETE, DESCRIBE

CRUD com API-Rest(verbos HTTP – GET, POST, PUT e DELETE)

Vamos desenvolver um código otimizado de uma API capaz de realizar as transações de inserção, leitura, atualização e exclusão de registros no MySQL através de uma aplicação Express que implementa as principais rotas HTTP (GET, POST, PUT e DELETE). Trabalharemos com o as funcionalidades assíncronas de conexão e acesso ao banco disponíveis no modulo mysql2/promise.

Nesta atividade teremos **dois módulos**, o modulo **bd.js** contendo o script de conexão com o banco e métodos de transação(CRUD) com o banco. Teremos também o modulo **index.js** contendo a aplicação Express e as rotas que consomem os métodos exportados no bd.js.

1. Para iniciar a atividade, crie um subdiretório dentro de Cap05 com o nome **at03-CrudCompleto** e depois acesse-o via console;

2. Inicialize o projeto, instale os módulos **express** e **mysql2** e faça as alterações no arquivo package.json para executar com o nodemon conforme os passos 2,3 e 4 da atividade anterior.
3. Crie um novo arquivo no subdiretório at03-CrudCompleto com o nome **bd.js**. Aqui implementaremos o script de conexão e os métodos do CRUD, todos de forma assíncrona.
4. No bd.js insira o código que trata da conexão com o banco, implemente o seguinte código:

```
//Objeto de conexão
const chaveAcesso = {
  host: 'localhost',
  user: 'nodejs',
  password: '1234',
  database: 'amigo_do_pet'
}
//Função de conexão
const conectar = async () => {
  if (global.statusConexao){
    return global.conexao
  }
  const mysql = require('mysql2/promise')
  const con = await mysql.createConnection(chaveAcesso)
  global.conexao = con
  global.statusConexao=true
  console.log('Conectado ao Banco: ', global.statusConexao)
  return con
}
```

5. Agora vamos implementar de uma função que é capaz de executar qualquer transação com o banco com base em instruções SQL e retorne uma resposta de sucesso ou erro. Segue o código:

```
//Função que recebe a string SQL e executa a transação no banco
const executarQuery = async (sql, dados='') => {
  let con = await conectar()
  try {
    let respBd = await con.query(sql, dados)
    respBd= respBd[1]? respBd[0]: respBd
    return await {dados:[...respBd], status:200, message:'Sucesso'}
  } catch (erro) {
    return {status:400, message:'Inconsistência nas informações: '+erro}
  }
}
```

6. Logo abaixo implementaremos uma função capaz de consultar todos os campos de uma tabela e retorná-los em um array. Essa função será utilizada em todos os métodos do CRUD para a construção da string contendo a instrução SQL. Segue o código:

```
//Função que obtem os campos de uma tabela
const obterCampos = async(tabela)=>{
  let con = await conectar()
  try {
    let sql= `DESCRIBE ${tabela}`
    let campos = await con.query(sql)
    return await campos[0]
  } catch (erro) {
    return `${tabela} não encontrada: ${erro}`
  }
}
```

7. Por fim, implementaremos os métodos a serem exportados: inserir, ler, atualizar e deletar:

```
//Métodos do CRUD a serem exportados e consumidos pelo app principal
module.exports={
  inserir: async (tabela, dados)=>{
    let campos = await obterCampos(tabela) //Obter campos do bd
    campos = campos.map((el)=>el.Field) //Tratamento na string com campos
    campos.shift()
    let argumentos = ''
    campos.forEach(el =>argumentos+=',?')//String com sequência de '?'
    argumentos = argumentos.slice(1)
    campos = campos.toString()
    const sql=`INSERT INTO ${tabela} (${campos}) VALUES (${argumentos});`
    return await executarQuery(sql, dados) //Salvar dados
  },
  ler: async (tabela, id='') => {
    let sql = id==='?' ? `SELECT * FROM ${tabela}` : `SELECT * FROM ${tabela}
WHERE id = ${id};`
    let linhas = await executarQuery(sql)
    linhas.dados = linhas.dados.length==0? linhas.dados=[{message:'Nenhum
registro encontrado'}]:linhas.dados
    return await linhas
  },
  atualizar: async(tabela, dados, id) => {
    let campos = await obterCampos(tabela)
    campos.shift()
    campos = campos.map((el)=>el.Field+'=?')
    campos = campos.toString()
    let sql= `UPDATE ${tabela} SET ${campos} WHERE id =?`
    dados.push(id)
    return await executarQuery(sql, dados)
  },
  deletar: async (tabela, id) => {
    let sql = `DELETE FROM ${tabela} WHERE id = ${id};`
    return await executarQuery(sql)
  }
}
```


8. Vamos criar o módulo **idex.js** no subdiretório **at03-CrudCompleto** e implementar o código instanciando a aplicação Express com os parâmetros já vistos em atividades anteriores, segue o código:

```
const express = require('express')
const app = express()
const bd = require('./bd')
var porta = '3200'
app.use(express.urlencoded({extended:true}))
app.use(express.json())
```

9. Vamos agora implementar as rotas POST, GET, PUT e DELETE que consomem os métodos do CURD no modulo bd.js. Iniciaremos com a rota POST – **“/cadastrar/:tabela”**, segue o código:

```
app.post('/cadastrar/:tabela', async (req, res)=>{
  try {
    let dados = Object.values(req.body).map((val)=>val)
    let tabela = req.params.tabela
    let respBd= await bd.inserir(tabela, dados)
    res.json(respBd).status(200)
  } catch (erro) {
    res.json(erro).status(400)
  }
})
```

10. Acrescente o código das próximas duas rotas GET: **“/consultar/:tabela”** busca todos os registros e **“/consultar/:tabela/:id”** busca apenas um registro baseado no ID. Segue o código:

```
app.get('/consultar/:tabela', async (req, res) => {
  try {
    let tabela = req.params.tabela
    let respBd = await bd.ler(tabela)
    res.json(respBd).status(200)
  } catch (erro) {
    res.json(erro).status(400)
  }
})
app.get('/consultar/:tabela/:id', async (req, res) => {
  try {
    let {tabela, id}= req.params
    let respBd = await bd.ler(tabela, id)
    res.json(respBd).status(200)
  } catch (erro) {
    res.json(erro).status(400)
  }
})
```

11. Implemente rota PUT: **“/editar/:tabela/:id”** que atualizar os registros. Segue o código:

```
app.put('/editar/:tabela/:id', async (req, res) => {
  try {
    let {tabela, id} = req.params
    let dados = Object.values(req.body).map((val)=>val)
    let respBd = await bd.atualizar(tabela, dados, id)
    res.json(respBd).status(200)
  } catch (erro){
    res.json(erro).status(400)
  }
})
```

12. Por último, vamos implementar a rota DELETE: “/excluir/:tabela/:id” que realiza a exclusão de registros. Segue o código:

```
app.delete('/excluir/:tabela/:id', async (req, res) => {
  try {
    let {tabela, id} = req.params
    let respBd = await bd.deletar(tabela, id)
    res.json(respBd).status(200)
  } catch (erro) {
    res.json(erro).status(400)
  }
})
```

13. Para finalizar o script do modulo index.js não podemos esquecer da linha que inicia o serviço da aplicação para “escutar” as requisições no protocolo HTTP:

```
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

14. Execute o script index.js através do comando **npm start**. Caso algum erro for relatado execute o processo de debug conferindo as linhas indicadas no erro ou nas funções relacionadas. Testaremos as funcionalidades de transação com o banco de dados da nossa API na próxima atividade utilizando a ferramenta de teste POSTMAN.

Atividade 4 – Testes de funcionalidades da API-Rest

Essa atividade tem como objetivo:

- Realizar um teste de todas as funcionalidades da API, enviando requisições a todas as rotas criadas – GET, POST, PUT e DELETE

Teste funcionalidades da API utilizando o POSTMAN

Até o momento para consumir os recursos da API utilizamos formulários HTML. Porém os formulários HTML só são capazes de enviar requisições HTTP utilizando os verbos

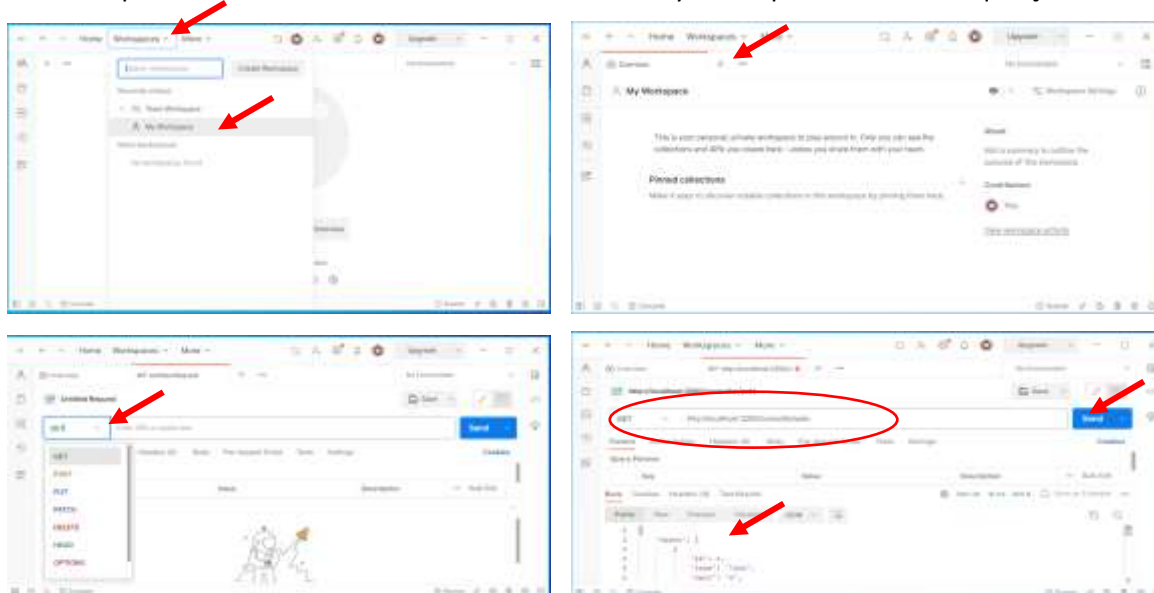
GET e POST por isso iremos utilizar uma ferramenta chamada POSTMAN que nos permite realizar o teste completo da API.

Existe uma versão portable do Postman nos recursos deste curso, ele também pode ser baixado na no link <https://www.postman.com/downloads/>, pode ser utilizado online, como uma extensão do chrome ou ainda extensão do VSCODE. É necessário realizar um cadastro ou fazer login com conta google.

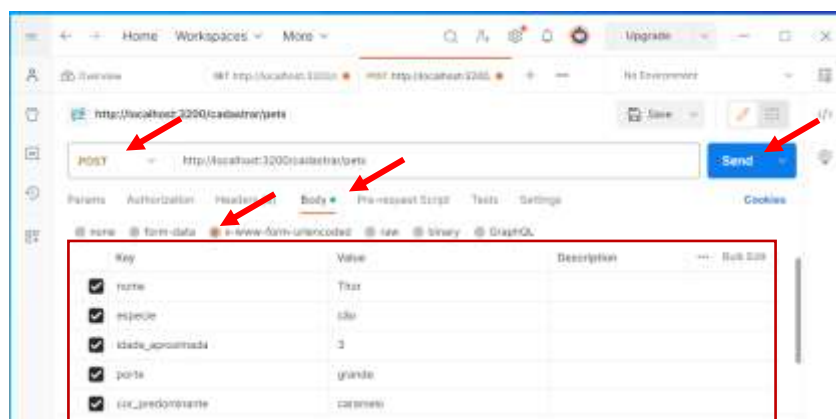
Esta ferramenta será utilizada também no próximo capítulo.

1. Inicie a atividade instalando o Postman e realizando o login, pode-se utilizar a conta Google.
2. Com a aplicação rodando, envie requisições HTTP para cada rotas da API e verifique o retorno.

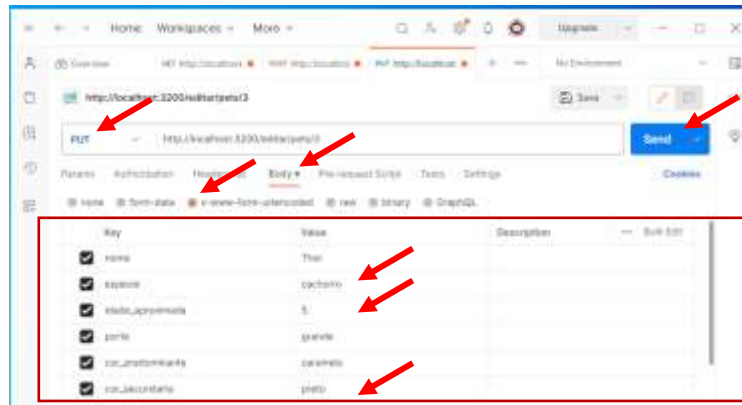
É muito importante que cada rota seja enviada para o verbo HTTP correspondente e o caminho esteja exato. Envie uma requisição GET para a rota <http://localhost:3200/consultar/pets>. Veja exemplos nas abaixo de como acessar a interface My Workspace e realizar requisições:



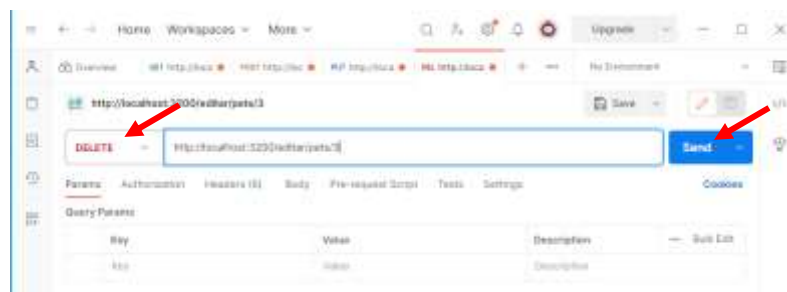
3. Crie outra aba e envie uma requisição para a rota POST <http://localhost:3200/cadastrar/pets>. Neste caso precisamos passar os dados no corpo da requisição, siga as indicações na imagem a seguir:



4. Crie outra aba e envie uma requisição para a rota PUT <http://localhost:3200/editar/pets/3>. Neste caso os dados também precisam ser passados no corpo da requisição, mas também é necessário passar o id do registro a ser editado, como no exemplo o id 3, altere alguns dados do último registro e após a transação confira as alterações no banco de dados. Para realizar um teste siga as indicações na imagem a seguir:



5. Crie outra aba e envie uma requisição para a rota DELETE <http://localhost:3200/excluir/pets/3>. Desta vez, nenhum dado será enviado no corpo da requisição, mas é necessário passar o id do registro a ser excluído, como no exemplo o id 3 será removido, após a transação confira no banco de dados se o registro 3 foi removido. Para realizar um teste siga as indicações na imagem a seguir:



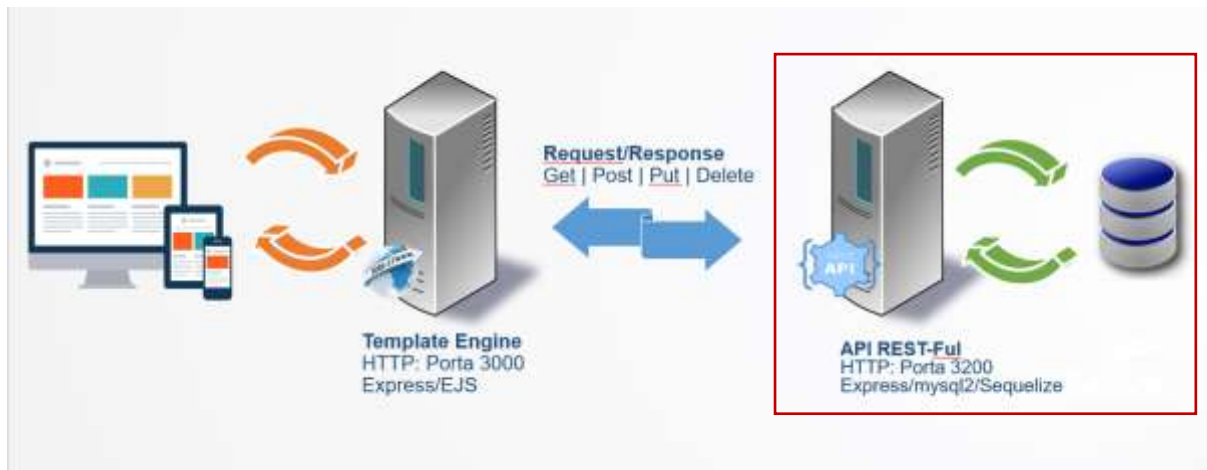
Exercícios extras

Estas atividades extras têm como objetivo criar novas tabelas de dados. Realizar testar as funcionalidades da API realizando as operações de CRUD. Exemplo:

1. Crie outra tabela no banco de dados, como por exemplo “usuários”, com campos como:
 - ✓ id
 - ✓ nome
 - ✓ cpf
 - ✓ email
 - ✓ senha
2. Realize os testes de funcionalidade utilizando o Postman nas rotas GET, POST, PUT e DELETE para a nova tabela.
3. Outras tabelas podem ser criadas a fim de repetir o procedimento de testes.

CAPÍTULO 6 – Manipulação de dados utilizando ORM

A proposta para esta aplicação é desenvolver dois servidores, conforme a imagem a seguir:



Chegou a hora de desenvolver uma versão funcional da API-Rest que fornecerá serviços para a aplicação web.

Neste capítulo você vai ver:

- Manipular tabelas do banco de dados utilizando models do ORM Sequelize;
- Criar relacionamento entre tabelas(models);
- Configuração de CORS;
- Utilização de Middleware;
- Organização do código em módulos no diretório “controllers”;
- Utilização do módulo “consign” para organizar o código.

Atividade 1 – Introdução e instalação do ORM - Sequelize

A sigla ORM significa **Object Relational Mapping**, tem como função principal aproximar o paradigma de orientação a objetos ao paradigma de **bancos de dados relacionais** (MySQL, PostgreSQL, SQL Server, Oracle e outros). Basicamente mapeia as entidades (tabelas) através de models(classes), a fim de abstrair complexidades do SGBD, tais como instruções SQL e relacionamentos entre tabelas.

Essa atividade tem como objetivo:

- Instalar o módulo Sequelize;
- Criar o módulo de conexão com o banco;
- Criar um model;
- Utilizar os principais métodos de manipulação de dados.

Comandos:

- **npm init -y | npm install express mysql2 sequelize**

Exemplo de aplicação CRUD utilizando Express e Sequelize

Vamos ver como criar uma conexão com o banco de dados, um model e utilizar os principais métodos do Sequelize.

1. Para iniciar a atividade, crie um diretório **Cap06**, dentro dele crie um subdiretório com o nome **"at01-Sequelize"** e abra no VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:
 - ✓ npm init -y
 - ✓ npm install express mysql2 sequelizeInsira também o **"start"**: "nodemon index.js" (arquivo package.json vide exemplos anteriores)
3. Crie no MySQL um novo banco(Schema) com o título de "teste".
4. Crie um módulo chamado "bdConexao.js" e insira o seguinte código:

```
//Desestruturação da classe Sequelize
const {Sequelize} = require('sequelize')
//Instancia de um objeto Sequelize
const sequelize = new Sequelize(
  //Lista de parâmetros: banco, usuário e senha
  'teste', 'nodejs', '1234', {
    host:'localhost', //nome DNS ou IP do servidor
    dialect:'mysql', //SGBD utilizado
    charset: 'utf8', //tabela de caracteres
    collate: 'utf8_general_ci', //colação
    timezone:"-03:00" //fuso horário
  })
try {
  //metodo de autenticação que conecta ao banco
  sequelize.authenticate()
  console.log('Conectado ao banco')
} catch (erro) {
  console.log('Não foi possível conectar: ', erro )
}
module.exports = sequelize
```

5. Vamos criar o model para representar a entidade(tabela) usuário. Para isso crie um subdiretório no projeto chamado “models” que irá abrigar os models. Crie dentro de “models” um arquivo chamado “usuario.js” e insira o seguinte código:

```
//Desestruturação das classes DataTypes e Model do módulo sequelize
const { DataTypes, Model } = require('sequelize')

//Importação do módulo de conexão
const sequelize = require('../bdConexao')

//Classe Usuario herdando a classe Model
class Usuario extends Model{}
Usuario.init({
  //Atributos da entidade usuário
  nome: {
    type: DataTypes.STRING,
    allowNull: false
  },
  cpf: {
    type: DataTypes.STRING,
    allowNull: false
  }
},{
  sequelize,
  modelName: 'usuario'
})

sequelize.sync() //Cria a tabela caso não exista
module.exports = Usuario

/*
  Consulte todos os tipos de dados da classe DataTypes em:
  https://sequelize.org/docs/v6/core-concepts/model-basics/#data-types
*/
```

6. Saia do diretório “models” e no diretório raiz do projeto crie o módulo index.js e insira o seguinte código:

```

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

//Instância do objeto (Model)
const usuario = new require('./models/usuario')
var porta = '3200'

app.use(express.urlencoded({extended:false}))
app.use(express.json())

app.get('/', (req, res)=>res.send('API - Amigo do Pet'))

app.get('/consultar/usuarios/:id?', async (req, res)=>{
  //O método findOne busca um registro baseado em uma condição, por
  exemplo o id
  // O método findAll obtém todos os registros da entidade
  let dados = req.params.id? await
usuario.findOne({where:{id:req.params.id}}) :
  await usuario.findAll()
  res.json(dados)
})

app.post('/cadastrar/usuarios', async (req, res)=>{
  let dados = req.body
  //Método create insere novo registro
  let respBd = await usuario.create(dados)
  res.json(respBd)
})

app.put('/atualizar/usuarios/:id', async (req, res) => {
  let id = req.params.id
  let dados = req.body
  //Método update atualiza o registro baseado no id
  let respBd = await usuario.update(dados, {where:{id:id}})
  res.json(respBd)
})

app.delete('/excluir/usuarios/:id', async (req, res) => {
  let id = req.params.id
  //Método destroy exclui um registro baseado no id
  let respBd = await usuario.destroy({where:{id:id}})
  res.json(respBd)
})

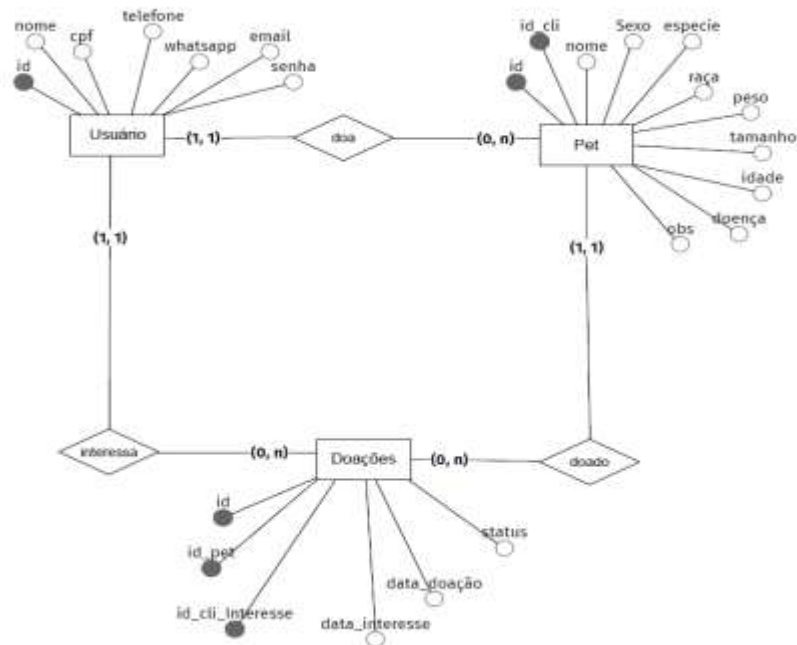
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))

```

7. Rode o código com o comando **npm start**.
8. Execute o procedimento de testes nas rotas GET, POST, PUT e DELETE utilizando o Postmann conforme atividade 4 do capítulo 5.

Atividade 2 – Desenvolvimento de API – REST

Para esta atividade vamos considerar as entidades – Usuário – Pet – Doações os atributos e relacionamentos definidos no MER a seguir:



Essa atividade tem como objetivo:

- Criar models com definição de relacionamentos entre as entidades;
- Estruturar o código da aplicação em módulos no diretório controllers.

Comandos:

- **npm init -y | npm install express mysql2 sequelize cors**

Desenvolvimento da versão 1.0 da API do sistema “Amigo do Pet”.

Nesta aplicação vamos definir as tabelas, campos e relacionamentos do banco de dados através de models do Sequelize. E estruturar o código da aplicação em módulos, utilizando a biblioteca “consign” e o diretório controllers.

1. Para iniciar a atividade, crie no diretório **Cap06** um subdiretório com o nome “**at02-APIRest_Amigo_do_Pet-V_1_0**” e abra no VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:
 - ✓ npm init -y
 - ✓ npm install express mysql2 sequelize cors

Insira também o **"start"**: "nodemon index.js" (arquivo pakete.json vide exemplos anteriores)

3. Crie no MySQL um novo banco(**Schema**) com o título de **"amigo_do_pet"** - Caso o banco de dados dos capítulos anteriores ainda existir, deve ser apagado e criado novamente.
4. Crie um módulo chamado **"bdConexao.js"** e insira o seguinte código:

```
const {Sequelize} = require('sequelize');
const sequelize = new Sequelize(
  'amigo_do_pet', 'nodejs', '1234', {
    host:'localhost',
    dialect:'mysql',
    charset: 'utf8',
    collate: 'utf8_general_ci',
    timezone:"-03:00"
  })
try {
  sequelize.authenticate()
  console.log('Conectado ao banco')
} catch (erro) {
  console.log('Não foi possível conectar: ', erro )
}
module.exports = sequelize
```

5. Crie no diretório raiz do projeto o módulo index.js e insira o seguinte código:

```
const express = require('express')
//Módulo consign facilita a organização de módulos e faz carga automático
de scripts
const consign=require('consign')
const app = express()
const cors = require('cors')
var porta = '3200'
app.use(cors()) //Configura política de segurança CORS
app.use(express.urlencoded({extended:false}))
app.use(express.json())
app.get('/', (req, res)=>res.send('API - Amigo do Pet'))

consign() //Execução do consign
  .include('./controllers/rotas') //Inclui módulos contidos no diretório
específico
  .into(app) //passa a instância do Express o para os módulos

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

6. Crie um subdiretório no projeto chamado **"models"** para os models. Crie dentro de **"models"** um arquivo chamado - **"usuario.js"** e insira o seguinte código:

```

const { DataTypes, Model } = require('sequelize')
const sequelize = require('../bdConexao')
class Usuario extends Model{}
Usuario.init({
  nome: {
    type: DataTypes.STRING(50),
    allowNull: false
  },
  cpf: {
    type: DataTypes.STRING(14),
    allowNull: false
  },
  telefone: {
    type: DataTypes.STRING(14),
    allowNull: true
  },
  whatsapp: {
    type: DataTypes.STRING(14),
    allowNull: true
  },
  email: {
    type: DataTypes.STRING(50),
    allowNull: false
  },
  senha: {
    type: DataTypes.STRING,
    allowNull: false
  }
},{
  sequelize,
  modelName: 'usuario'
})
sequelize.sync()
module.exports = Usuario

```

7. Crie dentro de “**models**” outro arquivo chamado - “**pet.js**” e insira o seguinte código:

```

const { DataTypes, Model } = require('sequelize')
const sequelize = require('../bdConexao')
const usuario = new require('../usuario')
class Pet extends Model{}
Pet.init({
  nome: {
    type: DataTypes.STRING(50),
    allowNull: false
  },
  sexo: {
    type: DataTypes.STRING(1),
    allowNull: false
  },
  especie: {
    type: DataTypes.STRING(50),
    allowNull: false
  },
  raca: {
    type: DataTypes.STRING(50),
    allowNull: true
  },
  peso: {
    type: DataTypes.INTEGER,
    allowNull: true
  },
  tamanho: {
    type: DataTypes.STRING(10),
    allowNull: false
  },
  idade: {
    type: DataTypes.INTEGER,
    allowNull: true
  },
  doenca: {
    type: DataTypes.STRING,
    allowNull: true
  },
  obs: {
    type: DataTypes.STRING,
    allowNull: true
  }
},{
  sequelize,
  modelName: 'pet'
})
usuario.hasMany(Pet) // Usuario tem muitos Pets 1-p-M
Pet.belongsTo(usuario) //Pet pertence a Usuário 1-p-1
sequelize.sync()
module.exports = Pet

```

8. Crie dentro de “**models**” um arquivo chamado - “**doacao.js**” e insira o seguinte código:

```
const { Sequelize, DataTypes, Model } = require('sequelize')
const sequelize = require('../bdConexao')
const usuario = new require('../usuario')
const pet = new require('../pet')
class Doacao extends Model{}
Doacao.init({
  data_interesse: {
    type:Sequelize.DATEONLY,
    defaultValue: DataTypes.NOW
  },
  data_doacao:{
    type:Sequelize.DATEONLY,
    allowNull: true
  },
  status: {
    type: DataTypes.STRING,
    defaultValue:"Cadastrada"
  }
},{
  sequelize,
  modelName:'doacoes'
})
pet.hasMany(Doacao) //Muitos Pets têm muitas Doações - M-p-M
Doacao.belongsTo(pet)

usuario.hasMany(Doacao) //Muitos Usuários têm muitas Doações - M-p-M
Doacao.belongsTo(usuario)

sequelize.sync()
module.exports = Doacao
```

9. Crie um subdiretório no projeto chamado “**controllers**” e dentro dele outro subdiretório chamado “**rotas**”, conforme definido no método include do consign.
10. Dentro do subdiretório “rotas” vamos criar um módulo para implementarmos as rotas de cada entidade. Vamos começar criando em “rotas” um arquivo “**usuario.js**” e inserindo o seguinte código:

```

const model = new require('../models/usuario')
const rota = 'usuarios'
module.exports = (app)=>{
  app.get(`/${rota}/:id?`, async (req, res)=>{
    try {
      let dados=req.params.id?
        await model.findOne({where:{id:req.params.id}}):
        await model.findAll()
      res.json(dados).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.post(`/${rota}`, async (req, res)=>{
    try {
      let dados = req.body
      const salt = bcrypt.genSaltSync()
      dados.senha = bcrypt.hashSync(dados.senha, salt)
      let respBd = await model.create(dados)
      res.json(respBd).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.put(`/${rota}/:id`, async (req, res) => {
    try {
      let id = req.params.id
      let dados = req.body
      console.log(dados)
      let respBd = await model.update(dados, {where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.delete(`/${rota}/:id`, async (req, res) => {
    try {
      let id = req.params.id
      let respBd = await model.destroy({where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
}

```

11. Agora crie em “rotas” um módulo chamado “**pet.js**” e insira o seguinte código:

```

const model = new require('../../models/pet')
const usuario = new require('../../models/usuario')
const rota = 'pets'
module.exports = (app)=>{
  app.get(`/${rota}/:id?`, async (req, res)=>{
    try {
      let dados = req.params.id?
        await model.findOne({include:[{model:usuario}],
{where:{id:req.params.id}}}) :
        await model.findAll({include:[{model:usuario}], {raw:
true, order:[['id','DESC']]})
      res.json(dados).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.post(`/${rota}`, async (req, res)=>{
    try {
      let dados = req.body
      console.log(dados)
      let respBd = await model.create(dados)
      res.json(respBd).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.put(`/${rota}/:id`, async (req, res) => {
    try {
      let id = req.params.id
      let dados = req.body
      console.log(dados)
      let respBd = await model.update(dados, {where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.delete(`/${rota}/:id`, async (req, res) => {
    try {
      let id = req.params.id
      let respBd = await model.destroy({where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
}

```

12. Para finalizar, crie em “rotas” um módulo chamado “doacao.js” e insira o seguinte código:

```

const model = new require('../../models/doacao')
const usuario = new require('../../models/usuario')
const pet = new require('../../models/pet')
const rota = 'doacoes'
module.exports = (app)=>{
  app.get(`/${rota}/:id?`, async (req, res)=>{
    try {
      let dados = req.params.id?
        await model.findOne({include:[{model:usuario},
{model:pet}]}], {where:{id:req.params.id}}) :
        await model.findAll({include:[{model:usuario},
{model:pet}]}], {raw: true, order:[['id','DESC']]})
      res.json(dados).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.post(`/${rota}`, async (req, res)=>{
    try {
      let dados = req.body
      let respBd = await model.create(dados)
      res.json(respBd).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.put(`/${rota}/:id`, async (req, res) => {
    try {
      let id = req.params.id
      let dados = req.body
      let respBd = await model.update(dados, {where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.delete(`/${rota}/:id`, async (req, res) => {
    try {
      let id = req.params.id
      let respBd = await model.destroy({where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
}

```

13. Rode o código com o comando **npm start** e execute os testes de funcionalidades da API utilizando o Postman.

Atividade 3 – Autenticação e criptografia

Para realizar o processo de autenticação da API utilizaremos a biblioteca JWT(JSON Web Token) que armazenar de forma segura e compacta objetos JSON que armazenam informações de autenticação usuário em um token criptografado utilizando o padrão Base64.

A senha do usuário é um dado sensível e não pode ser gravado em texto puro no banco de dados pois ficariam expostas. Desta forma, é necessário gerar um hash(senha criptografada) que será gravada no banco de dados. Durante o processo de login a senha digitada também transformada em hash e comparada com o hash gravado no banco.

Essa atividade tem como objetivo:

- Realizar o processo de autenticação na API;
- Gerar um token para autenticação utilizando “JWT”;
- Utilizar senhas criptografadas utilizando a biblioteca “bcrypt”;
- Criar middleware de autenticação em rotas protegidas.

Desenvolvimento da versão 1.1 da API do sistema “Amigo do Pet”.

Nesta aplicação vamos criar um middleware de autenticação e vincular às rotas protegidas, bem como desenvolver funções para gravar hash de senha e realizar o processo de login.

1. Para iniciar a atividade, duplique o diretório “at02-APIRest_Amigo_do_Pet-V_1_0”, renomeie para “at03-APIRest_Amigo_do_Pet-V_1_1” e abra no VSCode;
2. Emita o comando de instalação das novas dependências JWT e bcrypt:
npm install bcrypt jsonwebtoken
3. No subdiretório “controllers” crie um módulo chamado “auth.js” e insira o seguinte código:

```

const jwt = require('jsonwebtoken')
const bcrypt = require('bcrypt')
const jwtSecret = 'errtyytuczvxvdgghfrytddvfgfvzcvfehrew'
const model = new require('../models/usuario')
module.exports={
  criptografarSenha: async(senha)=>{
    const salt = bcrypt.genSaltSync(12)
    return bcrypt.hashSync(senha, salt)
  },
  //https://www.base64decode.org/
  gerarToken: async(usuario)=> await jwt.sign(usuario, jwtSecret,
{expiresIn:'1h'}),

  //Compara o hash da senha enviada na requisição com o hash do banco
  validarSenha: async(senha, hashSenha)=> await bcrypt.compare(senha,
hashSenha),

  validarToken:(req, res, next)=>{
    try {
      let token = req.headers.authorization
      token = token.split(' ');
      token = token[1]
      jwt.verify(token, jwtSecret, (erro, dados)=>{//Verifica a
validade do token
        if (erro){
          res.json({message:'Token inválido!', error:erro
}).status(400)
        } else {
          req.token = token //Insere o token na requisição
          req.usuarioAtual = {...dados} //Insere os dados do usuario
atual na requisição
          next() //CallBack que executa a proxima função(no caso a
rota protegida)
        }
      })
    } catch (erro) {
      res.json({message:'Não existe token na
requisição.'}).status(400)
    }
  }
}

```

4. Crie outro módulo em “controllers” com o nome “validacao.js” com o seguinte código:

```

const auth = require('./auth')
module.exports = {
  validarCadastro: async(dados, model)=>{
    let usuario = await model.findOne({ where: { email: dados.email }
  })

    if (usuario!=null){//Verifica se o email já está cadastrado
      return {erro:'email inválido', message:"Email já cadastrado!"}
    }
    if (dados.senha != dados.confirmacao){ //verifica se a senha e a
    confirmação conferem
      return {erro:'senha', message:"Senhas não coincidem!"}
    }
    return {validacao:true} //retorna o status de validação verdadeiro
  },
  validarLogin: async (dados, model)=>{
    //Carrega os dados do banco
    let usuario = await model.findOne({ where: { email:dados.email } })
    if ( usuario==null){ //verifica se o email enviado pela requisição
    existe
      return {message:'Conta de email inválida.', autenticado:false}
    } else {
      //verifica se o hash da senha enviada confere com o hash do
    banco
      let authSenha = await auth.validarSenha(dados.senha,
    usuario.senha)
      //usuario = authSenha? {...usuario} : {message:'Senha
    inválida'}
      return authSenha? {usuario,
    autenticado:true}:{erro:{message:'Senha inválida'}, autenticado:false}
    }
  }
}

```

5. No diretório “rotas” crie um módulo “login.js” e insira o seguinte código:

```

const model = new require('../models/usuario')
const auth = require('../auth')
const validacao = require('../validacao')

module.exports = (app) => {
  app.post('/login', async (req, res) => {
    try {
      let dados = req.body
      let validaLogin = await validacao.validarLogin(dados, model)
      if (validaLogin.autenticado) { // Verifica se o email e senha são
consistentes
        let {id, nome, email} = validaLogin.usuario.dataValues
        dados = {id, nome, email} // Desestruturação dos dados validados
        let token = await auth.gerarToken(dados) // Gera um token JWT
        return res.json({dados, autenticado: true, token: token}).status(200)
      } else {
        return res.json(validaLogin).status(200)
      }
    } catch (error) {
      return res.json(error).status(400)
    }
  })

  // Teste para verificar autenticação
  app.get('/testeAuth', auth.validarToken, async (req, res) => {
    const usuarioAtual = req.usuarioAtual
    const token = req.token
    res.json({usuarioAtual, token})
  })
}

```

6. Agora basta inserir as seguintes linhas de importação dos métodos de autenticação em cada módulos que define rotas das entidades (no diretório /controllers/rotas)

```

const auth = require('../auth')
const validacao = require('../validacao')

```

7. É necessário atualizar as rotas POST e PUT da entidade “usuário”, para que passe a gravar o hash da senha ao invés de senhas com texto puro. No arquivo “./controllers/rotas/usuario.js” realize a importação dos módulos “auth” e “validação” e reescreva as rotas POST e PUT conforme os códigos a seguir:

```

app.post(`/${rota}`, async (req, res)=>{
  try {
    let dados = req.body
    let dadosLogin = await validacao.validarCadastro(dados, model)
    if (dadosLogin.validacao){
      dados.senha = await auth.criptografarSenha(dados.senha)
      let respBd = await model.create(dados)
      delete respBd.dataValues.senha
      res.json(respBd).status(201)
    } else {
      res.json(dadosLogin).status(200)
    }
  } catch (error) {
    res.json(error).status(422)
  }
})

app.put(`/${rota}/${id}`, auth.validarToken, async (req, res) => {
  try {
    let id = req.params.id
    let {nome, cpf, telefone, whatsapp} = req.body
    //Para evitar atualização de dados de Login, atualizar estes
    dados exigem regras específicas de validação
    let dados = {nome:nome, cpf:cpf, telefone:telefone,
    whatsapp:whatsapp}
    console.log(dados)
    let respBd = await model.update(dados, {where:{id:id}})
    console.log(respBd)
    res.json(respBd).status(200)
  } catch (error) {
    res.json(error).status(400)
  }
})

```

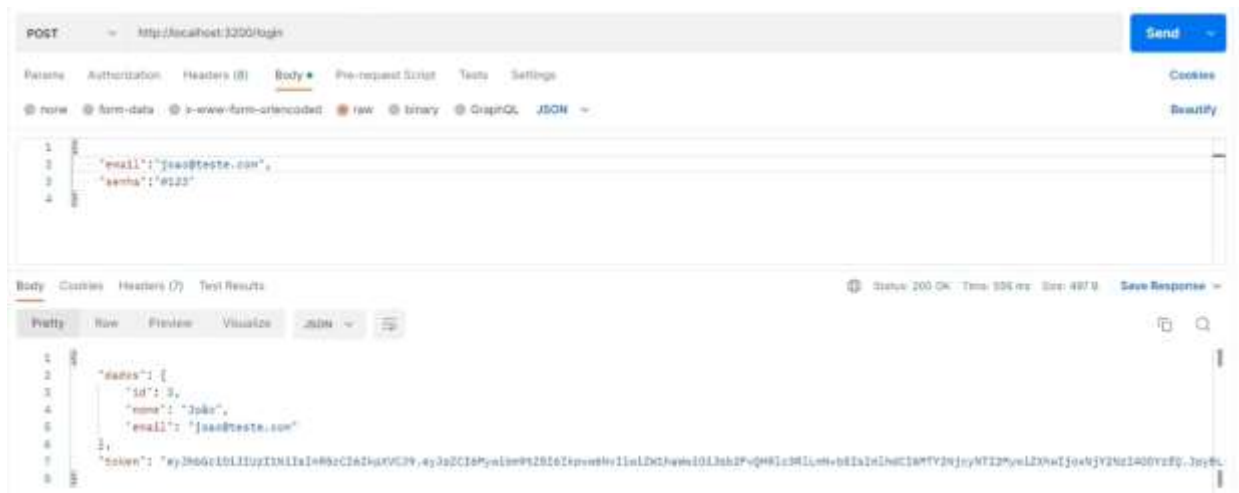
8. E inserir **EM TODAS AS ROTAS** que devem ser **PROTEGIDAS** o meddleware “**auth.validarToken**”, seguindo a sintaxe do exemplo abaixo:

```

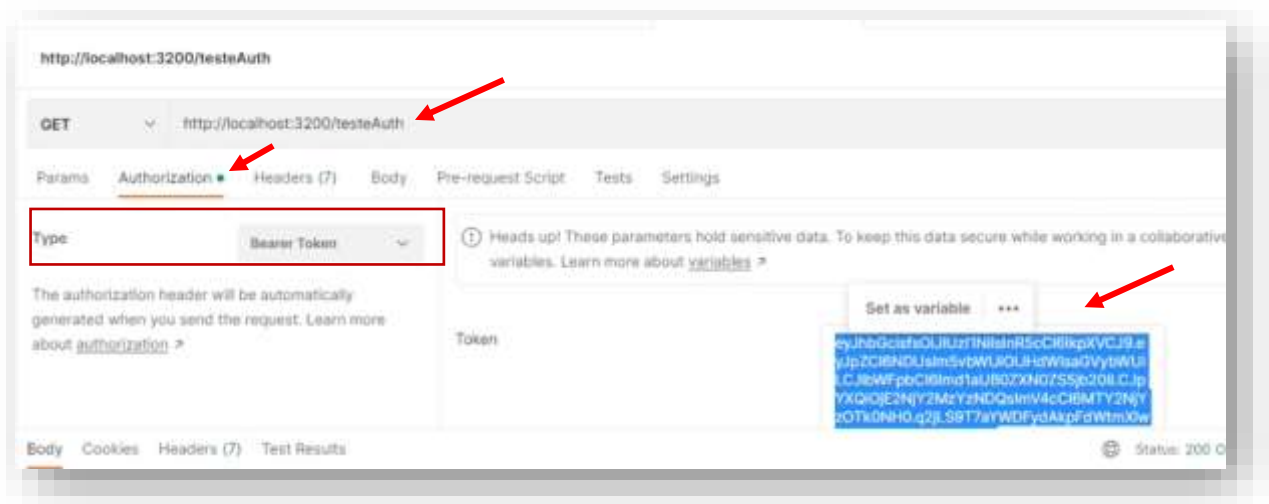
app.put(`/${rota}/${id}`, auth.validarToken, async (req, res) => {

```

9. Vamos realizar primeiro um teste ao inserir registro de alguns usuários e depois observar o campo senha no banco de dados e perceber que o campo senha não contém a senha em texto puro, e sim um **hash** (senha criptografada).
10. Ao executar a rota “/login” com dados consistentes(combinação de e-mail e senha que existam no banco de dados), a resposta no Postman será a seguinte:



11. Agora é possível testar o acesso a uma rota protegida. Para isso, acesse uma rota protegida como no exemplo da imagem abaixo, **copiando o token gerado pela rota login** e colando em no cabeçalho “Authorization” com a clausula “Type” configurada com o valor “Bearer Token”:



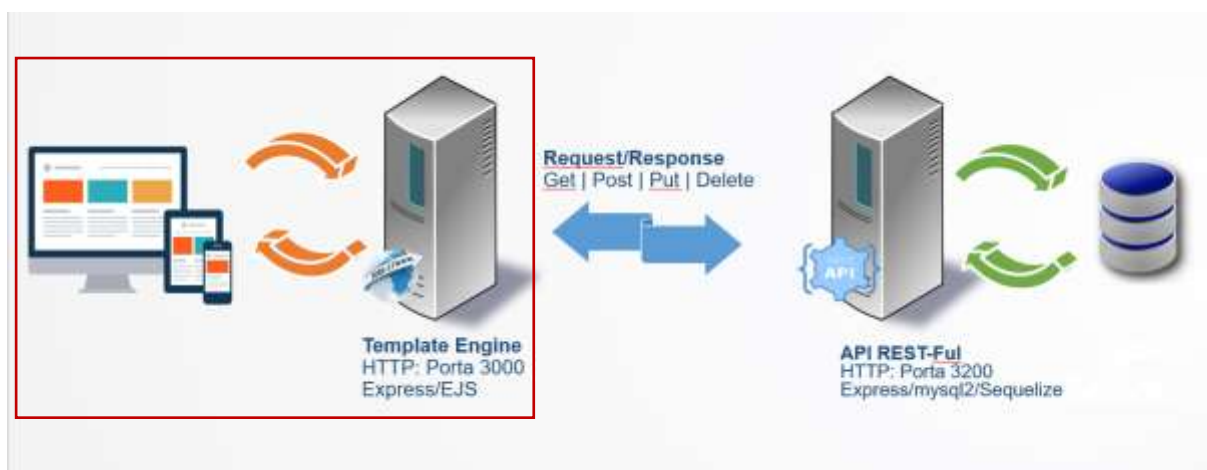
Exercícios extras

Estas atividades extras têm o objetivo aperfeiçoar o processo proteção de rotas e teste de funcionalidade utilizando o Postman.

1. Inserir o middleware “**auth.validarToken**” nas rotas das entidades “pets” e “doações” que devem ser protegidas;
2. Fazer login e realizar teste acessando rotas GET, POST, PUT e DELETE com e sem o token e verificando o comportamento da aplicação (se nega ou autoriza o acesso conforme a passagem do token)

CAPÍTULO 7 – View Engine EJS e integração front-end com API

Neste capítulo vamos trabalhar na aplicação que irá consumir os serviços fornecidos pela API e renderizar *templates* HTML(views) a partir dos dados fornecidos pela API.



Neste capítulo você vai ver:

- Utilizando a biblioteca EJS;
- Gerar códigos HTML utilizando as principais estruturas de programação;
- Utilizar a biblioteca Axios para fazer requisições HTTP a partir do front-end.

Atividade 1 – Introdução ao EJS e estrutura condicional IF

Essa atividade tem como objetivo:

- Iniciar uma aplicação Express com o EJS
- Instalar a biblioteca EJS
- Sintaxe de estrutura condicional IF utilizando o EJS

O EJS (Embedded JavaScript Templating) é um Template Engine em aplicações Node.js.

Comandos:

- **npm init -y | npm install express ejs cors**

Aplicação Express/EJS

Vamos ver a criação de uma aplicação que combina os módulos Express e EJS a fim de gerar respostas às requisições renderizando templates.

1. Para iniciar a atividade, crie um diretório **Cap07**, dentro dele crie um subdiretório com o nome **“at01-EJS-Introdução_estruturalF”** e abra com o VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:

- ✓ **npm init -y**
- ✓ **npm install express ejs cors**

Insira também a entrada **"start": "nodemon index.js"** (arquivo **package.json** vide exemplos anteriores).

3. Crie o arquivo **“index.js”** e insira o seguinte código:

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

var porta = '3000'
app.use(express.urlencoded({extended:true}))
app.use(express.json())

// Configura o Express p/ usar o EJS como View engine
app.set('view engine','ejs')

//Define diretório de arquivos estáticos front-end(css, imagens, javascript)
app.use(express.static('public'))

//Rota padrão com parâmetro opcional
app.get('/:nome?', async (req, res)=>{
  let {nome} = req.params

  //Responde uma view com um objeto contendo dados a serem renderizados
  res.render('index', {nome})
})

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

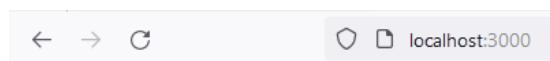
4. Agora é hora de criarmos o diretório que irá guardar os arquivos dos templates. Crie na raiz do projeto um diretório chamado **“views”**. Este diretório é utilizado por padrão pelo EJS.
5. Os templates são criados utilizando a estrutura e a linguagem de marcação HTML, porém para serem renderizados pelo **“view engine - EJS”** precisam ser criados com a **extensão EJS**. Então crie um arquivo **“index.ejs”** (dentro do diretório **“views”**) com o seguinte código:


```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Home</title>
</head>
<body>
  <h1>Olá EJS</h1>
  <h2>Seja bem vindo!</h2>
</body>
</html>

```

6. A rota padrão / da aplicação envia o "index.ejs" como resposta a requisição HTTP. Rode o código e acesse a rota padrão <http://localhost:3000> e perceba que a página foi rederizada.



Olá EJS

Seja bem vindo!

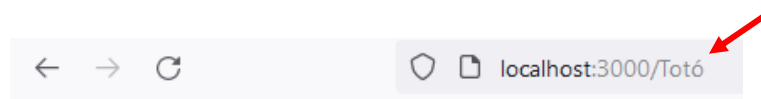
7. Até então, não existe diferença entre uma página HTML comum, mas as funcionalidades das views EJS vão muito além da linguagem de marcação. Pode-se utilizar estruturas complexas como condicionais e laços de repetição. A próxima implementação é um exemplo da sintaxe da estrutura condicional IF em uma view EJS ao verificar **se "nome" foi ou não** passado como parâmetro na rota padrão. Edite o <body> do "index.ejs" conforme a seguir:

```

<body>
  <h1>Olá EJS</h1>
  <%if (nome){%>
    <h2>Seja bem vindo <%= nome %>!</h2>
  <%} else {%>
    <h2>Seja bem vindo!</h2>
  <%}%>
</body>

```

8. Rode o código e teste a rota padrão passando um parâmetro, como no exemplo <http://localhost:300/Totô> e observe o resultado:



Olá EJS

Seja bem vindo Totó!

9. Vamos aperfeiçoar o exemplo acrescentando arquivos estáticos de estilo, imagens e scripts. Para isso crie um diretório na raiz da aplicação com o nome **“public”**, que foi predefinido no middleware `“app.use(express.static('public'))”` do arquivo `“index.js”`. **Copie** a pasta e **“img”** fornecidas no material do Capítulo 7 e **cole** dentro do diretório `“public”`. Crie um diretório chamado **“css”** e dentro dele um arquivo **“style.css”** com o seguinte código:

```
body{
  background: url('../img/background.jpg');
}
h1{
  text-align: center;
  color: blue;
}
h2{
  text-align: center;
  color: red;
}
img{
  display: block;
  margin: 0 auto;
  width: 800px;
  height: 600px;
}
```

10. Agora vincule o arquivo de estilo em cascata na view, inserindo a seguinte linha na tag `<head>` do arquivo `“index.ejs”`:

```
<link rel="stylesheet" href="/css/style.css">
```

11. Ajuste também no arquivo `“index.ejs”` no fim da tag `<body>` acrescentando o seguinte código:

```
<%if (especie=='cachorro'){%>
  
<%} else if (especie=='gato') {%>
  
<%} else {%>
  
<%}%>
```

12. Mais um ajuste é necessário no arquivo “index.js” para realizar o tratamento de um dado adicional “espécie”. Edite a rota padrão da seguinte forma:

```
//Rota padrão com parâmetros opcionais
app.get('/:nome?/:especie?', async (req, res)=>{
  let {nome, especie} = req.params

  //Responde uma view com um objeto contendo dados a serem renderizados
  res.render('index', {nome, especie})
})
```

13. Rode o código e teste a aplicação. Pode-se utilizar como exemplo as seguintes URLs <http://localhost:3000/> | <http://localhost:3000/Totó/cachorro> | <http://localhost:3000/Felix/gato> com resultados conforme as respectivas imagens:



Atividade 2 – Implementando a estrutura de repetição no EJS

Essa atividade tem como objetivo:

- Apresentar a sintaxe da **estrutura de repetição ForEach** em views EJS;
- Criar páginas dinâmicas a partir da iteração de estrutura de dados(vetor de objetos).

Páginas dinâmicas exibindo dados

Vamos criar uma aplicação que recebe um objeto contendo dados e cria uma view exibindo estes dados dinamicamente. A exemplo de atividades do capítulo 6, nesta atividade também utilizaremos a biblioteca “**consign**” para organizar o código das rotas em diretórios e módulos.

1. No diretório **Cap07**, crie um subdiretório com o nome “**at02-EJS-EstruturaForEach**” e abra com o VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:

✓ **npm init -y**

✓ **npm install express ejs cors consign**

Insira também a entrada **"start": "nodemon index.js"** (arquivo **package.json** vide exemplos anteriores).

3. Crie o arquivo **"index.js"** e insira o seguinte código:

```
const express = require('express')
const app = express()
const consign=require('consign')
var porta = '3000'
app.use(express.urlencoded({extended:true}))
app.use(express.json())

// Configura o Express p/ usar o EJS como View engine
app.set('view engine','ejs')

//Define diretório para arquivos estaticos(css, imagens, js(front-end))
app.use(express.static('public'))

consign()
  .include('./controllers/rotas')
  .into(app)

app.get('/', async (req, res)=>{

  res.render('index')
})

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

4. Por estar utilizando o consign, crie o diretório **"controllers"** e dentro dele crie o subdiretório **"rotas"**, conforme foi especificado em `.include('./controllers/rotas')`. Dentro de **rotas**, crie um arquivo **"pets.js"** com o seguinte código:

```
module.exports =(app)=>{
  app.get('/pets', async (req, res)=>{
    //Vetor de objetos com dados de exemplo a serem renderizados na view
    let pets=[
      {nome:'Totó', especie:'cachorro', idade:3},
      {nome:'Feliz', especie:'gato', idade:2},
      {nome:'Betoven', especie:'cachorro', idade:1},
    ]
    res.render('pets', {pets:pets})
  })
}
```

5. Crie o diretório **"views"** na raiz do projeto e crie o arquivo **"index.ejs"** com o seguinte código:

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="/css/style.css">
  <title>Home</title>
</head>
<body>
  <h1>Olá EJS</h1>
  <%if (nome){%>
    <h2>Seja bem vindo! Conheça <%= nome %>!</h2>
  <%} else {%>
    <h2>Seja bem vindo</h2>
  <%}%>
</body>
</html>

```

6. Agora crie um arquivo em views chamado “**pets.ejs**” com o seguinte código:

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- CSS -->
  <link rel="stylesheet" href="css/normalize.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.3.0/font/bootstrap-icons.css" >
  <link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"/>
  <link rel="stylesheet" href="/css/style.css">
<!-- JavaScript -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/js/bootstrap.bundle.min.js"></script>

  <title>Pets</title>
</head>
<body>
  <div class="container">
    <div class="card">
      <div class="card-title">

      </div>
      <div class="card-body">

```

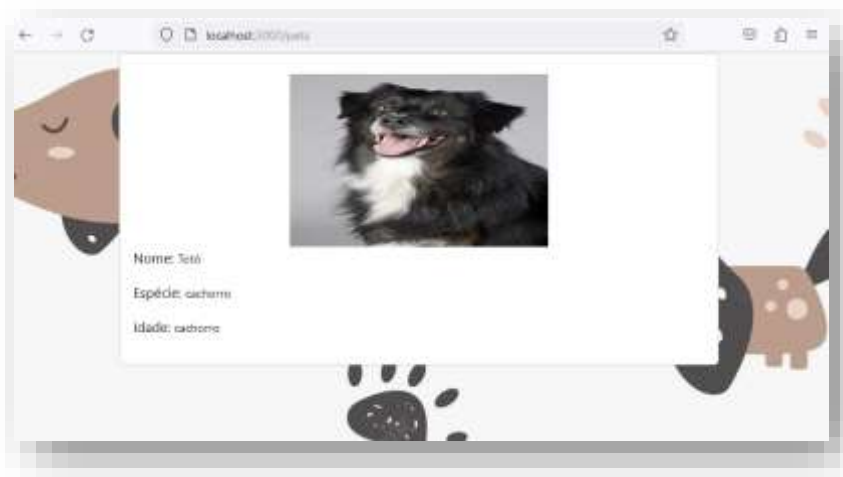
```

    <div class="card-img">
      <%if (pets[0].especie=='cachorro'){%>
        
      <%} else if (pets[0].especie=='gato') {%>
        
      <%} else {%>
        
      <%}%>
    </div>
    <div class="card-text">
      <p>Nome: <small><%= pets[0].nome %></small></p>
      <p>Espécie: <small><%= pets[0].especie %></small></p>
      <p>Idade: <small><%= pets[0].especie %></small></p>
    </div>
  </div>
</div>
</div>
</body>
</html>

```

Obs.: As tags com a importação CSS e JS do bootstrap(podem ser obtidas com o link CDN no site get.bootstrap.com).

7. Rode o código e realize o teste na rota <http://localhost:300/pets> e verifique que está aparecendo apenas a exibição dos dados do primeiro elemento do vetor, isso porque ficou definido no código a exibição somente o primeiro elemento do vetor(`pets[0]`). Veja a exibição abaixo:



8. Agora vamos aperfeiçoar o código para exibir todos os dados que forem passados para a view. Para isso precisaremos de uma estrutura de repetição para iterar o vetor contendo os dados, utilizaremos a função `forEach()`. Refatore o código do arquivo "pets.ejs" dentro da tag `<body>` conforme o código a seguir:

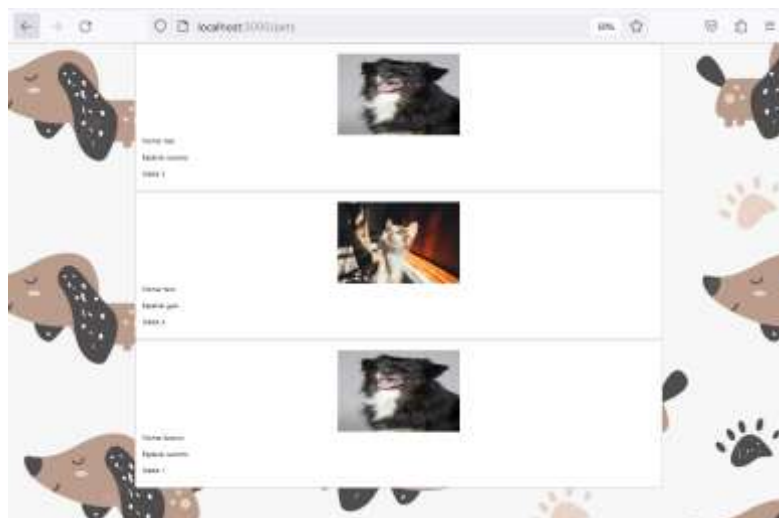
```

<body>
  <div class="container">

    <!--Estrutura forEach iterando do vetor contendo os dados-->
    <% pets.forEach((pet)=>{ %>
      <div class="card">
        <div class="card-title">
        </div>
        <div class="card-body">
          <div class="card-image">
            <%if (pet.especie=='cachorro'){%>
              
            <% } else if (pet.especie=='gato') {%>
              
            <% } else {%>
              
            <%}%>
          </div>
          <div class="card-text">
            <p>Nome: <small><%= pet.nome %></small></p>
            <p>Espécie: <small><%= pet.especie %></small></p>
            <p>Idade: <small><%= pet.idade %></small></p>
          </div>
        </div>
      </div>
    <% }) %>
  </div>
</body>

```

9. Rode o código e realize o teste na rota <http://localhost:3000/pets> e verifique agora todos os dados foram renderizados. Veja o resultado na imagem abaixo:



Atividade 3 – Utilização de Partials no EJS

Partial é uma parte reutilizável de código HTML e pode ser incluída em várias views conforme a necessidade. Pode ser usada para definir partes que se repetem em diferentes páginas do site, como por exemplo um cabeçalho, rodapé, menus, barra lateral ou qualquer outra seção comum entre as páginas. O conceito de Partials ajuda a manter o código limpo, modularizado e organizado. Promove a reutilização de código, evita duplicações e facilita a manutenção por atualizar apenas um único módulo e a modificação ter efeito em todas as views que utiliza a partial.

Essa atividade tem como objetivo:

- Aprender a utilizar o recurso de Partials no EJS;
- Reestruturar a aplicação em partial, a fim de manter o mesmo código do cabeçalho, menu e rodapé para todas as views;

Aplicação estruturada em Partials

Para criar um partial em EJS, define-se uma seção de código HTML em um arquivo separado e depois pode incluí-lo em outras páginas sempre que for necessário utilizando a tag `<%- include('/partials/arquivo.ejs') %>`.

1. No diretório **Cap07**, crie um subdiretório com o nome **“at03-EJS-Partials”** e abra com o VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:

- ✓ **npm init -y**
- ✓ **npm install express ejs cors**

Insira também a entrada **"start"**: "nodemon index.js" (arquivo package.json vide exemplos anteriores).

3. Crie o arquivo **“index.js”** e insira o seguinte código:


```

const express = require('express')
const app = express()
const consign=require('consign')
var porta = '3000'
app.use(express.urlencoded({extended:true}))
app.use(express.json())

// Configura o Express p/ usar o EJS como View engine
app.set('view engine','ejs')

//Define diretório para arquivos estaticos(css, imagens, js(front-end))
app.use(express.static('public'))

consign()
  .include('./controllers/rotas')
  .into(app)

app.get('/', async (req, res)=>{

  res.render('index')
})

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))

```

4. Crie a estrutura de diretórios, no diretório raiz do projeto crie o diretório “controllers” e dentro de “controles” o subdiretório “rotas”.
5. **Copie** os diretórios **public e views** fornecidos como “recursos” do capítulo 07. Nestas pastas contêm arquivos estáticos (css, scripts e imagens do front-ent) e também páginas HTML estáticas que iremos dividir em partials.
6. No diretório “**views**” crie um subdiretório chamado “**partials**”. E renomeie os arquivos “**index.html**” para “**index.ejs**” e “**login.html**” para “**login.ejs**”.
7. No arquivo “**index.ejs**” recorte o trecho do **código equivalente ao cabeçalho**(linhas as 1 a 17) e substitua pela entrada:

```
<%- include('partials/head.ejs') %>
```

8. No subdiretório “**partials**” crie um arquivo chamado “**head.ejs**” e cole o trecho do código equivalente ao cabeçalho que recortado:

```

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- CSS -->
  <link rel="stylesheet" href="css/normalize.css">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.3.0/font/bootstrap-icons.css">
  <link rel="stylesheet"
href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"/>
  <link rel="stylesheet" type="text/css" href="css/style.css">

  <!-- JavaScript -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/js/bootstrap.bundle.min
.js"></script>
  <script src="./js/index.js"></script>

```

9. No arquivo “index.ejs” recorte o trecho do código equivalente ao menu (linhas as 23 a 64 demarcados pelo comentário `<!--navBar-->`) e substitua pela entrada:

```
<%- include('partials/menu.ejs') %>
```

10. No subdiretório “partials” crie um arquivo chamado “**menu.ejs**” e cole o trecho do código equivalente a navbar que recortado:

```

<div class="container-fluid">
  <nav class="row navbar navbar-expand-lg navbar-light bg-light fixed-top clearfix"
role="navigation">
    <div class="col-sm-12 col-md-12 col-lg-2">
      <div class="row">
        <div class="col-sm-6 col-md-10 col-lg-12 mx-auto" style="max-width: 200px;">
          <a class="navbar-brand" href="/">
            
          </a>
        </div>
        <div class="col-sm-2 col-md-2 col-lg-0 my-auto py-2" style="max-width: 100px;">
          <div class="mx-auto">
            <button id="btnToggleMenu" class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#menuColapse" aria-controls="menuColapse" aria-
expanded="true" aria-label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
            </button>
          </div>
        </div>
      </div>
    </div>
    <div class="col-sm-12 col-md-12 col-lg-10">
      <div class="container">
        <div class="collapse navbar-collapse" id="menuColapse">
          <ul id="listaMenu" class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a href="/" class="nav-link clearfix fs-5 text-warning"
onclick="toggleMenu()">HOME</a>
            </li>
          </ul>
          <ul class="navbar-nav mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link fs-5 text-warning" aria-current="page"
href="/login">Login</img></a>
            </li>
          </ul>
        </div>
      </div>
    </div>
  </nav>
  <hr>
</div>

```

11. No arquivo “index.ejs” recorte o trecho do código equivalente ao carousel (linhas as 65 a 81 demarcados pelo comentário <!-- Carousel -->) e substitua pela entrada:

```
<%- include('partials/carousel.ejs') %>
```

12. No subdiretório “partials” crie um arquivo chamado “**carousel.ejs**” e cole o trecho do código equivalente a carousel que recortado:

```
<div class="container-fluid">
  <div class="row">
    <div class="col-12">
      <div id="idCarousel" class="carousel slide carousel-fade" data-bs-
ride="carousel">
        <div class="carousel-inner">
          <div class="carousel-item active" data-bs-interval="2000">
            
          </div>
          <div class="carousel-item" data-bs-interval="2000">
            
          </div>
          <div class="carousel-item" data-bs-interval="2000">
            
          </div>
          <div class="carousel-item" data-bs-interval="2000">
            
          </div>
        </div>
      </div>
    </div>
  </div>
</div><!-- Carousel -->
```

13. No arquivo “index.ejs” recorte o trecho do código equivalente ao footer(linhas as 177 a 211 demarcados pelo comentário <!--Rodapé-->) e substitua pela entrada:

```
<%- include('partials/footer.ejs') %>
```

14. No subdiretório “partials” crie um arquivo chamado “**footer.ejs**” e cole o trecho do código equivalente a footer que recortado:

```

<!--Rodapé-->
<footer class="footer">
  <div class="container-fluid bg-warning my-10">
    <div class="row my-10">
      <div class="col-7 d-flex">
        <div class="row mx-auto">
          <div class="col-3 col-sm-3 d-flex">
            <a href="index.html">
              
            </a>
          </div>
          <div class="col-9 col-sm-9 mx-auto py-auto my-auto">
            <h5 class="fs-5 text-center text-light text-break my-auto">
              <strong>® 2022 Ong Amigo do Pet - Diretos
reservados</strong></h5>
            </div>
          </div>
        </div>
      </div>
      <div class="col-5 d-flex me-0">
        <!--Newsletter-->
        <div class="my-2 mx-sm-auto me-md-0 text-center">
          <h5 class="display-7 fs-6 text-light my-1 text-sm-left
text-md-justify ms-md-3">Siga-nos</h5>
          <a class="btn btn-outline-light mx-sm-auto ms-md-2 me-md-
2" href="http://api.whatsapp.com/send?phone=5517997851320" target="_blank">
            <i class="fab fa-whatsapp img-fluid"></i>
          </a>
          <a class="btn btn-outline-light mx-sm-auto me-md-2"
href="#" target="_blank">
            <i class="fab fa-instagram img-fluid"></i>
          </a>
          <a class="btn btn-outline-light mx-sm-auto me-md-2"
href="#" target="_blank">
            <i class="fab fa-facebook-square img-fluid"></i>
          </a>
        </div>
      </div>
    </div>
  </div>
</footer> <!-- Fim Rodapé-->
</body>
</html>

```

15. As paritais criadas podem ser reaproveitadas em todas as views que forem compostas por estruturas similares. Para exercitar, repita o procedimento realizado em index.ejs e substitua o

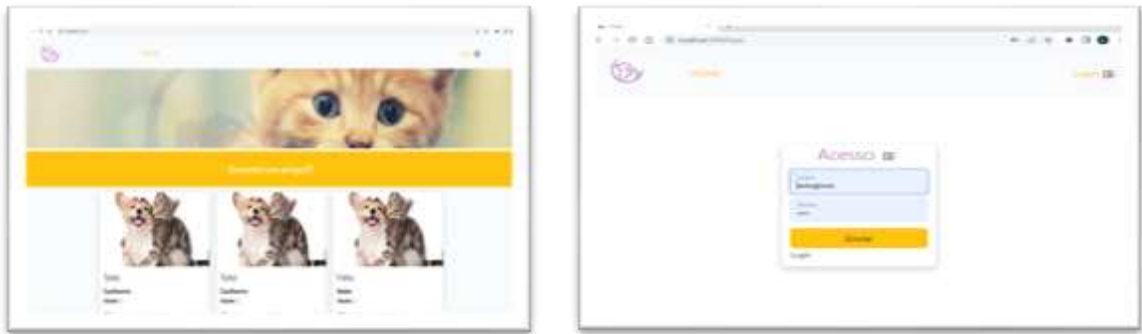
código HTML que definem cabeçalho e navbar pelas suas respectivas parciais. O arquivo **login.ejs** deve ficar com o código similar ao que segue:

```
<%- include('partials/head.ejs') %>
<title>Login</title>
</head>
<body>
  <div class="container-fluid">
    <!--Menu-->
    <%- include('partials/menu.ejs') %>
  </div>
  <hr>
  <main class="my-5">
    <div class="container login my-5 py-5">
      <div class="row justify-content-center">
        <div class="col-md-4 text-center">
          <div class="card shadow rounded">
            <div class="text-center">
              <strong class="display-6 text-center text-roxo">Acesso </img></strong>
            </div>
            <div class="card-body navddg">
              <form action="" method="">
                <div class="form-floating mb-2">
                  <input type="email" class="form-control"
id="floatingInput" autofocus placeholder="Login">
                  <label for="floatingInput">Login</label>
                </div>
                <div class="form-floating">
                  <input type="password" class="form-control"
id="floatingPassword" placeholder="Senha">
                  <label for="floatingPassword">Senha</label>
                </div>
                <input type="submit" href="#" class="w-100 py-2 mt-3 mb-1 btn
btn-lg btn-warning">Login</a>
              </form>
            </div>
          </div>
        </div>
      </div>
    </div>
  </main>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" />
</body>
</html>
```

16. No subdiretório **“/controllers/rotas”** crie um arquivo para a rota login com nome **“login.js”**, com o seguinte código:

```
module.exports =(app)=>{
  app.get(`/login`, async (req, res)=>{
    res.render('login')
  })
}
```

17. Inicie a aplicação e teste as rotas raiz “/” e “/login”, conforme imagens abaixo:



Atividade 4 – Integração entre o front-end e a API utilizando Axios

Axios é uma biblioteca JavaScript baseada em Promises que faz requisições HTTP de uma forma otimizada e é amplamente utilizada para conectar aplicações web a APIs para enviar e receber dados.

Essa atividade tem como objetivo:

- Integrar a aplicação web front-end com a API-Rest;
- Renderizar views dinâmicas exibindo informações fornecidas pela API;

Integração da aplicação front-end com API utilizando Axios

O Axios pode transformar automaticamente a resposta em diversos formatos, como JSON, XML ou texto simples, tornando o processo de lidar com diferentes tipos de dados mais simples. Nesta atividade vamos ver um exemplo da criação de uma requisição HTTP do tipo GET. Será necessário baixar a biblioteca via npm e importá-la para o módulo da rota.

1. Para iniciar, replique a pasta do exercício anterior e renomeie para “**at04-ConsumindoDadosAPI**”, e abra com o VSCode. Desta forma vamos aproveitar a estrutura de diretório, dependências e código da atividade anterior, ficando faltando apenas a dependência da biblioteca axios. Realize o procedimento de instalação do módulo do axios: comando no console:

✓ **npm install axios**

2. Edite o arquivo index.js para realizar uma requisição a API para obter os registros de pets passar a resposta para ser a view exibir a resposta. Altere o código conforme indicado em vermelho:

```

const express = require('express')
const app = express()
const cors = require('cors')
app.use(cors())
const axios = require('axios')
const consign=require('consign')
var porta = '3000'

//Variavel global que define a url do servidor da API
global.urlServer = 'http://localhost:3200'

// Configura o Express p/ usar o EJS como View engine
app.set('view engine','ejs')

app.use(express.urlencoded({extended:true}))
app.use(express.json())

//Define diretório para arquivos estaticos(css, imagens, js(front-end))
app.use(express.static('public'))

consign()
  .include('./controllers/rotas')
  .into(app)

app.get(`/`, async (req, res)=>{
  const rota = 'pets'
  let uri=`${urlServer}/${rota}`
  let dados = await axios.get(uri)
  dados = [...dados.data]
  res.render('index', {dados:dados})
})

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))

```

3. Precisamos ajustar o arquivo “index.ejs” para exibir os cartões contendo os dados recebidos pela API e não os dados estáticos de exemplo. Para isso precisamos substituir o código HTML por um forEatch. Edite o arquivo para que fique com o código a seguir:

```

<%- include('partials/head.ejs') %>
  <title>Home</title>
</head>
<body>
  <div class="container-fluid"></div><!-- main container-fluid -->
    <!-- Menu -->
    <%- include('partials/menu.ejs') %>

```



```

<!-- Carousel -->
<%- include('partials/carousel.ejs') %>

<!-- Inicio do conteúdo -->
<div class="container-fluid bg-light mt-2">
  <div class="py-2 mx-auto bg-warning">
    <hr>
    <h2 class="display-2 text-align-justify text-center text-break text-light">Encontre um amigo!!!</h2>
    <hr>
  </div>

  <div class="pt-3 container"><!--Div de Itens - Conteúdo-->
    <div class="row"> <!-- Linha de conteúdo -->

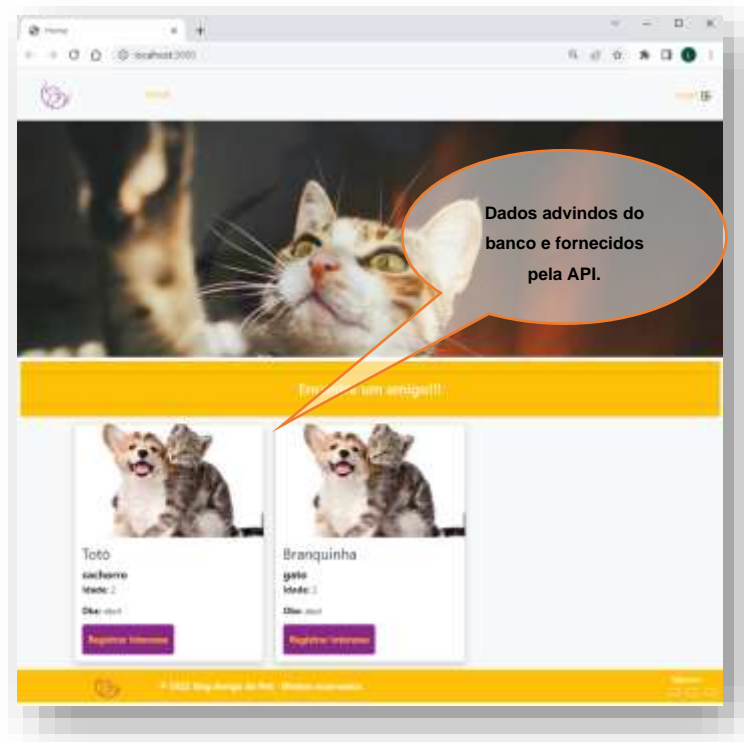
      <% dados.forEach((pet)=>{ %>
        <!-- 1º item de conteúdo -->
        <div class="col-md-4 col-sm-12 mb-3">
          <div class="card shadow rounded">
            
            <div class="card-body">
              <h3 class="card-title display-5 fs-2"><strong>
                <%= pet.nome %></strong></h3>
              <strong class="card-text text-break fs-4">
                <%= pet.especie %></strong>
              <p class="card-text text-break fs-5">
                <strong>Idade: </strong><small>
                  <%= pet.idade %></small></p>
              <p class="card-text fs-5">
                <strong>Obs: </strong><small>
                  <%= pet.obs %></small></p>
              <a href="#" class="btn btn-rosa py-3 fs-5">
                <strong>Registrar Interesse</strong></a>
            </div>
          </div>
        </div>
        <!-- Fim 1º item de conteúdo -->
      <% }) %>

    </div> <!-- Fim linha de conteúdo -->
  </div><!-- Fim itens de conteúdo -->
</div> <!-- Fim Container conteúdo -->
</div><!-- Fim do main container-fluid -->

<!-- Rodapé-->
<%- include('partials/footer.ejs') %>

```

4. Para testar o código é necessário que o serviço da API esteja em execução e sem nenhum problema. Para isso, inicialize à última versão da API no capítulo 6 - acesse o diretório "`\Cap06\at03-APIRest_Amigo_do_Pet-V_1_1`" no CMD e emita o comando de inicialização (npm start).
5. Agora é possível testar nossa aplicação que irá consumir os dados fornecidos pela API. No console do VSCode emita o comando "npm start" e verifique que a página index está exibindo os dados fornecidos pela aplicação que estavam armazenados no banco de dados.



Exercícios extras

Esta atividade tem o objetivo ampliar o conhecimento em fazer requisições HTTP assíncronas.

1. Pesquise as diferenças sobre **Fetch API** e **Axios**.
2. Pesquise e implemente a rota padrão "/" realizando a requisição GET utilizando o método nativo `fetch()`.

CAPÍTULO 8 – Autenticação, Área Exclusiva, Cadastro e Transações

Neste capítulo, continuaremos o desenvolvimento da aplicação, focando em recursos avançados, como autenticação, áreas exclusivas para usuários autenticados, cadastro de informações e a realização de transações na API.

- ✓ Neste capítulo, você aprenderá sobre:
- ✓ Implementação de autenticação em uma aplicação Node.js.
- ✓ Criação de áreas exclusivas acessíveis somente para usuários autenticados.
- ✓ Desenvolvimento de funcionalidades de cadastro de informações.
- ✓ Realização de transações utilizando a API.

Atividade 1 – Autenticação utilizando bearer Token JWT

Essa atividade tem como objetivo:

- Realizar requisições HTTP com verbo POST usando a biblioteca axios;
- Gerar um token de autenticação JWT e armazená-lo em um cookie e no session storage do navegador;
- Enviar o token para acessar rotas protegidas da API.

Comandos:

- **npm init -y | npm install** express ejs cors cookie-parser

Aplicação Login e Autenticação

Vamos ver a criação de uma aplicação que envia a combinação de usuário e senha para a API e recebe e armazena o token ou trata um erro exibindo uma mensagem.

1. Dentro do diretório "**Cap08**", crie um subdiretório com o nome "**at01-Login_e_Autenticação**" e abra com o VSCode.;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:
 - ✓ **npm init -y**
 - ✓ **npm install** express ejs consign axios cors cookie-parserInsira também a entrada "**start**": "nodemon index.js" (arquivo package.json vide exemplos anteriores).
3. Crie o arquivo "**index.js**" e insira o seguinte código:

```

const express = require('express')
const app = express()
const cors = require('cors')
app.use(cors())
const cookieParser = require('cookie-parser')
app.use(cookieParser())
const consign=require('consign')
const requests = require('./controllers/requests')
var porta = '3000'
global.urlServer = 'http://localhost:3200'
app.use(express.urlencoded({extended:true}))
app.use(express.json())

// Configura o Express p/ usar o EJS como View engine
app.set('view engine','ejs')

//Define diretório para arquivos estaticos(css, imagens, js(front-end))
app.use(express.static('public'))

consign()
  .include('/controllers/rotas')
  .into(app)

app.get(`/`, async (req, res)=>{
  try {
    dados = await requests.obterPets(`pets`)
    res.status(200)
    res.render('index', {dados:dados})
  } catch (error) {
    console.log(error)
    res.status(400)
    res.render('index')
  }
})

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))

```

4. Crie um diretório “**controllers**” e dentro dele crie um arquivo chamado “**requests.js**”, onde vamos escrever e exportar método especializados para realizar requisições HTTP assíncronas utilizando os verbos: GET, POST, PUT e DELETE.
5. O arquivo “**requests.js**” possui métodos para realizar operações com a API, especialmente relacionados à autenticação e manipulação de cookies. Insira o seguinte código:

```

const axios = require('axios')
module.exports={
  obterPets: async (rota)=>{

    let uri = `${urlServer}/${rota}`
    let dados = await axios.get(uri)
    return [...dados.data]
  },
  realizarLogin: async (req, rota)=>{
    let uri = `${urlServer}/${rota}`
    let resp = await axios.post(uri, {...req.body})
    return resp.data
  },
  gravarCookie:(res, token)=>{
    res.cookie('Authorization', token, {
      //httpOnly: true,
      secure: true,
      sameSite: 'strict',
      //expires: new Date(Date.now() + 60 * 60 * 1000), //+1 hora com
data/hra definida
      maxAge: 60 * 60 * 1000 //+1 hora em milesegundos
    })
  },
  excluirCookie:(res)=>{
    //res.cookie('Authorization', 'undefined', {maxAge: 60 * 60 *
100000})
    res.cookie('Authorization', 'undefined', { expires: new Date(0) })
  }
}

```

6. Agora vamos escrever as rotas que irão utilizar os métodos. Dentro do diretório **“controllers”** crie no subdiretório **“rotas”** um arquivo chamado **“login.js”**, neste arquivo vamos implementar o código responsável por carregar o template de login, a rota post responsável por fazer a autenticação com a API e a rota de logoff. Insira o seguinte código no arquivo:

```

const requests = require('../requests')
module.exports =(app)=>{
  app.get(`/login`, async (req,
res)=>res.render('login',{dados:{message:false}}))
  app.post('/login', async (req, res)=>{
    try {
      let dados = await requests.realizarLogin(req, 'login')
      if (dados.autenticado){
        requests.gravarCookie(res, dados.token)
        res.render('area_exclusiva', {dados})
      } else{
        dados.status=401
        res.render('login', {dados})
      }
    } catch (error) {
      let dados={message:"Login Inválido!", status:401, erro:error}
      res.render('login', {dados})
    }
  })
  app.get('/logoff', async (req, res) => {
    try {
      console.log('/logoff')
      requests.excluirCookie(res)
      res.render('index')
    } catch (error) {
      console.log('Erro:', error)
    }
  })
}

```

7. Crie um segundo arquivo no subdiretório “rotas” com o nome **“area-exclusiva.js”** que abrigará a rota responsável por verificar se o acesso está autenticado e carregar o template da área exclusiva. Insira o seguinte código:

```

module.exports =(app)=>{
  app.get('/area-exclusiva', (req, res)=>{
    //Resgata o token à partir do cookie
    const token = req.headers.cookie.split('=')[1];
    if (token!==undefined){
      dados.autenticado=true
      dados.token=token
      res.render('area_exclusiva', {dados})
    }
  })
}

```

- Para facilitar o desenvolvimento, copie as pastas “**public**” e “**views**” do último exercício do **capítulo 7(at04-ConsumindoDadosAPI)** para o diretório raiz do projeto.
- Agora, no diretório “**views**” crie um arquivo com o nome “**area_exclusiva.ejs**” e insira o seguinte código:

```
<%- include('partials/head.ejs') %>
<% if (typeof dados !== 'undefined') { %>
  <% if (dados.autenticado !== 'undefined' && dados.token !== 'undefined') { %>
    <script>
      //Insere o bearer token no sessionStorage
      const token = '<%= dados.token %>'
      sessionStorage.setItem('Authorization', `Bearer ${token}`)
    </script>
  <% } %>
<% } %>
<title>Área Restrita - Amigo do Pet</title>
</head>
<body>
  <div class="container-fluid">
    <!-- navbar -->
    <%- include('partials/menu.ejs') %>
  </div>
  <br>
  <!-- Formulario Login -->
  <main>
    <div class="card mt-5" id="panel-tab">
      <div class="mb-3 card-body">
        <div class="bg-light m-auto w-100 d-block">
          <ul class="nav nav-tabs" id="myTab" role="tablist">
            <li class="nav-item" role="presentation">
              <button class="nav-link active" id="perfil-tab"
data-bs-toggle="tab" data-bs-target="#perfil-pane" type="button" role="tab"
aria-controls="perfil-pane" aria-selected="true">
                <small class="fs-6">Meu Perfil</small></button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="pets-tab" data-bs-
toggle="tab" data-bs-target="#pets-pane" type="button" role="tab" aria-
controls="pets-pane" aria-selected="false">
                <small class="fs-6">Meus Pets</small></button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="interesse-tab" data-bs-
toggle="tab" data-bs-target="#interesse-pane" type="button" role="tab" aria-
controls="interesse-pane" aria-selected="false">
                <small class="fs-6">Meus Interesses</small></button>
            </li>
          </ul>
          <div class="tab-content" id="myTabContent">
```

```

        <div class="tab-pane fade show active" id="perfil-pane"
role="tabpanel" aria-labelledby="perfil-tab" tabindex="0">
            <p>Perfil</p>
        </div>
        <div class="tab-pane fade show" id="pets-pane"
role="tabpanel" aria-labelledby="pets-tab" tabindex="1">
            <p>Pets</p>
        </div>
        <div class="tab-pane fade" id="interesse-pane"
role="tabpanel" aria-labelledby="interesse-tab" tabindex="2">
            <p>Interesses</p>
        </div>
    </div>
</div>
</main>
<%- include('partials/footer.ejs') %>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.
js" integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
</body>
</html>

```

10. Para ter um menu dinâmico com opções referentes à quando se está ou não autenticado, modifique a partial “**menu.js**” (no subdiretório “views/partials”) para o código a seguir:

```

<div class="container-fluid">
    <nav class="row navbar navbar-expand-lg navbar-light bg-light fixed-top
clearfix" role="navigation">
        <div class="col-sm-12 col-md-12 col-lg-2">
            <div class="row">
                <div class="col-sm-6 col-md-10 col-lg-12 mx-auto" style="max-
width: 200px;">
                    <a class="navbar-brand" href="/">
                        
                    </a>
                </div>
                <div class="col-sm-2 col-md-2 col-lg-0 my-auto py-2"
style="max-width: 100px;">
                    <div class="mx-auto">
                        <button id="btnToggleMenu" class="navbar-toggler"
type="button"
data-bs-toggle="collapse" data-bs-
target="#menuColapse"

```



```

        aria-controls="menuColapse" aria-expanded="true"
aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
</div>
</div>
</div>
</div>
<div class="col-sm-12 col-md-12 col-lg-10">
    <div class="container">
        <div class="collapse navbar-collapse" id="menuColapse">
            <ul id="listaMenu" class="navbar-nav me-auto mb-2 mb-lg-
0">
                <li class="nav-item">
                    <a href="/" class="nav-link clearfix fs-5 text-
warning" onclick="toggleMenu()">HOME</a>
                </li>
            </ul>
            <ul class="navbar-nav mb-2 mb-lg-0">
                <li class="nav-item">
                    <a id="menuLogin" class="nav-link fs-5 text-
warning" aria-current="page" onclick="excluiToken()"><img></a>
                </li>
            </ul>
        </div>
    </div>
</div>
</nav>
<hr>
</div>
<script>
    let menuLogin=document.getElementById('menuLogin')
    let listaMenu=document.getElementById('listaMenu')
    let logado=false
    if (sessionStorage.getItem('Authorization')){
        menuLogin.innerHTML='Sair </img>'
        logado=true
        menuLogin.href='/'
        let li = document.createElement('li')
        li.classList.add('nav-item')
        let a = document.createElement('a')
        a.classList.add('nav-link', 'clearfix', 'fs-5', 'text-warning')
        a.href='/area-exclusiva'
        a.innerHTML = 'AREA EXCLUSIVA'
        li.appendChild(a)
        listaMenu.appendChild(li)
    } else {

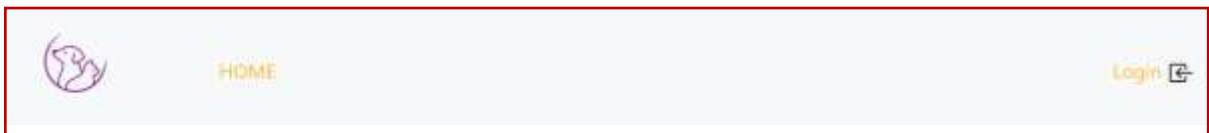
```

```

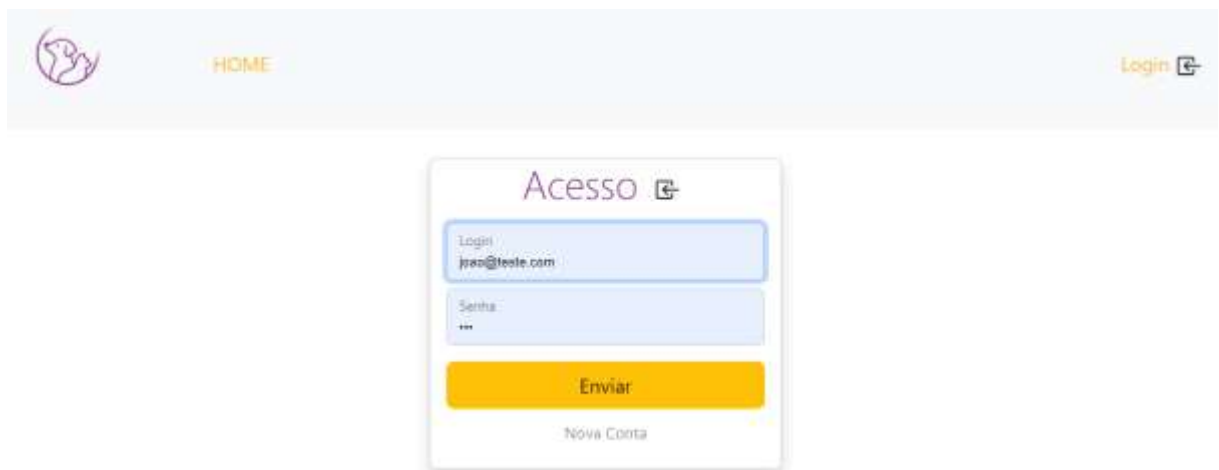
        menuLogin.innerHTML='Login'
        menuLogin.href='/login'
    }
    var excluirtoken={()=>{
        if (logado){
            sessionStorage.removeItem('Authorization')
            document.cookie = `Authorization=; maxAge:-1; path=/;`
            logado=false
        }
    }
}
</script>

```

11. Para podermos testar este código é necessário termos o **serviço da nossa API inicializado** e “escutando” na porta 3200. Portanto, navegue até o diretório da última versão da nossa API(último exercício do **capítulo 6 - at03-APIRest_Amigo_do_Pet-V_1_1**) e inicialize o serviço com o comando **“npm start”**.
12. Com o serviço da API rodando, agora podemos testar nosso código. Execute o comando **“npm start”** também no exercício atual. Certifique-se que tenha um usuário cadastrado no banco de dados e se lembre a senha de cadastro. Caso não se lembre, pode utilizar o postman para cadastrar um novo usuário.
13. Com as credenciais de um e-mail e senha válidos acesse a rota **/login** e realize testes de autenticação.
14. Poderemos observar a seguinte exibição nos templates:



Menu sem autenticação



Acesso ao menu que aponta para a rota /login

Acesso

Login

joao@teste.com

Senha

...

Enviar

Conta de email inválida.

Nova Conta

Acesso

Login

joao@teste.com

Senha

...

Enviar

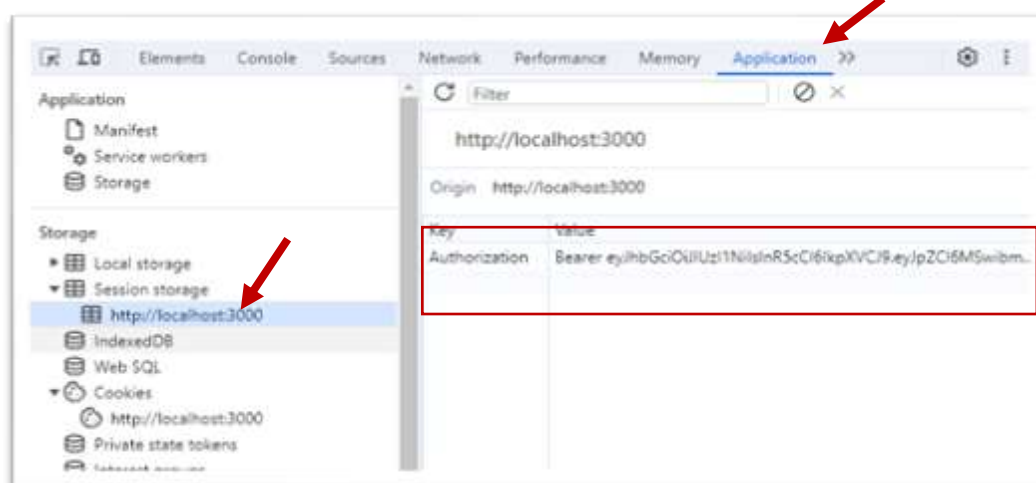
Senha inválida.

Nova Conta

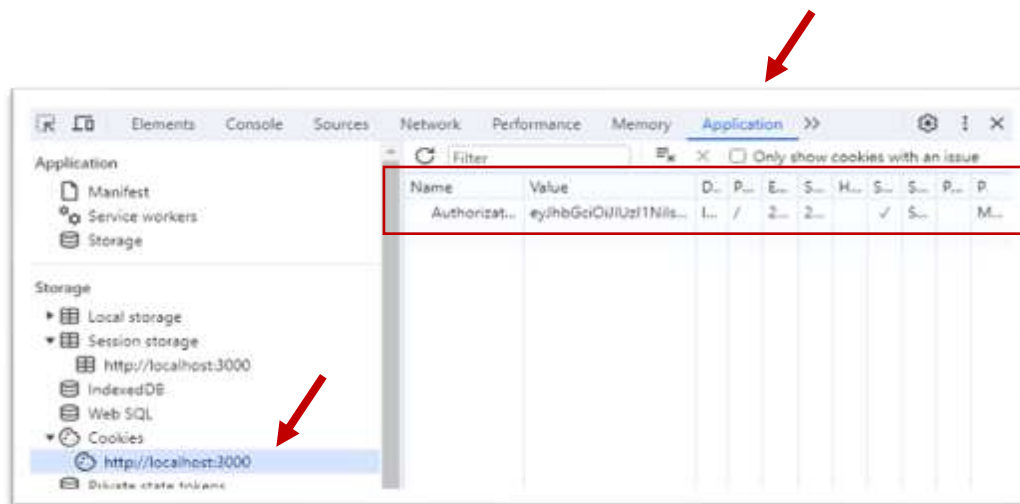
Tratamento de erro com dados de e-mail e/ou senha inconsistentes



Menu autenticado e página da área exclusiva após acertar os dados de login.



Observe o token em session storage(ferramentas de desenvolvedor no navegador com a tecla F12).



Token na área da cookie (abra as ferramentas de desenvolvedor no navegador utilizando a tecla F12).

Atividade 2 – Aperfeiçoamento da API para proteger rotas e atender as demandas de cadastro e transações

Essa atividade tem como objetivo:

- Preparar a API para tratar requisições em rotas protegidas pelo token;
- Aperfeiçoar a API para atender requisições de dados e novos cadastros;

Aperfeiçoamento da API

1. Copie para o diretório "Cap08" o diretório da última versão da nossa API que faz parte do capítulo6 at03-APIRest_Amigo_do_Pet-V_1_1, renomeie a cópia para "at02-APIRest_Amigo_do_Pet-V_1_2" e abra com o VSCode;
2. Refatore o código do "index.js" para o seguinte código:

```
const express = require('express')
const consign=require('consign')
const app = express()
const cors = require('cors')
app.use(cors({
  origin: '*', // Permite todas as origens
  credentials: true // Se necessário para permitir o envio de cookies
}))
var porta = '3200'
app.use(express.urlencoded({extended:false}))
app.use(express.json())
app.get('/', (req, res)=>res.send('API - Amigo do Pet'))
consign()
  .include('./controllers/rotas')
  .into(app)
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

3. Agora vamos iniciar o processo de **refatoração das rotas protegidas**, as linhas alteradas haverá a **indicação** de um **comentário “//alterar”**. Apenas as rotas POST “/login” e “/usuários” não devem ser **protegidas pelo middleware “auth.validarToken,”** (destacado pelo retângulo vermelho) que verifica a autenticidade do token e retorna os dados do usuário autenticado. Vamos iniciar com o código do arquivo “/controllers/rotas/ usuário.js”:

```
const model = new require('../models/usuario')
const auth = require('../auth')
const validacao = require('../validacao')
const rota = 'usuarios'
module.exports = (app) => {
  app.get(`/${rota}`, auth.validarToken, async (req, res) => { //rota
    protegida
    try {
      let id = req.usuarioAtual.id //Alterar
      let dados = await model.findByPk(id)
      delete dados.dataValues.senha
      res.json(dados).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.post(`/${rota}`, async (req, res) => {
    try {
      let dados = req.body
      let dadosLogin = await validacao.validarCadastro(dados, model)
      if (dadosLogin.validacao) {
        dados.senha = await auth.criptografarSenha(dados.senha)
        let respBd = await model.create(dados)
        delete respBd.dataValues.senha
        res.json(respBd).status(201)
      } else {
        res.json(dadosLogin).status(200)
      }
    } catch (error) {
      res.json(error).status(422)
    }
  })
  app.put(`/${rota}`, auth.validarToken, async (req, res) => {
    try {
      let id = req.usuarioAtual.id //alterar
      let {nome, cpf, telefone, whatsapp} = req.body
      //Atualizar dados de login exigem regras específicas de validação
      let dados = {nome: nome, cpf: cpf, telefone: telefone,
        whatsapp: whatsapp}
      let respBd = await model.update(dados, {where: {id: id}})
      res.json(respBd).status(200)
    }
  })
}
```

```

    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.delete(`/${rota}`, auth.validarToken, async (req, res) => {
    try {
      let id = req.usuarioAtual.id //alterar
      //A exclusão de registro exigem regras específicas de validação
      além deste escopo
      let respBd = await model.destroy({where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
}

```

Vale observar que as rotas PUT e DELETE que utilizavam parâmetros na URL, deixou de utiliza-los e passaram a utilizar o “id” do usuários que foram extraídos do token, está é uma boa prática.

4. Agora vamos refatorar o código do arquivo do diretório “/controllers/rotas/pets.js”, o mesmo padrão será seguido para os próximos passos:

```

const model = new require('../models/pet')
const usuario = new require('../models/usuario')
const auth = require('../auth')
const rota = 'pets'
module.exports = (app)=>{
  app.get(`/${rota}/:id?`, async (req, res)=>{
    try {
      let dados = req.params.id? await
      model.findOne({where:{id:req.params.id}}) : //Alterar
      await model.findAll({include:[{model:usuario}], {raw: true,
      order:[['id', 'DESC']]})
      res.json(dados).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  //Criar esta rota
  app.get(`/${rota}`, auth.validarToken, async (req, res) => {
    try {
      let id=parseInt(req.usuarioAtual.id)
      let dados = await model.findAll({where:{usuarioId:id}}, {raw:
      true, order:[['id', 'DESC']]})
      res.json(dados).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
}

```

```

    }
  )),
  app.post(`/${rota}`, auth.validarToken, async (req, res)=>{
    try {
      let dados = req.body
      dados.usuarioId=req.usuarioAtual.id //Alterar
      let respBd = await model.create(dados)
      res.json(respBd).status(200)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.put(`/${rota}`, auth.validarToken, async (req, res) => {
    try {
      let id = req.usuarioAtual.id //Alterar
      let dados = req.body
      console.log(dados)
      let respBd = await model.update(dados, {where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
  app.delete(`/${rota}`, auth.validarToken, async (req, res) => {
    try {
      let id = req.usuarioAtual.id //Alterar
      let respBd = await model.destroy({where:{id:id}})
      res.json(respBd)
    } catch (error) {
      res.json(error).status(400)
    }
  })
}

```

5. Agora vamos refatorar o código do arquivo do diretório **“/controllers/rotas/ doacao.js”**:

```

const model = new require('../models/pet')
const usuario = new require('../models/usuario')
const auth = require('../auth')
const rota = 'pets'
module.exports = (app)=>{
  app.get(`/${rota}/:id?`, async (req, res)=>{
    try {
      let dados = req.params.id? await
model.findOne({where:{id:req.params.id}}) : //Alterar
      await model.findAll({include:[{model:usuario}], {raw: true,
order:[['id', 'DESC']]})
      res.json(dados).status(200)
    }
  })
}

```

```

        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.get(`/obter/${rota}`, auth.validarToken, async (req, res) => { //Criar
esta nova rota
        try {
            let id=parseInt(req.usuarioAtual.id) //Alterar
            let dados = await model.findAll({where:{usuarioId:id}}, {raw:
true, order:[['id','DESC']]})
            res.json(dados).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    }),
    app.post(`//${rota}`, auth.validarToken, async (req, res)=>{
        try {
            let dados = req.body
            dados.usuarioId=req.usuarioAtual.id //Adiciona o Id do usuario
autenticado
            let respBd = await model.create(dados)
            res.json(respBd).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.put(`//${rota}`, auth.validarToken, async (req, res) => {
        try {
            let id = req.usuarioAtual.id //Alterar
            let dados = req.body
            let respBd = await model.update(dados, {where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.delete(`//${rota}`, auth.validarToken, async (req, res) => {
        try {
            let id = req.usuarioAtual.id //Alterar
            let respBd = await model.destroy({where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
}

```

6. Agora vamos refatorar o código do arquivo do diretório **“/controllers/rotas/ login.js”**:


```

const model = new require('../models/usuario')
const auth = require('../auth')
const validacao = require('../validacao')

module.exports = (app) => {
  app.post(`/login`, async (req, res) => {
    try {
      let dados = req.body
      let validaLogin = await validacao.validarLogin(dados, model)
      if (validaLogin.autenticado) { //Verifica se email e senha são
        consistentes
          let {id, nome, email} = validaLogin.usuario.dataValues
          dados = {id, nome, email} //Desestruturação dos dados
          validados
          let token = await auth.gerarToken(dados) //Gera um token
          JWT
          return res.json({dados, autenticado:true,
            token:token}).status(200)
        } else {
          return res.json(validaLogin).status(401)
        }
      } catch (error) {
        return res.json(error).status(400)
      }
    }
  })
}

```

7. Após concluir as alterações, rode o código e faça testes de funcionalidades utilizando o postman para verificar possíveis inconsistências e erros de digitação.

Atividade 3 – Aperfeiçoamento do módulo requests.js

Essa atividade tem como objetivo:

- Criar métodos que realizar requisições HTTP usando a biblioteca Axios que aceitam argumentos dinâmicos que podem lidar com requisições protegidas ou não;

Aperfeiçoamento do módulo requests

1. Copie para o diretório do exercício “at01-Login_e_Autenticação” do capítulo 8, renomeie a cópia para “at03-Metodos_Especialistas_HTTP” e abra com o VSCode;
2. Refatore o código do arquivo “requests.js” que se encontra no diretório “controllers”, para o seguinte código:

```

const axios = require('axios')
module.exports={
  requisicaoGet:async (...dataReq) => {
    try {
      dataReq[0]=`${urlServer}/${dataReq[0]}`
      if (dataReq[1]!==undefined){
        let config = {
          headers: {
            'Authorization':dataReq[1] // Adiciona o cabeçalho de
            autorização com o token
          }
        }
        dataReq[1]= config
      }
      let resp = await axios.get(...dataReq)
      return resp.data
    } catch (error) {
      return {status:400, message:'A requisição não pode ser
      respondida!', erro:error}
    }
  },
  requisicaoPost:async (...dataReq) => {
    try {
      dataReq[0]=`${urlServer}/${dataReq[0]}`
      if (dataReq[2]!==undefined){
        let config = {
          headers: {
            'Authorization':dataReq[2] // Adiciona o cabeçalho de
            autorização com o token
          }
        }
        dataReq[2]= config
      }
      let resp = await axios.post(...dataReq)
      return resp.data
    } catch (error) {
      return {status:400, message:'A requisição não pode ser
      respondida!', erro:error}
    }
  },
  requisicaoPut:async (rota, dados, token) => {
    try {
      let uri=`${urlServer}/${rota}`
      let config = {
        headers: {
          'Authorization':token // Adiciona o cabeçalho de
          autorização com o token
        }
      }
    }
  }
}

```

```

        let resp = await axios.put(uri, dados, config)
        return resp.data
      } catch (error) {
        return {status:400, message:'A requisição não pode ser
respondida!', erro:error}
      }
    },
    requisicaoDelete:async (rota, token) => {
      try {
        let uri=`${urlServer}/${rota}`
        let config = {
          headers: {
            'Authorization':token // Adiciona o cabeçalho de
autorização com o token
          }
        }
        let resp = await axios.delete(uri, config)
        return resp.data
      } catch (error) {
        return {status:400, message:'A requisição não pode ser
respondida!', erro:error}
      }
    },
    gravarCookie:(res, token)=>{
      res.cookie('Authorization', token, {
        //httpOnly: true,
        secure: true,
        sameSite: 'strict',
        //expires: new Date(Date.now() + 60 * 60 * 1000), //+1 hora com
data/hra definida
        maxAge: 60 * 60 * 1000 //+1 hora em milesegundos
      })
    },
    excluirCookie:(res)=>{
      //res.cookie('Authorization', 'undefined', {maxAge: 60 * 60 * 100000})
      res.cookie('Authorization', 'undefined', { expires: new Date(0) })
    }
  }
}

```

3. Vamos precisar refatorar os arquivos “**index.js**” e o arquivo “**login.js**” que fazem uso do módulo “requests.js”. Vamos iniciar pelo “index.js” substituindo a linha a seguir:

```
dados = await requests.obterPets(`pets`)
```

por:

```
dados = await requests.requisicaoGet(`pets`)
```

4. Agora é a vez do arquivo “login.js” que se encontra no diretório “/controllers/rotas”, substitua a linha:

```
let dados = await requests.realizarLogin(req, 'login')
```

por:

```
let dados = await requests.requisicaoPost('login', req.body)
```

5. Rode o código e verifique se existe alguma inconsistência no código após a refatoração. Caso exista corrija o problema antes de prosseguir para a próxima atividade.

Atividade 4 – Cadastrar, editar e exibir registros de usuários

Essa atividade tem como objetivo:

- Criar editar e exibir registros na tabela usuários;

Criação de rotas e templates para manipular dados de usuários

1. Copie para o diretório do exercício “at03-Metodos_Especialistas_HTTP” do capítulo 8, renomeie a cópia para “at04-Cadastrar_editar_exibir_usuarios” e abra com o VSCode;
2. Vá até o template “login.ejs” na pasta “views” e altere a linha:

```
<a class="navbar navbar-nav navbar-item navbar-link" href="/cadastrar/usuario">Nova  
Conta</a>
```

Por:

```
<a class="navbar navbar-nav navbar-item navbar-link" href="/novo-usuario">Nova  
Conta</a>
```

3. Agora crie um arquivo chamado “usuario.js” no diretório “/contorllers/rotas” e insira o seguinte código:

```
const requests = require('../requests')  
module.exports =(app)=>{  
  app.get(`/novo-usuario`, async (req, res)=>res.render('novo_usuario')),  
  app.post('/novo-usuario', async (req, res)=>{  
    try {  
      let dados = await requests.requisicaoPost('usuarios', req.body)  
      res.render('confirmacao', dados)  
    } catch (error) {  
      let dados={message:"Não foi possível realizar o cadastro.",  
status:401, erro:error}  
      res.render('login', dados)  
    }  
  })  
}
```

```

    }
  )),
  app.post('/editar-perfil', async(req, res) => {
    let dados={}
    if (req.headers.cookie==='Authorization=' || req.headers.cookie===''){
      dados.autenticado=false
      dados.token=undefined
      res.render('login',{dados:{message:'Sessão expirou. Faça
login!'}})
    } else {
      const token = req.headers.cookie.split('=')[1]
      if (token!==undefined){
        dados.autenticado=true
        dados.token=token

        //Realiza a atualização na API
        let respApi = await requests.requisicaoPut('usuarios',
req.body, `Bearer ${dados.token}`)
        dados.message=respApi>0?"Dados atualizados com sucesso!":"Os
dados não puderam ser gravados."
        dados.alert=respApi>0?"success":"danger"

        //Atualizar dados e recarregar página
        dados.meusDados = await requests.requisicaoGet(`usuarios`,
`Bearer ${dados.token}`)
        dados.meusPets = await requests.requisicaoGet(`obter/pets`,
`Bearer ${dados.token}`)
        meusInteresses = await requests.requisicaoGet(`doacoes`,
`Bearer ${dados.token}`)
        dados.meusInteresses = meusInteresses.map(m=>m.pet)

        res.render('area_exclusiva', {dados})
        //res.redirect('/area-exclusiva')
      }
    }
  })
}

```

4. Agora vamos criar um formulário de cadastro para usuários. Crie um arquivo na pasta “views” com o nome de “**novo_usuario.ejs**” e insira o seguinte código:

```

<%- include('partials/head.ejs') %>

<title>Novo Usuário</title>
</head>
<body>
  <div class="container"></div><!-- main container-fluid -->
  <!--navBar-->
  <%- include('partials/menu.ejs') %>

```

```

    <!-- Inicio do conteúdo -->
    <div class="row card bg-light mt-2">
        <div class="col-12 card-body py-2 mx-auto bg-warning">
            <hr>
            <h2 class="display-2 text-align-justify text-center text-
break text-light">Novo Usuário</h2>
            <hr>
        </div>
        <div class="pt-3 container"><!--Div de Itens - Conteúdo-->
            <div class="row px-5"> <!-- Linha de conteúdo -->

                <div class="col-12 mb-3"> <!-- 1º item de conteúdo -->
                    <div class="card shadow rounded">
                        <div class="card-body">
                            <form name="formClientes"
id="formClientes" method="post" action="/novo-usuario">
                                <!-- Dados pessoais -->
                                <%- include('partials/cadastro_usuario')%>
                                <br><hr><br>

                            </div>
                            <div class="card-footer">
                                <div class="row">
                                    <div class="col-12"></div>
                                    <div class="col-2 d-flex justify-
content-end ms-auto me-0">
                                        <button class="btn btn-primary
form-control">Gravar</button>

                                    </div>
                                </div>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
<!-- Rodapé-->
<%- include('partials/footer.ejs') %>

    <script>
        var frm = document.forms
        //Tabular com Enter
        function tabularComEnter(e) {
            if (e.keyCode == 13) {
                if ((e.target.type != 'submit') ||
(e.target.tagName != 'BUTTON')) {
                    e.preventDefault();
                }
            }
        }
    </script>

```

```

        let foco = document.activeElement;
        for (let i in frm) {
            for (let j in frm[i]) {
                if (frm[i][j] === foco) {
                    frm[i][parseInt(j) + 1].focus()
                    break;
                }
            }
        }
    }
}

function formatarCPF(cpf) {
    //mais formatos de validação
    https://aurelio.net/regex/html5/pattern.html
    const elementoAlvo = cpf
    const cpfAtual = cpf.value
    let cpfAtualizado
    // xxx.xxx.xxx-xx
    cpfAtualizado = cpfAtual.replace(
        /(\d{3})(\d{3})(\d{3})(\d{2})/,
        function (regex, argumento1, argumento2,
argumento3, argumento4) {
            return argumento1 + '.' + argumento2 + '.' +
argumento3 + '-' + argumento4
        }
    );
    elementoAlvo.value = cpfAtualizado;
}

function copiarTelefone(){
    console.log(frm.telefone)
    console.log(frm.whatsapp)
    frm[0].whatsapp.value = frm[0].telefone.value
    frm[0].whatsapp.focus()
}
</script>
</body>
</html>

```

5. Vamos criar a partial com os campos do formulário para cadastrar novo usuário. Crie um arquivo com o nome **"cadastro_usuario"** no diretório **"views/partial"** com o seguinte código:

```

<div class="row align-items-center">
  <div class="col-5 ms-5 my-3 ps-5">
    <label for="nome" class="form-label">Nome</label>
    <input type="text" id="nome" name="nome" required="required"
      minlength="4" placeholder="Ex.: João da Silva" autofocus

```

```

        class="form-control" />
    </div>

    <div class="me-5 col-3">
        <label for="cpf" class="form-label">CPF</label>
        <!--Forçar usuario digitar no formato pattern="\d{3}\.\d{3}\.\d{3}-\d{2} somente números [0-9]{11}"-->
        <input type="text" id="cpf" name="cpf" required="required"
            pattern="\d{3}\.\d{3}\.\d{3}-?\d{2}" placeholder="Somente
Números"
            class="form-control" onblur="formatarCPF(this)" />
    </div>

    <!--pattern="pattern="[0-9]{2}\/[0-9]{2}\/[0-9]{4}$" min="2012-01-01"
max="2014-02-18"-->
    <div class="my-5 col-4 ms-5 my-3 ps-5">
        <label class="fom-label" for="telefone">Celular</label>
        <input type="text" id="telefone" name="telefone" required="required"
class="form-control" />
    </div>
    <div class="me-5 my-5 col-4">
        <label class="fom-label" for="whatsapp">Whatsapp</label>
        <input type="text" id="whatsapp" name="whatsapp" class="form-control"
onfocus="copiarTelefone()" />
    </div>
    <br><hr><br>
    <div class="row mx-5 px-5">
        <div class="mx-5 col-8 px-5">
            <label class="fom-label" for="email">E-mail</label>
            <input type="email" id="email" name="email" class="form-control"
required="required"/>
        </div>
    </div>

    <div class="row mx-5 px-5">
        <div class="col-4 ms-5 my-3 ps-5">
            <label class="fom-label" for="senha">Senha</label>
            <input type="password" id="senha" name="senha" class="form-
control" required="required"/>
        </div>
        <div class="col-4 me-5 my-3 pe-5">
            <label class="fom-label" for="confirmacao">Confirmação da
senha</label>
            <input type="password" id="confirmacao" name="confirmacao"
class="form-control" required="required"/>
        </div>
    </div>
</div>

```


6. Agora vamos criar um template para confirmar o cadastro para usuários. Crie um arquivo na pasta "views" com o nome de "confirmacao.ejs" e insira o seguinte código:

```
<%- include('partials/head.ejs') %>

<title>Registro</title>
</head>
<body>
  <div class="container-fluid mb-5">
    <!-- navbar -->
    <%- include('partials/menu.ejs') %>
  </div>
  <hr>
  <!-- Formulario Login -->
  <main>
    <div class="container login my-5 py-5">
      <div class="row justify-content-center">
        <div class="col-4 alert alert-success" role="alert">
          <div class="card shadow rounded">
            <div class="text-center">
              <strong class="fs-2 text-center text-roxo">Registro Efetuado!</strong>
            </div>
            <div class="card-body navddg">
              <h2 class="text-success fs-4 badge text-wrap">
                Registro realizado com sucesso!</h2>
              <a class="navbar navbar-nav nav-item nav-link text-info" href="/login"><strong class="fs-5">Login</strong></a>
            </div>
          </div>
        </div>
      </div>
      <div class="alert alert-danger" role="alert">
        <%= dados.message %>
      </div>
    <div class="text-center">
      <strong class="fs-5">Login</strong>
    </div>
  </main>

  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
</body>
</html>
```

7. Neste momento já é possível realizar o cadastro de um novo usuário e realizar login na aplicação. Pode-se realizar este teste.
8. Após o teste podemos perceber a que a área exclusiva foi carregada, porém, nenhum dado foi carregado. Vamos agora criar a exibição da aba “meus dados” da área exclusiva. Para isso crie um arquivo no diretório “/views/partial” com o nome “perfil.ejs” e insira o seguinte código:

```
<div class="row d-flex justify-content-center align-items-center vh-80 pb-1
mb-1 ">
  <div class="col-md-9 col-sm-12 mx-5 px-5 py-5 mb-3">
    <div class="card shadow rounded px-5 mx-5">
      <div class="text-center mt-5 px-5">
        <% if (typeof dados !== 'undefined' && dados.message ){%>
          <div class="alert alert-<%= dados.alert %>" role="alert">
            <%= dados.message %>
          </div>
        <% } %>
        <strong class="display-6 text-center text-roxo py-5"> Meu
Perfil
          <button onclick="habilitarForm()" class="btn">
            
          </button>
        </strong>
      </div>
      <hr>
      <div class="card-body px-5 py-3">
        <div class="justify-content-center mx-5">
          <form method="post" action="/editar-perfil" >
            <div class="row align-items-center">
              <div class="col-10 ms-5 my-3">
                <label for="nome" class="form-
label">Nome</label>
                <input type="text" id="nome" name="nome"
required="required" minlength="4" class="form-control" value="<%=
dados.meusDados.nome %>" readonly/>
              </div>
              <div class="ms-5 col-10">
                <label for="cpf" class="form-
label">CPF</label>
                <!--Forçar usuario digitar no formato
pattern="\d{3}\.\d{3}\.\d{3}-\d{2} somente números [0-9]{11}"-->
                <input type="text" id="cpf" name="cpf"
required="required" pattern="\d{3}\.\d{3}\.\d{3}-\d{2}"
class="form-control" value="<%=
dados.meusDados.cpf %>" readonly/>
              </div>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```



```

    frm.whatsapp.readOnly=edicao
    frm.nome.focus()
    frm.btnGravar.classList.add('d-flex')
    frm.btnGravar.classList.toggle('d-none')
  }
</script>

```

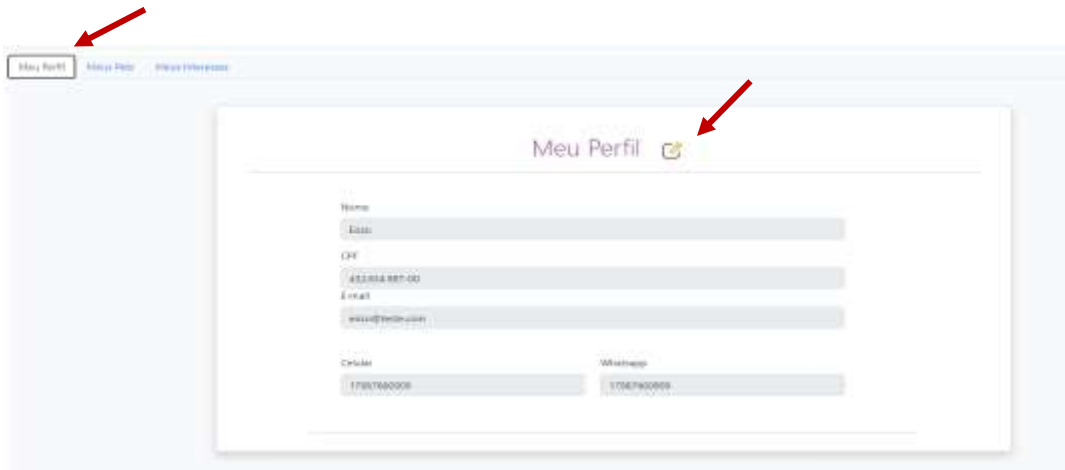
9. Refatore o arquivo **“area-exclusiva.js”** do diretório **“/controllers/rotas”** para o seguinte código:

```

const requests = require('../requests')
module.exports =(app)=>{
  app.get('/area-exclusiva', async (req, res)=>{
    try {
      let dados={}
      if (req.headers.cookie==='Authorization=' || req.headers.cookie===''){
        dados.autenticado=false
        dados.token=undefined
        res.render('login',{dados:{message:'Sessão expirou. Faça login!'}})
      } else {
        const token = req.headers.cookie.split('=')[1]
        if (token!==undefined){
          dados.autenticado=true
          dados.token=token
          dados.meusDados = await requests.requisicaoGet(`usuarios`,
`Bearer ${token}`)
          dados.meusPets = await requests.requisicaoGet(`obter/pets`,
`Bearer ${token}`)
          dados.meusInteresses = await requests.requisicaoGet(`doacoes`,
`Bearer ${token}`)
          dados.meusInteresses = await
requests.requisicaoGet(`doacoes`, `Bearer ${token}`)
          meusInteresses = await requests.requisicaoGet(`doacoes`,
`Bearer ${token}`)
          dados.meusInteresses = meusInteresses.map(m=>m.pet)
          res.render('area_exclusiva', {dados})
        }
      }
    } catch (error) {
      res.render('login',{dados:{message:'Sessão expirou. Faça login!'}})
    }
  })
}

```

10. A partir de agora os dados do usuário serão exibidos na área exclusiva e será possível editar o registro.



Tela Meu Perfil da área exclusiva.

Atividade 5 – Realizar transações de registro interesse nos Pets

Essa atividade tem como objetivo:

- Registrar interesse nos Pets;

Criação de rota, modal e exibição de registros de interesse nos Pets

1. Copie para o diretório do exercício “at04-Cadastrar_editar_exibir_usuarios” do capítulo 8, renomeie a cópia para “at05-Registro_de_interesse” e abra com o VSCode;
2. No arquivo “area-exclusiva.js” do diretório “/controllers/rotas” acrescente a seguinte rota em “module.exports”:

```
app.post("/interesse", async (req, res)=>{
  let dados = {}
  if (req.headers.cookie==='Authorization=' || req.headers.cookie===''){
    dados.autenticado=false
    dados.token=undefined
    res.render('login',{dados:{message:'Sessão expirou. Faça login!'}})
  } else {
    const token = req.headers.cookie.split('=')[1]
    if (token!==undefined){
      dados = req.body
      dados.status = 'Andamento'
      let resp = await requests.requisicaoPost(`cadastrar/doacoes`,
dados, `Bearer ${token}`)
      dados.message=resp.message
      dados.autenticado=true
      dados.token=token
      res.json(resp)
    }
  }
})
```

3. No diretório de partials crie um arquivo com o nome “modal_interesse.ejs” e insira o código:

```

<!-- Modal -->
<div class="modal" id="modalInteresse">
  <div class="modal-dialog">
    <div class="modal-content">

      <!-- Cabeçalho do Modal -->
      <div class="modal-header">
        <h4 class="modal-title">Registrar Interesse</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
      </div>

      <!-- Corpo do Modal -->
      <div class="modal-body">
        <form id="frmInteresse">
          <div class="row">
            <div class="col-4">
              <label for="data" class="form-label">Data</label>
              <input type="date" class="form-control" name="data" readonly>
            </div>
          </div>
          <br>
          <h3>Dados do Pet:</h3>

          <div class="mb-3 row">
            <input type="text" name="idPet" hidden readonly>
            <div class="col-md-4 col-sm-12">
              <label for="nomePet" class="form-label">Nome do Pet</label>
              <input type="text" class="form-control" name="nomePet"
readonly>
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="idade" class="form-label">Idade</label>
              <input type="text" class="form-control" name="idade" readonly>
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="raca" class="form-label">raca</label>
              <input type="text" class="form-control" name="raca" readonly>
            </div>
          </div>
          <br>
          <h3>Dados do Usuario:</h3>
          <hr>
          <div class="mb-3 row">
            <input type="text" name="idUsuario" hidden readonly>
            <div class="col-md-4 col-sm-12">
              <label for="nomeUsuario" class="form-label">Nome</label>
              <input type="text" class="form-control" name="nomeUsuario"
readonly>

```

```

        </div>
        <div class="col-md-4 col-sm-6">
            <label for="telefone" class="form-label">Telefone</label>
            <input type="text" class="form-control" name="telefone"
readonly>
        </div>
        <div class="col-md-4 col-sm-6">
            <label for="whatsapp" class="form-label">Whatsapp</label>
            <input type="text" class="form-control" name="whatsapp"
readonly>
        </div>
    </div>
</div>
</form>
<br>
<p class="fst-normal">
    Declaro ao confirmar este formulário que é de meu interesse
    adotar o Pet acima descrito e por isso
        <span class="fw-bolder"> concordo em compartilhar </span> os
    seguintes dados pessoais:
        <span class="fw-bolder"> Nome, Telefone, WahatsApp </span>

    </p>
    <div class="form-check">
        <input class="form-check-input" type="checkbox" value=""
id="checkDeAcordo">
        <label class="form-check-label" for="flexCheckDefault">
            Esou ciênte e concordo com o encaminhamento dos meus dados.
        </label>
    </div>
    <hr>
    <!-- Botão de envio -->
    <button type="button" class="btn btn-primary"
onclick="submeterDados()">Confirmar Interesse</button>
    </div>
</div>
</div>
</div>
</div>
<script>
    // Instanciar um objeto Modal
    let divInteresse = document.getElementById('modalInteresse')
    let modalInteresse = new bootstrap.Modal(divInteresse)
    let frmInteresse=document.getElementById('frmInteresse')
    let checkDeAcordo = document.getElementById('checkDeAcordo')
    async function abrirModalInteresse(id) {
        // Abrir o modal
        if ( logado){
            let url = `${urlServidor}/pets/${id}`
            let pet = await axios.get(url)
            pet = pet.data

```

```

url = `${urlServidor}/usuarios`
let config = {
    headers: {
        'Authorization':sessionStorage.getItem('Authorization')
// Adiciona o cabeçalho de autorização com o token
    }
}

let usuario = await axios.get(url, config)
usuario = usuario.data
carregarInteresseFrm(pet, usuario)
modalInteresse.show()
} else {
    window.location.href = "/login";
}
}

function carregarInteresseFrm(pet, usuario){
    frmInteresse.data.value = new Date().toISOString().split('T')[0]
    checkDeAcordo.checked=false
    frmInteresse.idPet.value = pet.id
    frmInteresse.nomePet.value = pet.nome
    frmInteresse.idade.value = pet.idade
    frmInteresse.raca.value = pet.raca
    frmInteresse.idUsuario.value = usuario.id
    frmInteresse.nomeUsuario.value = usuario.nome
    frmInteresse.telefone.value = usuario.telefone
    frmInteresse.whatsapp.value = usuario.whatsapp
}

async function submeterDados(){

    if (checkDeAcordo.checked) {
        let dados = {petId:frmInteresse.idPet.value,
usuarioId:frmInteresse.idUsuario.value}
        let resp = await axios.post('/interesse', dados)
        fecharModalInteresse()
        alert(resp.data.message)
    } else{
        alert('Para confirmar seu interesse é necessário que confirme a
ciência do compartilhamento de dados.')
    }
}

function fecharModalInteresse() {
    // Fechar o modal
    modalInteresse.hide();
}

function postInteresse() {
    fecharModalInteresse();
}

</script>

```


4. Acrescente a linha de importação da partial anterior no arquivo do template “index.ejs” no diretório “views”, conforme orientações a seguir:

```
<%- include('partials/modal_interesse.ejs') %> <!--Inserir apenas esta
linha(próximo da linha 57), as demais são referência.-->
    <div class="pt-3 container"><!--Div de Itens - Conteúdo-->
        <div class="row"> <!-- Linha de conteúdo -->

                <% dados.forEach((pet)=>{ %>
```

5. Agora vamos exibir os dados de interesse na área exclusiva, para isso crie outro arquivo no diretório de partial como nome “meus_interesses.ejs” e insira o seguinte código:

```
<hr>
<div class="row"> <!-- Linha de conteúdo -->
<strong class="display-6 text-center text-roxo py-5"> Meus
Interesses</strong>
<% dados.meusInteresses.forEach((pet)=>{ %>
<div class="col-md-4 col-sm-12 mb-3"> <!-- 1º item de conteúdo -->
<div class="card shadow rounded">

<div class="card-body">
<h3 class="card-title display-5 fs-2"><strong><%= pet.nome %></strong></h3>
<strong class="card-text text-break fs-4"><%= pet.especie %></strong>
<p class="card-text text-break fs-5"><strong>Idade: </strong><small><%=
pet.idade %></small></p>
<p class="card-text fs-5"><strong> Obs: </strong><small><%= pet.obs
%></small></p>
<a href="#" class="btn btn-roxo py-3 fs-5"><strong>Excluir</strong></a>
</div>
</div>
</div> <!--Fim 1º item de conteúdo -->
<% }) %>
</div> <!-- pets -->
```

6. Insira a linha de importação da partial anterior no arquivo “area-exclusiva.ejs” no diretório “views”, conforme orientação a seguir:

```
<div class="tab-pane fade show" id="interesse-pane" role="tabpanel" aria-
labelledby="interesse-tab" tabindex="2">
    <!--Interesses-->
        <%- include('partials/meus_interesses.ejs')%>
    <!--Interesses-->
</div>
```

7. A partir de agora já podem ser inseridos registros de interesse nos pets e serem exibidos na área exclusiva na aba meus interesses.

Atividade 6 – Cadastrar, exibir pets e lista de interessados

Essa atividade tem como objetivo:

- Cadastrar, exibir pets e lista de interessados;

Criação de rota, modal e exibição de registros de interesse nos Pets

1. Copie para o diretório do exercício “at05-Registro_de_interesse” do capítulo 8, renomeie a cópia para “at06-Cadastrar_exibir_Pets” e abra com o VSCode;
2. Vamos iniciar criando uma lista de “meus pets” na área exclusiva. Para isso cria no diretório de partials um arquivo com o nome “meus_pets.ejs” e insira o seguinte código:

```
<strong class="display-6 text-center text-roxo py-5"> Meus Pets
  <button onclick="abrirModalAddPts()" class="btn">
    </img>
  </button>
</strong>
</div>
<hr>
<div class="row"> <!-- Linha de conteúdo -->
  <% dados.meusPets.forEach((pet)=>{ %>
    <div class="col-md-4 col-sm-12 mb-3"> <!-- 1º item de conteúdo -->
      <div class="card shadow rounded">
        
        <div class="card-body">
          <h3 class="card-title display-5 fs-2"><strong><%= pet.nome
%></strong></h3>
          <strong class="card-text text-break fs-4"><%= pet.especie %></strong>
          <p class="card-text text-break fs-5"><strong>Idade:
</strong><small><%= pet.idade %></small></p>
          <p class="card-text fs-5"><strong>Obs:
</strong><small><%= pet.obs %></small></p>
          <button class="btn btn-success py-3 fs-5"
onclick="abrirModalInteressados(this.value)" value="<%= pet.id
%>"><strong>Interessados</strong></button>
          <button class="btn btn-roxo py-3 fs-
5"><strong>Excluir</strong></button>
        </div>
      </div>
    </div> <!--Fim 1º item de conteúdo -->
  <% }) %>
</div> <!-- pets -->
```

3. Inclua a partil anterior no arquivo “area-exclusiva.ejs”, conforme orientação abaixo:

```
<!--pets-->
<div class="pt-3 container"><!--Div de Itens - Conteúdo-->
  <%- include('partials/meus_pets.ejs')%>
</div><!-- Fim da tab -->
```

4. Após isso os pets já podem ser exibidos na página de área exclusiva. Vamos agora avançar e construir o template para cadastrar pets. Ainda no diretório de partial crie um arquivo com o nome “**modal_addPet.ejs**” e insira o seguinte código:

```
<!-- Modal -->
<div class="modal" id="modalFrmAdicionarPets">
  <div class="modal-dialog">
    <div class="modal-content">
      <!-- Cabeçalho do Modal -->
      <div class="modal-header">
        <h4 class="modal-title">Adicionar Pets</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
      </div>
      <!-- Corpo do Modal -->
      <div class="modal-body">
        <form id="frmAddPets" method="POST" action="/novo-pet">
          <hr>
          <div class="mb-3 row">
            <div class="col-md-8 col-sm-12">
              <label for="nome" class="form-label">Nome</label>
              <input type="text" class="form-control" name="nome">
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="sexo" class="form-label">Sexo</label>
              <select type="text" class="form-select" name="sexo">
                <option value="M">Macho</option>
                <option value="F">Fêmea</option>
              </select>
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="especie" class="form-label">Especie</label>
              <select type="text" class="form-select" name="especie">
                <option value="cão">Cão</option>
                <option value="gato">Gato</option>
                <option value="tartaruga">Tartaruga</option>
              </select>
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="raca" class="form-label">raca</label>
              <input type="text" class="form-control" name="raca">
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="peso" class="form-label">Peso</label>
              <input type="text" class="form-control" name="peso">
            </div>
            <div class="col-md-4 col-sm-6">
              <label for="tamanho" class="form-label">Tamanho</label>
              <input type="text" class="form-control" name="tamanho">
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

```

        </div>
        <div class="col-md-4 col-sm-6">
            <label for="idade" class="form-label">Idade</label>
            <input type="text" class="form-control" name="idade">
        </div>
        <div class="col-12">
            <label for="doenca" class="form-label">Doenca</label>
            <textarea class="form-control" name="doenca"
rows="2"></textarea>
        </div>
        <div class="col-12">
            <label for="obs" class="form-label">Obs.:</label>
            <textarea class="form-control" name="obs" rows="5"></textarea>
        </div>
    </div>

    <hr>
    <input type="submit" class="btn btn-primary" value="Salvar
Registro">
</form>
</div>
</div>
</div>
</div>
</div>
<script>
    // Instanciar um objeto Modal
    let divFrmAdicionarPets = document.getElementById('modalFrmAdicionarPets')
    let modalInteresse = new bootstrap.Modal(divFrmAdicionarPets)
    let frmAddPets=document.getElementById('frmAddPets')

    async function abrirModalAddPts() {
        // Abrir o modal
        if ( logado){
            modalInteresse.show()
        } else {
            window.location.href = "/login";
        }
    }

    function fecharModalFrmAdicionarPets() {
        // Fechar o modal
        modalInteresse.hide();
    }
</script>

```

5. Ainda no diretório de partial crie outro arquivo com o nome “**modal_verInteressados.ejs**” que exibira a lista de usuários interessados no pet. Insira o seguinte código:

```
<!-- Modal -->
```

```

<div class="modal" id="modalInteressados">
  <div class="modal-dialog modal-lg">
    <div class="modal-content">
      <!-- Cabeçalho do Modal -->
      <div class="modal-header">
        <h4 class="modal-title">Interessados no Pet</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
      </div>
      <!-- Corpo do Modal -->
      <div class="modal-body">
        <div class="container mt-5">
          <hr>
          <p>Lista de interessados: </p>
          <table class="table table-striped">
            <thead>
              <tr>
                <th scope="col">Data</th>
                <th scope="col">Nome</th>
                <th scope="col">Telefone</th>
                <th scope="col">WhatsApp</th>
                <th scope="col">Ação</th>
              </tr>
            </thead>
            <tbody id="tabelaCorpo">
              <!-- Linhas serão adicionadas aqui via JavaScript -->
            </tbody>
          </table>
          <hr>
        </div>
      </div>
    </div>
  </div>
</div>
<script>
  // Instanciar um objeto Modal
  let divInteressados = document.getElementById('modalInteressados')
  let modalInteressados = new bootstrap.Modal( divInteressados)
  let tabelaCorpo = document.getElementById('tabelaCorpo')
  tabelaCorpo.innerHTML = ''

  async function abrirModalInteressados(id) {
    // Abrir o modal
    if (logado){
      url = `${urlServidor}/obter/doacoes/${id}`
      let config = {
        headers: {
          'Authorization':sessionStorage.getItem('Authorization')
        }
      }
      // Adiciona o cabeçalho de autorização com o token

```

```

    }
  }
  let interesse = await axios.get(url, config)

  construirTabela(interesse.data)

} else {
  window.location.href = "/login"
}
}

function construirTabela(interesses){
  if (interesses.length === 0){
    alert('Ainda não existem interessados para este Pet.')
    fecharModalInteressados()
  } else {
    interesses.forEach(interesse => {
      const tr = document.createElement('tr')
      tr.innerHTML = `
        <td>${interesse.data_interesse}</td>
        <td>${interesse.usuario.nome}</td>
        <td>${interesse.usuario.telefone}</td>
        <td>${interesse.usuario.whatsapp}</td>
        <td>
          <button class="btn btn-success" value="${interesse.usuario.id}" onclick="">
            <i class="bi bi-check"></i>
          </button>
        </td>
      `
      tabelaCorpo.appendChild(tr)
      modalInteressados.show()
    })
  }
}

function fecharModalInteressados() {
  // Fechar o modal
  modalInteressados.hide();
}

</script>

```

6. Inclua as duas partials anteriores no arquivo “**area-exclusiva.ejs**” (no final uma linha antes do footer), conforme orientação a seguir:

```

</main>

<%- include('partials/modal_addPet.ejs')%>
<%- include('partials/modal_verInteressados.ejs')%>
<%- include('partials/footer.ejs') %>

```

7. Pronto, agora a aplicação será capaz de inserir novos registros de pets, exibir os pets na área exclusiva e exibir interessados.

Exercícios extras

Esta atividade tem o objetivo ampliar o conhecimento no desenvolvimento de regras de negócio de uma aplicação.

1. Desenvolva novas funcionalidades para editar e excluir registros de pets.
2. E para um usuário não registrar interesse em um pet mais de uma vez.

APENDICE

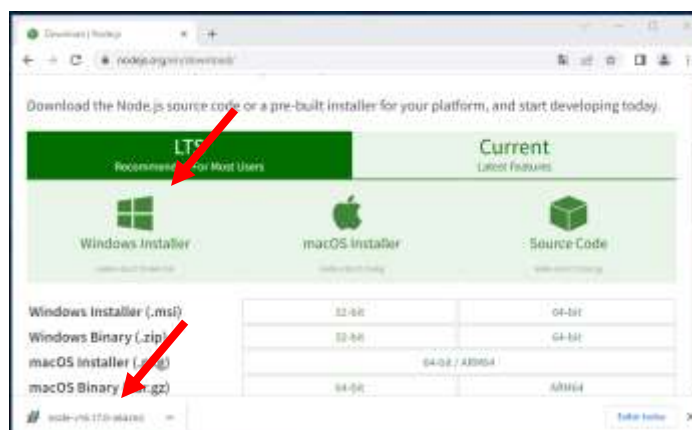
Instruções para o download e instalação do Node.js

Instruções para realizar o download e instalação do Node.js e suas principais dependências.

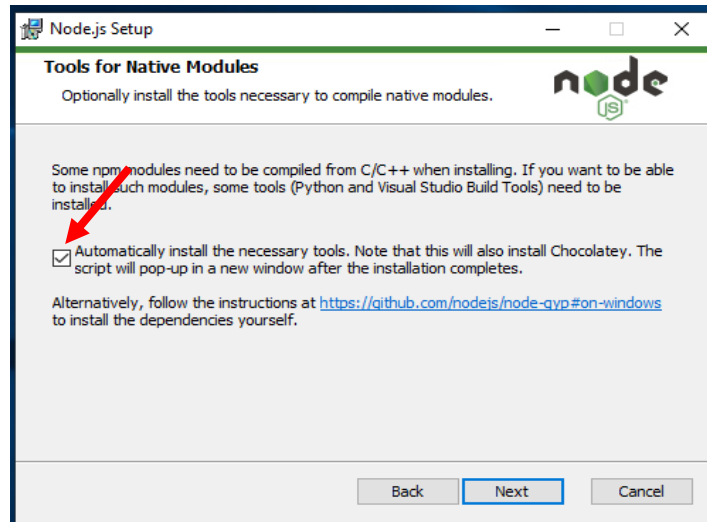
Obs.: Ao realizar esta atividade, procure utilizar sempre a última versão estável (LTS), a versão utilizada neste material é a 16.17.0, os passos podem sofrer pequenas alterações ao serem lançadas novas versões.

Realizar o download e instalar o Node.js

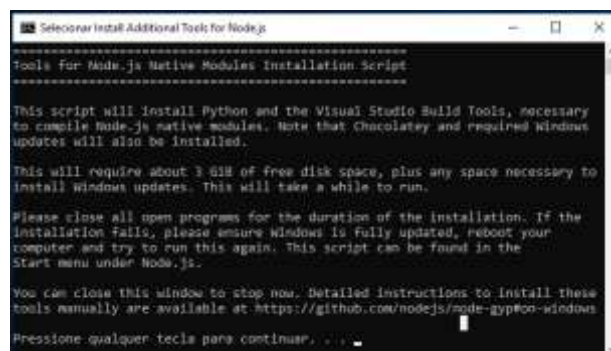
1. Acesse a url <https://nodejs.org/en/download/> e para realizar o download da versão mais atual, clique na opção compatível com o seu sistema operacional, por exemplo: Windows;
2. Após a conclusão do download execute o arquivo, conforme a ilustração abaixo:



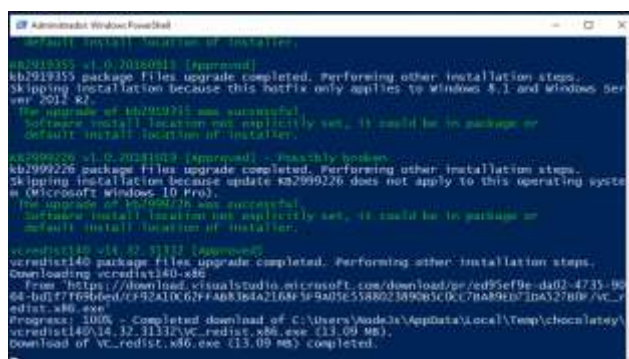
3. Siga os passos do assistente de instalação clicando no botão “Next”, até o passo “**Tools for Native Modules**” e ative a opção “**Automatically install the necessary tools**” para realizar a instalação de ferramentas necessárias para um bom funcionamento do Node.js.



4. Depois clique em próximo e por fim instalar após a conclusão clique em finish.
5. Uma tela do prompt se abrirá uma tela do prompt e iniciará a instalação do Chocolatey e várias bibliotecas e módulos nativos, pressione qualquer tecla para prosseguir com a instalação.



6. Após a janela se fechar, abrirá uma nova janela do Power Shell, onde poderá ser visto o progresso da instalação. Aguarde até a finalização, este processo pode ser bastante demorado, em muitos casos até mais de uma hora.

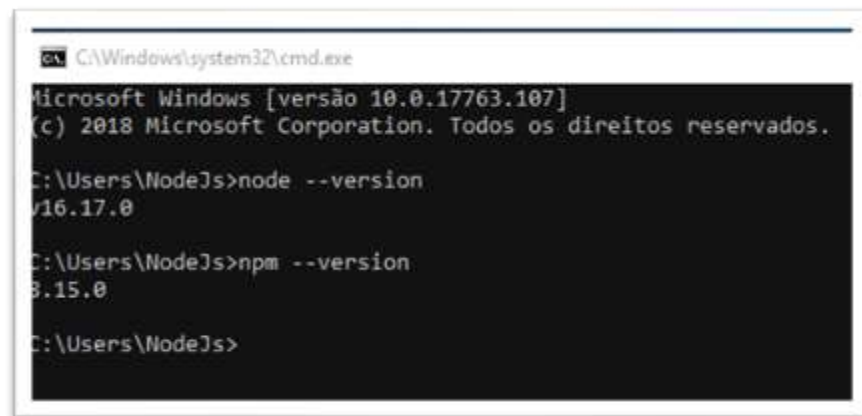


7. Após algum tempo, a instalação finalizará e o power shell será encerrado.
8. Para executar testes digite CMD na pesquisa do Windows e acesse o prompt de comandos.
9. Depois digite os seguintes comandos:

node -- version

npm – version

10. Se os comandos retornarem suas respectivas versões tudo foi instalado devidamente



```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 10.0.17763.107]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\NodeJs>node --version
v16.17.0

C:\Users\NodeJs>npm --version
8.15.0

C:\Users\NodeJs>
```

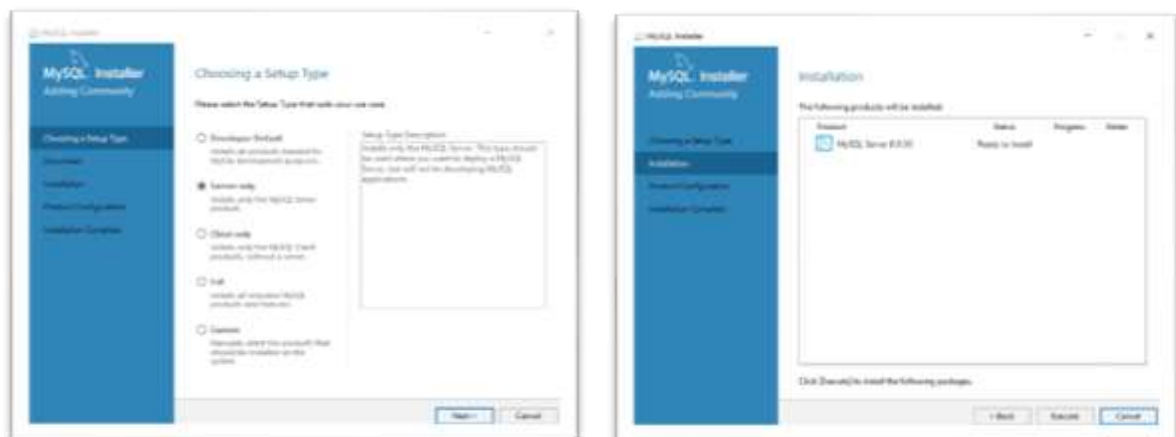
Instruções para o download e instalação do MySQL

Instruções para realizar o download e instalação do MySQL

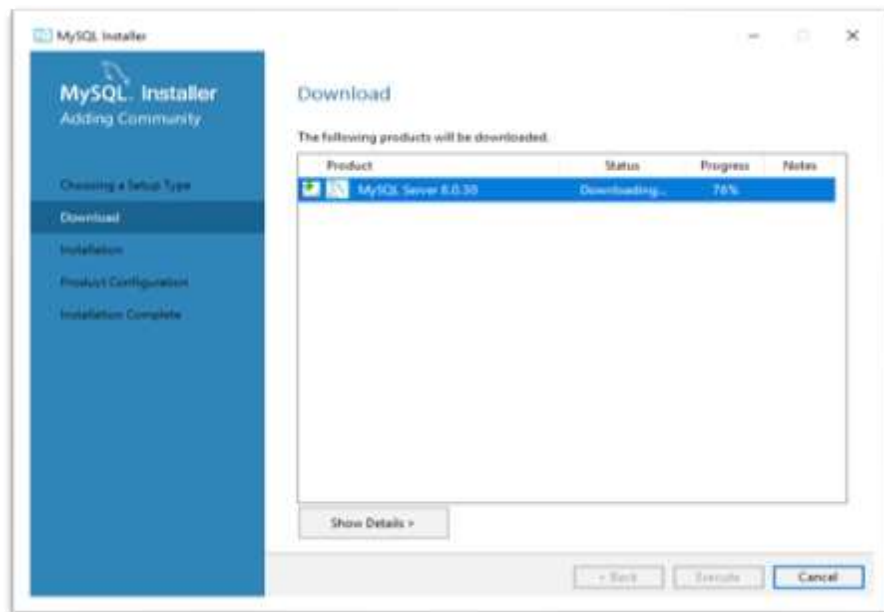
Obs.: Ao realizar esta atividade, procure utilizar sempre a última versão estável (LTS), a versão utilizada neste material é a 8.0.3.0, os passos podem sofrer pequenas alterações ao serem lançadas novas versões.

Realizar o download e instalar o Node.js

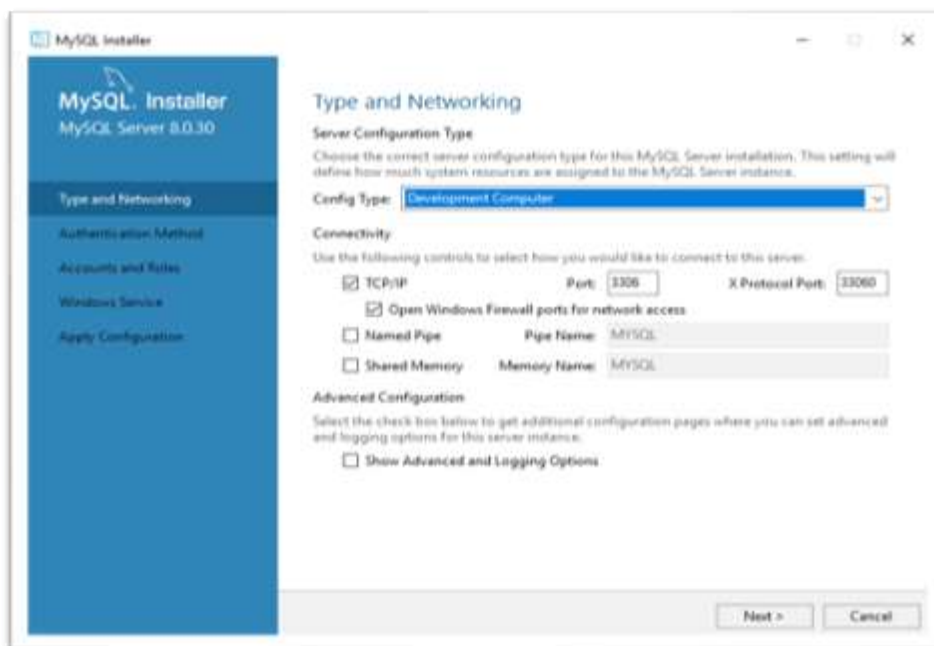
1. Acesse a url <https://dev.mysql.com/downloads/mysql/> e para realizar o download da versão mais atual, clique na opção compatível com o seu sistema operacional, por exemplo: Windows;
2. Após baixar o arquivo de instalação, execute-o e siga as instruções do assistente sempre clicando no botão next ou outro que indique a sequencia da instalação. Escolha as opções conforme imagens a seguir:



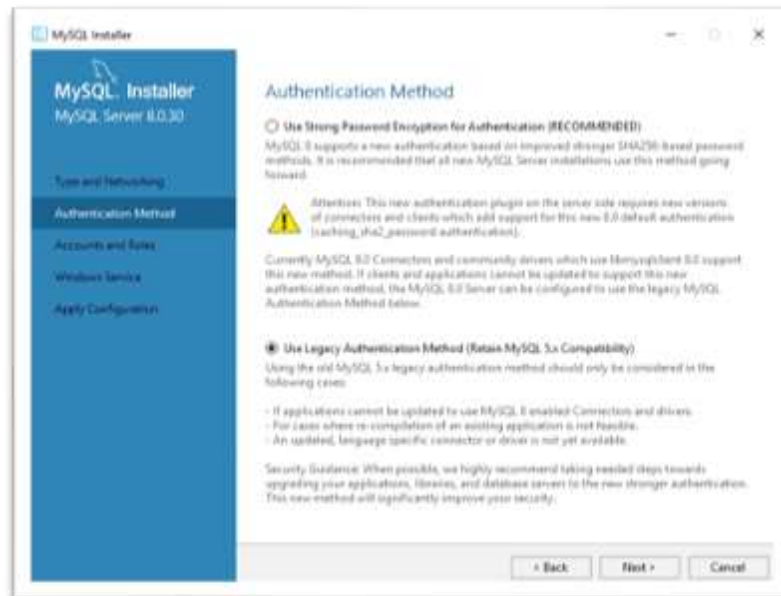
3. Aguarde a completar o download do servidor do MySQL e todas as suas dependências e depois clique em Execute, conforme imagem abaixo:



4. Na próxima tela, defina as configurações do serviço MySQL conforme definições abaixo:



5. Para uma configuração simplificada em um ambiente de testes e desenvolvimento utilize a opção de autenticação Legacy Authentication



6. Para finalizar, crie um acesso local definindo um usuário e senha conforme exemplo a baixo:

