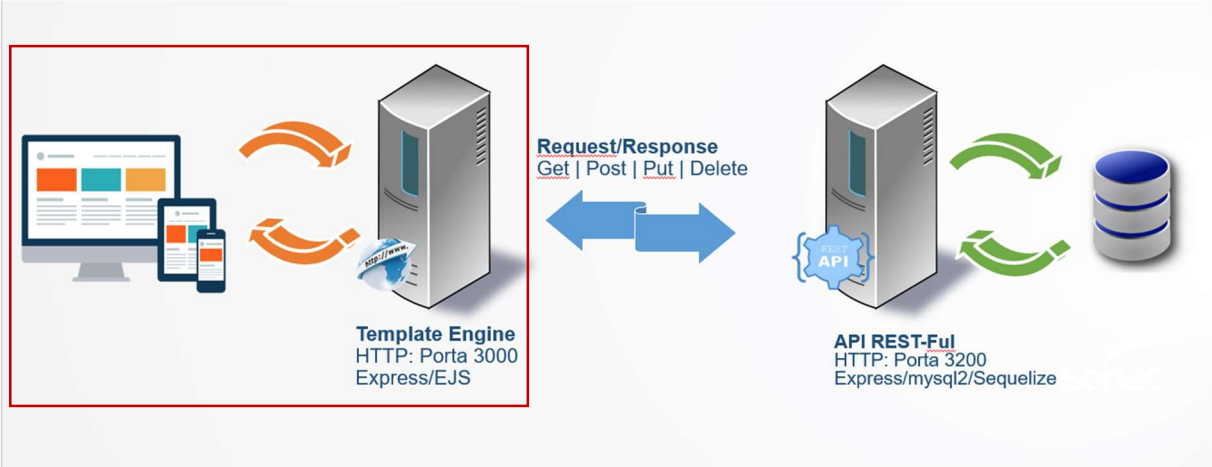
CAPÍTULO 7 – View Engine EJS e integração front-end com API

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_estruturaIF**" 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

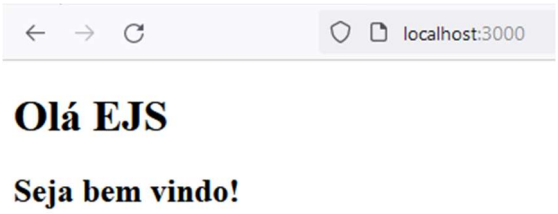
   Insira também a entrada **"start"**: "nodemon index.js" (arquivo pakete.json vide exemplos anteriores).
3. Crie o arquivo "**index.js**" e insira o seguinte código:

```
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.



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:

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 Capitulo 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">
```

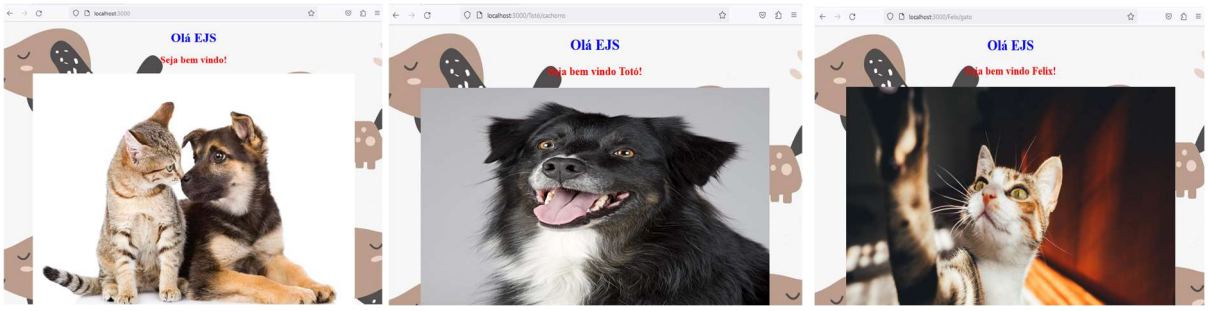
11. Ajuste também no arquivo "index.ejs" no fim da tag <body> acrescentando o seguinte código:

```
   <%if (especie=='cachorro'){%>
        <img src="/img/dog01.jpg" alt="">
    <%} else if (especie=='gato') {%>
        <img src="/img/cat01.jpg" alt="">
    <%} else {%>
        <img src="/img/default.png" alt="">
    <%}%>
```

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 capitulo 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 pakete.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>
```

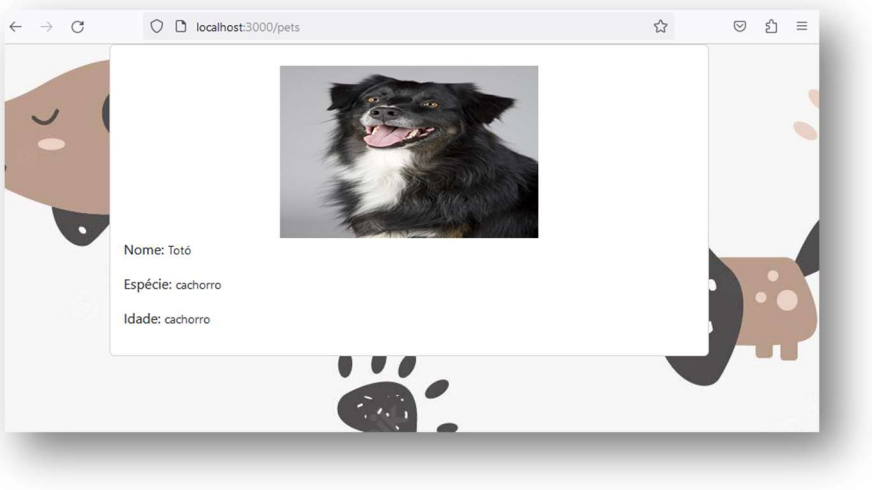
6. Agora crie um arquivo em views chamado “**pets.ejs**” com o seguinte código:

```
<!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'){%>
                        <img src="/img/dog01.jpg" alt="">
                    <%} else if (pets[0].especie=='gato') {%>
                        <img src="/img/cat01.jpg" alt="">
                    <%} else {%>
                    <img src="/img/default.png" alt="">
                    <%}%>
                </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).

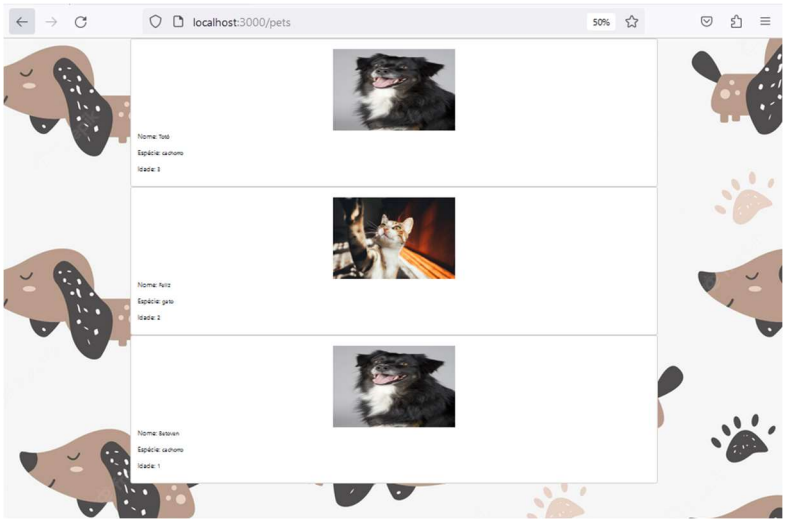
7. Rode o código e realize o teste na rota http://localhost:300/pets e verifique que está aparecendo apenas a exibição dos dados do primeiro elemento do vetor, isso porque ficou definido no código a exibição somente o primeiro elemento do vetor(pets[0]). Veja a exibição abaixo:



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(). Refatóre 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'){%>
                        <img src="/img/dog01.jpg" alt="">
                    <% } else if (pet.especie=='gato') {%>
                        <img src="/img/cat01.jpg" alt="">
                    <% } else {%>
                        <img src="/img/default.png" alt="">
                    <%}%>
                </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:300/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 entres 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 modulo 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 códio 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 pakete.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="/">
                        <img src="img/logo.png" alt="Logomarca da empresa" class="logoImg
img-fluid align-text-center">
                     </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 src="img/iconLogin.svg" class="px-2 img-fluid"></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">
            <img class="w-100 img-fluid" src="/img/slide1.png" class="d-block
w-100" alt="...">
          </div>
          <div class="carousel-item" data-bs-interval="2000">
            <img class="w-100 img-fluid" src="/img/slide2.png" class="d-block
w-100" alt="...">
          </div>
          <div class="carousel-item" data-bs-interval="2000">
            <img class="w-100 img-fluid" src="/img/slide3.png" class="d-block
w-100" alt="...">
          </div>
          <div class="carousel-item" data-bs-interval="2000">
            <img class="w-100 img-fluid" src="/img/slide4.png" class="d-block
w-100" alt="...">
          </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">
                            <img src="img/logo.png" alt="Logomarca da empresa"
class="logoImg img-fluid align-text-center">
                        </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 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 paritals 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

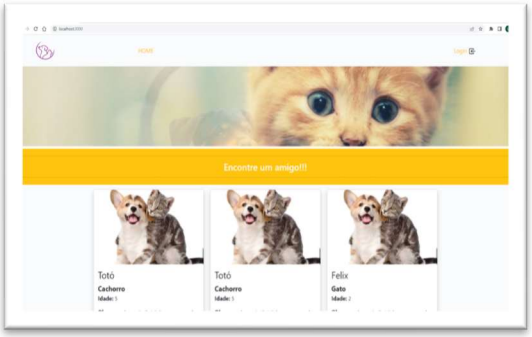
código HTML que definem cabeçalho e navbar pelas suas respectivas partials. O arquivo **login.ejs** deve ficar com o código similar ao que segue:

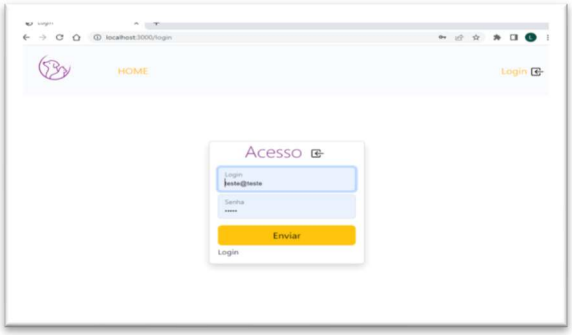
```
<%- include('partials/head.ejs') %>
<title>Login</title>
</head>
<body>
    <div class="container-fluid">
            <!--Menu-->
            <%- include('partials/menu.ejs') %>
    </div>
    <hr>
    <main class="my-5">
        <div class="container login my-5 py-5">
            <div class="row 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
src="img/iconLogin.svg" class="text-primary px-2 img-fluid"></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:

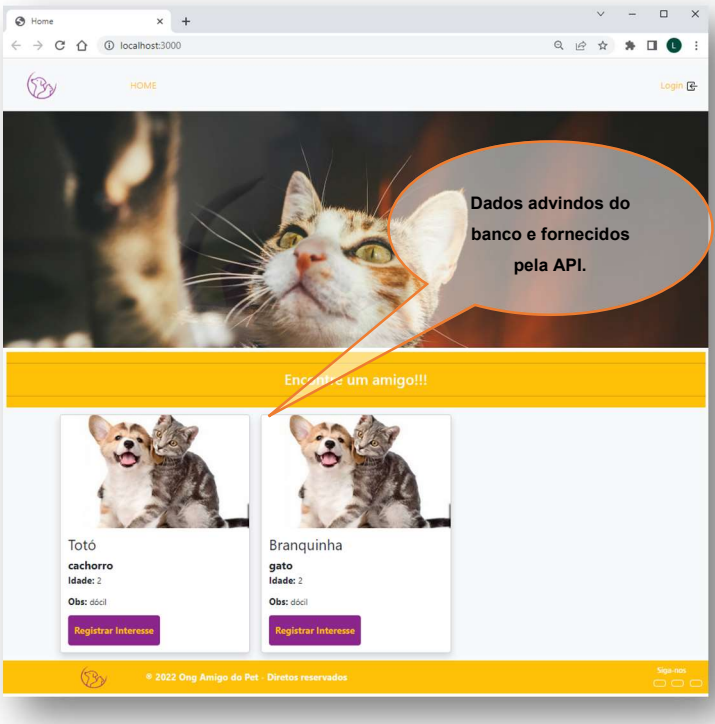
```
const express = require('express')
const app = express()
const cors = require('cors')
app.use(cors())
const axios = require('axios')
const consign=require('consign')
var porta = '3000'

//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 = 'consultar/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="dysplay-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">
                             <img src="img/petDefault.jpg"
                                class="card-img-top img-fluid" alt="...">
                           <div class="card-body">
                              <h3 class="card-title display-5 fs-2"><strong>
                                 <%= pet.nome %></strong></h5>
                              <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>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 capitulo 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().