Instituto Federal de Educação, Ciência e
Tecnologia de São Paulo campus Votuporanga

Apostila da disciplina Desenvolvimento Web 3 (DW3)

Prof. Dr. Evandro Jardini
eajardini@ifsp.edu.br

Esta apostila possui conteúdo extraído de diversos materiais como livros, apostilas, notas, artigos, etc elencados nas de Referências Bibliografias.

Sumário

Capítulo 1

Computação Distribuída utilizando o Formato *JavaScript Object Notation* (JSON)

1.1 Introdução

Nesse capítulo, iremos estudar o desenvolvimento de software utilizando utilizando **arquitetura Computação Distribuída** utilizando o formato de representação de estruturas denominado de *JavaScript Object Notation* (JSON).

Neste tipo de arquitetura temos componentes denominados de **servidores** que processam as requisições, controlam as regras de negócios e fazem acesso a Servidores de Banco de Dados (SGBDs) e componentes denomina-

dos de **clientes** que controlam as interface com o usuário. Esses componentes rodam de forma desacoplados um dos outros.

Iremos aprender durante este capítulo:

- O formato JSON e sua estrutura
- Serviços de Webservices
- APIs e REST
- Computação *Back-End* e *Front-End*.

1.2 Computação Distribuída

Computação distribuída é o processo de realizar uma tarefa usando vários computadores, cada um com seu próprio hardware e software. A computação distribuída é usada para uma variedade de tarefas, como programação cliente-servidor, computação em nuvem, Big data e jogos online.

- A **programação cliente-servidor** é um modelo de computação em que os recursos são compartilhados entre os clientes e os servidores. Os clientes solicitam recursos aos servidores e os servidores fornecem esses recursos aos clientes.
  - **Importante**: no modelo cliente-servidor não é caracterizado exclusivamente pelo uso de diversos computadores, mas o uso de diversos processos ora fazendo o papel de cliente e ora fazendo o papel de

servidores. Por exemplo: em um computador pode estar rodando um processo do *Enterprise Resource Planning* (ERP) com as regras de negócio e um processo do SGBD. O **processo do ERP** é considerado servidor para os computadores clientes deste sistema, mas é considerado cliente para o **processo do SGBD**.

- **Computação em nuvem**: a computação distribuída é usada para fornecer serviços de computação em nuvem. Isso inclui serviços como armazenamento de dados, computação e processamento de dados.
- **Jogos online**: a computação distribuída é usada para executar jogos online. Isso é importante para garantir que os jogos sejam executados de forma suave e que todos os jogadores tenham uma experiência consistente.
- **Big data**: a computação distribuída é usada para processar grandes quantidades de dados. Isso é importante para uma variedade de tarefas, incluindo análise preditiva, aprendizado de máquina e mineração de dados.

1.2.1 Vantagens da Computação Distribuída

Algumas vantagens:

- A computação distribuída pode permitir que tarefas sejam **executadas mais rapidamente** do que se fossem executadas em um único computador. Isso ocorre porque vários computadores podem trabalhar na tarefa ao mesmo tempo.
- A computação distribuída pode permitir que tarefas sejam executadas em computadores que estão localizados em diferentes locais. Isso pode ser útil para tarefas que precisam acessar dados que estão localizados

em diferentes locais, ou para tarefas que precisam ser executadas rapidamente, mesmo que os computadores não estejam localizados na mesma rede.

1.2.2 Desvantagens da Computação Distribuída

Algumas desvantagens:

- A computação distribuída pode ser mais **complexa de gerenciar** do que a computação em um único computador. Isso ocorre porque os administradores de sistema precisam garantir que todos os computadores estejam funcionando corretamente e que eles estejam se comunicando entre si de forma eficaz.
- A computação distribuída pode ser mais cara do que a computação em um único computador. Isso ocorre porque as organização precisam comprar e manter vários computadores.
- Como os sistemas executam de forma desacoplada, a equipe de desenvolvimento é formada por diversos profissionais tornando mais complexa a implementação do sistema.

1.3 *JavaScript Object Notation* (JSON)

*O JavaScript Object Notation (*JSON) é um formato leve para armazenar e transportar dados. JSON é frequentemente usado quando os dados são enviados de um servidor para uma página da web, para outro servidor, para aplicações em celulares, etc.

O JSON é autodescritivo e fácil de se entender, ler e escrever por humanos e também pode ser facilmente interpretado por computadores.

Arquivos JSON são usados para armazenar e transmitir dados entre diferentes aplicativos. Eles são usados em uma ampla variedade de aplicativos, incluindo:

- Requisições Asynchronous JavaScript and XML (AJAX)
- *Application programming interface* (API) web
- Aplicativos móveis
- Bancos de dados não relacionais
- Ferramentas de análise de dados

Os arquivos JSON podem ser abertos e manipulados em uma variedade de linguagens de programação, incluindo JavaScript, Python, Java e PHP. Também existem ferramentas disponíveis para converter arquivos JSON para outros formatos de dados, como XML, CSV e HTML.

1.3.1 Dados JSON

Os dados JSON são gravados **como pares** de *nome-valor*. Um par *nome-valor* consiste em um nome de campo (entre aspas duplas), seguido por **dois pontos**, seguido por um valor:

```
1 "firstName":"John"
```

1.3.2 Objetos JSON

Os objetos JSON são escritos entre **chaves**. Assim como em *JavaScript*, os objetos podem conter vários pares nome/valor:

```
1 {"firstName":"John", "lastName":"Doe"}
```

1.3.3 Matrizes JSON

As matrizes JSON são escritas entre colchetes. Assim como em *JavaScript*, um *array* pode conter objetos:

Listagem 1.1: *Array* de objetos JSON.

```
1 "employees":[
2   {"firstName":"John", "lastName":"Doe"},
3   {"firstName":"Anna", "lastName":"Smith"},
4   {"firstName":"Peter", "lastName":"Jones"}
5 ]
```

No exemplo da listagem 1.1 acima, o objeto *funcionários* é um *array*. Ele contém três objetos. Cada objeto é um registro de uma pessoa (com um nome e um sobrenome).

1.3.4 Objetos JSON Encadeados

Os objetos JSON podem ser encadeados formando vários níveis de profundidade:

Listagem 1.2: Objetos JSON encadeados.

```
"superheroes": {
  "squadName": "Super hero squad",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "powers": ["Million tonne punch","Damage resistance","Superhuman reflexes"]
    },
  ]
}
```

1.3.5 Acessando os dados de uma objeto JSON

Para acessar os dados armazenados em um objeto JSON, temos de passar o caminho que vai deste o nome da chave do nível mais elevado até o nome da chave do nível mais interno. Veja os exemplos

- Na listagem 1.1 queremos os nomes do terceiro registro:
  console.log(employees[2].firstname);

- Na listagem 1.2 queremos os dados do poder *Damage resistance*, mas vamos ver um exemplo em *javascript* dentro de uma página html:

```
1 <script>
2   var superHeroes = '{"squadName": "Super hero squad", "active": true,"members": [{"
     name": "Molecule Man","age": 29,"powers": ["Radiation resistance", "Turning tiny
     "]},{"name": "Madame Uppercut","age": 39,"powers": ["Million onne punch","Damage
     resistance","Superhuman reflexes"]}]}';
3
4   var superHeroesJSON = JSON.parse(superHeroes);
5
6   console.log("superHeroesJSON:" + superHeroesJSON.members[1].powers[1]);
7 </script>
```

1.4 *WebServices*

Um *webservice* é um componente de software que permite a comunicação entre aplicações diferentes. Ele é uma forma de compartilhar recursos e serviços entre diferentes sistemas, independentemente de sua plataforma ou linguagem de programação.

Os *webservices* são baseados em tecnologias web, como JSON, HTML, XML e HTTP. Isso os torna fáceis de usar e integrar com outras aplicações e que pode ajudar as organizações a compartilhar recursos e serviços de forma eficiente e eficaz.

Os *webservices* são usados em uma ampla variedade de aplicações, incluindo:

- Integração de sistemas;
- Serviços de *e-commerce*;
- Serviços de localização;
- Serviços de jogos e
- Serviços de mídia social

Aqui estão alguns dos benefícios dos *webservices*:

- **Flexibilidade**: Os *webservices* podem ser usados para comunicar entre aplicações diferentes, independentemente de sua plataforma ou linguagem de programação;

- **Escalabilidade**: Os *webservices* podem ser facilmente escaláveis para atender às necessidades de uma organização em crescimento;
- **Segurança**: Os *webservices* podem ser usados para proteger a privacidade e a segurança dos dados;
- **Confiabilidade**: Os *webservices* podem ser usados para garantir que os serviços sejam entregues de forma confiável e consistente e
- **Eficiência**: Os *webservices* podem ser usados para automatizar tarefas e processos, o que pode economizar tempo e dinheiro.

1.5 *Application Programming Interface* (API) e *Representational State Transfer* (REST)

Uma API é um conjunto de funções que foram **implementadas em um programa de computador** que são disponibilizados para que outros programas possam utilizá-las diretamente de forma simplificada; sem envolver-se em detalhes da implementação.

Representational State Transfer (REST) não é um protocolo ou padrão, mas sim um **conjunto de restrições de arquitetura**. Os desenvolvedores de API podem implementar a arquitetura REST de maneiras variadas.

Um *endpoint* é um **endereço** de rede que identifica um **recurso** ou **serviço**. Em uma API, um *endpoint* é o **endereço de rede que identifica um recurso específico** que pode ser acessado pela API. Os *endpoints* são geralmente nomeados de forma descritiva para refletir o recurso que eles representam. Por exemplo, o *endpoint* para o recurso *getAllUsers* (nome da função que retorna os usuários do sistema) pode ser chamado de */usuario* ou */systemusers*

Quando um cliente faz uma solicitação a um *endpoint* de uma *API REST*, essa API transfere uma representação do estado do recurso ao solicitante. Essa informação é entregue via HTTP utilizando um dos vários formatos possíveis: JSON, HTML, XML, etc. O **formato JSON** é o mais usado porque, apesar de seu nome, é independente de qualquer outra linguagem e pode ser lido por máquinas e humanos.

As *APIs REST* são baseadas nos métodos HTTP padrão, como **GET**, **POST**, **PUT** e **DELETE**. Esses métodos são usados para acessar, criar, atualizar e excluir recursos.

Os recursos são objetos que são expostos pela API. Eles podem ser entidades como *clientes, produtos ou pedidos*.

Alguns exemplos de *endpoints*:

- /users: este *endpoint* retorna uma lista de todos os usuários.
- /products: este *endpoint* retorna uma lista de todos os produtos.
- /orders: este *endpoint* retorna uma lista de todas os pedidos.

1.5.1 Exemplo de API REST

A seguir temos uma API implementada em *NodeJS* que compartilha o recurso cujo *endpoint* é ***api/v1/users*** dentro do domínio *exemple.com*:

Listagem 1.3: Exemplo de código para criar uma API para o *endpoint api/v1/users*.

```
1 api.get("/api/v1/users", appUsers.getAllUsers);
```

Aqui vemos uma API que ao ser chamada por uma aplicação cliente, vai executar a função *getAllUsers* que retorna um JSON contendo todos os usuários do sistema.

Para consumir esta API, podemos desenvolver em *javascript* o seguinte código:

Listagem 1.4: Exemplo de requisição para consumir a API *api/v1/users*.

```
<script>
  const request = new XMLHttpRequest();
  request.open("GET", "https://example.com/api/v1/users");
  request.send();
  request.onload = () => {
    if (request.status === 200) {
      const users = JSON.parse(request.responseText);
      console.log(users);
    } else {
      console.log("Erro:", request.status);
    }
  };
</script>
```

Este código fará uma solicitação ao *endpoint* /api/v1/users da API em *example.com*. Se a solicitação for bem-sucedida, o código imprimirá a lista de usuários na console. Se a solicitação falhar, o código imprimirá o código de erro na console.

1.5.1.1 Segurança em *API Rest*

Uma API em produção fica disponível para acessada pela Internet. Assim, qualquer pessoa que tiver o endereço da API poderá solicitar seus serviços. Caso a API não tenha algum sistema de segurança, pessoas não autorizadas podem obter dados não autorizados.

Uma das maneiras de se implementar segurança com APIs é utilizar o recurso de *token JSON Web Token* (JWT). Um *token* JWT é um pequeno trecho de dados codificado com JSON que é usado para transmitir informações de segurança entre duas partes como uma forma de autenticação. Os *tokens* JWT são normalmente usados em aplicativos da web para representar a identidade de um usuário e permitir que eles acessem recursos protegidos.

Os tokens JWT são compostos por três partes:

- ***Header*** (Cabeçalho): define o tipo de token (JWT) e o algoritmo de assinatura usado.

```
1 {
2     "alg": "HS256",
3     "typ": "JWT"
4 }
```

- ***Payload*** (Carga): é o corpo do *token* e contém as informações de segurança. Ele contém as *claims* (informações) da entidade tratada, normalmente o usuário autenticado. As *claims* podem ser de 3 tipos:
  - *Reserved*: atributos não obrigatórios que são usados na validação do *token* pelos protocolos de segurança das APIs.

  - *Public*: atributos que usamos em nossas aplicações. Normalmente armazenamos as informações do usuário autenticado na aplicação.
  - *Private*: atributos definidos especialmente para compartilhar informações entre aplicações.

- ***Signature*** (Assinatura): é um *hash* da cabeçalho e da carga e é usada para verificar a integridade do *token*.

```
1 HS256SHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret_key)
```

Cujo resultado seria algo assim:

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
2 eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
3 SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Iremos usar o *token* JWT em nossa aplicação.

1.6 Computadores *Back-End* e *Front-End*

1.6.1 Computador *Back-End*

Um computador *back-end* é um computador que **executa as funções de processamento e armazenamento de um aplicativo ou sistema**.

Normalmente, é um servidor que não é diretamente acessível aos usuários.

Os computadores *back-end* geralmente **são mais poderosos do que os computadores *front-end***, pois precisam lidar com grandes quantidades de dados e processamento.

Os computadores *back-end* são usados em uma ampla variedade de aplicativos, incluindo:

- Sites e aplicativos da web;
- Bancos de dados;
- Sistemas de arquivos;
- Servidor de e-mail;
- Servidor de rede.

1.6.2 Computador *Front-End*

Um computador *front-end* é um computador que **interage diretamente com o usuário**.

Normalmente, é um **computador pessoal ou um dispositivo móvel**.

Os computadores *front-end* geralmente **são menos poderosos do que os computadores *back-end***, pois não precisam lidar com grandes quantidades de dados e processamento.

Os computadores *front-end* são usados em uma **ampla variedade de aplicativos,** incluindo:

- Navegadores da web;
- Processadores de texto;
- Planilhas;
- Editores de imagens;
- Jogos

1.7 Finalizando

Neste capítulo foram apresentados diversos conceitos referentes a Computação Distribuída. Nos próximos capítulos iremos aprender a desenvolver softwares contendo os conceitos apresentado aqui.

Capítulo 2

Implementação do Servidor *Back-End*

2.1 Introdução

Nesse capítulo, iremos estudar o desenvolvimento de software no papel de *back-end*. Como dito, *back-end* é um computador que **executa as funções de processamento e armazenamento de um aplicativo ou sistema**.

Para o desenvolvimento do sistema exemplo, iremos usar o Node.js e alguns *frameworks* que facilitam a tarefa do programador na implementação de códigos. Iremos implementar as operações de *Create*, *Read*, *Update* e *Delete* (**CRUD**) com registros no SGBD Postgres. E ao término do capítulo, teremos APIs prontas para ser usadas pela aplicação *front-end*.

2.1.1 Diretório padrão da disciplina

Nossos programas ficarão dentro do diretório padrão *dw3*. Assim, sempre que formos criar um novo sistema, iremos criá-lo dentre desse diretório.

Para criar o diretório *dw3* faça:

1. Abra um terminal em seu sistema operacional.
2. Digite os seguintes comandos:

```
1 mkdir dw3
2 cd dw3
```

Para verificarmos em qual diretórios estamos, digite o comando

```
1 pwd
```

Caso o retorno não seja dw3, temos que entrar dentro dele com o comando *cd*.

2.1.2 Repositório Remoto no *Github* da disciplina

Após criado o diretório padrão da disciplina, vamos criar o **Repositório no *Github*** para armazenar os exemplos desenvolvidos durante a aula. **Porém**, antes de criar o *repositório*, o aluno deve verificar se tem *token* ativo para usar no momento de fazer o *push*. Caso não tenho o *token* ativo, o discente deverá criá-lo.

O aluno(a) deverá criar um **repositório remoto vazio no *Github*** sem o arquivo *README.md* no *Github* e em seguida seguir as instruções para criar o **repositório local** dentro do **diretório dw3**.

Adicionalmente, antes de efetuar o primeiro *git add .* , deve criar um arquivo de nome .gitignore:

```
1 touch .gitignore
```

Neste arquivo serão indicados os diretórios e arquivos que não deverão ser enviados ao **repositório remoto**. Por exemplo, a cada novo servidor *Node.js*, devemos indicar dentro do arquivo *.gitignore* o caminho do diretório **node_modules**, pois é um arquivo onde ficam instaladas as bibliotecas do *Node.js*.

Feito isso, pode dar prosseguimento na criação do **repositório local**.

A **ideia** é que a cada novo exemplo de servidor que criarmos, iremos enviá-lo para o repositório remoto, fazendo com que **o aluno tenha um portfólio** para apresentar no futuro com profissional da área.

2.2 *Node.js*

Node.js, ou simplesmente *Node*, é um *framework* de desenvolvimento de software de código aberto, gratuito e de multiplataforma. **Ele é escrito em *JavaScript***.

Node.js é usado para criar aplicativos da web que são responsivos e escaláveis. É indicado para aplicativos que precisam lidar com grandes quantidades de dados ou que precisam ser executados em vários dispositivos.

Algumas características são:

- É escrito em *JavaScript*, que é uma linguagem de programação popular e bem conhecida.
- Usa o motor *JavaScript* V8 do Google Chrome, que é um dos motores *JavaScript* mais rápidos disponíveis.
- É multiplataforma, o que significa que pode ser usada para criar aplicativos que podem ser executados em Windows, Mac e Linux.
- É um *framework* de desenvolvimento de software de código aberto, gratuito e com uma grande comunidade de usuários e desenvolvedores.

2.2.1 Instalação do Node.js

Para instalarmos o Node temos opções de acordo com o Sistema Operacional (SO) que estamos utilizando. O **site oficial do Node é *https://nodejs.org***.

Para sabermos qual a versão do Node que nosso computador possui, basta digitar o comando

```
1 node -v
2 npm -v
```

A instalação do Node no Windows, Mac e Linux por meio do site ***https://nodejs.org*** é feita de acordo com as instruções na *url https://nodejs.org/pt/download/package-manager*.

A instalação no Linux **por meio do apt-get** é de acordo com as instruções da *url https://deb.nodesource.com* .

2.2.2 Criando a primeira aplicação em Node.js

Para criarmos uma aplicação usando o Node, fazemos os seguintes passos dentro do **diretório padrão da disciplina**:

1. Crie um diretório chamado de 1_primeiro e entre dentro dele:

```
1 mkdir 1_primeiro
2 cd 1_primeiro
```

2. Instale os *frameworks* digitando o comando:

```
1 npm i express dotenv
```

   (a) Onde*:*
   express: é um *framework* que facilita a vida do programador.
   dotenv: permite a leitura de variáveis dentro de arquivos env.

3. Crie o arquivo *.env*:

```
1 touch .env
```

4. Dentro do arquivo *.env* coloque as seguintes linhas:

```
APP_NAME=1_Primeiro
PORT = 40000
```

5. Crie o arquivo *app*.js

```
touch app.js
```

6. Dentro do arquivo *app.js* coloque as seguintes linhas:

```
1  //@ Importa as bibliotecas
2  const express = require("express")
3  require("dotenv").config();
4
5  //@ Configura o servidor
6  const app = express();
7  const port = process.env.PORT;
8
9  //@ Cria uma rota para o endereço raiz.
10 app.get("/", (req, res) => {
11   res.send("Hello DW3!")
12 })
13
14 //@ Inicia o servidor
15 app.listen(port, () => {
16   console.log("Executando a aplicação:" , process.env.APP_NAME);
17   console.log("Example app listening on port:", port);
18 })
```

7. Para executar o servidor digite:

```
node app.js
```

8. Abra um navegador e digite o endereço *http://localhost:40000*.

9. Para encerrar a execução do servidor Node, no terminal, digite *ctrl+C*.

10. Envie para o **repositório remoto** a aplicação. Não se esqueça de acrescentar o diretório *node_modules* no arquivo *.gitignore*.

```
## arquivo .gitignore em dw3/
node_modules/

## Comando git para enviar os arquivos locais para o repositório remoto
git add .
git commit -m "texto"
git push -u origin main
-- vai solicitar o usuário do github
-- vai solicitar a senha que é o TOKEN do usuário
```

2.2.3 Criando a segunda aplicação em Node.js

Vamos criar uma outra aplicação usando o padrão *Model*, *View* , *Controller* (MVC), porém, como é servidor back-end, não teremos *View* e nem *Model* por não utilizarmos SGBD. Usaremos o VSCode para implementar nosso servidor.

1. Dentro do diretório padrão, crie a outra aplicação:

```
mkdir 2_segundo
cd 2_segundo
```

2. Vamos usar o *VSCode* e instalar a extensão *Rest Client* desenvolvido por Huachao Mao. Ela vai servir para fazermos testes de API.

3. Instale os *frameworks* digitando o comando:

```
## Desta fez não iremos instalar o dotenv e nem fazer uso dele
npm i express
```

4. Crie os seguintes diretórios:

```
1 mkdir controller
2 mkdir routes
3 mkdir tests
```

5. Dentro do diretório *controller*, crie o arquivo *ctlHello.js* com o seguinte código:

```
1  ## arquivo: controller/ctlHello.js
2
3  const hello = (req, res) => (async () => {
4   res.json({ status: "ok", "mensagem": "Olá segundo!" });
5  })();
6
7  const helloUser = (request, res)  => (async () =>{
8      const { username } = request.body
9      res.json({ status: "ok", "nomeusuario": username });
10 } )();
11
12 module.exports = {
13     hello,
14     helloUser,
15 }
```

6. Dentro do diretório *route*s, crie o arquivo *route.js* com o seguinte código:

```
1  ## arquivo: routes/route.js
2
3  //@ Importa as bibliotecas e arquivos
4  const express = require("express");
5  const routerApp = express.Router();
6  const appHello = require("../controller/ctlHello");
7
8  //@ Configura as rotas
9  routerApp.get("/", appHello.hello);
10 routerApp.post("/helloUser", appHello.helloUser);
11
12 //@ Exporta a variável com as rotas
13 module.exports = routerApp;
```

7. Crie o arquivo *app*.js

```
1  touch app.js
```

8. Dentro do arquivo *app.js* coloque as seguintes linhas:

```
## arquivo: app.js

//@ Importa as bibliotecas e arquivos
const express = require("express");
const router = require("./routes/route");

//@ Configura o servidor
const app = express();
const port = 40000;
app.use(express.json());
app.use(router);

//@ Inicia o servidor
app.listen(port, () => {
  console.log("App listening at port ${port}")
})
```

9. Para executar o servidor digite:

```
node app.js
```

10. Agora, vamos criar os teste de requisição. Vamos usar a extensão *REST Client* por *Huachao Mao* do *VScode*. Dentro do diretório *tests*, crie o arquivo *tests.rest* com o seguinte código:

```
### arquivo: tests/tests.rest

### teste de API /
GET http://localhost:40000/ HTTP/1.1
content-type: application/json

### teste de API helloUser
POST http://localhost:40000/helloUser HTTP/1.1
content-type: application/json

{
    "username": "John Doe"
}
```

11. Execute os testes do arquivo *tests.rest:*

```
1 - Clique em cima de um teste
2 - Com o botão direito selecione a opção Send Request
```

12. Para encerrar a execução do servidor Node, no terminal, digite *ctrl+C*.

13. Envie para o **repositório remoto** a aplicação. Não se esqueça de acrescentar o diretório *node_modules* no arquivo *.gitignore*.

```
## arquivo .gitignore em dw3/
node_modules/

## Comando git para enviar os arquivos locais para o repositório remoto
git add .
git commit -m "texto"
git push -u origin main
-- vai solicitar o usuário do github
-- vai solicitar a senha que é o TOKEN do usuário
```

2.2.4 Exercícios

Crie uma API para receber dois números e uma operação aritmética. A API deve retornar a operação aritmética indicada durante a requisição da API.

O formato da requisição será:

```
POST http://localhost:40000/calculadora HTTP/1.1
content-type: application/json

{
  "num1": 4,
  "num2": 2,
  "operacao": "+"
}
```

Requisitos:

1. Criar uma aplicação em Node para realizar cálculos com as 4 operações aritméticas;
2. A aplicação terá estrutura semelhante ao exemplo da seção 2.2.3;
3. Implementar as 4 operações aritméticas dentro do arquivo *controller/calculadora.js* cujo nomes e assinaturas das funções seguirão o modelo da função de somar: fSoma(num1Par, Num2Par);

4. Ainda no arquivo de *controller/calculadora.js* acrescente mais uma função chamada de *fCalculo (const fCalculo = (request, res) ...)* e fazendo sua implementação. Será essa função que irá ser usada para a criação da API e portanto sua assinatura deve seguir o modelo do *ctlHello.js* da seção 2.2.3;

5. Alterar o arquivo de *route/route.js* acrescentando as APIs do tipo *post* referente na nova função no *controller* e

6. Alterar o arquivo de *tests* acrescentando o teste da API conforme mostrado acima.

2.3 Servidor *Back-End do* Sistema Acadêmico

Vamos dar início a implementação do servidor *back-end* de nossa aplicação exemplo.

Nossa aplicação consta de um sistema acadêmico com os módulos *Aluno* e *Curso*. **Analise seu código no repositório remoto antes de continuar**.

O *script* do banco de dados do sistema está pronto e será explicado na próxima seção.

2.3.1 Banco de dados

No repositório remoto da disciplina no *Github*, há o arquivo *dw3/backend/databaseConfig*.sql cujo conteúdo é mostrado abaixo:

```
1 ----- Cria um banco de dados
2 -- create database dw3;
```

```
3
4  create table IF NOT EXISTS cursos (
5      cursoid bigserial constraint pk_cursos PRIMARY KEY,
6      codigo varchar(50) UNIQUE,
7      descricao VARCHAR(60),
8      ativo boolean,
9      deleted boolean DEFAULT false
10 );
11
12 insert into cursos values
13     (default, 'BSI', 'Bacharelado em Sistemas de Informação', true),
14     (default, 'DIREITO', 'Bacharelado em Direito', true),
15     (default, 'LETRAS', 'Licenciatura em Letras', true),
16     (default, 'ADM', 'Bacharelado em Administração', false)
17     ON CONFLICT DO NOTHING;
18
19 create table IF NOT EXISTS alunos (
20     alunoid bigserial constraint pk_alunos PRIMARY KEY,
21     prontuario varchar(10) UNIQUE,
22     nome varchar(50),
23     endereco VARCHAR(60),
```

```
24      rendafamiliar numeric(8,2),
25      datanascimento date,
26      cursoid bigint constraint fk_aluno_curso REFERENCES cursos,
27      deleted boolean DEFAULT false
28  );
29
30  insert into alunos values
31      (default, 'pront1', 'José das Neves', 'Rua A, Votuporanga', 6891.60, '2000-01-31',
32          (SELECT cursoid from CURSOS where codigo = 'BSI')),
33      (default, 'pront2', 'Maria Silveira', 'Rua B, São José do Rio Preto', 7372.41, '
        2002-03-12',
34          (SELECT cursoid from CURSOS where codigo = 'DIREITO'))
35  ON CONFLICT DO NOTHING;
36
37  create table IF NOT EXISTS usuarios (
38      usuarioid bigserial constraint pk_usuarios PRIMARY KEY,
39      username varchar(10) UNIQUE,
40      password text,
41      deleted boolean DEFAULT false
42  );
43
```

```
44 CREATE EXTENSION if NOT EXISTS pgcrypto;
45
46 insert into usuarios values
47     (default, 'admin', crypt('admin', gen_salt('bf'))), -- senha criptografada com bcrypt
48     (default, 'qwe', crypt('qwe', gen_salt('bf'))) -- senha criptografada com bcrypt
49 ON CONFLICT DO NOTHING;
50
51 -- Usado para exercícios
52
53 create table IF NOT EXISTS clientes (
54     clienteid bigserial constraint pk_clientes PRIMARY KEY,
55     codigo varchar(50) UNIQUE,
56     nome VARCHAR(60),
57     endereco VARCHAR(50),
58     ativo boolean,
59     deleted boolean DEFAULT false
60 );
61
62 insert into clientes values
63     (default, 'CLI01', 'João da Silva', 'Rua A1', true),
64     (default, 'CLI02', 'Marcia Almeida', 'Rua B2', true)
```

```
65      ON CONFLICT DO NOTHING;
66
67  create table IF NOT EXISTS pedidos (
68      pedidoid bigserial constraint pk_pedidos PRIMARY KEY,
69      numero bigint UNIQUE,
70      data DATE,
71      valortotal numeric(9,2),
72      clienteid bigint constraint fk_pedido_cliente REFERENCES clientes,
73      deleted boolean DEFAULT false
74  );
75
76  insert into pedidos values
77   (default, 234, '2020-01-31', 6891.60, (SELECT clienteid from CLIENTES where codigo = '
       CLI01'))
78   ON CONFLICT DO NOTHING;
```

A sintaxe do código é compatível com o SGBD *PostgreSQL*. Analise e entenda o código antes de prosseguir com a implementação do servidor *back-end* sistema exemplo

2.3.2 Procedimentos iniciais

Para desenvolver nosso servidor *back-end*, vamos realizar os seguintes procedimentos:

1. Vamos criar o diretório chamado dw3backend:

```
1 mkdir dw3backend
```

2. Dentro dele vamos instalar os frameworks:

```
1 npm i express pg body-parser dotenv jsonwebtoken bcryptjs
```

   Não se esqueça de adicionar o ***node_modules*** no arquivo *.gitignore*.

3. Crie o arquivo *.env* com o seguinte conteúdo:

```
1 APP_NAME=dw3
2 SECRET_API=9as1%12#xz0#@
3 # Banco de dados
4 DB_NAME=dw3
5 DB_USER = postgres
6 DB_PASS=postdba
7 DB_HOST=172.17.0.1
```

4. Dentro dele, vamos criar os diretórios:

```
1 mkdir apps # contém todas as nossas aplicações
2 mkdir database # contém a configuração do banco de dados
3 mkdir routes # contém as rotas da aplicação
```

5. Crie o arquivo *app.js* com o seguinte conteúdo:

```
const express = require('express');
const bodyParser = require('body-parser');
require('dotenv').config();

const app = express(); const port = 40000;

app.listen(port, () => {
    console.log(`App listening at port ${port}`)
})
```

6. Execute o servidor:

```
node app.js
```

7. Para parar o servidor, *digite ctrl + c*.

Importante: a apostila poderá conter apenas parte dos arquivos de código. Para a **versão completa**, use os arquivos disponibilizados no ***github***.

2.3.3 Configurando o arquivo de rotas

Dentro do diretório *routes*:

1. Crie o arquivo *router.js* com o seguinte conteúdo:

```
1  const express = require("express");
2  const routerApp = express.Router();
3
4  // middleware that is specific to this router
5  routerApp.use((req, res, next) => {
6    next();
7  });
8
9  routerApp.get("/", (req, res) => {
10   res.send("Olá mundo!");
11 });
12
13 //@ Rotas de Alunos
14
15 //@ Rotas de Cursos
16
17 // Rota Login
18
19 module.exports = routerApp;
```

2. Altere o arquivo *app*.js para ficar assim:

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  require('dotenv').config();
4
5  const router = require('./routes/router');
6
7  const app = express();
8  const port = 40000;
9
10 // app.set('view engine', 'ejs');
11 app.use(bodyParser.urlencoded({ extended: false, }));
12 app.use(express.json());
13
14 //@ Utiliza o routerApp configurado em ./routes/route.js
15 app.use(router);
16
17 app.listen(port, () => {
18   console.log(`App listening at port ${port}`)
19 })
```

3. Execute o servidor e faça o teste com o *browser* na rota / :

```
# No console
node apps.js

# No browser
localhost:40000/
```

4. Para o servidor com *ctrl*+*c*.

2.3.4 Configurando o banco de dados

Dentro do diretório *database*:

1. Crie o arquivo *databaseconfig*.js com o seguinte conteúdo:

```
const Pool = require('pg').Pool
const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASS,
  port: process.env.DB_PORT,
```

```
8  })
9
10 module.exports = {
11     query: (text, params) => pool.query(text, params),
12   };
```

2.3.5 Configurando o diretório de aplicação: *apps*

O **diretório *apps*** contém os código de nossa aplicação. Será usado o padrão *Model*, *View* e *Controller* (MVC) , porém sem o *View*. *Model*, *controller* e, também, *tests* serão subdiretórios dentro do diretório referente ao módulo.

A estrutura de diretório será semelhante à utilizada por aplicações *Django* em que cada módulo de nosso sistema terá seu próprio diretório. **Este será o padrão adotado pela disciplina**.

Como nossa aplicação contém apenas 3 módulos, *alunos, cursos e login*, iremos criar estes três diretórios:

```
1 mkdir alunos
2 mkdir cursos
3 mkdir login
```

2.3.6 Considerações sobre a implementação dos módulos do sistema

A seguir, iremos ver os passos necessários para implementar o módulo de login. Essa sequencia de passos deverá ser feita para cada novo módulo. **Assim, irei descrever o módulo de *login* e os demais serão criados na mesma sequência**. **São sempre 6 passos** que serão descritos na próxima seção.

Outra coisa, será mostrado na apostila e no *github* todo o código pronto para depois fazermos os testes, porém na prática, o *software* deve ser testado a cada nova rotina pronta. Não faça tudo para testá-lo, teste a cada nova rotina em desenvolvimento.

2.3.6.1 As operações de CRUD

Os módulos de *aluno* e *curso* conterão, em seus respectivos arquivos de *model* e *controller* , 5 operações do tipo CRUD, as quais são:

1. *getAllXXX*
2. *getXXXByID,*
3. *insertXXX,*
4. *updateXXX,*
5. *deleteXXX*

Cada operação CRUD será implementado em forma de função. É importante que os mesmos nomes de funções utilizado no *model* seja utilizada no *controller*. Isso facilita a vida do programador.

2.3.7 Configurando o módulo de *login*

Dentro do *apps/login* faça:

2.3.7.1 Crie os diretórios:

```
mkdir controller
mkdir model
mkdir tests
```

2.3.7.2 Configurando o *model*

Dentro do diretório *apps/login/model* crie o arquivo *mdlLogin.js* com o seguinte conteúdo:

```
const db = require("../../../database/databaseconfig");

const GetCredencial = async (loginPar) => {
    return (
      await db.query(
        "select username, password " +
          "from usuarios where username = $1 and deleted = false", [loginPar]
      )
    ).rows;
  };
```

```
11
12 module.exports = {
13     GetCredencial,
14   };
```

2.3.7.3 Configurando o *controller*

Dentro do diretório *apps/login/controller* crie o arquivo *ctlLogin.js* com o seguinte conteúdo:

```
1  const jwt = require("jsonwebtoken");
2  const bCrypt = require("bcryptjs");
3  const mdlLogin = require("../model/mdlLogin");
4
5  const Login = async (req, res, next) => {
6    const credencial = await mdlLogin.GetCredencial(req.body.username);
7    if (credencial.length == 0) {
8      return res.status(200).json({ message: "Usuário não identificado!" });
9    }
10
11   if (bCrypt.compareSync(req.body.password, credencial[0].password)) {
12     //auth ok
13     const username = credencial[0].username;
```

```
14      const token = jwt.sign({ username }, process.env.SECRET_API, {
15        expiresIn: 600, // expires in 10min
16      });
17      return res.json({ auth: true, token: token });
18    }
19    res.status(200).json({ message: "Login inválido!" });
20  };
21
22  function AutenticaJWT(req, res, next) {
23    const tokenHeader = req.headers["authorization"];
24    if (!tokenHeader)
25      return res
26        .status(200)
27        .json({ auth: false, message: "Não foi informado o token JWT" });
28
29    const bearer = tokenHeader.split(" ");
30    const token = bearer[1];
31
32    jwt.verify(token, process.env.SECRET_API, function (err, decoded) {
33      if (err)
34        return res
```

```
35          .status(200)
36          .json({ auth: false, message: "JWT inválido ou expirado" });
37
38      req.userId = decoded.id;
39      next();
40    });
41  }
42
43  const Logout = (req, res, next) => {
44    res.json({ auth: false, token: null });
45  };
46
47  module.exports = {
48    Login,
49    Logout,
50    AutenticaJWT,
51  };
```

2.3.7.4 Configurando o *test*

Dentro do diretório *apps/login/tests* crie o arquivo *testLogin.rest* com o seguinte conteúdo:

```
### teste de login
POST http://localhost:40000/Login HTTP/1.1
content-type: application/json
//Authorization: Bearer eyJhbGciOiJIUzI

{
    "username": "qwe",
    "password": "qwe"
}

### teste de logout
POST http://localhost:40000/Logout HTTP/1.1
content-type: application/json
//Authorization: Bearer eyJhbGciOiJIUzI1NiI
{

}
```

2.3.7.5 Configure as rotas

Abra o arquivo routes/router.js e o altere para ficar assim:

```
1  const express = require("express");
2  const routerApp = express.Router();
3
4  const appLogin = require("../apps/login/controller/ctlLogin");
5
6  // middleware that is specific to this router
7  routerApp.use((req, res, next) => {
8    next();
9  });
10
11 routerApp.get("/", (req, res) => {
12   res.send("Olá mundo!");
13 });
14
15 //Rotas de Alunos
16
17 //Rotas de Cursos
18
19 // Rota Login
20 routerApp.post("/Login", appLogin.Login);
21 routerApp.post("/Logout", appLogin.Logout);
```

```
22
23 module.exports = routerApp;
```

2.3.7.6 Execute os testes

Após ter implementado os códigos, execute o servidor e faça os testes:

```
1 # No console para iniciar o servidor
2 node app.js
3 # No VSCode para fazer os testes
4 No arquivo de testes do módulo login, clique com direito sobre um teste e escolha a opção
    Send Request. Analise a resposta.
```

Você deve testar cada uma das rotinas até verificar se está livre de erros. Quanto mais testes, menos chances de erros passarem. **Lembre-se**: arrumar *bug* enquanto o *software* está em desenvolvimento é MENOS custoso do que arrumar *bug* após o *software* entrar em produção.

2.3.8 Configurando o módulo de *aluno*

O módulo de *aluno* está completo no repositório do *github*. Para seu desenvolvimento, é necessário implementar os 6 passos descritos no módulo de *login*. Fique atendo para o nome dos diretórios e arquivos.

2.3.8.1 Crie os diretórios

Dentro do diretório *apps/alunos*, você deve criar os 3 diretórios *controller, model* e *tests*.

2.3.8.2 Configurando o model

Você deve criar o arquivo *apps/alunos/**model/mdl**Alunos.js* cujo conteúdo está no *github*.

2.3.8.3 Configurando o controller

Você deve criar o arquivo *apps/alunos/**controller/ctl**Alunos.js* cujo conteúdo está no *github*.

2.3.8.4 Configurando o test

Você deve criar o arquivo *apps/alunos/**tests/test**Alunos.rest* cujo conteúdo está no *github*.

2.3.8.5 Configurando as rotas

Nesta etapa o arquivo *routes/router.js* deverá ficar assim:

```
1 const express = require("express");
2 const routerApp = express.Router();
3
4 const appAlunos = require("../apps/alunos/controller/ctlAlunos");
5 const appLogin = require("../apps/login/controller/ctlLogin");
```

```
6
7  // middleware that is specific to this router
8  routerApp.use((req, res, next) => {
9    next();
10 });
11
12 routerApp.get("/", (req, res) => {
13   res.send("Olá mundo!");
14 });
15
16 //Rotas de Alunos
17 routerApp.get("/getAllAlunos", appAlunos.getAllAlunos);
18 routerApp.post("/getAlunoByID", appLogin.AutenticaJWT, appAlunos.getAlunoByID);
19 routerApp.post("/insertAlunos", appLogin.AutenticaJWT, appAlunos.insertAlunos);
20 routerApp.post("/updateAlunos", appAlunos.updateAlunos);
21 routerApp.post("/DeleteAlunos", appAlunos.DeleteAlunos);
22
23 //Rotas de Cursos
24
25 // Rota Login
26 routerApp.post("/Login", appLogin.Login);
```

```
27 routerApp.post("/Logout", appLogin.Logout);
28
29 module.exports = routerApp;
```

2.3.8.6 Execute os testes

Execute os testes conforme orientação feita no módulo de *login*.

2.3.9 Configurando o módulo de curso

O módulo de *curso* está completo no repositório do *github*. Para seu desenvolvimento, é necessário implementar os 6 passos descritos no módulo de *login*. Fique atendo para o nome dos diretórios e arquivos.

2.3.9.1 Crie os diretórios

Dentro do diretório *apps/cursos*, você deve criar os 3 diretórios como criado no módulo de login.

2.3.9.2 Configurando o model

Você deve criar o arquivo *apps/cursos/**model/mdl**Cursos.js* cujo conteúdo está no *github*.

2.3.9.3 Configurando o controller

Você deve criar o arquivo *apps/cursos/**controller/ctl**Cursos.js* cujo conteúdo está no *github*.

2.3.9.4 Configurando o test

Você deve criar o arquivo *apps/cursos/**tests/test**Cursos.rest* cujo conteúdo está no *github*.

2.3.9.5 Configurando as rotas

Nesta etapa o arquivo *routes/router.js* deverá ficar assim:

```
1  const express = require("express");
2  const routerApp = express.Router();
3
4  const appAlunos = require("../apps/alunos/controller/ctlAlunos");
5  const appCursos = require("../apps/cursos/controller/ctlCursos");
6  const appLogin = require("../apps/login/controller/ctlLogin");
7
8  // middleware that is specific to this router
9  routerApp.use((req, res, next) => {
10   next();
11 });
12
13 routerApp.get("/", (req, res) => {
14   res.send("Olá mundo!");
15 });
```

```
16
17 //Rotas de Alunos
18 routerApp.get("/getAllAlunos", appAlunos.getAllAlunos);
19 routerApp.post("/getAlunoByID", appLogin.AutenticaJWT, appAlunos.getAlunoByID);
20 routerApp.post("/insertAlunos", appLogin.AutenticaJWT, appAlunos.insertAlunos);
21 routerApp.post("/updateAlunos", appAlunos.updateAlunos);
22 routerApp.post("/DeleteAlunos", appAlunos.DeleteAlunos);
23
24 //Rotas de Cursos
25 routerApp.get("/GetAllCursos", appCursos.GetAllCursos);
26 routerApp.post("/GetCursoByID", appCursos.GetCursoByID);
27 routerApp.post("/InsertCursos", appCursos.InsertCursos);
28 routerApp.post("/UpdateCursos", appCursos.UpdateCursos);
29 routerApp.post("/DeleteCursos", appCursos.DeleteCursos);
30
31 // Rota Login
32 routerApp.post("/Login", appLogin.Login);
33 routerApp.post("/Logout", appLogin.Logout);
34
35 module.exports = routerApp;
```

2.3.9.6 Execute os testes

Execute os testes conforme orientação feita no módulo de *login.*

2.4 Exercícios

No arquivo de *script* banco de dados desta disciplina, há a criação de duas tabelas clientes e pedidos. Implemente um servidor *back-end* para elas.

2.5 Finalizando

Vimos como criar um servidor para rodar como *back-end*. No próximo capítulo, iremos trabalhar com servidor *front-end*.

Capítulo 3

Implementação do Servidor *Front-End*

3.1 Introdução

Nesse capítulo, iremos estudar o desenvolvimento de software no papel de *front-end*. Como dito, *front-end* é um computador que **interage diretamente com o usuário**. Normalmente, é um **computador pessoal ou um dispositivo móvel**.

Para o desenvolvimento do sistema exemplo, iremos usar o Node.js e alguns *frameworks* que facilitam a tarefa do programador na implementação de códigos no *front-end*. Iremos **implementar os formulários das operações** de *Create*, *Read*, *Update* e *Delete* (**CRUD**). E ao término do capítulo, a interface do sistema executando e interagindo-se com o servidor *back-end*.

3.2 Configuração inicial

Vamos, agora, fazer a configuração inicial de nosso servidor *front-end*.

3.2.1 Instalando as bibliotecas necessárias

Para desenvolver o servidor *front-end*, iremos instalar, inicialmente, o *framework express-generator*:

```
1 sudo npm i -g express-generator
```

Terminada a instalação, vamos criar nossa aplicação *front-end* cujo diretório será *dw3frontNode.* Para isso, dentro do diretório *dw3* digite o comando:

```
1 express -e --view=vash dw3frontNode
```

onde *vash* é um gerenciador de *template* cuja sintaxe é baseada em *Razor* da plataforma *dot.net*. O sítio eletrônico do *vash* é *https://github.com/kirbysayshi/vash* .

Terminada a instalação, entre no diretório *dw3frontNode*, entre no diretório com o comando *cd* e dê o comando *ls* e verifique os diretórios criados pelo *express-generator*:

```
1 cd dw3frontNode
2 ls
3 # deveremos ver os seguintes arquivos e diretórios
4 bin/
5 public/
```

```
routes/
views/
app.js
package.json
```

Apague o diretório *public/stylesheets*.

Em seguida, analise o conteúdo de cada um dos diretórios.

3.2.2 Instalando mais bibliotecas

Após analisar o conteúdo de cada diretório, vamos instalar as bibliotecas restantes:

```
1 npm i express-session axios dotenv body-parser moment http-errors
```

3.2.3 Arquivo *env* para configuração do servidor

Vamos criar um arquivo *.env* para configurar nosso servidor:

1. Crie um arquivo *.env* com o seguinte conteúdo:

```
1 PORT=40100
2 ## A variável servidor_dw3 é guarda o endereço do servidor back-end.
3 SERVIDOR_DW3="http://localhost:40000"
```

3.2.4 Arquivo App.js inicial

Apos instalar as bibliotecas, deixe o arquivo *app.js* com o seguinte conteúdo

```
1  //app.js
2  // DEBUG=dw3frontnode:* npm start
3
4  var createError = require('http-errors');
5  var express = require('express');
6  var path = require('path');
7  var cookieParser = require('cookie-parser');
8  var logger = require('morgan');
9
10 const session = require('express-session');
11 require('dotenv').config();
12
13 var indexRouter = require('./routes/index');
14
15
16 var app = express();
17
18 // view engine setup
19 app.set('views', path.join(__dirname, 'views'));
```

```
20 app.set('view engine', 'vash');
21
22 app.use(logger('dev'));
23 app.use(express.json());
24 app.use(express.urlencoded({ extended: false }));
25 app.use(cookieParser());
26
27 app.use(
28   session({
29     secret: "palavrasecreta",
30     resave: false,
31     saveUninitialized: true,
32     cookie: { maxAge: null },
33   })
34 );
35
36 app.use(express.static(path.join(__dirname, 'public')));
37
38 app.use('/', indexRouter);
39
40
```

```
41 // catch 404 and forward to error handler
42 app.use(function(req, res, next) {
43   next(createError(404));
44 });
45
46 // error handler
47 app.use(function(err, req, res, next) {
48   // set locals, only providing error in development
49   res.locals.message = err.message;
50   res.locals.error = req.app.get('env') === 'development' ? err : {};
51
52   // render the error page
53   res.status(err.status || 500);
54   res.render('error');
55 });
56
57 module.exports = app;
```

3.2.5 Executando o servidor *front-end*

Para rodar nosso servidor, faça:

1. Execute o comando:

```
DEBUG=dw3frontnode:* npm start
```

3.2.6 Finalizando a criação de diretórios

Uma vez que o servidor *front-end* executou sem erros, vamos criar a estrutura de diretórios. Devemos criar os seguintes diretórios:

```
mkdir app
```

Assim, no final teremos a seguinte estrutura de diretórios:

```
app/ ## aqui colocaremos as regras de validação dos módulos login, alunos e cursos.
bin/ ## não vamos mexer aqui
public/ ## aqui colocaremos os arquivo CSS e JS
routes/ ## aqui colocaremos as rotas de nosso sistema separadas em arquivos distintos.
views/ ## aqui colocaremos os formulários em html dos módulos login, alunos e cursos.
```

3.3 Implementando *templates* necessários para o sistema

Vamos criar diretório de bibliotecas *css* e *js* dentro do *public* e o diretório de *views*.

3.3.1 Diretórios de bibliotecas *public*

Antes de criarmos os *templates* do sistema, vamos baixar arquivos *css* e *js* necessários para a execução do site. O diretório *public* é o responsável por guardar as bibliotecas adicionais.

3.3.1.1 Diretório *public/js*

Crie dentro do diretório *raiz-do-projeto/public* o diretório *js*:

```
1 mkdir public/js
2 cd public/js
```

Dentro deste diretório crie **5.3.0**:

```
1 mkdir 5.3.0
```

dentro dele, salve o arquivo

- *bootstrap.min.js*
- *bootstrap.min.js.map*
- *popper.js_2.11.8_popper.min.js*

disponíveis no repositório remoto da disciplina no *Github*. Vamos pegar de lá para manter a compatibilidade das versões.

IMPORTANTE: para salvar as biliotecas, entre dentro dela no Github e clique no **botão download**. Se salvar clicando com o botão direito do mouse não funciona.

Agora, crie o diretório ***js/datatables***

```
1 mkdir js/datatables
```

e copie os arquivos

- *html5-2.4.1_datatables.min.js*
- *moment_2.29.2_moment.min.js*
- *pdfmake_0.2.7_pdfmake.min.js*
- *pdfmake_0.2.7_vfs_fonts.js*

do *Github* para cá.

Crie o diretório ***js/fontawesome***

```
1 mkdir js/fontawesome
```

e copie o arquivo

- *v6.3.0_js_all.js*

do *Github* para cá.

Por fim, copie para ***js*** os arquivo

- *axios.min.js*
- *jquery.maskMoney.min.js*
- *scripts.js*

do *Github* para cá.

3.3.1.2 Diretório *public/css*

Crie dentro do diretório *raiz-do-projeto/public* o diretório css:

```
1 mkdir public/css
2 cd public/css
```

Dentro deste diretório crie **5.3.0**:

```
1 mkdir 5.3.0
```

dentro dele, salve o arquivo

- *bootstrap.min.css*

disponíveis no repositório remoto da disciplina no *Github*. Vamos pegar de lá para manter a compatibilidade das versões.

Agora, crie o diretório **css/*datatables***

```
1 mkdir js/datatables
```

e copie os arquivos

- *datatables.min.css*

do *Github* para cá.

Por fim, copie para ***css*** o arquivo

- *styles.css*

do *Github* para cá.

3.3.2 Diretório *views*

Dentro do diretório *raiz-do-projeto/views*, crie os seguintes arquivos com seus respectivos conteúdos:

raiz-do-projeto/views/header.vash

```
1  <!--views/header.vash -->
2  <head>
3      <meta charset="utf-8" />
4      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
5      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"
       />
6      <meta name="description" content="" />
7      <meta name="author" content="" />
8      <title>@model.title</title>
9      <link href="/css/styles.css" rel="stylesheet" />
10     <link href="/css/datatables/datatables.min.css" rel="stylesheet">
11 </head>
```

raiz-do-projeto/views/footer.vash

```
<!--views/footer.vash -->
<footer class="py-4 bg-light mt-auto">
    <div class="container-fluid px-4">
        <div class="d-flex align-items-center justify-content-between small">
            <div class="text-muted">Copyright &copy; Your Website 2023</div>
            <div>
                <a href="#">Privacy Policy</a>
                &middot;
                <a href="#">Terms &amp; Conditions</a>
            </div>
        </div>
    </div>
</footer>
```

Para o módulo de login isso é o suficiente, porém vamos terminar de criar os *templates* necessários e não precisar voltar mais nesta tarefa.

raiz-do-projeto/views/layout.vash

```
1  <!-- views/layout.vash-->
2  @html.include("header")
3
4  <body class="sb-nav-fixed">
5      <nav class="sb-topnav navbar navbar-expand navbar-dark bg-dark">
6          <!-- Navbar Brand-->
7          <a class="navbar-brand ps-3" href="index.html">Start Bootstrap</a>
8          <!-- Sidebar Toggle-->
9          <button class="btn btn-link btn-sm order-1 order-lg-0 me-4 me-lg-0" id="
       sidebarToggle" href="#!"><i
10                 class="fas fa-bars"></i></button>
11         <!-- Navbar Search-->
12         <form class="d-none d-md-inline-block form-inline ms-auto me-0 me-md-3 my-2 my-md
       -0">
13             <div class="input-group">
14
15             </div>
16         </form>
17         <!-- Navbar-->
18         <ul class="navbar-nav ms-auto ms-md-0 me-3 me-lg-4">
```

```
19          <li class="nav-item dropdown">
20              <a class="nav-link dropdown-toggle" id="navbarDropdown" href="#" role="
    button" data-bs-toggle="dropdown"
21                  aria-expanded="false"><i class="fas fa-user fa-fw"></i></a>
22              <ul class="dropdown-menu dropdown-menu-end" aria-labelledby="
    navbarDropdown">
23                  <li>
24                      <hr class="dropdown-divider" />
25                  </li>
26                  <li><a class="dropdown-item" href="/Logout">Logout</a></li>
27              </ul>
28          </li>
29      </ul>
30   </nav>
31   <div id="layoutSidenav">
32       <div id="layoutSidenav_nav">
33           <nav class="sb-sidenav accordion sb-sidenav-dark" id="sidenavAccordion">
34               <div class="sb-sidenav-menu">
35                   <div class="nav">
36                       <div class="sb-sidenav-menu-heading">Principal</div>
37                       <a class="nav-link" href="/">
```

```
38                          <div class="sb-nav-link-icon"><i class="fas fa-tachometer-alt
    "></i></div>
39                          Home
40                      </a>
41                      <div class="sb-sidenav-menu-heading">Módulos</div>
42
43                      <a class="nav-link collapsed" href="#" data-bs-toggle="collapse"
    data-bs-target="#collapsePages"
44                          aria-expanded="false" aria-controls="collapsePages">
45                          <div class="sb-nav-link-icon"><i class="fas fa-book-open"></i
    ></div>
46                          Acadêmico
47                          <div class="sb-sidenav-collapse-arrow"><i class="fas fa-angle
    -down"></i></div>
48                      </a>
49                      <div class="collapse" id="collapsePages" aria-labelledby="
    headingTwo"
50                          data-bs-parent="#sidenavAccordion">
51                          <nav class="sb-sidenav-menu-nested nav accordion" id="
    sidenavAccordionPages">
52                              <a class="nav-link collapsed" href="#" data-bs-toggle="
```

```
    collapse"
53                                  data-bs-target="#pagesCollapseAuth" aria-expanded="
    false"
54                                  aria-controls="pagesCollapseAuth">
55                                  Alunos
56                                  <div class="sb-sidenav-collapse-arrow"><i class="fas
    fa-angle-down"></i></div>
57                              </a>
58                              <div class="collapse" id="pagesCollapseAuth" aria-
    labelledby="headingOne"
59                                  data-bs-parent="#sidenavAccordionPages">
60                                  <nav class="sb-sidenav-menu-nested nav">
61                                      <a class="nav-link" href="/alunos">Manutenção de
    alunos</a>
62                                  </nav>
63                              </div>
64                              <a class="nav-link collapsed" href="#" data-bs-toggle="
    collapse"
65                                  data-bs-target="#pagesCollapseError" aria-expanded="
    false"
66                                  aria-controls="pagesCollapseError">
```

```
67                                  Cursos
68                                  <div class="sb-sidenav-collapse-arrow"><i class="fas
    fa-angle-down"></i></div>
69                              </a>
70                              <div class="collapse" id="pagesCollapseError" aria-
    labelledby="headingOne"
71                                  data-bs-parent="#sidenavAccordionPages">
72                                  <nav class="sb-sidenav-menu-nested nav">
73                                      <a class="nav-link" href="/cursos">Manutenção de
    cursos</a>
74                                  </nav>
75                              </div>
76                          </nav>
77                      </div>
78
79                  </div>
80              </div>
81              <div class="sb-sidenav-footer">
82                  <div class="small">Bem vindo:</div>
83                  @model.userName
84              </div>
```

```
85              </nav>
86          </div>
87
88          <!-----Início Páginas filhas ----->
89          <div id="layoutSidenav_content">
90              <!-----Início Páginas filhas ----->
91              @html.block('content')
92              <!-----Fim Páginas filhas ----->
93
94              @html.include('footer')
95          </div>
96
97          <!-----Fim Páginas filhas ----->
98      </div>
99
100     <script src="/js/5.3.0/popper.js_2.11.8_popper.min.js" ></script>
101     <script src="/js/5.3.0/bootstrap.min.js"></script>
102     <script src="/js/scripts.js"></script>
103     <script src="/js/fontawesome/v6.3.0_js_all.js" crossorigin="anonymous"></script>
104     <script src="/js/datatables/pdfmake_0.2.7_pdfmake.min.js"></script>
105     <script src="/js/datatables/pdfmake_0.2.7_vfs_fonts.js"></script>
```

```
106     <script src="/js/datatables/html5-2.4.1_datatables.min.js"></script>
107     <script src="/js/datatables/moment_2.29.2_moment.min.js"></script>
108     <script src="/js/jquery.maskMoney.min.js"></script>
109     <script src="/js/axios.min.js"></script>
110 </body>
```

3.4 Implementando o módulo de *Login*

Inicialmente, vamos implementar o módulo *front-end* de login.

Para isso, vamos deixar o conteúdo do arquivo *routes/index.js* da seguinte forma:

```
1  // Arquivo routes/index.js
2
3  var express = require('express');
4  var loginApp = require("../app/login/controller/ctlLogin")
5  var router = express.Router();
6
7  //Função necessária para evitar que usuários não autenticados acessem o sistema.
8  function authenticationMiddleware(req, res, next) {
9      // Verificar se existe uma sessão válida.
10     isLogged = req.session.isLogged;
11
12     if (!isLogged) {
13       res.redirect("/Login");
14     }
15     next();
16 };
17
```

```
18 /* GET home page. */
19 router.get('/', authenticationMiddleware, function (req, res, next) {
20     userName = req.session.userName;
21     res.render('index', { "title": 'Página principal', "userName": userName});
22
23 });
24
25 /* GET login page. */
26 router.get('/Login', loginApp.Login);
27
28 /* POST login page. */
29 router.post('/Login',loginApp.Login);
30
31 /* GET logout page. */
32 router.get('/Logout', loginApp.Logout);
33
34 module.exports = router;
```

3.4.1 Criando o *Controller* de *Login*

Crie os diretórios:

```
1 mkdir app/login
2 cd app/login
3 mkdir controller
```

Em seguida, vamos criar o arquivo *ctlLogin.js* em *app/login/controller*

Deixe o conteúdo dele assim:

```
1  //Arquivo app/login/controller/ctlLogin.js
2
3  axios = require("axios");
4
5  const Login = async (req, res) => {
6    let resp;
7
8    if (req.method == "POST" && req.body.username !== "" ) {
9      console.log("[ctlLogin.js] Valor ServidorDW:", process.env.SERVIDOR_DW3);
10     resp = await axios.post(process.env.SERVIDOR_DW3 + "/Login", {
11       username: req.body.username,
12       password: req.body.password
13     });
14     if (!resp.data.auth) {
15         res.render("login/login", {
16         title: "Login",
17         message: resp.data.message
18       });
19     } else {
```

```
20        session = req.session;
21        session.isLogged = true;
22        session.userName = req.body.username;
23        session.token = resp.data.token;
24        return res.redirect("/");
25      }
26    }
27    res.render("login/login", { title: "Login", message: "", });
28  }
29
30  function Logout(req, res) {
31    session = req.session;
32    session.isLogged = false;
33    session.token = false;
34    res.redirect("/Login");
35  }
36  module.exports = {
37    Login,
38    Logout,
39  };
```

3.4.2 Criando a *View* de *Login*

No diretório *raiz-do-projeto/views*, crie o diretório *login*:

```
1 mkdir ./views/login
```

Crie o arquivo *login.vash* com o conteúdo:

```
1  <!--Arquivo views/login.vash-->
2
3  <!DOCTYPE html>
4  <html lang="en">
5  @html.include("header")
6
7  <body>
8      <div class="login ">
9          <div class="card mx-auto col-lg-4 mt-5" style="background-color:lightblue;">
10             <div class="card mx-auto col-10  mt-5 mb-5">
11                 <div class="card-body">
12                     <h4 class="card-title" style="text-align: center;">Login</h4>
13                     <form class="form-login" action="Login" method="POST">
14                         <label for="labelusername" class="form-label mt-4">Informe seu
    usuário</label>
15                         <input type="text" name="username" class="form-control" /><br />
```

```
16                          <label for="labelpassword" class="form-label mt-2">Informe seu
    password</label>
17                          <input type="password" name="password" class="form-control" /><br
     />
18                          <div class="text-center">
19                              <input class="btn btn-primary mt-4 mx-auto" type="submit"
    value="Login" />
20                          </div>
21
22                          @if(model.message) {
23                          <p>
24                              <label style="color:Red"> @model.message </label>
25                          </p>
26                          }
27                      </form>
28                  </div>
29              </div>
30          </div>
31      </div>
32      <script src="js/5.3.0/bootstrap.min.js" crossorigin="anonymous"></script>
33  </body>
```

```
34 </html>
```

Pronto, já podemos rodar o teste de login. Para isso digite na raiz-do-projeto

```
1 DEBUG=dw3frontnode:* npm start
```

3.5 Implementando o módulo de Alunos

Vamos implementar o módulo de alunos

A regra geral é:

1. Crie o *controller.*
2. Crie a *rota.*
3. Crie a *view.*

3.5.1 Criando o *Controller* de *Alunos*

Crie os diretórios:

```
1 mkdir app/alunos
2 cd app/alunos
3 mkdir controller
```

Em seguida, vamos criar o arquivo *ctl*Alunos.*js* em *app*/alunos/*controller*

Deixe o conteúdo dele assim:

```
1  //Arquivo app/alunos/controller/ctlAlunos.js
2
3  const axios = require("axios");
4  const moment = require("moment");
5
6  //@ Abre o formulário de manutenção de alunos
7  const getAllAlunos = (req, res) =>
8    (async () => {
9      userName = req.session.userName;
10     try {
11       resp = await axios.get(process.env.SERVIDOR_DW3 + "/getAllAlunos", {});
12       //console.log("[ctlLogin.js] Valor resp:", resp.data);
13       res.render("alunos/view_manutencao", {
14         title: "Manutenção de alunos",
15         data: resp.data,
16         userName: userName,
17       });
18     } catch (erro) {
19       console.log("[ctlAlunos.js|getAllAlunos] Try Catch:Erro de requisição");
```

```
20      }
21    })();
22
23  //@ Função para validar campos no formulário
24  function validateForm(regFormPar) {
25    //@ *** Regra de validação
26    //@ Como todos os campos podem ter valor nulo, vou me preocupar
27    //@ com campo datanascimento. Caso ele tenha valor "", vou atribuir null a ele.
28
29    if (regFormPar.datanascimento == "") {
30      regFormPar.datanascimento = null;
31    }
32
33    return regFormPar;
34  }
35
36  //@ Abre e faz operações de CRUD no formulário de cadastro de alunos
37  const insertAlunos = (req, res) =>
38    (async () => {
39      var oper = "";
40      var registro = {};
```

```
41      var cursos = {};
42      userName = req.session.userName;
43      token = req.session.token;
44      try {
45        if (req.method == "GET") {
46          oper = "c";
47          cursos = await axios.get(
48            process.env.SERVIDOR_DW3 + "/GetAllCursos",
49            {}
50          );
51          //console.log("[crlAlunos|insertAlunos] valor de cursos:", cursos.data.registro);
52          registro = {
53            alunoid: 0,
54            prontuario: "",
55            nome: "",
56            endereco: "",
57            rendafamiliar: "0.00",
58            datanascimento: "",
59            cursoid: 0,
60            deleted: false,
61          };
```

```
62
63          res.render("alunos/view_cadAlunos", {
64            title: "Cadastro de alunos",
65            data: registro,
66            curso: cursos.data.registro,
67            oper: oper,
68            userName: userName,
69          });
70        } else {
71          oper = "c";
72          const alunoREG = validateForm(req.body);
73          resp = await axios.post(
74            process.env.SERVIDOR_DW3 + "/insertAlunos",
75            {
76              alunoid: 0,
77              prontuario: alunoREG.prontuario,
78              nome: alunoREG.nome,
79              endereco: alunoREG.endereco,
80              rendafamiliar: alunoREG.rendafamiliar,
81              datanascimento: alunoREG.datanascimento,
82              cursoid: alunoREG.cursoid,
```

```
83            deleted: false,
84          },
85          {
86            headers: {
87              "Content-Type": "application/json",
88              Authorization: "Bearer " + token,
89            },
90          }
91        );
92
93        console.log("[ctlAlunos|insertAlunos] resp:", resp.data);
94        if (resp.data.status == "ok") {
95          registro = {
96            alunoid: 0,
97            prontuario: "",
98            nome: "",
99            endereco: "",
100           rendafamiliar: "0.00",
101           datanascimento: "",
102           cursoid: 0,
103           deleted: false,
```

```
104           };
105         } else {
106           registro = alunoREG;
107         }
108         cursos = await axios.get(
109           process.env.SERVIDOR_DW3 + "/GetAllCursos",
110           {}
111         );
112         oper = "c";
113         res.render("alunos/view_cadAlunos", {
114           title: "Cadastro de alunos",
115           data: registro,
116           curso: cursos.data.registro,
117           oper: oper,
118           userName: userName,
119         });
120       }
121     } catch (erro) {
122       console.log(
123         "[ctlAlunos.js|insertAlunos] Try Catch: Erro não identificado",
124         erro
```

```
125       );
126     }
127   })();
128
129 //@ Abre o formulário de cadastro de alunos para futura edição
130 const viewAlunos = (req, res) =>
131   (async () => {
132     var oper = "";
133     var registro = {};
134     var cursos = {};
135     userName = req.session.userName;
136     token = req.session.token;
137     try {
138       if (req.method == "GET") {
139         const id = req.params.id;
140         oper = req.params.oper;
141
142         parseInt(id);
143         resp = await axios.post(
144           process.env.SERVIDOR_DW3 + "/getAlunoByID",
145           {
```

```
146             alunoid: id,
147           },
148           {
149             headers: {
150               "Content-Type": "application/json",
151               Authorization: "Bearer " + token,
152             },
153           }
154         );
155
156         if (resp.data.status == "ok") {
157           registro = resp.data.registro[0];
158           registro.datanascimento = moment(registro.datanascimento).format(
159             "YYYY-MM-DD"
160           );
161           cursos = await axios.get(
162             process.env.SERVIDOR_DW3 + "/GetAllCursos",
163             {}
164           );
165           console.log("[ctlAlunos|viewAlunos] GET oper:", oper);
166
```

```
167           res.render("alunos/view_cadAlunos", {
168             title: "Cadastro de alunos",
169             data: registro,
170             curso: cursos.data.registro,
171             oper: oper,
172             userName: userName,
173           });
174         }
175       } else {
176         // Código vai entrar quando o usuário clicar no botão Alterar e requisição for
    POST
177         oper = "vu";
178         console.log("[ctlAlunos|viewAlunos] POST oper:", oper);
179         const alunoREG = validateForm(req.body);
180         console.log("[ctlAlunos|viewAlunos] POST id:", alunoREG.id);
181         const id = parseInt(alunoREG.id);
182         resp = await axios.post(
183           process.env.SERVIDOR_DW3 + "/updateAlunos",
184           {
185             alunoid: id,
186             prontuario: alunoREG.prontuario,
```

```
187             nome: alunoREG.nome,
188             endereco: alunoREG.endereco,
189             rendafamiliar: alunoREG.rendafamiliar,
190             datanascimento: alunoREG.datanascimento,
191             cursoid: alunoREG.cursoid,
192             deleted: false,
193           },
194           {
195             headers: {
196               "Content-Type": "application/json",
197               Authorization: "Bearer " + token,
198             },
199           }
200         );
201
202         if (resp.data.status == "ok") {
203           res.json({ status: "ok" });
204         } else {
205           res.json({ status: "erro" });
206         }
207       }
```

```
208     } catch (erro) {
209       res.json({ status: "[ctlAlunos.js|viewAlunos] Aluno não pode ser alterado" });
210       console.log(
211         "[ctlAlunos.js|viewAlunos] Try Catch: Erro não identificado",
212         erro
213       );
214     }
215   })();
216
217 //@ Remove o aluno selecionado
218 const DeleteAlunos = (req, res) =>
219   (async () => {
220     var oper = "";
221     userName = req.session.userName;
222     token = req.session.token;
223     try {
224       oper = "v";
225       const id = parseInt(req.body.id);
226
227       resp = await axios.post(
228         process.env.SERVIDOR_DW3 + "/DeleteAlunos",
```

```
229         {
230           alunoid: id,
231         },
232         {
233           headers: {
234             "Content-Type": "application/json",
235             Authorization: "Bearer " + token,
236           },
237         }
238       );
239
240       if (resp.data.status == "ok") {
241         res.json({ status: "ok" });
242       } else {
243         res.json({ status: "erro" });
244       }
245     } catch (erro) {
246       console.log(
247         "[ctlAlunos.js|DeleteAlunos] Try Catch: Erro não identificado",
248         erro
249       );
```

```
250     }
251   })();
252
253 module.exports = {
254   getAllAlunos,
255   viewAlunos,
256   insertAlunos,
257   DeleteAlunos,
258 };
```

3.5.2 Criando as Rotas de *Alunos*

Dentro do diretório *routes/* crie o arquivo *rte_alunos.js* com o seguinte conteúdo:

```
1 //Arquivo routes/rte_alunos.js
2
3 var express = require('express');
4 var alunosApp = require("../app/alunos/controller/ctlAlunos")
5
6 var router = express.Router();
7
8 //Função necessária para evitar que usuários não autenticados acessem o sistema.
```

```
9  function authenticationMiddleware(req, res, next) {
10     // Verificar se existe uma sessão válida.
11     isLogged = req.session.isLogged;
12
13     if (!isLogged) {
14       res.redirect("/Login");
15     }
16     next();
17 };
18
19 /* GET métodos */
20 router.get('/', authenticationMiddleware, alunosApp.getAllAlunos);
21 router.get('/insertAlunos', authenticationMiddleware, alunosApp.insertAlunos);
22 router.get('/viewAlunos/:id/:oper', authenticationMiddleware, alunosApp.viewAlunos);
23
24 /* POST métodos */
25 router.post('/insertAlunos', authenticationMiddleware, alunosApp.insertAlunos);
26 router.post('/DeleteAlunos', authenticationMiddleware, alunosApp.DeleteAlunos);
27 router.post('/viewAlunos', authenticationMiddleware, alunosApp.viewAlunos);
28
29
```

```
30 module.exports = router;
```

No arquivo *app.js*, logo abaixo da linha *var indexRouter = require('./routes/index')*, acrescente a linha:

```
1 // Arquivo app.js
2 ...
3 var indexRouter = require('./routes/index');
4 var alunosRouter = require('./routes/rte_alunos');
```

Ainda no arquivo *app.js*, logo abaixo da linha *app.use('/', indexRouter)*, acrescente a linha:

```
1 // Arquivo app.js
2 ...
3 app.use('/', indexRouter);
4 app.use('/alunos', alunosRouter);
```

3.5.3 Criando as Visões de Alunos

Entre no diretório *views/* e o diretório *alunos/*:

```
1 cd views
2 mkdir alunos
```

Dentro do diretório *alunos/* crie os arquivos:

1. *views/alunos/view_manutencao.vash*

```
1  <!-- Arquivo views/alunos/view_manutencao.vash -->
2  @html.extend('layout', function(model){
3
4  @html.block('content', function(model){
5
6  <main>
7
8      <div class="container-fluid px-4">
9          <h1 class="mt-4">@model.title</h1>
10         <ol class="breadcrumb mb-4">
11             <li class="breadcrumb-item active">@model.title</li>
12         </ol>
13         <div class="row">
14             <div class="col-xl-12">
15                 <div class="card mb-4">
16                     <div class="col-2 ms-1 mt-1 mb-2">
17
18                         <button type="button" class="btn btn-primary"
19                             onclick="window.location.href = '/alunos/insertAlunos'">
```

```
    Novo aluno</button>
20                  </div>
21                  <div class="card-header">
22                      <i class="fas fa-table me-1"></i>
23                      Listagem de alunos
24                  </div>
25
26
27                  <div class="card-body">
28                      <table id="example" class="stripe" style="width:100%">
29                          <thead>
30                              <tr>
31                                  <th>Ação</th>
32                                  <th>Prontuário</th>
33                                  <th>Nome</th>
34                                  <th>Data Nascimento</th>
35                                  <th>Curso</th>
36                              </tr>
37                          </thead>
38                          <tbody>
39                              @model.data.registro.forEach(function(registro){
```

```
40                                  <tr>
41                                      <td><a href="/alunos/viewAlunos/@registro.
    alunoid/v" class="btn btn-info btn-sm"><i
42                                          class="fa fa-magnifying-glass"></i>
    Visualizar</a></td>
43                                      <td>@registro.prontuario</td>
44                                      <td>@registro.nome</td>
45                                      <td>@registro.datanascimento</td>
46                                      <td>@registro.descricao</td>
47                                  </tr>
48                                  })
49                              </tbody>
50                              <tfoot>
51                                  <tr>
52                                      <th>Ação</th>
53                                      <th>Prontuário</th>
54                                      <th>Nome</th>
55                                      <th>Data Nascimento</th>
56                                      <th>Curso</th>
57                                  </tr>
58                              </tfoot>
```

```
59                          </table>
60                      </div>
61                  </div>
62              </div>
63          </div>
64      </div>
65  </main>
66
67  })
68
69  })
70
71  <script>
72      var table = new DataTable('#example', {
73          dom: 'B<"top"fi>rt<"bottom"lp><"clear">',
74          columnDefs: [
75              {
76                  targets: 3,
77                  render: DataTable.render.datetime('DD/MM/YYYY')
78              }
79          ],
```

```
80          buttons: [
81              'copy', 'csv', 'pdf'
82          ],
83      });
84
85      table.buttons().container()
86          .appendTo($('.col-sm-12:eq(0)', table.table().container()));
87  </script>
```

2. *views/alunos/view_cadAlunos.vash*

```
1   <!-- Arquivo views/alunos/view_cadAlunos.vash -->
2   @html.extend('layout', function(model){
3
4   @html.block('content', function(model){
5
6
7   <main>
8
9       <div class="container-fluid px-4">
10          <h1 class="mt-2">@model.title</h1>
```

```
11          <ol class="breadcrumb mb-2">
12              <li class="breadcrumb-item active">@model.title</li>
13          </ol>
14          <div class="row">
15              <div class="col-xl-12">
16
17                  <div class="card-header">
18
19                  </div>
20
21                  <div class="card-body">
22                      <div class="col-xl-5">
23                          <form  id="form">
24                              <div class="mb-1" style="display: none">
25                                  <input type="text" name="id" value="@model.data.
    alunoid" class="form-control" id="id">
26                              </div>
27                              <div class="mb-1">
28                                  <label for="prontuario" class="form-label">
    Prontuário</label>
29                                  <input type="text" name="prontuario" value="@model.
```

```
    data.prontuario" class="form-control"
30                                  id="prontuario" @(model.oper=='v' ? 'disabled' :
     '' )>
31                          </div>
32
33                          <div class="mb-1">
34                              <label for="nome" class="form-label">Nome</label>
35                              <input type="text" name="nome" value="@model.data.
    nome" class="form-control" id="nome"
36                                  @(model.oper=='v' ? 'disabled' : '' )>
37                          </div>
38
39                          <div class="mb-1">
40                              <label for="endereco" class="form-label">Endereco</
    label>
41                              <input type="text" name="endereco" value="@model.
    data.endereco" class="form-control"
42                                  id="endereco" @(model.oper=='v' ? 'disabled' : '
    ' )>
43                          </div>
44
```

```
45                              <div class="mb-1">
46                                  <label for="rendafamiliar" class="form-label">Renda
    familiar</label>
47                                  <input type="text" name="rendafamiliar" value="
    @model.data.rendafamiliar"
48                                      class="form-control" id="rendafamiliar" data-
    thousands="." data-decimal=","
49                                      @(model.oper=='v' ? 'disabled' : '' )>
50                              </div>
51
52                              <div class="mb-1">
53                                  <label for="" class="form-label">Curso</label>
54                                  <select class="form-select" aria-label="Default
    select example" id="cursoid"
55                                      @(model.oper=='v' ? 'disabled' : '' ) name="
    cursoid">
56                                      @model.curso.forEach(function(item){
57                                      @if(model.data.cursoid === item.cursoid) {
58                                      <option value="@item.cursoid" selected>@item.
    descricao</option>
59                                      } else {
```

```
60                                  <option value="@item.cursoid">@item.descricao</
    option>
61                                  }
62                                  });
63                              </select>
64                          </div>
65
66                          <div class="mb-4">
67                              <label for="datanascimento" class="form-label">Data
    nascimento <span
68                                      style="color: red;">*</span></label>
69                              <input type="date" name="datanascimento" value="
    @model.data.datanascimento"
70                                  class="form-control" id="datanascimento" @(model
    .oper=='v' ? 'disabled' : '' )>
71                          </div>
72
73                          @if (model.oper=="c") {
74                          <button type="submit" class="btn btn-success me-2"
    onclick=""
75                              formmethod="POST">Salvar</button>
```

```
76                          }
77
78                      </form>
79                      <div>
80                          @if (model.oper=="vu") {
81                          <button type="button" class="btn btn-warning me-2 mb-3"
    onclick="alteraRegistro()">Salvar
82                              Alteração</button>
83                          }
84
85                          @if (model.oper=="v") {
86                          <button type="button" class="btn btn-warning me-2 mb-3"
87                              onclick="window.open('/alunos/viewAlunos/' + $('#id
    ').val() + '/vu', '_self')">Alterar</button>
88                          }
89
90                          @if (model.oper=="v") {
91                          <button type="" class="btn btn-danger mb-3" onclick="
    deleteAlunos()">Remover</button>
92                          }
93                      </div>
```

```
94                          <button type="button" class="btn btn-primary" onclick="
    window.location.href = '/alunos'">Fechar
95                              sem salvar</button>
96                      </div>
97                  </div>
98              </div>
99          </div>
100     </div>
101
102 </main>
103
104 })
105
106 })
107
108 <script>
109
110     $(function () {
111         $('#rendafamiliar').maskMoney();
112     })
113
```

```
114     $('#form').on('submit', function () {
115         if ($('#datanascimento').val() == "") {
116             alert("Informe a data de nascimento");
117             return false;
118         } else {
119             $('#rendafamiliar').val($('#rendafamiliar').maskMoney('unmasked')[0]);
120             return true;
121         }
122     });
123 
124     async function alteraRegistro() {
125         // event.preventDefault();
126         resp = await axios.post("/alunos/viewAlunos", {
127             id: $("#id").val(),
128             prontuario: $("#prontuario").val(),
129             nome: $("#nome").val(),
130             endereco: $("#endereco").val(),
131             rendafamiliar: $('#rendafamiliar').maskMoney('unmasked')[0], //Retira a
    formatação de money
132             datanascimento: $("#datanascimento").val(),
133             cursoid: $("#cursoid").val(),
```

```
134         }, {
135             headers: {
136                 'Content-Type': 'application/json',
137             }
138         });
139         //console.log("[view_cadAlunos.vash|alteraRegistro] valor resp:", resp.data.
    status);
140         if (resp.data.status == "ok") {
141             alert("Aluno alterado com sucesso!");
142         } else {
143             alert("Houve erro ao alterar os dados dos alunos!");
144         }
145     }
146
147     async function deleteAlunos(prontuarioPar) {
148         resp = await axios.post("/alunos/DeleteAlunos", {
149             id: $("#id").val(),
150         }, {
151             headers: {
152                 'Content-Type': 'application/json',
153             }
```

```
154         });
155
156         if (resp.data.status == "ok") {
157             alert("Aluno removido com sucesso!");
158             window.open("/alunos", "_self");
159         } else {
160             alert("Houve erro ao remover os dados dos alunos!");
161         }
162     }
163
164     async function testeFormData() {
165         const form = document.getElementById("form");
166         const formData = new FormData(form);
167         console.log("Prontuario: ", formData.get("prontuario"));
168     }
169 </script>
```

3.6 Implementando o módulo de Cursos

O desenvolvimento do módulo de *Cursos* possui a mesma sequência do de *Alunos*. Você deverá seguir a regra de:

1. Criar o *controller*.

2. Criar a *rota.*

3. Criar a *view.*

Porém, aqui, não iremos colocar o código fonte aqui, desta forma, você usará o código disponível no repositório remoto do *Github*.

3.6.1 Criando o *Controller* de *Cursos*

Crie os diretórios:

```
1 mkdir app/cursos
2 cd app/cursos
3 mkdir controller
```

Em seguida, vamos criar o arquivo *ctl*Cursos.*js* em *app*/cursos/*controller*
Deixe o conteúdo com o mesmo do *Github*.

3.6.2 Criando as Rotas de *Cursos*

Dentro do diretório *routes*/ crie o arquivo *rte*_cursos.*js* cujo conteúdo será o do Github.

No arquivo *app.js*, logo abaixo da linha *var alunosRouter = require('./routes/rte_alunos');*, acrescente a linha:

```
1 // Arquivo app.js
2 ...
3 var indexRouter = require('./routes/index');
4 var alunosRouter = require('./routes/rte_alunos');
5 var cursosRouter = require('./routes/rte_cursos');
```

Ainda no arquivo *app.js*, logo abaixo da linha *app.use('/alunos', alunosRouter);*, acrescente a linha:

```
1 // Arquivo app.js
2 ...
3 app.use('/', indexRouter);
4 app.use('/alunos', alunosRouter);
5 app.use('/cursos',  cursosRouter);
```

3.6.3 Criando as Visões de Alunos

Entre no diretório *views/* e o diretório cursos/:

```
1 cd views
2 mkdir cursos
```

Dentro do diretório cursos/ crie os arquivos:

1. *views/cursos/view_manutencao.vash*
   Cole o conteúdo do repositório remoto no *Github*.

2. *views/cursos/view_cadCursos.vash*
   Cole o conteúdo do repositório remoto no *Github*.

3.7 Finalizando

Vimos como criar um servidor de *Front-End*. Este modelo pode ser utilizado para o desenvolvimento de aplicações reais e favorece a divisão de responsabilidades entre profissionais desenvolvedores de APIs e profissionais responsáveis por projetar a interface do sistema. Este modelo deixa o desenvolvimento produtivo, pois separa as competências dos profissionais permitindo que eles possam se especializar em suas áreas.