ATIVIDADE - API E COMPONANTES BASICOS
ALUNO: Lucas Marques de Oliveira

01 - o CONTROLLER é responsável por controlar métodos que são as ações que o sistema faz, e ao criar rotas que se conectam com esses métodos o sistema funciona.

02 -

```php
<?php

namespace App\Models;
```

```php
use Illuminate\Database\Eloquent\Model;
```

```php
class Produto extends Model
{
    protected $table = 'produto';
```

```php
    protected $fillable = ['nome', 'valor'];
}
```

CONTROLLER:

```php
public function index()
    {
        $produto = Produto::all();
        return response()->json($produto, 201);
    }
```

ROTA:

```php
Route::get('produto/listar', [ProdutoController::class,
'index'])->name('produto.listar');
```

03 - Route Model Bindig permite a vinculação automática de modelos Eloquent às rotas. Isso significa que, ao definir uma rota, você pode especificar um modelo que será automaticamente recuperado com base no identificador fornecido na URL.

Ex: Considere uma rota definida da seguinte forma: Route::get('/usuarios/{usuario}', 'UsuarioController@show');. Neste caso, o Laravel tentará encontrar um usuário com o ID correspondente ao parâmetro {usuario}. Se o usuário existir, ele será injetado automaticamente no método show do controlador.

04 - COMANDO para criar um CONTROLLER:

```
php artisan make:controller ProdutoController --resource
```

CÓDIGO:

```
<?php

namespace App\Http\Controllers;
```

```
use App\Models\Produto;
use Illuminate\Http\Request;
```

```
class ProdutoController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $produto = Produto::all();
        return response()->json($produto, 201);
    }
```

```
    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $produto = Produto::create($request->all());
```

```
        return response()->json($produto, 201);
    }
```

```
    /**
     * Display the specified resource.
     */
    public function show(string $id)
    {
        $produto = Produto::find($id);
        return response()->json($produto, 201);
```

```
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, string $id)
    {
        $produto = Produto::find($id);

        $produto->update($request->all());

        return response()->json($produto, 201);
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy(string $id)
    {
        $produto = Produto::find($id);

        $produto->delete();

        return response()->json('Produto removido com
sucesso!', 201);
    }
}
```

05 - CODIGO:

```
public function store(Request $request)
    {
        $produto = Produto::create($request->all()); // o
método CREATE aceita um array de atributos, cria um modelo e
o insere no banco de dados.

        return response()->json($produto, 201);
    }
```

- no POSTMAN:
Seleciona o metodo de requisição POST(enviar),

- ENDPOINT:

Coloca (URL do servidor seguido da rota a ser testada, sempre com o nome "api" na frente. ex:
http://127.0.0.1:8000/api/produto/criar

- HEADERS:
Colocar as seguintes chaves e valor
Key: Value:
Content-Type application/json
Accept application/json

Caso o envio seja atraves de formulário é necessário a seguinte chave-valor:
X-CSRF-TOKEN (token criado)

- BODY:
Selecionar a opção "RAW",
Implementar o corpo da informação a ser cadastrada (em JSON):

```
[
    {
        "nome" : "Teclado",
        "valor" : "R$ 100,00"
    }
]
```

Para criar o TOKEN:

Em uma nova aba do POSTMAN
- no endpoint colocar a URL do servidor seguida da rota para criação de tokens. ex:
http://127.0.0.1:8000/api/token
(obs. metodo de requisição GET).

- Criado o TOKEN, copiar e colar no HEADERS da requisição POST.

06 - CÓDIGO do CONTROLLER:

```
public function index()
    {
        $produto = Produto::all(); // O método all retorna o
array subjacente representado pela coleção:

        return response()->json($produto, 201);
    }
```

no POSTMAN:
Seleciona o metodo de requisição GET(obter)

- ENDPOINT:
Coloca (URL do servidor seguido da rota a ser testada, sempre com o nome "api" na frente. ex:
http://127.0.0.1:8000/api/produto/listar -> para testar a listagem de produtos).

- Resposta da API:

```
[
```

```
    {
        "id": 1,
        "nome": "Teclado",
    "valor": "R$ 100,00",
        "created_at": "2024-12-19T12:47:16.000000Z",
        "updated_at": "2024-12-19T12:50:55.000000Z"
    }
]
```

07 - CODIGO no CONTROLLER:

```
public function update(Request $request, string $id)
    {
        $produto = Produto::find($id); // O método FIND
retorna o modelo que tem uma chave primária que corresponde
à chave fornecida.

        $produto->update($request->all()); // O método
UPDATE deve ser usado para atualizar registros existentes no
banco de dados.
```

```
        return response()->json($produto, 201);
    }
```

no POSTMAN:
Seleciona o método PUT

ENDPOINT:
Coloca (URL do servidor seguido da rota a ser testada, sempre com o nome "api" na frente. ex:
http://127.0.0.1:8000/api/produto/atualizar,

Para requisiçoes do tipo PUT (No caso do PUT edita os dados no BODY):

- HEADERS:
Colocar as seguintes chaves e valor

Key:	Value:
Content-Type	application/json
Accept	application/json

Caso o envio seja atraves de formulário é necessário a seguinte chave-valor:
X-CSRF-TOKEN (token criado)

- BODY:
Selecionar a opção "RAW",
Implementar o corpo da informação a ser ATUALIZADA (em JSON):

```
[    {
```

```
        "nome" : "Teclado",
        "valor" : "R$ 150,00"
    }
]
```

08 - CODIGO:

```
public function destroy(string $id)
    {
        $produto = Produto::find($id);

        $produto->delete(); // O método delete do construtor
de consultas pode ser usado para excluir registros da tabela.

        return response()->json('Produto removido com
sucesso!', 201);
    }
```

no POSTMAN:
Seleciona o método DELETE

ENDPOINT:
Coloca (URL do servidor seguido da rota a ser testada, sempre com o nome "api" na frente. ex:
http://127.0.0.1:8000/api/produto/deletar,

Para requisiçoes do tipo DELETE:

- HEADERS:
Colocar as seguintes chaves e valor

Key:	Value:
Content-Type	application/json
Accept	application/json
X-CSRF-TOKEN	(token criado)

09 - O Eloquent ORM é uma poderosa ferramenta do Laravel que permite interagir com o banco de dados usando uma abordagem orientada a objetos. Ele mapeia tabelas para modelos, tornando operações CRUD mais simples e intuitivas.

- Como o Eloquent facilita o CRUD:
Leitura (Read)

Produto::all() retorna todos os registros.
Produto::find($id) retorna um registro específico.
Permite consultas mais complexas com métodos como where, orderBy, etc.
Criação (Create)

Produto::create($request) permite criar registros diretamente sem escrever SQL.
Gerencia automaticamente timestamps (created_at, updated_at).
Atualização (Update)

update() atualiza os campos fornecidos sem necessidade de SQL explícito.
Exclusão (Delete)

delete() remove registros com simplicidade.

10 - Seguir essas práticas assegura que sua API seja segura, robusta e fácil de manter:

- Use FormRequest sempre que possível.
- Centralize as regras de validação.
- Personalize mensagens de erro.
- Utilize regras pré-definidas do Laravel.
- Crie regras personalizadas para cenários específicos.
- Sempre sanitizar dados antes de processá-los.
- Valide relacionamentos e arrays adequadamente.