INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

ENSINO MÉDIO INTEGRADO - INFORMÁTICA
Disciplina de Desenvolvimento Web | Backend
Aula 05: Framework Laravel - Pacote Sanctum

Gil Eduardo de Andrade

Conceitos Preliminares

(https://laravel.com/docs/13.x/sanctum)

O Pacote *Laravel Sanctum*

O Laravel Sanctum oferece um sistema de autenticação extremamente leve para SPAs (aplicações de página única), aplicativos móveis e APIs simples baseadas em tokens. O Sanctum permite que cada usuário do seu aplicativo gere múltiplos tokens de API para sua conta. Esses tokens podem receber permissões/escopos que especificam quais ações eles podem executar.

Funcionamento

O Laravel Sanctum existe para resolver dois problemas distintos, que serão apresentados e discutidos a seguir, antes de iniciarmos a instalação, configuração e uso do pacote.

Tokens de API

Primeiramente, o Sanctum é um pacote simples que pode ser usado para emitir tokens de API para usuários sem a complexidade do (*)OAuth. Esse recurso foi inspirado no GitHub e em outros aplicativos que emitem "tokens de acesso pessoal". Por exemplo, imagine que as "configurações da conta" do seu aplicativo tenham uma tela onde o usuário pode gerar um token de API para sua conta. Você pode usar o Sanctum para gerar e gerenciar esses tokens. Esses tokens geralmente têm um longo tempo de expiração (anos), mas podem ser revogados manualmente pelo usuário a qualquer momento.

(*)OAuth: é um protocolo de autorização que permite a aplicativos acessarem dados ou recursos de outros serviços em seu nome, sem precisar da sua senha. Ele funciona entregando "chaves" digitais temporárias (tokens) em vez de suas credenciais - página oficial.

O Sanctum oferece esse recurso armazenando tokens de API do usuário em uma única tabela do banco de dados e autenticando as solicitações HTTP recebidas por meio do ***Authorization*** *header* (cabeçalho de autorização), que deve conter um token de API válido.

Autenticação SPA

Em segundo lugar, o Sanctum existe para oferecer uma maneira simples de autenticar aplicativos de página única (SPAs) que precisam se comunicar com uma API baseada em Laravel. Esses SPAs podem estar no mesmo repositório que seu aplicativo Laravel ou em um repositório completamente separado, como um SPA criado usando React, Svelte, VueJS ou Angular.

Para essa funcionalidade, o Sanctum não utiliza tokens de nenhum tipo. Em vez disso, o Sanctum usa os serviços de autenticação de sessão baseados em cookies integrados ao Laravel. Normalmente, o Sanctum utiliza o ***web authentication guard*** do Laravel para tal. Isso proporciona os benefícios da proteção contra CSRF, autenticação de sessão e também protege contra o vazamento das credenciais de autenticação via XSS.

O Sanctum só tentará autenticar usando cookies quando a solicitação recebida se originar do seu próprio frontend SPA. Quando o Sanctum examina uma solicitação HTTP recebida, ele primeiro verifica a presença de um cookie de autenticação e, caso não encontre nenhum, examina o *Authorization header* em busca de um token de API válido.

APLICANDO OS CONCEITOS: SANCTUM

Continuação da Aplicação Criada na Aula Anterior

1) Instalando o Pacote Sanctum

O primeiro passo é a instalação do pacote Laravel Sanctum via Composer. Certifique-se de executar este comando dentro do container Docker da sua aplicação, para garantir que as dependências sejam instaladas corretamente no ambiente de desenvolvimento.

Comando
(no terminal de comando - dentro da pasta da aplicação)

Subindo a Aplicação Laravel

```shell
docker compose up
```

Comando
(no terminal de comando - dentro da pasta da aplicação)

Instalando o Sanctum

```shell
docker compose exec app composer require laravel/sanctum
```

2) Publicando Arquivos de Configurações e Migrações do Sanctum

As migrações criam a tabela ***personal_access_tokens*** na base de dados, utilizada para armazenar os tokens de API.

Comando
(no terminal de comando - dentro da pasta da aplicação)

Configurando / Criando as Migrações

```shell
docker compose exec app php artisan vendor:publish
--provider="Laravel\\Sanctum\\SanctumServiceProvider"
```

Efetuando as Migrações

```shell
docker compose exec app php artisan migrate
```

3) Configurando a Modelo "User"

Para que o modelo "***User***" possa emitir tokens de API, ele precisa usar a *trait* "***HasApiTokens***", fornecida pelo Sanctum. Sendo assim, vamos editar o arquivo "***app/Models/User.php***", adicionando a *trait*.

Arquivo: "app/Models/User.php"

```php
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Database\Factories\UserFactory;
use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Attributes\Hidden;
use Illuminate\Database\Eloquent\Factories\HasFactory;
```

```php
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens; // Linha Adicionada

#[Fillable(['name', 'email', 'password', 'role_id'])]
#[Hidden(['password', 'remember_token'])]
class User extends Authenticatable
{
    /** @use HasFactory<UserFactory> */
    use HasFactory, Notifiable, HasApiTokens; // Linha Alterada

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }

    public function role() {
        return $this->belongsTo('\App\Models\Role');
    }
}
```

4) Configurando Middleware e CORS

Middleware do Sanctum

No Laravel 13, a configuração de middlewares é feita no arquivo: "***bootstrap/app.php***". É preciso garantir que o middleware "***Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful***" esteja presente no grupo api ou em um grupo de middleware que você usará para suas rotas de API. Para autenticação baseada em token, o middleware "***auth:sanctum***" é o mais relevante.

Arquivo: "bootstrap/app.php"

```php
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        api: __DIR__.'/../routes/api.php', // Linha Adicionada
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {
        // Linhas Adicionadas
        $middleware->api(prepend: [
            Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();
```

CORS

Para permitir que aplicações móveis (ou qualquer frontend) em domínios diferentes se comuniquem com a API, é necessário configurar o ***(*)CORS (Cross-Origin Resource Sharing)***. Para tal é preciso alterar o arquivo "***config/cors.php***" (se ele não existir, é possível publicá-lo utilizando os seguintes passos:

Criar, manualmente, o arquivo "routes/api.php"

Arquivo: "routes/api.php"

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
```

Comando

(no terminal de comando - dentro da pasta da aplicação)

Publicando o Arquivo "config/cors.php"

```shell
docker compose exec app php artisan config:publish cors
```

(*)CORS (Compartilhamento de Recursos com Origens Diferentes): é um mecanismo de segurança dos navegadores que permite que um site acesse recursos ou APIs de um domínio diferente daquele que o carregou. Ele atua como um "porteiro", verificando se o servidor de destino autoriza a solicitação - página de referência.

Arquivo: "config/cors.php"

```php
<?php

return [
    // Linha Alterada
    'paths' => ['api/*', 'sanctum/csrf-cookie', 'login', 'logout'],

    'allowed_methods' => ['*'],

    'allowed_origins' => ['*'],

    'allowed_origins_patterns' => [],

    'allowed_headers' => ['*'],

    'exposed_headers' => [],

    'max_age' => 0,

    'supports_credentials' => false,
];
```

***OBS.:** em produção, **NUNCA** use **'allowed_origins' => ['*']**. Substitua o "*" pelos domínios específicos de suas aplicações móveis, isso evitará vulnerabilidades de segurança.*

5) Adaptando Policies e Serviço de Permissão

Para que seja possível acessar e verificar as permissões do usuário autenticado via API (Sanctum) vamos adaptar a codificação, visto que agora as permissões não poderão ficar carregadas em sessão, como fizemos com na autenticação via Breeze.

Arquivo: “app/Http/Policies/CursoPolicy.php”

PHP

```php
// Passamos o Usuário Logado para o método isAuthorized()
<?php

namespace App\Policies;

use App\Models\Curso;
use App\Models\User;
use App\Services\PermissionService;

class CursoPolicy {

     public function __construct(protected PermissionService $service)
{}

    public function viewAny(User $user): bool {
        return $this->service->isAuthorized('curso.index', $user);
    }

    public function view(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.show', $user);
    }

    public function create(User $user): bool {
         return $this->service->isAuthorized('curso.create', $user);
    }

    public function update(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.edit', $user);

    }

    public function delete(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.delete', $user);

    }
    public function restore(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.delete', $user);
    }

    public function forceDelete(User $user, Curso $curso): bool
```

```
    {
        return $this->service->isAuthorized('curso.delete', $user);
    }
}
```

Arquivo: “app/Http/Policies/DisciplinaPolicy.php”

PHP

```
// Passamos o Usuário Logado para o método isAuthorized()
<?php

namespace App\Policies;

use App\Models\Disciplina;
use App\Models\User;
use App\Services\PermissionService;

class DisciplinaPolicy {

     public function __construct(protected PermissionService $service)
{}

    public function viewAny(User $user): bool {
        return $this->service->isAuthorized('disciplina.index', $user);
    }

    public function view(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.show', $user);
    }

    public function create(User $user): bool {
        return $this->service->isAuthorized('disciplina.create', $user);
    }

    public function update(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.edit', $user);
    }

    public function delete(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.delete', $user);
    }
```

```
    public function restore(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.delete', $user);
    }

    public function forceDelete(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.delete', $user);
    }
}
```

Arquivo: “app/Http/Services/PermissionService.php”

```php
<?php

namespace App\Services;

use App\Repositories\PermissionRepository;

class PermissionService extends BaseService {

    public function __construct(protected PermissionRepository $repository) {}

    protected function getRepository(): mixed {
        return $this->repository;
    }

    public function getPermissions($role) {
        $arr = Array();
        $perm = $this->repository->list(
            ['resource'],
            ['field' => 'role_id', 'value' => $role], 'resource_id'
        );

        foreach($perm as $item) {
            $arr[$item->resource->name] = true;
        }

        return $arr;
```

```
    }

    public function loadPermissions($role) {

        $arr_permissions = $this->getPermissions($role);

        // dd($arr_permissions);
        session(['user_permissions' => $arr_permissions]);
    }

    public function isAuthorized($resource, $user) {

        $permissions = session('user_permissions');

        if(!isset($permissions)) $permissions = $this->getPermissions($user->role_id);

        if(array_key_exists($resource, $permissions)) {
            return true;
        }
        return false;
    }
}
```

6) Criando a Controller de Autenticação API

Para recepcionar e gerenciar as requisições HTTP (autenticação) vindas através do arquivo de rotas da API, vamos criar, manualmente, uma classe de Controle (*AuthController*) dentro do seguinte diretório "***app/Http/Controllers/Api/***"

Arquivo: "app/Http/Controllers/Api/AuthController.php"

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Validation\ValidationException;
use App\Services\PermissionService;
```

```php
class AuthController extends Controller {

    protected $permissionService;

    public function __construct(PermissionService $permissionService) {
        $this->permissionService = $permissionService;
    }
    public function login(Request $request) {

        $request->validate([
            'email' => ['required', 'string', 'email'],
            'password' => ['required', 'string'],
            'device_name' => ['required', 'string'],
            // Nome do dispositivo para o token
        ]);

        if (! Auth::attempt($request->only('email', 'password'))) {
            throw ValidationException::withMessages([
                'email' => ['As credenciais fornecidas estão incorretas.'],
            ]);
        }

        $user = Auth::user();

        // Carregando as permissões para o usuário autenticado via API
        $this->permissionService->getPermissions($user->role_id);
        $token = $user->createToken($request->device_name)->plainTextToken;
        return response()->json([
            'token' => $token,
            'user' => $user,
            'permissions' => session('user_permissions') // Retorna as permissõe
        ]);
    }

    public function logout(Request $request) {
        $request->user()->currentAccessToken()->delete();
        return response()->json(
            ['message' => 'Token revogado com sucesso.']
        );
    }
}
```

OBS.: *o* ***PermissionService*** *é injetado e o método* ***getPermissions()*** *é invocado após a autenticação ser bem-sucedida. Isso garante que as permissões sejam carregadas para o usuário da API, de forma a replicar a lógica existente para o Breeze. As permissões são então retornadas como resposta a requisição de login.*

7) Adaptando as Controllers de Curso e Disciplina

Para reaproveitar o código, já implementado, das controladoras de Curso e Disciplina, vamos adaptá-los para identificar se a requisição recebida partiu do ***middleware web*** (*internas a aplicação*) ou do ***middleware api*** (*externas a aplicação*).

Arquivo: "app/Http/Controllers/CursoController.php"

PHP

```php
<?php

namespace App\Http\Controllers;

use App\Models\Curso;
use App\Http\Requests\CursoRequest;
use Illuminate\Support\Facades\Gate;
use App\Services\CursoService;

class CursoController extends Controller {

    public function __construct(protected CursoService $service) {}

    public function index() {

        Gate::authorize('viewAny', Curso::class);
        $data = $this->service->all(['disciplina', 'aluno'], [], 'nome');
        // Linha Adicionada
        if (request()->is('api/*')) return response()->json($data);

        return view('curso.index', compact(['data']));
    }

    public function create() {

        Gate::authorize('create', Curso::class);
        // Linha Adicionada
        if (request()->is('api/*')) return response()->json(['message' => 'Create!']);
```

```
        return view('curso.create');
    }

    public function store(CursoRequest $request) {

        Gate::authorize('create', Curso::class);
        $curso = $this->service->store($request->validated());
        // Linha Adicionada
        if ($request->is('api/*'))  return response()->json($curso, 201);

        return redirect()->route('curso.index');
    }

    public function show(string $id) {
        $curso = $this->service->find($id);
        Gate::authorize('view', $curso);

        if(isset($curso)) {
            // Linha Adicionada
            if(request()->is('api/*')) return response()->json($curso);

            return view('curso.show', compact(['curso']));
        }

        return "<h1>Curso não encontrado!</h1>";
    }

    public function edit(string $id) {
        $curso = $this->service->find($id);
        Gate::authorize('update', $curso);

        if(isset($curso)) {
            // Linha Adicionada
            if(request()->is('api/*')) return response()->json($curso);

            return view('curso.edit', compact(['curso']));
        }

        return "<h1>Curso não encontrado!</h1>";
    }
```

```php
public function update(CursoRequest $request, string $id) {
    $curso = $this->service->find($id);
    Gate::authorize('update', $curso);

    if(isset($curso)) {
        $updated = $this->service->update($request->validated(), $id);
        // Linha Adicionada
        if(request()->is('api/*')) return response()->json($updated);

        return redirect()->route('curso.index');
    }

    return "<h1>Curso não encontrado!</h1>";
}

public function destroy(string $id) {
    $curso = $this->service->find($id);
    Gate::authorize('delete', $curso);

    if(isset($curso)) {
        $this->service->remove($id);

        // Linhas Adicionadas
        if(request()->is('api/*'))
              return response()->json(
                    ['message' => 'Curso removido com sucesso.']
              );

        return redirect()->route('curso.index');
    }

    return "<h1>Curso não encontrado!</h1>";
}

public function audit(string $id) {
    $curso = $this->service->find($id);
    Gate::authorize('delete', $curso);

    if(isset($curso)) {
        $data = $this->service->audit($id);
        return view('curso.audit', compact(['data']));
    }
```

```
        return "<h1>Não encontrado!</h1>";
    }
}
```

Arquivo: “app/Http/Controllers/DisciplinaController.php”

PHP

```php
<?php

namespace App\Http\Controllers;

use App\Models\Disciplina;
use App\Http\Requests\DisciplinaRequest;
use App\Services\CursoService;
use App\Services\DisciplinaService;
use Illuminate\Support\Facades\Gate;

class DisciplinaController extends Controller {

    public function __construct(
        protected DisciplinaService $service,
        protected CursoService $cursoService
    ) {}

    public function index() {

        Gate::authorize('viewAny', Disciplina::class);
        $data = $this->service->all(['curso'], [], 'nome');
        // Linha Adicionada
        if (request()->is('api/*')) return response()->json($data);

        return view('disciplina.index', compact(['data']));
    }

    public function create() {
        Gate::authorize('create', Disciplina::class);
        $cursos = $this->cursoService->all([], [], 'nome');
        // Linha Adicionada
        if (request()->is('api/*')) return response()->json($cursos);
```

```
        return view('disciplina.create', compact(['cursos']));
    }

    public function store(DisciplinaRequest $request) {

        Gate::authorize('create', Disciplina::class);
        $disciplina = $this->service->store($request->validated());
        // Linha Adicionada
        if ($request->is('api/*'))  return response()->json($disciplina, 201);

        return redirect()->route('disciplina.index');
    }

    public function show(string $id) {

        $disciplina = $this->service->find($id, ['curso']);
        Gate::authorize('view', $disciplina);

        if(isset($disciplina)) {
            // Linha Adicionada
            if(request()->is('api/*')) return response()->json($disciplina);

            return view('disciplina.show', compact(['disciplina']));
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }

    public function edit(string $id) {
        $disciplina = $this->service->find($id, ['curso']);
        Gate::authorize('update', $disciplina);
        $cursos = $this->cursoService->all([], [], 'nome');

        if(isset($disciplina)) {
            // Linha Adicionada
            if(request()->is('api/*')) return response()->json($disciplina);

            return view('disciplina.edit', compact(['disciplina', 'cursos']));
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }
```

```
    public function update(DisciplinaRequest $request, string $id) {

        $disciplina = $this->service->find($id);
        Gate::authorize('update', $disciplina);

        if(isset($disciplina)) {
            $updated = $this->service->update($request->validated(), $id);
            // Linha Adicionada
            if(request()->is('api/*')) return response()->json($updated);

            return redirect()->route('disciplina.index');
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }

    public function destroy(string $id) {

        $disciplina = $this->service->find($id);
        Gate::authorize('delete', $disciplina);

        if(isset($disciplina)) {
            $this->service->remove($id);
            // Linhas Adicionadas
            if(request()->is('api/*'))
                  return response()->json(
                        ['message' => 'Disciplina removida com sucesso.']
                  );

            return redirect()->route('disciplina.index');
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }
}
```

8) Criando as Rotas de Autenticação via API

Arquivo: “routes/api.php”

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AuthController;
use App\Http\Controllers\CursoController;
use App\Http\Controllers\DisciplinaController;

Route::post('/login', [AuthController::class, 'login']);
Route::post('/logout', [AuthController::class, 'logout'])
      ->middleware('auth:sanctum');

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/user', function (Request $request) {
        return $request->user();
    });

    // Cursos
    Route::apiResource('cursos', CursoController::class);
    // Disciplinas
    Route::apiResource('disciplinas', DisciplinaController::class);
});
```

9) Testando a Aplicação

AUTENTICANDO E OBTENDO O TOKEN:

No Terminal de Comandos

Autenticando Usuário - Papel de Professor (POST)

```shell
curl -X POST http://localhost:15000/api/login \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -d '{
           "email": "antonio@gmail.com",
           "password": "@1234@5678",
           "device_name": "curl_test"
         }'
```

OBS.: *Deve retornar 200 OK com o* ***TOKEN*** *e os dados do usuário autenticado.*

No Terminal de Comandos

Obter Dados do Usuário Autenticado - Papel de Professor (GET)

```shell
curl -X GET http://localhost:15000/api/user \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR"
```

OBS.: *Deve retornar 200 OK com os dados do usuário autenticado.*

No Terminal de Comandos

Autenticando Usuário - Papel de Coordenador (POST)

```shell
curl -X POST http://localhost:15000/api/login \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -d '{
           "email": "rafaela@gmail.com",
           "password": "@1234@5678",
           "device_name": "curl_test"
         }'
```

OBS.: *Deve retornar 200 OK com o* ***TOKEN*** *e os dados do usuário autenticado.*

No Terminal de Comandos

Obter Dados do Usuário Autenticado - Papel de Coordenador (GET)

```shell
curl -X GET http://localhost:15000/api/user \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_COORDENADOR"
```

OBS.: *Deve retornar 200 OK com os dados do usuário autenticado.*

TESTANDO AS PERMISSÕES PARA OS PAPÉIS:

No Terminal de Comandos

Listar Curso - Papel de Professor (GET)

```shell
curl -X GET http://localhost:15000/api/cursos \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR"
```

OBS.: *Deve retornar 403 Forbidden / Acesso Negado.*

No Terminal de Comandos

Listar Curso - Papel de Coordenador (GET)

```shell
curl -X GET http://localhost:15000/api/cursos \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_COORDENADOR"
```

OBS.: *Deve retornar 200 OK com a lista de cursos cadastrados no banco.*

No Terminal de Comandos

Criar Curso - Papel de Professor (POST)

```shell
curl -X POST http://localhost:15000/api/cursos \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR" \
     -d '{
           "nome": "TÉCNICO EM LOGÍSTICA",
           "duracao": "4"
         }'
```

OBS.: *Deve retornar 403 Forbidden / Acesso Negado.*

No Terminal de Comandos

Criar Curso - Papel de Coordenador (POST)

```shell
curl -X POST http://localhost:15000/api/cursos \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer TOKEN_COORDENADOR" \
     -d '{
           "nome": "TÉCNICO EM MANUTENÇÃO",
           "duracao": "4"
         }'
```

OBS.: *Deve retornar 201 Created, juntamente com os dados do curso criado.*

No Terminal de Comandos

Criar Disciplina - Papel de Professor (POST)

```shell
curl -X POST http://localhost:15000/api/disciplinas \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR" \
     -d '{
           "nome": "LÍNGUA PORTUGUESA",
           "carga_horaria": "2",
           "curso_id": "1"
         }'
```

OBS.: *Deve retornar 201 Created, juntamente com os dados da disciplina criada.*

No Terminal de Comandos

Criar Disciplina - Papel de Coordenador (POST)

```shell
curl -X POST http://localhost:15000/api/disciplinas \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR" \
     -d '{
           "nome": "PROGRAMAÇÃO DISTRIBUIDA",
           "carga_horaria": "4",
           "curso_id": "2"
         }'
```

OBS.: *Deve retornar 201 Created, juntamente com os dados da disciplina criada.*

EFETUANDO OUTROS TESTE:

No Terminal de Comandos

Buscar um Curso - Papel de Professor (GET)

```shell
curl -X GET http://localhost:15000/api/cursos/1 \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR"
```

No Terminal de Comandos

Buscar um Curso - Papel de Coordenador (GET)

```shell
curl -X GET http://localhost:15000/api/cursos/1 \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_COORDENADOR"
```

No Terminal de Comandos

Atualizar Curso - Papel de Professor (PUT)

```shell
curl -X PUT http://localhost:15000/api/cursos/1 \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR" \
     -d '{
           "nome": "TÉCNICO EM MEIO AMBIENTE",
           "duracao": "3 anos"
         }'
```

No Terminal de Comandos

Atualizar Curso - Papel de Coordenador (PUT)

```shell
curl -X PUT http://localhost:15000/api/cursos/1 \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer TOKEN_COORDENADOR" \
     -d '{
           "nome": "TÉCNICO EM PRODUÇÃO CULTURAL",
           "duracao": "3 anos"
         }'
```

No Terminal de Comandos

Remover Curso - Papel de Professor (POST)

```shell
curl -X DELETE http://localhost:15000/api/cursos/1 \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR"
```

No Terminal de Comandos

Remover Curso - Papel de Coordenador (POST)

```shell
curl -X DELETE http://localhost:15000/api/cursos/1 \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_COORDENADOR"
```

No Terminal de Comandos

Efetuando Logout - Papel de Professor (POST)

```shell
curl -X POST http://localhost:15000/api/logout \
     -H "Accept: application/json" \
     -H "Authorization: Bearer TOKEN_PROFESSOR"
```

No Terminal de Comandos

Efetuando Logout - Papel de Coordenador (POST)

```shell
curl -X POST http://localhost:15000/api/logout \
     -H "Accept: application/json" \
     -H "Authorization: Bearer SEU_TOKEN_AQUI"
```