Helder Neves

hmrneves@gmail.com

Helder’s api

Uma simples API direcionada para um blog baseada na framework Laravel.

Índice

Conteúdo

[Introdução 4](#_Toc45744284)

[Planeamento 5](#_Toc45744285)

[Funcionalidades da API 6](#_Toc45744286)

[Base de Dados 8](#_Toc45744287)

[Diagrama ER 8](#_Toc45744288)

[Autenticação da API 9](#_Toc45744289)

[Middlewares 10](#_Toc45744290)

[Modelo de Respostas da API 11](#_Toc45744291)

[Documentação da API 12](#_Toc45744292)

[Autenticação 12](#_Toc45744293)

[*Registo* 12](#_Toc45744294)

[*Login* 13](#_Toc45744295)

[*Logout – Requer Autenticação* 13](#_Toc45744296)

[*Logout de todos os dispositivos – Requer Autenticação* 13](#_Toc45744297)

[*Reposição de Password* 14](#_Toc45744298)

[*Criar Nova Password* 14](#_Toc45744299)

[Categorias 15](#_Toc45744300)

[*Listar Todas as Categorias* 15](#_Toc45744301)

[*Listar uma Categoria* 15](#_Toc45744302)

[*Registar Categoria – Requer Autenticação* 15](#_Toc45744303)

[*Atualizar Dados de uma Categoria – Requer Autenticação* 16](#_Toc45744304)

[*Eliminar uma Categoria – Requer Autenticação* 16](#_Toc45744305)

[Posts 17](#_Toc45744306)

[*Listar Todos os Posts* 17](#_Toc45744307)

[*Obter Dados de um Post* 17](#_Toc45744308)

[*Regista um Novo Post na Base de Dados – Requer Autenticação* 17](#_Toc45744309)

[*Atualiza um Post existente na Base de Dados – Requer Autenticação* 18](#_Toc45744310)

[*Elimina um post – Requer Autenticação* 18](#_Toc45744311)

[Comentários 19](#_Toc45744312)

[*Registar um comentário – Requer Autenticação* 19](#_Toc45744313)

[*Atualiza um Comentário – Requer Autenticação* 19](#_Toc45744314)

[*Elimina um Comentário – Requer Autenticação* 20](#_Toc45744315)

[Pesquisa 20](#_Toc45744316)

[*Por Keyword* 20](#_Toc45744317)

[*Por Categoria* 20](#_Toc45744318)

[Logs – Requer Autenticação 21](#_Toc45744319)

[Subscritores 21](#_Toc45744320)

[*Listar Todos – Requer Autenticação* 21](#_Toc45744321)

[*Subscrever* 21](#_Toc45744322)

[*Pedir Token Para Cancelamento da Subscrição* 22](#_Toc45744323)

[*Cancelar Subscrição* 22](#_Toc45744324)

[Administração – Gerir Posts 23](#_Toc45744325)

[*Listar Todos os Posts – Requer Autenticação* 23](#_Toc45744326)

[*Listar Posts a Aguardar Aprovação – Requer Autenticação* 23](#_Toc45744327)

[*Aprovar Post Pendente – Requer Autenticação* 23](#_Toc45744328)

[*Eliminar Post Pendente – Requer Autenticação* 24](#_Toc45744329)

[Administração – Gerir Comentários 25](#_Toc45744330)

[*Listar Comentários Pendentes – Requer Autenticação* 25](#_Toc45744331)

[*Aprovar Comentário Pendente – Requer Autenticação* 25](#_Toc45744332)

[*Elimina Comentário Pendente – Requer Autenticação* 25](#_Toc45744333)

[Administração – Gerir Utilizadores 26](#_Toc45744334)

[*Listagem de Utilizadores – Requer Autenticação* 26](#_Toc45744335)

[*Listagem dos Posts de um Utilizador – Requer Autenticação* 26](#_Toc45744336)

[*Listagem dos Comentários de um Utilizador – Requer Autenticação* 27](#_Toc45744337)

[*Atualiza o Grupo de um Utilizador – Requer Autenticação* 27](#_Toc45744338)

[Administração – Grupos de Utilizadores 28](#_Toc45744339)

[*Listagem – Requer Autenticação* 28](#_Toc45744340)

[Utilizadores 29](#_Toc45744341)

[*Ver Perfil de Utilizador – Requer Autenticação* 29](#_Toc45744342)

[*Atualizar Email – Requer Autenticação* 29](#_Toc45744343)

[*Atualizar Password – Requer Autenticação* 29](#_Toc45744344)

[*Atualizar Avatar – Requer Autenticação* 30](#_Toc45744345)

[Documentação da WebApp 31](#_Toc45744346)

[Autenticação 31](#_Toc45744347)

[Dados Apresentados 31](#_Toc45744348)

[*Grupo dos Administradores* 31](#_Toc45744349)

[*Grupo dos Utilizadores Gerais* 31](#_Toc45744350)

[Implementação da Aplicação 32](#_Toc45744351)

[Software Necessário 32](#_Toc45744352)

[Implementação 34](#_Toc45744353)

[Dificuldades no Desenvolvimento 35](#_Toc45744354)

[Conclusão 36](#_Toc45744355)

Introdução

Este projeto resultou no desenvolvimento de uma API[[1]](#footnote-1) com base ideológica de um blog.

A ideia foi levada avante pelo facto de ser um excelente tema para desenvolver entidades relacionais, bem como o fluxo de dados entre elas, proteção de rotas via Middleware, unificação de respostas da API via Trait e aprofundamento/prova de conhecimentos da framework Laravel.

Planeamento

O planeamento deste projeto não seguiu a ordem habitual, ou seja, não foram definidas metas de desenvolvimento e nem prazos, uma vez que foi desenvolvido à medida da aquisição de conhecimentos.

Funcionalidades da API

Uma aplicação de front-end, poderá ter acesso às seguintes funcionalidades:

1. Posts
   1. Criar
   2. Ler
   3. Atualizar – Permissão: Administradores ou Proprietário
   4. Eliminar – Permissão: Administradores
2. Comentários
   1. Registar um comentário a um post ou a um comentário
   2. Atualizar – Permissão: Administradores ou Proprietário
   3. Eliminar – Permissão: Administradores ou Proprietário
3. Categorias
   1. Criar – Permissão: Administradores
   2. Ler
   3. Atualizar – Permissão: Administradores
   4. Eliminar – Permissão: Administradores
4. Subscritores
   1. Listar todos os subscritores – Permissão: Administradores
   2. Subscrever
   3. Pedir token para cancelar subscrição
   4. Cancelar subscrição por token
5. Pesquisar posts
   1. Por keyword no título
   2. Por categoria
6. Utilizadores
   1. Ver perfil de utilizador
   2. Atualizar email – Permissão: Proprietário
   3. Atualizar password – Permissão: Proprietário
   4. Atualizar avatar – Permissão: Proprietário
7. Logs
   1. Analisar logs de um utilizador – Permissão: Administradores
8. Autenticação
   1. Registar novo utilizador
   2. Login
   3. Logout do dispositivo
   4. Logout de todos os dispositivos
   5. Pedir token para efetuar reposição de password
   6. Repor password através de token
9. Gestão – Permissão: Administradores
   1. Gerir posts pendentes: lista-los, aprová-los e eliminá-los
   2. Gerir comentários pendentes: lista-los, aprová-los e eliminá-los
   3. Analisar atividade de um utilizador
   4. Atualizar grupo de um utilizador

Base de Dados

O sistema de gestão de dados deste projeto está assente em MySQL.

Diagrama ER[[2]](#footnote-2)

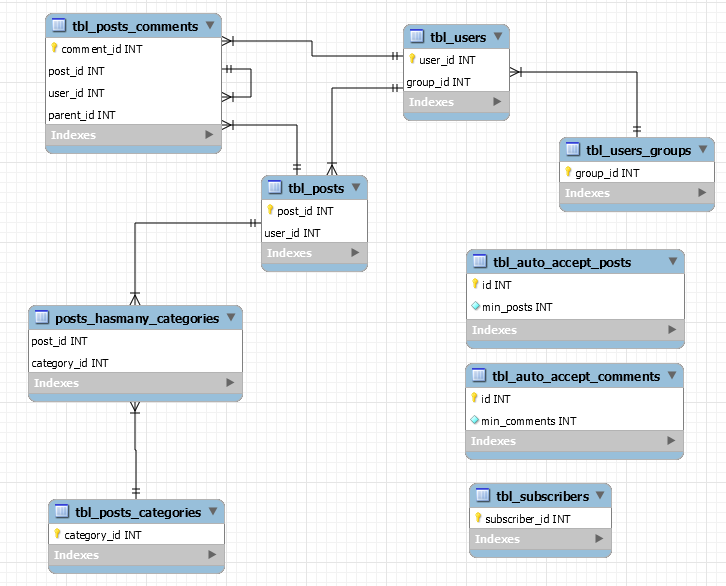


Figura 1 - Diagrama ER

Pelo diagrama, podemos verificar que o projeto conta com várias relações de 1:N[[3]](#footnote-3) e uma relação de N:M[[4]](#footnote-4).

Autenticação da API

Ao invés das sessões que as aplicações web usam, as API utilizam tokens para autenticar os utilizadores e não mantêm estados das sessões entre chamadas.

A API usa o pacote Laravel Passport[[5]](#footnote-5) para efetuar toda a gestão de autenticação por tokens.

Middlewares

Um Middleware[[6]](#footnote-6) é um mecanismo de filtragem dos pedidos HTTP que entram numa aplicação.

Este projeto conta com dois Middlewares implementados no ficheiro de roteamento da aplicação:

1. Auth:**@param**
   1. **@param** api
   2. Responsável por proteger rotas que necessitem que haja um utilizador autenticado
2. CheckPermissions:**@param**
   1. **@param** *group\_id*
   2. Responsável por proteger rotas que necessitem que o utilizador autenticado pertença a um grupo especifico

Modelo de Respostas da API

Traits[[7]](#footnote-7) são um mecanismo para reuso de código em linguagens de herança única, como o PHP. Um Trait tem o intuito de reduzir algumas limitações da herança única permitindo ao desenvolvedor reutilizar conjuntos de métodos livremente, numa variedade de classes independentes, vivendo em diferentes hierarquias de classes. A semântica da combinação de Traits e classes é definida de uma maneira que reduz a complexidade, e evita os problemas típicos associados a herança múltipla e Mixins.

De forma a unificar as respostas da API, foi criado um Trait para o efeito. Todas as respostas da API seguem o mesmo padrão não precisando assim, de estar a criar a mesma função a cada retorno de dados numa chamada à API.

Esta funcionalidade é particularmente útil no caso da necessidade de alterar algo na resposta. A ausência de um Trait para o efeito, resultaria em centenas de edições das repostas nos controladores da API.

**protected** **function** response(**bool** $success = **null**, **int** $code = **null**, **string** $message = **null**, $data = [])

{

**return** response()

->json(

[

'response' => [

'success' => $success,

'code' => $code,

'message' => $message,

'data' => $data,

]

],

$code

)

->withHeaders([

'Accept' => 'application/json',

'Content-Type' => 'application/json',

]);

}

Documentação da API

Abaixo será descrito as possíveis chamadas à API, bem como os métodos e parâmetros aceites.

Todos os chamadas à API que contenham um asterisco vermelho, requerem autenticação, ou seja, a chamada tem de conter um Bearer Token. Exemplo: Logout \*.

Autenticação

### Registo

Ação → Regista um novo utilizador.

Postman → 1. Authentication > register()

Método → POST

URL → api/v1/auth/register

Form Data →

1. name – String
2. user\_name – String
3. password – String, Mínimo 8 caracteres
4. password\_confirm – String, Mínimo 8 caracteres
5. email – Email válido

### Login

Ação → Autentica um utilizador

Postman → 1. Authentication > login()

Método → POST

URL → api/v1/auth/login

Form Data →

1. email – Email válido
2. password – String, Mínimo 8 caracteres

### Logout – Requer Autenticação

Ação → Efetua o logout do utilizador. Elimina o token obtido.

Postman → 1. Authentication > logout()

Método → POST

URL → api/v1/auth/logout

### Logout de todos os dispositivos – Requer Autenticação

Ação → Efetua o logout e elimina todos os tokens atribuídos a esse utilizador.

Postman → 1. Authentication > logout\_all\_devices()

Método → POST

URL → api/v1/auth/logout-all-devices

### Reposição de Password

Ação → Pede a reposição de password e obtém um token único para o efeito.

Postman → 1. Authentication > request\_token()

Método → POST

URL → api/v1/auth/request-reset-password-token

Form Data →

1. email – Email válido

### Criar Nova Password

Ação → Permite criar uma nova password utilizando o token previamente obtido.

Postman → 1. Authentication > set\_new\_password()

Método → POST

URL → api/v1/auth/set-new-password

Form Data →

1. email – Email válido
2. token – Token obtido na chamada anterior (request\_token())
3. password – String, Mínimo 8 caracteres
4. passsword\_confirmation – String, Mínimo 8 caracteres

Categorias

### Listar Todas as Categorias

Ação → Listagem de todas as categorias.

Postman → 2. Categories > índex()

Método → GET

URL → api/v1/categories

### Listar uma Categoria

Ação → Lista os dados de uma categoria em especifico.

Postman → 2. Categories > show()

Método → GET

URL → api/v1/categories/{category\_id}

### Registar Categoria – Requer Autenticação

Ação → Regista uma nova categoria na base de dados.

Postman → 2. Categories > store()

Método → POST

URL → api/v1/categories

Form Data →

1. name – String

### Atualizar Dados de uma Categoria – Requer Autenticação

Ação → Atualiza os dados de uma categoria já registada

Postman → 2. Categories > update()

Método → POST

URL → api/v1/categories/{category\_id}

Form Data →

1. \_method = PUT
2. name – String

### Eliminar uma Categoria – Requer Autenticação

Ação → Elimina uma categoria da base de dados.

Postman → 2. Categories > destroy()

Método → POST

URL → api/v1/categories/{category\_id}

Form Data →

1. \_method = DELETE

Posts

### Listar Todos os Posts

Ação → Lista todos os posts da base de dados com paginação.

Postman → 3. Posts > index()

Método → GET

URL → api/v1/posts

### Obter Dados de um Post

Ação → Lista os dados de um post, contendo também as informações do autor, cas categorias e dos comentários associados.

Postman → 3. Posts > show()

Método → GET

URL → api/v1/posts/{slug}

### Regista um Novo Post na Base de Dados – Requer Autenticação

Ação → Regista um novo post na base de dados.

Postman → 3. Posts > store()

Método → POST

URL → api/v1/posts

Form Data →

1. title – String | Mínimo 10 Caracteres
2. body – String | Mínimo 20 caracteres
3. image – MIMES: jpg, png, gif | Tamanho máximo 3MB | Opcional
4. categories[] – Array de Id’s.

### Atualiza um Post existente na Base de Dados – Requer Autenticação

Ação → Atualiza um post existente na base de dados.

Postman → 3. Posts > update()

Método → POST

URL → api/v1/posts/{post\_id}

Form Data →

1. \_method = PUT
2. title – String | Mínimo 10 Caracteres
3. body – String | Mínimo 20 caracteres
4. image – MIMES: jpg, png, gif | Tamanho máximo 3MB | Opcional
5. categories[] – Array de Id’s.

### Elimina um post – Requer Autenticação

Ação → Elimina um post

Postman → 3. Posts > destroy()

Método → POST

URL → api/v1/posts/{post\_id}

Form Data →

1. \_method = DELETE

Comentários

### Registar um comentário – Requer Autenticação

Ação → Regista um novo comentário a um post/comentário.

Postman → 4. Comments > store()

Método → POST

URL → api/v1/comments/store

Form Data →

1. post\_id – Campo numérico
2. comment – String | Mínimo 10 caracteres
3. parent\_id – Opcional (a preencher quando o comentário é feito a um comentário)

### Atualiza um Comentário – Requer Autenticação

Ação → Atualiza um comentário

Postman → 4. Comments > update()

Método → POST

URL → api/v1/comments/{comment\_id}

Form Data →

1. \_method = PUT
2. comment – String | Mínimo 10 caracteres

### Elimina um Comentário – Requer Autenticação

Ação → Elimina um comentário

Postman → 4. Comments > destroy()

Método → POST

URL → api/v1/comments/{comment\_id}

Form Data →

1. \_method = DELETE

Pesquisa

### Por Keyword

Ação → Pesquisa posts pela keyword fornecida. A pesquisa é feita pelo título.

Postman → 5. Search Posts > By Title Keyword

Método → GET

URL → api/v1/search/posts/by-keyword/{keyword}

### Por Categoria

Ação → Pesquisa posts pela categoria.

Postman → 5. Search Posts > By Category ID

Método → GET

URL → api/v1/search/posts/by-category/{category\_id}

Logs – Requer Autenticação

Ação → Obtem os logs de um utilizar em especifico.

Postman → 6. Logs > Search Logs by user\_id

Método → GET

URL → api/v1/logs/search/{user\_id}

Subscritores

### Listar Todos – Requer Autenticação

Ação → Lista todos os subscritores.

Postman → 7. Subscribers > index()

Método → GET

URL → api/v1/subscribers

### Subscrever

Ação → Regista nova subscrição.

Postman → 7. Subscribers > subscribe()

Método → POST

URL → api/v1/subscribers

Form Data →

1. name – String
2. email – Email válido

### Pedir Token Para Cancelamento da Subscrição

Ação → Pede um token único para efetuar o cancelamento da subscrição.

Postman → 7. Subscribers > request\_unsuscribe\_token()

Método → POST

URL → api/v1/subscribers/request-unsubscribe-token

Form Data →

1. email – Email válido

### Cancelar Subscrição

Ação → Cancela a subscrição de um utilizador utilizando o token previamente obtido.

Postman → 7. Subscribers > unsubscribeByToken()

Método → POST

URL → api/v1/subscribers/unsubscribe-by-token

Form Data →

1. email – Email válido
2. token – String

Administração – Gerir Posts

Todas as chamadas dentro desta categoria, requerem que o utilizador pertença ao grupo dos administradores.

### Listar Todos os Posts – Requer Autenticação

Ação → Lista todos os posts.

Postman → 8. Administration > Manage Posts > posts()

Método → GET

URL → api/v1/administration/manage-posts/posts

### Listar Posts a Aguardar Aprovação – Requer Autenticação

Ação → Lista todos os posts que aguardam aprovação

Postman → 8. Administration > Manage Posts > pending\_posts()

Método → GET

URL → api/v1/administration/manage-posts/pending-posts

### Aprovar Post Pendente – Requer Autenticação

Ação → Aprova um post que está pendente de aprovação.

Postman → 8. Administration > Manage Posts > approve\_posts()

Método → POST

URL → api/v1/administration/manage-posts/pending-posts/aprove

Form Data →

1. post\_id – Numérico

### Eliminar Post Pendente – Requer Autenticação

Ação → Elimina um post que está pendente de aprovação, ou seja, não aprovado.

Postman → 8. Administration > Manage Posts > delete\_pending\_posts()

Método → POST

URL → api/v1/administration/manage-posts/pending-posts/delete

Form Data →

1. post\_id – Numérico

Administração – Gerir Comentários

Todas as chamadas dentro desta categoria, requerem que o utilizador pertença ao grupo dos administradores.

### Listar Comentários Pendentes – Requer Autenticação

Ação → Lista todos os comentários que aguardam aprovação.

Postman → 8. Administration > Manage Comments > comments()

Método → GET

URL → api/v1/administration/manage-comments/pending-comments

### Aprovar Comentário Pendente – Requer Autenticação

Ação → Aprova um comentário que está pendente de aprovação.

Postman → 8. Administration > Manage Comments > approve()

Método → POST

URL → api/v1/administration/manage-comments/pending-comments/approve

Form Data →

1. comment\_id – Numérico

### Elimina Comentário Pendente – Requer Autenticação

Ação → Elimina comentário pendente.

Postman → 8. Administration > Manage Comments > delete()

Método → POST

URL → api/v1/administration/manage-comments/pending-comments/delete

Form Data →

1. comment\_id – Numérico

Administração – Gerir Utilizadores

Todas as chamadas dentro desta categoria, requerem que o utilizador pertença ao grupo dos administradores.

### Listagem de Utilizadores – Requer Autenticação

Ação → Lista todos os utilizadores registados.

Postman → 8. Administration > Manage Users > list\_users()

Método → GET

URL → api/v1/administration/manage-users/list-users

### Listagem dos Posts de um Utilizador – Requer Autenticação

Ação → Lista os posts de um utilizador

Postman → 8. Administration > Manage Users > list\_user\_posts()

Método → GET

URL → api/v1/administration/manage-users/list-user-posts

Query String Params →

1. user\_id – Numérico
2. limit – Numérico – Limite de resultados
3. order\_by – Enum [ asc / desc ]

### Listagem dos Comentários de um Utilizador – Requer Autenticação

Ação → Lista os comentários de um utilizador

Postman → 8. Administration > Manage Users > list\_user\_comments()

Método → GET

URL → api/v1/administration/manage-users/list-user-comments

Query String Params →

1. user\_id – Numérico
2. limit – Numérico – Limite de resultados
3. order\_by – Enum [ asc / desc ]

### Atualiza o Grupo de um Utilizador – Requer Autenticação

Ação → Atualiza o grupo de um utilizador

Postman → 8. Administration > Manage Users > update\_user\_group()

Método → POST

URL → api/v1/administration/manage-users/list-users

Form Data →

1. user\_id – Numérico
2. group\_id – Numérico

Administração – Grupos de Utilizadores

Todas as chamadas dentro desta categoria, requerem que o utilizador pertença ao grupo dos administradores.

### Listagem – Requer Autenticação

Ação → Lista todos os possíveis grupos de utilizador.

Postman → 8. Administration > Manage Groups > index()

Método → GET

URL → api/v1/administration/manage-user-groups/list-groups

Utilizadores

### Ver Perfil de Utilizador – Requer Autenticação

Ação → Lista os dados do perfil de um utilizador.

Postman → 9. Users > índex()

Método → GET

URL → api/v1/users/{user\_id}

### Atualizar Email – Requer Autenticação

Ação → Atualiza o email de um utilizador

Postman → 9. Users > update\_email()

Método → POST

URL → api/v1/users/update-email

Form Data →

1. user\_id – Numérico
2. email – Email válido | Único na tabela de utilizadores

### Atualizar Password – Requer Autenticação

Ação → Atualiza a password de um utilizador

Postman → 9. Users > update\_password()

Método → POST

URL → api/v1/users/update-password

Form Data →

1. user\_id – Numérico
2. password – String | Mínimo 8 caracteres
3. password\_confirmation – String | Mínimo 8 caracteres

### Atualizar Avatar – Requer Autenticação

Ação → Atualiza o avatar de um utilizador

Postman → 9. Users > update\_avatar()

Método → POST

URL → api/v1/users/update-avatar

Form Data →

1. user\_id – Numérico
2. avatar – File | MIMES: jpg, png, gif | Tamanho máximo 2MB

Documentação da WebApp

Este projeto conta com um simples dashboard para apresentação de estatísticas.

Autenticação

A autenticação é feita através da autenticação por defeito do Laravel. Desta forma, os utilizadores da API poderão aceder ao dashboard.

Dados Apresentados

### Grupo dos Administradores

Caso o utilizador pertença ao grupo de administradores ( *group\_id 1* ), poderá analisar as seguintes estatísticas:

1. Numero total de posts
2. Utilizador com mais posts
3. Numero total de comentários
4. Utilizador com mais comentários
5. Numero total de categorias
6. Categoria mais associada a posts
7. Total de post pessoais
8. Total de comentários pessoais

### Grupo dos Utilizadores Gerais

Caso o utilizador pertença ao grupo de utilizadores gerais ( *group\_id 2* ), somente poderá analisar o total de posts e comentários pessoais.

Implementação da Aplicação

Software Necessário

De forma a implementar a aplicação, deverá garantir que o computador/servidor contem o seguinte software instalado:

1. GIT
   1. Necessário para clonar o repositório da aplicação
   2. Download – <https://git-scm.com/downloads>
2. Composer
   1. Necessário para instalar os pacotes de PHP requeridos pela aplicação
   2. Download – <https://getcomposer.org/download/>
3. Postman
   1. Necessário para testar as chamadas à API
   2. Download – <https://www.postman.com/downloads/>
4. Laragon
   1. É recomendável o uso do servidor web Laragon, pelo facto da facilidade de configuração e por já trazer o Node JS
   2. Download – <https://sourceforge.net/projects/laragon/files/releases/4.0/laragon-full.exe>
5. PHP
   1. É recomendável o uso da versão 7.4.x do PHP
   2. Download para Windows – <https://windows.php.net/downloads/releases/php-7.4.8-Win32-vc15-x64.zip>
   3. Download para Linux – <https://www.php.net/downloads.php>
6. PHP Redis
   1. A cache do projeto é efetuada com o recurso ao Redis. Desta forma, deve garantir que a instalação do PHP contempla a extensão do Redis
   2. É possível alterar a cache através da modificação do ficheiro .env (não recomendado):
      1. CACHE\_DRIVER=file
      2. DEBUGBAR\_DRIVER=file
      3. SESSION\_DRIVER=file
   3. Downloads
      1. Download da Extensão Redis para Linux – <https://pecl.php.net/package/redis>
      2. Download da Extensão Redis para Windows – <https://pecl.php.net/package/redis/5.1.1/windows>

Implementação

1. Inicializar uma linha comandos e navegar até à pasta que irá conter o projeto. A pasta recomendada é a pasta ***www/*** ou ***public\_html/*** do servidor web do computador/servidor.
2. Na linha de comandos, executar o seguinte comando para clonar o repositório:
   1. git clone https://github.com/hmrneves1/blog-api.git
3. Na linha de comandos, navegar ate dentro da pasta do projeto clonado
4. Na linha de comandos, executar o seguinte comando para instalar as dependências de pacotes da framework:
   1. composer cc && composer self-update && composer install --prefer-dist --optimize-autoloader –vvv
5. Dentro da pasta do projeto, efetuar uma copia do ficheiro ***.env.example*** para **.env.** Reconfigurar somente mediante indicação.
6. Criar uma nova base de dados. Ex.: blog-api
7. Criar um utilizador, ex.: blog\_api\_user, e dar permissões totais à base de dados do 5º ponto
8. Configurar o nome da base de dados, utilizador e password no ficheiro **.env**, bloco **# Database connection**
9. Dentro da pasta do projeto, executar o seguinte comando para limpar todas as caches da mesma:
   1. php artisan optimize:clear
10. Dentro da pasta do projeto, executar o seguinte comando criar as tabelas da base de dados e semeá-las com dados aleatórios:
    1. php artisan migrate:fresh –seed
11. Definir o **DOCUMENT\_ROOT** do servidor web para a pasta **public/** dentro da pasta do projeto
12. Abrir o Postman e importar a coleção de chamadas localizada na pasta **\_\_postman\_collection/** dentro do projeto

Dificuldades no Desenvolvimento

Houve uma componente da aplicação que não foi possível desenvolver por falta de conhecimento.

Podemos registar comentários a posts, no entanto também podemos comentar comentários. A tabela dos comentários tem um atributo ***parent\_id***. Este atributo guarda o ID do comentário (***comment\_id***) ao qual estamos a comentar.

A dificuldade foi ordenar os comentários.

A resolução deste problema passa por criar uma função de ordenação e utilizá-la recursivamente para ordenar os comentários.

Exemplo:

1. Comentário N.º 1
   1. 1º comentário ao comentário N.º 1
      1. 1º comentário ao 1º comentário do comentário N.º 1
2. Comentário N.º 2
3. …

Foi possível criar uma relação no modelo de comentários para que possamos ordenar comentários, no entanto essa ordenação era só de 1º nível. Ou seja, um comentário com o ID **x** ia buscar todos os comentários em que o **parent\_id** fosse **x**.

Esta relação foi eliminada, dado que só retornava comentários ordenados ao 1º nível, o que não era o pretendido.

Conclusão

Este projeto conta com uma API assente na base ideológica de um blog e todas as suas funcionalidades básicas.

Como explicado anteriormente, houve dificuldades em criar uma função recursiva para ordenar os comentários a comentários. Isto, deve-se a falta de conhecimento prático no desenvolvimento da função em si e também por falta de tempo para aprofundar este tema.

Esta formação, bem como o projeto, foram ótimos para aprofundar o conhecimento da linguagem PHP, da framework Laravel e da arquitetura e modelo de bases de dados.

Em suma, será mais fácil desempenhar as funções de back-end no âmbito empresarial.

1. Application Programming Interface [↑](#footnote-ref-1)
2. Diagrama de Entidade-Relação [↑](#footnote-ref-2)
3. Relação 1:N – 1 para Muitos – Ex.: 1 post pode ter associados N comentários e 1 comentário só pode ter 1 post associado [↑](#footnote-ref-3)
4. Relação N:M – Muitos para Muitos – Ex.: 1 post pode ter N categorias associadas e 1 categoria pode ter N posts associados [↑](#footnote-ref-4)
5. <https://laravel.com/docs/7.x/passport> [↑](#footnote-ref-5)
6. <https://laravel.com/docs/7.x/middleware#introduction> [↑](#footnote-ref-6)
7. <https://www.php.net/manual/en/language.oop5.traits.php> [↑](#footnote-ref-7)