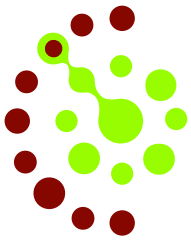

IPBeja

Instituto Politécnico de Beja

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Projecto – Implementação

Tiago Alexandre Baptista Pacheco - 20296

Beja, 30 de janeiro de 2023

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Projecto – Implementação

Tiago Alexandre Baptista Pacheco - 20296

Orientado por :

Luís Carlos Bruno, IPBeja

Relatório do Projecto – Implementação

2024

Resumo

Projecto – Implementação

Este projeto aborda a implementação bem-sucedida de uma aplicação web, denominada Forueddit, que se baseia na sólida estrutura Laravel para o desenvolvimento da sua API própria. A aplicação utiliza o padrão MVC (Model-View-Controller) para organizar e gerir eficientemente o fluxo de dados, proporcionando uma experiência robusta e intuitiva aos utilizadores. Ao oferecer diferentes tipos de utilizadores, como guest, utilizador normal e moderador, a plataforma promove uma participação diversificada. Destaca-se a funcionalidade de tradução, criação de publicação e validação de traduções, enriquecendo a interação na comunidade. Além disso, a presença ativa de uma equipa de moderação reforça a integridade da plataforma, assegurando um ambiente seguro e respeitoso. O relatório abrange a implementação detalhada da aplicação web, desde a modelagem da base de dados até os protótipos de interface, fornecendo uma visão abrangente do desenvolvimento e das estratégias adotadas para criar o Forueddit. **Palavras-chave**: *Forueddit*, *API*, *MVC*, *Laravel*.

Abstract

Projecto – Implementação

This project addresses the successful implementation of a web application, called Forueddit, which is based on the solid Laravel framework for developing its own API. The application uses the MVC (Model-View-Controller) pattern to efficiently organize and manage the data flow, providing a robust and intuitive experience for users. By offering different types of users, such as guest, normal user and moderator, the platform promotes diverse participation. The translation, publication creation and translation validation functionality stands out, enriching interaction in the community. Furthermore, the active presence of a moderation team reinforces the integrity of the platform, ensuring a safe and respectful environment. The report covers the detailed implementation of the web application, from database modeling to interface prototypes, providing a comprehensive view of the development and strategies adopted to create Forueddit. **Keywords**: *Forueddit*, *API*, *MVC*, *Laravel*.

Conteúdo

Lista de Figuras

List of Listings

Capítulo 1

Introdução

Este projeto destaca-se pela implementação de uma API própria, construída em Laravel, e pela adoção de um modelo de arquitetura MVC que integra todas as facetas da aplicação. Ao longo deste relatório, é explorado detalhadamente os elementos fundamentais que compõem o Forueddit, desde a caracterização dos tipos de utilizadores até a todas as funcionalidades disponíveis.

Este relatório tem como objetivo apresentar não apenas a estrutura técnica e funcionalidades da aplicação, mas também contextualizar o Forueddit.

Capítulo 2

Forueddit

O Forueddit é uma aplicação web inovadora que oferece aos utilizadores um fórum dinâmico, permitindo a criação, tradução e votação de publicações. Neste ambiente interativo, os utilizadores têm a liberdade de expressar ideias, compartilhar conteúdo e interagir com outros membros da comunidade. Além disso, a funcionalidade de votação (thumbs up/down) proporciona uma forma rápida e eficaz de avaliar a qualidade das publicações.

Uma característica distintiva do Forueddit é a presença de uma moderação ativa. Esta equipa verifica e valida traduções, garantindo a precisão e qualidade das informações disponíveis. Além disso, desempenham um papel crucial na identificação e eliminação de publicações que violem as diretrizes da comunidade, promovendo um ambiente seguro e respeitoso para todos os utilizadores.

Com uma interface intuitiva e funcionalidades robustas, o Forueddit representa uma plataforma aberta e colaborativa, onde a participação ativa da comunidade é incentivada e valorizada.

HomePage do Forueddit

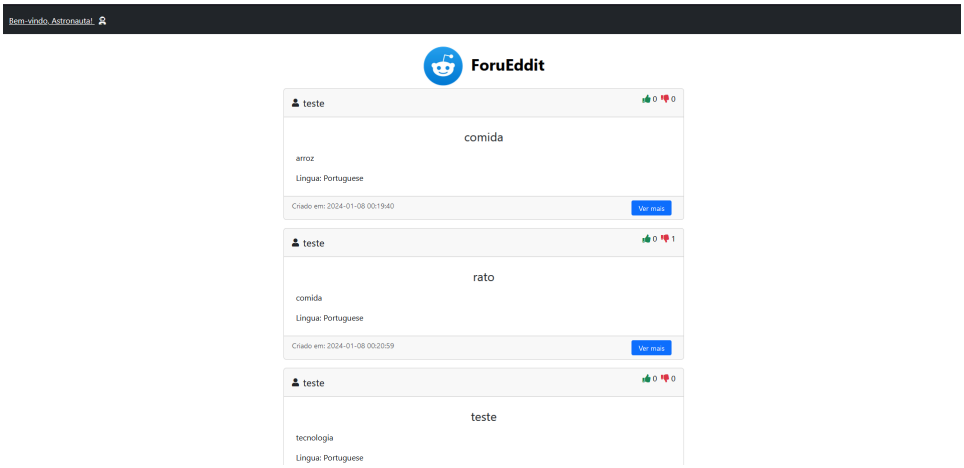

Figura 2.1: *HomePage Forueddit*

Tipos de Utilizador Disponíveis

- Guest: O utilizador "guest"é aquele que interage com a aplicação sem criar uma conta ou efetuar login. Este tipo de utilizador possui acesso limitado a certas funcionalidades da aplicação. Pode visualizar conteúdo público, mas as opções de participação, como criar publicações, votar ou traduzir, estao restritas. Os utilizadores "guest"podem explorar o Forueddit de forma anónima, mas a capacidade de envolvimento completo requer a criação de uma conta.

- Utilizador Forueddit: O utilizador da plataforma, após criar uma conta e efetuar login, tem acesso a uma gama mais ampla de funcionalidades na aplicação. Pode criar e publicar conteúdo, votar nas publicações (thumbs up/down) e traduzir o conteúdo disponível.

- Moderador: Os moderadores são utilizadores com privilégios especiais designados para manter a ordem, qualidade e respeito dentro da comunidade Forueddit. Têm a responsabilidade de validar traduções, monitorizar e reportar violações das diretrizes da comunidade e eliminar conteúdo inadequado. Os moderadores desempenham um papel crucial na promoção de um ambiente seguro e saudável, intervindo quando necessário para garantir que a experiência de todos os utilizadores seja positiva.

2.1 Melhorias efetuadas da análise e desenho do sistema

Depois da fase de análise e desenho do sistema, foram identificadas várias oportunidades de melhoria, resultando em ajustes significativos para aprimorar a eficiência, segurança e usabilidade do sistema.

2.2 Atualização da base de Dados

Uma das principais melhorias foi a otimização do desempenho do sistema, revendo o modelo da base de dados para garantir uma melhor eficiência na manipulação e organização dos dados no sistema. O modelo da base de dados ao longo do desenvolvimento da implementação do sistema foi mudando, pois ao longo da implementação foram evidenciadas necessidades, isto é, necessidade em criar tabelas de apoio.

A imagem abaixo mostra o modelo primário da base de dados.

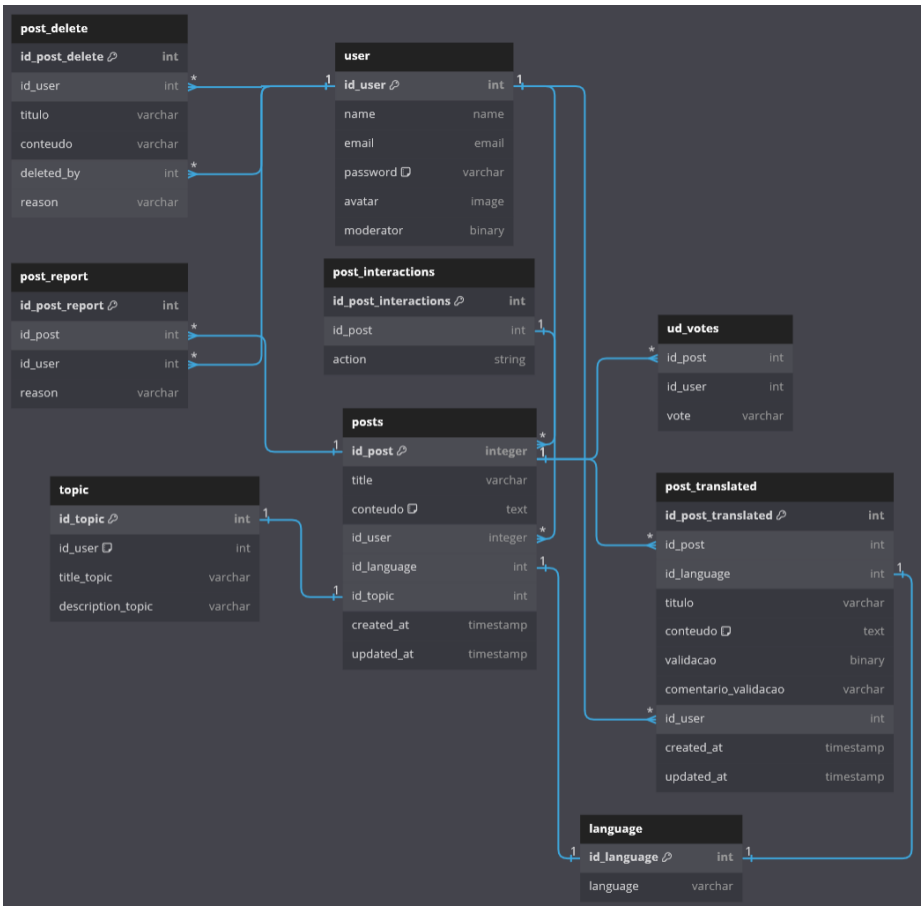

Figura 2.2: *Modelo Físico*

user (Dados do Utilizador):

- **id_user**: Identificação única do utilizador.
- **name**: Nome do utilizador.
- **email**: Endereço de email do utilizador.
- **password**: Palavra-passe do utilizador com nota para encriptação.
- **moderator**: Tipo de utilizador.

posts (Publicações):

- **id_post**: Identificação única da publicação.
- **titulo**: Título da publicação.
- **conteudo**: Corpo ou conteúdo da publicação com uma nota.
- **id_user**: Chave estrangeira que faz referência ao utilizador que criou a publicação.
- **created_at**: Data e hora de criação da publicação.
- **id_language**: Chave estrangeira que indica o idioma da publicação.
- **id_topic**: Chave estrangeira que faz referência ao tópico ao qual a publicação pertence.

post_interactions (Interações com a publicação):

- **id_post_interactions**: Identificação única das Interações.
- **id_post**: Identificação única da publicação
- **action**: Interação Realizada.

language (Idioma):

- **id_language**: Identificação única do idioma.
- **language**: Nome do idioma.
- **language_voice**: Som da linguagem.

post_translated (Publicação Traduzida):

- **id_post_translated**: Identificação única da publicação traduzida.
- **id_post**: Chave estrangeira que faz referência à publicação original.
- **id_language**: Chave estrangeira que indica o idioma da tradução.
- **titulo**: Título da publicação traduzida.
- **conteudo**: Corpo ou conteúdo da publicação traduzida com uma nota.
- **validação**: Validação realizada ou não.
- **comentário_validação**: Comentário relacionado com a validação.
- **id_user**: Chave estrangeira que faz referência ao utilizador que fez a tradução.
- **created_at**: Data e hora de criação da tradução.

topic (Tópico):

- **id_topic**: Identificação única do tópico.
- **id_user**: Chave estrangeira que faz referência ao criador do tópico.
- **title_topic**: Título do tópico.
- **description_topic**: Descrição do tópico.

UDVote (Votos):

- **id_votes**: Identificação única dos Votos.
- **id_post**: Identificação única da publicação.
- **id_user**: Identificação única do Utilizador.
- **vote**: Voto positivo, ou negativo.

post_deleted (Publicações Eliminadas):

- **id_post_delete**: Identificação única dos posts eliminados.
- **title**: Titulo da publicação eliminada.
- **content**: Conteúdo da publicação eliminada.
- **id_user**: Identificação única do utilizador.
- **deleted_by**: Identificação de quem eliminou.
- **reason**: Razão da Eliminação.

post_report (Posts Reportados):

- **id_post_report**: Identificação única dos posts reportados.
- **id_post**: Identificação única dos posts.
- **id_user**: Identificação única dos utilizadores.
- **reason**: Razão do report.

2.2.1 Adaptação ao longo do trabalho

De modo a haver uma melhor organização, decidiu se retirar a tabela relativa aos users, colocando esta tabela numa base de dados própria, para que não esteja dentro da base de dados da nossa api. Esta alteração foi uma dica proveniente do professor, de modo a melhorar a eficácia e desempenho das base de dados.

Ao longo do trabalho e na implementação mais concretamente nos meus casos de uso que serão abordados mais a frente, foi necessário criar uma tabela denominada de postsAPI_ids, de modo a haver uma organização dos dados gerados por uma api externa, que ajudou a realizar um dos casos de uso.

Segue a estrutura da tabela

postsAPI_IDs (Organizador de IDs):

- **id_postsAPI_ID**: Identificação única dos postsAPI_IDs.
- **local_post_id**: Identificação única dos posts localmente (Usa ID_post, da tabela posts).
- **api_post_id**: Identificação única do post id gerado na api externa.
- **api_translated_post_id**: Identificação única do id gerado ao traduzir um post na api externa.

2.3 Aprimoramento da Segurança

Com base em feedbacks e considerações adicionais de segurança, reforçamos as medidas de proteção existentes e implementamos novas camadas de segurança. Atualizamos as políticas de autenticação e autorização para garantir um controle mais preciso sobre o acesso aos recursos da aplicação.

2.4 Ajustes na Interface do Utilizador

Após análise de usabilidade e feedback proveniente da apresentação da fase de análise de desenho, realizamos ajustes na interface do utilizador para tornar a experiência mais intuitiva e agradável. Reorganizamos elementos de tela, refinamos fluxos de trabalho e incorporamos sugestões de design para melhorar a navegabilidade e eficácia global da aplicação.

Capítulo 3

Implementação

Neste capítulo são descritas as principais decisões e ações desenvolvidas na parte da implementação técnica deste projeto. A implementação foi realizada em Laravel, sendo uma framework de desenvolvimento web em PHP amplamente utilizado, conhecido por sua estrutura elegante e eficiente.

3.1 Definição da arquitetura do sistema na integração da aplicação com a API

A arquitetura do Laravel segue o padrão Model-View-Controller (MVC), proporcionando uma estrutura organizada para a construção de aplicações web modernas. O diagrama de blocos a seguir ilustra os principais elementos que compõem a arquitetura global da nossa api e como se interliga com as views.

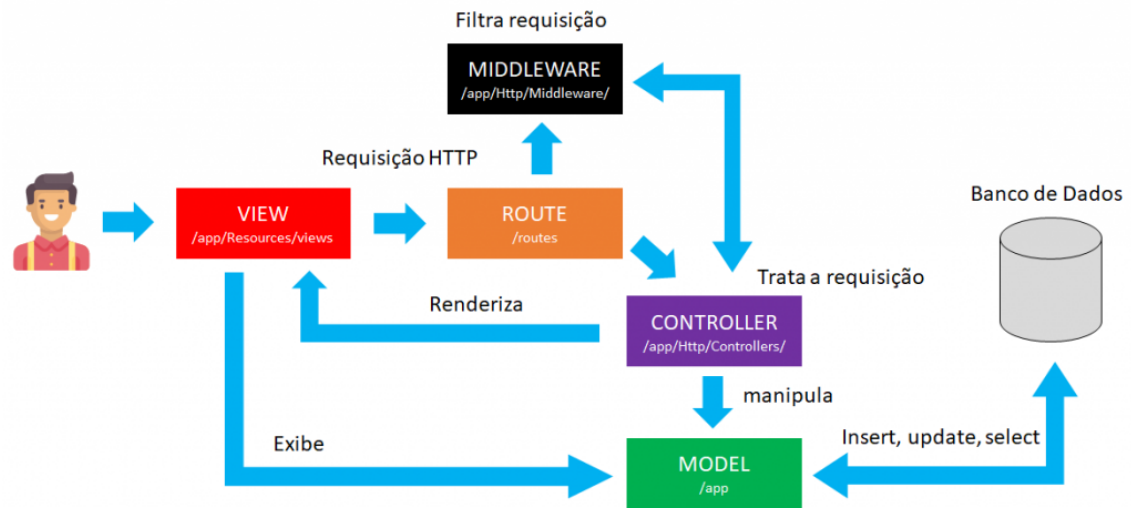

Figura 3.1: *Diagrama de Blocos*

1. **Frontend (Web):** Camada responsável pela interface do utilizador, desenvolvida com HTML, CSS/Bootstrap, JavaScript. Interage com os controllers web e as views.
2. **Web Controllers:** Controllers web que recebem requisições HTTP do frontend. Coordenam a lógica do sistema, manipulam dados e interagem com o modelo (Model).
3. **Views:** Responsáveis por exibir dados na interface do utilizador. Recebem informações dos controllers e Views e são renderizadas para os utilizadores finais.
4. **Model:** Camada de modelo que lida com a interação com o base de dados, utilizando o Eloquent ORM. Responsável por representar e manipular os dados da aplicação.
5. **Base de Dados:** Sistema de gestão de base de dados onde os dados são armazenados.
6. **API Controllers:** Controllers responsáveis por conter todos os métodos CRUD.

No nosso sistema existe uma ligação entre os nossos controladores da api, com os controladores "web", sendo estes os que retornam as views. Existe também uma ligação entre a nossa api e uma api externa denominada de TheFluentMe, de modo a ajudar no processo de traduções, que será abordado mais à frente. A imagem acima representa uma forma geral de como funciona todos o processo entre user e aplicação, realçando as ligações que são feitas de modo ao utilizador puder usar o sistema de forma correta.

3.2 Tecnologias usadas

Ao longo do desenvolvimento do projeto, diversas tecnologias foram utilizadas para garantir a eficiência, robustez e escalabilidade da aplicação. A seguir, é destacado as principais tecnologias e ferramentas que desempenharam um papel crucial no processo de implementação:

1. **Laravel:**
    - O framework PHP Laravel foi escolhido como a framework para o desenvolvimento da aplicação. A sua arquitetura e as suas funcionalidades integradas, como o Eloquent ORM e o sistema de rotas, proporcionaram um ambiente de desenvolvimento eficaz e produtivo.
2. **Bootstrap:**
    - Para o desenvolvimento do frontend, o framework Bootstrap foi adotado. Com sua biblioteca de componentes e estilos predefinidos, o Bootstrap acelerou a criação de interfaces responsivas e visualmente atrativas.
3. **Laravel Sanctum:**
    - O Laravel Sanctum foi utilizado para controladores de autenticação na aplicação. Essa ferramenta proporciona uma solução eficaz para autenticação de API, garantindo a segurança das interações entre frontend e backend.
4. **XAMPP (phpMyAdmin):**
    - O ambiente XAMPP foi utilizado para fornecer uma solução local de servidor web, base de dados e interpretação PHP. O phpMyAdmin, incluído no XAMPP, facilitou a administração da base de dados MySQL utilizado pela aplicação.
5. **GitHub:**
    - A plataforma GitHub foi escolhida para o controle de versões e colaboração no desenvolvimento. Através do GitHub, foi possível gerir alterações no código e rastrear problemas de um forma eficiente.

Estas tecnologias foram cuidadosamente selecionadas para atender às necessidades específicas do projeto, proporcionando um ambiente de desenvolvimento consistente e eficiente.

3.3 Desenvolvimento da API

3.3.1 Estrutura da API REST

A API REST foi projetada seguindo os princípios do estilo arquitetural RESTful. Abaixo estão alguns dos principais elementos da estrutura da API:

Endpoints

A API oferece os seguintes endpoints, cada um servindo a um propósito específico:

Exemplo da Estrutura

- `/users`: Endpoint para manipulação de dados de utilizadores.
- `/posts`: Endpoint para operações relacionadas a publicações.

Formato de Resposta

A API retorna dados no formato JSON. Isso foi escolhido devido à sua simplicidade e facilidade de interpretação.

3.3.2 Principais Decisões de Implementação

Autenticação e Autorização

A autenticação é gerida através do Laravel Sanctum, proporcionando tokens de acesso para utilizadores autenticados. A autorização é baseada em papéis e capacidades para garantir que apenas utilizadores autorizados possam realizar certas operações.

Tratamento de Erros

Erros são tratados de forma consistente, com respostas detalhadas contendo códigos de status HTTP apropriados. A API utiliza códigos de status padrão, como 200 para sucesso, 400 para solicitação inválida e 404 para recurso não encontrado.

Exemplos de Requisições e Respostas

A seguir estão exemplos de requisições e respostas para alguns dos principais endpoints:

```
GET /users/1

200 OK
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com"
}
```

3.3.3 Especificação da interface

A nossa api desenvolvida tem todas as suas rotas bem definidas, onde abaixo é representada uma tabela com todos os endpoints baseada na norma OpenAPI e na interface Swagger.

API	Description	Request body	Response body
POST /login	User authentication	{username, password}	{token}
POST /register	User registration	{username, email, password}	{user}
GET /fetch-languages	Fetch and store languages	None	{languages}
POST /send-post	Send a post	{postContent}	{post}
POST /send-post-translate/{api_post_id}/{id_language}	Send post to translate	{postContent}	{translatedPost}
GET /moderation/{postId}	Get moderation status	None	{moderationStatus}
GET /delete/{postId}	Get delete status	None	{deleteStatus}
POST /delete-post/{postId}	Delete a post	None	{deleteStatus}
POST /validation/{postId}	Post validation	None	{validationStatus}
GET /moderations-translation/{postId}	Get moderation status for translation	None	{moderationStatus}
GET /validate-by-id/{postId}	Get validation status by ID	None	{validationStatus}
GET /dashboard	Get dashboard data	None	{dashboardData}
GET /reports	Get reports	None	{reports}
GET /posts	Get all posts	None	{posts}
GET /posts/{id}	Get a post by ID	None	{post}
POST /posts	Add a new post	{postContent}	{post}
PUT /posts/{id}	Update a post	{postContent}	None
DELETE /posts/{id}	Delete a post	None	None
POST /posts/{postId}/report	Report a post	None	{reportStatus}
GET /posts/create	Create a post	None	{postCreationStatus}
GET /post/translation	Get all post translations	None	{postTranslations}
GET /post/translation/{id}	Get a post translation by ID	None	{postTranslation}
POST /post/translation/{id}	Add a new post translation	{translationContent}	{postTranslation}
PUT /post/translation/{id}	Update a post translation	{translationContent}	None
DELETE /post/translation/{id}	Delete a post translation	None	None
GET /topics	Get all topics	None	{topics}
GET /topics/{id}	Get a topic by ID	None	{topic}
POST /topics	Add a new topic	{topicContent}	{topic}
PUT /topics/{id}	Update a topic	{topicContent}	None
DELETE /topics/{id}	Delete a topic	None	None

Tabela 3.1: Rotas API

3.4 Decisões de implementação

No desenvolvimento da API, optamos pela abordagem *code-first*. Isso significa que o design e a estrutura da base de dados foram definidos utilizando as funcionalidades de migração do Laravel, assim como os modelos. Essa escolha foi feita para facilitar a manutenção do esquema da base de dados, e facilitar todas as alterações que poderiam ser necessárias. Antes da criação da API era necessários que as base de dados estivessem de acordo com o pretendido, para que toda a api funcionasse corretamente.

Seguir se ia 2 exemplos de modelos que foram usados para estruturar uma das tabelas de cada base de dados

Modelo para Estruturar a tabela Post_translated

O modelo `Post_translated` representa uma entidade relacionada à tradução de posts, onde é referente à Base de Dados da API de seu nome posts. A seguir, é apresentada uma descrição das principais características e ligações do modelo:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post\_translated extends Model
{
    use HasFactory;

    protected $connection = 'posts';
    protected $table = 'post_translated';
    protected $primaryKey = 'id_post_translated';
    protected $fillable = ['id_post', 'id_language', 'titulo', 'conteudo', 'validacao',
    'comentario_validacao', 'id_user'];


    public function post()
    {
        return $this->belongsTo(Post::class, 'id_post', 'id_post');
    }

    public function user()
    {
        return $this->belongsTo(User::class, 'id_user', 'id_user');
    }
```

```
28
29      public function language()
30      {
31          return $this->belongsTo(Language::class, 'id_language', 'id_language');
32      }
33  }
```

Listing 3.1: Modelo/Estrutura da tabela De Traduções

Este modelo está configurado para utilizar uma conexão chamada 'posts',isto é, a base de dados, associado à tabela 'post_translated', com uma chave primária personalizada 'id_post_translated'. As colunas preenchíveis incluem informações como o ID do post, ID do utilizador, título, conteúdo, validação, e comentário de validação.

Além disso, o modelo possui três relações definidas:

- `post()`: Relaciona o post traduzido ao modelo `Post`.
- `user()`: Relaciona o utilizador associado ao post traduzido ao modelo `User`.
- `language()`: Relaciona o idioma da tradução ao modelo `Language`.

Estas relações são estabelecidas através dos métodos `belongsTo`, indicando a relação das chaves estrangeiras entre os modelos.

Base de Dados dos Utilizadores de seu nome users

Modelo para Estruturar a tabela users

O modelo `Post_translated` representa uma entidade relacionada à tradução de posts. A seguir é apresentada uma descrição das principais características e relações do modelo:

```php
<?php
namespace App\Models;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
// sanctum
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{

    protected $connection = 'users';
    protected $primaryKey = 'id_user';
    protected $table = 'users';
    use HasApiTokens, HasFactory, Notifiable;
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        'moderator',
    ];
    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
    /**
     * The attributes that should be cast to native types.
     *
     * @var array
```

```
41      */
42     protected $casts = [
43         'email_verified_at' => 'datetime',
44     ];
45
46     public function posts()
47     {
48         return $this->hasMany(Post::class, 'id_user', 'id_user');
49     }
50
51     public function votes()
52     {
53         return $this->hasMany(Vote::class, 'id_user', 'id_user');
54     }
55
56 }
```

Listing 3.2: Modelo/Estrutura da tabela De Traduções

Este modelo está configurado para utilizar uma conexão chamada 'users',isto é, a base de dados, associado à tabela 'users', com uma chave primária personalizada 'id_user'. As colunas preenchíveis incluem informações como name,email,password e moderator.

Explicação dos principais pontos do modelo:

- **Classe User Extends Authenticatable:** A classe User estende Authenticatable, indicando que é uma classe de utilizador autenticável no Laravel. Usa os traits HasApiTokens, HasFactory, e Notifiable.
- **Propriedades Protegidas ($fillable, $hidden, $casts):** Define alguns atributos protegidos, como $fillable e $hidden. Configura o nome da tabela, chave primária e conexão da base de dados.
- **Métodos Relacionados (posts(), votes()):** O método posts indica que um utilizador pode ter muitos posts (uma relação de um para muitos com o modelo Post). O método votes indica que um utilizador pode fazer muitos votos (uma relação de um para muitos com o modelo Vote).

Todos os outros modelos que temos seguem esta ideia dos explicados anteriormente, onde teve de haver um cuidado especial, pois caso os modelos não estivessem como pretendido depois na inserção de valores na base de dados poderia causar problemas, assim como as interligações entre as base de dados e os controladores.

Criação das Migrações

A criação de migrações no Laravel desempenha um papel fundamental na definição e manutenção da estrutura da base de dados. As migrações são scripts de PHP que permitem estruturar o esquema da base de dados ao longo do tempo, facilitando a colaboração entre desenvolvedores e o controle de alterações na estrutura da base de dados.

No contexto de modelos, as migrações são frequentemente utilizadas para definir a estrutura da tabela associada a um determinado modelo. Essa prática segue a abordagem *code-first*, em que o design da base de dados é orientado pelo código relativo aos modelos.

Ao criar migrações para modelos, é possível especificar detalhes como os campos da tabela, os tipos de dados, restrições, chaves estrangeiras e outras configurações relacionadas ao armazenamento de dados. Dessa forma, as migrações tornam se uma parte essencial do processo de desenvolvimento, garantindo que a estrutura da base de dados esteja alinhada com as necessidades do modelo correspondente.

A seguir, é apresentada um exemplo prático de criação de migrações para modelos no Laravel, demonstrando como essa abordagem favorece a consistência entre o código da aplicação e o esquema da base de dados.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreatePostTranslatedTable extends Migration
{
    public function up()
    {
        Schema::connection('posts')->create('post_translated', function (Blueprint $table) {
            $table->id('id_post_translated');
            $table->unsignedBigInteger('id_post');
            $table->unsignedBigInteger('id_language')->nullable();
            $table->string('titulo');
            $table->text('conteudo');
            $table->tinyInteger('validacao')->default(0)->nullable();
            $table->text('comentario_validacao')->nullable();
            $table->unsignedBigInteger('id_user');
            $table->timestamps();

            $table->foreign('id_user')
                ->references('id_user')
                ->on('users.users')
                ->onDelete('cascade')
                ->onUpdate('cascade');

```

```
28              $table->foreign('id_post')
29                  ->references('id_post')
30                  ->on('posts.posts')
31                  ->onDelete('cascade')
32                  ->onUpdate('cascade');
33
34              $table->foreign('id_language')
35                  ->references('id_language')
36                  ->on('language')
37                  ->onDelete('cascade')
38                  ->onUpdate('cascade');
39          });
40      }
41
42      public function down()
43      {
44          Schema::connection('posts')->dropIfExists('post_translated');
45      }
46  }
```

Listing 3.3: Modelo/Estrutura da tabela De Traduções

3.5 Casos de Uso Da Implementação

Durante a fase de implementação do projeto, a minha responsabilidade concentrou-se na execução de dois casos de uso e na implementação de uma funcionalidade estatística, todos relacionados à gestão de traduções de publicações na plataforma.

Tradução de Publicações

Um dos casos de uso que abordei foi a tradução de publicações. Este processo envolveu a criação de uma interface intuitiva para permitir que os utilizadores contribuíssem com traduções para diversas línguas. Implementei lógica de backend para armazenar essas traduções associadas às publicações originais, garantindo uma estrutura de dados eficiente e fácil de manter.

Validação de Publicações por Moderadores

Outro caso de uso crítico que desenvolvi envolveu a validação de traduções por moderadores. Desenvolvi uma interface específica para moderadores reverem e validarem traduções propostas, incluindo a capacidade de fornecer feedback aos tradutores.

Visualização de Dados Estatísticos

Além dos casos de uso, contribuí para a implementação de uma funcionalidade estatística crucial. Desenvolvi um sistema de visualização de dados gráficos que permite aos utilizadores e administradores monitorizarem o desempenho das traduções. Isso inclui gráficos detalhados sobre o número de traduções por post, a quantidade de traduções validadas e a quantidade de traduções eliminadas. Essa funcionalidade estatística fornece insights valiosos sobre a eficácia do processo de tradução e a participação da comunidade.

Essas implementações não apenas contribuem para a expansão das funcionalidades da plataforma, mas também promovem a colaboração eficaz entre utilizadores, moderadores e administradores, assegurando a qualidade e relevância das traduções na comunidade.

3.6 Definições dos Controladores

3.6.1 Caso de Uso - Traduzir Publicação - Controlador APIFluentMeController

Este Controlador possui funcionalidades relacionadas à gestão de idiomas e tradução de publicações, assim como um método relativo a enviar a publicação criada para a API do FluentMe, de modo a facilitar a tradução, sendo este método desenvolvido pelo o colega David.

Funcionalidades Implementadas

- **fetchAndStoreLanguages():**
  - Verifica se a tabela `Language` na base de dados está vazia.
  - Realiza uma pedido à API para obter a lista de idiomas suportados.
  - Armazena esses idiomas na tabela `Language`.
- **sendPost($titulo, $conteudo, $id_language):**
  - Envia uma pedido POST à API para criar uma nova publicação.
  - Recebe como parâmetros o título da publicação, o conteúdo e o ID do idioma.
- **sendPostToTranslate($api_post_id, $id_language):**
  - Envia uma pedido POST à API para traduzir uma publicação existente.
  - Recebe como parâmetros o ID da publicação na nossa API e o ID do idioma para o qual a tradução deve ser feita.

A implementação deste controlador proporciona uma integração eficiente com a API do The Fluent Me, permitindo a gestão de idiomas e a realização de operações relacionadas à tradução de publicações de forma simplificada.

3.6.2 Caso de Uso - Traduzir Publicação - Comando FetchLanguages

O comando é projetado para armazenar as linguas na base de dados utilizando a API do controlador `APIFluentMeController`.

Funcionalidades Implementadas

Explicação do Código

- O comando está definido no namespace `App->Console->Commands`.
- Ele estende a classe `Command` do Laravel, que fornece funcionalidades básicas para criar comandos personalizados.
- O controlador `APIFluentMeController` é importado, indicando que será utilizado para realizar as operações desejadas.

```
class FetchLanguages extends Command
{
    protected $signature = 'fetch:languages';
    protected $description = 'Fetch and store languages';
}
```

- A classe `FetchLanguages` é uma extensão da classe `Command` e define a assinatura e a descrição do comando.
- O comando pode ser chamado usando `php artisan fetch:languages`.
- A descrição indica que o objetivo do comando é listar e armazenar linguagens.

```
public function handle()
{
    // Instancia o controlador
    $fluentMeController = new APIFluentMeController();

    // Chama a função para ir listar e armazenar as linguagens
    $fluentMeController->fetchAndStoreLanguages();

    $this->info('Languages fetched and stored successfully.');
}
```

- O método `handle` é chamado quando o comando é executado.
- Ele instancia o controlador `APIFluentMeController`.
- Chama a função `fetchAndStoreLanguages` no controlador, responsável por listar e armazenar as linguagens.
- Em seguida, exibe uma mensagem de sucesso.

Este comando instancia o controlador `APIFluentMeController` e chama o método `fetchAndStoreLanguages()` para listar e armazenar as informações das linguagens. Uma mensagem informativa é exibida na consola indicando o sucesso da operação. O comando `FetchLanguages` desempenha um papel crucial na integração de informações sobre linguagens na aplicação. A sua execução é essencial para manter atualizadas as informações sobre as linguagens disponíveis.

O comando pode ser executado utilizando o seguinte comando Artisan:

```
php artisan fetch:languages
```

3.6.3 Caso de Uso - Traduzir Publicação - Controlador APIPostTranslatedController

Este Controlador possui funcionalidades relacionadas a criação, leitura, atualização e exclusão de traduções (*Crud Methods*).

Funcionalidades Implementadas

Métodos do Controlador

`index`

- Retorna todos os registos da tabela `Post_translated` em formato JSON.

`show`

- Retorna o registo mais recente da tabela `Post_translated` em formato JSON.

`store`

- Processa a tradução de um post.
- Valida o idioma da tradução.
- Envia a publicação para tradução usando uma API externa.
- Atualiza as tabelas `PostApiId`, `Post_translated`, e `PostInteraction` em caso de sucesso.

`update`

- Permite a atualização de um registo de `Post_translated` com base no ID fornecido.

`destroy`

- Exclui um registo de `Post_translated` com base no ID fornecido.

O controlador `APIPostTranslatedController` desempenha um papel crucial na gestão de traduções de posts, envolvendo operações de criação, leitura, atualização e exclusão. Ele integra-se com uma API externa para facilitar o processo de tradução, garantindo consistência e eficiência.

Casos relevantes de codificação

Dentro do controlador das traduções é importante ver em detalhe o que o método store faz, de modo a perceber todas as suas fases.

- **Validação de Dados:**
  - A função `validate` é usada para garantir que o campo `id_language` na pedido seja numérico e obrigatório.
- **Obtenção do utilizador e ID da publicação na API:**
  - O utilizador autenticado é obtido através do `Auth::user()`.
  - O ID da publicação na API é recuperado da tabela `PostApiId` com base no `local_post_id`.
- **Envio para Tradução na API:**
  - Uma instância do controlador externo `APIFluentMeController` é criada.
  - O método `sendPostToTranslate` é chamado para enviar a publicação para tradução na API externa.
- **Processamento da Resposta da API:**
  - A resposta da API é verificada para garantir que a operação foi bem-sucedida.
  - Os dados da resposta são decodificados usando `json_decode`.
- **Atualização das Tabelas e Criação de registos:**
  - A tabela `PostInteraction` é atualizada para registrar a ação de tradução.
  - A tabela `PostApiId` é atualizada ou criada para armazenar o ID da publicação traduzida.
  - Uma nova entrada é criada na tabela `Post_translated` para armazenar os detalhes da tradução.
- **Resposta JSON:**
  - Se a tradução for bem-sucedida, uma resposta JSON de sucesso é retornada com os dados da tradução.
  - Em caso de falha, uma resposta JSON de erro é retornada junto com o status HTTP correspondente.

```php
public function store(Request $request, $id): \Illuminate\Http\JsonResponse
    {
        $request->validate([
            'id_language' => 'required|numeric',
        ]);

        $user = Auth::user();
        $api_post_id = PostApiId::where('local_post_id', $id)->value('api_post_id');

        $apiFluentMeController = new APIFluentMeController();
        $apiResponse = $apiFluentMeController->sendPostToTranslate($api_post_id,
        $request->id_language);

        if ($apiResponse->successful()) {
            // Decodifica a resposta da API
            $translatedPostId = json_decode($apiResponse->body())->post_id;
            $translatedLanguageId = json_decode($apiResponse->body())->post_language_id;
            $translatedPost_title = json_decode($apiResponse->body())->post_title;
            $translatedPost_content = json_decode($apiResponse->body())->post_content;

            PostInteraction::create([
                'id_post' => $id,
                'action' => 'traducao',
            ]);
            // Atualiza a tabela postsAPI_IDs tendo em conta o local_post_id
            PostApiId::updateOrCreate(
                ['local_post_id' => $id],
                ['api_translated_post_id' => $translatedPostId]
            );

            // Salva na tabela Post_translated a traduçao
            $post_translated = Post_translated::create([
                'id_post' => $id,
                'id_language' => $translatedLanguageId,
                'titulo' => $translatedPost_title,
                'conteudo' => $translatedPost_content,
                'id_user' => $user->id_user
            ]);
            return response()->json(['data' => $post_translated], 201);
        } else {
            // Em caso de falha no envio para a API
            return response()->json(['error' => 'Erro ao enviar post para a API.'],
            $apiResponse->status());
```

Listing 3.4: Método Store do Controlador APIPostTranslated

3.6.4 Caso de Uso - Traduzir Publicação - Controlador APIPostController

Este Controlador foi criado para os casos de uso do colega David, mas decidi adicionar uma particularidade no método show que ele criou. A ideia passava por ser apresentada também as traduções disponíveis do post que se estava a ver, de modo a proporcionar ao utilizador uma melhor experiência de utilização do sistema.

Funcionalidades Implementadas

Método `show`

O método `show` é responsável por listar informações de um post específico e as suas traduções associadas com base no ID fornecido.

- **Parâmetro:** *id* - Representa o identificador único do post a ser exibido.
- **Operações Realizadas:**
  - Utiliza o modelo `Post` para encontrar o post na base de dados com o ID fornecido.
  - Utiliza o modelo `Post_translated` para recuperar todas as traduções associadas ao post com o mesmo ID.
- **Retorno:** Retorna uma resposta JSON contendo os dados do post e as traduções associadas.

```
1 public function show($id): \Illuminate\Http\JsonResponse
2     {
3         $post = Post::findOrFail($id);
4         $posts_translated = Post_translated::where('id_post', $id)->get();
5 
6         return response()->json(['data' => $post, 'data2' => $posts_translated]);
7     }
```

Listing 3.5: Método show do controlador da API APIPostController

3.6.5 Caso de Uso - Validar Tradução - Controlador APIModController

Este Controlador possui funcionalidades relacionadas coma validação das traduções.

Funcionalidades Implementadas

Método `Validation(Request $request, $postId)`

- **Validação da pedido:** O método inicia validando os dados recebidos pela pedido usando a função `validate`. O campo 'validacao' é exigido e o campo 'comentario_validacao' é opcional (nullable).
- **Consulta à base de dados:** Utiliza o modelo `Post_translated` para obter o primeiro registo onde o `id_post` é igual ao `$postId` e a `validacao` não é igual a 1.
- **Tomada de Decisão com Base na `validacao`:** Se o valor de 'validacao' no pedido for 0, exclui o registo da tradução (`$post_translated`) e cria uma interação do post indicando que a tradução foi eliminada. Se o valor for diferente de 0, atualiza os campos 'validacao' e 'comentario_validacao' no registo e cria uma interação do post indicando que a tradução foi validada.
- **Resposta JSON:** Retorna uma resposta JSON contendo os dados do post traduzido.

Método `moderations_translation($postId)`

- **Consulta à base de dados:** Utiliza o modelo `Post_translated` para obter todos os registos onde o `id_post` é igual ao `$postId`.
- **Resposta JSON:** Retorna uma resposta JSON contendo todos os dados dos posts traduzidos associados ao post específico.

Método `validateById($postId)`

- **Consulta à base de dados:** Utiliza o modelo `Post_translated` para obter o primeiro registo onde o `id_post` é igual a `$postId` e a `validacao` não é igual a 1.
- **Resposta JSON:** Retorna uma resposta JSON contendo os dados do post traduzido não validado.

Estes métodos estão relacionados à moderação de traduções no sistema, onde a moderação pode validar ou eliminar traduções, obter traduções associadas a um post ou verificar se existe uma tradução não validada para um post específico.

```php
public function Validation(Request $request, $postId): \Illuminate\Http\JsonResponse
    {
        $request->validate([
            'validacao' => 'required',
            'comentario_validacao'
        ]);
        $post_translated = Post_translated::where('id_post', $postId)
            ->where('validacao', '!=', 1)
            ->first();
        $id_post_translated = Post_translated::where('id_post', $postId)
            ->where('validacao', '!=', 1)
            ->value('id_post_translated');

        if ($request->input('validacao') == 0) {
            $post_translated->delete();
            PostInteraction::create([
                'id_post' => $postId,
                'action' => 'traducao_eliminada',
            ]);
        } else {  $post_translated->update([
                'validacao' => $request->validacao,
                'comentario_validacao' => $request->comentario_validacao,
            ]);
            PostInteraction::create([
                'id_post' => $postId,
                'action' => 'traducao_validada',
            ]);
        }
        return response()->json(['data' => $post_translated], 201);
    }

    public function moderations_translation($postId): \Illuminate\Http\JsonResponse
    {
        $post_translated = Post_translated::where('id_post', $postId)->get();
        return response()->json(['data' => $post_translated], 201);
    }

    public function validateById($postId): \Illuminate\Http\JsonResponse
    { $post_translated = Post_translated::where('id_post', $postId)
            ->where('validacao', '!=', 1)
            ->first();
        return response()->json(['data' => $post_translated], 201);
    }
```

Listing 3.6: Métodos do APIModController

3.7 Definições dos Modelos

A implementação dos controladores da API é uma parte fundamental do desenvolvimento do sistema, especialmente quando se trata de casos de uso específicos, onde para isso é importante ter modelos bem estruturados. Na minha parte da implementação foram necessário criar um conjunto de modelos que ajudaram a que os controladores criados funcionassem corretamente.

3.7.1 Language

O modelo `Language` foi criado para gerir informações relacionadas às línguas suportadas pela plataforma. Este modelo é essencial para as operações de tradução.

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Artisan;

class Language extends Model
{
    use HasFactory;

    protected $connection = 'posts';
    protected $table = 'language';
    protected $primaryKey = 'id_language';
    protected $fillable = ['language','language_voice'];

}
```

Listing 3.7: Modelo Language

3.7.2 Post_Translated

O modelo `Post_Translated` desempenha um papel central na manipulação de traduções de publicações. Ele armazena dados como título traduzido, conteúdo traduzido e status de validação.

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post_translated extends Model
{
    use HasFactory;

    protected $connection = 'posts';
    protected $table = 'post_translated';
    protected $primaryKey = 'id_post_translated';
    protected $fillable = ['id_post', 'id_language', 'titulo', 'conteudo', 'validacao',
    'comentario_validacao', 'id_user'];


    public function post()
    {
        return $this->belongsTo(Post::class, 'id_post', 'id_post');
    }

    public function user()
    {
        return $this->belongsTo(User::class, 'id_user', 'id_user');
    }

    public function language()
    {
        return $this->belongsTo(Language::class, 'id_language', 'id_language');
    }
}
```

Listing 3.8: Modelo Post_Translated

3.7.3 PostApiId

O modelo `PostApiId` foi desenvolvido para mapear e associar IDs de posts na API externa com os IDs correspondentes no sistema. Isso é crucial para garantir consistência e integridade nas operações de tradução.

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class PostApiId extends Model
{
    use HasFactory;

    protected $connection = 'posts';
    protected $table = 'postsAPI_IDs';
    protected $primaryKey = 'id_postsAPI_ID';
    protected $fillable = ['local_post_id', 'api_post_id', 'api_translated_post_id'];
}
```

Listing 3.9: Modelo PostApiId

3.7.4 PostInteraction

O modelo `PostInteraction` foi criado para registar interações específicas relacionadas aos posts. Ele armazena informações sobre ações como eliminação de tradução ou validação bem-sucedida, proporcionando um histórico detalhado das atividades na plataforma.

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class PostInteraction extends Model
{
    use HasFactory;

    protected $connection = 'posts';
    protected $table = 'post_interactions';
    protected $primaryKey = 'id_post_interactions';
    protected $fillable = ['id_post', 'action'];

    public function post()
    {
        return $this->belongsTo(Post::class, 'id_post');
    }
}
```

Listing 3.10: Modelo/Estrutura da tabela De Traduções

A criação desses modelos e controladores é um passo crucial para garantir que a API atenda aos requisitos dos casos de uso designados. Os modelos são estruturados para armazenar dados relevantes, enquanto os controladores fornecem uma interface eficiente para interagir com esses dados. Essa abordagem modular e orientada a casos de uso contribui para um sistema coeso e adaptável.

3.8 Escolha das Rotas

Todas as rotas escolhidas para a api foram cuidadosamente criadas de modo a não haver confusão entre as rotas de modo a facilitar todo o acesso a essa mesmas rotas. Com base nesta tabela consegue se ver todas as rotas criadas, sendo que algumas servirão para os casos de uso do colega David, outras para os meus casos de uso, assim como casos mútuos.

API	Description	Request body	Response body
`POST /login`	User authentication	`{username, password}`	`{token}`
`POST /register`	User registration	`{username, email, password}`	`{user}`
`GET /fetch-languages`	Fetch and store languages	None	`{languages}`
`POST /send-post`	Send a post	`{postContent}`	`{post}`
`POST /send-post-translate/{api_post_id}/{id_language}`	Send post to translate	`{postContent}`	`{translatedPost}`
`GET /moderation/{postId}`	Get moderation status	None	`{moderationStatus}`
`GET /delete/{postId}`	Get delete status	None	`{deleteStatus}`
`POST /delete-post/{postId}`	Delete a post	None	`{deleteStatus}`
`POST /validation/{postId}`	Post validation	None	`{validationStatus}`
`GET /moderations-translation/{postId}`	Get moderation status for translation	None	`{moderationStatus}`
`GET /validate-by-id/{postId}`	Get validation status by ID	None	`{validationStatus}`
`GET /dashboard`	Get dashboard data	None	`{dashboardData}`
`GET /reports`	Get reports	None	`{reports}`
`GET /posts`	Get all posts	None	`{posts}`
`GET /posts/{id}`	Get a post by ID	None	`{post}`
`POST /posts`	Add a new post	`{postContent}`	`{post}`
`PUT /posts/{id}`	Update a post	`{postContent}`	None
`DELETE /posts/{id}`	Delete a post	None	None
`POST /posts/{postId}/report`	Report a post	None	`{reportStatus}`
`GET /posts/create`	Create a post	None	`{postCreationStatus}`
`GET /post/translation`	Get all post translations	None	`{postTranslations}`
`GET /post/translation/{id}`	Get a post translation by ID	None	`{postTranslation}`
`POST /post/translation/{id}`	Add a new post translation	`{translationContent}`	`{postTranslation}`
`PUT /post/translation/{id}`	Update a post translation	`{translationContent}`	None
`DELETE /post/translation/{id}`	Delete a post translation	None	None
`GET /topics`	Get all topics	None	`{topics}`
`GET /topics/{id}`	Get a topic by ID	None	`{topic}`
`POST /topics`	Add a new topic	`{topicContent}`	`{topic}`
`PUT /topics/{id}`	Update a topic	`{topicContent}`	None
`DELETE /topics/{id}`	Delete a topic	None	None

Tabela 3.2: API rotas

3.9 Desenvolvimento da App MVC

Com a parte dos controladores da API explicados assim como os seus modelos e rotas, passamos então para os controladores WEB, isto é, aqueles que fazem o sistema ter vistas de modo a que os utilizadores possam mexer no nosso website com a melhor qualidade possível.

3.9.1 Decisões de implementação

Modelos Existentes na APP MVC

A app mvc como referido no desenvolvimento da api, tem um conjunto de modelos que ajudaram que todo o sistema funciona se de forma correta. Os modelos de seguida referenciados, englobam todos os casos de uso que a aplicação web contem.

- Modelo Language
- Modelo Post_Translated
- Modelo Post
- Modelo PostApiId
- Modelo PostDelete
- Modelo Post Interaction
- Modelo Post Report
- Modelo UDVote
- Modelo User

Criação de um Sistema de Login e Suas Vistas

Para garantir a segurança do acesso aos dados da API, no sistema foi implementado autenticação utilizando o sistema de tokens do Laravel Sanctum. Apenas utilizadores autenticados têm permissão para aceder a certos conteúdos provenientes na api e no sistema. Além disso, foram implementadas políticas de autorização para controlar o acesso a recursos específicos com base nas permissões do utilizador autenticado.

Foram Criados/Usados dois controladores

AuthController

O `AuthController` é responsável por gerir a autenticação, registo e logout de utilizadores na aplicação.

`signin(Request $request)`

- Este método lida com a autenticação do utilizador.
- Utiliza `Auth::attempt` para tentar autenticar o utilizador com base no email e password fornecidos no `Request`.
- Se a autenticação for bem-sucedida, gera um token de acesso, recupera o nome do utilizador e redireciona para a rota 'home'.
- Se a autenticação falhar, retorna um erro indicando que a autenticação não foi autorizada.

`signup(Request $request)`

- Este método lida com o processo de registo de um novo utilizador.
- Utiliza o `Validator` para validar os campos do formulário.
- Se a validação falhar, retorna um erro com as mensagens de validação.
- Se a validação for bem-sucedida, cria um novo utilizador na base de dados, encriptando a password.
- Gera um token de acesso para o novo utilizador e redireciona para a página de login com uma mensagem de sucesso.

`logout(Request $request)`

- Este método lida com o processo de logout do utilizador.
- Utiliza `Auth::logout()` para efetuar o logout.
- Invalida a sessão atual, regenera o token da sessão e redireciona para a página inicial.

`showLoginForm()` e `showRegisterForm()`

- `showLoginForm()` retorna a view para o formulário de login.
- `showRegisterForm()` retorna a view para o formulário de registo.

BaseController

O BaseController é um controlador destinado a fornecer métodos comuns para controladores derivados no contexto de autenticação (AUTH)

- **Classe `BaseController`:**
  - A classe `BaseController` é uma classe de base para outros controladores no namespace `AUTH`.
- **Método `sendResponse`:**
  - Este método é responsável por criar e enviar uma resposta JSON de sucesso.
  - Recebe dois parâmetros: `$result` (dados a serem incluídos na resposta) e `$message` (mensagem de sucesso).
  - Retorna uma resposta JSON com status 200 (OK).
- **Método `sendError`:**
  - Este método é responsável por criar e enviar uma resposta JSON de erro.
  - Recebe três parâmetros: `$error` (mensagem de erro), `$errorMessages` (detalhes adicionais de erro, opcional) e `$code` (código de status HTTP, padrão é 404).
  - Retorna uma resposta JSON com o código de status fornecido e possíveis mensagens de erro adicionais.

Ambos os métodos `sendResponse` e `sendError` são úteis para estruturar as respostas JSON em controladores, simplificando a lógica do return de sucesso ou erro. Eles encapsulam o formato comum das respostas e podem ser reutilizados em vários pontos da aplicação.

Com a adição destes controlador, é novamente importante conter um modelo que funcione e esteja de encontro com o pretendido. Segue a explicação do modelo criado, para a fase de autenticação.

Modelo User

- **Configuração da Conexão e Chave Primária:**
  - `$connection`: Especifica a conexão da base de dados que esta model representa.
  - `$primaryKey`: Define o nome da chave primária para a tabela.
  - `$table`: Especifica o nome da tabela associada ao modelo.
- **Traits Utilizadas:**
  - `HasApiTokens`: Fornece métodos para autenticação via API usando Laravel Sanctum.
  - `HasFactory`: Fornece um método de fábrica para a criação de registos do modelo.
  - `Notifiable`: Adiciona suporte para notificações.
- **Atributos `$fillable`, `$hidden` e `$casts`:**
  - `$fillable`: Define os atributos que podem ser atribuídos.
  - `$hidden`: Especifica os atributos que devem ser ocultados.
  - `$casts`: Define a conversão de tipos para determinados atributos.
- **Relações:**
  - `posts()`: Define a relação de um utilizador para muitos posts.
  - `votes()`: Define a relação de um utilizador para muitos votos.

Este modelo é essencial para gerir informações dos utilizadores no sistema, incluindo suas interações com posts e votos.

```php
<?php
namespace App\Models;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
// sanctum
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{

    protected $connection = 'users';
    protected $primaryKey = 'id_user';
    protected $table = 'users';
    use HasApiTokens, HasFactory, Notifiable;
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        'moderator',
    ];
    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function posts()
    {
```

```
48          return $this->hasMany(Post::class, 'id_user', 'id_user');
49      }
50
51      public function votes()
52      {
53          return $this->hasMany(Vote::class, 'id_user', 'id_user');
54      }
55
56  }
```

Listing 3.11: Modelo User

Vista Login e Registo

Foram criadas 2 vistas bastante coesas e objetivas, com o objetivo do utilizador ter uma boa experiências com o sistema.

View - Login

A página é limpa e fácil de usar, com um formulário centralizado para a inserção de email e password. Se o utilizador já tentou fazer login antes e houve um sucesso ou erro, uma mensagem correspondente será exibida.

O design da página inclui um convite para o registo, com um link direto para a página de registo, incentivando novos utilizadores a criar uma conta. Em resumo, a visualização proporciona uma experiência direta e intuitiva para os utilizadores que desejam acessar o sistema.

```
1   @extends('layout')
2       <title>Login</title>
3       @section('content')
4       @if(session('success'))
5           <div class="alert alert-success">{{ session('success') }}</div>
6       @elseif(session('error'))
7           <div class="alert alert-danger">{{ session('error') }}</div>
8       @endif
9
10  <body class="bg-light">
11      <div class="container">
12          <div class="row justify-content-center">
13              <div class="col-md-6">
14                  <div class="card my-5">
15                      <div class="card-body">
16                          <h2 class="card-title text-center mb-4">Login</h2>
17                          <form action="{{ route('login') }}" method="post">
```

```
18                          @csrf
19                          <div class="mb-3">
20                              <label for="email" class="form-label">Email:</label>
21                              <input type="email" name="email" class="form-control"
22                              required>
23                          </div>
24                          <div class="mb-3">
25                              <label for="password" class="form-label">Senha:</label>
26                              <input type="password" name="password" class="form-control"
27                              required>
28                          </div>
29                          <div class="d-grid">
30                              <button type="submit" class="btn btn-primary">Entrar
31                              </button>
32                          </div>
33                      </form>
34                      <p class="mt-3 text-center">Ainda não tem uma conta?
35                      <a href="{{ route('register') }}">Crie uma agora.</a></p>
36                  </div>
37              </div>
38          </div>
39      </div>
40  </div>
41
42  @endsection
```

Listing 3.12: View Login

View - Register

Esta view possui um formulário simples para inserir nome, email e password. Há também a opção de se tornar moderador. Mensagens de sucesso ou erro podem ser exibidas. Após preencher, basta clicar em "Registrar". Se já possui uma conta, pode fazer login (link de login).

```
@extends('layout')
<title>Registrar</title>

@section('content')
    @if(session('success'))
        <div class="alert alert-success">{{ session('success') }}</div>
    @elseif(session('error'))
        <div class="alert alert-danger">{{ session('error') }}</div>
    @endif
    <div class="container mt-5">
        <h2 class="mb-4">Registrar</h2>

        <form action="{{ route('register') }}" method="post">
            @csrf
            <div class="mb-3">
                <label for="name" class="form-label">Nome:</label>
                <input type="text" name="name" class="form-control" required>
            </div>
            <div class="mb-3">
                <label for="email" class="form-label">Email:</label>
                <input type="email" name="email" class="form-control" required>
            </div>
            <div class="mb-3">
                <label for="password" class="form-label">Senha:</label>
                <input type="password" name="password" class="form-control" required>
            </div>
            <div class="mb-3">
                <label for="confirm_password" class="form-label">Confirmar Senha:</label>
                <input type="password" name="confirm_password" class="form-control"
                required>
            </div>
            <div class="mb-3 form-check">
                <input type="checkbox" name="moderator" class="form-check-input"
                value="1">
                <label for="moderator" class="form-check-label">Moderador</label>
            </div>
            <button type="submit" class="btn btn-primary">Registrar</button>
        </form>

        <p class="mt-3">Já tem uma conta? <a href="{{ route('login') }}">
```

```
41          Faça login.</a></p>
42      </div>
43
44  @endsection
```

Listing 3.13: View Registo

Criação do Web Controller e suas Vistas Referente ao Caso de Uso - Traduzir uma Publicação

Este Controlador possui funcionalidades relacionadas a criação, leitura, atualização e exclusão de traduções (Crud Methods), onde vai instanciar os métodos do controlador da API, referente a traduções.

O controlador contém os seguintes métodos:

- `index()`
  Método para exibir a página inicial de traduções. Chama o método 'index' do controlador da API de tradução.
- `show($id)`
  Método para exibir detalhes de uma tradução específica. Chama o método 'show' da API de tradução e passa os dados para a view 'translations.showTranslation'.
- `create()`
  Método para exibir o formulário de criação de traduções.
- `store($request, $id)`
  Método para armazenar uma nova tradução. Chama o método 'store' da API de tradução e redireciona com mensagens de sucesso ou erro.
- `edit($id)`
  Método para exibir o formulário de edição de uma tradução existente.
- `update($request, $id)`
  Método para atualizar uma tradução. Chama o método 'update' da API de tradução.
- `destroy($id)`
  Método para excluir uma tradução. Chama o método 'destroy' da API de tradução.

Vista para a Criação de uma Tradução

Para a criação de uma tradução foi criado um formulário para o efeito, onde o utilizador ao clicar em Traduzir iria aparecer lhe uma opção para selecionar qual a língua que deseja traduzir. Este formulário foi criado dentro da view associado ao método show dos posts criada mais para o efeito dos casos de uso do colega David, onde se vem em detalhe o post.

View representativa das Traduções e suas Permissões tendo em conta tipo de utilizador

Primeiramente na view , foi adicionado um botão no qual a ideia passava por , caso seja clicado iria aparecer uma formulário para efetivamente realizar a tradução. Para isso foi usado um básico script de javascript para o efeito. Este formulário procura um ambiente simples e intuitivo ao utilizador assim como melhorar a usabilidade de todo o sistema. Foi também adicionada umas condições a nível dos utilizadores, de modo a que só os utilizadores que não tenham criado o post possam efetivamente traduzi-la. Foi também colocado que caso seja um utilizador seja moderador não tenha acesso a traduzir publicações.

```
@auth
        @if(auth()->user()->id_user !== $post->user->id_user)
            @if(!auth()->user()->moderator)
            <div class="d-flex justify-content-center align-items-center">
                <button id="translateButton" class="btn btn-primary d-block mt-3">
                Traduzir</button>
            </div>
            @endif
        @endif
```

Listing 3.14: View show - botão traduzir

```
<form id="translationForm" action="{{ route('posts.translation.store', ['id' =>
$post->id_post]) }}" method="post" style="display: none; text-align: center;">
            @csrf
            <select name="id_language" class="form-select mb-3">
                @foreach($languages as $language)
                    <option value="{{ $language['id_language'] }}">{{
                    $language['language'] }}</option>
                @endforeach
            </select>
            <button type="submit" class="btn btn-success btn-sm">
            Concluir Tradução</button>
        </form>
    @endauth
```

```
14
15      <script>
16          $(document).ready(function() {
17              $("#translateButton").click(function() {
18                  $("#translationForm").toggle();
19              });
20          });
21      </script>
```

Listing 3.15: View show - Formulário Tradução

Este Web Controller usa os seguintes modelos

- Modelo - Language;
- Modelo - Post_Translated;
- Modelo - PostApiId;
- Modelo - PostInteraction.

Este Web Controller tem as rotas web

- /posts-translation;
- /posts-translation/create;
- /posts/id/translate';
- /posts-translation/id';
- /posts-translation/id/edit';
- /posts-translation/id;
- /posts-translation/id;

Criação do Web Controller e suas Vistas Referente ao Caso de Uso - Validação de uma Publicação

Este Controlador possui funcionalidades relacionadas com o Controlador da API (API-ModControler) referido anteriormente, onde vai instanciar os métodos do controlador da API, referente à validação.

O controlador contém os seguintes métodos:

- Cria uma instância do controlador `APIModController`.
- Chama o método `moderations_translation($postId)` no `APIModController`.
- Obtém a resposta da API e extrai os dados traduzidos do post.
- Retorna uma view chamada `mod.moderation_Translation`, passando os posts traduzidos como dados.

Function `validateById($postId)`

- Cria uma instância do controlador `APIModController`.
- Chama o método `validateById($postId)` no `APIModController`.
- Obtém a resposta da API e extrai os dados do post validado.
- Retorna uma view chamada `mod.validation`, passando os posts validados como dados.

Function `validate_Translation($request, $postId, $apiModController)`

- Chama o método `Validation($request, $postId)` no `$apiModController`.
- Verifica se o código de status da resposta da API é 201 (Created).
- Se for, redireciona para a rota `posts.moderation_translation` com uma mensagem de sucesso.
- Se não for, redireciona para a rota `home` com uma mensagem de erro.

Vista para a Validação das Traduções

Para a validação das traduções foi criado um formulário para o efeito, onde o utilizador ao clicar em validar tradução iria redirecioná lo para o formulário referido.

View representativa do botão para validar

Esta view, onde foi colocado um botão para validar, é a view do método show dos posts , onde só aparece o botão caso o utilizador autenticado seja o moderador.

```
@if(auth()->user()->moderator)
            <div class="d-flex justify-content-center align-items-center">
                <a href="{{ route('posts.moderation_translation', ['id'
                => $post->id_post]) }}" class="btn btn-warning btn-n d-block mx-auto mt-3">
                    <i class="fas fa-edit me-2"></i> Validar Tradução
                </a>
            </div>
        @endif
```

Listing 3.16: View show - Botão Validação

View representativa das Traduções Não Validadas

Primeiramente na view , aos serem listadas todas as traduções da publicações não validadas foi adicionado um botão no qual a ideia passava por , caso seja clicado iria para a página referente a validações para efetivamente realizar a validação ou exclusão da tradução.

```
@extends('layout')
<title>Moderar Post</title>

@section('content')
    @if(session('success'))
        <div class="alert alert-success">{{ session('success') }}</div>
    @elseif(session('error'))
        <div class="alert alert-danger">{{ session('error') }}</div>
    @endif

    <div class="container mt-5">
        <h2 class="mb-4">Moderar Post</h2>
        <hr>

        @if($posts_translated->isEmpty())
            <p>Não existe traduções para validar.</p>
        @else
            @foreach ($posts_translated as $translatedPost)
```

```
19                @if ($translatedPost->validacao != 1)
20                    <div class="border p-3 mb-3">
21                        <p class="mb-2">Title: {{ $translatedPost->titulo }}</p>
22                        <p class="mb-2">Content: {{ $translatedPost->conteudo }}</p>
23                        <p class="mb-2">Author: {{ $translatedPost->user->name }}</p>
24                        <p class="mb-2">Created at: {{ $translatedPost->created_at }}</p>
25                        <p class="mb-2">Lingua Traduzida: {{ $translatedPost->
26                        language->language }}</p>
27
28                        <form action="{{ route('posts.validateById', ['id' =>
29                        $translatedPost->id_post]) }}" method="get">
30                            @csrf
31                            <button type="submit" class="btn btn-success">
32                            Validar</button>
33                        </form>
34                        <hr>
35                    </div>
36                @endif
37            @endforeach
38        @endif
39    </div>
40
41    <a href="{{ route('home') }}" class="btn btn-secondary mt-3">
42    Voltar para a lista de posts</a>
43
44 @endsection
```

Listing 3.17: View que lista traduções não validadas por post

View representativa das Validações

Na view representativa das validações, existe um conjunto de opções, onde inicialmente é mostrado qual a tradução que está para ser validada, assim como uma checkbox para se for clicada, quererá dizer que o moderador valida a tradução, podendo colocar um comentário. Caso não selecione a checkbox e finalizar a validação, a tradução é eliminada.

```
1  @extends('layout')
2  <title>Validar Post - {{ $posts_translated->titulo }}</title>
3
4  @section('content')
5      @if(session('success'))
6          <div class="alert alert-success">{{ session('success') }}</div>
7      @elseif(session('error'))
8          <div class="alert alert-danger">{{ session('error') }}</div>
9      @endif
10
```

```
<div class="container mt-5">
    <h2>Validar {{ $posts_translated->titulo }}</h2>

    <h3>Título: {{ $posts_translated->titulo }}</h3>
    <p>Conteúdo: {{ $posts_translated->conteudo }}</p>
    <p>Autor: {{ $posts_translated->user->name }}</p>
    <p>Criado em: {{ $posts_translated->created_at }}</p>

    <form action="{{ route('posts.validate_Translation', ['id' =>
    $posts_translated->id_post]) }}" method="post"
        class="mt-3">
        @csrf

        <div class="mb-3">
            <label for="validacao" class="form-label">Validação:</label>
            <div class="form-check">
                <input type="hidden" name="validacao" value="0">
                <input type="checkbox" name="validacao" id="validacao" class=
                "form-check-input" value="1">
                <label for="validacao" class="form-check-label">Validar</label>
            </div>
        </div>

        <div id="comentario_container" class="mb-3" style="display:none;">
            <label for="comentario_validacao" class="form-label
            Comentário Validação:</label>
            <textarea name="comentario_validacao" id="comentario_validacao"
            class="form-control" rows="4"
                cols="50"></textarea>
        </div>

        <button type="submit" class="btn btn-primary">Finalizar</button>
    </form>

    <script>
        document.getElementById('validacao').addEventListener('change', function () {
            var comentarioContainer = document.getElementById('comentario_container');
            if (this.checked) {
                comentarioContainer.style.display = 'block';
            } else {
                comentarioContainer.style.display = 'none';
            }
        });
    </script>

    <a href="{{ route('home') }}" class="btn btn-secondary mt-3">
    Voltar para a lista de posts</a>
```

```
58      </div>
59
60  @endsection
```

Listing 3.18: View formulário validação

Este Web Controller usa os seguintes modules com base na API

- Modelo - Language;
- Modelo - Post_Translated;
- Modelo - PostInteraction.

Este Web Controller tem as rotas web

- /posts/id/moderation_translation;
- /posts/id/moderationTranslation/validate;
- /posts/id/moderationTranslation/validate;

3.9.2 Criação dos Web Controller e suas Vistas Referente ao Caso de Uso - Dados Estatísticos

Este Controlador possui funcionalidades relacionadas ao visionamento de métricas relativas a publicações criadas, e todas as interações que os utilizadores teem com elas. Cliques em ver mais, numero de reports, numero de traduções por dia, numero de traduções validadas e numero de traduções eliminadas.

Controlador do dashbord do Utilizador

O `UserController` é um controlador web onde é responsável por lidar com as ações relacionadas ao utilizador, especialmente aquelas relacionadas ao (`dashboard`).

- **Método `dashboard()`:**
  Este método representa a ação associada à exibição do dashboard do utilizador:
  - **Autenticação do utilizador:**
    Utiliza `Auth::user()` para obter o utilizador autenticado. Isso pressupõe que apenas utilizadores autenticados podem acessar ao dashboard.
  - **Consulta de Posts do utilizador:**
    Obtém os posts associados ao utilizador autenticado utilizando `$user->posts()` e carrega também as relações de votos (`votes`) associadas a esses posts.
  - **Renderização da View:**
    Retorna a view `dashboard.user` com os posts do utilizador passados como variável (`compact('userPosts')`).

```php
<?php

namespace App\Http\Controllers\WEB;

use App\Http\Controllers\WEB\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\User;
use App\Models\Post;

class UserController extends Controller
{
    public function dashboard()
    {
        $user = Auth::user();
        $userPosts = $user->posts()->with('votes')->get();
        return view('dashboard.user', compact('userPosts'));
    }
}
```

Listing 3.19: View Dashboard

Método ShowGraph no controlador WebPostController

Este método tem como objetivo criar gráficos interativos tendo em conta as métricas de cada post.

- **Método `showGraph()`:** O método `showGraph` é responsável por apresentar um gráfico visualizando as interações de uma publicação específica. A seguir, estão as etapas realizadas por este método:
  - **Obtenção da publicação:** Obtém a publicação correspondente ao ID fornecido utilizando o modelo `Post` e a função `find`.
  - **Obtenção da Métrica de Interesse:** Verifica a solicitação HTTP para obter o parâmetro chamado `metric`, utilizado para filtrar as interações relacionadas à publicação. Se não estiver presente, é atribuído um valor padrão de 'view_more'.
  - **Obtenção de Interactions:** Obtém as interações associadas à publicação e à métrica de interesse utilizando o modelo `PostInteraction`.
  - **Agrupamento e Contagem por Data:** Agrupa as interações por data e calcula o número de interações para cada dia.
  - **Preparação de Dados para o Gráfico:** Organiza os dados necessários para renderizar o gráfico.
  - **Renderização da View:** Retorna uma view chamada 'posts.graph', passando dados com as datas e contagem de interações para cada métrica.

```php
public function showGraph(Request $request, $id)
    {
        $post = Post::find($id);
        $metric = $request->input('metric', 'view_more', 'report', 'tradução',
        'tradução_validada', 'tradução_eliminada');
        $interactions = PostInteraction::where('id_post', $id)
            ->where('action', $metric)
            ->get();

        $clicks = $interactions->groupBy(function ($interaction) {
            return $interaction->created_at->format('d-m-Y');
        })->map->count();

        $labels = $clicks->keys()->toArray();

        return view('posts.graph', compact('labels', 'clicks', 'post', 'metric'));
    }
```

Listing 3.20: Método showgraph

Vista para visualização dos dados estatísticos relativos às publicações

Para a visualização dos dados estatísticos foram criadas 2 views, onde no dashbord aparecia um lista com os posts que o utilizador tinha , assim como os dados relativo a traduções, validações etc. É acompanhado de gráficos de barras para demonstrar as métricas por dia.

View representativa do Dashboard

Esta view tem como objetivo listar as publicaçoes com a informaçoes das suas métricas

```
@extends('layout')

@section('content')
    @if(session('success'))
        <div class="alert alert-success">{{ session('success') }}</div>
    @elseif(session('error'))
        <div class="alert alert-danger">{{ session('error') }}</div>
    @endif

    <div class="container mt-5 d-flex justify-content-center align-items-center">
        <div>
            <h1 class="text-center fw-bold">Dashboard do {{ auth()->user()->name }}</h1>

            <h2 class="mt-4 text-center">Os Teus Posts</h2>

            @foreach($userPosts as $post)
                <div class="card mb-3 mx-auto" style="max-width: 800px;">
                    <div class="card-header d-flex justify-content-between">
                        <div class="d-flex align-items-center">
                            <i class="fas fa-user me-2"></i>
                            <p class="card-text fs-5">{{ $post->user->name }}</p>
                        </div>
                        <div class="d-flex">
                            <p class="card-text me-2">
                                <i class="fas fa-thumbs-up text-success"></i>
                                {{ $post->votes ? $post->votes->where('vote', 'up')
                                ->count() : 0 }}
                            </p>
                            <p class="card-text">
                                <i class="fas fa-thumbs-down text-danger"></i>
                                {{ $post->votes ? $post->votes->where('vote', 'down')
                                ->count() : 0 }}
                            </p>
                        </div>
                    </div>
                    <div class="card-body p-4">
                        <h5 class="card-title text-center mb-3 fs-4">{{ $post->titulo }}
                        </h5>
```

```
39                      <p class="card-text fs-6">{{ $post->conteudo }}</p>
40                      <p class="card-text">Lingua: {{ $post->language ? $post->
41                      language->language : 'N/A' }}</p>
42                  </div>
43
44                  <div class="text-center mx-auto">
45                      <h4 class="mt-3 fw-bold">Métricas de Interação</h4>
46                      <p class="card-text">Cliques em "Ver mais": {{
47                      $post->interactions ? $post->interactions->where('action',
48                      'view_more')->count() : 0 }}</p>
49                      <p class="card-text">Traduções: {{
50                      $post->interactions ? $post->interactions->where('action',
51                      'traducao')->count() : 0 }}</p>
52                      <p class="card-text">Traduções Validada: {{
53                      $post->interactions ? $post->interactions->where('action',
54                      'traducao_validada')->count() : 0 }}</p>
55                      <p class="card-text">Traduções Eliminadas: {{
56                      $post->interactions ? $post->interactions->where('action',
57                      'traducao_eliminada')->count() : 0 }}</p>
58                      <p class="card-text">Reports: {{
59                      $post->interactions ? $post->interactions->where('action',
60                      'report')->count() : 0 }}</p>
61                      <a href="{{ route('posts.graph', ['id' => $post->id_post]) }}
62                      " class="btn btn-primary mt-3 mb-3">Ver Gráfico</a>
63                  </div>
64          </div>
65              @endforeach
66
67              <div class="d-flex justify-content-center align-items-center">
68                  <a href="{{ route('home') }}" class="btn btn-secondary mt-3">
69                  Voltar para a lista de posts</a>
70              </div>
71          </div>
72      </div>
73  @endsection
```

Listing 3.21: View Dashboard

View representativa do gráfico de métricas

Esta view tem como objetivo criar gráficos interativos para que o utilizador possa ver as métricas de um post por ele criado ao longo do dia.

```
@extends('layout')
<title>Gráfico - {{ ucfirst($metric) }}</title>

@section('content')
    @if(session('success'))
        <div class="alert alert-success">{{ session('success') }}</div>
    @elseif(session('error'))
        <div class="alert alert-danger">{{ session('error') }}</div>
    @endif

    <div class="container mt-5">
        <h2 class="mb-4 fw-bold">Gráfico - {{ ucfirst($metric) }}</h2>

        <form action="{{ route('posts.graph', ['id' => $post->id_post]) }}"
        method="get" class="mb-3">
            <label for="metric" class="form-label">Escolha a métrica:</label>
            <select name="metric" id="metric" class="form-select">
                <option value="view_more" {{ $metric === 'view_more' ? 'selected'
                : '' }}>Ver Mais</option>
                <option value="report" {{ $metric === 'report' ? 'selected' :
                '' }}>Report</option>
                <option value="tradução" {{ $metric === 'traducao' ? 'selected' :
                '' }}>Traduções</option>
                <option value="tradução_validada" {{ $metric === 'traducao_validada'
                ? 'selected' : '' }}>Tradução_validada</option>
                <option value="tradução_eliminada" {{ $metric === 'traducao_eliminada'
                ? 'selected' : '' }}>Tradução_eliminada</option>
            </select>
            <div class="d-flex justify-content-center align-items-center">
                <button type="submit" class="btn btn-primary mt-3">
                Atualizar Gráfico</button>
            </div>
        </form>

        <canvas id="interactionChart" width="400" height="200"></canvas>
        <div class="d-flex justify-content-center align-items-center">
            <a href="{{ route('user.dashboard') }}" class="btn btn-secondary mt-3">
            Voltar à Dashboard do {{ auth()->user()->name }}</a>
        </div>
    </div>

    <script src="/bootstrap-5.3.2-dist/js/bootstrap.bundle.js"></script>
    <script>
```

```
44          var ctx = document.getElementById('interactionChart').getContext('2d');
45          var interactionChart = new Chart(ctx, {
46              type: 'bar',
47              data: {
48                  labels: {!! json_encode($labels) !!},
49                  datasets: [{
50                      label: '{{ ucfirst($metric) }}',
51                      data: {!! json_encode($clicks->values()) !!},
52                      backgroundColor: 'rgba(75, 192, 192, 1)',
53                      borderColor: 'rgb(0, 0, 0)',
54                      borderWidth: 2,
55                      fill: false
56                  }]
57              },
58              options: {
59                  scales: {
60                      y: {
```

Listing 3.22: View Gráfico de métricas

Este Web Controller usa os seguintes com base na API

- Modelo - Posts;
- Modelo - Post_Translated;
- Modelo - PostInteraction.

Este Web Controller tem as rotas web

- /dashboard/user;
- /posts/id/grafico;

Capítulo 4

Conclusão

Em conclusão, a implementação da aplicação web Forueddit foi bem-sucedida, apoiada por uma API própria desenvolvida em Laravel e um modelo de arquitetura MVC abrangente.

A escolha do Laravel para o desenvolvimento da API proporcionou uma estrutura robusta e flexível, permitindo uma comunicação eficiente entre o frontend e o backend. A arquitetura MVC, por sua vez, demonstrou ser crucial para a organização e escalabilidade do projeto, garantindo uma manutenção facilitada e uma separação clara de responsabilidades entre os diferentes componentes da aplicação.

A presença de diferentes tipos de utilizadores, desde os utilizadores anónimos até aos moderadores dedicados, destaca a capacidade da aplicação de acomodar uma ampla variedade de interações online. A inclusão de funcionalidades como criação de publicações, tradução e moderação, proporciona uma experiência rica e dinâmica aos utilizadores, promovendo a participação ativa e a criação de uma comunidade envolvente. Por fim referir que o projeto poderia sim estar melhor, mais no sentido de como os web controllers se comunicam com a api. Houve alguma dificuldade em usar as rotas da api, portanto foi decidido instanciar os métodos da api em vez de usar as rotas. Numa forma geral o projeto vai de encontro com o que nos propusemos , portanto é motivo para estar satisfeito.