News Article Comparison and Sentiment Analysis System

1. Introduction

This project implements a news article comparison and sentiment analysis system. The system fetches news articles from a Macedonian news aggregator (Time.mk) and then compares them for similarity using a natural language processing (NLP) approach. Additionally, it performs sentiment analysis on the news articles to determine the tone of each article. The primary technologies used in this project include Laravel, Python, and several libraries for NLP and sentiment analysis.

The system supports several functionalities, including fetching news, comparing articles, analyzing sentiment, and allowing users to like, bookmark, and store articles. This report outlines the project's architecture, key components, implementation details, and the challenges encountered.

2. Project Architecture

The project is built using a combination of Laravel for web services and Python scripts for text processing. Below are the major components:

- News Fetching Module (Python): Scrapes Macedonian news articles.

- Comparison Module (Python): Uses machine learning models for comparing the similarity of news articles and custom user-provided texts.

- Sentiment Analysis Module (Python): Analyzes the sentiment of each article.

- User Interface (Laravel + Blade): Allows users to interact with articles, view results, and manage bookmarks and likes.

3. System Components and Implementation

3.1 Controllers in Laravel

The heart of the system's backend is implemented in Laravel controllers, which handle routing, data management, and interactions with Python scripts for NLP tasks. Here's a detailed breakdown of the controllers:

NewsController

The `NewsController` manages the flow of fetching, storing, comparing, and displaying news articles. It integrates with Python scripts for tasks like fetching news articles, comparing their content, and analyzing sentiment.

1. fetchNews()

This method fetches news from the scraper script (`fetch_news.py`), decodes the result, and stores the articles in the database. It also calls the `compare.py` Python script to compare the similarity of news articles.

```
public function fetchNews()
{    set_time_limit(300);
    $result =
shell_exec("C:\Users\Denica\PhpstormProjects\iisok_project\\venv\Scripts\python.exe
C:\Users\Denica\PhpstormProjects\iisok_project\python\\fetch_news.py");



    $data = json_decode($result, true);

    // Save articles to the database
    if (is_array($data)) {
        foreach ($data as $article) {
            NewsArticle::updateOrCreate(
                ['url' => $article['url']], // Ensure uniqueness
                [
```

```
                    'title' => $article['title'],
                    'content' => $article['content'],
                    'source' => $article['source']
                ]
            );
        }
    }

    $tempFilePath =
'C:\\Users\\Denica\\PhpstormProjects\\iisok_project\\storage\\app\\private\\temp_articles
.json';
    Storage::disk('local')->put('temp_articles.json', $result);

    $filePath = escapeshellarg($tempFilePath);
    // Construct the shell command with arguments
    $comparison =
escapeshellcmd("C:\Users\Denica\PhpstormProjects\iisok_project\\venv\Scripts\python.exe
C:\Users\Denica\PhpstormProjects\iisok_project\python\\compare.py $filePath");
    $output = shell_exec($comparison);

    //Execute the shell command
    $comparisonResults = json_decode($output, true);
    Storage::delete('temp_articles.json');

    // Check if the comparisonResults are valid
    if ($comparisonResults) {
        // Save comparison results to the database
        foreach ($comparisonResults as $result) {
            $article1 = NewsArticle::firstOrCreate(['title' => $result['news1_title'],
'content' => $result['news1_content']]);
            $article2 = NewsArticle::firstOrCreate(['title' => $result['news2_title'],
'content' => $result['news2_content']]);

            Comparison::updateOrCreate(
                [
                    'article1_id' => $article1->id,
                    'article2_id' => $article2->id
                ],
                ['similarity' => $result['similarity_original']]
            );
        }
    } else {
        return response()->json([
            'status' => 'error',
            'message' => 'Failed to process comparison results.',
        ], 500);
    }
```

```
    return view('news', [
        'fetchedNews' => $data,
        'news' => $comparisonResults
    ]);
}
```

2. showDetails($id)

Displays details for a specific article, including the sentiment analysis results and the top 10% most similar articles. The similarity comparison is performed dynamically if not already available in the database.

```
public function showDetails($id)
{
    set_time_limit(300);
    // Find the article by ID
    $article = NewsArticle::findOrFail($id);

    // Check if sentiment analysis has already been done
    if ($article->sentiment==null or $article->sentiment=='null') {
        // Perform sentiment analysis on the selected article

        $sentimentResult =
escapeshellcmd("C:\\Users\\Denica\\PhpstormProjects\\iisok_project\\venv\\Scripts\\python
.exe C:\\Users\\Denica\\PhpstormProjects\\iisok_project\\python\\sentiment_analysis.py
$article->content");
        $output = shell_exec($sentimentResult);

        // Decode the JSON result from Python (assuming the sentiment script returns
JSON)
        $sentiment = json_decode($output, true);
        if($sentiment==null or $sentiment == 'null'){
            return response()->json([
                'error' => 'Custom error message',
            ], 400);
        }
        // Save the sentiment analysis to the database
        $article->sentiment = json_encode($sentiment);
        $article->save();
    } else {
        // Sentiment already exists, retrieve from the database
        $sentiment = json_decode($article->sentiment, true);
    }

    // Retrieve the stored similarities for this article
```

```
    $similarArticles = $this->getTopSimilarArticles($article);

    return view('details', [
        'article' => $article,
        'sentiment' => $sentiment,
        'similarArticles' => $similarArticles,
    ]);
}
```

3. compareSavedArticles(Request $request)

Allows users to compare two saved articles. It uses a form where users select two articles, and the similarity score between them is calculated and displayed.

```
public function compareSavedArticles(Request $request)
{
    $article1 = NewsArticle::findOrFail($request->article1_id);
    $article2 = NewsArticle::findOrFail($request->article2_id);

    // Call the function to calculate similarity (could be from a Python script or
Laravel function

    $comparison = Comparison::where(function ($query) use ($article1, $article2) {
        $query->where('article1_id', $article1->id)
            ->where('article2_id', $article2->id);
    })->orWhere(function ($query) use ($article1, $article2) {
        $query->where('article1_id', $article2->id)
            ->where('article2_id', $article1->id);
    })->first();

    // If the comparison doesn't exist, do the comparison
    if (!$comparison) {
        $similarity = $this->compareArticles($article1->content, $article2->content);
        if($similarity==0.0){
            return response()->json([
                'error' => 'Custom error message',
            ], 400);
        }
        // Save the new comparison to the database
        $comparison = Comparison::create([
            'article1_id' => $article1->id,
            'article2_id' => $article2->id,
            'similarity' => $similarity,
        ]);
    }

    // Return the view with the similarity result
```

```
    return view('comparison_result', [
        'article1' => $article1,
        'article2' => $article2,
        'similarity' => $comparison->similarity
    ]);
}
```

5. getTopSimilarArticles($article)

The system dynamically determines the top 10% most similar articles for each article based on a comparison score. This functionality ensures that users can explore similar content across multiple news articles, providing additional context or alternative viewpoints on a topic.

The process of determining the top 10% involves iterating through the articles stored in the database, comparing them against the currently viewed article, and selecting the articles with the highest similarity scores. This is achieved using Python scripts integrated into Laravel through shell execution. Here's how the system sorts and retrieves similar articles:

```
protected function getTopSimilarArticles($article)
{
    $allArticles = NewsArticle::where('id', '!=', $article->id)->get();
    $similarArticles = [];

    foreach ($allArticles as $otherArticle) {
        // Check if the comparison already exists in the database
        $comparison = Comparison::where(function ($query) use ($article, $otherArticle) {
            $query->where('article1_id', $article->id)
                ->where('article2_id', $otherArticle->id);
        })->orWhere(function ($query) use ($article, $otherArticle) {
            $query->where('article1_id', $otherArticle->id)
                ->where('article2_id', $article->id);
        })->first();

        // If the comparison doesn't exist, do the comparison
        if (!$comparison) {
            $similarity = $this->compareArticles($article->content, $otherArticle-
>content);
            if($similarity==0.0){
                return response()->json([
                    'error' => 'Custom error message',
                ], 400);
            }
```

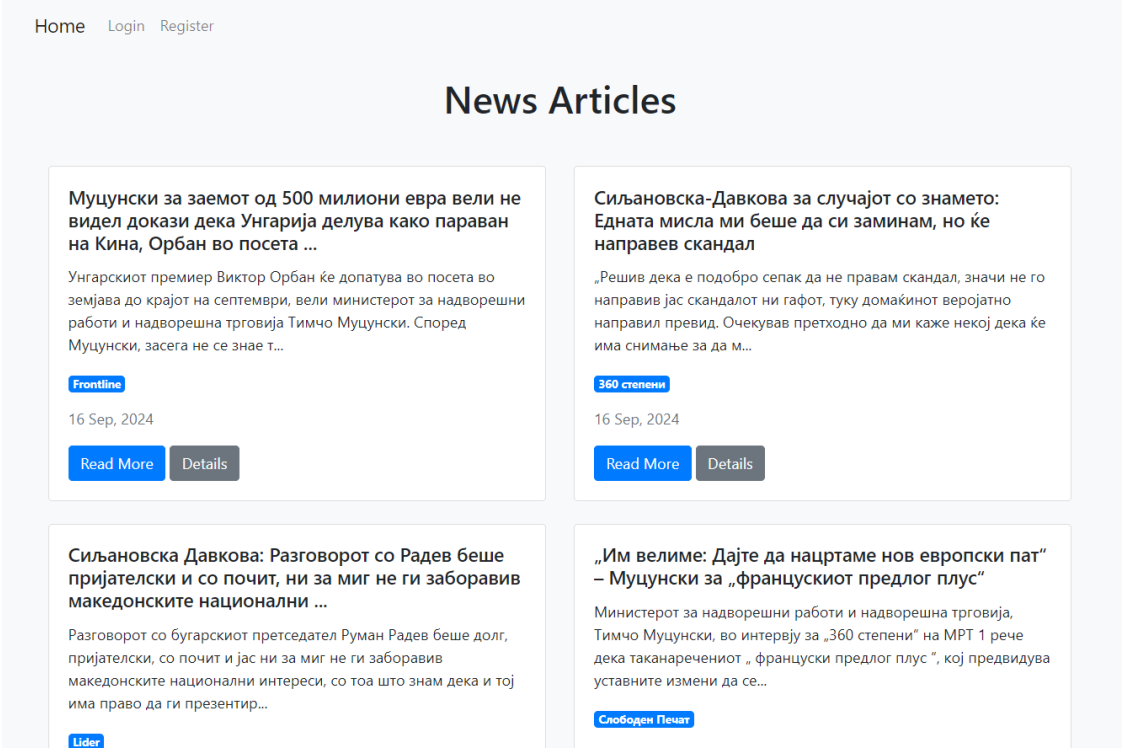
```
            // Save the new comparison to the database
            $comparison = Comparison::create([
                'article1_id' => $article->id,
                'article2_id' => $otherArticle->id,
                'similarity' => $similarity,
            ]);
        }

        $similarArticles[] = [
            'article' => $otherArticle,
            'similarity' => $comparison->similarity
        ];
    }

    usort($similarArticles, function ($a, $b) {
        return $b['similarity'] - $a['similarity'];
    });

    return array_slice($similarArticles, 0, ceil(count($similarArticles) * 0.10));
}
```

3.2 Views

Laravel's Blade templating engine is used to render the views. The system includes several important views for displaying articles, comparison results, and user actions.

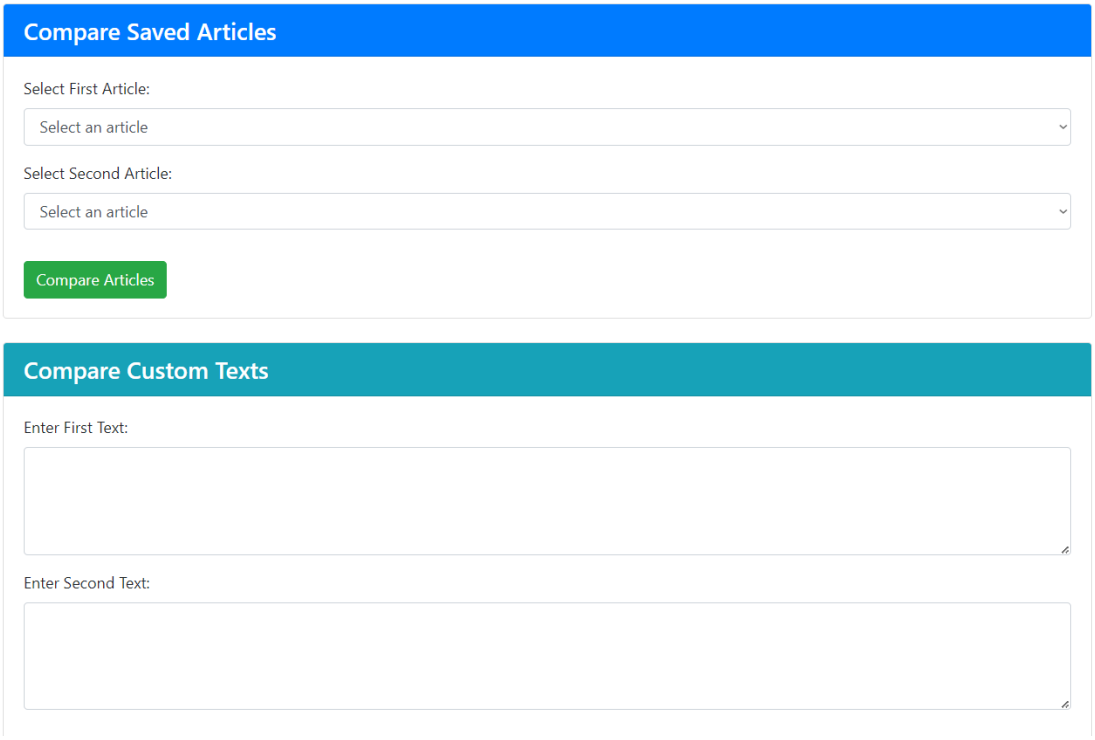
news.blade.php

This view displays the list of fetched news articles along with their comparison results. It provides an option to store the fetched articles in the database.



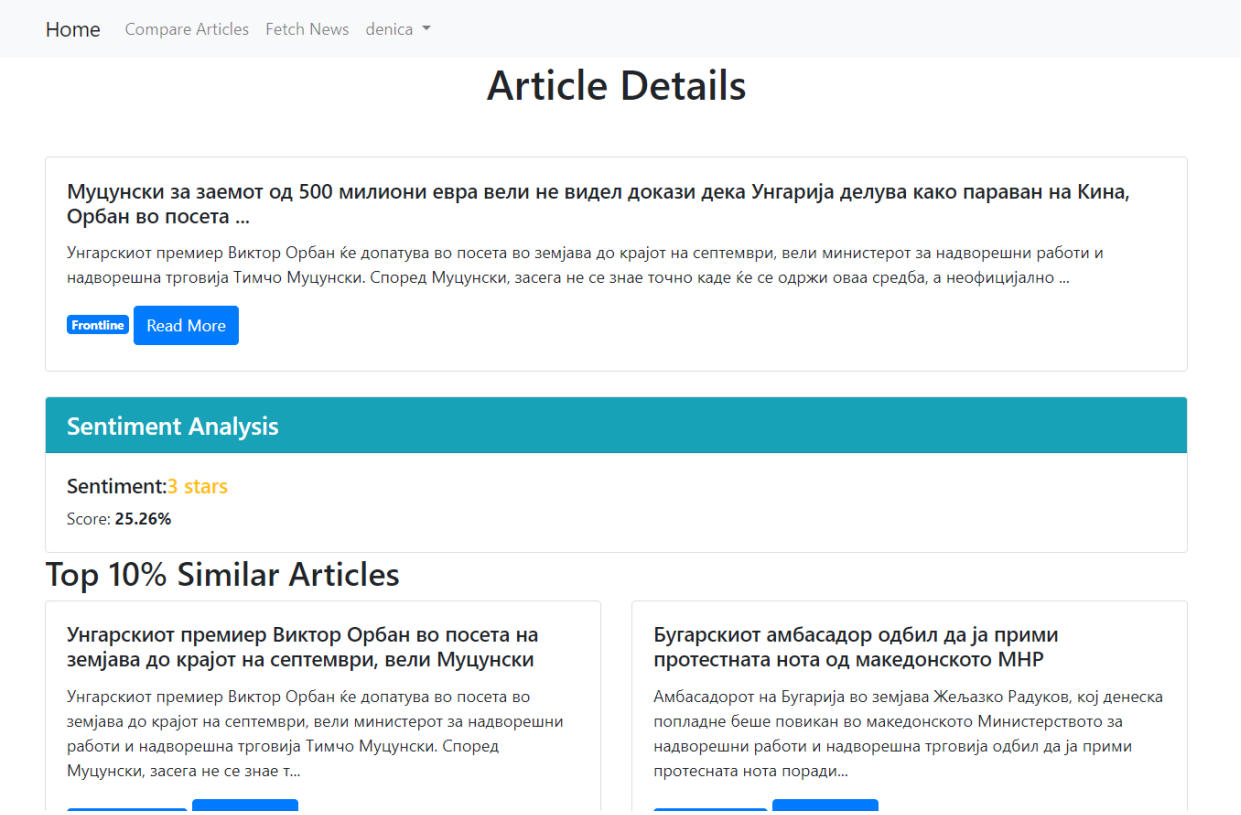
compare.blade.php

This view allows users to either select two saved articles to compare or enter custom texts for direct comparison.



details.blade.php

Displays the detailed view of an article, including the sentiment analysis and top 10% most similar articles.



4. System Components and Implementation

4.1 News Fetching Module

The news-fetching functionality is implemented using Python's BeautifulSoup library for web scraping. The system fetches the latest Macedonian news articles from the site time.mk. Once the articles are fetched, they are processed and stored in the database.

```
def fetch_news():
    url = "https://time.mk/?new=true&topic=makedonija"
    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'lxml')
        articles = soup.find_all('div', class_='cluster')
        news = []

        for article in articles:
            content = article.find(class_='snippet').text.strip()
```

```
            source = article.find(class_='source').text.strip() if
article.find(class_='source') else 'Unknown'
            title_h1 = article.find('h1')
            title = title_h1.find('a').text
            link = article.find('h1').find('a')['href']
            news.append({
                'title': title,
                'url': link,
                'content': content,
                'source': source
            })
        return news
    else:
        return []

if __name__ == "__main__":
    news = fetch_news()
    print(json.dumps(news, ensure_ascii=False, indent=2))
```

4.2 Article Comparison Module

The comparison module employs machine learning models, particularly BERT (Bidirectional Encoder Representations from Transformers), to calculate the similarity between two articles. The comparison is done both in the original Macedonian and in English, using Google Translate API for translation.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
model = BertModel.from_pretrained('bert-base-multilingual-cased')
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
sys.stderr = io.TextIOWrapper(sys.stderr.buffer, encoding='utf-8')
translator = Translator()
sys.stdout.flush()

def encode_sentence(sentence):
    inputs = tokenizer(sentence, return_tensors='pt', truncation=True, padding=True,
max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1)

def cosine_similarity(vec1, vec2):
    cos_sim = torch.nn.functional.cosine_similarity(vec1, vec2)
    return cos_sim.item() * 100  # Convert to percentage

def compare_news(news1, news2):
```

```
    vec1 = encode_sentence(news1)
    vec2 = encode_sentence(news2)
    return cosine_similarity(vec1, vec2)

def translate_to_english(text):
    translation = translator.translate(text, src='mk', dest='en')
    return translation.text

def compare_news_articles(news):
    results = []
    for i in range(len(news) - 1):
        for j in range(i + 1, len(news)):
            original_similarity = compare_news(news[i]['content'], news[j]['content'])
            translated_content1 = translate_to_english(news[i]['content'])
            translated_content2 = translate_to_english(news[j]['content'])
            translated_similarity = compare_news(translated_content1,
translated_content2)

            results.append({
                'news1_title': news[i]['title'],
                'news1_content': news[i]['content'],
                'news1_url': news[i]['url'],
                'news2_title': news[j]['title'],
                'news2_content': news[j]['content'],
                'news2_url': news[j]['url'],
                'similarity_original': f"{original_similarity:.2f}",
                'similarity_translated': f"{translated_similarity:.2f}"
            })
    return results

def load_input_data(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            input_data = json.load(file)
            return input_data
    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
        sys.exit(1)
    except json.JSONDecodeError:
        print("Error: Failed to decode JSON input.")
        sys.exit(1)

if __name__ == "__main__":
    file_path = sys.argv[1]
    news_articles = load_input_data(file_path)
    results = compare_news_articles(news_articles)
    print(json.dumps(results, ensure_ascii=False, indent=2))
```

4.3 Sentiment Analysis Module

Sentiment analysis of articles is achieved using a pretrained NLP model (nlptown/bert-base-multilingual-uncased-sentiment) in Python. This model predicts whether the sentiment of the article is positive, neutral, or negative.

```
model_name = "nlptown/bert-base-multilingual-uncased-sentiment"
sentiment_analyzer = pipeline("sentiment-analysis", model=model_name)

def analyze_sentiment(text):
    result = sentiment_analyzer(text)
    return result[0]  # Assuming the result is a list of dictionaries

if __name__ == "__main__":
    # Get the article text from the command-line arguments
    article_text = sys.argv[1]

    # Analyze sentiment
    sentiment = analyze_sentiment(article_text)

    # Print the result as JSON
    print(json.dumps(sentiment))
```

5. Database and Models

The Laravel database models manage the relationships between news articles, comparisons, users, bookmarks, and likes.

Key Models:

- NewsArticle: Represents news articles fetched from external sources.

- Comparison: Stores similarity scores between two articles.

- User: Manages user profiles and their interactions with articles (likes and bookmarks).

- Bookmark and Like: Store relationships between users and articles.

6. User Interactions

The system includes several user-facing features like article comparison, bookmarking, and sentiment analysis. The user can:

- Like Articles: Managed in the `likes` table.

- Bookmark Articles: Stored in the `bookmarks` table.

- Compare Articles: Both custom and saved articles can be compared.

- View Sentiment Analysis: Displayed along with each article's details.

7. Performance Optimization

Given the computationally intensive tasks of comparing text data and running sentiment analysis on large datasets, performance optimization is crucial for ensuring the system runs efficiently. Several measures were implemented to improve the overall performance of the system:

Caching Similarity Scores: The comparison results are cached after being computed for the first time. This avoids repeated and unnecessary execution of the Python comparison scripts on articles that have already been compared.

Asynchronous Processing: Time-intensive tasks like fetching news articles or performing sentiment analysis can take time, so the system uses background processing (or allows long-running HTTP requests) to prevent the frontend from freezing while waiting for results.

Efficient Text Truncation: Since the BERT model used for similarity comparison can only process up to 512 tokens, the system automatically truncates articles longer than this limit. This ensures that the processing remains within the model’s capacity while still providing meaningful comparisons.

8. Challenges and Solutions

1. Dynamic Web Scraping Some sites dynamically load content, requiring careful scraping using `BeautifulSoup` in Python.

2. Handling Large Text Comparisons: To prevent truncation issues, articles are limited to 512 tokens before comparison using BERT.

3. Performance Optimization: The system caches comparison results to avoid redundant computations.

4. User Experience: Simple forms and clear displays are used to improve user interactions and article management.

9. Conclusion

The project achieves a comprehensive system for comparing news articles and analyzing their sentiment. With Laravel managing the backend and Python for text processing, the system is capable of providing real-time news comparisons, sentiment analysis, and user-friendly interactions.