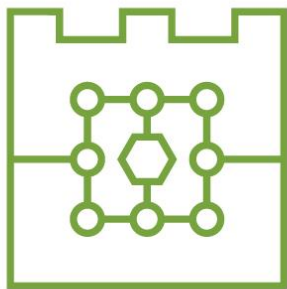


Przetwarzanie Języka Naturalnego

PROJEKT

Grammar and Spell Checker



Politechnika Krakowska
Wydział Informatyki
i Telekomunikacji

Wydział Informatyki
i Telekomunikacji
kierunek Informatyka
Rok IV, semestr VIII

Wysopal Wioletta
Zdeb Tomasz
Zimirski Piotr

Spis treści

1. Cel projektu	3
2. Technologie	3
3. Analiza	3
3.1. Struktura aplikacji	4
3.2. Analiza działania i użyteczności	4
4. Kod źródłowy	5
4.1. Uruchamianie aplikacji	8
4.2. Dalsze możliwości rozwoju	8

1. Cel projektu

Aplikacja *Grammar and Spell Checker* ma na celu poprawę leksykalno-gramatyczną tekstu wprowadzonego w języku angielskim. Jej głównym celem jest korygowanie błędów ortograficznych oraz gramatycznych. Aplikacja stanowi projekt edukacyjny realizowany w ramach przedmiotu *Przetwarzanie Języka Naturalnego*.

Aplikacja wykorzystuje słownik *enchant* oraz narzędzia *language_tool_python*, aby zidentyfikować i poprawić błędy ortograficzne w podanym tekście. Dzięki temu użytkownik otrzymuje nie tylko poprawiony tekst, ale także szczegółową informację o wykrytych błędach. Wykorzystanie *language_tool_python* umożliwia także analizę gramatyczną tekstu, identyfikując błędy takie jak niepoprawne użycie czasów, błędne konstrukcje zdaniowe czy problemy z interpunkcją.

Interfejs aplikacji umożliwia łatwe korzystanie zarówno z funkcji korekty tekstu wpisanego bezpośrednio w formularzu, jak i przesłanego w pliku. Możliwe jest przysyłanie plików w dowolnym formacie tekstowym.

2. Technologie

W niniejszym projekcie Python pełni kluczową rolę jako język *backendowy*, obsługujący logikę aplikacji. Zostały użyte biblioteki takie jak *Flask* do tworzenia i obsługi serwera webowego, *language_tool_python* do korekty gramatycznej oraz *enchant* do sprawdzania ortografii.

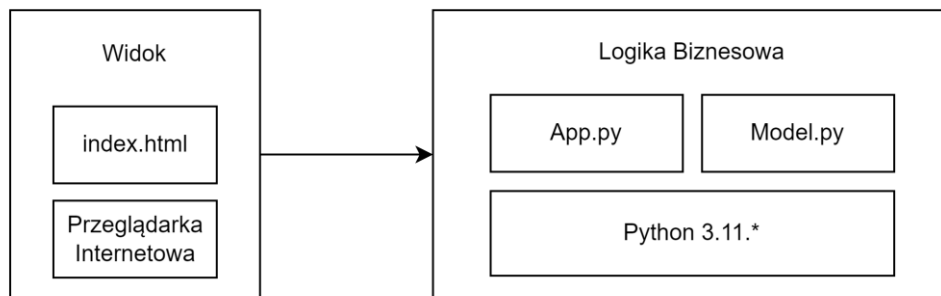
W projekcie zastosowano również język znaczników HTML do zbudowania podstawowej struktury interfejsu użytkownika, włączając w to formularze do wprowadzania tekstu i przysyłania plików. HTML definiuje również podstawowe elementy interakcji z użytkownikiem, takie jak przyciski i pola tekstowe. Do stylizacji i układu strony zastosowano framework Bootstrap, zapewniający gotowe komponenty CSS bez konieczności pisania stylowania strony od podstaw. Jako tło strony wykorzystano plik SVG *background-pattern*. Każdy element tekstu (litera lub cyfra) jest stylizowany i rozmieszczony pod różnymi kątami oraz w różnych kolorach.

3. Analiza

Aplikacja *Spell and Grammar Checker* jest zaprojektowana do sprawdzania i korygowania błędów ortograficznych oraz gramatycznych w tekście podanym przez użytkownika. Na poziomie backendu aplikacja korzysta z zaawansowanych narzędzi do przetwarzania języka naturalnego, tym samym stając się przykładem zastosowania technologii AI w praktycznych narzędziach.

3.1.Struktura aplikacji

Aplikacja składa się z trzech głównych części: serwera `Flask`, modułu `Model.py` zaimplementowanego do korekty tekstu, oraz interfejsu użytkownika zdefiniowanego w pliku `index.html`.



Serwer `Flask` pozwala na interakcję z użytkownikiem, przetwarzając żądania HTTP i renderując odpowiedzi na podstawie danych wejściowych użytkownika. Serwer definiuje trzy główne trasy: jedna do wyświetlania strony głównej, druga do przetwarzania tekstu podanego bezpośrednio przez użytkownika, a trzecia do przetwarzania tekstu z plików. Każda z tych tras odbiera dane oraz przetwarza je za pomocą modułu `Model.py`.

Moduł `Model.py` zawiera klasę `SpellCheckerModule`, która używa `language_tool_python` do korekty gramatycznej oraz biblioteki `enchant` do sprawdzania pisowni. Klasa `TextCorrector` wykorzystuje model `T5` z biblioteki `transformers` w celu optymalizacji korekty tekstu. Opracowanie korekt odbywa się poprzez analizę tekstu podzielonego na tokeny, gdzie każdy token jest analizowany, a następnie – gdy jest taka potrzeba – korygowany. W procesie korekty uwzględniane są zarówno reguły gramatyczne, jak i słownikowe sugestie.

Frontend zaimplementowano w HTML i CSS z wykorzystaniem frameworka `Bootstrap`. Formularze HTML służą do zbierania danych wejściowych od użytkownika, które są następnie przesyłane do serwera.

3.2.Analiza działania i użyteczności

Podstawową funkcją aplikacji jest identyfikacja i korekta błędów w tekście wprowadzonym przez użytkownika. Użytkownik ma możliwość wprowadzenia tekstu zarówno przez formularz tekstowy, jak i przez wysłanie pliku, co daje elastyczność w sposobie korzystania z aplikacji. Wyniki korekty są wyświetlane na tej samej stronie, co wprowadzony tekst ułatwiając tym samym porównanie oryginału z poprawionym tekstem oraz analizę wykrytych błędów.

4. Kod źródłowy

Plik *index.html*:

```
index.html X background-pattern.svg app.py Model.py
1 <!doctype html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1">
6 <title>Grammar and Spell Checker</title>
7 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QktZyPjEj15vSmaRU90FeRpk6fctnYnDr5pKlyf2bRjKh03Hjy6HwALEwZn" crossorigin="anonymous">
8 <link href="/static/css/styles.css" rel="stylesheet">
9 </head>
10
11 <body>
12 <div class="text-center">Grammar And Spell Checker</div>
13
14 <div class="container">
15 <form action="/text" method="post" enctype="multipart/form-data">
16 <div class="form-group">
17 <label for="text">Type your text here:</label>
18 <textarea class="form-control" id="text" name="text"></textarea>
19 </div>
20 <button type="submit" class="btn btn-primary">Correct</button>
21 </form>
22
23 <div class="corrected-section">
24 <div class="text-center">
25 <h5>Corrected text:</h5>
26 <p>{{ corrected_text }}</p>
27 <h5>Mistakes found:</h5>
28 <p>{{ found_mistakes }}</p>
29 </div>
30 <div class="text-center">
31 <p>No text provided or corrected.</p>
32 </div>
33 </div>
34
35
36 <div class="container">
37 <form action="/file" method="post" enctype="multipart/form-data">
38 <div class="form-group">
39 <label for="file">Upload a file:</label>
40 <input type="file" name="file" class="form-control">
41 </div>
42 <button type="submit" class="btn btn-primary">Correct</button>
43 </form>
44
45 <div class="corrected-section">
46 <div class="text-center">
47 <h5>Corrected file text:</h5>
48 <p>{{ corrected_file_text }}</p>
49 <h5>Mistakes found:</h5>
50 <p>{{ found_file_mistakes }}</p>
51 </div>
52 <div class="text-center">
53 <p>No file provided or corrected.</p>
54 </div>
55 </div>
56
57 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrVf3tW6L6cW2024/2" crossorigin="anonymous"></script>
58 </body>
59 </html>
```

Plik *background-pattern.svg*:

```
index.html background-pattern.svg X app.py Model.py
static > css > background-pattern.svg
1 <svg width="200px" height="200px" xmlns="http://www.w3.org/2000/svg">
2 <text x="20" y="30" font-family="Arial" font-size="16" fill="#3c64fe" transform="rotate(5 20 30)">A</text>
3 <text x="50" y="70" font-family="Arial" font-size="16" fill="#f3cae2" transform="rotate(-20 50 70)">K</text>
4 <text x="150" y="30" font-family="Arial" font-size="16" fill="#95e0e2" transform="rotate(15 150 30)">M</text>
5 <text x="30" y="90" font-family="Arial" font-size="16" fill="#95d7fa" transform="rotate(-5 30 90)">B</text>
6 <text x="100" y="50" font-family="Arial" font-size="16" fill="#958cfa" transform="rotate(10 100 50)">C</text>
7 <text x="160" y="110" font-family="Arial" font-size="16" fill="#888888" transform="rotate(-15 160 110)">D</text>
8 <text x="60" y="150" font-family="Arial" font-size="16" fill="#c0c0c0" transform="rotate(20 60 150)">E</text>
9 <text x="120" y="190" font-family="Arial" font-size="16" fill="#e17d40" transform="rotate(-10 120 190)">F</text>
10 <text x="10" y="160" font-family="Arial" font-size="16" fill="#f3f35e" transform="rotate(0 10 160)">1</text>
11 <text x="90" y="220" font-family="Arial" font-size="16" fill="#4c6ec7" transform="rotate(-5 90 220)">2</text>
12 <text x="180" y="170" font-family="Arial" font-size="16" fill="#666666" transform="rotate(25 180 170)">3</text>
13 <text x="40" y="120" font-family="Arial" font-size="16" fill="#8864c7" transform="rotate(15 40 120)">4</text>
14 <text x="140" y="80" font-family="Arial" font-size="16" fill="#00b890" transform="rotate(-25 140 80)">5</text>
15 <text x="190" y="140" font-family="Arial" font-size="16" fill="#00e26d" transform="rotate(30 190 140)">6</text>
16 <text x="70" y="100" font-family="Arial" font-size="16" fill="#4c6ec7" transform="rotate(-15 70 100)">7</text>
17 <text x="130" y="160" font-family="Arial" font-size="16" fill="#7fc940" transform="rotate(20 130 160)">8</text>
18 <text x="180" y="60" font-family="Arial" font-size="16" fill="#80af5e" transform="rotate(-30 180 60)">9</text>
19 <text x="80" y="200" font-family="Arial" font-size="16" fill="#ecaf5e" transform="rotate(10 80 200)">10</text>
20 </svg>
21
```

Plik *Model.py*:

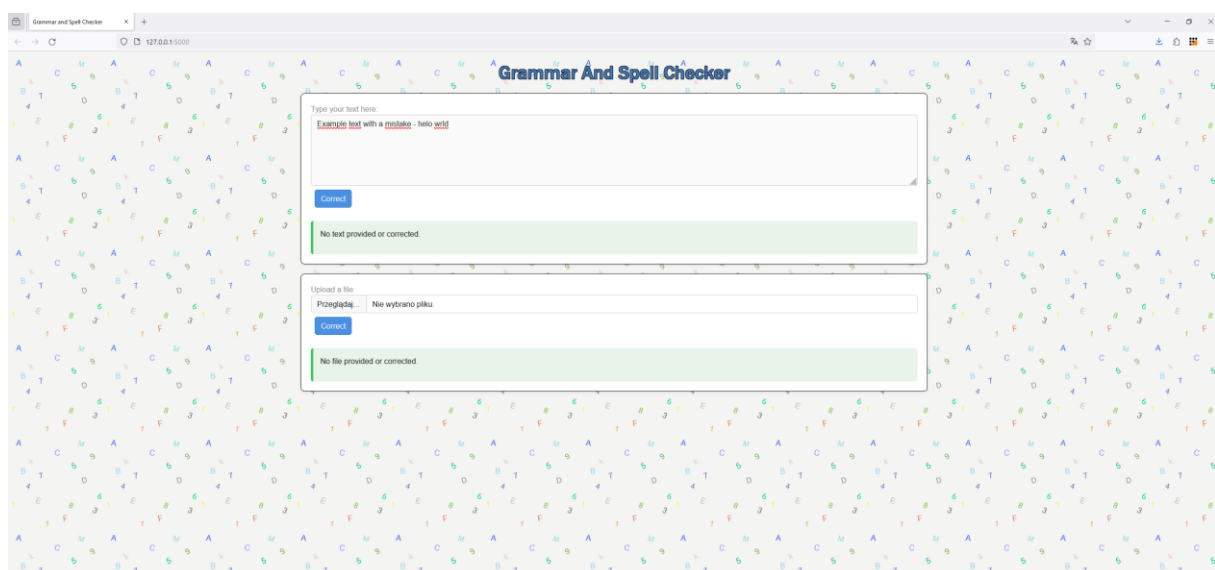
```
index.html * background-pattern.svg app.py Model.py X
1 import language_tool_python
2 import enchant
3 import re
4
5 class SpellCheckerModule:
6     def __init__(self):
7         self.tool = language_tool_python.LanguageTool('en-US')
8         self.d = enchant.Dict("en_US")
9
10    def correct_spell(self, text):
11        tokens = re.findall(r'\w+|[\^\w\s]', text, re.UNICODE)
12        corrected_words = []
13        mistakes = []
14
15        space_and_punctuation_pattern = r'(\s+|[\^\w\s]+)'
16        parts = re.split(space_and_punctuation_pattern, text)
17        new_parts = []
18
19        for part in parts:
20            if part and re.fullmatch(r'\w+', part) and not self.d.check(part):
21                suggestions = self.d.suggest(part)
22                if suggestions:
23                    new_parts.append(suggestions[0])
24                    mistakes.append((part, suggestions[0]))
25                else:
26                    new_parts.append(part)
27            else:
28                new_parts.append(part)
29
30        intermediate_text = ''.join(new_parts)
31
32        matches = self.tool.check(intermediate_text)
33        final_text = language_tool_python.utils.correct(intermediate_text, matches)
34
35        for match in matches:
36            original_text_segment = intermediate_text[match.offset:match.offset + match.errorLength]
37            corrected_segment = match.replacements[0] if match.replacements else original_text_segment
38            if original_text_segment != corrected_segment:
39                mistakes.append((original_text_segment, corrected_segment))
40
41        return final_text, mistakes
42
43    def correct_text_from_file(self, text):
44        return self.correct_spell(text)
45
46    def correct_grammar(self, text):
47        matches = self.tool.check(text)
48        found_mistakes = [text[match.offset:match.offset + match.errorLength] for match in matches]
49        corrected_text = language_tool_python.utils.correct(text, matches)
50        return corrected_text, found_mistakes if found_mistakes else "No grammar mistakes found."
```

Plik *app.py*:

```
index.html  background-pattern.svg  app.py  Model.py

1 from flask import Flask, request, render_template
2 from Model import SpellCheckerModule
3
4 app = Flask(__name__)
5 spell_checker_module = SpellCheckerModule()
6
7 @app.route('/')
8 def index():
9     return render_template('index.html')
10
11 @app.route('/text', methods=['POST'])
12 def text_process():
13     corrected_text = ""
14     found_mistakes = ""
15     if request.method == 'POST':
16         text = request.form['text']
17         if text:
18             corrected_text, initial_mistakes = spell_checker_module.correct_spell(text)
19             found_mistakes = ", ".join(f"{orig} - {corr}\n" for orig, corr in initial_mistakes) if initial_mistakes else "No mistakes found.\n"
20         else:
21             corrected_text = "No text provided."
22             found_mistakes = "No text provided."
23     return render_template('index.html', corrected_text=corrected_text, found_mistakes=found_mistakes)
24
25 @app.route('/file', methods=['POST'])
26 def file_process():
27     corrected_file_text = ""
28     found_file_mistakes = ""
29     if request.method == 'POST':
30         file = request.files['file']
31         if file:
32             readable_file = file.read().decode('utf-8', errors='ignore')
33             corrected_file_text, file_initial_mistakes = spell_checker_module.correct_spell(readable_file)
34             all_mistakes = [f"{orig} - {corr}\n" for orig, corr in file_initial_mistakes]
35
36             corrected_file_text, file_grammar_mistakes = spell_checker_module.correct_grammar(corrected_file_text)
37
38             if file_grammar_mistakes and not isinstance(file_grammar_mistakes, str):
39                 all_mistakes.extend(f"{match[0]} - {match[1]}\n" for match in file_grammar_mistakes)
40
41             found_file_mistakes = ", ".join(all_mistakes) if all_mistakes else "No mistakes found.\n"
42         else:
43             corrected_file_text = "No file provided."
44             found_file_mistakes = "No file provided."
45     return render_template('index.html', corrected_file_text=corrected_file_text, found_file_mistakes=found_file_mistakes)
46
47 if __name__ == "__main__":
48     app.run(debug=True)
49
```

Widok aplikacji w oknie przeglądarki internetowej:



4.1. Uruchamianie aplikacji

Procedura uruchamiania aplikacji została przedstawiona jako następujący zestaw kroków:

1. Zainstaluj język Python w wersji 3.11.9 (instalator na platformę MS Windows dołączony do projektu)
2. Zainstaluj środowisko uruchomieniowe języka Java w wersji 21.34 (instalator na platformę MS Windows dołączony do projektu)
3. Zlokalizuj ścieżkę w której zainstalowany został język Python i zapamiętaj ją.
Przykładowa ścieżka:
C:\Users\<Nazwa_Użytkownika>\AppData\Local\Programs\Python\Python311\python.exe
4. Zlokalizuj ścieżkę instalacji menedżera zależności: *pip*, zazwyczaj znajduje się on w tym samym katalogu co instalacja języka Python w podkatalogu: *Scripts*. Przykładowa ścieżka:
C:\Users\<Nazwa_Użytkownika>\AppData\Local\Programs\Python\Python311\Scripts\pip3.11.exe
5. Będąc w katalogu głównym aplikacji stwórz wirtualne środowisko języka Python przy pomocy komendy **python -m venv** . Uwaga! Komendę python zastąp **pełną ścieżką** uzyskaną w jednym z poprzednich kroków
6. Przejdź do katalogu **Scripts**
7. Uruchom skrypt **activate** (używając natywnego wiersza poleceń MS Windows należy uruchomić skrypt **activate.bat**)
8. Przejdź do katalogu głównego przy pomocy komendy **cd ..**
9. Zainstaluj zależności przy pomocy komendy **pip install -r requirements.txt** Uwaga! Komendę pip zastąp **pełną ścieżką** uzyskaną w jednym z poprzednich kroków
10. Uruchom aplikację przy pomocy komendy: **python app.py** Uwaga! Komendę python zastąp **pełną ścieżką** uzyskaną w jednym z poprzednich kroków
11. Uruchom przeglądarkę internetową
12. Przejdź do adresu: <http://127.0.0.1:5000/>

4.2. Dalsze możliwości rozwoju

Rozwój aplikacji *Spell and Grammar Checker* może przybrać różnorodne formy. Jednym z kluczowych ulepszeń mogłoby być wprowadzenie obsługi wielu języków, umożliwiając tym samym korzystanie z aplikacji użytkownikom niezależnie od ich narodowości. Można by także zaimplementować korektę wprowadzonego z klawiatury na bieżąco tak, aby użytkownik od razu otrzymywał poprawiony tekst. Ponadto można by także zaimplementować obsługę większej ilości formatów plików, na przykład PDF bądź Word. W

celu zwiększenia poprawności korekty możliwe jest wykorzystanie na przykład technologii uczenia maszynowego lub głębokiego uczenia, podnosząc tym samym jakość i dokładność korekt, zwłaszcza w bardziej złożonych kontekstach gramatycznych i stylistycznych.

Integracja aplikacji z innymi platformami i narzędziami, takimi jak edytory tekstu, klienty email oraz systemy zarządzania treścią, otworzyłaby nowe możliwości dla użytkowników – mogliby oni wówczas korzystać z funkcji korekty bezpośrednio w ramach używanych aplikacji.

W celu ochrony prywatności wprowadzanych danych można by zaimplementować rozbudowane metody szyfrowania. Zapewniłoby to użytkownikom większą kontrolę nad ich danymi, co jest obecnie niezwykle istotne, zwłaszcza w świetle rosnących wymagań dotyczących prywatności i ochrony danych.

Optymalizacja wydajności aplikacji, zarówno poprzez ulepszenie kodu, jak i infrastruktury serwerowej, zapewniłaby szybsze i bardziej efektywne przetwarzanie zapytań. Wpłynęłoby to na jakość obsługi większej ilości użytkowników.