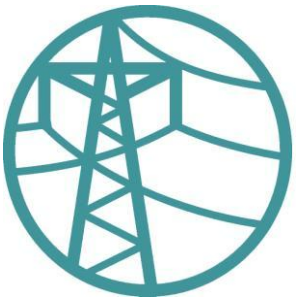


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Projekt z przedmiotu Programowanie w języku Python

Dawid Rokosz, Łukasz Pado, Patryk Słąba

Generator opisów Django

Rzeszów, 2023

1. Wstęp

Celem projektu było stworzenie aplikacji webowej która wykorzystuje API OpenAI do uzupełniania opisów produktów z wprowadzonej listy. Aplikacje należało wykonać wykorzystując framework Django.

2. Technologie

Django: Jest to framework służący do tworzenia aplikacji webowych w języku Python. Zapewnia wiele gotowych rozwiązań do autentykacji użytkowników, zarządzania bazą danych.

OpenAI API: Jest to oficjalne API dostarczone przez OpenAI, które umożliwia korzystanie z zaawansowanych modeli języka generatywnego do generowania tekstu na podstawie wprowadzonych zapytań.

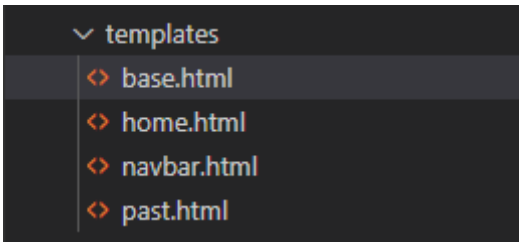
Python: Przy tworzeniu aplikacji wykorzystano głównie biblioteki python, oraz biblioteki z nim powiązane np. pip. Python pozwolił na automatyczną instalację frameworka Django, utworzenia projektu Django czy migracji modelu do bazy danych.

Bootstrap: To framework do tworzenia responsywnych interfejsów użytkownika. Zawiera gotowe komponenty, style i skrypty JavaScript, które umożliwiają szybkie tworzenie atrakcyjnych interfejsów użytkownika.

3. Wykonane prace

Templates

Pakiet Django odpowiada za utworzenie całej struktury projektu, w celu wyświetlenia się utworzonej strony w witrynie skonfigurowano ustawienia projektu. Szablon base.html przygotowuje stronę do wyświetlenia. W oparciu o Boostrap przygotowano wygląd strony składającej się z dwóch podstron: strona główna (szablon home.html) zawierająca element generujący opisy do przedmiotów, oraz strona z listą wcześniej wygenerowanych opisów (szablon past.html). Dodatkowo w szablonie głównym umieszczono szablon navbar.html który tworzy pasek nawigacyjny u góry strony który pozwala na nawigację pomiędzy podstronami.



Views.py

Główne funkcje które odpowiadają za wytworzenie i wyświetlanie opisów znajdują się w pliku views.py. Plik zawiera 3 funkcje główne:

home

```
def home(request):
    if request.method == "POST":
        question = request.POST['question']
        past_responses = request.POST['past_responses']

        # Check if question already exists in database
        if Past.objects.filter(question=question).exists():
            # Retrieve past response from database and display a warning message
            past_responses = Past.objects.get(question=question).answer
            messages.warning(request, f'Response for question "{question}" has
already been given.')
        else:
            # Set OpenAI API key
            openai.api_key =
"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"
            openai.Model.list()

            try:
                # Make API call to OpenAI to generate a response
                response = openai.Completion.create(
                    model="text-davinci-003",
                    prompt=question,
                    temperature=0,
                    max_tokens=450,
                    top_p=1.0,
                    frequency_penalty=0.0,
                    presence_penalty=0.0
                )
```

```
                response = (response["choices"][0]["text"]).strip()
                if "1new1" in past_responses:
                    # Add new record indicator to past_responses
                    past_responses = f"<span style='color: green;'>[New Record
Added]</span><br/><br/>{response}"
                else:
                    past_responses =
f"{past_responses}<br/><br/>{response[:500]}"

                # Save the question and its response to the database
                record = Past(question=question, answer=response)
                record.save()

                return render(request, 'home.html', {"question": question,
"response": response, "past_responses": past_responses})
            except Exception as e:
                return render(request, 'home.html', {"question": question,
"response": e, "past_responses": past_responses})

        return render(request, 'home.html', {"question": question, "response":
past_responses, "past_responses": past_responses})

    return render(request, 'home.html', {})
```

Funkcja home jest funkcją widoku, która obsługuje widok strony głównej w aplikacji internetowej. Przyjmuje ona obiekt “request” jako parametr, który reprezentuje żądanie HTTP wysłane przez klienta do serwera. Parametr “request” zawiera informacje takie jak metoda żądaniem dane POST i inne metadane.

Na początku ma miejsce uzyskane z metody POST pytania i ostatnich odpowiedzi na pytania. Następnie ma miejsce weryfikacja czy taki produkt nie znalazł już opisu w bazie danych aby nie wykorzystywać nadmiernie zasobów OpenAI. W przypadku wystąpienia takiego pytania pobierana jest odpowiedź z bazy danych.

W przypadku braku takiego produktu w bazie, następuje ustawienie klucza API OpenAI do uwierzytelnienia z API. Kolejno po ustawieniu klucza ma miejsce pobranie słownika modelu API, a następnie w metodzie try przygotowane są parametry zapytania do API, wraz z jego treścią. Zapytanie jest procesowane oraz aktualizowane są poprzednie zapytania. Odpowiedź zapisywana jest do bazy danych. Następnie ma miejsce wyrenderowanie treści odpowiedzi w HTML.

W przypadku gdy nastąpi błąd podczas wywołania API, na stronie zostanie wyświetlony błąd. W przypadku gdy nie nastąpi metoda POST, zwracana jest pusta odpowiedź.

past

```
def past(request):
    # Create a Paginator object to paginate the Past objects with 5 objects per
page
    p = Paginator(Past.objects.all(), 5)

    # Get the current page number from the request GET parameters
    page = request.GET.get('page')

    # Get the Page object for the current page using the Paginator object
    pages = p.get_page(page)

    # Retrieve all Past objects from the database
    past = Past.objects.all()
    nums = "a" * pages.paginator.num_pages

    # Render the 'past.html' template with the retrieved Past objects, Page
object, and 'nums' string
    return render(request, 'past.html', {"past":past, "pages":pages,
"nums":nums})
```

Ta funkcja obsługuje żądanie wyświetlenia obiektów klasy past z podziałem na strony. Wykorzystuje klasę Paginator do wyświetlania 5 obiektów klasy. Pobiera aktualny numer strony z parametrów GET żądania, pobiera obiekt Page dla aktualnej strony, wszystkie obiekty klasy Past z bazy danych, a następnie renderuje szablon ‘past.html’ z pobranymi obiektami klasy Past, obiektem Page i listą ‘nums’.

delete_past

```
def delete_past(request, Past_id):
    past = Past.objects.get(pk=Past_id)
    past.delete()
    messages.success(request, "Delete ANS and QUE done!")
    return redirect('past')
```

Funkcja obsługuje działanie przycisku usunięcia odpowiedzi na liście Past.