ДОДАТОК А

Програмний код вебдодатка

Лістинг А.1 – Програмний код файлу main.py

```
from fastapi import FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from fastapi.responses import JSONResponse
import language_tool_python
import os
import nltk
nltk.download('punkt')
os.environ["TRANSFORMERS_NO_TF"] = "1"
from transformers import pipeline
import requests
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from lime.lime_text import LimeTextExplainer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import
TfidfVectorizer
import json


app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Продовження лістингу А.1

```
OPENROUTER_API_KEY = "" # ключ приховано в цілях безпеки

corrector = pipeline("text2text-generation",
model="vennify/t5-base-grammar-correction")

class TextRequest(BaseModel):
    text: str

class ExplainRequest(BaseModel):
    language: str

def correction_languagetool(text):
    tool = language_tool_python.LanguageTool('en-US')
    matches = tool.check(text)
    corrected_text =
language_tool_python.utils.correct(text, matches)
    return corrected_text

def correction_gpt(text):
    prompt = f"""
    You are an English writing assistant. The user will
provide you with a text that contains grammatical,
spelling, punctuation, and lexical mistakes.

    Your task:
    1. Correct only lexical and semantic mistakes in the
text, if there are any. Make sure all the sentences make
sence separately and as the whole text. Pick up the style
of the text - formal, informal or something in between.
Make corrections according to that style - for example, if
the text is clearly informal, do not turn the informal
frases into formal ones.
    2. Output only the corrected version at the end. Do
not include the word \"corrected\" at the beginning.
```

Продовження лістингу А.1

```
    Text: {text}
    """

    headers = {
        "Authorization": f"Bearer {OPENROUTER_API_KEY}",
        "Content-Type": "application/json",
        "HTTP-Referer": "https://yourdomain.com",
        "X-Title": "text-correction-app"
    }

    data = {
        "model": "openai/gpt-4o-2024-11-20",
        "messages": [
            {"role": "user", "content": prompt}
        ]
    }

    response = requests.post(
        "https://openrouter.ai/api/v1/chat/completions",
        headers=headers,
        json=data
    )

    try:
        res_json = response.json()
        if 'choices' not in res_json:
            return "⚠ Виникла помилка під час обробки
відповіді."
        corrected_text =
res_json['choices'][0]['message']['content']
        return corrected_text.strip()
    except Exception as e:
        print("⚠ GPT ERROR:", str(e))
```

Продовження лістингу А.1

```
        return "⚠ Виникла помилка під час обробки
відповіді."

def correct_text_by_sentences(text, max_length=128):
    sentences = nltk.sent_tokenize(text)
    corrected_sentences = []

    for sent in sentences:
        correct_grammar = corrector(
            sent,
            max_length=max_length,
            do_sample=False,
            num_beams=4,
            early_stopping=True
        )
        correct_spelling =
correction_languagetool(correct_grammar[0]['generated_text
'])
        correct_semantic =
correction_gpt(correct_spelling)
        corrected_sentences.append(correct_semantic)

    return " ".join(corrected_sentences)


@app.post("/correct")
def correct_and_explain(req: TextRequest):
    start_text = req.text
    correct_text = correct_text_by_sentences(start_text)

    texts = [start_text, correct_text]
    labels = [1, 0]

    vectorizer = TfidfVectorizer()
    classifier = LogisticRegression()
```

Продовження лістингу А.1

```
    X = vectorizer.fit_transform(texts)
    classifier.fit(X, labels)

    pipe = make_pipeline(vectorizer, classifier)

    explainer = LimeTextExplainer(class_names=["correct",
"error"])
    explanation = explainer.explain_instance(
        start_text,
        pipe.predict_proba,
        num_features=6
    )

    features = explanation.as_list()
    lime_explanation_json = {
        "text": start_text,
        "top_tokens": [
            {"token": word, "importance": score}
            for word, score in features
        ],
        "predicted_label":
int(pipe.predict([start_text])[0]),
        "class_names": explanation.class_names,
    }

    # Зберігаємо LIME JSON
    with open("lime_explanation.json", "w", encoding="utf-
8") as f:
        json.dump(lime_explanation_json, f, indent=4)

    # Зберігаємо виправлений текст
    with open("corrected_text.txt", "w", encoding="utf-8")
as f:
        f.write(correct_text)
```

Продовження лістингу А.1

```
    # Зберігаємо HTML з LIME
    with open("text_explanation.html", "w", encoding="utf-
8") as f:
        f.write(explanation.as_html())

    return JSONResponse({
        "corrected_text": correct_text,
        "lime_result": lime_explanation_json
    })


class ExplainRequest(BaseModel):
    language: str

def generate_explanation_prompt(json_data, corrected_text,
language="en"):

    text = json_data.get("text", "")
    predicted_label = json_data.get("predicted_label", "")
    class_names = json_data.get("class_names", [])
    features = json_data.get("top_tokens", [])

    label_name = class_names[predicted_label] if
class_names and predicted_label < len(class_names) else
str(predicted_label)
    important_features = [f for f in features if
abs(f.get("importance", 0)) > 0]

    explanation_lines = [
        "You are an expert in English linguistics and
writing.",
        "A machine learning model analyzed the following
sentence and identified potentially problematic or
incorrect parts.",
        "",
```

Продовження лістингу А.1

```
        f"Text input from the user is: \"{text}\"",
        f"The corrected version is:\n{corrected_text}",
        f"🧠 Model's prediction: {label_name.lower()}",
        "",
        "The XAI method used to explain the model's
decision is LIME. It considers the following tokens
important for the correction model's decision:"
    ]

    if not important_features:
        explanation_lines.append("None - the model didn’t
find any strongly influencing words.")
    else:
        for i, feature in enumerate(important_features,
1):
            token = feature.get("token", "")
            weight = feature.get("importance", 0)
            explanation_lines.append(f"{i}. '{token}'
(importance score: {weight:.4f})")

    explanation_lines.append("")
    explanation_lines.append(
        f"Please explain in English why these tokens might
indicate that the sentence is incorrect or correct."
        f"Compare the initial text and the corrected one,
use the differences to explain the corrections, as well as
your own knowledge of the language and common sense. Do
not make up other corrections, only rely on differences
between two texts and the LIME's explanation."
        f"Use the simple language, explain as an English
language teacher to a student of a medium-level knowledge.
Do not include technical details in the explanation."
```

Продовження лістингу А.1

```
        f"Explain every token LIME found important from
the list above, make it an unnumbered list. Include a
small introduction and conclusion to sound more natural."
        f"Focus on grammar, semantics, or other writing
issues the model may have found."
        f"Explain using rules of english language but keep
the explanations brief and simple."
        f'Format your explanation using HTML instead of
Markdown. Use <strong> for bold and <h4> or <p> for
structure. Do not use multiple <br> tags for spacing –
instead, use <p> or <div> with minimal bottom margin. Do
not use <br> between <h4> and <p>. Wrap the entire output
in <div style="font-family: Arial, sans-serif; line-
height: 1.4; margin: 0; padding: 0;">.\n'
        )

    return "\n".join(explanation_lines)


@app.post("/explain")
def explain(req: ExplainRequest):
    lang = req.language.lower()

    try:
        with open("lime_explanation.json", "r",
encoding="utf-8") as f:
            json_data = json.load(f)
    except FileNotFoundError:
        return JSONResponse(content={"explanation": "⚠
No LIME explanation found."})

    try:
        with open("corrected_text.txt", "r",
encoding="utf-8") as f:
            corrected_text = f.read()
```

Продовження лістингу А.1

```
    except FileNotFoundError:
        corrected_text = ""

    prompt = generate_explanation_prompt(json_data,
corrected_text, language=lang)

    headers = {
        "Authorization": f"Bearer {OPENROUTER_API_KEY}",
        "Content-Type": "application/json",
        "HTTP-Referer": "https://yourdomain.com",
        "X-Title": "lime-text-explainer"
    }

    data = {
        "model": "openai/gpt-4o-2024-11-20",
        "messages": [
            {"role": "user", "content": prompt}
        ]
    }

    try:
        response =
requests.post("https://openrouter.ai/api/v1/chat/completio
ns", headers=headers, json=data, timeout=30)
        gpt_reply =
response.json()["choices"][0]["message"]["content"]
    except Exception as e:
        gpt_reply = f"⚠ GPT error: {e}"

    try:
        with open("text_explanation.html", "a",
encoding="utf-8") as f:
            f.write("<hr><h3>🧠 Natural Language
Explanation:</h3>")
```

Продовження лістингу А.1

```
            html_reply = gpt_reply.replace('\n', '<br>')
            f.write(f"<div>{html_reply}</div>")
    except Exception as e:
        print(f"⚠ Could not write GPT explanation: {e}")

    try:
        with open("text_explanation.html", "r",
encoding="utf-8") as f:
            html = f.read()
    except FileNotFoundError:
        html = "⚠ No explanation file found."

    return JSONResponse(content={"explanation": html,
"prompt": prompt})
```

Лістинг А.2 – Програмний код файлу index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>TextTune</title>
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <div class="logo">
            <img src="TextTune.png" alt="TextTune logo"
style="height: 60px;">
        </div>
        <p>
            Enter the text you want to check for mistakes
and press the arrow.<br>
```

Продовження лістингу А.2

```
            To get the corrections explained, press
Explain button.<br>
            Please note: both correction and explanation
take some time, as ML models are working.
        </p>

        <div class="text-box">
            <textarea id="input-text" placeholder="Enter
your text"></textarea>
            <button class="arrow-btn"
onclick="correctText()">➡</button>
            <textarea id="output-text"
placeholder="Corrected text will appear here"
readonly></textarea>
        </div>

        <button class="btn"
onclick="explainCorrections()">Explain</button>

        <iframe id="explanation-frame" style="width: 100%;
height: 600px; border: none; display: none;"></iframe>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Лістинг А.3 – Програмний код файлу styles.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #eee;
    text-align: center;
    margin: 0;
    padding: 0;
```

Продовження лістингу А.3

```
    width: 100%;
    min-height: 100vh;
    box-sizing: border-box;
    overflow-y: auto;
}

.container {
    width: 80%;
    max-width: 1200px;
    min-height: 100vh;
    padding: 20px;
    background: #eee;
    margin: 0 auto;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: flex-start;
}

.logo {
    padding: 10px 20px;
    border-radius: 5px;
    margin-top: 10px;
}

.text-box {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 100%;
    margin-top: 20px;
}

textarea {
```

Продовження лістингу А.3

```
    width: 40%;
    height: 300px;
    padding: 10px;
    border: 1px solid #ccc;
    background: #f6f6f6;
    border-radius: 5px;
    resize: none;
    font-size: 16px;
}

.arrow-btn {
    padding: 15px 20px;
    font-size: 20px;
    background: lightblue;
    border: none;
    cursor: pointer;
    border-radius: 5px;
}

.btn {
    margin-top: 20px;
    padding: 12px 24px;
    background: #90ee90;
    border: none;
    cursor: pointer;
    font-size: 18px;
    border-radius: 5px;
}
```

Лістинг А.4 – Програмний код файлу script.js

```
async function correctText() {
    const inputText = document.getElementById("input-
text").value.trim();
```

Продовження лістингу А.4

```
    const outputArea = document.getElementById("output-
text");
    const iframe = document.getElementById("explanation-
frame");

    if (!inputText) {
        outputArea.value = "⚠ Please enter some text.";
        return;
    }

    outputArea.value = "⏳ Correcting...";
    iframe.style.display = "none"; // ховаємо старе
пояснення

    try {
        const response = await
fetch("http://127.0.0.1:8000/correct", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ text: inputText }),
        });

        const data = await response.json();
        outputArea.value = data.corrected_text || "✅
Done, but no correction returned.";
    } catch (error) {
        outputArea.value = "⚠ Error: " + error.message;
    }
}

async function explainCorrections() {
```

Продоження лістингу А.4

```
    const iframe = document.getElementById("explanation-
frame");
    iframe.style.display = "none";
    iframe.src = "";

    try {
        const response = await
fetch("http://127.0.0.1:8000/explain", {
            method: "POST",
            headers: { "Content-Type": "application/json"
},
            body: JSON.stringify({ language: "en" })  //
фіксована англійська
        });

        const data = await response.json();
        const blob = new Blob([data.explanation], { type:
'text/html' });
        const url = URL.createObjectURL(blob);
        iframe.src = url;
        iframe.style.display = "block";

    } catch (error) {
        alert("⚠ Error generating explanation: " +
error.message);
    }
}
```

ДОДАТОК Б

Представлення користувацького інтерфейсу

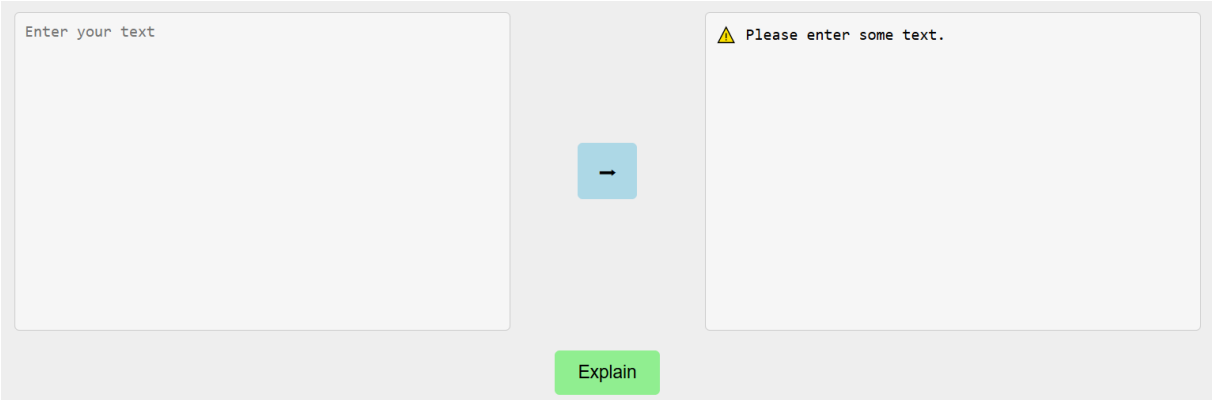

Рисунок Б.1 – Помилка при отриманні порожнього вхідного тексту

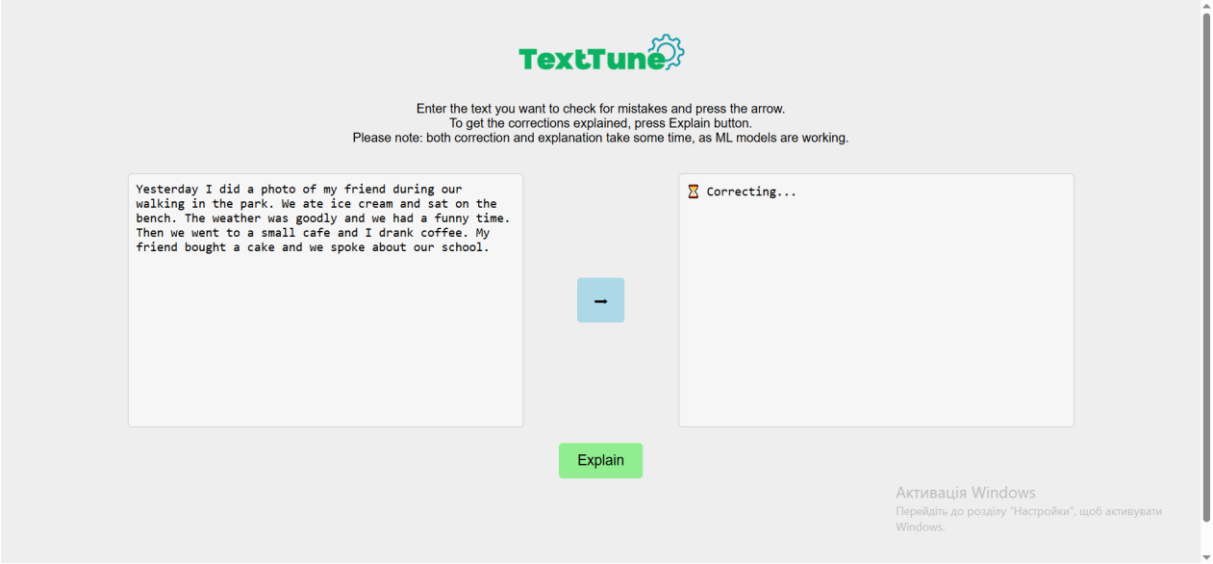

Рисунок Б.2 – Повідомлення про обробку тексту

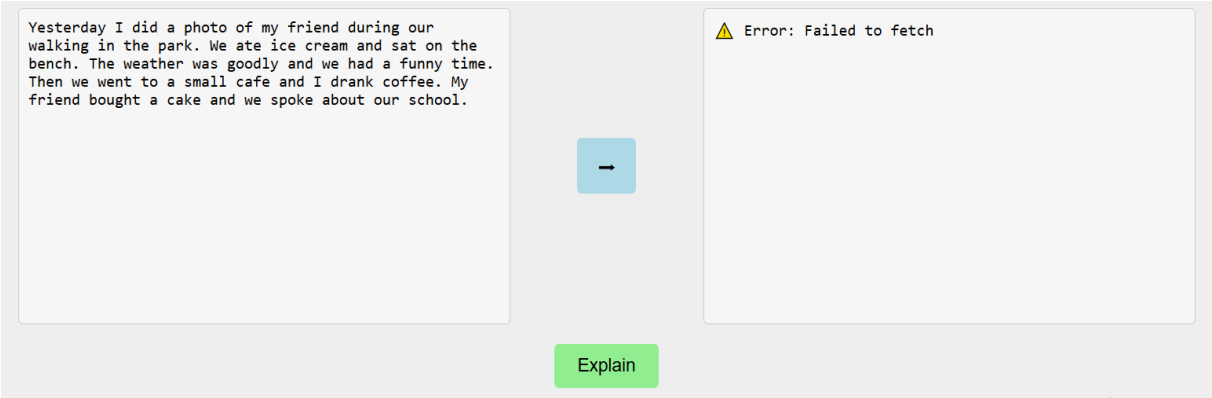

Рисунок Б.3 – Повідомлення про помилку на сервері під час виправлення

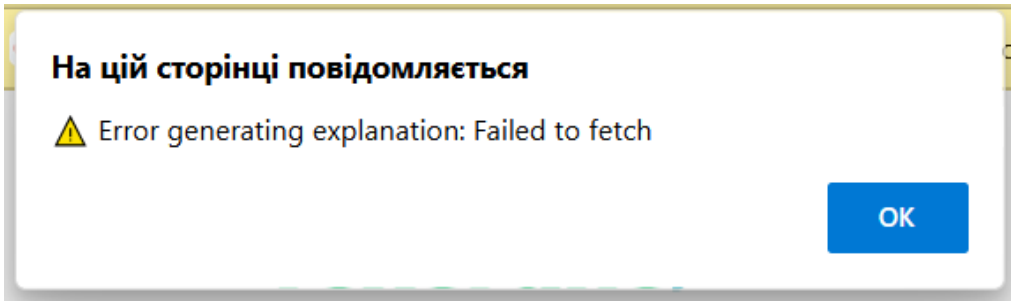

Рисунок Б.4 – Повідомлення про помилку на сервері під час пояснення

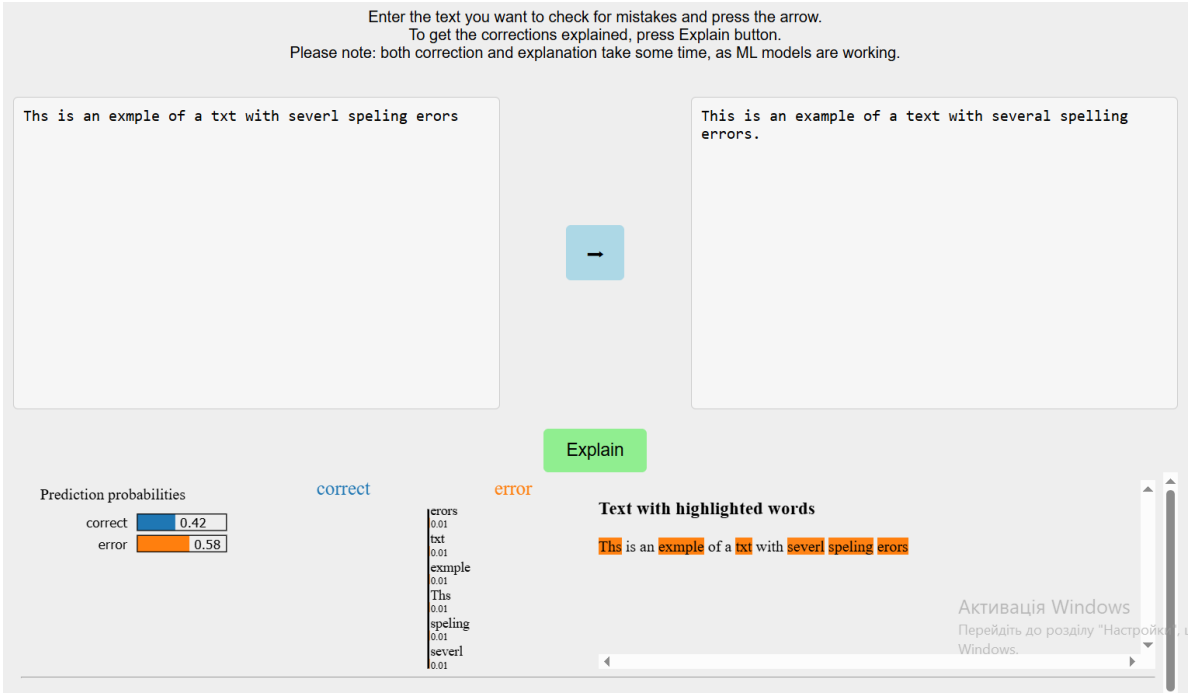

Рисунок Б.5 – Пояснення орфографічних помилок від LIME

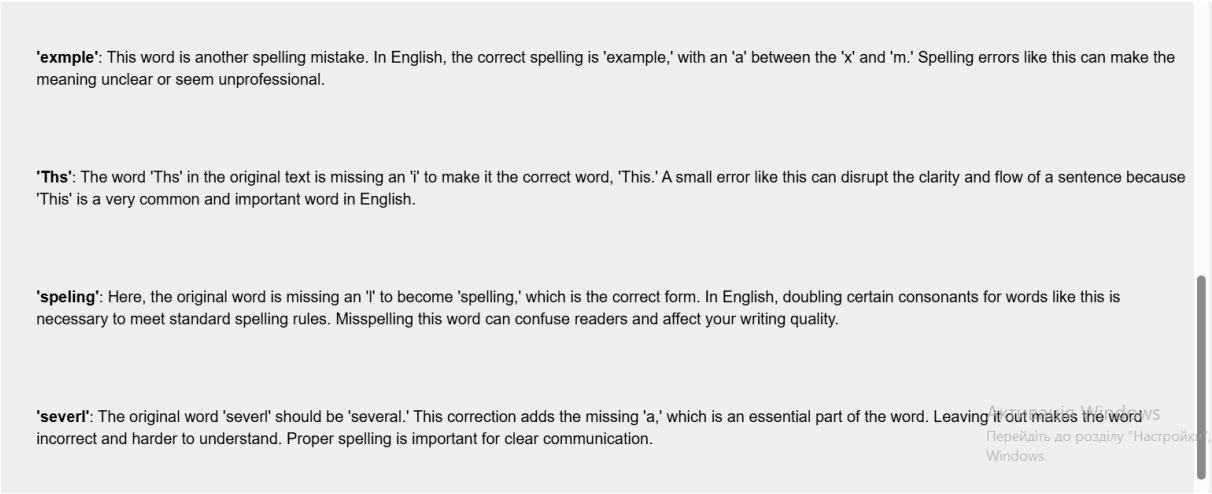

Рисунок Б.6 – Приклад пояснення орфографічних помилок природною мовою

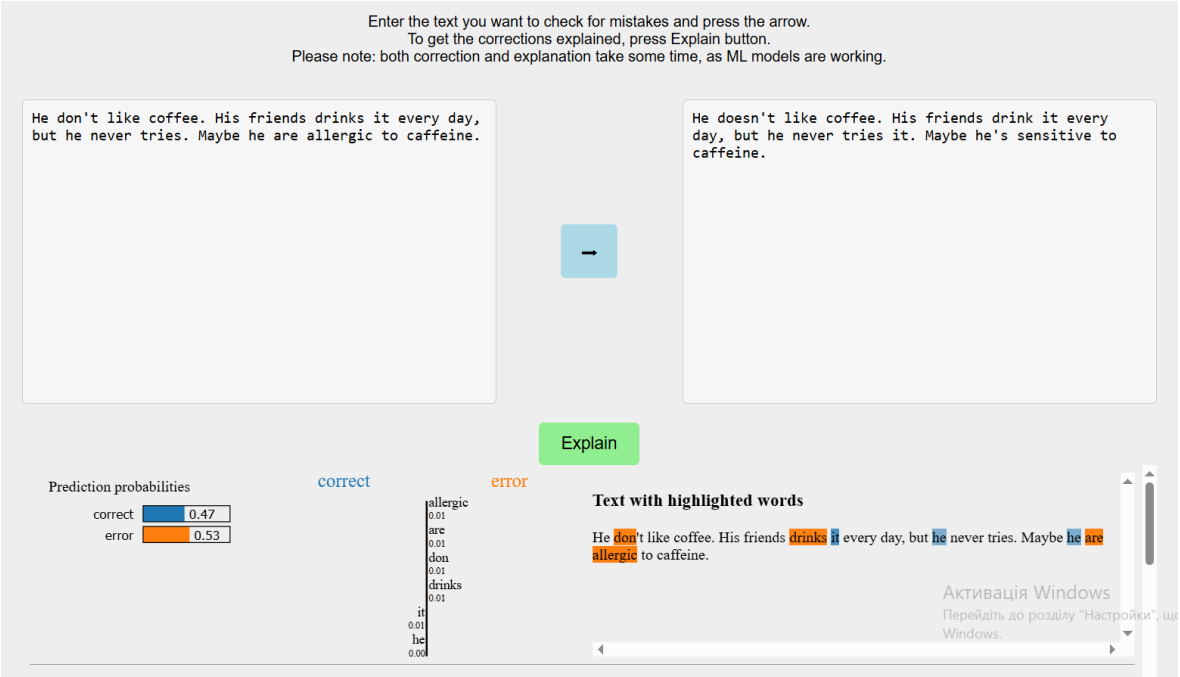

Рисунок Б.7 – Пояснення граматичних помилок від LIME

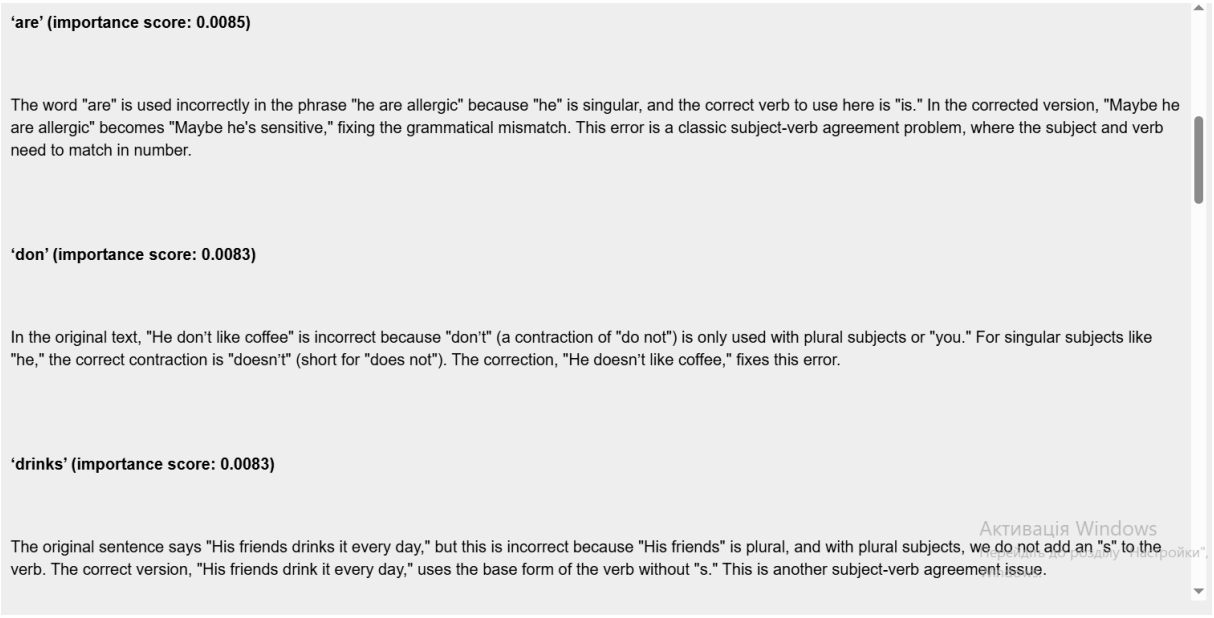

Рисунок Б.8 – Інтерпретація пояснення граматичних помилок природною мовою

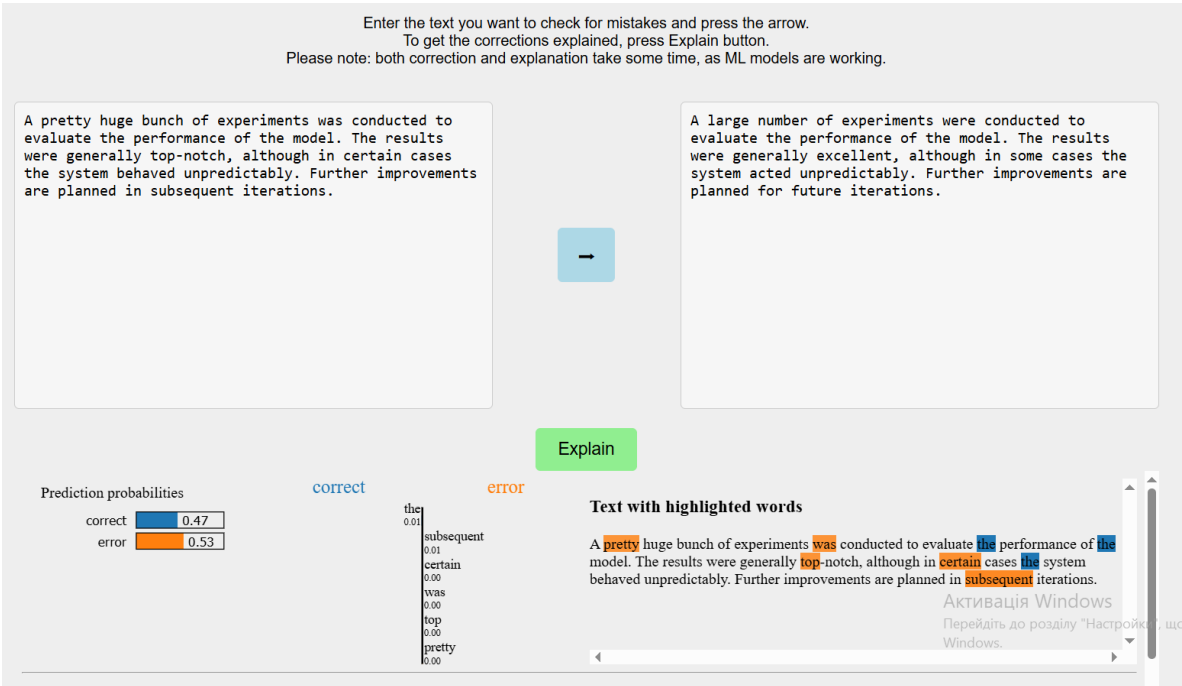

Рисунок Б.9 – Пояснення стилістичних помилок від LIME

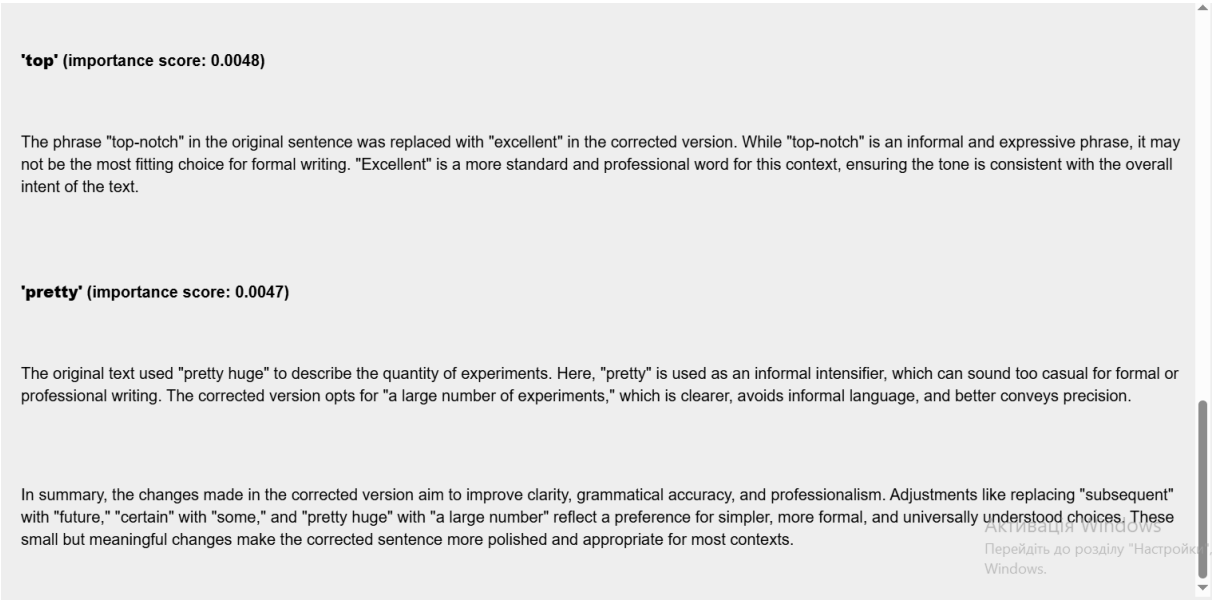

Рисунок Б.10 – Пояснення стилістичних помилок природною мовою

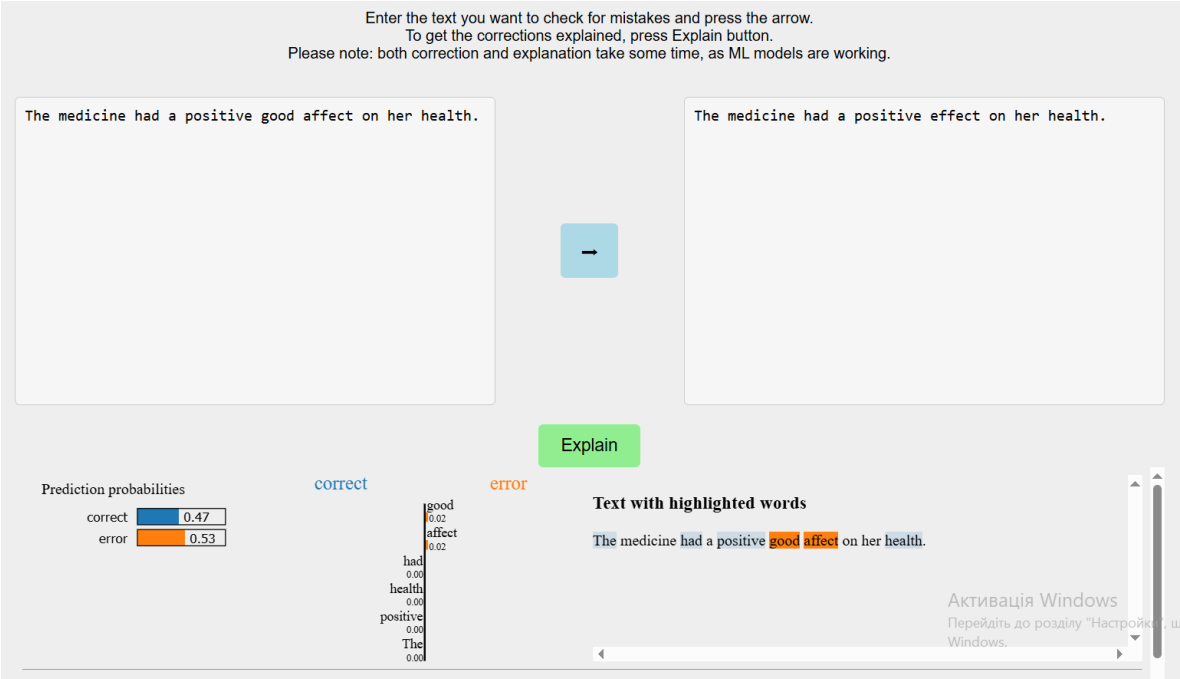

Рисунок Б.11 – Пояснення лексичних помилок від LIME

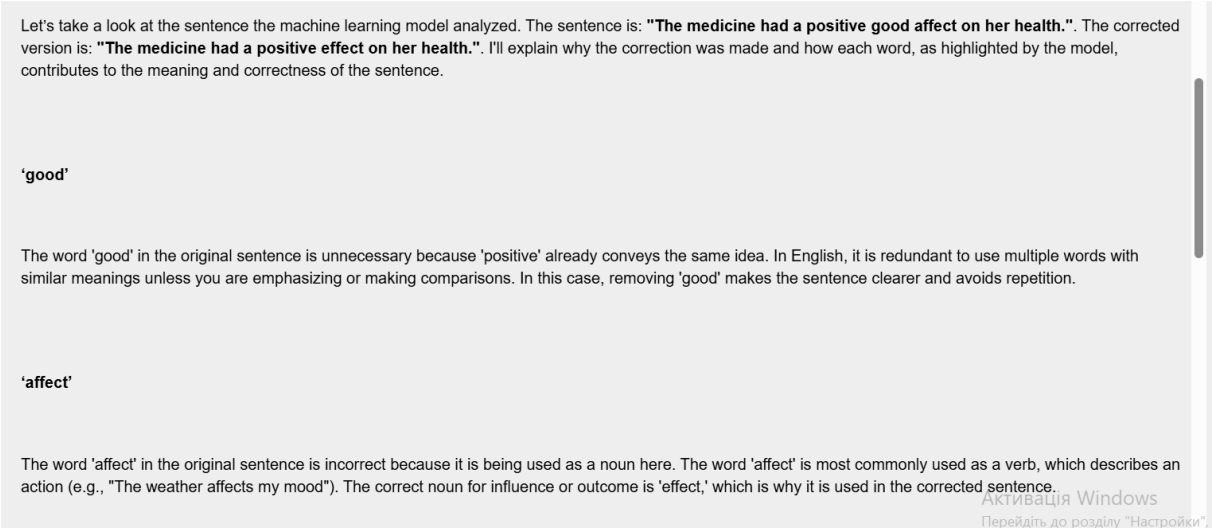

Рисунок Б.12 – Пояснення лексичних помилок природною мовою

Рисунок Б.13 – Пояснення семантичних помилок від LIME

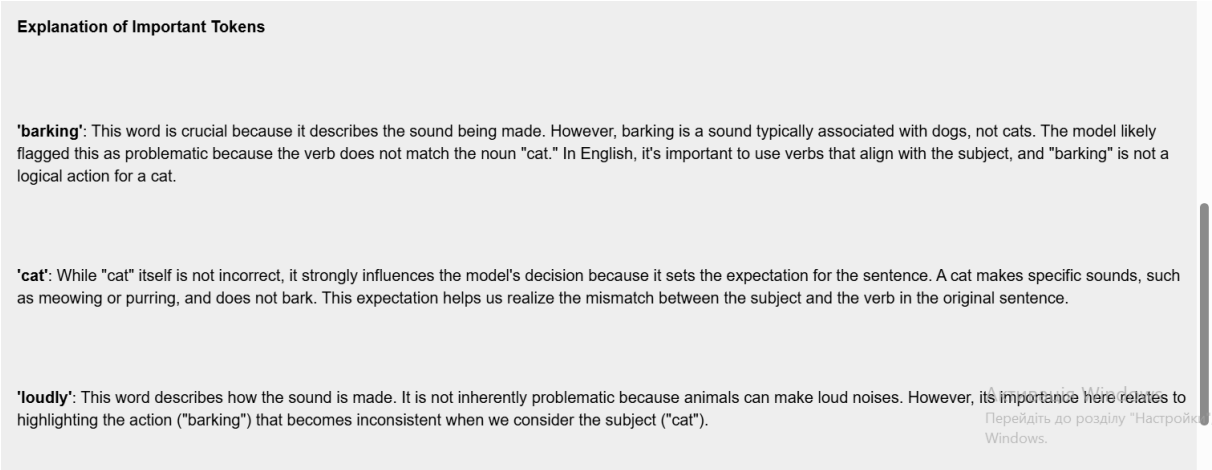

Рисунок Б.14 – Пояснення семантичних помилок природною мовою

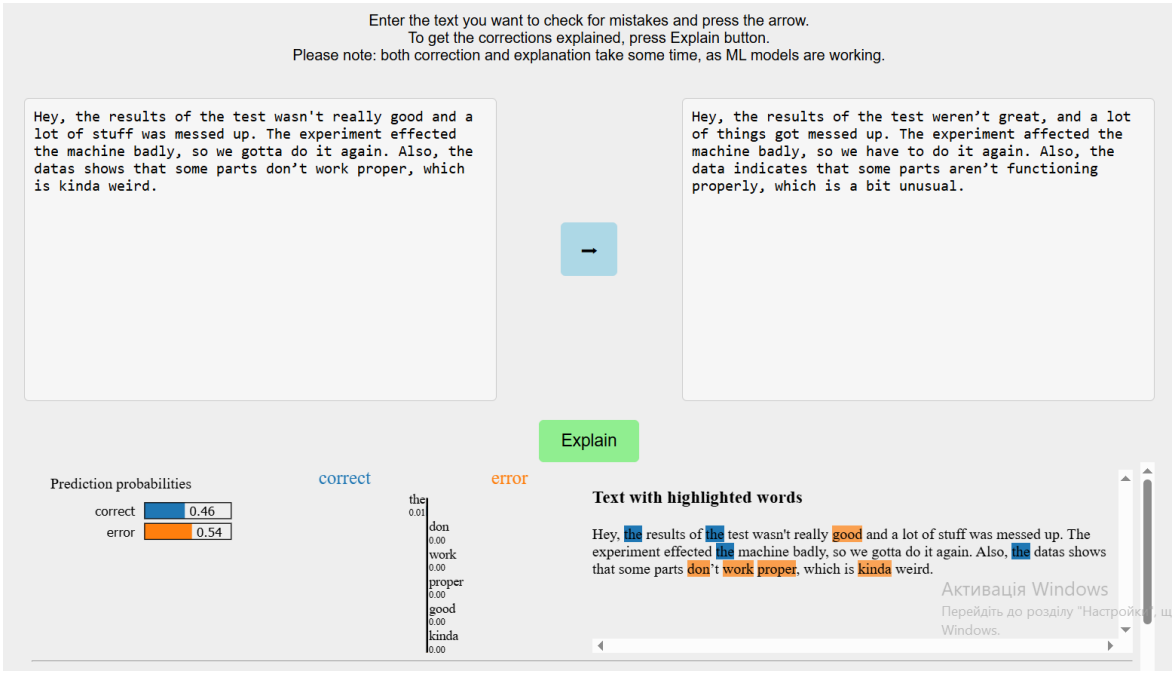

Рисунок Б.15 – Пояснення комбінованих помилок від LIME

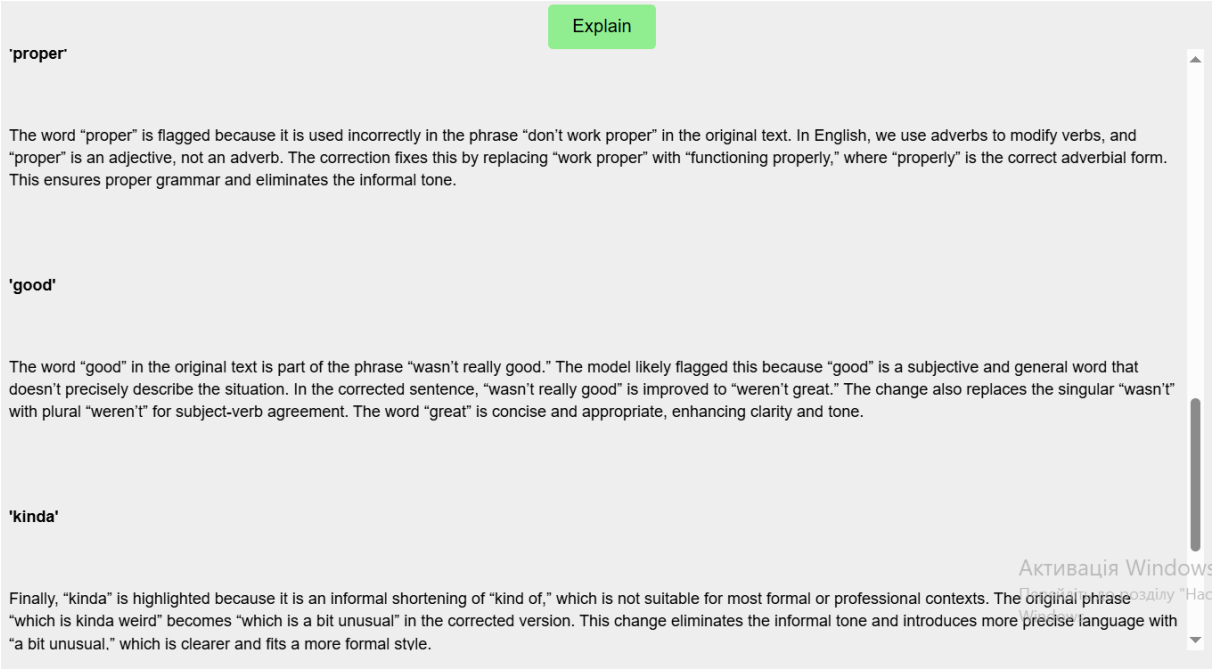

Рисунок Б.16 – Пояснення комбінованих помилок природною мовою

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	90 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Стрелкова А.О.			Інтеграція технологій Explainable AI та великих мовних моделей для виправлення та пояснення помилок в англомовних текстах	Шифр групи	Код напр./спец.
Перевірив		Головянко М.В.				ІТШІ-21-5	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					