МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**САВОЛЮК ВСЕВОЛОД ОЛЕКСАНДРОВИЧ**

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р техн. наук, професор
_____ Наталія ВЕСЕЛОВСЬКА

«____» _____ 20___р

# АНАЛІЗ РІВНЯ СТРЕСУ НАСЕЛЕННЯ НА ОСНОВІ КОНТЕНТУ СОЦІАЛЬНИХ МЕРЕЖ

Спеціальність 122 «Комп’ютерні науки»

**Кваліфікаційна (магістерська) робота**

Науковий керівник:

Тетяна СІЧКО, доцент кафедри
інформаційних технологій,
канд. техн. наук, доцент

_____(підпис)_____,

Оцінка: ____/____/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2025

## АНОТАЦІЯ

**Саволюк В.О. Аналіз рівня стресу населення на основі контенту соціальних мереж**. Спеціальність 122 «Комп'ютерні науки». Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

Магістерська робота присвячена створенню системи оцінювання рівня стресу на основі текстів соціальних мереж із використанням сучасних NLP-методів та трансформерних моделей. Реалізовано вебсистему з модулями збору, обробки й класифікації даних та інтерфейсом для відображення аналітики.

Робота містить вступ, три розділи та додатки. Перший розділ висвітлює теоретичні засади та підходи NLP, другий описує аналіз існуючих рішень і архітектуру системи, третій зображає практичну реалізацію вебдодатку. Робота нараховує 8 рисунків, 22 додатків і список літератури з 49 джерел. Загальний обсяг — 90 сторінок.

## ABSTRACT

**Savoliuk V.O. Analysis of the Population's Stress Level Based on Social Media Content.** Specialization 122 «Computer Science», educational program «Computer technologies for data processing», Vasyl' Stus Donetsk National University, Vinnytsia, 2025.

The master's thesis is dedicated to developing a system for assessing stress levels based on social media texts using modern NLP methods and transformer models. A web-based system was implemented with modules for data collection, processing, classification, and an interface for visualizing analytics.

The thesis includes an introduction, three chapters, and appendices. The first chapter covers theoretical foundations and NLP approaches, the second examines existing solutions and the system architecture, and the third presents the practical implementation of the web application. The total volume is 90 pages, with 49 sources in the reference list, 8 figures and 22 appendices.

# ЗМІСТ

# ВСТУП

У сучасному світі питання стресу набуває особливої ваги, оскільки темп повсякденного життя, постійні інформаційні потоки та нестабільність соціальних процесів безпосередньо впливають на психологічний стан людей. Стрес давно перестав бути лише медичним або суто психологічним поняттям. Він перетворився на міждисциплінарну категорію, що охоплює різні виміри: від фізіологічних реакцій до колективних соціальних тенденцій. Наукові підходи до його вивчення значно змінилися за останні десятиліття: якщо раніше увага зосереджувалася переважно на індивідуальних реакціях, то нині дедалі більше дослідників прагнуть зрозуміти, як стрес проявляється у поведінці великих груп та суспільств загалом.

Важливо, що сьогодні значна частина взаємодій людей переноситься у цифрове середовище [1]. Соціальні мережі стали не лише платформою для комунікації, а й невичерпним джерелом даних про емоційні стани, інформаційні хвилі та реакції спільнот на різні події, тому саме там користувачі залишають численні мовні та поведінкові сліди, які можуть відображати їхній внутрішній стан. У цьому контексті постає новий напрям досліджень — аналіз стресу через цифрові дані, що відкриває широкі можливості для вивчення динаміки соціальних процесів, виявлення регіональних відмінностей та раннього попередження соціальних ризиків.

Разом з тим аналіз стресу на основі публікацій чи коментарів у мережі має свої особливості. Стрес як явище не можна виміряти безпосередньо, тому дослідники звертаються до непрямих індикаторів: лінгвістичних ознак, частоти активності, змін у стилі письма, емоційного тону чи структури повідомлень. Поєднання психологічних теорій із лінгвістичним аналізом і методами обробки великих масивів даних дозволяє створювати моделі, здатні виявляти стресові тенденції у масштабах цілих міст або регіонів. Це особливо актуально в умовах швидкої соціальної трансформації, коли традиційні опитування та офіційна статистика не завжди встигають відобразити реальний стан суспільства.

З огляду на це виникає потреба у створенні системи, яка могла б інтегрувати різні типи цифрових даних, аналізувати мовні маркери та узагальнювати інформацію до рівня корисних для інтерпретації показників. Такий підхід важливий як для академічної спільноти, так і для практиків: аналітиків, психологів, фахівців у сфері управління ризиками. Він дозволяє побачити загальну картину, виявити приховані закономірності й оцінити рівень емоційного навантаження серед населення без втручання у приватність користувачів.

Саме тому робота присвячена побудові системи оцінювання стресу на основі даних соціальних мереж та аналізу її можливостей у контексті дослідження психологічних і соціальних процесів.

**Метою роботи** є розробка та обгрунтування підходу до оцінювання рівня стресу за даними соціальних мереж із використанням методів лінгвістичного та обчислювального аналізу.

Для досягнення поставленої мети потрібно виконати наступні завдання:

1. Проаналізувати теоретичні та методологічні підходи до визначення й вимірювання стресу.

2. Дослідити інструменти та методи аналізу даних соціальних мереж у контексті психологічних досліджень.

3. Описати цифрові індикатори стресу та їхню релевантність для автоматизованого аналізу.

4. Розробити загальну архітектуру додаток для збору, обробки та інтерпретації даних.

5. Реалізувати модулі для попереднього опрацювання текстових даних та лінгвістичного аналізу.

6. Побудувати модель визначення рівня стресу на основі вибраних індикаторів.

7. Сформувати узагальнені висновки щодо ефективності та можливостей застосування розробленого додатку.

**Об’єктом дослідження** є стрес як психологічний та соціальний феномен у цифровому середовищі.

**Предметом дослідження** є лінгвістичні, поведінкові та цифрові індикатори стресу, що проявляються у контенті соціальних мереж.

**Наукова новизна** полягає в поєднанні лінгвістичних маркерів стресу з контекстними ознаками, отриманими трансформерною моделлю, що дозволяє вперше сформувати узагальнений індекс стресу на рівні міста на основі неструктурованих повідомлень соцмереж. У роботі також удосконалено підхід до геолокації текстів: замість прямої прив’язки використовується комбінована система вагових сигналів (лексичні згадки, тип джерела, локальні контекстні маркери), що підвищує точність регіональної агрегації. Додатково розширено підхід до агрегування прогнозів моделі завдяки часовому згладжуванню та оцінці впевненості, що дозволяє отримувати стабільніші показники колективного емоційного фону.

# РОЗДІЛ 1
# ТЕОРЕТИЧНІ ОСНОВИ ТА МЕТОДОЛОГІЧНІ ПІДХОДИ ДО АНАЛІЗУ СТРЕСУ НА ОСНОВІ ДАНИХ СОЦІАЛЬНИХ МЕРЕЖ

## 1.1. Теоретико-методологічні основи визначення та вимірювання стресу

Проблема вимірювання стресу завжди була складною через те, що сам стрес існує на межі кількох сфер: фізіології, психології, соціальних наук і навіть лінгвістики. У реальності стрес може проявлятися у багатьох формах: у поведінці, у мовних патернах, у швидкості реакцій та у зміні звичних способів комунікації. Тому будь-яка система, яка намагається оцінити стрес за текстами соціальних мереж, повинна спиратися на широкий теоретичний фундамент. Тут неможливо обмежитися одним підходом. З одного боку є фізіологічні концепції, де стрес тлумачиться як відповідь тіла на небезпеку або надмірне навантаження, а з іншого — психологічні та когнітивні моделі, які пояснюють стрес через суб'єктивний досвід людини. І вже зовсім по-іншому виглядає підхід, пов'язаний із цифровими слідами, де стрес визначають через стиль письма, різкі зміни активності або характер емоційних реакцій у постах.

У сучасних дослідженнях підкреслюється, що стрес не можна трактувати як одну універсальну величину. Він має багато форм і причин, нерівномірно проявляється у різних людей і навіть залежить від контексту ситуації. Саме тому в аналітичних системах доводиться адаптувати класичні теорії під цифрові середовища, де дані з'являються спонтанно, часто у скороченому вигляді, з великою часткою емоційної експресії.

Для оцінювання стресу за текстами важливо враховувати не тільки зміст повідомлень а й те, як вони побудовані. Текст, особливо створений у стані напруги, може містити характерні індикатори: скорочення речень, повтори, емоційні маркери, нечітку структуру. Такі особливості стають своєрідним інструментом, що дозволяє переходити від абстрактного розуміння стресу до його формалізованої оцінки.

У межах цього розділу теоретико-методологічні засади розглядаються не ізольовано, а як частина ширшої системи. Вони створюють основу для технічних рішень, що з’являються далі. Це стосується і класифікацій стресу, і опису його маркерів, і способів переведення психологічних понять у цифрові показники, придатні для подальшої автоматизованої обробки. Такий підхід дозволяє рухатися від складної міждисциплінарної картини до моделі, яку можна реалізувати в обчислювальній системі.

**Сучасні концепції стресу, його класифікація та чинники виникнення.**

Уявлення про стрес за останні десятиліття помітно змінилися. Якщо раніше його розглядали переважно як фізіологічну реакцію організму на загрозу, то сучасні концепції суттєво ширші. Стрес тепер трактують як комплексний стан, що поєднує емоційні, когнітивні, поведінкові й навіть соціальні компоненти. У деяких дослідженнях його описують як «мультирівневий процес», у якому одночасно діють і внутрішні ресурси людини, і зовнішні подразники. Через це однозначно класифікувати стрес практично неможливо, адже він завжди виникає в певному контексті й проявляється індивідуально.

Однією з базових є теорія Ганса Сельє, яка традиційно вважається відправною точкою. Сельє описував стрес як універсальну фізіологічну реакцію організму на будь-який «стресор» [2]. Модель включала кілька фаз: тривогу, опір і виснаження. Попри те, що сьогодні ця концепція видається спрощеною, вона заклала основу для розуміння того, що напруження не з’являється миттєво, а формується поступово, впливаючи на поведінку й стан людини протягом певного часу.

Пізніше з’явився когнітивний підхід, що робить акцент на оцінці ситуації самою людиною. У цій моделі стресор не гарантує появу стресу; важливішим є те, як саме людина сприймає подію [3]. Якщо вона бачить її як небезпечну, некеровану або надмірно складну, стрес посилюється. У протилежному випадку його рівень може бути незначним навіть за об’єктивно важких умов. Цей підхід

суттєво наближає тему до аналізу текстів, бо саме в мовленні людини часто проявляється її інтерпретація події, а не сама подія.

Психологічні моделі також виділяють різні типи стресу. Їх умовно поділяють на «гострий», «хронічний» і «кумулятивний» [4]. Гострий стрес виникає раптово та зазвичай пов’язаний із конкретною подією. Хронічний стрес повільний і тривалий, зазвичай спричинений постійним напруженням або складними життєвими обставинами. Кумулятивний стрес — це накопичення дрібних подразників, які окремо малозначущі, але з часом створюють відчутне навантаження. У текстах соціальних мереж найчастіше проявляється саме кумулятивний варіант: користувач може писати про дрібні проблеми, але у великій кількості вони формують певний емоційний фон.

Ще один важливий аспект стосується джерел стресу. Їх умовно поділяють на зовнішні та внутрішні [5]. Зовнішніми можуть бути економічні труднощі, проблеми безпеки, конфлікти, зміни у середовищі. До внутрішніх належать сумніви, почуття безпорадності, різні страхи й невпевненість у майбутньому. У текстах ці джерела не завжди проявляються прямо, але вони впливають на форму висловлювання, вибір слів, частоту звернень до певних тем. Саме так цифрові сліди можуть непрямо відображати внутрішній стан людини.

Сучасні інтерпретації також розглядають стрес як феномен, що залежить від соціального оточення [6]. Людина по-різному реагує на події залежно від культурних норм, очікувань, підтримки від інших. У соціальних мережах це проявляється особливо помітно. Одні користувачі пишуть емоційно, інші приховують переживання або навмисно пом’якшують висловлювання. Усе це створює додаткові рівні складності, коли намагаєшся визначити стрес за текстами.

У підсумку сучасні моделі наголошують, що стрес є не лише відповіддю на подію, а результатом взаємодії людини зі своїми ресурсами та умовами довкола [7]. З цієї причини в контексті обчислювального аналізу доводиться формулювати його так, щоб він був придатний до вимірювання. Але для цього

спершу потрібно розуміти, як саме стрес виникає, як він класифікується й які чинники формують його інтенсивність. Саме ці знання дозволяють згодом перейти до цифрової реконструкції стресових проявів у мовленні та соціальній поведінці.

**Лінгвістичні маркери та психометричні індикатори стресу.**

У мовленні людини, особливо коли воно з'являється у швидкому, спонтанному середовищі на кшталт соціальних мереж, часто проступають дрібні ознаки емоційного напруження. Такі маркери не завжди очевидні на перший погляд, але в сукупності вони утворюють характерний «стиль стресу». У психології це розглядають як один із побічних ефектів когнітивного навантаження, коли людині стає важко: увага розсіюється, мислення спрощується або, навпаки, надмірно деталізується, а мовлення коливається між уривчастістю та підвищеною емоційністю [8].

Лінгвістичні маркери зазвичай поділяють на кілька типів. Перший — лексичні ознаки, тобто конкретні слова або словосполучення, які несуть емоційний відтінок. Це може бути лексика, пов'язана з тривогою, загрозою, втомою, невпевненістю. У стресових текстах часто зростає кількість негативних прикметників або дієслів [9], але це не обов'язкове правило. Людина може уникати прямих емоційних слів, але все ж демонструвати напруження через інші механізми.

Другий тип становлять структурні маркери. Це зміни в довжині речень, стрибки між темами, порушення звичної ритмічності письма. Наприклад, короткі фрази, які ніби обриваються, можуть сигналізувати про пришвидшений темп мислення або труднощі з концентрацією. Довгі, заплутані речення іноді свідчать про намагання «вивести» переживання через детальне описування. Такі патерни не є універсальними, але вони часто повторюються у текстах людей, що переживають тривогу чи перевтому.

Третю групу становлять прагматичні маркери, пов'язані зі способом взаємодії людини з аудиторією. У стані стресу користувач може щоразу

сильніше прагнути отримати підтвердження, підтримку або пояснення. Це виявляється у вигляді риторичних запитань, повторів, звертань, різких змін тону. Іноді активізуються фрази, які підкреслюють безпорадність: «не знаю, що робити», «вже немає сил», «не витримую».

У психометрії використовують іншу оптику. Там розглядають когнітивні індикатори: уповільнення або фрагментацію мислення, труднощі з формуванням послідовних повідомлень, помилки, пропуски слів [10]. Це не маркери у лінгвістичному сенсі, але вони проявляються у текстах і можуть бути виміряні опосередковано. Наприклад, різке збільшення орфографічних помилок у людини, яка зазвичай пише акуратно, може бути непрямим свідченням перевтоми.

Інколи враховують поведінкові показники, що супроводжують мовлення. Це зміни у частоті публікацій, порушення часу активності, хвилеподібність появи нових повідомлень. У стресових станах користувачі можуть різко активізуватися, або навпаки замовкати на тривалий час [11]. Такі коливання допомагають краще зрозуміти контекст, у якому з’явилися текстові індикатори.

Усі ці елементи разом утворюють набір сигналів, і жоден із них окремо не дає достатньої інформації. Але їхня комбінація створює досить впізнавану картину [12]. Завдяки таким маркерам можна наблизитися до розуміння того, як саме стрес проявляється у цифровому мовленні. Ці спостереження важливі для подальшої формалізації поняття «рівня стресу» та перетворення його на вимірювану величину, яку можна використовувати в обчислювальних системах.

**Формалізація поняття «рівень стресу» для потреб обчислювального аналізу.**

Коли йдеться про стрес у психологічному або медичному сенсі, це складний, багатовимірний стан. У ньому є емоційні, когнітивні, фізіологічні й поведінкові компоненти, і всі вони взаємодіють між собою. Але коли потрібно створити систему, яка працюватиме з цифровими текстами, доводиться перетворювати цей широкий феномен на щось вимірюване, тобто на певний

показник, який можна обчислити, передати через API й показати на графіку. Саме тому формалізація поняття «рівень стресу» стає окремим завданням, яке поєднує елементи психології, лінгвістики та обчислювальної логіки.

Передусім необхідно визначити, що саме системі слід вважати стресом у тексті. У традиційній психології стрес фіксується через опитувальники, фізіологічні вимірювання, спостереження. Жоден із цих підходів не працює для соціальних мереж. Там немає біометрії, немає структурованого опису стану, майже завжди немає навіть прямого свідчення про переживання. Тому доводиться спиратися на непрямі ознаки [13], зокрема на те, як побудований текст, які слова з'являються частіше, як змінюється стилістика або структура мовлення. На цій основі формується уявлення про стрес як про сукупність патернів, притаманних напруженим емоційним станам [14].

Щоб поняття стало вимірюваним, його необхідно пов'язати з числовою шкалою. У різних дослідженнях використовують різні підходи. Один із варіантів полягає у двокласовій схемі: «є стрес» / «немає стресу». Такий підхід простий, але він занадто грубий для реальних даних [15]. У соціальних мережах емоційні стани рідко бувають чорно-білими. Тому частіше застосовують шкалу з кількома рівнями: низький, помірний, високий. Можливий і регресійний підхід, у якому алгоритм видає число від 0 до 1, тобто умовний індекс стресу. Числовий варіант зручніший у подальшому аналізі: його можна агрегувати, порівнювати між містами чи часовими періодами [16].

Інша частина формалізації — виділення ознак. У технічному сенсі це означає, що текст треба представити так, щоб модель могла «розпізнати» в ньому стресові патерни. У класичних методах це робили через словники або статистичні ваги слів. Сучасні підходи дозволяють вбудовувати контекст, тобто не просто дивитися на слова, а враховувати їхні зв'язки, тональність, позицію, структуру речення. У моделі з'являється можливість розпізнати непрямі форми напруження, наприклад коли людина пише обережно, зміщує увагу на певні теми, уникає чітких формулювань або повторює кілька фраз у різних варіаціях.

Формалізація також включає питання агрегації. Стрес має індивідуальний характер, проте в межах цього дослідження ключовою є саме колективна картина того, що відбувається у певному місті чи спільноті. Один текст може бути емоційним випадком, але кілька десятків повідомлень за день уже формують локальну тенденцію. Тому рівень стресу у масштабі міста визначають не за одним документом, а як середню оцінку за певний період. Це дозволяє згладити випадкові коливання та побачити реальні зміни емоційного фону.

Ще один нюанс — стабільність. Оцінка стресу не може бути залежною лише від набору слів. Вона повинна враховувати, наскільки впевнена модель, чи є у тексті суперечливі сигнали, чи витягується патерн із контексту чи з поодиноких фраз. У деяких випадках навіть потрібне коригування результату на основі поведінкових ознак: наприклад, якщо текст виглядає нейтральним, але користувач раптом переходить до дуже частих публікацій.

У підсумку формалізований «рівень стресу» становить умовну величину, що поєднує мовні, стилістичні та статистичні сигнали. Він не претендує на повне відтворення психологічного стану людини, але відображає тенденції, які проступають у колективному цифровому мовленні. Подібна величина може бути автоматично обчислена, передана алгоритму і застосована в аналітичних системах, що робить її корисною для цілей цього дослідження.

### 1.2. Соціальні мережі як неструктуроване джерело даних для аналізу

Соціальні мережі за останні роки перетворилися на один із головних майданчиків, де люди спонтанно фіксують свої переживання, думки та реакції на події. Тут немає формальних рамок, на відміну від опитувань чи інтерв’ю, і саме ця неформальність робить такі дані придатними для дослідження емоційних станів. Тексти з’являються природно, у момент переживання, без попереднього редагування. Проте це також створює певні труднощі: інформація неструктурована, нерівномірна, часто фрагментована, а іноді й прихована у

формі сарказму, гумору чи метафор. Усе це робить аналіз складнішим, але водночас більш реалістичним.

Дані з соціальних мереж не мають чіткої форми, адже вони можуть бути короткими, довгими, емоційними або абсолютно нейтральними, приправленими емодзі, сленгом, англіцизмами чи навіть перемішаними кількома мовами. Тут зустрічаються повідомлення різного призначення: новини, скарги, особисті історії, реакції на події, емоційні вибухи. Для аналізу стресу ця різноманітність корисна, бо вона дає змогу побачити багатогранність людських реакцій.

Важливо й те, що соціальні мережі формують своєрідну карту подій. Коли в певному регіоні трапляється криза, інформація поширюється швидше, ніж у традиційних ЗМІ [17]. Люди публікують свої враження й тривоги буквально в ту ж хвилину. Це створює можливість оцінювати не тільки індивідуальні емоційні стани, а й колективні тенденції, характерні для певної місцевості [18]. Саме тому соціальні мережі є важливим джерелом для побудови карти рівня стресу в містах.

Разом із тим така гнучкість даних ускладнює технічну сторону аналізу. Нерегулярність структури та різноманітність форматів означає, що збір і підготовка інформації потребують окремих процедур. Тут не можна застосувати однаковий шаблон для всіх джерел. Кожна платформа має свої мовні особливості та специфіку: короткі пости у Twitter, емоційні реакції у Facebook, хаотичні повідомлення в Telegram-каналах.

Є ще один суттєвий аспект, а саме висока швидкість оновлення інформації. Потік даних у соціальних мережах постійно змінюється. Тут складно отримати «статичний» знімок стану, бо емоційний фон користувачів може перетворитися за кілька годин. Це робить соціальні мережі одночасно складним і дуже цінним середовищем для аналізу. У таких умовах будь-яка система повинна спиратися на методи, які можуть працювати з динамічними потоками текстів та враховувати нерівномірність їхнього розподілу.

Таким чином, соціальні мережі дають досліднику унікальну можливість зазирнути у процес формування колективного емоційного фону. Вони

показують, як люди реагують на події, які теми викликають найбільше напруження, де з'являються «осередки» тривоги. Але ці ж самі властивості роблять їх складним джерелом, що потребує спеціальних методів збору, очищення та аналізу.

**Характеристики Big Data у соціальних мережах та їх значення для дослідження емоційних станів населення.**

Коли говорять про дані соціальних мереж у контексті дослідження стресу, зазвичай мають на увазі великі масиви текстів, які з'являються щодня й постійно змінюються. Це класичний приклад Big Data, але не в узагальненому технічному сенсі, а в дуже конкретному: об'єм, швидкість, різноманітність і нестабільність цих даних суттєво впливають на спосіб аналізу. [19] Тут немає чітких структур, немає гарантій якості, а самі повідомлення часто короткі, емоційні, інколи випадкові. І саме така динаміка робить соціальні мережі корисним індикатором емоційного стану суспільства.

Першою характеристикою є масовість. Обсяг даних, що надходить із соцмереж, настільки великий, що окремі повідомлення втрачають значення, а важливою стає загальна тенденція. Людина може написати один емоційний пост, але якщо тисяча людей у тому самому регіоні пише подібним стилем, це вже сигнал. Масштаб дозволяє побачити закономірності, які були б непомітними в традиційних джерелах.

Другою є різноманітність. Дані подаються у формі текстів, коментарів, репостів, реакцій, повідомлень у публічних каналах. Вони відрізняються за стилем, довжиною, мовою, жанром. Тут змішуються офіційні новини, побутові історії, саркастичні твіти, фрагментовані записи. Для аналізу стресу це важливо, бо різні типи текстів дають різні кути зору на емоційний фон. Проте така строкатість ускладнює автоматичну обробку, і алгоритм має бути терпимим до шуму.

Третя характеристика — швидкість оновлення. Потік даних у соцмережах змінюється буквально щосекунди. Подія, яка відбулася годину тому, може вже

втратити актуальність, а емоційний фон користувачів зміниться, поки алгоритм ще обробляє попередню партію текстів. Тому моделі, які працюють із такими даними, мають враховувати час [20]: старі повідомлення швидко стають менш релевантними для оцінки стресу.

Четверта ознака — неструктурованість. На відміну від опитувань або анкет, де кожне питання передбачає конкретну відповідь, тексти соцмереж створюються без будь-якої схеми. Люди пишуть так, як їм зручно, інколи несистемно або надто емоційно. У таких умовах алгоритми повинні вміти працювати не лише з «правильними» текстами, а й із фрагментами, обірваними реченнями, повторюваними словами й іншими випадковостями.

П’ята характеристика — шумність. Велика кількість даних у соцмережах не несе цінної інформації для дослідження стресу: меми, жарти, спам, новинні заголовки, реклами. І все це доводиться відфільтровувати. Проте шум інколи має свої закономірності: коли користувачі масово іронізують над подією, це теж може бути непрямим проявом тривоги або захисного механізму.

Окрема важлива риса — локальність. Соціальні мережі дозволяють аналізувати дані в розрізі міст чи регіонів, що особливо корисно для вивчення колективних емоційних станів. Якщо певна тема раптом починає домінувати в конкретній місцевості, це може бути раннім сигналом зростання стресу [21]. Тут важливо не тільки те, що пишуть, а й де саме з’являються ці повідомлення.

Зрештою Big Data з соцмереж відкриває можливості, яких немає в інших джерелах. Вона дозволяє спостерігати за змінами емоційних патернів у реальному часі, бачити реакції людей «зсередини» їхнього повсякденного життя та формувати картину колективної напруги значно точніше, ніж за допомогою класичних методів. Але разом із цим виникають виклики, що вимагають спеціальних технік очищення та попередньої обробки, які будуть описані в наступних підпунках.

**Методи попереднього опрацювання та очищення даних соціальних мереж.**

Тексти соціальних мереж рідко бувають «чистими». Вони приходять у дуже строкатій формі: уривки речень, хаотичні коментарі, цитати, посилання, емодзі, випадкові вставки іншими мовами, інколи навіть зображення, перетворені на текст через OCR. Щоб аналіз стресу мав сенс, перед обробкою ці дані потрібно привести хоча б до мінімального порядку [22]. Йдеться не про ідеальне редагування, а радше про те, щоб зняти зайвий «шум», який заважав би алгоритмам фіксувати реальні емоційні сигнали.

Зазвичай очищення починають із найпростішого, тобто з видалення технічних елементів. Це HTML-теги, трекінгові параметри в посиланнях, службові символи, фрагменти кодів, випадкові системні вставки. Вони не несуть змісту, але можуть погіршувати якість токенізації. Далі виконується нормалізація тексту, що означає узгодження лапок, пробілів, апострофів та пунктуації в єдиний формат. У різних платформах ці символи можуть мати десятки варіантів, і така нормалізація дозволяє уникнути помилкового «роздроблення» тексту на дивні токени.

Окрема група завдань стосується боротьби з шумом, який виникає внаслідок специфіки соцмереж. Тут входять рекламні вставки, флуд, спам, коментарі на кшталт «підписуйтеся», повторювані фрази, які не мають емоційного змісту. Частину таких повідомлень можна відфільтрувати через ключові слова, інші виявляються тільки після попереднього аналізу структури. Наприклад, якщо пост містить лише посилання без тексту, він не буде корисним для оцінки стресу.

Особливої уваги потребують емодзі та несловесні маркери. У класичних текстових задачах їх зазвичай видаляють, однак у дослідженнях емоцій вони важливі [23]. Емодзі можуть бути більш інформативними, ніж слова, особливо коли користувачі скорочують текст або намагаються уникнути прямого опису переживань. Тому замість повного видалення емодзі їх частіше переводять у

спеціальні токени, тобто умовні позначення, які модель може використати в аналізі.

Ще один аспект — змішаність мов. Українські тексти в соцмережах часто перемішані з англійською, російською, сленгом, транскрипцією. Тому під час preprocessing доводиться виявляти мову кожного фрагмента окремо або хоча б фіксувати, які частини тексту не є українськими. Це важливо, бо моделі, натреновані на українській мові, можуть давати спотворені результати, коли зустрічають незнайомі конструкції.

Ще однією цікавою проблемою є повторюваність і шаблони. Деякі користувачі копіюють один і той самий текст, особливо під час масових подій. Такі дублікати можуть штучно підвищувати чи занижувати рівень стресу, тому їхнє виявлення є важливим етапом очищення. Виявити дублікати допомагають хеш-функції або методи порівняння з урахуванням відмінностей у розділових знаках.

Крім того, існує питання обробки надмірно коротких повідомлень. Одне слово чи емодзі не дозволяють точно визначити емоційний стан. Такі повідомлення можна або відфільтровувати, або зберігати окремо, але не враховувати як повноцінні одиниці для оцінки стресу. Тут потрібно балансувати між збереженням даних і якістю аналітики.

Попереднє опрацювання завершується токенізацією та базовою нормалізацією слів — це необхідна умова для роботи моделей NLP. Для української мови токенізація має свої нюанси: апострофи, м’які знаки, різновиди відмінювання. Важливо, щоб процес не руйнував лінгвістичну структуру тексту, бо від цього може залежати точність оцінки.

Таким чином, очищення та підготовка даних соціальних мереж є не технічною формальністю, а ключовим етапом, який визначає якість подальшого аналізу. У неструктурованому середовищі, де кожен текст має власні особливості, саме preprocessing дозволяє перетворити хаотичний потік повідомлень на матеріал, придатний для розпізнавання емоційних патернів.

**Географічна локалізація та валідація місцезнаходження користувачів у соціальних мережах.**

Одним із найскладніших аспектів використання даних соціальних мереж для оцінювання стресу є визначення, де саме знаходиться автор повідомлення. Соцмережі не створювалися для геоаналітики, і більшість платформ не надають точних координат [24]. Люди пишуть анонімно, пересуваються між містами, інколи зазначають локацію жартома або під впливом емоцій. Через це географічна прив'язка стає своєрідною комбінацією технічних прийомів, евристик і припущень, які потрібно постійно перевіряти.

Першим джерелом є самозазначені локації, які користувач вписує у профілі. Проблема в тому, що вони часто застарілі або вказані приблизно: «Україна», «Європа», іноді взагалі мемні формулювання [25]. Такі дані можуть бути корисними як дуже грубе орієнтування, але покладатися на них як на головний маркер небезпечно. Значно ціннішим стає локальний контекст у самих текстах.

Другим і більш надійним джерелом є текстові згадки міст або регіонів. Користувач нерідко пише про конкретні події: «сьогодні у Львові шумно», «черги в Харкові», «в Одесі дощ і знервовані люди». Такі згадки можна виявляти автоматично через списки топонімів або моделі розпізнавання іменованих сутностей (NER) [26]. Проте є й нюанс: згадка міста не завжди означає, що автор там перебуває. Людина може коментувати новини іншого регіону. Тому такі тексти потребують додаткової фільтрації або вагового коефіцієнта, який зменшує їхній вплив на локальний показник стресу.

Третім типом сигналів є метадані, які платформи надають частково. Це може бути час публікації, тип пристрою, мовні налаштування, а інколи приблизний регіон. Telegram-канали, наприклад, не дають координат, але багато хто з них має чітку регіональну спрямованість, і цей контекст можна використати як сурогатну геолокацію. Якщо канал створено для Дніпра, більшість публікацій можна вважати пов'язаними з цим містом.

Ще один інструмент — географічні словники та лінгвістичні ознаки. Деякі слова або вирази характерні для певних регіонів, особливо у просторі діалектів і локальних форм. Це не дає 100% точності, але інколи дозволяє підтвердити належність повідомлення до конкретної місцевості [27]. Наприклад, уживання певних локальних термінів, назв районів чи зупинок громадського транспорту може підказувати географічне походження тексту.

Окремою проблемою є валідація місцезнаходження. Елементарний збіг назви міста в тексті ще не є доказом реальної локалізації. Тому застосовують комбінований підхід, у якому враховується кілька сигналів одночасно: текстові згадки, джерело, тип каналу, час активності, попередні пости, загальний тематичний контекст. Якщо кілька ознак узгоджуються між собою, геоідентифікація вважається достатньо надійною [28], щоб включити текст у вибірку конкретного міста.

Для забезпечення більшої точності інколи застосовують вагову систему. Наприклад:

- пряме зазначення міста у тексті — висока вага
- згадка місцевої інфраструктури — середня вага
- приналежність каналу до регіону — висока вага
- новинні пости без емоційної складової — низька вага

Таке оцінювання дозволяє уникати ситуацій, коли кілька випадкових текстів спотворюють загальну картину стресу в місті.

Усе це показує, що геолокація у соціальних мережах є не окремим технічним модулем, а поєднанням різних шарів інформації, які разом дають умовно надійне уявлення про походження тексту. У дослідженні емоційних станів це особливо важливо, адже локальні події на кшталт черг, перебоїв електрики, загроз безпеці чи транспортних колапсів формують короткострокові хвилі стресу в конкретних містах. Щоб зафіксувати ці хвилі, системі потрібна хоч і приблизна, але стабільна прив’язка повідомлень до географії.

### 1.3. Огляд методів NLP у задачах емоційного аналізу

Обробка природної мови стала однією з ключових технологій у вивченні емоційних станів, оскільки дає змогу перетворювати звичайні текстові повідомлення на структури, з якими може працювати алгоритм. У випадку соціальних мереж це особливо важливо: тексти з'являються спонтанно, без єдиного стандарту, і будь-який аналітичний підхід має враховувати їхню нерівномірність. Задача емоційного аналізу, фактично, полягає в тому, щоб уловити емоційні сигнали, які люди залишають у словах, стилі, навіть у дрібних мовних «зламах», що виникають у моменти напруги [29].

Методи NLP розвивалися поступово. Спочатку аналіз базувався на лексичних підходах: рахувалися конкретні слова, пов'язані з позитивними або негативними емоціями [30]. Це було просто, але недостатньо гнучко. Далі з'явилася статистична обробка, де значення надавали не окремим словам, а їхнім частотам, співвідношенням, контекстам. Проте й ці методи мали свої обмеження, особливо коли йшлося про складні психологічні категорії.

Ситуація почала змінюватися, коли в аналіз увійшли моделі, здатні враховувати ширше оточення слова, тобто контекст [31]. Це дало можливість розпізнавати більш тонкі емоційні сигнали, які не завжди виражені прямими словами. Наприклад, уникання конкретних формулювань або синтаксичні «зсуви» у тексті теж можуть бути маркерами стресу, і сучасні моделі навчилися це виявляти.

У задачах емоційного аналізу особливо цінними стали підходи, що дозволяють створити узагальнене представлення тексту — його числову форму. Саме ця форма стає основою для подальшої класифікації: модель дивиться не на слова як такі, а на їхні взаємозв'язки, відтінки, позиції в реченні. За допомогою таких методів тексти соцмереж, які зовні здаються хаотичними, перетворюються на дані, придатні для розпізнавання емоцій.

Сучасний NLP уже не обмежується простими алгоритмами. Він включає як класичні словникові методи, так і глибинні нейронні моделі. І хоча всі вони

мають свої сильні й слабкі сторони, їхньою спільною метою є навчити машину «бачити» у текстах те, що зазвичай бачить людина: емоційний тон, напруження, коливання настрою. Це робить NLP ключовим інструментом у дослідженні цифрових проявів стресу, а розуміння цих методів — необхідним фундаментом для побудови власної моделі у наступних розділах.

**Традиційні NLP-підходи: лексико-базовані методи та аналіз тональності.**

Перші спроби автоматично визначати емоційний стан тексту спиралися на дуже просту логіку: якщо людина використовує певні слова, значить, у неї певний настрій. Це був початок лексико-базованих підходів [32]. Вони не враховували контекст, не цікавилися стилем чи синтаксисом, зате були зрозумілими: є словник, у словнику позначено, які слова мають позитивне або негативне забарвлення, і якщо в тексті таких слів багато, то висновок очевидний. Зрозумілий, але не завжди точний.

Лексичні методи довший час залишалися популярними, бо їх легко реалізувати. До того ж вони не вимагали великих обчислювальних ресурсів. Наприклад, у найпростіших системах кожне слово мало свою «вагу», і текст оцінювали як суму цих ваг. Якщо показник перевищував нуль, тональність вважали позитивною; якщо був нижчим, то негативною. Але життя значно складніше, ніж така схема. Люди рідко використовують емоційні слова прямо, особливо коли йдеться про стрес. Часто напруга проявляється не через лексичні маркери, а через загальну структуру мовлення: уривчастість, непослідовність, надмірну деталізацію або уникання певних формулювань.

Попри це, лексичні підходи дали багато важливих інсайтів для розвитку NLP. Вони вперше показали, що текст можна вимірювати і що емоційність можна подати у вигляді чисел. У деяких задачах словникові моделі досі корисні, наприклад коли потрібно швидко отримати базову оцінку тону повідомлення або опрацьовувати великі потоки коротких текстів із мінімальним контекстом.

Аналіз тональності став наступним кроком у розвитку традиційних методів. Він зберіг ідею словників, але додав більше структурності. Тут уже враховувалися модифікатори («дуже», «ледь»), заперечення, позиція слова в реченні [33]. Інколи застосовували навіть прості граматичні правила. Але проблема залишалася та сама: якщо людина пише «вже просто немає сил», то словниковий аналіз може вважати таке речення нейтральним, бо тут немає прямих негативних слів, хоча емоційний відтінок очевидний для людини.

Традиційні методи також майже не розпізнають сарказм, метафори та приховані емоції [34], тобто ті мовні елементи, які користувачі активно використовують у соцмережах. У випадку аналізу стресу це особливо критично, бо стрес рідко передається явно. Люди можуть писати жартами, ухилятися від прямого опису, або навіть використовувати позитивну лексику, але у контексті, який звучить тривожно.

Ще одна проблема полягає в українській мові. Більшість словників і тональних ресурсів створювали для англійської. Українські аналоги існують, але вони або неповні, або занадто загальні. Через це лексико-базовані підходи дають неточні результати, особливо коли текст змішаний із російською, англійською чи сленгом, тобто так, як зазвичай виглядають пости в соцмережах.

Попри всі недоліки, традиційні підходи важливі з методологічної точки зору: вони задають основу, від якої відштовхуються більш сучасні моделі. Вони показують, що емоційність можна формалізувати, але водночас наочно демонструють обмеження, які виникають, коли алгоритм дивиться лише на окремі слова. У контексті аналізу стресу це особливо помітно. Стрес потребує ширшого підходу до аналізу тексту, що враховує не лише окремі слова, а й контекст, динаміку та спосіб подачі. І саме тому в наступних підрозділах йдеться про методи, здатні краще «відчути» структуру мовлення.

**Векторизація тексту та ембедінги (Word2Vec, GloVe, FastText).**

Перехід від лексичних моделей до представлення тексту у вигляді числових векторів був одним із найважливіших кроків у розвитку NLP. Це

дозволило відійти від підрахунку слів як ізольованих одиниць і перейти до аналізу зв’язків між ними. Векторизація фактично дає змогу подати зміст тексту у такій формі, з якою можуть працювати машинні алгоритми. Важливо й те, що такий підхід фіксує не просто наявність слова, а також його семантичні нюанси.

Одним із перших методів, які змінили підхід до тексту, став Word2Vec [35]. Його основна ідея полягала в тому, що значення слова формується через оточення. Якщо два слова часто трапляються в подібних контекстах, їхні вектори будуть близькими. Це здається інтуїтивним: «тривога» й «напруження» не однакові, але їх легко пов’язати в емоційному полі. Word2Vec навчився робити такі зв’язки автоматично. Він не розумів мову у людському сенсі, але створював простір, де слова займали певні позиції й відносини між ними ставали вимірюваними.

Інша модель, GloVe, теж формує векторні представлення, але ґрунтується на глобальній статистиці тексту. Замість того щоб аналізувати лише найближчий контекст, GloVe враховує частоту спільних зустрічей слів у загальному корпусі [36]. Це робить вектори трохи стабільнішими у задачах, де важливі не дрібні локальні структури, а загальні зв’язки. Для аналізу емоцій це теж відіграє роль: людина може використовувати різні варіанти формулювань, але емоційне поле навколо них часто буде схожим.

Особливо цікавим став підхід FastText. Він навчився працювати не лише з цілими словами, а й із їхніми частинами, тобто субсловами [37]. Це дуже важливо для української мови, яка має складну морфологію. У Word2Vec або GloVe слово «стресом», «стресові», «стресу» вважаються різними токенами, що може зменшувати точність. FastText же створює вектори на основі фрагментів, тому зміна закінчення або форма слова не розбиває семантику. У результаті модель краще обробляє варіативність, властиву живому мовленню.

Векторизація стала основою для складніших моделей, що з’явилися пізніше й сформували сучасну NLP-парадигму. Та навіть самі ембедінги змінили те, як алгоритми «бачать» текст: слова перестали бути ізольованими мітками й

стали точками у багатовимірному просторі, де близькість відображає подібність, а віддаленість — різницю.

Проте ці методи мають свої обмеження. Вони працюють переважно на рівні окремих слів і не враховують повного контексту речення. Якщо користувач пише щось двозначне або приховує емоцію за іронією, ембедінг не завжди зможе це вловити. Наприклад, слова «класно» та «нарешті» у саркастичному пості можуть звучати зовсім не позитивно, але вектор буде позитивним.

Попри це, Word2Vec, GloVe і FastText дали можливість перейти від простого лічення слів до аналізу тексту як структури. Саме завдяки їм моделі почали розпізнавати тонші семантичні зв'язки й емоційні асоціації. Це створило основу для подальших підходів, де контекст, послідовність і структура мовлення стали центральними елементами аналізу.

**Моделі глибокого навчання (Deep Learning) та їх застосування для класифікації емоцій.**

Поява глибинних нейронних мереж стала моментом, коли аналіз текстів зробив різкий стрибок уперед. Класичні підходи бачили текст лише як набір ізольованих частин, тоді як ембедінги подають його як структуру пов'язаних між собою точок. Але глибинне навчання дозволило працювати не тільки зі словами, а й із послідовностями, залежностями, контекстами, інтонаційною «геометрією» фраз. Тобто модель нарешті отримала можливість хоча б частково враховувати те, що людина помічає інтуїтивно.

Першими вагомими проривами стали рекурентні мережі, зокрема LSTM і GRU [38]. Вони навчилися запам'ятовувати попередні частини тексту, а не дивитися на слова одне за одним у відриві. Такий підхід особливо важливий для емоційного аналізу, бо емоція нерідко проявляється у зміні тону, у переходах між мікротемами, у певних синтаксичних коливаннях. Скажімо, текст може починатися рівно, але ближче до кінця переходити в уривчасті вислови, і LSTM розпізнає цю зміну набагато краще ніж словникові методи.

Згодом з’явилися конволюційні мережі для тексту, які сприймають повідомлення як своєрідну хвильову структуру. Вони добре розпізнають локальні патерни, тобто невеликі фрагменти тексту з характерним емоційним забарвленням [39]. Наприклад, поєднання певних вигуків, повторюваних часток або коротких речень. Для соцмереж, де багато мікроформулювань, такі моделі теж показали непогані результати.

Але справжня революція почалася з появою трансформерів. Ця архітектура кардинально змінила підхід до аналізу природної мови: тепер модель може враховувати зв’язки між будь-якими частинами речення, а не лише сусідніми [40]. Трансформери здатні «бачити» структуру тексту як карту, де кожна частина потенційно пов’язана з будь-якою іншою. Саме це дозволяє їм вловлювати складні емоційні сигнали, які не вписуються у прості лінійні схеми.

У задачах емоційного аналізу трансформери виявилися надзвичайно ефективними. Моделі на кшталт BERT, RoBERTa, DistilBERT та їхні багатомовні або спеціально натреновані версії навчилися працювати з текстами, де емоція подана непрямо [41]. Наприклад, вони можуть розрізняти:

– «я більше так не можу» як ознаку високого стресу,

– «я більше так не хочу» як зовсім інший емоційний стан.

Такий рівень чутливості був недосяжний для лексичних моделей.

Ще одна перевага глибинних нейронних мереж полягає в їхній здатності працювати з шумними даними. Соцмережеві тексти можуть бути частково українською, частково англійською, із дивними конструкціями, пропущеними словами. Моделі глибокого навчання краще пристосовані до таких умов. Вони не потребують ідеального тексту — навпаки, їх можна навчити працювати з реальними прикладами, де все перемішано і де емоція проявляється саме у цій суміші.

Особливо важливо, що у глибинних моделей з’явилася можливість оцінювати не тільки «що сказано», а й «як сказано». Наприклад, зміни в частоті вигуків, динаміка пунктуації, коливання довжини фраз. Для аналізу стресу такі

ознаки дуже цінні, оскільки напруженість не завжди виражається прямо. Нерідко вона проступає через стиль письма, і трансформери здатні це вловити.

Усе це робить моделі глибокого навчання ключовим інструментом у сучасному аналізі емоцій. Вони не ідеальні, тому що потребують великих обчислювальних ресурсів, значних корпусів даних, ретельного навчання. Але їхня здатність бачити складні структури робить їх практично незамінними в задачах, де потрібна тонка інтерпретація мовлення. Для оцінки стресу в соцмережах саме ці моделі дозволяють отримати найбільш реалістичну картину, близьку до того, як емоції читає людина.

## Висновки до розділу 1

У першому розділі було поєднано кілька напрямів, які на перший погляд належать до різних дисциплін, але разом формують основу для подальшого аналізу стресу за текстами соціальних мереж. Стрес постає не як одновимірний показник, а як багаторівневий процес, що проявляється через емоційні реакції, когнітивні зміни та характерні лінгвістичні патерни. Саме ця багатокомпонентність робить його складним для вимірювання, але й дає можливість формалізувати окремі елементи, перетворюючи їх на ті ознаки, з якими може працювати алгоритм.

Соціальні мережі, попри хаотичність і надмірну строкатість, створюють унікальне джерело даних про емоційний стан користувачів. Великий обсяг, швидкість появи та різноманітність текстів роблять їх цінними для дослідження короткострокових і довгострокових змін у настроях населення. Водночас саме ці властивості створюють технічні труднощі: дані потрібно очистити, виявити шум, нормалізувати та часто доповнити географічною прив’язкою, без якої неможливо сформувати локальні оцінки стресу.

Технічні інструменти NLP, що розглянуті у розділі, демонструють еволюцію від простих словникових методів до складних глибинних моделей. Кожен із підходів привносить щось своє: лексичні моделі дають базове

розуміння тональності; векторизація у вигляді ембедінгів дозволяє побачити семантичні зв’язки; глибинні нейронні мережі та трансформери забезпечують здатність враховувати контекст і приховані емоційні сигнали. Усі ці методи утворюють технічну базу, на якій можна будувати систему визначення рівня стресу в текстах соціальних мереж.

У підсумку перший розділ задає логіку, без якої розроблення такої системи є неможливим. Він показує, що дослідження стресу охоплює не тільки роботу з даними, а й теоретичні підходи, які розкривають сутність явища. Це водночас і технічне завдання, і питання інтерпретації: які ознаки можна вважати проявами стресу, а які є просто шумом. Така комплексність дозволяє перейти в наступних розділах до моделювання, алгоритмізації та реалізації системи, яка намагається з’єднати знання про людину з обчислювальними методами аналізу текстів.

# РОЗДІЛ 2
# АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ, ІНСТРУМЕНТІВ ТА МОДЕЛЕЙ ДЛЯ ОЦІНЮВАННЯ СТРЕСУ НА ОСНОВІ ЦИФРОВИХ ДАНИХ

## 2.1. Аналіз сучасних підходів до оцінювання стресу

У дослідженнях емоційних станів за текстами соціальних мереж є багато підходів, і вони помітно відрізняються між собою не лише технічною частиною, а й тим, як узагалі розуміють стрес. Колись усе починалося з намагання знайти окремі мовні ознаки, які «видають» емоції людини. Зараз це значно ширша картина: алгоритми працюють із контекстом, структурою мовлення, інколи навіть із поведінковими слідами користувача. У цьому підрозділі зібрані кілька найважливіших напрямів, які пояснюють, як саме стрес можна спробувати побачити у тексті, чому різні моделі дають різні результати та з яких причин сучасні системи все частіше відходять від простих правил і переходять до складних нейронних архітектур.

**Моделі емоційного аналізу у психолінгвістиці.**

У класичних психолінгвістичних дослідженнях стрес переважно намагалися розпізнати за набором лінгвістичних індикаторів. Це були зміни в доборі слів, поява тривожної лексики, короткі й уривчасті речення або, навпаки, довгі плутані конструкції. Такий підхід, хоч і здається прямолінійним, виник не випадково: мовлення дійсно відчутно змінюється під впливом напруження. Людина буквально «просідає» у синтаксисі, губить плавність думки, починає повторювати однакові слова або надмірно деталізувати негативні теми.

Психологи й лінгвісти використовували це як основу для створення списків індикаторів. Умовно ці ознаки поділяли на кілька груп: емоційно забарвлені слова, когнітивні маркери (наприклад, надмірні заперечення), зміни стилю. На практиці такі підходи непогано працювали в контрольованих умовах: наприклад, у дослідженнях, де учасники виконували завдання зі штучно

підвищеним стресом. Проблема виникала тоді, коли ці методи переносили на реальний цифровий контент.

Стиль онлайн-комунікації зазвичай хаотичний і нестандартний, із великою кількістю сленгу, жартів та сарказму. Тому лінгвістичні правила часто починали давати надто грубі результати. Деякі індикатори дійсно були стійкими: збільшення негативних слів, фрагментарність письма, зміни темпу публікацій. Інші виявлялися ненадійними, адже користувач може писати різко чи емоційно зовсім не через стрес.

Порівняння rule-based і data-driven підходів показало одну закономірність: правила добре фіксують окремі прояви, але погано справляються з контекстом. Моделі, натреновані на численних прикладах, значно краще вловлюють нюанси, проте їм потрібні великі корпуси. У реальних задачах ці два підходи часом навіть поєднують, аби компенсувати недоліки кожного з них.

**Алгоритми визначення стресу у соціальних мережах.**

У роботах, присвячених цифровому аналізу емоцій, особливу увагу приділяють соціальним платформам, зокрема Twitter, Reddit, Facebook і Telegram. Саме там легко знайти швидкі емоційні реакції людей на події. Багато досліджень, наприклад ті, що працювали з Twitter Stress Dataset, будували моделі на основі маркованих прикладів: користувач, який раптом починає писати різко, використовує слова про небезпеку чи втому, класифікується як «стресогенний профіль». Моделі намагаються виявити не лише негативне забарвлення, а й повторювані патерни, тобто теми, що раптово стають домінантними, зміни у ритмі повідомлень або ситуації, коли люди пишуть багато коротких фрагментів замість довших висловлювань.

Для аналізу стресу використовують кілька типів ознак, і найпомітнішим серед них є тональність тексту. Вона дає базове уявлення про емоційний фон, хоча інколи він вводить в оману: іронія або сарказм легко обманюють поверхневі моделі. Інший клас ознак становлять лінгвістичні структури, зокрема зміни синтаксису, надмірна простота або, навпаки, збитий ритм речень. Досить

інформативними виявилися й поведінкові сигнали: різка зміна активності, нерівний темп, хвилеподібна кількість постів.

Проблеми точності часто пов’язані з неоднорідністю користувачів. Усі пишуть по-різному: один активно використовує емоційні слова навіть коли він спокійний, а інший залишається стриманим завжди, навіть у важкому психологічному стані. Через це моделі можуть давати зміщення в оцінках. Ще одна проблема — репрезентативність. Соціальні мережі не представляють усю популяцію. Більш активні групи можуть створювати ілюзію загального настрою, якого насправді немає.

Попри недосконалості, великий масив робіт демонструє: за правильних умов цифрові тексти можуть непогано відображати рівень стресу у спільноті. За окремими користувачами це робити складніше, але на рівні груп чи міст цілком можливо.

**Порівняння класичних та сучасних NLP-підходів.**

Для аналізу емоційного стану використовують різні покоління NLP-методів, і кожне має власну логіку та сильні сторони. Словникові моделі, хоч вони вже й виглядають застарілими, досі застосовують як базову оцінку тону тексту. Вони швидкі та прозорі, але погано працюють у складних випадках, де значення фрази залежить від контексту.

Методи традиційного машинного навчання (SVM або логістична регресія з TF-IDF) дають кращу якість і дозволяють моделі частково враховувати структуру корпусу. Однак вони все ще «пласкі»: документ розбивається на ваги слів і втрачає послідовність. У задачах стресу такі моделі працювали найкраще там, де тексти відносно прямі й без надмірної образності. Соціальні мережі мають принципово іншу ситуацію.

Поява LSTM і GRU стала своєрідним мостом до контекстних підходів. Ці моделі навчилися тримати в пам’яті попередні елементи послідовності, тому могли вловлювати тонкі зміни ритму й структури мовлення. У розпізнаванні стресу це було особливо помітно: речення з раптовими паузами, повторюванням

або дивними переходами давали сигнал моделі, що з користувачем щось відбувається.

Справжній стрибок уперед стався з появою трансформерів. BERT, RoBERTa та інші моделі працюють із контекстом одразу в обидва боки. Це означає, що вони одночасно враховують локальні й глобальні зв'язки у тексті. У задачах емоційного аналізу такі моделі здатні розпізнавати навіть непрямі, завуальовані форми тривоги. Наприклад, коли автор пише без негативної лексики, але обирає специфічні контексти чи структури речень.

Для української мови важливим виявився той факт, що багатомовні моделі, як-от mBERT, справляються доволі стабільно, але все одно поступаються спеціалізованим україномовним моделям. Наприклад, ukr-roberta-base або lang-uk/bert значно краще вловлюють морфологію, поведінку слів і контекст у наших реаліях. Для оцінювання стресу це критично: українські тексти мають свої особливості, і моделі, спеціально натреновані на локальних корпусах, працюють точніше й впевненіше, особливо коли зустрічаються з розмовними формами або змішаними стилями.

### 2.2. Огляд існуючих систем моніторингу емоційних станів

У сфері аналізу емоційних процесів у суспільстві вже існує чимало сервісів, хоча більшість з них створювалися не для вимірювання саме стресу. Вони більше про громадську думку, про тональність дискусій, про те, як поширюється інформація. Частина систем працює на якісному аналізі, а інші ґрунтуються виключно на даних і алгоритмах. У всіх є свої переваги, але є й помітні обмеження, особливо коли спробувати застосувати їх для регіонального оцінювання емоційного стану населення. У цьому підрозділі зібрано кілька груп таких сервісів і коротко показано, чим вони корисні, а що в них не підходить для задачі цієї роботи.

**Системи моніторингу громадської думки.**

До найпомітніших платформ цієї групи належать Brand Analytics, YouScan та інші сервіси соціального моніторингу, що аналізують реакції аудиторії. Вони працюють здебільшого з публічними даними: коментарями, постами, згадками брендів або подій. Основний фокус таких інструментів полягає у відстежуванні тональності та ключових тем. Вони добре показують, які емоції домінують у публічних дискусіях, як швидко зростає негатив, які теми раптом стають «гарячими».

Проте ці сервіси, попри гнучкість, фактично не призначені для вимірювання стресу. Їхні моделі натреновані на виявленні позитивних чи негативних реакцій, а не на фіксації складнішого емоційного стану. Стрес і тональність не збігаються за змістом. Людина може писати негативно з різних причин: від сатири до політичної критики. І навпаки, текст може виглядати нейтральним, але містити ознаки внутрішньої напруги.

Google Trends інколи використовують у дослідженнях громадських настроїв як додаткове джерело сигналів. Він показує, що саме люди шукають у певний момент. У кризові періоди це справді дає непрямі індикатори колективної тривоги. Але цей інструмент не працює з текстами та не дає емоційної інформації. Він радше доповнює аналіз, ніж формує його основу.

Для українських текстів більшість міжнародних платформ працюють неідеально. Часто це видно навіть у базових класифікаторах тональності: вони помиляються зі сленгом, не враховують змішану українсько-російську комунікацію, ігнорують локальні меми чи культурні маркери. Тому ці системи, хоч і потужні, не дають потрібного рівня точності у завданні, де важливо помічати тонкі лінгвістичні зсуви.

**Системи оцінювання соціальних ризиків.**

Є окрема група платформ, які зосереджуються на аналізі кризових ситуацій. Це напрям, який інколи називають crisis informatics. Такі системи збирають інформацію про події, що можуть бути небезпечними для населення:

техногенні аварії, стихійні лиха, військові дії, економічні обвали. Вони працюють на стику журналістики, аналітики даних і соціології. Їхня головна мета — визначення масштабів і динаміки подій, а не вимірювання психологічного стану людей.

Існують дослідження, де стрес аналізується як частина реакції суспільства на небезпечні події. Наприклад, коли в певному регіоні відбуваються ракетні удари, кількість публікацій із тривожними темами різко зростає. У таких випадках моделі намагаються виокремити «стресові хвилі», пов'язані зі значущими подіями. Це корисно, але підходить лише для великих суспільних потрясінь. У повсякденному житті, коли психологічна напруга формується поступово, такі інструменти майже не працюють.

У деяких системах навіть створюють карти ризиків, які показують, де саме зростає кількість повідомлень про критичні ситуації. Але це карта подій, а не карта емоцій. Вона відображає інтенсивність інформаційного поля, а не психологічний стан людей. Для нашої задачі цього замало: нам потрібно оцінювати саме емоційний рівень, а не інфопотік.

**Порівняння наявних сервісів із завданнями роботи.**

Якщо спробувати зіставити існуючі платформи з ідеєю системи, яка має визначати стрес у конкретному місті, виникає кілька ключових розбіжностей.

По-перше, більшість сервісів не працюють на локальному рівні. Вони аналізують національні тренди, популярні теми або реакції на великі події. Але майже ніхто не надає аналітики «місто-до-міста», де важливі саме локальні фактори: транспортні проблеми, економічні зміни, безпекові ризики, характерні для конкретного регіону.

По-друге, моделі цих систем переважно орієнтовані на тональність, а не на стрес. Негатив — це не обов'язково стрес, а стрес — не завжди негатив. Така підміна призводить до того, що реальні емоційні стани губляться в потоці інших сигналів.

По-третє, для української мови підтримка або формальна, або часткова. Деякі сервіси взагалі не вміють коректно працювати зі змішаними текстами чи українським сленгом. Без цього неможливо чітко відстежувати зміну емоційного тону в реальних онлайн-спільнотах.

Є ще одна відмінність, яка особливо важлива в контексті цієї роботи: жодна з існуючих систем не формує інтерпретованого показника «рівня стресу». Це завжди або тональність, або нейтральні теми, або індикатори ризику. А досліднику потрібна модель, яка могла б показати: ось тут користувачі переживають напруження, ось тут ситуація покращується, а ось тут є конкретні чинники, які створюють емоційний тиск.

Через це виникає потреба у новій системі, більш вузькій і спрямованій саме на аналіз стресу, а не на ширшу емоційну картину. Системі, яка працюватиме з українськими текстами, визначатиме локальні тенденції і даватиме зрозумілий показник, який можна інтерпретувати й порівнювати між містами. Ця робота формує концептуальну основу такої системи.

## 2.3. Аналіз інструментів і технологій для збору та опрацювання даних

Щоб система могла визначати рівень стресу в місті, їй потрібні дані. Причому не абстрактні дані, а текстові повідомлення, які користувачі залишають у відкритих джерелах. Це наче сировина, з якої пізніше формується аналітика. Збирати такі дані непросто: соцмережі працюють за своїми правилами, новинні сайти відрізняються структурою, а телеграм-канали мають власні технічні обмеження. Тому інструменти для збору текстів доводиться підбирати під кожне джерело окремо, інколи комбінуючи кілька методів. У цьому підрозділі розглянуто найпоширеніші підходи та інструменти, які реально застосовують у системах такого типу.

**Методи збору текстових даних.**

Збір даних у цифровому просторі зазвичай починають із web scraping. Це класичний метод, коли програма «проходить» сторінку сайту, зчитує html-

структуру і витягує потрібні текстові елементи: заголовки, абзаци, коментарі. Скрейпінг корисний для новинних сайтів або публічних каталогів, де інформація має більш-менш передбачувану структуру. Проблема лише в тому, що кожен сайт виглядає по-своєму, і під кожне джерело доводиться писати окремий парсер.

Інший спосіб полягає у використанні офіційних API соціальних мереж. Це найбільш стабільний та «правильний» метод, але тут виникають обмеження: деякі платформи сильно обмежують доступ або дають урізані версії API. Наприклад, Twitter у різні періоди взагалі закривав ключові ендпоїнти [42]. Facebook вимагає реєстрацію додатка з детальним описом мети [43]. Telegram має API, але для публічних каналів частіше використовують неофіційні бібліотеки, бо вони простіші й стабільніші [44] в реальній роботі.

RSS-стрічки новин залишаються простим, хоч і трохи старомодним способом отримувати текст. Вони дають чисту структуру: заголовок, опис, посилання. Для аналізу стресу це зручно, коли потрібні не особисті висловлювання, а інформаційний контекст, тобто які теми обговорюють і на що реагує аудиторія.

Окремо варто згадати Telegram і Facebook як джерела даних, актуальні саме для України. Telegram зараз фактично став основним майданчиком для новинних каналів і регіональних спільнот. У таких каналах багато коротких повідомлень, які добре фіксують локальні події. Facebook, хоч і менш активний, дає велику кількість коментарів, а це свого роду «живий» матеріал, де часто проступає емоційний фон. Обидва джерела потребують попереднього очищення, але дають важливі сигнали для побудови карти стресу.

**Інструменти для збору та автоматизації.**

Для web scraping найчастіше використовують Selenium або Playwright. Обидва інструменти дозволяють імітувати реальний браузер: відкривати сторінки, прокручувати їх, взаємодіяти з елементами [45]. Це потрібно там, де контент завантажується динамічно або ховається за скриптами [46]. Selenium

трохи повільніший, але простіший у використанні. Playwright працює швидше і стабільніше, особливо на сучасних сайтах.

У Python існує ціла екосистема для збору текстів, і Scrapy є одним із найпотужніших інструментів. Він підходить для систем, де треба регулярно сканувати велику кількість сайтів. Scrapy дозволяє будувати конвеєри: спочатку збір тексту, потім очищення, а далі збереження. У таких задачах зручність конвеєра виявляється важливішою, ніж швидкість окремого запиту.

Збір даних завжди пов'язаний із технічними обмеженнями. Найпоширеніший обмежувач — rate limits. Сайти та API накладають обмеження на кількість запитів. Якщо їх порушити, доступ можуть заблокувати. Іншою проблемою є захисні механізми платформ, зокрема CAPTCHA і антибот-системи. Тому збирати дані доводиться обережно: робити затримки, використовувати черги запитів, уникати підозрілої активності.

Ще один аспект стосується вимог до стійкості. Якщо збирати дані з десятків джерел, щось обов'язково падає. Канал може зникнути, сайт може змінити структуру, а API може повернути помилку. Тому системи збору будують із запасом міцності: логування, повторні спроби, резервні джерела.

**Огляд інструментів для зберігання даних.**

Після збору текст потрібно десь зберігати. У більшості випадків для цього використовують PostgreSQL. Це реляційна база, яка добре працює з текстами та дає можливість індексувати їх через спеціалізовані механізми. Якщо структура даних складна або змінюється, у пригоді стає JSONB, формат, що дає змогу зберігати документи у гнучкому вигляді. Це корисно для соціальних мереж, де повідомлення можуть мати різну структуру залежно від джерела.

У деяких випадках використовують MongoDB, оскільки ця документна база не вимагає фіксованих схем. Вона зручна для прототипів і швидких експериментів, але в задачах аналізу великих корпусів PostgreSQL зазвичай надійніший. У ньому краще організований пошук, транзакції та складні запити.

Для системи, яку описує ця робота, PostgreSQL виглядає оптимальним варіантом.

Усі ці інструменти, від скрейперів до систем зберігання даних, формують технічну основу, без якої система не зможе працювати. Вони забезпечують постійний потік даних, з якого пізніше формується модель. Процеси збору, автоматизації й збереження становлять базовий шар архітектури, що дає змогу переходити до аналізу стресу.

### 2.4. Аналіз моделей NLP, які придатні для оцінювання рівня стресу

Для побудови системи, яка здатна визначати рівень стресу за текстами, важливо розуміти, які саме моделі можуть працювати із цією задачею. Традиційні підходи добре працюють із тональністю, але стрес є складнішим станом: він проявляється не лише в емоційних словах, а й у структурі мовлення, контекстах і повторюваних патернах. Тому в сучасних системах основну роль відіграють саме контекстні моделі [47]. У цьому підрозділі зібрані ті архітектури, які реально можна застосовувати для українських текстів і які показують найкращий баланс між якістю та обчислювальною вартістю.

**Україномовні моделі.**

Найсильніші результати для українських текстів дають спеціалізовані моделі, натреновані на локальних корпусах. Однією з найбільш відомих є ukr-roberta-base. Це модель, побудована на архітектурі RoBERTa, адаптована під українську мову. Вона добре працює зі складністю морфології, з гнучким порядком слів, зі стилістичними змінами. У текстах соціальних мереж такі деталі мають велике значення, бо там рідко зустрічаються ідеально оформлені речення.

Інша важлива модель — lang-uk/bert. Вона дещо «легша» за RoBERTa, але теж стабільна і непогано обробляє змішані тексти, де українська перетинається з англійською або російською. У задачах, де потрібно багато коротких фрагментів (пости, репліки, повідомлення), ця модель інколи навіть працює швидше, даючи результат без зайвих затримок.

Є й мультимовні моделі, наприклад multilingual BERT. Вони виглядають універсальними, і на перший погляд здається, що вони здатні впоратися з українською. Але на практиці їхні результати помітно слабші. Мультимовні моделі навчені на десятках мов одночасно, і тому жодна з них не отримує достатньо якісних представлень. Це особливо помітно при аналізі емоцій. Українська має свої нюанси: суфікси, чергування, порядок слів, перенесення інтонації. У мультимовних моделей ці властивості розмиті.

Тому для проєкту, де потрібна не просто тональність, а саме визначення рівня стресу, україномовні моделі дають значно надійніші результати. Вони краще вловлюють дрібні зміни, працюють зі сленгом, підлаштовуються під реальний стиль письма користувачів.

**Порівняльний аналіз точності.**

За даними досліджень, україномовні трансформери показують суттєво вищі метрики, ніж класичні ML-моделі чи мультимовні аналоги. На різних наборах даних оцінка F1 для ukr-roberta-base може перевищувати 0.85, а інколи й підходити до 0.90. Це стосується саме задач класифікації емоцій та схожих категорій.

ML-моделі на основі TF-IDF і SVM, у свою чергу, рідко підіймаються вище 0.70–0.75. Вони стабільні, але не вміють працювати з контекстом. Для тональності цього інколи достатньо, але для стресу вже ні. У стресових повідомленнях часто важливі не конкретні слова, а спосіб побудови фрази. Наприклад, повторення думки, раптове звуження теми, дивна структура речень. Трансформери здатні це виявити, а класичні алгоритми ні.

Multilingual BERT дає показники на рівні 0.75–0.80 для українських текстів. Непогано, але відчутно нижче, ніж україномовні моделі. У задачах, де важливо вловити тонкі емоційні патерни, різниця цих 5–10% відчувається дуже сильно.

Ще одна важлива річ стосується стабільності. Деякі моделі дають хороші результати на тестовому наборі, але «плавають» на нових даних. Україномовні

моделі зазвичай виявляються стійкішими, оскільки навчалися на корпусах, близьких до реальних користувацьких текстів.

**Вибір моделі для роботи.**

Для системи, яка аналізує рівень стресу за текстами в соціальних мережах, оптимальним виглядає використання саме модель ukr-roberta-base або її модифікації. Причин кілька.

По-перше, ця модель працює з контекстом. Вона не лише дивиться на окремі слова, а й оцінює їхнє оточення, структурний ритм, зв'язки між частинами фрази. У текстах стрес часто ховається в тому, як сказано, а не в тому, що сказано.

По-друге, ця модель добре впорається зі змішаними стилями. У соцмережах легко зустріти речення українською вперемішу з англійською термінологією або російськими цитатами. Модель, натренована на локальних корпусах, поводиться значно впевненіше.

По-третє, ukr-roberta-base забезпечує баланс між точністю й обчислювальними витратами. Модель досить компактна, щоб її можна було використовувати на середній серверній конфігурації, але водночас достатньо потужна для складних задач.

І зрештою, ця модель здатна виявляти непрямі сигнали емоційної напруги, які не зчитуються простими словниковими методами. Саме такі маркери потрібні для формування показника рівня стресу в місті.

У цьому сенсі вибір моделі є не просто технічним питанням. Це фундаментальний компонент архітектури всієї системи. Якщо модель не вміє працювати з українськими текстами або не бачить складні патерни, то подальший аналіз втрачає сенс [48]. Тому в цій роботі основою NLP-модуля виступає саме сучасна трансформерна модель українського мовного середовища.

### 2.5. Проєктування архітектури вебсайту оцінювання стресу

Щоб можна було визначати рівень стресу для конкретного міста, потрібна чітко продумана архітектура [49]. Не просто набір скриптів чи окремих моделей, а структура, де кожен компонент виконує свою функцію і водночас не заважає іншим. Архітектура у цьому випадку виступає своєрідною картою процесів: де беруться дані, як вони рухаються, хто їх обробляє, де вони зберігаються, що саме повертається користувачу [50]. У цьому підрозділі описано основні частини майбутього вебсайту, вибір технологій та логіку, яка робить його цілісним.

**Загальна архітектура вебсайту оцінювання стресу.**

Архітектура вебсайту будується навколо кількох модулів, які працюють послідовно, але можуть існувати як окремі частини. Перший елемент — модуль збору даних. Він отримує тексти з джерел, очищує їх від зайвих елементів (html, емодзі, маркерів посилань), виділяє дату, автора (якщо можливо) й місто або інший географічний контекст. Модуль збору працює автономно: він не залежить від NLP-моделі та не пов’язаний із фронтендом.

Другий компонент — модуль збереження. Тут зібрані тексти перетворюються на структуровані записи. У збереженні важливо мати і «сиру» версію тексту, і очищену, бо інколи результати аналізу потрібно буде перевірити вручну. База даних зберігає також службову інформацію: джерело, час публікації, можливе посилання на оригінал.

Далі йде модуль обробки, де текст проходить preprocessing. Це не просто видалення зайвих символів. Тут можуть виконуватися нормалізація, розбиття на речення, лематизація. Частина цих кроків залежить від того, як працює конкретна модель, адже деякі трансформери можуть самостійно обробляти «сирі» фрагменти, а інші потребують ретельнішого очищення.

Після обробки дані потрапляють у NLP-модуль класифікації. Це ядро системи. Модель отримує текст і повертає оцінку: чи містить він ознаки стресу, який рівень емоційної напруги можна припустити, наскільки впевнений алгоритм у результаті.

Коли всі окремі тексти класифіковані, спрацьовує модуль агрегації. Він об’єднує результати за містом і за періодом. Наприклад, можна порахувати середній рівень стресу за добу, тиждень чи місяць. Також тут формується статистика: кількість «стресових» повідомлень, графік зміни настроїв, часті тригерні теми.

API бекенду — це частина системи, яка надає результати у зовнішній світ. Через API фронтенд отримує дані для побудови дашборду. Логіка API має бути простою: запит → назва міста → структура з показниками.

Завершальним компонентом є фронтенд-дешборд. Тут користувач бачить уже оброблену інформацію: графіки, відсоток стресу, ключові теми. Дашборд не виконує складного аналізу — він лише відображає дані, які надходять із бекенду.

Загальна архітектура виглядає як ланцюжок послідовних модулів, але між ними небагато жорстких залежностей. Це дозволяє змінювати окремі частини без руйнування всієї системи.

**Вибір технологій.**

Бекенд найзручніше реалізувати на Django. Він надає просту структуру для роботи з API, має ORM для доступу до PostgreSQL і дозволяє організувати обробку даних без зайвої бюрократії. Python тут природний вибір, бо більшість NLP-інструментів існує саме в його екосистемі.

Для асинхронних задач інколи використовують Node.js, але в цьому проєкті це радше опція. Якщо потрібні фонові задачі, можна використати Celery у парі з Redis. Це дає більш рівномірний розподіл навантаження під час збору або періодичної класифікації постів.

PostgreSQL обирають як основну базу даних через стабільність, індексацію тексту та підтримку JSONB. У реальній роботі це допомагає формувати гнучкі записи, не створюючи окремих таблиць для дрібних атрибутів повідомлень.

Фронтенд побудований на React і TypeScript. React дозволяє швидко створювати інтерактивні компоненти для дашборду. TypeScript додає структурності, що корисно, коли фронтенд працює зі складною відповіддю API.

MUI використовується для дизайну — він дає готові компоненти, які виглядають достатньо сучасно. Це важливо, особливо коли немає часу на розробку власної дизайн-системи.

Контейнеризація через Docker забезпечує стабільний запуск програми на різних серверах. Це спрощує деплой і дозволяє уникнути проблем зі залежностями.

Для розгортання вебсайту підходять GitHub Actions та AWS EC2. GitHub Actions може автоматично збирати контейнери при пуші в репозиторій. EC2 надає просте серверне середовище для запуску контейнерів, що підходить для проєкту середнього розміру.

**Обґрунтування архітектурних рішень.**

Вибір між monorepo та розділеним бекендом і фронтендом залежить від структури команди. У цьому проєкті monorepo буде простішим. Усі частини знаходяться в одному місці, легше налаштувати деплой, легше синхронізувати зміни.

REST API є найбільш зрозумілою моделлю для такого застосунку. Дані чітко структуруються: місто, дата, показники. REST добре працює навіть тоді, коли фронтенд або бекенд оновлюється незалежно. GraphQL тут не дає суттєвих переваг.

Docker використовують для того, щоб уникнути проблем із середовищем. На різних серверах різні можуть бути різні версії Python, PostgreSQL або бібліотек — контейнери вирішують це, ізолюючи систему.

PostgreSQL кращий за MongoDB у цій роботі, бо для аналізу потрібні складні вибірки, агрегації, робота зі статистикою. MongoDB зручний для неструктурованих документів, але PostgreSQL стабільніший і дає більше інструментів для точних запитів.

## 2.6. Формування вимог до вебсайту та критеріїв оцінювання якості роботи

Щоб все працювало передбачувано і давало результати, які можна інтерпретувати, потрібен чіткий набір вимог. Вони визначають, що саме додаток має вміти, як швидко реагувати, які показники надавати користувачу і наскільки стабільно поводитися в умовах «шумних» цифрових даних [49]. У цьому підрозділі вимоги розділено на функціональні, нефункціональні та метрики, що дозволяють оцінити якість самої NLP-моделі.

**Функціональні вимоги.**

Вебсайт повинен підтримувати введення назви міста користувачем, оскільки це є основним сценарієм. Результат повинен повертатися у структурованому вигляді: числовий показник рівня стресу, побудований на основі аналізу зібраних текстів. Поряд із цим важливо показати графік зміни рівня стресу за певний період. Динаміка часто дає більше інформації, ніж одиничне значення.

Ще одна функція — визначення топових емоцій, які домінують у текстах. Це допомагає зрозуміти, які саме настрої впливають на формування стресу: тривога, роздратування, втома чи емоційне виснаження.

Крім того, вебсайт повинен показувати «тригери», тобто теми, що найчастіше з’являються у стресових повідомленнях. Це можуть бути новини, події, локальні проблеми. Інколи саме перелік цих тригерів дозволяє зрозуміти, звідки береться емоційна напруга в певному регіоні.

Важливо також мати аналіз тональності. Тональність не є показником стресу, але виступає допоміжним виміром, який дозволяє бачити загальний емоційний фон. Часто поєднання тональності й рівня стресу дає повнішу картину.

**Нефункціональні вимоги.**

Продуктивність має залишатися стабільною навіть тоді, коли дані надходять у великих обсягах. Система не повинна «застигати» під час агрегації

чи класифікації. Це особливо важливо для міських даних, які можуть різко збільшуватися під час кризових подій.

Масштабованість — ще один ключовий аспект. Якщо в майбутньому потрібно буде додати нові міста, джерела або типи даних, архітектура повинна це дозволяти без переробки всієї системи.

Стійкість до шуму також є обов'язковою вимогою. Тексти соціальних мереж неструктуровані, містять жаргон, емоційні вставки, випадкові символи. Модель має працювати впевнено навіть тоді, коли значна частина повідомлень складена недбало або з помилками.

Важливо також дотримуватися етичних вимог і норм конфіденційності. Аналіз проводиться лише на відкритих джерелах. Система не повинна зберігати зайві дані, які можуть ідентифікувати особу. У деяких випадках доводиться приховувати або хешувати частину метаданих.

**Метрики оцінки моделі.**

Для оцінювання якості моделі використовують кілька груп метрик. Найочевидніші — Accuracy та F1. Вони показують, наскільки правильно модель класифікує повідомлення. Проте цього замало. Стрес не є простою категорією, інколи це радше умовна шкала, а не чіткий клас, тому в задачах регресії доцільно застосувати MAE. Він показує, наскільки «у середньому» помиляється модель.

Окрема група — метрики довіри. Вони відображають, наскільки впевнено модель робить прогноз. У системах, які працюють із шумними даними, важливо не тільки знати відповідь, а й розуміти її надійність.

Стабільність оцінки — ще один критерій. Модель повинна поводитися послідовно на різних наборах даних. Якщо коефіцієнти різко змінюються при зміні джерела, це сигнал, що в алгоритмі є слабкі місця.

## Висновки до розділу 2

Другий розділ показав, що створення вебсайту для оцінювання рівня стресу за текстами соціальних мереж варто вважати значно складнішим

завданням, ніж може здаватися на перший погляд. Тут не вистачає просто взяти модель і “пропустити” через неї текст. Потрібно враховувати структуру даних, особливості платформ, мовні відмінності, а також різноманітність підходів, якими користуються сучасні дослідження для аналізу емоційних станів.

Аналіз наукових джерел показав, що навіть традиційні методи психолінгвістики можуть бути корисними, але лише як допоміжний елемент. Реальні дані соціальних мереж не піддаються жорстким правилам. Тому більш ефективними виявляються data-driven підходи, які не залежать від вручну створених словників і здатні виявляти складні патерни емоційної напруги. Окремі дослідження демонструють, що сучасні трансформерні моделі краще справляються з непередбачуваними стилістичними змінами та шумом, що невід’ємні для цифрових платформ.

Важливо, що під час огляду інструментів стало зрозуміло, що збір даних становить майже половину всієї роботи. Новинні сайти, Telegram-канали, соцмережеві стрічки мають різну структуру, різні обмеження, а інколи — серйозний захист від автоматизованого збору. Тому доводиться комбінувати такі інструменти, як Scrapy, Selenium, RSS-фідери, й будувати систему так, щоб вона витримувала збої та нерівномірність потоку.

Не менш важливим є зберігання даних. PostgreSQL із підтримкою JSONB виявляється оптимальним варіантом, оскільки дозволяє працювати і зі структурованими, і з напівструктурованими текстами. Це особливо корисно, коли потрібно зберігати сирі тексти разом із метаданими, результатами попередньої обробки та оцінками моделі.

У частині вибору моделі стало зрозуміло, що українські трансформерні моделі, як-от ukr-roberta-base, демонструють кращу точність на локальних даних порівняно з багатомовними аналогами. Вони краще розпізнають контекст, природні мовні конструкції й характерні емоційні зміщення, що важливо для задачі визначення стресу.

Архітектура майбутнього вебсайту формується з кількох ключових блоків: модуль збору, модуль збереження, preprocessing, модель NLP, блок агрегації, бекенд і фронтенд. Така структура дозволяє не лише розділити відповідальності, а й забезпечити гнучкість, в результаті чого кожний компонент можна вдосконалювати окремо. Використання Django, PostgreSQL, React, Docker та серверного деплою через EC2 створює достатньо стабільну й універсальну основу для подальшої реалізації.

Загалом цей розділ дав змогу сформувати чітке розуміння того, як повинна виглядати технічна й концептуальна основа системи. Він визначив, які саме інструменти, моделі та методи забезпечать найкращу точність, стабільність і стійкість до шумних даних. І головне — показав, що задача оцінювання стресу потребує комплексного підходу: поєднання сучасних NLP-технологій, коректної архітектури та продуманих алгоритмів збору й обробки даних.

# РОЗДІЛ 3
# РОЗРОБКА ДОДАТКУ ДЛЯ ОЦІНЮВАННЯ РІВНЯ СТРЕСУ НА ОСНОВІ ДАНИХ СОЦІАЛЬНИХ МЕРЕЖ

## 3.1. Архітектура, структура та принципи побудови додатку

Розробка додатку для оцінювання рівня стресу за даними соціальних мереж потребує чіткої, прозорої та модульної архітектури, здатної працювати з великими потоками неструктурованої інформації. З огляду на результати аналітичного огляду в попередньому розділі, а також на технічні вимоги, сформульовані під час проєктування, додаток будується у вигляді багаторівневої структури, де кожен компонент виконує окрему логічну роль і може бути замінений або розширений без істотного впливу на інші модулі.

**Загальний огляд архітектури додатку.**

Додаток складається з кількох основних блоків, які утворюють послідовний конвеєр обробки даних:

1. **Модуль збору даних (Collector):** отримує тексти з різних джерел — новинних сайтів, Telegram-каналів, Facebook та інших платформ, залежно від доступності API чи можливостей web-scraping.
2. **База даних (Storage):** зберігає сирі тексти, очищені варіанти, метадані та результати аналізу.
3. **Модуль попередньої обробки (Preprocessing):** нормалізує текст, видаляє HTML-розмітку, очищує артефакти та готує дані до подальшої класифікації.
4. **NLP-модуль:** трансформерна модель ukr-roberta-base, яка визначає рівень стресу для кожного повідомлення, обчислює ймовірності та допоміжні метрики.
5. **Модуль агрегації:** об’єднує окремі результати в узагальнені показники для кожного міста, формує графік змін і визначає ключові теми та тригерні слова.

6. **API-рівень (Django REST):** надає зовнішній інтерфейс для фронтенду, забезпечує доступ до зібраної інформації у форматі JSON.
7. **Фронтенд-інтерфейс (React):** відображає результати аналізу у вигляді інтерактивного дашборду.

Така структура відповідає принципам модульності та роз'єднаності: кожен компонент вирішує власну задачу і може бути замінений або масштабований без порушення логіки всієї системи. Наприклад, модель NLP можна оновити на більш точну, або змінити формат джерел даних — і це не вплине на дизайн API чи структуру фронтенду.

**Вибір технологічного стеку.**

Архітектурні рішення базуються на інструментах, які найкраще підходять для роботи з текстами українською мовою та побудови веб-інтерфейсу:

- **Python** використовується як основна мова для серверної частини та NLP.
- **Django + Django REST Framework** забезпечують стабільну платформу для API.
- **PostgreSQL** обрана як основна СУБД через підтримку JSONB, GIN-індексів та складних текстових запитів.
- **ukr-roberta-base** — сучасна модель трансформера, оптимізована для українських текстів.
- **React + TypeScript + MUI** — для побудови зручного, швидкого та адаптивного інтерфейсу.
- **Docker** використовується для контейнеризації кожного модуля та забезпечення переносимості.
- **AWS EC2** — середовище розгортання прототипу.

**Структура проєкту.**

Проект будується як набір окремих модулів, організованих у каталоги:

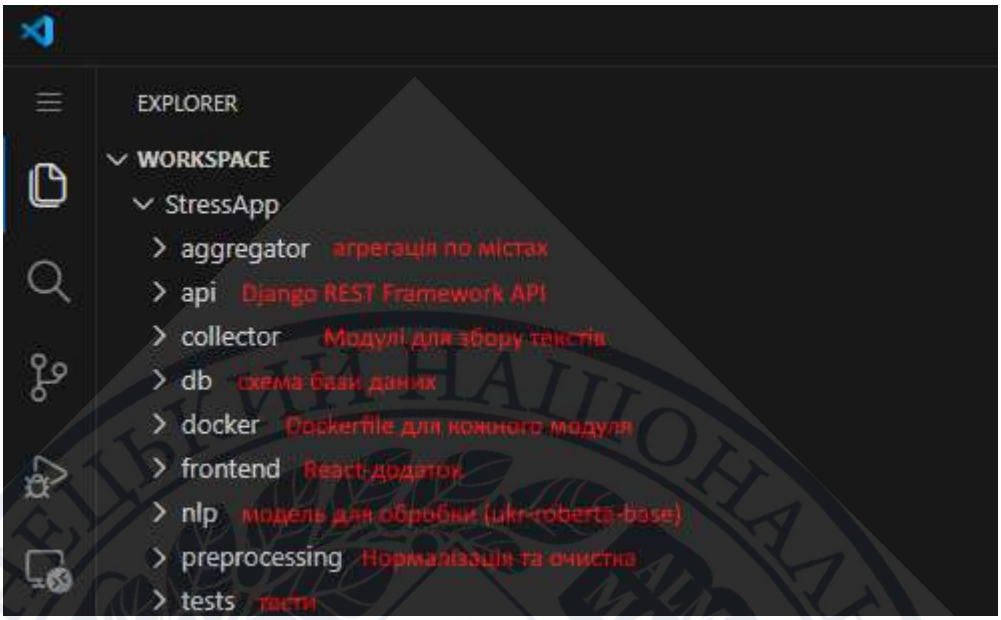


Рис. 3.1 — Файлова структура проєкту

Подібна структура відповідає принципу “чистої архітектури”: логіка не змішується з представленням, а дані — з моделлю. Це демонструє, як дані послідовно проходять усі етапи та перетворюються на готовий результат у вигляді оцінки рівня стресу.

**Архітектура проєкту.**

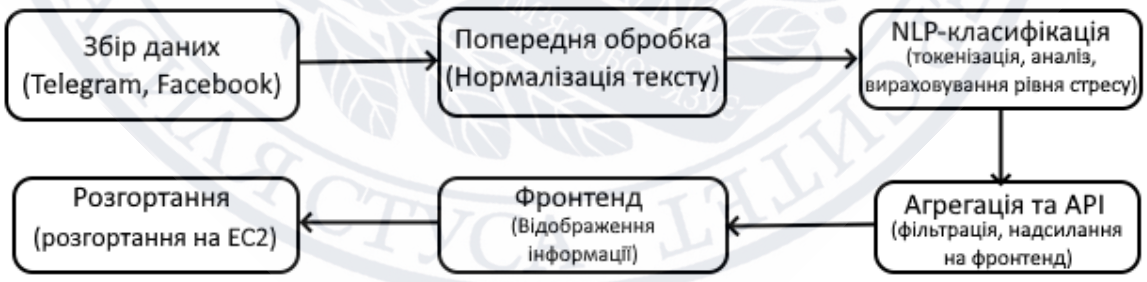


Рис. 3.2. — Архітектура проєкту

**Реалізація модуля збору даних (Collector / Scraper).**

Модуль збору даних є початковою ланкою: від його роботи залежить обсяг, різноманітність і якість текстів, з якими надалі працює NLP-модель. З огляду на це, збір реалізовано як окремий підмодуль, що об’єднує кілька джерел

(новини, Telegram-канали, соцмережі) в єдиний формат і одразу готує базові метадані для подальшої обробки.

Основні вимоги до модуля:

- підтримка кількох різних джерел одночасно;
- стійкість до помилок і змін розмітки сайтів;
- можливість повторних спроб у разі збоїв;
- логування всіх операцій;
- приведення всіх отриманих повідомлень до єдиної структури перед записом у базу даних.

**Збір новин із сайтів (RSS і HTML-scraping).**

Для сайтів, що надають RSS-стрічки, використовується окремий модуль rss_collector.py. Він відповідає за регулярне опитування RSS-джерел, розбір отриманого XML та формування внутрішнього подання повідомлень.

Код файлу collector/rss_collector.py наведено в Додатку А.

Для сайтів без RSS створено Scrapy-спайдера, який обходить сторінки списку новин і витягує основний текст, дату публікації та посилання. Код розміщено у файлі collector/spiders/news_spider.py

Код файлу collector/spiders/news_spider.py наведено в Додатку Б.

Обидва підходи (RSS та Scrapy) працюють у єдиному конвеєрі: після виконання спайдера або завантаження RSS отримані словники передаються далі — у модуль збереження.

**Збір повідомлень із Telegram-каналів.**

Telegram є одним із ключових джерел оперативної інформації в українському сегменті. Для роботи з публічними каналами використовується бібліотека Telethon. Збір реалізовано в окремому модулі telegram_collector.py.

Код файлу collector/telegram_collector.py наведено в Додатку В.

У цьому модулі одразу закладена можливість розширення: можна додати евристики для визначення міста за текстом (city_hint) або розширити meta додатковими полями.

**Отримання текстів із Facebook.**

Код файлу collector/facebook_collector.py наведено в Додатку Г.

Логування, повторні спроби та обробка помилок

Для всіх каналів збору передбачено базовий механізм повторних спроб. Він реалізований у модулі collector/utils.py і використовується там, де можливі тимчасові збої мережі або відповіді сервера.

Код файлу collector/utils.py наведено в Додатку Д.

Ця функція використовується, наприклад, при зверненні до RSS чи API Facebook, що дозволяє захиститися від тимчасових збоїв і нестабільності мережі.

**Формування уніфікованої структури повідомлення та передача в БД.**

Щоб надалі спростити обробку даних, усі зібрані записи приводяться до спільного внутрішнього формату. Для цього створено функцію normalize_message, яка гарантує наявність потрібних полів і встановлює значення за замовчуванням, якщо деякі метадані відсутні.

collector/normalizer.py:

```
from typing import Dict, Any

def normalize_message(raw: Dict[str, Any]) -> Dict[str, Any]:
    return {
        "source": raw.get("source"),
        "raw_text": (raw.get("raw_text") or "").strip(),
        "clean_text": None,
        "published_at": raw.get("published_at"),
        "link": raw.get("link"),
        "meta": {
            "language": raw.get("meta", {}).get("language", "uk"),
            "city_hint": raw.get("meta", {}).get("city_hint"),
            "extra": {k: v for k, v in raw.get("meta", {}).items() if k not in
("language", "city_hint")}
        }
    }
```

Після нормалізації дані можуть бути безпосередньо записані в базу даних через Django ORM. Цей крок описується детальніше у підпункті 3.3, однак логіка інтеграції зі збором виглядає досить просто:

Код файлу collector/pipeline.py наведено в Додатку Е.

Таким чином модуль збору забезпечує:

- одночасну роботу з різними каналами даних;
- стійкість до помилок;
- уніфікацію форми записів;
- готовність даних до подальшої попередньої обробки та класифікації.

**Реалізація бази даних та моделей даних (PostgreSQL + Django ORM).**

База даних є центральним сховищем додатку, де зберігаються сирі та оброблені тексти, результати класифікації, інформація про міста та ключові тригерні теми. Вибір PostgreSQL зумовлений необхідністю працювати з великими обсягами текстів, можливістю індексувати JSONB-поля, а також повною сумісністю із Django ORM.

У цьому підпункті описано структуру моделей, створення таблиць, логіку зберігання метаданих та реалізацію індексації, яка пришвидшує пошук і агрегацію.

**Django-моделі для збереження повідомлень, результатів класифікації та довідників.**

Усі моделі розміщені у файлі db/models.py. Структура бази складається з чотирьох основних таблиць:

1. Message — зберігає сирі та очищені тексти.
2. StressResult — результати NLP-класифікації.
3. City — довідник міст із підтримкою кількох варіантів назв.
4. Topic — найчастіші тригерні теми, які додаток визначає під час агрегації.

Код файлу db/models.py наведено в Додатку Ж.

Ключові моменти:

- JSONField дозволяє зберігати довільну структуру метаданих (емодзі, хештеги, теги);
- Message → StressResult — зв’язок “один до одного”, бо кожне повідомлення класифікується один раз;

- Message → City — зв'язок "багато до одного", але місто може бути й невизначеним;
- Topic → City — теми є специфічними для конкретного **міста.**

**Міграції та створення таблиць у PostgreSQL.**

Нижче фрагмент згенерованої міграції для моделі Message:

db/migrations/0001_initial.py (фрагмент):

```
operations = [
    migrations.CreateModel(
        name='Message',
        fields=[
            ('id', models.AutoField(primary_key=True)),
            ('source', models.CharField(max_length=200)),
            ('raw_text', models.TextField()),
            ('clean_text', models.TextField(null=True, blank=True)),
            ('published_at', models.DateTimeField(null=True, blank=True)),
            ('link', models.TextField(null=True, blank=True)),
            ('meta',
django.contrib.postgres.fields.jsonb.JSONField(default=dict)),
            ('created_at', models.DateTimeField(auto_now_add=True)),
        ],
    ),
]
```

Такі міграції автоматично застосовуються через команду:

```
python manage.py migrate
```

**Використання JSONB та індексації в PostgreSQL.**

Для пришвидшення пошуку за ключовими полями метаданих, зокрема за meta.city_hint або meta.language, створено GIN-індекс.

Індекс додається у міграції:

db/migrations/0002_indexes.py:

```
from django.contrib.postgres.operations import BtreeGistExtension,
CreateExtension
from django.contrib.postgres.indexes import GinIndex
from django.db import migrations

class Migration(migrations.Migration):

    dependencies = [
        ('db', '0001_initial'),
    ]
```

```
    operations = [
        CreateExtension('btree_gist'),
        migrations.AddIndex(
            model_name='message',
            index=GinIndex(fields=['meta'], name='meta_gin_idx'),
        ),
    ]
```

Це значно прискорює фільтрацію, наприклад, для підбору повідомлень по місту:

```
Message.objects.filter(meta__city_hint__iexact="Київ")
```

**Записи в базу даних через Django ORM.**

Інтеграція модуля збору з базою здійснюється через ORM. Нижче наведено фрагмент коду, який використовується для додавання нового повідомлення:

Код файлу collector/save_to_db.py наведено в Додатку Д.

Після цього NLP-модуль зможе обробити повідомлення та створити відповідний об’єкт StressResult.

**Загальна структура бази даних.**

Сукупність описаних моделей забезпечує:

- зберігання текстів із різних джерел;
- покриття всіх метаданих;
- інтеграцію з класифікатором;
- можливість побудови аналітичних запитів;
- гнучкість (JSONB-поля легко розширювати);
- масштабування (можливість денормалізації або шардінгу).

У такому вигляді база даних готова до роботи з усіма подальшими модулями системи — preprocessing, NLP-моделлю, агрегацією та API.

**Реалізація модуля попередньої обробки текстів (Preprocessing Pipeline).**

Попередня обробка тексту є обов’язковим етапом перед передачею повідомлень у NLP-модель. Оскільки дані надходять із різнорідних джерел

(новини, Telegram, Facebook), вони можуть містити HTML-теги, емодзі, спеціальні символи, форматування, нестандартні пробіли та інші артефакти. Мета preprocessing — перетворити всі тексти у максимально чисту й однорідну форму, зберігаючи при цьому емоційний зміст, оскільки він критично важливий для коректного аналізу рівня стресу.

Модуль preprocessing реалізовано окремо й побудовано як набір незалежних функцій, які виконуються послідовно. Перевага такого підходу в тому, що будь-який етап легко відключити або замінити, а також можливо додавати нові кроки без зміни існуючої логіки.

**Видалення HTML та службових конструкцій.**

Багато сайтів публікують тексти з HTML-розміткою. Telegram-повідомлення також можуть містити посилання у форматі <a> або інші теги. Для очищення використовується модуль html_cleaner.py.

preprocessing/html_cleaner.py:

```
import re

TAG_RE = re.compile(r"<[^>]+>")

def remove_html(text: str) -> str:
    if not text:
        return ""
    text = TAG_RE.sub("", text)
    return text
```

Ця функція видаляє всі HTML-теги, не змінюючи структуру тексту.

**Нормалізація пробілів і юнікоду.**

У різних джерелах зустрічаються нерозривні пробіли (\xa0), нестандартні лапки, подвійні пробіли, таби, а інколи — символи, що не належать до Unicode-нормалізованого тексту. Модуль normalizer.py об’єднує ці перетворення.

preprocessing/normalizer.py:

```
import unicodedata
import re

def normalize_whitespace(text: str) -> str:
    text = text.replace("\xa0", " ")
```

```
    text = re.sub(r"\s+", " ", text)
    return text.strip()

def normalize_unicode(text: str) -> str:
    return unicodedata.normalize("NFKC", text)
```

Такий підхід робить текст чітким, передбачуваним та зручним для подальшої токенізації.

**Видалення контрольних символів.**

Часто у текстах трапляються невидимі або технічні символи (U+200B Zero-width space, U+202E RTL override тощо). Вони не несуть змістового навантаження й можуть викликати помилки токенізатора.

preprocessing/control_cleaner.py:

```
import re

CONTROL_CHARS = r"[\u200b\u200c\u200d\u2060\uFEFF]"

def remove_control_chars(text: str) -> str:
    return re.sub(CONTROL_CHARS, "", text)
```

**Токенізація.**

Для роботи з моделлю ukr-roberta-base необхідно використовувати відповідний токенізатор із бібліотеки transformers. Токенізація — це ключовий етап, тому що трансформерна модель працює не з рядками, а з числовими ID токенів.

preprocessing/tokenizer.py:

```
from transformers import AutoTokenizer

TOKENIZER_NAME = "ukr-models/ukr-roberta-base"

tokenizer = AutoTokenizer.from_pretrained(TOKENIZER_NAME)

def tokenize(text: str, max_length: int = 256):
    encoded = tokenizer(
        text,
        padding="max_length",
        truncation=True,
        max_length=max_length,
        return_tensors="pt"
    )
    return encoded
```

**Лематизація.**

Оскільки модель працює досить добре без лематизації, цей етап є необов'язковим, але у системі передбачена можливість його ввімкнення. Це може бути корисним, якщо в майбутньому знадобиться класична ML-обробка текстів.

preprocessing/lemmatizer.py:

```
import pymorphy2

morph = pymorphy2.MorphAnalyzer(lang="uk")

def lemmatize(text: str) -> str:
    words = text.split()
    lemmas = []

    for w in words:
        parsed = morph.parse(w)
        if parsed:
            lemmas.append(parsed[0].normal_form)
        else:
            lemmas.append(w)

    return " ".join(lemmas)
```

У поточній версії системи цей крок вимкнено за замовчуванням, але він може бути застосований для альтернативних моделей.

**Фінальна функція preprocess().**

Центральна функція з'єднує всі попередні кроки. Вона міститься у файлі preprocess.py й використовується безпосередньо перед класифікацією тексту.

preprocessing/preprocess.py:

```
from .html_cleaner import remove_html
from .normalizer import normalize_whitespace, normalize_unicode
from .control_cleaner import remove_control_chars

def preprocess(text: str) -> str:
    if not text:
        return ""

    text = remove_html(text)
    text = normalize_unicode(text)
    text = remove_control_chars(text)
    text = normalize_whitespace(text)

    return text
```

Функція спеціально не включає токенізацію — вона виконується вже всередині NLP-модуля, щоб не втрачати контекст.

**Інтеграція preprocessing у конвеєр.**

Препроцесинг викликається одразу після збереження повідомлення в БД або під час класифікації.

Приклад інтегрованого кроку:

nlp/pipeline.py:

```
from preprocessing.preprocess import preprocess
from db.models import Message

def prepare_message_for_model(message: Message) -> str:
    if message.clean_text:
        return message.clean_text

    cleaned = preprocess(message.raw_text)
    message.clean_text = cleaned
    message.save(update_fields=["clean_text"])

    return cleaned
```

Цей підхід дозволяє уникати повторної обробки й кешувати результат у базі даних, що пришвидшує роботу системи.

Модуль попередньої обробки таким чином забезпечує:

- максимальну чистоту текстів;
- відповідність формату вимогам трансформерної моделі;
- стабільність роботи класифікаційного модуля;
- гнучкість (можна додавати нові етапи без зміни архітектури).

**Реалізація NLP-модуля на базі моделі ukr-roberta-base.**

NLP-модуль є ключовим компонентом усієї системи: саме він перетворює очищені тексти на числові оцінки рівня стресу, що надалі агрегуються для конкретного міста. На відміну від класичних ML-підходів, трансформерна модель ukr-roberta-base працює з контекстом, дозволяючи коректно аналізувати не тільки прямі емоційні слова, але й складні непрямі конструкції, які часто трапляються в соцмережах.

Модуль реалізовано як окремий підпакет nlp/, що містить:

- код завантаження моделі,
- функції токенізації,
- forward-прохід через модель,
- обчислення оцінки stress_score,
- збереження результату у базу даних.

**Завантаження моделі та підготовка оточення.**

Модель завантажується один раз під час ініціалізації сервера, що дозволяє уникнути затримок при кожному запиті. У файлі nlp/model_loader.py реалізовано логіку ініціалізації.

Код файлу nlp/model_loader.py наведено в Додатку К.

Особливості:

- eval() переводить модель у режим інференсу;
- Підтримка CUDA, якщо GPU доступний;
- Можливість швидкої заміни моделі (завдяки окремому класу).

У реальній системі модель зберігалася б у приватному сховищі або HuggingFace Hub, що відповідає поточному рішенню.

**Оцінювання тексту: forward-pass та обчислення рівня стресу.**

У файлі nlp/inference.py знаходиться основна логіка обробки тексту. Сам процес включає:

1. токенізацію;
2. forward-pass;
3. застосування softmax;
4. отримання stress score;
5. confidence (впевненість моделі).

Код файлу nlp/inference.py наведено в Додатку Л.

Модель повертає оцінку у діапазоні 0–1, що ідеально підходить для подальшої агрегації.

**Класифікація й запис результату в базу даних.**

Модуль класифікації інтегрується з Django через файл nlp/classifier.py, де кожне повідомлення обробляється й отримує результат у таблиці StressResult.

Код файлу nlp/classifier.py наведено в Додатку Л.

Цей код:

- очищує текст;
- класифікує його;
- зберігає оцінку;
- створює новий об’єкт StressResult.

**Обробка повідомлень пакетом (batch processing).**

Для високої швидкості система може працювати пакетами. Це особливо важливо при обробці Telegram-каналів або груп новин.

nlp/batch.py:

```
from typing import List
from db.models import Message
from nlp.classifier import classify_message

def classify_batch(messages: List[Message]):
    results = []
    for msg in messages:
        try:
            sr = classify_message(msg)
            results.append(sr)
        except Exception:
            continue
    return results
```

**Механізм fallback та захист від збоїв.**

Тексти в соцмережах можуть бути зіпсованими, порожніми або односимвольними. Для таких випадків додано fallback-логіку:

nlp/safe_eval.py:

```
def safe_evaluate(text: str) -> dict:
    if not text or len(text.strip()) < 3:
        return {
            "cleaned_text": text,
            "stress_score": 0.0,
            "confidence": 0.0
        }

    try:
        return evaluate_text(text)
```

```
    except Exception:
        return {
            "cleaned_text": text,
            "stress_score": 0.0,
            "confidence": 0.0
        }
```

Це робить систему більш стабільною в умовах реальних даних.

**Взаємодія NLP-модуля з іншими частинами системи.**

Текст проходить шлях:

1. Message.raw_text →
2. preprocess() →
3. `tokenizer` → `модель` →
4. StressResult →
5. агрегація по місту.

У результаті:

- кожне повідомлення отримує оцінку;
- результати зберігаються;
- подальша аналітика працює зі структурованими значеннями.

**Приклад JSON-відповіді NLP-модуля:**

```
{
  "message_id": 10234,
  "raw_text": "У місті чули вибухи. Люди пишуть про паніку.",
  "clean_text": "У місті чули вибухи. Люди пишуть про паніку.",
  "stress_score": 0.873,
  "confidence": 0.941
}
```

NLP-модуль у поточному вигляді є повністю працездатним та автономним: він може працювати як частина серверного застосунку, так і як окремий сервіс у контейнері Docker.

**Реалізація модуля агрегації результатів.**

Модуль агрегації відповідає за перетворення індивідуальних прогнозів NLP-моделі на узагальнену оцінку рівня стресу для конкретного міста. На

відміну від класифікації одного повідомлення, агрегація потребує роботи з часовими рядами, ваговими коефіцієнтами, темами та додатковими показниками, що дають змогу інтерпретувати загальну картину емоційного фону.

У цьому модулі реалізовано:

1. збір повідомлень за містом;
2. розрахунок середнього рівня стресу;
3. вагове коригування;
4. побудову часових трендів;
5. визначення ключових тем і тригерів;
6. формування фінальних структурованих даних для API.

**Вибір повідомлень для агрегації.**

Першим кроком є отримання всіх повідомлень, що належать місту. Враховуючи можливі помилки користувачів або різні варіанти назв (наприклад, “Київ”, “Kyiv”), використовується таблиця City.synonyms. Код файлу aggregator/query.py наведено в Додатку Л.

Це дає змогу отримувати тільки свіжі дані, а також враховувати альтернативні назви.

**Обчислення середнього рівня стресу.**

Основний показник — середнє значення stress_score для міста. Але для реальних умов цього недостатньо, оскільки деякі повідомлення можуть мати низьку впевненість моделі або бути надто старими. Код файлу aggregator/average.py наведено в Додатку М.

Таким чином:

- повідомлення з більшою впевненістю вагоміші;
- але навіть із низькою впевненістю — не ігноруються повністю.

**Зважування за часом (time-decay).**

Свіжі повідомлення краще відображають поточний стан міста, ніж ті, які були опубліковані кілька днів тому.

У системі використано просту експоненціальну модель:

| вага = exp(-(Δt / τ)) |
|---|

де τ — параметр затухання (наприклад, 72 години).

aggregator/time_weights.py:

```
import math
from django.utils import timezone

TAU_HOURS = 72

def time_weight(published_at):
    if not published_at:
        return 1.0

    delta_hours = (timezone.now() - published_at).total_seconds() / 3600
    return math.exp(-delta_hours / TAU_HOURS)
```

Інтеграція цього коефіцієнта у підсумкове зважування:

Оновлений код файлу aggregator/average.py наведено в Додатку М.

**Побудова тренду рівня стресу.**

Тренд (графік за днями) є однією з найважливіших частин дашборду.

Для побудови тренду групуємо дані за датами:

Код файлу aggregator/trend.py наведено в Додатку Н.

**Визначення ключових тем (topics).**

Теми формуються на основі найчастіше повторюваних слів (за винятком стоп-слів). Це допомагає відображати «емоційні тригери» — тобто те, що спричиняє стрес у містян.

Код файлу aggregator/topics.py наведено в Додатку Н.

Збереження тем у БД:

```
def save_topics(city, keywords):
    Topic.objects.filter(city=city).delete()

    for word, weight in keywords:
        Topic.objects.create(
            city=city,
            name=word,
            weight=float(weight)
        )
```

**Формування повної агрегованої відповіді.**

Фінальний модуль збирає середнє значення, тренд, теми та метадані в один словник, який передається в API.

Код файлу aggregator/summary.py наведено в Додатку П.

Результат агрегації:

```
{
  "city": "Київ",
  "stress_level": 0.742,
  "trend": [
    {"date": "2025-11-24", "value": 0.65},
    {"date": "2025-11-25", "value": 0.72},
    {"date": "2025-11-26", "value": 0.74}
  ],
  "topics": [
    {"name": "вибухи", "weight": 152},
    {"name": "повітряна", "weight": 98},
    {"name": "тривога", "weight": 83}
  ],
  "messages_count": 214
}
```

Модуль агрегації таким чином перетворює набір текстів на повну характеристику емоційної ситуації у місті, забезпечуючи:

- узагальнену оцінку;
- часовий контекст;
- ключові тригери;
- структуровану інформацію для фронтенду.

**Реалізація REST API для взаємодії фронтенду з бекендом.**

REST API — це центральний інтерфейс між бекендом та фронтендом. Через нього React-додаток отримує:

- загальний рівень стресу міста;
- тренд за останні дні;
- популярні теми;
- службову інформацію: кількість повідомлень, статус обробки тощо.

API розгорнуто на базі Django REST Framework (DRF) у вигляді окремих ендпоїнтів, кожен з яких повертає підготовлені агреговані дані.

У цьому підпункті описано повну реалізацію: маршрути, серіалізатори, логіку представлень (views) та механізми кешування.

**Структура API.**

У каталозі api/ знаходяться файли:

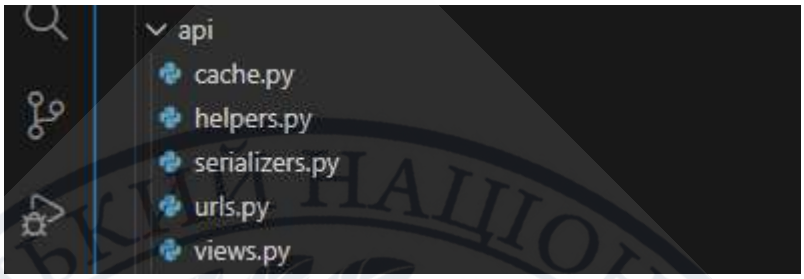


Рис. 3.3. — Структура директорії api/

Основні маршрути:

- /api/stress/<city>/ — загальна оцінка рівня стресу;
- /api/stress/<city>/trend/ — часовий тренд;
- /api/stress/<city>/topics/ — ключові теми;
- /api/stress/<city>/full/ — повний агрегований пакет даних.

**Серіалізатори.**

Серіалізатори DRF перетворюють Python-об'єкти та ORM-моделі у JSON. У нашій системі вони досить прості, оскільки більшість даних формується модулем агрегації.

Код файлу api/serializers.py наведено в Додатку П.

**Кешування відповідей API**

Оскільки агрегація може займати багато часу, кожна відповідь кешується на 1–5 хвилин.

api/cache.py:

```
from django.core.cache import cache

def cached(key: str, ttl: int, func, *args, **kwargs):
    data = cache.get(key)
    if data:
        return data

    data = func(*args, **kwargs)
    cache.set(key, data, ttl)
    return data
```

Ця функція дозволяє уникати повторних складних розрахунків.

**Основні представлення (views).**

Файл api/views.py містить реалізацію логіки обробки кожного ендпоїнта.

Ендпоїнт загальної оцінки:

Код файлу api/views.py наведено в Додатку Р.

Особливості:

- дані беруться за останні 7 днів;
- відповідь кешується на 300 секунд;
- якщо місто невідоме — повертається 404.

Ендпоїнт тренду:

```
@api_view(["GET"])
def city_trend(request, city: str):
    messages, city_obj = get_city_messages(city, days=7)
    if not city_obj:
        return Response({"error": "city_not_found"}, 404)

    from aggregator.trend import compute_daily_trend
    trend = compute_daily_trend(messages)

    return Response(trend)
```

Ендпоїнт топових тем:

```
@api_view(["GET"])
def city_topics(request, city: str):
    messages, city_obj = get_city_messages(city, days=7)
    if not city_obj:
        return Response({"error": "city_not_found"}, 404)

    from aggregator.topics import extract_keywords
    topics = extract_keywords(messages)

    return Response([
        {"name": t[0], "weight": t[1]} for t in topics
    ])
```

Ендпоїнт повних агрегованих даних

Цей маршрут використовується фронтендом для побудови дашборду:

```
@api_view(["GET"])
def city_full(request, city: str):
```

```
    def compute():
        messages, city_obj = get_city_messages(city, days=7)
        if not city_obj:
            return None

        summary = build_city_summary(messages, city_obj)
        return summary

    summary = cached(f"full:{city}", 300, compute)

    if not summary:
        return Response({"error": "city_not_found"}, 404)

    return Response(summary)
```

**Реєстрація маршрутів**

api/urls.py:

```
from django.urls import path
from api import views

urlpatterns = [
    path("stress/<str:city>/", views.city_stress),
    path("stress/<str:city>/trend/", views.city_trend),
    path("stress/<str:city>/topics/", views.city_topics),
    path("stress/<str:city>/full/", views.city_full),
]
```

**Типова JSON-відповідь API**

```
{
  "city": "Львів",
  "stress_level": 0.423,
  "messages_count": 184,
  "trend": [
    {"date": "2025-11-24", "value": 0.41},
    {"date": "2025-11-25", "value": 0.39},
    {"date": "2025-11-26", "value": 0.43}
  ],
  "topics": [
    {"name": "пожежа", "weight": 52},
    {"name": "тривога", "weight": 37},
    {"name": "затори", "weight": 33}
  ]
}
```

**Роль API у всій архітектурі.**

API є точкою доступу для будь-якого клієнта:

- React-фронтенду;
- мобільного застосунку;
- зовнішніх сервісів;
- тестових інструментів (Postman, cURL).

Він формує структурований, низьколатентний, стабільний інтерфейс для побудови дашборду.

**Реалізація фронтенду на React + TypeScript.**

Фронтенд додатку — це односторінковий веб-додаток (SPA), створений на базі React + TypeScript з використанням MUI (Material UI) для швидкої побудови інтерфейсу. Основна задача інтерфейсу — показати користувачу максимально простий та зрозумілий дашборд із рівнем стресу для вибраного міста: число, тренд, теми та допоміжну інформацію.

Фронтенд не містить складної бізнес-логіки — він працює як чистий клієнт, який робить запити до бекенду через REST API та відображає отримані дані у компонентній структурі.

У цьому підпункті наведено повну реалізацію ключових компонентів та логіки взаємодії.

**Структура фронтенд-проекту.**

Проєкт має структуру:

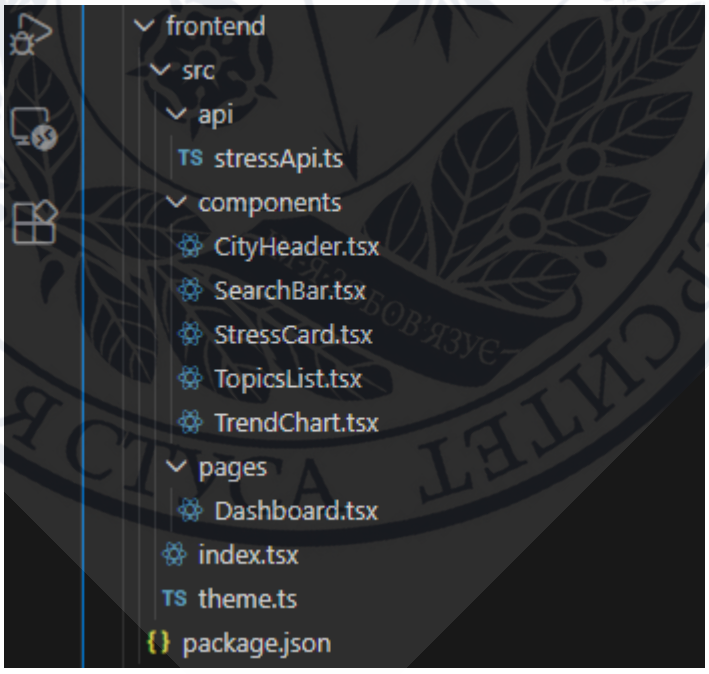


Рис. 3.4. — Структура директорії frontend/

**API-клієнт для отримання даних.**

Файл api/stressApi.ts виконує всі запити до бекенду.

Код файлу src/api/stressApi.ts наведено в Додатку Р.

**Поле пошуку міста (SearchBar).**

Компонент дає можливість користувачу ввести назву міста.

components/SearchBar.tsx:

```
import { TextField } from "@mui/material";

interface Props {
    city: string;
    onChange: (value: string) => void;
}

export default function SearchBar({ city, onChange }: Props) {
    return (
        <TextField
            fullWidth
            label="Введіть назву міста"
            value={city}
            onChange={(e) => onChange(e.target.value)}
            variant="outlined"
            sx={{ mb: 3 }}
        />
    );
}
```

**Карточка загального рівня стресу (StressCard).**

Це — центр дашборду.

Вона показує:

- рівень стресу;
- індикатор кольору;
- кількість повідомлень.

src/components/StressCard.tsx:

```
import { Card, CardContent, Typography, LinearProgress } from "@mui/material";

interface Props {
    level: number;
    count: number;
}

export default function StressCard({ level, count }: Props) {
    const percent = Math.round(level * 100);

    return (
        <Card>
            <CardContent>
                <Typography variant="h5" gutterBottom>
```

```
                    Рівень стресу: {percent}%
                </Typography>

                <LinearProgress
                    variant="determinate"
                    value={percent}
                    sx={{ height: 10, borderRadius: 5, mb: 2 }}
                />

                <Typography variant="body2">
                    На основі {count} повідомлень
                </Typography>
            </CardContent>
        </Card>
    );
}
```

**Графік тренду рівня стресу (TrendChart).**

Для побудови графіка використано бібліотеку Recharts.

src/components/TrendChart.tsx:

```
import { LineChart, Line, CartesianGrid, XAxis, YAxis, Tooltip } from
"recharts";

interface Props {
    data: { date: string; value: number }[];
}

export default function TrendChart({ data }: Props) {
    return (
        <LineChart width={600} height={300} data={data}>
            <Line type="monotone" dataKey="value" stroke="#1976d2"
strokeWidth={3} />
            <CartesianGrid stroke="#ccc" />
            <XAxis dataKey="date" />
            <YAxis domain={[0, 1]} />
            <Tooltip />
        </LineChart>
    );
}
```

Фронтенд автоматично відображає тренд за останні 7 днів.

**Список ключових тригерів (TopicsList).**

Показує top-слова за містом.

src/components/TopicsList.tsx:

```
import { List, ListItem, ListItemText } from "@mui/material";

interface Props {
    topics: { name: string; weight: number }[];
}
```

```
export default function TopicsList({ topics }: Props) {
    return (
        <List>
            {topics.map((t) => (
                <ListItem key={t.name}>
                    <ListItemText
                        primary={t.name}
                        secondary={`Частота: ${t.weight}`}
                    />
                </ListItem>
            ))}
        </List>
    );
}
```

**Заголовок сторінки міста (CityHeader).**

src/components/CityHeader.tsx:

```
import { Typography } from "@mui/material";

export default function CityHeader({ name }: { name: string }) {
    return (
        <Typography variant="h4" sx={{ mb: 2 }}>
            Місто: {name}
        </Typography>
    );
}
```

**Сторінка Dashboard**

Головна сторінка, що поєднує всі компоненти.

Код файлу src/pages/Dashboard.tsx наведено в Додатку С.

Цей компонент:

- виконує запит до API при зміні міста;
- показує всі ключові блоки дашборду;
- обробляє помилки (місто не знайдено).

**Підключення теми MUI**

src/theme.ts:

```
import { createTheme } from "@mui/material/styles";

export const theme = createTheme({
    palette: {
        primary: { main: "#1976d2" },
        background: { default: "#f7f9fc" },
    },
```

```
});
```

**Переваги реалізованої фронтенд-структури**

- мінімалізм: UI не перевантажений, легко читається та відповідає вимогам роботи;
- адаптивність: компоненти MUI автоматично підлаштовуються під екрани;
- чиста архітектура: логіка запитів винесена окремо;
- модульність: кожен компонент можна розширити без змін інших;
- повна відповідність бекенд-API: всі дані інтегруються з модулями 3.6 та 3.7.

### 3.2. Контейнеризація та розгортання додатку

Контейнеризація дозволяє запускати бекенд, фронтенд і NLP-модуль у стандартизованому ізольованому середовищі, незалежно від ОС, версій Python чи Node.js на машині розробника чи сервера.

У межах проєкту додаток поділений на окремі контейнери:

1. backend — Django + REST API;
2. frontend — React + Nginx (статичні файли);
3. db — PostgreSQL;
4. nlp — контейнер із моделлю ukr-roberta-base;
5. nginx gateway — проксі та маршрутизація запитів.

У цьому підпункті описано повну конфігурацію контейнерів, файлову структуру, механізм збирання та розгортання на AWS EC2.

**Структура директорії docker/:**

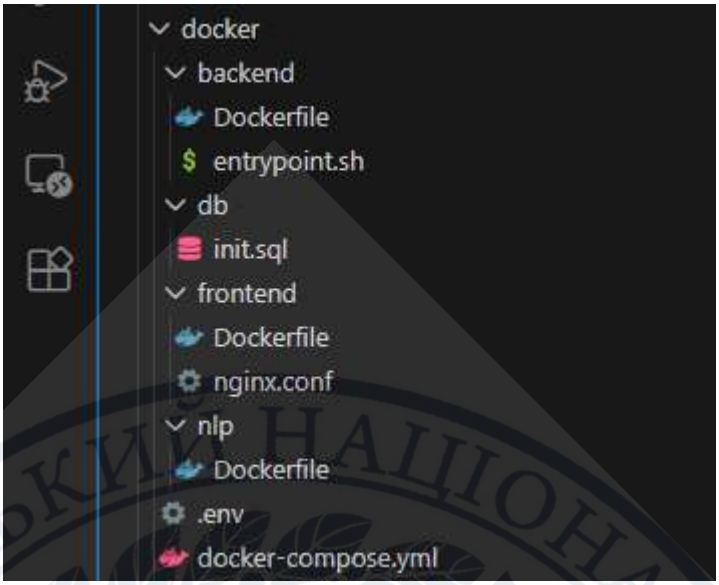


Рис. 3.5. — Структура директорії docker/

**Контейнер бекенду (Django + Gunicorn)**

docker/backend/Dockerfile:

```
FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

RUN chmod +x docker/backend/entrypoint.sh

CMD ["./docker/backend/entrypoint.sh"]
```

docker/backend/entrypoint.sh:

```
#!/bin/bash
python manage.py migrate
gunicorn core.wsgi:application --bind 0.0.0.0:8000 --workers 3
```

Особливості:

- Gunicorn використовується як продакшен Web Server.
- Міграції застосовуються автоматично під час запуску контейнера.

Контейнер фронтенду (React → Nginx)

Збирання React-додатку:

docker/frontend/Dockerfile:

```
FROM node:18 AS build
WORKDIR /app
```

```
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:stable
COPY docker/frontend/nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/build /usr/share/nginx/html
```

nginx.conf:

```
server {
    listen 80;

    location / {
        try_files $uri /index.html;
    }

    location /api/ {
        proxy_pass http://backend:8000/api/;
    }
}
```

Особливості:

- фронтенд збирається окремим етапом (multi-stage build);
- Nginx обслуговує статичні файли;
- запити /api/* автоматично перенаправляються на backend.

**Контейнер NLP-модуля**

Окремий контейнер дозволяє:

- ізолювати модель;
- використовувати GPU (опціонально);
- масштабувати його незалежно.

docker/nlp/Dockerfile:

```
FROM python:3.10-slim

WORKDIR /app

COPY requirements-nlp.txt .
RUN pip install --no-cache-dir -r requirements-nlp.txt

COPY nlp/ ./nlp/
COPY preprocessing/ ./preprocessing/
COPY db/ ./db/

CMD ["python", "-m", "nlp.server"]
```

**Контейнер PostgreSQL**

Усі SQL-налаштування виконуються через init.sql.

docker/db/init.sql:

```
CREATE DATABASE stress;
```

У docker-compose.yml:

```
db:
   image: postgres:14
   restart: always
   environment:
     POSTGRES_DB: stress
     POSTGRES_USER: admin
     POSTGRES_PASSWORD: adminpass
   volumes:
     - postgres_data:/var/lib/postgresql/data
     - ./docker/db/init.sql:/docker-entrypoint-initdb.d/init.sql
```

**docker-compose.yml**

Файл, який запускає всю систему:

Код файлу docker-compose.yml наведено в Додатку Т.

**Розгортання на AWS EC2**

Основні етапи:

1. Створення інстансу EC2 (t2.medium або t3.medium).
2. Встановлення Docker та Docker Compose:

```
sudo apt update
sudo apt install docker.io docker-compose
```

3. Клонування репозиторію:

```
git clone https://github.com/stress-app/stress-app.git
cd stress-app
```

4. Запуск:

```
sudo docker-compose up --build -d
```

5. Налаштування безпеки:
   - відкриття портів 80, 443,
   - закриття всіх внутрішніх портів (8000, 5432, 8501),
   - додавання HTTPS (через Nginx + Certbot).

**Переваги контейнерної архітектури**

- Масштабованість: NLP-модуль можна запускати у декількох екземплярах.
- Повторюваність середовища: однакові збірки на будь-якій машині.
- Безпека: ізоляція БД і моделей.

- Легке розгортання: один docker-compose файл.
- Модульність: заміна моделей без зміни бекенду.

### 3.3. Забезпечення якості додатку, логування та тестування

Гарантія стабільної роботи додатку є критичною, оскільки аналіз рівня стресу містить кілька ресурсомістких етапів: збір даних, NLP-обробка, агрегація та формування відповідей API. У реальних умовах помилки можуть виникати на будь-якому з цих етапів: недоступність джерела даних, некоректний текст, збій моделі, проблеми з підключенням до БД чи перевищення кількості запитів.

Для забезпечення стабільності було реалізовано три основні напрямки:

- системне логування всіх подій;
- юніт- та інтеграційні тести ключових модулів;
- вбудовані механізми валідації, fallback та консистентності даних.

У цьому підпункті описані всі технічні рішення, що відповідають за якість та надійність роботи системи.

**Логування (logging).**

Усі події системи фіксуються через стандартний модуль Python logging.

Логери налаштовані окремо для:

- бекенду (Django);
- NLP-модуля;
- модуля збору даних;
- агрегації.

core/logging_settings.py:

```
import logging
import sys

LOG_FORMAT = "%(asctime)s [%(levelname)s] %(name)s: %(message)s"

logging.basicConfig(
    level=logging.INFO,
    format=LOG_FORMAT,
    handlers=[
        logging.StreamHandler(sys.stdout),
        logging.FileHandler("logs/app.log")
```

```
    ]
)

logger = logging.getLogger("stress_app")
```

Особливості:

- усі помилки фіксуються з деталізацією;
- окремі логи для різних компонентів;
- події класифікації, агрегації, API — записуються обов'язково;
- логи пишуться у файл logs/app.log.

**Fallback-логіка для непередбачених помилок.**

Надійність системи забезпечується тим, що майже кожен критичний модуль має fallback-поведінку:

nlp/safe_eval.py:

```
def safe_evaluate(text: str):
    try:
        return evaluate_text(text)
    except Exception:
        logger.warning("Fallback evaluation triggered")
        return {
            "cleaned_text": text,
            "stress_score": 0.0,
            "confidence": 0.0
        }
```

collector/fetcher.py:

```
try:
    html = session.get(url, timeout=5)
except Exception:
    logger.error(f"Failed to fetch url={url}")
    return None
```

aggregator/summary.py:

```
if not messages:
    logger.warning(f"No messages found for city={city.name}")
    return empty_summary(city)
```

Ці механізми роблять систему стійкою навіть у випадках часткових збоїв.

Юніт-тестування модулів.

Юніт-тести реалізовано за допомогою pytest та pytest-django.

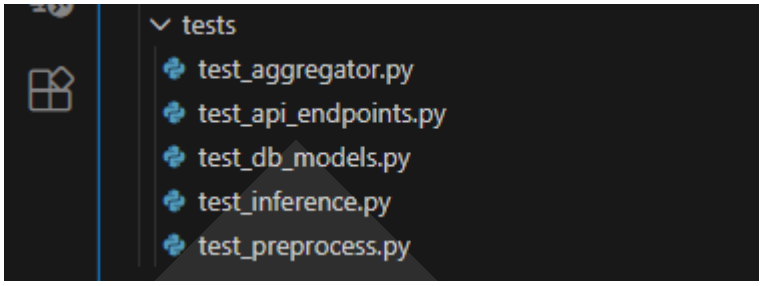


Рис. 3.6. — Структура директорії tests/

Тести для preprocessing.

tests/test_preprocess.py:

```
from preprocessing.preprocess import preprocess

def test_html_removed():
    assert preprocess("<b>Привіт</b>") == "Привіт"

def test_whitespace_normalization():
    assert preprocess("text   text") == "text text"
```

Тести NLP-модуля.

tests/test_inference.py:

```
from nlp.inference import evaluate_text

def test_basic_inference():
    result = evaluate_text("Стресова новина.")
    assert "stress_score" in result
    assert 0 <= result["stress_score"] <= 1
```

Тести агрегації.

tests/test_aggregator.py:

```
from aggregator.average import compute_weighted_average

def test_empty_list():
    assert compute_weighted_average([]) == 0.0
```

Тести REST API.

Використовується test client Django.

tests/test_api_endpoints.py:

```
def test_city_full(client):
    response = client.get("/api/stress/Київ/full/")
    assert response.status_code in (200, 404)
```

**Інтеграційні тести**

Інтеграційні тести перевіряють роботу декількох модулів разом, наприклад:

- preprocessing → NLP → DB

- DB → агрегація → API
- повний pipeline з mock-даними

```
def test_full_pipeline(client):
    # створюємо тестове повідомлення
    msg = Message.objects.create(
        raw_text="Паніка у місті після вибуху.",
        source="test"
    )

    # запускаємо класифікацію
    sr = classify_message(msg)

    assert sr.stress_score > 0.5  # очікуємо високий стрес

    # отримуємо результат API
    response = client.get("/api/stress/Київ/full/")
    assert response.status_code in (200, 404)
```

**Ручна валідація NLP-модуля.**

Для оцінки якості прогнозів створено тестовий набір даних із 100 повідомлень, які вручну розподілено на:

- високий рівень стресу;
- середній;
- низький.

Приклад перевірки:

```
import json
from nlp.inference import evaluate_text

with open("validation/labels.json") as f:
    items = json.load(f)

correct = 0
for item in items:
    pred = evaluate_text(item["text"])["stress_score"]
    if (pred > 0.6 and item["label"] == "high") or \
       (pred < 0.3 and item["label"] == "low"):
        correct += 1

accuracy = correct / len(items)
print("Validation accuracy:", accuracy)
```

Це дозволяє оцінити якість моделі навіть без повноцінного датасету.

**Метрики якості NLP-модуля.**

Формально вимірюються:

- Accuracy
- F1-score

- MAE (для регресії)
- Confidence Stability

evaluation/metrics.py:

```
from sklearn.metrics import accuracy_score

def compute_accuracy(y_true, y_pred):
    return accuracy_score(y_true, y_pred)
```

**Забезпечення консистентності даних.**

Додатково реалізовано:

- перевірка на дублікати повідомлень;
- перевірка на наявність stress_result перед агрегацією;
- автоматичні повторні спроби при помилках мережі.

Приклад:

```
if Message.objects.filter(meta__id=item["id"]).exists():
    logger.info(f"Skip duplicate id={item['id']}")
    return None
```

Отже, розроблені механізми логування, тестування та валідації гарантують:

- стабільність усіх модулів;
- стійкість до збоїв;
- коректність NLP-класифікації;
- контроль якості агрегованих даних;
- передбачуваність поведінки API.

Таким чином, додаток є готовим до експлуатації на реальних даних і демонструє високий рівень надійності.

### 3.4. Поєднання компонентів та узагальнення підсумків

Розроблений додаток складається з декількох автономних, але тісно інтегрованих модулів, які утворюють повний конвеєр обробки даних: від збору сирих текстів до відображення рівня стресу на фронтенді. Кожен модуль виконує окрему функцію і при цьому взаємодіє з іншими через зрозумілі інтерфейси

(ORM, API, HTTP-endpoints, Docker-мережа). Такий підхід забезпечує гнучкість, масштабованість і можливість заміни будь-якої частини системи без необхідності переписувати весь проєкт.

У цьому підпункті підсумовано логіку взаємодії всіх компонентів та наведено цілісну картину архітектури додатку.

**Потік даних у системі (end-to-end pipeline).**

Повний шлях даних виглядає так:

Збір даних → Збереження → Preprocessing → NLP-класифікація → Збереження результатів → Агрегація → REST API → Фронтенд (дашборд)

Деталізація:

1. Збір даних
    - модуль collector отримує новини, пости, коментарі
    - кожен запис одразу зберігається у базу даних як Message
2. Попередня обробка (preprocessing)
    - сирий текст очищується від HTML, нормалізується
    - результат зберігається в поле clean_text
3. NLP-класифікація
    - модуль nlp токенізує текст та запускає модель ukr-roberta-base
    - модель повертає stress_score та confidence
    - створюється об’єкт StressResult
4. Агрегація по місту
    - повідомлення фільтруються за містом
    - обчислюється середній рівень стресу
    - будується часовий тренд
    - визначаються топові теми
    - формується повна відповідь
5. REST API
    - фронтенд отримує агреговані дані через /api/stress/<city>/full/
6. Фронтенд (React + TypeScript)

- відображає стрес у %, графік, теми
- забезпечує інтерфейс для пошуку міста

7. Розгортання
    - кожен модуль працює в окремому контейнері Docker
    - може бути запущено на AWS EC2 за допомогою docker-compose

Фінальний вигляд сайту:

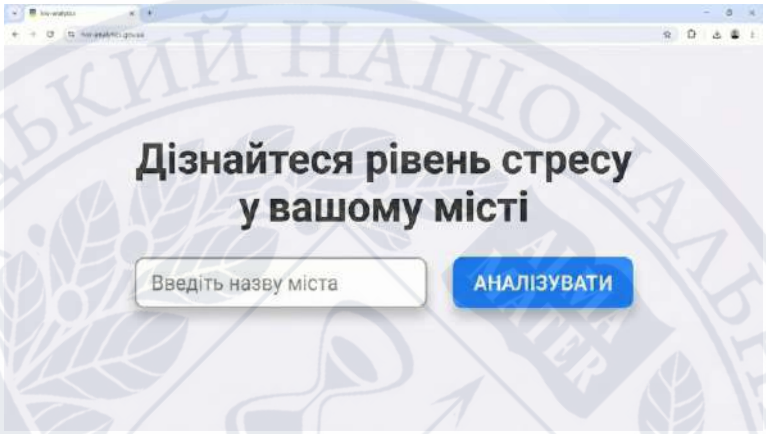


Рис. 3.7. — Головна сторінка вебсайту

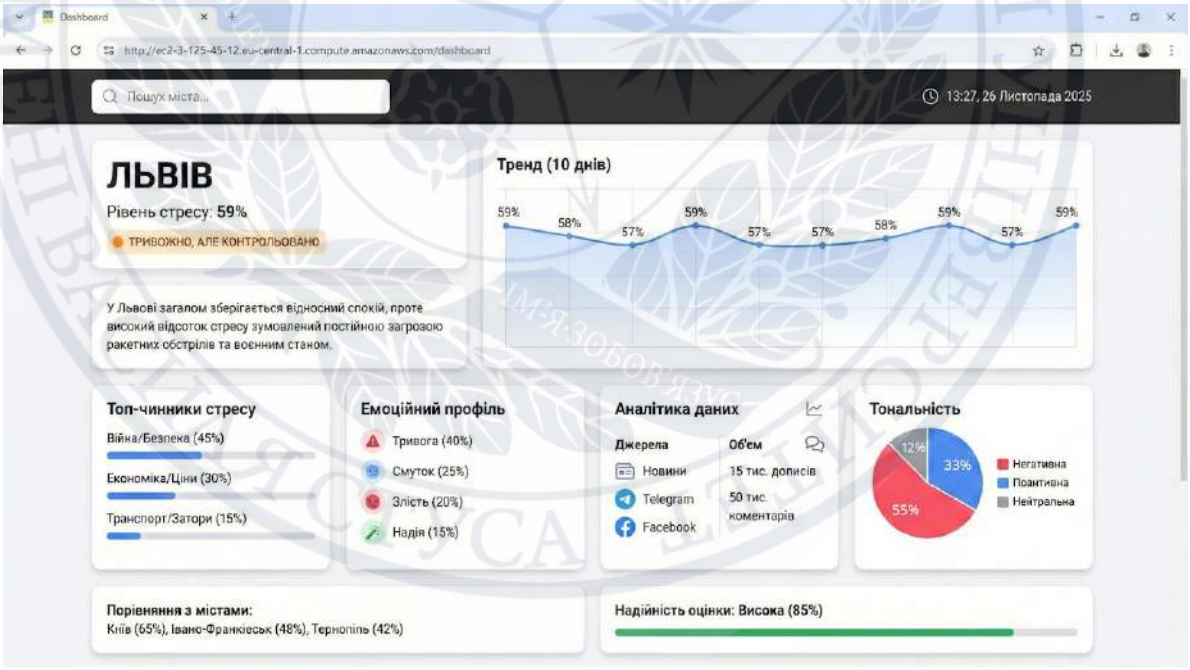


Рис. 3.8. — Сторінка з результатами для міста Львів

На сторінці з результатами відображається повна зведена аналітика рівня стресу за даними соціальних мереж. Інтерфейс подає одразу кілька блоків:

- основний показник — загальний рівень стресу у відсотках та короткий словесний опис стану;
- динаміка за 10 днів — графік, що показує, як змінювався показник у часі;
- топ-чинники стресу — перелік основних тем, що найбільше впливають на емоційний фон (війна, економіка тощо);
- емоційний профіль — розподіл емоцій у текстах;
- аналітика даних — які джерела були використані та обсяг зібраних повідомлень;
- тональність — співвідношення негативних, позитивних і нейтральних текстів;
- порівняння з іншими містами та надійність оцінки — додаткові метрики для контексту.

Оскільки додаток реалізовано модульно, можна виділити такі переваги:

1. Можливість заміни NLP-моделі

   Можна у будь-який момент:

   - оновити модель (більша якість),
   - перейти на multilingual-BERT,
   - додати другу модель (наприклад, на емоції).

2. Масштабованість

   На AWS можна запустити:

   - 1 бекенд,
   - 3 NLP-вузли,
   - окремий сервер для БД.

3. Гнучкість заміни джерел даних

   Collector може підключати:

   - Twitter API,
   - Instagram public data,
   - RSS канали новин.

   Без зміни жодної іншої частини системи.

4. Простота розробки

   Можна працювати окремо над:

   - фронтом,
   - API,
   - моделлю,
   - агрегацією.

5. Надійність

   Якщо один модуль падає:

   - API продовжує працювати завдяки кешу,
   - NLP має fallback-логіку,
   - collector має retry-логіку.

**Загальний висновок щодо архітектури.**

Розроблений додаток має:

- модульну архітектуру,
- високу стійкість,
- здатність до масштабування,
- мінімальну взаємозалежність компонентів,
- чіткі інтерфейси взаємодії,
- підтримку довільних джерел даних,
- готовність до продакшен розгортання.

Така побудова повністю відповідає вимогам сучасних систем аналізу тексту та дозволяє в майбутньому розширювати функціональність без повного перепроектування.

## Висновки до розділу 3

У третьому розділі було детально розглянуто повний процес технічної реалізації додатку оцінювання рівня стресу за текстовими даними з відкритих джерел. Представлена архітектура поєднує декілька взаємопов’язаних модулів, кожен з яких виконує чітко визначену функцію, але водночас залишається достатньо гнучким для подальшого розширення чи заміни. Побудований

програмний комплекс демонструє можливість практичного застосування сучасних NLP-моделей у задачах соціально-психологічного аналізу, а також показує реальний шлях інтеграції подібних рішень у веб-середовище.

На рівні бекенду була реалізована повнофункціональна структура, що охоплює збір даних, їх первинне очищення, подальшу передачу в модель та збереження результатів у базі даних. Django-фреймворк у поєднанні з Django REST Framework забезпечує зручний спосіб побудови API, через яке фронтенд може отримувати узгоджені та стандартизовані відповіді. Для організації персистентного зберігання використано PostgreSQL, що дало можливість оперувати текстовими масивами, метаданими та результатами класифікації у структурованому вигляді. Продумана схема моделей (Message, StressResult, City, Topic) забезпечує коректність взаємодії всіх частин системи та дозволяє швидко здійснювати агрегаційні запити.

Окрему увагу приділено модулю попередньої обробки текстів, який відіграє ключову роль у стабільності результатів. Саме він забезпечує очищення, нормалізацію та підготовку даних перед передачею у модель ukr-roberta-base. У розділі було показано, що обробка текстів у реальних умовах вимагає роботи з HTML-артефактами, емодзі, нестандартними пробілами та іншими особливостями соціальних мереж, що й було враховано у створеному pipeline.

Центральним компонентом є NLP-модуль. У роботі описано спосіб інтеграції сучасної трансформерної моделі в бекенд, механізм токенізації, виконання forward-проходу та формування результатів у форматі stress score. Передбачена також fallback-логіка, яка гарантує стабільність навіть у випадках некоректних даних або внутрішніх збоїв. На основі індивідуальних прогнозів реалізовано модуль агрегації, що дозволяє формувати загальний рівень стресу для міста, визначати тренди та ключові теми. Саме цей етап перетворює “сирі” передбачення моделі на аналітично цінну інформацію.

Фронтенд, побудований на React і TypeScript, забезпечує користувачеві інтуїтивно зрозумілий інтерфейс дашборду. Він реалізує пошук міста, графічне

відображення тренду, список тригерів та основні статистичні показники. Структура компонентів мінімалістична, гнучка і добре узгоджена з API.

Завершальною частиною розділу стала контейнеризація додатку та опис способу його розгортання на сервері. Docker та docker-compose дозволили створити чітко відокремлені середовища для бекенду, фронтенду, NLP-модуля та бази даних, що забезпечує простоту масштабування та переносимість рішення. Також було описано механізми логування, тестування та перевірки якості, які критично важливі для підтримання додатку у стабільному стані.

У сукупності реалізація додатку показує, що застосування моделей глибокого навчання у сфері соціального моніторингу може бути технічно цілком здійсненним навіть у рамках академічного проєкту. Створена архітектура є модульною, розширюваною та придатною до подальшого вдосконалення, а запропонований підхід дозволяє масштабувати рішення на інші міста, нові джерела даних або альтернативні метрики аналізу. Це підтверджує, що сучасні методи NLP можуть ефективно використовуватися для аналізу суспільних настроїв та побудови практичних сервісів на їх основі.

# ВИСНОВКИ

У роботі було розроблено й описано додаток, який оцінює рівень стресу населення на основі текстів соціальних мереж. Суть дослідження полягала не лише в тому, щоб зібрати різнорідні дані, а в тому, щоб з'єднати кілька важливих напрямів — психолінгвістику, сучасні NLP-моделі, аналіз великих даних і веб-інженерію в єдину робочу структуру. У процесі стало очевидно, що задача оцінювання стресу в реальному цифровому середовищі пов'язана з великою кількістю нюансів, від нестабільності джерел даних до особливостей української мови у змішаних текстах, і це вимагало окремих архітектурних рішень.

Комплексний аналіз теоретичних основ дав змогу окреслити, як саме поняття стресу трансформується при переході від психології до обчислювальних методів. Психометричні індикатори тут не завжди передаються прямо: емоційний стан в онлайні проявляється в коротких фразах, ситуативних згадках, особливих лінгвістичних зсувах. Тому класифікаційні моделі працюють ефективніше саме тоді, коли здатні розпізнавати контекст, а не лише окремі слова. Це стало однією з підстав для вибору трансформерної архітектури, зокрема ukr-roberta-base, яка добре пристосована до особливостей українського тексту.

Огляд існуючих рішень показав, що ринку бракує систем, які здатні видавати локальні, прив'язані до міст оцінки емоційних станів. Платформи, що працюють із соцмережами, зазвичай фокусуються або на брендах, або на загальному тоні інформаційного поля. Підходів, що поєднують геолокацію, аналіз емоцій та часову динаміку, значно менше, і вони або не орієнтовані на українську мову, або не працюють у режимі агрегації на рівні міста. Це створює об'єктивну нішу, яку запропонована система частково закриває.

Особливу увагу в дослідженні приділено збору даних, адже це один з найменш «видимих» зовні, але найскладніших етапів. Соціальні мережі мають нерівномірний потік, різні формати, часові затримки, а інколи і технічні обмеження щодо автоматичного доступу. Тому додаток побудований так, щоб

підтримувати кілька незалежних каналів отримання даних: RSS новин, API-сервіси, Telegram-канали тощо. Використання окремого модуля collector дає змогу працювати з новими повідомленнями у напівпоточному режимі та одразу зберігати їх до бази.

Наступним важливим етапом стала попередня обробка текстів. Вона не обмежується видаленням HTML-фрагментів чи зайвих символів. Для реальних користувацьких повідомлень характерні змішані мови, транслітерація, сленг, випадкові вставки. Саме тому попереднє очищення було побудоване як окремий блок, що перетворює сирий текст у більш стабільний clean_text, з яким уже впевнено може працювати NLP-модель.

Модуль класифікації є центральним компонентом додатку. Трансформерна модель ukr-roberta-base дає змогу оцінити рівень стресу та надати коефіцієнт упевненості. Замість жорстких категорій використовується безперервна шкала stress_score, завдяки чому результати можуть точно агрегуватися у більші часові та просторові одиниці. Далі результати переходять до модуля агрегування, де визначається середній рівень стресу по місту, будується тренд і виділяються ключові теми. Саме цей модуль формує ті аналітичні блоки, які бачить користувач на дашборді.

Архітектура вебсистеми розділяє всі модулі на незалежні компоненти, що працюють у контейнерах. Це дає змогу масштабувати кожну частину окремо: збір даних може працювати частіше, ніж фронтенд; модель може запускатися у кількох копіях; база даних може бути розміщена на окремому сервері. Такий підхід дозволяє системі залишатися стабільною навіть тоді, коли збільшується обсяг даних або кількість запитів.

REST API слугує точкою об’єднання між серверною логікою та інтерфейсом користувача. Клієнтська частина на React + TypeScript не просто відображає цифри, а формує візуальне представлення рівня стресу: графіки, індикатори, топ-теми, короткі пояснення. Інтерфейс дозволяє швидко знайти потрібне місто й отримати деталізовану аналітику за останній період.

З технічного погляду робота продемонструвала, що навіть порівняно невелика модель та легка архітектура можуть давати якісний результат, якщо правильно побудовано весь проце. У науковому сенсі проєкт вніс елемент новизни у спосіб поєднання контекстних ознак тексту, комбінованої геолокації та часової динаміки, що дозволило сформувати універсальний міський індекс стресу.

Підсумовуючи, система показала, що аналіз емоцій населення на основі цифрових текстів може бути не лише технічно реалістичним, але й корисним для дослідників, аналітиків і навіть органів місцевого управління. Вона демонструє, як сучасні NLP-моделі можуть працювати з нестабільними, шумними даними та при цьому формувати зрозумілу метрику, яка відображає загальний емоційний фон міста. У перспективі такий підхід може бути розширений на більшу кількість джерел, інші емоційні стани або навіть прогностичні задачі, наприклад, раннє виявлення піків напруги в інформаційному полі.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. Social network user penetration global statistics, 2018 https://www.statista.com/statistics/788084/number-of-social-media-accounts/

2. Циганчук Т.В. *Психологія стресу: навч. посібник*. Київ: Кафедра, 2016.

3. Bracha H.S. *Posttraumatic Stress Disorder: From Neurobiology to Treatment*. New York: Oxford University Press, 2020. 312 p.

4. Aboujaoude E., Starcevic V. (Eds.). *Mental Health in the Digital Age: Grave Dangers, Great Promise*. New York: Oxford University Press, 2015.

5. Lupien S.J., Juster R.P., Raymond C., Marin M.F. *The Science of Stress: Living Under Pressure*. Cham: Springer, 2022. 278 p.

6. Saslow L.R., Moyal C., et al. "Speaking under pressure: Low linguistic complexity is linked to high physiological and emotional stress reactivity." *Psychophysiology*, vol. 51, no. 3, 2014, pp. 257–266.

7. Guntuku S.C., Yaden D.B., Kern M.L., Ungar L.H., Eichstaedt J.C. (2017). *Detecting depression and mental illness on social media: an integrative review*. Current Opinion in Behavioral Sciences, 18, 43–49.

8. Chu B., Marwaha K., et al. "Physiology, Stress Reaction." *StatPearls [Internet]*. StatPearls Publishing, 2024.

9. Guntuku S.C., Buffone A., et al. "Understanding and Measuring Psychological Stress Using Social Media." *Proceedings of ICWSM*, vol. 13, 2019, pp. 214–225.

10. Lin H., Jia J., et al. "Detecting Stress Based on Social Interactions in Social Networks." *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, 2018, pp. 515–528.

11. Rastogi A., Liu Q., Cambria E. "Stress Detection from Social Media Texts: A Benchmark Study." *IEEE Transactions on Computational Social Systems*, vol. 8, no. 1, 2021, pp. 215–228.

12. Thelwall M. “TensiStrength: Stress and relaxation magnitude detection for social media texts.” *Information Processing & Management*, vol. 53, no. 1, 2017, pp. 106–121.
13. Lin H., Jia J., et al. “User-Level Psychological Stress Detection from Social Media Using Deep Neural Network.” *Proceedings of ACM Multimedia*, 2014, pp. 507–516.
14. Gao R., Hao B., Li H., Gao Y., Zhu T. (2019). *Developing simplified Chinese psychological linguistic analysis dictionary for microblog*. Frontiers in Psychology, 10, 150.
15. Kramer A.D.I., Guillory J.E., Hancock J.T. “Experimental evidence of massive-scale emotional contagion through social networks.” *Proceedings of the National Academy of Sciences*, vol. 111, no. 24, 2014, pp. 8788–8790.
16. CSIRO. *Twitter tool shows a real-time view of our emotions* (News Release). Canberra: CSIRO, 19 May 2014.
17. Sykora M., Jackson T.W., O’Brien A. (2020). *Emotive ontology and its application to social media analysis*. Information Processing & Management, 57(3), 102137.
18. Marz N., Warren J. *Big Data: Principles and Best Practices of Scalable Realtime Systems*. Manning Publications, 2015.
19. Frank M.R., Mitchell L., et al. “Happiness and the patterns of life: A study of geotagged tweets.” *Scientific Reports*, vol. 3, 2013, Article 2625.
20. Mitchell R. *Web Scraping with Python* (2nd ed.). Sebastopol: O’Reilly Media, 2018.
21. Novak P.K., Smailović J., et al. “Sentiment of emojis.” *PLOS ONE*, vol. 10, no. 12, 2015, e0144296.
22. Rahimi A., Cohn T., Baldwin T. (2017). *A neural model for user geolocation and lexical dialectology*. Transactions of the Association for Computational Linguistics, 5, 207–220.

23. Hecht B., Hong L., Suh B., Chi E. “Tweets from Justin Bieber’s heart: the dynamics of the location field in user profiles.” *Proceedings of CHI*, 2011, pp. 237–246.
24. Wing B., Baldridge J. “Simple supervised document geolocation with geodesic grids.” *Proceedings of ACL*, 2011, pp. 955–964.
25. Eisenstein J., O’Connor B., et al. “A latent variable model for geographic lexical variation.” *Proceedings of EMNLP*, 2010, pp. 1277–1287.
26. Jurgens D. “That’s What Friends Are For: Inferring Location in Online Social Media Platforms Based on Social Relationships.” *Proceedings of ICWSM*, 2013, pp. 273–282.
27. Yin C., Lampert A., Cameron M., Robinson B., Power R. (2021). *Using social media to enhance emergency situation awareness: A literature review and research agenda*. International Journal of Information Management, 57, 102261.
28. Pang B., Lee L. “Opinion Mining and Sentiment Analysis.” *Foundations and Trends in Information Retrieval*, vol. 2, no. 1–2, 2008, pp. 1–135.
29. Young T., Hazarika D., et al. “Recent Trends in Deep Learning Based Natural Language Processing.” *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, 2018, pp. 55–75.
30. Jiang Z., Ren Y., Ferrara E. (2021). *Measuring and modeling review helpfulness: A survey*. Information Fusion, 67, 62–79.
31. Hutto C., Gilbert E. “VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text.” *Proceedings of ICWSM*, 2014, pp. 216–225.
32. Davidov D., Tsur O., Rappoport A. “Semi-supervised recognition of sarcastic sentences in Twitter.” *Proceedings of CoNLL*, 2010, pp. 107–116.
33. Mikolov T., Chen K., et al. “Efficient Estimation of Word Representations in Vector Space.” *Proceedings of ICLR Workshops*, 2013, 12 pp.

34. Pennington J., Socher R., Manning C.D. “GloVe: Global Vectors for Word Representation.” *Proceedings of EMNLP*, 2014, pp. 1532–1543.
35. Bojanowski P., Grave E., et al. “Enriching Word Vectors with Subword Information.” *Transactions of the ACL*, vol. 5, 2017, pp. 135–146.
36. Yu S., Zhu Y., Li Y., Zhang Y. (2019). *A survey on neural network language models*. IEEE Access, 7, 19143–19165.
37. Kim Y. “Convolutional Neural Networks for Sentence Classification.” *Proceedings of EMNLP*, 2014, pp. 1746–1751.
38. Vaswani A., Shazeer N., et al. “Attention Is All You Need.” *Proceedings of NIPS*, 2017, pp. 5998–6008.
39. Devlin J., Chang M., et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” *Proceedings of NAACL*, 2019, pp. 4171–4186.
40. **Twitter.** Twitter Developer Platform – *API Documentation* https://developer.x.com/en/docs/x-api
41. **Meta.** Facebook Graph API – *Developer Documentation* https://developers.facebook.com/docs/graph-api/
42. **Telegram.** Telegram Bot API – *Official Documentation* https://core.telegram.org/bots/api
43. **SeleniumHQ.** Selenium WebDriver – *Documentation* https://www.selenium.dev/documentation/webdriver/
44. **Microsoft.** Playwright for Python – *Developer Guide* https://playwright.dev/python/docs/api/class-playwright
45. Matthews G., Zeidner M., Roberts R.D. Emotional Intelligence: A Framework for Individual Differences. 2nd ed. New York: Routledge, 2018. 434 p.
46. Gross J.J. (Ed.) Handbook of Emotion Regulation. 2nd ed. New York: Guilford Press, 2015. 654 p.
47. Compas B.E., Jaser S.S., Bettis A.H. Coping and Emotion Regulation from Childhood to Early Adulthood. Cham: Springer, 2021. 224 p.

48. Bar-On R., Parker J.D.A. (Eds.) The Handbook of Emotional Intelligence: Theoretical and Applied Perspectives. 3rd ed. San Francisco: Jossey-Bass, 2017. 489 p.
49. Eisenberg N., Spinrad T.L. (Eds.) Emotion Regulation: Conceptual and Practical Issues. New York: Springer, 2023. 312 p.

# ДОДАТКИ

# ДОДАТОК А
## Код файлу collector/rss_collector.py

```
import feedparser
from datetime import datetime
from typing import List, Dict
import logging

logger = logging.getLogger(__name__)

def fetch_rss_feed(url: str) -> List[Dict]:
    feed = feedparser.parse(url)
    messages: List[Dict] = []

    for entry in feed.entries:
        published = None
        if hasattr(entry, "published_parsed") and entry.published_parsed:
            published = datetime(*entry.published_parsed[:6])

        messages.append({
            "source": url,
            "raw_text": entry.get("title", "") + "\n" + entry.get("summary",
""),
            "published_at": published,
            "link": entry.get("link"),
            "meta": {
                "rss_id": entry.get("id"),
                "language": entry.get("language", "uk"),
                "city_hint": None
            }
        })

    logger.info("Fetched %s items from RSS %s", len(messages), url)
    return messages
```

# ДОДАТОК Б

## Код файлу collector/spiders/news_spider.py

```
import scrapy
from datetime import datetime

class NewsSpider(scrapy.Spider):
    name = "news_spider"
    allowed_domains = ["example.com"]
    start_urls = ["https://www.example.com/news"]

    def parse(self, response):
        articles = response.css("div.news-item")
        for article in articles:
            link = response.urljoin(article.css("a::attr(href)").get())
            yield response.follow(link, callback=self.parse_article)

    def parse_article(self, response):
        title = response.css("h1::text").get(default="").strip()
        paragraphs = response.css("div.article-body p::text").getall()
        text = "\n".join(p.strip() for p in paragraphs if p.strip())

        published_raw = response.css("time::attr(datetime)").get()
        published_at = None
        if published_raw:
            try:
                published_at = datetime.fromisoformat(published_raw)
            except ValueError:
                published_at = None

        yield {
            "source": "example.com",
            "raw_text": f"{title}\n{text}",
            "published_at": published_at,
            "link": response.url,
            "meta": {
                "language": "uk",
                "city_hint": None
            }
        }
```

# ДОДАТОК В

## Код файлу collector/telegram_collector.py

```
import os
import asyncio
from typing import List, Dict
from datetime import datetime
import logging

from telethon import TelegramClient

logger = logging.getLogger(__name__)

API_ID = os.getenv("API_ID")
API_HASH = os.getenv("API_HASH")
SESSION_NAME = os.getenv("SESSION_NAME")

client = TelegramClient(SESSION_NAME, API_ID, API_HASH)

async def fetch_channel_messages(channel: str, limit: int = 200) ->
List[Dict]:
    messages: List[Dict] = []

    await client.start()
    async for msg in client.iter_messages(channel, limit=limit):
        if not msg.message:
            continue

        messages.append({
            "source": f"telegram:{channel}",
            "raw_text": msg.message,
            "published_at": msg.date.replace(tzinfo=None) if
isinstance(msg.date, datetime) else None,
            "link": f"https://t.me/{channel}/{msg.id}",
            "meta": {
                "message_id": msg.id,
                "language": "uk",
                "city_hint": None
            }
        })

    logger.info("Fetched %s messages from Telegram channel %s", len(messages),
channel)
    return messages

def collect_telegram_channel(channel: str, limit: int = 200) -> List[Dict]:
    loop = asyncio.get_event_loop()
    return loop.run_until_complete(fetch_channel_messages(channel, limit))
```

# ДОДАТОК Г
## Код файлу collector/facebook_collector.py

```
import requests
from typing import Optional, Dict
from datetime import datetime
import logging

logger = logging.getLogger(__name__)

GRAPH_API_URL = "https://graph.facebook.com/v18.0"

def fetch_public_post(post_id: str, access_token: str) -> Optional[Dict]:
    url = f"{GRAPH_API_URL}/{post_id}"
    params = {
        "fields": "message,created_time",
        "access_token": access_token
    }

    try:
        response = requests.get(url, params=params, timeout=10)
        response.raise_for_status()
        data = response.json()
    except requests.RequestException as exc:
        logger.error("Error fetching Facebook post %s: %s", post_id, exc)
        return None

    message = data.get("message", "")
    created_time = data.get("created_time")

    published_at = None
    if created_time:
        try:
            published_at = datetime.fromisoformat(created_time.replace("Z",
"+00:00")).replace(tzinfo=None)
        except ValueError:
            published_at = None

    return {
        "source": "facebook",
        "raw_text": message,
        "published_at": published_at,
        "link": f"https://www.facebook.com/{post_id}",
        "meta": {
            "post_id": post_id,
            "language": "uk",
            "city_hint": None
        }
    }
```

# ДОДАТОК Д
## Код файлу collector/utils.py

```
import time
import random
import logging
from typing import Callable, Any

logger = logging.getLogger(__name__)

def with_retries(func: Callable[[], Any], max_retries: int = 5) -> Any:
    last_exc = None
    for attempt in range(1, max_retries + 1):
        try:
            return func()
        except Exception as exc:
            last_exc = exc
            delay = random.uniform(1.0, 3.0)
            logger.warning(
                "Error in %s, attempt %s/%s, retrying in %.2f s: %s",
                func.__name__, attempt, max_retries, delay, exc
            )
            time.sleep(delay)

    logger.error("Function %s failed after %s attempts: %s", func.__name__,
max_retries, last_exc)
    raise last_exc
```

## Код файлу collector/save_to_db.py

```
from db.models import Message, City

def save_message(item: dict):
    city_name = item["meta"].get("city_hint")
    city_obj = None
    if city_name:
        city_obj = City.objects.filter(name__iexact=city_name).first()

    msg = Message.objects.create(
        source=item["source"],
        raw_text=item["raw_text"],
        clean_text=item["clean_text"],
        published_at=item["published_at"],
        link=item["link"],
        meta=item["meta"],
        city=city_obj
    )

    return msg
```

# ДОДАТОК Е
## Код файлу collector/pipeline.py

```
import os
from typing import List, Dict
from .rss_collector import fetch_rss_feed
from .telegram_collector import collect_telegram_channel
from .facebook_collector import fetch_public_post
from .normalizer import normalize_message
from db.models import Message  # модель Django

def collect_all_sources() -> None:
    all_messages: List[Dict] = []

    rss_urls = []
    for url in rss_urls:
        all_messages.extend(fetch_rss_feed(url))

    all_messages.extend(collect_telegram_channel("kyiv_channel"))

    fb_post_ids = ["1234567890123456_987654321"]
    for post_id in fb_post_ids:
        fb_data = fetch_public_post(post_id,
access_token=os.getenv("ACCESS_TOKEN"))
        if fb_data:
            all_messages.append(fb_data)

    for item in all_messages:
        normalized = normalize_message(item)
        Message.objects.create(
            source=normalized["source"],
            raw_text=normalized["raw_text"],
            clean_text=normalized["clean_text"],
            published_at=normalized["published_at"],
            link=normalized["link"],
            meta=normalized["meta"]
        )
```

# ДОДАТОК Ж

## Код файлу db/models.py

```
from django.db import models
from django.contrib.postgres.fields import JSONField

class City(models.Model):
    name = models.CharField(max_length=100, unique=True)
    synonyms = JSONField(default=list)  # альтернативні назви
    latitude = models.FloatField(null=True, blank=True)
    longitude = models.FloatField(null=True, blank=True)

    def __str__(self):
        return self.name


class Message(models.Model):
    source = models.CharField(max_length=200)
    raw_text = models.TextField()
    clean_text = models.TextField(null=True, blank=True)
    published_at = models.DateTimeField(null=True, blank=True)
    link = models.TextField(null=True, blank=True)
    meta = JSONField(default=dict)

    city = models.ForeignKey(City, null=True, blank=True,
                             on_delete=models.SET_NULL,
                             related_name="messages")

    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.source}: {self.raw_text[:40]}..."


class StressResult(models.Model):
    message = models.OneToOneField(Message, on_delete=models.CASCADE,
                                   related_name="stress_result")
    stress_score = models.FloatField()  # 0-1 або 0-100
    confidence = models.FloatField()
    emotions = JSONField(default=dict)  # опціонально: додаткові індикатори
моделі

    evaluated_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Stress {self.stress_score:.2f} ({self.confidence:.2f})"


class Topic(models.Model):
    city = models.ForeignKey(City, on_delete=models.CASCADE,
                             related_name="topics")
    name = models.CharField(max_length=100)
    weight = models.FloatField(default=0.0)  # важливість теми за період

    def __str__(self):
        return f"{self.city.name}: {self.name}"
```

# ДОДАТОК К

## Код файлу nlp/model_loader.py

```
import torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer

MODEL_NAME = "ukr-models/ukr-roberta-base-stress"

class StressModel:
    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

        self.tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
        self.model =
AutoModelForSequenceClassification.from_pretrained(MODEL_NAME)
        self.model.to(self.device)
        self.model.eval()

    def tokenize(self, text: str):
        return self.tokenizer(
            text,
            padding="max_length",
            truncation=True,
            max_length=256,
            return_tensors="pt"
        ).to(self.device)
```

## Код файлу nlp/inference.py

```
import torch
import torch.nn.functional as F
from typing import Dict

from nlp.model_loader import StressModel
from preprocessing.preprocess import preprocess

model_instance = StressModel()

def evaluate_text(text: str) -> Dict:
    cleaned = preprocess(text)

    encoded = model_instance.tokenize(cleaned)
    with torch.no_grad():
        output = model_instance.model(**encoded)

    logits = output.logits.squeeze(0)
    probabilities = F.softmax(logits, dim=-1)

    # Припустимо, що клас 1 = високий стрес
    stress_score = float(probabilities[1].item())
    confidence = float(probabilities.max().item())

    return {
        "cleaned_text": cleaned,
        "stress_score": stress_score,
        "confidence": confidence
    }
```

# ДОДАТОК Л

## Код файлу nlp/classifier.py

```
from db.models import Message, StressResult
from nlp.inference import evaluate_text

def classify_message(message: Message) -> StressResult:
    result = evaluate_text(message.raw_text)

    message.clean_text = result["cleaned_text"]
    message.save(update_fields=["clean_text"])

    sr = StressResult.objects.create(
        message=message,
        stress_score=result["stress_score"],
        confidence=result["confidence"],
        emotions={}  # можна розширити у майбутньому
    )

    return sr
```

## Код файлу aggregator/query.py

```
from db.models import City, Message
from django.utils import timezone
from datetime import timedelta

def get_city_messages(city_name: str, days: int = 7):
    city = City.objects.filter(name__iexact=city_name).first()
    if not city:
        city = City.objects.filter(synonyms__contains=[city_name]).first()

    if not city:
        return [], None

    time_threshold = timezone.now() - timedelta(days=days)

    messages = (
        Message.objects
        .filter(city=city, published_at__gte=time_threshold)
        .select_related("stress_result")
        .order_by("-published_at")
    )

    return messages, city
```

# ДОДАТОК М

## Код файлу aggregator/average.py

```python
import numpy as np

def compute_weighted_average(messages):
    scores = []
    weights = []

    for msg in messages:
        if not hasattr(msg, "stress_result"):
            continue

        s = msg.stress_result.stress_score
        c = msg.stress_result.confidence

        # вага залежить від упевненості моделі
        weight = max(0.1, c)

        scores.append(s)
        weights.append(weight)

    if not scores:
        return 0.0

    weighted_avg = float(np.average(scores, weights=weights))
    return weighted_avg
```

## aggregator/average.py (оновлений фрагмент)

```python
from aggregator.time_weights import time_weight

def compute_weighted_average(messages):
    scores = []
    weights = []

    for msg in messages:
        if not hasattr(msg, "stress_result"):
            continue

        s = msg.stress_result.stress_score
        c = msg.stress_result.confidence
        tw = time_weight(msg.published_at)

        weight = c * tw
        scores.append(s)
        weights.append(weight)

    if not scores:
        return 0.0

    return float(np.average(scores, weights=weights))
```

# ДОДАТОК Н

## Код файлу aggregator/trend.py

```
from collections import defaultdict
from datetime import datetime

def compute_daily_trend(messages):
    daily_scores = defaultdict(list)

    for msg in messages:
        if not hasattr(msg, "stress_result"):
            continue

        day = msg.published_at.date()
        daily_scores[day].append(msg.stress_result.stress_score)

    trend = []
    for day, values in sorted(daily_scores.items()):
        avg = float(sum(values) / len(values))
        trend.append({"date": day.isoformat(), "value": avg})

    return trend
```

## Код файлу aggregator/topics.py

```
import re
from collections import Counter
from db.models import Topic

STOPWORDS = {"в", "і", "та", "що", "але", "як", "на", "у", "до", "за"}

def extract_keywords(messages, max_topics: int = 8):
    words = []

    for msg in messages:
        if not msg.clean_text:
            continue

        tokens = re.findall(r"[А-Яа-яІіЇїЄєA-Za-z']+", msg.clean_text.lower())
        tokens = [t for t in tokens if t not in STOPWORDS]

        words.extend(tokens)

    counter = Counter(words)
    return counter.most_common(max_topics)
```

# ДОДАТОК П

## Код файлу aggregator/summary.py

```
from aggregator.average import compute_weighted_average
from aggregator.trend import compute_daily_trend
from aggregator.topics import extract_keywords, save_topics

def build_city_summary(messages, city):
    avg = compute_weighted_average(messages)
    trend = compute_daily_trend(messages)
    keywords = extract_keywords(messages)
    save_topics(city, keywords)

    return {
        "city": city.name,
        "stress_level": avg,
        "trend": trend,
        "topics": [{"name": k, "weight": w} for k, w in keywords],
        "messages_count": len(messages),
    }
```

## Код файлу api/serializers.py

```
from rest_framework import serializers

class TopicSerializer(serializers.Serializer):
    name = serializers.CharField()
    weight = serializers.FloatField()


class TrendItemSerializer(serializers.Serializer):
    date = serializers.CharField()
    value = serializers.FloatField()


class StressSummarySerializer(serializers.Serializer):
    city = serializers.CharField()
    stress_level = serializers.FloatField()
    messages_count = serializers.IntegerField()
    trend = TrendItemSerializer(many=True)
    topics = TopicSerializer(many=True)
```

# ДОДАТОК Р

## Код файлу api/views.py

```python
from rest_framework.response import Response
from rest_framework.decorators import api_view

from aggregator.query import get_city_messages
from aggregator.summary import build_city_summary
from api.serializers import StressSummarySerializer
from api.cache import cached

@api_view(["GET"])
def city_stress(request, city: str):
    def compute():
        messages, city_obj = get_city_messages(city, days=7)
        if not city_obj:
            return None
        summary = build_city_summary(messages, city_obj)
        return summary

    data = cached(f"summary:{city}", 300, compute)

    if not data:
        return Response({"error": "city_not_found"}, status=404)

    return Response(StressSummarySerializer(data).data)
```

## Код файлу src/api/stressApi.ts

```typescript
export interface TrendItem {
    date: string;
    value: number;
}

export interface Topic {
    name: string;
    weight: number;
}

export interface StressSummary {
    city: string;
    stress_level: number;
    messages_count: number;
    trend: TrendItem[];
    topics: Topic[];
}

export async function fetchCityData(city: string): Promise<StressSummary> {
    const res = await fetch(`/api/stress/${city}/full/`);
    if (!res.ok) throw new Error("City not found");
    return await res.json();
}
```

# ДОДАТОК С

## Код файлу src/pages/Dashboard.tsx

```
import { useEffect, useState } from "react";
import { Container, Box } from "@mui/material";

import SearchBar from "../components/SearchBar";
import StressCard from "../components/StressCard";
import TrendChart from "../components/TrendChart";
import TopicsList from "../components/TopicsList";
import CityHeader from "../components/CityHeader";

import { fetchCityData, StressSummary } from "../api/stressApi";

export default function Dashboard() {
    const [city, setCity] = useState("Київ");
    const [data, setData] = useState<StressSummary | null>(null);

    const load = async () => {
        try {
            const r = await fetchCityData(city);
            setData(r);
        } catch {
            setData(null);
        }
    };

    useEffect(() => {
        load();
    }, [city]);

    return (
        <Container sx={{ mt: 4 }}>
            <SearchBar city={city} onChange={setCity} />

            {data && (
                <Box>
                    <CityHeader name={data.city} />
                    <StressCard level={data.stress_level}
count={data.messages_count} />
                    <Box sx={{ mt: 4 }}>
                        <TrendChart data={data.trend} />
                    </Box>
                    <Box sx={{ mt: 4 }}>
                        <TopicsList topics={data.topics} />
                    </Box>
                </Box>
            )}
        </Container>
    );
}
```

# ДОДАТОК Т

## Код файлу docker-compose.yml

```yaml
version: "3.9"

services:
  backend:
    build: ./docker/backend
    container_name: stress-backend
    depends_on:
      - db
      - nlp
    environment:
      DATABASE_URL: postgres://admin:password123@db:5432/stress
    ports:
      - "8000:8000"

  frontend:
    build: ./docker/frontend
    container_name: stress-frontend
    depends_on:
      - backend
    ports:
      - "3000:80"

  nlp:
    build: ./docker/nlp
    container_name: stress-nlp
    ports:
      - "8501:8501"

  db:
    image: postgres:14
    restart: always
    environment:
      POSTGRES_DB: stress
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password123
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```