МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Аліна САВЧЕНКО
"____"_____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Клієнт–серверні засоби інтелектуального аналізу інформаційних ресурсів мережі Інтернет"

Виконавець: студент групи УС-211М Куц Кіріл Юрійович

Керівник: к.т.н., доцент кафедри КІТ Климова Асія Сабирівна

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Куца Кіріла Юрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Клієнт–серверні засоби інтелектуального аналізу інформаційних ресурсів мережі Інтернет»
затверджена наказом ректора від 12.10.2021 за № 2229/ст.

2. Термін виконання роботи: з 12.10.2021 по 31.12.2021.

3. Вихідні дані до роботи: мова гіпертекстової розмітки HTML, CSS, TypeScript, Node.js, Nest.js, NLP.

4. Зміст пояснювальної записки: вступ, загальна характеристика, принципи побудови засобів інтелектуального аналізу інформаційних ресурсів мережі інтернет, аналіз методів та технологій для розробки засобу аналізу інформаційних ресурсів, розробка засобу інтелектуального аналізу інформаційних ресурсів мережі інтернет.

5. **Перелік обов'язкового ілюстративного матеріалу:** рисунки, діаграми, презентація.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт.	12.10.2021 – 14.10.2021	
2.	Загальний аналіз роботи сервісів інтелектуального аналізу інформаційних ресурсів мережі Інтернет.	15.10.2021 – 20.10.2021	
3.	Опис функціональних можливостей, аналіз переваг та недоліків розглянутих сервісів.	20.10.2021 – 26.10.2021	
4.	Формування переліку функціональних можливостей та вимог до розроблюваного програмного комплексу.	26.10.2021 – 30.10.2021	
5.	Аналіз архітектурних підходів та технологій для розробки сервісу.	30.10.2021 – 06.11.2021	
6.	Опис функціональних можливостей додатку із точку зору користувача, планування розробки.	07.11.2021 – 14.11.2021	
7.	Розробка програмного комплексу.	15.11.2021 – 02.12.2021	
8.	Опис функціоналу програмного комплексу для аналізу інформаційних ресурсів мережі Інтернет.	03.12.2021 – 06.12.2021	
9.	Оформлення пояснювальної записки, створення презентації та доповіді.	07.12.2021 – 10.12.2021	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи: _____ Асія КЛИМОВА
(підпис керівника)

Завдання прийняв до виконання: _____ Кіріл КУЦ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Клієнт–серверні засоби інтелектуального аналізу інформаційних ресурсів мережі Інтернет” складається із вступу, трьох розділів, висновків, списку бібліографічних посилань використаних джерел і містить 89 сторінок тексту, 35 рисунків. Список використаних джерел містить 8 найменувань.

Об’єкт розробки – засіб інтелектуального аналізу інформаційних ресурсів мережі Інтернет.

Мета роботи – розробка засобу інтелектуального аналізу інформаційних ресурсів мережі Інтернет, який надає аналітичну інформацію по аналізованому тексту.

Об’єктом дослідження – програма, аналітичний інструмент інтелектуального аналізу інформаційних ресурсів.

Предмет дослідження – додаток для інтелектуального аналізу інформаційних ресурсів мережі інтернет.

Перелік ключових слів: ПРОГРАМНИЙ КОМПЛЕКС, ІНТЕЛЕКТУАЛЬНИ АНАЛІЗ, АНАЛІЗ ТЕКСТУ, АНАЛІТИЧНІ ДАНІ, ЗБІР АНАЛІТИЧНИХ ДАНИХ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API – Application programming interface
CSS – Cascading Style Sheets
DOM – Document Object Model
HIPPA – Health Insurance Portability and Accountability Act
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
JS – JavaScript
JSON – JavaScript Object Notation
NER – Named-entity recognition
NLP – Natural Language Processing
PCI – Payment Card Industry
Sass – Syntactically Awesome Style Sheets
SSL – Secure Sockets Layer
TLS – Transport Layer Security
URL – Uniform Resource Locator
VPS – Virtual private server
XML – eXtensible Markup Language
ОС – операційна система

ВСТУП

Сучасні технології активно розвиваються. Ера інтернету надає усім людям доступ до величезної кількості інформації. Її безперервний потік слугує своєрідним повсякденним інформаційним фоном.

Люди розробили спеціальні засоби пошуку інформації в мережі інтернет, з метою структуризації даних. Пошукові системі тісно інтегрувались в наше життя, та надають можливості пошуку інформації та відповіді на будь-яке питання.

Важко переоцінити потребу у засобах інтелектуального аналізу інформаційних ресурсів в часи коли інформації довкола так багато.

Аналіз тексту (ТА) — це техніка машинного навчання, яка використовується для автоматичного вилучення цінної інформації з неструктурованих текстових даних. Компанії використовують інструменти аналізу тексту, щоб швидко переварювати онлайн-дані та документи та перетворювати їх на корисну інформацію.

Використовують аналіз тексту, щоб витягти конкретну інформацію, як-от ключові слова, імена або інформацію про компанію, з тисяч електронних листів, або класифікувати відповіді опитування за настроєм і темою.

Аналіз тексту дає якісні результати, а аналіз тексту — кількісні результати. Якщо машина виконує аналіз тексту, вона визначає важливу інформацію в самому тексті, але якщо вона виконує текстову аналітику, вона виявляє закономірності в тисячах текстів, що призводить до графіків, звітів, таблиць тощо.

Скажімо, менеджер служби підтримки хоче знати, скільки запитів у службу підтримки було вирішено окремими членами команди. У цьому випадку вони використовували б текстову аналітику, щоб створити графік, який візуалізує швидкість розгляду окремих запитів.

Однак, імовірно, менеджер також хоче знати, яка частка квитків призвела до позитивного чи негативного результату?

Аналізуючи текст у кожному квитку та наступні обміни, менеджери служби підтримки клієнтів можуть побачити, як кожен агент обробляв квитки та чи були клієнти задоволені результатом.

В основному, проблема аналізу тексту полягає в розшифровці неоднозначності людської мови, тоді як в аналітиці тексту це виявлення закономірностей і тенденцій на основі числових результатів.

Навчивши машини працювати над упорядкуванням та аналізом ваших текстових даних, можна отримати величезні знання та переваги.

Інструменти аналізу тексту дозволяють компаніям структурувати величезну кількість інформації, як-от електронні листи, чати, соціальні мережі, квитки на підтримку, документи тощо, за секунди, а не за дні, тож ви можете перенаправляти додаткові ресурси на важливіші бізнес-завдання.

Компанії переповнені інформацією, і сьогодні коментарі клієнтів можуть з'являтися де завгодно в Інтернеті, але буває важко стежити за всім цим. Аналіз тексту змінює гру, коли справа доходить до виявлення невідкладних справ, де б вони не з'являлися, 24/7 та в режимі реального часу. Навчаючи моделі аналізу тексту виявляти вирази та настрої, які мають на увазі негативність або терміновість, компанії можуть автоматично позначати твіти, огляди, відео, квитки тощо та вжити заходів раніше, ніж пізніше.

Люди роблять помилки. Факт. І чим виснажливіше і трудомісткіше завдання, тим більше вони допускають помилок. Навчаючи моделі аналізу тексту відповідно до ваших потреб і критеріїв, алгоритми можуть аналізувати, розуміти та сортувати дані набагато точніше, ніж люди.

Класифікація тексту – це процес призначення попередньо визначених тегів або категорій неструктурованому тексту. Це вважається одним з найкорисніших методів обробки природної мови, оскільки він настільки універсальний і може організовувати, структурувати та класифікувати майже будь-яку форму тексту, щоб передавати значущі дані та вирішувати проблеми. Обробка природної мови (NLP) — це техніка машинного навчання, яка дозволяє комп'ютерам розбивати й розуміти текст так само, як це робить людина.

Клієнти вільно залишають свої думки про бізнес і продукти під час взаємодії з клієнтами, в опитуваннях та в Інтернеті. Аналіз настроїв використовує потужні алгоритми машинного навчання, щоб автоматично зчитувати й класифікувати за

полярністю думок (позитивні, негативні, нейтральні) і далі, у почуття й емоції письменника, навіть контекст та сарказм.

Наприклад, за допомогою аналізу настроїв компанії можуть позначати скарги чи термінові запити, тому з ними можна розглянути негайно – навіть запобігти PR-кризі в соціальних мережах. Класифікатори настроїв можуть оцінювати репутацію бренду, проводити дослідження ринку та допомагати покращувати продукти за допомогою відгуків клієнтів.

Іншим поширеним прикладом класифікації тексту є тематичний аналіз (або моделювання теми), який автоматично організує текст за темою. Якщо ми використовуємо категорії тем, як-от Ціни, Підтримка клієнтів і Простота використання, цей відгук про продукт буде класифікуватися як Простота використання.

Класифікатори тексту також можна використовувати для визначення наміру тексту. Виявлення намірів або класифікація намірів часто використовуються для автоматичного розуміння причини зворотного зв’язку клієнтів. Це скарга? Або клієнт пише з наміром придбати продукт? Машинне навчання може читати розмови чат-бота або електронні листи та автоматично спрямовувати їх до відповідного відділу або співробітника.

Вилучення тексту – це ще один широко використовуваний метод аналізу тексту, який витягує фрагменти даних, які вже існують у будь-якому тексті. Ви можете отримувати такі речі, як ключові слова, ціни, назви компаній та специфікації продуктів із новин, оглядів продуктів тощо.

Можна автоматично заповнювати електронні таблиці цими даними або виконувати вилучення разом з іншими методами аналізу тексту, щоб категоризувати та витягувати дані одночасно.

Екстрактор розпізнавання іменованих об’єктів (NER) знаходить об’єкти, які можуть бути людьми, компаніями чи місцеположеннями та існувати в текстових даних. Результати відображаються з міткою відповідної сутності.

Частота слів — це метод аналізу тексту, який вимірює слова або поняття, які найчастіше зустрічаються в даному тексті, за допомогою числової статистики TF-IDF (термін частота інверсної частоти документа).

Можна застосувати цю техніку для аналізу слів або виразів, які клієнти найчастіше використовують у розмовах про підтримку. Наприклад, якщо слово «доставка» найчастіше зустрічається в наборі негативних заявок у службу підтримки, це може свідчити про те, що клієнти незадоволені вашою службою доставки.

Колокація допомагає визначити слова, які зазвичай зустрічаються разом. Наприклад, у відгуках клієнтів на веб-сайті бронювання готелів слова «повітря» і «кондиціонування» частіше зустрічаються разом, а не окремо. Біграми (два сусідніх слова, наприклад, «кондиціонер» або «підтримка клієнтів») і триграми (три сусідніх слова, наприклад, «не в офісі» або «продовження») — це найпоширеніші типи спільного розташування, на які вам потрібно звернути увагу.

Спільне розташування може бути корисним для виявлення прихованих семантичних структур і покращення деталізації розуміння, підраховуючи біграми та триграми як одне слово.

Конкордантність допомагає визначити контекст і екземпляри слів або набору слів. Наприклад, нижче наведено відповідність слова «простий» у наборі оглядів додатків:

У цьому випадку узгодженість слова «простий» може дати нам швидке уявлення про те, як рецензенти використовують це слово. Його також можна використовувати для декодування неоднозначності людської мови до певної міри, дивлячись на те, як слова використовуються в різних контекстах, а також аналізуючи більш складні фрази.

Дуже часто слово має кілька значень, тому розшифровка значень слів є основною проблемою обробки природної мови. Візьмемо, наприклад, слово «світло». Чи стосується текст ваги, кольору чи електричного приладу? Розумний аналіз тексту з розшифровкою значень слів може розрізняти слова, які мають більше одного значення, але лише після навчання моделей для цього.

Текстові кластери здатні розуміти та групувати велику кількість неструктурованих даних. Хоча алгоритми кластеризації менш точні, ніж алгоритми класифікації, алгоритми кластеризації реалізуються швидше, тому що вам не потрібно позначати приклади для навчання моделей. Це означає, що ці розумні алгоритми видобувають інформацію та роблять прогнози без використання навчальних даних, інакше відомих як машинне навчання без нагляду.

Google — чудовий приклад того, як працює кластеризація. Коли ви шукаєте термін у Google, ви коли-небудь замислювалися, як потрібні лише секунди, щоб отримати відповідні результати? Алгоритм Google розбиває неструктуровані дані з веб-сторінок і групує сторінки в кластери навколо набору подібних слів або n-грам (всіх можливих комбінацій суміжних слів або літер у тексті). Отже, сторінки з кластера, які містять більшу кількість слів або n-грам, релевантних пошуковому запиту, відображатимуться першими в результатах.

Аналіз тексту може бути використанний в діапазоні залежно від бажаних результатів. Його можна застосувати до:

- цілих документів: отримує інформацію з повного документа або параграфа: наприклад, загальне враження від огляду клієнта;
- окремі речення: отримує інформацію з конкретних речень: наприклад, більш детальну думку кожного речення відгуку клієнта;
- підречення: отримує інформацію з підвиразів у реченні: наприклад, основні почуття кожної одиниці думки відгуку клієнта.

Коли ви знаєте, як ви хочете розділити свої дані, ви можете почати їх аналізувати.

Щоб автоматично аналізувати текст за допомогою машинного навчання, вам потрібно впорядкувати свої дані. Більшість цього робиться автоматично, і ви навіть не помітите, що це відбувається. Однак важливо розуміти, що в автоматичному аналізі тексту використовується ряд методів обробки природної мови (NLP), як показано нижче.

Токенізація, тегування частин мови та синтаксичний аналіз.

Токенізація — це процес розбиття рядка символів на семантично значущі частини, які можна проаналізувати (наприклад, слова), при відкиданні безглуздих фрагментів (наприклад, пробілів).

Після того, як маркери були розпізнані, настав час їх класифікувати. Тегування частини мови відноситься до процесу присвоєння виявленим лексемам граматичної категорії, наприклад іменника, дієслова тощо.

З усіма категоризованими лексемами та мовною моделлю (тобто граматикою) система тепер може створювати більш складні уявлення текстів, які вона аналізуватиме. Цей процес відомий як розбір. Іншими словами, синтаксичний розбір відноситься до процесу визначення синтаксичної структури тексту. Для цього алгоритм розбору використовує граматику мови, якою був написаний текст. Різні уявлення будуть результатом розбору одного і того ж тексту з різними граматиками.

Граматики залежностей можна визначити як граматики, які встановлюють спрямовані відношення між словами речень. Розбір залежностей — це процес використання граматики залежності для визначення синтаксичної структури речення

Граматики структурних фраз моделюють синтаксичні структури, використовуючи абстрактні вузли, пов’язані зі словами та іншими абстрактними категоріями (залежно від типу граматики), і ненаправлені відносини між ними. Розбір виборчого округу відноситься до процесу використання граматики округу для визначення синтаксичної структури речення.

Створення і лемматизація відносяться до процесу видалення всіх афіксів (тобто суфіксів, префіксів тощо), приєднаних до слова, щоб зберегти його лексичну базу, також відому як корінь або основа, або його словникову форму чи лему. Основна відмінність між цими двома процесами полягає в тому, що створення коренів зазвичай базується на правилах, які обрізають початок і закінчення слів (і іноді призводять до дещо дивних результатів), тоді як лемматизація використовує словники та набагато складніший морфологічний аналіз.

Щоб забезпечити більш точний автоматизований аналіз тексту, нам потрібно видалити слова, які надають дуже мало семантичної інформації або взагалі не мають значення. Ці слова також відомі як стоп-слова: a, and, or, the тощо.

Існує багато різних списків стоп-слів для кожної мови. Однак важливо розуміти, що вам може знадобитися додати слова до цих списків або видалити слова з них залежно від текстів, які ви хочете проаналізувати, і аналізів, які ви хочете виконати.

Можливо, ви захочете зробити якийсь лексичний аналіз домену, з якого надходять ваші тексти, щоб визначити слова, які слід додати до списку стоп-слів.

Класифікація тексту (також відома як категоризація тексту або тегування тексту) відноситься до процесу присвоєння тегів текстам на основі його вмісту.

У минулому класифікація тексту виконувалася вручну, що займало багато часу, було неефективним і неточним. Але моделі аналізу тексту автоматичного машинного навчання часто працюють за лічені секунди з неперевершеною точністю.

Найпопулярніші завдання класифікації тексту включають аналіз настроїв (тобто виявлення, коли в тексті говориться щось позитивне чи негативне про певну тему), виявлення теми (тобто визначення тем, про які йдеться у тексті) та виявлення намірів (тобто виявлення мети чи основного наміру). тексту), серед іншого, але існує ще багато інших програм, які можуть вас зацікавити.

Існує багато алгоритмів машинного навчання, які використовуються в класифікації текстів. Найбільш часто використовуваними є алгоритми сімейства Naive Bayes (NB), машини опорних векторів (SVM) і алгоритми глибокого навчання.

Сімейство алгоритмів наївного Байєса базується на теоремі Байєса та умовних ймовірностях входження слів зразка тексту в слова набору текстів, що належать даному тегу. Вектори, які представляють тексти, кодують інформацію про те, наскільки ймовірно, що слова в тексті зустрічаються в текстах даного тега. За допомогою цієї інформації можна обчислити ймовірність належності тексту будь-якому заданому тегу в моделі. Після того, як усі ймовірності були обчислені для вхідного тексту, модель класифікації поверне тег з найвищою ймовірністю як вихідний результат для цього введення.

Однією з головних переваг цього алгоритму є те, що результати можуть бути досить хорошими, навіть якщо немає багато навчальних даних.

Машини опорних векторів (SVM) — це алгоритм, який може розділити векторний простір текстів із тегами на два підпростори: один простір, який містить більшість векторів, що належать даному тегу, і інший підпростір, який містить більшість векторів, які не належать до цей тег.

Моделі класифікації, в основі яких використовується SVM, перетворюють тексти у вектори та визначать, до якої сторони кордону, що розділяє векторний простір для даного тега, належать ці вектори. Залежно від того, де вони приземляються, модель дізнається, належать вони до певного тегу чи ні.

Найважливішою перевагою використання SVM є те, що результати зазвичай кращі, ніж результати, отримані з Naive Bayes. Однак для SVM потрібно більше обчислювальних ресурсів.

Глибоке навчання — це набір алгоритмів і методів, які використовують «штучні нейронні мережі» для обробки даних так само, як і людський мозок. Ці алгоритми використовують величезні обсяги навчальних даних (мільйони прикладів) для створення семантично багатих уявлень текстів, які потім можуть бути подані в моделі на основі машинного навчання різних видів, які будуть робити набагато точніші прогнози, ніж традиційні моделі машинного навчання.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА, ПРИНЦИПИ ПОБУДОВИ ЗАСОБІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ІНФОРМАЦІЙНИХ РЕСУРСІВ МЕРЕЖІ ІНТЕРНЕТ

Метою цього розділу є надання загальної характеристики та аналіз принципів роботи засобів для інтелектуального аналізу інформаційних ресурсів мережі інтернет. Детальний аналіз складових інформаційних ресурсів мережі інтернет дасть змогу сформувати план та вимоги щодо того, які частини та якими алгоритмами будуть аналізовані результуючим додатком. Ознайомлення з існуючими алгоритмами інтелектуального аналізу дасть змогу сформувати основні принципи та вимоги до додатку, скласти план розробки.

1.1. Аналіз складових інформаційних ресурсів мережі інтернет

Інформаційні ресурси мережі інтернет здебільшого представлені у вигляді веб-сторінок. Веб-сторінка, в свою чергу, формується з використанням мови розмітку гіпертексту HTML.

HTML – мова розмітки гіпертексту. Основою цієї мови є “теги”. Теги – спеціальний елемент, котрий формує структуру документу. Теги можуть бути вкладені один в другий, або використовуватись як окремий елемент. Саме використання тегів надало можливість передавати форматований контент через мережу інтернет.

Основні складові сучасного інформаційного ресурсу мережі інтернет наступні:

- шапка;
- основна частина;

- бокова панель;
- футер.

Іноді також можуть зустрічатись додаткові, допоміжні елементи:

- модальні вікна;
- пуш-сповіщення.

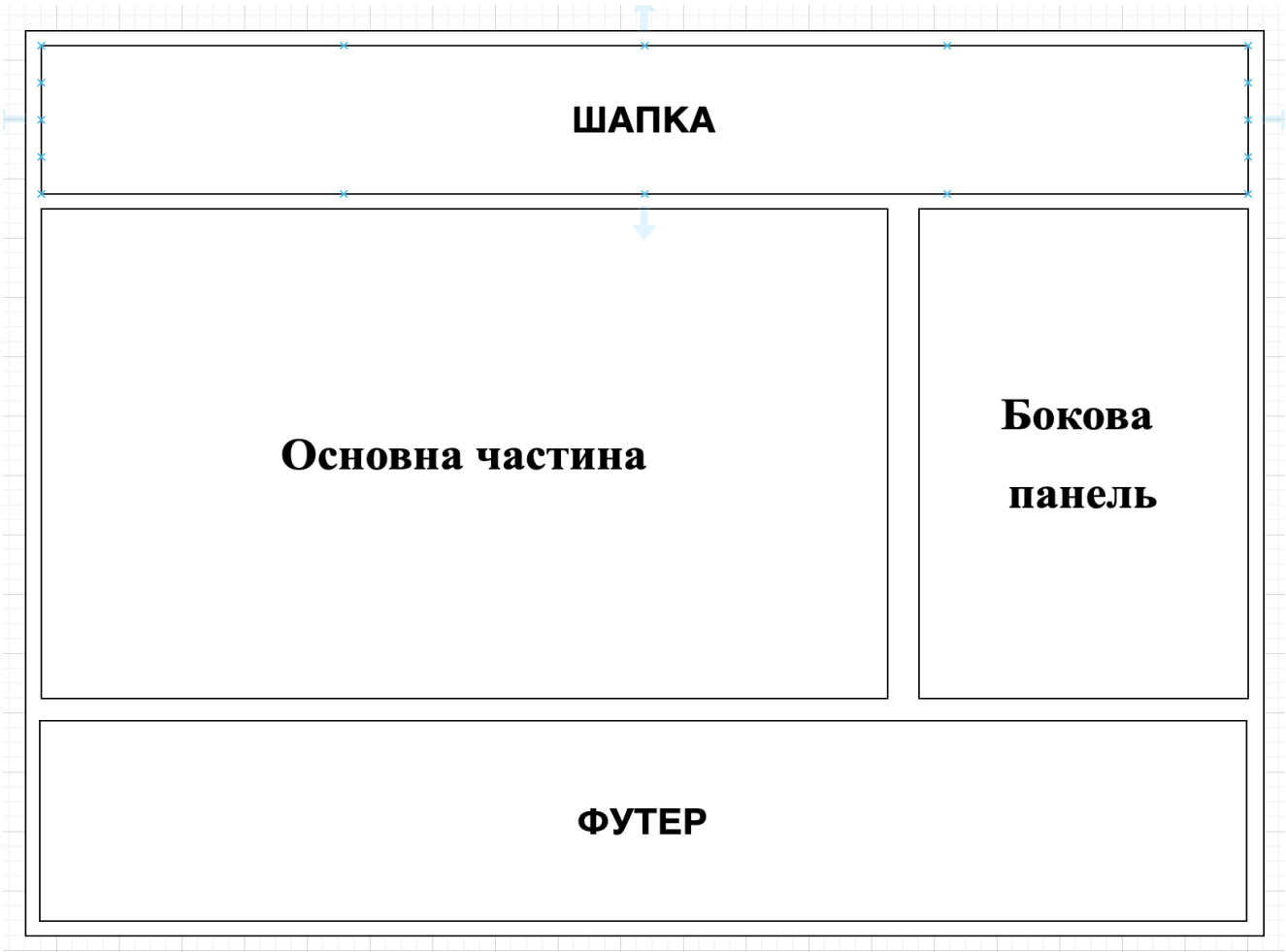

Рис. 1.1. HTML структура веб-сторінки

При інтелектуальному аналізі тексту усі складові веб-сторінки, окрім її основної частини, не є важливими, а тому алгоритм що її аналізує, або попередній етап обробки, повинен містити крок видалення зайвої інформації.

Оскільки основну цінність несе інформація збережена в текстовому вигляді варто зазначити які HTML теги найчастіше використовуються для її форматування, зокрема:

- Тег “p” - параграф тексту;
- Тег “h1” - заголовок першого рівня;

- Тег “h2” - заголовок другого рівня;
- Тег “h3” - заголовок третього рівня;
- Тег “h4” - заголовок четвертого рівня;
- Тег “h5” - заголовок п’ятого рівня;
- Тег “h6” - заголовок шостого рівня;
- Тег “span” - речення;
- Тег “strong” - виділення тексту жирним шрифтом;
- Тег “i” - виділення тексту шрифтом із нахилом (italic).

Саме з використанням описаних тегів результуюча веб сторінка має можливість відображати текст у відформатованому вигляді і й, відповідно, при розробці потрібно мати на увазі що саме вони містять основну цінність та текст для подальшого інтелектуального аналізу.

1.2. Формування переліку складових інформаційних ресурсів для інтелектуального аналізу

Інформаційні ресурси мережі інтернет складаються із веб-сторінок, котрі в свою чергу містять ієрархію блоків що форматують інформацію. Інформація на веб-сторінках може мати різний вигляд та формат, медіа тип. Зокрема сучасні засоби розробки веб-сторінок надають можливість крім тексту відображати наступні формати інформації:

- аудіо;
- картинки;
- відео;
- вбудовані фрейми.

Окрім того, останні стандарти дозволяють додавати на сторінку “Custom elements” – компоненти, що інкапсулюють в собі функціональну логіку, а зовні виглядають на звичайний тег.

Нові стандарти HTML в значній мірі розширюють функціональні можливості мови розмітки та її взаємодію з іншими інструментами веб-розробки.

Загалом, варто відзначити, що розроблюваний додаток при інтелектуальному аналізі повинен вміти:

- відокремлювати текст від усіх інших типів контенту;
- із отриманого тексту видаляти ті частини, що не належать основному контенту сторінки;
- аналізувати отриманий текст засобами інтелектуального аналізу.

Сформовані вимоги стануть основою майбутнього додатку та першим із розроблюваних модулів - модуль експорту тексту із веб-ресурсу.

Функціонально цей модуль повинен отримувати URL адресу в якості вхідного параметру та віддавати текст у якості вихідного.

Окремо варто зазначити, що деякі інформаційні веб-ресурси мають додатковий рівень захисту при завантаженні сторінки. Це зумовлено необхідністю захисту від різноманітних хакерських атак, зокрема DDOS атак.

DDOS атака - різновид хакерської атаки, при якій до ресурсу (найчастіше веб-ресурсу) здійснюється велика кількість звернень. Основна мета такого різновиду хакерської діяльності – унеможливити доступ користувачів до ресурсу.

Саме через потребу захисту від подібного роду атак значна кількість інформаційних веб-ресурсів додає окремий рівень захисту, обмежуючи обробку запитів виключно від справжніх користувачів.

Це потрібно враховувати при розробці сервісу отримання даних. Запит повинен містити коректно встановлені параметри HTTP заголовків та правильно обробляти випадки, коли веб-сторінка недоступна або рівень захисту ресурсу не дає можливості отримати її вміст.

1.3. Огляд алгоритмів інтелектуального аналізу легкості сприйняття інформаційних ресурсів

Сучасних світ потребує нових засобів аналізу інформації. Зокрема гостро стоїть проблема інтелектуального аналізу інформаційних ресурсів. А оскільки основним

контентом інформаційних ресурсів є текст - потреба в актуальних засобах аналізу тексту є вкрай актуальною.

Засоби аналізу тексту різноманітні. Починаються вони із сбору базових статистичних даних:

- кількість речень у тексті;
- кількість слів у тексту;
- кількість символів у тексті;
- кількість “довгих” слів у тексті;
- кількість “довгих” речень у тексті.

Ці три базові параметри дають можливість оцінити рівень складності прочитання та засвоєння інформацію. Для цього використовуються різноманітні формули скорингу рівня читабельності.

1.4. Формула аналізу легкості читання за Flesch

Flesch Reading Ease – показник, що визначає легкість читання тексту. Може коливатись у межах значень від 0 до 100. Чим нижче значення – тим легше читати текст.

Був придуманий наприкінці XX століття. Вперше був офіційно використаний в Армії США для визначення складності читання технічних інструкцій.

Із роками активно набирає популярності. На його основі складають вимоги до офіційних документів, щоб лімітувати складність їх читання та зробити доступними для розуміння кожного.

Формула має такий вигляд: 206.835 – 1.015 (кількість слів/кількість речень) – 84.6 (загальна кількість складів/кількість слів).

Аналіз формули показує пряму залежність результату від кількості довгих речень (перша частина формули) і довгих слів (друга частина формули).

1.5. Формула аналізу легкості читання за Flesch–Kincaid

Ця формула, так само як і попередня, створена задля надання можливості швидкої оцінки складність читання тексту. Особливість цієї формули полягає в тому, що на відміну від попередньої вона не має чітких меж можливих значень.

Широке застосування знайшла у сфері навчання - як універсальний спосіб оцінки складності прочитання тексту. Результат розрахунку формули - теоретична кількість років читача, необхідних для сприйняття тексту.

Сама формула виглядає наступним чином: 0.39 (кількість слів/кількість речень) +11.8 (загальна кількість складів/кількість слів) + 15.59.

1.6. Індекс туманності за Ганінгом

Третя за популярністю та частотою використання формула для аналізу легкості читання та усвідомлення інформації в форматі тексту. Історія цієї формули була розпочата Робертом Ганнінгом, бізнесменом у сфері публікацій газет та книг. Ця сфера активно потребувала можливостей аналізу легкості читання опублікованих матеріалів, тому ця формула й набула своєї популярності.

Ідея формули полягає в тому, що результат її обрахунків визначає умовну кількість років навчання для розуміння тексту. Чим більше результуючі значення – тим складніше читати текст. Результат прямо залежить від довжини слів та кількості складних слів.

Сама формула має наступний вигляд: 0.4 * (кількість слів/кількість речень) +100 (кількість “складких” слів/кількість слів).

1.7. Аналіз засобів перевірки легкості читання

Існує велика кількість засобів для інтелектуального аналізу легкості читання тексту. Проаналізувавши функціональність найкращих із них можна зробити висновки та висунути додаткові умови до розробленого додатку. Зокрема потрібно звернути увагу на наступні пункти:

- швидкість роботи;
- зручність використання;
- точність результатів.

Саме ці параметри є ключовими для користувача при визначенні чи буде він використовувати той чи інший інструмент в майбутньому.

1.8. Readable – сервіс інтелектуального аналізу читабельності

Readable - сервіс перевірки легкості читання. Працюють за SAAS моделлю, надаючи користувачам доступ до додатку за місячною підпискою.

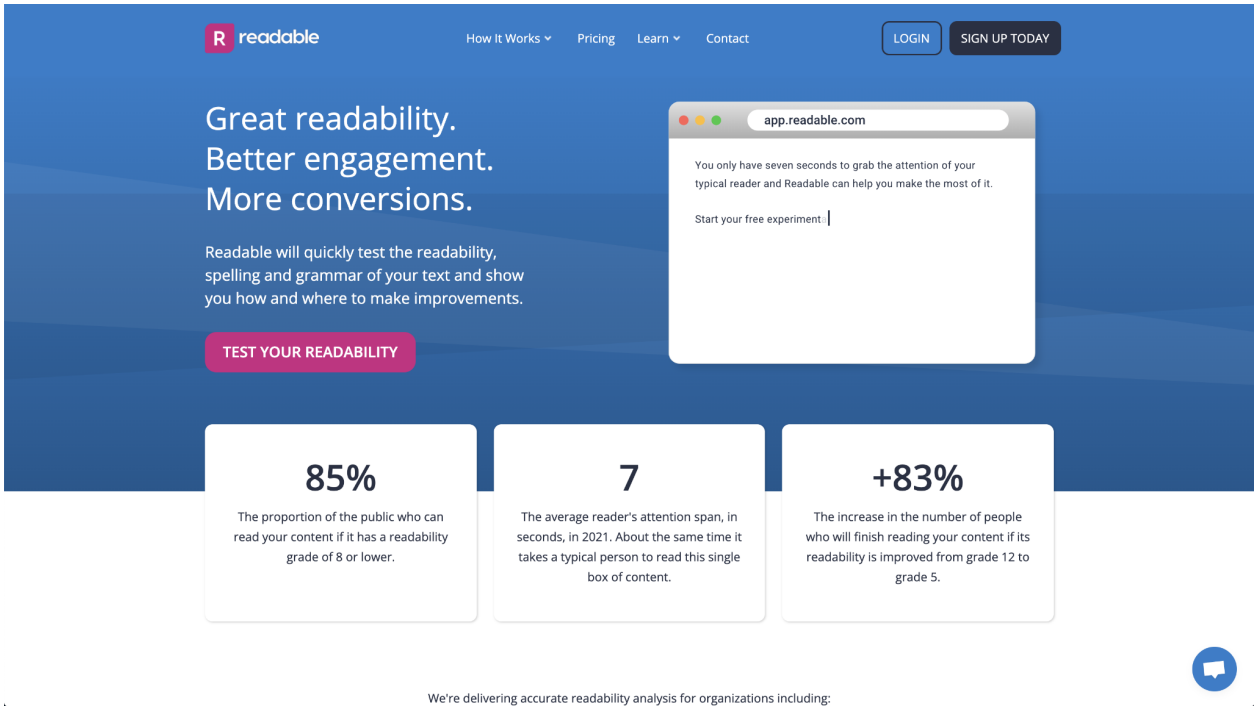

Рис. 1.2. Головна сторінка сервісу Readable

Головна сторінка дає можливість перейти на сторінку безпосередньої перевірки тексту. Безкоштовна версія дає можливість оцінити легкість читання за наступними параметрами:

- статистичні дані тексту (кількість слів, речень та символів);
- розрахункові значення формул оцінки легкості читання;
- вкладка із “проблемами” де перелічені усі довгі слова та речення, що ускладнюють читання тексту.

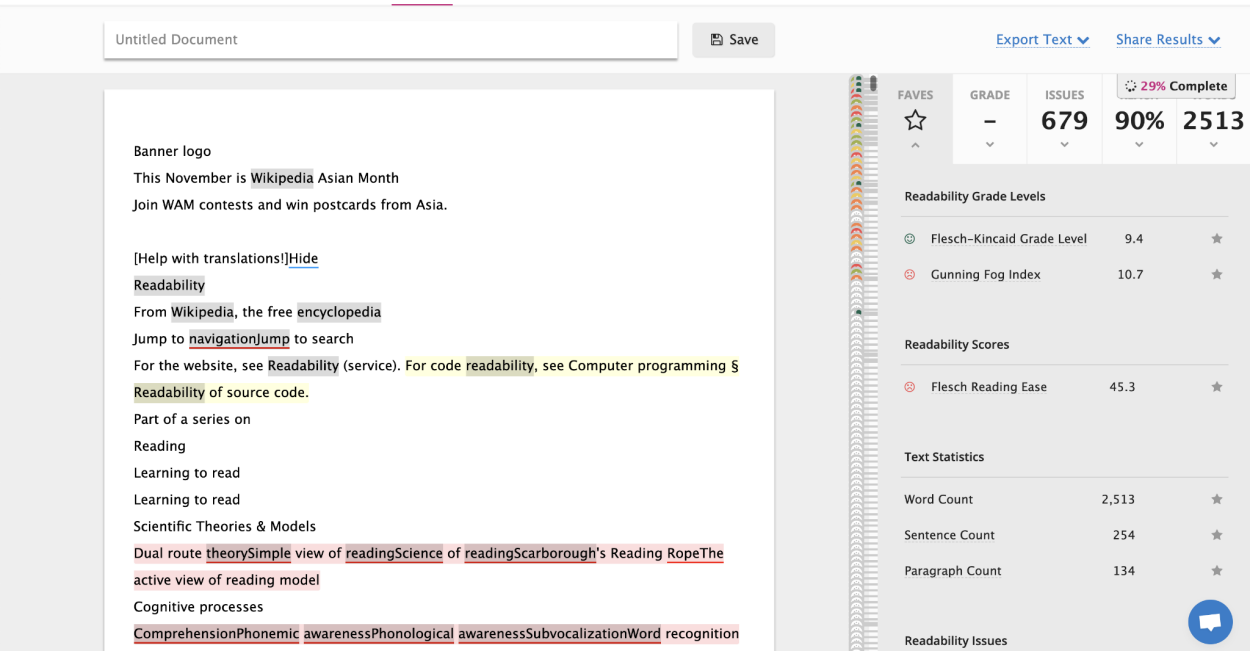

Рис. 1.3. Результат аналізу тексту у сервісі Readable

Окрім виведення результатів перевірки, сервіс, для зручності використання, має наступний додатковий функціонал:

- підсвітка складових тексту для легкості пошуку деталей проблеми, що розглядається;
- функціонал додавання параметрів до “Обраних”. Після того як користувач позначає той чи інший параметр за “обраний” він буде виводитись на самій першій вкладці сайдбару.

Загалом варто відзначити, що сервіс Readable надає можливість швидко та зручно перевірити читабельність тексту за різними показниками. Він зручний у використанні, має продуманий інтерфейс на надає можливості кастомізації під свої потреби.

1.9. Hemingwayapp – редактор та сервіс перевірки читабельності тексту

Один із найчастіше додатків для аналізу легкості читання тексту. На відміну від попередньо розглянутого, не є SAAS сервісом та не має платної моделі користування. Повністю безкоштовний додаток має інтерактивний редактор з підсвіткою проблемних частин, дає рекомендації по покращенню тексту.

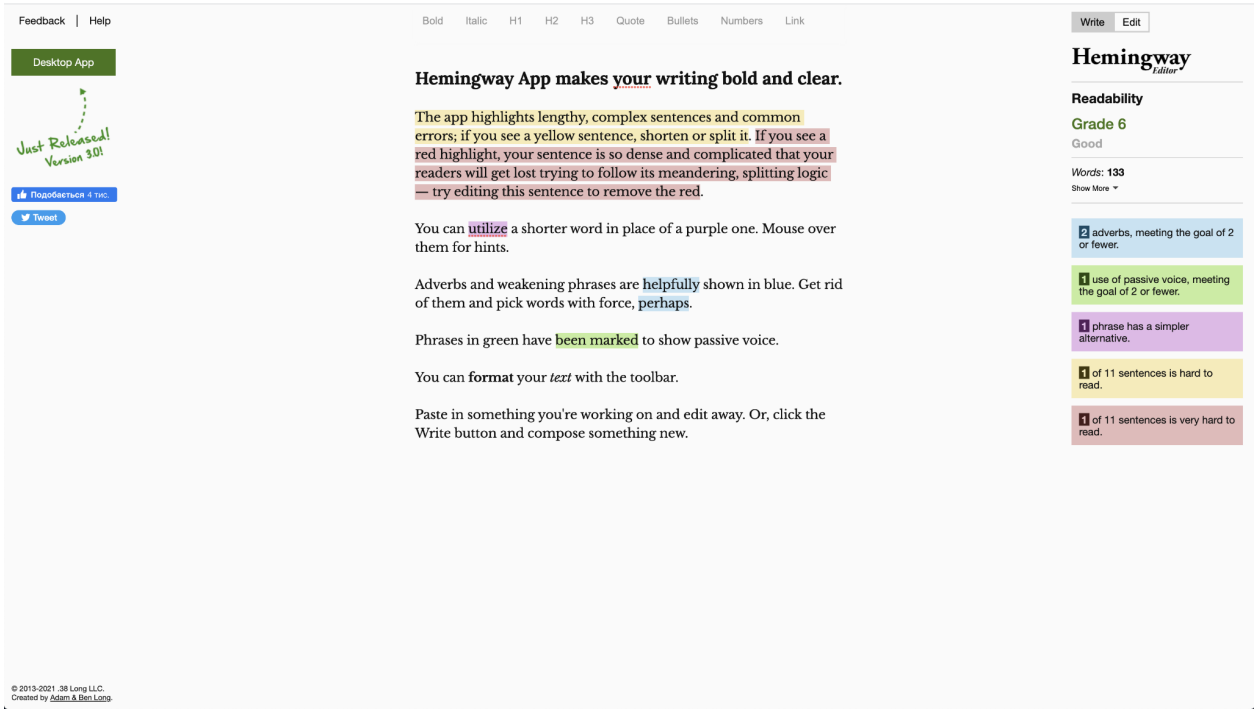

Рис. 1.4. Hemingwayapp – сервіс перевірки читабельності тексту

Має зручний інтерфейс та функціонал автоматичної перевірки при зміні тексту. Окремо варто відзначити наявність альтернативної версії у вигляді нативного додатку. Має окремі версії для Windows та Mac OS.

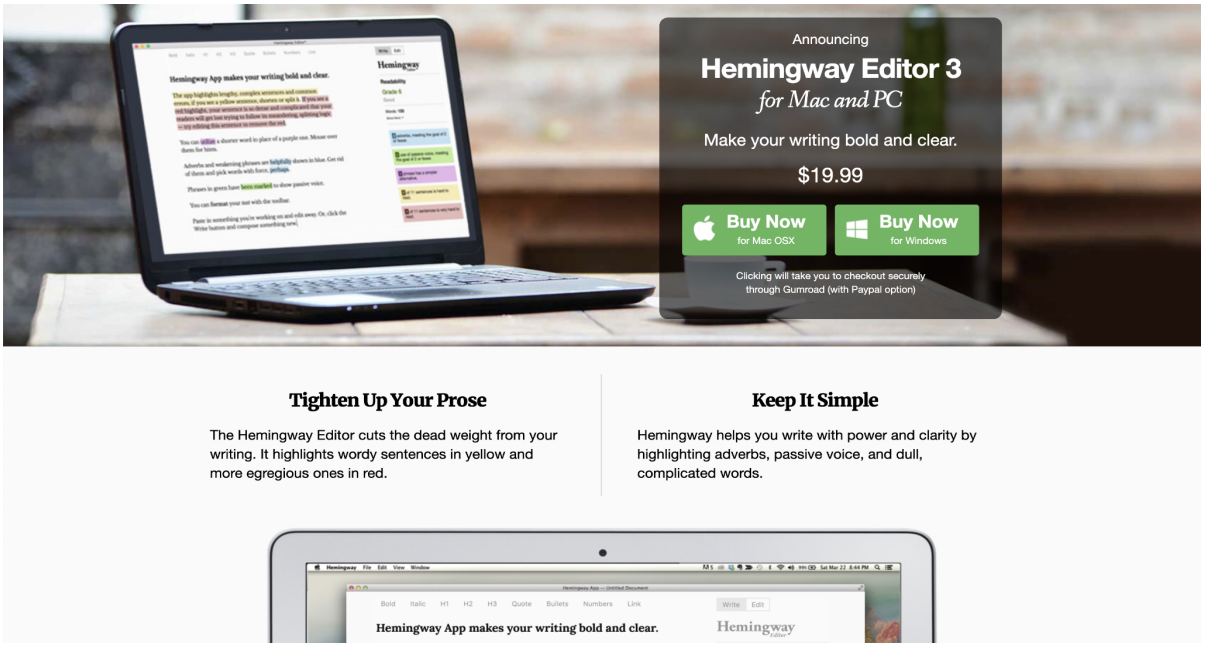

Рис. 1.5. Hemingwayapp – продажна сторінка нативної версії додатку

Саме нативна версія додатку і є основним способом монетизації сервісу. Вона надає такі переваги:

- можливість перевірки тексту без підключення до мережі;
- форматування тексту;
- публікація тексту напряму на популярну платформу для ведення блогів Medium та сайтів створених на базі CMS WordPress;
- експорт результатів написання тексту у HTML та docx форматах.

Загалом варто відзначити, що додаток має дуже зручний, але лімітований функціонал. Безумовно, рекомендації що він надає дають можливість покращити легкість читання тексту, але його можливості та функціонал в значній мірі програють можливостям попередньо розглянутого сервісу Readable.

1.10. Readability.io – онлайн-сервіс перевірки параметрів читабельності тексту

Readability.io – ще один цікавий сервіс перевірки читабельності. Має простий інтерфейс, можливість аналізу тексту веб-сторінки, та безкоштовну модель використання. Окремо надає програмний інтерфейс для інтеграції в свої додатки.

Сервіс розраховує наступні показники:

- формула Flesch-Kincaid;
- індекс Колман-Лиау;
- Automatic Readability Index SMOG (Simple Measure of Gobbledygook);
- формула Дейла-Чейла.

Окрім того, сервіс обраховує загальні статистичні показники тексту, зокрема:

- кількість букв;
- кількість слів;
- кількість речень;
- відсоток використання “важких” слів та складових тексту.

Сервіс вважає, що саме ці показники дають можливість оцінити рівень важкості сприйняття тексту.

Інтерфейс сервісу зручний у використанні, але немає локалізацій під англійську, українську або будь-які інші мови крім російської. До недоліків сервісу

також можна віднести те, що відсутній інтерактивний редактор, підказки із виділенням проблемних частин тексту.

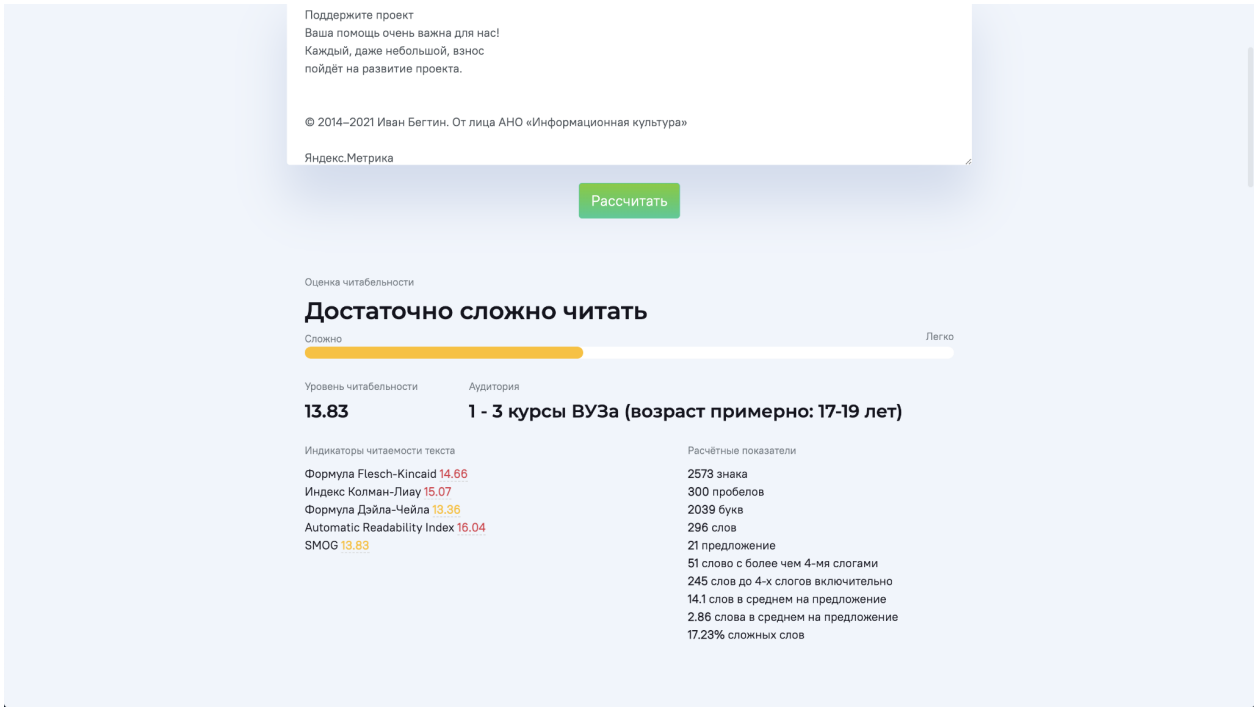

Рис. 1.6. Результат перевірки тексту у сервісі Readability.io

1.11. Загальна характеристика алгоритмів інтелектуального аналізу інформаційних ресурсів

Алгоритми аналізу тексту не зупинились на аналізі складності сприйняття тексту. Перші кроки в аналізі, що будувався на основі статистичних даних, стали скоріше додатковими аніж основними факторами при аналізі, поступившись сучасним алгоритмам NLP.

NLP (Natural Language Processing – обробка природної мови) — це галузь, що поєднує лінгвістику, обчислення, а також штучний інтелект. Правильне розуміння природної мови має вирішальне значення для широкого спектру програм, зокрема:

- віртуальних помічників;
- чат-ботів;
- голосових помічників;

Ці програми, як правило, включають процесор природної мови, метою якого є вилучення взаємодій і намірів, а також пов’язаної інформації та метаданих із частини простої природної мови та перетворення їх у те, що може обробляти машина.

Обробка природної мови - важлива проблема що стоїть перед розробниками величезної низки типів додатків. Перші спроби проаналізувати текст були зроблені ще у далекі 1950-і роки, на основі створеного алгоритму розробники планували створити програму для автоматичного перекладу тексту. Експеримент на жаль не вдався, його фінансування було значно скорочено, але на його основі майбутні розробники досягали кращих результатів.

Зокрема у 1960-х розробник із MIT створив програму “SHRDLU” - одну із перших розробок в сфері NLP. У програмі користувач веде розмову з комп’ютером, переміщуючи об’єкти, називаючи колекції та запитуючи стан спрощеного «світу блоків», по суті, віртуального боксу, заповнення різними блоками.

```
Person: PICK UP A BIG RED BLOCK.
Computer: OK. (does it)
Person: GRASP THE PYRAMID.
Computer: I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
Person: FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX.
Computer: BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.
Computer: OK. (does it)
Person: WHAT DOES THE BOX CONTAIN?
Computer: THE BLUE PYRAMID AND THE BLUE BLOCK.
Person: WHAT IS THE PYRAMID SUPPORTED BY?
Computer: THE BOX.
Person: HOW MANY BLOCKS ARE NOT IN THE BOX?
Computer: FOUR OF THEM.
Person: IS AT LEAST ONE OF THEM NARROWER THAN THE ONE WHICH I TOLD YOU TO PICK UP?
Computer: YES, THE RED CUBE.
Person: IS IT SUPPORTED?
Computer: YES, BY THE TABLE.
Person: CAN THE TABLE PICK UP BLOCKS?
Computer: NO.
Person: CAN A PYRAMID BE SUPPORTED BY A BLOCK?
Computer: YES.
Person: CAN A PYRAMID SUPPORT A PYRAMID?
Computer: I DON'T KNOW.
Person: STACK UP TWO PYRAMIDS.
Computer: I CAN'T.
Person: WILL YOU PLEASE STACK UP BOTH OF THE RED BLOCKS AND EITHER A GREEN CUBE OR A PYRAMID?
Computer: OK. (does it)
Person: WHICH CUBE IS SITTING ON THE TABLE?
Computer: THE LARGE GREEN ONE WHICH SUPPORTS THE RED PYRAMID.
Person: IS THERE A LARGE BLOCK BEHIND A PYRAMID?
Computer: YES, THREE OF THEM: A LARGE RED ONE, A LARGE GREEN CUBE, AND THE BLUE ONE.
Person: PUT A SMALL ONE ONTO THE GREEN CUBE WHICH SUPPORTS A PYRAMID.
Computer: OK. (does it)
Person: PUT THE LITTLEST PYRAMID ON TOP OF IT.
Computer: OK. (does it)
Person: DOES THE SHORTEST THING THE TALLEST PYRAMID'S SUPPORT SUPPORTS SUPPORT ANYTHING GREEN?
Computer: YES, THE GREEN PYRAMID.
Person: WHAT COLOR IS IT?
Computer: BY "IT", I ASSUME YOU MEAN THE SHORTEST THING THE TALLEST PYRAMID'S SUPPORT SUPPORTS.
Computer: RED
Person: HOW MANY THINGS ARE ON TOP OF GREEN CUBES?
Computer: I'M NOT SURE WHAT YOU MEAN BY "ON TOP OF "IN THE PHRASE "ON TOP OF GREEN CUBES " .
 DO YOU MEAN:
```

Рис. 1.7. Приклад використання програми “SHRDLU”

1970-і ознаменувався початком розробки додатків що імплементують концепцію онтології - формалізацію назв та іменувань. Окрім того, почалась розробка перших чат-ботів.

1980-ті та початок 1990-х ознаменували розквіт символічних методів у НЛП. Основними галузями того часу були дослідження аналізу на основі правил (наприклад, розробка HPSG як обчислювальної операціоналізації генеративної граматики), морфології (наприклад, дворівнева морфологія), семантики (наприклад, алгоритм Леска) та інших областях розуміння природної мови (наприклад, у теорії риторичної структури). Інші напрямки досліджень були продовжені, наприклад, розробка чат-ботів з Racter і Jabberwacky.

Загалом усі ці спроби призвели до активного розвитку науки NLP та наявності низки рішень із відкритим вихідним кодом, що доступні для використання при розробці.

На основі цих відкритих рішень будуються важливі продукти, що вирішують різноманітні питання.

1.12. Використання NLP для розробки чат-ботів

У сучасному світі чат-боти використовуються повсякденно. Велики корпорацію розробляють свої рішення з метої зменшити навантаження на відділення підтримки користувачів, адже чим менше звернень - тим менша кількість співробітників потрібна. Окрім того, чат-боти стають в нагоді й при формуванні замовлень або залишення заявок, тобто на будь-якому з етапів воронки продажів. Факт використання чат-ботів у сфері комерції дає можливість їм, а на рівні з ними й NLP, розвиватися шаленими темпами.

Чат-боти поділяють на дві групи, в залежності від принципу ідентифікації клієнтського засобу та відповіді на нього. Зокрема відрізняють наступні типи.

- Чат-боти, що працюють за попередньо сформованим обмеженим переліком ключових слів. Зазвичай їх значно легше створювати, адже вони не

потребують значних зусиль. Із мінусів варто відзначити функціонал, що обмежується початковими налаштуваннями.

- Чат-боти побудовані на основі штучного інтелекту на NLP. Цей різновид чат-ботів значно важчий у розробці. Вони здатні до самонавчання та накопичення знань на основі попередніх взаємодій із користувачами.

Загальна схема роботи чат-ботів виглядає наступним чином:

1. одержання запиту від клієнта;
2. розбір запиту - розуміння висловлювання та визначення намірів клієнта в контексті його бізнес-кейсу;
3. виконання дій згідно із заздалегідь визначеним сценарієм (скриптом) з обробки клієнтського кейсу;
4. генерація відповіді природною мовою;
5. збереження запиту, контексту та параметрів діалогу для обробки наступних звернень;
6. надсилання відповіді клієнту.

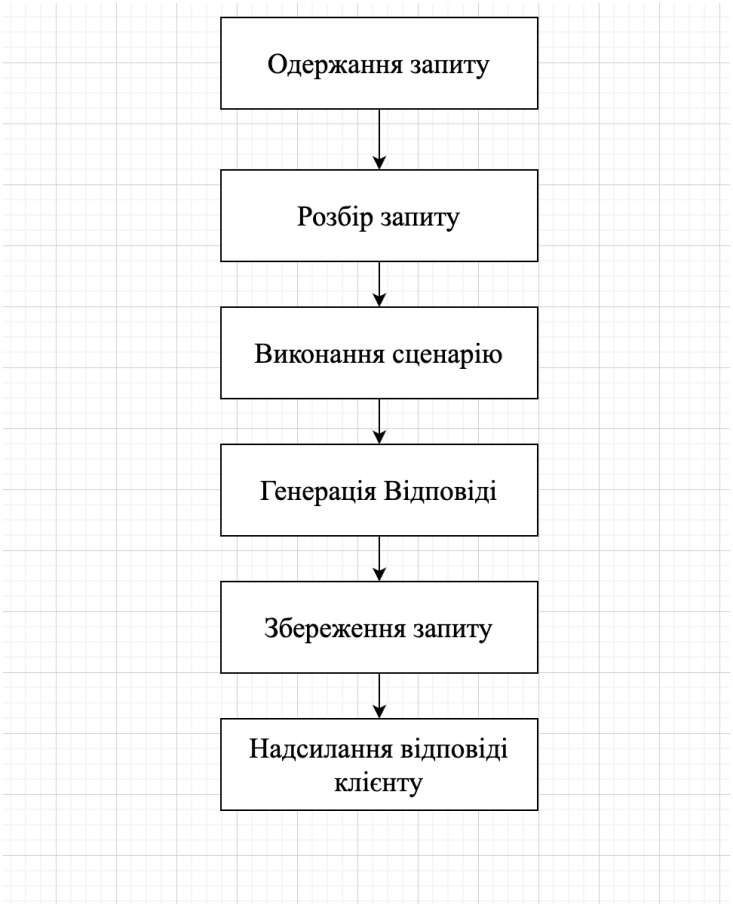

Рис. 1.8. Типова блок-схема роботи чат-ботів

Найбільш складним етапом роботи є аналіз клієнтського запиту. Чат-боти, що самонавчаються, на базі Machine Learning використовують для цього методи NLU і NLP. Наприклад, для текстових чат-ботів процес аналізу включає наступні етапи:

- попередня обробка тексту – токенізація (розбиття на слова), виправлення друкарських помилок, лематизація та стемінг (визначення нормальної форми слів та частин мови), відкидання стоп-слів (артиклі, вигуки, спілки та ін.), розширення запиту за допомогою словників синонімів, додаток інформацією про значимість окремих слів, розширення запиту деревом синтаксичного аналізу та результатами дозволу займенників, і навіть визначення іменованих сутностей;
- класифікація запиту на основі прикладів фраз та ML-алгоритмів або формальних правил (шаблонів); ранжування гіпотез класифікації відповідно до поточного контексту розмови;
- вилучення параметрів запиту із фрази користувача.

Цікаво, що приблизно до 2014 року при розробці чат-ботів переважно використовувався rule-based підхід (підхід на основі формальних правил), суть якого полягає у виділенні семантично значущих елементів фраз та їх кодифікації.

Далі на основі цих результатів створювали сценарії діалогів за допомогою скриптових мов програмування, наприклад PHP, або Javascript.

Однак, після 2015 року розвиток алгоритмів семантичної близькості текстів, технологій синтезу та розпізнавання мови, призвело до поширення нових підходів до класифікації текстів та навчання NLU-систем.

Зокрема, тут варто відзначити WikiMatrix та CCMatrix – величезні датасети для NLP-задач та машинного перекладу, що базується на визначенні семантичної близькості словосполучень у різних мовах.

Таким чином, більшість сучасних чат-ботів влаштовані на базі останніх досягнень у галузі Data Science: NLU та NLP-техніки, методи розпізнавання мови та обробки текстів за допомогою нейромереж та інших інструментів штучного інтелекту.

1.13. Використання NLP для при розробці алгоритмів пошукових систем мережі інтернет

Інтернет сьогодні – це величезна кількість інформації. Щоденно створюється величезна кількість інформаційних ресурсів, порталів, відео та картинок.

Але усе це різноманіття інформацію не має цінності, у випадку якщо користувач не має до нього доступу. Задля того, щоб переглянути веб-сторінку - потрібно мати її адресу. Але звісно, застарілі довідники із переліком веб-сайтів аж ніяк не можуть слугувати джерелом інформації щодо адрес шуканих веб-ресурсів, релевантних тій чи іншій темі.

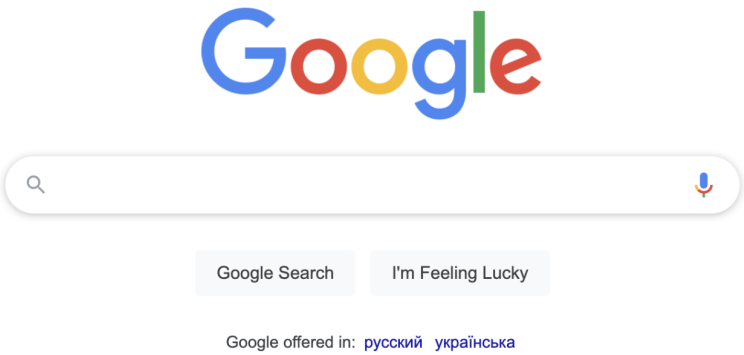

Рис. 1.9. Головний екран пошукової системи “Google”

На допомогу приходять пошукові системи. Ми настільки до них звикли що й не помічаємо як вони непомітно, але безперервно, супроводжують наші інтернет активності. Саме вони слугують відправною точкою при пошуку інформації в інтернеті. Працюють вони за простим принципом.

1. Користувач відкриває веб-сторінку пошукової системи.
2. Користувач заповнює форму, задаючи пошукові параметри (найчастіше вона складається із одного поля - пошукового запиту).

3. Користувач відправляє форму та переадресовується на сторінку із результатами пошуку.

Рис. 1.10. Типова блок-схема роботи пошукової системи

Але як при цьому надавати користувачу максимально релевантні його пошуковому запиту відповіді? За яким принципом сортувати результати пошуку?

Окрім того, важливо щоб пошук не займав великої кількості часу. Користувач може не дочекатись результатів пошуку і покинути пошукову у систему у випадку, якщо підбір результатів пошуку займатиме значний період часу.

Сучасні пошукові системи працюють дуже швидко. Зазвичай, користувач отримує результати пошуку за лічені секунди. Це досягається шляхом комбінації багатьом технологій. Зокрема, використовується і NLP.

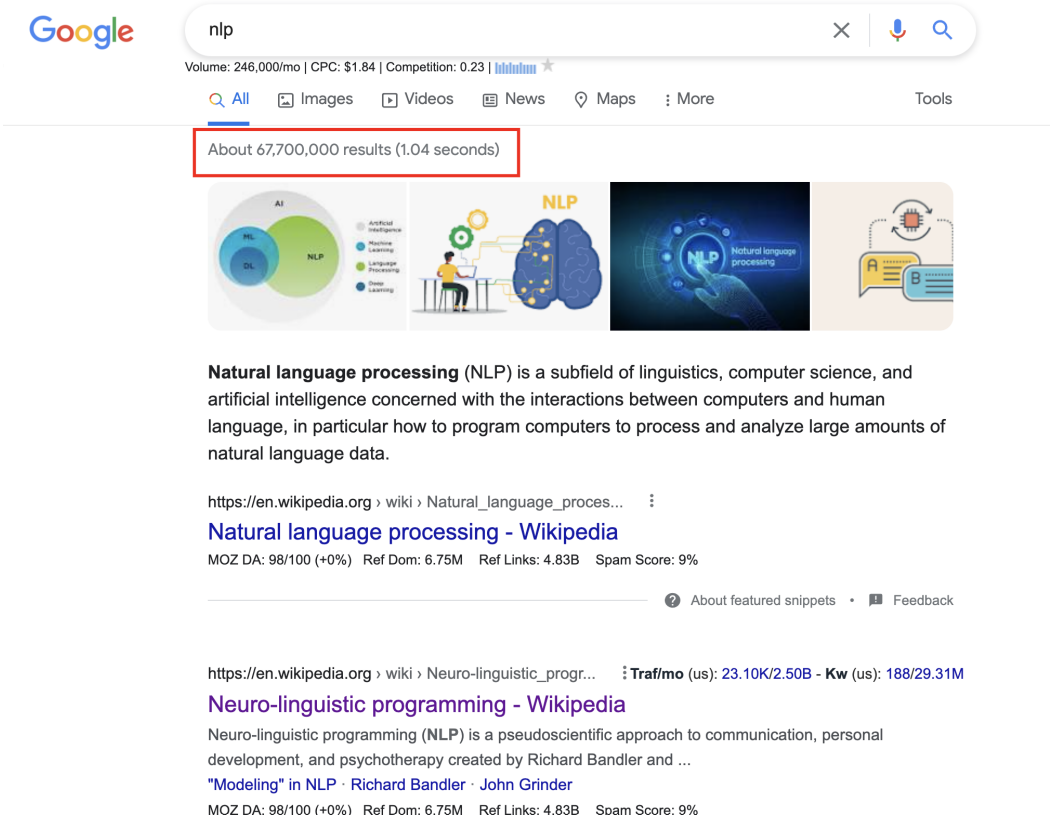

Рис. 1.11. Приклад результатів пошуку із виділеними показниками часу, що зайняв пошук, та кількістю результатів

Перед пошуковими системами стоїть нелегке завдання. Серед усієї кипи веб-сторінок потрібно обрати саме ті, що максимально релевантні до пошукових запитів клієнтів. Але ж які бувають типи пошукових запитів?

Прийнято розподіляти пошукові запити за їх пошуковими намірами, зокрема виділяють наступні:

- інформаційний;
- транзакціонний;
- комерційний.

Кожен із перелічених різновидів пошукових запитів потребує окремого типу веб-ресурсів на сторінці із результатами.

Алгоритми пошукових систем розвивались паралельно із розвитком усіх технологій. На початку своєї історії більшість із них виконувала пошук за простим алгоритмом - у випадку якщо шукана фраза міститься в тексті – текст релевантний.

Знаючи про цей принцип пошукові оптимізатори (люди, що працюють над покращенням позицій веб-ресурсів на сторінках із результатами пошуку) почали ним зловживати, додаючи на сторінку величезне полотно беззмістовних ключових слів, що не несли жодної користі користувачам.

Звісно подібна поведінка власників ресурсів змусила пошукові системи переглянути свої алгоритми, адже користувачі були в значній мірі незадоволені результатами пошуку, адже не отримували релевантних результатів.

Крім усіх інших засобів боротьби були покращенні алгоритми пошуку. Зокрема вони перестали працювати за принципом простого пошуку, а почали приймати до увагу зв'язок між словами, словоформи та релевантні ключові слова. Усе це стало можливим завдяки активному розвитку NLP – технологій обробки природної мови.

Пошук перестає працювати за принципом “ключових слів” та все більше базує свої результати за принципами “семантичного пошуку”.

Семантичний пошук – це техніка пошуку даних, у якій пошуковий запит має на меті не тільки знайти ключові слова, але й визначити намір і контекстне значення слів, які людина використовує для пошуку.

Разом із впровадженням та реалізацією ідеї семантичного пошуку актуальним стало поняття LSI слів. Воно набуло широкої популярності в колі пошукових оптимізаторів на маркетологів.

LSA (латентний семантичний аналіз/індекс) — це техніка в обробці природною мовою аналізу відносин між набором документів та термінами, які вони містять, шляхом створення набору понять, пов’язаних з документами та термінами.

LSI заснований на принципі, що слова, які вживаються в одному контексті, як правило, мають подібне значення. Ключовою особливістю LSI є його здатність витягувати концептуальний зміст тексту шляхом встановлення асоціацій між термінами, які зустрічаються в подібних контекстах.

Пошукові системи почали формувати результати пошуку базуючись не тільки на принципі наявності ключового слова в тексті, а й на переліку LSI слів. Паралельно додавши фільтри від веб-сайтів, що не містять корисної інформації та зловживають ключовими словами їм вдалось в значній мірі покращити результати пошуку, як з

точки зору якості вебресурсів, так і з точки зору відповідності самих ресурсів намірам користувача та його пошуковим запитом.

LSI додає важливий крок до процесу індексування, аналізу, та пошуку документів, він досліджує набір документів, щоб побачити, які із них максимально релевантні. LSI вважає документи, які мають багато спільних слів, семантично близькими, а документи з меншою кількістю слів — менш близькими.

Для формування переліку LSI слів потрібно проаналізувати значні об'єми даних, саме тому сучасні пошукові системи використовують великі серверні потужності в своїй роботі.

Окрім пошуковий систем NLP може бути використаний для реалізації функціоналу пошуку у інтернет магазині.

1.14. Використання NLP для при розробці алгоритмів пошуку в інтернет-магазинах

Інтернет-магазини сьогодні користуються шаленою популярністю. Можливість обрати речі не виходячи за межі житла, зручно порівнювати та читати відгуки – усе це щоденно приваблює величезну кількість користувачів до вітрин інтернет ресурсів.

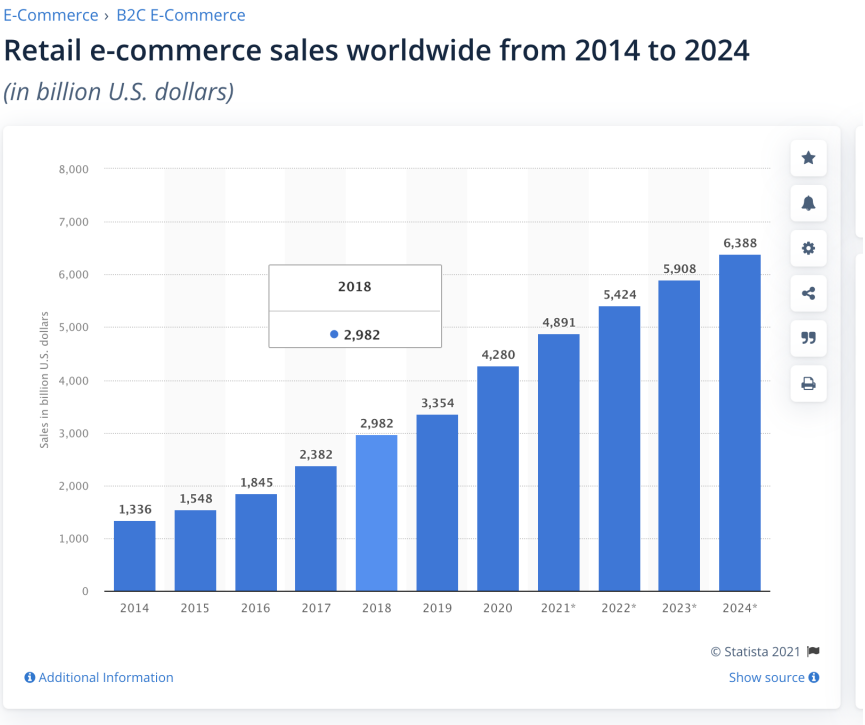

Рис. 1.12. Графік витрат людей у онлайн-магазинах

Статистика відзначає щорічний ріст об'єму коштів котрі люди витрачають в інтернет магазинах, що свідчить про активний розвиток цієї індустрії.

В свою чергу активний ріст провокує й активну розробку нових рішень, як для покращення тунелів продажів і для покращення показників кінцевої конверсії, так і для покращення зручності інтерфейсу користувача.

Один із важливих складових будь-якого інтернет магазину є функціонал пошуку. Маючи великий асортимент зручний пошук - обов'язкова складова, адже коли користувач має на меті придбання конкретного товару, змушувати його шукати продукт власноруч – погана ідея.

Добре коли користувач точно знає що шукає та має на руках назву товару. Причому, назва повинна повністю співпадати з назвою, що міститься на сайті. Але що як точної назви користувач не знає, або переплутав порядок слів, помилився літерою чи взагалі не знає конкретної назви.

Саме в цей момент бізнес онлайн-продажів потребує NLP алгоритмів, що сформують до кожного товару LSI слова, та нададуть можливість користувачу шукати не тільки за прямим входженням в назву продукту, а й за релевантними словами. Це в значній мірі покращує користувацький досвід, конверсії та збільшує оберт бізнесу.

Один із готових рішень multisearch.io декларує, що в середньому його клієнти побачили ріст кількості продажів на 30%, що свідчить про безумовно користь подібних алгоритмів.

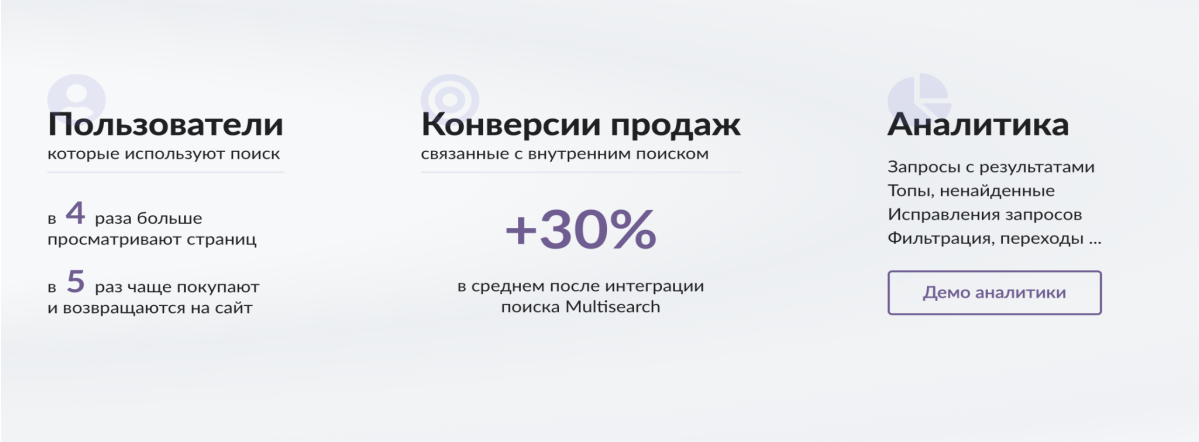

Рис. 1.13. Покращення показників при додаванні функціоналу розумного пошуку згідно із статистикою multisearch.io

1.15. Розмічування частин мови

Важлива складова NLP, відповідає за аналіз тексту в розрізі частин мови. Текст розбивається на речення, у котрих в свою чергу для кожного слово визначається граматична форма. Аналіз відбувається на основі контексту слово та попередньо нетренованої моделі.

Складність полягає у тому, що для розмічування частин мови недостатньо мати перелік типу ключ-значення, адже кожне окреме слово може приймати різне значення в залежності від контексту його використання.

Частини мови визначені в NLP дещо відрізняються від тих, що визначаються лінгвістикою. Їх набір залежить від обраної моделі. Більша кількість частин мови зумовлена складністю розробки алгоритму розмітки при використанні стандартних частин мови.

1.16. Розпізнавання іменованих сутностей

Процес розпізнавання іменованих сутностей (NER) - важливий етап в процесі інтелектуального аналізу тексту. Його суть полягає у розпізнаванні в не структурованому тексті іменовані сутності (як от: персона, дати, адреса, роки, організації та інші).

Існує велика кількість NER платформ, зокрема популярності набули:

- GATE;
- OpenNlp;
- SpaCy.

Алгоритм роботи NER складається із двох основних кроків:

- розпізнавання назв;
- класифікація розпізнаних назв.

NER підходить для будь-якої ситуації, в якій корисний огляд на високому рівні великої кількості тексту. За допомогою NER можливо з першого погляду зрозуміти тему або сенс тексту та швидко згрупувати тексти за їх релевантністю чи схожістю.

Деякі відомі випадки використання NER включають:

- категоризація резюме – прискорити процес найму, узагальнивши резюме кандидатів; покращити внутрішні робочі процеси, класифікуючи скарги та запитання співробітників;
- підтримка клієнтів – можна легко зменшити час відповіді, розділивши запити, скарги та запитання користувачів за категоріями та фільтруючи за пріоритетними ключовими словами;
- пошукові та рекомендаційні системи – підвищує швидкість та релевантність результатів пошуку та рекомендацій, узагальнивши описовий текст, огляди та обговорення;
- класифікація вмісту – простіше відображати вміст і отримувати уявлення про тенденції, визначаючи теми і назви публікацій у блозі та новинних статей;
- охорона здоров'я – покращує стандарти догляду за пацієнтами та зменшує робоче навантаження, витягуючи важливу інформацію з лабораторних звітів;
- сфера навчання – дозволяє студентам та дослідникам швидше знаходити відповідний матеріал, узагальнюючи документи та архівний матеріал і виділяючи ключові терміни, теми та теми. Цифрова платформа ЄС для культурної спадщини Europeana використовує NER, щоб зробити історичні газети доступними для пошуку.

1.17. Аналіз “настрою” тексту

Аналіз настрою тексту - невід'ємна складова природного аналізу тексту. Повсякчас використовується з метою оцінки:

- клієнтський відгуків та форм зворотного зв'язку;
- соціальних мереж;
- онлайн-ресурсів;
- медицинських даних.

В поєднанні із іншими технологіями допомагає сформувати розуміння аудиторії по тому чи іншому питанню. Зокрема прикладом може слугувати збір політичної інформації за наступним алгоритмом:

- збір географічних даних користувачів соціальних мереж;

- аналіз їх соціальної активності;
- розпізнавання іменованих сутностей та фільтр по тим сутностям, які аналізуються;
- складання результуючого звіту із сегментованому аудиторіями та показниками “настрою” по аналізованій тематиці.

Таким чином аналіз “настрою” тексту є невід'ємною складової природного аналізу тексту та відіграє важливу роль при формуванні звітів.

1.18. Аналіз на граматичні помилки

Один із важливих пунктів при аналізі інформаційних ресурсів є перевірка коректності аналізованого тексту з точки зору правил граматики. Ця перевірка є невід'ємною складовою адже напряму впливає на скоринг аналізовано ресурсу, довіру до нього з точки зору аналізатора.

Формуючи правила перевірки граматики потрібно вважати наступне:

- різні мови мають різні правила;
- слова можуть писатись схоже одно до одного але мати різне значення.

Аналіз синтаксису — це другий етап процесу проектування компілятора, на якому введений рядок перевіряється на підтвердження правил і структури формальної граматики. Він аналізує синтаксичну структуру та перевіряє, чи введений вхід відповідає правильному синтаксису мови програмування чи ні.

Синтаксичний аналіз у процесі проектування компілятора відбувається після фази лексичного аналізу. Він також відомий як дерево синтаксису або синтаксичне дерево. Дерево розбору розроблено за допомогою попередньо визначеної граматики мови. Синтаксичний аналізатор також перевіряє, чи відповідає дана програма правилам, передбаченими граматикою. Якщо це задовольняє, синтаксичний аналізатор створює дерево розбору цього тексту. В іншому випадку буде відображатися повідомлення про помилку.

Перевірка орфографії запускає різні алгоритми для виправлення помилок. Цей алгоритм такий.

- Спочатку він сканує текст, щоб виділити окремі або пару слів.
- Це стосується лексем/слів зі списком слів у словнику.
- Якщо слова не збігаються з жодним словом, він запускає алгоритм редагування відстані, щоб запропонувати найближчі слова або список слів.

Подібно до алгоритмів перевірки орфографії, алгоритм перевірки граматики також витягує речення з тексту та звіряє кожне слово з реченням, переглядаючи інформацію, таку як частини мови, відповідно до розташування в реченні. Крім того, спираючись на кілька правил, алгоритм виявляє помилку в узгодженні часу, числі, порядку слів тощо.

Висновки до розділу 1

Основна мета розділу – це аналіз принципів та надання характеристик засобам інтелектуального аналізу інформаційних ресурсів мережі інтернет. Були описані основні характеристики існуючим сервісам для аналізу.

Розглянуті сервіси надали змогу сформулювати перелік вимог до розробленого додатку, зокрема:

- додаток повинен мати зручний інтерфейс;
- додаток повинен містити інтерактивний редактор;
- інтерактивний редактор повинен містити функціонал підсвітки складових тексту.

Крім того, були розглянути алгоритми природного аналізу тексту, перелічені їх техніки та можливості. Майбутній додаток повинен містити реалізацію використання таких можливостей, зокрема підтримувати:

- розпізнавання частин мови;
- розпізнавання сутностей.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ЗАСОБУ АНАЛІЗУ ІНФОРМАЦІЙНИХ РЕСУРСІВ

Метою цього розділу є аналіз, порівняння та вибір технологій та архітектури майбутнього додатку. Детальний аналіз та вибір платформи, архітектуру додатку загалом, контрактів взаємодії клієнт-серверної частини дасть змогу сформувати максимально зручні та комфортні умови для розробки та подальшої підтримки майбутнього додатку.

2.1. Аналіз платформ та форматів реалізації додатку

Існує велика кількість варіантів для реалізації додатку, усі вони можуть бути умовно розділені на наступні категорії за форматом використання:

- додатки із інтерфейсом у вигляді командного рядка;
- нативні додатки із графічним інтерфейсом;
- вебдодатки із графічним інтерфейсом.

2.2. Аналіз додатків із інтерфейсом у вигляді командного рядка

Додатки із інтерфейсом у вигляді командного рядка – найперший формат реалізації додатків. У ті далекі роки, коли ера комп'ютерів тільки починається, обчислювальних потужностей не вистачає для того, щоб відображати зручний графічний інтерфейс. Не зважаючи на те, що зараз ситуація змінилась та більшість користувачів та розробників надають перевагу новому формати із графічним інтерфейсом, програми з інтерфейсом командного рядка не втрачають свою популярність.

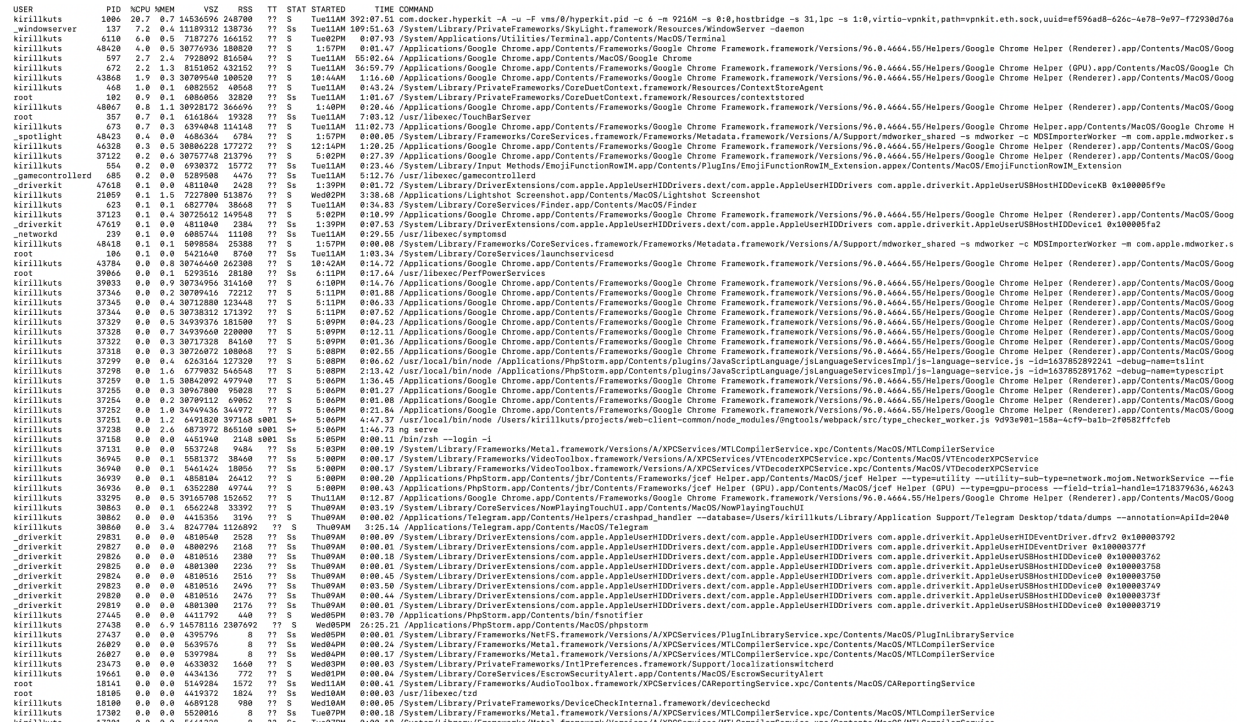

Рис. 2.1. Приклад використання додатку із CLI інтерфейсом

Додатки із інтерфейсом у вигляді командного рядка часто використовуються програмістами та системними адміністраторами, в інженерних та наукових середовищах, а також технічно досвідченими користувачами персональних комп’ютерів. CLI також популярні серед людей з вадами зору, оскільки команди та відповіді можна відображати за допомогою дисплеїв Брайля.

Програма, яка обробляє інтерфейс, називається інтерпретатором командного рядка або процесором командного рядка. Операційні системи реалізують інтерфейс командного рядка в оболонці для інтерактивного доступу до функцій або служб операційної системи.

2.3. Аналіз нативних додатків із графічним інтерфейсом

Нативними додатками називають додатки, що розроблені для використання на певній платформі. Зокрема, прикладами таких додатків можуть бути:

- браузери: Google Chrome, Edge, Safari;
- Microsoft Office;
- Adobe Photoshop.

Кожен із перелічених додатків має свою версію під певну операційну систему та платформу. Це зручно, адже надає можливості відкритої взаємодії із користувачем, операційною системою. Збереження файлів, доступ до інших процесів – усе це безумовні переваги розробки нативних додатів.

Але є і недоліки. Зокрема, розробка версії додатку під кожну окрему платформу займає велику кількість часу та ресурсів. Адже кожна операційна система має свої відмінності у взаємодії, програмного інтерфейсі та правах доступу.

Окрім того, на відміну від вебдодатків, нативні додатки повинні бути окремо інстальовані в систему. Також, нерідко виникають ситуації, коли для коректного функціонування того чи іншого додатку потрібно окремо встановити інший пакет, на основі якого функціонує встановлений додаток. Добре, коли при встановленні цього додаткового пакету не виникає додаткових проблем, але можливі й інші сценарії.

- Встановлена версія ОС не відповідає вимогам встановленого додатку.
- Вже встановлена інша версія цього ж самого додатку.
- Цей додаток був встановлений, але некорректно виделаний. Файли конфліктують.

Подібні сценарії створюють складнощі при поширені додатку, що призводить до погіршення показників конверсії. Окрім того, при розробці нативних додатків перед бізнесом постає питання ресурсів, для розробки під кожну платформу. У випадку, якщо бізнес обере шлях розробки під виключно одну платформу – різко зменшується число потенційних користувачів.

Окрім того, розроблюваний додаток потрібно постійно підтримувати та актуалізовувати відповідно до оновлень операційних систем.

Варто також зазначити, що сучасний світ розробки надає інструменти розробки додатків одразу під декілька платформ. Це реалізується шляхом портування функціоналу взаємодії із API операційної системи. Безумовною перевагою такого підходу є швидкість розроби, але є і мінуси. Зокрема, обмеження що накладаються самою платформою. Окрім того, саме портування інтерфейсу платформи сповільнює роботу додатку, що в свою чергу може призвести до гіршого досвіду користувача.

Однією з таким платформ для розробки є Electron. Він дозволяє розробляти додатки написані із використанням HTML, CSS, JS та портувати їх відразу під декілька платформ. Даний інструмент користується попитом серед різноманітних проектів, зокрема із використанням цього інструменту написані додатки:

- Visual Studio Code;
- Facebook Messenger;
- Twitch;
- Slack;
- Figma.

Довіра таких серйозних компаній свідчить про надійність та стабільність роботи. Самий інструмент має відкритий вихідний код. Також до переваг можна віднести низький поріг входу та легкість налаштування.

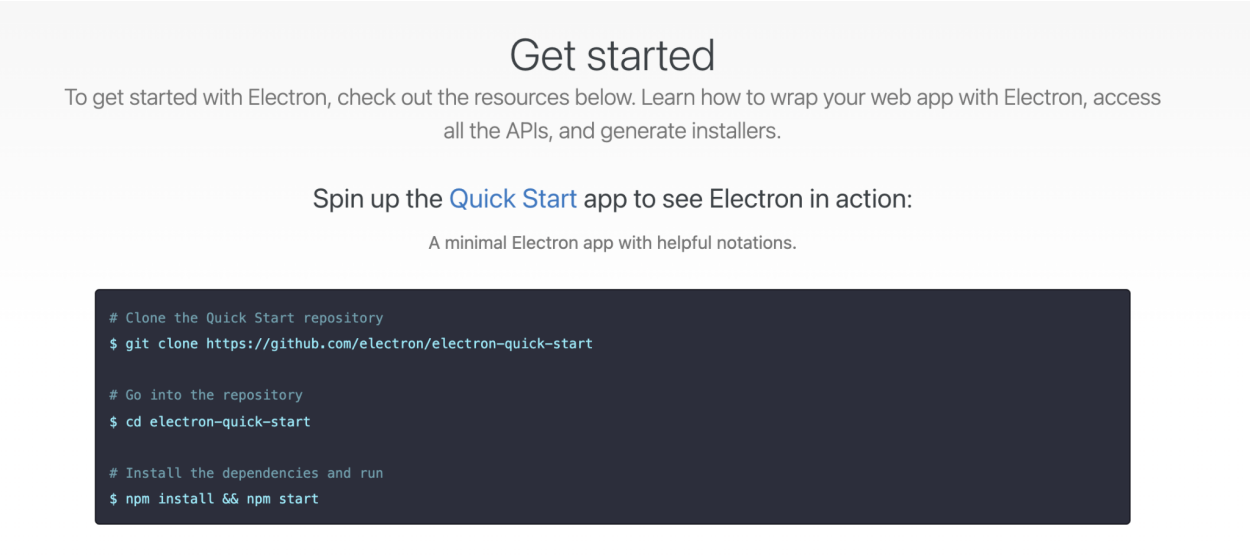

Рис. 2.2. Налаштування Electron перед початком розробки

2.4. Аналіз веб-додатків із графічним інтерфейсом

Веб-додатки стають набирають значної популярності з роками. Вперш за все, це пов'язано з їх основною особливістю - кросплатформеність. Для роботи веб-додатку потрібен тільки браузер, котрий встановлений на більшості сучасних пристроїв за замовчуванням.

Веб-додатки це ті самі вебсайти, але з більшою кількістю бізнес-логіки. Її імплементація в браузерному середовищі стала можливою завдяки активному розвитку мови програмування JavaScript.

JavaScript (JS) — це легка, інтерпретована або скомпільована в режимі реального часу мова програмування. Хоча вона й найбільш відома як мова сценаріїв веб-сторінок, багато не браузерних середовищ також використовують її, наприклад Node.js, Adobe Acrobat, Apache CouchDB.

JavaScript — заснована на прототипах, багатопарадигмальна, однопотокова динамічна мова, яка підтримує об’єктно-орієнтований, імперативний та декларативний (наприклад, функціональне програмування) стилі.

Мова програмування JavaScript активно розвивається. Стандартами для JavaScript є специфікація мови ECMAScript (ECMA-262) і специфікація API інтернаціоналізації ECMAScript (ECMA-402).

Саме мова JavaScript і слугує тим інструментом, на основі котрого й формується клієнтська частина веб-додатку. Окрім того, при розробці клієнтської частини використовуються такі інструменти як:

- HTML (Hyper Text Markup Language) - мова розмітки;
- CSS (Cascading Style Sheets) - мова створення стилів.

Загалом, будь який веб-додаток має дві основні складові.

- Клієнтська частина: виконується безпосередньо в браузері. Відповідальна за коректне відображення інформації, взаємодії із користувачем, отримання та передачу інформації до серверу.
- Серверна частина: відповідальна за отримання інформації від клієнта, збереження інформації у Базі Даних, її коректне оновлення та передачу до клієнта.

Взаємодія між наведеними складовими веб-додатків може відбуватись у різних форматах. Найчастіше обирають HTTP API для передачі та отримання даних до клієнта (або іншого додатку). Клієнт формує потрібен запит, обираючи заздалегідь погоджену адресу, тип методу та параметри. А сервер, у свою чергу, отримавши запит, обробляє його, модифікуючі та зберігаючі отримані дані, надсилає результат роботи назад до клієнта.

HTTP API може бути побудований на основі різних типів архітектур. Сучасні веб-додатки найчастіше використовують одну із двох найпопулярніших моделей:

- RESTful API;
- GraphQL API.

Кожна з них має свої переваги та недоліки, але важливо притримуватись обраного курсу та розробляти базуючись на одному й тому самому типі архітектури.

REST API (також відомий як RESTful API) — це інтерфейс програмування прикладних програм (API або веб-API), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-сервісами RESTful. REST розшифровується як репрезентаційна передача стану і була створена вченим-комп'ютерником Роєм Філдінгом.

REST — це набір архітектурних обмежень, а не протокол чи стандарт. Розробники API можуть реалізувати REST різними способами.

Коли запит клієнта здійснюється через RESTful API, він передає стан ресурсу запитувачу. Ця інформація надається в одному з кількох форматів через HTTP:

- JSON (нотація об'єктів Javascript);
- HTML;
- XLT;
- звичайний текст.

JSON є найбільш популярним форматом обміну даними, оскільки, незважаючи на свою назву, він не залежить від мови, а також він легко читається людьми і комп'ютерами.

Окрему важливу роль у RESTful моделі API відіграють заголовки HTTP запитів. Вони зберігають важливу службову інформацію, як от токен авторизації, метадані запиту, ідентифікатор ресурсу. Також заголовки можуть бути використані для імплементації логіки кешування результатів – однієї із стратегій оптимізації швидкодії роботи додатку.

Оскільки REST – це методологія, а не конкретний інструмент розробки, є певні рекомендації та правила, при використанні котрих реалізованих програмний інтерфейс вважається RESTful:

- Архітектура клієнт-сервер: складається з клієнтів, серверів і ресурсів, із запитами, які керуються через HTTP.
- Зв’язок клієнт-сервер без стану: інформація про клієнта не зберігається між запитами на отримання, і кожен запит є окремим і не підключеним.
- Єдиний інтерфейс між компонентами, щоб інформація передавалася у стандартній формі. Для цього потрібно:
    - запитувані ресурси є ідентифікованими та відокремленими від представлень, надісланих клієнту.
    - Клієнт може маніпулювати ресурсами за допомогою представлення, яке вони отримують, оскільки подання містить достатньо інформації для цього.
    - Повідомлення, що повертаються клієнту, мають достатньо інформації, щоб описати, як клієнт повинен її обробити.
- Багатошарова система, яка організовує кожен тип серверів (відповідальних за безпеку, балансування навантаження тощо), передбачала отримання запитуваної інформації в ієрархії, невидимі для клієнта.
- Code-on-demand (опціонально): можливість надсилати виконуваний код із сервера клієнту, коли його запитують, що розширює функціональні можливості клієнта.

Хоча REST API має відповідати цим критеріям, що дещо ускладнює процес розробки, він все ще вважається легшим у використанні, ніж альтернативи, як наприклад SOAP, який має конкретні вимоги, зокрема: обмін повідомленнями XML, а також вбудовану безпеку та відповідність транзакціям, які роблять його повільніше і важче.

Альтернативою, що активно набирає популярність, можна відзначити GraphQL. GraphQL — це мова запитів і середовище виконання на стороні сервера для інтерфейсів програмування прикладних програм (API), які надають клієнтам запитувані ними дані та нічого крім них.

GraphQL розроблено, з метою внесення альтернативи до варіантів вирішення питання розробки API, та щоб зробити результат роботи більш:

- швидким;
- гнучкими;
- зручними для розробників.

Однією з основних відмінностей та переваг у порівнянні із REST, є те, що GraphQL дозволяє розробникам створювати запити, які витягують дані з кількох джерел даних за один виклик API.

Розробники GraphQL API створюють схеми для опису всіх можливих даних, які клієнти можуть запитувати через розроблювану службу. Схема GraphQL складається з типів об’єктів, які визначають, який тип об’єкта ви можете запитати та які поля він має. Коли надходять запити, GraphQL перевіряє запити на відповідність схемі. Потім GraphQL виконує перевірені запити.

Який сьогодні найкращий спосіб створити API? Напевно, на думку спадає REST, але якщо ви збираєтеся інвестувати в створення нового програмного забезпечення, ймовірно, варто розглянути кілька різних варіантів і вибрати найкращий серед них.

GraphQL виділяється як альтернатива архітектурі REST API в основному (але не тільки), тому що він забезпечує доступний API за дизайном. Він також має власну мову запитів і середовище виконання для виконання запитів за допомогою функцій, які називаються резольверами.

Спочатку розроблений у 2012 році у Facebook як краще рішення для отримання даних для мобільних пристроїв із недостатньою потужністю, GraphQL був відкритий у 2015 році. У 2018 році його передали під опіку Linux Foundation, який підтримує інші важливі проекти, такі як Node.js, Kubernetes, і, звісно, сам Linux.

Загальний рух навколо GraphQL дуже обнадійливий для тих, хто хоче його прийняти. Його популярність стрімко зростає протягом останніх кількох років, як це видно, наприклад, у Stack Overflow Trends. Є також кілька історій успіху в авторитетних компаніях, таких як PayPal, Netflix і Coursera, де GraphQL допоміг створити гнучкі та високопродуктивні API у великих, складних архітектурах.

Однак, враховуючи динамічний технологічний ландшафт, у якому ми працюємо сьогодні, вам буде пробачено за скептичні настрої. Чи може GraphQL бути

ще однією модою? Якщо це працює для цих компаній, чи це обов'язково означає, що буде працювати і для вас? Давайте обговоримо переваги та проблеми GraphQL, щоб ви могли прийняти зважене рішення.

GraphQL — чудовий вибір для організацій з кількома командами та системами, які хочуть зробити свої дані легко доступними через єдиний уніфікований API.

Як технологія API, розроблена для гнучкості, GraphQL є потужним інструментом як для розробників, так і для споживачів API, а також для організацій, що стоять за ними. У цьому розділі ми розглянемо деякі з ключових областей, де GraphQL блискуче.

Незалежно від того, скільки баз даних, служб, застарілих систем і сторонніх API ви використовуєте, GraphQL може приховати цю складність, надаючи єдину кінцеву точку, з якою клієнти можуть спілкуватися. Сервер GraphQL відповідає за отримання даних з потрібних місць, і клієнтам ніколи не потрібно знати деталі про те, звідки надходять різні фрагменти даних. В результаті екосистема GraphQL забезпечує максимальну гнучкість, коли мова йде про те, щоб зробити дані легко доступними для клієнтів і внутрішніх користувачів.

Ще одна величезна перевага для клієнтів API GraphQL полягає в тому, що вони можуть запитувати саме ті дані, які їм потрібні, навіть між пов'язаними об'єктами. Це особливо важливо, оскільки різні клієнти мають різні вимоги до даних, або через різну бізнес-логіку, або тому, що вони просто представляють різне уявлення про дані (наприклад, Інтернет або мобільний), а також можуть мати різні апаратні обмеження.

Для порівняння, набагато важче ефективно отримати нетривіальні дані з REST API. Запит даних з однієї кінцевої точки часто повертає більше даних, ніж насправді потрібно (перебір), тоді як запит даних про декілька пов'язаних об'єктів зазвичай вимагає або кількох викликів API (недобірка) або виділених кінцевих точок для конкретних запитів клієнта (що дублює зусилля). GraphQL вирішує цю проблему, обслуговуючи саме ті дані, які запитує кожен клієнт, ні більше, ні менше.

Екосистема GraphQL містить ряд інструментів, які роблять роботу з GraphQL легкою. Такі інструменти, як GraphiQL і GraphQL Playground, забезпечують багатий досвід, дозволяючи розробникам перевіряти та випробовувати API з мінімальними

зусиллями, завдяки функціям самодокументування, які ми розглянемо в наступному розділі.

Крім того, інструменти генерації коду, такі як GraphQL Code Generator, можна використовувати для подальшого прискорення розробки, тоді як існують інші інструменти та найкращі методи для вирішення конкретних проблем, зокрема:

Кешування на стороні клієнта доступне з коробки в кількох клієнтських бібліотеках.

Розбиття на сторінки на основі курсора дає можливість пропонувати розбиття на сторінки між списками даних.

DataLoader покращує продуктивність шляхом групування запитів на вибірку даних, а також забезпечує базовий рівень кешування на стороні сервера.

API GraphQL побудовано навколо системи типів, яка визначає назву та тип кожного поля, а також відносини між різними сутностями. Ця система типу, або схема, використовується для перевірки запитів, надісланих клієнтом. Схему можна запитати за допомогою функції, яка називається інтроспекція, яка часто використовується для створення документації та коду, які будуть використовуватися під час інтеграції API на стороні клієнта.

Як наслідок, для використання добре документованого API під час використання GraphQL потрібні мінімальні зусилля. Це забезпечує велику прозорість для розробників, які вперше працюють з API, і робить розробку більш гладкою та ефективною.

Зазвичай REST API надають кілька версій одного API, щоб він міг змінюватися без збоїв король існуючої функціональності. GraphQL заохочує інший підхід до модифікацій API: еволюцію.

Якщо необхідні зміни, що порушують роботу (наприклад, перейменувати поле або змінити його тип), ви можете ввести нове поле і відмовитися від старого, можливо, повністю видаливши його пізніше, коли ви впевнені, що воно більше не використовується. Це означає, що ви все ще можете змінити свій API, зберігаючи зворотну сумісність і єдиний API.

Розробник API приєднує кожне поле в схемі до функції, яка називається резольвером. Під час виконання коду обробки запиту, привязана функція-резолвер викликається для отримання значення.

З точки зору клієнта існує два типи запитів: зчитування та мутації. З точки зору моделі CRUD запит на читання є еквівалентним “зчитуванню”. Усі інші (створювати, оновлювати та видаляти) обробляються мутаціями.

Безумовна перевага веб-додатків є їх масштабованість. Оскільки вони не потребують встановлення, не мають зовнішніх залежностей та доступні користувачам з великим спектром пристроїв, веб-додатки все частіше стають вибором для моделі розробки нових продуктів.

Серед недоліків веб-додатків можна зазначити проблему того, що основні обрахування відбувається на сервері, а отже серверна частина потребуватиме значних ресурсів. На відміну від нативних додатків, де є можливість це в більшій мірі перекласти на клієнта.

Для продуктів із невеликим навантаженням ця проблема не стоїть гостро, але коли мова йде про додатки із великою кількістю користувачів, або потребую у “важких” обрахуваннях - витрати на серверну складову можуть досягати значних фінансових видатків.

Окрім того варто зазначити, що налаштування серверної архітектури - складне питання що нерідко потребує окремих спеціалістів, котрі займаються налаштуванням та моніторингом систем.

Існують наступні варіанти для розгортання та налаштування серверної частини:

- спільний хостинг-сервер;
- VPS хостинг;
- виділений сервер.

Кожен із перелічених варіантів має свої переваги та недоліки. Вибір потрібно робити розуміючи потреби свого додатку, та не забувати про такі важливі речі як безпека додатку. На безпеку роботи додатку та рівні захисту впливають такі складові, як:

- засоби моніторингу мережі;

- сертифікати безпеки (наприклад, TLS, SSL, HIPAA і PCI);
- інструменти аварійного відновлення;
- системи сканування шкідливих програм;
- засоби налаштування авторизації користувача;
- фільтр спаму;
- брандмауери;
- системи SFTP-підключення;
- засоби протидії DDoS-атакам, SQL-ін’єкціям, запобігання фішингу.

Окрім того, кожен із хостинг-провайдерів повинен надавати інформацію щодо пропонованих ним основних показників працездатності своїх серверів.

- Гарантія безвідмовної роботи. Клієнти можуть втратити доступ до критичних програм і даних, якщо сервер вийде з ладу. Краще обирати хостинг-провайдерів, що декларують гарантію працездатності не менше 99% часу.
- Регламент компенсацій у випадку непрацездатності серверів. Надійні компанії прописують такі умови у договорі оренди серверних потужностей.
- Звітність про результати діяльності - безумовна важлива складова, що повинна бути регламентована та регульована договором.

Розгортання проекту на спільному хостингу — найпростіша і економічно ефективна форма розміщення проекту. Фактично ресурси одного фізичного сервера віртуалізуються і стають доступними для кількох орендарів у рівних пропорціях. Спільний хостинг ідеально підходить для базових персональних веб-сайтів і веб-програм, які мають невелику кількість користувачів, не потребують особливих технічних налаштувань та мають обмежені вимоги до продуктивності. Окрім того, спільний хостинг надає вже налаштовані засоби безпеки. Звісно, є й недоліки. Той факт, що засоби безпеки налаштовані заздалегідь забирає більшість можливостей її кастомізації, але з іншого боку це в значній мірі економить час. Є й інші недоліки, зокрема оскільки всім орендарям виділяється обмежена кількість можливостей, може виникнути ситуації коли розроблюваний вами веб-ресурс буде недоступний для ваших користувачів через те, що вичерпається ресурсна квота на хостингу.

VPS (virtual private server) хостинг - хостинг, котрий пропонує хостинг для кількох орендарів, але наступного рівня - кожен орендар ділиться деякими, але не всіма, ресурсами одного апаратного сервера і отримує більше контролю над середовищем хостингу. Кожен VPS запускає свою власну операційну систему (ОС) і програми, а також резервує власну частину ресурсів машини (пам’ять, обчислення тощо).

VPS значно надійніший та гнучкіший у порівнянні з “спільним” хостингом. Можливість гнучких кастомізацій дозволяє налаштувати засоби безпеки на значно глибшому та професійному рівні.

З іншого боку такі зручні можливості вимагають вищого рівня знань від спеціаліста, що налаштовує середовище, а отже й займає більше часу.

Також існує ще більш професійний, але й дорожчий, варіант для розміщення своїх проектів - оренда виділеного сервера.

Виділений сервер — це форматі хостингу у вигляді оренди сервера виключно для одного клієнта. Розроблюваний додаток має доступ до абсолютно всіх ресурсів одного апаратного сервера та не має потреби розділяти їх з будь-ким. У порівнянні з іншими формами хостингу, переліченими вище, виділений сервер забезпечує найбільший рівень ізоляції від інших серверів і клієнтів.

2.5. Інструменти розробки клієнтської частини веб-додатку

Клієнтська частина веб-додатку – це та частина додатку, що працює на стороні клієнта, найчастіше – у браузері.

Браузер – різновид програмного забезпечення для комп’ютера, телефона або іншого пристрою, що надає користувачу можливість взаємодії із веб-сторінками. Для відображення вебсторінок браузер отримує від серверу файли трьох основних типів:

- HTML;
- CSS;
- JavaScript.

Веб-розробники використовують ці інструменти для створення клієнтської частини веб-додатків (а також деякі додаткові, що після обробки перетворюються на них).

При завантаженні проходять декілька етапів. Потрібно чітко розуміти які саме, адже це напряму впливатиме на якість і швидкість роботи розробленого додатку.

Першим етапом завантаження сторінки є навігація. Вона відбувається щоразу, коли користувач запитує сторінку. Вводячи її URL адресу, надсилаючи форму, натискаючи по посиланню, тощо.

Навігація, в свою чергу, починається с DNS-пошуку. Звернення у мережі інтернет відбувається по IP адресам. Цей формат взаємодії зручний для комп'ютерів, але геть не зручний для користувачів, адже запам'ятовувати 16 цифр для кожного сайту значно важче, аніж коротку URL адресу.

Оскільки користувачі оперують URL адресами, першим кроком для браузера, при завантаженні ресурсу, є отримання IP адреси, що прив’язана до запитуваної URL-адреси. Цей процес називається DNS-lookup. Більшість сучасних пристроїв вміють кешувати на певний час отриманий результат та уникати витрат часу при зверненні до цієї ж веб-адреси в майбутньому. Це важливо враховувати при проектуванні веб-сторінки. У випадку якщо сторінка матиме потребу у завантаженні додаткових ресурсів їх краще розмістити по тій самій адресі, аби уникнути витрат часу на додаткові DNS пошуки.

Після вдалого отримання IP адреси завантажуваного ресурсу, але перед передачею самих даних, браузер та сервер переходять до фази “знайомство”. “Знайомство” між браузером і сервером відбувається шляхом трьох “рукостискань”. Фактично від браузера до сервера надсилаються три запити, у котрих вони формують контракт взаємодії.

Крім того при встановленні безпечного з'єднання (через https протокол) виконується ще один запит для встановлення правил взаємодії із точки зору безпеки. Цей етап має назву “TLS Negotiation”.

Безумовно останній крок може бути уникнути, якщо використовувати формат http замість https, але цей крок вартує затраченого часу, адже тоді клієнт-сервер

можуть вільно обмінюватись даними. Адже навіть якщо трафік буде перехвачений - його буде вкрай важко “прочитати”, адже він буде зашифрований. Така схема взаємодії формує додатковий рівень захисту, що позитивно впливає на рівень довіри користувача.

Після успішного встановлення зв'язку та проходження усіх етапів налаштування взаємодії браузер надсилає початковий запит. Найчастіше це HTTP GET запит на отримання HTML сторінки. Після отримання запиту сервер формує відповідь, встановлюючи потрібні HTTP заголовки та надсилаючи тіло HTTP запиту у вигляді HTML.

```
<!doctype HTML>
<html>
 <head>
  <meta charset="UTF-8"/>
  <title>Приклад Сторінки</title>
  <link rel="stylesheet" src="styles.css"/>
  <script src="myscript.js"></script>
</head>
<body>
  <h1 class="heading">Моя Сторінка</h1>
  <p>Параграф із<a href="https://example.com/about">посиланням</a></p>
  <div>
    <img src="myimage.jpg" alt="image description"/>
  </div>
  <script src="anotherscript.js"></script>
</body>
</html>
```

Рис. 2.3. Приклад базової HTML сторінки

Отримавши перший блок даних, браузер відразу починає його аналіз. Синтаксичний розбір – один із перших етапів відображення сторінки, його ідея полягає в тому, що браузер перетворює отримані дані у два відображення: DOM і CSSOM, які використовуються рендерером (складовою браузера) для відображення сторінки на екрані користувача.

DOM – внутрішнім представленням розмітки для браузера. Ним можна маніпулювати використовуючи для цього різноманітні браузерні програмні інтерфейси, використовуючи мову програмування JavaScript.

Об’єктна модель документа (DOM) — це інтерфейс програмування для веб-документів. Він представляє сторінку у зручному для програмних змін структури, вигляду та змісту вигляді. DOM-представлення документу являє собою комбінацію вузлів і об'єктів.

Дерево DOM описує вміст документа. Елемент <html> є першим тегом і кореневим вузлом дерева документів. Дерево відображає зв’язки та ієрархії між різними тегами. Теги, вкладені в інші теги, є дочірніми вузлами. Чим більша кількість вузлів DOM, тим більше часу потрібно для побудови дерева DOM.

Коли синтаксичний аналізатор знаходить неблокуючі ресурси, наприклад зображення, браузер запитає ці ресурси та продовжить аналіз. Так само синтаксичний розбір не зупиняється й при опрацюванні підключені стилів. Але теги <script>, зокрема підключені без атрибутів async або defer, блокують відтворення та припиняють розбір HTML, браузер переключається на розбір скриптів. Саме тому потрібно бути дуже обережним при додаванні нових тегів. Кожен із них сповільнює відображення сторінки, що в свою чергу негативно впливає на досвід користувача.

2.6. Аналіз TypeScript - інструменту розробки нового покоління

TypeScript – це сучасна мова програмування, розроблена за підтримки компанії Microsoft. Фактично являє собою над-множину для мови програмування JavaScript. Одна із основних можливостей, що додаються TypeScript до JavaScript це статична типізація. Динамічна типізація - один із основних недоліків мови програмування JavaScript. При розробці великих корпоративних додатків ця особливість нерідко призводить до появи помилок, котрі дуже важко відлагодити.

TypeScript може бути використаний як для розробки клієнтської, так і серверної частини додатку. Фактично, він може бути використаний будь-де, де підтримується

виконання JavaScript коду, адже компилятор TypeScript перетворює написаний на TypeScript код у код JavaScript.

Сам компілятор TypeScript написаний на TypeScript і скомпільований е JavaScript. Він ліцензований за ліцензією Apache 2.0. TypeScript включено як першокласну мову програмування в Microsoft Visual Studio 2013 Update 2 і новіших версій, поряд із C# та іншими мовами Microsoft.

Офіційне розширення дозволяє Visual Studio 2012 також підтримувати TypeScript. Варто відзначити, що Андерс Хейлсберг (провідний архітектор C# і творець Delphi і Turbo Pascal) працював над розробкою TypeScript.

Рис. 2.4. Приклад перетвореня TypeScript коду на JavaScript

За останні кілька років TypeScript стає все популярнішим. Angular, одна з найбільших інтерфейсних фреймворків, використовує TypeScript. Близько 60% програмістів JS вже використовують TypeScript, а 22% хочуть спробувати. Чому? Історично JavaScript став основною мовою для написання сценаріїв веб-сторінок і програм в Інтернеті. Тепер можна використовувати JavaScript як на інтерфейсі, так і на серверній частині за допомогою таких фреймворків, як Node.js і Deno.

Насправді, дослідження показує, що 15% усіх помилок JavaScript можна виявити за допомогою TypeScript.

Свобода динамічного введення часто призводить до помилок, які не тільки знижують ефективність роботи програміста, але також можуть зупинити розробку через збільшення витрат на додавання нових рядків коду.

Таким чином, нездатність JavaScript включати такі речі, як типи та перевірки помилок під час компіляції, робить його поганим вибором для серверного коду в підприємствах і великих кодових базах. Як говорить їх слоган, TypeScript - це JavaScript, який масштабується.

TypeScript, по суті, є лінтером JS. Або JS з документацією, яку може зрозуміти компілятор.

Тому, на відміну від інших мов, таких як CoffeeScript (який додає синтаксичний цукор) або PureScript (який зовсім не схожий на JavaScript), вам не потрібно багато вчитися, щоб почати писати код TypeScript.

Типи в TS є необов'язковими, і кожен файл JS є дійсним файлом TypeScript. Хоча компілятор скаржиться, якщо у ваших початкових файлах є помилки типу, він повертає вам файл JavaScript, який працює так само, як і раніше. Де б ви не були, TypeScript зустріне вас там, і легко розвивати свої навички поступово.

TypeScript скомпільовано в JavaScript. Таким чином, TS можна використовувати де завгодно JS: як інтерфейс, так і бекенд.

JavaScript є найпопулярнішою мовою для реалізації сценаріїв для інтерфейсу програм і веб-сторінок. Таким чином, TypeScript можна використовувати для тих самих цілей, але він блискуче в складних корпоративних проектах на стороні сервера.

Типи – це спосіб відрізнити правильні програми від неправильних, перш ніж ми їх запустимо, описуючи в нашому коді, як ми плануємо використовувати наші дані. Вони можуть варіюватися від простих типів, таких як число і рядок, до складних структур, ідеально змодельованих для нашої проблемної області.

Мови програмування діляться на дві категорії: статично типізовані або динамічно типізовані.

У мовах зі статичною типізацією тип змінної повинен бути відомий під час компіляції. Якщо ми оголошуємо змінну, компілятор повинен знати (або не можна вивести), чи це буде число, рядок чи логічне значення. Подумайте про Java.

У мовах з динамічним типом це не обов'язково. Тип змінної відомий лише під час запуску програми. Подумайте про Python.

TypeScript може підтримувати статичну типізацію, тоді як JavaScript ні.

TypeScript має різноманітні основні типи, як-от Boolean, Number, String, Array, Tuple тощо. Деякі з них не існують у JS; Ви можете дізнатися більше про них в документації TypeScript.

На додаток до них, ось деякі інші типи, які ми хочемо представити, щоб продемонструвати виразність TS:

На відміну від JavaScript, код TypeScript є надійнішим і легшим для реорганізації. Це дозволяє розробникам уникати помилок і набагато легше виконувати переписування.

Типи скасовують більшість дурних помилок, які можуть проникнути в кодові бази JavaScript, і створюють швидкий цикл зворотного зв'язку, щоб виправити всі маленькі помилки під час написання нового коду та рефакторингу.

2.7. Типи архітектурних рішень при розробці веб-додатків

Проектування вебдодатків – складне питання, що потребує додаткового розгляду, адже існують не тільки різні архітектурні рішення клієнт-серверної взаємодії, а й відмінні між собою варіанти побудови архітектури кожної із цим частин окремо.

До прикладу серверна частина вебдодатку може бути розроблена у вигляді монолиту або мікросервісу.

Монолітна кодова база – класичне архітектурне рішення, при котрому уся бізнес-логіка зберігається в одному проекті. Таких підхід добре працює для простий та середніх за рівнем складності розробки проектів, але її недоліки починають перевищувати її переваги пропорційно розвитку проекту.

Зокрема такий підхід призводить до значного ускладнення розуміння проекту. Безумовно, монолітна кодова база містить модулі, компоненти та інші юніти, але загалом процес передачі знань для того проекту значно важчий.

Окрім того, безумовним недоліком є те, що при внесенні змін до одного із модулів є ризик зачепити інші, сусідні юніти, котрі здавалось би не мають відношення до першочергових змін. Таке нерідко трапляється у великих проектах.

Постійний ріст рівня складності системи ускладнює й її подальшу розробку та сповільнює розвиток. Окрім того, при бажанні змінити технологію розробки, або банально оновити версію – масштаб завдання значний та потребує окремого виділення часу.

Саме тому все частіше розробники почали надавати перевагу розробці із використанням мікросервісної архітектури. Ідея просто - розбити великий та складний додаток на менші за розміром та об'ємом логіки проекти. Все частіше намагаються притримуватись правила, що кожен новий мікросервіс повинен виконувати певну частину бізнес-логіки та бути частково чи повністю ізольований.

Такий підхід до розробки поліпшує ситуацію із точки зору складності розробки. Адже вносити зміни та оновлювати проект, що має один функціонал значно легше. Окрім того, зменшується ризик своїми змінами призвести до помилок у роботі системи.

Легше тестування та внесення змін - безумовні переваги мікросервісної архітектури. Але є і недоліки. Робота над проектом що базується на мікросервісній архітектурі пов’язана із взаємодією декількох проектів, що в свою чергу може уповільнити процес внесення простих змін. Окрім того, оновлення проектів хоч і є менш складним (адже ризик, що виникнуть проблеми через несумісність версій значно менший), але теж займає значну кількість часу, адже кількість проектів активно збільшується по мірі розвитку проекту.

Деякі компанії практикують комбінувати груп споріднених за функціоналом або бізнес-логікою мікро сервісів у монорепозиторії.

2.8. Монорепозиторії для веб-додатків

Монорепозиторії — спосіб збереження кодової бази у системі керування версіями (наприклад, Git) у вигляді одного репозитория, що містить багато проектів.

Хоча ці проекти можуть бути пов’язаними, вони часто логічно незалежні й керуються різними командами.

Деякі компанії розміщують весь свій код в одному репозиторії, спільному доступі для всіх. Монорепо може досягати колосальних розмірів. Наприклад, передбачається, що Google має найбільше сховище коду в історії, яке має десятки сотень комітів на день і розміром понад 80 ТБ. Іншими компаніями, які, ведуть великі монорепозитории, є Microsoft, Facebook і Twitter.

На перший погляд, вибір між монорепо та мульти-репозиторіями може здатися незначним, але це рішення суттєво вплине на робочий процес розвитку вашої проекту. Переваг монорепозиторія наступні.

- Видимість: кожен учасник проекту бачить увесь чужий код. Ця властивість сприяє кращій співпраці та взаємодії між командами — розробник з іншої команди може виправити помилку у коді колеги, про існування якої ви навіть не здогадувалися. Особливо це стає актуальним у випадку із інтеграцією кількох додатків або змін у контрактах між ними.
- Простіше керування залежностями: менеджер пакетів не потрібен, оскільки всі модулі розміщені в одному репозиторії.
- Єдине джерело істини: одна версія кожної залежності означає, що немає конфліктів версій і пекла залежностей.
- Послідовність: дотримуватись стандартів якості коду та уніфікованого стилю легше, коли у вас є вся ваша кодова база в одному місці.
- Спільна часова шкала: критичні зміни в API або спільних бібліотеках відкриваються негайно, що змушує різні команди спілкуватися наперед і об’єднувати зусилля. Кожен інвестує в те, щоб йти в ногу зі змінами.
- Неявний CI: безперервна інтеграція гарантована, оскільки весь код вже уніфікований в одному місці.
- Уніфікований CI/CD: ви можете використовувати той самий процес розгортання CI/CD для кожного проекту в репозиторії.
- Уніфікований процес збірки: ми можемо використовувати спільний процес збірки для кожної програми в репозиторії.

З іншого боку, як і будь-який підхід, монорепозиторії мають і свої недоліки, зокрема:

- погана продуктивність: монорепозиторії важко збільшити. Такі команди, як git blame, можуть зайняти невиправдано багато часу, IDE починають відставати, продуктивність знижується, а тестування всього репозиторію на кожному коміті стає неможливим;
- зламаний CI/CD: зламаний CI/CD у master гілці впливає на всіх, хто працює в монорепозиторії. Це можна розглядати як проблему, або як хорошу мотивацію підтримувати тести чистими та актуальними;
- крива навчання: крива навчання для нових розробників є крутішою, якщо репозиторій охоплює багато тісно пов'язаних проектів;
- великі обсяги даних: монорепозиторії може досягати громіздких обсягів даних і комітів на день;
- право власності: зберегти право власності на файли складніше, оскільки такі системи, як Git або Mercurial, не мають вбудованих дозволів до каталогу;
- огляд коду: сповіщення можуть бути дуже шумними. Наприклад, у GitHub є обмежені налаштування сповіщень, які не найкраще підходять для сніжної гірки запитів на перегляду коду. Розробникам доведеться налаштовувати фільтри на рівні поштових клієнтів.

Немає універсального рішення, які б підходили для кожного випадку використання. Деякі компанії можуть вибрати монорепозиторії на деякий час, а потім вирішити, що їм потрібно перейти на мультирепозиторії або навпаки, тоді як інші можуть вибрати комбінацію. Зрештою, це більше не про технології, а про культуру. Культуру роботи та спілкування.

2.9. Огляд NX – сучасного засобу організації кодової бази

Nx — це розумна і розширювана платформа для зберігання коду проекту, його збіірка. Вона допомагає створювати, тестувати та будувати проекту будь-якого

масштабу. До переваг платформи належить той факт, що вона безперешкодно інтегрується з сучасними технологіями та фреймворками.

Окрім того, платформа надає такі сучасні можливості як:

- виконання завдань на основі розподіленого графіка;
- кешування обчислень;
- розумне відновлення проектів, які постраждали;
- потужні генератори коду;
- підтримка редактора програми GitHub.

Nx допомагає вам розробляти додатки Angular з повністю інтегрованою підтримкою сучасних інструментів і бібліотек, таких як Jest, Cypress, ESLint, Storybook, NgRx.

Nx побудовано на технологічно-агностичному ядрі, яке підтримує модульні одиниці коду та розуміє графік залежності між ними. Розробники розуміють абстракцію, що закладена в додаток, наприклад: ваш додаток містить дві сторінки, кожна з яких використовує спільний компонент кнопки. Nx вміє проводити розумний аналіз і виявляти ці самі принципи взаємодії, використовує цю інформацію, щоб зрозуміти структуру проекту, надати поради та покращити продуктивність. Базуючись на цьому алгоритмі була створена ціла екосистема плагінів.

По мірі розростання проекту цінність Nx збільшується в геометричній прогресії, адже час який економить використання цього засобу розробки значно збільшується.

Nx особливо добре працює для монорепозиторіїв. Кожна нова програма, додана до монорепозиторія, додає ще більше можливостей для спільного використання коду та інструментів, але це часто відбувається за ціною уповільнення роботи CI процесу. Nx гарантовано покращує цей процес, покращуючи стандартні алгоритми запуску.

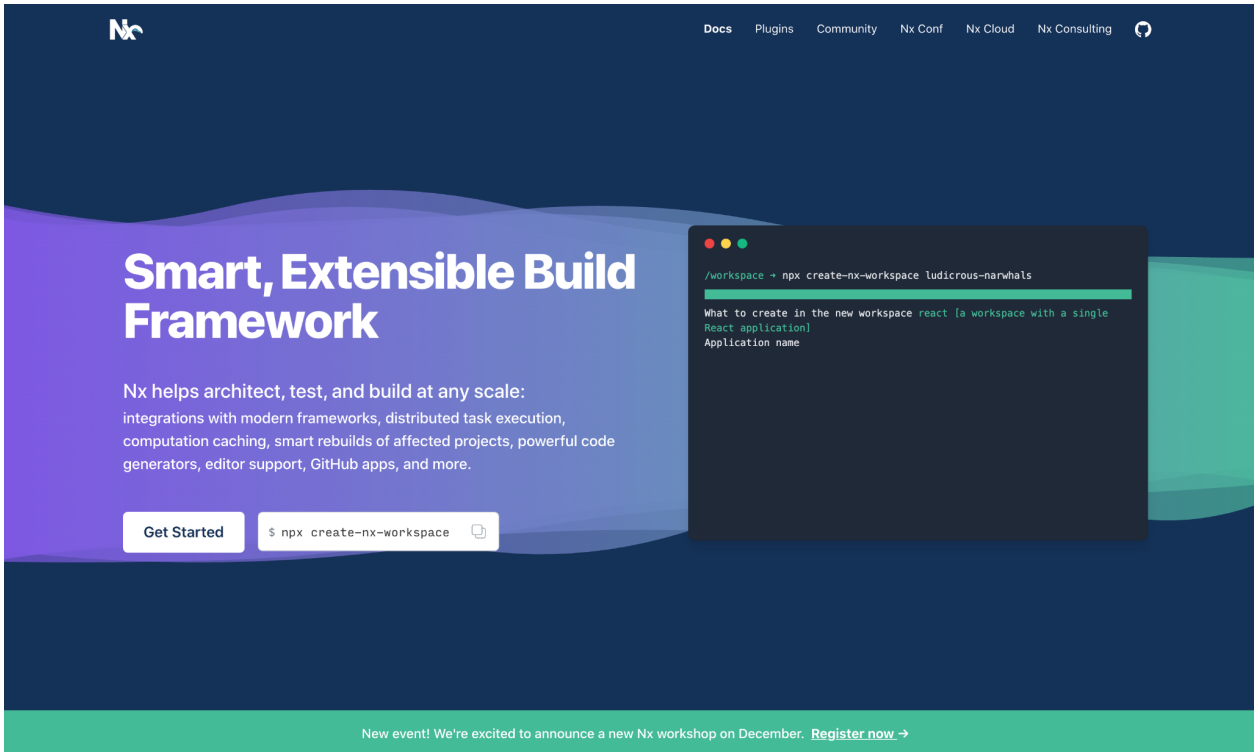

Рис. 2.5. Головна сторінка інструменту Nx

2.10. Огляд засобу серверної розробки – Node.js

Node.js — це середовище виконання JavaScript з відкритим вихідним кодом та міжплатформним. Це популярний інструмент що використовується у проектах різноманітного масштабу. Інструмент запускає движок V8 (ядро Google Chrome) поза браузером. Це дозволяє Node.js бути дуже продуктивним та працювати на будь-якому типі платформ.

Цікавою відмінністю є те, що додаток написаний із використанням Node.js виконується в одному процесі, не створюючи новий потік для кожного запиту. Node.js надає в стандартній бібліотеці набір асинхронних примітивів введення-виводу, які запобігають блокування коду JavaScript, і загалом бібліотеки в Node.js написані з використанням не блокуючих парадигм, що робить поведінку блокування скоріше винятком, ніж нормою.

Коли Node.js виконує операцію вводу-виводу, наприклад, читання з мережі, доступ до бази даних або файлової системи, замість того, щоб блокувати потік і витрачати цикли ЦП на очікування, Node.js відновить операції, коли відповідь

повернеться, перемикаючись тим часом на виконання наступної операції. Це дозволяє Node.js обробляти тисячі одночасних підключень з одним сервером, не додаючи потреби керування паралельністю потоків, що може бути критичним джерелом помилок при масштабуванні проекту.

Окремою важливою перевагою Node.js є те, що оскільки мільйони розробники клієнтьскої частини веб-додатків, які пишуть JavaScript для браузера, можуть писати код на стороні сервера без необхідності вивчати зовсім іншу мову.

Звісно є певна відмінність у підходах, що використовуються, але це гарне підґрунтя для активного та легкого розвитку технології.

У Node.js нові стандарти ECMAScript можна використовувати без проблем, на відміну від розробки під звичні JavaScript розробникам браузери. Оскільки не доведеться чекати, поки всі користувачі оновлять свої браузери – розробник самостійно вирішує, яку версію ECMAScript використовувати, змінюючи версію Node.js. Окрім того, додатковими конфігураціями можливий запуск окремих експериментальних функцій. Це нерідко стає у нагоді про розробці із використанням останніх архітектурних рішень.

Node.js – низькорівнева платформа. З метою пришвидшення розробки спільності розробників активно додає та оновлює існуючі Node.js бібліотеки та плагіни. Багато з них з часом стали популярними та невід'ємними частинами платформи, зокрема:

- Nest.js: повнофункціональний фреймворк на основі TypeScript, зосереджений на ергономіці, стабільності та впевненості розробника. Adonis — одна з найшвидших вебфреймворків Node.js.
- Express: це один із найпростіших, але потужних способів створення вебсервера. Його мінімалістичний підхід, бездумний, зосереджений на основних функціях сервера, є ключем до його успіху.
- Fastify: Вебфреймворк, спрямований на забезпечення найкращого досвіду розробника з найменшими витратами та потужною архітектурою плагінів. Fastify — одна з найшвидших вебфреймворків Node.js.

- Gatsby: генератор статичних сайтів, заснований на React, на базі GraphQL з дуже багатою екосистемою плагінів.
- hapi: багатий фреймворк для створення додатків і служб, який дозволяє розробникам зосередитися на написанні логіки багаторазового використання замість того, щоб витрачати час на створення інфраструктури.

Node.js – фреймворк, що дозволяє поєднувати класичні архітектурні рішення, популярні тренди розробки та сучасні засоби.

2.11. Аналіз веб-фреймворку Angular

Angular – фреймворк нового покоління, розроблений для спрощення, структуризації та систематизації розробки клієнтської частини веб-додатків. Активно розвивається на набирає популярності із роками. Основні конкуренти: Vue.js та React.

Мовою програмування за замовчуванням використовує TypeScript, але є можливість переналаштувати під використання JavaScript. Тим не менш, не зважаючи на можливість таких налаштувань на практиці частіше надають перевагу використанню мови програмування за замовчуванням, адже сильно цінують внесок статичної типізації та інших додаткових можливостей, котрі надає JavaScript.

Angular — це добре відомий фреймворк, створений для створення веб-додатків. Він добре підходить для малих і великих програм, тому його дійсно варто вивчати та використовувати. Вибір однієї із версій потребує додаткового визначення потреб проекта, адже вони відмінні.

Ймовірно, для більшості фронтенд-розробників Angular є добре відомим фреймворком, створеним для створення веб-додатків. Якщо ми хочемо використовувати Angular у нашому проекті, ми повинні вибрати одну з його версій – AngularJS чи іншу.

AngularJS – або, як дехто віддає перевагу: Angular 1 – був створений у 2009 році. Він дає нам двостороннє прив’язування даних і дає змогу бачити зміни даних у JavaScript, які автоматично відображаються в інтерфейсі користувача. Крім того, AngularJS має директиви, які дозволяють нам створювати більше розділеного та

багаторазового коду, ніж будь-коли раніше. Як правило, це дозволяє програмістам писати програми на архітектурі MVC або MVVM (іноді її називають простіше архітектурою MVW (Model-View-Whatever). Вона зробила крок вперед у тестуванні зовнішніх додатків завдяки своєму механізму впровадження залежностей, який допомагає імітувати залежності.

Angular — чудова структура. Він має багато покращень з точки зору AngularJS. Він буде ставати все більш популярним з роками, а, отже, він добре підходить як для малих, так і для великих програм, тому його дійсно варто вивчати та використовувати. Коли говориться про Angular, варто окремо зауважити, що є також ReactJS, який є трохи противником Angular, він має свої відмінності, переваги та недоліки.

При побудові додатку розробник має можливість використати наступні блоки, для формування додатку:

- компоненти – слугують основою додатку. Компоненти у тій чи іншій формі існують майже у кожному різновиді фреймворків, навіть більше – сучасний HTML стандарт починає підтримувати компоненти як структурні блоки. Зазвичай компоненти створюються довкола певної функції, та об'єднуються в групи за спорідненим функціоналом;
- сервіси – широка категорія, що охоплює будь-яку функцію яка потрібна додатку. Сервіс, як правило, є класом з вузькою, чітко визначеною метою. Angular відокремлює компоненти від сервісів з метою покращення можливостей до перевикористання коду та модульності додатку;
- модулі – Angular додатки є модульними, що реалізується шляхом власної імплементації механізму під назвою NgModules. NgModules – це контейнери для цілісного блоку коду, призначеного для домену програми, робочого процесу або тісно пов'язаного набору можливостей. Вони можуть містити компоненти, сервіси та інші файли коду, область дії яких визначається вмістом NgModule. Вони можуть імпортувати функціональні можливості, які експортуються з інших модулів NgModules, і експортувати власні функції для використання іншими модулями

NgModules. Такий архітектурний підхід дозволяє зберегти гнучкість проекти при збільшенні його розміру.

Окрім того, при розробці також використовуються Директиви, Пайпи та деякі інші технічні можливості.

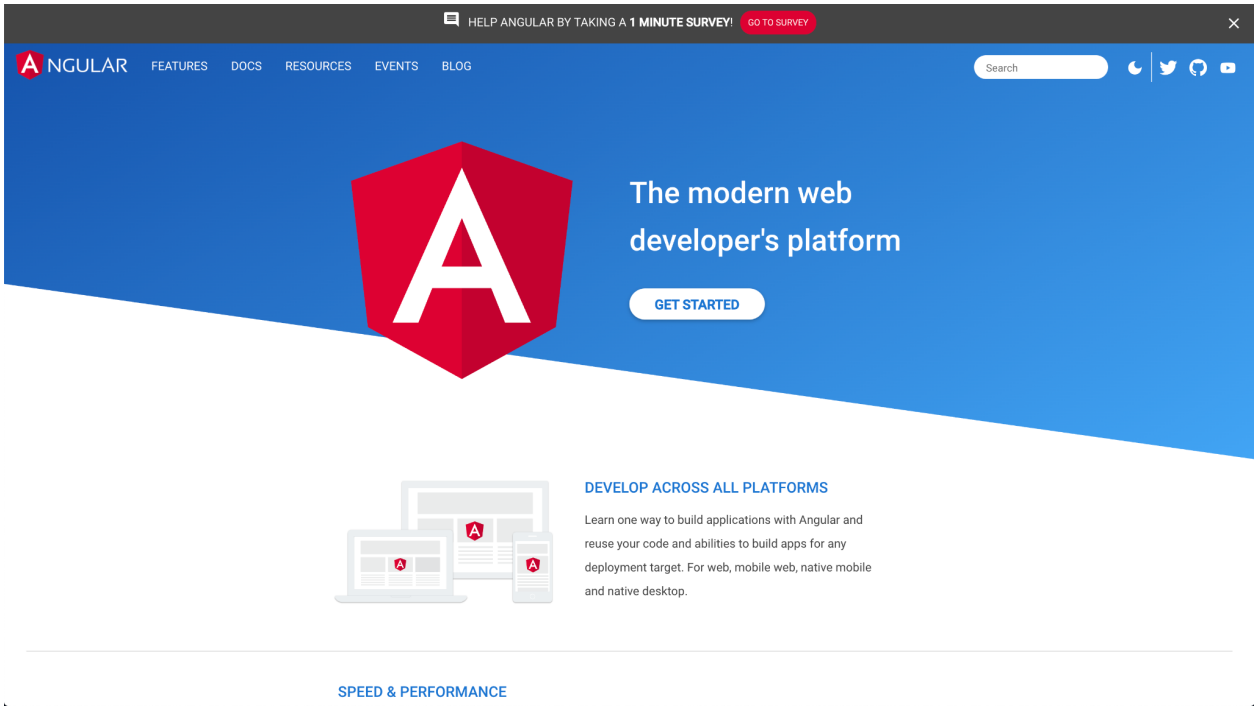

Рис. 2.6. Angular – сучасний фреймворк на розробки

Висновки до розділу 2

Основною метою розділу був опис сучасних технологій для розробки додатків. Були розглянуті різноманітні варіанти для розробки, їх переваги та недоліки.

Обраним типом додатку став вебдодаток. Також були детально описані архітектурні підходи до реалізації цього типу додатку. Окремо увагу було приділену способам збереження та керування кодом. Розглянуті архітектурні підходи до імплементації додатку дали можливість обрати оптимальний шлях для імплементації.

Також були проаналізовані та оглянуті засоби реалізації клієнтьскої частини додатку. Була описана робота базових інструментів як: HTML, CSS, JavaScript та більш складних засобів розробки, зокрема: Angular, TypeScript. Їх переваги та недоліки дали можливість сформувати зручний перелік інструментів для розробки додатку.

РОЗДІЛ 3
РОЗРОБКА ЗАСОБУ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ІНФОРМАЦІЙНИХ РЕСУРСІВ МЕРЕЖІ ІНТЕРНЕТ

Метою цього розділу є опис процесу реалізації засобу інтелектуального аналізу інформаційних ресурсів мережі інтернет. Будуть детально розглянуті основні складові проекту, їх функціональність та принципи побудови. Огляд основних функціональних можливостей розробленого додатку надасть змогу скласти загальний опис використання додатку.

2.1. Розробка архітектури додатку

Для реалізації додатку була обрана комбінована архітектура. Клієнтська частина виконана у вигляді статичного веб-додатку із використанням базових інструментів розробки, таких як: JavaScript, CSS, HTML.

Серверна частина базується на основі Node.js, використовує фреймворк Nest.js і Express.js для розгортання серверу.

Архітектура обох частин додатків – моноліт.

Серверна складова імплементована у вигляді програмного інтерфейсу (API), що дає можливість її подальшого використання при розробці нативних (платформених) додатків, зокрема рішень для мобільних пристроїв та планшетів. Окрім того, за потреби, додаток може бути інтегрований у інший сервіс з метою розширення його функціоналу.

Відповідно до вимог, клієнтська частина повинна містити наступні складові:

- ознайомча (лендінг) сторінка;

- сторінки авторизації: сторінки для логіну, реєстрації та сбросу пароля;
- сторінка інтелектуального аналізу.

Серверна частина, в свою чергу, повинна надавати інтерфейс взаємодії із базою даних, обробляти помилки та перевіряти дані на коректність. Зокрема, повинні бути доступні наступні типи запитів.

- Модуль авторизації: реалізована можливість надіслати запит на авторизацію користувача, реєстрацію або сброс пароля.
- Модуль інтелектуального аналізу веб-ресурсу: повинен містити програмну логіку інтелектуального аналізу веб-ресурсу.

3.2. Головний модуль додатку

Перше, що бачить користувач на початку взаємодії із додатком – це головний модуль. Головний модуль виконує важливу роль – ознайомлює користувача із функціоналом додатку та мотивує його пройти реєстрацію чи авторизацію.

Основні елементи головного модуля:

- заголовок;
- опис додатку;
- кнопка “поклик-до-дії”;
- ознайомчий із інтерфейсом додатку медіа файл-картинка.

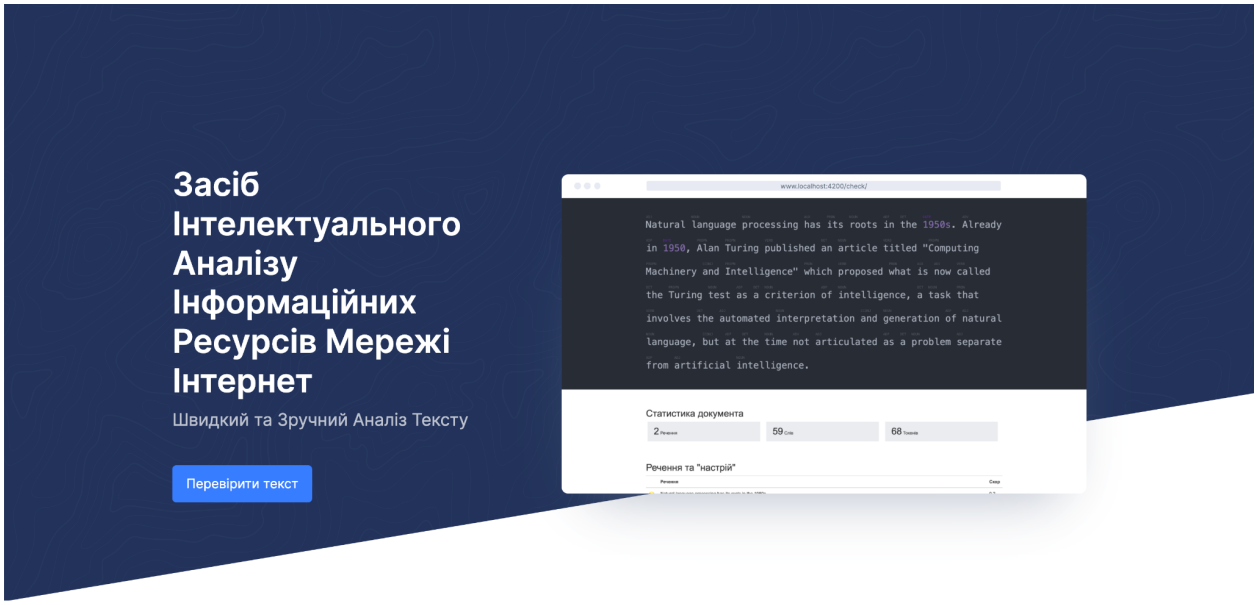

Рис. 3.1. Головний модуль додатку

Кнопка “Перевірити текст” працює за наступним алгоритмом.

- Якщо користувач авторизований – вона переадресовує користувача на сторінку інтелектуального аналізу.
- Якщо користувач був авторизований раніше – вона переадресовує користувача на сторінку логіну.
- Якщо про користувача немає жодних відомостей – переадресація відбудеться на сторінку реєстрації.

3.3. Система авторизації

Система авторизації – невід'ємна складова будь-якого продукту. Для її реалізації потрібно додати функціональну підтримку до серверної складової та імплементувати зв'язок із нею на стороні клієнта.

Сервер повинен містити базу даних, де і будуть збережені клієнти. Важливо зауважити, що забороняється зберігати паролі користувачів у їх “чистому” вигляді, і дуже рекомендується хешувати їх перед збереженням.

Це робиться насамперед задля того, щоб у випадку потенційному хакерської атаки зловмисники не отримали доступ до паролів користувачів, адже деякі з них можуть використовувати одні й ті самі паролі у різних системах.

Таким чином серверна частина повинна надавати програмний інтерфейс (API) для наступних методів:

- реєстрація: перший метод що використовує новий користувач. Повинен містити наступні поля: email, password;
- логін: метод, що буде викликаний щоразу при авторизації користувача в системі. Запит передає такі поля: email, password;
- відновлення паролю: запит ініціюється, коли користувач повідомляє системи, що забув пароль та бажає його відновити.

Кожен із описаних методів повинен коректно обробляти помилки та у випадку їх наявності повідомляти про це користувача API.

Відповідно до розглянутого попередньо архітектурного рішення для розробки серверної частини авторизації були створенні інтерфейси користувача для взаємодії із ними.

3.3.1. Розробка інтерфейсу реєстрації

Інтерфейс реєстрації дуже простий. Його головний елемент – форма реєстрації. Вона містить наступні поля,

- Поле для вводу електронної пошти: електронна пошта - основний спосіб ідентифікації користувача в системі. Вона повинна бути коректно відформатована та унікальна в рамках системи. Саме тому клієнтська частина перед відправкою запиту перевіряє коректність формату електронної адреси, а серверна в свою чергу перевіряє чи немає в системі користувача з такою самою електронною адресою.
- Поля вводу паролю: для запиту бажаного паролю до облікового запису використовується пара полей для ввода пароля: один основний і дублюючий. Використовується дублююче поле з метою убезпечити користувача від помилки, адже вірогідність того, що користувач двічі введе однаковий пароль із опечаткою значно менша, аніж у випадку із одним полем (без дублюючого). Окрім перевірки на

відповідність значень між двома полями, перевіряється і коректність з точки зору мінімальних умов до складності паролю, зокрема пароль повинен щонайменше містити 8 символів. Такі заходи безпеки надають можливість гарантувати вищий рівень захисту даних користувача.

- Поле погодження із правилами “Політики конфіденційності”: використовується з метою надання користувачу офіційної інформації щодо політики конфіденційності.

При натисканні на кнопку “Реєстрація” відбувається перевірка коректності введених даних. У випадку, якщо всі дані пройшли перевірку – форма надсилається на сервер та відображається проміжковий стан “Завантаження”, індикатором якого є анімація завантаження що відображається поруч із текстом кнопки відправки форми.

У випадку, якщо одне або більше полей у формі містять не коректні дані - форма не може бути відправлена. Некоректні поля відсічуються червоним а під ними виводиться допоміжне повідомлення із порадами щодо того, як виправити помилку, що виникла.

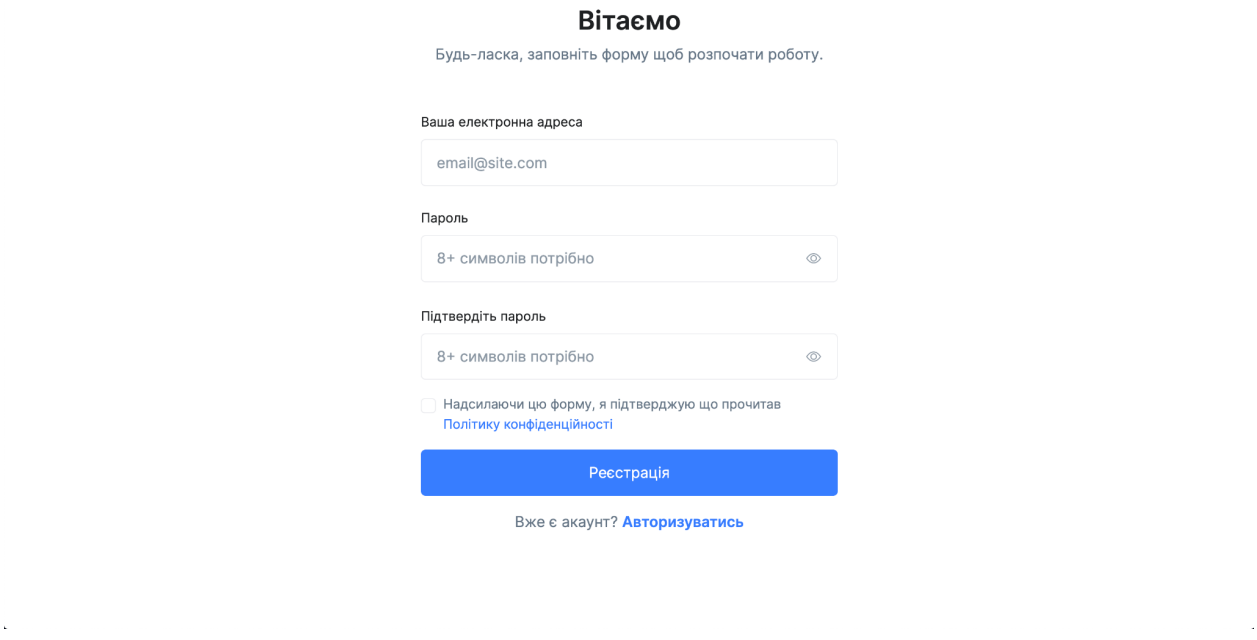

Рис. 3.2. Інтерфейс реєстрації

При отриманні запиту на реєстрацію серверний додаток обробляє його наступним чином.

1. Перевірка коректності (валідності) отриманий даних.
2. Перевірка унікальності введеної електронної адреси.
3. Хешування паролю.
4. Збереження даних користувача у БД.
5. Формування та повернення до клієнта токена авторизації.

У випадку, якщо виникає та чи інша помилка – сервер надсилає відповідь, що містить ідентифікатор помилки. Такі відповіді сервера повинні бути очікуваними та оброблятись клієнтом. Приклад відповіді на запит реєстрації у випадку, якщо надіслана електронна пошта використана у іншому обліковому записі представлена на Рис. 3.3.


```
{
    "statusCode":400,
    "message":"EMAIL_TAKEN",
    "error":"Bad Request"
}
```


Рис. 3.3. Приклад відповіді сервера

3.3.2. Розробка інтерфейсу логіну

Інтерфейс логіну логіну використовується користувачем, що вже має обліковий запис у системі та бажає заново пройти авторизацію. Форма логіну схожа до попередньо розглянутої форми реєстрації, але має відмінності, зокрема деякі спрощення.

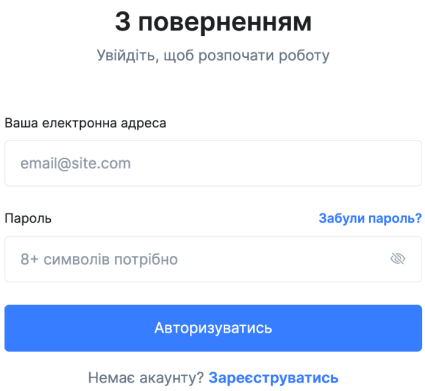

Рис. 3.4. Інтерфейс логіну

Складові форми для логіну наступні.

- Поле для ввода електронної пошти: електронна пошта використовується як основний спосіб ідентифікації користувача у базі даних. Клієнтська частина перед відправкою запиту перевіряє коректність формату електронної адреси.
- Поле для вводу пароля. Має ті самі правила валідації, що і поле при реєстрації.

При заповненні форми та натисканні кнопки “Авторизуватись” формується запит до сервера в котрому й передаються перелічені параметри у форматі JSON.

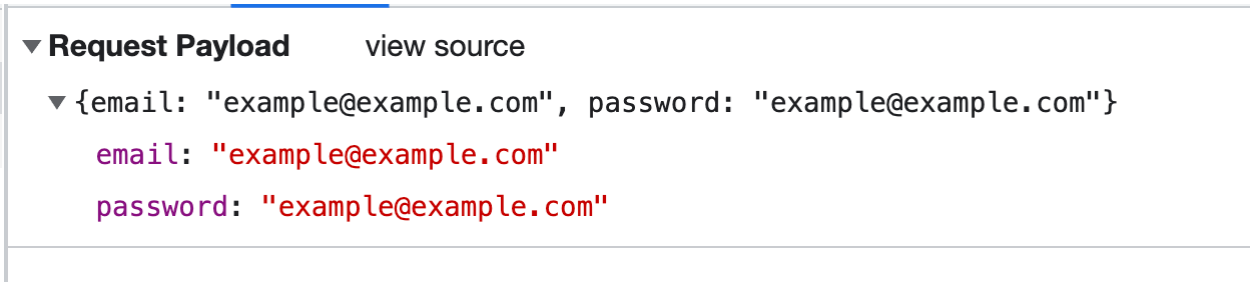

Рис. 3.5. Дані запиту на логін користувача

Сервер, в свою чергу, отримавши запит, обробляє його за наступним алгоритмом:

1. перевіряє, чи база даних містить запис про користувача з такою електронною адресою;
2. хешує переданий пароль, та порівнює його із хешем пароля, збереженим при реєстрації;
3. у випадку, якщо усі перевірки успішно пройдені сервер авторизує користувача генеруючи JWT-токен.

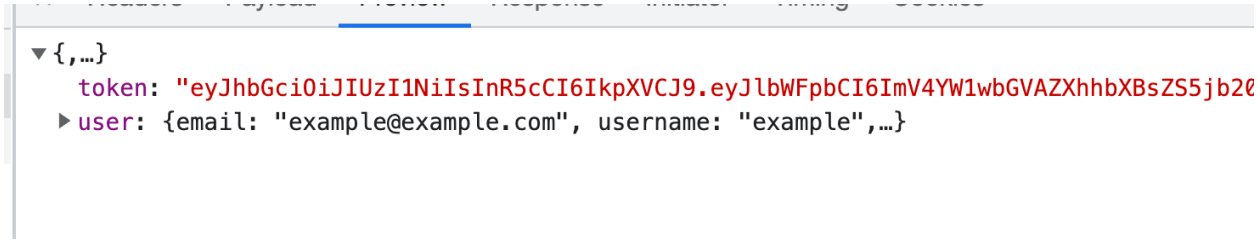

Рис. 3.6. Відповідь серверу на успішний запит авторизації

У випадку, якщо користувача із введеною електронною адресою не буде знайдено сервер поверне помилку та її детальний опис, в котрому вкаже, що саме викликало її:

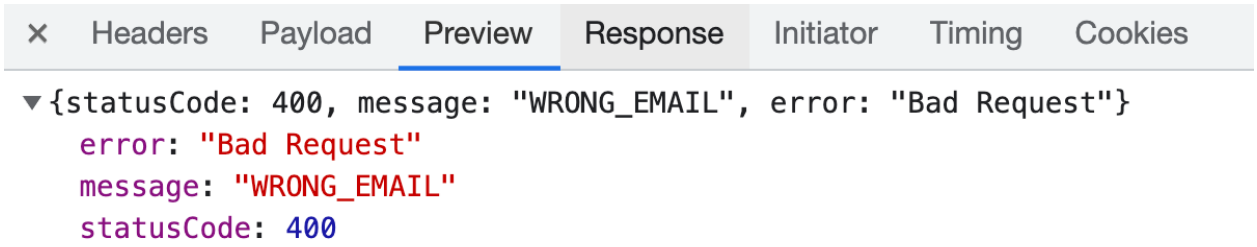

Рис. 3.7. Відповідь серверу на некоректно введену електронну адресу

Клієнту ж, в свою чергу, при отриманні відповіді про неуспішну авторизацію виводиться відповідне повідомлення із вказівками щодо того, де саме виникла помилка.

Окрім полів форми та кнопку відправки запиту, сторінка логіну містить посилання на сторінки реєстрації та скидання паролю.

3.3.3. Розробка інтерфейсу скидання паролю

Сторінка скидання паролю використовується користувачем у випадку, якщо він забув пароль та хоче відновити доступ до свого облікового запису. Алгоритм скидання пароля наступний.

1. Користувач заходить на сторінку скидання паролю.
2. Користувач вводить електронну адресу, що прив'язана до його облікового запису.
3. Користувач надсилає форму із запитом на скидання паролю.
4. У випадку, якщо користувач із введеною електронною адресою існує, на неї (електронну адресу) надсилається лист із посиланням на форму, для введення нового паролю для облікового запису до якого прив'язана електронна адреса.
5. Користувач переходить по отриманому посиланню та заповнює форму відновлення пароля, відправляє її.
6. Сервер успішно обробляє запит та встановлює новий пароль для облікового запису.

Цей сценарій використання починається із сторінки скидання паролю, що містить форму з одним полем для вводу електронної адреси та кнопку відправки запиту на відновлення доступу.

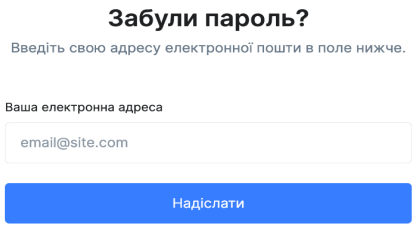

Рис. 3.8. Форма скидання паролю до облікового запису

3.4. Розробка інтерфейсу інтелектуального аналізу інформаційних ресурсів

Інтерфейс сторінки для інтелектуального аналізу інформаційних ресурсів складається із кількох основних частин:

- інтерактивне поле для введення тексту для інтелектуального аналізу;
- блок із статистикою документа;
- блок із результатами перевірки “настрою” тексту;
- блок із переліком виділених із тексту сутностей;
- блок із результатами перевірки частоти слів.

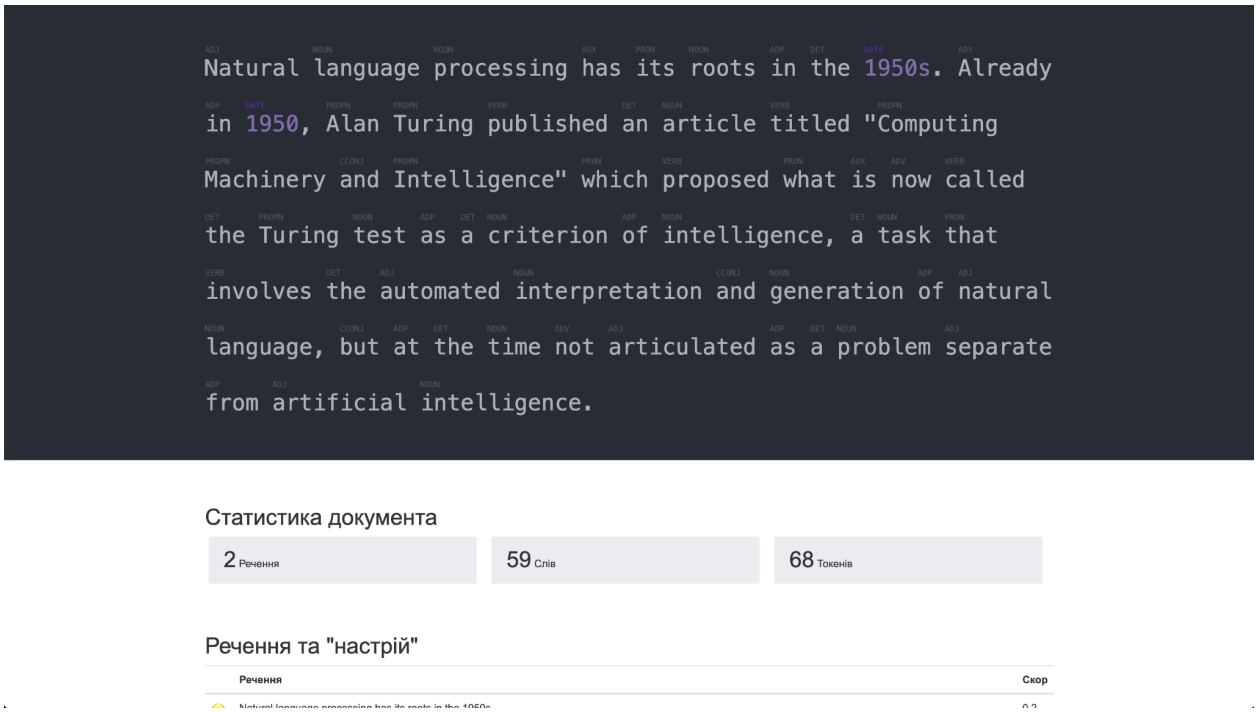

Рис. 3.9. Інтерфейс сторінки інтелектуального аналізу

Сторінка розроблена на базі статичного шаблону HTML. Стилізація реалізована за допомогою CSS. Інтерактивність: отримання даних та їх відображення функціонує на базі JS-коду.

3.4.1. Розробка інтерфейсу інтерактивного поля для вводу тексту

Інтерактивне поле для вводу тексту може бути реалізоване декількома способами.

- Елемент “textarea” – мультирядкове поле для введення тексту. Просте та надійне рішення для отримання тексту від користувача. Працює швидко та не має проблем із сумісністю у різних версіях браузеру. До недоліків можна віднести обмежений функціонал, відсутність підтримки форматування, а також можливості реалізувати підсвітку чи примітки до складових введеного тексту.
- Використання елементу “div” з атрибутом “contenteditable”. Цікаве рішення що стало доступний у нових версіях браузеру. Дає можливість створити підсвітку частин тексту. До недоліків можна віднести те, що редагування в різних браузерах було важким протягом тривалого часу через відмінності в створеній розмітці між браузерами. Наприклад, навіть те, що відбувається, при натисканні кнопки “Enter” (щоб створити новий рядок тексту всередині елемента), обробляється по-різному в основних браузерах (Firefox вставляє елементи
, IE/Opera використовує <p>, Chrome/Safari використовує). Подібні проблеми значно ускладнюють використання цього формату отримання даних від користувача, але тим не менш його переваги перевищують недоліки і саме елемент із атрибутом “contenteditable” був використаний при розробці додатку.

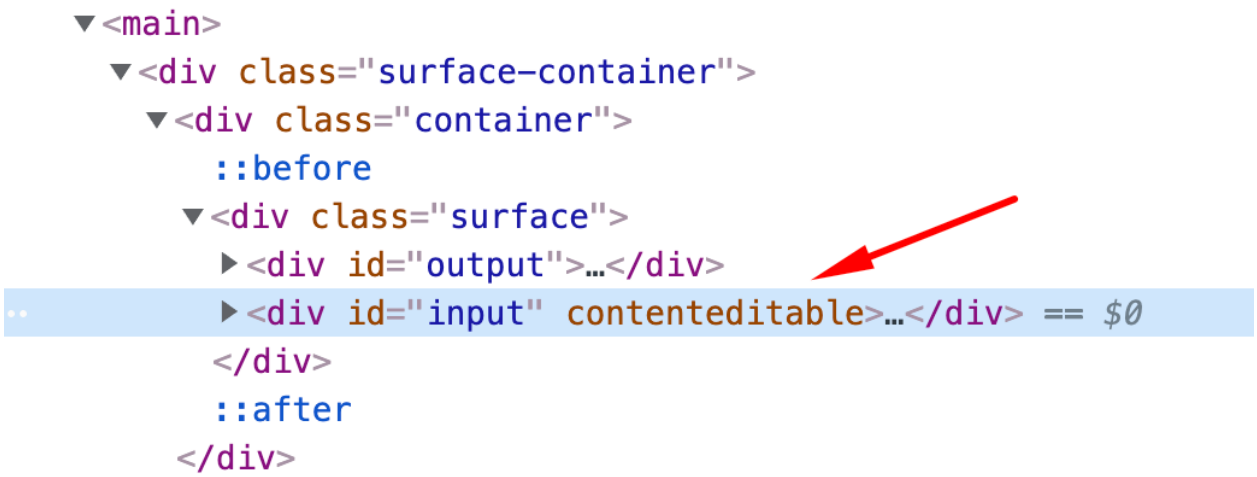

Рис. 3.10. HTML-код інтерактивного поля для вводу тексту

При використанні HTML тегу із атрибутом “contenteditable” потрібно бути дуже обережним не тільки в питанні відмінностей тонкостей його функціональної реалізації між браузерами, а й в питанні того, що отриманий елемент підтримує збереження та відображення усіх типів контенту, а не тільки тексту, зокрема:

- зображення;
- відео;
- аудіо;
- iFrame;
- тощо.

З метою обмеження типів контенту, що може бути доданий до обробку використовується додатковий функціонал, реалізований із використанням підписки на подію, що викликав користувач при виконанні операції “Вставка” у поле для введення контенту для перевірки.

Логіка частки даних що користувач надає у форму реалізована на базі браузерного API “clipboardData” та “execCommand”. JS-код, що реалізує логіку очистки наданих користувачем даних представлений на Рис 3.11.

```
input.addEventListener('paste', (e) => {
  e.preventDefault();
  const content = e.clipboardData.getData("text/plain");
  document.execCommand("insertHTML", false, content);
});
```

Рис. 3.11. JS-код очистки наданих користувачем даних

Додаток підтримує функціонал перевірки нового контенту при зміні його значення у полі для вводу тексту. Це реалізується шляхом використання браузерних можливостей додавання обробника до певного івенту користувача. Зокрема для реалізації логіки перевірки тексту при його зміні був використаний івент під назвою “keyup”. JS-код, що реалізує логіку нової перевірки тексту при його зміні представлений на Рис.3.12.

```
input.addEventListener('keyup', function () {
  getInfo(input.innerText);
  return false;
});
```

Рис. 3.12. JS-код, що реалізує логіку нової перевірки тексту при його зміні

Після успішної перевірки нового тексту редактор отримує HTML код змісту змісту DIV тегу із підписаними типами слів, сутностями.

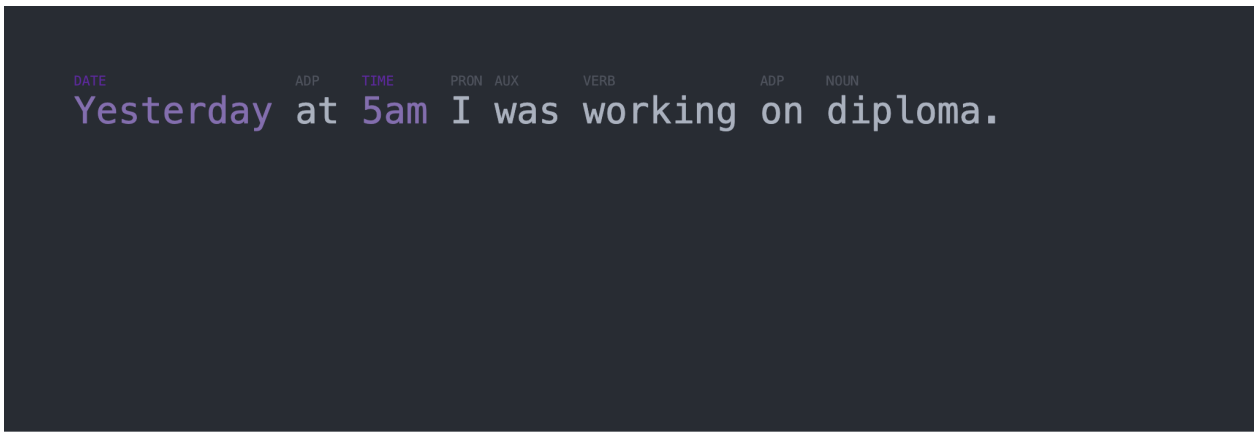

Рис. 3.13. Результат тегування введеного для аналізу тексту

3.4.2. Блок із статистикою тексту

Блок із статистикою аналізованого тексту знаходиться відразу під інтерактивним редактором. Він також оновлює дані щоразу, коли змінюється значення аналізованого тексту. Сам блок складається із наступних складових:

- кількість речень;
- кількість слів;
- кількість токенів.

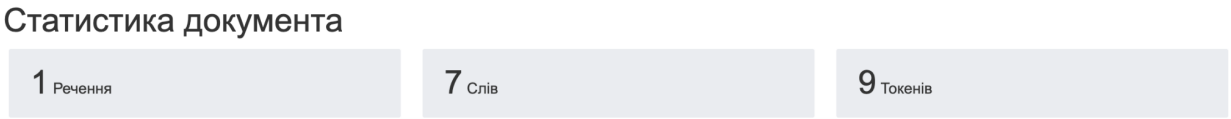

Рис. 3.14 – Блок із статистикою тексту

3.4.3. Блок аналізу настрою тексту

Аналіз настрою тексту – важлива складова при інтелектуальному NLP аналізі контенту. Може бути використаний для різноманітних завдань, зокрема збору статистики лояльності до бренду, політичної партії чи окремого продукту.

Блок аналізу настрою тексту відображає результати у вигляді табличка. Табличка складається із трьох стовпців:

- емоджи-смайлик, що відображає настрій речення;
- саме речення;
- цифрове значення “настрою”.

Кожен новий рядок у таблиці – окреме речення. Проаналізувати загальний “настрій” тексту можна підсумувавши цифрове значення для усіх речень.

Звісно краще розглядати кожне речення окремо. У поєднанні із функціоналом отримання сутносте, що були згадані у тексті цей інструмент надає потужні можливості для маркетингового або будь-якого іншого соціального аналізу.

Речення та "настрій"

	Речення	Скор
😕	Yesterday I watched really awful movie.	-0.6
😊	I've spent a great evening with her yesterday.	0.6
😊	The movie we have watched was very interesting, I'm highly suggesting you to watch it.	0.4
😊	Satisfied with the purchase.	0.4
😶	The sound is clear.	0
😊	Bluetooth does not disappear.	0.2
😊	Heavy in the hands, it feels solid	0.4
😶	You can take just that for nature.	0
😊	Well assembled, long battery life at medium power, good sound.	0.7035
😕	The low frequencies are a bit lacking, the radio is unsuccessfully implemented	-0.625

Рис. 3.15. Блок із статистикою “настрою” тексту

3.4.4. Блок аналізу сутностей

Після блоку аналізу “настрою” сторінка перевірки містить блок із переліком отриманих із тексту сутностей. Ідентифікуються наступні типи сутностей:

- дати;
- час;
- URL-адреси;
- вирази, що описують кількість грошей;
- вирази, що описують черговість;
- відсотки;
- тривалість;
- хештеги.

Усі сутності відображаються одна за одному в рядок. Кожен елемент містить дві складові:

- тип сутності;
- слово.

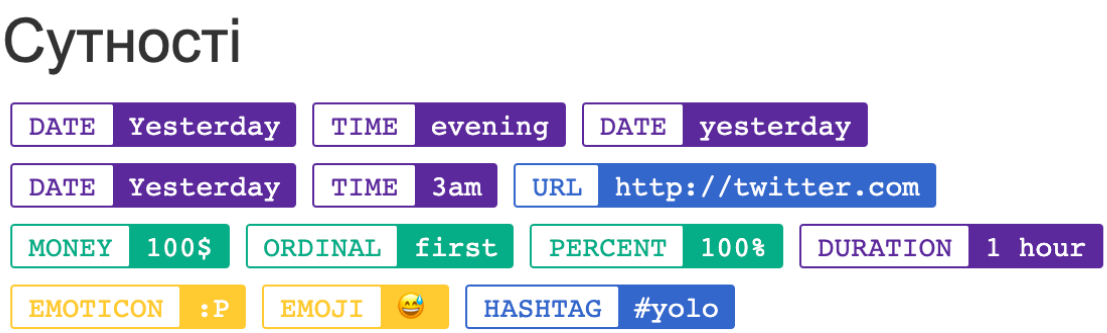

Рис. 3.16. Блок із переліком ідентифікованих сутностей

3.4.5. Блок аналізу частоти слів

Завершаючим блоком сторінки результатів інтелектуального аналізу інформаційного тексту є блок, що надає інформацію по частоті слів, котрі вживаються в тексті. Цей блок відображає інформацію отриману від серверного API. Основна мета цього блоку даних є ознайомлення користувача із статистикою частоти використання слів що присутні в тексті.

Статистика базується на системному підході що містить в собі алгоритми стемінгу та підрахунку результатів на основі статистичних даних отриманих після очистки тексту та переведення його у стеммінгову форму. Результат обрахунку відображається у зручному для користувача вигляді, а саме – таблиці.

Таблиця, що відображає результати підрахунку статистики частоти використання слів має два стовпці: слово та його частота.

Кожен рядок цієї таблиці – окреме слово, для котрого була прорахована частота використання.

Результати відсортовані по частоті використання від більшого до меншого.

Частота слів

Слово	Частота
yesterday	3
spent	2
watched	1
awful	1
movie	1

Рис. 3.17. Блок статистики частоти слів

Висновки до розділу 3

В рамках даного розділу були описані принципи та результати розробки засобу засобу інтелектуального аналізу інформаційних ресурсів мережі інтернет. Були детально розглянуті основні складові додатку, зокрема: головна сторінка, сторінки авторизації та сторінка безпосереднього аналізу.

Були детально розібрані усі складові кожної із сторінок, принципи їх роботи та взаємодії з серверною частиною.

Опис взаємодії із серверною частиною також містив в собі детальний огляд принципи обробок помилок на внепланових ситуацій.

Результат розробки складається із багатьох модулів що комбінуючись між собою формують широкі аналітичні можливості.

ВИСНОВКИ

В процесі роботи над дипломним проектом була проаналізована проблематика інтелектуального аналізу інформаційних ресурс. Ця тема є актуальною та важливою, адже структуризація інформації сьогодні надає потужні можливості для оптимізації бізнес-процесів.

Широко використовується в організаціях, орієнтованих на знання, аналіз тексту — це процес вивчення великих колекцій документів, щоб знайти нову інформацію або допомогти відповісти на конкретні питання дослідження.

Видобуток тексту визначає факти, відносини та твердження, які інакше залишилися б похованими в масі текстових великих даних. Після вилучення ця інформація перетворюється в структуровану форму, яку можна додатково аналізувати або представляти безпосередньо за допомогою групованих таблиць HTML, ментальних карт, діаграм тощо. Для обробки тексту використовуються різноманітні методики, одна з найважливіших з них. це обробка природної мови (NLP).

Структуровані дані, створені за допомогою аналізу тексту, можна інтегрувати в бази даних, сховища даних або інформаційні панелі бізнес-аналітики та використовувати для описової, директивної або прогнозної аналітики.

Розуміння природної мови допомагає машинам «читати» текст, імітуючи здатність людини розуміти природну мову, наприклад англійську, іспанську чи китайську. Обробка природної мови включає як розуміння природної мови, так і створення природної мови, яка моделює здатність людини створювати текст природною мовою, наприклад. узагальнити інформацію або взяти участь у діалозі.

Як технологія, обробка природної мови досягла повноліття за останні десять років, і такі продукти, як Siri, Alexa і голосовий пошук Google, використовують НЛП для розуміння запитів користувачів і відповіді на них. Складні програми для вивчення тексту також були розроблені в таких різноманітних областях, як медичні дослідження, управління ризиками, обслуговування клієнтів, страхування (виявлення шахрайства) та контекстна реклама.

Сучасні системи обробки природної мови можуть аналізувати необмежену кількість текстових даних без втоми та послідовним, неупередженим чином. Вони можуть розуміти поняття в складних контекстах і розшифровувати неоднозначність мови, щоб витягти ключові факти та взаємозв'язки, або надати резюме. Враховуючи величезну кількість неструктурованих даних, які створюються щодня, від електронних медичних карт (EHR) до публікацій у соціальних мережах, ця форма автоматизації стала важливою для ефективного аналізу текстових даних.

Машинне навчання — це технологія штучного інтелекту (ШІ), яка надає системам можливість автоматично вчитися на досвіді без необхідності явного програмування, і може допомогти вирішувати складні проблеми з точністю, яка може конкурувати або навіть іноді перевершувати людей.

Однак машинне навчання вимагає добре підібраних введених даних для навчання, і вони зазвичай недоступні з таких джерел, як електронні медичні записи (EHR) або науковою літератури, де більшість даних є неструктурованим текстом.

При застосуванні до EHR, записів клінічних випробувань або повної текстової літератури, обробка природною мовою може витягти чисті, структуровані дані, необхідні для стимулювання передових моделей прогнозування, що використовуються в машинному навчанні, тим самим зменшуючи потребу в дорогих, ручних анотаціях навчальних даних.

Хоча традиційні пошукові системи, як-от Google, тепер пропонують уточнення, такі як синоніми, автозаповнення та семантичний пошук (історія та контекст), переважна більшість результатів пошуку вказують лише на місцезнаходження документів, залишаючи шукачам проблему необхідності витрачати години вручну. отримання необхідних даних шляхом читання окремих документів.

Обмеження традиційного пошуку посилюються зростанням обсягу великих даних за останнє десятиліття, що допомогло збільшити кількість результатів, які повертає один запит пошуковою системою, як-от Google, з десятків тисяч до сотень мільйонів.

Сектори охорони здоров'я та біомедицини не є винятком. Дослідження, проведене Міжнародною корпорацією даних (IDC) у грудні 2018 року, показало, що

обсяг великих даних, за прогнозами, зростатиме швидше в галузі охорони здоров'я, ніж у виробництві, фінансових послугах чи медіа протягом наступних семи років: спостерігається сукупний річний темп зростання (CAGR) у 36%.

Онтології, словники та користувацькі словники — це потужні інструменти, які допомагають у пошуку, вилученні та інтеграції даних. Вони є ключовим компонентом багатьох інструментів аналізу тексту та надають списки ключових понять з іменами та синонімами, які часто розташовані в ієрархії.

Пошукові системи, інструменти текстової аналітики та рішення для обробки природної мови стають ще потужнішими, якщо вони впроваджені з онтологіями, що стосуються домену. Онтології дозволяють зрозуміти справжнє значення тексту, навіть якщо воно виражається різними способами (наприклад, Tylenol проти Acetaminophen). Техніки НЛП розширюють силу онтологій, наприклад, дозволяючи зіставляти терміни з різним написанням і беручи до уваги контекст.

Специфікація онтології включає словник термінів і формальні обмеження щодо її використання. Підготовлена до підприємства обробка природної мови вимагає ряду словників, онтологій та пов'язаних стратегій для визначення понять у їх правильному контексті.

В ході роботи над дипломним проектом був розроблений додаток що дозволяє засобами сучасних технологій проаналізувати текст та отримати аналітичні його дані.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Итан Браун. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. = Web Development with Node and Express / Итан Браун;. — Санкт-Петербург: Питер, 2017. — 336 с.
2. Дакетт, Джон. Javascript и jQuery. Интерактивная веб-разработка. — М., 2017. — 640 с.
3. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 4-е изд. — СПб.. — М.: «Диалектика», 2016. — 768 с.
4. Фримен Эрик, Фримен Элизабет. Изучаем HTML, XHTML и CSS = Head First HTML with CSS & XHTML. — П.: «Питер», 2010. — 656 с.
5. Дэвид Сойер Макфарланд. Новая большая книга CSS = CSS: The Missing Manual. — Санкт-Петербург: Питер, 2017. — 720 с.
6. Гото Келли, Котлер Эмили. Веб-редизайн, 2-е издание. — СПб.: «Символ-Плюс», 2006. — 416 с.
7. Дженнифер Нидерст Роббинс. HTML5, CSS3 и JavaScript. Исчерпывающее руководство = Learning Web Design, 4th Edition / пер. англ. М.А. Райтман. — 4-е издание. — М.: «Эксмо», 2014. — 528 с.
8. Питер Лабберс, Брайан Олберс, Фрэнк Салим. HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений = Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. — М.: «Вильямс», 2011. — 272 с.