Державний вищий навчальний заклад

«Прикарпатський національний університет імені Василя Стефаника»

Факультет математики та інформатики  
Кафедра комп’ютерних наук та інформаційних систем

**ДИПЛОМНА РОБОТА**

на здобуття другого (магістерського) рівня вищої освіти

на тему

«Аналіз неструктурованих текстових даних»

Виконав: студент 6 курсу,

групи КНМ-21

спеціальності 122 «Комп’ютерні науки»

Глодан О.З.

Керівник: кандидат технічних наук, доцент Горєлов В.О.

Рецензент: к.т.н. доц. Іляш Ю.Ю.

Івано-Франківськ – 2021 р.

Державний вищий навчальний заклад

«Прикарпатський національний університет

імені Василя Стефаника»

Факультет математики й інформатики

Кафедра комп’ютерних наук та інформаційний технологій

Освітньо-кваліфікаційний рівень магістра

Спеціальність 122 Комп’ютерні науки

Затверджено на засіданні кафедри\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Протокол №\_\_\_\_\_ від\_\_\_\_\_\_\_  
Завідувач кафедри

Петришин Л. Б. \_\_\_\_\_\_\_\_\_\_\_\_

**ЗАВДАННЯ**

**НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ**

Глодану Оресту Зіновійовичу

1. Тема роботи: Аналіз неструктурованих текстових даних

Керівник роботи: кандидат технічних наук, доцент Горєлов В.О

1. Перелік питань, які потрібно розробити: розробка методів аналізу несруктурованих текстових даних, сервіс чату та аналізу тональності відгуків на книги
2. Дата видачі завдання: 23.09.2020

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| **п/п** | **Назва етапів роботи** | **Строк виконання етапів роботи** | **Примітка** |
| 1 | Пошук джерел, наукових публікацій та дослідження алгоритмів | 10.11.2020 – 05.01.2021 |  |
| 2 | Вибір технологій для написання програми. | 05.01.2021 – 20.01.2021 |  |
| 3 | Вибір алгоритмів для реалізації проектного завдання | 20.01.2021 – 14.02.2021 |  |
| 4 | Аналіз документації модулів | 14.02.2021 – 05.03.2021 |  |
| 5 | Проектування бази даних | 05.03.2021 – 15.03.2021 |  |
| 6 | Реалізація серверної частини | 15.03.2020 – 26.06.2021 |  |
| 7 | Проектування інтерфейсу програми | 26.06.2021 – 04.07.2021 |  |
| 8 | Реалізація Front-end модулів для імплементації програмного інтерфейсу | 04.07.2021 – 05.09.2021 |  |
| 9 | Аналіз та рефакторинг коду серверної та клієнтської частини. | 05.09.2021 – 08.10.2021 |  |
| 10 | Аналіз роботи наявних компонентів. | 08.10.2021 – 17.11.2021 |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глодан О.З

(Підпис)

Керівник робои \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горєлов В.О

(Підпис)

АНОТАЦІЯ

«Аналіз неструктурованих текстових даних»

Глодан О.З., студент, спеціальність «Комп’ютерні науки та інформаційні технології»

Кандидат технічних наук, доцент Горєлов В. О.

Івано-Франківськ – 2021 р.

У роботі наведено огляд методів аналізу неструктурованих текстових даних, досліджено алгоритм на основі теореми Байєса та обробки природньої мови. Запропоновано використання теореми для блокування не відповідних повідомлень та використання методів обробки мови та складеного рейтингу тональності слів для аналізу відгуків на книгу. Розроблено програмне забезпечення для забезпечення роботи сервісів з додатковими інтерактивними взаємодіями

Дипломна робота викладена на 101 сторінках, містить 31 рисунків, 3 таблиці, лістинги коду з модулями та компонентами, а також 26 формул.

ABSTRACT

"Analysis of unstructured text data"

Hlodan O., student, specialty "Computer Science and Information Technology"

Ass. Prof., Ph.D. Horelov V.

Ivano-Frankivsk - 2021

The paper reviews the methods of analysis of unstructured text data, investigates an algorithm based on Bayes' theorem and natural language processing. The use of the theorem for blocking inappropriate messages and the use of language processing methods and compiled word tonality rating for the analysis of responses to the book are proposed. Software has been developed to ensure the operation of the feedback and discussion service with additional interactive interactions with the content of the services.

The diploma is presented in 101 pages, contains 31 figures, 3 tables, code listings with modules and components, as well as 26 formulas.

ЗМІСТ

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 10](#_Toc90601880)

[ВСТУП 12](#_Toc90601881)

[РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ОПРАЦЮВАННЯ НЕСТРУКТУРОВАНОЇ ІНФОРМАЦІЇ 15](#_Toc90601882)

[1.1 Інформація. Типи інформації. 15](#_Toc90601883)

[1.2 Проблеми аналізу текстових даних методами машинного навчання. 19](#_Toc90601884)

[1.3 Метадані. NLP. Як засіб обробки неструктурованих текстових даних. 20](#_Toc90601885)

[1.3.1 Метадані. 20](#_Toc90601886)

[1.3.2 NLP 21](#_Toc90601887)

[1.4 Методи машинного навчання аналізу неструктурованих текстових даних. 23](#_Toc90601888)

[1.4.1 Загальний огляд методів машинного навчання. 23](#_Toc90601889)

[1.4.2 Метод Байєса. 24](#_Toc90601890)

[1.4.3 Метод опорних векторів. 25](#_Toc90601891)

[1.4.4 Дерево прийняття рішень. 26](#_Toc90601892)

[1.4.5 Лінійний дискриммінаційний аналіз. 28](#_Toc90601893)

[1.4.6 Випадковий ліс. 29](#_Toc90601894)

[1.5 Постановка задачі. 30](#_Toc90601895)

[1.5.1 Мета проекту. 31](#_Toc90601896)

[1.5.2. Тема проекту. 31](#_Toc90601897)

[1.6 Висновки 32](#_Toc90601898)

[РОЗДІЛ 2 МОДЕЛЮВАННЯ ОСНОВНИХ АЛГОРИТМІВ АНАЛІЗУ ТЕКСТОВИХ ДАНИХ 33](#_Toc90601899)

[2.1 NLP. Методи та їх моделі. 33](#_Toc90601900)

[2.1.1 Tokenization. 33](#_Toc90601901)

[2.1.2 Stemming and Lemmatization 34](#_Toc90601902)

[2.1.2.1 Алгоритм Портера Стеммера 35](#_Toc90601903)

[2.1.3 Bag of words 38](#_Toc90601904)

[2.1.4. Stop words removal. 39](#_Toc90601905)

[2.1.5 Висновки 40](#_Toc90601906)

[2.2 Наївний класифікатор Байєса. 40](#_Toc90601907)

[2.3 SVM. Support Vector Machine 50](#_Toc90601908)

[2.3.1 Основні поняття про Support Vector Machine. 50](#_Toc90601909)

[2.3.1.1 SVM та лінійні проблеми 51](#_Toc90601910)

[2.3.1.2. SVM та нелінійні проблеми 54](#_Toc90601911)

[РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУ НЕСТРУКТУРОВАНИХ ТЕКСТОВИХ ДАНИХ 58](#_Toc90601912)

[3.1 Вибір технологій. 58](#_Toc90601913)

[3.2 Обгрунтування вибраних технологій. 58](#_Toc90601914)

[3.2.1 Web Sockets. Введення. 58](#_Toc90601915)

[3.2.1.1. Long polling (HTTP) 60](#_Toc90601916)

[3.2.1.2 Web sockets. Детальний огляд. 61](#_Toc90601917)

[3.2.1.3. Аналіз 63](#_Toc90601918)

[3.2.1.4. Висновки 65](#_Toc90601919)

[3.2.2. Node.js та Express.js 66](#_Toc90601920)

[3.2.2.1. Node, як засіб розробки серверів. 66](#_Toc90601921)

[3.2.2.2 Express.js 69](#_Toc90601922)

[3.2.3 Бази даних. PostgreSQL. 70](#_Toc90601923)

[3.2.3.1 Основні поняття про БД. 70](#_Toc90601924)

[3.2.3.2 Важливість СКБД. 72](#_Toc90601925)

[3.2.3.3. PostgreSQL 73](#_Toc90601926)

[3.2.3.4. Sequelize 74](#_Toc90601927)

[3.2.4. JWT Token. Авторизація. 75](#_Toc90601928)

[3.2.5. AWS S3 Bucket. 78](#_Toc90601929)

[3.3. Практична реалізація. 79](#_Toc90601930)

[3.3.1. Імплементація сервісу аутентифікації та авторизації. 79](#_Toc90601931)

[3.3.2. Процес імплементації чату, та спам-фільтру. 85](#_Toc90601932)

[3.3.2.1 Визначення схем чату. 85](#_Toc90601933)

[3.3.2.2 Імплементація веб-сокетів. 87](#_Toc90601934)

[3.3.2.3. Імплементація алгоритму Байєса. 90](#_Toc90601935)

[3.3.2.4 Візуалізація чату 92](#_Toc90601936)

[3.3.3 Імплементація аналізу відгуку. 93](#_Toc90601937)

[3.4.1 Дослідження розробленої системи 96](#_Toc90601938)

[ВИСНОВКИ 99](#_Toc90601939)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 101](#_Toc90601940)

[Додаток А. 103](#_Toc90601941)

[Додаток Б 108](#_Toc90601942)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ML – Machine Learning (машинне навчання).

Data Mining – Добування даних, глибинний аналіз даних.

Data Science – Наука про дані.

WS– web sockets.

MVC – Model View Controller, архітектурний шаблон, що використовується для розробки програмного забезпечення.

ORM – Object Relation Mapping.

ООП – Об’єктно-орієнтоване програмування.

БД – База даних.

SQL – Structured Query Language, мова програмування для взаємодії користувача з базами даних.

API – Application Programming Interface.

СКБД **–** система керування базами даних.

JSON **–** JavaScript Object Notation**,** запис об’єктів JSON.

JWT – JSON Web Token.

SWM **–** support vector machine

NLP **–** Natural language processing

SWR **–** Stop words removal

LDA **–** Linear discriminant analysis

HTTP **–** HyperText transfer protocol

AWS **–** Amazon Web Service

NBC **–** Naïve Bayes Classifier

TDM **-** Матриця термінового документа

# ВСТУП

*Мета роботи.*Проаналізувати методи обробки нестуктурованих текстових даних та застосувати їх в розробці програмної платформи з продажу та пошуку книг.

*Актуальність.*Попит породжує пропозиції. В силу великих потреб організацій швидко та якісно обробляти тонну неструктурованої інформації - це породжує велику потребу фахівців, котрі можуть реалізувати поданий процес. За допомогою таких алгоритмів, компанії можуть покращувати процеси в маркетингу, виправляти та покращувати свій сервіс з чим збільшувати своїх клієнтів, а з ними і доходи. Вдосконалити процеси статистики та аналізу продукту, основуючись на відгуках клієнтів, що дасть змогу уникати помилок в подальшому. Подані приклади зможуть зберегти кошти, та більше того збільшити прибутки. Виходячи з цього можна зробити висновок, що методи аналізу неструктурованої інформації дадуть наступне:

* прибутки;
* економію часу;
* покращення сервісу;
* статистику та аналіз продукту;

*Практична значимість.* Інформація та дані заповнюють життя. Важко уявити світ без інформації, вона є повсюду, починаючи від новин в газетах закінчуючи телебаченням чи інтернетом. Дані можуть бути представлені в багатьох видах: чи то зображення та відео, музика, звичайний текст в статтях чи книжках, в листах чи в мейлах. Дані можна поділити на два типи: структуровані та неструктуровані. Щодо перших ми можемо представити їх у вигляді excel таблиць, баз даних, чи звичайної категоризації відео чи музики. Це все що ми можемо розмістити у правильному чи звичному нам порядку. Інший вид даних - неструктуровані дані. До цих можна віднести відео, музику, листи, статті, чатування в соціальних мережах. Все що знаходиться за межами чітко визначених меж бази даних чи категорій. Таку інформацію важче обробляти та працювати з нею. Відсоток такої інформації за певними підведеними статистиками складає 80-90%. Що означає, масу корисної інформації, яку проаналізувати та упорядкувати, задача не з простих.

В силу цього, впродовж вже декількох років все більшої популярності набирає галузь програмування «Машинне начання». За допомогою алгоритмів машинного навчання ми можемо проаналізувати неструктуровані дані, такі як текст, мейли, відео, аудіо та багато іншого. Ми можемо реалізувати алгоритм який б виявляв та правильно опрацьовував спам в листах, блокував листи чи статті, які містять неприйнятний вміст, категоризував статті за певними тегами чи темами та багато іншого.

Згадана методологія вирішує ряд проблем, з якими можна зіштовхнутись в бізнесі.

В ідеальному світі для організацій та компаній всі дані є структурованими - чітко відсортованими за категоріями, мітками, стовпцями та полями, синхронізовані та зібрані по всій організації, так як вони доступніші, з ними легше і швидше працювати. А економія часу є критично важливою.

*Новизною* в дослідженні є застосування підходів до обробки неструктурованих даних, для покращення роботи сервісу з онлайн продажу, та пошуку книг

*Об’єктом дослідження* є неструктуровані дані, їх відмінності від структурованих.

*Предметом дослідження є* методи аналізу неструктурованих даних.

*Методи дослідження.* Аналіз та використання методів обробки природної мови та методів машинного навчання для аналізу неструктурованих тестових даних.

*Апробація результатів роботи.* Роботу було апробовано на міжнародній науково-технічній конференції здобувачів вищої освіти та молодих вчених «Комп’ютерні науки, інформаційні технології та системи управління» (CSYSC). На Konferencja Studenckich Kół Naukowych Pionu Górniczego AGH, робота посіла 1 місце на конкурсі наукових робіт що проводиться в рамках конференції.

*Структура роботи.* Робота складається зі вступу, трьох розділів та висновки. Робота містить 3 таблиці, 30 рисунків, 26 формул та додатки.

# РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ОПРАЦЮВАННЯ НЕСТРУКТУРОВАНОЇ ІНФОРМАЦІЇ

## 1.1 Інформація. Типи інформації.

Дані - це джерело життя бізнесу, і вони надходять у величезну різноманітність форматів - від строго сформованих реляційних баз даних до вашого останнього повідомлення у Twitter. Усі ці дані у всіх різних форматах можна сортувати в одну з двох категорій: структуровані та неструктуровані дані.

Що ж таке структуровані дані?

Структуровані дані - це дані, які були заздалегідь визначені та відформатовані до встановленої структури перед їх розміщенням у сховищі даних, яке часто називають схемою для запису. Найкращим прикладом структурованих даних є реляційна база даних: дані відформатовані у точно визначені поля, такі як ім’я, прізвище, серія паспорту, адреса, тощо. Для того, щоб їх можна було легко запитувати за допомогою SQL (декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних).

Розглянемо плюси структурованих даних, вони мають три основні переваги:

- Легко використовується алгоритмами: Найбільшою перевагою структурованих даних є те, наскільки легко вони можуть бути використані за допомогою машини. Специфічний та організований характер структурованих даних дозволяє легко маніпулювати та запитувати ці дані;

- Легко використовується алгоритмами бізнес-користувачами: Ще однією перевагою структурованих даних є те, що вони можуть бути використані простим бізнес користувачем з розумінням теми, до якої стосуються дані. Немає необхідності глибоко розуміти різні типи даних або взаємозв’язок цих даних. Це відкриває бізнес-користувачам доступ до даних самообслуговування.

- Легкі в підтримці. Над такими даними легко проводити різного роду операції, такі як: сортування, фільтрування, оновлення.

Мінуси структурованих даних зосереджені у відсутності гнучкості даних. Ось деякі потенційні недоліки використання структурованих даних:

- Заздалегідь визначена мета обмежує використання: Хоча визначення даних у схемі запису є великою перевагою для структурованих даних, правда також, що дані із заздалегідь визначеною структурою можна використовувати лише за призначенням. Це обмежує її гнучкість і варіанти використання.

- Обмежені можливості зберігання: Структуровані дані зазвичай зберігаються в сховищах даних. Сховища даних - це системи зберігання даних із жорсткими схемами. Будь - яка зміна вимог означає оновлення всіх цих структурованих даних для задоволення нових потреб; це призводить до значних витрат ресурсів і часу. Частину вартості можна зменшити за допомогою хмарного сховища даних, оскільки це забезпечує більшу масштабованість та усуває витрати на технічне обслуговування, створюване наявністю обладнання на місці. Значні витрати ресурсів, для клієнта та власника бізнесу означають суттєві втрати у доходах, та бюджету компанії.

Структуровані дані - це основа для систем управління запасами та банкоматів. Він може бути створений людиною або машиною. Це ті дані, які розробники програмного забезпечення використовують щоденно, вони можуть бути представлені у різних форматах даних, таких як json, xml та інші. Зберігаються в хмарних сховища, базах данних, різного роду файлах, xls, csv і т.д.

Іншим видом даних є неструктуровані дані. Це дані, що зберігаються у своєму рідному форматі і не обробляються до їх використання. Ці дані, можна знайти повсюду, в електронній пошті, публікаціях у соціальних мережах, презентаціях, чатах, зображення, аудіо, відео на ютубі та багато іншого.

Плюси неструктурованих даних. Вони так само як структуровані мають свої слабкі та сильні сторони для конкретних потреб бізнесу.

Неструктуровані дані мають кілька позитивних моментів:

- Свобода рідного формату. Оскільки такі дані зберігаються у своєму рідному форматі, дані не визначаються поки вони не потрібні. Це призводить до збільшення кількості випадків використання, оскільки призначення даних є адаптивним. Це дозволяє вченим з даних готувати та аналізувати лише необхідні дані. Рідний формат також дозволяє використовувати різноманітні формати файлів у базі даних, оскільки дані, які можна зберігати, не обмежені певним форматом. Це означає, що компанія має більше даних для збору.

- Більш швидкі темпи накопичення: Ще однією перевагою неструктурованих даних є швидкість накопичення даних. Немає необхідності заздалегідь визначати дані, а це означає, що їх можна швидко і легко зібрати.

- Велика кількість даних. Не структуровані дані часто зберігаються в хмарних «озерах» даних, що дозволяє зберігати великі їх масиви. Хмарні сховища даних також дозволяють встановлювати ціни на сховище з оплатою за використання, що допомагає скоротити витрати та полегшити масштабування.

Попри позитивні моменти, у неструктурованих даних є також свої суттєві недоліки:

- Вони вимагають знання в області Data Sciences. Найбільший недолік.

- Спеціалізовані інструменти. Через те що аналіз таких даних автоматизується відносно недавно, неструктуровані дані вимагають спеціальних інструментів для маніпулювання, що означає, що вони більш характерні та категоричні за своєю природою.

Неструктуровані дані є швидше якісними, а не кількісними, що означає, що вони більш характерні та категоричні за своєю природою.

Такий тип даних добре підходить для визначення ефективності маркетингової кампанії або розкриття потенційних тенденцій покупки через соціальні мережі та огляд веб-сайтів. Він також може бути дуже корисним для підприємства, допомагаючи моніторингу даних, виконувати свою роботу якісніше.

При аналізі різниці між двома типами даних, все зводиться до декількох пунктів. Їх можна виділити, відповівши на деякі питання. Які наведені у таблиці 1.1. Таблиця визначає декілька головних питань – відповідей:

- Хто?;

- який тип?

- де?;

- як?;

Таблиця 1.1

Структурована та не структурована інформація

|  |  |  |
| --- | --- | --- |
|  | Структуровані дані | Неструктуровані дані |
| Хто може використовувати? | Легкий у доступі | Вимагає спеціальних навичок в експертизі |
| Який тип даних підтримується? | Лише відомі типи даних | Конгломеративні типи даних |
| Де можна зберігати? | В більшості, в хмарах/базах даних | У будь-яких текста/аудіо/відео |
| Як можна зберігати? | Відомий формат (json) | Нативний формат |

Структуровані дані є дуже специфічними і зберігаються у заздалегідь визначеному форматі, де неструктуровані дані - це сукупність багатьох різноманітних типів даних, які зберігаються у їх рідних форматах. Це означає, що структуровані дані використовують переваги схеми при записі, а неструктуровані дані використовують схему при читанні. Проте неструктуровані дані дають вагому перевагу у бізнесі, так як вони несуть велику кількість корисної інформації, яку звичайно важче проаналізувати, але вона принесе набагато більше користі.

Така інформація допоможе передбачити та спрогнозувати майбутній план розвитку бізнесу, дасть деякі уявлення про різні шляхи вирішення поставлених проблем, тим самим проаналізувавши її компанія зможе уникнути труднощів у майбтньому.

## 1.2 Проблеми аналізу текстових даних методами машинного навчання.

Згідно з останнім дослідженням, як вже було згадано раніше, близько 80% усіх даних, отриманих на підприємствах, є у вигляді текстів. З нього можна отримати багато знань.

Але текстовий аналіз ML також викликає деякі проблеми:

Складність. Перетворення тексту у формат, який може бути оброблений комп’ютером, вимагає декількох кроків. Наприклад, якщо ми вирішуємо проблему класифікації тексту, нам потрібно зібрати дані, виявити ключові слова в ній, визначити кількість класів, згрупувати дані відповідно до цих класів і описати ці процеси математичними термінами. Це виклик як інтелектуально, так і з точки зору людських/грошових/часових ресурсів.

Концептуальні боротьби. Комп’ютери не розуміють понять, які стоять за словами, тому працювати з омографами їм важко. Програмісти повинні придумати деякі ефективні інструменти для визначення неоднозначності слів, щоб працювати з такими реченнями, як «Police police Police police police police Police police». Наприклад, Google Translate наразі не може впоратися з цим реченням.

Розуміння культури. Розуміння людської мови означає розуміння її емоцій. Однією з найскладніших емоцій, які сприймає комп’ютер, є сарказм. Продовжуючи тему багатозначності, одне й те саме значення в різних культурах може бути виражене різними словами, такими як сленг або місцеві варіанти. Для британця "джемпер" - це для американця "светр". Комп'ютерна програма повинна мати досвід та культурну освіту для ефективного спілкування з носіями мови, які використовують менш традиційні форми мови.

В допомогу з цими проблемами приходить технологія під назвою NLP та метадані, котрі несуть з собою ряд переваг в підготуванні даних до їх подальшою обробкою методами машиного навчання.

Вони допомагають обробити та «почистити» текст, для того, щоб алгоритмам було простіше працювати з текстом.

## 1.3 Метадані. NLP. Як засіб обробки неструктурованих текстових даних.

### 1.3.1 Метадані.

Метадані - це дані, що визначають, надають інформацію про інші дані. Вони відіграють важливу роль в управлінні, зберіганні та аналізі неструктурованих даних [1].

Яскравим прикладом таких даних, можна пригадати дані до фотографії. Коли ми робимо знімок за допомогою цифрової камери чи смартфону, кожне таке зображення має пов'язані з ним метадані, такі як дата, час, назва файлу та геолокація. Кожна публікація в блозі містить метадані, які включають заголовок, автора, URL -адресу, дату публікації, теги, категорію тощо. Веб-сторінка містить такі метадані, як заголовок сторінки, URL -адреса, опис сторінки.

На додаток до цих стандартних полів, ви можете визначити додаткові поля спеціальних метаданих на основі ваших вимог, щоб вказати характер або вміст неструктурованих даних. Таким чином, метадані можуть полегшити подальший пошук та аналіз.

Оскільки наразі немає загальногалузевих стандартів щодо метаданих, кожен розробник ПЗ може визначити власні. Ефективне використання метаданих допомагає впорядкувати, автоматизувати процес отримання даних.

### 1.3.2 NLP

NLP - natural language processing - обробка натуральної мови. Так як, неструктуровані текстові дані, в більшості випадків використовують нативний формат даних, а саме просту людську мову, їх потрібно обробляти перед аналізом, саме в такій ситуації, допомагають методи обробки натуральної мови. Їх є доволі багато, проте виділимо найважливіші з них:

- Bag of word

- Tokenization

- Stop words removal

- Stemming

- Lemmatization

- Topic modeling

Розглянемо кожну з методологій окремо.

Bag of word - основною метою даного методу є підрахунок входжень слів, фраз чи конструкцій в поданий текст, не враховуючи семантики чи контексту.

Tokenization - використовується для розбиття тексту на лексеми. Токенізація розрізає текст на речення та слова і не звертає уваги на розділові знаки.

Stop words removal - це процес видалення артиклів та прийменників, таких як «the» та «to» з тексту, оскільки вони не несуть ніякої цінної інформації для аналізу тексту. Більшість методів аналізу неструктурованих текстових даних починаються з аналізу базового списку стоп слів, після чого, вони доповнюються іншими основними словами з аналізуючого тексту.

Stemming - це процес, який видаляє афікси, або доповненням до кореневого слова за допомогою префіксу перед словом або суфіксу після слова. Вирішення може допомогти згрупувати різні форми слова, наприклад, "гітара" та "гітарист".

Lemmatization - це процес перетворення слів у їх словникову форму, відому як «лема». Наприклад, видаляються часи, тому і «навчання», і «навчене» стають «навчати». Синоніми також уніфікуються під час процесу лемматизації, наприклад, «дім» стає «будинком». Лемматизація враховує контекст вживання слова, оскільки одне й те саме слово може мати різне значення залежно від того, де і як воно вживається.

Topic modeling - це інструмент для видобутку тексту, який допомагає розкрити теми в тексті та знайти групи слів, пов’язаних з різними темами. Конкретні слова, які часто зустрічаються в документі, вказують на теми, що містяться, та їх значення. Наприклад, припустимо, що слова "економіка" та "ВВП" будуть частіше зустрічатися в документі про економіку, а "судноплавство" та "логістика" - частіше, якщо мова йде про логістику. Тоді можна вважати, що документ, що містить у 9 разів більше слів, пов'язаних з логістикою, ніж слів, пов'язаних з економікою, складає 90% про логістику та 10% про економіку.

Підсумовуючи, робимо висновок, що всі ці методи призначені для збагачення тексту таким чином, щоб він надавав ще більшої цінності.

Обробка природної мови йде рука в руку з аналізом неструктурованих текстових даних та аналітикою тексту. Вона підраховує, групує, класифікує слова для витягнення структури та значення з великого масиву даних. Аналіз тексту використовується для вивчення текстового вмісту та отримання нових змінних із необробленого тексту, які можуть бути візуалізовані, відфільтровані або використані як вхідні дані для моделей прогнозування чи інших статичних методів.

NLP є невід’ємним процесом обробки та аналізу неструктурованих масивів даних, що обробляються щоденно [2].

## 1.4 Методи машинного навчання аналізу неструктурованих текстових даних.

### 1.4.1 Загальний огляд методів машинного навчання.

Машинне навчання - це все ще про передбачення. Методи якого допомагають передбачити та відповісти на одне важливе питання - “Що дальше?”. Звідси виходить запитання. Що ми передбачаємо? Та що саме ми маємо на увазі під визначення “Що дальше?”.

Важливо розділити та визначити два важливі терміни в машинному навчанні, а саме: Дані та алгоритми.

Дані + алгоритми = Машинне навчання. Ця дуже проста формула несе в собі теперішнє та майбутнє, адже більшість популярних ресурсів та платформ використовують так чи інакше машинне навчання.

Дані - це те що подається на машину навчання. Наприклад, якщо ви намагаєтесь описати алгоритм, який буде передбачати погоду, то вам слід, організувати навчальний сет даних на основі минулих статистик, так щоб список включав максимальну та мінімальну точку, наприклад температури повітря, швидкість руху вітру чи будь що інше.

Якщо ми уважно спостерігатимемо, у вхідних даних, які ми маємо, завжди буде той чи інший малюнок. Наприклад, максимальний і мінімальний діапазони температур можуть потрапляти в одну область; або швидкість вітру може бути дещо подібною для певного сезону тощо. Машинне навчання допомагає дуже глибоко аналізувати такі закономірності. І тоді дає змогу спрогнозувати результати проблеми, для якої ми її розробили.

Алгоритм аналізує, представляє, знаходить пробіли та заповнює їх.

Алгоритми - це методи, які використовуються машинами для роботи над введенням даних до них.

### 1.4.2 Метод Байєса.

Це метод класифікації, заснований на теоремі Байєса з припущенням незалежності серед предикатів. Простими словами, класифікатор Naive Bayes припускає, що наявність певного об’єкта в класі не пов’язана з наявністю будь-якого іншого об’єкта [3].

Наприклад, фруктом можна вважати банан, якщо він жовтий чи зелений, має форму дуги. Навіть якщо ці ознаки залежать один від одного або від наявності інших ознак, усі ці властивості незалежно один від одного сприяють збільшенню ймовірності того, що цей фрукт є яблуком, і тому він відомий як «Naive».

Модель Naive Bayes проста у побудові та особливо корисна для дуже великих наборів даних. Поряд із простотою, відомо, що Naive Bayes перевершує навіть дуже складні методи класифікації.

Розглянемо плюси цього алгоритму.

- Легкий в реалізації

- Він не вимагає багато навчальних даних

- Обробляє як безперервні так і дискретні дані

- Він легкий у масштабуванні в кількостях прогнозів та точок даних

- Швидкий. Його можна використовувати для прогнозування в реальному часі

### 1.4.3 Метод опорних векторів.

Одним з найпопулярніших методів класифікації є метод опорних векторів (SVM). Він також вирішує регресійні проблеми, проте на даному етапі нас цікавлять його класифікаційні властивості.

Метою алгоритму є створення «ідеальної» лінії або межі рішення, яка може розділити n-вимірний простір на класи, так щоб ми легко могли визначити положення нової точки даних в категорії. Цю межу також називають гіперплощиною.

SVM вибирає крайні точки (вектори), які допомагають у створенні гіперплощини. Ці крайні випадки називаються опорними векторами, від цього і назва метод опорних векторів [4].

Плюси:

- Він працює дуже добре з чітким полем розмежування

- Він ефективний у просторах великих розмірів.

- Він ефективний у випадках, коли кількість розмірів більша за кількість зразків.

- Він використовує підмножину навчальних точок у функції прийняття рішень (так званих опорних векторах), тому він також ефективно працює з пам'яттю.

Мінуси:

- Він не працює добре, якщо у нас великий набір даних, оскільки необхідний час навчання вищий

- Це також не дуже добре працює, коли у наборі даних більше шуму, тобто цільові класи перекриваються

- SVM безпосередньо не надає оцінок ймовірності, вони розраховуються за допомогою дорогої п'яти кратної перехресної перевірки. Він включений у відповідний метод SVC бібліотеки scikit-learn Python.

### 1.4.4 Дерево прийняття рішень.

Дерево прийняття рішень - це метод, який надає вирішення правил в ієрархічній послідовності. Він складається з двох елементів - вузли (node) та листки (leaf). У вузлах знаходяться правила які потрібно вирішити, відповідно проводиться перевірка відповідності прикладів до цього правила, по будь-якому атрибуту, множини яка навчається. [5]

В найпростішому варіанті, в результаті перевірки, множина прикладів, попавших у вузол, розбивається на дві підмножини, в одну з яких попадають приклади, які задовольняють правило, а другі ні.

Після чого до кожної підмножини знову застосовується правило і процедура повторюється рекурсивно, до тих пір, поки не буде досягнуто деяке стоп слово алгоритма. В результаті, в останньому вузлі перевірка і розбиття не відбудеться і цей вузол стане листком.

Розглянемо переваги та недоліки алгоритму.

Переваги алгоритму:

- Легко читати та тлумачити

Однією з переваг дерев рішень є те, що їх результати легко читаються та інтерпретуються, не вимагаючи статистичних знань. Наприклад, при використанні дерев рішень для представлення демографічної інформації про клієнтів, співробітники відділу маркетингу можуть читати та інтерпретувати графічне представлення даних, не вимагаючи статистичних знань. Дані також можуть дати важливу інформацію про ймовірності, витрати та альтернативи різним стратегіям, сформульованим відділом маркетингу.

- Легкий у підготовці

Порівняно з іншими методами прийняття рішень, дерева рішень докладають менше зусиль для підготовки даних. Однак користувачі повинні мати готову інформацію для створення нових змінних з можливістю передбачення цільової змінної. Вони також можуть створювати класифікації даних без складних обчислень. У складних ситуаціях користувачі можуть поєднувати дерева рішень з іншими методами.

- Потрібно менше очищення даних

Ще однією перевагою дерев рішень є те, що після створення змінних потрібно менше очищення даних. Випадки відсутніх значень та відхилень мають менше значення для даних дерева рішень.

Недоліки:

- Одним з обмежень дерев рішень є те, що вони в значній мірі нестійкі в порівнянні з іншими предикторами прийняття рішень. Невелика зміна даних може призвести до значних змін у структурі дерева прийняття рішень, які можуть передати результат, відмінний від того, що користувачі отримають у звичайній події. Отриману зміну результату можна контролювати за допомогою алгоритмів машинного навчання, таких як посилення та складання пакетів.

- Менш ефективний у прогнозуванні результату безперервної змінної. Крім того, дерева рішень менш ефективні для прогнозування, коли основною метою є передбачення результату безперервної змінної. Це пояснюється тим, що дерева рішень, як правило, втрачають інформацію при класифікації змінних у кілька категорій.

### 1.4.5 Лінійний дискриммінаційний аналіз.

Лінійний дискримінаційний аналіз - це лінійний класифікатор меж прийняття рішень, створений методом підбору умовних «щільностей» класу до даних та застосування до них правила Байєса [6].

Це один з найпопулярніших методів моделювання тем. Приклад використання алгоритму дуже простий. Кожен документ складається з різних слів, і в кожній темі також є різні слова, що належать їй. Мета методу знайти теми, до яких належить документ, на основі слів у ньому.

Він також відомий, як метод зменшення розмірності. Оригінальна методика була розроблена Рональдом А. Фішером в 1936 році і отримала назву «Лінійний дискримінант» або «Дискримінантний аналіз Фішера». Оригінальний лінійний дискримінант був описаний як двокласова техніка. Пізніше версію з кількома класами було узагальнено К.Р Рао як аналіз множинної дискримінації. Всі вони просто називаються лінійним дискримінаційним аналізом.

LDA - це метод класифікації, який вважається частиною створення конкурентних моделей машинного навчання. Ця категорія зменшення розмірності використовується в таких областях, як розпізнавання зображень та прогнозований аналіз у маркетингу.

Плюси:

- Це простий, швидкий та портативний алгоритм. Він все ще перевершує деякі алгоритми (логістична регресія), коли його припущення виконуються.

Мінуси:

- Це вимагає нормального припущення про розподіл за характеристиками предикторами.

- Іноді не підходить для змінних кількох категорій.

### 1.4.6 Випадковий ліс.

Випадкові ліси в машинному навчанні - це комплексний метод навчання класифікації, регресії та інших операцій, які залежать від безлічі «дерев рішень» під час навчання. Вони швидкі, гнучкі, представляють надійний підхід до видобутку високо розмірних даних і є розширенням дерев рішень класифікації та регресії, про які ми говорили вище.

Загалом ансамблеве навчання можна визначити як модель, яка дає прогнози шляхом поєднання окремих моделей. Модель ансамблю, як правило, є більш гнучкою з меншою упередженістю та меншою дисперсією. Ансамблеве навчання має два популярних методи:

- Обмеження: кожне окреме дерево для випадкової вибірки з набору даних і тренування за випадковою підмножиною даних, що призводить до різних дерев

- Підвищення: Кожне окреме дерево /модель вчиться на помилках попередньої моделі та вдосконалюється

«Випадковий ліс» повинен мати кількість дерев між 64–128 деревами.

Різниця між випадковим лісом та деревом рішень

Випадковий ліс - це, по суті, сукупність дерев рішень. Дерево рішень будується на цілому наборі даних, використовуючи всі особливості/змінні, що представляють інтерес, тоді як випадковий ліс випадковим чином вибирає спостереження/рядки та конкретні ознаки/змінні для побудови кількох дерев рішень, а потім усереднює результати [5].

Плюси:

- Добре працює з нелінійними даними.

- Ефективно працює на великому наборі даних.

- Краща точність, ніж інші алгоритми класифікації.

Мінуси:

- Випадкові ліси виявляються упередженими при роботі з категоріальними змінними.

- Повільне навчання.

## 1.5 Постановка задачі.

З аналізу видно, які проблеми вирішують розглянуті алгоритми, на основі зроблених досліджень, зроблено висновок та поставлено задачу імплементації досліджених алгоритмів в сервіс аналізу відгуків та керування вмістом чату, запобігаючи надсилання повідомлень котрі не попадають під категорії обговорення. На основі сучасних мов програмування, технологій та методів аналізу неструктурованих текстових дани– розробити сервіси, для побудови поставлених задач.

### 1.5.1 Мета проекту.

Метою даного проекту є розробка веб-аплікації засобами мов програмування, та за допомогою алгоритмічної бази створення платформи для опрацювання неструктурованої інформації, та її подальшою обробкою для виділення з неї корисної інформації. Дослідження методів обробки природної мови та її аналіз.

### 1.5.2. Тема проекту.

Темою проекту є створення веб-аплікації – «Система книжок». Де в користувачів буде наступний функціонал.

* Створення нової книжки для обговорення.
* Перегляд існуючих книжок.
* Участь в обговоренні книжки в чатах.
* Скачування книжок з платформи.
* Перегляд відгуків на книжку.
* Перегляд статистики тональності відгуків на книжку.
* Пошук, фільтування книжок за темами.

Додатково також, реалізувати функціонал відстежування спаму в повідомленнях в чаті обговерення книжки, щоб запобігати не потрібній інформації, рекламі та інших шкідливих повідомлень. Також передбачити функціонал для скривання неприємного вмісту, нецензурної лексики та інших.

## 1.6 Висновки

Алгоритми машинного навчання можуть підвищити ефективність аналізу тексту, це водночас складний, ретельний та дуже корисний процес, котрий приносить бізнесу багато переваг. Результати роботи машинного навчання можуть бути дуже корисними, якщо правильно його застосовувати.

Так мільйони компаній можуть аналізувати відгуки своїх користувачів, легко, автоматизувавши цей процес. Вони не витрачають додаткові кошти, на людей які раніше займались цим процесом. А використовують методологію машинного навчання, яка робить це за них.

Вище описані алгоритми можуть аналізувати та передбачати дані, на основі тренування на моделях. Правильно використовуючи їх ви отримуєте неймовірні переваги у швидкості, зменшенні витрат часу та економії коштів, ви можете відстежувати тенденції та проблеми ринку, створювати візуалізовані звіти для менеджерів. Допомагати їм зв’язок з кінцевим користувачем.

# РОЗДІЛ 2. МОДЕЛЮВАННЯ ОСНОВНИХ АЛГОРИТМІВ АНАЛІЗУ ТЕКСТОВИХ ДАНИХ

## 2.1 NLP. Методи та їх моделі.

В розділі 1 було згадано більшість з цих моделей та дано короткі визначені, в даному розділі ми розглянемо їх більш детально, наведемо приклади їх використання та складемо блок схеми, та інші корисні діаграми для їх розуміння та реалізації.

### 2.1.1 Tokenization.

Це процес сегментації тексту на речення та слова. По суті, це завдання розрізати текст на частини, які називаються лексемами, і водночас викинути певні символи, наприклад, розділові знаки. Наслідуючи наш приклад, результат токенізації буде таким, який зображено на рисунку (2.1).

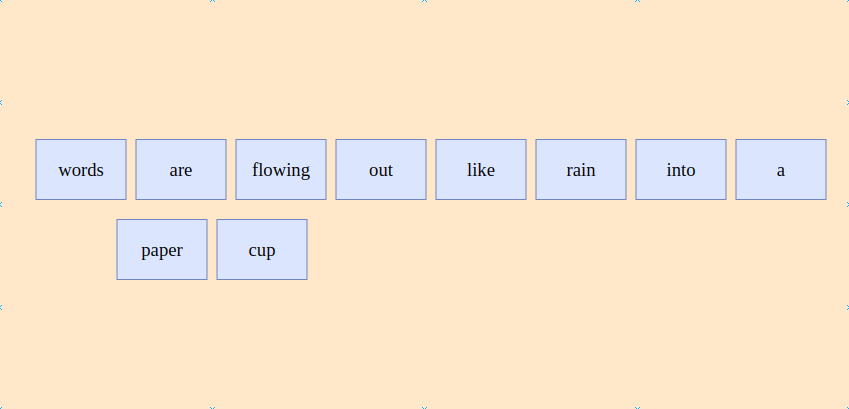


Рисунок 2.1 – Представлення тексту за допомогою токенізації

Як ми можемо бачити текст ніби розрізаний на частини, які в подальшому ми можемо використовувати та проводити однорідні операції над кожним з ним.

Хоча в цьому випадку це може здатися досить простим, а також у таких мовах, як англійська, що розділені слова порожнім простором (так звані сегментовані мови) не всі мови поводяться однаково, і якщо ми подумаєте, одних порожніх пробілів недостатньо навіть для того, щоб англійська могла правильно виконувати токенізацію. Розподіл на порожні пробіли може розбити те, що слід вважати однією лексемою, наприклад, у випадку певних назв (наприклад, Сан-Франциско чи Нью-Йорк) або запозичених іноземних фраз (наприклад, laissez faire) [7].

### 2.1.2 Stemming and Lemmatization

Зростання обсягу даних та інформації в Інтернеті за останні пару років стало найвищим за весь час. Ці величезні дані та інформація вимагають необхідних інструментів та методів для того, щоб з легкістю отримувати висновки.

«Stemming» - це процес скорочення складових (або іноді похідних) слів до їхньої основи слова чи кореня - зазвичай це письмова форма слова. Наприклад, те, що робить корінь, в основному відрізає всі суфікси. Тож після того, як ви застосуєте «stemming» на основі слова «playing», воно стає  «play», або подібне, «asked» стає «ask».

Під лемматизацією зазвичай розуміють дії з належним використанням словникового запасу та морфологічним аналізом слів, які зазвичай мають на меті виключити флексійні закінчення та повернути основну або словникову форму слова, яка відома як лема. Простими словами, лематизація стосується леми слова, яка передбачає скорочення форми слова після розуміння частини мови (POS) або контексту слова в будь - якому документі.

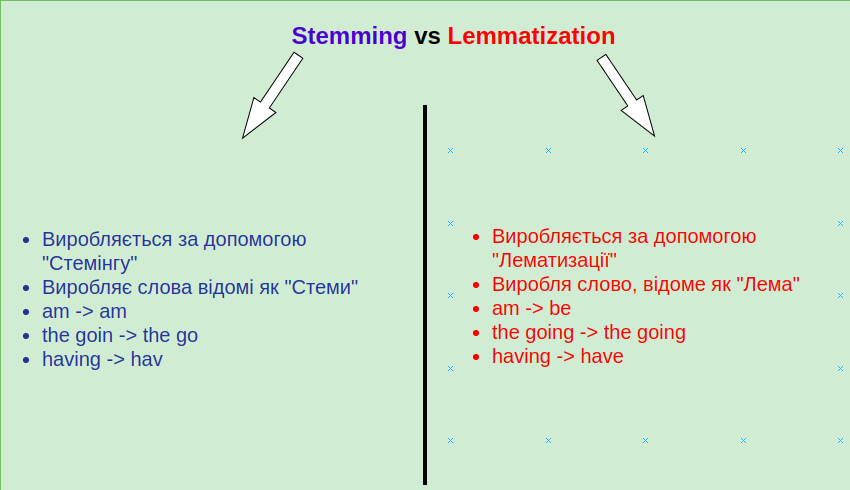


Рисунок 2.2 – Різниця між Lemmatization та Stemming

На рисунку 2.2 наведена різниця між Стемізацією та Лематизацією.

Розрізняють багато видів Стемінгу. Розглянемо один з найбільш популярних.

#### 2.1.2.1 Алгоритм Портера Стеммера

Це один із найпопулярніших методів стемінгу, запропонований у 1980 р, винайдений Мартіном Портером. Він базується на ідеї, що суфікси англійської мови складаються з комбінації менших та простіших суфіксів. Цей «stemmer» відомий своєю швидкістю та простотою.

Основні програми Porter Stemmer включають видобуток даних та пошук інформації. Однак його застосування обмежується лише англійськими словами. Крім того, група коренів відображається на одному і тому ж корені, і вихідний корінь не обов’язково має значення. Алгоритми мають досить довгий характер і, як відомо, є найдавнішим стовбуром.

Приклад: EED -> EE означає «якщо слово має принаймні один голосний та приголосний плюс закінчення EED, змініть закінчення на EE», оскільки agreed стає agree.

Як і кожен алгоритм, зокрема методи стемінгу та його різновид «Портер Стемінг» має свої плюси та мінуси.

Перевага: він дає найкращий результат порівняно з іншими штамбовими машинами і має меншу частоту помилок.

Обмеження: Вироблені морфологічні варіанти - це не завжди справжні слова.

Стемінг є не менш важливою складовою у підготовці даних до подальшої їх обробки машинним навчанням, адже нам потрібно звести всі слова в тексті до “однорідної маси”, щоб в подальшому правильно їх обробити та вирахувати з них потрібні нам дані.

Класичний алгоритм Портера Стемінга включає в себе п’ять кроків реалзіації, що відображені та кожен крок деталізований на блок схемі на рисунку 2.3.

Алгоритм також має широке застосування в в аналізі тексту, наприклад тональності, коли потрібно отримати «чисте» слово, нам не потрібно розглядати всі слова, та всі його види, у зв’язку з тим що ми оцінюватимемо слово лише за його тональністю, тому достатньо використати алгоритм для отримання одної форми та проаналізувати її, за допомогою певних ознак слів. Складається в суму та отримується певна категорія чи ознака тексту. Яка в подальшому використовується для складання певних графіків, за якими аналітики можуть передбачати згадані проблеми, покращувати продукт та його аналіз, що забезпечить якісний розвиток бізнесу в майбутньому [8].

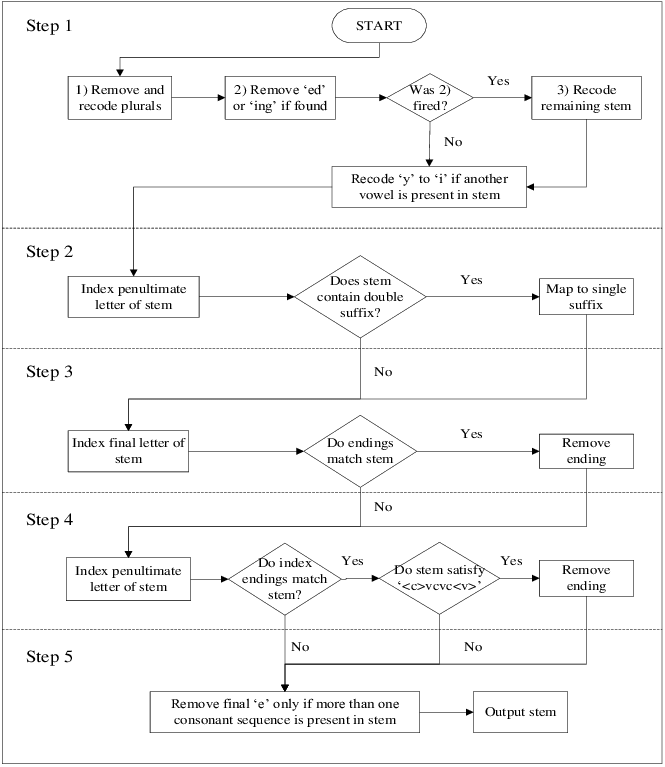


Рисунок 2.3 – Імплементація стемінгу

### 2.1.3 Bag of words

Техніка «bag of words» використовується для попередньої обробки тексту та вилучення всіх функцій із текстового документа для використання в моделюванні машинного навчання. Це також представлення будь - якого тексту, який детально описує чи пояснює входження слів у корпус (документ). Його також називають «bag» через його механізм, тобто він стосується лише того, чи є в документі відомі слова, а не розташування слів.

Давайте візьмемо приклад, щоб детальніше зрозуміти «Bag of words». Візьмемо два текстових документи.

Перший. «John is wrathful on Janin and he is wrathful on Lucy.»

Другий. «Lucy like dogs.»

Вище можемо побачити два тексти, як документи, ми розглядаємо обидва документи як різну сутність і складаємо список усіх слів, присутніх в обох документах, крім розділових знаків.

В результаті отримуємо наступний список.

«John», «is», «wrathful», «on», «Megy», «and», «he», «Lucy», «like», «dogs».

Потім ми створюємо ці документи у вектори (це також можна назвати, створенням тексту у числах, що називається векторизацією в ML) для подальшого моделювання.

Презентація «John is wrathful on Janin and he is wrathful on Lucy.» у векторній формі як [1,1,1,1,1,1,1,0,0], і така ж, як у «Lucy like dogs.» має векторну форму як [1,0,0,0,0,0,0,0,1,1]. Отже, техніка «Bag of words» в основному використовується для створення покоління з текстових даних [9].

### 2.1.4. Stop words removal.

Також одна з основних технік природної обробки мови. Використовується для видалення чи фільтрації непотрібних для контексту слів. У обробці природною мовою, марні слова (дані), називаються стоп - словами. Від чого і назва методу «stop words removal».

Стоп - слово - це загальновживане слово (наприклад, «the», «a», «an», «in»), яке пошукова система запрограмована ігнорувати як під час індексування записів для пошуку, так і при їх отриманні в результаті пошукового запиту.

Ми б не хотіли, щоб ці слова займали місце в нашій базі даних або забирали дорогоцінний час на обробку. Для цього ми можемо легко видалити їх, зберігаючи список слів, які, на нашу думку, є «марними» словами.

В таблиці 2.1 наведено приклад виконання алгоритму «Stop words removal».

Таблиця 2.1

Приклад виконання алгоритму «Stop words»

|  |  |
| --- | --- |
| «The film was not so good.» | “movie, good” |
| «Can listening be exhausting» | “Listening, exhausing” |
| «I like reading, so i read» | “Like, reading, read” |

Розглянемо подані речення детальніше. Перше речення виражає негативний зміст, те що фільм є не дуже добрий. Проте після видалення стоп - слів, ми отримали зовсім інше значення, те що фільм навпаки є хорошим. Тому при класифікації ми можемо отримати невірний результат. Враховуючи це нам потрібно вибирати список стоп слів, відповідно до завдання яке перед нами стоїть. Імплементація даного методу є доволі простою, та легко реалізується в кожній сучасній мові програмування [10].

В загальному вона складається з трьох етапів.

- Отримати слова з тексту за допомогою токенізації.

- Перевірити наявність слова в списку стоп слів.

- Вилучити їх з тексту, та додати до результуючого тексту.

### 2.1.5 Висновки

Розглянуті методи NLP допомагають у підготовці тексту до її подальшої оброки методами машинного навчання. Вони також доволі корисні і в одиночній роботі, для обробки неструктурованих даних. Так наприклад алгоритм POS (part of speech) допомагає в різного роду завданнях, починаючи від підготовки тексту для його аналізу на сентиментність чи є основою для його граматичного аналізу. Методи Lemmatization, Tokenization та інші допомагають в оцінці слова, як окремої одиниці тексту.

## 2.2 Наївний класифікатор Байєса.

Як вже було згадано в першому розділі, це метод класифікації, заснований на теоремі Байєса з припущенням незалежності серед предикторів.

Як ми вже сказали в основі NBC лежить теорема Байєса.

де P (c | d) - ймовірність події що клас d відноситься до класу c, що нам потрібно розрахувати;

P (d | c) - ймовірність появи документа d серед всіх документів класу c;

P (c) - безумовна ймовірність зустріти документ класу c в області документів;

P (d) - безумовна ймовірність документа d в області документів.

Розглянемо наступну задачу. Нам потрібно профільтрувати повідомлення, та визначити чи містить воно спам чи ні.

Припустимо ми маємо сет повідомлень, які ми будемо вважати «нормальними». Використовуючи їх ми можемо побудувати гістограму (рис. 2.4) ймовірності появи деякого слова в повідомленні.

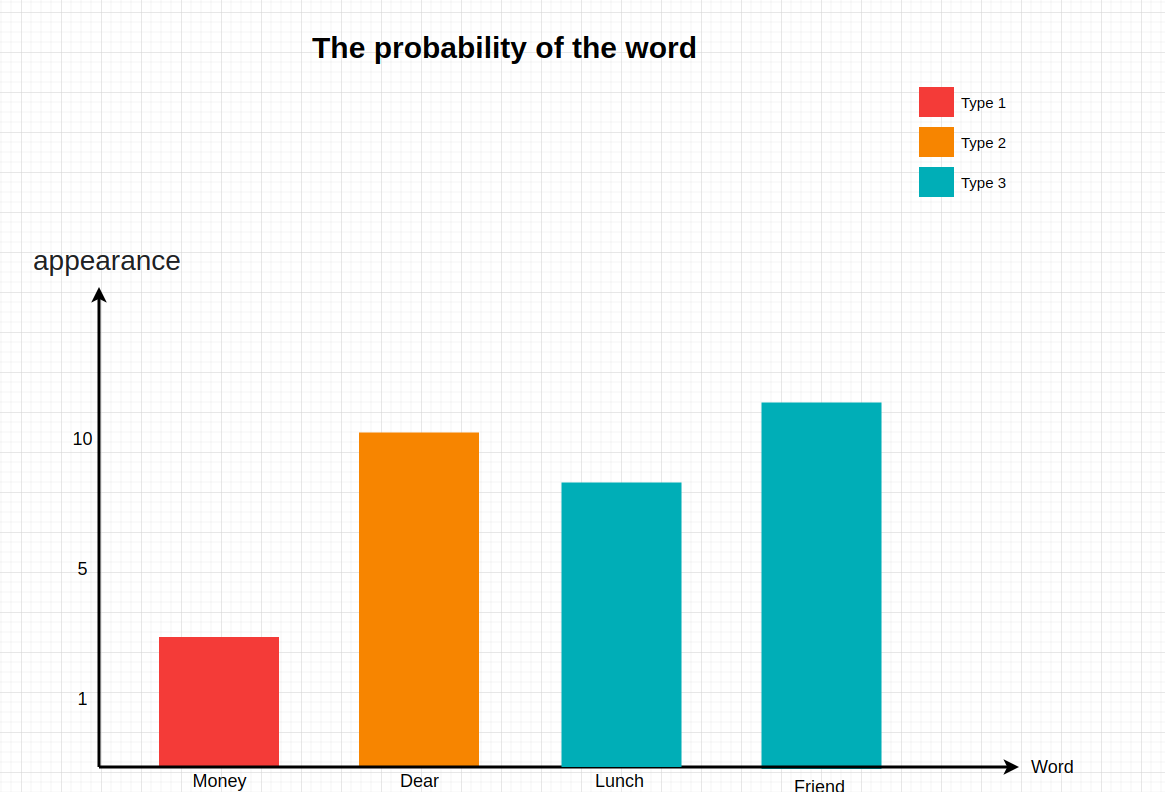


Рисунок 2.4 – Ймовірність появи слова в реченні

На основі поданої гістограми, виведемо вище згадані формули. Припустимо у нас слово money - з’явилось в повідомленнях 3 рази, dear - 12, lunch - 8, friend - 14. Отже, ймовірність появи слова dear обрахується за наступною формулою

де W – ймовірність появи слова,

d – кількість появи слова в тексті

N – кількість всіх слів в тексті.

Підрахувавши формулу ми отримаємо 12 / 37 = 0.32. За таким самим принципом, підрахуємо наступні слова та отримаємо результати, які наведені в таблиці 2.2.

Таблиця 2.2

Ймовірність появи слів в тексті

|  |  |
| --- | --- |
| Слово | Ймовірність |
| money | 0.08 |
| dear | 0.32 |
| lunch | 0.21 |
| friend | 0.37 |

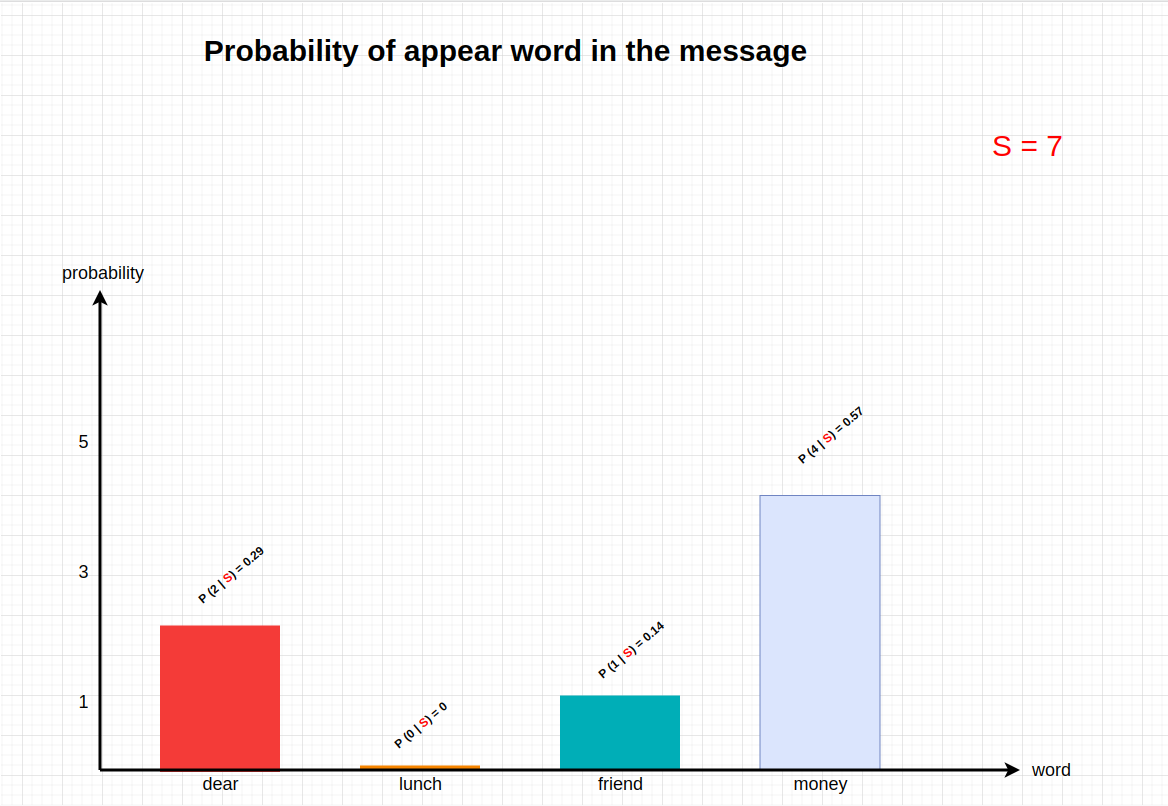


Рисунок 2.5 – Гістограма ймовірностей

На рисунку 2.5 наведено ймовірність появи кожного з розглянутих нами слів у повідомленнях, що несуть спам характер.

Тепер припустимо ми отримали нове повідомлення, що складається з двох слів «Dear friend», нам потрібно визначити чи дане повідомлення потрапить в спам чи в кошик з нормальними меседжами.

Ми розпочинаємо аналіз з того, що ідентифікуємо кожне повідомлення як «нормальне», незважаючи на його вміст. Припущенням може бути будь яка ймовірність, яку ми хочемо, проте загальна здогадка має бути оцінена на основі навчальних даних (training set).

Припустимо в навчальних даних було 14 повідомлень, 8 з яких були «нормальні», решта 4 – «спам». Виходячи з цього за формулою (2.2) обрахуємо оцінку «нормальних» повідомлень. Вона буде рівна 0.67.

Після цього розглянемо кожне слово в поданому меседжі, та порахуємо ймовірність класифікування повідомлення.

Підставивши формулу (2.3) числа з таблиці 2.2 та обчислимо її отримаємо наступний результат.

Ймовірність того що повідомлення, котре ми розглядаємо буде «нормальним» рівне 0.09. За таким самим принципом обрахуємо ймовірність того що подане повідомлення є «спамом». Результат такого припущення складатиме - 0.01.

В загальному ми можемо зробити висновок, що повідомлення «Dear Friend» є «нормальним» повідомленням, так як ймовірність такої події 0.09, а ймовірність, того що меседж є «спамом» - 0.01, виходячи з порівняння повідомлення є «нормальним», що має сенс.

Ускладнемо задачу. Розглянемо наступне повідомлення – «Lunch money, money, money, money». Так як на основі припущень, котрі ми зробили на початку ймовірність появи money в тренованих даних «спам» повідомлень є набагато більшою ніж в «нормальних», логічно було б припустити, що повідомлення котре ми розглядаємо є спамом.

Проте зробимо підрахунки на основі вище наведених формул та обрахуємо ймовірність кожної з подій.

Провівши не складні математичні операції, ми отримуємо результат, що дане повідомлення є «нормальним» в 0.000002, що є надзвичайно малим. Проте, так як в повідомленні присутнє слово «Lunch», а ймовірність появи такого в «спам» повідомленнях рівне 0, то з обчислення формули (2.3) для спам повідомлення ми отримуємо 0. Так як множення на 0 дасть нам в результати той самий нуль.

Виходячи з цього, вводять так званий альфа коефіцієнт, та припускають що кожне слово повинно зустрітись в повідомленнях щонайменше один раз, так ймовірність появи слова «lunch» в спам повідомленнях буде не нуль, а 0.09, що в результаті дасть нам більший результат для припущення, що повідомлення є спамом, в підсумку давши нам правильну класифікацію поданого прикладу.

Чому «Наївний Байєс» називають таким «Наївним». Це те, що він припускає, що всі слова мають однаковий порядок. Наприклад для повідомлення «Dear friend» результат ймовірності «нормального» класифікування буде 0.08, так само і для “Friend dear”.

Однакове ставлення до всіх порядків слів дуже відрізняється від того, як ми спілкуємось. Кожна мова має граматичні правила, загальні фрази, але «Naive Bayes» ігнорує їх всіх. Натомість Naive Bayes ставиться до мови так, як до простого мішка, повного слів, і кожне повідомлення - це випадкова жменька їх. Naive Bayes ігнорує всі правила, тому що стежити за кожною фразою у мові було б неможливо. Тим не менш, хоча Naive Bayes наївний, він має тенденцію працювати напрочуд добре, коли відокремлює звичайне повідомлення від спаму.

Розібравшись в задачах, сформуємо деяку послідовність дій, котрі потрібно виконати щоб реалізувати вище згаданий алгоритм.

Крок 1. Потрібно визначити перед умови для класифікатора Naive Bayes. Застосування класифікатора Байєса для класифікації тексту нескінченні. Єдина умова - мати наявний набір прикладів для кожної категорії (класу), у який ми хочемо об’єднати/класифікувати фрагменти тексту.

Наприклад, класифікації списків вакансій нам потрібен набір оголошень про вакансії, для яких відомо, що вони є цільовими (тобто позначеними) для «розробників програмного забезпечення», набір списків вакансій, відомих для пошуку  «дизайнерів», та набір вакансій, орієнтованих на «маркетологів».

У цьому випадку існує три класи («розробка програмного забезпечення», «дизайн» та «маркетинг»). Зі зразками списків вакансій у кожній категорії ми можемо навчити класифікатора Naive Bayes, щоб нові списки вакансій автоматично класифікувалися.

Іншим прикладом є розглянуті нами задачі, - це класифікація Spam/Ham, тому існує два класи: «Spam» та «Ham» (тобто не спам). Як згадувалося раніше, єдиною необхідною умовою є наявність наявного навчального набору електронних листів, відомих як спам, та навчального набору електронних листів, відомих як Ham.

Крок 2. Обчислення матриці термінових документів (TDM) для кожного класу.

Матриця термінового документа (TDM) складається зі списку частот слів, що з'являються в наборі документів.

Матриця TDM - це розріджена прямокутна матриця з n слів і m документів. І кажуть, що він рідкий, оскільки містить переважно нулі. Запис (i, j) матриці TDM представляє частоту слова «i» у документі «j».

Для англійського класифікатора спаму, який враховує всі слова англійською мовою, кількість слів (n) становить приблизно 170 тисяч. Що стосується кількості прикладів у класі, загальне правило таке: чим більше, тим веселіше. Однак на практиці кілька тисяч повідомлень дають обґрунтовані прогнози. Для завдання виявлення спаму фактичним джерелом даних із маркуванням спаму є SpamAssassin.

На рисунку 2.6, наведені приклади формування таких матриць у формі csv файлу.

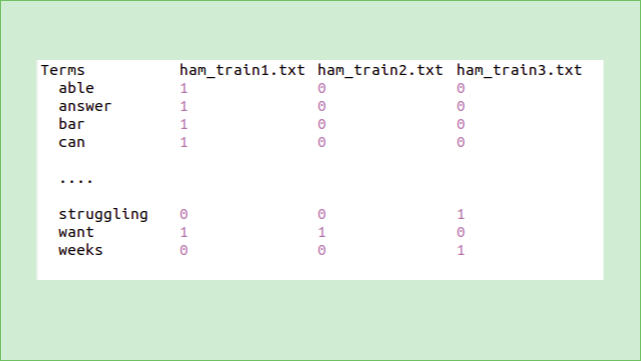


Рисунок 2.6 – CSV файл навчальних ham даних

Таку саму матрицю складаємо для сету текстових документів котрі містять спам.

Крок 3. Обчислення частот. Після того, як матриці TDM обчислюються для кожного класу, наступним кроком є обчислення частоти та появи кожного терміну: (рис 2.7)

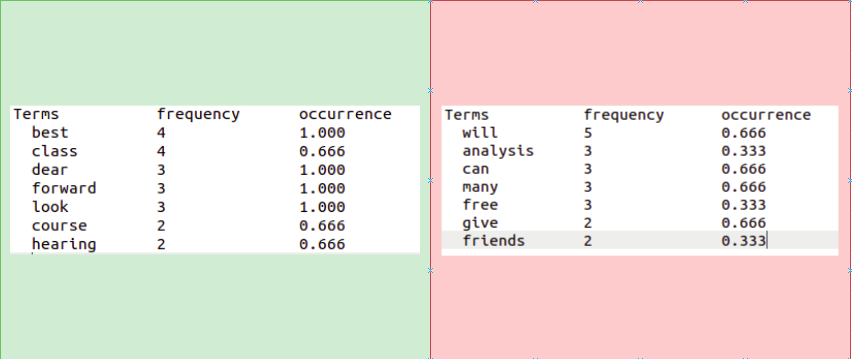


Рисунок 2.7 – Результат обчислення частот появи термінів

З підрахунку необробленої частоти можна зробити висновок, що спам - повідомлення містяться у високочастотних термінах, таких як «free», тоді як повідомлення Ham містяться у високочастотних термінах, таких як «exam». Таким чином класифікатор зможе відрізнити один клас від іншого.

Стовпець «frequency» - це кількість разів, коли кожен термін з'являвся у всіх документах, тобто сума стовпців матриці TDM. Стовпець «occurrence» - це відсоток часу, протягом якого кожен термін з'являвся у всіх документах.

І стовпці «frequency» і «occurrence» можуть бути використані для обчислення оцінок ймовірності, однак, стовпець «occurrence» є кращим, оскільки він обмежений між нулем і одиницею.

Крок 4. Пригадайте правило наївного Байєса. З елементарної ймовірності, яка ймовірність того, що станеться подія А, враховуючи, що також сталася інша подія В? Якщо казати словами, то яка ймовірність того, що A дано B?

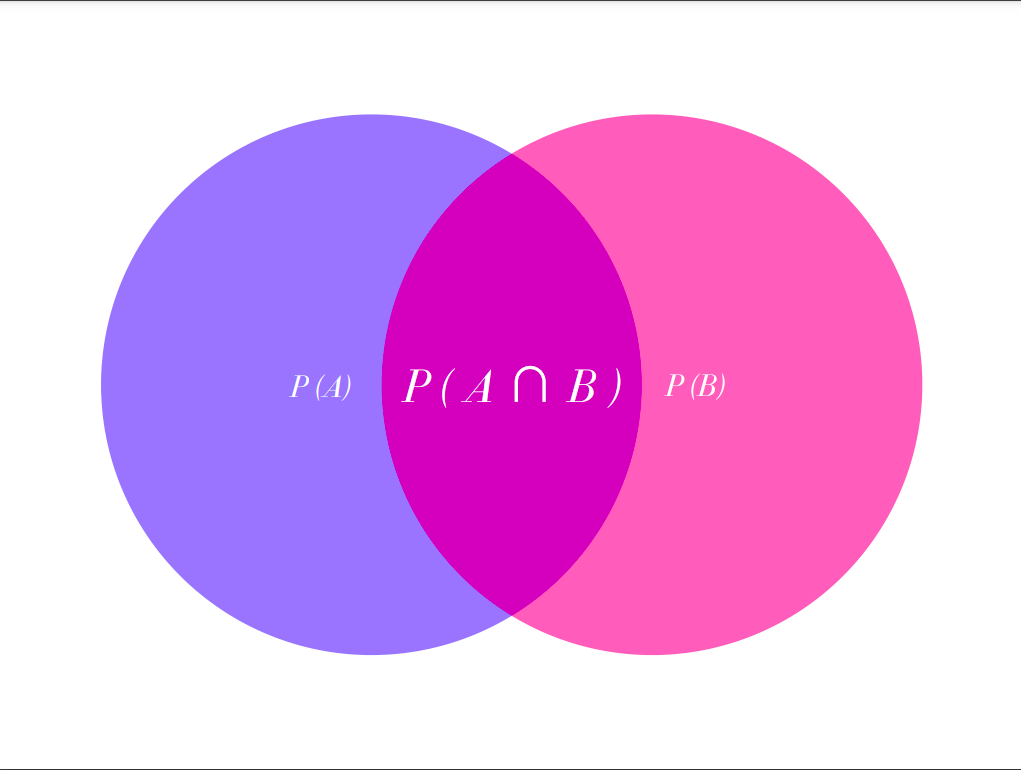


Рисунок 2.8 – Діаграма Венна

З діаграми Венна, що наведена на рисунку 2.8. ми можемо побачити, що ймовірність A з урахуванням B - це ймовірність того, що A і B (перетин) відбулися, поділена на ймовірність того, що B сталося, ця діаграма пояснює формулу, яку ми згадували раніше для розв’язання задач.

А як щодо ймовірності того, що B сталося, враховуючи, що A також сталося? Дотримуючись попередньої формули, маємо, що:

З діаграми Венна наведеної на рисунку 2.8, також випливає наступне:

З формули 2.6 виходить що вона рівна формулі (2.1) на початку цього розділу. Враховуючи значення теореми Байєса в теорії ймовірності, кожен термін має назву, тому перепишемо формулу 2.6 в наступну.

де «Prior» - P (A) та «evidence» - P (B) - відносяться до ймовірностей спостереження А та B незалежно один від одного.

«posterior» та  «likelihood» є умовними ймовірностями спостереження за даною B і навпаки.

Повертаючись до нашого прикладу класифікації електронної пошти в Spam або Ham, ймовірність того, що ми хочемо обчислити:

де x - вектор ознак, що містить слова, що надходять із листів зі спамом (або Ham):

«Наївне» припущення, яке робить класифікатор Naive Bayes, полягає в тому, що ймовірність спостереження за словом не залежить одна від одної. Результатом є те, що «ймовірність» є продуктом індивідуальної ймовірності побачити кожне слово в наборі листів зі спамом або хамом. Ми розрахували ці ймовірності на кроці 3 і зберегли їх у стовпці «поява». Формально:

Крок 5. Обчислення ймовірності того, що вхідний лист є спамом або «нормальним».

Враховуючи, що ми обчислили базу ймовірностей для термінів, що з’являються в електронних листах спаму або ham, ми можемо переходити до останнього кроку класифікатора Naive Bayes, який є класифікацією. Офіційне правило прийняття рішення таке:

Це означає, що для кожної вхідної електронної пошти нам потрібно обчислити ймовірність того, що така електронна пошта буде спамом і ham (тобто для кожного класу), і наша остаточна достовірність буде зроблена за найбільшою ймовірністю [11].

## 2.3 SVM. Support Vector Machine

### 2.3.1 Основні поняття про Support Vector Machine.

Метод опорних векторів є популярним методом класичної класифікації. Першочергове завдання SVM, це знайти роздільну лінію (або гіперплощину) між даними двох класів. Отже - це алгоритм, який приймає вхідні дані і виводить рядок, який розділяє ці класи, якщо це можливо. Вхідні точки, тобто два класи позначаються за допомогою векторів. Задача алгоритму знайти ту саму роздільну лінію між списком з векторів.

Цей метод можна застосувати до будь-якого типу векторів, які кодують будь-які дані. Це означає, щоб svm використати для класифікації тексту, нам потрібно перетворити текстові дані у вектори.

Для початку, дамо визначення вектору. Це величезні списки чисел, які представляють набір координат у деякому просторі.

Отже, коли машина підтримки векторів визначає межу прийняття рішень, про яку згадувалось вище, вона вирішує де провести «найкращу лінію», або як її ще називають гіперплощину, що розділяє простір на два підпростори, один для векторів, що належать до даної категорії, а інший для векторів що йому не належать.

Отже за умови, що ми можемо знайти векторні представлення для текстових даних, ми можемо застосувати алгоритм SVM до проблем класифікації тексту та отримати дуже хороші результати.

Щоб перетворити текстові дані в вектор нам потрібно використати методи NLP, що були описані в першому підрозділі поточного розділу. Ми можемо використати метод «bag of words», щоб отримати числову величину появи слова в тексті. Таким чином ми отримаємо потрібний нам вектор, для його подальшого використання в навчальній моделі алгоритму SVM.

SVM Kernel. Алгоритм SVM реалізований на практиці за допомогою «kernel». «Kernel» перетворює простір вхідних даних у необхідну форму. SVM використовує техніку, яка називається «Kernel trick». Тут «kernel» займає низькомірний вхідний простір і перетворює його на простір вищих розмірів. Іншими словами, вона перетворює нероздільну проблему в роздільні, додаючи до неї більше вимірів. Найбільш корисний у задачі нелінійного поділу. «Kernel trick» допоможе вам створити більш точний класифікатор.

Розділяють три види SVM «ядра»:

- Linear kernel

- Polynomial kernel

- Radial basis function kernel

«Linear kernel» може бути використаний, як звичайний добуток будь-яких двох даних спостережень. Добутком між двома векторами є сума множення кожної пари вхідних значень.

«Linear kernel» використовується, коли дані є лінійно розділимими, тобто їх можна розділити за допомогою однієї лінії. Це однин з найбільш поширених ядер. Найчастіше він використовується, коли у певному наборі даних є велика кількість значень. Навчання SVM з «Linear kernels» відбувається швидше, ніж з будь -яким іншим [12].

#### 2.3.1.1 SVM та лінійні проблеми

Класифікацію можна розглядати як завдання з розділення класів у просторі ознак.

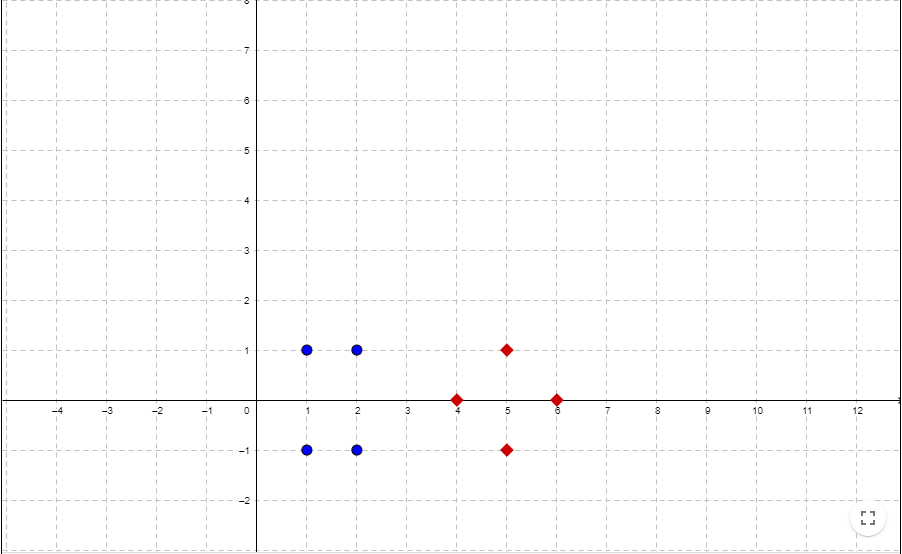


Рисунок 2.9 – Класифікація двох класів

Ось два класи червоний клас і синій, і завдання полягає в тому, щоб знайти межу між цими двома класами і зробити окрему гіперплощиною лінію.

У Support Vector Machine ми робимо оптимальну лінію для розділення цих двох класів.

Тут ми вибираємо три опорні вектори для початку.

Використаємо вектори доповнені одиницею, та позначимо їх за допомогою тих самих змінних з тильдою.

Тепер нам потрібно знайти 3 параметри α1, α2 та α3 на основі наступних 3 лінійних рівнянь:

Після підстановки та спрощення поданих рівнянь ми потримаємо наступну систему:

Розв’язавши систему 2.17 ми отримаємо наступне рішення:

α1 = -3.25, α2 = -3.25 та α3 = 3.5.

Гіперплощина, яка розрізняє позитивний клас від негативного, визначається за наступною формулою:

За формулою 2.18 знайдемо гіперплощину для нашого випадку який зображений на рисунку 2.9.

Підставивши у формулу 2.19 значення обраховані у системі 2.17, отримаємо наступний вектор

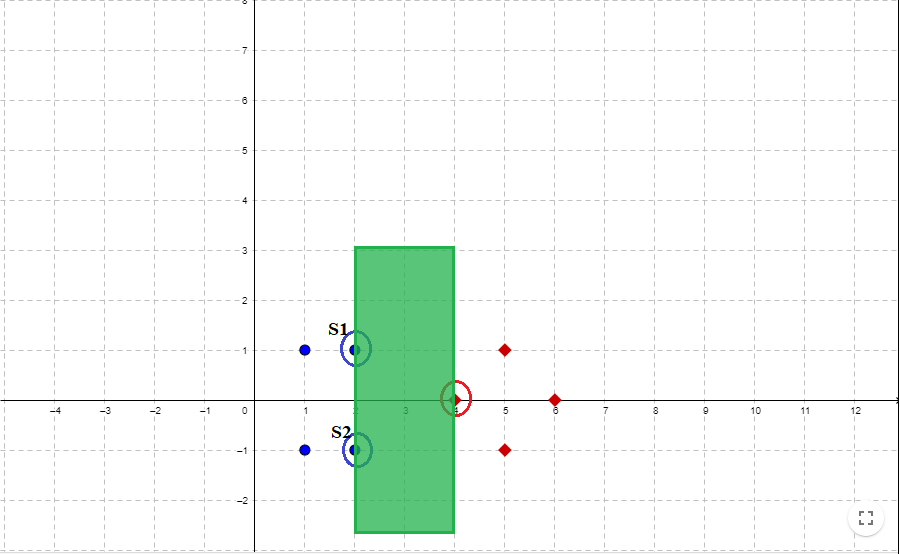


Рисунок 2.10 – Знайдена гіперплощина

Це очікувана поверхня прийняття рішень LSVM.

#### 2.3.1.2. SVM та нелінійні проблеми

Тепер розглянемо нелінійний приклад, розмежування даних, такий який зображений на рисунку 2.11. Очевидно, що провести лінію класифікації так просто ми не можемо, так як немає чітко вираженої гіперплощини розділення між червоним та синім класом. Отже тут нам потрібно знайти функцію нелінійного відображення, яка може перетворити цей простір в новий простір ознак, так щоб, ми мали змогу знайти розділювану гіперплощину, за методом описаним в попередньому розділі.

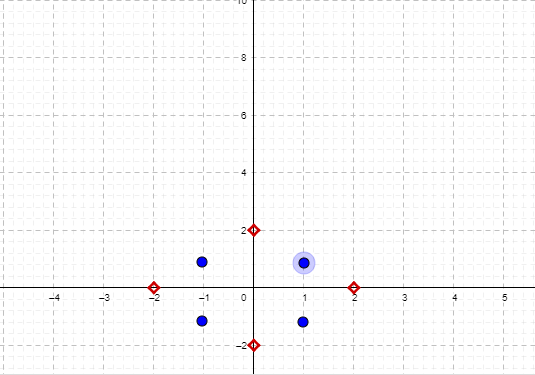


Рисунок 2.11 – Простір для нелінійного розділення.

Визначимо множину синіх точок:

Та множину червоних точок:

Розглянемо наступну функцію відображення:

Застосувавши дану функцію до множини синіх точок, нічого не зміниться, так як умова в першій функції системи не виконається.

Множина червоних точок трансформується в наступну:

Що в результаті нам дасть наступну множину, яка зображена на рисунку 2.12.

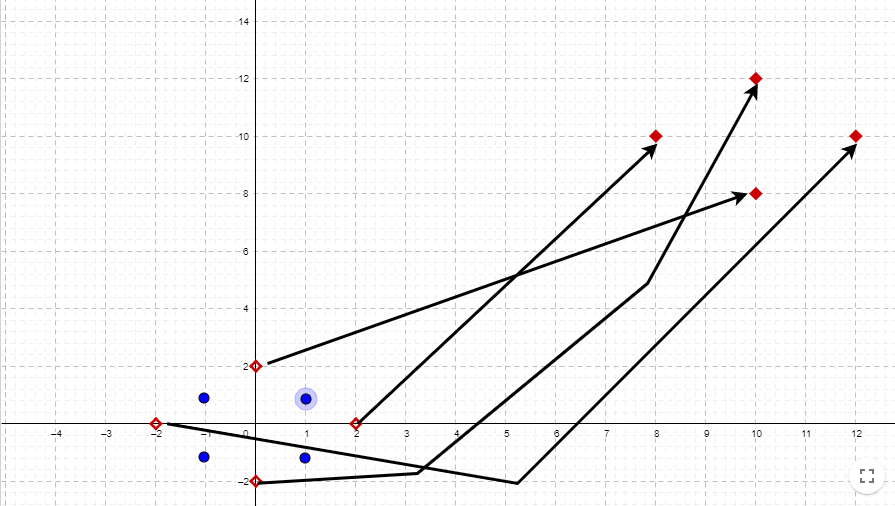


Рисунок 2.12 – Трансформована множина синіх точок

Дальше застосувавши формули 2.14 – 2.17 ми отримаємо наступні значення: α1 = 0.859, α2 = 0.859 та α3 = -1.4219.

Підставивши ці значення у формулу 2.19 отримаємо наступні значення:

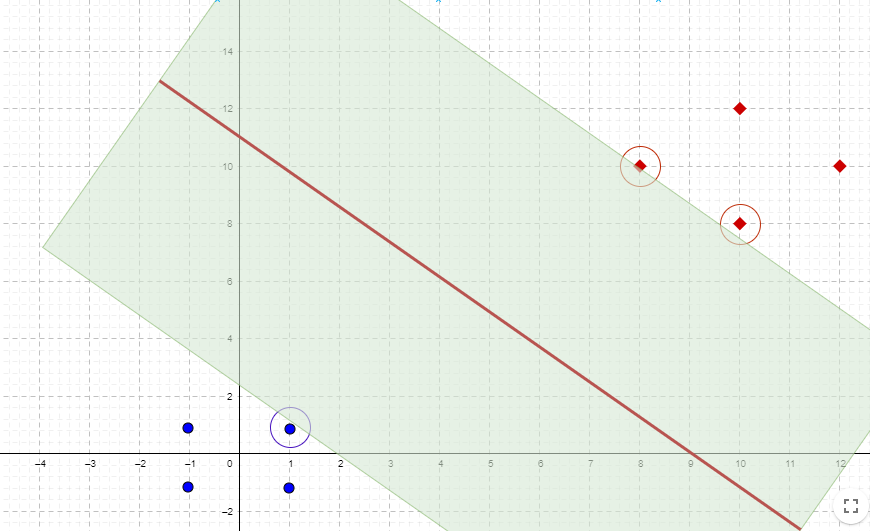


Рисунок 2.13 – Гіперплощина розмежування

На рисунку 2.13 зоображена гіперплощина розмежування для множини точок, з прикладу на рисунку 2.11.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУ НЕСТРУКТУРОВАНИХ ТЕКСТОВИХ ДАНИХ

## 3.1 Вибір технологій.

Для імплементації поставленого завдання було обрано наступний ряд технологій:

- Postgres SQL

- Node.js

-Amazon S3 Bucket

-Web Sockets

- Sequelize.js

- JWT Token

## 3.2 Обгрунтування вибраних технологій.

### 3.2.1 Web Sockets. Введення.

WebSockets вперше були описані в 2008 році і користувалися широкою підтримкою браузера приблизно з 2010 року. До того, як WebSockets з'явився, Інтернет в режимі реального часу існував, але його було важко досягти, як правило, повільніше, і він доставлявся шляхом злому існуючих веб-технологій, які не були розроблені. для додатків реального часу. Це тому, що Інтернет був побудований на основі протоколу HTTP, який спочатку був повністю розроблений як механізм запиту-відповіді.

Він працював за наступним принципом:

Відкрийте з’єднання, опишіть, що ви хочете, отримайте відповідь, а потім закрийте з’єднання.

На початку Інтернету це було хороше рішення, тому що тоді ми насправді мали справу лише з текстовим документом і, можливо, кількома додатковими ресурсами (зазвичай зображеннями).

Проблематика такої поведінки була в тому що, веб-додатки спочатку розроблялися навколо моделі клієнт/сервер, де веб-клієнт завжди є ініціатором транзакцій, запитуючи дані від сервера. Таким чином, не було механізму для сервера, щоб самостійно надсилати або передавати дані клієнту без попереднього запиту клієнта.

Щоб подолати цей недолік, розробники веб-додатків можуть реалізувати техніку, яка називається довгим опитуванням HTTP, коли клієнт опитує сервер, запитуючи нову інформацію. Сервер утримує запит відкритим, поки не будуть доступні нові дані. Після доступу сервер відповідає і надсилає нову інформацію. Коли клієнт отримує нову інформацію, він негайно надсилає інший запит, і операція повторюється. Або ж можна реалізувати протокол веб-сокетів.

Проаналізуємо різницю між двома підходами, та дослідимо ефективніший метод.

#### 3.2.1.1. Long polling (HTTP)

Довге опитування, по суті, є більш ефективною формою оригінальної техніки опитування. Виконання повторних запитів до сервера витрачає ресурси, оскільки кожне нове вхідне з’єднання має бути встановлене, заголовки HTTP повинні бути проаналізовані, має бути виконаний запит на нові дані, а також має бути згенерована відповідь (зазвичай без нових даних) і доставлено. Після цього, з’єднання потрібно закрити, а всі ресурси очистити. Замість того, щоб повторювати цей процес кілька разів для кожного клієнта, доки нові дані для даного клієнта не стануть доступними, тривале опитування — це техніка, при якій сервер вирішує утримувати з’єднання клієнта відкритим якомога довше, надаючи відповідь лише після того, як дані стануть доступними або досягнуто порогового значення тайм-ауту.

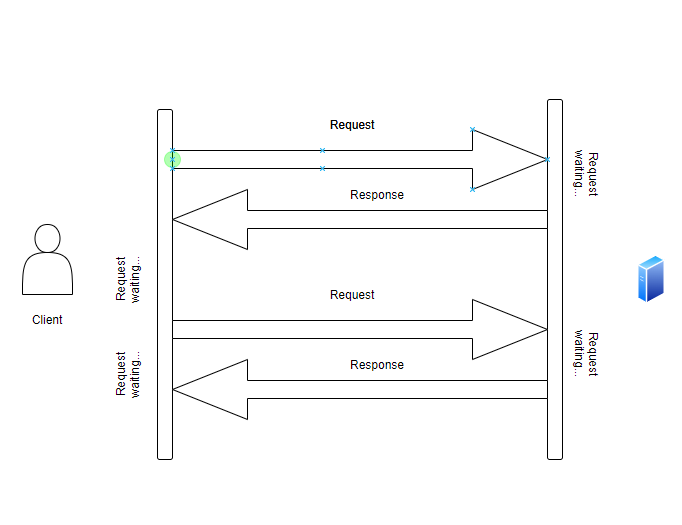


Рисунок 3.1 – Long Polling

Розглянемо позитивні та негативні моменти такого підходу.

Плюси:

- Long polling реалізовано на задній частині XMLHttpRequest, який майже повсюдно підтримується пристроями.

Мінуси:

- Довге опитування на сервері відбувається набагато інтенсивніше.

- Надійне впорядкування повідомлень може бути проблемою з тривалим опитуванням, оскільки є можливість одночасного виконання кількох запитів HTTP від ​​одного клієнта. Наприклад, якщо у клієнта відкриті дві вкладки браузера, які споживають той самий ресурс сервера, а програма на стороні клієнта зберігає дані в локальному сховищі, такому як localStorage, немає вбудованої гарантії, що повторювані дані не будуть написані не один раз.

- Залежно від реалізації сервера підтвердження отримання повідомлення одним екземпляром клієнта також може призвести до того, що інший екземпляр клієнта взагалі ніколи не отримає очікуване повідомлення, оскільки сервер може помилково вважати, що клієнт уже отримав дані, які він очікує.

#### 3.2.1.2 Web sockets. Детальний огляд.

Web Sockets — це тонкий транспортний шар, побудований поверх стеку TCP/IP пристрою. Мета полягає в тому, щоб надати розробникам веб-додатків те, що, по суті, є максимально наближеним до необробленого рівня зв’язку TCP для розробників веб-додатків, додаючи кілька абстракцій, щоб усунути певні тертя, які в іншому випадку існували б щодо того, як працює Інтернет. Вони також враховують той факт, що Інтернет має додаткові міркування безпеки, які необхідно враховувати для захисту як споживачів, так і постачальників послуг [13].

Розглянемо відмінності між Web Sockets та Long Polling, за допомогою порівнянь підходів до їх реалізації. Для цього зообразимо загальну схему роботи Web Sockets на рисунку 3.2.

Опишемо наведену схему.

Заголовок Connection повідомляє серверу, що клієнт хоче домовитися про зміну способу використання сокета. Супровідне значення Upgrade вказує, що транспортний протокол, який зараз використовується через TCP, має змінитися.

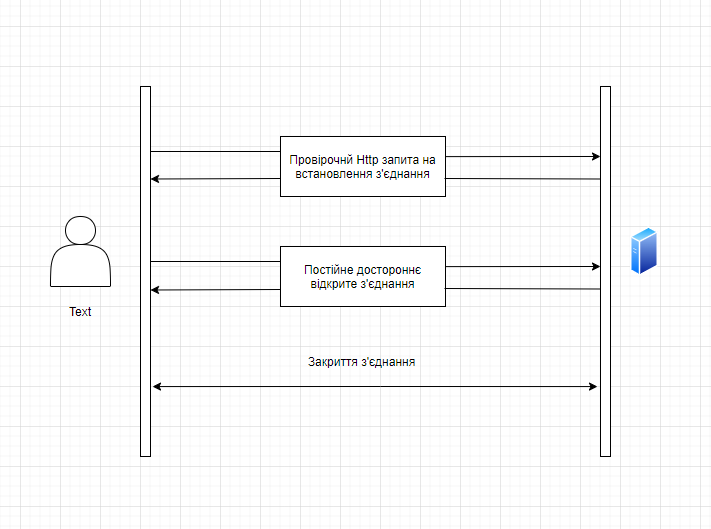


Рисунок 3.2 – Схема роботи Web Sockets

Тепер, коли сервер знає, що клієнт хоче оновити протокол, що використовується в даний момент, через активний сокет TCP, сервер знає, що шукає відповідний заголовок Upgrade, який вкаже йому, який транспортний протокол клієнт хоче використовувати протягом часу, що залишився з'єднання. Як тільки сервер бачить WebSocket як значення заголовка Upgrade, він знає, що розпочався процес рукостискання WebSocket.

Після того як з’єднання встановлено, як клієнт так і сервер можуть обмінюватись повідомленнями.

Розглянемо плюси та мінуси такого підходу:

Плюси:

- WebSockets зберігає унікальне з'єднання відкритим, усуваючи проблеми із затримкою, які виникають при тривалому опитуванні.

- WebSockets зазвичай не використовують XMLHttpRequest, і тому заголовки не надсилаються щоразу, коли нам потрібно отримати більше інформації від сервера. Це, у свою чергу, зменшує дороге навантаження даних, що надсилаються на сервер.

Мінуси:

- WebSockets не відновлюються автоматично, коли з’єднання припиняються – це те, що потрібно реалізувати самостійно, і є частиною причин, чому існує багато бібліотек на стороні клієнта.

- Браузери старше 2011 року не можуть підтримувати підключення WebSocket, але це стає все менш актуальним.

#### 3.2.1.3. Аналіз

Аналізуючи два підходи варто розглянути одну з найважливіших характеристик, а саме – масштабування.

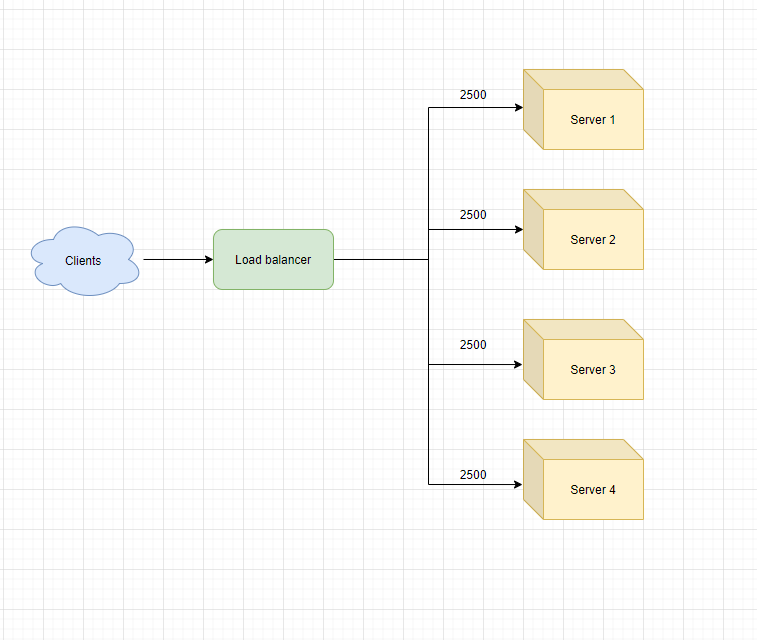


Рисунок 3.3 – Схема згрупованих серверів з 10000 з’єднань

Розглянемо приклад з масштабуванням клієнтів та запитів на сервер, у двох різних випадках з Long Polling та Web Sockets.

Уявімо, що в нас є 4 сервери, які згруповані між собою, та розподіляють нагрузку в 10000 з’єднань.

Тепер припустимо ми захотіли додати ще один сервер та очікуємо хвилину. Що станеться в такому випадку з існуючими з’єднаннями? У випадку з Web Sockets, всі з’єднання залишаться на перших чотирьох серверах тривалий час, а п’ятий сервер буде простоювати. Балансувальник навантаження є достатньо розумним, щоб маршрутизувати нові з’єднання на новий сервер додатків, щоб збалансувати кількість одночасних з’єднань, щоб цей ефект з часом зменшився. Однак кількість часу, необхідного цій системі для повернення до рівноваги, невідома і теоретично нескінченна.

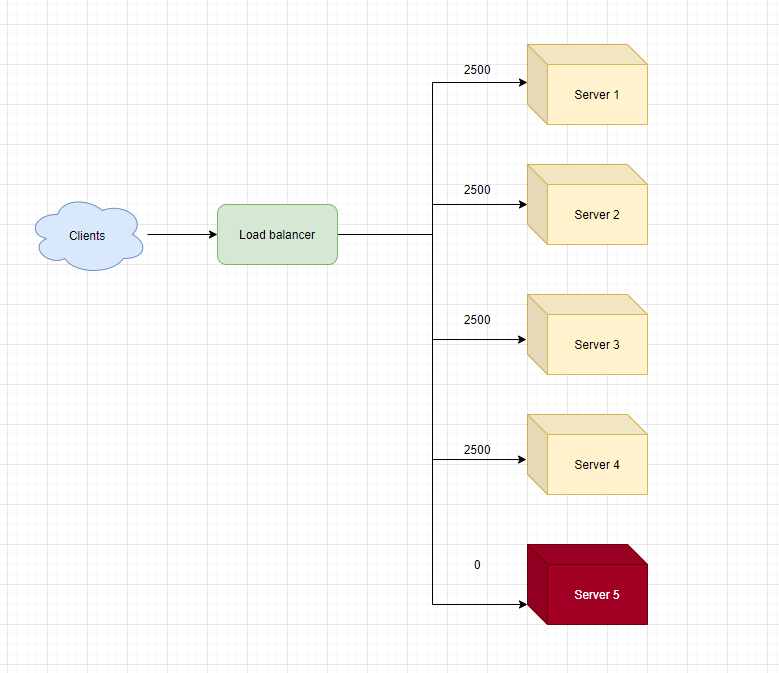


Рисунок 3.4 – Схема після додавання серверу

В свою чергу Long Polling через постійне опитування серверу, не буде мати такої проблеми, та з’єднання відновлять баланс між розподіленням на згруповані сервери після наступних запитів, та на кожен сервер припаде приблизно по 2000 з’єднань, тобто автоматичне масштабування відбудеться успішно.

#### 3.2.1.4. Висновки

Довге опитування є набагато більш ресурсомістким на серверах, тоді як WebSockets мають надзвичайно легкий слід на серверах. Довге опитування також вимагає багато стрибків між серверами та пристроями. І ці шлюзи часто мають різні уявлення про те, як довго типове з’єднання може залишатися відкритим. Якщо він залишається відкритим занадто довго, щось може його вбити, можливо, навіть коли він робив щось важливе.

WebSockets підходять для багатьох додатків, які вимагають постійного повнодуплексного високочастотного зв'язку з низькою затримкою, таких як програми чату. Однак будь-яку архітектуру WebSocket, яку можна звести до проблеми напівдуплексу, ймовірно, можна перемоделювати для використання тривалого опитування для покращення мінливості продуктивності програми, зниження операційної складності та сприяння повній стабільності системи.

Проте можна чітко стверджувати що вибір технології точно залежить від потреб вирішення певної проблеми.

В даному практичному завданні було обрано технологію веб сокетів для побудови чату, в якому користувачі могли б обговорювати вибрану книгу, та спілкуватись з однодумцями.

### 3.2.2. Node.js та Express.js

#### 3.2.2.1. Node, як засіб розробки серверів.

Для реалізації поставленої задачі було обрано мову програмування Node.js.

Node.js – це оточення JavaScript, яке реалізоване за моделлю асинхронно подій, він побудований для розробки масштабованих аплікацій. Для кожного нового з’єднання на Node.js викликається фукнція зворотного виклику. Цей підхід доволі сильно відрізняється від більш стандартного з більш загальною моделлю, де використвовуться паралельні операційні потоки. Такий підхід є відносно неефективним та дуже важким у використанні. На Node.js відсутні будь які блокування процесів, адже жодна з функцій не працює напряму з I/O, тому процеси не можуть блокуватись.

Однією з головних переваг Node є кросплатформне середовище виконання з відкритим вихідним кодом. Також в Node.js є ряд інших важливих переваг:

- Асинхронність. Асинхрований і керований подіями – усі API бібліотеки асинхронні, тобто не блокують основний потік. По суті, це озачає, що сервер на основі Node.js ніколи не чекає, поки API поверне дані. Сервер переходить до наступного API після його виклику, а механізм сповіщень про події допомагає серверу отримати відповідь від попереднього виклику API.

- Висока швидкість. Будучи побудованими на Google Chrome`s V8 JavaScript Engine, що використовується більшістю браузерів, бібліотеки Node.js є дуже швидкі у виконанні коду.

- Однопотоковий, але високомасштабований – Node.js використовує однопотокову модель з циклом подій. Механізм подій допомагає серверу реагувати неблокуючим способом і робить сервер масштабовним, на відміну від традиційних підходів, які створюють обмежені потоки для обробки запитів. Node.js використовує одну потокову програму, і та сама програма може надавати обслуговування набагато більшої кількості запитів, ніж традиційні сервери такі як Apache HTTP Server.

- Без буферизації – програми Node.js ніколи не буферизують дані. Ці програми просто виводять дані фрагментами.

- Ліцензія — Node.js випущено під ліцензією MIT. Група ліцензай розроблених Массачусетським технологічним інститутом для розповсюдження вільного програмного забезпечення.

Node.js використовує кілька основних модулів, що представленні на рисунку 3.5. у своїй розробці, що дозволяють суттєво скоротити час розробки програмного забезпечення.

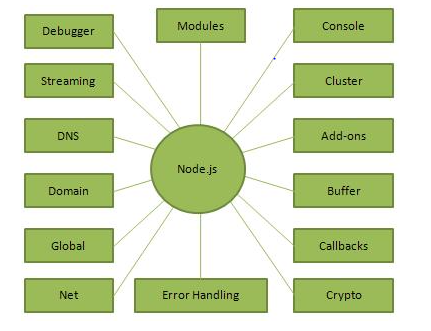


Рисунок 3.5 – Основні модулі Node.js

Варто також згадати і про недоліки Node.js. Серед таких можна виділити декілька головних:

- Проблема асинхронності.

Одна з головних переваг, також може стати і великим недоліком. В більшій степені Node полагається на так званні callback функції, які запускаються після завершення кожної задачі в черзі, тому збереження ряду задач в черзі також може сильно ускладнити програму, утруднюючи розуміння коду.

- Стабільність.

Хоча основні модулі Node.js є таки добре реалзованими, і достатньо документованими та вони можуть вважатись достатньо стабільними для їх використання, в NPM реєстрі є дуже багато інструментів, які або мають низьку якість, або ж не мають чітко визначеної документаці, вони не пройшли певні етапи тестування, та не можуть вважатись стабільними, хоча з першого погляду, вони можуть виконувати ваші потреби, проте в деяких спеціальних випадках, про які розробники можуть не замислюватись, пакети можуть міняти свою поведінку, та працювати не так як ви очікували. Виходячи з цього розробникам, потрібно докласти більше зусиль для пошуку підходящих пакетів, після чого протестувати їх на спеціальні випадки, та проаналізувати свої майбутні потреби.

- Низька продуктивність при роботі з важкими обчислювальними задачами.

Проблема виникає кожен раз, коли Node отримує важку обчислювальну задачу. Та через свою природу оброблення запитів через event loop, тобто Node обробляє всі запити асинхронно через Js код, що дає в певній мірі високу продуктивність на простих задачах, коли сервер отримає важку задачу, він напривить всі доступні ресурси ЦП для його першої обробки, а потім відповідсть на інші запити, що призведе до загальної затримки циклу виконання подій.

Роблячи висновок з розглянутих плюсів та мінусів, Node добре підходить для малих аплікацій, також для швидкої розробки додатків, адже Node легко конфігурована, та багато-функціональна для розробки додатків.

#### 3.2.2.2 Express.js

Фреймворк для веб-розробки на Node.js. Швидкий, гнучкий та простий у використанні. Express – це мінімальний і гнучки фреймворк веб-додатків node.js, що забезпечує надійний набір функцій для створення одно- та багатосторінкових додатків, а також гібридних веб-додатків.

Express класифікується як інструмент «Microframeworks (Backend)».

Розглянемо деякий функціонал, який пропонує Express.js:

- Надійна маршрутизація

- Http utils (переспрямування, кешування та інші)

- Перегляд системи, яка підтримує 14+ шаблонних механізмів

Він добре підходить для швидкої розробки веб-застосунків, та має низький поріг входження, добре налаштовуємий та має хорошу підтримку у вигляді документації.

### 3.2.3 Бази даних. PostgreSQL.

#### 3.2.3.1 Основні поняття про БД.

Бази даних чи не одна з найважливіших складових успішного проекту, правильно побудована архітектура бази даних, це ключ до успішного виконання замовлення, розвитку продукту, та його подальшої підтримки.

Існують багато видів баз даних, які підходять під певні потреби користувачів. Розглянемо та проаналізуємо деякі з них, та опишемо їх детально:

- Document databases.

Різновид нереляційної бази даних, призначеної для зберігання та запиту даних у вигляді документів, подібних до JSON. Бази даних документів полегшують зусилля розробника щодо зберігання та запиту даних у базі даних, використовуючи той самий формат моделі документа, що використовується в коді програми. Такі бази даних є гнучко масштабованими, навідмінно від реляційної моделі управління базою даних, яку важко масштабувати, моделі документної чи NoSql бази, використовують нові вузли, що робить їх прозорими для розширення. Вони потребують менше експертного керування, оскільки вже мають можливості автоматичного відлагодження. Такі бази є економічні, тоді як реляційні вимагають дорогих власних серверів і систем зберігання.

- Graph databases

Тип бази даних NoSQL, база даних Graph замінює таблиці на графи, що мають вершини та ребра. У той час як вершини представляють сутності, такі як особа або місце, ребра представляють відносини між двома вузлами.

- Ієрархічні бази даних

Дотримується моделі даних, в якій дані організовані в деревоподібну структуру. Дані зберігаються як записи, а різні записи з’єднуються за допомогою посилань. Кожен запис містить набір полів, тоді як кожне поле має лише одне значення.

- Бази даних NoSQL

Бази даних NoSQL – альтернатива традиційній реляційній базі даних, база даних NoSQL надзвичайно корисна під час роботи з гігантськими наборами розподілених даних. На відміну від баз даних SQL, які базуються на таблицях, бази даних NoSQL базуються на документах, парах ключ-значення, графічних базах даних або сховищах із широкими стовпцями. Крім цього, між базами даних SQL і NoSQL є кілька інших помітних відмінностей.

#### 3.2.3.2 Важливість СКБД.

Є багато переваг, пов’язаних із можливістю ефективно обробляти великі обсяги даних, і найважливішими серед них є такі:

- Підвищення ефективності

Системи керування базами даних є важливими для бізнесу, оскільки вони пропонують ефективний спосіб обробки великих обсягів і різних типів даних. Можливість ефективного доступу до даних дозволяє компаніям швидше приймати зважені рішення.

- Універсальність

Системи керування базами даних неймовірно універсальні. До них можна отримати доступ на комп’ютерах, планшетах і навіть мобільних пристроях.

- Дозволяє категоризувати та структурувати наявні дані

Системи керування базами даних дозволяють організаціям і окремим особам класифікувати та структурувати доступні дані. Наприклад, база даних для університету може зберігати інформацію про вступ студентів (наприклад, ім’я, очікуваний рік випуску, національність), а також може зберігати оцінки студентів та середній бал.

- Мультидоступ

Advantages-of-database-management-systemСистема управління базою даних дає можливість кільком авторизованим користувачам отримувати доступ до однієї бази даних з різних місць, різними способами, для виконання різних намірів.

Наприклад, HR-адміністратор може оцінити щорічне підвищення заробітної плати, отримавши доступ до поточної зарплати співробітників і збільшення приросту, визначених його керівником, щоб визначити цю суму. У той же час команда розробників може отримати доступ до тієї ж бази даних, щоб найняти тимчасового працівника на заміну захворілого члена команди.

#### 3.2.3.3. PostgreSQL

Для виконання поставленої задачі було обрано базу даних PostgreSql. PostgreSQL — це система управління базами даних корпоративного класу з відкритим вихідним кодом. Він підтримує як SQL, так і JSON для реляційних і нереляційних запитів для розширюваності та відповідності SQL. PostgreSQL підтримує розширені типи даних і функції оптимізації продуктивності, які доступні лише в дорогих комерційних базах даних, таких як Oracle і SQL Server.

Переваги Postgres:

- PostgreSQL може запускати динамічні веб-сайти та веб-програми як варіант стека LAMP.

- Попереднє ведення журналу PostgreSQL робить його високовідмовостійкою базою даних.

- Вихідний код PostgreSQL вільно доступний під ліцензією з відкритим вихідним кодом. Це дає свободу використовувати, змінювати та впроваджувати його відповідно до потреб бізнесу.

- Щоб вивчити Postgres, не потрібно багато навчання, оскільки його легко використовувати.

- Низький рівень обслуговування та адміністрування як для вбудованого, так і для корпоративного використання PostgreSQL.

Недоліки:

- Зміни, внесені для підвищення швидкості, вимагають більше роботи, ніж MySQL, оскільки PostgreSQL зосереджується на сумісності

- Багато програм з відкритим кодом підтримують MySQL, але можуть не підтримувати PostgreSQL

- За показниками продуктивності він повільніше, ніж MySQL.

Зі всього вище описаного однією з найважливіших переваг, чому було обрано PostgreSQL це безкоштовна ліцензія, та низький поріг входження.

#### 3.2.3.4. Sequelize

Sequelize — це інструмент Node.js ORM на основі обіцянок для Postgres, MySQL, MariaDB, SQLite та Microsoft SQL Server. Він має надійну підтримку транзакцій, відносини, швидке та ліниве завантаження, реплікацію читання та багато іншого функціоналу.

ORM - Об’єктно-реляційне відображення (ORM) — це техніка, яка дозволяє запитувати дані з бази даних і керувати ними за допомогою об’єктно-орієнтованої парадигми. Говорячи про ORM, більшість людей мають на увазі бібліотеку, яка реалізує техніку об’єктно-реляційного відображення, звідси фраза «ORM».

Бібліотека ORM — це цілком звичайна бібліотека, написана певною мовою, яка інкапсулює код, необхідний для маніпулювання даними, тому нам більше не потрібно використовувати SQL; а ми безпосередньо взаємодіємо з об’єктом тією ж мовою, якою користуємося.

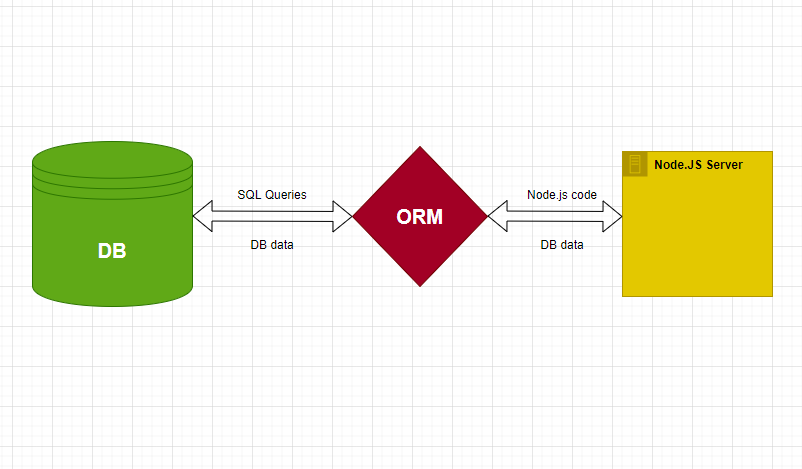


Рисунок 3.6 – Загальний принцип ORM

### 3.2.4. JWT Token. Авторизація.

JSON Web Token (JWT) — це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити й довіряти, оскільки вона має цифровий підпис. JWT можна підписати за допомогою секретного (за допомогою алгоритму HMAC) або пари відкритих і закритих ключів за допомогою RSA або ECDSA.

В загальному вигляді JWT містить в собі наступні пукти:

- Header

- Signature

- Payload

Зазвичай він набирає наступного вигляду

xxxx.yyyyyyyy.zzzz.

Заголовок зазвичай складається з двох частин: типу токена, яким є JWT і використуовуваного алгоритму підписання, наприклад HMAC, SHA256 чи RSA. Потім цей JSON кодується Base64Url, щоб утворити першу частину JWT.

Корисне навантаження

Друга частина токена — це корисне навантаження, яка містить вимоги. Claims — це заяви про сутність (як правило, користувача) та додаткові дані. Існує три види claims: зареєстровані, публічні та приватні claims.

Public claims: це набір попередньо визначених заяв, які не є обов’язковими, але рекомендованими, щоб забезпечити набір корисних, взаємодіючих заяв. Деякі з них: iss (емітент), exp (термін дії), sub (предмет), aud (аудиторія) та інші.

Private claims: це спеціальні позови, створені для обміну інформацією між сторонами, які погоджуються їх використовувати, і не є ні зареєстрованими, ні публічними претензіями.

Щоб створити частину підпису, ви повинні взяти закодований заголовок, кодоване корисне навантаження, секрет, алгоритм, зазначений у заголовку, і підписати це.

Підпис використовується для перевірки того, що повідомлення не було змінено в дорозі, і, у випадку маркерів, підписаних закритим ключем, він також може підтвердити, що відправник JWT є тим, за кого він це каже.

Склавши це все разом, ми отримуємо JSON Web Token, за допомогою якого ми будемо реалізовувати авторизацію. Клієнт з кожним запитом через Web Sockets та через HTTP протоколи буде прикріплювати підпис, за допомогою якого ми будемо визначати, який саме користувач запитує дані, та чи є в нього доступ на ці дані.

В сервісі чату, ми зможемо перевіряти, чи повідомлення належить даному користувачеві, чи іншим «юзерам». Відповідно до цього ми будемо застосовувати ті чи інші стилі.

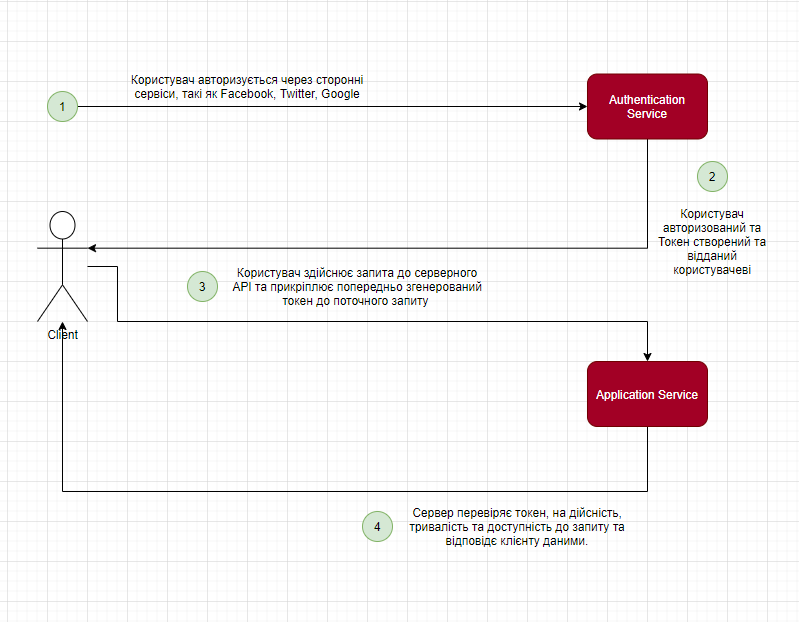


Рисунок 3.7 – Загальний принцип роботи з JWT Token

На рисунку 3.7 також зоображено додатковий сервер аутентифікації у вигляді сторонніх сервісів, таких як Google, Twitter та інші, які через додаткові протоколи такі як Auth0 чи OAuth2.0. Такий сервер є опційним та не обов’язковим.

В ході розробки поставленого завдання буде релізовано власний сервер аутентифікації, який буде генерувати токен самостійно та надавати його клієнту, без втручання сторонніх API [14].

### 3.2.5. AWS S3 Bucket.

Amazon Simple Storage Service (Amazon S3) — це служба зберігання об’єктів, яка пропонує провідну в галузі масштабованість, доступність даних, безпеку та продуктивність. Клієнти всіх розмірів і галузей можуть використовувати Amazon S3 для зберігання та захисту будь-якої кількості даних для різних випадків використання, таких як озера даних, веб-сайти, мобільні додатки, резервне копіювання та відновлення, архівування, корпоративні програми, пристрої Інтернету речей та великі дані. аналітика. Amazon S3 надає функції керування, щоб ви могли оптимізувати, організувати та налаштувати доступ до своїх даних відповідно до ваших конкретних вимог бізнесу, організації та відповідності.

В поданій задачі було обрано AWS S3 Bucket, як хмарну технологія, збергігання даних, а саме книжки, для їх подальшого скачування іншими користувачами. А також для використання S3 використовується AWS SDK для Javascript, що забезпечує, інтеграцію AWS в додатки розроблені на базі Node.js [15].

Загальний принцип зберігання даних на Amazon S3 Bucket наступний:

Користувач завантажує файл на платформі, розроблений API посилає даний файл на сервіс Amazon, останній зберігає файл та повертає його адресу, API зберігає адресу файлу в базі даних, та при кожному наступному запиті на поданий файл за допомогою адреси витягує поданий файл та повертає його клієнту.

Плюси даного сервісу наступні:

- Можливість зберігання великого об’єму інформації

- Розвантаженість серверу.

Так як ми не зберігаємо жодну інформацію не пов’язану з source кодом ми не «засмічуємо» сервер надлишковою інформцією. Дозволяючи працювати йому в оптимальному режимі, зберегши функцію зберігання великого потоку та об’єму даних.

## 3.3. Практична реалізація.

### 3.3.1. Імплементація сервісу аутентифікації та авторизації.

Так як було згадано в розділі 3.2, для верифікації особи на нашій платформі буде здійснено імплементацію з JWT Token, для цього розробимо сервіс, що буде здійснювати даний алгоритм. Для виконання цієї задачі було використано бібліотеку «jsonwebtoken», котра надає функціонал шифрування та дешифрування інформації, котру нам потрібно перевіряти.

Для збереження юзера в базі даних, було створено наступну схему (рис. 3.7)

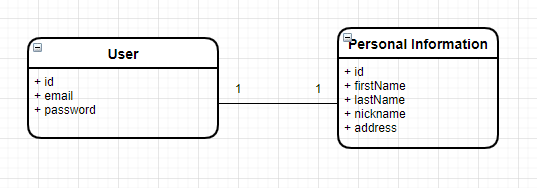


Рисунок 3.8 – Схема User-Personal Info в Базі даних

Після цього був реалізований алгоритм, зоображений на рисунку 3.9, для логіну юзера на платформі.

На блок схемі позначені можливі випадки, коли сервер може відхилити запит на доступ до ресурсів чи реєстрацію користувача.

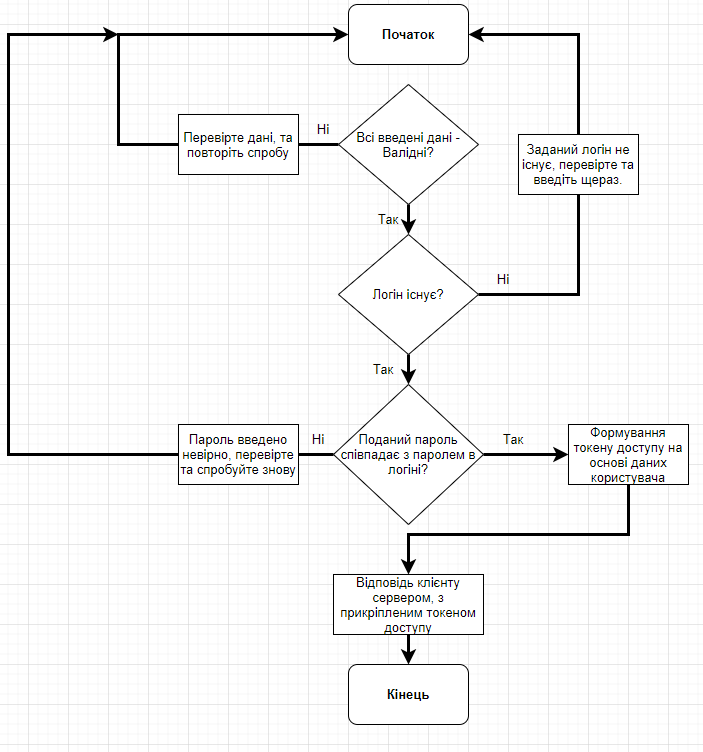


Рисунок 3.9 – Блок-схема алгоритму авторизації

Варто зазначити, що перевірка паролю, та його зберігання в базі даних повинно відбуватись у хешованому, а не шифрованому вигляді, через мірування безпеки. Так як хешування – це одностороння операція, тобто захешований пароль, конвертувати у звичайни ми не можемо а шифрування – це двостороння операця, тобто будь-хто, має ключ, зможе розшифрувати пароль та получити конфедеційні дані, так само, через цю умову пароль не може зберігатись у своєму «нативному» вигляді.

Також у токена, котрий який формується є певні його характеристики, котрі перевіряються на рівні доступу до будь-яких ресурсів.

- Підпис

- Час дії

- Дані

При його розшифруванню перевіряється тип юзера, якщо це потрібно та перевіряється можливість даного юзера звертатись до конкретного функціоналу, наприклад: звичайний юзер не повинен мати доступ до функцій адміна. Також перевіряється час дії, якщо токен не проходить наступне рівннянн, яке наведене у формулі 3.1

де Tn – теперішній час,

Tk – час виходу токена в мілісекундах.

Якщо дана перевірка здійснюється, то такий токен вважається не валідним, а отже повертається помилка токена клієнту.

Як вже було сказано раніше, за допомогою токену буде здійснюватись перевірка на можливість здійснення конкртної операції – конкретним юзером, якщо така перевірка не проходить то повертається HTTP статус 403 – що несе в собі інформацію про заборону доступу до такого функціоналу.

Процедура реєстрації на платформі здійснюється за наступною блок-схемою, що наведена на рисунку 3.10

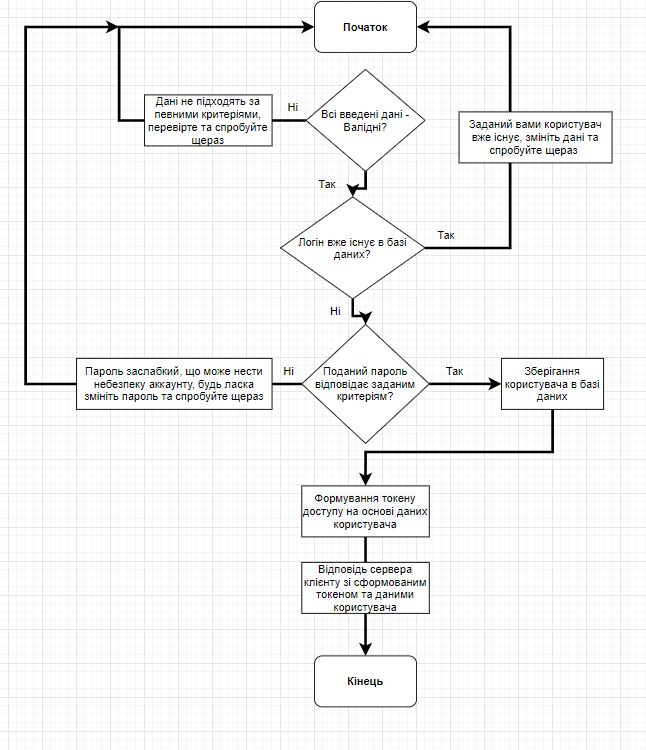


Рисунок 3.10 – Блок-схема авторизації користувача

Перевірка паролю здійснюватиметься за певними критеріями та буде визначатись складність паролю за трьома категоріям: Складний, нормальний, та слабкий. Якщо пароль попадає під категорю – слабкий, то такий вважатимемо не валідним, та просимо користувача ввести новий, або доповнити старий.

Визначемо критерії для кожної категорії:

- Сильний (Strong password)

Містить щонайменше одну малу літеру, одну велику літеру, одну цифру, один спеціальний символ і має принаймі вісім символів.

- Нормальний (Normal password)

Якщо пароль містить щонайменше шість символів і відповідає всім іншим вимогам «сильного паролю», або не містить цифри, але відповідає решті вимог.

- Слабкий (Weak password)

Якщо введений пароль не відповідає вимогам сильного або середнього рівня, він вважається слабким.

Для цього реалізуємо перевірку за наведеними схемами, та зберігатимемо стан паролю в локальній змінній, забороняючи користувачеві закінчувати реєстрацію, якщо його введений пароль не відповідає нашим характеристикам.

Також виводитимемо підказку, в якому стані знаходиться наш пароль, та нашими вимогами, щоб користувач розумів, що йому потрібно вводити, там мав змогу зробити пароль, таким який відповідає заданими нами критеріями безпеки. Для відображення статусу пароля використовуватимемо наступні кольори:

- Зелений (Strong Password);

- Жовтий (Normal Password);

- Червоний (Weak Password).

Відповідно до описаних правил, виводитимемо відповідну підказку користувачу.

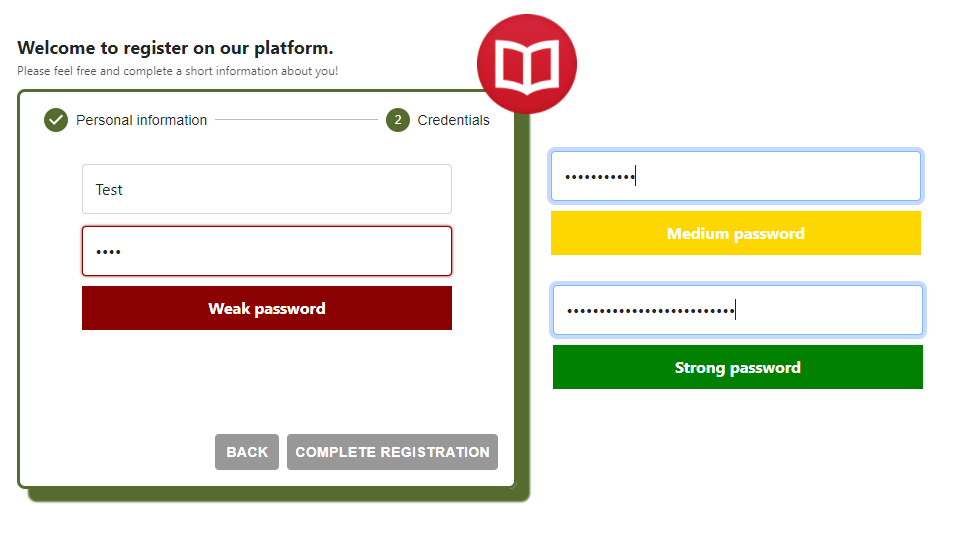


Рисунок 3.11 – Процес реєстрації на платформі

А також процес аутентифікації, котрий містить поля для введення мейл-адреси, для того що перевірити даного користувача на наявність його в базі даних сервера та співпадння паролів.

При перевірці даних, перед цим як було показано на блок-схемі (рис. 3.9) перевіряються данні на валідність, а саме:

- Email

Повинен бути дійсним мейлом, та відповідати всім вимогам стандарту мейл-адрес, не повинен бути пустим.

На рисунку 3.12 зоображено процес аутентифікації користувача на платформі, де користувач платформи повинен ввести email та пароль, для доступу до ресурсів платформи.

У разі відхилення сервером запиту про доступ до платформи, користувач отримує відповідне сповіщення.

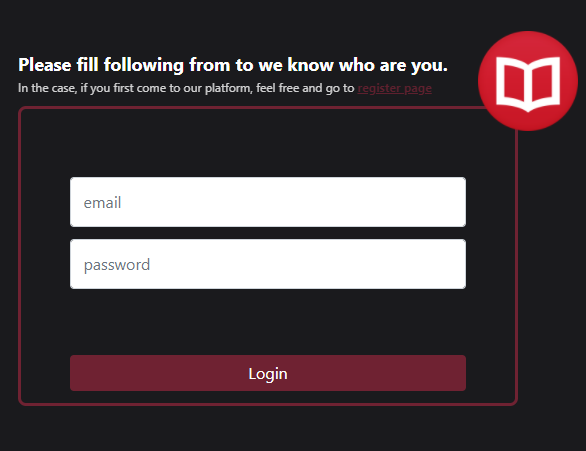


Рисунок 3.12 – Процес аутентифікації

### 3.3.2. Процес імплементації чату, та спам-фільтру.

#### 3.3.2.1 Визначення схем чату.

Для імплементації чату та алгоритму класифікації спаму нам потрібно визначити декілька схем:

- room

- message

- frequency\_table

- naïve\_bayes\_data

Зообразимо вище згадані схеми на діаграмі (рис. 3.13), та опишемо їх.

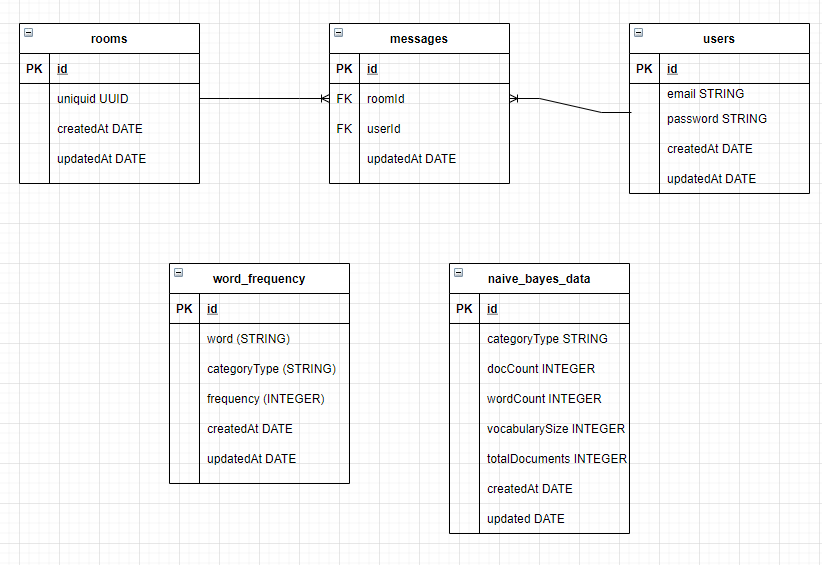


Рисунок 3.13 – Схеми чату бази даних

Отже основні таблички створені для реалізації чату – rooms, messages, users. Кожна книжка має свою кімнату для обговорення, та при її створенні їй присвоюється унікальний номер (UUID), який складається зі 16 символів. Відповідно можна вважати, що чат = кімнаті. Кімната в свою чергу зв’язана зв’язком «one to many» з табличкою повідомлень через зовнішній ключ roomId та звичайно кожен користувач може писати в цю кімнату, отже в зв’язок між табличками користувачі та повідомлення також має місце бути «one to many». За допомогою таких зв’язків ми з легкістю зможемо витягувати дані про кімнату, її повідомлення та повідомлення конкретного користувача.

Також на схемі 3.13 зоображені таблиці потрібні для реалізації алгоритму «Наївного Байєса», а саме word\_frequency для обрахунку частоти появи слова в тій чи іншій категоріїї та таблиця naïve\_bayes\_data для обліку всіх документів, слів в тій чи іншій категорії, які в подальшому будуть використані для реалізації алгоритму та обрахування ймовірності категоризації повідомленян як спам чи не спам.

#### 3.3.2.2 Імплементація веб-сокетів.

Для імплементації чату була використана технологія Web Sockets, яка була описана в пункті 2.1 даного розділу.

Сокети дозволяють встановити повнодуплексне з’єднання між клієнтами та сервером, та дає змогу в реальному часі обмінюватись повідомленнями. Для реалізації схеми зоображеної на рисунку 3.2, потрібно імплементувати деякі так званні «Listeners» - це слухачі, котрі дозволяють реагувати певним чином на подію, котру ми прослуховуємо.

Для реалізації поставленої задачі, на стороні серверу, нам потрібно прослуховувати наступні дії:

* connection
* message
* close

Розглянемо кожну з них детально, та опишемо що потрібн робити при потраплянні в callback кожного з цих подій.

Першим слухачем є connection. За допомогою Node.js та бібліотеки ws (web sockets), створимо сервер, виокремивши його в сервіс та опишемо згадану подію.

Для того щоб створити сокет-сервер, тобто відкрити з’єднання для клієнтів, використаймо наступну команду, що наведена в лістингу А.1. Після того, як сокет сервер створено потрібно описати наших слухачів.

Connection. Для цього виклачемо команду on бібліотеки «ws» та передамо їй функції onConnect, яка описана в лістингу А.3, як можна побачити, ми використовуємо той самий метод аутентифікації користувача, що правда, через обмеженість сокетів, вони не мають заголовку, токен доступу потрібно передавати через query. Ідентифікуємо користувача, та заносимо в список слухачів даного з’єднання.

Та передамо функції прослуховування «message» та «close» даному користувачеві, де перший слухач чекає на повідомлення, котре буде надіслано користувачеві, а другий очікує закриття з’єднання.

Відповідно передамо функцію «handleMessage», яка представлена в лістингу 3.1, в яку передаються дані про сокет з’єднання та токен користувача, який в подальшому буде використаний для витягу з нього даних про останнього, та визначення чи належить йому ітерований меседж чи ні. Належність повідомлення певному користувачу здійснюється за допомогою зовнішнього ключа, який було представлено в базі даних на рисунку 3.13.

const messageHandle = async (wsClient, message, wsServer, token) => {  
 try {  
 const user = JwtService.*verifyUser*(token) || {};  
  
 const jsonMessage = ***JSON***.parse(message);  
 const { roomId, data: messageData } = jsonMessage;  
  
 const classifier = bayes();  
 const test = await classifier.categorize(messageData);  
  
 if (test === EmailClassifier.*SPAM*) {  
 wsClient.send(***JSON***.stringify(ApiResponse.*sendSocketException*('Sorry, but we detected spam in your message.'), null, '\t'));  
 return;  
 }  
  
 await MessageService.*createMessage*(user, { text: messageData, roomId})  
  
 await ***Bb***.map(***Object***.values(clients.*clientsList*), async (client) => {  
 const newMessageList = await MessageService.*getMessages*(client.user, roomId);  
 client.socket.send(***JSON***.stringify(ApiResponse.*sendThroughSocket*(newMessageList), null, '\t'));  
 })  
 }

*Лістинг 3.1 Функція «handleMessage».*

Для простоти подальшого опису поданого алгоритму наведемо блок-схему що описуєм згадану функцію.

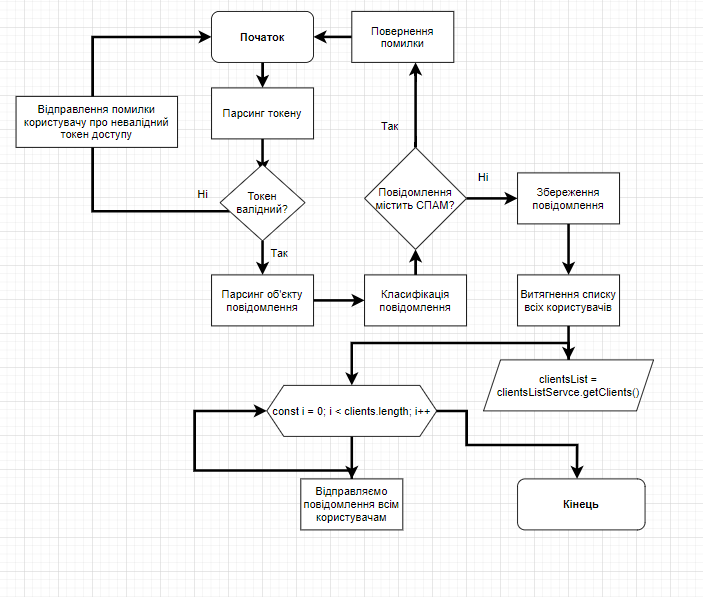


Рисунок 3.14 – Алгоритм обробки повідомлення

Як можна бачити спершу дістається дані юзера з токену, після чого відувається парсинг JSON об’єкта який прийшов з клієнта, дістається ідентифікаційний номер кімнати та класифікується меседж, якщо повідомлення класифіковане алгоритмом Байєса, як спам таке повідомлення вважається таким що не пройшло перевірку та сервер повертає вийняток, зі статусом 404 (Bad Request) та повідомленням про те, що повідомлення містить потеційним спам.

Витягнення списку повідомлень та його збереження обробляється у розробленому сервісі «MessageService», який відповідає за опрацювання бізнес логіки чату та наведений в лістингу А.4.

#### 3.3.2.3. Імплементація алгоритму Байєса.

Для розробки спам фільтру використано алгоритм Байєса, котрий заснований на формулі 2.1. Для розробки даного алгоритму, нам потрібно сворити модуль, з наступними функціями:

**-** train;

**-** categorize;

- tokenProbability;

- frequencyTable;

- tokenize;

- initialize;

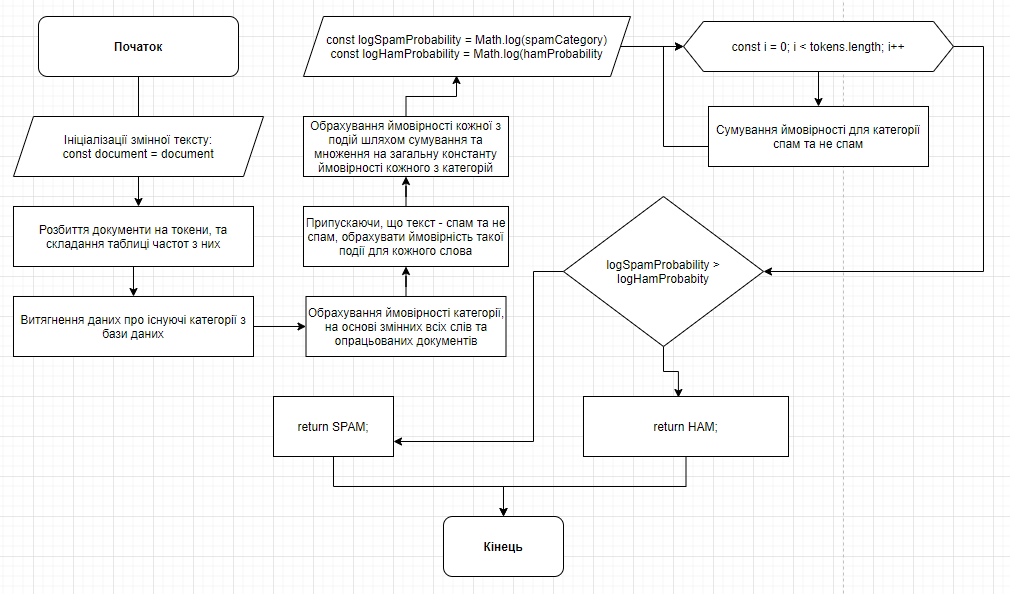


Рисунок 3.15 – Алгоритм обрахування категорії

Ключовим методом даного алгоритму, розглядаючи в цілому для нас є функція categorize, саме вона буде здійснювати визначення чи містить наше повідомлення потеційний спам чи ні.

Розглянемо зоображений на рисунку 3.15 алгоритм.

Спочатку проводиться навчання алгоритму, тобто йому передається набір текстів, котрі попередньо вже визначені як spam чи ham, алгоритм розбиває текст на токени, обробляє їх та визначає ймовірність появи кожного з них, підраховує скільки раз слово появлялось в ham/spam повідомленнях та заносить у відповідно табличку дані (word\_frequency) після чого, на основі цих даних за алгоритмом 3.15 здійснюється обрахування ймовірності слів в повідомленні як спам. Після обрахування ймовірностей робиться припущення що подане повідомлення може містити спам на віддається клієнту помилка, що в повідомленні міститься спам, як це було описано у функціїї «handleMessage» у пункті 3.2.2 даного розділу.

Одним з основних обчислень, є також перевірка ймовірності слова на ту чи іншу категорію, яка здійснюється за наступною формулою:

де probability – ймовірність того що слова належить тій чи іншій категорії, в нашому випадку (Spam, Ham),

frequency – частота появи розглядаємого слова в певній категорії

countOfWord – кількість всіх слів в даній категорії

vocabularyLength – кількість всіх слів загалом.

#### 3.3.2.4 Візуалізація чату

Для реалізації чату було використано фронт-енд частини в аплікації ми використовуємо React.js та Web Sockets та стороні клієнта, так само як на сервері ми опишемо з’єдання з нашими сокетами. Для цього виокремимо це в сервіс «socket-service». Також опишемо з’єднання з сокетом та messageHandler. Повний код сервісу наведено в лістингу А.5. Для перевірки та рендерингу повідомлень створимо наступну функцію в компонені «Сhat».

const send = () => {  
 const fakeRoom = 'a7566bda-24fa-4e52-b4a7-0ad603349605';  
 setMessage('');  
 if (!floodProtection.check()) {  
 enqueueSnackbar('Please stop flooding, and wait for a seconds', { variant: 'error' });  
 return;  
 } else {  
 if (isSocketConnectionOpen()) {  
 sendMessage(***JSON***.stringify({data: message, roomId: fakeRoom}));  
 } else {  
 setTimeout(reopenConnection, 5000);  
 }  
 }  
}

*Лістинг 3.2. Опис фукції відправлення повідомлення*

У функції відправки повідомлення використовуються декілька сервісів:

- socket-service.js

- flood-protection.js

FloodProtection сервіс, для запобіганню відправці повідомення швидше ніж вказана кількість повідомлень за вказану кількість часу, для цього побудуємо його (лістинг А.6) та виводитимо відповідну помилку, якщо користувач флудить, також використовуватимемо socket-service (лістинг А.5) для встановлення сокет-з’єднання з сервером.

Приклад реалізованої компоненти наведено на рисунку Б.1, також на рисунку Б.2 представлена реалізація повідомлень, з відповідною відповіддю сервера про помилку, чи про успішно доставлене повідомлення.

### Імплементація аналізу відгуку.

Відповідно до поставленої задачі, було реалізовано алгоритм аналізу відгуку. Аналіз відгуку виконаний в кілька етапів:

- Позбавлення апострофів – скорочення.

В англійській мові дуже часто два слова скорочуються в одне, за допомогою апострофа, наприклад речення: «I will do something», можна скоротити в наступну форму «I`ll do something» віпповідно аналізувати таку конструкцію складніше, тому нам потрібно її перетворити в сталу форму.

- Привести весь текс до нижнього реєстру.

- Видалити всі пунктуації з тексту, та пробіли, лишити «чисті» слова.

- Провести алгоритм токенізації тексту для перетворення тексту на масив слів.

- Виконати алгоритм «Stop words».

- Виконати стемізацію тексту, для перетворення слів в початкову форму.

- Оцінити кожне слово, згідно його рейтингу підсумувати, та отримати результат.

- Якщо сума більша за 0, вважати такий текст позитивним, менша за 0 – негативним, в іншому випадку нейтральним.

Для зберігання та відображення відгуків, реалізуємо сторінки відображення детальної інформації про книжку. Де розміщена детальна інформація та статистика про книжку, фото обкладинки та посилання на скачування.

Всі потрібні ресурси будемо зберігати за допомогою хмарної технології AWS S3 Bucket. Для цього скористаємось Aws SDK для Node.js та опишемо сервіс для збереження файлів в нашому хмарному сховищі.

Попередньо сконфігуривши всі потрібні дані, створимо S3 Bucket, давши йому унікальне їм’я та вибравши потрібний регіон. Після чого згенеруємо ключі, та запишемо їх в .env файл.

Після чого створимо сервіс, який буде відповідати за роботу нашої «хмари». (Додаток А.7) За допомогою створеної функції зберігатимемо при створені нової книжки всі потрібні нам файли в ньому (рис. 3.16).

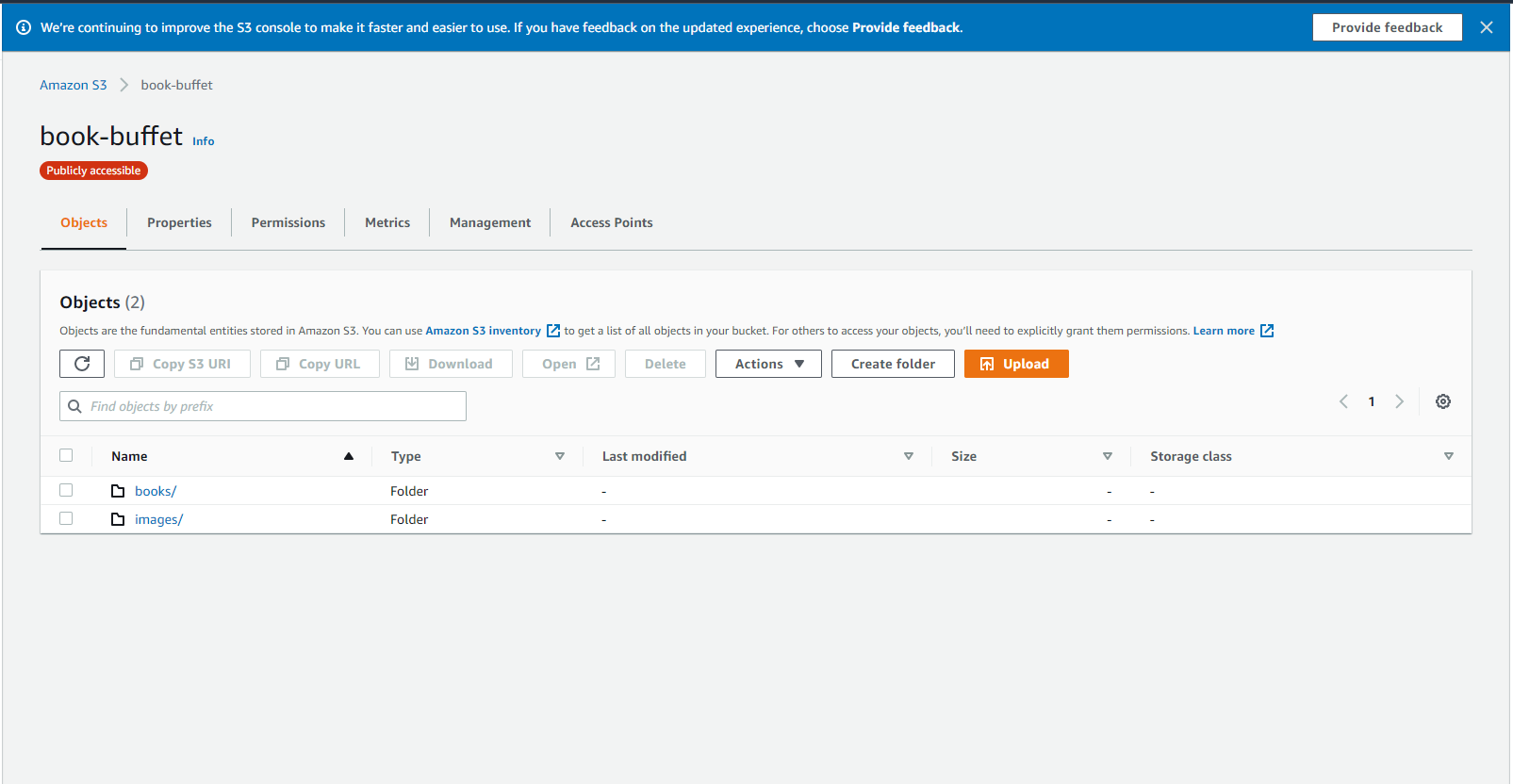


Рисунок 3.16 – S3 Bucket «Book-buffet»

На рисунку 3.16 зоображена детальна сторінка книжки. На сторінці присутні декілька «активних» кнопок: «Favorite», «Download», «Go to buy».

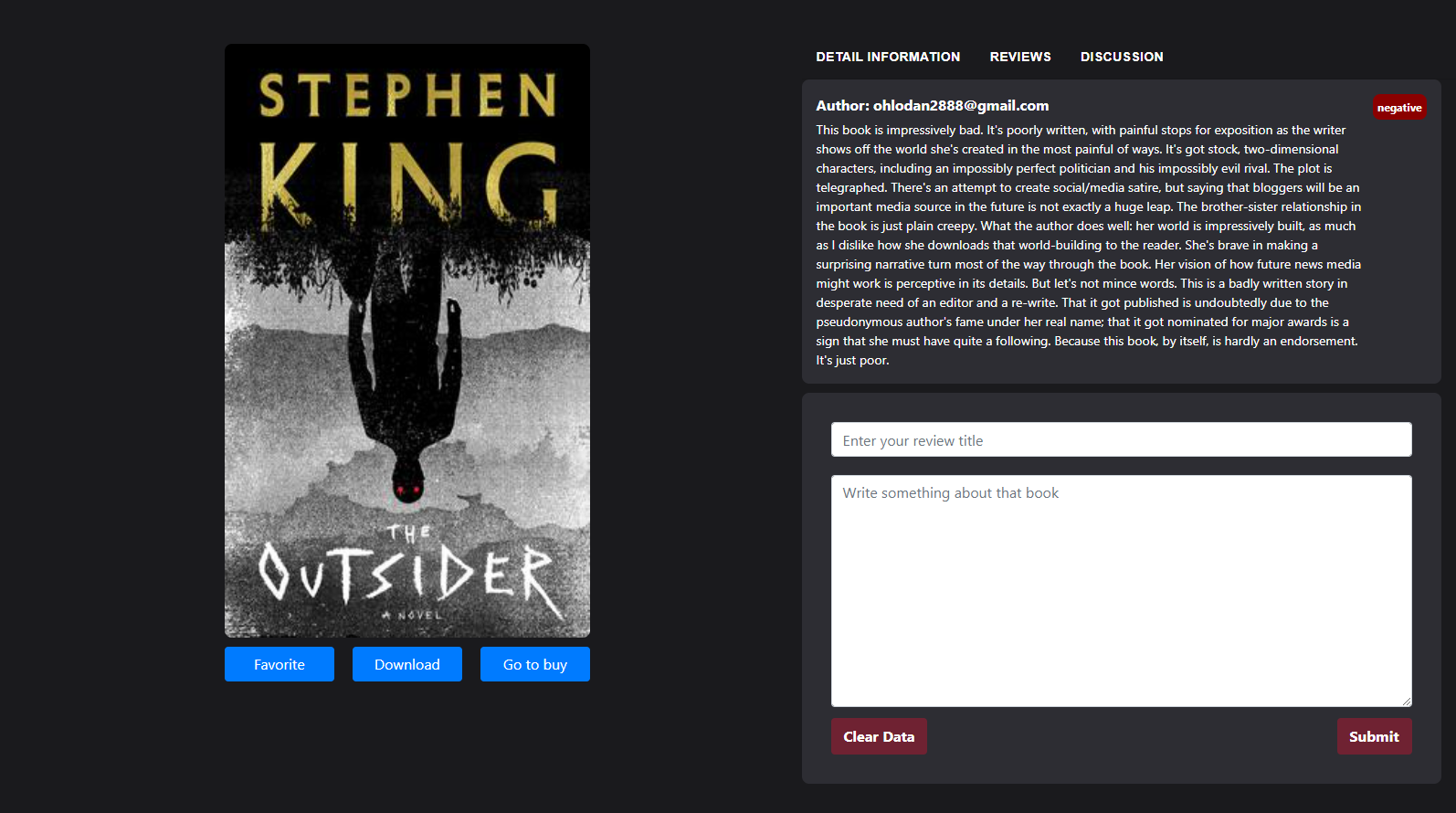


Рисунок 3.17 – Детальна сторінка книжки

Також на рисунку користувач знаходиться на вкладці «Review», де він може переглядати вже існуючі відгук користувачів та додавати свої власні. На основі вище поданого алгоритму, система аналізує відгук та видає один з трьох результатів:

* Positive
* Negative
* Neutral

Та зберігає відгук в базу даних. Помічений відповідною позначкою.

В ході розробки практичного завдання також реалізована адмін панель, для перегляду статистики відгуків по певних книжках, фільтрування книг за певними критеріями та їх пошук.

Повний вміст зоображений в додатках Б.

### 3.4.1 Дослідження розробленої системи

На основі розробленої системи, зробимо наступні висновки:

- Ефективність.

- Швидкість виконання.

- Точність.

На основі розробленого сервісу для запобіганню відправки «спам» повідомлень визначемо вище наведені характеристики для нього.

Ефективність роботи сервісу та точність класифікації.

Для аналізу ефективності роботи поданого сервісу, візьмемо декілька повідомлень:

1. You are a winner U have been specially selected receive 1000 cash or a 4\* holiday (flights inc) speak to a live operator claim 0960962714.
2. Hi there. What do you think about this book?
3. Congratulations! you won a 5000$ and if you want to receive money, you just need to send sms on the following number: 0123211.
4. I found this book interesting, the plot excited me, and i do recommend to read the book to all my friends.
5. Maybe someone want to buy my book?
6. If somebody doesn't know where you can buy the best cakes. Welcome to my bakermarket!

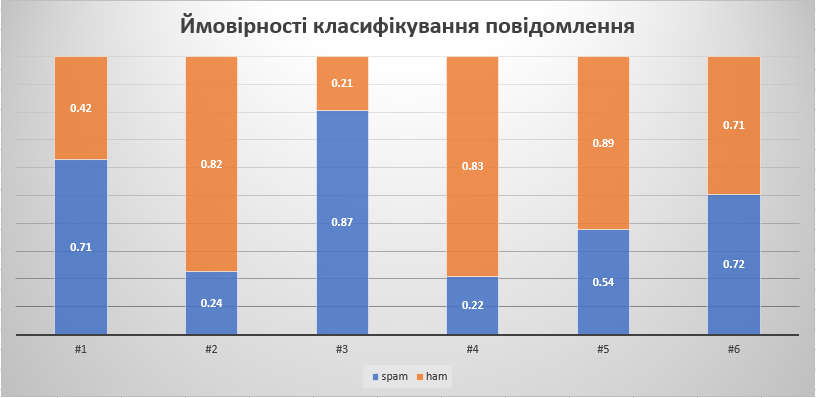


Рисунок 3.18 – Ймовірності класифікування повідомлення

Виходячи з наступного класифікування «спам» повідомення – це масове розсилання кореспонденції рекламного чи іншого характеру людям, які не висловили бажання її одержувати, отримані результати очікувались наступні: повіомлення під номерами 1,3,5,6 – «спам», 2,4 – «звичайні».

З діаграми на рисунку 3.18, зоображено, що після класифікування системою, отримані результати були дещо іншими. А саме повідомлення під номером 5 – класифіковане як «звичайне» та повідомлення під номером 6 – знаходилось на межі класифікації.

Проблеми класифікації повідомлень, розробленою системою наступні:

- Малий обсяг навчальних даних.

- Похибка алгоритмів обробки природної мови

З рештою повідомлень, алгоритм впорався, та отримані результати – які очікувались.

Середня швидкість виконання класифікації на поданих прикладах складає – 361.379 мілісекунд «чистого» часу на сервера, що дорівнює ~600 мілісекунд, враховуючи час на відправлення повідомлення сокетом клієнту.

# ВИСНОВКИ

Оскільки понад велика кількість корпоративних даних є неструктурованими, зрозуміло, що ці дані містять важливу бізнес-ідею, яка залишається невикористаною.

Не беручи до уваги, що структуровані дані мають велике значення, неструктуровані дані надають величезну кількість знань, які цифри та статистика просто не можуть пояснити.

Організації повинні знайти способи управління та аналізу неструктурованих даних, щоб вони могли використовувати їх для прийняття важливих бізнес-рішень, надаючи їм конкурентну перевагу над своїми конкурентами. Компанії, які не враховують неструктуровані дані, втрачають величезну кількість бізнес-аналітики.

Так за допомогою аналізу не структурованих даних, можна розпізнавати зображення, аудіо, відео. Проводити класифікацію, аналізувати текстові дані.

Наприклад фільтрація спам повідомлень, дозволяє позбутись менеджерів чатів чи сервісів, краще аналізувати відгуки на онлайн платформах, що в подальшому може покращити сервіс компаній, після перегляду аналітики по своїй продукції.

Отже неструктуровані дані відіграють дуже важливу роль в нашому житті, тому щоб бути попереду конкурентів компаніям потрібно пристосовуватись до теперішніх реаліїв, де світ наповнений такою інформацією, яку потрібно обробляти правильно та видобувати з неї корисну її частину, щоб рухатись вперед та бути на крок вперед інших.

Виявлення прихованої цінності у ваших неструктурованих даних відкриває раніше закриті двері, які ведуть до безлічі невикористаних ідей. Хоча видобуток неструктурованих даних має свої проблеми, визнання їх цінності та використання правильних інструментів може дати вашому бізнесу конкурентну перевагу та заощадити тисячі людино-годин при ручній обробці, які ведуть за собою втрату не одних мільйонів грошових одиниць.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* 1. What is metadata? – стаття. Режим доступу до ресурсу - https://www.opendatasoft.com/blog/2016/08/25/what-is-metadata-and-why-is-it-important-data (дата звернення - 14.07.2021)
  2. R. Kibble. Introduction to natural language processing Department of Computing, Goldsmiths., 2013
  3. Sergios Theodoridis. Machine Learning. A Bayesian and Optimization Perspective., 2020
  4. Vojislav Kecman. Support Vector Machines – An Introduction., 2005
  5. Chris Smith. Decision Trees and Random Forests: A Visual Introduction For Beginners., 2017
  6. Linear Discriminant Analysis for Machine Learning – стаття. Режим доступу до ресурсу - <https://www.knowledgehut.com/blog/data-science/linear-discriminant-analysis-for-machine-learning> (дата звернення - 18.07.2021)
  7. Tokenization – стаття. Режим доступу до ресурсу - <https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html> (дата звернення - 25.07.2021)
  8. Porter Stemmer Algorithm – стаття. Режим доступу до ресурсу - <https://vijinimallawaarachchi.com/2017/05/09/porter-stemming-algorithm/> (дата звернення – 30.08.2021)
  9. A Gentle Introduction to the Bag-of-Words Model – стаття. Режим доступу до ресурсу - <https://machinelearningmastery.com/gentle-introduction-bag-words-model/> (дата звернення – 05.09.2021)
  10. Siddiqui T. and Tiwary U.S., “Natural Language Processing and Information Retrieval”, Oxford University press, 2008.
  11. Naïve Bayes for Machine Learning – стаття. Режим доступу до ресурсу - <https://blog.floydhub.com/naive-bayes-for-machine-learning/> (дата звернення 10.09.2021)
  12. C. Cortes and V. Vapnik. Support-vector networks. Machine Learning, November 1995
  13. Andrew Lombards. WebSocket: Lightweight Client-Server Communications., 2015
  14. JSON Web Token – Grant authentication – стаття. Режим доступу до ресурсу - <https://developers.docusign.com/platform/auth/jwt/> (дата звернення 05.10.2021)
  15. AWS S3 – стаття. Режим доступу до ресурсу <https://searchaws.techtarget.com/definition/Amazon-Simple-Storage-Service-Amazon-S3> (дата звернення 15.09.2021)

# Додаток А.

* 1. Створення веб-сокет серверу.

const wsServer = new WebSocket.Server({ port: 9000 });

* 1. Socket сonnection Listener.

wsServer.on('connection', (wsClient, req) => onConnect(wsClient, wsServer, req));

* 1. onConnect callback.

const onConnect = async (wsClient, wsServer, req) => {  
 const query = url.parse(req.url, true).query;  
 const { token, roomId } = query;  
  
 const user = JwtService.verifyUser(token) || {};  
 clients.saveClient(user, wsClient);  
  
 const initialMessages = await MessageService.getMessages(user, roomId);  
 wsClient.send(JSON.stringify(ApiResponse.sendThroughSocket(initialMessages), null, '\t'))  
  
 wsClient.on('message', (message) => {  
 messageHandle(wsClient, message, wsServer, token);  
 })  
 wsClient.on('close', (message) => {  
 console.log('connection lost');  
 })  
}

* 1. MessageService.js

const models = require('../database/models');  
const { uuid } = require('uuidv4');  
const Bb = require('bluebird');  
  
class MessageService {  
  
 static populateMessage(user, message) {  
 const { id } = user;  
  
 message.ownMessage = message.userId === id;  
  
 return message;  
 }  
  
 static async getMessages(user, roomId) {  
  
 const { id } = await models.Room.findOne({where: {uniqueId: roomId}})  
  
 const messages = await models.Message.findAll({  
 where: {  
 roomId: id,  
 },  
 include: {  
 association: 'user', attributes: ['email']  
 },  
 raw: true,  
 limit: 10,  
 order: [['createdAt', 'DESC']]  
 });  
  
 let resultMessages = [];  
  
 if (messages && messages.length > 0) {  
 resultMessages = messages.map((message) => this.populateMessage(user, message));  
 }  
  
 return resultMessages;  
 }  
  
 static async createMessage(user, { text, roomId }) {  
 const { id } = user;  
  
 let createdRoomId = null;  
  
 const room = await models.Room.findOne({where: { uniqueId: roomId }});  
  
 if (!room) {  
 const roomIdUUID = uuid();  
 const createdRoom = await models.Room.create({ uniqueId: roomIdUUID }, { returning: true });  
 createdRoomId = createdRoom.id;  
 } else {  
 createdRoomId = room.id;  
 }  
  
 const message = await models.Message.create({userId: id, text, roomId: createdRoomId}, { returning: true });  
  
 return message;  
 }  
}

* 1. socket-service.js (Front-end part)

let socket = null;  
  
const initializeSocket = (onMessageCallback) => {  
  
 const token = localStorage.getItem('token');  
 const fakeRoomId = 'a7566bda-24fa-4e52-b4a7-0ad603349605';  
  
 socket = new WebSocket('ws://localhost:9000?token=' + token + '&roomId=' + fakeRoomId);  
  
 socket.onopen = (ev) => {console.log('socket connection was opened')}  
  
 socket.onmessage = (ev) => { handleMessage(ev, onMessageCallback); }  
  
 socket.onclose = (ev) => {console.log('socket connection was closed');}  
  
 return socket;  
}  
  
const sendMessage = (data) => {  
 socket.send(data);  
}  
  
const reopenConnection = () => {  
 socket = null;  
 initializeSocket();  
}  
  
const handleMessage = (ev, onMessageCallback) => {  
 const data = JSON.parse(ev.data);  
 onMessageCallback(data);  
}  
  
const isSocketConnectionOpen = () => {  
 return socket.readyState === socket.OPEN;  
}  
  
export {  
 initializeSocket,  
 sendMessage,  
 isSocketConnectionOpen,  
 reopenConnection  
}

* 1. flood-service.js (Front-end part)

const now = () => Math.floor(Date.now() / 1000);  
  
export default class FloodProtection {  
 constructor(options = {}) {  
 const defaultOptions = {  
 ratingMessage: 1, // unit: messages  
 perSeconds: 80, // unit: seconds  
 };  
  
 this.options = Object.assign({}, defaultOptions, options);  
  
 this.size = this.options.ratingMessage;  
 this.lastCheckOnFlooding = now();  
 }  
  
 check() {  
 const current = now();  
  
 const timeCompleted = current - this.lastCheckOnFlooding;  
 this.lastCheckOnFlooding = current;  
 this.allowance += timeCompleted \* (this.options.ratingMessage / this.options.perSeconds);  
  
 if (this.size > this.options.ratingMessage) {  
 this.size = this.options.ratingMessage; // throttle  
 }  
  
 const canSendMessaged = this.size >= 1.0;  
  
 if (canSendMessaged) {  
 this.size -= 1.0;  
 }  
  
 return canSendMessaged;  
 }  
}

* 1. AwsService.js

const AWS = require('aws-sdk');  
  
const env = process.env.NODE\_ENV ;  
  
const s3 = new AWS.S3({  
 accessKeyId: process.env.aws\_access\_key\_id,  
 secretAccessKey: process.env.aws\_secret\_access\_key,  
});  
  
const saveFile = async (file, folder) => {  
 const params = {  
 Bucket: process.env.bucket\_name + folder,  
 Key: file.name, // File name you want to save as in S3  
 Body: file.data  
 };  
  
 return new Promise((resolve, reject) => {  
 s3.upload(params, function(err, data) {  
 if (err) {  
 reject(err);  
 }  
 resolve(data.Location);  
 });  
 });  
}  
  
module.exports = {  
 saveFile  
}

# Додаток Б

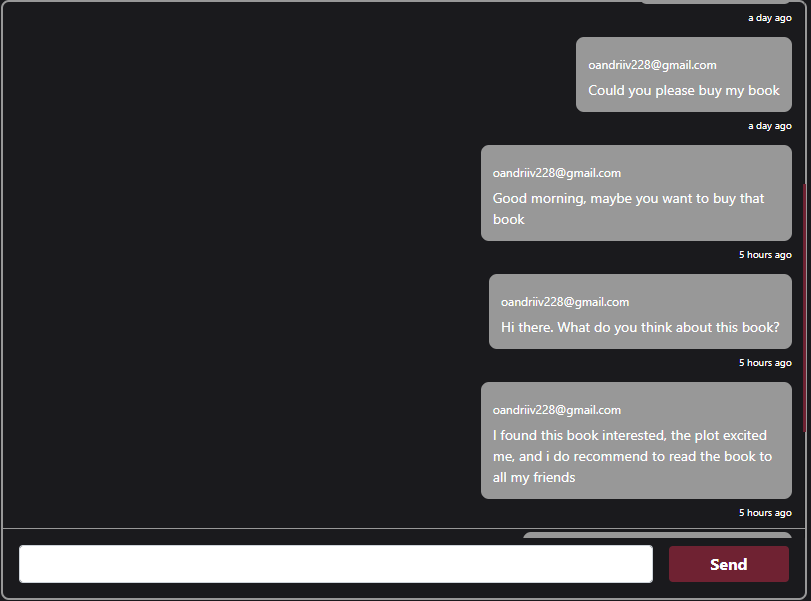


Рисунок Б.1- Компонента-Чат

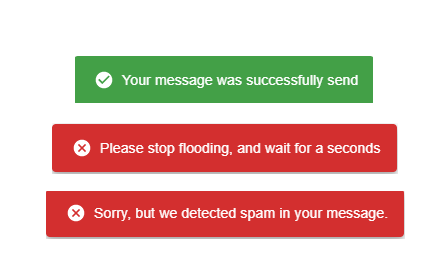


Рисунок Б.2 - Типи повідомлень чату

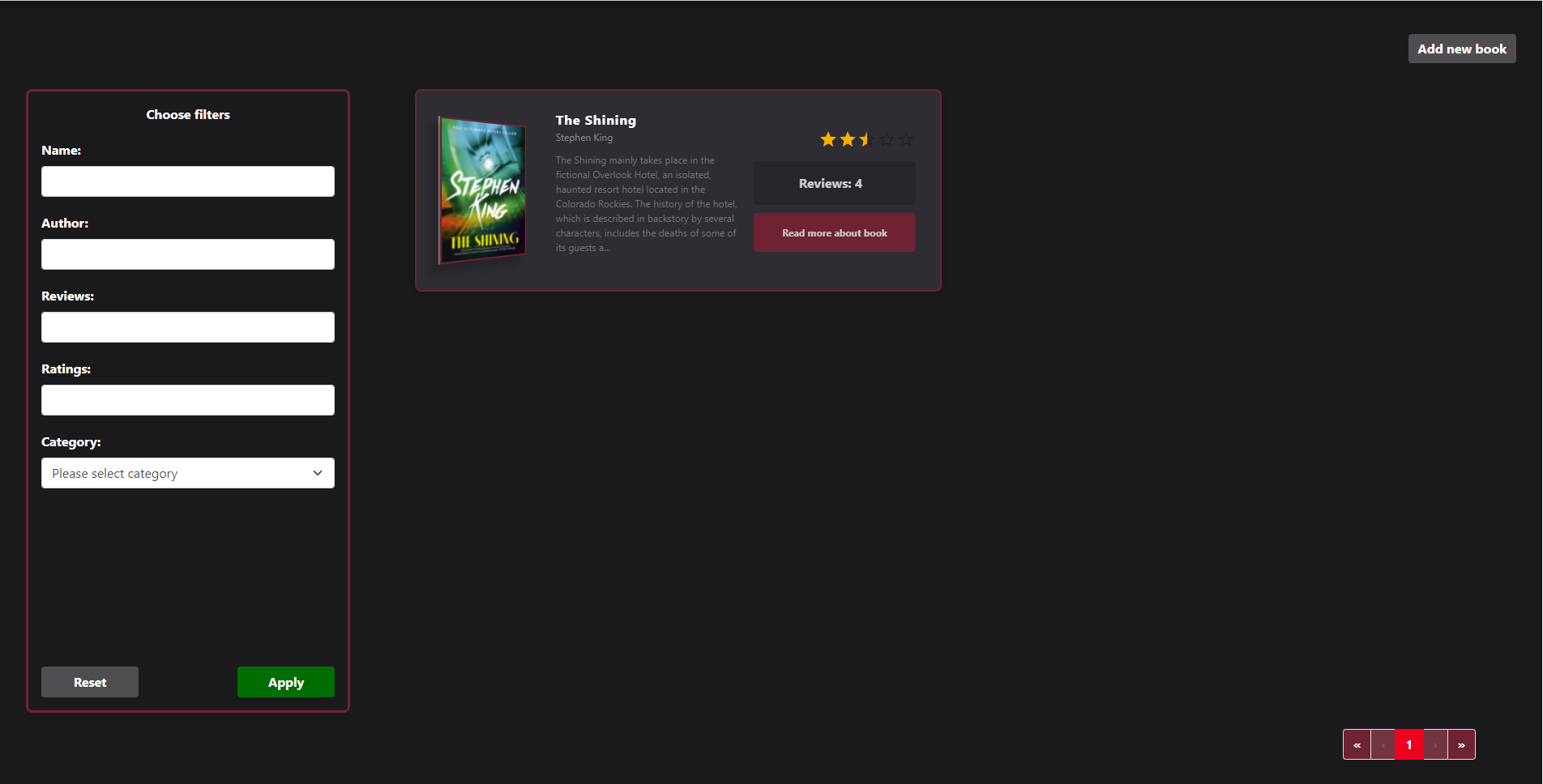


Рисунок Б.3 – Основна сторінка платформи

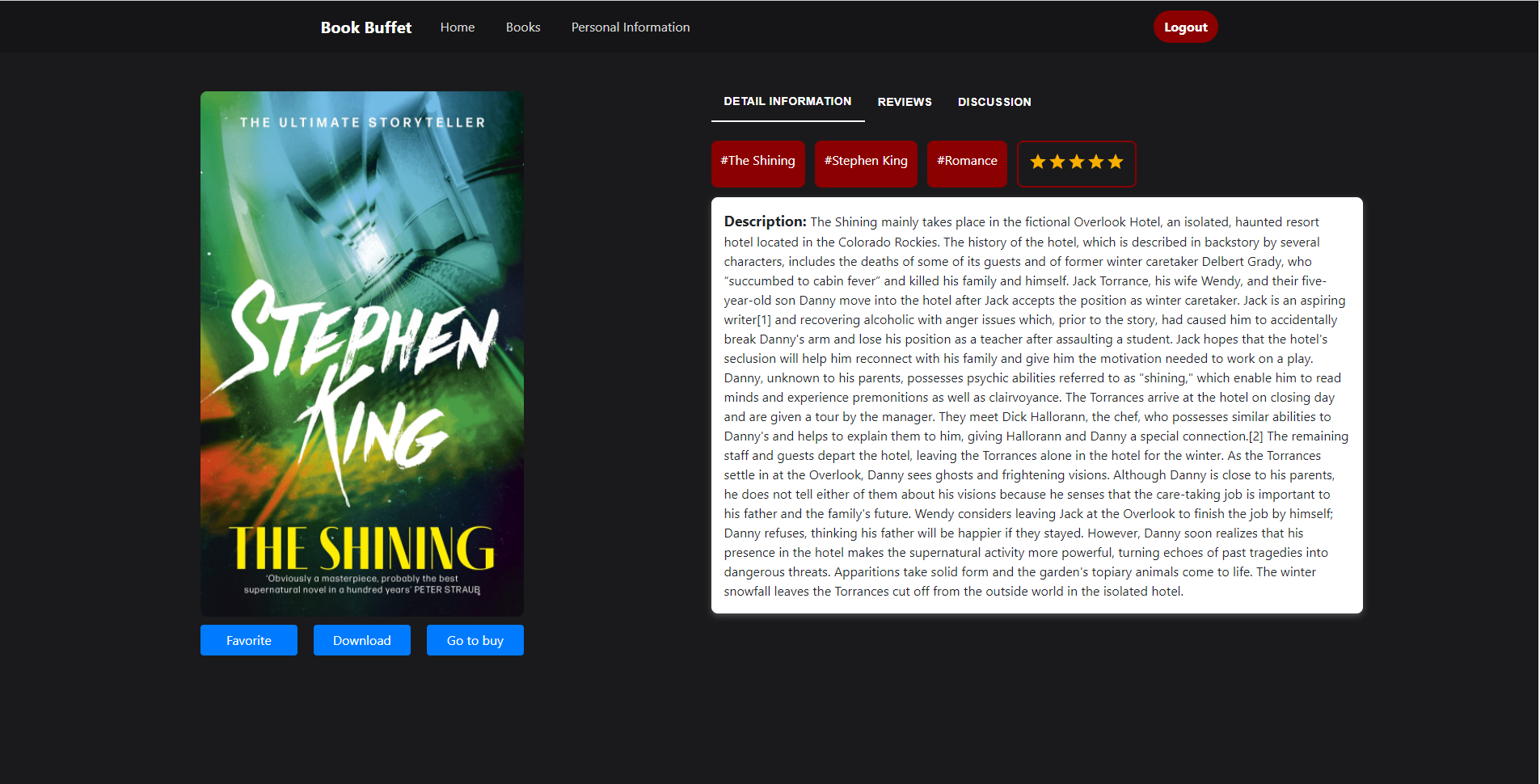


Рисунок Б.4 – Розгорнута сторінка книжки

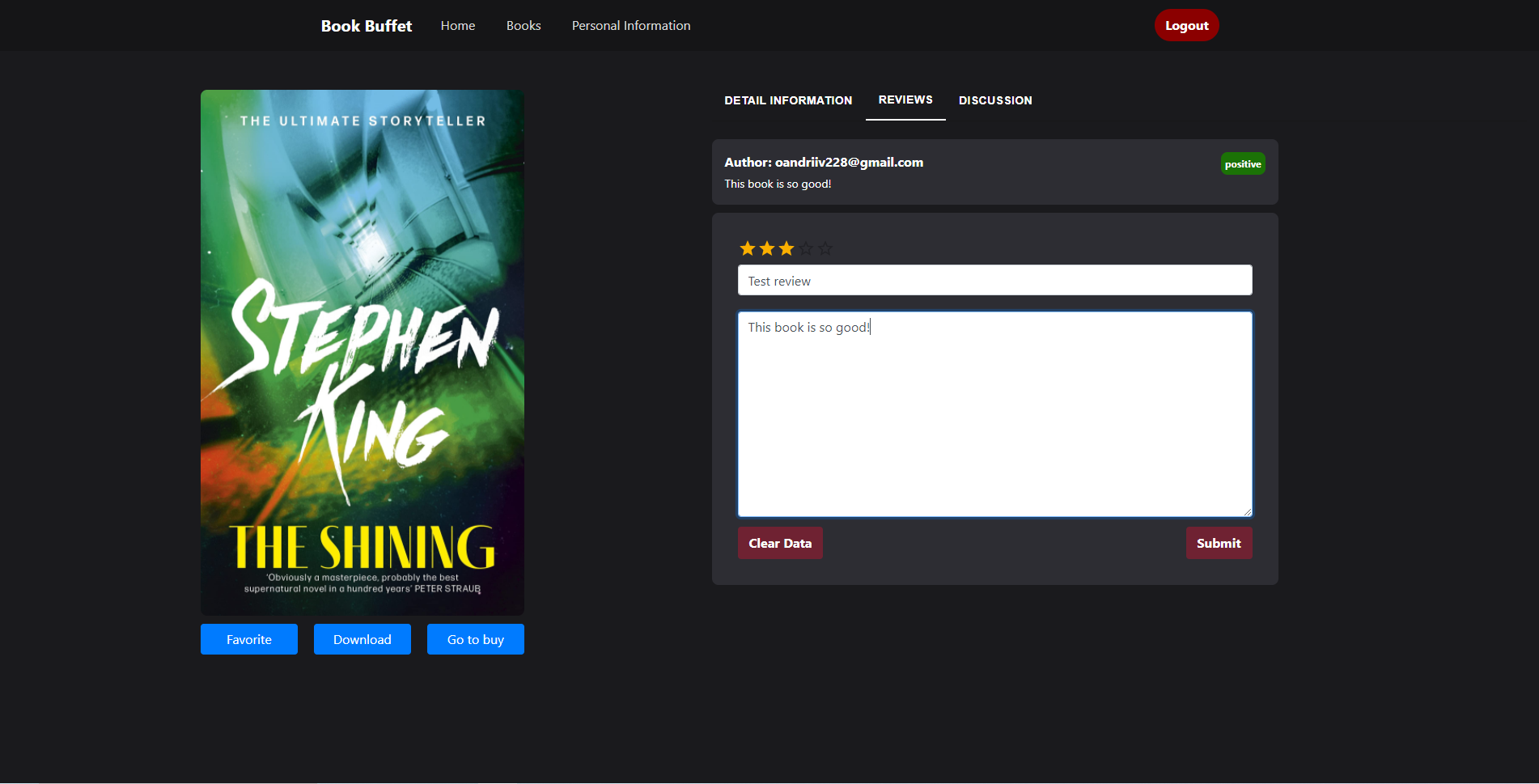


Рисунок Б.5 – Сторінка відгуків

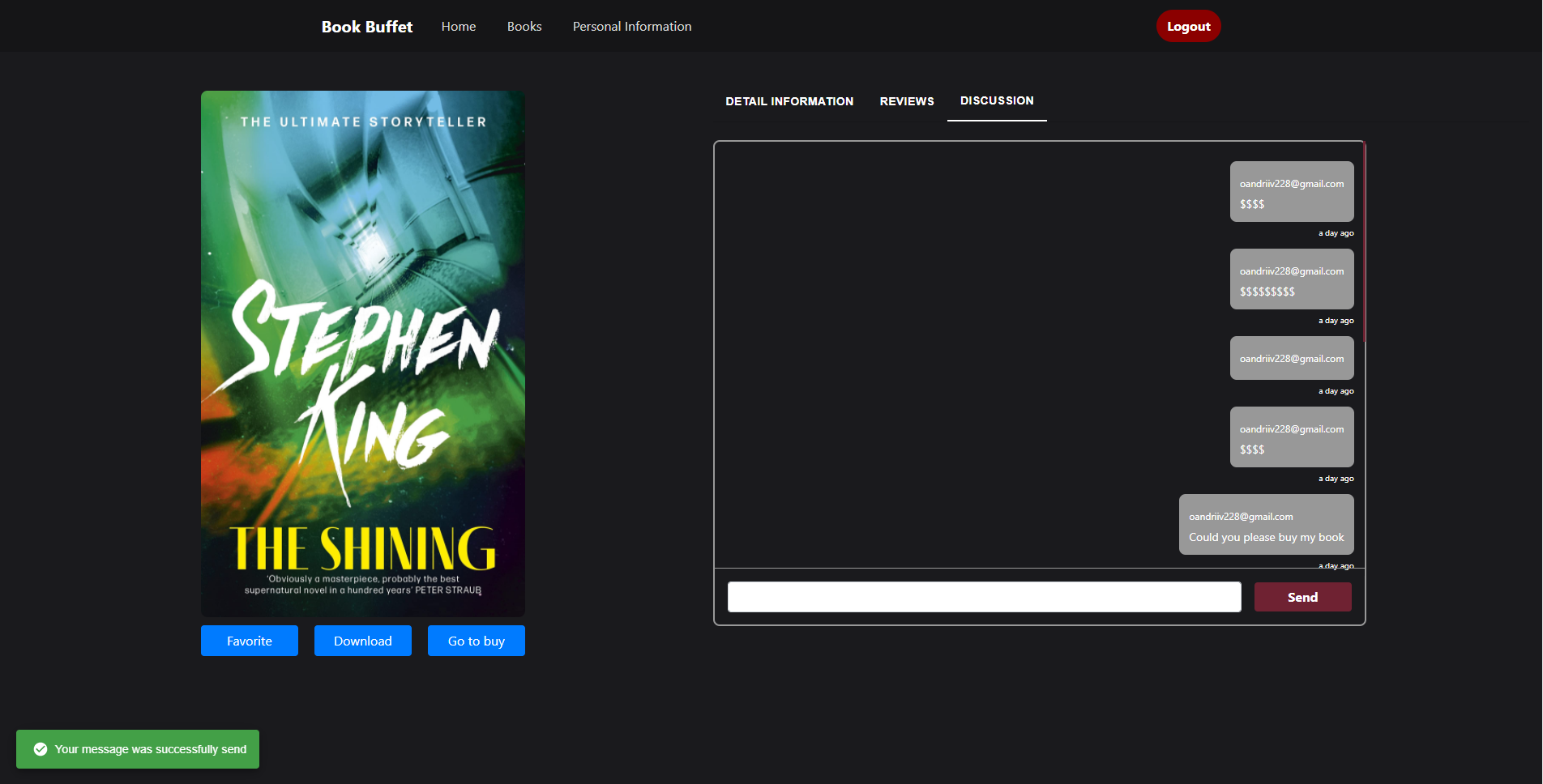


Рисунок Б.6 – Розгорнута сторінка чату

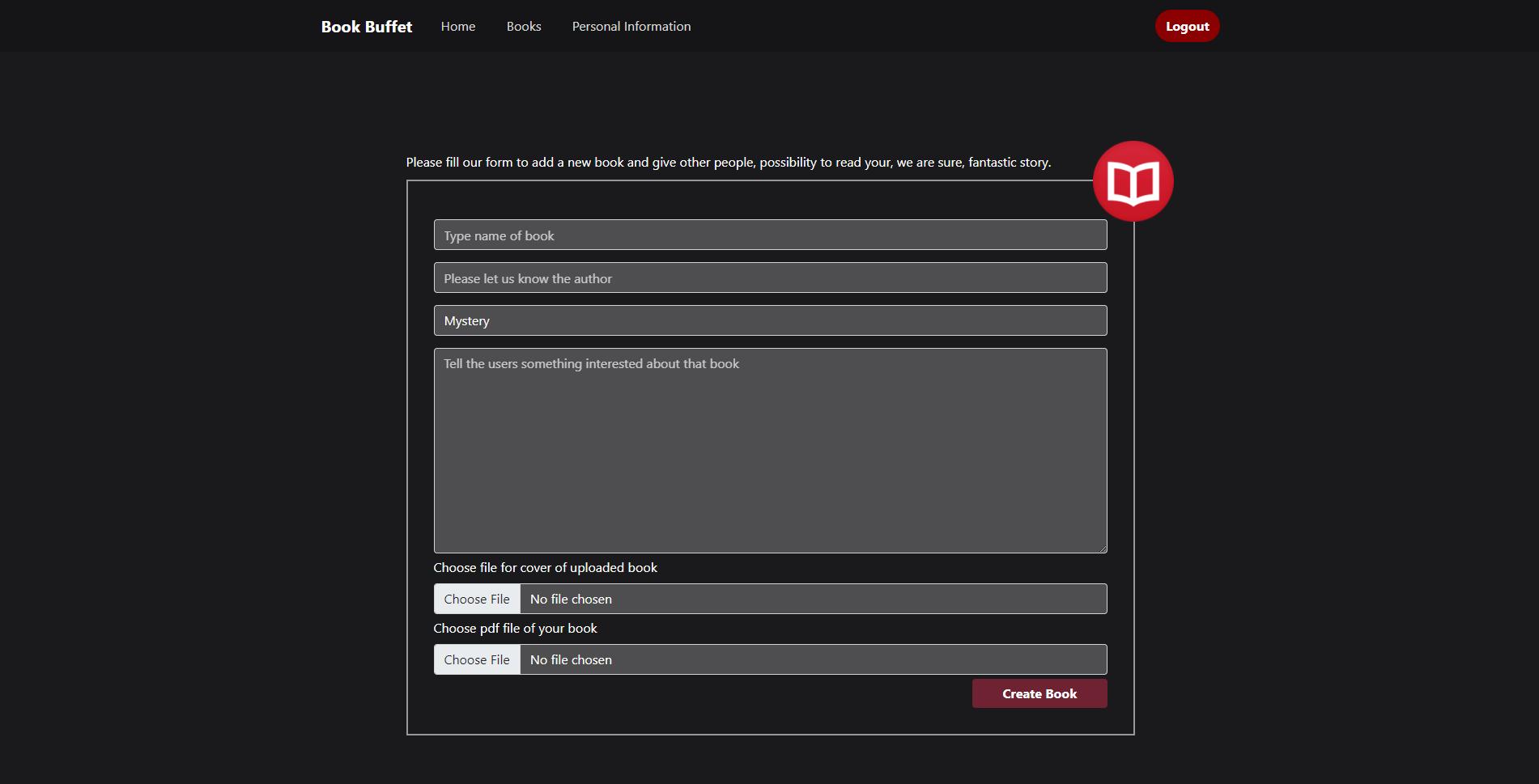


Рисунок Б.7 – Додавання нової книги

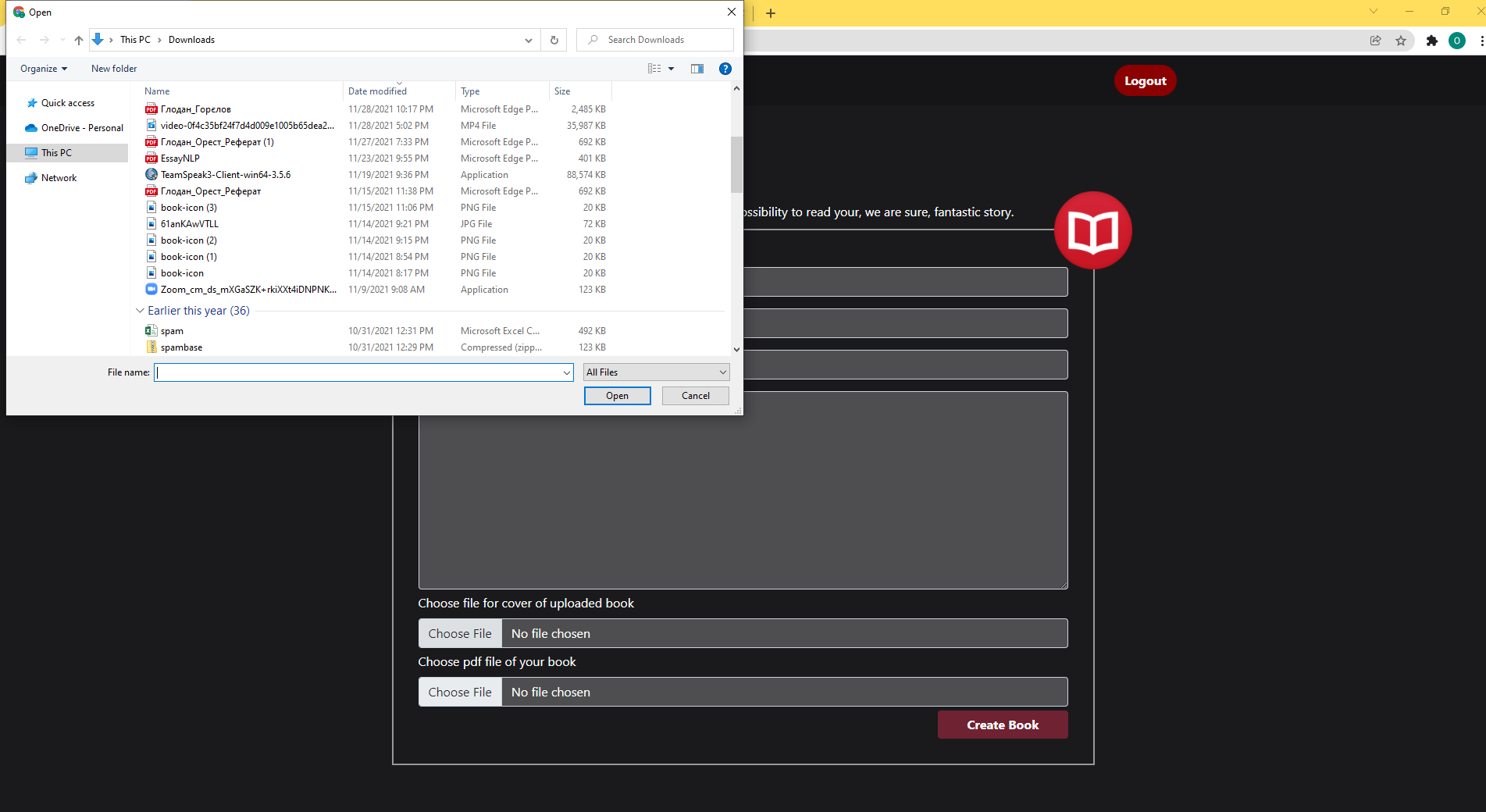


Рисунок Б.7 – Режим вибору книги