**АНОТАЦІЯ**

В даній магістерській кваліфікаційній роботі розроблено спеціалізовану програму аналізу груп індивідуумів у соціальній мережі. Система розроблена за допомогою технологій Java EE. Система проводить збір даних з соціальної мережі Facebook. Має можливість оповіщення користувача, та аналізу і візуалізації даних. Доступ до системи відбувається з будь-якого пристрою. Реалізовано прототип програмної системи у вигляді інтернет-сервісу аналізу соціальної мережі.

**ANNOTATION**

In this master work was developed specialized analysis program group individuals in the social network. The system was developed using technologies Java EE. The system collects data from the social network Facebook. Also has the ability to alert the user, and the analysis and data visualization. Access to the system is on any device. Implemented prototype software system in the form of online social network analysis service.

**ЗМІСТ**

[ВСТУП 8](#_Toc437177233)

[1. АНАЛІТИЧНИЙ ОГЛЯД 10](#_Toc437177234)

[1.1. Соціальні мережі та інтернет-сервіси 10](#_Toc437177235)

[1.1.1. Поняття соціальної мережі 10](#_Toc437177236)

[1.1.2. Методи аналізу соціальних мереж 12](#_Toc437177237)

[1.3. Використання теорії графів для аналізу соціальних мереж 15](#_Toc437177238)

[1.4. Розподілена обробка даних за допомогою Hadoop 17](#_Toc437177239)

[1.4.1. Архітектура та принципи роботи 18](#_Toc437177240)

[1.4.2. Розподілена файлова система HDFS 20](#_Toc437177241)

[1.4.3. MapReduce і обробка великомасштабних графів 23](#_Toc437177242)

[1.4.4. Алгоритми на графах в моделі MapReduce 24](#_Toc437177243)

[2.ВИБІР ІНСТРУМЕНТІВ ТА СОЦІАЛЬНОЇ МЕРЕЖІ 26](#_Toc437177244)

[2.1. Вибір мови програмування для реалізації проекту 26](#_Toc437177245)

[2.2. Вибір мови JavaScript для реалізації інтерфейсу користувача 28](#_Toc437177246)

[2.3. Використання JSON для обміну даними 29](#_Toc437177247)

[2.4. Вибір соціальної мережі 30](#_Toc437177248)

[2.5. Вибір інструменту для парсингу веб сторінок 31](#_Toc437177249)

[2.6. Вибір засобів для реалізації бази даних 33](#_Toc437177250)

[2.7. Вибір веб сервера 36](#_Toc437177251)

[3. Реалізація спеціалізованої програми аналізу груп індивідуумів в соціальній мережі 38](#_Toc437177252)

[3.1. Розробка алгоритму роботи спеціалізованої програми аналізу груп індивідуумів в соціальній мережі 38](#_Toc437177253)

[3.2. Реалізація серверної частини спеціалізованої програми аналізу груп індивідуумів в соціальній мережі 42](#_Toc437177254)

[3.2.1. Структура проекту 42](#_Toc437177255)

[3.2.2. Авторизація та реєстрація користувача в системі 43](#_Toc437177256)

[3.2.3. Збір даних з соціальної мережі 47](#_Toc437177257)

[3.3 Реалізація бази даних 51](#_Toc437177258)

[3.4. Реалізація клієнтської частини спеціалізованої програми аналізу груп індивідуумів в соціальній мережі 53](#_Toc437177259)

[4. Опис проведених досліджень 58](#_Toc437177260)

[4.1. Розробка MapReduce програми 58](#_Toc437177261)

[4.2. Підготовка даних 59](#_Toc437177262)

[4.3. Теорія шести рукопотискань 60](#_Toc437177263)

[4.4. Число Данбара 63](#_Toc437177264)

[РОЗДІЛ 5. ЕКОНОМІЧНА ОЦІНКА ПРОЕКТНОГО РІШЕННЯ 65](#_Toc437177265)

[5.1. Економічна характеристика проектного рішення 65](#_Toc437177266)

[5.2. Розрахунок витрат на розробку програмного забезпечення 66](#_Toc437177267)

[5.3. Вибір і обґрунтування аналога 69](#_Toc437177268)

[5.4. Визначення комплексного показника якості проектної розробки 71](#_Toc437177269)

[5.5. Визначення експлуатаційних витрат 73](#_Toc437177270)

[5.6. Розрахунок ціни споживання проектного рішення 75](#_Toc437177271)

[5.7. Визначення показників економічної ефективності 75](#_Toc437177272)

[5.8. Висновки 77](#_Toc437177273)

[ВИСНОВКИ 78](#_Toc437177274)

[Список використаних джерел 79](#_Toc437177275)

[ДОДАТКИ 80](#_Toc437177276)

[А. Лістинг програми 80](#_Toc437177277)

# **ВСТУП**

Соціальна мережа – це спільнота людей, об’єднаних однаковими інтересами, уподобаннями або тих, хто має інші причини для безпосереднього спілкування між собою.

У теорії соціальних мереж розглянуто соціальні взаємовідносини у вигляді зв’язків між вузлами. Вузли є відособленими акторами у мережах, а зв'язки відповідають стосункам між акторами. Існує велика кількість різних типів зв'язків між вузлами. У найпростішій формі соціальна мережа є відображенням усіх зв'язків, які мають відношення до дослідження, між вузлами. Мережі можуть використовуватися для встановлення соціального капіталу окремих акторів. Ці концепції часто відображаються на діаграмі соціальної мережі, на якій вузлам відповідають точки, а зв'язкам – лінії.

Останнім часом дослідження соціальних мереж набуло великої популярності. В публічному доступі з’явилися персональні дані людей (факти біографії, відео, фото, аудіо-матеріали, маршрути подорожей, коментарі та інші дані) та зв’язки між ними. Таким чином, соціальні мережі є унікальним джерелом особистих даних та інтересів реальних людей. Одним із способів вивчення соціальних мереж є соціальний граф, де основними елементами є взаємодія соціальних суб’єктів.

У математиці графами називають абстрактні структури даних, що описують структурні відносини між об'єктами. Сьогодні графи широко використовуються для моделюванні даних у різних предметних областях, потребують з'ясування та визначення правил і схем відносин об'єктів, а також можливих відхилень. Такими предметними областями є Web-графи, соціальні мережі, семантичний Web, бази знань, бібліографічні мережі і т. д. Постійно збільшуються обсяги графових даних цих додатків призводять до гострої необхідності використовувати масштабовані системи, що дозволяють ефективно обробляти величезні масиви даних.

Кожні два роки розмір «цифрового світу» подвоюється. Так за оцінками IDC розмір на 2006 р. становив 0,18 зетабайт, а на 2015 рік він збільшився в десятки раз і становить 7,9 зетабайт. За прогнозами в 2020 році «цифровий світ» буде містити 44 зетабайт.

# **АНАЛІТИЧНИЙ ОГЛЯД**

## **1.1. Соціальні мережі та інтернет-сервіси**

### **1.1.1. Поняття соціальної мережі**

Соціальна мережа являє собою структуру, що будується на зв’язках між групами людей, які мають схожі уподобання. В якості інтернет-сервісу соціальну мережу можна розглядати як майданчик, на якому може відбуватись зв’язок між особою та групою за своєрідними інтересами. Основна мета таких сайтів у тому, щоб забезпечити усіх користувачів ресурсами для комунікації між собою чи групою людей відео, зображення, музика, чати, блоги та інше.

Фактично соціальні мережі з'явилися з формуванням соціуму як такого, а от ідея про те, що взаємодія людей між собою, можна образно представити у вигляді мережі (у вузлах якої знаходяться індивідууми, а з’єднуючі їх лінії можуть інтерпретуватися як відображення взаємодії в групах), виникла значно пізніше, задовго до створення Інтернету. Ще в 1902 році відомий соціолог Ч. Кули писав: «Людину можна представити як точку перетину будь-якої кількості ліній, які позначають соціальні групи, при цьому число ліній відповідає числу груп, до яких належить даний індивід». [1]

Термін «соціальні мережі» був запроваджений в 1954 році соціологом з Манчестерской школи Джеймсом Барнсом в роботі «Класи і зборів в норвезькому острівному приході», що увійшла до збірки «Людські стосунки». У другій половині XX століття поняття «соціальна мережа» стало дуже популярним у колах західних дослідників, суспільстві і загальновживаним в англійській мові. З часом в якості вузлів соціальних мереж почали розглядати не тільки людей як агентів соціуму, а й будь-яких інших акторів, які можуть мати соціальні зв'язки, для прикладу фільми, книги, музика, міста, країни і т. д.[1]

У ході розвитку технологій в дослідженні соціальної мережі появились такі терміни, як аналіз соціальної мережі (Social Network Analysis, SNA), теорія соціальної мережі і т. д..

Побудова та аналіз соціальних мереж використовуються в різноманітних областях. Багато підприємців, наприклад, відзначають, що при побудові нового бізнесу велику роль відіграє саме їх соціальна мережа, що складається з родичів, друзів, сусідів, покупців, продавців, кредиторів і т. п. Аналітики стали вивчати різні аспекти розвитку бізнесу на базі використання SNA. Аналіз соціальних мереж надає цікаву інформацію при вивченні колективної поведінки. В основі різних громадських організацій і рухів, як правило, лежать певні соціальні зв'язки, причому для спільних дій у групі необхідний набір зв'язків критичної маси.

Коли з'явилися комп'ютери, їх почали застосовувати для відображення та кількісного аналізу соціальних мереж. Можна сказати, що весь веб-простір - це соціальна мережа, яка може бути описана величезним графом, вершини якого - це окремі сторінки, а гіперпосилання - дуги графа, що зв'язують вершини. Більш того, структура соціальної мережі (люди, їхні знайомства) і структура веб-мережі мають багато спільного саме за рахунок наявності і в тій, і в іншій мережі соціального компоненту. Обидві мережі вкрай неоднорідні. Дійсно, соціальна значимість людей різна, відповідно різна і потреба оточення у створенні контактів з конкретною особистістю. Певною мірою соціальна значимість людей визначається двома параметрами: їх знаннями та їх становищем (зв'язками). Аналогічно сайти можна розділити на дві категорії: «авторитети» (містять багато корисної інформації за деякою темою) і «концентратори» (мають безліч зв'язків - посилань на цікаві сайти). Як показали дослідження фахівців з IBM, AltaVista і Compaq, структура веб-простору неоднорідна: у ньому можна виділити так зване ядро, що складається з «концентраторів» і «авторитетів», і периферійні сайти, що володіють невеликою «вагою» в мережевому співтоваристві. Протягом життя сайт або набирає вагу і входить в ядро, або вмирає.

Багато інтернет-сервісів, що надають користувачам можливість комунікувати, автоматично формують соціальні мережі. Досягнувши певного етапу провайдери зрозуміли, що необхідно створити сервіс, головною метою якого буде накопичення соціального капіталу, тобто особистих ділових зв'язків у вигляді соціальної мережі. Таким чином був створений інтернет-сервіс у вигляді соціальної мережі. Подібні сервіси називаються також соціальними-послугами мережі (від анг. Social networking service), тобто послугами по підтримці соціальних кіл спілкування та мереж, які працюють за допомогою Інтернету. Підтримання соціальних мереж досягається шляхом автоматизації поширення оголошень по встановленим зв'язкам між людьми. Нерідко підтримка соціальної мережі включається в різні види послуг, де потрібна база даних облікових записів, які накопичують особисті дані про користувачів. Найперше це присутнє у послугах, які надають можливість особистого спілкування між користувачами. Умовно ресурси з побудови соціальних мереж можна розділити на співтовариства загального характеру і професійні бізнес-спільноти.

Таким чином, про соціальну мережі можна говорити в різних аспектах: як про соціальне явище (людям властиво встановлення соціальних зв'язків), як про універсальний інструмент соціологічного аналізу, заснованому на теорії графів, і нарешті, як про інтернет-послуги або інтернет-сервісі з побудови соціальній мережі у всесвітній павутині для отримання соціального капіталу.

### **1.1.2. Методи аналізу соціальних мереж**

Соціальну мережу, можна математично представити у вигляді графу, в якому вершини відображають користувачів мережі, а ребра - відносини.

Головною течією аналізу соціальних мереж є візуалізація соціальних графів. Важливе значення має візуалізація, оскільки сама можливість побачити мережу дозволяє зробити важливі висновки про характер взаємодії користувачів, не вдаючись до інших методів аналізу графа. Існує чотири підходи що до аналізу соціальних мереж:

1. Ресурсний – розглядає можливості акторів по залученню індивідуальних і мережевих ресурсів для досягнення певної мети і диференціює акторів, які знаходяться в ідентичних структурних позиціях соціальної мережі, за їх ресурсами. Як індивідуальні ресурси можуть виступати знання, престиж, багатство, етнічність, стать (гендерна ідентичність). Під мережевими ресурсами розуміються вплив, статус, інформація, капітал. [2]

2. Нормативний – вивчає рівень довіри між акторами, а також норми, правила та санкції, які впливають на поведінку акторів у соціальній мережі та процеси їх взаємодій. У цьому випадку аналізуються соціальні ролі, які пов'язані з даним ребром мережі, наприклад, відношення керівника і підлеглого, дружні або родинні зв'язки. Комбінація індивідуальних і мережевих ресурсів актора з нормами і правилами, що діють у даній соціальній мережі, утворює його «соціальний капітал». [2]

3. Структурний – акцентує увагу на геометричній формі та інтенсивності взаємодій (вазі ребер). Усі актори розглядаються як вершини графа, які впливають на конфігурацію ребер і інших акторів мережі. Особлива увага приділяється взаємному розташуванню вершин, центральності, транзитивності взаємодій. Для інтерпретації результатів у даному випадку використовуються структурні теорії і теорії мережевого обміну. [2]

4. Динамічний – увага акцентована на змінах у мережевій структурі з часом. Вивчаються причини зникнення і появи ребер мережі; зміни структури мережі при зовнішніх діях; стаціонарні конфігурації соціальної мережі. [2]

У соціальному аналізі застосовуються такі аналітичні методи:

1. методи знаходження локальних властивостей суб'єктів;
2. аналіз діад і тріад;
3. методи визначення еквівалентності акторів, включаючи їх структурну еквівалентність;
4. блокові моделі і ролева алгебра;
5. методи теорії графів;
6. кореспондентський аналіз і топологічні методи, що представляють мережу як деякий симпліціальний комплекс.
7. імовірнісні моделі;

Основна особливість оперативної соціальної мережі, що є джерелом соціальних даних є, що користувач розширює свою мережу досить швидко, що не вимагає додаткових витрат використовуючи соціальних аналітиків. Тому як результат створюються соціальні графи великих розмірів, за допомогою яких здійснюються різноманітні дослідження різних властивостей соціальної мережі.

Дослідниками вже визначені десятки ознак/показників, що характеризують структуру соціальних мереж: діаметр і середня відстань (малий діаметр соціальної мережі обумовлює короткий ланцюжок розповсюдження впливу по мережі), розподіл ступенів (статечне/без масштабне розподілення у соціальній мережі обумовлює наявність домінуючих вузлів (хабів), пов'язаних з сотнями, тисячами і навіть мільйонами інших вузлів, в більшості своїй мають всього по декілька зв'язків), показники пов'язаності і інші показники.

На основі певної множини ключових структурних ознак і характерних їх значень можна розробити моделі соціальних мереж, що володіють структурним схожістю з реальними мережами. Довгий час все складні мережі розглядалися як повністю випадкові, і мережі моделювалися з'єднанням випадкових вершин випадковим чином. Проте 1969 року Мілгрем (Milgram) виявив, що будь-яких двох людей «розділяє всього шість кроків». З'явилося безліч статичних моделей так званого «малого світу». Однак незважаючи на те, що статичні моделі дають можливість врахувати складність мережі, вони не пояснюють причину появи безмасштабних мереж, їх природу. Найбільш поширеною динамічної моделлю появи таких структур є модель «багатий стає багатшими», в якій в зростаючому графі ймовірність появи нових зв'язків у вершини пропорційна її ступеня, тобто кількості вже наявних зв'язків. Існують і інші підходи до пояснення причин появи структури соціальній мережі. Наприклад, в теоретико-ігрових моделях вартість додавання нових зв'язків впливає на механізм переважного приєднання нових зв'язків (тобто для агентів переважні ті чи інші конфігурації зв'язків).

У даний час створено доволі багато систем аналізу соціальних мереж, ці системи найчастіше використовуються соціологами для їхніх досліджень: Negopy, UCINET, Pajek, Netminer, Visone, NetDraw, SNA/R, StOCNET, InFlow, GUESS, NetworkX, JUNG, BGL/Python, prefuse та інші.

## **1.3. Використання теорії графів для аналізу соціальних мереж**

Соціальну мережу можна відобразити ​​у вигляді неорієнтованого соціального графу Z (X, Y), що містить безліч X користувачів і безліч V ребер, представляють соціальні зв'язку, такі як дружба, споріднення, довіри тощо. Теорія графів містить дуже багато корисного та цікавого для таких областях, як мережі та соціологія.

Граф Z =(X, Y ) можна зобразити за допомогою рисунка на площині, який називають діаграмою графа Z. Вершини графа Z ставляться у об’єктивну відповідність точки площини; точки, що відповідають вершинам v i w, з’єднуються лінією (відрізком або кривою) тоді і тільки тоді, коли v i w суміжні вершини. Зрозуміло, що діаграма графа змінюватиме свій вигляд у залежності від вибору відповідних точок на площині. На Рис. 1.1 зображені діаграми графів Z1 i Z2 .[5]

Занумеруємо всі вершини графа Z натуральними числами від 1 до n. Матрицею суміжності A графа Z називається квадратна n\*n - матриця, в якій елемент aij i-го рядка і j-го стовпчика дорівнює 1, якщо вершини vi та vj з номерами i та j суміжні, і дорівнює 0 у противному разі. [5]

Графи можна задавати також за допомогою матриць.

Z1

Z2

Рис. 1.1. Граф

Для графів Z1 i Z2 маємо відповідно

A1= i A2=

Очевидно, що матриці суміжності графів - симетричні.

Занумеруємо всі вершини графа Z числами від 1 до n і всі його ребра - числами від 1 до m. Матрицею інцидентності B графа Z називається n\*m-матриця, в якій елемент bij i-го рядка і j-го стовпчика дорівнює 1, якщо вершина vi з номером i інцидентна ребру ej з номером j, і дорівнює 0 у противному разі.[5]

Для графів Z1 і Z2 маємо (ребра графів нумеруємо в тому порядку, в якому вони виписані)

B1= і B2=

Ще одним способом завдання графів є списки суміжності. Кожній вершині графа відповідає свій список. У список, що відповідає вершині v, послідовно записуються всі суміжні їй вершини. Для графів Z1 і Z2 маємо списки :

|  |  |
| --- | --- |
| Z1 | Z2 |
| v1: v3,v4  v2: v3,v4  v3: v1,v2,v4  v4: v1,v2,v3 | v1: v2,v4,v5  v2: v1,v3,v4  v3: v2,v5  v4: v1,v2,v5  v5: v1,v3,v4 |

Обрання одного з способів завдання графа залежить від поставленої задачі яку потрібно розв’язати.

## **1.4. Розподілена обробка даних за допомогою Hadoop**

Обробка великих масивів даних в фреймворку Apach Hadoop реалізується за допомогою технології MapReduce. Технологія MapReduce була реалізована в Google в 2004 році.

Розробники Google Джеффрі Дін і Санджай Гемават в 2004 році опублікували статтю, в якій запропонували наступне рішення для обробки великих обсягів даних (індексованих документів, логів запитів і т.п.): величезний масив інформації ділиться на частини, і обробка кожної з цих частин доручається окремому сервера. Як правило, дані обробляються на тих же серверах, де вони й зберігаються, що дозволяє прискорити процес обробки і уникнути зайвих переміщень даних між серверами. Після цього отримані результати об'єднуються в єдине ціле.[6]

Фахівці Google у згаданій вище статті обмежилися лише описом основних алгоритмів, не зупиняючись на подробицях реалізації. Однак цієї інформації виявилося для розробників Hadoop цілком достатньо, щоб створити власний фреймворк MapReduce. [6]

MapReduce використовується в таких проектах як - Facebook, Yahoo, Last.Fm, Google та інших. IDC говорить, що переважна більшість користувачів об'єднує Hadoop з іншими базами даних для виконання аналізу великих даних. Майже 39% респондентів кажуть, що вони використовують бази даних NoSQL з HBase, Cassandra і MongoDB, і майже 36% говорять, що вони використовують Greenplum і Vertica в поєднанні з Hadoop. Gartner недавно провела опитування серед користувачів Hadoop, з'ясувати, що :

* 53% працюють з інтерактивним SQL
* 18% працюють з системами управління базами даних
* 14% опрацьовують потокові дані
* 9% використовуються для пошуку
* 6% знаходять використання в графах

### **1.4.1. Архітектура та принципи роботи**

Завдяки унікальній архітектурі масштабується фізичного кластеру та елегантною інфраструктурі обробки, спочатку розробленої Google, Hadoop сприяла вибухового зростання в новій галузі обробки великих даних. Hadoop також створила багату і різноманітну екосистему додатків, включаючи Apache Pig (потужна мова сценаріїв) і Apache Hive (сховище даних з SQL-подібним інтерфейсом).

На жаль, ця екосистема побудована на парадигмі програмування, яка не здатна вирішити всі проблеми великих даних. Хоча інструменти Pig і Hive спрощують специфічну модель програмування, яку надає MapReduce, вона не є панацеєю для великих даних.

Щоб поліпшити спільне використання, масштабованість і надійність кластера Hadoop, був обраний ієрархічний підхід до інфраструктури кластеру. Зокрема, вузькоспеціалізована функціональність MapReduce була замінена новим набором демонів, який відкриває інфраструктуру для нових моделей обробки.

Використанням JobTracker і TaskTracker був головним недоліком MRv1 через обмеження масштабування і наявності режимів відмов, викликаних мережевими витратами. Ці демони були також характерні для моделі обробки MapReduce. З метою ліквідувати цю залежність JobTracker і TaskTracker були видалені з YARN і замінені новим набором демонів, що не прив'язаних до додатків (Рис. 2.2).

Коренем ієрархії YARN є ResourceManager. Цей демон керує всім кластером і призначенням додатків базовим обчислювальним ресурсів. ResourceManager розподіляє ресурси (обчислювальні ресурси, пам'ять, пропускна здатність і т.д.) для базових NodeManager (агент YARN на вузлі). Крім того, ResourceManager взаємодіє з ApplicationMaster при розподілі ресурсів і з NodeManager при запуску та моніторингу базових додатків. У цьому контексті ApplicationMaster виконує деякі функції свого попередника TaskTracker, а ResourceManager виконує роль JobTracker.[4]

Кожним екземпляром програми, яке виконується в YARN, управляє демон ApplicationMaster. Він відповідає за отримання ресурсів із ResourceManager і моніторинг (за допомогою NodeManager) виконання і споживання ресурсів контейнерів (розподіл ресурсів процесора, пам'яті і т.д.). Хоча в даний час ресурси є традиційними (ядра процесора, пам'ять), завтра можуть з'явитися нові ресурси, специфічні для конкретної задачі (наприклад, графічні процесори або спеціалізовані пристрої обробки). Демони ApplicationMaster з точки зору YARN є користувальницькою кодом, отже, становлять потенційну загрозу безпеці. YARN припускає, що ApplicationMaster містять безліч помилок або навіть шкідливого коду і відповідно ставиться до них як непривілейованих коду.

ResourceManager

NameNode

NodeManager

DataNode

ApplicationMaster

Клієнт

Клієнт

NodeManager

DataNode

ApplicationMaster

NodeManager

DataNode

Контейнер

NodeManager

DataNode

Контейнер

Контейнер

NodeManager

DataNode

Контейнер

Головний вузол

Підпорядковані

вузли

Рис. 2.2. Проста ілюстрація архітектури кластера Hadoop

NodeManager управляє всіма вузлами в кластері YARN. NodeManager надає сервіси на кожному вузлі кластера, починаючи з контролю управління контейнером протягом його життєвого циклу і закінчуючи моніторингом ресурсів і відстеженням стану вузла. На відміну від MRv1, керувала завданнями map і reduce через слоти, NodeManager управляє абстрактними контейнерами, які являють собою ресурси на вузлі, доступні для конкретного додатка. YARN раніше використовує шар HDFS з головними вузлами NameNode для сервісів метаданих і DataNode для сервісів зберігання, реплікованих по всьому кластеру.

Використання кластера YARN починається з клієнтського запиту, в якому вказується додаток. ResourceManager виділяє необхідні ресурси для контейнера і запускає ApplicationMaster для обслуговування зазначеного додатка. Використовуючи протокол запиту ресурсів, ApplicationMaster виділяє контейнери ресурсів для програми на кожному вузлі. ApplicationMaster контролює контейнер до завершення програми. Після завершення програми ApplicationMaster скасовує виділення контейнера в ResourceManager і цикл завершується.

З усього вищесказаного стає ясно, що колишню архітектуру Hadoop сильно обмежував JobTracker, що відповідав за управління ресурсами і планування завдань в кластері. Нова архітектура YARN відмовляється від цієї моделі; тепер за управління використанням ресурсів додатками відповідає ResourceManager, а за управління виконанням завдань - ApplicationMasters. Ця зміна усуває вузьке місце, а також підвищує можливість масштабування кластерів Hadoop до набагато більших змін, ніж раніше. Крім того, крім традиційної MapReduce, YARN дозволяє одночасно застосовувати різні моделі програмування, в тому числі обробку графів, ітераційну обробку, машинне навчання і кластерні обчислення, використовуючи стандартні схеми передачі інформації, такі як Message Passing Interface.

### **1.4.2. Розподілена файлова система HDFS**

Hadoop Distributed File System (HDFS) є розподілена файлова система призначена для роботи з програмним і апаратним забезпеченням. Вона є дуже похожа на існуючі розподілені файлові системи. Тим не менш, відмінності від інших розподілених файлових систем є суттєвими. HDFS є дуже відмовостійкою і призначений для розгортання на дешевому обладнанні. HDFS забезпечує високу пропускну здатність доступу до даних програми і підходить для програм, які мають великі набори даних. HDFS спрощує кілька вимог POSIX, щоб дозволити потоковий доступ до файлу даних системи. HDFS була спочатку побудована як інфраструктура для веб-проекту пошуку двигуна Apache Nutch. HDFS є частиною проекту Apache Hadoop.

Клієнт

NameNode

DataNode

DataNode

DataNode

DataNode

Метадані

Мережа на основі

TCP/IP

Рис. 2.3. Розподілена файлова система

Основний NameNode-сервер фіксує всі транзакції, пов'язані зі зміною метаданих файлової системи, в log-файлі, званому EditLog. При запуску основного NameNode-сервера, він зчитує образ HDFS (розташований у файлі FsImage) і застосовує до нього всі зміни, накопичені в EditLog. Потім записується новий образ вже із застосованими змінами і система починає роботу вже з чистим log-файлом. Слід зауважити, що дану роботу NameNode-сервер виконує один раз при його першому запуску. У подальшому, подібні операції покладаються на вторинний NameNode-сервер. FsImage і EditLog зрештою зберігаються на основному сервері.

Ослаблена модель цілісності даних, реалізована в файлової системі, не гарантує ідентичність реплік. Тому HDFS перекладає перевірку цілісності даних на клієнтів. При створенні файлу клієнт розраховує контрольні суми кожні 512 байт, які в подальшому зберігаються на DataNode-сервері. При зчитуванні файлу, клієнт звертається до даних і контрольних сумах. І у разі їх невідповідності відбувається звернення до іншої репліці.

При запису даних у HDFS використовується підхід, що дозволяє досягти високої пропускної здатності. Додаток веде запис в потоковому режимі, при цьому HDFS-клієнт кешує записувані дані в тимчасовому локальному файлі. Коли у файлі накопичуються дані на один HDFS-блок, клієнт звертається до NameNode-серверу, який реєструє новий файл, виділяє блок і повертає клієнтові список datanode-серверів для зберігання реплік блоку. Клієнт починає передачу даних блоку з тимчасового файлу перший DataNode-серверу зі списку. DataNode-сервер зберігає дані на диску і пересилає наступного DataNode-серверу у списку. Таким чином, дані передаються в конвеєрному режимі і реплікуются на необхідній кількості серверів. Після закінчення запису, клієнт повідомляє NameNode-сервер, який фіксує транзакцію створення файлу, після чого він стає доступним в системі .

В силу забезпечення збереження даних (на випадок відкату операції), видалення в файлової системі відбувається за певною методикою. Спочатку файл переміщується в спеціально відведену для цього/trash директорію, а вже після закінчення певного часу, відбувається його фізичне видалення

HDFS була спеціально розроблена для підтримки потокового доступу до файлів великого об'єму, які записуються однократно. Коли клієнтові необхідно записати файл в системі HDFS, цей процес починається з кешування файлу в тимчасове локальне сховище клієнта. Коли об'єм Кешована даних перевищує заданий розмір блоку HDFS, вузлу NameNode надсилається запит на створення файлу. Вузол NameNode відповідає клієнту, посилаючи у відповідь ідентифікатор вузла DataNode і блок призначення. Також повідомляються ті вузли DataNode, які будуть містити репліки цього файлового блоку. Коли клієнт починає передачу свого тимчасового файлу першому вузлу DataNode, вміст файлового блоку негайно ретранслюється вузлам репліки з використанням конвеєрної технології. Крім того, клієнти відповідають за створення файлів, що містять контрольні суми, які також зберігаються в тому ж самому просторі імен HDFS. Після передачі останнього файлового блоку вузол NameNode фіксує факт створення файлу у своєму постійному сховищі метаданих (в файлах EditLog і FsImage).

### **1.4.3. MapReduce і обробка великомасштабних графів**

У моделі програмування MapReduce функція Map приймає пари ключ/ значення і формує з них набір проміжних пар ключ/значення. Всі проміжні значення, пов'язані з одним і тим же проміжним ключем, групуються і передаються на обробку функції Reduce. Функція Reduce отримує проміжні ключі з наборами відповідних їм значень і об'єднує їх. [4]

Map

Map

Map

Map

Reduce

Reduce

Reduce

Reduce

Вхідні дані

Результат

Переміщення та сортування

Рис. 2.4. Архітектура MapReduce системи

На рівні реалізації проміжні пари ключ/значення буферезує в пам'яті. Періодично буферезовані дані скидаються на локальний диск і розбиваються на області відповідною функцією. Відомості про місце розташування цих буферизованих пар на локальному диску передаються спеціальному головному екземпляру диктофону, що відповідає за передачу цих відомостей reduce-обробникам. Коли reduce-обробник отримує дані про місцезнаходження, він зчитує буферизовані дані з локальних дисків map-обробників. Після цього буферизовані дані сортуються за проміжними ключам таким чином, щоб всі однакові ключі виявилися згрупованими. Reduce-обробник передає ключ і відповідний набір проміжних значень на вхід користувача функції Reduce, результат роботи якої додається в підсумковий результуючий файл для цього розділу reduce.

MapReduce дозволяє розробникам не піклуватися про зайвих подробицях роботи розподілених додатків, наприклад, про розподіл і плануванні даних або про забезпечення відмовостійкості. Базова модель MapReduce не підходить для задач обробки графів, оскільки більшість графових алгоритмів є ітеративними і в них так чи інакше використовуються переходи по графу. Таким чином, ефективність графових обчислень сильно залежить від пропускної здатності між вузлами-обробниками, оскільки структури графів передаються по мережі після кожної ітерації. Наприклад, базова модель MapReduce не має прямої підтримки ітеративних додатків для аналізу даних. Для реалізації такої підтримки від розробника може знадобитися вручну запустити кілька завдань MapReduce і управляти ними за допомогою програми-драйвера. На практиці при ручному управлінні ітеративними додатками в MapReduce виникають дві ключові проблеми:

* Навіть якщо більша частина даних не змінюється від ітерації до ітерації, всі дані доводиться повторно завантажувати і обробляти на кожній ітерації, що призводить до витрат на виконання операцій введення/ виводу, а також до завантаження мережі і ресурсів процесора.
* Умова завершення може включати в себе перевірку досягнення контрольної точки. Ця умова може припускати виконання додаткової задачі MapReduce в кожній ітерації, що знову ж збільшує завантаження ресурсів (виконання додаткових завдань, читання додаткових даних з диска і їх передача по мережі).

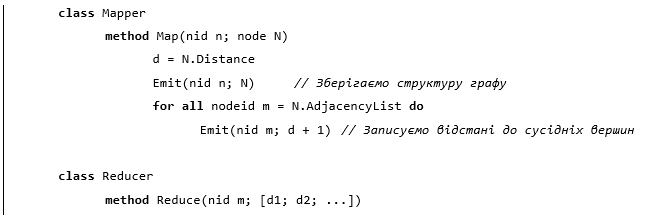
### **1.4.4. Алгоритми на графах в моделі MapReduce**

Для пошуку найкоротшого шляху від однієї вершини графу до всіх, використано модифікації алгоритму Дейкстри та пошуку в ширину.

Алгоритм Дейкстри полягає в покроковому переборі всіх вершин графа G та присвоєнні мітки, яка є мінімальною відстанню від початкової вершини. Для початку присвоїмо початковій вершині – 0, та всім іншим відстань рівну - ∞. Перебираємо всі вершини графа, вибираючи вершину u, з найменшою відстанню. Для кожної сусідньої вершини u, крім вже відвіданих, вибираємо найменше між значенням мітки даної вершини та значенням нової довжини шляху, що становить відстань вершини u плюс довжина ребра. [4]

Основою алгоритму Дейкстри є пріоритетна глобальна черга, з відсортованим списком вузлів. Це неможливо в MapReduce, так як модель програмування не передбачає механізму обміну глобальними даними. Замість цього використано пошук в ширину, який досить легко масштабується.

Алгоритм працює шляхом зіставлення всіх верши графа з їх сусідніми вершинами, створюючи пару ключ / значення, де ключем являється сусідня вершина, а значенням є відстань та поточна вершина графа. Відстань до сусіднього вузла визначається сумою відстані від початкового вузла до поточного та довжина ребра до сусідньої вершини. На вході Reduce отримає значення всіх ключів, що є вершинами графа та список шляхів, які ведуть до відповідного вузла. Для кожної вершини вибираємо найменше значення шляху та обновляємо структуру графа.



Даний алгоритм є циклічним, де кожна ітерація відповідає новому MapReduce завданню. Від ітерації до ітерації ми повинні зберігати структуру графа. Це досягається передачею вхідних даних Map на вихід разом з результатами обчислення відстаней до сусідніх вершин. В Reduce ми повинні розрізняти структуру графа від списку значень відстаней. Знайдену мінімальну відстань оновити в структурі графа та подати на вихід Reduce. Результат виконання поточної ітерації слугуватиме вхідними даними для наступних ітерації. Кількість ітерацій для опрацювання всіх вершин дорівнює діаметру графа. У нашому випадку кількість ітерацій ≤ 6.

# **2**.**ВИБІР ІНСТРУМЕНТІВ ТА СОЦІАЛЬНОЇ МЕРЕЖІ**

## **Вибір мови програмування для реалізації проекту**

Для розробки програми було обрано, як основну мову програмування Java оскільки вона найкраще підходить для виконання проекту. Java – мова, що інтерпретується. Компілятор Java генерує байт-код який виконується на віртуальній машині Java (JVM – Java Virtual Machine), а не звичайний машинний код для процесора і системи, на якій виконується програма. Для того щоб почати виконання програми на Java, потрібно використати віртуальну машину Java, що буде виконувати машинні інструкції створеного байт-коду. Байт-код Java не залежить від конкретної архітектури і тому програми на Java мають змогу працювати на будь-яких машинах, де була встановлена JVM.

Java є динамічною мовою. Будь-який клас Java може бути завантажений загружчиком класів під час робити програми. Бібліотеки які є внутрішніми також завантажуються динамічно.

Мова Java є розподіленою мовою. З цього випливає, що Java забезпечує досить широку підтримку роботи в інтернет мережі на доволі високому рівні. Клас який реалізований в Java, URL разом з потрібними класами пакету java.net надає можливість читати інформацію з файлу, який знаходиться в мережі так само, як з локального.

В Java реалізований автоматичний збирач сміття(garbage collector) який призначений керувати пам'яттю під час життєвого циклу об'єкта. Програмісту не потрібно дбати про час життя об’єкта, йому потрібно тільки його створити, а віртуальна машина сама видалить його з пам’яті коли об’єкт стане не потрібним. В той час, коли на об’єкт не посилається жодний з живих об’єктів, збирачу сміття повідомляється що він може забрати цей об’єкт з пам’яті. Однак, втрата пам'яті все ж може відбутися, коли код створений програмістом, містить посилання на об’єкти які вже не потрібні, якщо об’єкт використовується у контейнерах даних.

Java має реалізовану систему винятків чи ситуацій, під час роботи програми виникають випадкові помилки, наприклад:

* ділення на нуль
* невірний тип даних
* закінчення пам’яті
* незнайдений файл

Однією з властивостей концепції JVM полягає в тому, що виключення які відбулись під час роботи програми не зупиняють виконання програми. Є цілий ряд інструментів, які приєднуються до середовища під час роботи та кожного разу, коли відбулась певна помилка, зберігають інформацію в пам’ять для відновлення роботи програми. Такі засоби [автоматичного опрацювання](https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B0_%D0%BE%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D0%B8%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D1%8C&action=edit&redlink=1) помилок надають усю інформацію про помилки які відбулись під час роботи програми.

В Java присутня [строга](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%97) типізація даних, кожний клас та параметр має свій [тип](https://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), який етапі [компіляції](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) є відомий.

Сучасна віртуальна машина Java не аби як покращилась від початку її випуску, зросла швидкість її роботи вона стала дуже оптимізованою. Проводились випробування, де віртуальна машина Java порівнювалась зі звичайними компіляторами, результатом випробувань було, що швидкість виконання програми на віртуальній машині нічим не відрізняється від виконання на звичайному компілятору.

Java Platform, Enterprise Edition (Java EE) є стандартом в секторі корпоративного програмного забезпечення. Java EE створена з використанням Java Community Process, за участю галузевих експертів, комерційних і відкритих джерел, організацій користувачів Java і величезного числа людей.

Сьогодні, Java EE пропонує великий вибір корпоративного програмного забезпечення з більш ніж 20 сумісних з Java EE технологій, на сьогоднішній день існує 7 реалізацій Java EE.

Платформа Java EE призначена, щоб допомогти розробникам створювати дуже великі, мережеві додатки багаторівневі, масштабовані, надійні, безпечні і т.д. Назва таких додатків "корпоративні додатки", так називаються тому, що ці програми призначені для вирішення проблем, що виникають великих підприємств. Корпоративні програми корисні для великих корпорацій, установ і урядів, бізнесу. Переваги корпоративного програмного забезпечення є корисними у всіх галузях.

Платформа Java EE зменшує складність розробки корпоративних додатків, забезпечуючи середню модель розвитку, API і виконання, які дозволяють розробникам сконцентруватися на функціональності.

## **2.2. Вибір мови JavaScript для реалізації інтерфейсу користувача**

Для розробки клієнтської частини програми було обрано мову програмування JavaScript. Мова JavaScript є динамічною, об'єктно-орієнтованою мовою програмування. Що є реалізацією стандарту ECMAScript. Мова javaScript виконується на стороні клієнта в браузері, це дає можливість коду взаємодіяти з усіма ресурсами клієнта, асинхронно обмінюватися даними з серверною частиною, керувати браузером, змінювати структуру, є засоби для валідації та відображення вмісту форми, відкривати нові вікна і наповнювати динамічного вміст сторінки, змінювати зображення. Можливість використовувати JavaScript з CSS, щоб зробити DHTML (Dynamic Hypertext Markup Language). Сценарії JavaScript виконуються тільки на сторінках, які знаходяться у вікні браузера. Коли користувач припиняє переглядати цю сторінку, будь-які сценарії, які були запущені на ньому відразу ж зупиняються. Виняток становлять лише Cookies або різні інтерфейси на стороні клієнта.

JavaScript не має нічого спільного з Java, JavaScript спочатку назвали LiveWire, а потім LiveScript, коли він був створений Netscape.

В JavaScript є майже всі властивості [об'єктно-орієнтованої мови](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), але в ній реалізована концепція прототипів і тому підтримка об'єктів в ній є іншою ніж в традиційних мовах [ООП](https://uk.wikipedia.org/wiki/%D0%9E%D0%9E%D0%9F). Крім цього, в JavaScript є декілька особливостей, які присутні в функціональних мовах,  об'єкти як списки, функції як [об'єкти першого класу](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D1%88%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%83), [анонімні функції](https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%BD%D0%BE%D0%BD%D1%96%D0%BC%D0%BD%D1%96_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97&action=edit&redlink=1), [замикання](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D0%B8%D0%BA%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) (closures), каррінг ([currying](https://uk.wikipedia.org/w/index.php?title=Currying&action=edit&redlink=1)) це робить мову більш гнучкою.

Як і більшість мов JavaScript реалізує C подібний синтаксис, але для порівняння з мовою [С](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) має такі радикальні відмінності:

* функції які є об'єктами першого класу
* анонімні функції
* [опрацювання винятків](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D0%B8%D0%BD%D1%8F%D1%82%D0%BA%D1%96%D0%B2)
* приведення типів
* автоматичне управління пам’яттю
* об'єкти, які мають можливість динамічно змінювати тип завдяки механізму [прототипів](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

В JavaScript охоплює доволі багато вбудованих об'єктів: Object, Error, Array, String, Global, Boolean, Number, RegExp, Math, Date, Function. Крім цього, JavaScript включає багато вбудованих операцій, які не завжди є функціями чи методами, також javaScript містить набір вбудованих операторів, які керують логікою роботи програм. Синтаксис мови JavaScript є дуже подібним до синтаксису мови Java (який наслідується від C), але набагато простіший за Java, що полегшує умови для вивчення мови.

JavaScript, сьогодні є однією з найпопулярніших мов програмування. Технологія AJAX, механізми шаблонізації на клієнті, розвиток стандарту HTML5 з новими API для JavaScript, поява node.js як серверної платформи на JavaScript- все це сприяє просуванню JavaScript в індустрії розробки програмного забезпечення. Незважаючи на існування безлічі мов, які компілюються в JavaScript, більшість розробників все ж вважають за краще використовувати сам JavaScript через простоту в налагодженні і зовнішньої схожості з вже відомими їм мовами програмування.

## **2.3. Використання JSON для обміну даними**

Формат JSON (JavaScript Object Notation) є простим форматом обміну даними. Легкий для сприйняття людиною, щоб читати і писати та зручний для машин для розбору і генерації. Він заснований як підмножина мови програмування JavaScript, є стандартом ECMA-262 3rd Edition. Текстовий формат JSON створений для заміни громіздкому XML для асинхронної передачі впорядкованої інформації між сервером та клієнтом. JSON має декілька переваг перед XML, він займає набагато менше місця, допускає складні конструкції в атрибутах і просто інтегрується за допомогою JavaScript в об'єкти.

JSON створюється на двох структурах:

* Впорядкований список значень. У більшості мов це втілено як вектор, масив, послідовність або список.
* Колекція пара ім'я/значення. У різних мовах, це виконано як об'єкт, хеш-таблиця, словник або асоціативний масив.

Формат JSON є універсальною структурою даних. На сьогоднішній день майже всі мови програмування підтримують його у тій чи іншій формі. Це має сенс, що формат даних, які є взаємозамінними з мовами програмування також грунтується на цих структурах.

## **2.4. Вибір соціальної мережі**

На сьогоднішній день є достатньо різнотипних соціальних мереж, кожна з яких має окремий набір можливостей, ось короткий перелік найбільш популярних соціальних мереж:

1.**Facebook** є найпопулярнішою соціальною мережею на даний час, що дозволяє зареєстрованим користувачам створювати профілі, завантажувати фотографії та відео, відправляти повідомлення і залишатися на зв'язку з друзями, родиною та колегами. За статистичними даними Nielsen Group, інтернет-користувачі в межах Сполучених Штатів витрачають більше часу на Facebook, ніж будь-який інший веб-сайт.

2.**Twitter** cоціальна мережа, яка є мережею мікроблогів, що дозволяє користувачам надсилати короткі текстові повідомлення, так звані твіти. Користувач Twitter може транслювати твіти та переглядати твітів інших користувачів за допомогою багатьох платформ.

3.**Google+** є проектом соціальної мережі від Google, призначений для більш тісної взаємодії ​​ніж це має місце в інших соціальних мереж .

4.**LinkedIn** є сайт соціальної мережі, розроблений спеціально для бізнес-спільноти. Мета сайту є надання зареєстрованим користувачам створювати і підтримувати список ділових контактів.

5.**Вконтакте** соціальна мережа популярна здебільшого серед користувачів із [Росії](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D1%81%D1%96%D1%8F), [Білорусі](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BB%D0%BE%D1%80%D1%83%D1%81%D1%8C), [Казахстану](https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%B7%D0%B0%D1%85%D1%81%D1%82%D0%B0%D0%BD), [України](https://uk.wikipedia.org/wiki/%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B0) та [Узбекистану](https://uk.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B1%D0%B5%D0%BA%D0%B8%D1%81%D1%82%D0%B0%D0%BD). Дана мережа є копією мережі Facebook та містить деякі додаткові функції. Оскільки соціальна мережа контролюється російською ФСБ від її використання краще відмовитись.

6.**WeUA** одна з українськихсоціальних мереж , що була створена у 2014 році. Ідея для створення української соціальної мережі для заміни зарубіжних появилась доволі давно, перші намагання створити вітчизняну соціальну мережу здійснювались ще в 2008-2009 роках. Саме тоді українським користувачам Інтернету запропонували полишити Вконтакті, Однокласники  та Facebook і перейти у свої соціальні мережі ВРеале, Українці та Друзі.

7. [**Друзі**](http://druzi.org.ua/) також є українською соціальною мережею, яка має доволі високу аудиторію відвідувачів. Проект був розпочатий в 2009 році й на даний час його аудиторія складає приблизно 400 тисяч користувачів.

Отже для реалізації проекту було вибрану соціальну мережу Facebook, оскільки це найпопулярніша соціальна мережа у світі яка містить 1,5 мільярда користувачів на 2015 рік  з яких 968 млн. відвідує свої акаунти щодня. Також Facebook мітить дуже зручний і зрозумілий набір API.

## **2.5. Вибір інструменту для парсингу веб сторінок**

Парсинг (анг. Parse - розбір) називають процес аналізу та розбору деякого контенту на складові елементи за допомогою парсерів. Найбільш відомими парсерами в інтернет мережі є пошукові роботи, що аналізують сторінки, зберігають отримані дані у себе в базі та пізніше при пошуку видають потрібні дані. Парсинг сайтів це - послідовний синтаксичний аналіз інформації, розміщеної на інтернет-сторінках. Web-парсинг використовується для автоматизованого збору даних або контенту з якого-небудь сайту або сервісу. Зазвичай, процес парсинг відбувається за допомогою серверних мов програмування: Java, Perl, PHP та інших. Результати парсингу часто записуються в базу даних чи у файл, в зручному для читання або обробки форматі, найчастіше в XML.

В епоху бурхливого зростання Мережі і жорстокої конкуренції вже всім ясно, що успішний веб-проект немислимий без розміщення великої кількості інформації на сайті. Обслуговування величезного потоку інформації яка динамічно змінюється не зможе забезпечити людина чи навіть злагоджена команда операторів. Часом інформація змінюється щохвилини і в ручному режимі оновлювати її навряд чи доцільно та можливо.

У порівнянні з людиною, комп'ютерна програма-парсер:

* швидко обійде тисячі веб-сторінок.
* акуратно відокремить технічну інформацію від «людської».
* безпомилково відбере потрібне і відкине зайве.
* ефективно упакує кінцеві дані в необхідному вигляді.

На сьогодні існує велика кількість бібліотек за допомогою яких можна парсити веб сторінки. Бібліотека Jsoup для роботи з HTML сторінками. Яка забезпечує дуже зручний API для витягання і обробки даних, з використанням з DOM, CSS, JQuery методів.

Jsoup реалізує специфікацію HTML5 WHATWG і аналізує HTML в DOM так само як і сучасні браузери, також надає можливість очистити і розібрати HTML з URL, знайти і витягти дані, використовуючи DOM обходу або CSS селектори, маніпулювати HTML елементами, атрибутами та текстом, Jsoup призначений для вирішення багатьох задач опрацювання HTML.

Jericho HTML Parser це Java бібліотека, яка дозволяє аналізувати і маніпулювати частинами HTML документа. Це бібліотека з відкритим вихідним кодом, яке розповсюджується за ліцензією Eclipse Public (EPL), GNU Lesser General Public License (LGPL), а Apache Licence.

Selenium WebDriver - це програмна бібліотека для управління браузерами. WebDriver являє собою драйвери для різних браузерів і клієнтські бібліотеки на різних мовах програмування, призначені для управління цими драйверами.

По суті використання такого веб-драйвера зводиться до створення бота, що виконує всю ручну роботу з браузером автоматизовано.

Бібліотеки WebDriver доступні на мовах Java, JavaScript, Python, .Net (C #), Ruby, драйвери реалізовані для браузерів Firefox, InternetExplorer, Safari, iOS (а також Chrome і Opera). Найчастіше Selenium WebDriver використовується для тестування функціоналу веб-сайтів/веб-орієнтованих додатків. Функціонал WebDriver дозволяє використовувати його не тільки для тестування, а й для адміністрування веб-сервісів, скоротивши до можливої межі кількість дій, які здійснюються в ручну. Також однією з незамінних особливостей Selenium WebDriver є очікування завантаження сторінки. Сюди можна віднести випадки, коли парсинг даних на сторінці неможливий через динамічні контент який міститься на сторінках і стає видимий після певних дій користувача, такі сторінки не вдасться опрацювати за допомогою інших парсерів, оскільки вони не можуть збирати динамічні дані. Selenium WebDriver дозволяє в таких випадках «очікувати», як очікувала би людина, поки на сторінці, наприклад, не з'явиться потрібний контент.

Найкраще для опрацювання соціальної мережі Facebook підходить бібліотека Selenium WebDriver, оскільки сторінки Facebook містять динамічний контент який з’являються поступово. Також всі інші бібліотеки не в змозі пройти авторизацію в соціальній мережі.

## **2.6. Вибір засобів для реалізації бази даних**

Для реалізації бази даних обрано СКБД PostgreSql. PostgreSQL — [об'єктно-реляційна](https://uk.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85&action=edit&redlink=1) [система керування базами даних](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) (СКБД). Дана СКБД є хорошою альтернативою комерційним СКБД  [IBM DB2](https://uk.wikipedia.org/wiki/DB2), [Microsoft SQL Server](https://uk.wikipedia.org/wiki/Microsoft_SQL_Server), [Oracle Database](https://uk.wikipedia.org/wiki/Oracle_Database) та інші, також таким СКБД як [MySQL](https://uk.wikipedia.org/wiki/MySQL), [Firebird](https://uk.wikipedia.org/wiki/Firebird), [SQLite](https://uk.wikipedia.org/wiki/SQLite), що також є з відкритим кодом.

Якщо порівнювати PostgreSQL з [FreeBSD](https://uk.wikipedia.org/wiki/FreeBSD), [Apache](https://uk.wikipedia.org/wiki/Apache) чи [MySQL](https://uk.wikipedia.org/wiki/MySQL), що є проектами з відкритим кодом, то PostgreSQL не ведеться тільки однією компанією, над її розробкою працює велика кількість компаній та людей, які мають бажання розробляти цю СКБД та реалізувати у неї свої ідеї і найновіші досягнення.

Сервер PostgreSQL розроблений за допомогою [мови C](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Який найчастіше розповсюджується як набір текстових файлів із [вихідним кодом](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%80%D1%86%D0%B5%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4). Для використання СКБД потрібно [відкомпілювати](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%86%D1%96%D1%8F) файли і помістити їх у відповідний каталог, деталі інсталяції детально описані в документації.

Реалізація Postgres СКБД почалася в 1986. Початкові концепції системи представлені в Task Postgres, а визначення первісної моделі даних показано в Model data Postgres. Postgres використовувався для реалізації багатьох досліджень та виробництва. Серед них: система аналізу фінансових даних, пакет контролю за продуктивністю реактивних двигунів, база даних пересування астероїдів і кілька географічних інформаційних систем. Postgres також використовувався як навчальний посібник в декількох університетах. В 1992 році, Postgres став первинним адміністратором даних для Sequoia 2000, наукового обчислювального проекту.

Postgres за цей час пережив кілька основних версій. Перша система "demoware" заробила в 1987 і була представлена ​​в 1988 на ACM-SIGMOD конференції. Ми випустили Версію 1, описувану в Реалізація Postgres, для декількох зовнішніх користувачів в червні 1989. У відповідь на критику перших правила системи (Коментар до системи правил Postgres), система правил була переконструювати (Про правила, процедури, кешуванні і уявленнях в системах баз даних ) та Версія 2 побачила світ у червні 1990 з новою системою правил. Версія 3 з'явилася в 1991 і додала підтримку для складеного адміністратора зберігання, покращився виконавець запитів, і була переписана система правил перезапису. Здебільшого, після випуску Postgres95 (дивися нижче) вся увага була перенесено на портативність і надійність. Основні можливості PostgreSql:

1. Усі функції можуть виконуватись код безпосередньо на сервері бази даних. Такі функції можна написати на чистому [SQL](https://uk.wikipedia.org/wiki/SQL), що містить деякі стандартні програмні оператори, такі як цикли та розгалуження. Набагато ефективнішою буде функція, що розроблена за допомогою високорівневої мови програмування, з яку підтримує PostgreSQL. Перелік таких мов:

* Вбудована мова, [pgSQL](https://uk.wikipedia.org/w/index.php?title=PL/pgSQL&action=edit&redlink=1), що схожа на процедурну мову [PL/SQL](https://uk.wikipedia.org/wiki/PL/SQL) від компанії [Oracle](https://uk.wikipedia.org/wiki/Oracle_Corporation).
* Мови для розробки сценаріїв: [PL/Tcl](https://uk.wikipedia.org/w/index.php?title=PL/Tcl&action=edit&redlink=1), [PL/Python](https://uk.wikipedia.org/w/index.php?title=PL/Python&action=edit&redlink=1),  [PL/Ruby](https://uk.wikipedia.org/w/index.php?title=PL/Ruby&action=edit&redlink=1), [PL/sh](https://uk.wikipedia.org/w/index.php?title=PL/sh&action=edit&redlink=1), [PL/Perl](https://uk.wikipedia.org/w/index.php?title=PL/Perl&action=edit&redlink=1).
* Високорівневі мови програмування [C](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))++, [Java](https://uk.wikipedia.org/wiki/Java), С .

Деякі функції мають можливість виконуватись із привілеями які надані користувачу, який здійснив виклик, чи написав цю функцію.

2. PostgreSQL підтримує індекси таких типів: [R-дерево](https://uk.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE),  [B-дерево](https://uk.wikipedia.org/wiki/R-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), GiST, GIN, [хеш](https://uk.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88). Користувач має можливість створювати нові типи індексів.

3. В PostgreSQL підтримується одночасна модифікацію БД декількома користувачами, це реалізується механізмом Multiversion Concurrency Control (MVCC). Це довзволяє забезпечувати імунітет до брудного читання і повної серіалізації, та виконувати вимоги ACID.

4. PostgreSQL реалізує великий набір вбудованих типів даних. Такі як: цілі, числові типи, з фіксованою комою, плаваючою комою, грошові типи, символьні типи різної довжини, двійкові типи, булевий тип, типи дати та часу, геометричні примітиви, перерахування, мережеві типи, [JSON](https://uk.wikipedia.org/wiki/JSON)-дані, [XML](https://uk.wikipedia.org/wiki/XML)-дані. Також, користувач має можливість створювати власні типи та розробляти для них механізм індексування GiST.

5. Таблиці мають можливість наслідувати набори полів та характеристики від інших таблиць. Всі дані, що були додані до створеної таблиці, автоматично будуть брати участь в запитах до батьківської таблиці.

PostgreSQL реалізує більшість SQL: 2011 стандарту, обробляє складні запити SQL, використовуючи безліч методів індексації, які не доступні в інших базах даних; має нові погляди і матеріалізовані уявлення, тригери, зовнішні ключі; підтримує функції і процедури, і легко маштабована, і має велику кількість розширень. На додаток до можливості працювати з основними базами даних з відкритим вихідним кодом, PostgreSQL підтримує міграцію з них, за допомогою своєї обширної підтримки стандарту SQL і доступних інструментів міграції. Тригери реалізуються як функція, що ініціалізується DML-операціями.

Наприклад, коли виконується операція UPDATE запускається певний тригер, який зберігає в іншу таблицю кількість оновлень поля, реалізуючи таким чином статистику. Тригери можуть бути написані на різних мовах програмування.

## **2.7. Вибір веб сервера**

Для виконання всієї логіки програми вибрано веб сервер Apache Tomcat. Apache Tomcat, часто згадується як Tomcat, є веб-сервер з відкритим вихідним кодом, розроблений Apache Software Foundation (ASF). Tomcat реалізує кілька специфікацій Java EE, включаючи Servlet, Java JavaServer Pages (JSP), Java EL і WebSocket, та забезпечує "pure" з’єднання HTTP веб-сервера для середовища Java.

Tomcat розроблявся в якості довідок сервлет реалізації Джеймса Дункана, архітектором програмного забезпечення в Sun Microsystems. Пізніше він допоміг зробити проект з відкритим вихідним кодом. Tomcat підтримується відкритим співтовариством розробників під егідою Apache Software Foundation, випущений під ліцензією Apache 2.0 ліцензія і програмне забезпечення є відкритим вихідним кодом.

Tomcat 4.x був випущений з Catalina (сервлет контейнер), Coyote (конектор HTTP) і Jasper (рушій JSP).

Catalina контейнер сервлетів Tomcat. Catalina реалізує специфікації Sun Microsystems для сервлетів і JavaServer Pages (JSP). У Tomcat, Realm елемент являє собою "базу даних" імен користувачів, паролів і ролей (за аналогією з Unix груп), призначений цим користувачам. Різні реалізації Realm дозволяють Catalina бути інтегровані в середовищах, де вже створені такі відомості.

Jasper розбирає файли JSP для компіляції в Java файли та перетворює їх в сервлети. Під час роботи, Jasper виявляє зміни в файлах JSP і компілює їх.

Сluster був доданий, щоб керувати великими додатками. Він використовується для балансування навантаження, що може бути досягнуто багатьма методами. Підтримка кластеризації в даний час вимагає JDK версії 1.5 або більш пізньої версії.

Отже для реалізації спеціалізованої програми аналізу груп індивідуумів у соціальній мережі обрано мову розробки програми Java та мову для розробки інтерфейсу користувача JavaScript, вибрано соціальну мережу для пошуку і аналізу груп індивідуумів Facebook, обрано СКБД PostgreSql, також було обрано веб сервер.

# **3. Реалізація спеціалізованої програми аналізу груп індивідуумів в соціальній мережі**

## **3.1. Розробка алгоритму роботи спеціалізованої програми аналізу груп індивідуумів в соціальній мережі**

Алгоритм роботи спеціалізованої програми аналізу груп індивідуумів в соціальній мережі зображено на рис. 3.1. Початок роботи системи починається з авторизації користувача для входження у систему. Якщо користувач раніше уже авторизувався у системі то відбудеться зчитування даних з бази даних, якщо ж ні, то відбудеться синхронізація з Facebook, якщо користувач надасть право на збір його особистих даних, отримані дані будуть збережені в базу даних. Після успішного входження в систему користувач буде перенаправлений на особисту сторінку, де він зможе визначитись з наступним кроком дій. Якщо користувач вибрав операцію збору даних, відбудеться перенаправлення на відповідну сторінку де він задає кількість кіл збору інформації. Відбудеться збір інформації з його сторінки соціальної мережі, якщо він задав коло спілкування більше одного то і з відповідних користувачів соціальної мережі. Вся зібрана інформація зберігається у базу даних та у файл. По завершенню збору даних з соціальної мережі відбудеться повідомлення користувача за допомогою електронної пошти. Якщо користувач обрав візуалізацію даних його буде перенаправлено на сторінку візуалізації. Користувач обирає потрібні йому дані для візуалізації, якщо таких даних немає йому потрібно буде перейти на сторінку збору даних та розпочати збір даних. Якщо раніше дані вже візуалізувались вони є структуровані та збережені в базу даних, відбудеться зчитування з бази даних та відображення на сторінці соціального графу, якщо це нові дані, які були нещодавно зібрані з соціальної мережі їх потрібно проаналізувати, та зберегти у відповідну структуру, визначити зв’язки вершин графа, ініціалізувати кожну вершину такими даними, як id, ім’я, прізвище, посиланням на особисту сторінку соціальної мережі, фотографією. Далі дані зберігаються для подальшого використання та відправляються користувачу.



Рис. 3.1. Алгоритм роботи спеціалізованої програми аналізу груп індивудуумів в соціальній мережі.

Архітектурна схема спеціалізованої програми аналізу груп індивідумів в соціальній мережі зображена на рис. 3.2.

Рис. 3.2. Архітектурна схема програми.

Програма реалізує стандартний архітектурний шаблон Модель-Вид-Контролер MVC. Суть цього шаблону щоб розділити програму на три основні частини і це дасть можливість створити гнучкий дизайн програми, що в подальшому спростить її підтримку. Модель забезпечує зберігання даних, та реалізує бізнес логіку. Компонент вигляд забезпечує надання даних користувачеві, це здійснюється за допомогою JSP технології. Контролер керує компонентами, отримує запити зі сторони користувача, опрацьовує їх та повідомляє модель про зміну даних. В якості контролера виступають класи Servlet. В системі реалізований шаблон “Команда”, який полегшує опрацювання даних контролером. Звертання до бази даних відбувається через шаблон “Сервіс” який в свою чергу доступається до шаблону DAO, компонування цих двох шаблонів забезпечує об’єктний доступ до бази даних.

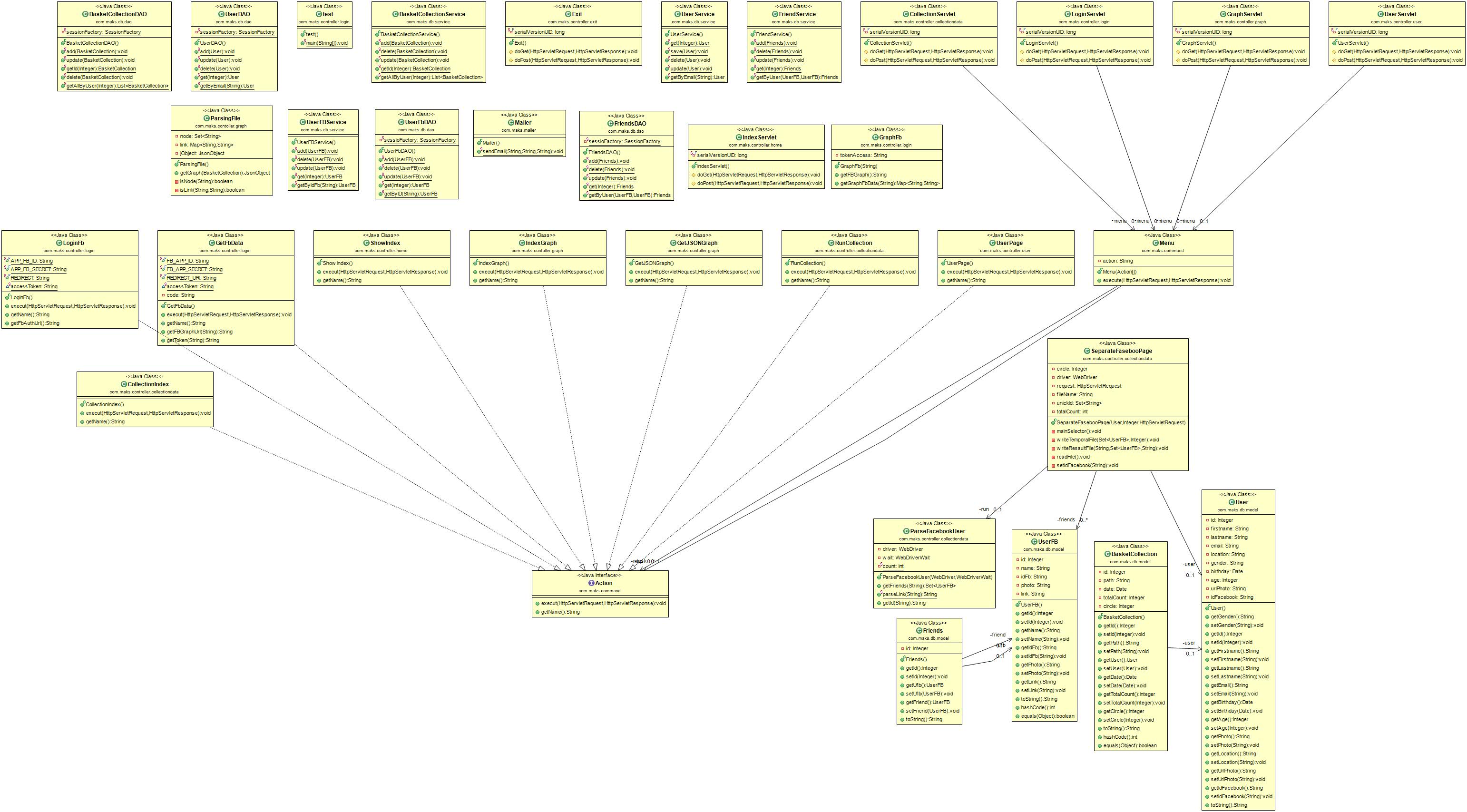


Рис. 3.3. Діаграма класів

## **3.2. Реалізація серверної частини спеціалізованої програми аналізу груп індивідуумів в соціальній мережі**

### **3.2.1. Структура проекту**

В основі стуркутури проекту лежить динамічний Web проект, який реалізований за допомогою Java EE технологій.

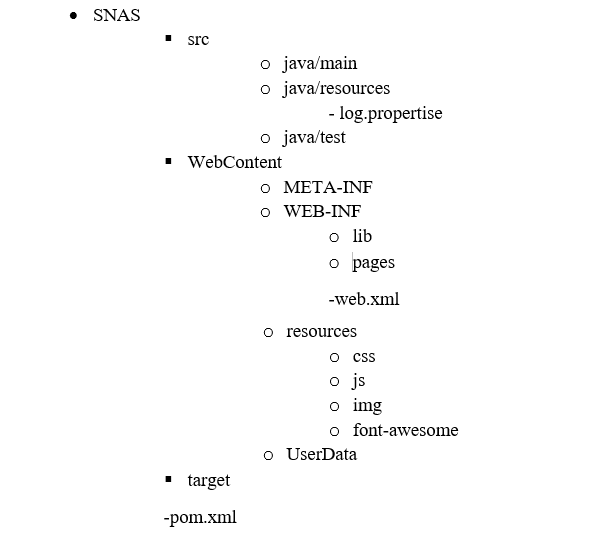


Рис. 3.4. Загальна структура проекту.

Загальна структура проекту зображена на рис. 3.4. Кореневим елементом в проекті є каталог SNAS в якому зберігаються усі складові проекту. Директорія src містить в собі підтеки, в java/main містяться пакети в яких зберігається source code, та сервлети, які будуть скомпільовані в бінарний код. Підтека java/test містить в собі набір Junit тестів, які покривають основну частину коду програми, а підтека java/resources зберігає файли налаштування та файли до логування. Тека WebContent є основною текою веб проекту, папка відображує в собі вміст файлу war який завантажується на сервер. Каталог WEB-INF містить в собі файл web.xml в якому описуються основні налаштування веб сервісу. Папка lib містить бібліотеки jar які будуть доступні web проекту. Папка pages зберігає html та jsp сторінки. В папці resources зберігаються допоміжні ресурси для веб проекту, таблиця стилів, javaScript файли, картинки, шрифти. Каталог UserData призначений для зберігання даних які завантажуються чи створюються користувачем під час роботи системи. Файл pom.xml є основним maven файлом який описує проект.

### **3.2.2. Авторизація та реєстрація користувача в системі**

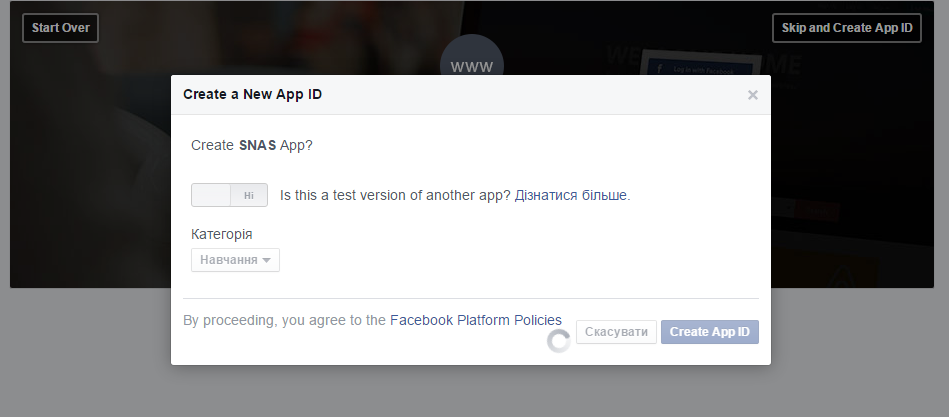
Оскільки програма орієнтована на соціальну мережу, реєстрація та авторизація користувачів відбувається за допомогою API соціальної мережі Facebook. Для того щоб мати можливість використовувати АPI соціальної мережі потрібно зареєструватись на сайті developers.facebook.com та створити новий додаток.

Рис. 3.3. Створення нового додатку для Facebook.

Після успішного створення додатку, потрібно його налаштувати для роботи з веб сервісом, вказати URL нашого сайту, обрати перелік даних які до яких користувач даватиме доступ здійснивши авторизацію та інші параметри. Головна сторінка додатку зображена на рис. 3.3. Створеному додатку присвоюється унікальний id та секретний ключ, ці два параметри в майбутньому будуть передаватись в запиті до створеного додатку.

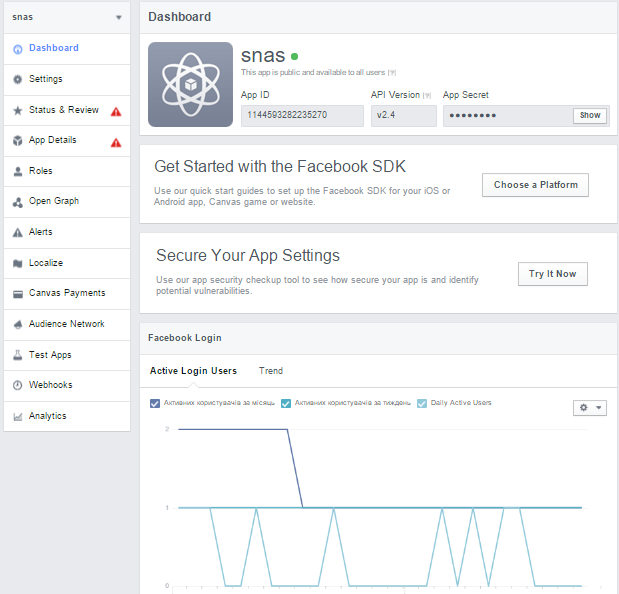


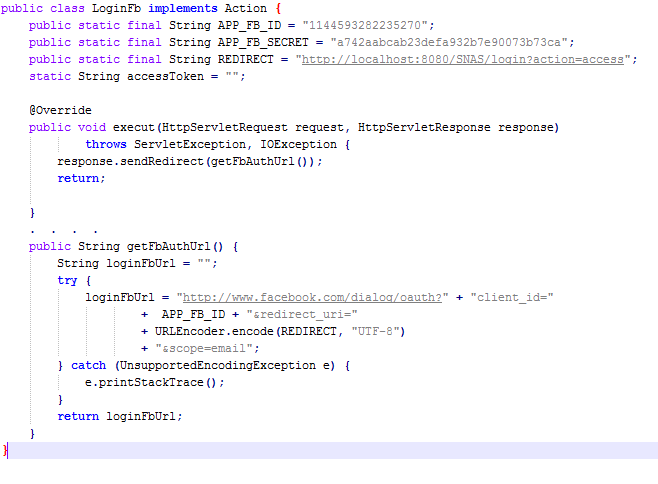
Рис. 3.4. Головна сторінка додатку.

Наступним кроком потрібно увійти у свій профіль соціальної мережі, якщо користувач залогований під час даної сесії браузера вхід виконається автоматично.

Під час першого входження в систему користувачу потрібно надати дозвіл доступу додатку до своїх даних, що зображено на рис. 3.5.

Рис. 3.5. Дозвіл доступу до даних.

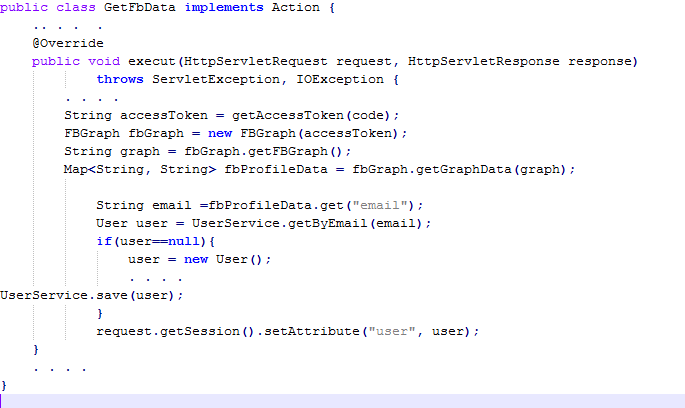
Авторизація користувача відбувається за допомогою сервлета “Login”. Коли користувач нажимає на кнопку логування то виконується клас LoginFB.

Даний клас містить константи та метод який формує url. Сформована url містить id додатку та секретний ключ, також як параметр передається url, за цією адресою відправляється хеш ключ, який передається як параметр при звертанні до Facebook Graph API.

Отримавши відповідь від сервера можна звертатись за потрібними даними соціальної мережі. Логіку отримання та опрацювання даних реалізовано в класі FBGraph.

За допомогою метода getGraphFbData() відбувається звертання до сховища даних соціальної мережі, поле graphURL містить url в якій в казано набір параметрів які повинні бути отримані з сервера, також останнім параметром потрібно передати ключ, який був отриманий раніше. Отримані дані зберігаються у форматі JSON.

За реєстрацію та авторизацію відповідає клас GetFbData. Якщо такий користувач вже зареєстрований в системі то дані дістаються з бази даних, якщо ж ні то такий користувач додається в базу даних і відбувається перенаправлення на особисту сторінку користувача.



### **3.2.3. Збір даних з соціальної мережі**

Соціальна мережа Facebook містить в собі великий набір API, за допомогою яких можна отримувати дані з соціальної мережі, до яких користувач надав доступ. Facebook проводить сильну політику захищеності даних і це перешкоджає отримувати дані які потрібні для роботи програми. За допомогою Fecabook API можна отримати дані лиш про конкретного користувача, для програми потрібно збирати дані про вибране користувачем коло спілкування. Тому було обрано фреймворк Selenium, який містить в собі реалізацію WebDriver за допомогою якого можна отримувати дані з веб сторінок. Мінусом такого збору даних є те, що парсинг веб сторінки відбувається в рази довше ніж звичайний запит до веб сервера.

Головний клас для збору даних з соціальної мережі є **SeparateFasebooPage.** Основним методом є mainSelector в якому реалізована головна логіка роботи збору даних.

**public** **class** **SeparateFasebooPage** {

**. . . . . .**

**private** **void** **mainSelector**() {

**FirefoxProfile** **profile** = **new** FirefoxProfile();

. . . .

driver = **new** FirefoxDriver(capabilities);

**. . . .**

**try** {

driver.get("https://www.facebook.com/");

driver.findElement(**By**.*id*("email")).sendKeys(user.getEmail());

driver.findElement(**By**.*id*("pass")).sendKeys(user.getKey());

driver.findElement(**By**.*id*("u\_0\_x")).click();

run = **new** ParseFacebookUser(driver, wait);

friends = run.getFriends(user.getIdFacebook());

writeResaultFile(user.getIdFacebook(), friends, fileName);

writeTemporalFile(friends, 1);

readFile();

**BasketCollection** **basket** = **new** BasketCollection();

. . . . .

**BasketCollectionService**.*add*(basket);

driver.close();

}

**private** **void** **writeResaultFile**(**String** id, Set<UserFB> setFriend,

**String** fileName) {

**for** (**UserFB** **userFB** : setFriend) {

**System**.***out***.println(userFB);

**UserFBService**.*add*(userFB);

write.write(userFB.getIdFb() + ",");

**Friends** **friend** = **new** Friends();

friend.setUfb(ufb);

userFB =**UserFBService**.*getByIdFb*(userFB.getIdFb());

friend.setFriend(userFB);

}

write.close();

}

**private** **void** **readFile**() {

**for** (**int** **i** = 1; i < circle; i++) {

**. . . .**

**while**((id= read.readLine())!=**null**){

**if**(!unickId.contains(id))

{

friends= run.getFriends(id);

writeResaultFile(id, friends, fileName);

**if**(i==circle)

**continue**;

writeTemporalFile(friends, i+1);

}

} }

Для парсингу сторінок обрано FireFox драйвер, який потрібно налаштувати та передати в WebDriver. Це відображено в наступних рядках коду.

**FirefoxProfile** **profile** = **new** FirefoxProfile();

. . . .

driver = **new** FirefoxDriver(capabilities);

Наступним кроком відбувається вхід в соціальну мережу. Пройшовши етап авторизації відбувається збір даних з соціальної мережі. Парсинг сторінки здійснює об’єкт ParseFacebookUser(driver, wait); який реалізує метод Set<UserFB> getFriends(String id); якому як параметр потрібно передати id користувача соціальної мережі, який повертає список всіх друзів користувача. Отриманий спикос друзів потрібно записати в результуючий файл і зберегти на сервері це виконує метод **writeResaultFile().** Таким чином відбувся збір першого кола друзів користувача. Якщо користувач обрав збір даних більше ніж одного кола друзів, то зібрані дані зберігаються також у проміжний файл, з якого будуть зчитуватись id користувачів для збору даних. Метод readFile() реалізую збір даних заданих наступних кругів спілкування користувача, раніше збережені ід користувачів зчитуються, опрацьовується їхнє коло спілкування та зберігається. Після збору даних відбувається зберігання основних результатів в базу даних. Оскільки парсинг сторінок займає доволі багато часу, по закінченню збору даних здійснюється повідомлення користувача за допомогою email, що збір даних завершився і вони готові до аналізу.

Парсинг сторінки конкретного користувача відбувається в класі **ParseFacebookUser.**

**public** **class** **ParseFacebookUser** {

**. . . . ..**

**public** **ParseFacebookUser**( WebDriver driver, **WebDriverWait** wait) {

. . . .

}

**public** Set<UserFB> **getFriends**(**String** id){

. . . . . . .

**if**(id.replaceAll("\\d\*", "").isEmpty()){

driver.get("https://www.facebook.com/profile.php?id="+id+"&sk=friends");

}**else**{

driver.get("https://www.facebook.com/"+id+"/friends?pnref=lhc");

}

wait.until(**ExpectedConditions**.*visibilityOfElementLocated*(**By**

.*id*("medley\_header\_friends")));

**. . . . . .**

((JavascriptExecutor) driver).executeScript("window.scrollBy(0,5000)", ""); //y value '400' can be altered

List<WebElement> **frends** = driver.findElements(**By**.*className*("\_698"));

**for** (WebElement **webElement** : frends) {

**UserFB** **ufb** = **new** UserFB();ufb.setIdFb( *parseLink*( webElement.findElement(**By**.*tagName*("a")).getAttribute("href")));

ufb.setLink(webElement.findElement(**By**.*tagName*("a")).getAttribute("href"));

. . . . .

}

.. . . .

**return** list;

}

. . . . . .

}

Перейшовши на сторінку з друзями користувача відбувається скролинг сторінки, це потрібно для того щоб відобразити усіх друзів. Наступним кроком знаходяться усі html елементи з класом ”\_698”, вони містять усю потрібну інформацію про користувача.

List<WebElement> **frends** = driver.findElements(**By**.*className*("\_698"));

Отримавши список об’єктів WebElement можна витягнути усі потрібні дані. З цього об’єкта витягується id користувача, ім’я, прізвище, та фотографія і створюється новий об’єкт UserFb, який зберігається в базу даних. Опрацювавши усі веб елементи формується список користувачів яких повертає метод.

#### **3.2.3.1. Формат даних**

Зібрані дані задаються в графі як список суміжності. Кожна вершина графа відповідає своєму списку. У список, що відповідає вершині v послідовно записуються всі суміжні їй вершини.

|  |
| --- |
| 100003795194298 100004059080855,petro.ryndych,valdikk,ulyana.chandruk |

де, 100003795194298 є вершина графа, а 100004059080855 суміжна їй вершини. Цей формат використовується при завантаженні даних та зберіганні їх.

Для пошуку даних використовується інший формат, який передбачає зберіганні відстані до головної вершини та до даної, а також весь шлях до даної точки :

|  |
| --- |
| 100003795194298 3: 100004059080855, 100002332323215, 1000040592389405 |

В даному прикладі пошук виконувався від вершини 100003795194298. Вершини 100004059080855 та 100002332323215 є проміжними в процесі пошуку. Кількість ребер між вершинами рівна 3. Кінцевою вершиною у пошуку буде 1000040592389405.

## **3.3 Реалізація бази даних**

База даних реалізована та зберігається в СУБД PostgreSQL. Для взаємодії бази даних та програми використано платформу ORM Hibernate. База даних складається з наступних таблиць: user, userbf, basket, friends, graphdata, role.

Таблиця user зберігає всі дані про користувача системи, та містить наступні поля.

Опис стовпців таблиці user:

* id integer NOT NULL,- унікальний ідентифікатор
* age integer, -вік користувача
* birthday date, -дата народження
* email character varying(255), -емейл адреса
* firstname character varying(255), -ім’я користувача
* lastname character varying(255), - прізвище
* gender character varying(255), -стать
* idfacebook character varying(255)-іd адреса Facebook сторінки
* location character varying(255), - місце проживання
* urlphoto character varying(255), - посилання на фотографію
* role integer, - роль користувача в системі

Таблиця userfb зберігає користувачів соціальної мережі, які буди виявлені під час збору даних.

Таблиці userfb містить наступні поля

* id integer NOT NULL,- унікальний ідентифікатор
* idfb character varying(255),-id Facebook сторінки
* link character varying(255),-посилання на сторінку Facebook
* name character varying(255), - ім’я та прізвище
* photo character varying(255), - фотографія користувача
* amountfriends integer, -кількість друзів в соціальній мережі

Таблиця basket зберігає інформація про зібрані дані з соціальної мережі.

Опис стовпців таблиці basket:

* id integer NOT NULL, - унікальний ідентифікатор
* circle integer, - кількість кіл спілкування
* date timestamp without time zone, - час збору
* path character varying(255), -шлях до файлу з інформацією
* totalcount integer, -загальна кількість зібраних користувачів
* user\_id integer, - користувач який збирав інформацію

Таблиця friends містить інформацію про взаємовідносини користувачів в соціальній мережі.

Опис стовпців таблиці friends:

* id integer NOT NULL, - унікальний ідентифікатор
* friend integer, - id ідентифікатор рядка таблиці userfb
* ufb integer, - id ідентифікатор рядка таблиці userfb

Таблиця graphdata містить опрацьовані дані для візуалізації. Ця таблиця використовується для оптимізації роботи системи, не потрібно кожний раз опрацьовувати дані для візуалізації.

Опис стовпців таблиці graphdata:

* id integer NOT NULL, - унікальний ідентифікатор
* graph json, - опрацьовані дані
* basketid integer, - ключ зібраних даних

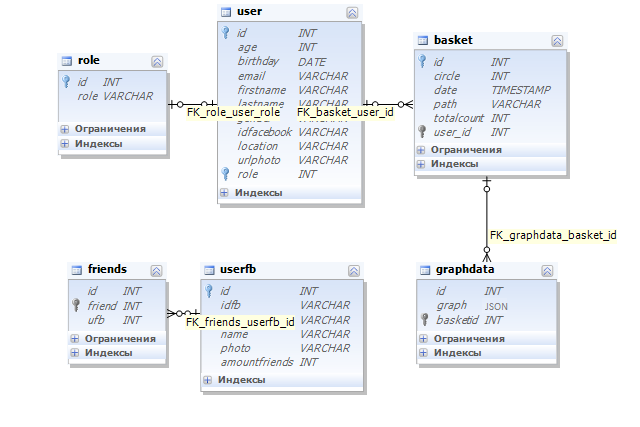
Таблиця role, призначена для зберігання ролей користувачів системи.

Рис. 3.6. Структурна схема бази даних “SNAS”.

В програмі для кожної таблиці розроблений клас модель який проанотований анотаціями JPA і який відображає в собі структуру рядка таблиці. Робота з базою даних здійснюється за допомогою патерна DAO, доступ до бази даних відбувається за допомогою класів сервісів які викликають потрібні методи в DAO.

## **3.4. Реалізація клієнтської частини спеціалізованої програми аналізу груп індивідуумів в соціальній мережі**

Клієнтська частина програми реалізована за допомогою мови javaScript та html5 з css. Для надання сучасного дизайну та зручного інтерфейсу використано фреймворк Boststrap, що є найбільш популярним веб фреймворком, який дуже зручний та простий у використанні.

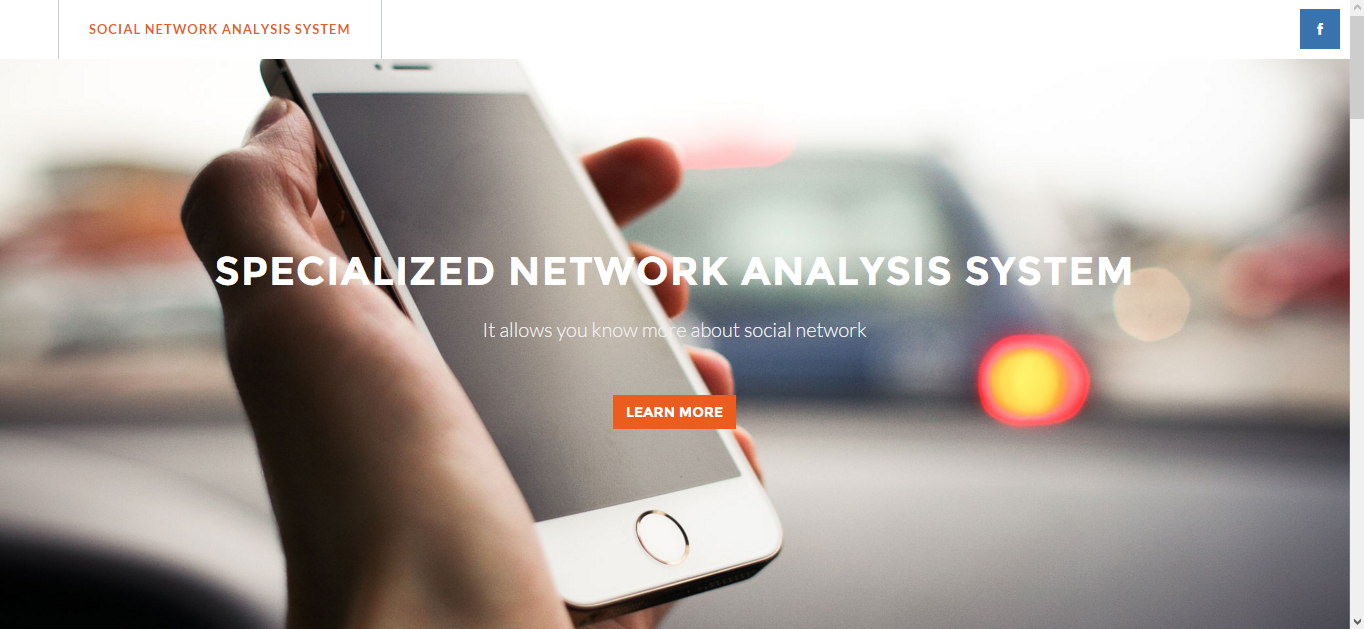
Коли користувач відвідує сайт, то він буде направлений на головну сторінку, яка містить загальну інформацію про систему та її можливості. Головна сторінка зображена на рис. 3.7.

Рис. 3.7. Головна сторінка системи.

Головна сторінка системи виконана у стилі “single page” застосунку. На ній знаходиться форма для зворотного зв’язку яка показана на рис. 3.8.

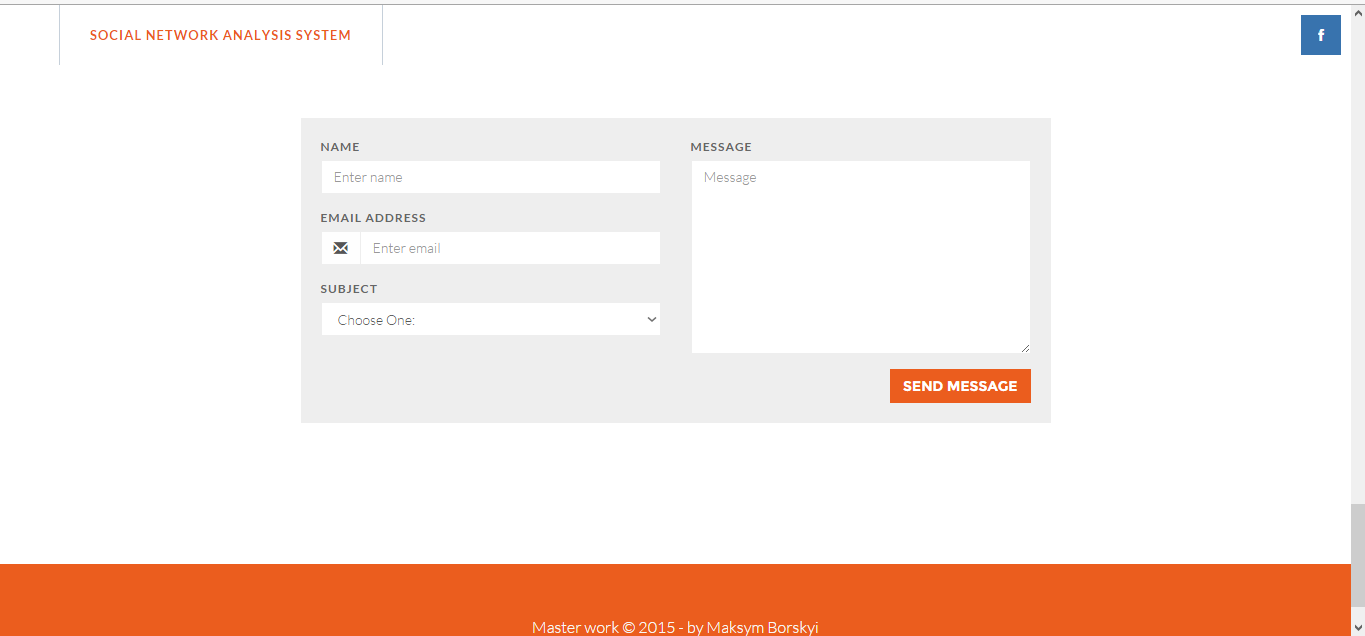


Рис. 3.8. Форма зворотного зв’язку.

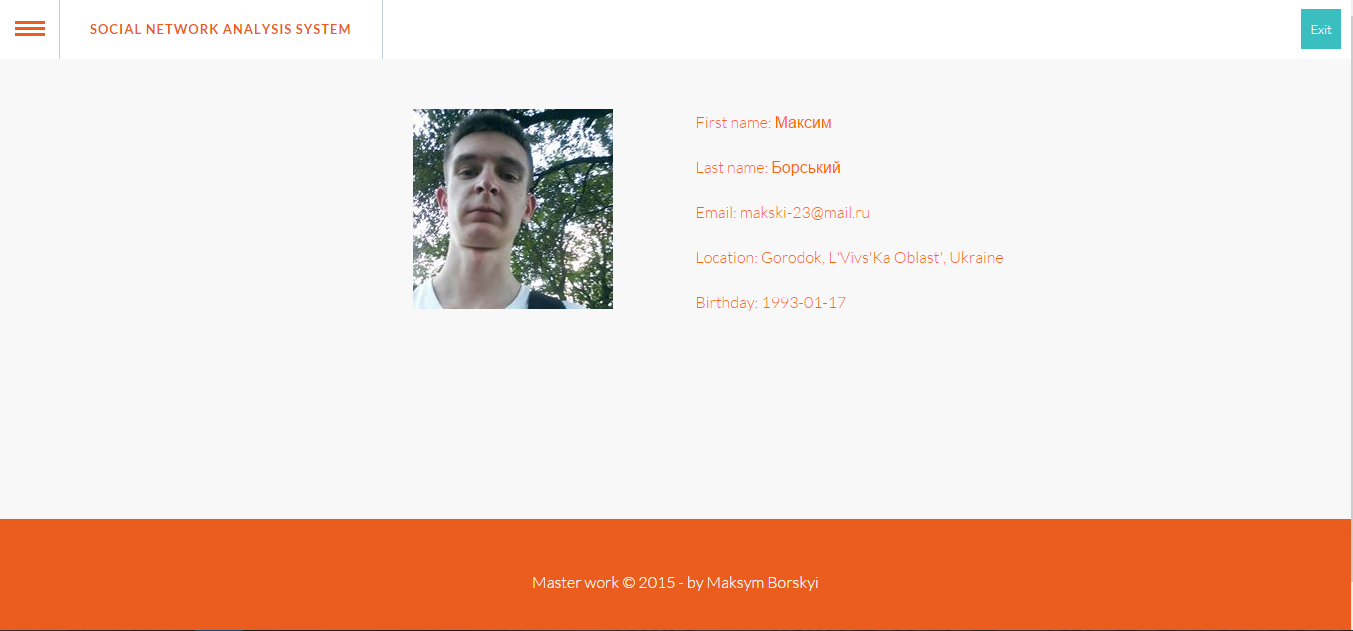
У правому кутку головної сторінки знаходиться посилання для входження в систему за допомогою соціальної мережі Facebook. Якщо користувач на зареєстрований в цій соціальній мережі нажаль йому не вдасться скористатись веб сервісом для аналізу соціальної мережі. Авторизувавшись за допомогою соціальної мережі користувача перенаправить в особистий кабінет який зображений на рис. 3.9.

Рис. 3.9. Сторінка користувача.

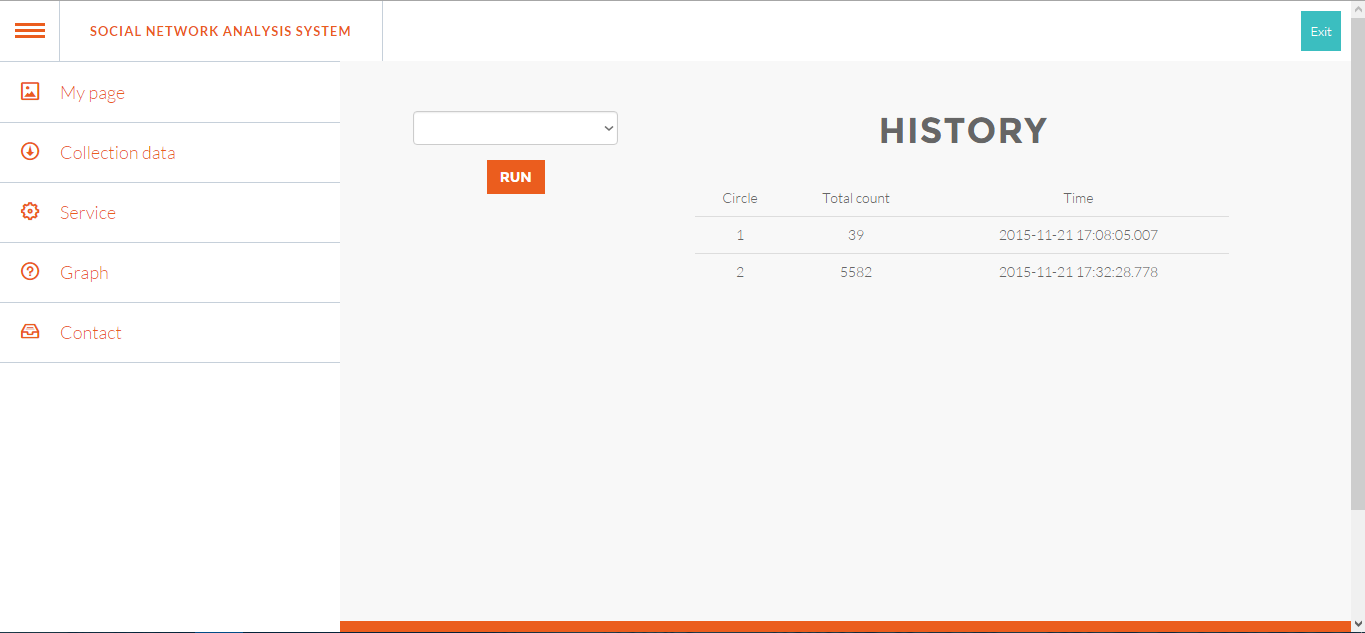
На сторінці користувача знаходиться особиста інформація про користувача також у лівому кутку з’явилось динамічне меню, через яке користувач матиме можливість доступатись до усіх сторінок сервісу.

Рис. 3.10. Сторінка збору даних.

На рис. 3.10. зображено сторінку для збору даних, користувач обирає яке коло спілкування йому потрібно зібрати і виконується збір даних з соціальної мережі, збір даних досить тривала операція тому після закінчення збору даних користувачу буде надіслано лист на електронну пошту з інформацію про готовність даних та можливість їх використання. Праворуч розміщена історія попереднього збору інформації.

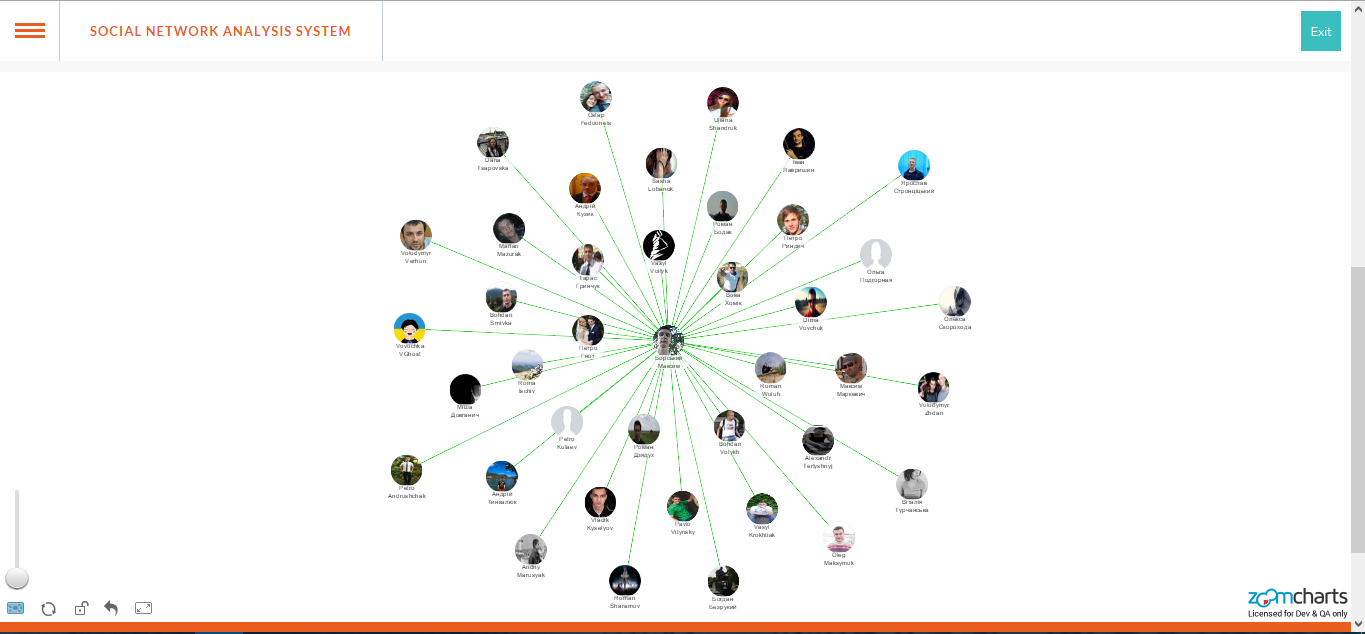


Рис. 3.11. Візуалізація соціального графа.

Перейшовши на сторінку візуалізації соціального графа рис. 3.11. користувач обирає йому потрібні дані, які були зібрані раніше з соціальної мережі та візуалізує їх. Візуалізація відбувається за допомогою javaScript бібліотеки zoomcharts, ця бібліотека містить в собі великий набір можливостей для відображення даних. Вибравши потрібну ітерацію даних, відбувається запит до сервера, на якому відбувається звертання до бази даних, якщо вже раніше ці дані були туди збережені, якщо їх там немає відбуваються обчислення соціального графа і структуровані дані відправляються користувачу за допомогою JSON та AJAX технології. Отримавши дані користувач має змогу маніпулювати ними. Є можливість видаляти певні ноди з соціального графа, переміщувати їх на потрібне місце, фокусуватись на певній ноді, закріпити листок графа на певній позиції. Присутня можливість маштабування графа. Зробивши подвій клік по конкретному листку графа відбудеться перенаправлення на сторінку користувача в соціальний мережі.

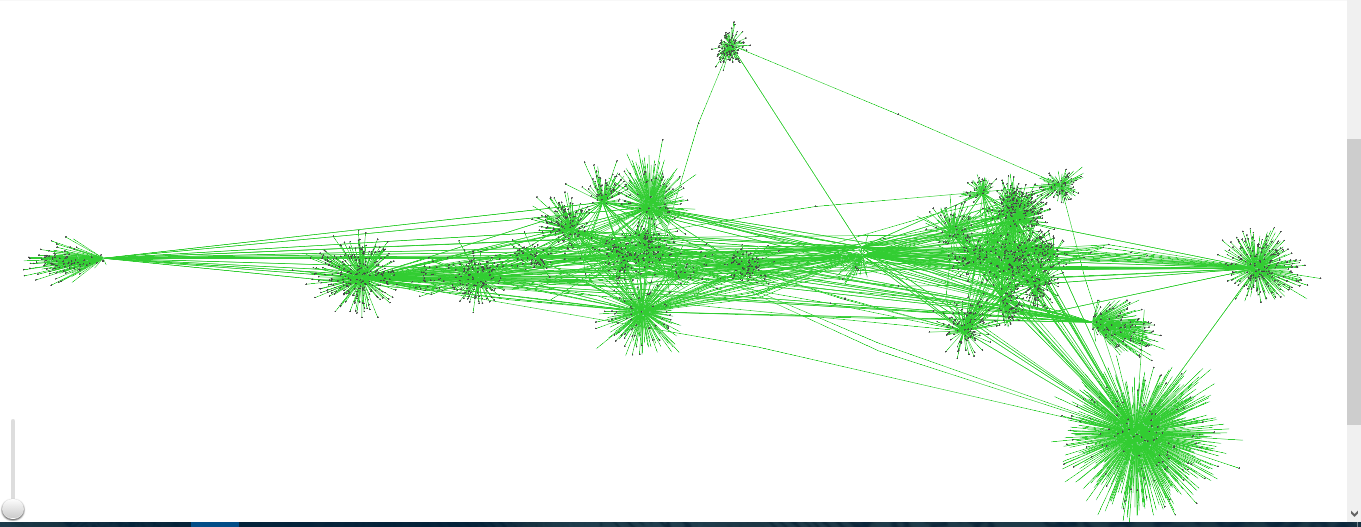


Рис. 3.12. Візуалізація другого кола спілкування.

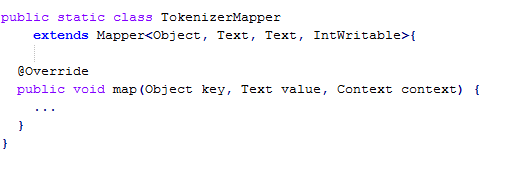
На рис. 3.12. зображено візуалізацію другого кола спілкування користувача соціальної мережі, яке містить 5585 листків які зв’язані між собою.

# **4. Опис проведених досліджень**

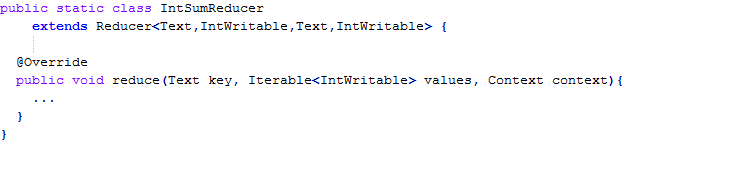
## **4.1. Розробка MapReduce програми**

Весь процес опрацювання даних побудований на парадигмі MapReduce. Суть її в тому, що ми поділяємо всю роботу на два етапи: Map і Reduce. Вхідними даними буде слугувати тека з файлами.

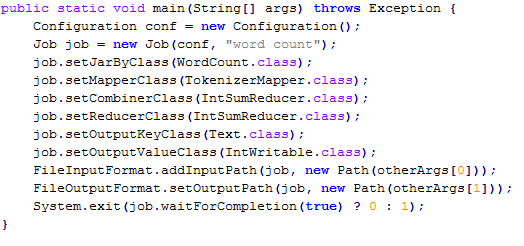
На Map кроці ми як аргумент отримуємо ключ і значення. Далі ці дані проходять обробку подаючи на вихід список ключів і список значень.



На етапі Reduce ми отримуємо в якості аргументу один ключ і всі від­повідні йому значення отримані на виході Map методу для їх подальшої обробки. У нашому випадку ми отримали шлях до файлу, де знайдений текст відповідний заданим шаблоном (ключ) і набір рядків, де були знайдені збіги (список значень).



У класі-драйвері відбувається настройка завдання (установка Map і Reduce, типу вхідних і вихідних даних і т. д.).



Оскільки я все упакував у виконуваний jar-файл, то запустити програму можна так:

|  |
| --- |
| <шлях до hadoop> /bin/hadoop jar/home/hduser/WordCount.jar <шлях з файлами для аналізу> <шлях для збереження результатів> |

Шляхом для збереження результатів повинна бути неіснуюча директорія. Якщо налаштувати Hadoop в pseudo-distributed mode, то дані зберігаються у файловій системі HDFS і їх звідти ще потрібно буде витягти.

## **4.2. Підготовка даних**

Для експерименту була вибрана соціальна мережа Facebook та мову програмування Java. В якості початкової вершини графа використовувався мій Facebook акаунт. Було зібрано та опрацьовано 5 кіл спілкування. Знайомими вважалися люди, які знаходилися в списку друзів, які були підтвердженні з обох сторін.

Для збору персональної інформації скористався фреймворком Selenium. Збір даних тривала за часом операція, яка здійснювалась декілька годин. Аналіз даних проводився в псевдо паралельному режимі Hadoop 2.2.0 з конфігурацією системи: Процесор Intel Core i5-3210M, 2.5 ГГц, 4Мб кеш-пам'ять, 2 ядра; 4Гб 1600МГц DDR3L SDRAM; HDD 600 Гб SATA (5400 об/хв); Windows 10.

## **4.3. Теорія шести рукопотискань**

Для пошуку найкоротшого шляху було використано алгоритм Дейкстри для MapReduce моделі програмування. Робота всіх ітерацій зайняла 4,5 год (Рис. 4.1).

Ефективність обчислень сильно залежать від пропускної можливості мережі між кластерами, через те, що структура графа передається по мережі після кожної ітерації. Оптимізувати цей алгоритм мені вдалося за рахунок використання оперативної пам’яті, в яку загружав відстані до відомих нам вершин. В такому випадку структура графа буде незмінна, а по мережі будуть передаватися тільки відстані між вершинами.

Враховуючи кількість зав’язків які індивід може підтримувати в соціальній мережі (Число Данбара), у кожного з нас, приблизно 150 знайомих. У кожного з них теж приблизно по 150, що дає коло спілкування 22,5 тис. чоловік. На третьому кроці це коло буде рівне 3,4 млн. чоловік, а на четвертому 5 млрд. Досить оптимістичні результати, але проблема в тому, що коло спілкування A досить суттєво перетинається з колом спілкування B. Тобто частина знайомих кожного з ваших друзів часткова входить в ваше коло знайомств.

Також важливу роль відіграє кастовість населення, тобто люди схильні спілкуватися з собі подібними. І наявність в одному колі друзів представників двох різ-них соціальних груп є дуже малою.

Рис. 4.1. Залежність часу обчислення від розміру вхідних даних

Проаналізувавши дані отримані з соціальної мережі, визначив кількість вершин та ребер на кожному кроці ітерації від початкової вершини (Табл. 4.1). Кількість ребер відповідає кількості знайомих без об’єднання спільних, а вершини це реальна кількість знайомих.

*Таблиця 4.1*

Кількість друзів

|  |  |  |
| --- | --- | --- |
| Коло спілкування | Ребра | Вершини |
| 1 | 39 | 39 |
| 2 | 10873 | 5582 |
| 3 | 25046986 | 4953068 |
| 4 | 2318488285 | 85509543 |

Круги спілкування для дослідження

1

2

3

4

5

Рис. 5.2. Кола друзів

Для підтвердження теорії шести було проведено дослідження. Результати дослідження показали, що переважаюча більшість людей знайомі через чотири рукопотискання (Рис. 4.3). Таким чином число рукостискань, між двома будь-якими користувачами соціальної мережі Facebook, в середньому становить 4,118. Ця цифра підтверджує теорію шести рукопотискань. Тож це дослідження можна вважати її підтвердженням.

Рис. 4.3. Статистика результату дослідження

Як приклад, побудовано граф з найкоротших шляхів від мене до засновника SpaceX Elon Musk. В результаті граф містив 332 вершину та 417 різних рукопотискань довжиною ланцюжка 4 рукопотискання. Для наглядного відображення кількість вершин відфільтрувалися з умовою кількість друзів менша тисячі. В результаті фільтрації граф зменшився до 28 вершин та 54 різних рукопотискань (Рис. 4.4).

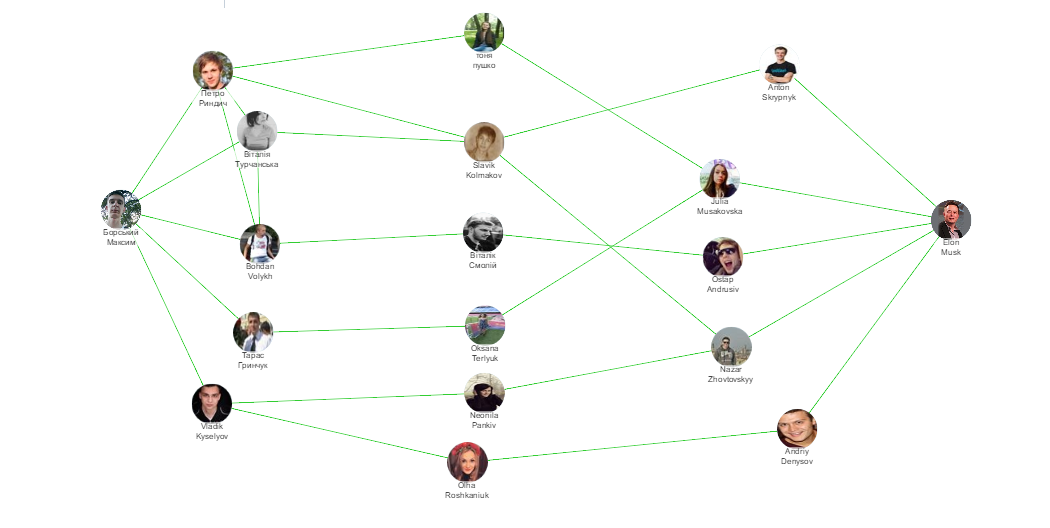


Рис. 4.4. Граф найкоротших шляхів до певної вершини

## **4.4. Число Данбара**

Маючи перед собою досить велику базу даних користувачів, вирішив перевірити Чило Данбара. Суть якого полягає в максимальна кількість людей, з якими людина може підтримувати стабільні соціальні відносини. На основі отриманих даних Данбар вивів математичну залежність де кількість осіб в групі лежить від 100 до 230, частіше всього вважають рівним 150.

З отриманих результатів побудував гістограму (Рис. 4.5). Для наглядності обрізав «хвіст» графіку виключивши користувачів кількість друзів, яких, більша тисячі. Це в основному відомі люди (музиканти, ведучі, шоумени, фотографи та інші) і навряд вони знають всіх своїх друзів.

Рис.4.5. Кількість друзів користувачів.

З отриманих результатів можна зробити висновок що найбільше користувачів мають по 100 друзів. Пік припадає від 50 до 200.

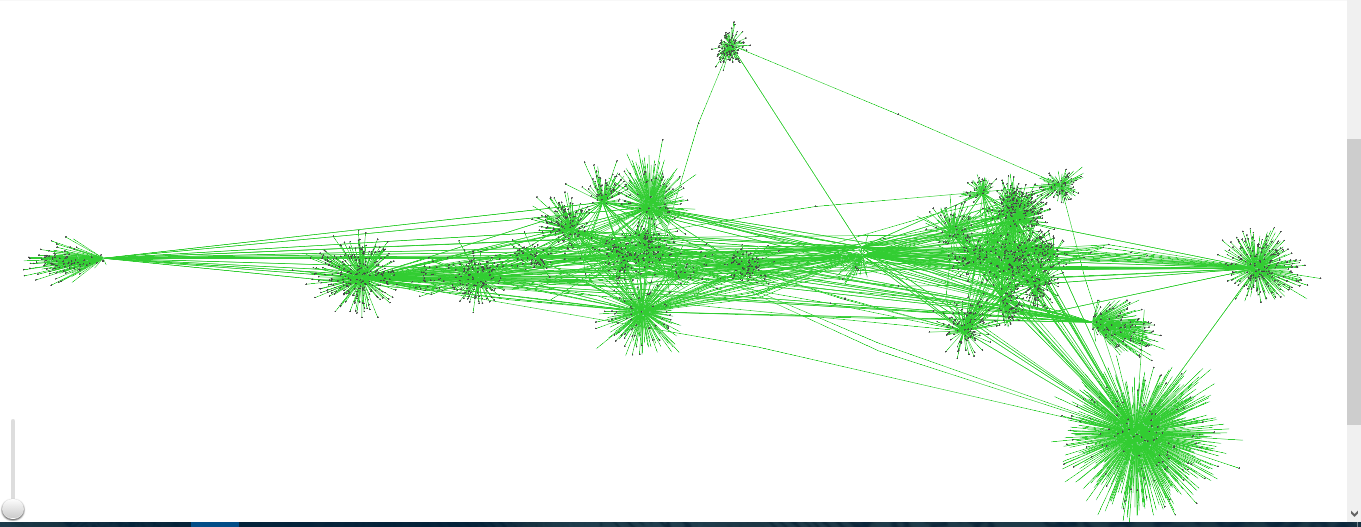
Візуалізація соціальних мереж, як один із способів її аналізу, має важливе значення. Вона дає можливість виявити неформальні товариства, знайти слабкі місця, визначити лідерів і т.д. (Рис.4.6).

Рис. 4.6. Граф другого кола спілкування, об’єкта дослідження

Проаналізувавши візуально соціальний граф можна прийти висновку що тут є три спільноти в які входить користувач. Це може бути друзі по місцю проживання, школа, університет з роботи та інш.

# **РОЗДІЛ 5. ЕКОНОМІЧНА ОЦІНКА ПРОЕКТНОГО РІШЕННЯ**

## **5.1. Економічна характеристика проектного рішення**

Метою магістерської роботи є аналіз соціальної мережі та розробка математичних моделей для дослідження даних, за допомогою яких можна аналізувати, оцінювати, вести порівнювальну характеристику між об’єктами соціальних мереж.

Дослідження соціальних мереж - це новий погляд на активну аудиторію, яка хоче впливати на подальший розвиток бренду, компанії. У соціальних мережах можна не тільки «зібрати» відгуки та коментарі клієнтів, але й почерпнути нові ідеї, які надалі можна використовувати для розвитку бізнесу та створення нових продуктів.

Деякі поширені додатки мережевого аналізу включають в себе збір і накопичення даних, моделювання розповсюдження мережі, моделювання мережі і вибірок, аналіз характерних ознак і поведінки користувача, ресурсна підтримка, що забезпечується співтовариством, аналіз взаємодії на основі місця розташування, соціальний обмін і відбір, розвиток систем рекомендацій, а також прогнозування зв'язків і аналіз об'єктів. У приватному секторі фірми використовують аналіз соціальних мереж для підтримки такої діяльності, як взаємодія та аналіз клієнтів, маркетинг і бізнес-аналітика. Використання аналізу соціальних мереж державним сектором включає в себе розвиток стратегій участі керівництва, аналіз індивідуальної і групової участі, використання засобів масової інформації заснованої на спільнотах.

Так само використовується в розвідувальних, контррозвідувальних та правоохоронних заходах. Розроблені методи дозволяють аналітикам відобразити на карті нелегальну або приховану організацію, таку як шпигунське коло, організовану злочинну громаду або вуличну банду.

Варто підкреслити, що аналіз і дослідження соціальних мереж дозволить отримати всі дані за цікавими темами, зібрати авторів та інформацію про них. Зробити різні зрізи мереж по поширенню інформації, знайти популярні теми, визначити лідера в соціальних зв’язках.

Така розробка є надзвичайно актуальною на сьогоднішній день, оскільки в зв’язку із швидким розвитком об’ємів даних, виникає необхідність у їх опрацюванні. Вона буде актуальною для спеціалістів, які працюють в сферах :

* аналізу і дослідження соціальних мереж;
* опрацювання великих об’ємів графів.

## **5.2. Розрахунок витрат на розробку програмного забезпечення**

Витрати на розробку і впровадження програмних засобів (К) включають:

К = К1+К2 (5.1)

де K1 - витрати на розробку програмних засобів, грн.

К2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на ЕОМ, грн.

Витрати на розробку програмних засобів включають:

- витрати на оплату праці розробників;

- витрати на відрахування у спеціальні державні фонди (Вф,);

- витрати на куповані вироби (Кв);

- витрати на придбання спецобладнання для експериментальних  
робіт (Об);

- накладні витрати (Н);

- інші витрати (Ів).

Витрати на оплату праці розробників проекту визначаються за формулою:

 (5.2)

де nij - чисельність розробників і-ої спеціальності j-ro тарифного

розряду, які приймають участь в проектуванні, чол;

tij - час, який затрачений на розробку проекту співробітника і-ої

спеціальності j-ro тарифного розряду, днів;

Cij - денна заробітна плата і-ої спеціальності j-ro тарифного розряду, грн;

 (5.3)

де Сij - основна місячна заробітна плата розробника і-ої спеціальності

j-гo тарифного розряду, грн.;

h - коефіцієнт, що визначає розмір додаткової заробітної плати;

р - середня кількість робочих днів у місяці.

Таблиця 5.1

**Вихідні дані для розрахунку витрат на оплату праці**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Посада виконавців | Місячний оклад, грн. | Середньоденна ставка, грн./дні |
| 1 | Доцент | 3461 | 164,81 |
| 2 | Консультант з економіки | 3036 | 144,57 |
| 3 | Студент | 720 | 34,30 |

Таблиця 5.2

**Розрахунок витрат на оплату праці**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Спеціальність розробника | Час розробки, ДНІ | Денна заробітна плата, грн. | Витрати на розробку, грн. |
| 1 | Доцент | 30 | 164,81 | 4944,30 |
| 2 | Консультант з економіки | 2 | 144,57 | 289,14 |
| 3 | Студент | 60 | 34,30 | 2058 |
|  | Разом |  |  | 7291,44 |

Величину відрахувань у спеціальні державні фонди визначають у процентному співвідношенні від суми основної та додаткової заробітної плати. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 36,2%від суми заробітної плати:

Вф=36,2:100\*З (5.4)

Вф=0,362\*7291,44=2639,50 грн.

Таблиця 5.3

**Розрахунок витрат на куповані вироби**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Найменування купованих виробів | Одиниця виміру | Ціна за одиницю виміру, грн. | Кількість купованих виробів | Сума, грн. |
| 1 | Папір (формат А4) | 500 листів | 70,00 | 1 | 70,0 |
| 3 | Блокнот | Шт. | 4,00 | 2 | 8,0 |
| 6 | Диск (DVD-RW) | Шт. | 9 | 1 | 9,0 |
| Всього | 87,0 |  |  |  |  |

При розробці даного програмного забезпечення спеціальне обладнання не використовувалось, тому витрати на спеціальне обладнання відсутні. Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими процентами до витрат на оплату праці:

 (5.5)

Н = 0,3\*7291,44=2187,43грн.

Інші витрати відображають видатки, які не враховані в інших статтях витрат. Вони розраховуються за встановленими процентами до витрат на оплату праці:

Ів =10:100\*З (5.6)

Ів=0,1\*7291,44=729,14 грн.

Витрати на розробку програмного забезпечення розраховуються за формулою:

К=3 + Вф + Кв + Об + Н + Ів (5.7)

К =7291,44+2639,5+87,0+2187,43+729,14=12934,51 грн.

Витрати на відлагодження і дослідну експлуатацію програмного забезпечення визначаються за формулою:

K2=SMr\*tBiд (5.8)

де SMr - вартість однієї машино-години роботи конкретного типу ЕОМ, грн/год;

tBід - машинний час, витрачений на відлагодження і дослідну

експлуатацію програмних засобів, год.

Загальна кількість днів роботи на ЕОМ рівна 60 днів. Середній щоденний час роботи на ЕОМ - 2 год, тому:

tвiд=60\*2=120год.

За даними обчислювального центру НУ "Львівська Політехніка" для ЕОМ типу IBM PC/AT SMг = 9 грн.

Отже:

К2= 9\*120 = 1080 грн

Таблиця 5.4

**Кошторис витрат на розробку програмного забезпечення**

|  |  |  |
| --- | --- | --- |
| № | Найменування елементів витрат | Сума витрат, грн. |
| 1 | Витрати на оплату праці | 7291,44 |
| 2 | Відрахування у спеціальні державні фонди | 2639,5 |
| 3 | Витрати на куповані вироби | 87,0 |
| 4 | Накладні витрати | 2187,43 |
| 5 | Інші витрати | 729,14 |
| 6 | Витрати на відлагодження і дослідну експлуатацію програмного забезпечення | 1080,0 |
|  | Всього | 14014,51 |

## **5.3. Вибір і обґрунтування аналога**

Для вирішення проектної задачі можуть бути використані декілька аналогів проектного рішення. Найбільш близькими за функціональними можливостями та призначенням може бути застонок TouchGraph Navigator вартістю близько 16 000 грн.

Провівши аналіз проектного рішення та його аналогів вибираємо наступні експлуатаційні показники для порівняння:

1. Показники призначення:

1.1 час проектування;

1.2 об'єм дискового простору;

1.3 Cross platform.

2. Показники надійності:

2.1 стійкість програми до некоректних дій користувача.

3. Ергономічні показники:

3.1 зручність інтерфейсу.

4. Естетичні показники:

4.1 Simple page usability.

5. Показники технологічності:

5.1 Швидкодію.

Таблиця 5.5.

**Визначення якості розроблюваного програмного забезпечення**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва показника | Одиниця | Числове значення показника | | Диференційна оцінка розроблюваного ПЗ |
|  | Виміру | проектне рішення | аналог |
| 1.Показники призначення |  |  |  |  |
| 1.1. Час проектування | год. | 1 | 1,5 | П |
| 1.2. Об'єм дискового простору | Мб | 90 | 50 | П |
| 1.3. Cross platform | % | 80 | 50 | П |
| 2. Показники надійності |  |  |  |  |
| 2.1. Стійкість програми до некоректних дій користувача | % | 80 | 65 | П |
| 3.Ергономічні показники |  |  |  |  |
| 3.1.Зручність інтерфейсу | Бали | 8 | 5 | В |
| 4. Естетичні показники |  |  |  |  |
| 4.1. Simple page usability | Бали | 10 | 6 | В |
| 5.Показники технологічності |  |  |  |  |
| 5.1. Швидкодія | % | 80 | 90 | В |

## **5.4. Визначення комплексного показника якості проектної розробки**

Комплексний показник рівня якості розроблюваного програмного забезпечення визначається порівнянням його показників з відповідними значеннями показників аналога за формулою:

 (5.9)

де аі- коефіцієнт вагомості і-го показника якості, що визначає його

відносну значущість у формуванні сукупної якості, визначається експертними методами з умови:

qі - показник рівня якості, визначений на основі одного часткового показника за формулами:

 (5.10)

або

 (5.11)

де Пі1 і Пі2 - числові значення і-го часткового показника якості відповідно аналога і розроблюваного програмного забезпечення.

Розрахунок за формулою (5.10) використовується, якщо збільшення числового значення показника відповідає покращенню якості.

Розрахунок за формулою (5.11) ведеться, якщо покращенню якості відповідає зменшення числового значення показників.

Таблиця 5.6.

**Визначення комплексного показника рівня якості проектного рішення за параметрами**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Назви показників | Одиниця вимірювання | Числові значення | | Частковий показник рівня якості  Показник якості | Коефіцієнт вагомості показника  вагомості | Показник  якості |
| Проектне рішення | Аналог |
| 1.Показники призначення |  |  |  |  |  |  |
| 1.1.Час проектування | год. | 1 | 1,5 | 1,5 | 0,2 | 0,3 |
| 1.2.Об'єм дискового простору | Мб | 90 | 50 | 1.8 | 0,2 | 0,36 |
| 1.3. Cross platform | % | 80 | 50 | 1.6 | 0,1 | 0,16 |
| 2.Показники надійності |  |  |  |  |  |  |
| 2.1. Стійкість програми до некоректних дій користувача | % | 80 | 65 | 1,2 | 0,2 | 0,24 |
| 3.Ергономічні показники |  |  |  |  |  |  |
| 3.1.Зручність інтерфейсу | Бали | 8 | 5 | 1.6 | 0.15 | 0,24 |
| 4.Естетичні показники |  |  |  |  |  |  |
| 4.1. Simple page usability | Бали | 10 | 6 | 1.6 | 0,2 | 0,33 |
| 5. Показники технологічності |  |  |  |  |  |  |
| 5.1. Швидкодія | % | 80 | 90 | 1.125 | 0,05 | 0,05 |
| Підсумок |  |  | — |  | 1 | 1,64 |

## **5.5. Визначення експлуатаційних витрат**

Експлуатаційні одноразові витрати програмного забезпечення і аналогу включають вартість підготовки даних і вартість машино-годин роботи ЕОМ (за час дії програми):

 (5.12)

де Еп - одноразові експлуатаційні витрати на проектне рішення (аналог), грн;

Е1п - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн;

Е2п - вартість машино-годин роботи ЕОМ для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати Веп визначаються за формулою:

 (5.13)

де Nп - періодичність експлуатації проектного рішення (аналогу), раз/рік.

Вартість підготовки даних для роботи на ЕОМ визначається за формулою:

 (5.14)

де 1 - номери категорій персоналу, який приймає участь у підготовці даних (1=1, 2, ... L);

n1, - чисельність співробітників 1-ої категорії, чол;

t1, - трудоємність роботи співробітників 1-ої категорії по підготовці

даних, год;

с1 — середнього динна ставка співробітника 1-ої категорії з врахуванням додаткової заробітної плати та відрахувань у спеціальні державні фонди, грн./год.

 (5.15)

де  - основна місячна заробітна плата працівника 1-ої категорії, грн.;

b - коефіцієнт, який враховує додаткову заробітну плату і відрахування у спеціальні державні фонди;

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: с° = 3200 грн. Тоді:

грн/год  
Трудоємність працівника по підготовці даних для проектного рішення складає 1 год, для аналога - 1,5 год.

Таблиця 5.7

**Розрахунок витрат на підготовку даних та реалізацію проектного рішення на ЕОМ**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Час роботи співробітників, год. | Середньогодинна заробітна плата, грн./год. | Витрати, грн. |
|  | Проектне рішення | | |
| 1 | 1 | 28,5 | 28,5 |
|  | Аналог | | |
| 1 | 1,5 | 28,5 | 42,75 |

Витрати на експлуатацію ЕОМ визначається за формулою:  
 (5.16)

де t - витрати машинного часу для реалізації проектного рішення (аналогу), год;

SMГ - вартість однієї машино-години роботи ЕОМ, грн/год.

Е2п = 1 \* 9 = 9 грн.

Е2а=1,5\*9 = 13,5 грн.

Еп=28,5+9=37,5 грн.

Еа=42,75+13,5 = 56,25 грн.

Веп= 37,5\*252=9450 грн.

Веа=56,25\*252=14175 грн.

## **5.6. Розрахунок ціни споживання проектного рішення**

Ціна споживання - це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

Цсп=Цп+Вепрv (5.17)

де Цп - ціна придбання проектного рішення, грн.:

 (5.18)

де Пр - норматив рентабельності;

Ко - витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, грн.;

Кк - витрати на доукомплектування технічних засобів на об'єкті, грн.;

Цп=14014,51\*(1+0,3) = 18218,863 грн.

Benpv - теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

 (5.19)

де Веп - річні експлуатаційні витрати, грн.;

T - строк служби проектного рішення, років;

R - річна ставка проценту банківського

 грн.

грн..

Цсп = 18218,863+7745,9=25964,76 грн.

Цса = 16000+11618,9=27618,9 грн.

## **5.7. Визначення показників економічної ефективності**

Коефіцієнт конкурентноздатності проектного рішення:

Ккз=Кяк\*Цса:Цсп

Ккз=1,64\*27618.9: 25964,76 =1,74

Економічний ефект в сфері проектування рішення:

Епр=Ца-Цп (5.21)

Епр =16000,0 – 18218,863= -2218,86 грн.

Річний економічний ефект в сфері експлуатації:

Екс=Веа-Веп

Екс=14175-9450=4725грн.

Додатковий економічний ефект у сфері експлуатації:



грн.

Сумарний ефект складає:

36569,31-2218,86 =34350,45 грн.

Таблиця 5.8

**Показники економічної ефективності проектного рішення**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування | Одиниці вимірювання | Значення показників | |
| Базовий варіант | Новий варіант |
| 1 | Капітальні вкладення | Грн. | - | 14014,51 |
| 2 | Ціна придбання | Грн. | 16000 | 18218,86 |
| 3 | Річні експлуатаційні витрати | Грн. | 14175 | 9450 |
| 4 | Ціна споживання | Грн. | 27618.9 | 25964,76 |
| 5 | Економічний ефект в сфері проектування | Грн. | - | -2218,86 |
| 6 | Економічний ефект в сфері експлуатації | Грн. | - | 4725 |
| 7 | Додатковий ефект в сфері експлуатації | Грн. | - |  |
| 8 | Коефіцієнт конкурентоздатності | Грн. | - | 1,74 |
| 9 | Сумарний ефект | Грн. | 34350,45 | |

## **5.8. Висновки**

В даному роздiлi була проведена економiчна характеристика програмного рiшення, що доводить можливiсть позитивних економiчних ефектiв. Для перевiрки цих ефектiв було розраховано витрати на розроблення i впровадження програмного рiшення, показник якостi та експлуатацiйнi витрати.

Розрахунки свiдчать про те, що розробка програмної системи є доцiльною, оскiльки забезпечується економiя капiталовкладень в розробку програмного продукту та значно покращується рiчний економiчний ефект, який отримується користувачами розробленої системи. Коефіцієнт конкурентноспроможності дорівнює 1,74.

Економiчна ефективнiсть розробки програми пiдтверджує її доцiльнiсть, адже значення додаткового економiчного ефекту в сферi експлуатацiї рiвне 36596,31 грн, а економiчного ефекту в сферi проектування – 2218,86 грн. Цiна придбання аналога становить 16 000 грн, а проектного рiшення — 18218,86 грн. Отримано додатній економічний ефект у розмірі 36569,31 грн, що обумовлений додатковим економічним ефектом у сфері експлуатації, значення якого, в свою чергу, спричинене меншими, у порівнянні з аналогом, витратами на підготовку даних та реалізацію проектного рішення на ЕОМ.  Отже, впровадження розробленої математичної моделі дозволяє економити значнi ресурси.

# **ВИСНОВКИ**

В даній магістерській кваліфікаційній роботі розроблено спеціалізовану програму аналізу груп індивідуумів у соціальній мережі. Програма реалізована у вигляді інтернет-сервісу, що дає можливість доступатись до неї з будь-якого пристрою. Збір даних проводиться з соціальної мережі Facebook, яка є найбільш популярна в наш час, це забезпечує якісну вибірку даних. Збір даних здійснюється без використання Facebook API, на відміну від аналогів.

Розглянуто технологію масштабного опрацювання даних MapReduce, для дослідження графових даних. На її основі розроблено програмне забезпечення для соціальних мереж та інтернет-ресурсів. Для масштабування використав проект Apache Hadoop.

Розроблено алгоритм, за допомогою якого проаналізував зв’язки між користувачами соціальної мережі Facebook. Відповідно до результатів дослідження, середня довжина шляху між двома будь-якими користувачами становить 4,118 «рукопотискань».

Також дослідження показало, що максимальна кількість користувачів мають по 120 друзів. Аналіз соціального графа має теж важливе значення. Він дає можливість виявити неформальні товариства, знайти слабкі місця, визначити лідерів і т. д.

Очікується, що подальше розширення сфери дослідження соціальних мереж, що обумовлено інтеграцією життя великої кількості людей з соціальними мережами.

Проведено розрахунок витрат на розробку проектного рішення. Здійснено порівняння з існуючим аналогом. Крім того, отримано додатній економічний ефект у розмірі 36569,31 грн. і тому розробка і впровадження цього проектного рішення є економічно доцільними.

# **Список використаних джерел**

1. Ритцер Джордж. Современные социологические теории. 5-е изд. / Джордж Ритцер. – СПб.: Питер, 2002. – 688 с.
2. Hanneman, Robert A. and Mark Riddle. Introduction to social network methods. – Riverside, CA: University of California, Riverside, 2005. – 322 с.
3. Newman M.E.J. The structure and function of complex networks // SIAM Review. 2003. – 256 с.
4. Tom White. Hadoop: The Definitive Guide, Third Edition.- O’Reilly, 2012.

– 686 с.

1. Уилсон Р. Введение в теорию графов. Пер. с. анг. -М.: Мир, 1977. – 208 с.
2. Чак Лєм. Hadoop в действии. - М.: ДМК Пресс, 2012. – 424 с.
3. Facebook. – [Режим доступу]: <https://facebook.com>.
4. Соціальна мережа – Wikipedia, the free encyclopedia [Режим доступу]: <https://uk.wikipedia.org/wiki/Соціальна_мережа>
5. Число Данбара: – Wikipedia, the free encyclopedia [Режим доступу]: <https://uk.wikipedia.org/wiki/Число_Данбара>
6. Борський М. С. Спеціалізована програма аналізу груп індивідуумів у соціальній мережі/ Борський М. С. // 73-а студентська науково-технічна конференція, секція «Комп’ютерна інженерія», підсекція «Спеціалізовані комп’ютерні системи», Львів, 16 жовтня 2015 р.: тези доп. / Нац. ун-т "Львів. політехніка". – Л. : Вид-во кафедри СКС, 2015 –C. 9-10.

# **ДОДАТКИ**

## **А. Лістинг програми**

|  |
| --- |
| com.maks.command.Action.class |
|  |

package com.maks.command;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public interface Action {

public void execut(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException;

public String getName();

}

|  |
| --- |
| com.maks.command.Menu.class |
|  |

package com.maks.command;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Menu {

private Map<String, Action> map;

private String action;

private Action task;

public Menu(Action... action) {

map = new HashMap<String, Action>();

for (Action item : action) {

map.put(item.getName(), item);

}

}

public void execute(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

action = request.getParameter("action");

task = map.get(action);

try{

task.execut(request, response);

}catch(NullPointerException e){

response.sendError(404);

return;

}

}

}

|  |
| --- |
| com.maks.contoller.GetJSONGraph.class |
|  |

package com.maks.contoller.graph;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.google.gson.JsonObject;

import com.maks.command.Action;

import com.maks.db.model.BasketCollection;

import com.maks.db.service.BasketCollectionService;

public class GetJSONGraph implements Action {

@Override

public void execut(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("application/json");

Integer id = Integer.parseInt(request.getParameter("basket"));

System.out.println(id);

BasketCollection basket = BasketCollectionService.getId(id);

System.out.println(basket);

ParsingFile parse = new ParsingFile();

JsonObject js = parse.getGraph(basket);

System.out.println(js);

response.getWriter().print(js);

}

@Override

public String getName() {

// TODO Auto-generated method stub

return "getGraph";

}

}

|  |
| --- |
| com.maks.contoller.GraphServlet.class |
|  |

package com.maks.contoller.graph;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.maks.command.Menu;

/\*\*

\* Servlet implementation class GraphServlet

\*/

@WebServlet("/GraphServlet")

public class GraphServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

Menu menu;

public GraphServlet() {

menu = new Menu(new IndexGraph(), new GetJSONGraph(), new ServiceGraph());

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

request.setCharacterEncoding("utf-8");

response.setCharacterEncoding("utf-8");

menu.execute(request, response);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

request.setCharacterEncoding("utf-8");

response.setCharacterEncoding("utf-8");

menu.execute(request, response);

}

}

|  |
| --- |
| com.maks.contoller.IndexGraph.class |
|  |

package com.maks.contoller.graph;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import com.maks.command.Action;

import com.maks.db.model.BasketCollection;

import com.maks.db.model.User;

import com.maks.db.service.BasketCollectionService;

public class IndexGraph implements Action {

@Override

public void execut(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

HttpSession session = request.getSession();

User user = (User) session.getAttribute("user");

List <BasketCollection> list = BasketCollectionService.getAllByUser(user.getId());

request.setAttribute("basketList", list);

request.getRequestDispatcher("WEB-INF/page/graph.jsp").forward(request, response);

}

@Override

public String getName() {

// TODO Auto-generated method stub

return "index";

}

}

|  |
| --- |
| com.maks.contoller.graph.ParsingFile.class |
|  |

package com.maks.contoller.graph;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Arrays;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Set;

import org.apache.commons.collections.map.HashedMap;

import com.google.gson.JsonArray;

import com.google.gson.JsonObject;

import com.maks.db.model.BasketCollection;

import com.maks.db.model.UserFB;

import com.maks.db.service.UserFBService;

public class ParsingFile{

private Set<String> node;

private Map<String, String> link;

private JsonObject jObject;

public ParsingFile() {

node = new HashSet<String>();

link = new HashMap<String, String>();

jObject = new JsonObject();

}

public JsonObject getGraph(BasketCollection basket){

File file = new File(basket.getPath()+"selectorPeople.txt");

JsonArray nodes = new JsonArray();

JsonArray links = new JsonArray();

try {

BufferedReader reader = new BufferedReader(new FileReader(file));

String line;

int id = 0;

while((line = reader.readLine())!=null){

String[] mas= line.split(",");

String first = mas[0];

if(isNode(first)){

UserFB user = UserFBService.getByIdFb(first);

JsonObject object1 = new JsonObject();

object1.addProperty("id", user.getIdFb());

object1.addProperty("name", user.getName());

object1.addProperty("link", user.getLink());

object1.addProperty("photo", user.getPhoto());

nodes.add(object1);

node.add(first);

}

for (int i = 1; i < mas.length; i++) {

UserFB user;

user = UserFBService.getByIdFb(mas[i]);

if(user!=null){

if(isNode(mas[i])){

JsonObject object1 = new JsonObject();

object1.addProperty("id", user.getIdFb());

object1.addProperty("name", user.getName());

object1.addProperty("link", user.getLink());

object1.addProperty("photo", user.getPhoto());

nodes.add(object1);

System.out.println(object1);

node.add(mas[i]);

}

if(isLink(first, mas[i])){

JsonObject objectLinks = new JsonObject();

objectLinks.addProperty("id", id);

objectLinks.addProperty("from", first);

objectLinks.addProperty("to", mas[i]);

links.add(objectLinks);

link.put(first, mas[i]);

id++;

}

}

}

}

jObject.add("nodes", nodes);

jObject.add("links", links);

} catch ( IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return jObject;

}

private boolean isNode(String id){

return !node.contains(id);

}

private boolean isLink(String id1, String id2){

boolean flag = true;

if(link.containsKey(id1)){

if(link.get(id1).equals(id2)){

flag = false;

}

}else if(link.containsKey(id2)){

if(link.get(id2).equals(id1)){

flag = false;

}

}

return flag;

}

}

|  |
| --- |
| com.maks.controller.collectiondata.CollectionIndex.class |
|  |

package com.maks.controller.collectiondata;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import com.maks.command.Action;

import com.maks.db.model.BasketCollection;

import com.maks.db.model.User;

import com.maks.db.service.BasketCollectionService;

public class CollectionIndex implements Action {

@Override

public void execut(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

HttpSession session = request.getSession();

User user = (User) session.getAttribute("user");

List<BasketCollection> list = BasketCollectionService.getAllByUser(user.getId());

request.setAttribute("basketList", list);

request.getRequestDispatcher("WEB-INF/page/collection.jsp").forward(request, response);

}

@Override

public String getName() {

// TODO Auto-generated method stub

return "index";

}

}

|  |
| --- |
| com.maks.controller.collectiondata.CollectionServlet.class |
|  |

package com.maks.controller.collectiondata;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.maks.command.Menu;

/\*\*

\* Servlet implementation class CollectionServlet

\*/

@WebServlet("/collection")

public class CollectionServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

Menu menu;

public CollectionServlet() {

menu = new Menu(new CollectionIndex(), new RunCollection());

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

menu.execute(request, response);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

menu.execute(request, response);

}

}

|  |
| --- |
| com.maks.controller.collectiondata.ParseFacebookUser.class |
|  |

package com.maks.controller.collectiondata;

import java.util.LinkedHashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.NoSuchElementException;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import com.maks.db.model.UserFB;

public class ParseFacebookUser {

private WebDriver driver;

private WebDriverWait wait;

private static int count;

public ParseFacebookUser( WebDriver driver, WebDriverWait wait) {

this.driver = driver;

this.wait = wait;

count = 0;

}

public Set<UserFB> getFriends(String id){

Set<UserFB> list = new LinkedHashSet<>();

try{

driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);

driver.manage().timeouts().setScriptTimeout(40, TimeUnit.SECONDS);

System.out.println("parse user " +id);

if(id.replaceAll("\\d\*", "").isEmpty()){

driver.get("https://www.facebook.com/profile.php?id="+id+"&sk=friends");

}else{

driver.get("https://www.facebook.com/"+id+"/friends?pnref=lhc");

}

wait.until(ExpectedConditions.visibilityOfElementLocated(By

.id("medley\_header\_friends")));

for (;;) {

try{

if( !driver.findElement(By.className("mbm")).getText().isEmpty()){

break;

}

}catch(NoSuchElementException e){

}

((JavascriptExecutor) driver).executeScript("window.scrollBy(0,5000)", ""); //y value '400' can be altered

try {

Thread.sleep(300);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

List<WebElement> frends = driver.findElements(By.className("\_698"));

for (WebElement webElement : frends) {

UserFB ufb = new UserFB();

ufb.setIdFb( parseLink( webElement.findElement(By.tagName("a")).getAttribute("href")));

ufb.setLink(webElement.findElement(By.tagName("a")).getAttribute("href"));

ufb.setName(webElement.findElement(By.className("fsl")).getText());

ufb.setPhoto("https://graph.facebook.com/"+getId(webElement.findElement(By.tagName("a")).getAttribute("data-hovercard"))+ "/picture?type=small");

list.add(ufb);

}

}catch(Exception e){

if(count<5){

count++;

list = getFriends(id);

}

count=0;

}

return list;

}

public static String parseLink(String href){

String id = null;

if(href.contains("id")){

Matcher matcher = Pattern.compile("\\d+").matcher(href);

if(matcher.find( ))

{

id = matcher.group();

}

}else{

Matcher matcher = Pattern.compile("[\\w|\\s|.]\*\\?").matcher(href);

if(matcher.find( ))

{

id = new StringBuffer(matcher.group()).reverse().replace(0, 1, "").reverse().toString();

}

}

return id;

}

public String getId(String link){

String id = null;

Matcher matcher = Pattern.compile("id=\\d\*").matcher(link);

if(matcher.find( ))

{

id = matcher.group().replace("id=", "");

}

return id;

}

}

|  |
| --- |
| com.maks.controller.collectiondata.RunCollection.class |
|  |

package com.maks.controller.collectiondata;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import com.maks.command.Action;

import com.maks.db.model.User;

public class RunCollection implements Action {

@Override

public void execut(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

long start = System.currentTimeMillis();

Integer circle = Integer.parseInt(request.getParameter("circle"));

HttpSession session = request.getSession();

User user = (User) session.getAttribute("user");

SeparateFasebooPage parse = new SeparateFasebooPage(user,circle,request);

System.out.println("time collection: "+(System.currentTimeMillis()-start)/1000);

}

@Override

public String getName() {

// TODO Auto-generated method stub

return "run";

}

}

|  |
| --- |
| com.maks.controller.collectiondata.SeparateFasebooPage.class |
|  |

package com.maks.controller.collectiondata;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.Writer;

import java.util.Date;

import java.util.HashSet;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import javax.servlet.http.HttpServletRequest;

import org.openqa.selenium.By;

import org.openqa.selenium.TimeoutException;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.firefox.FirefoxProfile;

import org.openqa.selenium.remote.CapabilityType;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import com.maks.db.dao.FriendsDAO;

import com.maks.db.model.BasketCollection;

import com.maks.db.model.Friends;

import com.maks.db.model.User;

import com.maks.db.model.UserFB;

import com.maks.db.service.BasketCollectionService;

import com.maks.db.service.FriendService;

import com.maks.db.service.UserFBService;

import com.maks.db.service.UserService;

public class SeparateFasebooPage {

private User user;

private Integer circle;

private Set<UserFB> friends;

private WebDriver driver;

private ParseFacebookUser run;

private HttpServletRequest request;

private String fileName;

private Set<String> unickId;

private int totalCount;

public SeparateFasebooPage(User user, Integer circle,

HttpServletRequest request) {

this.user = user;

this.circle = circle;

this.request = request;

unickId = new HashSet<String>();

mainSelector();

}

private void mainSelector() {

FirefoxProfile profile = new FirefoxProfile();

profile.setEnableNativeEvents(false);

profile.setPreference("permissions.default.image", 2);

profile.setPreference("webdriver.load.strategy", "unstable");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(CapabilityType.PAGE\_LOAD\_STRATEGY, "eager");

capabilities.setCapability(FirefoxDriver.PROFILE, profile);

driver = new FirefoxDriver(capabilities);

WebDriverWait wait = new WebDriverWait(driver, 10);

driver.manage().timeouts().pageLoadTimeout(40, TimeUnit.SECONDS);

driver.manage().timeouts().setScriptTimeout(40, TimeUnit.SECONDS);

try {

driver.get("https://www.facebook.com/");

wait.until(ExpectedConditions.visibilityOfElementLocated(By

.id("email")));

driver.findElement(By.id("email")).sendKeys("makski-23@mail.ru");

driver.findElement(By.id("pass")).sendKeys("maksym171993");

driver.findElement(By.id("u\_0\_x")).click();

wait.until(ExpectedConditions.visibilityOfElementLocated(By

.id("q")));

if(user.getIdFacebook()==null){

setIdFacebook(driver.findElement(By.className("\_2dpe")).getAttribute("href"));

}

} catch (TimeoutException e) {

driver.close();

mainSelector();

}

fileName = request.getServletContext().getRealPath("")

+File.separator+ "UserData" +File.separator+"\_" + user.getId()+new Date().getTime()

+File.separator;

run = new ParseFacebookUser(driver, wait);

friends = run.getFriends(user.getIdFacebook());

UserFB ufb = new UserFB();

ufb.setIdFb(user.getIdFacebook());

ufb.setName(user.getLastname()+" "+user.getFirstname());

ufb.setPhoto(user.getUrlPhoto());

UserFBService.add(ufb);

writeResaultFile(user.getIdFacebook(), friends, fileName);

writeTemporalFile(friends, 1);

unickId.add(user.getIdFacebook());

readFile();

BasketCollection basket = new BasketCollection();

basket.setCircle(circle);

basket.setDate(new Date());

basket.setPath(fileName);

basket.setUser(user);

basket.setTotalCount(totalCount);

System.out.println(basket);

BasketCollectionService.add(basket);

driver.close();

}

private void writeTemporalFile(Set<UserFB> setFriend, Integer iteration) {

try {

File file = new File(fileName+iteration+".txt");

Writer write = new FileWriter(file,true);

for (UserFB userFB : setFriend) {

if(userFB.getIdFb()!=null){

write.write(userFB.getIdFb() + "\n");

totalCount++;

}

}

write.flush();

write.close();

System.out.println(setFriend.size());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private void writeResaultFile(String id, Set<UserFB> setFriend,

String fileName) {

try {

File path = new File(fileName);

if (!path.exists()) {

path.mkdirs();

}

File file = new File(fileName+"selectorPeople.txt");

Writer write = new FileWriter(file,true);

write.write(id + ",");

UserFB ufb = UserFBService.getByIdFb(id);

for (UserFB userFB : setFriend) {

System.out.println(userFB);

UserFBService.add(userFB);

System.out.println(userFB);

write.write(userFB.getIdFb() + ",");

Friends friend = new Friends();

friend.setUfb(ufb);

userFB =UserFBService.getByIdFb(userFB.getIdFb());

friend.setFriend(userFB);

if(userFB==null){

}else{

FriendService.add(friend);

}

}

write.write("\n");

write.flush();

write.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private void readFile() {

for (int i = 1; i < circle; i++) {

try(BufferedReader read = new BufferedReader(new FileReader(new File(fileName+i+".txt")))) {

String id;

while((id= read.readLine())!=null){

if(!unickId.contains(id))

{

unickId.add(id);

friends= run.getFriends(id);

writeResaultFile(id, friends, fileName);

if(i==circle)

continue;

writeTemporalFile(friends, i+1);

}

}

} catch ( IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

private void setIdFacebook(String link){

String id= ParseFacebookUser.parseLink(link);

user.setIdFacebook(id);

UserService.update(user);

}

}

|  |
| --- |
| com.maks.db.dao.UserDAO.class |
|  |

package com.maks.db.dao;

import org.hibernate.Criteria;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

import org.hibernate.criterion.Restrictions;

import com.maks.db.model.User;

public class UserDAO {

private static SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();

public static void add(User user){

Session session = sessionFactory.openSession();

session.beginTransaction();

session.save(user);

session.getTransaction().commit();

session.close();

}

public static void update(User user){

Session session = sessionFactory.openSession();

session.beginTransaction();

session.update(user);

session.getTransaction().commit();

session.close();

}

public static void delete(User user){

Session session = sessionFactory.openSession();

session.beginTransaction();

session.delete(user);

session.getTransaction().commit();

session.close();

}

public static User get(Integer id){

Session session = sessionFactory.openSession();

session.beginTransaction();

User user = session.get(User.class, id);

session.getTransaction().commit();

session.close();

return user;

}

public static User getByEmail(String email){

Session session = sessionFactory.openSession();

session.beginTransaction();

User user =(User) session.createCriteria(User.class).add(Restrictions.eq("email", email)).uniqueResult();

session.getTransaction().commit();

session.close();

return user;

}

}

|  |
| --- |
| com.maks.db.dao.UserDAO.class |
|  |

package com.maks.db.model;

import java.util.Date;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.Temporal;

import javax.persistence.TemporalType;

@Entity

@Table(name="user")

public class User{

@Id

@GeneratedValue(strategy =GenerationType.AUTO)

private Integer id;

private String firstname;

private String lastname;

private String email;

private String location;

private String gender;

@Temporal(TemporalType.DATE)

private Date birthday;

private Integer age;

private String urlPhoto;

private String idFacebook;

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

public String getPhoto() {

return urlPhoto;

}

public void setPhoto(String urlPhoto) {

this.urlPhoto = urlPhoto;

}

public String getLocation() {

return location;

}

public void setLocation(String location) {

this.location = location;

}

public String getUrlPhoto() {

return urlPhoto;

}

public void setUrlPhoto(String urlPhoto) {

this.urlPhoto = urlPhoto;

}

public String getIdFacebook() {

return idFacebook;

}

public void setIdFacebook(String idFacebook) {

this.idFacebook = idFacebook;

}

@Override

public String toString() {

return "User [id=" + id + ", firstname=" + firstname + ", lastname="

+ lastname + ", email=" + email + ", location=" + location

+ ", gender=" + gender + ", birthday=" + birthday + ", age="

+ age + ", urlPhoto=" + urlPhoto + ", idFacebook=" + idFacebook

+ "]";

}

}