Зміст

[ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ 3](#_Toc11201364)

[ВСТУП 4](#_Toc11201365)

[1. Аналіз систем збору інформацію з різних джерел 6](#_Toc11201366)

[1.1 Аналіз сучасних агрегаторів інформації про спортивні події 11](#_Toc11201367)

[1.2 Аналіз особливостей сайтів-агрегаторів 14](#_Toc11201368)

[2. Аналіз засобів реалізації системи агрегації 17](#_Toc11201369)

[2.2 Типова архітектура системи агрегації 20](#_Toc11201370)

[2.3 Вибір методології розробки 21](#_Toc11201371)

[2.4 Парадигми програмування 23](#_Toc11201372)

[2.5 Вибір мови програмування і основних засобів розробки 24](#_Toc11201373)

[2.6 Фреймворки і бібліотеки, які необхідні для розробки системи 27](#_Toc11201374)

[2.6.1 Spring Framework 27](#_Toc11201375)

[2.6.2 Spring Boot 28](#_Toc11201376)

[2.6.3 Spring MVC 30](#_Toc11201377)

[2.6.4 Spring Data 31](#_Toc11201378)

[2.7 Вибір бази даних 32](#_Toc11201379)

[2.8 Синтаксичний аналізатор JSOUP 35](#_Toc11201380)

[2.9 Інструменти для тестування 35](#_Toc11201381)

[2.9.1 Фреймворк для тестування JUnit 35](#_Toc11201382)

[2.9.2 Інструмент для тестування API Postman 36](#_Toc11201383)

[2.10 Генератор HTML документів 37](#_Toc11201384)

[3. Реалізація системи агрегації 39](#_Toc11201385)

[3.1 Розробка модулів серверної частини системи агрегації 39](#_Toc11201386)

[3.1.1 Модуль з синтаксичними аналізаторами. 40](#_Toc11201387)

[3.1.2 Модуль з базою даних 44](#_Toc11201388)

[3.1.3 Веб-сервер 46](#_Toc11201389)

[4. Опис інтерфейсу та інструкція користування 50](#_Toc11201390)

[5. Висновок 55](#_Toc11201391)

[СПИСОК ЛІТЕРАТУРИ 57](#_Toc11201392)

# ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application programming interface) – прикладний програмний інтерфейс.

XML (Extensible Markup Language). – розширювана мова розмітки.

SQL (Structured query language) — декларативна мова програмування для взаємодії користувача з базами даних.

HTML (Hypertext Markup Language) - стандартна [мова розмітки](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D1%80%D0%BE%D0%B7%D0%BC%D1%96%D1%82%D0%BA%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) для створення [веб-сторінок](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D0%BE%D1%80%D1%96%D0%BD%D0%BA%D0%B0) і [веб-додатків](https://uk.wikipedia.org/w/index.php?title=%D0%92%D0%B5%D0%B1-%D0%B4%D0%BE%D0%B4%D0%B0%D1%82%D0%BA%D1%96%D0%B2&action=edit&redlink=1).

JVM (Java Virtual Machine) – віртуальна машина Java, частина середовища виконання Java, виконує інтерпретацію Java байт-коду та його оптимізацію.

ООП (Об’єктно орієнтоване програмування) - парадигма програмування, що базується на представленні програми у вигляді сукупності об’єктів, кожний з яких є екземпляром класу і виконує якусь певну функцію.

MVC (Model-View-Controller) - шаблон розділення даних додатку, користувацького інтерфейсу і керуючої логіки на три окремі компоненти: модель, представлення і контролер таким чином, що модифікація кожного компонента може відбуватись незалежно.

IoC (Injection of Control) – принцип ООП, який використовується для зменшення  в зменшення зв’язків у комп’ютерних програмах.

TDD (test-driven development)

SQL бази даних -

NoSQL бази даних –

DAO (Data Access Object) –

CRUD (Create Read Update Delete) –

HTTP (Hyper Text Transfer Protocol) -

# ВСТУП

Багато людей цікавиться спортивною індустрією і присвячують більшість свого часу на огляд та аналіз спортивних подій. Це хобі може приносити не лише задоволення, але й матеріальні прибутки. Охочі заробляти на своїх знаннях і займаючись своїм хобі часто приймають участь у букмекерській діяльності.

Букмекерська діяльність відома ще з часів Римської імперії. Місцем їх локації були арени, де проводилися гладіаторські бої. Цей процес називався тоталізатором.

Тоталізатор - це гра на результатах яких-небудь подій. Найчастіше тоталізатори орієнтуються на спортивні події, оскільки вони відбуваються часто і багато людей цікавиться спортом. Перед початком учасник може тільки здогадуватись про суму виграшу у випадку свого успіху. Учасники ставлять свої ставки перед подією. Виграють лише ті люди, які поставили на результат, що стане відомим після події. Виграш становить всю суму, що поставили учасники тоталізатора мінус відсоток який отримує організатор, а решта поділяється порівну між всіма переможцями.

Наступниками тоталізаторів є букмекерські контори, які мають дещо змінену систему ставок. Тоталізатори не роблять жодних передбачень і аналізів можливого результату, вони просто приймають ставки і виплачують виграші, тобто вони є посередниками. А букмекерські контори зобов’язані проводити аналіз можливого результату і враховувати ймовірність та різні фактори, які можуть впливати на нього. Букмекерські контори перед початком або під час події озвучує коефіцієнти на різні варіанти результату події. Учасники орієнтуються на свої аналітичні знання, інтуїцію або віру в певні зроблені ставки. Ті особи, шо поставили ставку на результат який відбувся, отримують виграш, що складається з початкової ставки помноженої на коефіцієнт. Учасники, які помилилися втрачають свою ставку і не отримують жодного виграшу.

На сьогодні, є багато популярних букмекерських контор таких як: Pari Match, 1xBet, MarathonBet і т. д. Кожна з цих установ надає платформу, де кожен бажаючий може робити ставки на різні події: спортивні, культурні, політичні. Вони змушені проводити статистичний аналіз на різні події, щоб приваблювати користувачів високими коефіцієнтами і бути прибутковою організацією. Тому, кожна така структура має певні індивідуальні алгоритми для обрахунку сприятливих коефіцієнтів, тобто в них відрізняються показники на події і їх результати.

Звичайному користувачеві було б зручніше користуватися певною платформою, яка б надавала хорошу аналітику і інформацію з букмекерських контор в одному місці. Аналітична система є актуальною, оскільки, має за мету спростити користувачеві вибір правильної платформи для участі у букмекерській діяльності, надавши можливість ознайомитись за короткий період часу з коефіцієнтами на різні результати подій на різних платформах, у зручному форматі.

Мета даної роботи полягає у створенні системи агрегації та порівняння ставок на спортивні події.

Для досягнення цієї мети необхідно реалізувати такі задачі:

1. Проаналізувати відповідні аналогічні системи, їх особливості та принципи функціонування.
2. Виконати проектування системи та спланувати роботу.
3. Визначитися із засобами проектування та реалізації системи.
4. Надати зручний інтерфейс для користувача.

Розробити інформаційну систему, яка буде виконувати такі функції:

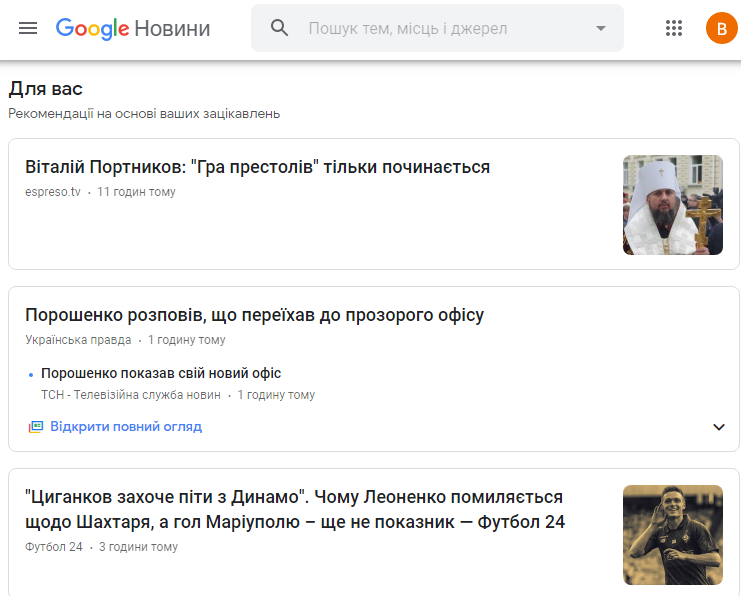
* збирає з різних букмекерських сайтів коефіцієнти ставок на спортивні події;
* порівнювати коефіцієнти з різних джерел і підсвічувати який коефіцієнт вигідніший на якому сайту
* збирати статистику учасників події і відображати її на сайті;

# Аналіз систем збору інформацію з різних джерел

Сайти агрегатори – це сайти, які збирають інформацію з різних сервісів, сортують і аналізують і відображають на якійсь одній платформі, най частіше це веб сайт. Основною метою таких сайтів є забезпечення зручного способу доступу до інформації з різних сайтів схожої тематики для користувачів, де вони можуть порівняти ціни на товари, якість послуг, і обрати те, що їм більше потрібно, не витрачаючи багато часу на пошуки інформацію на різних ресурсах.

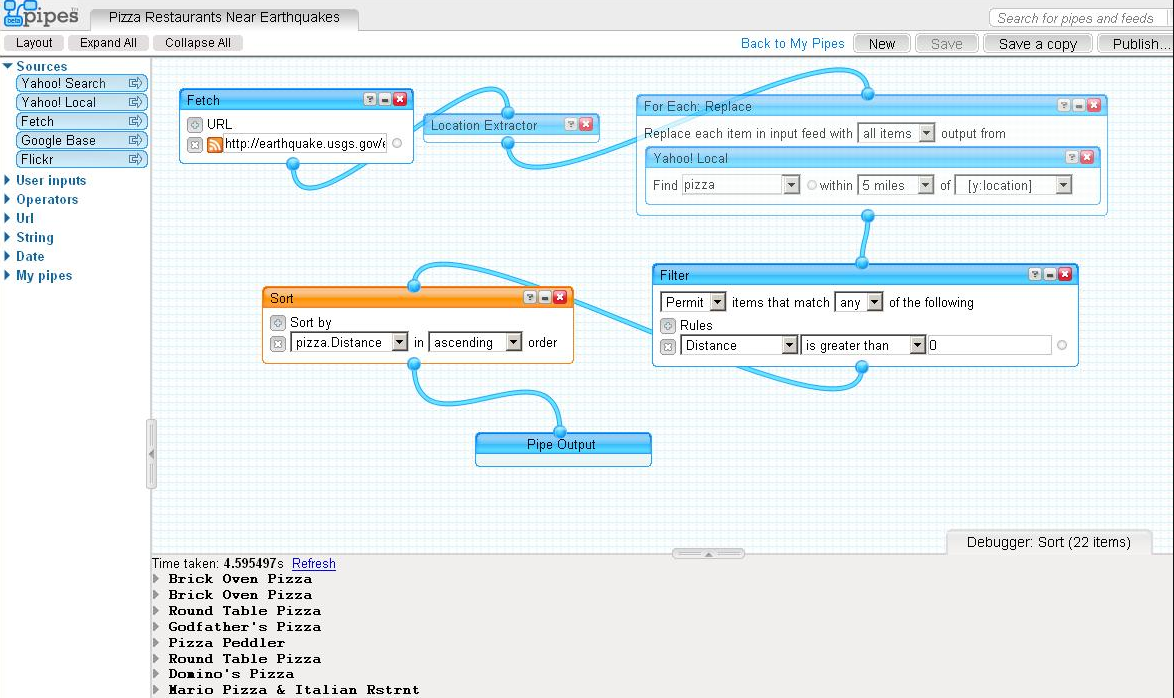
Агрегатори можна поділити за типом агрегації на такі види:

* **Користувацький агрегатор** – найбільш популярний і відомий тип, що об’єднує дані із декількох джерел, відображаючи це на зручному графічному інтерфейсі. Яскравим прикладом є більшість додатків від Google. Найпопулярнішими їх продуктами є: Google News та Google Maps (рис. 1.1)



*Рис. 1.1 Користувацький агрегатор Google News*

* **Агрегатори даних** – об’єднують близькі за типом дані з різних джерел. Найпопулярнішим таким агрегатором такого типу був Yahoo! Pipes але з 30 вересня 2015 року цей сервіс закритий (рис 1.2). Користувачі використовували графічний інтерфейс для створення додатків, які об’єднували новини, веб-сторінки та інші сервіси. Сайт надавав можливість отримувати інформацію з різних джерел і використовувати критерії за контролем отриманого контенту.



*Рис. 1.2 Агрегатор даних Yahoo! Pipes*

* **Бізнес агрегатор** – це комбінація користувацьких додатків і баз даних для агрегації і представлення інформації, розширення функціональності сервісу для взаємодії з користувачем (додавання параметрів, редагування). Кінцевий результат можна використовувати як бізнес додаток.

За типом сайти агрегатори поділяються так:

* **Сайт агрегатор новин** – це сервіси, які збирають новини з різних джерел інформації, відображають їх на своїй платформі та класифікують їх за певними категоріями

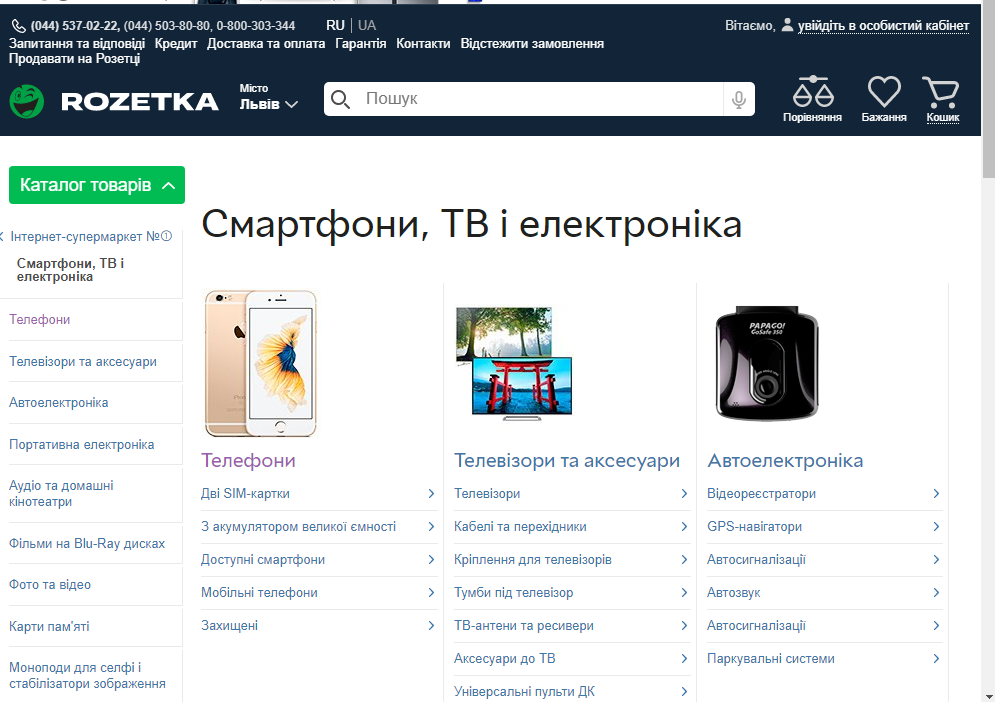
Джерелами інформації для таких сайтів можуть бути:

* Новинні сайти, на яких розміщені статті, замітки і інтерв’ю.
* Інформаційні стрічки, що надають конкретну і суху інформацію.
* Соціальні мережі.
* **Сайт агрегатор товарів** – це спеціальний ресурс, займається збором даних про різну продукцію. Завдяки цим сайтам користувачам дуже зручно у процесі вибору потрібного товару. Такі майданчики мають характеристики і ціни товарів. Користувачі можуть порівнювати товари за цими характеристиками і обирати потрібну продукцію серед аналогів за найбільш прийнятну ціну.
* **Платіжний агрегатор**– це сервіс, який займається збором всіх онлайн коштів, що надійшли на рахунок сайту, і далі на рахунок замовника. Саме ця система дозволяє працювати не тільки з готівковими платежами, але й з такими популярними платіжними системами як Visa або Mastercard, а також з різними віртуальними валютами. Це єдина система, яка дозволяє легко і швидко упорядкувати електронні розрахунки, а також забезпечити легальний обіг електронних грошей.
* **Сайт-агрегатор купонів** – це сайт, який спеціалізується на зборі інформації про актуальні акції, розпродажі, знижки у магазинах, кафе, різних розважальних центрах, що надають купони на свої послуги. На цих сайтах можна купити купони на безкоштовні замовлення або знижки. Сайтами такого типу найчастіше користуються особи, які роблять велику кількість покупок і часто ходять в різноманітні розважальні місця.
* **Агрегатор послуг** – це площадка, де зібрано структуровані дані про окремі напрямки ринку. Агрегатор, зазвичай, містить інформацію про виробників, постачальників, послуги і продукти одного або декількох суміжних напрямків.

Типові сервіси агрегатора послуг:

* Швидкий і розширений пошук.
* Перевірка підрядників.
* Системи арбітражу (допомога вирішення спірних ситуацій між сторонами угоди).
* “Безпечні угоди” (резервування коштів замовника до виконання всіх умов замовлення).

В Україні найпопулярнішим таким сервісом є Rozetka.ua (рис. 1.3)

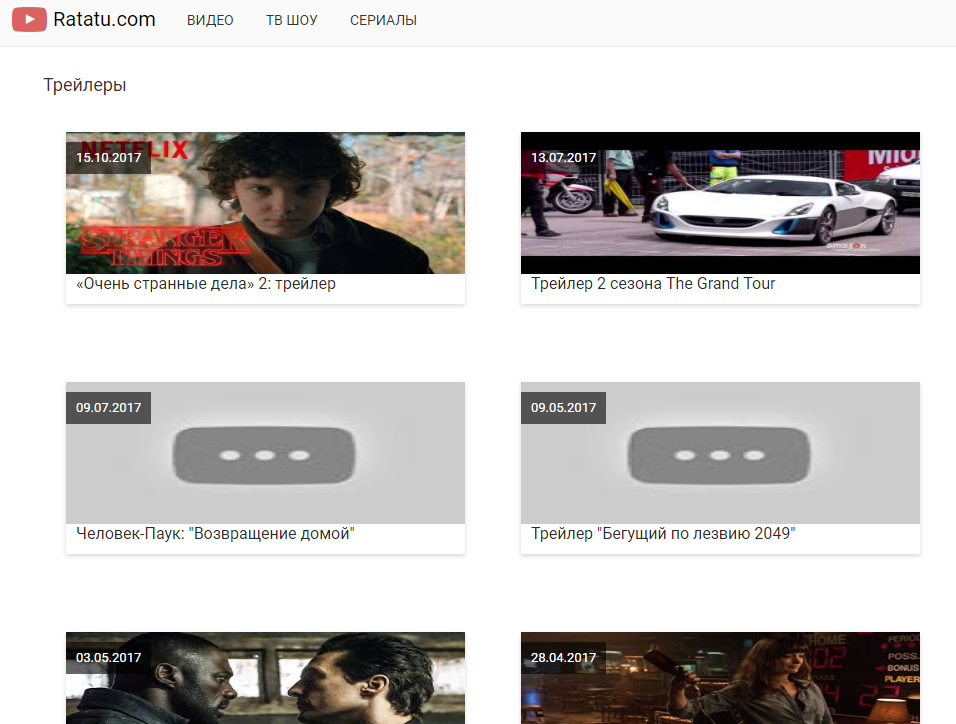


*Рис. 1.3 Агрегатор послуг Rozetka.ua*

* **Агрегатори каршерінга** – сервіси, які завантажують від операторів інформацію про наявність вільних машин і відображають їх на одній карті. Це оренда автомобіля, яка передбачає як по-хвилинне використання, так і більш тривале. Для оренди авто не потрібно кожного разу ходити в офіс і підписувати договори: автомобілі просто знаходяться на вулицях. Достатньо прив’язати смартфон до банківської картки, з якої буде зніматися оплата.

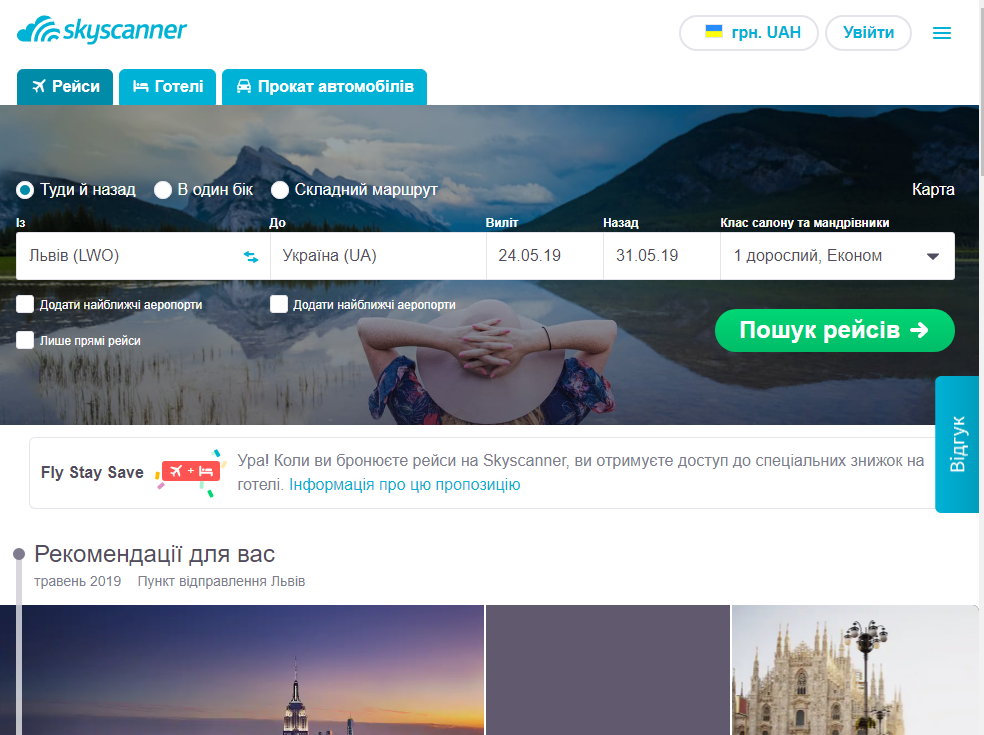
**Приклади сайтів-агрегаторів:**

**Ratatu.com** - зручний відеосервіс, не потребує реєстрації, ним може скористуватись кожен бажаючий. При завантаженні головної сторінки відображається блог з новинами. Користувач має доступ до найбільш цікавих подій з областей спорту, науки, розваг, інновацій у технологіях і т.д. Сервіс потрібен для тих, хто не хоче витрачати багато часу на налаштування і оптимізацію цікавого контенту і хоче без зусиль отримувати інформацію про найновіші події, які трапилися у світі. (рис. 1.4)



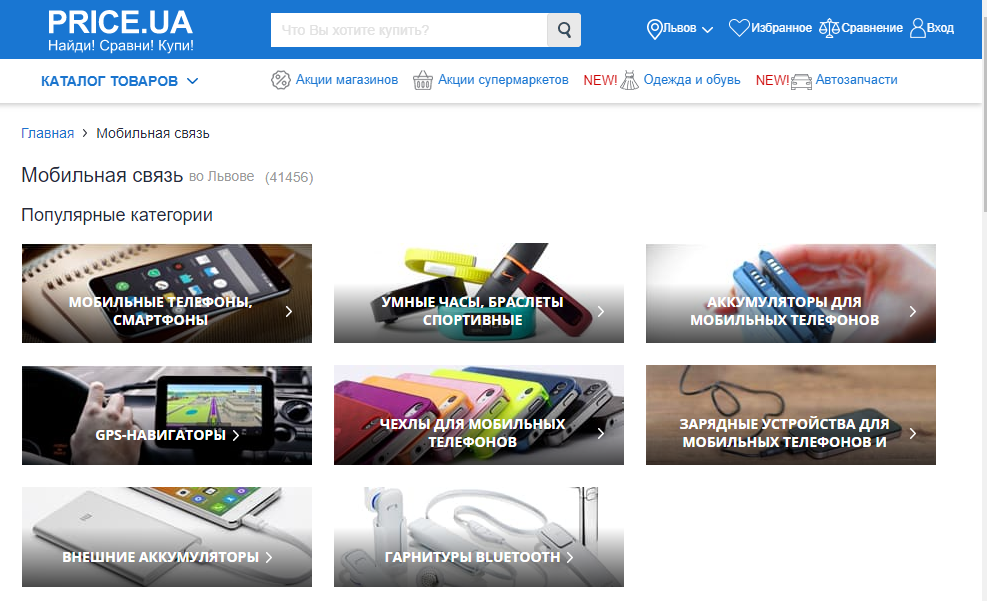
*Рис. 1.4 Агрегатор відео Ratatu.com*

**Scyscanner.ua** – агрегатор авіаквитків, готелів, прокат автомобілів (рис. 1.5). На сайті є можливість зручно і швидко знайти квиток на політ за найбільш привабливу ціну. Присутній аналіз коли ціни на квитки можуть найменшу ціну.



*Рис.1.5 Агрегатор квитків Scascanner.ua*

**Price.ua** – це один з найпопулярніших сервісів порівняння цін в Україні (рис. 1.6). Користувачі можуть порівнювати ціни на товари, підписуватися на сповіщення, які будуть повідомляти про змінення цін. Є можливість залишити або переглянути відгуки про товари, дізнатися детальні характеристики товара і його опис.



*Рис.1.6 Агрегатор товарів і цін Price.ua*

## Аналіз сучасних агрегаторів інформації про спортивні події

На сьогодні існує чимало програмних засобів, які створені для агрегації та порівняння ставок на спортивні події.

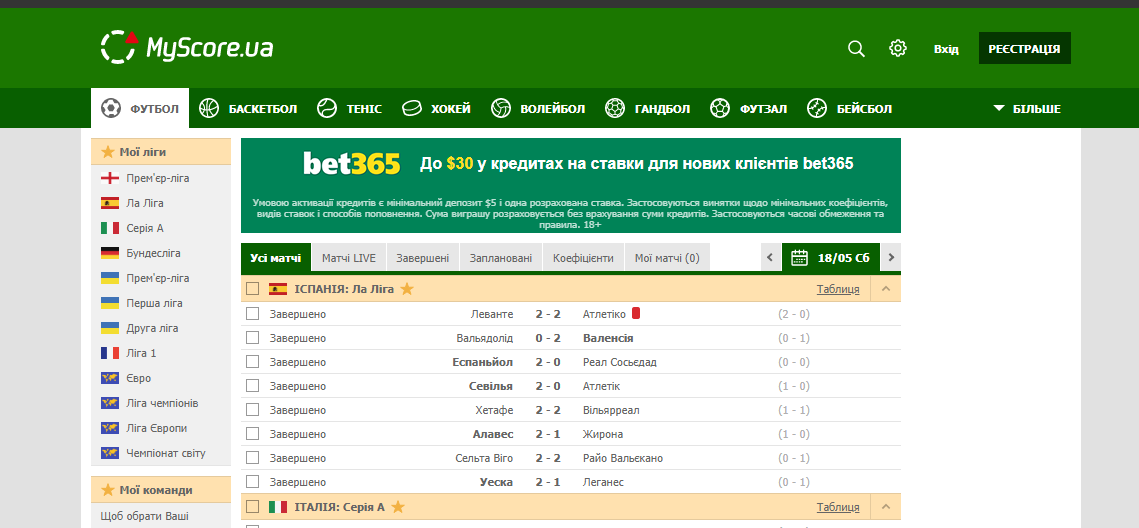
Основні функції, які мають бути у цій системі такі:

* Наведення деталей спортивних подій.
* Наведення статистики учасників спортивних подій.
* Зручний інтерфейс.
* Наведення коефіцієнтів з популярних букмекерських контор.
* Відображення результатів у режимі реального часу.
* Наведення інформації про різні види спорту.

Нижче наведено приклади таких засобів та проведено їх аналіз.

*1.1.1 MyScore*

*MyScore* – це веб сайт, де користувач в реальному часі може побачити результати спортивних матчів. Проаналізувати статистику команд, які беруть участь у події. Даний сайт відображає такі види спорту як футбол, волейбол, теніс, баскетбол і т.д. Як виглядає інтерфейс сайту можна побачити на рис.1.7



*Рис. 1.7 Інтерейс сайту Myscore*

*Переваги:*

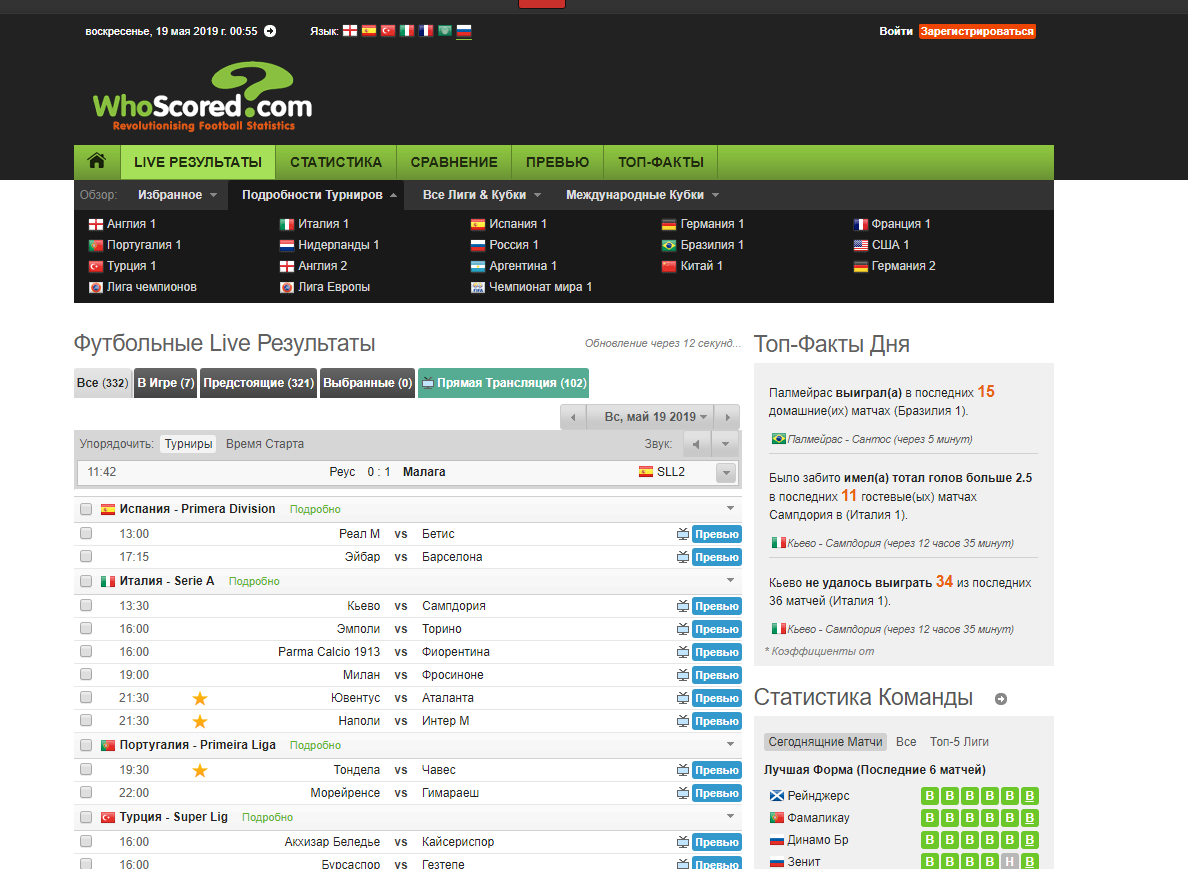
* Містить достатньо інформації про результати спортивних подій.
* Присутня мінімальна статистика команд.
* Є можливість вибору мови сторінки.
* У більшість видів спорту можна моніторити матчі у режимі реального часу.

*Недоліки:*

* Такий вид спорту як кіберспорт не відображається у реальному часі, тому результати на сайті висвітлюються із затримкою.
* Присутні помилки відображення інформації.

*1.1.2 WhoScored*

*WhoScored* – це ресурс, який містить детальну інформацію про кожен футбольний матч. Проводиться аналітика після кожного матчу, можна побачити оцінку ефективності кожного гравця. Будь-який досвідчений учасник букмекерської діяльності має знати і вміти використовувати цей ресурс для аналізу статистики і прогнозування результатів. Інтерфейс цього ресурсу зображено на рис. 1.8.



*Рис. 1.8 Інтефейс сервісу WhoScored*

*Переваги:*

* Детальний аналіз футбольних матчів.
* Детальна статистика ігор та гравців.
* Є можливість вибору мови сторінки.
* Статистика відображається у режимі реального часу.

*Недоліки:*

* Присутня інформація тільки про футбольні матчі.
* Відсутність україномовного інтерфейсу.
* Відсутність коефіцієнтів з букмекерських контор.

*1.1.3 BetVirus*

*BetVirus –* надає інформацію про минулі ігри, ймовірність результату у відсотках, графіках, таблицях, статистику ліг, коефіцієнти і рух у різних букмекерських конторах. Інтерфейс цього ресурсу зображено на рис. 1.9.



*Рис. 1.9 Інтерфейс сервісу BetVirus*

*Переваги:*

* Надає ймовірність результату у відсотках.
* Присутня статистика матчів.

*Недоліки:*

* Мала кількість спортивних ліг
* Присутні помилки відображення інформації

## Аналіз особливостей сайтів-агрегаторів

За своєю архітектурою сайт агрегатор дуже нагадує портал. Адже це і є сайт-портал, який протягом еволюції отримав певний набір унікальних функцій.

Головна функція агрегатора - це можливість підтягувати інформацію зі сторонніх сайтів. Саме ця функція допомагає в роботі багатьом компаніям з нерухомості, адже вони можуть розміщувати свої бази нерухомості пакетами прямо в агрегатор. Наприклад, турагенства, які можуть підтягувати інформацію про тури безпосередньо через ресурси тур-операторів. Або інтернет-магазини, які потребують збірки товарів.

**Переваги:**

1. Порівняння даних.

Сайти агрегатори надають інформацію яка може відрізнятися на різних ресурсах. Користувачі можуть швидко переглянути її використовуючи зручний інтерфейс, зробити аналіз, і опираючись на аналіз зробити найбільш вигідне рішення для себе.

1. Економія часу.

Користувачам не потрібно витрачати на пошуки інформації на різних ресурсах багато часу, оскільки все, що їх цікавить може знаходитись в одному місці.

1. Широкий асортимент

Ці сервіси надають багато видів інформації, що збирається з багатьох вузько спеціалізованих сервісів і відображається на одному графічному інтерфейсі. Це спрощує аналіз даної інформації, пришвидшує прийняття рішення користувача, який користується даною системою.

1. Доступ до інформації.

Часто такі сервіси мають великі масиви даних, тому розробляється зручний API для доступу до неї. Це API може бути корисне для інших розробників, що хочуть розробити власні системи, які будуть використовувати цю інформацію для подальшого аналізу і надання її користувачеві. Чим багатший буде функціонал системи тим більш буде задоволений користувач.

**Недоліки:**

1. Використання реклами.

Для розвитку і підтримування функціональності таким системам потрібні фінансові ресурси. Оскільки доступ до них вільний і ці системи не проводять жодних фінансових операцій, то з’являється потреба використовувати рекламу для заробітку.

1. Монетизація послуг.

Ці системи надають користувачу тільки інформацію про цікаві об’єкту, але вони не можуть проводити якісь фінансові операції. Можливість оплачувати певні послуги на сайтах, з яких ці системи збирають інформацію.

1. Залежність від API ресурсів.

Розробники агрегаційної системи завжди мають уважно слідкувати за джерелами інформації, бо найменша зміна інтерфейсу, API може призвести до некоректного стану роботи .

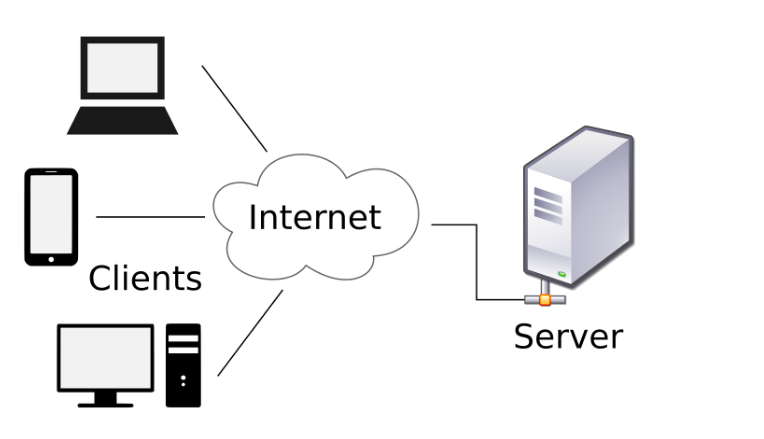
**Висновки:**

Проведено аналітичний огляд агрегаційних систем. В результаті проведеного дослідження та аналізу наявних систем агрегації коефіцієнтів на результати спортивних подій, можна зробити висновок, що більшість систем не надає пропозицій клієнту за певними критеріями, що враховані у розробленій системі.

# 2. Аналіз засобів реалізації системи агрегації

* 1. **Архітектура системи**

Клієнт-серверна архітектура набула популярності з розвитком інтернету (рис. 2.1).



*Рис. 2.2 Архітектура клієнт-сервер*

Клієнт-серверна архітектура є спільною архітектурою системи, де розділено рівні клієнт-сервер. Представляє централізовану ресурсну систему, де сервер має всі ресурси, отримує численні запити для обміну ресурсами з клієнтами, які робили запит. Клієнт і сервер можуть перебувати в одній мережі. Сервер є стабільним і масштабованим для повернення відповідей до клієнтів. Ця архітектура є сервіс-орієнтованою, що означає, обслуговування клієнтів не буде перерване. Клієнт-серверну архітектуру підпорядковує мережний трафік, відповідаючи на запити клієнтів, а не повну передачу файлів. Вона відтворює файловий сервер з базою даних сервера.

Клієнтські комп'ютери реалізують зв’язок, щоб дозволити користувачу комп'ютера запитувати послуги сервера та відображати результати, які повертає сервер. Сервери очікують запитів, які надходять від клієнтів, а потім повертають їх. Сервер, зазвичай, надає стандартизований простий інтерфейс для клієнтів, щоб уникнути апаратної та програмної несумісності. Ця архітектура корисна, якщо клієнт і сервер мають різні завдання, які вони регулярно виконують. Отримати інформацію від сервера можуть багато клієнтів одночасно.

Оскільки, програма–сервер може приймати запити від багатьох програм-клієнтів, її розміщають на спеціально виділену обчислювальну машину, налаштовану спеціальним способом, тому продуктивність цієї машини має бути високою. Через особливі ролі такої машини в мережі, специфіку її налаштування і програмного забезпечення, її називають сервером, а машини, які виконують клієнтські програми, відповідно, клієнтами.

**Переваги:**

* Відсутність дублюючого коду на клієнтах і серверах.
* Оскільки більшість опрацювань даних виконується на сервері, то вимоги до комп’ютерів клієнтів зменшуються.
* Всі дані зберігаються на сервері, який як правило краще захищений ніж комп’ютери клієнтів. На сервері простіше організувати контроль повноважень, щоб дозволити доступ до даних тільки клієнтам з відповідними правами доступу.

**Недоліки:**

* Неробочий сервер може зробити неробочою всю обчислювальну мережу. Неробочим сервером варто вважати, ресурсів якого не бракує для обслуговування всіх клієнтів, а також сервер який знаходиться на ремонті, профілактиці і т.д.
* Підтримка роботи даної системи вимагає окремого спеціаліста – системного адміністратора.
* Висока ціна обладнання.

Клієнт-серверна архітектура поділяється на 2 види двохрівневу і трьохрівневу.

**Двохрівнева архітектура** забезпечує взаємодію тільки двох програмних компонентів – клієнта і сервера. Двохрівнева поділяється ще на 2 види за типом клієнта.

Тонкий клієнт – вся логіка обробки даних відбувається на стороні сервера, а клієнтський компонент відповідає тільки за відображення даних.

Товстий клієнт – сервер лише керує постачанням даних, а обробка їх і формування відображення відбувається на стороні клієнта.

**Трьохрівнева архітектура** поділяється на 3 частини, а саме: презентаційний рівень (клієнтський рівень), рівень застосунку (бізнес-рівень) та рівень бази даних (рівень даних).

У клієнт-серверних системах, використовується додаткове проміжне програмного забезпечення, що означає, що запит клієнта йде на сервер через який середній рівень і відповідь сервера йде спочатку до проміжного рівня, а потім до клієнта. Ця архітектура захищає двохрівневу архітектуру і надає кращу продуктивність. Система обходиться дорого, але вона проста у використанні. Ідея проміжного рівня полягає в постановці баз даних, черги, виконання додатків, покращує гнучкість і забезпечує кращу продуктивність і т.д.

Архітектура збільшує продуктивність завдяки практиці рентабельним інтерфейсом користувача, удосконалених сховищ даних, розширенню підключень служб безпеки. Оскільки, ці 3 рівні розділені, то зміни в кожному з них ніяк не вплинуть на інші. Тому для реалізації системи агрегації обрано трьохрівневу клієнт-серверну архітектуру для розробки продукту.

## Типова архітектура системи агрегації

Типову архітектуру систем агрегація наведено на рис.2.2



*Рис. 2.2 Типова архітектура систем агрегації*

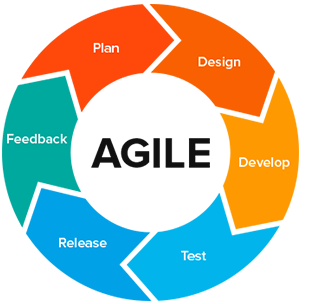
Модулі, що використовуються у цій архітектурі:

* Джерела інформації – зовнішні ресурси, з яких агрегатор збирає інформацію.
* Сервіс з синтаксичними аналізаторами – модуль, у якому знаходяться сервіси, які збирають інформацію з зовнішніх джерел. Найчастіше, вони використовують API, яке надають джерела. Оскільки багато ресурсів не надають публічний доступ до своєї інформації, то необхідно створювати власні інструменти для аналізу і видобування інформації, шляхом аналізу документів HTML, XML. Часто розробляється такими мовами програмування як JavaScript, Java, C#.
* База даних – це організована збірка даних, яка, зазвичай, зберігається та доступна в електронному вигляді з комп'ютерної системи. Якщо бази даних є більш складними, вони часто розробляються з використанням формальних методів проектування та моделювання. Найчастіше використовуються такі типи баз даних: SQL (MySql, PosgreSql, SQL Server), та NoSql (MongoDB, Neo4J, Redis)
* Веб-сервер – програмне забезпечення, яке контролює доступ веб-користувачів до розміщених на сервері файлів, відповідає на HTTP-запити користувачів і повертає їм результати. Часто реалізується такими інструментами: java(Spring Framework), C#(ASP.NET), Python(Django), PHP.

## Вибір методології розробки

 Agile – це серія підходів розробки програмного забезпечення, орієнтованих на використання ітеративної розробки, динамічне формування вимог та забезпечення їх реалізації в результаті постійного взаємозв’язку всередині робочих груп, які об’єднують спеціалістів різної кваліфікації.

Схему роботи цієї методології наведено на Рис 2.3.



*Рис 2.3 Agile процес*

Основні принципи методології Agile:

1. Вирішальне значення для налаштування продукту має споживач і степінь його залучення у створення продукту.

2. Для прийняття рішення команди мають бути високоефективними та дружніми.

3. Поетапна та циклічна робота становить основу процесу. Проект розділяється на чавстини, які мають бути завершені до закінчення певного терміну.

4. Фокус оцінки результативності скерований на періодичні представлення проміжних станів проекту.

5. Для розробки проекту в основу береться закон Парето, який каже що потрібно прикладати 20% зусиль , щоб отримати 80% ефективності. Що дозволяє не виконувати кожний окремий цикл до досконалості перед представленням результати клієнтові. Продукт вдосконалюється кожної ітерації.

6. Потрібно завершити попередній етап перед початком наступного.

Найбільш популярна версія Agile є Scrum методологія.

Scrum – встановлює правила управління процесом розробки і дозволяє використовувати існуючі практики кодування, коректуючи вимоги або включаючи тактичні зміни. Використання даної методики дозволяє виявляти і уникати відхилення від бажаного результату на більш ранніх етапах розробки програмного продукту.

Основні елементи Scrum:

* Product backlog – передбачає формування списку вимог, створеному по єдиному шаблону та відсортованому по пріоритетах.Якщо вимог немає, то проект завершується.
* Sprint backlog – вимоги до виконання найближчого спринту (етапу). Розділена на завдання без можливості додавати нові вимоги під час теперішнього спринту.
* Sprint Goal – мета спринту, яка має бути досягнута до кінця певного терміну. Тривалість одного спринту може тривати від 1 тижня до 1 місяця.
* Sprint BurnDown Chart – діаграма. яка обновлюється по завершенню завдань. По ній можна побачити динаміку розробки проекту та ефективність команди.

Для розробки системи агрегації обрано методологію Scrum.

## Парадигми програмування

Найбільш популярна парадигма програмування є ООП.

Парадигма програмування базується на представленні програми у вигляді сукупності об’єктів. Кожний з об’єктів є екземпляром класу і виконує певну функцію.

Основні принципи ООП:

* Абстракція – для виділення в предметі важливого для вирішення конкретного завдання створюються абстракції, які пізніше описуються в класах.
* Інкапсуляція – для швидкого і безпечного доступу до даних та операцій, використовуючи спеціальні модифікатори доступу, які обмежують видимість методів та полів відповідно до модифікатора. Дозволяють уникнути накопичення непотрібних зв’язків у коді.
* Наслідування - організації ієрархії об’єктів, спільні особливості, яких можна винести у клас-предок, а специфічний функціонал реалізовувати окремо. Дозволяє уникнути повторення коду та спростити розробку продукту.
* Поліморфізм – особливість системи, яке надає можливість використовувати об’єкти з однаковими інтерфейсами без інформації про тип і структуру об’єкта..

**Принципи програмування** – це принципи, які використовуються для проектування програм та написання коду у певному стилі. Ці стилі допомагають описати певні техніки, що можна використовувати для написання коду, який буде легко читати, розширювати або змінювати у майбутньому.

SOLID – це набір принципів, які дозволяють написати програми, що можуть динамічно розвиватися, розростати функціоналом та й просто забезпечує зручне читання програми іншими розробниками.

S (Single responsibility) – це принцип єдиного обов’язку, тобто клас має лише один обов’язок. Це дозволяє уникнути ситуацій коли щоб змінити одну операцію потрібно переписувати багато коду.

O (Open-closed) – об’єкти повинні бути відкриті для розширення функціональності та закриті для змін. Цей принцип реалізовується унаслідуванням класів та реалізацією інтерфейсів.

L (Liskov substitution) – сутності програми можна замінювати їх нащадками без переписування коду.

I (Interface segregation) – принцип який описує, що краще мати кілька схожих класів, які виконують власні специфічні функції, а не один великий універсальний клас.

D (Dependency inversion) - Залежності в системі мають базуватися на абстракціях. Абстракції не мають залежати від деталей реалізації їх, а навпаки деталі залежать від абстракцій. Тобто модулі вищих рівнів не залежать від модулів нижніх рівнів.

## Вибір мови програмування і основних засобів розробки

Для розробки системи обрано мову програмування Java. Ця мова тривалий час утримує лідируючі місця з популярності мов світу. Має велику спільноту, яка в будь який момент може допомогти з вирішенням проблем. Для розробки програм на Java написано багато різноманітних інструментів, які спрощують розробку.

Особливостями цієї мови програмування є:

1. Використання віртуальної машини для запуску програм

Це одна з основних характеристик Java. Написаний програмістом код компілюється у байт-код, який передається у JVM (Java Virtual Machine). Ця процедура надає можливість мати кроссплатформений код. Тобто програміст пише код для будь якої зручної для нього операційної системи і може запустити цей код на інших ОС. Наприклад, код написаний на мові С++ на ОС Windows, при запуску на Linux може викликати проблеми. Це одна з основних переваг мови, а також і недолік, оскільки код спочатку має запуститись у віртуальній машині, а потім транслюватись у машинний код. Це призводить до більших витрат часу для виконання програми.

1. Строго типізована мова.

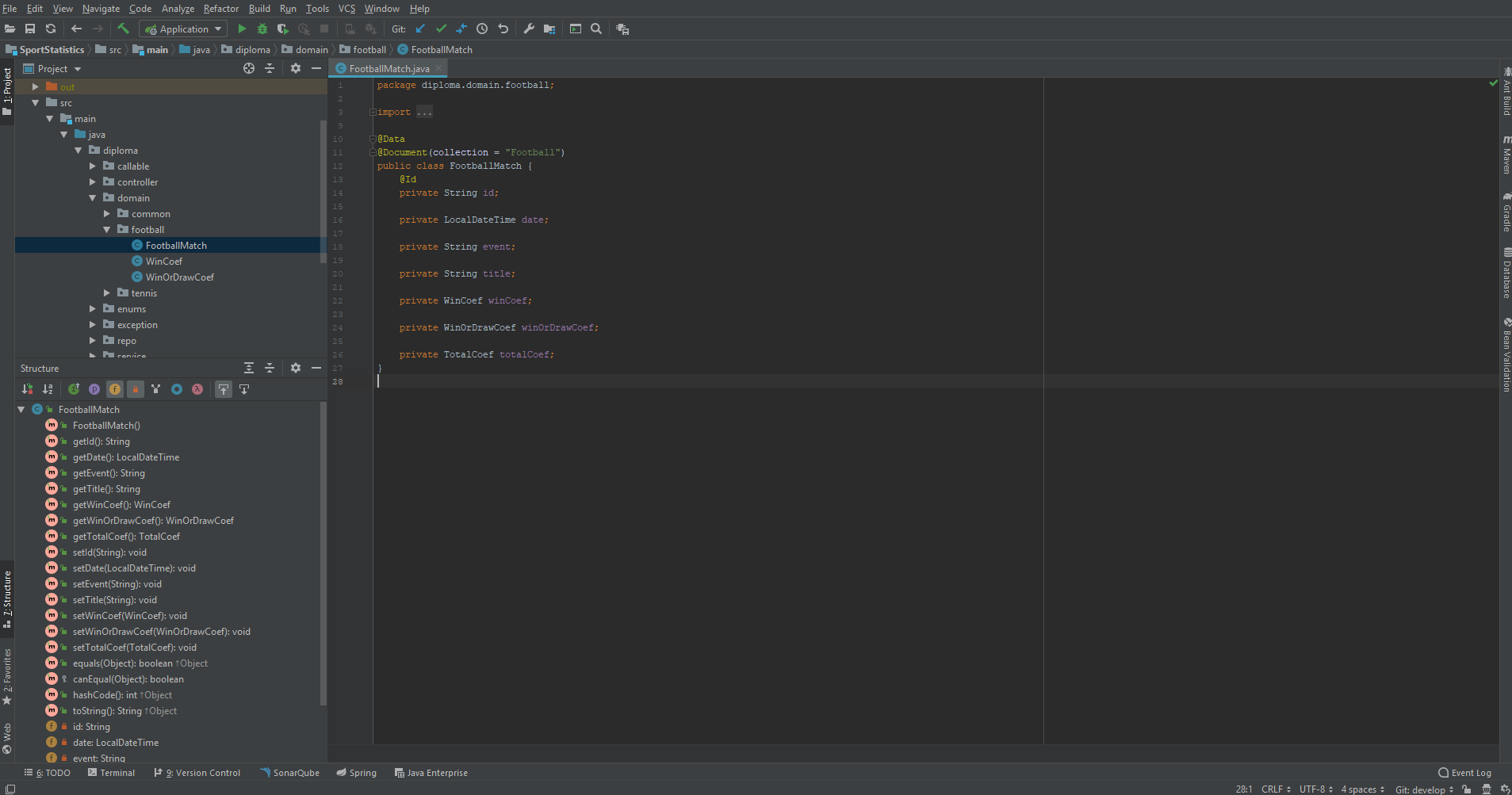
Ця особливість вказує на те, що кожна змінна має мати вказаний при ініціалізації тип. Це дозволяє зручно читати і підтримувати код у робочому стані. Не відбувається жодних не передбачуваних ситуацій, оскільки всі інструкції програміст вказує заздалегідь.

1. Реалізація ООП (об’єктно орієнтоване програмування).
2. Використання автоматичного збирача сміття.

У Java використовується автоматичний збирач сміття. Розробник створює об’єкти, а JVM відповідає за очищення пам’яті від не потрібних об’єктів. Це стається, якщо на нього немає посилань. Але ця функція має і недоліки, оскільки якщо бракує пам’яті для роботи програми, тоді запускається цей збирач сміття і програма може зависнути на кілька секунд, що інколи може мати погані наслідки.

**Середовище розробки:**

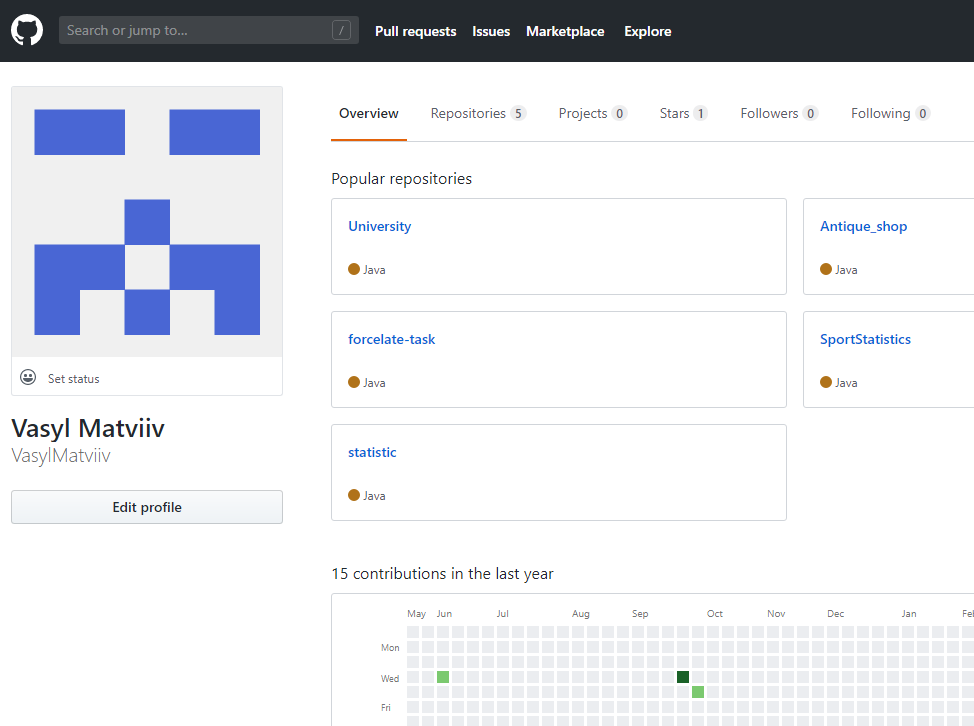
Для розробки системи обрано середовище IntelliJ IDEA (рис. 2.4). Дозволяє написання програм на різних мовах програмування, таких як Java, Python, Scala, PHP і т.д. Ця система сумісна з багатьма популярними інструментами розробників, такими як Maven, Gradle, GIT і т.д.



*Рис. 2.4 Середовище розробки IntelliJ IDEA*

**Система контролю версій**

Для контролю версій обрано систему GitHub. Це веб сервіс, який використовують для хостингу IT-проектів і їх спільної розробки (рис. 2.5).



*Рис. 2.5 Система контролю версій GitHub*

Функції GitHub:

* Розміщення коду.
* Можливість коментувати і коментувати виправлення коду.
* Об’єднання репозиторіїв.
* Відслідковування помилок.
* На сайті можна переглянути файли проектів з підсвіткою синтаксису.

**Система автоматичної збірки**

**Gradle –** система автоматичної збірки, яка пришвидшує розробку продукту, тим що завантажує різні бібліотеки, фреймворки. До моменту, як з’явилися системи автоматичної збірки потрібно було вручну прописувати в консолі всі залежності і шлях, звідки потрібно завантажувати бібліотеки. Програмісту потрібно тільки написати сценарій на мові Groovyі запустити його.

## Фреймворки і бібліотеки, які необхідні для розробки системи

### Spring Framework

Платформа Spring - популярна платформа додатків з відкритим кодом, що призначена для спрощення розробки на J2EE. Вона складається з контейнера, платформи управління елементами і набору інтегрованих служб для веб-інтерфейсів користувача, транзакцій і збереження стану. До складу платформи Spring Web Входить MVC - розширювана платформа MVC для створення веб-додатків [1].

Не дивлячись на те, що Spring Framework не надавав конкретної моделі програмування, він є широко поширеним серед Java-програмістів, головним чином як альтернатива і заміна Enterprise JavaBeans. Spring Framework забезпечує великим вибором Java-розробників в проектуванні. Він надає детальну документацію і прості у використанні засоби вирішення проблем, що виникають при створенні корпоративних додатків.

Найкорисніші переваги, які надає Spring Framework.

* Spring дозволяє розробникам розробляти корпоративні додатки з використанням POJOs. Використання тільки POJOs (Plain Old Java Object не вимагає продуктів EJB контейнера, такі як Application Server, але можна використовувати тільки надійні сервлет контейнери, такі як Tomcat або деякі комерційні продукти.
* Spring реалізований через модульну архітектуру. Розробник може використовувати тільки ті модулі, які йому потрібні і не завантажувати пам’ять не потрібними налаштуваннями, бібліотеками та класами.
* Spring не створює свої продукти, якщо є аналогічні випробувані часом інструменти, такі, як ORM фреймворки, засоби логування, JEE та інші, а надає зручний інтерфейс для долучення цих технологій у свій продукт.
* Тестування програм, написаних з використанням Spring є простим, оскільки в цей фреймворк переміщається код, який залежний від середовища. Крім того, через використання POJOs стає простіше використовувати ін’єкцію залежностей для постачання тестових даних.
* Spring надає зручне API для передачі специфічних повідомлень про помилки для різних технологій, таких як JDBC, Hibernate, JDO і т.д.
* Легкий IoC (Injection of Control) контейнер для зберігання об’єктів порівняно з EJB контейнером. Це корисно для розробки і розгортання програм на комп’ютерах з малою пам’яттю і CPU ресурсами.
* Spring надає послідовний інтерфейс керування транзакціями, який можна масштабувати до локальної транзакції (використовуючи одну базу даних) або до глобальних транзакцій (за допомогою JTA).
* Spring спрощує ініціалізацію і налаштування компонентів додатку, не роблячи істотних змін Java-коду. Ці налаштування можна написати у файлі xml, у спеціальному класі, написаному на Java і використовуючи анотації.

Технологія Spring є найбільш відома через інєкції залежностей (Dependency Injection - DI), що є різновидом інверсії контролю (IoC). IoC є загальною концепцією, і вона може бути виражена в багатьох різних напрямках. Ін'єкції залежностей є лише прикладом Інверсія контролю.

При написанні складних Java-додатків, класи повинні бути як найбільш незалежними від інших класів Java. Це збільшує можливість повторного використання цих класів і тестування їх незалежно від інших класів. Ін'єкції залежностей допомагають об’єднати ці класи разом і в той же час тримати їх незалежними. Компоненти можна додавати і видаляти (майже) незалежно один від одного. Також Spring спрощує тестування: в об’єкт, розроблений для роботи в IoC контейнері дуже легко втілювати тестові залежності і перевірити роботу тільки цього об’єкта.

### Spring Boot

**Spring Boot** — це корисний проект, метою якого є спрощення створення додатків на базі Spring. Він дозволяє найбільш простим способом створити веб-додаток, вимагаючи від розробника мінімум зусиль з його налаштування і написання коду.

Особливості Spring Boot.

* Простота керування залежностями.

Щоб прискорити процес управління залежностями, Spring Boot неявно упаковує необхідні сторонні залежності для кожного типу додатку на основі Spring і надає їх розробнику за допомогою так званих starter-пакетів (spring-boot-starter-data-jpa ,spring-boot-starter-web, і т . д.)

Starter-пакети є набором зручних дескрипторів залежностей, які можна долучити у свій додаток. Це дозволить отримати універсальне рішення для всіх технологій, пов'язаних зі Spring, позбавляючи програміста від зайвого пошуку прикладів коду та завантаження з них необхідних дескрипторів залежностей.

* Автоматична конфігурація.

Після вибору відповідного стартового пакету, Spring Boot спробує автоматично налаштувати Spring додаток, відповідно до JAR залежностей, які були додані.

Наприклад, якщо додати Spring-boot-starter-web, то Spring Boot автоматично налаштує такі зареєстровані компоненти, як ResourceHandlers, DispatcherServlet, MessageSource.

Автоматична конфігурація може бути повністю змінена в будь який момент за допомогою користувацьких налаштувань.

* Вбудована підтримка сервера додатків – контейнера сервлетів.

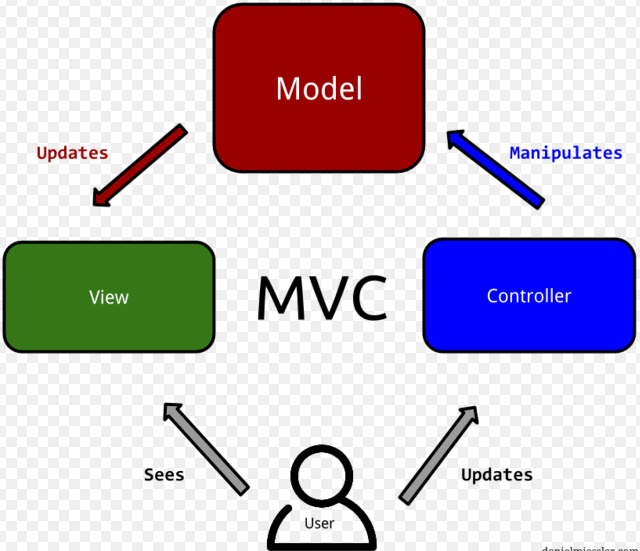
Тепер не потрібно турбуватися про налаштування контейнера сервлетів і розгортання програми на ньому. Додаток може запустити себе, як виконуваний файл JAR за допомогою вбудованого сервера.

Якщо потрібно використовувати окремий HTTP-сервер, достатньо відімкнути залежності за замовчуванням. Spring Boot надає окремі стартові пакети для різних HTTP-серверів.

Створення автономних веб-додатків за допомогою вбудованих серверів є не лише зручним для розробки, але і є дійсним рішенням для корпоративного рівня додатків і стає все більш корисним у світі мікросервісів. Можливість швидко упакувати весь сервіс (наприклад, аутентифікації користувача) в автономному режимі і повністю мобільному артефакті, який також надає API - робить установку і розгортання додатків значно простіше.

### **Spring MVC**

**Spring MVC** використовує архітектуру Модель-Відображення-Контролер (MVC) і готові компоненти, що використовуються для розробки гнучких і слабко зв’язаних веб-додатків. Патерн MVC забезпечує поділ на 3 різні рівні абстракції, що забезпечує слабкий зв’язок між ними (рис. 2.6) .



*рис. 2.6 Архітектура патерну MVC*

1. **Модель** містить дані додатку, які обробляються і надсилаються до клієнта, в основному це об’єкти POJO.
2. **Відображення** відповідає за відображення даних у клієнта, за їх оформлення і тип подачі. В основному це генерування HTML документів, які надсилаються до клієнта.
3. **Контролер** відповідає за обробку запитів користувачів і надсилання до них результатів.

Основним компонентом Spring MVC є DispatcherServlet, який обробляє і надсилає всі HTTP-запити. Порядок обробки запиту в DispatcherServlet зображено на рис. 2.7.



*Рис. 2.7 Обробка запиту в Spring Web MVC DispatcherServlet*

Нижче описано послідовність дій обробки запиту HTTP до DispatcherServlet:

1. Після надходження HTTP запиту, DispatcherServlet звертається до об’єкта HandlerMapping, щоб визначити який **Контролер** має прийняти запит.
2. **Контролер** отримує запит і викликає відповідний обслуговуючий метод використовуючи GET або POST метод. Метод отримує дані **Моделі**, які базуються на визначеній бізнес логіці і повертає в DispatcherServlet ім’я **Відображення**.
3. За допомогою ViewResolver DispatcherServlet визначає, яке **Відображення** потрібно використовувати, базуючись на отримане ім’я.
4. Після того, як **Відображення** створено, DispatcherServlet надсилає дані **Моделі** у вигляді атрибутів у **Відображення**, яке відображається у браузері клієнта.

### Spring Data

Один з основних модулів Spring Framework. Що надає зручний інтерфейс для використання баз даних. Для використання потрібно створювати POJO об’єкти на які будуть проектуватися дані з таблиць бази даних. Модуль найчастіше використовується з шаблоном DAO ( Data Access Object), який реалізує необхідний для роботи з джерелом даних механізму доступу. Використовуючи DAO бізнес-компоненти працюють з більш простим інтерфейсом. Він повністю приховує деталі реалізації джерела даних від клієнта. Дозволяє без переписування коду змінювати бази даних.

Spring Data надає можливість розробнику спростити написання коду для зв’язку до бази даних. Програміст може створювати інтерфейси, які успадковуються від інтерфейсу JPARepositoriy, в якому описані основно CRUD (CREATE READ UPDATE DELET) операції. Надає механізм створення методів для доступу до бази даних не реалізовуючи їх. Розробник лише має у власному інферейси створити ці методи з ім’ям, яке відповідає певним правилам. Цей модуль створює SQL операції аналізуючи назви цих методів. Для досвідчених розробників надає можливість прописувати вланоруч SQL операції над певними методам і Spring буде їх реалізовувати.

## Вибір бази даних

**База даних** (БД)– це програмне забезпечення, яке дозволяє зберігати великі масиви різних типів даних. Кожна база даних має певну модель (реляційна, документно-орієнтована, графова), яка надає зручний доступ до даних. Система управління базами даних (СУБД) – спеціальне програмне забезпечення для управління базами даних різних форм і розмірів.

Перед вибором типу БД, потрібно відповісти на кілька основних питань.

1. Чи потрібна чітка структура даних?

Дані на сайтах можуть змінюватися дуже швидко, тобто, кожного кварталу може змінювати відображення коефіцієнтів, додаватися нові показники, система має бути готова до динамічної зміни структури. Щоб не переписувати всі запити, а просто врахувати зміни і щоб система працювала коректно і безперебійно. Тому, гнучкість бази даних є дуже важливою.

1. Як часто будуть обновлятися дані?

Агрегатор має реагувати на будь-які зміни, які будуть відбуватися на джерелах інформації. Ці зміни особливо часто будуть траплятися при відображенні результату матчів які відбуваються у режимі реального часу. Тому, швидкість обновлення даних є дуже важлива.

1. Чи потрібен реляційний зв’язок?

Дані не пов’язані між собою, з кожного сайту надходять незалежні дані, які потрібно обробляти по різному. Тому, реляційний зв’язок не є важливим.

1. Яка кількість сутностей потрібна для відображення повноти інформації?

Кількість сутностей буде дорівнювати кількості сайтів помноженої на кількість підтримуваних видів спорту. Тобто, структура бази даних буде зростати дуже швидко, і сутності не будуть між собою зв’язаними.

1. Чи потрібна можливість масштабувати базу даних?

Якщо буде збільшуватись кількість підтримуваних сайтів і користувачів, тоді буде потреба нарощувати. Тому, потрібно передбачити можливість горизонтального масштабування, оскільки воно є більш орієнтоване на майбутні витрати.

Аналізуючи критерії, які було описано вище, обрано для порівняння 2 види бази даних (SQL, NoSQL ) і по одному представнику з кожної з них (MySQL, MongoDB).

SQL бази даних - це реляційні бази даних, які використовують (Structured Query Language, SQL) для визначення і опрацювання даних. З одного боку, це відкриває великі можливості для розробки. SQL одна з найбільш поширеніших мов запитів, що дозволяє уникнути багатьох ризиків і буде зручною, якщо потрібно працювати з комплексними запитами. Але є кілька обмежень, наприклад, при побудові запитів потрібно мати чітку структуру даних, тобто, при зміні якоїсь таблиці знадобиться переписувати більшість запитів і перезаписувати старі дані, щоб вони були коректними для нової таблиці.

NoSQL бази даних– це не реляційні бази даних, що мають динамічну структуру даних, які можуть зберігатися декількома способами: орієнтованими на колонки, документно-орієнтованими, у виді графів.

NoSQL база даних надає наступні переваги:

* Можете створювати документи, не задаючи їх структуру заздалегідь.
* Кожний документ може мати власну структуру.
* Присутня можливість додавати і редагувати поля під час роботи з базою даних.

Масштабування.

В більшості випадків SQL бази даних є вертикально масштабованими, тобто, можна збільшувати завантаження окремого сервера, нарощуючи потужність процесорів, збільшувати об’єми ОЗП або системи збереження даних. А NoSQL бази даних розраховані на горизонтальне масштабування, тобто, збільшуючи кількість серверів можна збільшити максимальний трафік. Для вертикального масштабування потрібно весь час обновлювати апаратне забезпечення, а при горизонтальному масштабуванні достатньо купити новий сервер і його налаштувати. Тобто, для маcштабування краще підходять нереляційні бази даних.

Переваги MySQL бази даних.

Надійність. Дуже розвинута база даних, що означає велику спільноту користувачів, багато прикладів і фахівців, які можуть допомогти з помилками.

* Сумісність. MySQL доступна на всіх основних платформах: Linux, Windows, Mac.
* Окупність: MySQL поширюється з відкритими кодом, і знаходиться у вільному доступі.

Переваги MongoDB.

* Динамічна схема. Ця СУБД дозволяє гнучко працювати зі схемою даних без необхідності змінювати самі дані.
* Масштабованість. MongoDB є горизонтально масштабованою, що дозволяє легко зменшити навантаження на сервера при великих обсягах даних.
* Швидкість. Висока продуктивність при виконанні простих запитів.

Отже, проаналізувавши ці бази даних, можна прийти висновку що для даної системи агрегації даних краще підходить MongoDB, оскільки вона дешевше масштабується, краще реагує на зміну схеми даних і швидше опрацьовує прості запити.

## Синтаксичний аналізатор JSOUP

Щоб збирати з сайтів інформацію про спортивні ставки і події потрібний інструмент, який зможе робити синтаксичний аналіз HTML сторінки, для цієї операції обрано бібліотеку Jsoup.

Основні функції, які надає бібліотека Jsoup:

* Отримувати і розбирати HTML із URL, файлу або стрічки.
* Знаходити і витягувати інформацію, використовуючи DOM-навігацію або CSS-селектори.
* Маніпулювати HTML елементами, атрибутами, текстом.
* Формувати HTML код.

В процесі роботи, JSOUP створює синтаксичне дерево, котрим можна в майбутньому маніпулювати.

## Інструменти для тестування

### Фреймворк для тестування JUnit

Сьогодні стає популярною техніка TDD (test-driven development), яка наголошує на тому, що потрібно спочатку написати тест на функціонал і тоді реалізовувати цей функціонал. Це призводить до того, що розробник в тесті має передбачити всі можливі ситуації, помилки, поведінку користувача, що допомагає написати логіку, яка буде містити всі ці фактори. Більшість коду буде покрита тестами. Це зменшує ризик на помилки програми і дозволяє проектувати чітку архітектуру, яка буде легко підтримуватися і тестуватися.

Для тестування обрано Junit. Цей фреймворком є з відкритим вихідним код, який розроблений для написання тестів.

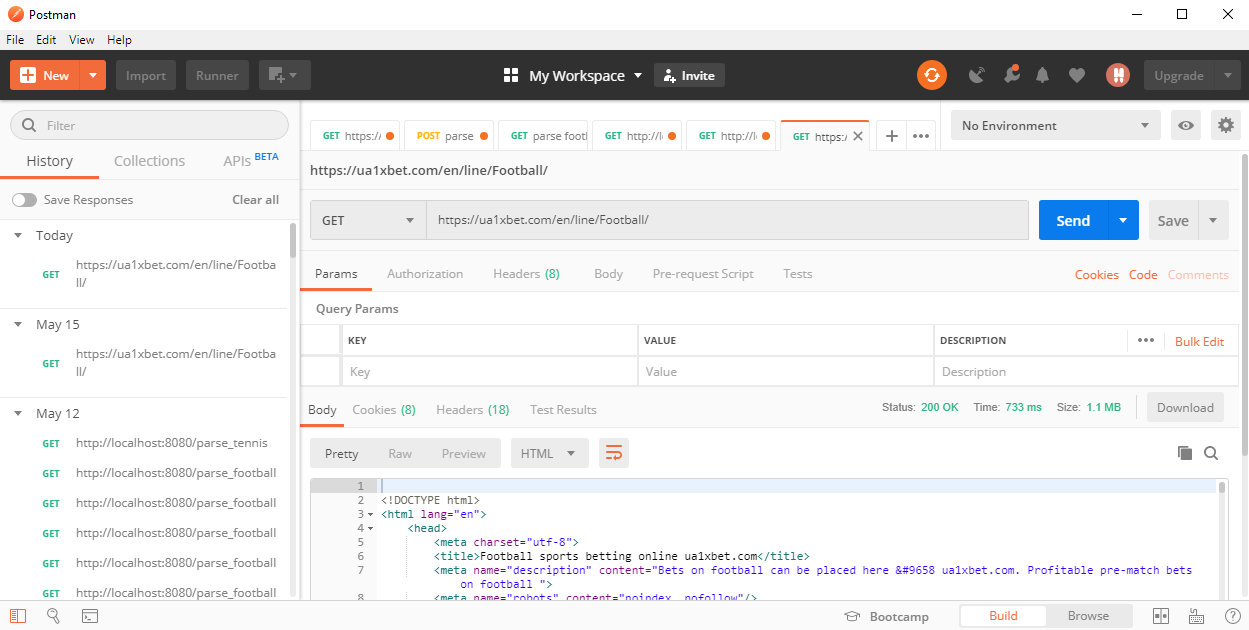
Основні характеристики, які має JUnit:

* Забезпечення анотацій для ідентифікації тестових методів.
* Забезпечення підтверджень для очікуваних тестових результатів.
* Забезпечення потоками для запуску тестів.
* Дозволяє швидко писати код.
* JUnit є дуже простим у використанні, що спрощує написання тестів для складних об’єктів і скорочує час на це.
* Тести можуть запускатися автоматично, перевіряють власні результати і надають швидкий звіт про їх проходження. Немає необхідності вручну аналізувати звіти про результати тестування.
* Тести JUnit можуть бути організовані у тестові блоки, які можуть складатися з інших тестових блоків.
* JUnit відображає тест із зеленою позначкою, якщо він пройшов успішно, і стає червоним якщо тест не пройдено з часом.

Фреймворк JUnit спочатку був розроблений для Java. З часом продукт став таким популярним, що його переписали для тестування коду на інших мовах програмування, це сімейство відоме як XUnit[2].

### Інструмент для тестування API Postman

**Postman** – одна з найбільш популярних систем для тестування API. В основному використовується для Rest API. Цю програму можна використовувати як програму на ПК, так як і звичайний плагін у браузері (рис. 2.8).



*Рис. 2.8 Інтерфейс програми Postman*

Основні функції Postman:

* Надсилати запити. Не потрібно використовувати браузер для перевірки своїх контролерів. Postman надає зручний інтерфейс для надсилання записів використовуючи різні методи HTTP, такі як GET, POST, PUT, DELETE, HEAD, OPTION.
* Підтримка різних типів збереження даних: xml, json, text, javascript, html.
* Можна писати автоматичні тести на мові JavaScript.
* Збереження запитів у колекції, якими можна поділитися з іншими розробниками. Можливість запускати всі запити, які містяться в колекції однією командою.

## Генератор HTML документів

Для повернення результату користувачеві з серверної частини, потрібно формувати HTML- сторінку. Для цієї операції передбачено багато інструментів, які орієнтуючись на шаблони генерують HTML-сторінки з даними.

Для розроблення системи агрегацію було обрано шаблонізатор Thymeleaf.

**Thymeleaf** – сучасний серверний механізм Java-шаблонів для веб-середовищ, здібний опрацьовувати HTML, XML, JavaSkript, CSS і навіть простий текст.

Основною метою **Thymeleaf** являється створення елегантного і зручного способу шаблонізації. Щоб досягнути цього, він базується на концепції Natural Templates, щоб додати власну логіку у файли шаблонів таким чином, щоб цей шаблон не впливав на відображення прототипудизайну. Це покращує кумунікацію в команді та зменшує різницю між дизайнерами та програмістами.

Усі типи шаблонів які опрацьовує Thymeleaf.

* HTML
* XML
* TEXT
* JAVASCRIPT
* CSS
* RAW

**Thymeleaf –** надає можливістьвизначати і налатовувати спосіб опрацювання шаблонів до тонкого рівня деталізації.

Об’єкт, який застосовує певну логіку до артефакту розмітки (тегу, текст, коментарю) називається процесом. Бібліотека Thymeleaf надає власний діалект і надає зручні засоби для створення своїх діалектів.

Thymeleaf має легкий і потужний синтаксис і тісну інтеграцію із екосистемою Spring.

Шаблони Thymeleaf дозволяють багаторазово використання. Можна створювати невеликі фрагменти шаблонів і повторно використовувати їх в інших шаблонах

**Висновки**

Проведено аналіз засобів реалізації системи агрегації і обрано основні інструменти. Для розробки системи агрегації використовується мотодологія Scrum. Для розробки обрано мову програмування Java та середовище IntelliJ IDEA, в якому буде писатися код. Для автоматичної збірки проекту використано система Gradle. Для контролю версій обрано веб сервіс GitHub. Щоб пришвидшити розробку системи обрано такі фреймворки: Spring Framework (Spring MVC, Spring Boot), JUnit, Jsoup. Для збереження даних обрано нереляційну базу даних MongoDB. Для генерації HTML- сторінок використовується шаблонізатор Thymeleaf.

# 3. Реалізація системи агрегації

## Розробка модулів серверної частини системи агрегації

Архітектура системи агрегації відповідає типовій клієнт-серверній архітектурі та містить наступні модулі (рис. 3.1).



*Рис. 3.1 Архітектура системи агрегації*

* Модуль з синтаксичними аналізаторами - забезпечує аналіз інформації з джерел даних і збереження її у базі даних.
* База даних – зберігає, і надає доступ до масиву даних системи.
* Веб-сервер – надає інтерфейс для користувача та виконує основну бізнес-логіку системи.

### Модуль з синтаксичними аналізаторами.

Основна мета системи агрегації є збірка інформації про коефіцієнти з різних сайтів букмекерських контор. Тому, роботу цього модуля буде продемонстровано на прикладі збору цих даних.

Для агрегації даних обрано сайт PariMatch, який не надає публічного API для зв’язку з його системою, тому збирати корисні дані можна способом синтаксичного аналізу HTML сторінки. Ефективність цього способу залежить від багатьох факторів.

Для кожного сайту і для кожної унікальної сторінки не можливо розробити єдиний синтаксичний аналізатор (парсер), що буде коректно працювати і повертати очікувані результати. Для кожного окремого випадку потрібно розробляти свій парсер. Синтаксичний аналізатор дуже чутливий до структури сторінки, яку він аналізує. Якщо зміниться дизайн сторінки, то потрібно буде знову аналізувати сторінку і переписувати його, щоб він повертав коректний результат. Алгоритм роботи реалізованого синтаксичного аналізатора зображено на рис. 3.2



*Рис. 3.2 Алгоритм роботи синтаксичного аналізатора.*

Алгоритм синтаксичного аналізатора такий:

1. Веб-сервер виконує запит на сервіс синтаксичного аналізатора і передає адресу досліджуваного сайту та вид спорту і виконується певний синтаксичний аналізатор.
2. Сервіс проходить по сайту і зчитує всі посилання на сторінки спортивних ліг певного виду спорту.
3. Опрацьовуються всі HTML сторінки, які знаходяться за посиланнями, що отримано на попередньому кроці. Опрацювання сторінок відбувається у багатопоточному режимі, тобто комп’ютер виконує аналіз кількох сторінок паралельно. Щоб забезпечити виконання завдань паралельно використовується об’єкт ExecutorService, який дозволяє виконувати завдання асинхронно, тобто програма не чекає на виконання завдання, а пізніше коли воно виконається, то буде отримано результат.

Якщо виникає виключна ситуація “SocketTimeoutException”, то сервіс через певний час спробує ще раз зчитати дані з цієї сторінки, але якщо виникає “ParserException” або “HttpStatusException” то це означає, що виникла помилка під час зчитування даних (структура сторінки не відповідає очікуваному шаблону) і сервіс записує помилку у файл з логами та продовжує аналізувати наступні сторінки.код який викликає парсер і обробляє посилання наведено у лістингу 3.1.

Лістинг 3.1:

*public List<FootballMatch> parseDataFootball() throws IOException {*

*try {*

*Document doc = Jsoup.connect(url).get();*

*ParserFootballHelper parser =new ParserFootballHelper(matchesFootball);*

*parser.parse(doc);*

*} catch (ParserException e) {*

*log.error(e.setAndReturnMessage(url));*

*} catch (HttpStatusException e) {*

*log.error(e.getUrl());*

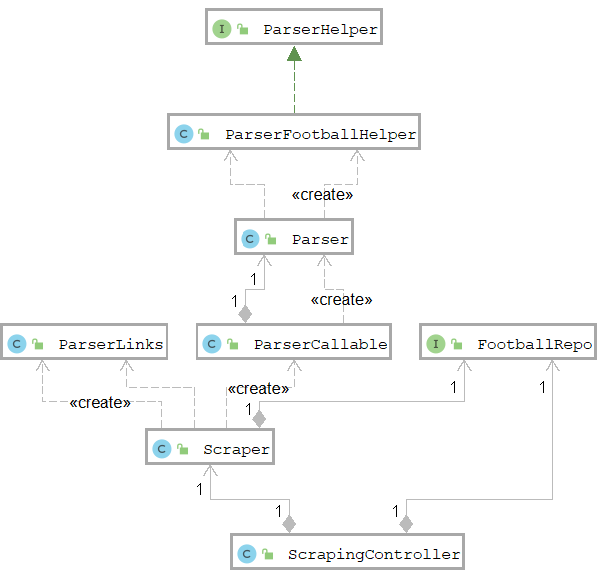
*}*

*return matchesFootball;*

*}*

1. Після аналізу сторінок за посиланнями, всі видобуті дані зберігаються у MongoDB.

Діаграму класів синтаксичного аналізатора для опрацювання виду спорту “Футбол” з сайту PariMatch зображено на рис. 3.3.



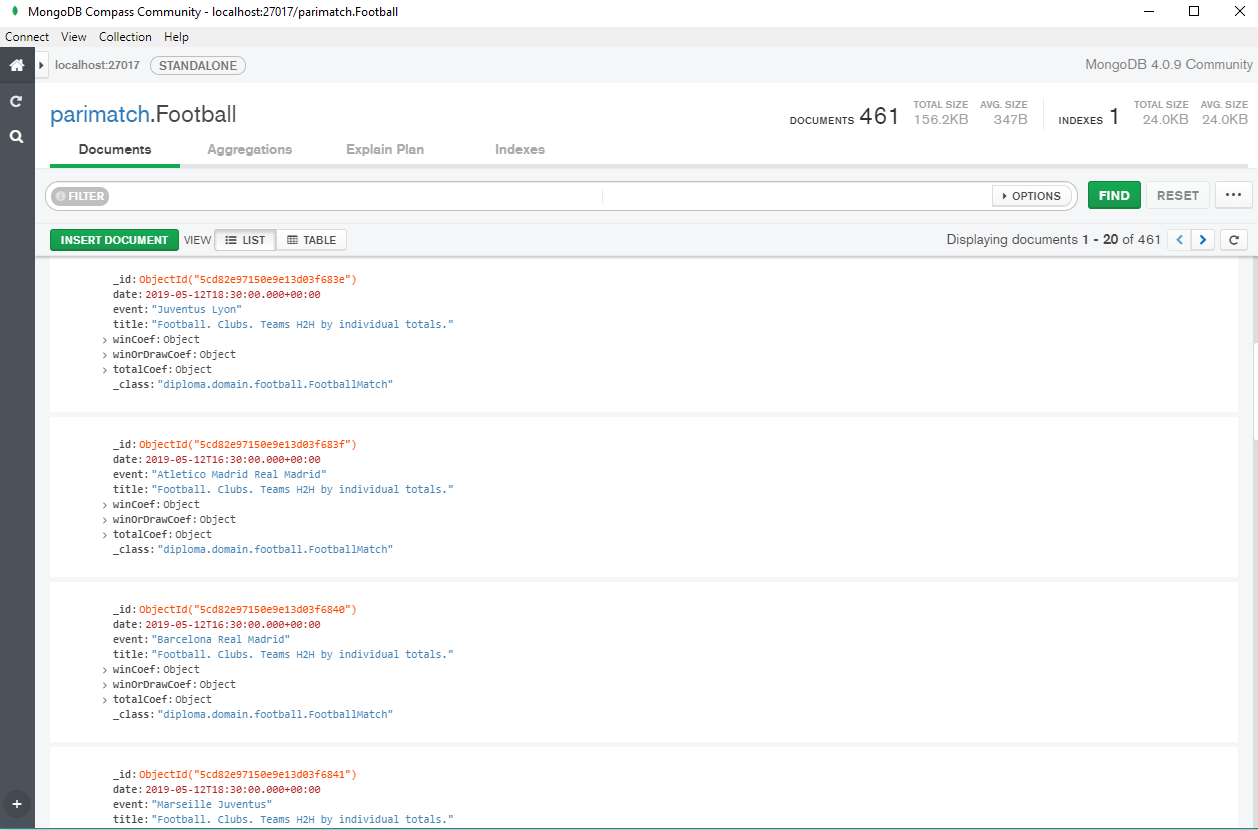
*Рис. 3.3 Діаграма класів синтаксичного аналізатора*

Кожен клас виконує певну свою функцію.

* Об’єкт ScraperController приймає запити від веб-сервера та викликає відповідний синтаксичний аналізатор.
* Scraper викликає потрібний парсер та зберігає результат у базу даних.
* ParserCallable це об’єкт, який обгортає парсер та запускає його в багатопоточному режимі.
* Об’єкт ParserLinks аналізує початкову HTML сторінку і збирає всі доступні посилання на сторінки всіх чемпіонатів певного виду спорту.
* FootballRepo – це об’єкт через який сервіс зв’язується з базою даних.
* Parser – об’єкт, який аналізує сторінку, збирає потрібні дані а обробляє помилки.
* ParserFootballHelper – це об’єкт, який виконує повний аналіз html-сторінки використовуючи бібліотеку Jsoup.

### Модуль з базою даних

Для збереження даних використовується нереряційна база даних MongoDB. Вона зберігає дані у документно-орієнованому вигляді. Швидко опрацьовує прості запити. Для зв’язку системи з базою даних використовується Spring Data MongoDB фреймворк. Який дозволяє швидко налаштувати базу даних та розробити інтерфейс для зв’язку системи з нею. Для налаштування та адміністрації MongoDB використовується клієнт MongoDB Compass Community (рис. 3.4).



*Рис. 3.4 Інтерфейс клієнту MongoDB Compass Community*

Для зберігання об’єктів потрібно створити POJO, через які будуть виконуватися операції над даними з бази даних. Ці об’єкти мають лише поля і методи, щоб отримувати значення або змінювати їх. Вони не мають жодної поведінки (лістинг 3.2).

Лістинг 3.2 клас FootballMatch

@Data

@Document(collection = "Football")

public class FootballMatch {

@Id

private String id;

private LocalDateTime date;

private String event;

private String title;

private WinCoef winCoef;

private WinOrDrawCoef winOrDrawCoef;

private TotalCoef totalCoef;}

Анотація @Data забезпечує автоматичне генерування спеціальних методів для доступу і зміни значень полів класу. Забезпечує зручний для читання код.

Анотація @Document доступна з модуля Spring Data MongoDB, вона вказує цьому модулю що потрібно створити документ у базі даних, який містить у собі колонки відповідні до полів класу над який вона вказується. Параметр collection вказує, як потрібно назвати документ куди будуть зберігатися дані.

Використання MobgoDB відбувається через розширення інтерфейсу MongoRepository, у якому є доступні стандартні операції для запису, вибірки та видалення записів. Для реалізації власних методів, достатньо, створити методи, назва яких відповідає певним правилам. Код класу FootballRepo у лістингу 3.3.

Лістинг 3.3 інтерфейс FootballRepo.

@Repository

public interface FootballRepo extends MongoRepository<FootballMatch,String>, CustomDocRepository {List<FootballMatch> findByEventAndTitle(String event, String title);

List<FootballMatch> findByDate(LocalDateTime dateTime);

FootballMatch updateFootballMatch(FootballMatch footballMatch);

Boolean deleteByEventAndTitle(String event, String title);}

Цей інтерфейс при запуску аналізувався модулем Spring Data MongoDB, та генерувався клас, який реалізовував прописані в ньому методи. Це пришвидшило розробку продукту, та призвело до зменшення можливостей для помилок розробника, приховавши від нього реалізацію.

Структура бази даних зображена на рис.3.5.



*Рис. 3.5 Структура бази даних*

Зв’язків між сутностями не має, тому що це незалежні таблиці і не потребують його.

### Веб-сервер

Це один з найважливіших модулів. Забезпечує обробку запитів користувача та керування сервісом із синтаксичними аналізаторами.

Для розроблення API обрано архітектурний стиль REST.

REST (Representational State Transfer — «передача стану представлення») — архітектурний стиль взаємозв’язку компонентів системи в мережі. REST представляє узгоджений набір обмежень, врахованих при проектуванні системи.

У модулі використовується Apache TomCat. Він може опрацьовувати HTTP-запити та використовується як сервлет-контейнер. Усі контролери створюють сервлети і вони розгортаються у веб-сервері.

Веб-сервер отримує від користувача лише GET-запити.

GET запити виконуються для отримання вказаного ресурсу. За допомогою GET запитів веб-сервер запускає синтаксичний аналізатор, який зберігає дані коефіцієнтів у базу даних.

Код контролера, який опрацьовує запити від користувача наведено на лістингу 3.4

Лістинг 3.4 Клас MainController

@RestController

public class MainController {

@Autowired

private FootballService footballService;

@Autowired

private TennisService tennisService;

@Autowired

private BasketballService basketballService;

@Autowired

private VolleyballService volleyballService;

@GetMapping("/football")

public String getFootball()ModelAndView model= new ModelAndView("football");model.addObject("footballMatch", footballService.getFootballMatches());

return model;}

@GetMapping("/tennis")

public String getTennis(){

ModelAndView model= new ModelAndView("tennis");model.addObject("tennisMatches", tennisService.getTennisMatches());

return model; }

@GetMapping("/basketball")

public String getBasketball(){

ModelAndView model= new ModelAndView("basketball");

model.addObject("basketballMatch",basketballService.getBasketballMatches()); return model; }

@GetMapping("/volleyball")

public String getVolleyball(){

ModelAndView model= new ModelAndView("volleyball");model.addObject("volleyballMatch", volleyballService.getVolleyballMatches());

return model; }}

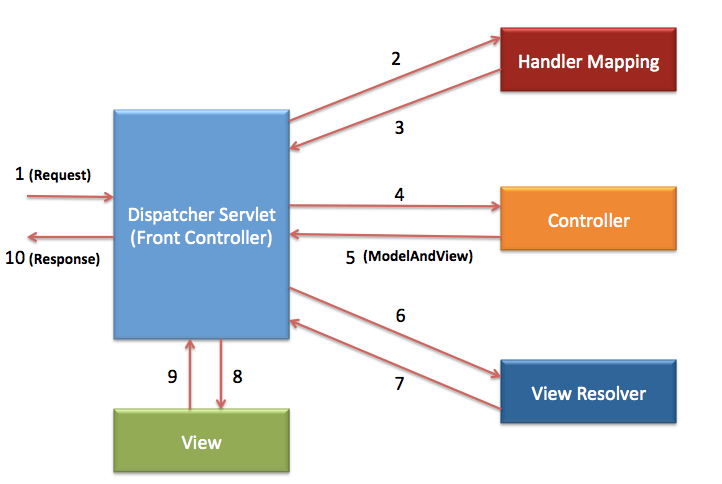
Для опрацювання HTTP-запитів система використовує Spring MVC. Цей фреймворк надає зручний інтерфейс та багато анотацій для розроблення веб-сервісів.

@RestController – анотація, що вказує на клас, який реагує на HTTP-запити.

@GetMapping – анотація вказує на метод в контролері, який має виконатися при відповідному HTTP-запиту.

У кожному методі викликається сервіс, який повертає всі матчі відповідного типу спорту з бази даних. Створює об’єкт ModelAndView і передає йому всі матчі. Для генерації HTML-сторінки використовується шаблонізатор Thymeleaf.

Алгоритм роботи веб-сервера зображено на рис.3.6



*Рис. 3.6 Алгоритм роботи веб-сервера*

1. Надходження запиту від користувача.
2. Передача запиту у HandlerMaping, що повертає контролер, який має опрацювати запит.
3. Повернення назви контролера.
4. Передача запиту до контролера і збереження результату опрацювання у ModelAndView.
5. Передача ModelAndView до DispatcherServlet.
6. Передача ModelAndView у ViewResolver, який визначає View( HTML- сторінку), яку має отримати користувач.
7. Повернення назви View.
8. Передача даних на View, де вони будуть відображатися використовуючи Thymeleaf шаблони.
9. Повернення готової відповіді до DispatcherServlet.
10. Повернення відповіді користувачу.

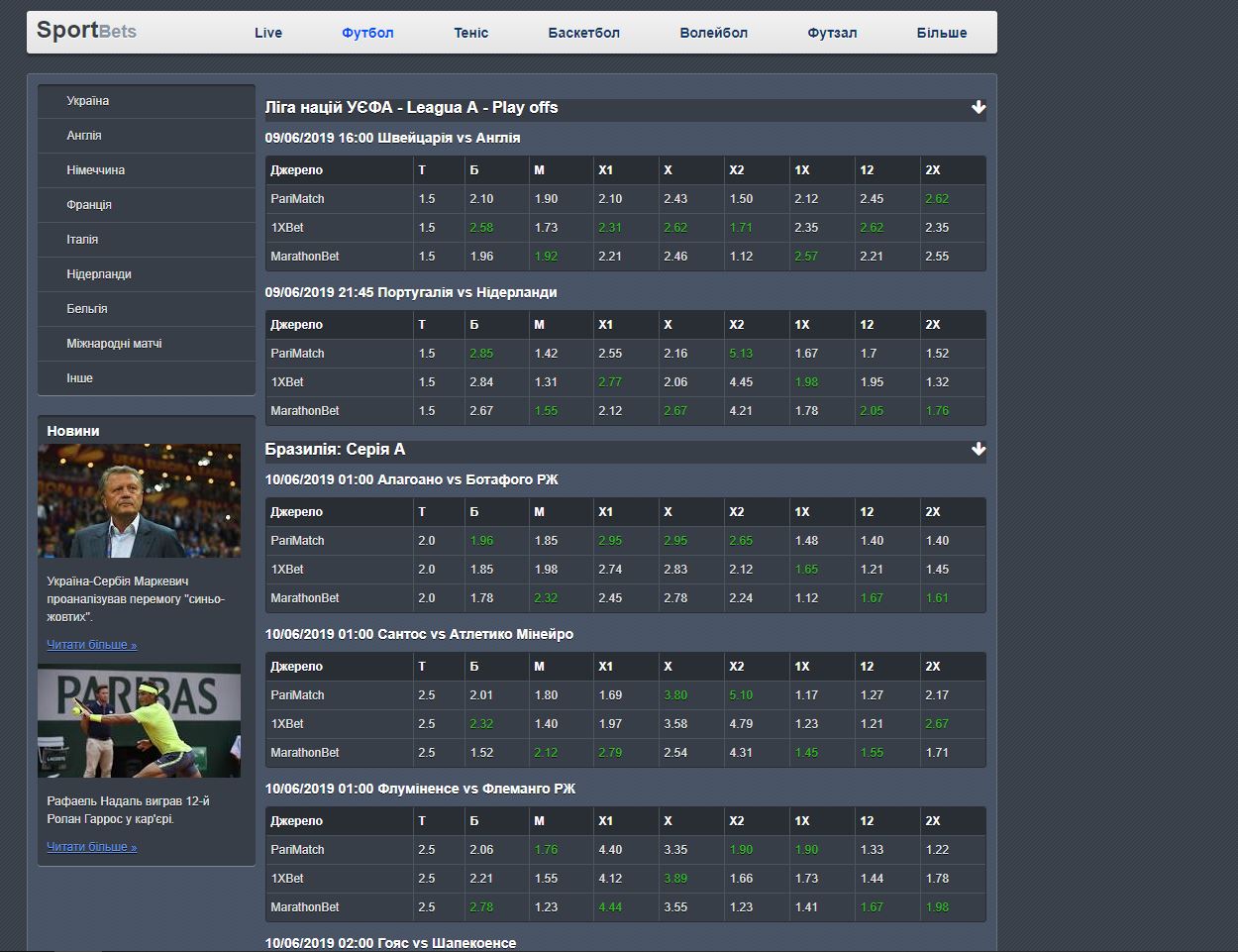
**Висновки**

Розроблено систему агрегації, яка складається з трьох модулів. Перший модуль сервіс із синтаксичними аналізаторами опрацьовує джерела інформації, використовуючи бібліотеку Jsoup. Модуль бази даних дозволяє зберігати великі масиви даних на твердий диск сервера та зручний доступ до них. Веб сервер відповідає на запити користувачів та повертає їм HTML сторінки.

# Опис інтерфейсу та інструкція користування

Для користувача розроблено зручний та інтуїтивно зрозумілий інтерфейс. Система агрегації не потребує реєстрації та авторизації користувача. Він одразу ж отримує доступ до усіх функцій системи.

Зображено головну сторінку на рис. 4.1.



*Рис. 4.1 Головна сторінка*

Сервіс називається SportBets. Клікнувши на логотип справа зверхукористувач обновляє сторінку та переправляється на головну сторінку якщо він не на ній.

Верхня панель складається з переліку видів спорту, якіє найбільш популярні серед користувачів. Клікнувши на кнопку “Більше” можна отримати повний перелік підтримуваних видів спорту.

На сторінці зліва є 2 блоки. Верхній блок відображає регіони ,які користувач може обрати, щоб ознайомитися з результатами лише цих ліг. Нижній блок відображає спортивні новини, які агрегуються з новинних сайтів. Клікнувши на новину користувача буде направлено на сайт-джерело новини.

Основний блок знаходиться посередині сторінки. На ньому відображено усі ліги, які підтримуються сайтом. На даний момент реалізована агрегацію інформації коефіцієнтів на ставки на спортивні події з цих сайтів: PariMatch, 1XBET, MarathonBet.

Над таблицями відображено назву спортивної ліги. Під нею дата та перелік учасників спортивної події. На сайті відображено лише найбільш популярні показники, за якими можна ставити ставки.

Тотал «T» – показник, який визначає поріг який має бути досягнуто або ні.

Більше «Б» - коефцієнт, який визначає прибуток при тому якщо результат буде більший ніж показник Тотал.

Більше «М» - коефцієнт, який визначає прибуток при тому, якщо результати буде менший ніж показник Тотал.

«X1» - коефцієнт, який визначає прибуток при тому, якщо переможе команда, яка грає на домашній арені або відображається зліва(знизу) на табло.

«X» - коефцієнт, який визначає прибуток при тому, якщо жодна з команд не виграє у основний час, а буде нічия.

«X2» - коефцієнт, який визначає прибуток при тому, якщо переможе команда, яка грає у гостях або відображається справа (знизу) на табло.

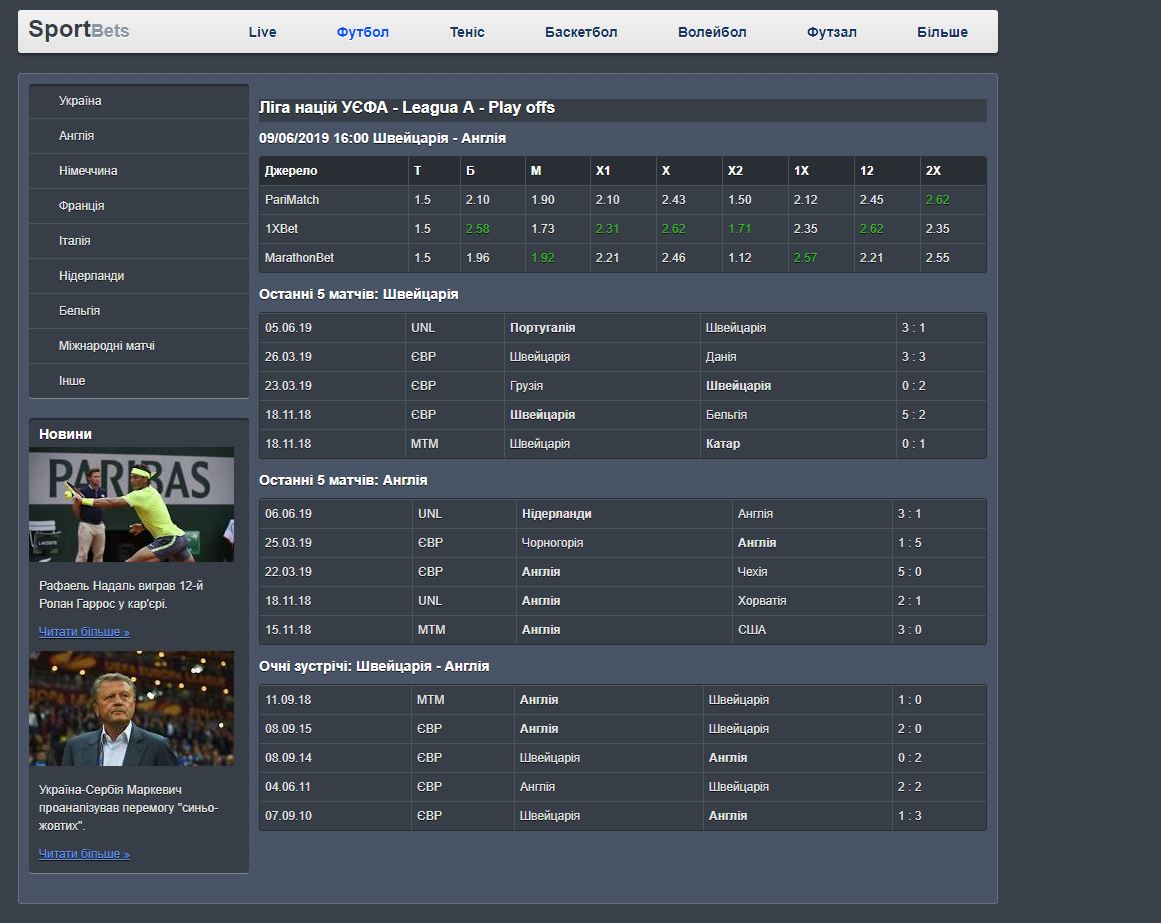
«1X» - коефцієнт, який визначає прибуток при тому, якщо команда, яка грає на домашній арені або відображається зліва (зверху) на табло, не програє тобто переможе або зіграє у нічию.

«12» - коефцієнт, який визначає прибуток при тому, якщо не буде нічиєї, тобто якась із команд обов’язково переможе.

«1X» - коефцієнт, який визначає прибуток при тому, якщо команда, яка грає на чужій арені або відображається справа (знизу) на табло, не програє тобто переможе або зіграє у нічию.

Коефіцієнти, які є найбільш вигідні у певній категоріє підсвічуються зеленим кольором, щоб зручніше було аналізувати дані користувачем.

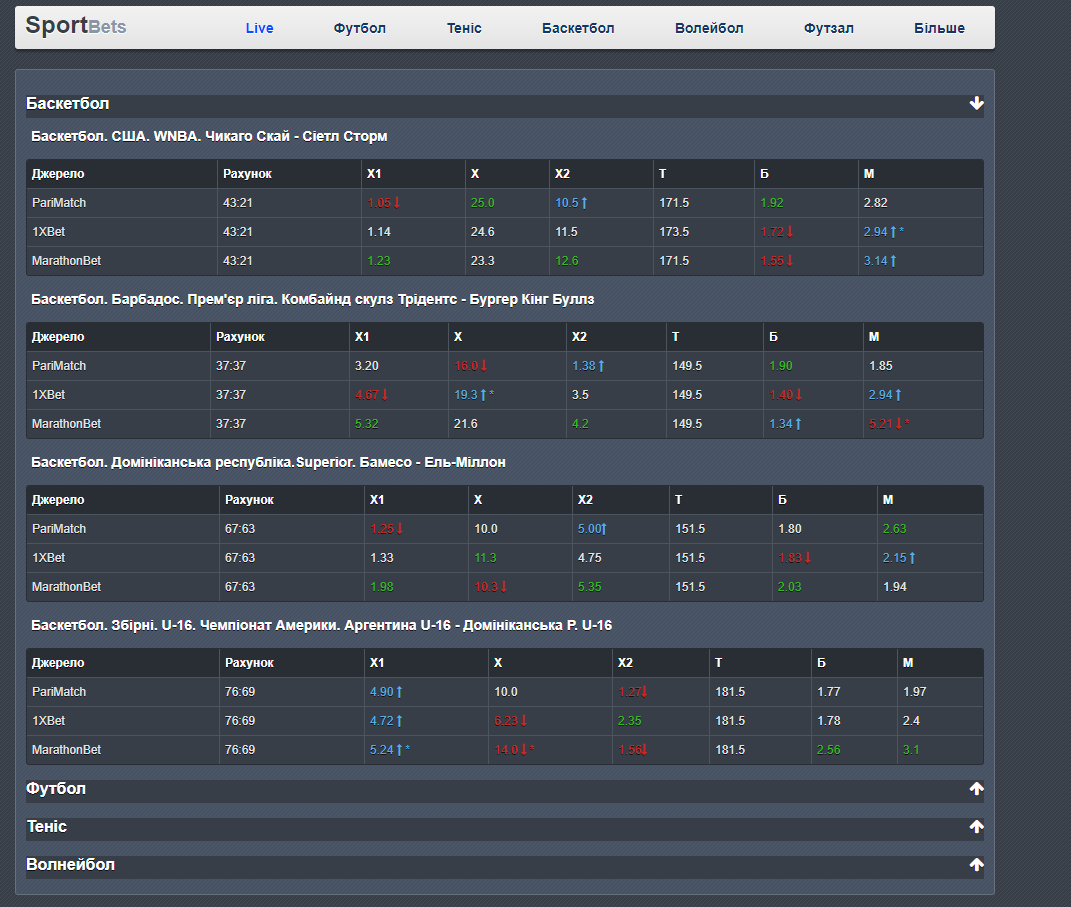
Клікнувши на назву певної спортивної події, користувача буде направлено на сторінку з інформацією про коефіцієнти на ставки цієї події та статистику учасників спортивної події (рис. 4.2).



*Рис. 4.2 Статистика учасників спортивної події*

На сторінці відображено останні 5 матчів кожного учасника та результати 5 очних зустрічей учасників події. Це дозволить користувачеві сфокусувати свою увагу лише на цікаву йому спортивну подію та ознайомитися з ефективністю кожного учасника.

У таблиці є інформація про дату коли відбулася подія, чемпіонат під егідою, якого відбулася подія, назви учасників та результат події.

Клікнувши на верхній панелі на текст “Live”, користувача буде направлено на сторінку з результатами спортивних подій в реальному часі (рис. 4.3).

*Рис. 4.3 Відображення коефіцієнтів у режимі реального часу*

На сторінці відображено розкритий блок з видом спорту Баскетбол. В цей момент не відбувалась жодна подія серед інших виді спорту тому ці блоки приховані.

Якщо коефіцієнт змінюється то інтерфейс відповідно реагує на його зміну. Якщо він більший порівняно останньої ітерації агрегування, то коефіцієнт відображений у синьому кольорі та із стрілкою вверх. Якщо коефіцієнт нижчий, то значення зафарбовується у червоний колір та відображається із стрілкою вниз.

Відображення найкращого результату для користувача також присутнє. Якщо коефіцієнт не змінювався то він зафарбовується у зелений колір, а якщо змінювався, то біля нього з’являється знак «\*» , що вказує на те, що зміни призвели до того що коефіцієнт є найбільш вигідним для користувача із різних ресурсів.

Схему дій користувача на сайті ображено на рис. 4.4.



*Рис. 4.4 Схема дій користувача*

**Висновок**

Розроблено графічний інтерфейс для користувача. Він є зручний та інтуїтивно зрозумілий для користувача. Доступні 2 види перегляду коефіцієнтів статичний та в режимі реального часу. Статичний вид потрібен для аналізу спортивної події , яка ще не почалася та ознайоми користувача із статистикою учасників та з коефіцієнтами на ставки на спортивні події на різних букмекерських конторах. А динамічний потрібен щоб аналізувати зміни коефіцієнтів на спортивну події в режимі реального часу, тобто під час проведення її.

# Висновок

В бакалаврській кваліфікацій роботі досліджено та реалізовано систему агрегації та порівняння ставок на спортивні події.

Оглянуто основні типи агрегаційних систем та їх призначення. Досліджено аналоги агрегаційних систем, які забирають інформацію про спортивні події. Визначено переваги та недоліки агрегаційних систем.

Основними етапами розробки агрегаційної системи є: проектування архітектури, програмна розробка, та створення графічного інтерфейсу.

Обрано клієнт-серверну архітектуру, як типову архітектуру для реалізації агрегаційних систем. Для розробки продукту обрано мову програмування Java, тому що вона на даний момент одна з найпопулярніших мов для розробки застосунків на серверній стороні. Має велику спільноту та великий багаж бібліотек та фреймворків, які спрощують та пришвидшують розробку програмного продукту. Як основний фреймворк обрано Spring Framework, бо він має багато незалежних модулів і використовує дизайнерський шаблон Dependency Injection. Що спрощує розробки системи і дозволяє уникати залежностей об’єктів один від одного.

Для агрегування даних обрано бібліотеку Jsoup, що дозволяє проводити синтаксичний аналіз HTML-сторінок, та видобування корисних даних.

Для збереження даних обрано нереляційну базу даних MongoDB, яка зберігає дані у документно-орієнтованому вигляді. Що дозволяє динамічно змінювати структуру документів та швидко виконувати прості CRUD – запити. Для зв’язку з базою даних використовується Spring Data MongoDB, що дозволяє уникнути написання багатьох запитів, бо цей фреймворк дозволяє декларативно створити їх. Для тестування обрано фреймворк Junit, який є інтегрований у Spring Framework, та було використано Postman для тестування API. Для генерування HTML-сторінок обрано шаблонізатор Thymeleaf.

Реалізовано агрегаційну систему з трьома модулями: сервіс із синтаксичними аналізаторами, модуль із базою даних та веб сервер. Сервіс із синтаксичними аналізаторами відповідає за аналіз HTML-сторінок та агрегацію даних. Модуль з базою даних забезпечує запис і вибірку інформацію з неї. Веб-сервер забезпечує доступ користувачів до системи агрегації через кінцеві точки, які були розроблені за архітектурою Rest, та реалізовані використовуючи Spring MVC.

Розроблено зручний користувацький інтерфейс, який є зрозумілий на інтуїтивному рівні та відображає усі функції, заради яких була реалізована система.

З економічної точки зору проект має усі умови для успішного розвитку.

# СПИСОК ЛІТЕРАТУРИ

1. Spring Framework Overview. - <http://www.tutorialspoint.com/spring/spring_overview.htm>
2. Unit тестування з JUnit. – <http://devcolibri.com/864> .