МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційно-аналітична технологія моніторингу ціноутворення онлайн-платформи Nintendo»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студента групи ІНм – 02 **Сирін Є.М.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Сиріну Єгору Миколайовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційно-аналітична технологія моніторингу ціноутворення онлайн-платформи Nintendo*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд існуючих рішень проблеми;2) Постановка завдання й формування завдань дослідження; 3) Проектування інформаційної системи; 4) Аналіз технологій для рішення задачі 5) Розробка системи; 6) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд існуючих рішень проблеми		
2.	Постановка задачі та формування завдань дослідження.		
3.	Опис архітектури		
4.	Розробка інформаційно-аналітичної системи		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 85 стор., 27 рис., 2 таблиці, 3 додатки, 20 літературних джерел.

Об'єкт дослідження — Інформаційно-аналітична технологія моніторингу ціноутворення онлайн-платформи Nintendo.

Мета роботи — розробка інформаційно-аналітичної технології, котра допоможе користувачеві відслідковувати ціни на необхідні товари онлайн-платформи.

Результати — було з'ясовано, що всі аналоги не мають необхідного повного функціоналу. Виходячи з аналізу проблеми було сформовано задачу дослідження. Було спроєктовано архітектуру інформаційної технології. Було реалізовано інформаційно-аналітичну систему у вигляді трьох компонентів. REST-додаток реалізовано з використанням мови JavaScript та фреймворку NodeJS. Веб-додаток створено з використанням мови програмування Java та фреймворку Spark. Додаток для розсилання електронних листів реалізовано з використанням Java. Усі компоненти інформаційної системи розміщено на сервері.. Готовий прототип можна переглянути за посиланням https://eshoperer.com. Мети дослідження було досягнуто в повному обсязі. Інформаційно-аналітична технологія реалізована у відповідності до завдань дослідження.

ІНФОРМАЦІЙНА СИСТЕМА, ESHOP, NODEJS, JAVA, JAVASCRIPT, SPARK, SQLITE, REST.

ЗМІСТ

ВСТУП

У сучасному світі з кожним роком зростає попит на придбання товарів онлайн. За допомогою мережі Інтернет є можливість здійснити будь-які покупки. А деякі фізичні товари стають менш розповсюдженими через наявність цифрових аналогів та можливості безпечно їх придбати. Так, наприклад, на даний момент дистрибуція відеокасет з фільмами чи платівок з музикою стала радше вузькопрофільною торгівлею з цільовою авдиторією колекціонерів. Індустрія кінематографу чи музична індустрія за останні роки значно змінилася завдяки можливості цифрової дистрибуції. З'явилося багато способів слухати музику чи дивитися кіно за різними цінами, велика кількість сервісів надають унікальні можливості для користувачів і при цьому мають різну цінову політику.

Ігрова індустрія відносно молода, у порівнянні з, наприклад, кіноіндустрією. Одна з платформ для споживання ігрового контенту – платформа на основі останньої ігрової консолі компанії Nintendo – Nintendo Switch. Цей пристрій було придбано 89 мільйонів разів, що говорить про популярність платформи. Є можливість придбання фізичних копій ігор. Але компанія також надає можливість придбання цифрових копій за допомогою платформи eShop. Цей спосіб є популярнішим, адже користувачу не потрібно чекати, поки фізичну копію у вигляді картриджу доставлять, чи йти до точки продажу. За час існування консолі Nintendo Switch користувачі придбали 632 мільйони цифрових копій ігор.

Актуальність дослідження.

Однією з особливостей платформи eShop є наявність регіональних цін. Вартість одного й того ж товару може варіюватися в залежності від обраної країни акаунту. Знижки на товари також є регіональними. У той же час, на відміну від багатьох інших цифрових платформ дистрибуції ігор, Nintendo не забороняє змінювати регіон акаунту. Користувач може в будь-який момент обрати іншу країну. Таким чином, існує можливість істотно зекономити на покупці певного товару, маючи інформацію про всі знижки для різних країн і

ціни на різні товари в різних регіонах. Зазначене засвідчує актуальність обраної тематики дослідження.

Метою цієї роботи є розробка інформаційно-аналітичної технології, котра дозволяє відслідковувати ціни на товари платформи eShop для доступних регіонів.

Для реалізації поставленої мети необхідно виконати наступні завдання:

- проаналізувати існуючі рішення проблеми та сформулювати задачу дослідження;
- спроектувати інформаційно-аналітичну технологію моніторингу цін онлайн-платформи;
- реалізувати інформаційну технологію.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Аналіз проблеми

Через популярність онлайн-платформи і можливість зміни регіону для економії на покупках було створено значну кількість інформаційно-аналітичних систем для моніторингу цін. Ці системи надають користувачу можливість відслідковувати ціни на конкретний товар. Деякі з систем надають можливість додавати товари до списку бажаного й отримувати сповіщення на електронну пошту, якщо товар продається зі знижкою.

Нижче подано аналіз найпопулярніших систем. На основі проаналізованих систем окреслено проблему дослідження.

Один з найпростіших способів моніторингу цін є власне перегляд цін на товари безпосередньо в **онлайн-магазині eShop** [1].

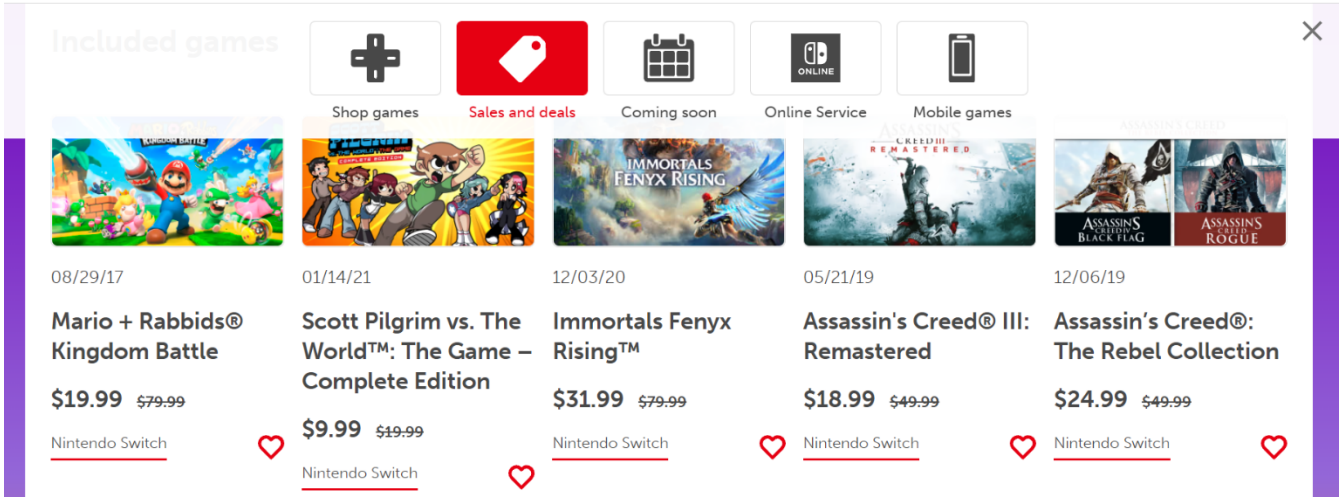

Рисунок 1.1 Онлайн-магазин eShop

Компанія Nintendo надає можливість додавати цифрові копії ігор до списку бажаних товарів.

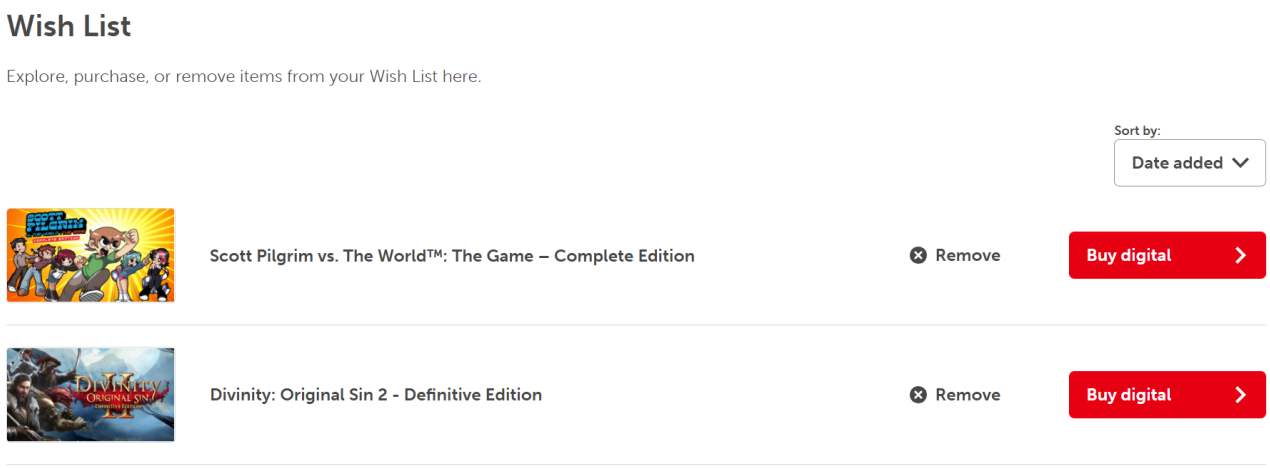

Рисунок 1.2 Список бажаних покупок

Користувач буде отримувати сповіщення про те, що товари з списку зараз продаються зі знижкою. З недоліків цього підходу слід відзначити те, що користувач бачить ціни тільки одного, обраного регіону. Тож для того, щоб знайти вигідну пропозицію фактично треба обрати кожен з регіонів і переглянути ціни.

Ресурс **Deku Deals** [2] має схожий з офіційним сайтом функціонал. Користувач має можливість додавати ігри до списку бажаного і отримувати листи про знижки на товари.

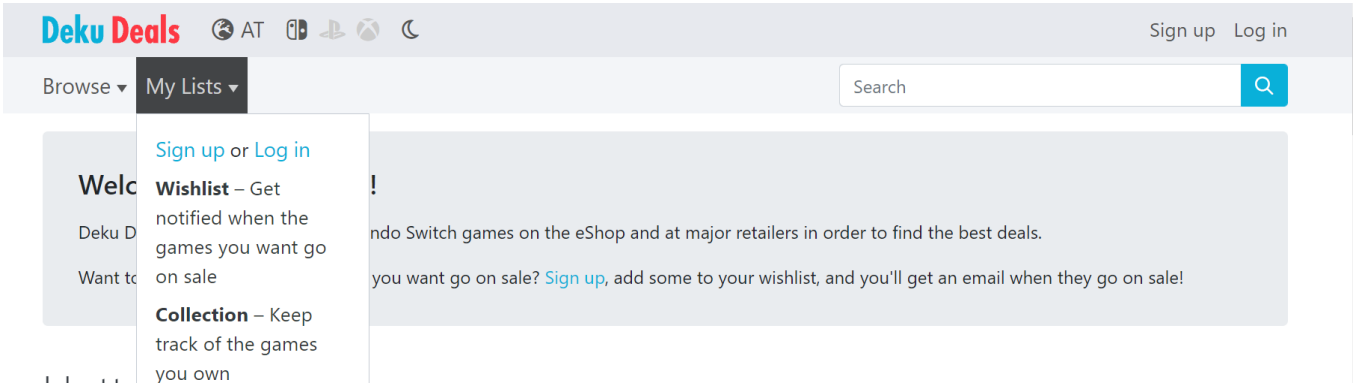

Рисунок 1.3 Ресурс Deku Deals

Особливостями цього ресурсу є можливість відслідковувати ціни на товари не тільки у вигляді електронних копій, а й фізичних на різних торгових майданчиках (Amazon, eBay).

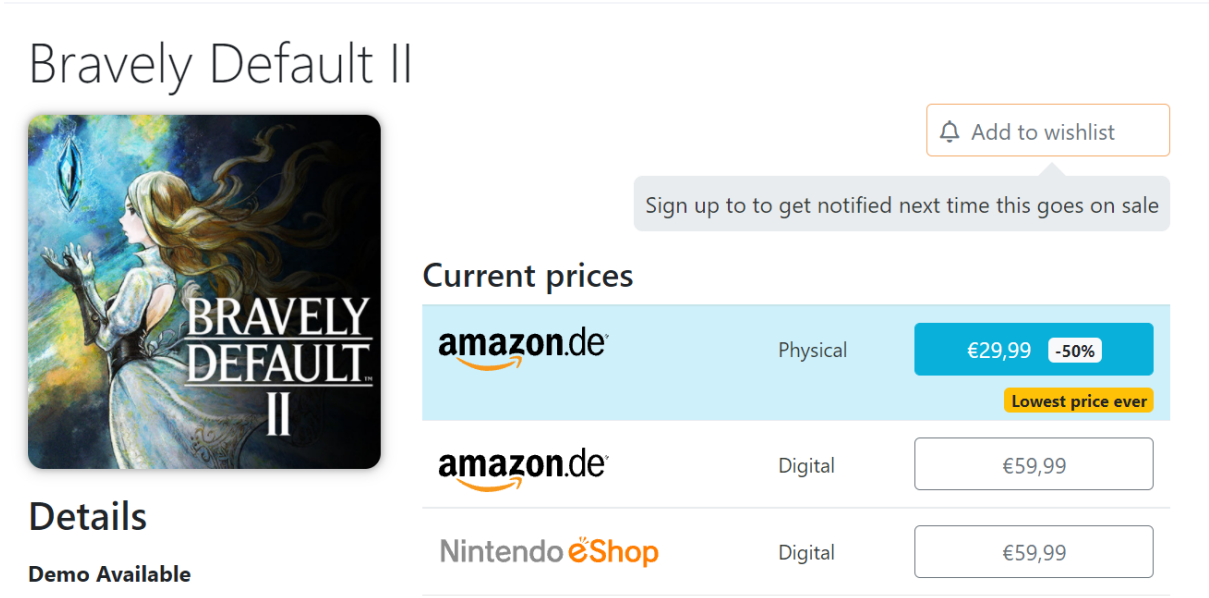

Рисунок 1.4 Варіанти для купівлі фізичних копій

Також є можливість бачити історію цін і, таким чином, розуміти вигідність пропозиції.

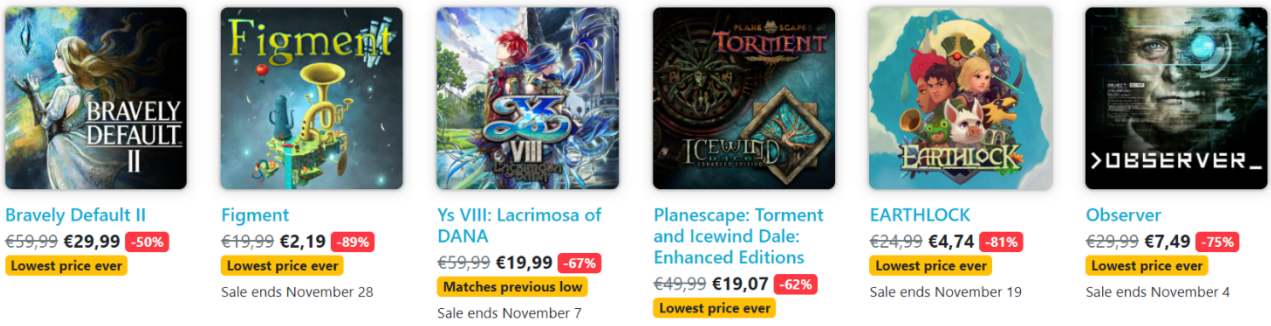

Рисунок 1.5 Оцінка вигідності пропозиції

Недолік даного ресурсу такий, як і в eShop, – користувач обирає регіон акаунту і бачить знижки, а також ціни безпосередньо для нього. Такий варіант не дає можливості обирати найвигідніші пропозиції.

Ресурс **esho-prices** [3] надає можливість перегляду цін на конкретний товар для всіх доступних регіонів. Вони відсортовані за зростанням ціни.

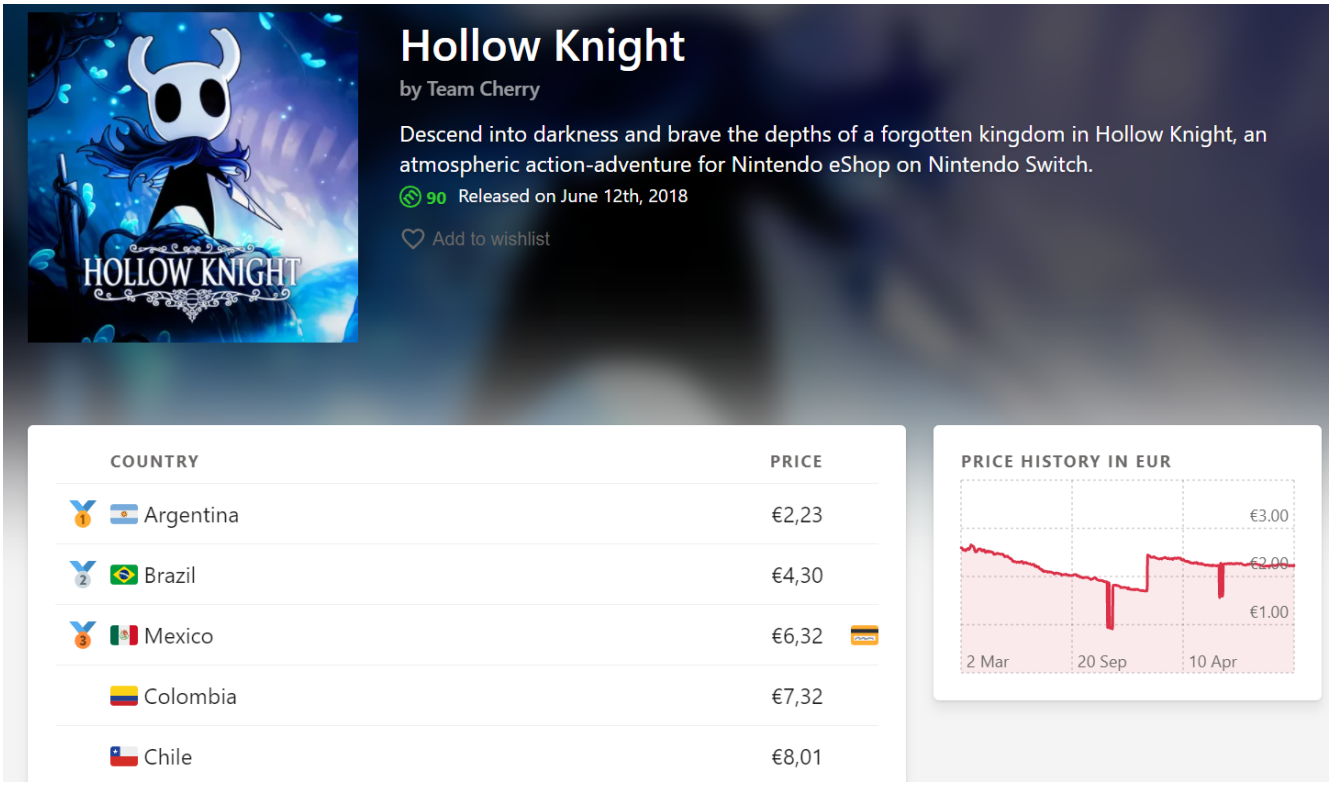

Рисунок 1.6 Ресурс eShop-prices

Є можливість додати гру до списку бажаних покупок. Але в той же час відсутній функціонал сповіщення через електронну пошту. Користувач немає облікового запису – список бажаного прив'язується до конкретного пристрою. Тож для того, аби дізнатися, що ціна на необхідний товар знизилася, потрібно щодня перевіряти на нього ціни, заходячи на сайт. Це може бути незручно, особливо з урахуванням прив'язки списку бажаного до одного пристрою.

Ресурс **switchgames.io** [4] має схожий до Deku Deals функціонал: користувач додає цифровий товар до кошика й отримує сповіщення кожного разу, коли гра продається зі знижкою. Можна побачити історію цін на товар.

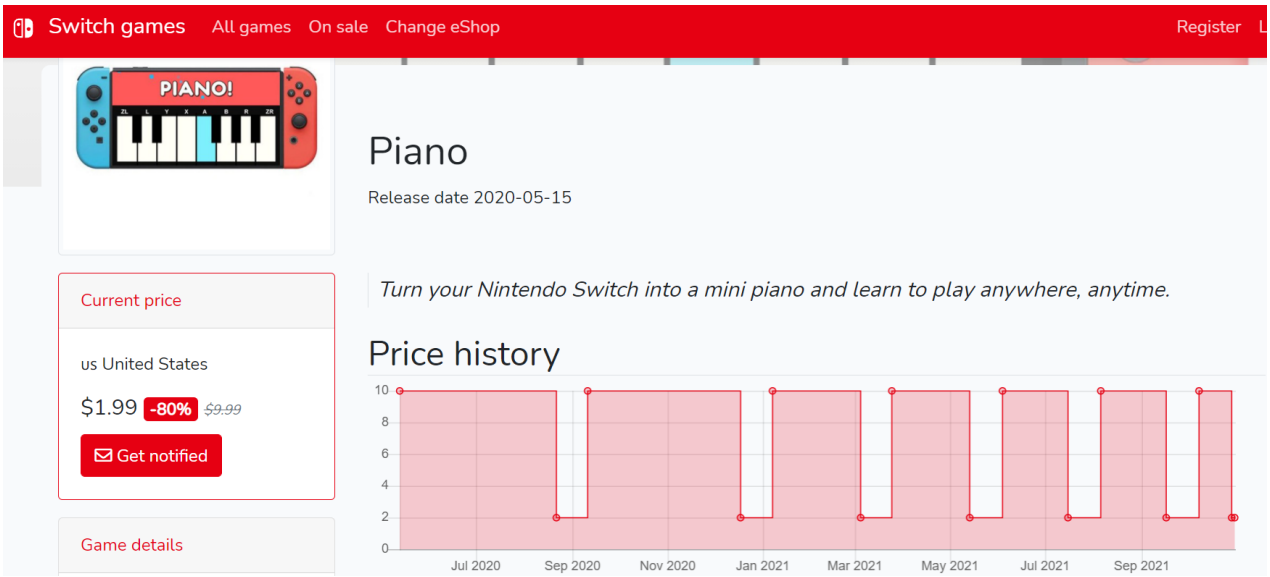

Рисунок 1.7 Історія цін на ресурсі switchgames.io

Недолік ресурсу теж аналогічний – користувач одразу обирає необхідний регіон:

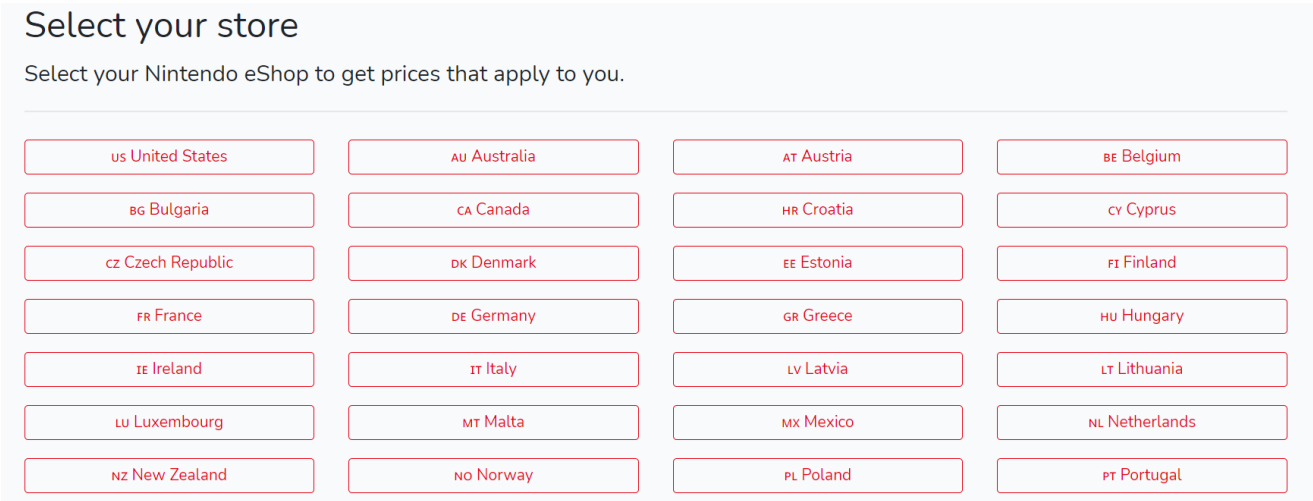

Рисунок 1.8 Обов'язковий вибір регіону на switchgames.io

Ресурс **NT Deals** є повним аналогом попереднього веб-сайту і надає такі ж можливості.

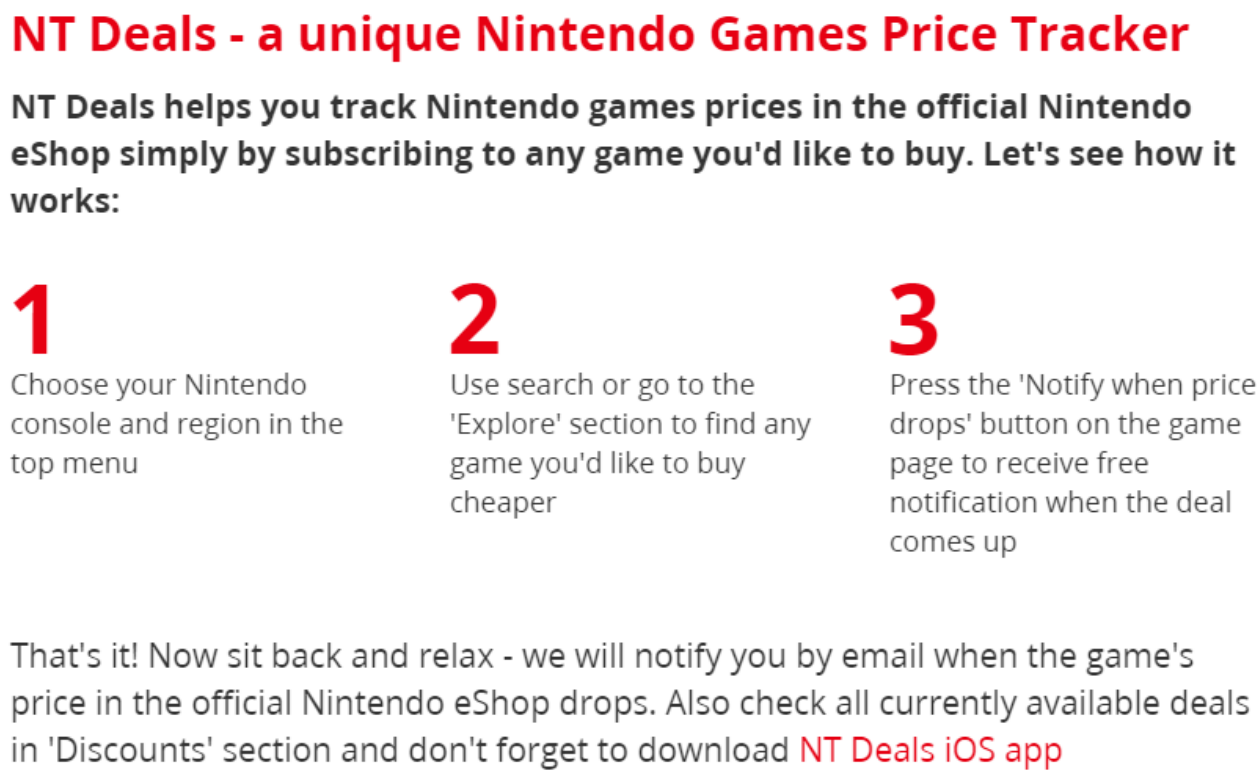

Рисунок 1.9 Інструкція з ресурсу NT Deals

Особливістю є наявність мобільного додатку. Також даний ресурс дозволяє відслідковувати ціни і на інших онлайн-платформах. Але такий підхід усе ще змушує користувача регулярно заходити на сайт (чи в додаток) для того, щоб перевірити ціни.

Ресурс **pricecharting** [5] створений для аналізу динаміки цін на окремі товари і комбінує в собі моніторинг різних торгових мереж. Але відсутня можливість обирати регіон, додаток створений виключно для США.

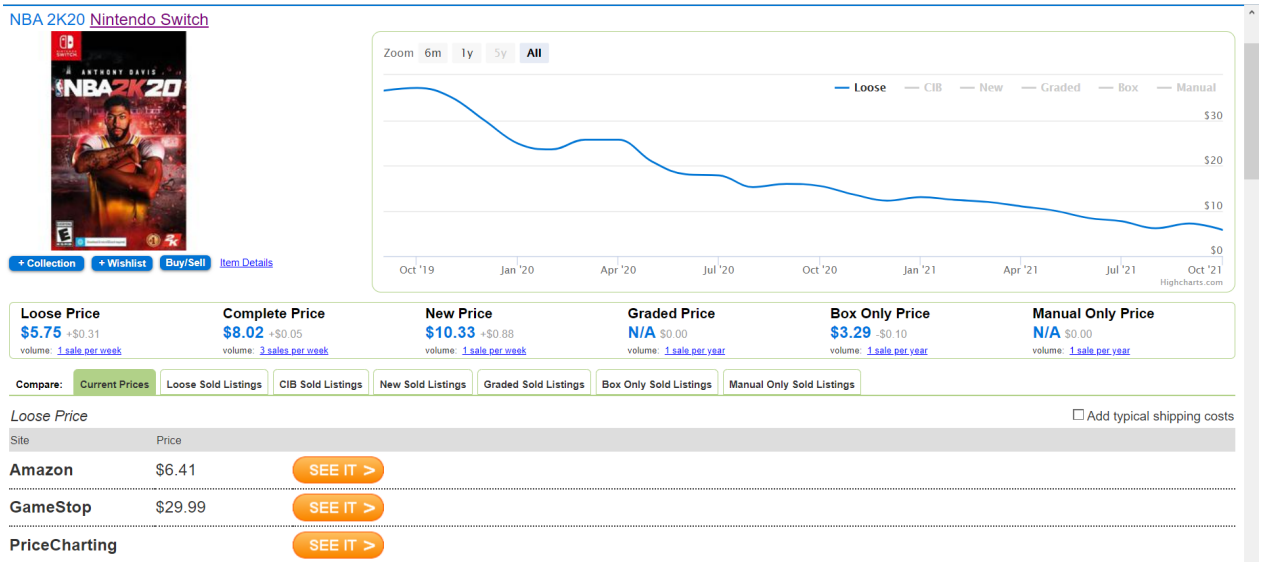

Рисунок 1.10 Ресурс pricecharting

Є ще багато інших сервісів для моніторингу цін, але всі вони мають схожий функціонал. У таблиці 1.1. узагальнено та подано порівняльну таблицю ресурсів.

Таблиця 1.1. Порівняльна характеристика існуючих ресурсів

Назва ресурсу	Можливість створення аккаунту і ведення списку бажаного	Можливість отримання повідомлення про знижки за допомогою електронної пошти	Можливість моніторингу цін в різних регіонах одночасно
eShop	+	+	-
Deku Deals	+	+	-
eshop-prices	-	-	+
switchgames	+	+	-
ntdeals	+	+	-
pricecharting	+	+	-

Відповідно до таблиці, жоден з популярних ресурсів не надає достатнього функціоналу для користувача. Додатковою проблемою є той факт, що оплата товарів відбувається через країну відповідного регіону. І, наприклад, банківські картки України не можуть бути використані в Аргентині чи Бразилії. Тож користувачу необхідна можливість виключати певні регіони, аби не отримувати сповіщення про знижки, якими неможливо скористатися.

Для отримання аналізу необхідних даних можна використовувати api. Існує декілька готових бібліотек, що виконують цю функцію.

Один із варіантів отримання таких даних є api компанії Nintendo. Таким чином, можна отримати дані про гру для конкретного регіону, знаючи ідентифікаційний номер гри.


```
{
  "personalized": false,
  "country": "JP",
  "prices": [
    {
      "title_id": 70010000009922,
      "sales_status": "onsale",
      "regular_price": {
        "amount": "1,620円",
        "currency": "JPY",
        "raw_value": "1620"
      },
      "gold_point": {
        "basic_gift_gp": "81",
        "basic_gift_rate": "0.05",
        "consume_gp": "0",
        "extra_gold_points": [],
        "gift_gp": "81",
        "gift_rate": "0.05"
      }
    }
  ]
}
```


Рисунок 1.11 Приклад відповіді від офіціного api Nintendo

Такий спосіб не є зручним, адже відсутня можливість отримати весь перелік. Також відсутня необхідна документація для розробників.

Бібліотека **nintendo-switch-eshop** має необхідний функціонал для розроблення на основі неї веб-додатку для моніторингу цін. Вона створена за допомогою node.js і має всю потрібну документацію [6].

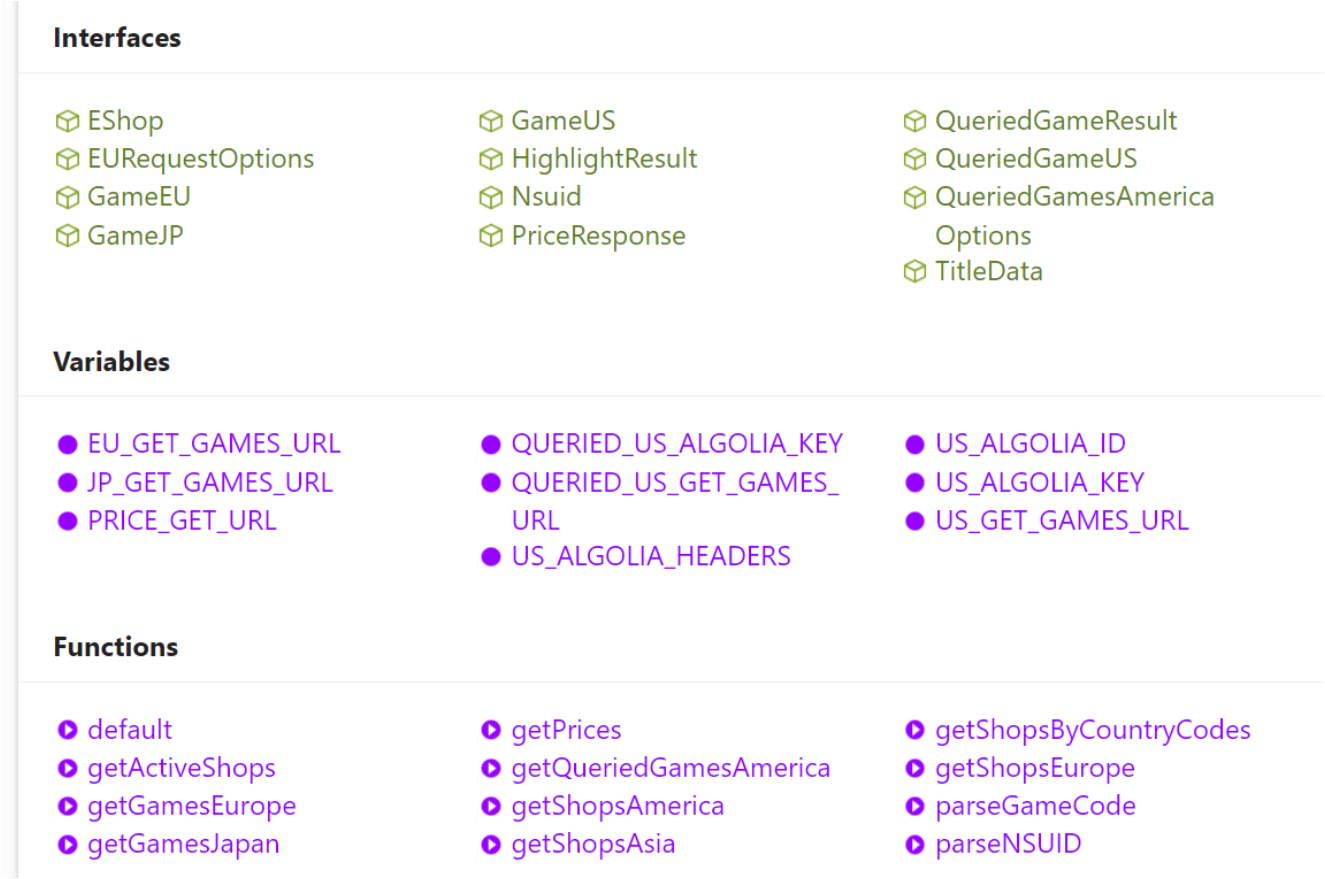

Рисунок 1.12 Документація бібліотеки nintendo-switch-eshop

Є можливість отримати список всіх товарів для конкретного регіону з їх ідентифікаційними номерами, ціною і інформацією про наявні знижки.

Отже, проблема дослідження реалізується в трьох аспектах:

- необхідно зібрати дані про всі товари для всіх регіонів у зручному для подальшої обробки форматі;
- потрібно надати користувачеві достатній функціонал для моніторингу цін не тільки в активний, але і пасивний спосіб. Повинна бути можливість перегляду найвигідніших пропозицій серед усіх регіонів, котрі задовольняють вимоги користувача. Моніторинг цін повинен

здійснюватися також системою та сповіщати користувача про бажані пропозиції;

- для сповіщення користувача необхідно реалізувати розсилку за допомогою електронної пошти.

1.2. Постановка задачі

Основною задачею дослідження є розробка інформаційно-аналітичної технології, котра допоможе користувачеві відслідковувати ціни на необхідні товари онлайн-платформи.

Для реалізації цієї задачі її умовно можна розділити на три частини. Перша – збір даних за допомогою арі. Друга – надання користувачеві можливості обирати необхідні товари та налаштовувати певні критерії. Третя – створення системи сповіщень всіх користувачів про наявність ціни на певний товар відповідно до обраних критеріїв.

Для збору даних необхідно створити окремий сервіс, що буде працювати за REST архітектурою. Інші компоненти інформаційної системи можуть звертатися для отримання необхідної інформації про конкретний товар чи всі товари одразу.

Користувачеві необхідно надати UI для зручного користування інформаційною системою. Для цього потрібно створити WEB-додаток з можливістю створення облікового запису, додавання товарів до списку бажаного та конфігурацією сповіщень (вибір регіонів, в яких треба перевіряти ціну та максимальна ціна, про яку треба сповіщати). Ідентифікацію користувача можна робити за допомогою електронної адреси. Інтерфейс повинен бути мінімалістичний і зрозумілий.

З метою сповіщення всіх користувачів про наявність пропозицій, що відповідають вказаним в особистому кабінеті критеріїв, має бути створено окремий додаток для розсилки електронних листів. Цей додаток може повинен отримувати дані з REST сервісу для збору даних. Розсилка повинна відбуватися

в визначений час, а після її здійснення додаток має завершувати роботу для економії ресурсів сервера.

Уся потрібна інформація повинна зберігатися в базі даних. Доступ до неї необхідно надати тільки WEB-додатку для запису і читання інформації, а також додатку сповіщення для читання.

Доступ до REST-додатку повинен бути тільки в рамках сервера. Усі зовнішні запити повинні бути заблоковані з ціллю економії ресурсів сервера.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Проектування архітектури інформаційної системи

Як було описано в першому розділі, інформаційна система має складатися з трьох компонентів. Така реалізація наближена до мікросервісної архітектури.

Мікросервіси — архітектурний стиль, за яким єдиний застосунок будується як сукупність невеличких сервісів, кожен з яких працює у своєму власному процесі та спілкується з рештою, використовуючи прості та швидкі протоколи передачі даних, зазвичай HTTP. Ці сервіси будуються навколо бізнес-потреб і розгортаються незалежно один від одного з використанням зазвичай повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі по собі вони можуть бути написані з використанням різних мов програмування і технологій зберігання даних [7].

Такий спосіб побудови інформаційної системи має свої переваги:

- незалежність кожного компоненту;
- простота в масштабуванні;
- простота в заміні реалізацій;
- стійкість до помилок - неправильна робота чи вихід з ладу одного компоненту не призводить до цілковитого колапсу

Як бачимо, використання мікросервісного підходу дасть змогу в подальшому розширювати систему або безперешкодно замінювати будь-які складові. Але з урахуванням того, що один з компонентів – це веб-додаток, то можна вважати що філософія мікросервісної архітектури не дотримується повністю. Цей компонент є досить великим і може бути розбитий на менші складові. Але це не є доцільно для даної інформаційної системи.

Мікросервіси мають також і недоліки. Серед них:

- складність налаштування і контролю правильності комунікації;
- дублювання коду;

- ускладнене тестування;
- ускладнене забезпечення безпечного доступу;

Для даної інформаційної системи складність комунікації між сервісами не є значним недоліком через невелику кількість окремих компонентів. Дублювання не є значним недоліком в разі створення системи одним розробником, адже навіть за наявності необхідності продублювати функціонал в іншому компонентові, вже відомо про спосіб його реалізації. Ускладнене тестування може стати помітним на останніх стадіях розробки, та це не є значною проблемою. Безпечний доступ для даної інформаційної системи можливо забезпечити за умови визначення компонентів, доступ до яких можна здійснювати з-за меж операційної системи. Таким має бути тільки веб-додаток.

Отже, наближеність до мікросервісної архітектури позитивно впливає на процес розробки та підтримки даної інформаційної системи. У подальшому використання цього принципу дозволить змінювати чи масштабувати як окремі компоненти, так і всю систему.

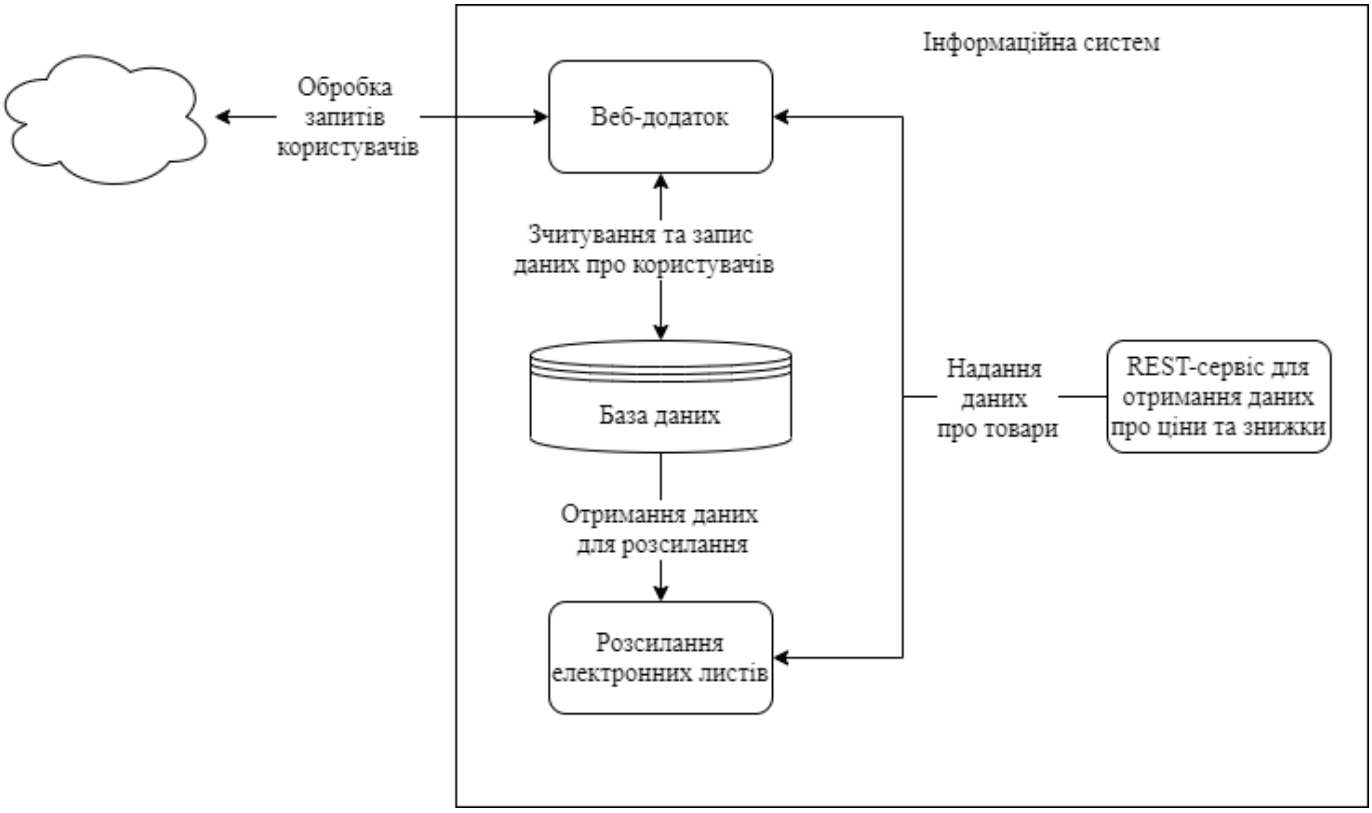

Рисунок 2.1 Архітектура інформаційної системи

Для **веб-додатку** можна використовувати шаблон проектування MVC. Модель–вигляд–контролер – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

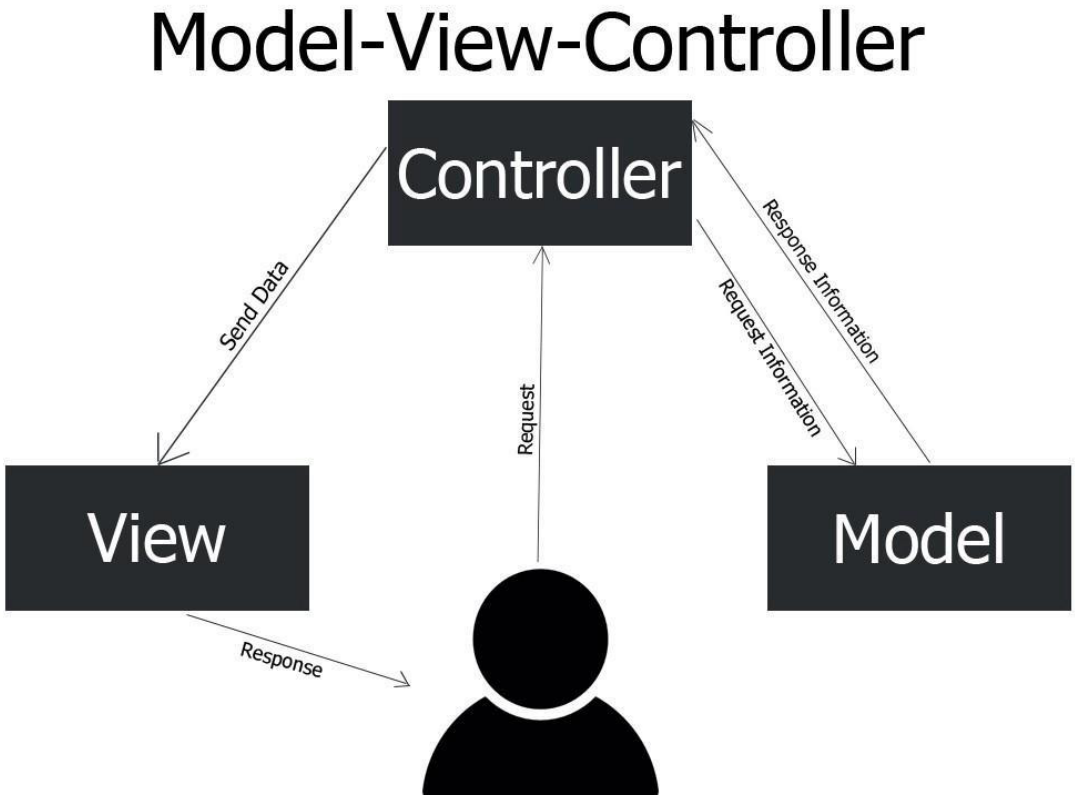

Рисунок 2.2 Схема патерну MVC

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Цей компонент буде основною складовою інформаційної системи. За допомогою веб-інтерфейсу користувач буде здійснювати реєстрацію, пошук необхідних товарів та конфігурацію розсилки.

Веб-додаток буде взаємодіяти з базою даних як для зчитування, так і для запису інформації. Також буде здійснюватися комунікація з REST-сервісом.

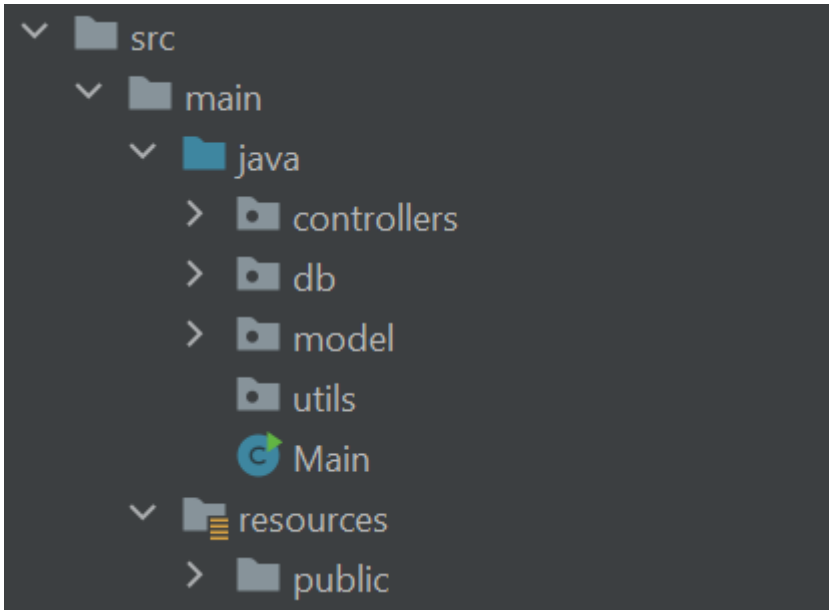

Рисунок 2.3 Структура проєкту веб-додатку

У пакеті controllers містяться всі контролери додатку. Пакет db містить необхідні для роботи з базою даних класи. Пакет utils містить необхідні інструменти для роботи додатку, такі як менеджер з'єднань з базою даних. Уся інформація з цих класів надається контролерам у вигляді екземплярів класів з пакету model. Пакет utils призначений для утилітарних класів. Клас Main є точкою входу нашого додатку. Компонент view-патерну MVC представлений пакетом public, у якому знаходяться html-файли. Саме вони і є інтерфейсом для користувача.

Оскільки тільки цей компонент буде доступний ззовні інформаційної системи, він потребує налаштування безпеки. Для будь-яких дій з даними користувача необхідно забезпечити попередню авторизацію.

Компонент інформаційної системи, призначений для **отримання інформації про товари та ціни**, може бути реалізовано з використанням REST-підходу.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST

закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє їй еволюціонувати з новими вимогами. [8].

Фактично ця частина знаходиться на межі клієнт-серверної архітектури та REST-сервісу. Звертатися до цього додатку будуть тільки два інших компоненти в односторонньому порядку.

Необхідний функціонал сервісу:

- надання списку всіх товарів онлайн-платформи eShop, включно з ідентифікаторами, посиланнями на зображення і назвою;
- надання цін та даних про знижки та їх термін дії для конкретної гри у відповідь на запит, що містить її ідентифікатор для всіх країн, що були передані в цьому запиті разом з ідентифікатором;
- надання цін та даних про знижки та їх термін дії на всі товари, ідентифікатори яких передано в запиті, для всіх країн, що були передані в запиті разом з ідентифікатором.

Для отримання цих даних компонент повинен надсилати запит чи декілька запитів до онлайн-платформи eShop. Для їх здійснення необхідно використовувати бібліотеку.

Усі запити, що будуть надходити до даного компоненту, мають бути лише від інших компонентів інформаційної системи. Це буде забезпечено на рівні операційної системи. Тож авторизація є зайвою.

У подальшому можливе масштабування цього компонента для розширення його функціональних можливостей. Також можливе створення ще одного

компоненту з таким принципом роботи, але для отримання інших даних. Наприклад:

- рейтинг конкретної гри чи списку ігор з запиту з ресурсу metacritic його альтернатив;
- час, необхідний для проходження певної гри, отриманий з ресурсу howlongtobeat.

Найпростішою складовою інформаційної системи є **додаток для розсилання електронних листів** користувачам. Цей компонент не буде працювати весь час. Необхідно, щоб один раз на день, у визначений час, здійснювався старт процесу. Протягом нього додаток збирає інформацію про всіх користувачів з бази даних. Кожен користувач має перелік ігор для моніторингу та перелік країн, у яких слідкувати за цінами непотрібно.

Якщо ціна на конкретну гру нижча хоча б в одному регіоні – вона буде додана до електронного листа. Дані про конкретний товар будуть отримуватися з REST-сервісу. Фактично, запитів до цього сервісу буде стільки ж, скільки користувачів з хоча б однією доданою для відслідковування грою в системі. Це може бути обмеженням у роботі системи за занадто значної кількості користувачів. Тож для даного компоненту необхідно налаштувати детальне логування часу роботи і підрахувати максимальний час роботи компоненту для одного користувача.

Забезпечення авторизації не має сенсу з урахуванням того, що додаток працює тільки на зчитування даних з бази та отримання від іншого компоненту. Вхідні запити просто не будуть оброблені.

Загальна архітектура інформаційної системи на основі мікросервісів дасть змогу додавати нові компоненти в майбутньому. Та окреслені вище сервіси зможуть забезпечити базовий функціонал при достатній гнучкості і простоті масштабування. Перераховані недоліки підходу не є істотною проблемою для даної інформаційної системи, у той час як переваги позитивно впливають на процес проєктування, розробки та підтримки.

2.2. Проєктування бази даних

Для роботи більшості додатків у сучасному світі необхідна база даних для збереження і зручної роботи з інформацією. У наш час існує багато варіантів СУБД.

Система управління базами даних – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних [9].

Для реалізації бази даних інформаційної системи слід використовувати реляційну модель. Реляційна база даних є сукупністю елементів даних, організованих у вигляді набору формально описаних таблиць, з яких дані можуть бути доступними або повторно зібрані багатьма різними способами без необхідності реорганізації таблиць бази даних [10].

Інформаційну систему спроектовано так, щоб вибір СУБД можна було здійснити вже під час розробки, а в разі необхідності перехід до СУБД не викликав значних проблем. Це може знадобитися при масштабуванні системи.

При проєктуванні бази даних слід привести структуру до нормальної форми за необхідності. Перехід до вищої нормальної форми не повинен ускладнювати типові запити до бази даних від додатків. Можлива надмірність може бути меншою проблемою, ніж заскладні запити.

Для проєктування бази даних інформаційної системи є створення ERD діаграми. Цей вид діаграми описує сутності та зв’язки між ними. На основі такої діаграми можна безпосередньо створити таблиці бази даних.

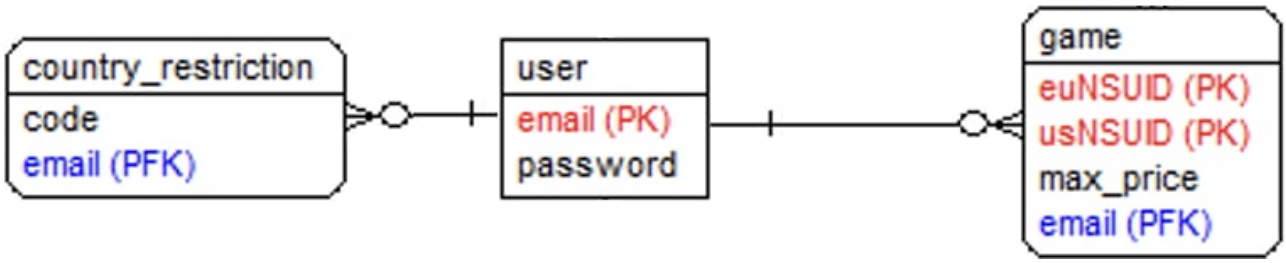

Рисунок 2.4 ERD-діаграма

Таким чином, діаграма містить три сутності. Опис цих сутностей необхідний для більш чіткої характеристики бази даних.

Таблиця 2.1 Опис сутностей діаграми ERD

Сутність	Опис сутності	Таблиця	Поле	Опис поля
1	2	3	4	5
User	Сутність користувача	users	email	Електронна адреса користувача, одночасно є унікальним ідентифікатором користувача
			password	Пароль користувача в зашифрованому вигляді
Country restriction	Зберігає дані про країни, в яких немає необхідності моніторити ціни і знижки	country_restrictions	code	Код країни
			email	Електронна адреса користувача (фактично ідентифікатор користувача)
Game	Гра, додана користувачем для відслідковування	games	euNSUID	Код гри в європейському регіоні
			usNSUID	Код гри в американському регіоні
			max_price	Верхня межа ціни, починаючи із якої необхідно сповіщати
			email	Електронна адреса користувача (фактично ідентифікатор користувача)

2.3 Планування життєвого циклу додатку

Життєвий цикл всієї інформаційно-аналітичної системи можна поділити на такі фази:

1. Аналіз проблеми та обґрунтування доцільності розробки інформаційної системи.
2. Проєктування складових інформаційної системи.
3. Розробка інформаційної системи.
4. Розгортання інформаційної системи на сервері.
5. Підтримка інформаційної системи.

Аналіз проблеми та доцільності розробки було здійснено в першому розділі. У результаті аналізу доведено, що існуючі рішення не є повними і мають недоліки.

Проєктування складових інформаційної системи було здійснено в другому розділі. В результаті цього кроку стає зрозумілою загальна архітектура інформаційної системи та обираються інструменти та засоби, що будуть використані під час розробки.

Під час **створення інформаційної системи** можна використовувати методології розробки, якщо це доцільно. Результатом цього кроку має бути готовий до наступного пункту продукт.

Для **розгортання інформаційної системи** на сервері необхідно заздалегідь підготувати інфраструктуру. Налагодити всі функції безпеки потрібно до повного запуску системи. Запланований запуск компоненту розсилки електронних листів необхідно налагодити на рівні операційної системи, що дасть змогу зекономити ресурси сервера.

На стадії **підтримки системи** важливу роль відіграє логування додатку і всіх помилок. Необхідно налаштувати моніторинг активності веб додатку та роботи всіх компонентів. У разі виникнення проблем це дасть змогу швидко реагувати. Також можна налаштувати форму зворотного зв’язку для користувачів.

2.4 Аналіз існуючих технологій для рішення задачі

Для рішення даної задачі необхідно проаналізувати можливі опції та вибрати необхідний варіант для:

- мова програмування кожного з компонентів;
- фреймворк для веб-додатку;
- СУБД;
- системи збирання додатків;
- операційна система серверу.

Мовою програмування для додатку розсилки та веб-додатку буде слугувати Java. Це зумовлено простотою написання коду та кросплатформеністю додатків. Також існує достатній вибір фреймворків для веб-додатку, написаних цією мовою. REST-сервіс буде створено з використанням Node.js – фреймоврком для мови програмування javascript. Це зумовлено тільки відсутністю альтернатив серед бібліотек, що дозволяють працювати з онлайн-платформою Nintendo.

Серед великої безлічі веб-фреймовркв для поданих задач підійде Spark. Це мікрофреймворк, що надасть всі необхідні функції для інформаційної системи. Але він є максимально простим у використанні, має структуровану документацію і використовує мінімум ресурсів сервера.

Серед значної кількості СУБД можна обрати SQLite як найпростіший в конфігурації варіант. Ця СУБД не потребує встановлення нових додатків в операційну систему і є фактично файлом, який містить всю інформацію. Ця СУБД не розрахована на велику кількість одночасних підключень. Тому у випадку масштабування системи можна перейти на PostgreSQL. Така міграція потребуватиме розширення потужності сервера, але не буде складною з урахуванням подібності синтаксисів цих СУБД. У той час налаштування резервного копіювання, доступів і підключень до SQLite є дуже простим, що зменшить час на розробку.

Для збирання Java-додатків очевидним варіантом є Maven, який на даний момент є один з найпопулярніших засобів для збирання застосунків та керування

підключеними бібліотеками. Для збирання додатку на Node.js достатньо використовувати стандартні можливості фреймоврка та пакетного менеджера npm.

Операційною системою можна обрати Ubuntu. Більшість варіантів Linux-дистрибутивів для серверів відрізняються е використанні ресурсів не значною мірою. Тож основним аргументом на користь саме цього дистрибутиву є документованість та розповсюдженість.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ

3.1. Реалізація REST-сервісу

Насамперед реалізовано REST-сервіс. Він не залежить від інших, але є необхідним для тестування складових системи.

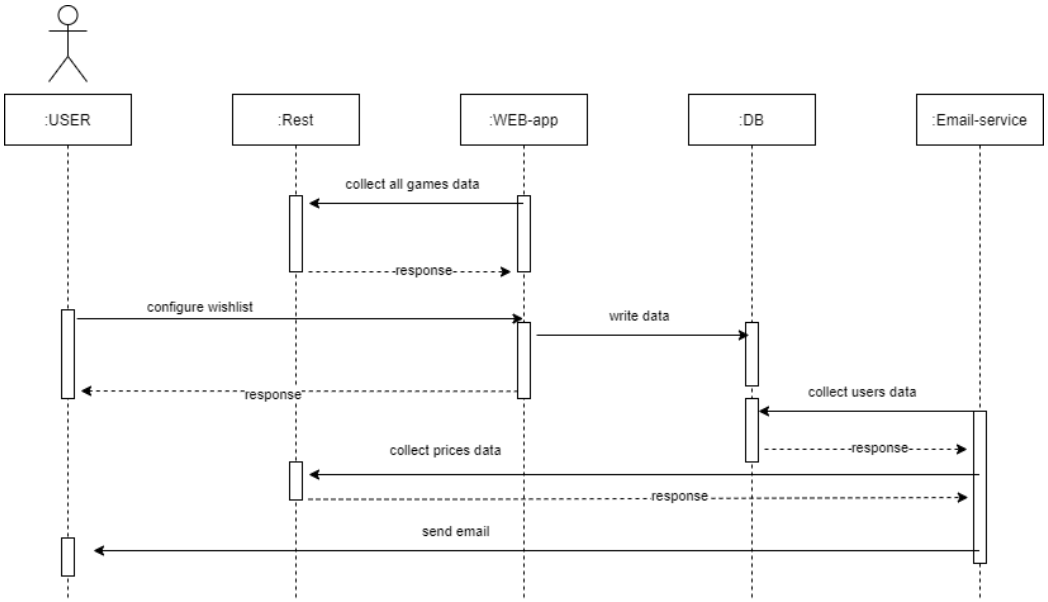

Рисунок 3.1 Діаграма послідовності

Для реалізації обрано мову програмування JavaScript з використанням фреймворка NodeJS. Фактично було створено веб-додаток, який буде відповідає на три можливих HTTP-запити:

- GET-запит «*/games*» – повертає дані про всі ігри, а саме: назву, посилання на зображення тощо. Приклад запиту та результату:

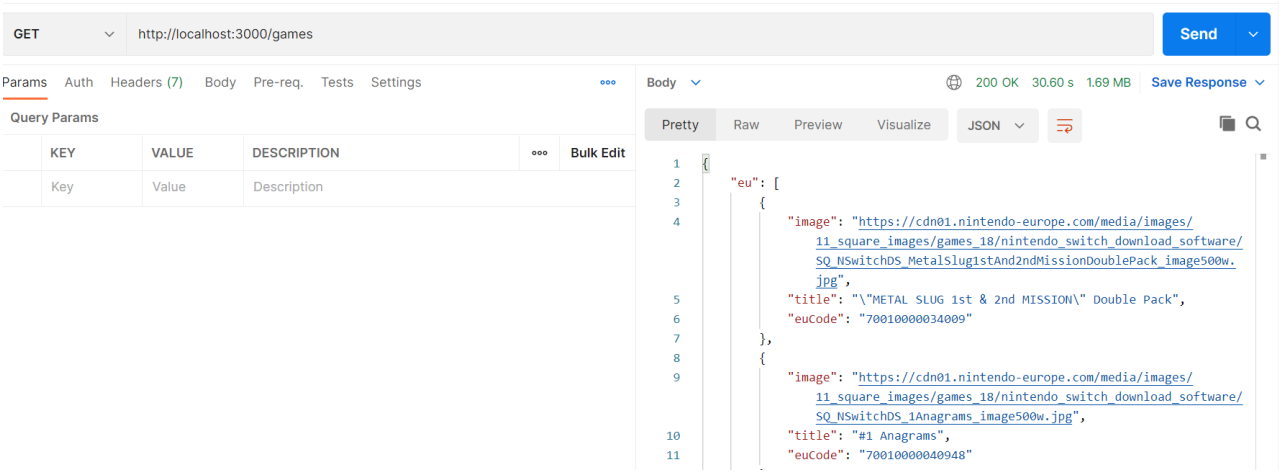

Рисунок 3.2 Приклад запиту /games

Як бачимо, час відповіді приблизно 30 секунд. Це зумовлено значною кількістю даних. В онлайн-магазині є приблизно 6500 ігор, а дані збираються двічі – окремо для європейського й американського регіонів.

- POST-запит «*/getEuGames*» – повертає дані про ціни на ігри, ідентифікатори яких було передано у вигляді JSON. Приклад запиту та результату:

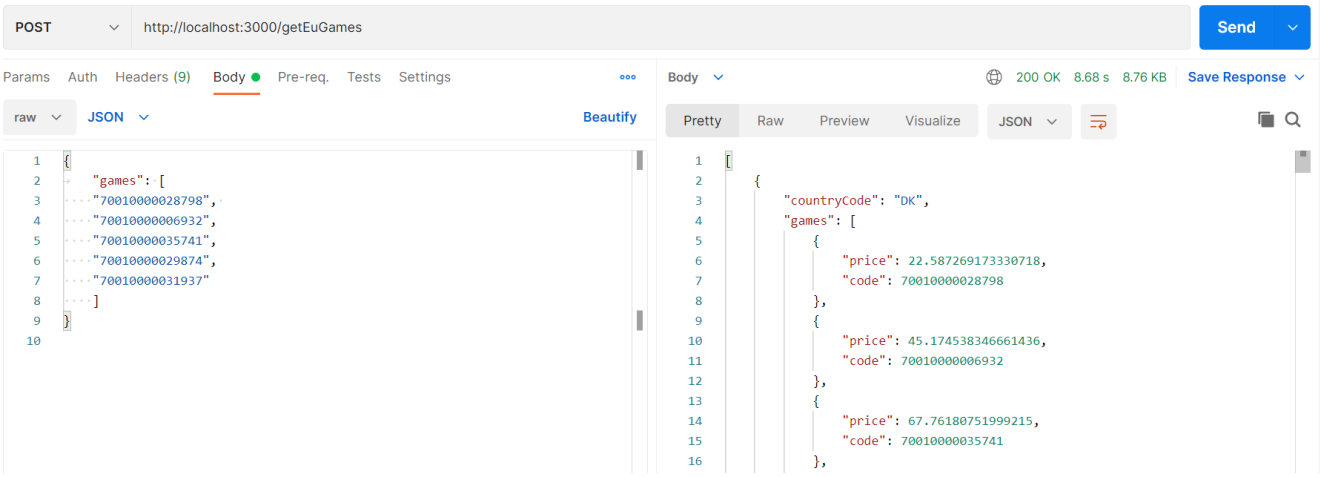

Рисунок 3.3 Приклад запиту /getEuGames

Час відповіді складає приблизно 8 секунд. Це зумовлено тим, що для переліку ігор робиться окремий запит у кожну країну регіону. Також було створено штучну затримку між кожним запитом для уникнення 403 помилки. Вибір POST-методу зумовлено зручністю передачі даних запиту.

- POST-запит «*/getUsGames*» – аналогічний за структурою запиту до попереднього, час відповіді рівноцінний. Зумовлено тією ж проблемою – уникненням помилки, що виникає при великій кількості одночасних запитів.

Оскільки ціни на конкретний товар API повертає в національній валюті, було використано веб-сервіс exchangeratesapi.io. Через обмеження безкоштовної версії курс валют оновлюється щодня. Для потреб даної інформаційної системи така точність є достатньою для кінцевого користувача.

Загальна структура сервісу досить проста і складається з одного js-файлу та допоміжних файлів та директорій, необхідних для роботи NodeJS.

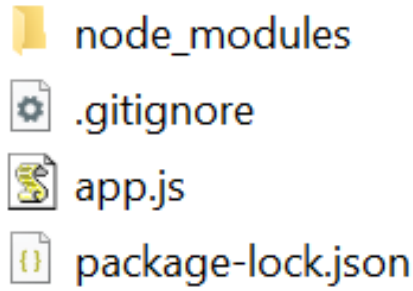

Рисунок 3.4 Структура проєкту

Для розгортання даного веб-додатку достатньо перенести цю структуру на необхідний сервер зі встановленим NPM та NodeJS. Збирання додатку відсутнє.

3.2. Реалізація веб-додатку

Веб-додаток є основною складовою інформаційної системи і взаємодіє з REST-сервісом через HTTP-протокол. Тож реалізація цього компоненту можлива тільки з наявності способу отримання даних про ігри.

На старті роботи додаток перевіряє наявність підключення до бази даних. Так як в реалізації інформаційної системи використовується SQLite, то фактично це зводиться до перевірки наявності файлу бази даних. У випадку відсутності файлу необхідно його створити та ініціалізувати базову структуру БД – усі необхідні таблиці.

Наступним кроком є отримання деталей про всі доступні ігри в американському та європейському регіонах. Веб-додаток робить запит до REST-сервісу, оброблює вхідні дані та зберігає їх у пам’яті. Цей процес необхідно повторювати регулярно для своєчасного оновлення бібліотеки ігор. Зберігання у пам’ять, а не в базу даних, відбувається через наявність чіткої кількості ігор. Оскільки ми знаємо кількість і тип даних, ми можемо зберігати їх не в базі даних, займаючи відповідний об’єм ОЗУ. Для розробленої системи такий варіант є прийнятим. У той же час, зберігання в пам’яті пришвидшує процес отримання

даних про конкретну гру – немає потреби звертатися до бази даних для отримання інформації.

Користувач робить усі налаштування через веб-додаток, але в першу чергу необхідно створити особистий профіль. Для цього необхідна електронна пошта та пароль. Після підтвердження пошти (введення одноразового коду, що був на неї відправлений) у базі даних буде створено нового користувача. Пароль зберігається в зашифрованому за допомогою BCrypt вигляді. Електронна пошта є ідентифікатором користувача, тож вона має бути унікальна.

Для пошуку ігор було створено головну сторінку. Потрапити на неї можна натиснувши на логотип. На сторінці є поле пошуку, де необхідно ввести назву. Ігри, що підходять за назвою, будуть з'являтися на сторінці без необхідності перезавантаження. Після цього необхідно натиснути на потрібну гру, що призведе до відкриття діалогового вікна.

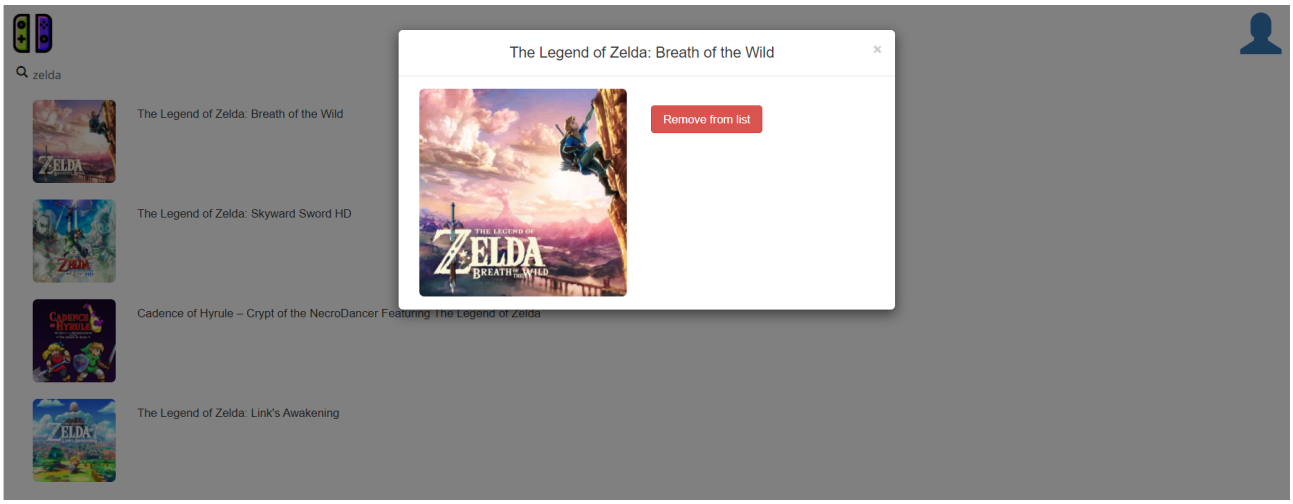

Рисунок 3.5 Приклад діалогового вікна головної сторінки

У діалоговому вікні може бути доступне поле для введення та кнопка «додати» або кнопка «видалити». Перший варіант відображається у випадку, коли гра ще не додана до бібліотеки. Можна ввести ціну гри, вище якої користувач не хоче отримувати повідомлення. Також це поле можна залишити порожнім – у такому разі кожного разу користувач буде отримувати повідомлення з найвигіднішою ціною. Після натискання «додати» гра буде додана до бібліотеки і з'явиться кнопка «видалити» замість попередньої.

Оскільки цю сторінку можливо відвідувати без попередньої авторизації, при натисканні на кнопку «додати» користувача буде перенаправлено на сторінку входу. Якщо ж гра вже була в бібліотеці, то кнопка «видалити» буде доступна одразу.

Для зручного перегляду та налаштування списку країн-винятків було створено особистий кабінет. Потрапити в нього можна натиснувши на іконку користувача в верхньому правому кутку. Якщо користувач ще не авторизувався – його буде перенаправлено на відповідну сторінку.

Особистий кабінет складається з двох секцій. Перша містить перелік країн, про ціни в яких користувача повідомляти не потрібно.

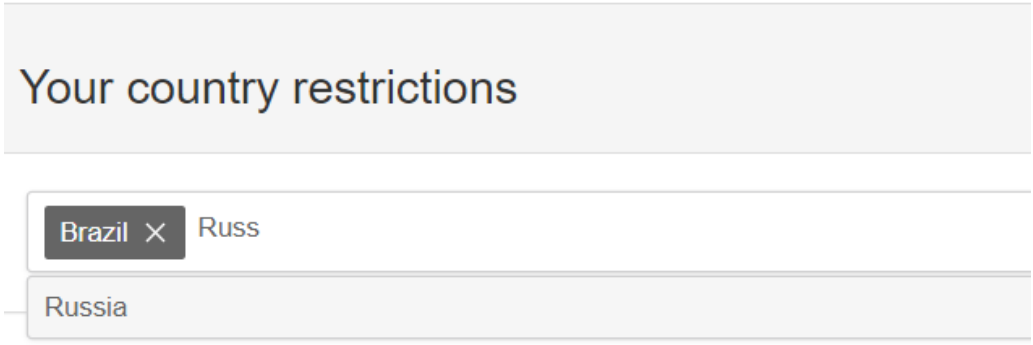

Рисунок 3.6 Вигляд секції для вибору країн-винятків

Друга – зі списком ігор користувача. Кожна гра містить поле введення з поточною ціною, кнопку «зберегти» та кнопку для видалення.

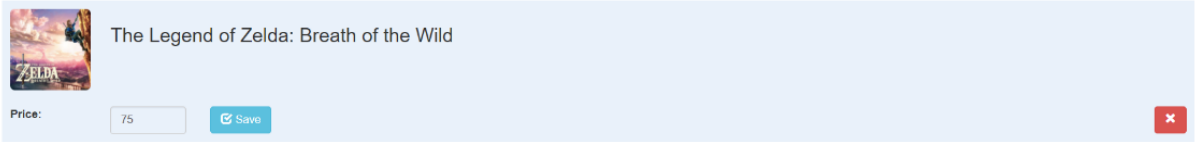

Рисунок 3.7 Приклад гри в особистому кабінеті

3.3. Реалізація додатку розсилання електронних листів

Компонент для розсилання електронних листів залежить напряму від двох попередніх. Для його роботи необхідно доступ до даних про ціни на ігри (тобто працюючий REST-сервіс) та наявна інформація в базі даних. Тож цей компонент було реалізовано в останню чергу.

Першим кроком роботи додатку є отримання даних про всіх користувачів та їх ігри. Для цього необхідно виконати декілька SQL-запитів та зберегти ці дані в пам’ять.

Наступним кроком є отримання даних про ціни на ігри. Для цього необхідно зробити два запити (для європейського та американського регіонів). Ідентифікаційні номери ігор отримуємо з попереднього кроку.

Також необхідно отримати дані про ігри (такі як назва) для подальшої обробки. У базі даних зберігається тільки ідентифікатор гри, тож для формування електронного листа це потрібно.

Наступним кроком є формування електронних листів та їх відправлення. Для надсилання використовується бібліотека simplejavamail. Кожному користувачу буде надіслано лист з цінами, але тільки у разі наявності відповідних пропозицій. Враховується як максимальна ціна, так і обмеження по країнах.

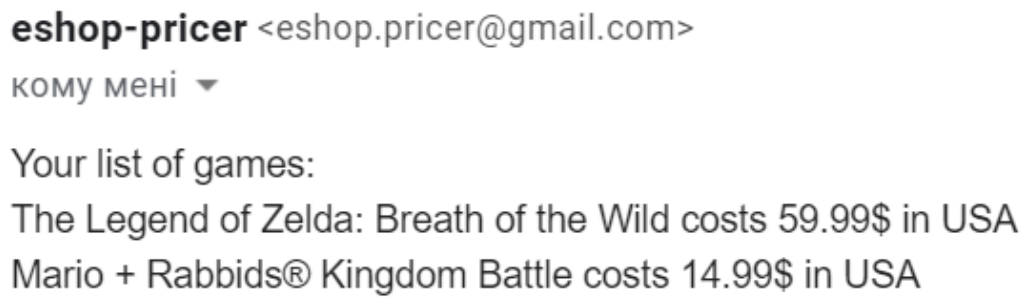

Рисунок 3.8 Приклад листа з цінами на ігри

Лист формується у вигляді звичайного тексту. У подальшому є можливість змінити оформлення шаблону.

3.4. Розгортання інформаційної системи

Результатом дослідження є готова функціонуюча інформаційна система. У першу чергу для цього необхідне середовище для її розгортання. Для цього

було обрано виділений сервер під керуванням ОС Ubuntu. Технічні характеристики сервера обрані відповідно до вимог компонентів.

Насамперед встановлення всього необхідного ПЗ, а саме:

- OpenJDK;
- NodeJS;
- Nginx;
- NPM.

Наступним кроком був запуск кожного з компонентів та налаштування безпеки. Для цього було закрито відповідні порти на налаштовано Nginx так, щоб відповідні запити на 80 порт перенаправлялися на необхідний нам порт веб-додатку. Таким чином, ми контролюємо доступ до файлів та решти компонентів інформаційної системи.

Після того, як вся інформаційна система працює, було налаштовано DNS для використання доменного імені. Це зроблено внутрішніми засобами платформи GoDaddy. У результаті доступ до веб-додатку можна отримати за адресою eshoperer.com.

Наступним кроком є забезпечення безпечного з’єднання для користувача. Для отримання HTTPS-з’єднання нам необхідно згенерувати SSL-сертифікат. Це було здійснено за допомогою утиліти certbot. Фактично згенерований сертифікат є безкоштовним сертифікатом від Let’s Encrypt.

Останнім кроком є налаштування задачі cron для регулярної роботи сервісу розсилання листів.

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
10 01 * * * /home/admin/java/runEmail.sh >/home/admin/java/mailCron.log 2>&1
```

Рисунок 3.9 Запис в cron

Розсилка буде здійснюватися щоночі о 01:10 за часом сервера. Усі отримані логи будуть направлені в файл mailCron.log.

Результатом деплою є повністю працююча інформаційна система. Структура файлів на сервері розділяє додатки на дві групи: ті, що мають доступ до бази даних, а також на ті, що такого доступу не мають.

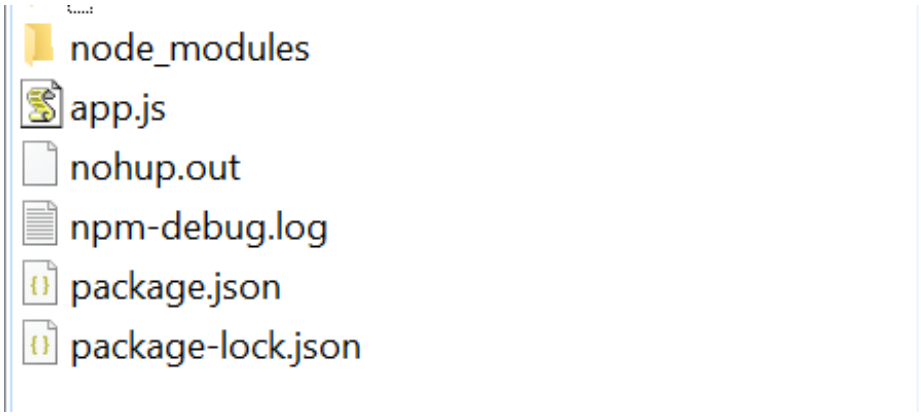

Рисунок 3.10 Структура директорії Node сервера

У директорії Node зберігаються всі необхідні файли для роботи REST-сервісу. Також є файл nohup.out, котрий містить логи програми.

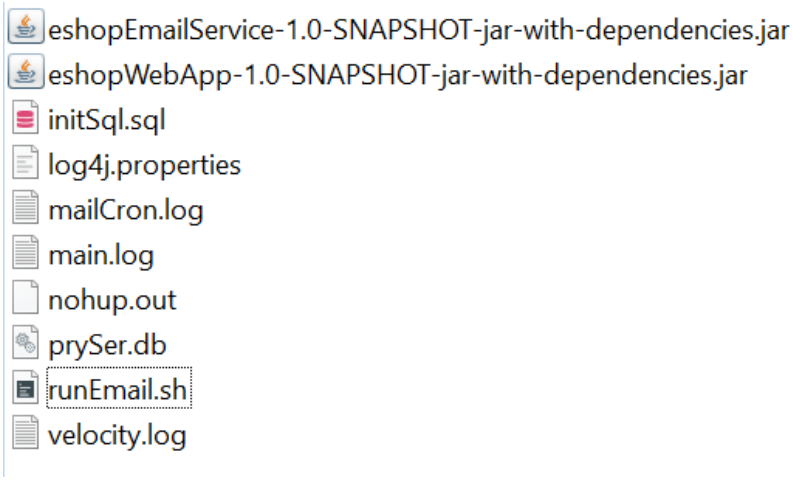

Рисунок 3.11 Структура директорії Java сервера

У директорії Java містяться всі необхідні файли для роботи додатку для розсилання та веб-додатку. Також є файл бази даних, скрипт для ініціалізації бази даних, логи та скрипт для запуску сервісу розсилки.

3.5 Аналіз ефективності

Найпростішим способом аналізу ефективності рішення проблеми даного дослідження є розрахунок заощаджень. Подані дані є актуальними на момент написання роботи, однак репрезентативні і для цілісного аналізу.

Для аналізу ефективності було взято три гри і пораховано їх найбільшу і найменшу вартість. Так, 3D Arcade Fishing коштує 350 гривень у чилійському регіоні. У той же час в швейцарському вона коштує 620 гривень. Економія становить 330 гривень або 42%.

Гра This War of Mine: Complete Edition на момент написання роботи має акційну ціну в 80 гривень в декількох країнах. У той же час в США акція відсутня, тож товар коштує 1090 гривень. Економія складає 1010 гривень, або 92%.

Аналогічна ситуація з State of Mind. У мексиканському регіоні цей товар коштує 22 гривні адже бере участь у розпродажу. Ціна без акції в Росії становить 664 гривні. Економія складає 642 гривні або 96%.

Як бачимо, ефективність залежить від наявності акції на той чи інший товар. Економія може бути незначною для деяких товарів. У той же існує ймовірність зекономити до понад 90%, використовуючи інформаційну систему.

ВИСНОВКИ

У результаті аналізу проблеми дослідження з’ясовано, що існують аналоги інформаційної технології, що дають змогу слідкувати за цінами на товари онлайн-платформи Nintendo, такі як Eshop-prices, Doku Deals та інші. Усі інформаційні системи мають схожий функціонал. Найпопулярніші варіанти проаналізовано, визначено їх переваги та недоліки. У результаті аналізу з’ясовано, що всі розглянуті додатки не мають необхідного повного функціоналу.

Проаналізувавши проблему, було сформовано задачу дослідження. Необхідно спроєктувати та розробити інформаційно-аналітичну технологію, котра буде дозволяти користувачеві проводити моніторинг цін на товари онлайн-платформи по всім можливим регіонам з можливістю отримувати сповіщення за допомогою електронної пошти.

Розроблено подальший план життєвого циклу інформаційної системи, який включає в себе кроки від аналізу до післярелізної підтримки.

Спроєктовано архітектуру інформаційної технології. Для зручності було обрано мікросервісний тип. Інформаційна система має поділятися на три компоненти: веб-додаток, додаток для розсилання електронних листів та REST-сервіс для отримання даних про товари. Результатом проєктування бази даних є ERD-діаграма. На її основі в подальшому буде написано скрипт для ініціалізації бази даних. Для кожного з компонентів інформаційної системи обрано необхідні технології для розробки. Також обрано СУБД, яка буде використовуватися в роботі.

Реалізовано інформаційно-аналітичну систему у вигляді трьох компонентів. REST-додаток реалізовано з використанням мови JavaScript та фреймворку NodeJS. Реалізовано три основних типи запитів – для отримання даних про ігри, для отримання цін на ігри в європейських та американських країнах. Веб-додаток створено з використанням мови програмування Java та фреймворку Spark. Цей компонент надає всі необхідні функції для кінцевого

користувача, такі як вибір ігор, гранично прийнятних цін та країн-виключень. Додаток для розсилання електронних листів реалізовано з використанням Java.

Усі компоненти інформаційної системи розміщено на сервері. Зроблено відповідні мережеві налаштування для безпечної роботи додатків. Налаштовано DNS та створено SSL-сертифікат для HTTPS-з’єднання. Готовий прототип можна переглянути за посиланням https://eshoperer.com

Отже, мети дослідження досягнуто в повному обсязі. Інформаційно-аналітична технологія реалізована у відповідності до завдань дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nintendo eShop. URL : https://www.nintendo.com
2. Deku Deals. URL : https://www.dekudeals.com
3. Nintendo eShop-prices URL : https://eshop-prices.com
4. Switch games price tracker URL : https://switchgames.io
5. Current & Historic Prices For Every Video Game. Prices for loose, complete, and brand new condition. URL : https://www.pricecharting.com
6. Nintendo Switch Eshop API URL : https://lmmfranco.github.io/nintendo-switch-eshop/
7. Microservices (a definition of this new architectural term) URL : https://martinfowler.com/articles/microservices.html
8. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. URL : https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
9. Avi Silberschatz, Henry F. Korth, S. Sudarshan. Database System Concepts. Database System Concepts. URL : https://www.db-book.com
10. Date C.J. An Introduction to Database Systems. 2004. URL : https://docs.google.com/file/d/0B9aJA_iV4kHYR1I1Q1MxQ2VzX0U/edit?resourcekey=0-m-SoWfxx0CbK6tjYrMttow
11. How To Secure Nginx with Let's Encrypt on Ubuntu 20.04. URL : https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-20-04
12. How to setup cron jobs in Ubuntu. URL : https://www.geeksforgeeks.org/how-to-setup-cron-jobs-in-ubuntu/
13. NodeJS. Документація. URL : https://nodejs.org/uk/docs/
14. Documentation. URL : https://www.sqlite.org/docs.html
15. How to Configure NGINX. URL : https://www.linode.com/docs/guides/how-to-configure-nginx/
16. Spark. Documentation. URL : https://sparkjava.com/documentation

17. Java Documentation. URL : https://docs.oracle.com/en/java/
18. npm Docs. URL : https://docs.npmjs.com
19. Official Ubuntu Documentation. URL : https://help.ubuntu.com
20. Довідковий центр. URL : https://ua.godaddy.com/help

ДОДАТКИ

Додаток А. Програмний код REST-сервісу

```
const hostname = '127.0.0.1';
const port = 3000;
const express = require('express')
const app = express()
var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended: true
}));
const nintendoSwitchEshop = require('nintendo-switch-eshop');
const http = require('http');
const https = require('https');
const DELAY = 250;
app.get('/games', (req, res) => {
    let pr = nintendoSwitchEshop.getGamesEurope();
    pr.then((value) => {
        let euGames = [];
        value.forEach(element => {
            let editedElement = {};
            editedElement.image = element.image_url_sq_s;
            editedElement.title = element.title;
            editedElement.euCode = element.nsuid_txt + "";
            euGames.push(editedElement)
        })

        let usPromice = nintendoSwitchEshop.getGamesAmerica();
        usPromice.then((value) => {
```

```
            let usGames = [];
            value.forEach(element => {
                let editedElement = {};
                editedElement.title = element.title;
                editedElement.usCode = element.nsuid;
                usGames.push(editedElement)
            })

            let result = {};
            result.eu = euGames;
            result.us = usGames;
            res.send(result);
        })

    })
})
app.post('/getEuGames', async (req, res) => {
    let euGameIds = req.body.games;
    let desiredCurr = req.body.currency;
    if (desiredCurr == null) desiredCurr = "USD"
    console.log(euGameIds);
    let euCountries = ['DK', 'EE', 'FI', 'IE', 'LV', 'LT', 'NO', 'SE', 'GB', 'HR', 'CY', 'GR',
'IT', 'MT', 'PT', 'SI', 'ES', 'BG', 'CZ', 'HU', 'PL', 'RO', 'RU', 'SK', 'AT', 'BE', 'FR', 'DE',
'LU', 'NL', 'CH', 'AU', 'NZ', 'ZA'];
    let promises = [];
    let resArr = [];
    let i = 1;
    euCountries.forEach(countryCode => {
        let delay = DELAY * i;
```

```
        promises.push(delayMyPromise(nintendoSwitchEshop.getPrices(countryCode,
euGameIds), delay))
        i += 1;
    })
    return Promise.all(promises).then(results => {
        results.forEach(r => {
            let element = {}
            element.countryCode = r.country;
            let games = [];
            r.prices.forEach(pr => {
                if (pr.sales_status == "onsale") {
                    let game = {};
                    let price;
                    let curr;
                    if (pr.discount_price != null) {
                        price = pr.discount_price.raw_value;
                        curr = pr.discount_price.currency;
                        game.end_date = pr.discount_price.end_datetime;
                    } else {
                        price = pr.regular_price.raw_value;
                        curr = pr.regular_price.currency;
                    }
                    let usdPrice = parseFloat(price) / parseFloat(rates[curr]) *
parseFloat(rates[desiredCurr]);
                    game.price = usdPrice;
                    game.code = pr.title_id;
                    games.push(game)
                }
            })
```

```
            element.games = games;
            resArr.push(element)
        })
        res.send(resArr);
    })
})
app.post('/getUsGames', async (req, res) => {
    let usGameIds = req.body.games;
    let desiredCurr = req.body.currency;
    if (desiredCurr == null) desiredCurr = "USD"
    console.log(usGameIds);
    let usCounties = ['BR', 'CA', 'MX', 'US']
    let promises = [];
    let resArr = [];
    let i = 1;
    usCounties.forEach(countryCode => {
        let delay = DELAY * i;
        promises.push(delayMyPromise(nintendoSwitchEshop.getPrices(countryCode,
usGameIds), delay))
        i += 1;
    })
    return Promise.all(promises).then(results => {
        results.forEach(r => {
            let element = {}
            element.countryCode = r.country;
            let games = [];
            r.prices.forEach(pr => {
                if (pr.sales_status == "onsale") {
                    let game = {};
```

```
                let price;
                let curr;
                if (pr.discount_price != null) {
                    price = pr.discount_price.raw_value;
                    curr = pr.discount_price.currency;
                    game.end_date = pr.discount_price.end_datetime;
                } else {
                    price = pr.regular_price.raw_value;
                    curr = pr.regular_price.currency;
                }
                let usdPrice = parseFloat(price) / parseFloat(rates[curr]) *
parseFloat(rates[desiredCurr]);
                game.price = usdPrice;
                game.code = pr.title_id;
                games.push(game)
            }
        })
        element.games = games;
        resArr.push(element)
    })
    res.send(resArr);
  })
})
function delayMyPromise(myPromise, myDelay) {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
        console.log(myDelay + " delayed")
        return resolve(myPromise);
    }, myDelay);
```

```
    });
}
let rates = null;
let lastTime = new Date();
let usd = null;
function getRates() {
    lastTime.setHours(0);
    lastTime.setMinutes(0);
    lastTime.setSeconds(0);
    if (rates == null || (new Date() - lastTime) > (60 * 60 * 1000 * 23)) {
        makeRequest();
    }
}
function makeRequest() {
    http.get('http://api.exchangeratesapi.io/v1/latest?access_key=', (resp) => {
        let data = '';
        resp.on('data', (chunk) => {
            data += chunk;
        });
        resp.on('end', () => {
            rates = JSON.parse(data).rates;
        });
    }).on("error", (err) => {
        console.log("Error: " + err.message);
    });
}
app.listen(port, () => {
    console.log(`Example app listening at http://localhost:${port}`)
    getRates();
```

})

Додаток Б. Програмний код веб-додатку

```
package controllers;

import db.SQLiteManager;
import model.Game;
import model.SelectedCountry;
import model.User;
import org.json.simple.JSONObject;
import spark.ModelAndView;
import spark.Route;
import spark.template.velocity.VelocityTemplateEngine;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

public class CabinetController {
    private static SQLiteManager sqliteManager = SQLiteManager.getInstance();

    public static Route getCabinetPage = (request, response) -> {
        if(request.session().attribute("user")==null) {
            response.redirect("/login");
            return null;
        }
        String email = ((User) request.session().attribute("user")).getEmail();
        //String email = "test@gmail.com";
```

```
        ArrayList<Game> games = sqliteManager.getGames(email);
        ArrayList<Game> filledGames = new ArrayList();
        for (Game game : games) {
            System.out.println(game.getEuCode()+" "+game.getUsCode());
        }
        Collection<JSONObject> jsonGames =
MainPageController.gamesMap.values();
        for (Game game : games) {
            for (JSONObject jsonGame : jsonGames) {

if(game.getEuCode().equals(jsonGame.get("euCode"))&&jsonGame.get("usCode")=
=null){
                    game.setTitle((String) jsonGame.get("title"));
                    game.setImageRef((String) jsonGame.get("image"));
                    game.setUsCode("undefined");
                    filledGames.add(game);
                }
                else if (game.getUsCode().equals(jsonGame.get("usCode")) &&
                        game.getEuCode().equals(jsonGame.get("euCode"))) {
                    game.setTitle((String) jsonGame.get("title"));
                    game.setImageRef((String) jsonGame.get("image"));
                    filledGames.add(game);
                }
            }
        }
        Map<String, Object> model = new HashMap<>();
        System.out.println(sqliteManager.getCountryRestrictions(email));
        model.put("codes",
generateSelect(sqliteManager.getCountryRestrictions(email)));
```

```
    model.put("games", filledGames);
    return new VelocityTemplateEngine().
            render(new ModelAndView(model, "public/cabinet.html"));
};

public static Route addCountryRestriction = (request, response) -> {
    if(request.session().attribute("user")==null) {
        return "no auth";
    }
    String email = ((User) request.session().attribute("user")).getEmail();
    //String email = "test@gmail.com";
    String code = request.queryParams("code");
    sqliteManager.addCountryRestriction(email, code);
    return "OK";
};

public static Route removeCountryRestriction = (request, response) -> {
    if(request.session().attribute("user")==null) {
        return "no auth";
    }
    String email = ((User) request.session().attribute("user")).getEmail();
    //String email = "test@gmail.com";
    String code = request.queryParams("code");
    sqliteManager.removeCountryRestriction(email, code);
    return "OK";
};

public static Route updatePrice = (request, response) -> {
    if(request.session().attribute("user")==null) {
```

```
            return "no auth";
        }
        String email = ((User) request.session().attribute("user")).getEmail();
        //String email = "test@gmail.com";
        String euCode = request.queryParams("euCode");
        String usCode = request.queryParams("usCode");
        String newPrice = request.queryParams("newPrice");
        try {
            Integer.parseInt(newPrice);
        } catch (Exception e) {
            return "wrong max price";
        }
        sqliteManager.updatePrice(email, euCode, usCode, newPrice);
        return "OK";
    };


    private static ArrayList<SelectedCountry> generateSelect(ArrayList<String>
countryList) {
        String countryCodes[] = {"DK", "EE", "FI", "IE", "LV", "LT", "NO", "SE",
"GB", "HR", "CY", "GR", "IT", "MT", "PT", "SI", "ES", "BG", "CZ", "HU", "PL",
"RO", "RU", "SK", "AT", "BE", "FR", "DE", "LU", "NL", "CH", "AU", "NZ", "ZA",
"BR", "CA", "MX", "US"};
        String countryNames[] = {
                "Denmark",
                "Estonia",
                "Finland",
                "Ireland",
                "Latvia",
```

"Lithuania",
"Norway",
"Sweden",
"United Kingdom",
"Croatia",
"Cyprus",
"Greece",
"Italy",
"Malta",
"Portugal",
"Slovenia",
"Spain",
"Bulgaria",
"Czech Republic",
"Hungary",
"Poland",
"Romania",
"Russia",
"Slovakia",
"Austria",
"Belgium",
"France",
"Germany",
"Luxembourg",
"Netherlands",
"Switzerland",
"Australia",
"New Zealand",
"South Africa",

```
            "Brazil",
            "Canada",
            "Mexico",
            "USA"
        };
        ArrayList<SelectedCountry> result = new ArrayList<SelectedCountry>();
        int i = 0;
        for (String code : countryCodes) {
            result.add(new SelectedCountry(code, countryList.contains(code) ?
"selected='selected'" : null, countryNames[i]));
            i += 1;
        }
        return result;
    }
}
package controllers;

import db.SQLiteManager;
import model.User;
import spark.ModelAndView;
import spark.Route;
import spark.template.velocity.VelocityTemplateEngine;

import java.util.HashMap;
import java.util.Map;

public class LoginController {

    private static SQLiteManager sqliteManager = SQLiteManager.getInstance();
```

```
    public static Route getLoginPage = (request, response) -> {
        Map<String, Object> model = new HashMap<>();
        return new VelocityTemplateEngine().
                render(new ModelAndView(model, "public/login.html"));
    };
    public static Route logIn = (request, response) -> {
        String login = request.queryParams("login");
        String password = request.queryParams("password");
        User user = new User(login, password);
        if (sqliteManager.loginUser(user)) {
            request.session().attribute("user", new User(user.getEmail(), ""));
            response.redirect("/cabinet");
            return 200;
        } else {
            Map<String, Object> model = new HashMap<>();
            model.put("msg", "Wrong email or password");
            return new VelocityTemplateEngine().
                    render(new ModelAndView(model, "public/login.html"));
        }
    };
}
package controllers;

import db.SQLiteManager;
import model.User;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
```

```
import org.json.simple.parser.ParseException;
import spark.ModelAndView;
import spark.Route;
import spark.template.velocity.VelocityTemplateEngine;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;

public class MainPageController {
   protected static Map<String, JSONObject> gamesMap = new HashMap<String,
JSONObject>();
   private static SQLiteManager sqliteManager = SQLiteManager.getInstance();
   private static LocalDate lastTime = null;
   public static Route getMainPage = (request, response) -> {
      Map<String, Object> model = new HashMap<>();

      return new VelocityTemplateEngine().
            render(new ModelAndView(model, "public/index.html"));
   };

   public static Route getGameDetails = (request, response) -> {
      if (request.session().attribute("user") == null) {
         return "not added";
```

```
        }
        String email = ((User) request.session().attribute("user")).getEmail();
        //String email = "test@gmail.com";
        String euGameId = request.queryParams("euGameId");
        String usGameId = request.queryParams("usGameId");
        return sqliteManager.isGameAdded(email, euGameId, usGameId) ? "added" :
"not added";
    };

    public static Route addGame = (request, response) -> {
        if (request.session().attribute("user") == null) {
            return "no login";
        }
        //String email = "test@gmail.com";
        String email = ((User) request.session().attribute("user")).getEmail();
        String euGameId = request.queryParams("euGameId");
        String usGameId = request.queryParams("usGameId");
        String maxPrice = request.queryParams("maxPrice");
        try {
            Integer.parseInt(maxPrice);
        } catch (Exception e) {
            return "Wrong max price";
        }
        //if (maxPrice.matches("[^0*[1-9]\\d*$]")) return "Wrong max price";
        sqliteManager.addGame(email, euGameId, usGameId, maxPrice);
        return "OK";
    };

    public static Route removeGame = (request, response) -> {
```

```
        if (request.session().attribute("user") == null) {
            return "no auth";
        }
        String email = ((User) request.session().attribute("user")).getEmail();
        //String email = "test@gmail.com";
        String euGameId = request.queryParams("euGameId");
        String usGameId = request.queryParams("usGameId");
        sqliteManager.removeGame(email, euGameId, usGameId);
        return "OK";
    };


    public static Route getGameByName = (request, response) -> {
        if (gamesMap == null) fillGamesList();
        if (lastTime.isBefore(LocalDate.now())) fillGamesList();
        String data = request.queryParams("text").replaceAll("[^a-zA-Z0-9]",
"").toLowerCase();
        JSONArray games = new JSONArray();
        for (String key : gamesMap.keySet()) {
            if (key.startsWith(data)) {
                JSONObject game = gamesMap.get(key);
                games.add(game);
            }
            if (games.size() == 5) break;
        }
        for (String key : gamesMap.keySet()) {
            if (games.size() == 5) break;
            if (key.contains(data) && !key.startsWith(data)) {
                JSONObject game = gamesMap.get(key);
```

```
                games.add(game);
            }
        }
        return games.toJSONString();
    };

    public static void fillGamesList() throws IOException, ParseException {
        lastTime = LocalDate.now();
        System.out.println("Started collecting all games");
        URL url = new URL("http://localhost:3000/games");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer content = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();
        con.disconnect();
//        StringBuilder content = new StringBuilder();
//        try (Stream<String> stream = Files.lines(Paths.get("demo.txt"),
StandardCharsets.UTF_8)) {
//            stream.forEach(s -> content.append(s));
//        }
        JSONObject root = (JSONObject) new JSONParser().parse(content.toString());
        JSONArray euGames = (JSONArray) root.get("eu");
        JSONArray usGames = (JSONArray) root.get("us");
```

```
        euGames.forEach(e -> {
            JSONObject element = (JSONObject) e;
            String code = (String) element.get("euCode");
            if (code.contains(","))
                element.put("euCode", code.substring(0, code.indexOf(",")));
            String title = ((String) element.get("title"))
                    .replaceAll("[^a-zA-Z0-9]", "");
            gamesMap.put(title.toLowerCase(), element);
        });
        usGames.forEach(e -> {
            JSONObject element = (JSONObject) e;
            String title = ((String) element.get("title"))
                    .replaceAll("[^a-zA-Z0-9]", "");
            if (gamesMap.containsKey(title.toLowerCase())) {
                gamesMap.get(title.toLowerCase()).put("usCode", (String)
element.get("usCode"));
            }
        });
        System.out.println("Collecting all games ended");
    }
}
package controllers;

import at.favre.lib.crypto.bcrypt.BCrypt;
import db.SQLiteManager;
import model.User;
import org.apache.commons.lang.RandomStringUtils;
import spark.ModelAndView;
import spark.Route;
```

```
import spark.template.velocity.VelocityTemplateEngine;
import utils.EmailService;

import java.util.HashMap;
import java.util.Map;

public class RegisterController {
    private static SQLiteManager manager = SQLiteManager.getInstance();
    public static Route getRegisterPage = (request, response) -> {
        Map<String, Object> model = new HashMap<>();
        return new VelocityTemplateEngine().
                render(new ModelAndView(model, "public/register.html"));
    };

    public static Route registerAccount = (request, response) -> {
        String login = request.queryParams("login");
        String password = request.queryParams("password");
        Map<String, Object> model = new HashMap<String, Object>();
        if(manager.checkIfEmailAddressExists(login)){
            model.put("msg", "this e-mail is already in use");
            return new VelocityTemplateEngine().
                    render(new ModelAndView(model, "public/register.html"));
        }
        String code = RandomStringUtils.random(7, true, true);
        request.session().attribute("code", code);
        User user = new User(login, BCrypt.withDefaults().hashToString(12,
password.toCharArray()));
        System.out.println("Code "+code);
        try {
```

```
            EmailService.sendCode(login, code);
        } catch (Exception e){
            model.put("msg", "can not send code to this mail for some reason");
            return new VelocityTemplateEngine().
                    render(new ModelAndView(model, "public/register.html"));
        }

        model.put("msg", "Write code from e-mail to confirm it");
        request.session().attribute("tempuser", user);
        return new VelocityTemplateEngine().
                render(new ModelAndView(model, "public/checkCode.html"));
    };

    public static Route checkCode = (request, response) ->{
        String code = request.queryParams("code");
        if(request.session().attribute("code").equals(code)){

System.out.println(((User)request.session().attribute("tempuser")).getEmail());
            manager.createNewUser((User)request.session().attribute("tempuser"));
            request.session().attribute("user", request.session().attribute("tempuser"));
            request.session().removeAttribute("tempuser");
            response.redirect("/main");
        }
        else {
            Map<String, Object> model = new HashMap<String, Object>();
            model.put("msg", "Wrong code");
            return new VelocityTemplateEngine().
                    render(new ModelAndView(model, "public/checkCode.html"));
        }
```

```
        return 200;
    };
}
package db;

import at.favre.lib.crypto.bcrypt.BCrypt;
import model.Game;
import model.User;
import org.apache.ibatis.jdbc.ScriptRunner;
import org.apache.log4j.Logger;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;
import java.sql.*;
import java.util.ArrayList;

public class SQLiteManager {

    private String url = "jdbc:sqlite:prySer.db";
    private Logger LOG = Logger.getLogger(SQLiteManager.class);
    private static SQLiteManager manager = new SQLiteManager();

    public static SQLiteManager getInstance() {
        return manager;
    }
```

```
    private SQLiteManager() {
        File file = new File("prySer.db");
        if (!file.exists()) {
            LOG.info("Creating database");
            try {
                file.createNewFile();
            } catch (IOException e) {
                LOG.error("Failed to create database", e);
            }
            File dbStructure = new File("initSql.sql");
            if (!dbStructure.exists()) {
                LOG.error("Failed to create db structure, no initSql.sql file");
                throw new RuntimeException("Failed to create db structure, no initSql.sql
file");
            }
            try {
                Class.forName("org.sqlite.JDBC");
            } catch (ClassNotFoundException e) {
                LOG.error("Failed to create db structure, driver error");
                throw new RuntimeException("Failed to create db structure, driver error",
e);
            }
            try (Connection connection = DriverManager.getConnection(url)) {
                ScriptRunner sr = new ScriptRunner(connection);
                sr.setEscapeProcessing(false);
                Reader reader = new BufferedReader(new FileReader(dbStructure));
                sr.runScript(reader);
            } catch (FileNotFoundException | SQLException e) {
```

```
            LOG.error("Failed to create db structure, error while running init script");
            file.delete();
            throw new RuntimeException("Failed to create db structure, error while
running init script", e);
        }
    }
}

public boolean checkIfEmailAddressExists(String emailAddress) {
    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
            String sql = "select count(*) from users where email=?;";
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, emailAddress);
            ResultSet resultSet = statement.executeQuery();
            resultSet.next();
            if (resultSet.getInt(1) == 0) return false;
            statement.close();
        }
    } catch (SQLException e) {
        LOG.error("Failed to check email", e);
    }
    return true;
}

public boolean loginUser(User user) {
    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
            String sql = "select password from users where email=?;";
```

```
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, user.getEmail());
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
               String password = resultSet.getString(1);
               BCrypt.Result result =
BCrypt.verifyer().verify(user.getPassword().toCharArray(), password);
               statement.close();
               return result.verified;
            }
            statement.close();
         }
      } catch (SQLException e) {
         LOG.error("Failed to login user", e);
      }
      return false;
   }

   public void createNewUser(User user) {
      try (Connection conn = DriverManager.getConnection(url)) {
         if (conn != null) {
            String sql = "insert into users values(?,?);";
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, user.getEmail());
            statement.setString(2, user.getPassword());
            statement.executeUpdate();
            statement.close();
         }
      } catch (SQLException e) {
```

```
            LOG.error("Failed to create user", e);
        }
    }

    public void addGame(String email, String euNsuid, String usNsuid, String
maxPrice) {
        try (Connection conn = DriverManager.getConnection(url)) {
            if (conn != null) {
                String sql = "insert into games values(?,?,?,?);";
                PreparedStatement statement = conn.prepareStatement(sql);
                statement.setString(1, euNsuid);
                statement.setString(2, usNsuid);
                statement.setString(3, email);
                statement.setString(4, maxPrice);
                statement.executeUpdate();
                statement.close();
            }
        } catch (SQLException e) {
            LOG.error("Failed to add game", e);
        }
    }

    public boolean isGameAdded(String email, String euNsuid, String usNsuid) {
        try (Connection conn = DriverManager.getConnection(url)) {
            if (conn != null) {
                String sql = "select count(*) from games where email=? and euNSUID = ?
and usNSUID = ?;";
                PreparedStatement statement = conn.prepareStatement(sql);
                statement.setString(1, email);
```

```
            statement.setString(2, euNsuid);
            statement.setString(3, usNsuid);
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                int count = resultSet.getInt(1);
                return count > 0;
            }
            statement.close();
        }
    } catch (SQLException e) {
        LOG.error("Failed to login user", e);
    }
    return false;
}

public void removeGame(String email, String euNsuid, String usNsuid) {
    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
            String sql = "delete from games where email=? and euNSUID = ? and
usNSUID = ?;";
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, email);
            statement.setString(2, euNsuid);
            statement.setString(3, usNsuid);
            statement.executeUpdate();
            statement.close();
        }
    } catch (SQLException e) {
        LOG.error("Failed to delete game", e);
```

```
        }
    }

    public void addCountryRestriction(String email, String code) {
        try (Connection conn = DriverManager.getConnection(url)) {
            if (conn != null) {
                String sql = "insert into country_restrictions values(?,?);";
                PreparedStatement statement = conn.prepareStatement(sql);
                statement.setString(1, email);
                statement.setString(2, code);
                statement.executeUpdate();
                statement.close();
            }
        } catch (SQLException e) {
            LOG.error("Failed to add country restriction", e);
        }
    }

    public void removeCountryRestriction(String email, String code) {
        try (Connection conn = DriverManager.getConnection(url)) {
            if (conn != null) {
                String sql = "delete from country_restrictions where email = ? and code =
?;";
                PreparedStatement statement = conn.prepareStatement(sql);
                statement.setString(1, email);
                statement.setString(2, code);
                statement.executeUpdate();
                statement.close();
            }
```

```
    } catch (SQLException e) {
        LOG.error("Failed to remove country restriction", e);
    }
}

public ArrayList<String> getCountryRestrictions(String email) {
    ArrayList<String> countryRestrictions = new ArrayList<String>();
    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
            String sql = "select code from country_restrictions where email=?;";
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, email);
            ResultSet resultSet = statement.executeQuery();
            while (resultSet.next()) {
                countryRestrictions.add(resultSet.getString(1));
            }
            statement.close();
        }
    } catch (SQLException e) {
        LOG.error("Failed to get country restrictions", e);
    }
    return countryRestrictions;
}

public ArrayList<Game> getGames(String email) {
    ArrayList<Game> games = new ArrayList<>();
    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
```

```
            String sql = "select euNSUID, usNSUID,max_price from games where
email=?;";
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, email);
            ResultSet resultSet = statement.executeQuery();
            while (resultSet.next()) {
                Game game = new Game();
                game.setEuCode(resultSet.getString(1));
                game.setUsCode(resultSet.getString(2));
                game.setLimitPrice(resultSet.getString(3));
                games.add(game);
            }
            statement.close();
        }
    } catch (SQLException e) {
        LOG.error("Failed to get games", e);
    }
    return games;
  }

  public void updatePrice(String email, String euCode, String usCode, String
limitPrice) {
    try (Connection conn = DriverManager.getConnection(url)) {
        if (conn != null) {
            String sql = "update games set max_price=? where email=? and
euNSUID=? and usNSUID = ?;";
            PreparedStatement statement = conn.prepareStatement(sql);
            statement.setString(1, limitPrice);
            statement.setString(2, email);
```

```
                statement.setString(3, euCode);
                statement.setString(4, usCode);
                statement.executeUpdate();
                statement.close();
            }
        } catch (SQLException e) {
            LOG.error("Failed to update price", e);
        }
    }
}
package model;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
public class Game {
    private String title;
    private String usCode;
    private String euCode;
    private String imageRef;
    private String limitPrice;

}
package model;
```

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
public class SelectedCountry {
    private String code;
    private String selected;
    private String countryName;
}
package model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class User {
    private String email;
    private String password;
}
package utils;
```

```
import org.simplejavamail.api.email.Email;
import org.simplejavamail.api.mailer.Mailer;
import org.simplejavamail.api.mailer.config.TransportStrategy;
import org.simplejavamail.email.EmailBuilder;
import org.simplejavamail.mailer.MailerBuilder;

public class EmailService {

    public static void sendCode(String emailAddress, String code){
        Email email = EmailBuilder.startingBlank()
                .from("Mailer", "mailer@eshoperer.com")
                .to(emailAddress, emailAddress)
                .withSubject("Code for registration on eshoperer.com")
                .withPlainText("Your code is "+code)
                .buildEmail();
        Mailer mailer = MailerBuilder
                .withSMTPServer("smtpout.secureserver.net", 587,
"mailer@eshoperer.com", "")
                .withTransportStrategy(TransportStrategy.SMTP_TLS)
                .buildMailer();
        mailer.sendMail(email);
    }
}
import controllers.CabinetController;
import controllers.LoginController;
import controllers.MainPageController;
import controllers.RegisterController;
import db.SQLiteManager;
import org.apache.log4j.Logger;
```

```
import org.apache.log4j.PropertyConfigurator;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import spark.ModelAndView;
import spark.Spark;
import spark.template.velocity.VelocityTemplateEngine;
import utils.EmailService;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Stream;

public class Main {
    private static Logger LOG = Logger.getLogger(Main.class);

    public static void main(String[] args) throws Exception {
        Spark.staticFiles.location("/public");
        PropertyConfigurator.configure("log4j.properties");
        MainPageController.fillGamesList();
        if(args.length!=0){
            Spark.port(Integer.parseInt(args[0]));
        }
        SQLiteManager sqliteManager = SQLiteManager.getInstance();
```

```
        Spark.get("/login", (req, res) -> LoginController.getLoginPage.handle(req, res));
        Spark.post("/login", (req, res) -> LoginController.logIn.handle(req, res));
        Spark.get("/register", (req, res) ->
RegisterController.getRegisterPage.handle(req, res));
        Spark.post("/register", (req, res) ->
RegisterController.registerAccount.handle(req, res));
        Spark.post("/checkCode", (req, res) ->
RegisterController.checkCode.handle(req, res));
        Spark.get("/main", (req, res) -> MainPageController.getMainPage.handle(req,
res));
        Spark.get("/gamesByName", (req, res) ->
MainPageController.getGameByName.handle(req, res));
        Spark.post("/addGame", (req, res) -> MainPageController.addGame.handle(req,
res));
        Spark.post("/getGameDetails", (req, res) ->
MainPageController.getGameDetails.handle(req, res));
        Spark.post("/removeGame", (req, res) ->
MainPageController.removeGame.handle(req, res));
        Spark.get("/cabinet", (req, res) -> CabinetController.getCabinetPage.handle(req,
res));
        Spark.post("/addCountryRestriction", (req, res) ->
CabinetController.addCountryRestriction.handle(req, res));
        Spark.post("/removeCountryRestriction", (req, res) ->
CabinetController.removeCountryRestriction.handle(req, res));
        Spark.post("/updatePrice", (req, res) ->
CabinetController.updatePrice.handle(req, res));
    }

}
```

Додаток В. Програмний код додатку для відправки листів

```
package model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@AllArgsConstructor
@ToString
public class UserGame {
    private String email;
    private String euCode;
    private String usCode;
    private String priceLimit;
}
import org.simplejavamail.api.email.Email;
import org.simplejavamail.api.mailer.Mailer;
import org.simplejavamail.api.mailer.config.TransportStrategy;
import org.simplejavamail.email.EmailBuilder;
import org.simplejavamail.mailer.MailerBuilder;

public class EmailService {

    public static void sendDaily(String emailAddress, String text){
        Email email = EmailBuilder.startingBlank()
                .from("Mailer", "mailer@eshoperer.com")
```

```
            .to(emailAddress, emailAddress)
            .withSubject("Your daily report for the deals")
            .withPlainText(text)
            .buildEmail();
        Mailer mailer = MailerBuilder
            .withSMTPServer("smtpout.secureserver.net", 587,
"mailer@eshoperer.com", "")
            .withTransportStrategy(TransportStrategy.SMTP_TLS)
            .buildMailer();
        mailer.sendMail(email);
    }
}
import db.SQLiteManager;
import model.UserGame;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.NumberFormat;
```

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Main {
   static HashMap<String, HashMap<String, String>> usGamePrices = new
HashMap<String, HashMap<String, String>>();
   static Map<String, JSONObject> gamesMap = new HashMap<String,
JSONObject>();

   public static void main(String[] args) throws Exception {
      Class.forName("org.sqlite.JDBC");
      fillGamesList();
      ArrayList<UserGame> userGames =
SQLiteManager.getInstance().collectGamesTable();
      HashMap<String, Set<String>> countryRestrictionsForUser =
SQLiteManager.getInstance().collectCountryRestrictions();
      Set<String> usGames =
userGames.stream().map(UserGame::getUsCode).collect(Collectors.toSet());
      System.out.println("US games " + usGames);
      JSONArray usPrices = makeRequest("getUsGames", usGames);
      HashMap<String, HashMap<String, Double>> usPricesForGame =
parseGames(usPrices);
      Set<String> euGames =
userGames.stream().map(UserGame::getEuCode).collect(Collectors.toSet());
      JSONArray euPrices = makeRequest("getEuGames", euGames);
      HashMap<String, HashMap<String, Double>> euPricesForGame =
parseGames(euPrices);
      HashMap<String, String> mails = new HashMap<String, String>();
```

```
        NumberFormat numberFormat = NumberFormat.getInstance();
        numberFormat.setMaximumFractionDigits(2);
        for (UserGame userGame : userGames) {
            System.out.println("Doing for " + userGame.toString());
            HashMap<String, Double> usGamePrices =
usPricesForGame.get(userGame.getUsCode());
            if (usGamePrices == null) usGamePrices = new HashMap<String, Double>();
            HashMap<String, Double> euGamePrices =
euPricesForGame.get(userGame.getEuCode());
            if (euGamePrices == null) usGamePrices = new HashMap<>();
            Set<String> countryCodes =
countryRestrictionsForUser.get(userGame.getEmail());
            if (countryCodes != null) {
                usGamePrices.keySet().removeAll(countryCodes);
                euGamePrices.keySet().removeAll(countryCodes);
            }
            String usCode = null;
            String euCode = null;
            String message = null;
            Double maxPrice = null;
            if (userGame.getPriceLimit() == null || userGame.getPriceLimit().equals(""))
maxPrice = Double.MAX_VALUE;
            else maxPrice = Double.valueOf(userGame.getPriceLimit());
            for (String code : usGamePrices.keySet()) {
                if (usGamePrices.get(code).compareTo(maxPrice) == -1) {
                    usCode = code;
                    break;
                }
            }
```

```
        for (String code : euGamePrices.keySet()) {
            if (euGamePrices.get(code).compareTo(maxPrice) == -1) {
                euCode = code;
                break;
            }
        }
        if (usCode != null) {
            if (euCode != null) {
                if (Double.compare(usGamePrices.get(usCode),
euGamePrices.get(euCode)) == -1) {
                    message = getGameName(userGame.getEuCode()) + " costs " +
numberFormat.format(usGamePrices.get(usCode)) + "$ in " +
getFullCountryName(usCode) + "\n";
                } else {
                    message = getGameName(userGame.getEuCode()) + " costs " +
numberFormat.format(euGamePrices.get(euCode)) + "$ in " +
getFullCountryName(euCode) + "\n";
                }
            } else {
                message = getGameName(userGame.getUsCode()) + " costs " +
numberFormat.format(usGamePrices.get(usCode)) + "$ in " +
getFullCountryName(usCode) + "\n";
            }
        } else if (euCode != null) {
            message = getGameName(userGame.getEuCode()) + " costs " +
numberFormat.format(euGamePrices.get(euCode)) + "$ in " +
getFullCountryName(euCode) + "\n";
        }
        if (message != null) {
```

```
            if (mails.containsKey(userGame.getEmail())) {
                mails.put(userGame.getEmail(), mails.get(userGame.getEmail()) +
message);
            } else {
                mails.put(userGame.getEmail(), "Your list of games:\n" + message);
            }
        }
    }
    sendMails(mails);
  }

  public static void sendMails(HashMap<String, String> mails) {
    for (String email : mails.keySet()) {
      EmailService.sendDaily(email, mails.get(email));
    }
  }

  public static String getGameName(String code) {
    for (String gameName : gamesMap.keySet()) {
      if (code.equals(gamesMap.get(gameName).get("euCode")) ||
          code.equals(gamesMap.get(gameName).get("usCode"))) {
        return (String) gamesMap.get(gameName).get("title");
      }
    }
    return null;
  }

  public static HashMap<String, HashMap<String, Double>>
parseGames(JSONArray gamePrices) {
```

```
        HashMap<String, HashMap<String, Double>> map = new HashMap<>();
        gamePrices.forEach(e -> {
            JSONObject root = (JSONObject) e;
            JSONArray games = (JSONArray) root.get("games");
            String code = (String) root.get("countryCode");
            games.forEach(gameJson -> {
                JSONObject game = (JSONObject) gameJson;
                String gameCode = ((Long) game.get("code")) + "";
                String stringPrice = game.get("price").toString();
                Double gamePrice = Double.parseDouble(stringPrice);
                if (map.containsKey(gameCode)) {
                    map.get(gameCode).put(code, gamePrice);
                } else {
                    HashMap<String, Double> gamePricesMap = new HashMap<>();
                    gamePricesMap.put(code, gamePrice);
                    map.put(gameCode, gamePricesMap);
                }
            });
        });
        for (String code : map.keySet()) {
            HashMap<String, Double> sorted =
                    map.get(code).entrySet().stream()
                            .sorted(Map.Entry.comparingByValue(Comparator.naturalOrder()))
                            .collect(Collectors.toMap(
                                    Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -> e1,
LinkedHashMap::new));
            map.put(code, sorted);
        }
        return map;
```

```
    }

    public static JSONArray makeRequest(String route, Set<String> games) throws
Exception {
        JSONObject json = new JSONObject();
        JSONArray jsonGames = new JSONArray();
        for (String gameId : games) {
            if (!gameId.equals("undefined"))
                jsonGames.add(gameId);
        }
        json.put("games", jsonGames);
        URL url = new URL("http://localhost:3000/" + route);
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("POST");
        con.setRequestProperty("Content-Type", "application/json; utf-8");
        con.setDoOutput(true);
        byte[] out = json.toJSONString().getBytes(StandardCharsets.UTF_8);
        int length = out.length;

        con.setFixedLengthStreamingMode(length);
        con.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
        con.connect();
        try (OutputStream os = con.getOutputStream()) {
            os.write(out);
        }
        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer content = new StringBuffer();
```

```
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();
        con.disconnect();
        JSONArray res = (JSONArray) new JSONParser().parse(content.toString());
        return res;
    }

    public static <K, V extends Comparable<? super V>> Map<K, V>
sortByValue(Map<K, V> map) {
        List<Map.Entry<K, V>> list = new ArrayList<>(map.entrySet());
        list.sort(Map.Entry.comparingByValue());

        Map<K, V> result = new LinkedHashMap<>();
        for (Map.Entry<K, V> entry : list) {
            result.put(entry.getKey(), entry.getValue());
        }

        return result;
    }

    public static void fillGamesList() throws IOException, ParseException {
        System.out.println("Started collecting all games");
        URL url = new URL("http://localhost:3000/games");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
```

```
        String inputLine;
        StringBuffer content = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();
        con.disconnect();
//        StringBuilder content = new StringBuilder();
//        try (Stream<String> stream = Files.lines(Paths.get("demo.txt"),
StandardCharsets.UTF_8)) {
//            stream.forEach(s -> content.append(s));
//        }
        JSONObject root = (JSONObject) new JSONParser().parse(content.toString());
        JSONArray euGames = (JSONArray) root.get("eu");
        JSONArray usGames = (JSONArray) root.get("us");
        euGames.forEach(e -> {
            JSONObject element = (JSONObject) e;
            String title = ((String) element.get("title"))
                    .replaceAll("[^a-zA-Z0-9]", "");
            gamesMap.put(title.toLowerCase(), element);
        });
        usGames.forEach(e -> {
            JSONObject element = (JSONObject) e;
            String title = ((String) element.get("title"))
                    .replaceAll("[^a-zA-Z0-9]", "");
            if (gamesMap.containsKey(title.toLowerCase())) {
                gamesMap.get(title.toLowerCase()).put("usCode", (String)
element.get("usCode"));
            }
```

```
        });
        System.out.println("Collecting all games ended");
    }

    public static String getFullCountryName(String countryCode) {
        String countryCodes[] = {"DK", "EE", "FI", "IE", "LV", "LT", "NO", "SE",
"GB", "HR", "CY", "GR", "IT", "MT", "PT", "SI", "ES", "BG", "CZ", "HU", "PL",
"RO", "RU", "SK", "AT", "BE", "FR", "DE", "LU", "NL", "CH", "AU", "NZ", "ZA",
"BR", "CA", "MX", "US"};
        String countryNames[] = {
                "Denmark",
                "Estonia",
                "Finland",
                "Ireland",
                "Latvia",
                "Lithuania",
                "Norway",
                "Sweden",
                "United Kingdom",
                "Croatia",
                "Cyprus",
                "Greece",
                "Italy",
                "Malta",
                "Portugal",
                "Slovenia",
                "Spain",
                "Bulgaria",
                "Czech Republic",
```

```
            "Hungary",
            "Poland",
            "Romania",
            "Russia",
            "Slovakia",
            "Austria",
            "Belgium",
            "France",
            "Germany",
            "Luxembourg",
            "Netherlands",
            "Switzerland",
            "Australia",
            "New Zealand",
            "South Africa",
            "Brazil",
            "Canada",
            "Mexico",
            "USA"
        };
        return countryNames[Arrays.asList(countryCodes).indexOf(countryCode)];
    }
}
```