Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ПІДБОРУ МУЗИЧНИХ ІНСТРУМЕНТІВ

Виконала: студентка 4 курсу, групи 3КН-21б
Спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Джулай А.О.
(прізвище та ініціали)
Керівник: асистент каф. КН
Шолота В. В.
(прізвище та ініціали)
«04» червня 2025 р.
Рецензент: к.т.н., доцент каф. АІІТ
Барабан М.В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Джулай Анастасії Олександрівні

1. Тема роботи: Розробка WEB-ресурсу для підбору музичних інструментів
керівник роботи: Шолота Владислава Владиславівна, асистент.
затверджені наказом вищого навчального закладу "20" березня 2025 року №97
2. Термін подання студентом роботи „30" травня 2025р.
3. Вихідні дані до роботи :
Мова програмування: об'єктно-орієнтована;
Кількість груп музичних інструментів: більше 2;
Кількість представлених одиниць для вибору: більше 10;
Обов'язковий елемент програмного забезпечення: рекомендаційний алгоритм та мультимедійність;
Інтерфейс користувача має бути інтуїтивно зрозумілим.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
– проаналізувати існуючі WEB-ресурси для підбору музичних інструментів, визначити доцільність розробки власного WEB-ресурсу;
– спроектувати програмний модуль WEB-ресурсу для підбору музичних інструментів;
– розробити алгоритм рекомендацій музичних інструментів на основі активності користувача;
– програмно реалізувати та протестувати WEB-ресурс для підбору музичних інструментів, розробити інструкцію користувача.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
- схема алгоритму рекомендацій для підбору музичних інструментів;
- схема алгоритму роботи WEB-ресурсу;
- загальна структурна схема компонентів WEB-ресурсу;
- UML-діаграма класів WEB-ресурсу;
- загальний вигляд інтерфейсу користувача;
- приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Шолота В.В., ас. каф. КН		

7. Дата видачі завдання „05" травня 2025року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ; обґрунтування доцільності розробки web-ресурсу для підбору музичних інструментів	05.05.25	07.05.25	
2	Проектування WEB-ресурсу для підбору музичних інструментів	08.05.25	11.05.25	
3	Розробка схеми алгоритму модуля підбору музичних інструментів	12.05.25	15.05.25	
4	Реалізація клієнтської та серверної частин WEB-ресурсу для підбору музичних інструментів	16.05.25	26.05.25	
5	Тестування WEB-ресурсу підбору музичних інструментів	27.05.25	29.05.25	
6	Розробка інструкції користувача	30.05.25	01.06.25	
7	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Джулай А.О.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Шолота В.В**
(підпис) (прізвище та ініціали

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 103 сторінок формату А4, які включають 25 рисунків і 3 таблиці. У списку використаних джерел зазначено 22 найменування.

Метою роботи є розширення функціональних можливостей WEB-ресурсу для підбору музичних інструментів.

У загальній частині роботи на основі аналізу існуючих технічних рішень, методів та технологій розробки програмного забезпечення доведена актуальність розробки WEB-ресурсу для підбору музичних інструментів. Розроблено алгоритм роботи сайту та алгоритм рекомендаційної системи на основі дій користувача, UML-діаграми класів, прецедентів та послідовностей WEB-ресурсу, що є документацією для створеного програмного забезпечення. Програмний продукт розроблено з використанням фреймворків React та Node.js.

Програмний продукт протестовано. За результатами тестування було визначено, що всі заявлені функції реалізовано правильно, програмне забезпечення відповідає поставленим вимогам.

Ключові слова: WEB-ресурс, мультимедійність, музичні інструменти, алгоритм рекомендацій.

ABSTRACT

The bachelor's thesis consists of 103 pages of A4 format, which include 25 figures and 3 tables. There are 22 names in the list of used sources.

The purpose of the work is to expand the functionality of the WEB resource for selecting musical instruments.

In the general part of the work, based on the analysis of existing technical solutions, methods and technologies of software development, the relevance of developing a WEB resource for selecting musical instruments is proven. The algorithm of the site operation and the algorithm of the recommendation system based on user actions, UML class diagrams, precedents and sequences of the WEB resource, which is the documentation for the created software, have been developed. The software product has been developed using the React and Node.js frameworks.

The software product has been tested. According to the results of the testing, it was determined that all the declared functions have been implemented correctly, the software meets the requirements.

Keywords: WEB resource, multimedia, musical instruments, recommendation algorithm.

ЗМІСТ

ВСТУП

Актуальність досліджень.

Музика є невід'ємною частиною сучасної культури, що дозволяє не лише споживати контент та отримувати емоційний досвід від легкого сприйняття широкого різноманіття творчих наробків різних країн та народів та, а й самому бути залученим до процесу створення нових або виконання вже відомих творів. Гра на музичних інструментах може бути як професійною діяльністю, так і хобі, що формує різні вимоги до пошуку та вибору користувачем потрібного йому інструменту.

В сучасному світі більшість запитів з пошуку, продажу, вибору тих чи інших речей (зокрема і музичних інструментів) здійснюється онлайн. Саме тому постає задача створення WEB-ресурсів, які врахують потреби користувача у подачі та представленні інформації та задовольнять його вимоги стосовно функціоналу.

У сфері пошуку музичних інструментів можна виділити декілька актуальних аспектів.

Важливою є доступність платформи як для професіоналів, так і для новачків. WEB-ресурси є доступними для широкого кола користувачів. Сайт, що вміщує в собі також освітній, пізнавальний контент, дозволяє прийняти рішення без додаткового пошуку в інтернеті, може залучати значно більшу кількість користувачів, аніж звичайний інтернет-магазин.

Другим аспектом є наявність мультимедійного контенту. Текстові описи музичних інструментів не надають повної інформації про його якості, зокрема, зовнішній вигляд та звучання. Для здійснення якісного та усвідомленого вибору в даній сфері користувачу важливо мати можливість почути інструмент. Використання мультимедійних інтерактивних вставок у WEB-ресурсі дозволяє зробити вибір більш усвідомленим та наблизити його до досвіду підбору інструменту в фізичному магазині зі збереженням всіх переваг онлайн-покупок.

Іншим аспектом є персоналізовані рекомендації. Користувачі будь-якого рівня, як новачки, так і професіонали, надають перевагу індивідуальному підходу, що спиратиметься на їх особистий досвід взаємодії з WEB-ресурсом і їх уподобання. Високу актуальність має розробка модулів, що здійснюють пропозицію товару на основі попередніх дій користувача.

Отже, WEB-ресурс для підбору музичних інструментів, що інтегрує в собі перелічені вище особливості за допомогою сучасних інформаційних технологій, є актуальним для подальшого дослідження та розробки.

Метою дослідження є розширення функціональних можливостей WEB-ресурсу для підбору музичних інструментів.

Об'єктом дослідження є процес розробки WEB-ресурсу для підбору музичних інструментів.

Предметом дослідження є алгоритми та програмні засоби для реалізації WEB-ресурсу для підбору музичних інструментів.

Задачі дослідження:

1. Провести аналіз предметної області, розглянути аналоги та обґрунтувати доцільність розробки WEB-ресурсу для підбору музичних інструментів.

2. Розробити структуру програмного забезпечення WEB-ресурсу для підбору музичних інструментів.

3. Розробити алгоритм для надання персональних рекомендацій з вибору музичних інструментів на основі активності користувача.

4. Програмно реалізувати WEB-ресурс для підбору музичних інструментів з використанням розробленого алгоритму.

5. Провести тестування роботи WEB-ресурсу для підбору музичних інструментів.

6. Розробити інструкцію користувача.

Апробація роботи. Робота апробувалася на конференції «LIV Всеукраїнська науково-технічна конференція факультету інтелектуальнихінформаційних технологій та автоматизації (2025)» у м. Вінниці [1].

Публікації. На науково-технічній конференції ВНТУ було опубліковано тези доповіді на тему «Перспективи розробки WEB-ресурсу підбору музичних інструментів» [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ WEB-РЕСУРСУ ПІДБОРУ МУЗИЧНИХ ІНСТРУМЕНТІВ

1.1 Огляд існуючих рішень

Вибір правильного музичного інструменту є ключовим для багатьох музикантів, оскільки він впливає на загальне звучання, комфорт та насолоду від гри. Неправильно обраний інструмент може призвести до сповільнення виконання, зниження якості звуку та навіть втрати інтерес до музики [2-4]. Головна проблема полягає у тому, як допомогти користувачам знайти їхній ідеальний інструмент з урахуванням їх особистих потреб та уподобань, і надати загальнодоступний ресурс, здатний задовольнити потребу у швидкому пошуку та заглибленню в тему.

Основна мета створення WEB-ресурсу – забезпечення користувачів точними та релевантними даними про різні музичні інструменти, а також надання корисних порад щодо їх вибору та використання [5]. Цей ресурс повинен стати доступним для широкої аудиторії користувачів та забезпечувати їх вичерпною інформацією для задоволення особистих музичних потреб.

На сьогодні існує велика кількість сайтів та програм, що зосереджуються на різних аспектах музичного виконання. Притому постійно ведеться розробка нових ресурсів, що закривають існуючі або виникаючі в процесі заглиблення у предметну область потреби користувача. Отже, потреба у розробці спеціалізованого WEB-ресурсу, що зосереджується на підборі музичних інструментів, є достатньо об’єктивно обгрунтованою.

Наведемо характерні особливості WEB-ресурсів [6]:

- велика інформаційна база про інструменти;
- безкоштовний доступ;
- наявність купівлі товарів;
- доступний інтерфейс;
- інтерактивність продукту;

На рисунку 1.1 та 1.2 наведено дизайн WEB-ресурсу головної сторінки сайту «Musicca» [7]. Даний ресурс є ознайомчо-навчальною платформою, що надає широкий спектр мультимедійних матеріалів для поглиблення знань стосовно музичних інструментів. Платформа є мультимовною та ставить собі за мету зробити можливим безкоштовне навчання музиці.

На головній сторінці сайту можна побачити заклик до вивчення теорії на платформі та переваги ресурсу. Зверху розташоване верхнє меню сайту, де пропонується реєстрація або вхід в акаунт. Нижче показані особливості та переваги сайту для вивчення музики та використання музичних інструментів.

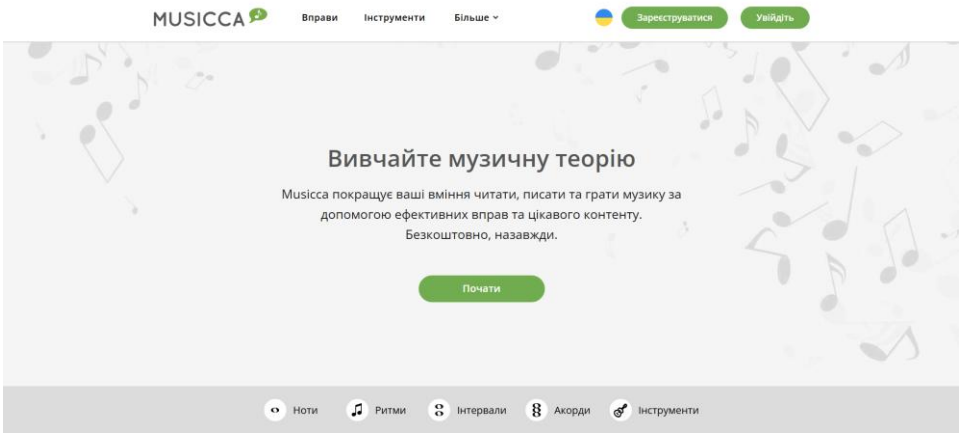

Рисунок 1.1 – Головний екран WEB-ресурсу «Musicca»

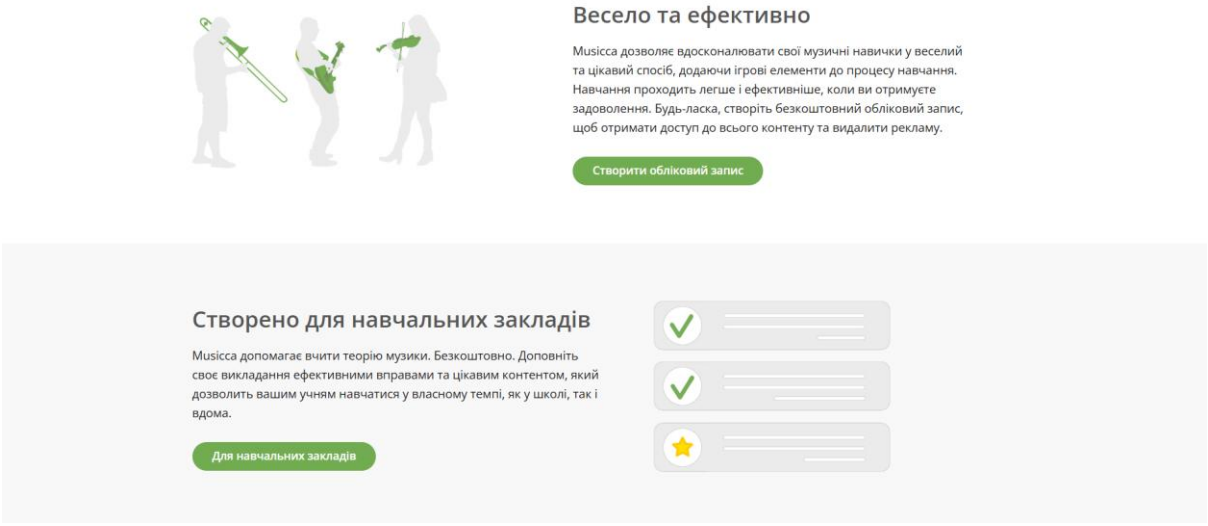

Рисунок 1.2 - Головний екран WEB-ресурсу «Musicca»

До переваг описаного WEB-ресурсу варто віднести зрозумілий інтерфейс та чудове розташування об'єктів на екрані, що заохочує відвідувачів скористатись платформою.

Проте дана платформа концентрується саме на навчальних аспектах, не даючи користувачам здійснити вибір товарів. Попри велику кількість інтерактивних елементів, основні напрямки свого розвитку користувач також обирає сам, так як сайт не має інтелектуалізованих блоків, зокрема, рекомендаційних алгоритмів.

Розглянемо WEB-ресурс «8notes» [8] на рисунку 1.3.

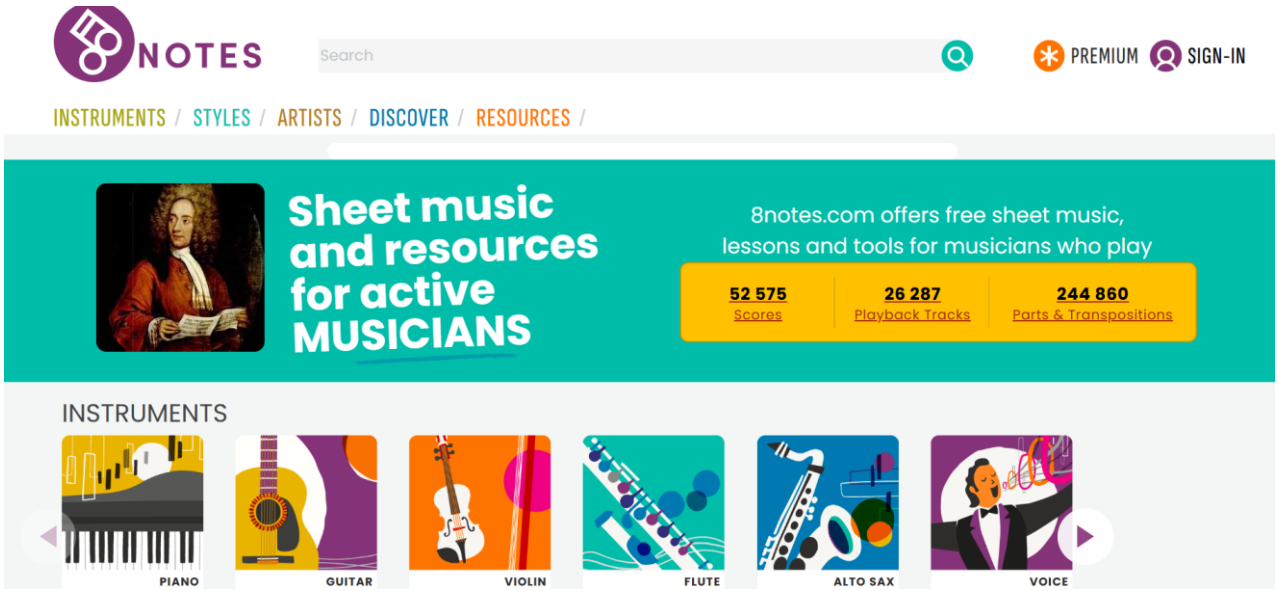

Рисунок 1.3 - Головний екран WEB-ресурсу «8notes»

Головна сторінка сайту пропонує користувачам різноманітні музичні ресурси, такі як ноти, уроки та медіа-ресурси для вивчення музики з різних інструментів. На верхній панелі сайту містяться меню для навігації та користувацькі доступи, такі як зареєструватися чи ввійти в акаунт. Нижче проведені основні ресурси, що надають змогу користувачам обрати інструмент та навчатися гри на ньому. За метою сайт подібний до ресурсу «Musicca», розглянутого вище.

Проаналізувавши даний ресурс «8notes», можемо визначити його переваги, а саме: широкий вибір музичних нот, розміщення високоякісних навчальних ресурсів та їх доступність для всіх користувачів.

Проте, як і в попередньому випадку, сайт переслідує в першу чергу ознайомчу мету і не може слугувати для покупки музичного інструменту і вибору його конкретної моделі.

На рисунку 1.4 та 1.5 представлено дизайн головної сторінки сайту «Metronom.ua» [9].

На головній сторінці відвідувачі можуть ознайомитися з різноманітними музичними інструментами, доступними для придбання. У верхній частині сторінки розташоване меню, яке дозволяє користувачам швидко переходити між категоріями товарів, здійснювати пошук інструментів, а також реєструватися або входити в особистий кабінет.

На відміну від перелічених вище навчальних музикознавчих ресурсів, «Метроном» є інтернет-магазином і надає велику кількість товарів для придбання. Відповідно, у інтерфейсі повинна бути відображена не лише інформація навчального характеру, а й інформація для покупки. Це впливає на структуру побудови сайту.

На рисунку 1.5 видно, що вже на головній сторінці сайту представлений широкий вибір музичних інструментів. Користувач може взаємодіяти з товарами, відкривати сторінку кожного товару окремо та переглядати її.

Основною перевагою сайту є інтуїтивно зрозумілий інтерфейс, що полегшує навігацію навіть для нових користувачів. Додатково сайт пропонує зручну систему фільтрації та сортування товарів, що допомагає швидко знайти потрібний інструмент.

Окрім цього, «Metronom.ua» надає детальні описи кожного інструменту, що включають характеристики, фотографії та відгуки інших користувачів, що сприяє прийняттю обґрунтованого рішення про покупку.

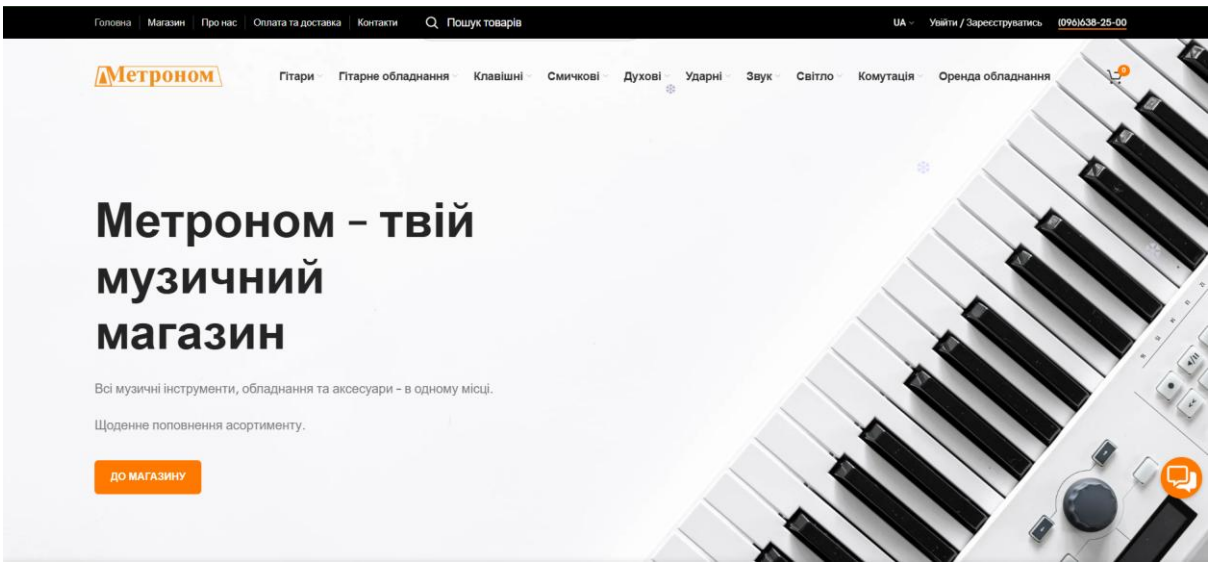

Рисунок 1.4 - Головний екран WEB-ресурсу «Metronom.ua»

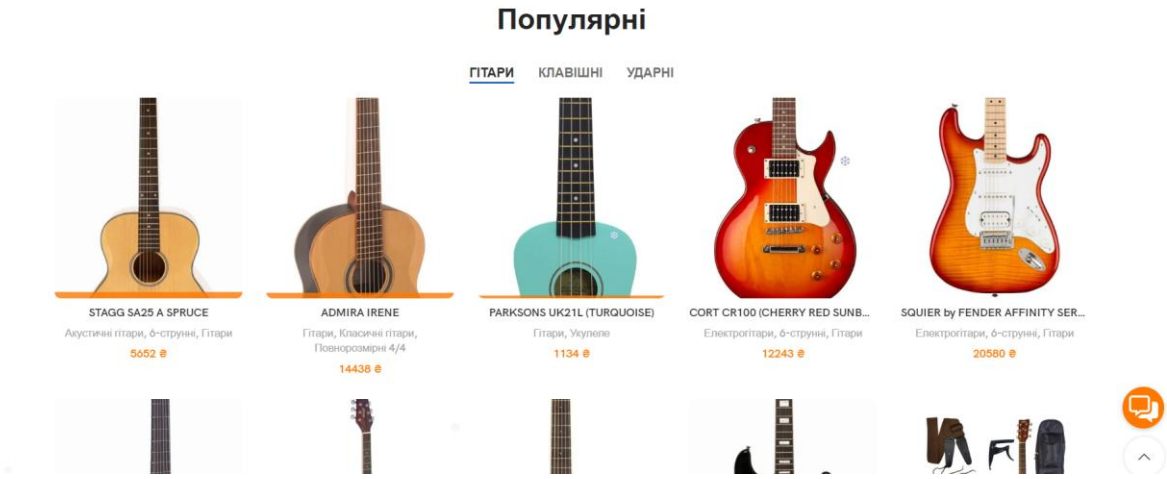

Рисунок 1.5 - Головний екран WEB-ресурсу «Metronom.ua»

Серед недоліків сайту можна відзначити відсутність навчальних інтерактивних матеріалів, що дозволяють новачкам краще орієнтуватись серед широкого вибору інструментів. З іншого боку, при розробці даного сайту, імовірно, не ставилась навчальна мета.

На рисунку 1.6 наведено дизайн головної сторінки сайту «Muztorg.ua» [10]. На рисунку 1.7 продемонстровано приклад відображення товарів на ресурсі. Головна сторінка пропонує користувачам широкий вибір музичних інструментів та аксесуарів, зручно організованих у категорії. Навігація по сайту є частиною інтерфейсу, що забезпечує легкий доступ до основних розділів та категорій

товарів, завдяки чому користувачі можуть швидко знайти потрібну інформацію. Верхнє меню містить опції для входу в особистий кабінет, перегляду кошика та пошуку по сайту.

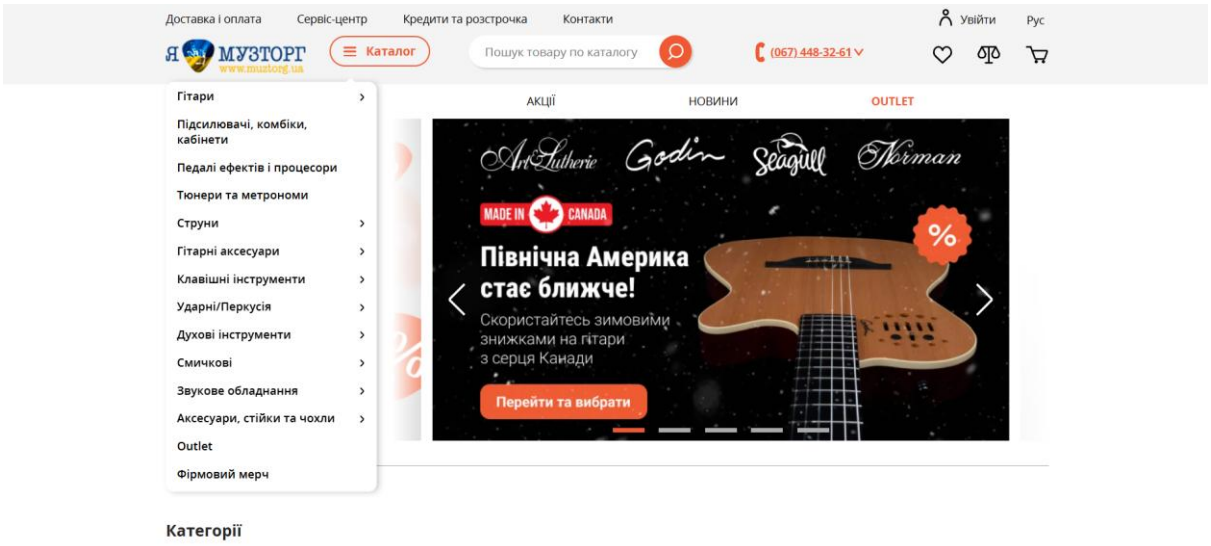

Рисунок 1.6 - Головний екран WEB-ресурсу «Muztorg.ua»

Рисунок 1.7 – Сторінка вибору товару WEB-ресурсу «Muztorg.ua»

Крім того, «Muztorg.ua» пропонує різні способи сортування і фільтрації товарів, що значно полегшує пошук необхідного інструменту. Детальні описи товарів, відгуки покупців та рекомендації сприяють інформованому вибору.

Як і у випадку сайту «Метроном», акценту на навчальних матеріалах не робиться, тому для користувача-новачка важкою задачею може стати здійснення вибору в умовах невизначеності, без підказок та мультимедійних матеріалів.

Окрім того, при розробці даного сайту використовувались застарілі на даний момент технології, його дизайн не відповідає сучасним вимогам. У планах адміністрації сайту є покращення візуального інтерфейсу користувача.

В таблиці 1.1 наведено порівняльну характеристику розглянутих додатків: «Musicca», «8notes», «Metronom.ua», «Muztorg.ua».

З таблиці 1.1 видно, що більшість описаних додатків мають такі переваги:

- доступність;
- ідентифікація;
- пошукова система.

Поміж цього, не всі додатки мають мультимедійність, а саме лише «Musicca» та «8notes».

Також не всі додатки збирають у собі всі типи інструментів. На сайтах «Musicca» та «Metronom.ua» відсутній розподіл за типами, що робить навігацію менш зручною та інтуїтивно-зрозумілою.

З цього можна зробити висновок, що створення WEB-ресурсу для підбору музичних інструментів, що врахує зауваження до аналогів, є доцільним та має практичне значення.

Такий WEB-ресурс буде включати в себе мультимедійність, тобто використання різних типів контенту, такі як: текст, аудіо, відео.

Також система буде мати реєстрацію та авторизацію, де для входу буде використовуватись логін та пароль. При користуванні ресурсом користувач буде мати доступ до бібліотеки інструментів, де зможе обрати той, що найбільше зручний для нього.

Таблиця 1.1 – Порівняльна характеристика популярних додатків для підбору музичних інструментів

Характеристика, що розглядається	«Musicca»	«8notes»	«Metronom.ua»	«Muztorg.ua».
Мультимедійність	Так	Так	Ні	Ні
Реєстрація та авторизація	Так	Так	Так	Так
Пошукова система	Ні	Так	Так	Так
Типи інструментів	Жодного	Кілька	Жодного	Всі типи
Використання сучасних веб-технологій в дизайні	Ні	Так	Так	Ні

Відмінною рисою WEB-ресурсу буде стильний та унікальний дизайн, який буде налаштований не лише гарно, але й лаконічно та практично у використанні сайту користувачем. Комбінація навчальної та економічної складової (продажу товарів) стане суттєвим розширенням функціональних можливостей в порівнянні з аналогами.

1.2 Формулювання вимог та постановка задачі

Значення правильного підбору музичних інструментів є важливим для розвитку музичної освіти та творчих можливостей особистості [2, 4]. Незважаючи на широкий доступ до музичних ресурсів, багато людей, особливо новачків у музиці, стикаються з труднощами при виборі інструментів, які б підходили для їхнього рівня навичок або музичних уподобань. Це може призводити до нерозуміння власних потреб у музиці, що ускладнює подальший

розвиток музичних здібностей та відвертає потенційних майбутніх музикантів від цього захоплення.

Інтерактивний сучасний WEB-ресурс для підбору музичних інструментіввбуде корисним для тих, хто шукає інструменти, що найкраще відповідають їхнім вимогам, незалежно від рівня підготовки або жанру музики, яку вони бажають виконувати. Це дозволить користувачам вибирати інструменти з урахуванням особливостей, таких як тип музики, зручність у використанні та вимоги до навичок гри.

Крім того, ресурс надаватиме детальну інформацію про кожен інструмент, можливості його застосування та навчальні матеріали, що допоможуть користувачам швидше адаптуватися до нового інструмента. Таким чином, закривається прогалина у знаннях користувача, йому не потрібно шукати додаткові ресурси для того, щоб дізнатись потрібну інформацію. Це водночас і сприяє покупкам, і покращує користувацький досвід використання WEB-ресурсу.

Розробка цього WEB-ресурсу вимагатиме врахування не лише функціональних вимог до системи, таких як наявність бібліотеки інструментів, зручного інтерфейсу для вибору та порівняння, а й нефункціональних вимог, наприклад, надійної ідентифікації користувача для персоналізованих рекомендацій.

Дизайн ресурсу буде створено таким чином, щоб він був зручним та практичним для користувачів усіх рівнів, а також мав стильний вигляд, що підкреслює інноваційність та функціональність платформи.

Інструменти реалізації будуть обрані для максимальної доступності ресурсу незалежно від пристрою, що використовує користувач (з певними базовими вимогами до його сучасності). Серверна та клієнтська частина WEB-ресурсу повинні бути виконані таким чином, щоб не лише вміщати увесь необхідний фунціонал, а й забезпечувати високу швидкість роботи програмного забезпечення.

WEB-ресурс «Підбір музичних інструментів» надаватиме такі можливості користувачам:

- доступ до бібліотеки інструментів з детальними характеристиками;
- персоналізовані рекомендації по інструментах;
- інтуїтивно зрозумілий інтерфейс для зручності використання;
- мультимедійні вставки, що допоможуть при виборі інструмента.

Розробка буде здійснюватися з використанням сучасних веб-технологій, орієнтуючись на забезпечення максимальної зручності та функціональності для користувачів на всіх етапах взаємодії з ресурсом.

1.3 Висновок

У цьому розділі було наведено аналіз існуючих додатків для підбору музичних інструментів, визначено актуальні в даній сфері функції WEB-ресурсів, зокрема, мультимедійність, використання сучасних технологій, тощо. Було розглянуто переваги та недоліки аналогів, визначено, що мультимедійність мають лише такі додатки, як «Musicca» та «8notes», а використання сучасних веб-технологій в дизайні забезпечується лише у додатках «8notes» та «Metronom.ua».

На основі проведеного аналізу вирішено розробити WEB-ресурс для підбору музичних інструментів з усуненням наведених недоліків та врахуванням визначених вище потреб користувача.

В розділі було поставлено задачі для подальшого дослідження та визначено базові вимоги до WEB-ресурсу, що розроблятиметься в межах бакалаврської роботи. Даний ресурс сприятиме вибору музичних інструментів відповідно до потреб та вподобань користувачів, надаватиме персоналізовані рекомендації та доступ до навчальних матеріалів. Крім того, результат розробки забезпечуватиме інтерактивний досвід через мультимедійні вставки та інтуїтивно зрозумілий інтерфейс. Дизайн ресурсу буде стильним, сучасним та орієнтованим на зручність користувачів, що підкреслюватиме інноваційність та функціональність платформи.

2 ПРОЕКТУВАННЯ WEB-РЕСУРСУ ДЛЯ ПІДБОРУ МУЗИЧНИХ ІНСТРУМЕНТІВ

2.1 Розробка структури програмного забезпечення ресурсу підбору музичних інструментів

Програмний модуль ресурсу підбору музичних інструментів є WEB-ресурсом, що складається з двох частин – клієнтської й серверної. Для спрощення розробки кожну частину поділено на окремі модулі. У серверній частині ці модулі реалізовано у форматі мікросервісів, тому архітектура проєкту є мікросервісною.

Мікросервісна архітектура передбачає поділ великого застосунку на невеликі незалежні мікросервіси, кожна з яких виконує чітко визначену функцію. Мікросервісна архітектура дозволяє розробникам більш ефективно розвивати та масштабувати складні додатки, так як мікросервіси можуть бути розгорнуті та масштабовані незалежно один від одного. Крім того, цей підхід дозволяє розробникам використовувати різні технології та мови програмування для кожного мікросервісу, що дає можливість використовувати найбільш підходящі інструменти для кожної конкретної задачі.

Кожен мікросервіс запаковується в контейнер. Контейнери (Docker) дозволяють запустити програму з усіма бібліотеками та залежностями як один пакет, забезпечують ізольоване середовище та підтримують горизонтальне масштабування – запуск кількох екземплярів для балансування навантаження [11].

Для програмного модуля підбору музичних інструментів створено такі модулі:

- Модуль авторизації, реєстрації та кабінету користувача;
- Модуль каталогу інструментів – пошуку, фільтрації й сортування за категоріями;

- Модуль сторінки інструмента;
- Модуль кошика покупок;
- Модуль персональних рекомендацій.

Кожен модуль складається зі сторінок. Наприклад, модуль авторизації містить сторінки реєстрації нового користувача та входу; модуль каталогу – сторінку переліку інструментів; модуль сторінки інструмента – детальний перегляд із медіа-прев'ю; модуль кошика – перегляд кошика та ініціацію оплати. Зобразимо схему роботи програмного модуля на діаграмі прецедентів для підбору музичних інструментів на рисунку 2.1.

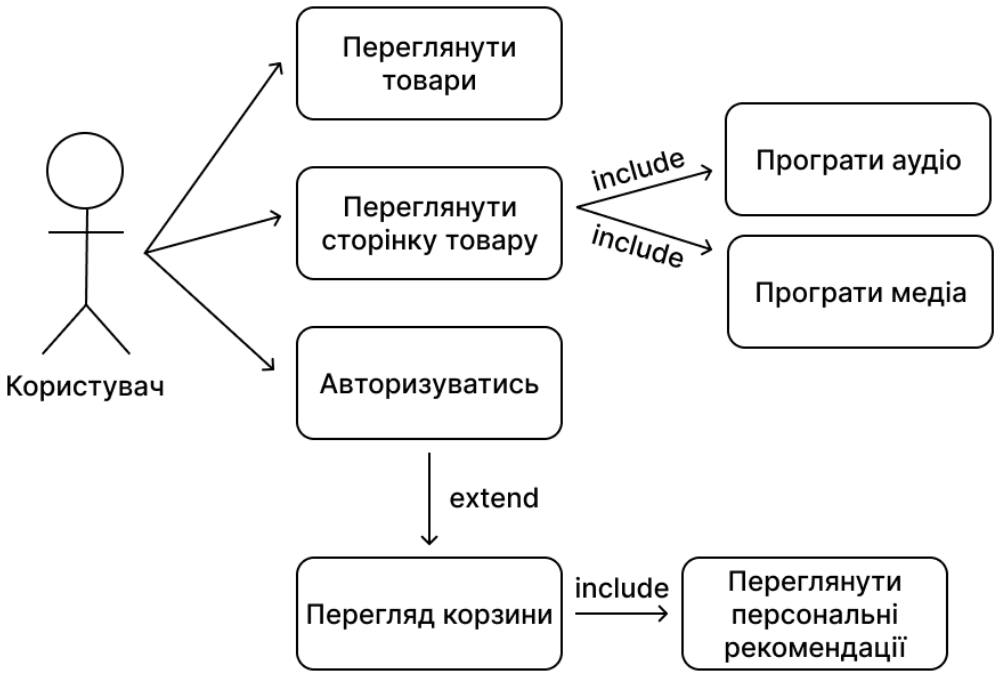

Рисунок 2.1 – Діаграма прецедентів для підбору музичних інструментів

На сторінці реєстрації користувач створює обліковий запис, указуючи електронну пошту й пароль. Дані передаються на сервер, де їх обробляє мікросервіс реєстрації. Після успішної реєстрації користувач авторизується: введені облікові дані надсилаються на мікросервіс авторизації, і в разі успіху система перенаправляє його на каталог інструментів.

Користувач може переглядати каталог, відкривати детальні сторінки інструментів, слухати аудіозаписи та дивитися відео-огляди. Додавши інструмент до кошика, він переходить на сторінку кошика, де бачить обрані позиції та підраховану вартість. Запити, що стосуються кошика, опрацьовує відповідний мікросервіс; якщо користувач авторизований, стан кошика зберігається у профілі.

Сервер зберігає інформацію у документно-орієнтованій базі даних Mongo DB. Активна колекція users містить облікові записи, електронні адреси та хешовані паролі користувачів; саме до неї прив’язані операції авторизації й реєстрації. Решта модулів отримують і віддають дані через API без прямого звернення до сховища, тому навантаження на базу мінімізоване.

Усі запити клієнтської частини надходять через єдиний API-шлюз і обробляються балансувальником навантаження, який рівномірно розподіляє трафік між екземплярами мікросервісів. Запити, що стосуються авторизації та реєстрації користувачів, маршрутизуються на відповідний мікросервіс і лише тоді взаємодіють із базою даних.

Загальна структурна схема компонентів програмного модуля підбору музичних інструментів зображена на рисунку 2.2.

На структурній схемі виділено дві основні частини програмного забезпечення: клієнтську та серверну. З’єднання між частинами відбувається на основі API. Користувач має безпосередній контакт з блоками клієнтської частини, такими як:

– модуль головної сторінки з інструментами;

– модуль авторизації та реєстрації;

– модуль корзини користувача;

– модуль сторінки інструменту;

– модуль системи рекомендацій.

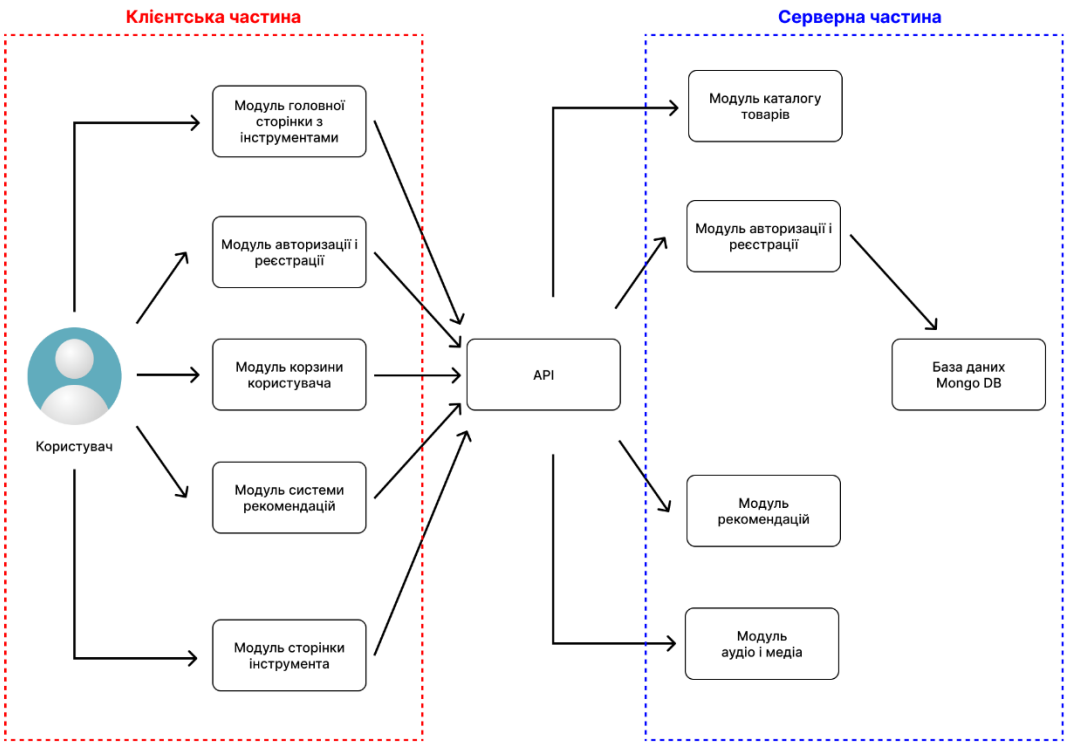

Рисунок 2.2 – Загальна структурна схема компонентів програмного модуля підбору музичних інструментів

Запити до серверної частини, ініційовані діями користувача, через API надходять до серверної частини та розподіляються по відповідних модулях:

– модуль каталогу товарів;

– модуль авторизації і реєстрації;

– модуль рекомендацій;

– модуль аудіо і медіа.

Серед всіх перелічених модулів лише модуль авторизації та реєстрації взаємодії з базою даних, що також інтегрована у серверну частину.

2.2 Розробка UML-діаграм

Перш ніж розпочати розробку веб-ресурсу, варто врахувати всі можливі підходи до побудови та реалізації UML-діаграм. Розглянемо їхнє визначення та класифікацію.

UML (Unified Modeling Language) — уніфікована мова моделювання, що використовується роз робниками програмного забезпечення для візуалізації процесів та роботи систем.

Це не мова програмування, скоріше набір правил та стандартів для створення діаграм. Вони дозволяють розробникам програмного забезпечення та інженерам «говорити однією мовою», не заглиблюючись у фактичний код свого продукту. Складання діаграм за допомогою UML — це чудовий спосіб допомогти іншим швидко зрозуміти складну ідею чи структуру. UML-діаграми поділяють на 14 типів. Покажемо типи UML-діаграм на рисунку 2.3.

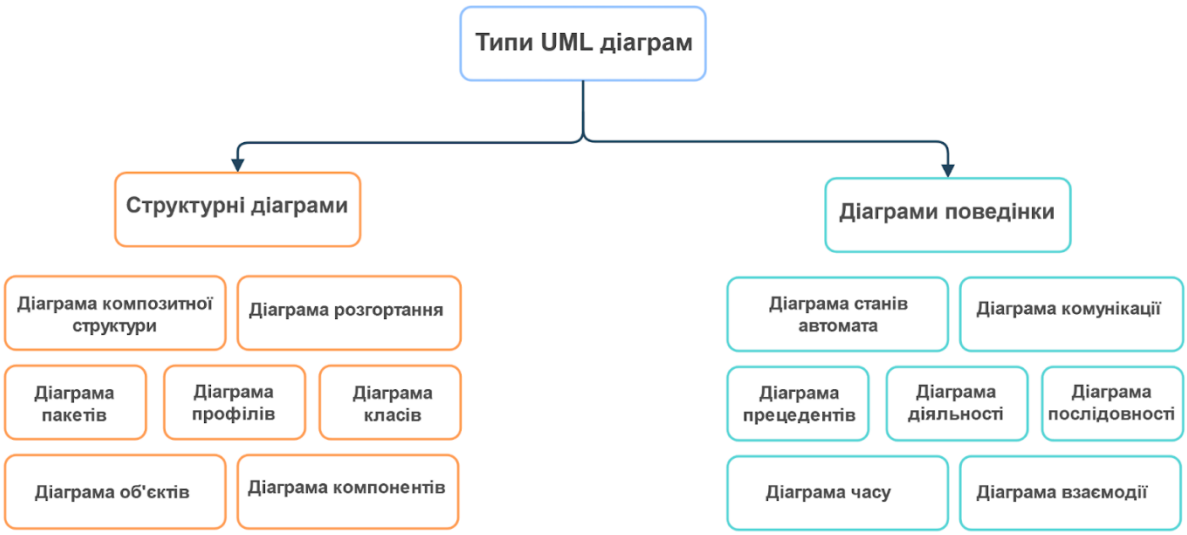

Рисунок 2.3 – Типи UML-діаграм

Структурні – містять діаграми класів (відображає структуру класів у системі та їхні взаємозв’язки), компонентів (дає змогу уявити структуру високорівневих компонентів системи та їхні залежності) і об’єктів (фокусується на конкретних об’єктах у системі та їхніх взаємодіях).

Поведінкові – включають діаграми випадків використання (описує взаємодію між системою та її користувачем), активностей (моделює потоки управління або процесів у системі) і послідовності (показує взаємодію між різними об’єктами в системі в певній послідовності подій). Засновуючись на цю інформацію, зобразимо діаграму класів та послідовностей.

UML-діаграму класів побудовано, спираючись на засадничі принципи об’єктно-орієнтованого програмування (ООП).

- Інкапсуляція — це принцип ООП, який полягає в приховуванні деталей реалізації об'єктів від «зовнішнього світу». Цей принцип стверджує, що вся важлива інформація міститься всередині об’єкта, а назовні доступна тільки вибрана інформація.
- Наслідування — це принцип, який дозволяє створювати нові класи на основі наявних (батьківських класів), з можливістю перевизначення або доповнення їх властивостей та методів. З його допомогою розробник може використати логіку батьківського класу у дочірньому. Це спрощує розробку та зменшує кількість дублювального коду.
- Поліморфізм — це принцип, який в певному сенсі доповнює наслідування. Тобто об'єкти різних класів можуть виконувати дії з однаковою назвою, використовуючи різний код.
- Абстракція допомагає зосередитися на головних аспектах системи та ігнорувати менш важливі деталі, які не впливають на головні функції. Абстракцію можна розглядати як розширення до інкапсуляції. Завдяки принципу абстракції кожен об’єкт відкриває лише певний механізм для використання.

UML-діаграми класів (Class diagram) – тип структурованої діаграми складного об’єкта статичної структури, яка представляє класи, атрибути, методи та взаємозв’язки [12, 13]. Діаграма класів може слугувати основною опорою для розробки програмного коду. Вона враховує структуру майбутнього програмного забезпечення і чітко окреслює вимоги до створених класів, визначає взаємозв’язки, які важливо врахувати при подальшій розробці проекту. Цей тип діаграм може бути корисним як перед створенням програмного забезпечення, слугуючи опорною схемою для програміста, так і після, адже забезпечує можливість вносити зміни, враховуючи існуючі зв’язки і не руйнуючи їх нововведеннями.

У серверній та клієнтській частині програмного модуля для підбору музичних інструментів є такі ключові класи: User, Category, Instrument, CartItem, Order, OrderItem.

Клас User містить поля id типу string, username типу string, email типу string, password типу string, createdAt типу Date та updatedAt типу Date для збереження облікових даних користувача й автоматичного відстеження часу створення та зміни запису в базі даних.

Клас Category має поля id типу string та name типу string, відповідаючи за групування інструментів за категоріями і використовується для фільтрації й навігації на фронтенді. Попри відсутність великої кількості полів, він відіграє важливу роль у побудові структури ресурсу та пов’язаний з класом Instrument. Варто зазначити, що цей зв’язок можна охарактеризувати типом «багато до одного»: інструмент може належати тільки до однієї категорії, проте в межах однієї категорії можна описати багато інструментів.

Клас Instrument визначає поля id типу string, title типу string, description типу string, imageUrl типу string та price типу number і представляє окремий товар з усіма його атрибутами; кожен екземпляр цього класу належить до певної категорії Category. В подальшому цей взаємозв’язок дозволяє групувати об’єкти класу «Instrument» та покращити видачу інформації користувачеві.

Клас CartItem має поля id типу string та quantity типу number і містить асоціації до об’єктів User та Instrument, вказуючи, який користувач додав який інструмент у кошик та в якій кількості. У візуальному відображенні цей клас представляється у вигляді вмісту корзини з товарами та повинен динамічно оновлюватися по мірі активності авторизованого користувача на сторінці.

Клас Order містить поля id типу string, totalPrice типу number та createdAt типу Date і відповідає за збереження загальної вартості й часу створення кожного замовлення, що належить певному користувачеві.

Клас OrderItem складається з полів id типу string, quantity типу number та price типу number і має зв’язки з класами Order та Instrument, фіксуючи деталі

кожної позиції в замовленні: який інструмент, скільки одиниць і за якою ціною було придбано.

На рисунку 2.4 зображена UML-діаграма класів клієнтської та серверної частини модуля підбору музичних інструментів.

В таблиці встановлені певні зв’язки між класами. Користувач може мати багато позицій у кошику і багато замовлень; кожна позиція у кошику і кожен елемент замовлення належить конкретному інструменту; інструмент, у свою чергу, відноситься до певної категорії. Ці асоціації зображено стрілками з вказанням кардинальностей, що дає змогу одразу побачити напрями й кратність взаємодій (наприклад, один користувач — багато замовлень).

Зображені на рисунку 2.4 класи в подальшому отримають своє втілення у відповідному функціоналі клієнтської та серверної системи.

Варто зазначити, що створена структура є легко адаптивною: при подальшому розширенні функціоналу WEB-ресурсу. Це важливий аспект, що дозволяє доповнювати та осучаснювати сайт у відповідності до поточного високого темпу розвитку WEB-технологій, інтегрувати нові компоненти та гнучко реагувати на розвиток користувача.

Реалізація WEB-ресурсу на основі діаграми класів також може потребувати створення додаткових (службових) класів, існування яких важко передбачити на етапі проектування. В такому разі, документація доповнюється по мірі розробки. Однак важливо, щоб створення додаткових структур не вносило корективи в загальну логіку зв’язку між класами та не розривало їх.

UML-послідовність (Sequence Diagram) – тип поведінкової діаграми, яка описує поведінкові аспекти складного об’єкта та враховує взаємодію об’єктів в часі.

Цей тип діаграм дозволяє відобразити тимчасові особливості передачі і прийому повідомлень об'єктами, використовуючи уточнення діаграм прецедентів.

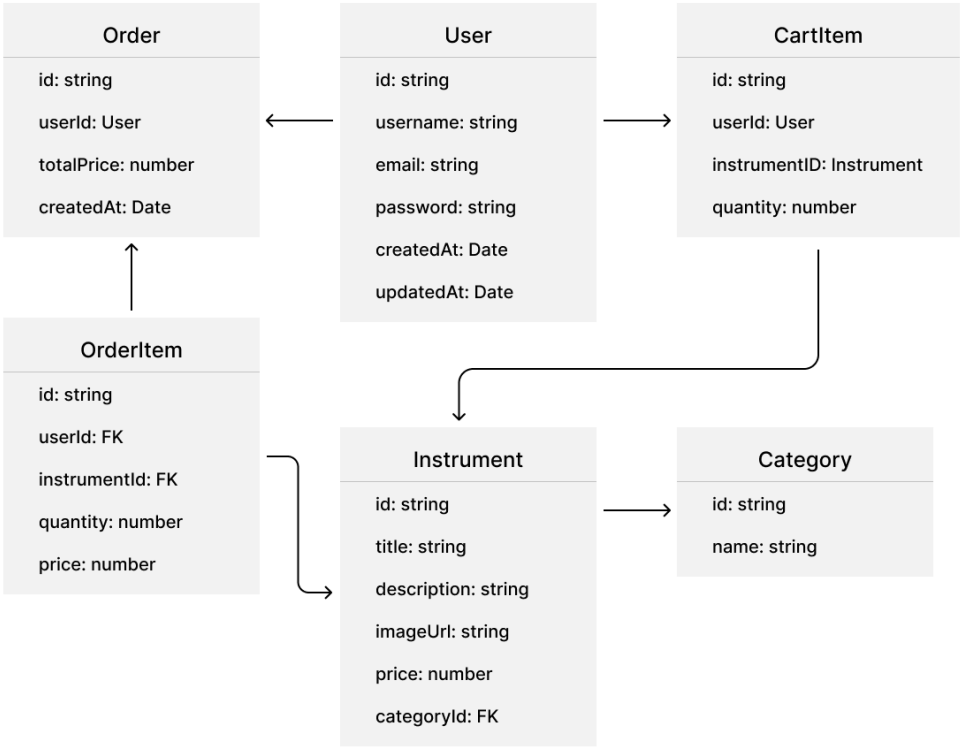

Рисунок 2.4 - UML-діаграма класів клієнтської та серверної частини модуля підбору музичних інструментів

Така діаграма дає правильне розуміння взаємодії між шарами та об’єктами для забезпечення правильного функціонування системи. На відміну від діаграми прецедентів (варіантів використання), діаграма послідовності дає можливість чітко визначити, який з компонентів системи відповідає за який запит користувача. Таким чином, вона грає важливу роль при відлагодженні системи та при розширенні технічного завдання від користувача/замовника з подальшим його пропрацюванням.

Проаналізуємо взаємодію між різними шарами системи, використовуючи UML-діаграму послідовностей, а саме: Користувач, Інтерфейс, AuthController, ProductController, CartController, Recommendation Controller. UML-діаграму послідовностей для підбору музичних інструментів зображено на рисунку 2.5.

Основним блоком, з яким взаємодіє користувач, є інтерфейс. Серверна частина (бекенд) прихована від користувача, і його відповідні дії на клієнтській

частині (інтерфейсі, фронтенді) слугують тригерами для реакцій відповідних компонентів.

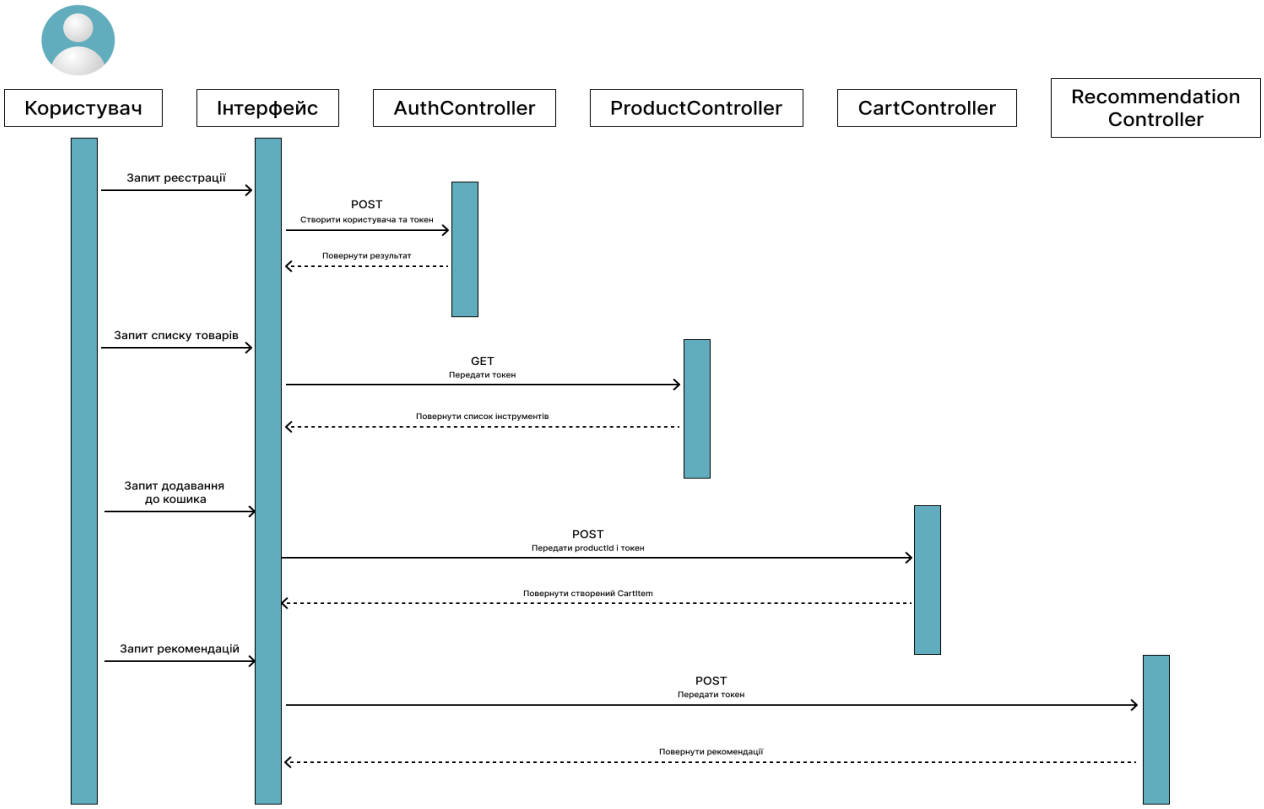

Рисунок 2.5 - UML-діаграма послідовностей для підбору музичних інструментів

Блок AuthController відповідає за процес автентифікації користувача. Він активується на основі POST-запиту з користувацькими даними (логін та пароль). Відповіддю на запит слугує сповіщення про успішність або неуспішність авторизації. Якщо авторизація пройшла успішно, користувач має можливість користуватися перевагами зареєстрованого, якщо ні, йому надходить повідомлення про неправильність логіна чи паролю, і надається можливість здійснити реєстрацію на сайті. Важливим є використання саме POST-запитів для даного типу взаємодії, так як за сучасними протоколами безпеки вимагається шифрування конфіденційної інформації.

Тоді як взаємодія користувача зі списком товарів активує ProductController і здійснюється на основі запиту GET, тобто такого, що передає інформацію

безпосередньо у посиланні (URL/URI). Дана інформація не носить персональний характер і відповідає за навігацію на сайті, тому доцільним є використання простішого запиту зі швидшим відгуком. В якості відповіді на запит користувач отримує сторінку з набором інструментів обраного ним типу.

І додавання до кошика, і запит стосовно рекомендацій теж здійснюються за допомогою POST-методу, так як вимагають взаємодії з особистими даними користувача: дані дії можливі лише для авторизованого користувача і використовують інформацію про поточну сесію та його поточну активність.

Запит на додавання до кошика запускає на виконання CartController, що відповідає за зміни у кошику користувача та їх відображення на інтерфейсі сайту.

Запит рекомендацій активує RecommendationController, що запускає алгоритм персональних рекомендацій на сайті і відправляє на інтерфейс відповідний перелік товарів.

Усі відповіді на користувацькі запити активують відповідні функції інтерфейсу. Інтерфейсна частина за допомогою розробленого функціоналу обирає варіант відображення інформації у зручній для користувача формі. Після цього користувач може переходити до наступної дії у потрібному йому порядку. Дії можуть виконуватися ту кількість разів, яку забажає користувач. У випадку, коли певні дії неможливі до виконання попередніх (зокрема, взаємодія з корзиною до реєстрації або авторизації), користувач отримуватиме відповідне повідомлення та переноситиметься на сторінку з потрібним йому функціоналом.

2.3 Розробка схеми алгоритму модуля для підбору музичних інструментів

Алгоритм – це впорядкований набір дій для вирішення певної задачі. Використання алгоритму у веб-ресурсах інтернет-магазинів сприяє:

- чіткому та структурованому поділу функціональності, що спрощує розробку та підтримку коду;
- зменшенню часу на внесення змін і відлагодження;

- підвищенню зрозумілості логіки взаємодії з користувачем.

Сфера електронної комерції максимально зацікавлена у залученні користувача до активності на сайті та у зростанні кількості його покупок. З іншого боку, при зростанні кількості товарів, збільшенні асортименту, ускладненні функціонування WEB-ресурсу постає проблема зменшення зацікавленості користувача та ускладненню пошуку. Таким чином, виникає задача пошуку способу демонстрації користувачеві бажаних товарів за мінімальний період часу.

В сучасних системах з великим об’ємом контенту задача пошуку та динамічної взаємодії з ним часто вирішується за допомогою рекомендаційних систем. Вони призначені для того, щоб продемонструвати користувачу саме той контент або товар, у якому він максимально зацікавлений, простимулювати покупку. Також наявність такої системи слугує додатковою перевагою WEB-ресурсу, як уже було означено в першому розділі бакалаврської роботи.

Проте при впровадженні рекомендаційного алгоритму постає проблема, по-перше, його інтеграції в загальну роботу ресурсу, щоб наявність пропозицій на сайті не була занадто нав’язливою, але водночас і не ігнорувалася функціоналом взагалі, по-друге, здійснення власне рекомендації, що повинно базуватися на певних математичних, статистичних та логічних принципах, адаптованих до предметної області та потреб користувача.

Існує ряд автоматизованих інструментів, які доречно задіювати при створенні WEB-ресурсів для електронної комерції, проте варто враховувати особливості їх реалізації та можливість інтеграції у обрану систему, зокрема, сумісність даних технологій [14 - 16].

Враховуючи особливості спроектованого WEB-ресурсу, в даному випадку основними проблемами, які потрібно вирішити при побудові рекомендаційного алгоритму, є такі:

– холодний старт, тобто відсутність даних про користувача або дії інших користувачів у системі;

– відсутність даних про конкретного користувача.

Такі характерні проблеми як масштабованість та накручування рейтингів, типові для великих WEB-систем, не є актуальними на даному етапі розвитку ресурсу.

Часто при побудові рекомендаційних систем використовуються методи колаборативної фільтрації, що виділяють item-based та user-based підхід. Підхід, оснований на об’єктах, використовує набори ознак товарів, якими зацікавився поточний користувач, та вишукує подібні у загальній системі. Підхід, оснований на користувачах, в якості основи бере подібні дії користувачі, приймаючи за даність, що в середньому поведінка більшості користувачів системи є типовою, а отже, якщо користувач А і користувач В мають багато точок перетину, то користувачу В можна рекомендувати ті товари, що придбав користувач А, але ще не придбав користувач В.

Враховуючи структуру спроектованого WEB-ресурсу для підбору музичних інструментів, на даному рівні недоцільно використовувати user-based підхід, так як на даний момент не передбачено систематизоване збереження інформації про користувача і загальна інтеграція цієї інформації у спільну систему.

Проте інформація про музичні інструменти (об’єкти, представлені на WEB-ресурсі) наявна в повному обсязі. Таким чином, було прийнято рішення спинитися на item-based підході та використовувати в якості опорних ознак дані про інструменти.

Рекомендаційний алгоритм спиратиметься на активність користувача під час поточної сесії. В результаті його дій певним заздалегідь визначеним стандартним чином формуються списки вподобань, а потім на їх основі оцінюються всі наявні інструменти. Користувачу рекомендується перелік з кількох товарів, які система визначає як найдоречніші для нього.

Існує багато готових рекомендаційних алгоритмів, в тому числі таких, що базуються на машинному навчанні. Проте при розробці даного WEB-ресурсу, враховуючи складність інтеграції сторонніх блоків і відсутність даних про активність користувачів для того, щоб сформувати трейнову та тестову вибірку

і здійснити навчання моделі, було прийнято рішення реалізувати алгоритм як чітку послідовність кроків (дій) з мінімальним використанням чужих моделей.

Загалом алгоритм можна описати лінгвістично (словами) або схематично (блок-схемою). Для розробки алгоритму роботи веб-ресурсу для підбору музичних інструментів обрано схематичний спосіб опису за допомогою структурних блоків. Перевага такого підходу в тому, що логіку взаємодії користувача з системою можна візуалізувати єдиним малюнком і передати програмісту без подальших уточнень.

Варто зазначити, що активність користувача не обов'язково відбувається у суворій послідовності (як і у випадку з UML-діаграмою послідовності). При розробці блок-схеми та текстового алгоритму було враховано варіативність дій користувача у тій мірі, в якій це можливо з використанням послідовностей в цілому.

Окрім основної блок-схеми функціонування сайту, також представлено окрему блок-схему алгоритму рекомендацій товарів.

Блок-схему веб-ресурсу для підбору музичних інструментів наведено на рисунку 2.6.

Алгоритм функціонування WEB-ресурсу складається з таких кроків:

Крок №1. Користувач заходить на головну сторінку веб-ресурсу та приймає рішення, чи бажає він здійснити пошук інструментів (або продовжити його).

- Якщо так → перехід до кроку №2.
- Якщо ні → перехід до кроку №7.

Крок №2. Користувач обирає тип музичного інструмента.

Крок №3. Користувач обирає конкретний інструмент із переліку

Крок №4. Система відображає детальну інформацію про обраний інструмент.

Крок №5. Система демонструє медіа-матеріали (фото, відео) обраного інструмента. За бажанням користувач може здійснити інтерактивну взаємодію з даними матеріалами.

Крок №6. Користувач вирішує, чи додати товар до корзини:

- Якщо так → товар додається до корзини і перехід до кроку №8.
- Якщо ні → повернення до кроку №2.

Крок №7. Система пропонує користувачу перейти до оформлення покупки (корзина).

- Якщо так → перехід до кроку №8.
- Якщо ні → завершення роботи алгоритму.

Крок №8. Перевірка статусу авторизації користувача:

- Якщо користувач авторизований → перехід до кроку №10.
- Якщо не авторизований → завантажується вікно авторизації (крок №9).

Крок №9. Користувач вводить логін і пароль; система перевіряє облікові дані:

- Якщо дані знайдено в базі → користувач авторизується і алгоритм повертається до кроку №8.
- Якщо не знайдено → повторний виклик процесу авторизації або повідомлення про помилку.

Крок №10. Система відображає вміст корзини та запускає алгоритм рекомендацій.

Крок №11. Користувач вирішує, чи оформити покупку:

- Якщо так → система переходить до оформлення замовлення і завершує роботу алгоритму.
- Якщо ні → завершення роботи алгоритму.

Нижче наведено алгоритм рекомендацій у вигляді впорядкованої послідовності дій для підбору трьох найрелевантніших товарів на основі історії взаємодій користувача. Використання такого алгоритму підвищує точність рекомендацій та зменшує час пошуку потрібних позицій.

Перевагою блок-схеми є наочна візуалізація логіки, чітке розділення етапів обробки даних і прийняття рішень, а також спрощена передача вимог між аналітиками та розробниками.

Для опису алгоритму застосовано схематичний спосіб за допомогою структурних блоків [17].

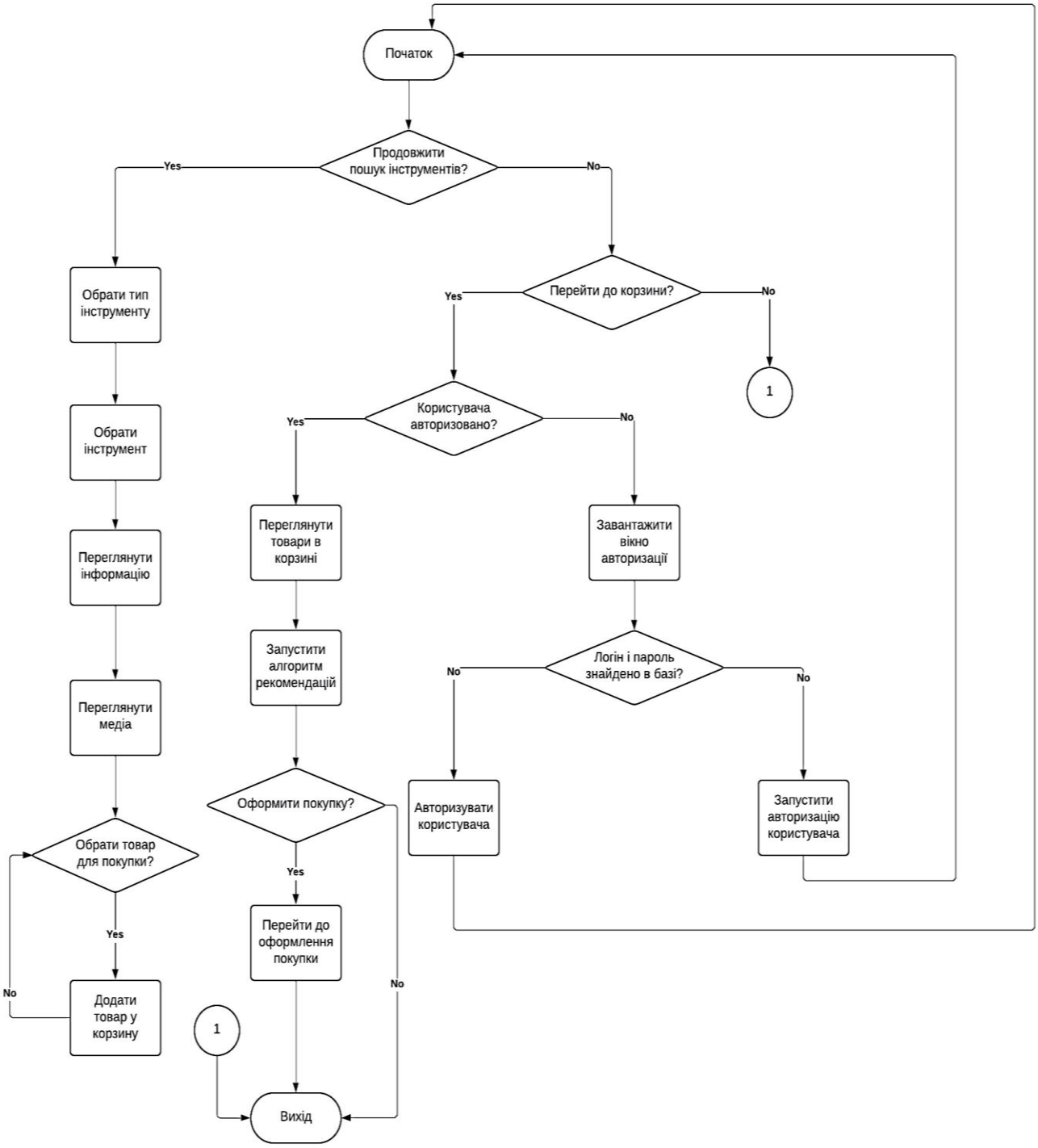

Рисунок 2.6 – Блок-схема роботи веб-ресурсу для підбору музичних інструментів

Блок-схему алгоритму системи рекомендацій веб-ресурсу для підбору музичних інструментів наведено на рисунку 2.7 та 2.8.

Алгоритм рекомендацій складається з таких кроків:

Крок №1. Система збирає для кожного товару статистику переглядів K[i] та визначає, які товари вже знаходяться в корзині.

Крок №2. Ініціалізуються масиви: L[i] := 0 (вага товару) і score[i] := 0 (накопичений бал схожості). Ініціалізовані масиви використовуватимуться для подальшого розподілу товарів за схожістю та їх видачі в рекомендаційній системі.

Крок №3. Для кожного товару, що міститься в корзині, присвоюється L[i] := 1.

Крок №4. Для кожного товару з історії переглядів, у якого K[i] > 0, встановлюється L[i] := 1, якщо K[i] = 1, або L[i] := 2, якщо K[i] > 1.

Крок №5. Для всіх товарів з L[i] > 0 і кожного їхнього атрибута j заповнюється матриця attr[i][j] := L[i].

Крок №6. На основі профільних товарів обчислюється середня ціна Price.Average.

Крок №7. Для кожного товару з L[i] > 0 перевіряється збіг його атрибутів із профілем користувача; якщо збігаються, до score[i] додається L[i].

Крок №8. У підалгоритмі здійснюється перевірка, чи належить ціна товару до меж Price.Average; у разі позитивної відповіді до score[i] додаються додаткові 2 бали.

Крок №9. Після оцінювання всіх товарів виконується сортування за score[i] у порядку спадання.

Крок №10. Формуються три товари з найвищими балами схожості та відображаються користувачу як рекомендації. Для відображення товарів запускається відповідний блок клієнтської частини.

Крок №11. Алгоритм завершено; за оновлення даних його можна запускати повторно.

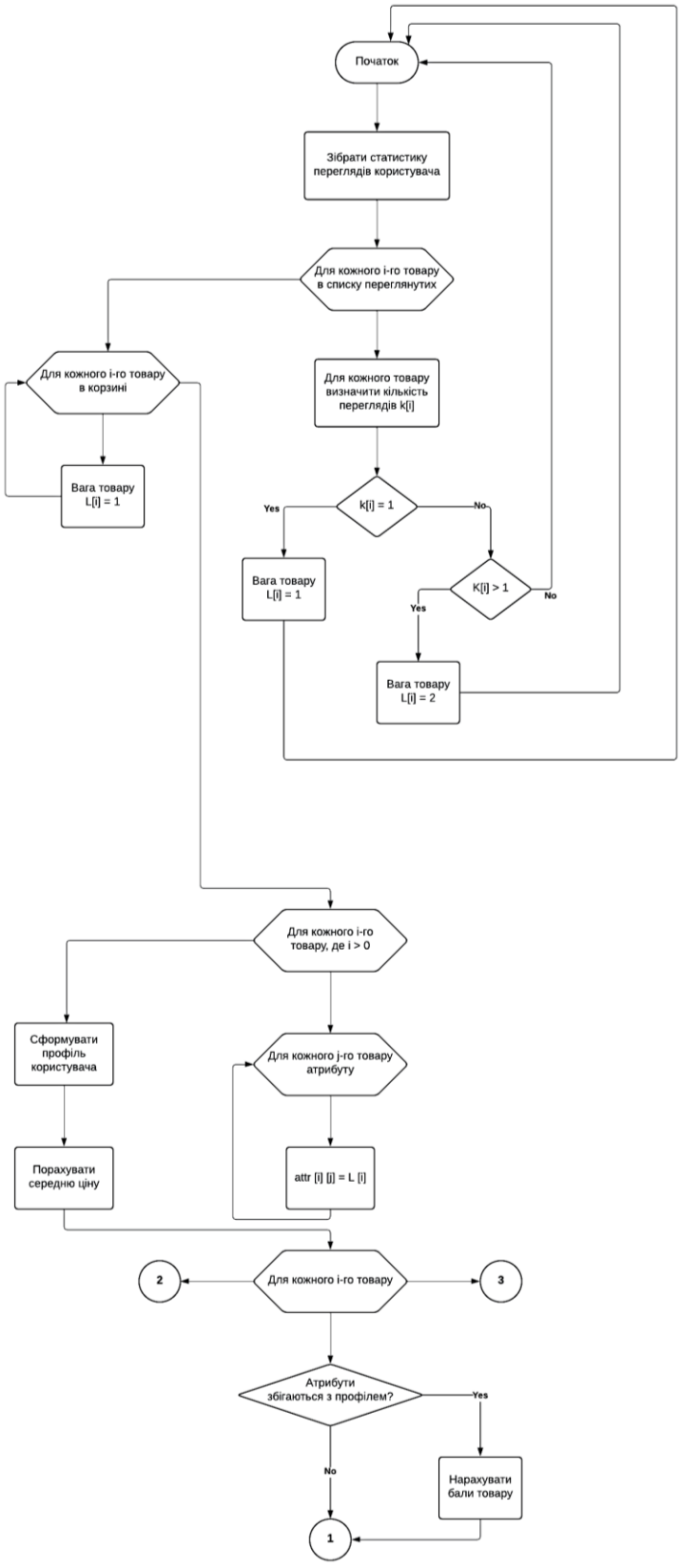

Рисунок 2.7 – Блок-схема алгоритму системи рекомендацій веб-ресурсу для підбору музичних інструментів

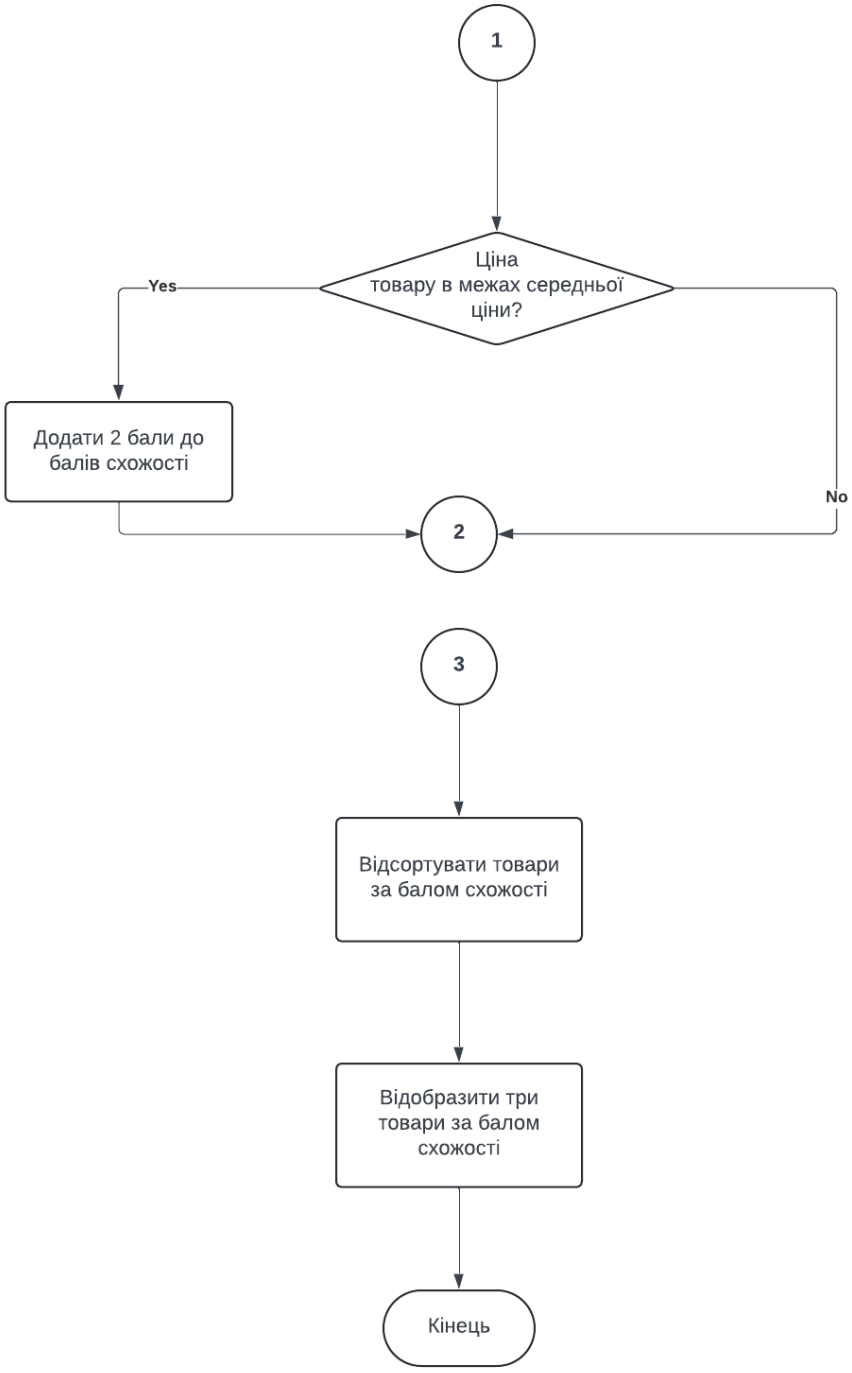

Рисунок 2.7.1 – Продовження

2.4 Висновок

У розділі було здійснено комплексне моделювання програмного модуля інтернет-магазину музичних інструментів із використанням як UML-діаграм, так і блок-схем. Для візуалізації основних взаємодій із користувачем створено діаграму прецедентів, що ілюструє ключові сценарії роботи з сайтом. Структурна схема компонентів розкриває внутрішню архітектуру модуля та зв’язки між його підсистемами. Діаграма класів відображає об’єктно-орієнтовану модель з властивостями й асоціаціями, а діаграма послідовностей демонструє хронологію викликів під час виконання типових операцій.

Крім того, було розроблено дві блок-схеми:

- блок-схема загальної логіки роботи сайту, що крок за кроком описує процес пошуку інструментів, додавання товарів до кошика та оформлення замовлення;
- блок-схема алгоритму рекомендацій, яка деталізує обчислення ваги товарів за історією переглядів і кошиком, побудову профілю користувача, перевірку цінового діапазону та відбір трьох найрелевантніших позицій.

Поєднання UML-діаграм і блок-схем дозволяє одразу охопити як статичну структуру, так і динамічну поведінку системи, а також забезпечує високу прозорість і гнучкість подальшої реалізації програмного модулю.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ ДЛЯ ПІДБОРУ МУЗИЧНИХ ІНСТРУМЕНТІВ

3.1 Обґрунтування вибору мови та бібліотек для розробки

У рамках розробки онлайн-магазину музичних інструментів було проведено аналіз двох архітектурних підходів: класичного клієнт-серверного рішення на базі PHP з фронтендом на HTML + CSS та сучасного SPA-стеку JavaScript (React на клієнті і Node.js/Express на сервері).

Першочергово розглянемо класичний підхід з PHP. Класичний підхід із PHP та HTML + CSS передбачає виконання всіх динамічних запитів на сервері за допомогою PHP скриптів і побудову готових HTML-сторінок, які віддаються користувачеві. Особливості цього підходу:

1. Кожна дія користувача, яка потребує нових даних, спричиняє повне перезавантаження сторінки;
2. Базове SEO-індексування працює без додаткових налаштувань завдяки прямому доступу пошуковиків до готового HTML;
3. Для запуску достатньо стандартного хостингу з підтримкою PHP і файлової системи;
4. Будь-яка динаміка понад прості форми потребує підключення окремих JavaScript-скриптів.

Перевагами класичного підходу на базі PHP із фронтендом на HTML / CSS є те, що розгортання можна здійснити на будь-якому стандартному хостингу без додаткових налаштувань, сам процес старту надзвичайно простий і не потребує складних інструментів чи бандлерів, а готовий HTML одразу «з коробки» індексується пошуковими системами та забезпечує високу SEO-дружність.

Недоліками є те, що кожна взаємодія з ресурсом, яка потребує оновлення даних, спричиняє повне перезавантаження сторінки і відчутну затримку для користувача, при зростанні навантаження сервер починає суттєво гальмувати через інтерпретацію PHP і генерацію HTML, а відсутність єдиної компонентної

структури ускладнює повторне використання коду та його підтримку у великих проєктах. Це стало причиною пошуку альтернатив при розробці WEB-ресурсів.

Класичний підхід досі активно використовується і для великих, і для малих проєктів, але вимагає значного вдосконалення.

Сучасний підхід із React на клієнті та Node.js / Express на сервері базується на принципах Single-Page Application, коли після початкового завантаження відбувається миттєве оновлення лише змінених фрагментів інтерфейсу без повних перезавантажень. Цей стек вирізняється такими рисами:

5. Чітка компонентна структура інтерфейсу дозволяє повторно використовувати блоки коду;

6. Асинхронні запити через fetch або axios оновлюють лише необхідні дані;

7. Неблокуюча модель введення-виведення Node.js ефективно обробляє численні одночасні API-запити;

8. Статична типізація TypeScript допомагає уникнути багатьох помилок ще на етапі компіляції та спрощує рефакторинг і подальшу модифікацію коду.

У стеку React + Node.js / Express перевагами є плавний та чуйний користувацький досвід завдяки SPA-архітектурі, коли після початкового завантаження оновлюються лише ті частини інтерфейсу, що змінилися; компонентна модель React спрощує організацію коду, його тестування та повторне використання; неблокуюча модель введення-виведення Node.js дозволяє ефективно обробляти велику кількість одночасних запитів, а TypeScript зменшує кількість помилок на етапі компіляції та полегшує рефакторинг.

Недоліками є те, що проєкт вимагає початкової конфігурації бандлерів (Vite, Webpack) і транспіляторів, що збільшує час на налаштування середовища розробки; перше завантаження клієнта може бути довшим через обсяг JavaScript-бандлу; а для досягнення повної SEO-дружності необхідне впровадження серверного рендерингу або пререндерингу, що додає ще один рівень складності. [18, 19].

Додатково складемо порівняльну таблицю цих двох стеків. Інформацію наведено у таблиці 3.1.

Таблиця 3.1 – Порівняльна таблиця двох стеків мов програмування

Критерій	**PHP + HTML / CSS**	**React + Node.js / Express**
Модель рендерингу	MPA – кожен запит повертає повний HTML	SPA – динамічне оновлення фрагментів через віртуальний DOM
Інтерактивність	мінімальна, лише базова динаміка через окремі скрипти	нативні реактивні оновлення UI без перезавантажень
Модель запитів	повні HTTP-запити	асинхронні AJAX/Fetch, WebSocket для реального часу
Навантаження на сервер	високе – кожен запит ініціює інтерпретацію PHP і генерує HTML	менше – сервер повертає JSON, статику віддає CDN
Масштабованість	обмежена – монолітна логіка обробки запитів	висока – мікросервіси, розділення API і статики

Отже, для інтернет-магазину музичних інструментів, де критично важлива швидка реакція інтерфейсу, можливість обробки великої кількості одночасних запитів і простота масштабування, оптимальним рішенням є використання стеку React + Node.js / Express замість класичного PHP + HTML /CSS.

3.2 Обґрунтування вибору середовища розробки

Інтегроване середовище розробки (IDE) - це програмний додаток, розроблений для спрощення розробки, тестування та налагодження коду.

IDE включають різні інструменти та функції, за допомогою яких завдання розробки стандартизуються відповідно до мови програмування, що використовується.

Обираючи середовище розробки проаналізуємо найпопулярніші редактори кодів для створення WEB-ресурсів: Visual Studio Code, JetBrains IDEs, Sublime Text.

Visual Studio Code (VS Code) – це безкоштовне інтегроване середовище розробки (IDE) від Microsoft [20]. Розробка сайтів і додатків включає VS Code, вирізняючись своєю легковаговістю та швидкістю завантаження, водночас забезпечуючи повний набір функцій сучасної IDE: інтегрований термінал, вбудований відладчик, «розумне» автодоповнення IntelliSense та широкий каталог розширень у Marketplace.

Важливим аспектом є те, що Visual Studio Code може використовуватися для різних мов програмування та не вимагатиме переходу між різними середовищами для інтеграції нових частин програмного забезпечення. Це робить його популярним вибором, зокрема, для WEB-розробників, адже у інтернет-системах часто виникає потреба суміщення різноманітних елементів реалізації і їх комбінація в цілісній системі. Не кожне середовище розробки дозволяє здійснити цю інтеграцію екологічно.

Незважаючи на помітні переваги в гнучкості, потребує достатньо оперативної пам’яті та може іноді уповільнюватися через надмірну кількість плагінів.

Дане середовище розробки користується великою популярністю як серед професійних розробників, так і серед початківців за рахунок низького «порогу входу» (безкоштовне та не вимагає специфічних знань для базового використання).

Загальний вигляд інтерфейсного вікна редактора коду Visual Studio Code наведено на рисунку 3.1.

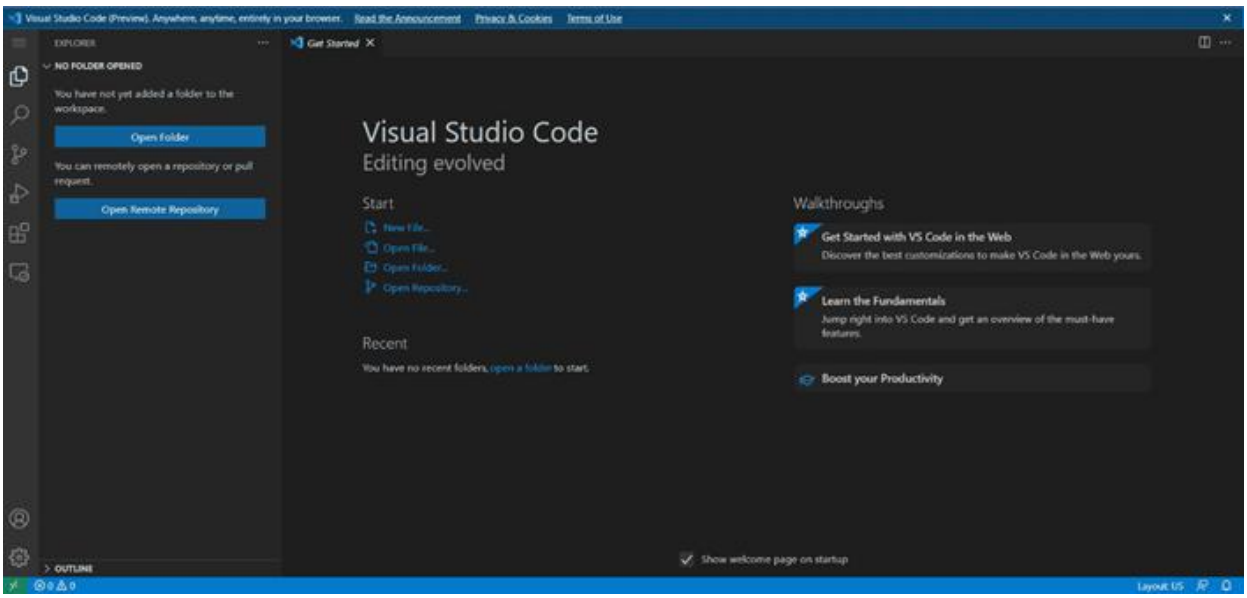

Рисунок 3.1 - Загальний вигляд інтерфейсного вікна редактора коду Visual Studio Code

JetBrains IDEs забезпечують найглибший аналіз коду «з коробки», автоматично виявляючи помилки в процесі набору тексту, пропонуючи контекстні виправлення і виконуючи складний рефакторинг. Вбудовані засоби для роботи з базами даних, контейнерами Docker та профілюванням додатків роблять ці IDE незамінними в проєктах великого масштабу. Водночас висока продуктивність потребує інвестицій у платну ліцензію та потужніші апаратні ресурси.

Дані IDE існують для різних мов програмування та різних екосистем розробки. Деякі з них мають окремі безкоштовні (стандартні) версії та корпоративні, що вимагають плати. Також існують варіанти, що розповсюджуються вільно лише для некомерційного використання [21].

Важлива перевага у вигляді широкого спектру функціональних можливостей з іншого боку і виступає недоліком даного сімейства середовищ розробки. Дані середовища вимагають досить багато обчислювальних ресурсів, ставлять високі вимоги до персонального комп’ютера користувача та можуть бути складними для новачка. З іншого боку, це зручний інструмент для

професійної роботи, що надає спектр можливостей, недоступний або менш зручно реалізований у інших середовищах.

Загальний вигляд інтерфейсного вікна редактора коду JetBrains IDE наведено на рисунку 3.2.

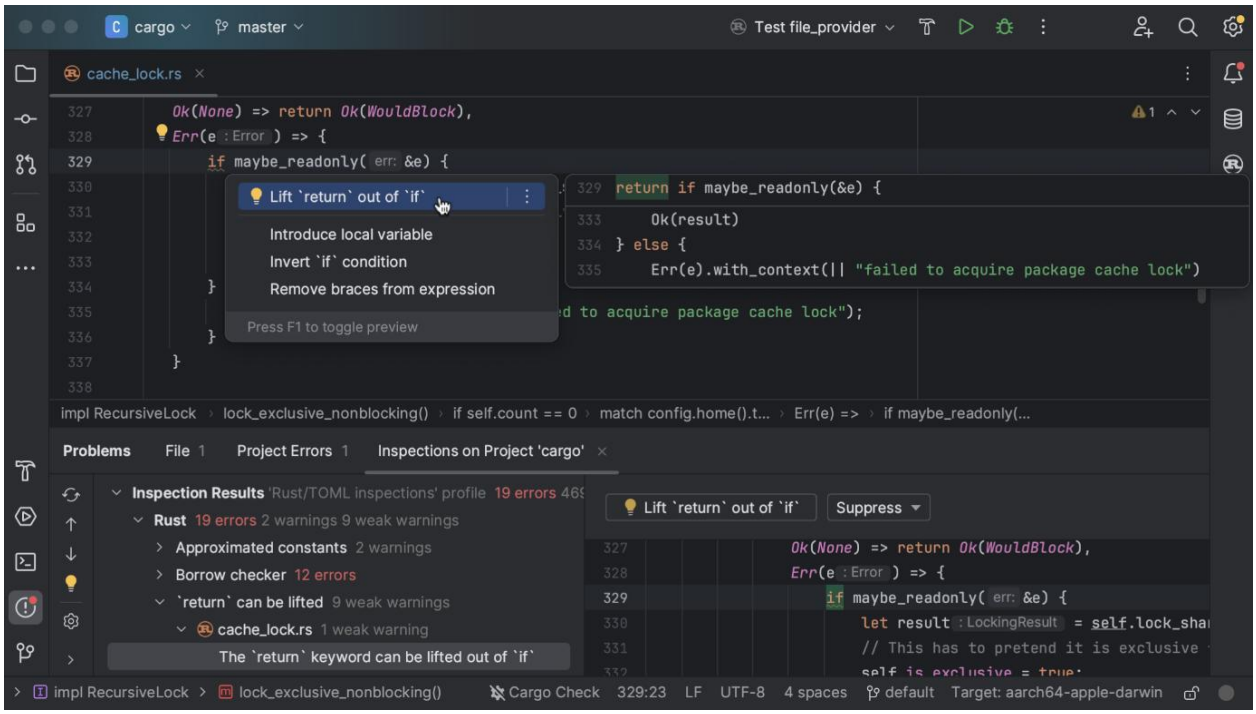

Рисунок 3.2 - Загальний вигляд інтерфейсного вікна редактора коду JetBrains IDE

Sublime Text має кардинально інший підхід: мінімалістичний інтерфейс і швидкодію, яка не відчутна жодними затримками. Така простота обмежує можливості вбудованого відлагодження та інтеграції з системами контролю версій і тестування, а підтримка розширень не настільки потужна, як у VS Code або JetBrains IDEs. Це робить його придатним для невеликих довідкових правок або роботи з текстом, але менш зручним для розробки великих SPA-додатків.

Загальний вигляд інтерфейсного вікна редактора коду Sublime Text наведено на рисунку 3.3.

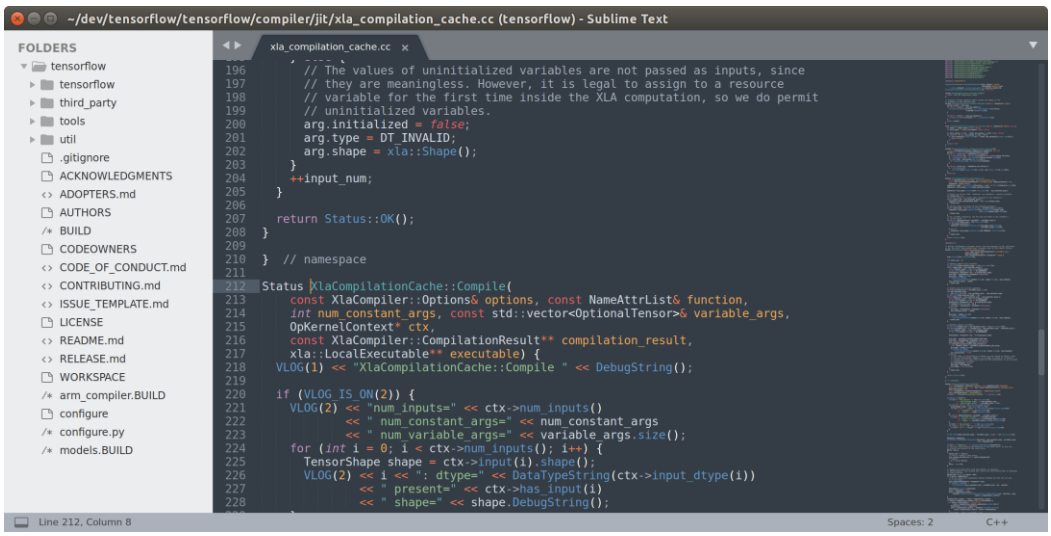

Рисунок 3.3 - Загальний вигляд інтерфейсного вікна редактора коду Sublime Text

Оскільки проєкт передбачає розробку сучасного WEB-ресурсу із необхідністю швидкого переходу між різними технологіями, автоматизованого відлагодження та інтеграції з Git, у якості основного редактора було обрано Visual Studio Code.

Завдяки поєднанню продуктивності, гнучкості налаштувань і безкоштовної ліцензії, VS Code надає універсальне середовище, яке відповідає вимогам командної розробки та дозволяє оптимізувати робочий процес. Дана IDE дає можливість повністю реалізувати всі поставлені програмні задачі та достягнути поставленої мети.

3.3 Розробка клієнтської та серверної частини

У сучасній архітектурі вебзастосунку чітко розмежовуються дві основні складові – клієнтська (frontend) та серверна (backend) частини, які разом забезпечують злагоджену роботу веб-ресурсу для підбору музичних інструментів. Клієнтська сторона відповідає за взаємодію безпосередньо з користувачем: саме тут формується інтерфейс, відтворюється структура сторінок, обробляються події натискання кнопок та змінюються дані в реальному часі без перезавантаження вікна браузера. Завдяки використанню HTML та CSS

формується базова розмітка й стиль, а JavaScript, доповнений сучасними підходами (React, TypeScript), робить цей інтерфейс інтерактивним – наприклад, дозволяє миттєво фільтрувати каталог інструментів або оновлювати вміст кошика покупок. При цьому інструменти збірки (Vite) та система типізації (TypeScript) забезпечують швидкий «гарячий» перезапуск під час розробки і мінімізують ризики помилок ще на етапі компіляції.

В точці входу ми виконуємо ініціалізацію. React монтує компонент App у кореневий елемент DOM, при цьому ми обгортаємо його в провайдер Redux для зберігання глобального стану й у BrowserRouter для клієнтської маршрутизації без перезавантаження сторінки.

В самому компоненті App.tsx описані маршрути, що відповідають за різні екрани, а саме: головна сторінка з переліком інструментів, сторінка категорії, деталі конкретного товару, кошик і особистий кабінет. Перехід між ними обробляється на клієнті — URL змінюється, але повна перезагрузка не відбувається, що забезпечує миттєву реакцію на дії користувача. При цьому вбудована логіка перевіряє наявність токену в Redux-слайсі auth. Якщо користувач не авторизований, спроба зайти в кошик одразу перенаправить його на головну:

```
export const fetchProducts = createAsyncThunk('products/fetchAll', async () =>
{
  const { data } = await axios.get('/api/products');
  return data;
});
```

Для керування станом застосовано Redux Toolkit. Всі сторінки звертаються до окремих «слайсів», які містять асинхронні операції з Axios до серверного API. Наприклад, на головній сторінці при завантаженні каталогу викликається thunk fetchProducts, який надсилає GET-запит і, отримавши список інструментів,

записує їх у сховище. Це дозволяє компонентам просто підписатися на зміни стану й відображати товари незалежно від того, коли саме прийшли дані.

```
export const fetchProducts = createAsyncThunk(
  'products/fetchAll',
  async () => {
    const { data } = await axios.get('/api/products');
    return data;
  }
);
```

Самі UI-компоненти розбиті на дві великі групи: «сторінки» (у папці pages) з основними екранами й «спільні» (у папці components або shared), куди винесені кнопки, картки продуктів, модальні вікна та інші дрібні елементи. Щоб зробити код більш гнучким, в проєкті використовуються кастомні React-хуки для типових операцій, наприклад, useToggle для відкриття/закриття модалки або useFetch для загального шаблону завантаження даних.

Стилізація здійснюється за допомогою SASS та модульних SCSS-файлів: кожна React-компонента має свій .module.scss, де описані лише її стилі.

Серверна частина має працювати як простий сервіс на JavaScript, що приймає запити від браузера, звертається до сховища даних і повертає результат у форматі JSON. Коли користувач натискає «Додати в кошик» або відкриває сторінку товару, клієнт надсилає HTTP-запит, сервер обробляє його, читає або змінює відповідні записи в базі й віддає назад необхідні поля – наприклад, список інструментів, підтвердження успішного логіну чи результат обчислення вартості замовлення.

Усередині серверу є простий модуль для роботи з даними: він читає товари, замовлення й профілі користувачів із MongoDB. Цю базу було обрано, так як вона є документоорієнтованою і не вимагає опису схеми таблиць, що робить її

зручною для використання у WEB-ресурсах, що не вимагають важкої структуризації даних [23].

У цьому випадку дані організовані у вигляді документів, де кожен інструмент має свою категорію, рейтинг і лічильник продажів. Коли потрібно отримати будь-які дані – сервер звертається до цієї колекції й віддає результат.

Система рекомендацій у веб-ресурсі зараз реалізується так: коли користувач заходить на сторінку з пропозиціями, клієнт надсилає запит до сервера, а сервер аналізує історію його замовлень та визначає категорії інструментів, які його цікавили.

Якщо користувач новий і ще не має замовлень, буде повернено загальні топ-продажі. Інакше зібрані категорії використовуються для пошуку інших товарів у тих самих групах, потім із цього набору видаляються вже придбані позиції, а залишок сортується за комбінованим показником популярності. Розглянемо кроки алгоритму рекомендацій покроково.

1. Спочатку отримуємо з бази даних всі замовлення поточного користувача та витягаємо з них категорії інструментів:

```
const orders = await Order.find({ user: userId });
const categories = [...new Set(
  orders.flatMap(o => o.items.map(i => i.category))
)];
```

2. Якщо користувач уже робив покупки, беремо всі продукти з цих категорій; якщо він ще новачок, просто завантажуємо загальні топ-продажі:

```
let candidates;
if (categories.length === 0) {
  candidates = await Product.find().sort({ soldCount: -1 }).limit(10);
} else {
  candidates = await Product.find({ category: { $in: categories } });
}
```

3. Щоб не рекомендувати те, що користувач уже має, відкидаємо всі продукти з його історії замовлень:

```
const purchasedIds = new Set(
  orders.flatMap(o => o.items.map(i => i.productId.toString()))
);
candidates = candidates.filter(p => !purchasedIds.has(p._id.toString()));
```

4. Для кожного залишеного кандидата рахуємо зважений бал за формулою:

score = rating * 0.7 + soldCount * 0.3, де:

rating — середня оцінка продукту (наприклад, відгуки користувачів)

soldCount — кількість продажів цього товару

Вибір коефіцієнтів 0.7 і 0.3 підкреслює, що ми надаємо пріоритет якісним товарам із хорошими рейтингами, але також враховуємо їхню популярність у продажах.

5. Після того, як у кожного кандидата є свій score, сортуємо список у спадному порядку і вибираємо перші N позицій (наприклад, 5):

```
scored.sort((a, b) => b.score - a.score);
const recommendations = scored.slice(0, 5).map(x => x.item);
```

В результаті клієнту повертається масив рекомендацій, де кожен інструмент відповідає найбільш релевантним уподобанням користувача: спочатку відбрані категорії, потім відфільтровані вже куплені, і нарешті найвищий зважений бал на основі рейтингу та популярності.

3.4 Тестування роботи програми і аналіз результатів

Тестування програмного забезпечення є невід’ємним етапом створення будь-якого веб-ресурсу. Воно дозволяє переконатися, що платформа надає

користувачам достовірну й актуальну інформацію. Завдяки перевірці працездатності, продуктивності та надійності модуля підбору музичних інструментів можна виявити й усунути можливі недоліки. Аналіз результатів тестування дає змогу зробити об’єктивний висновок про якість та ефективність розробленого рішення.

Для запуску веб-ресурсу переконайтеся, що на вашому комп’ютері встановлені Node.js та npm, а також глобально — pnpm (за потреби виконайте npm install -g pnpm).

Паралельно треба інсталювати MongoDB. Демон mongod має працювати, а для ручної роботи з базою ви можете використовувати консоль mongosh. Потім встановіть залежності (npm install) і запустіть сервер за допомогою npm run dev. У консолі з’явиться повідомлення типу «Server running at http://192.168.0.111:3000». Необхідно запам’ятати вказаний IP та порт. Після цього необхідно у файлі .env встановити значення порта.

Запустіть клієнт командою pnpm devVite покаже локальну та мережеву адреси. Відкрийте одну з них у браузері, щоб побачити інтерфейс магазину й переконатися, що всі запити до серверу коректно обробляються.

Після запуску програми з’явиться головна сторінка веб-ресурсу, яка подана на рисунку 3.4. У ній представлено меню, для легкої навігації та комфортного користування. Сайт виглядає стилістично гарно та є повністю зрозумілим користувачу.

Також є короткий опис кожного виду інструменту, навігаційна кнопка та розділ з подіями (рис. 3.5).

Кожна з кнопок на головній сторінці є клікабельною та реагує на дії користувача.

Реакція кнопок повністю відповідає технічному завданню та логічному задуму. Вона є передбачуваною та здійснюється в межах базових уявлень користувача про роботу сайту на основі досвіду користування подібними продуктами у різних сферах.

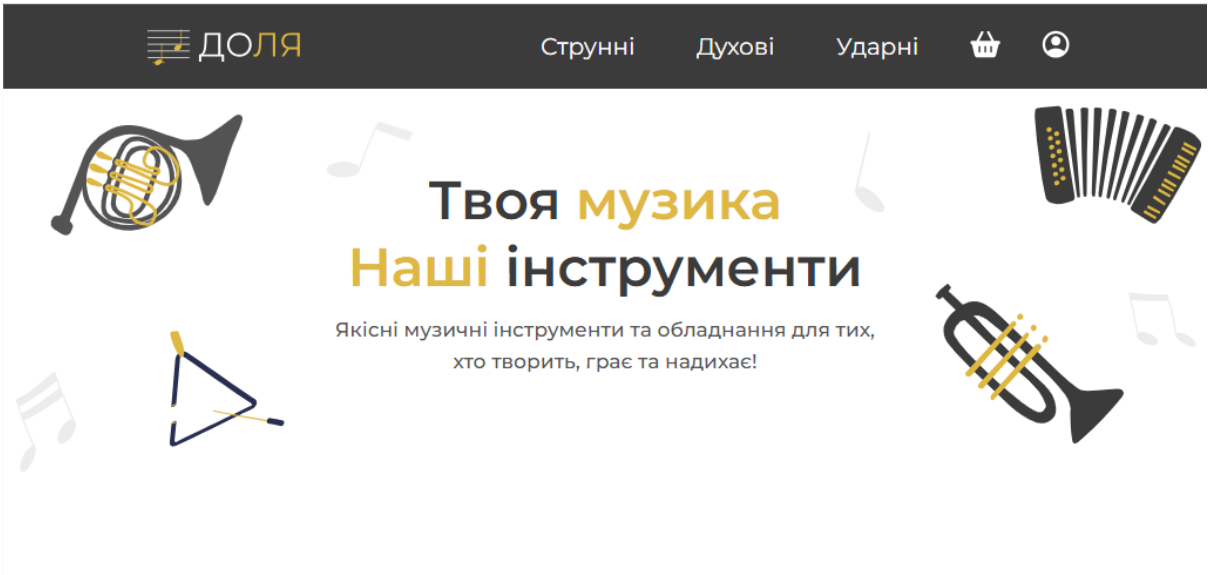

Рисунок 3.4 – Головна сторінка WEB-ресурку для підбору музичних інструментів

З «шапки» головної сторінки, що залишається видимою на будь-якій іншій сторінці сайту, можна перейти до категорій товарів (струнні, духові, ударні) або перейти до кошика чи вийти на сторінку авторизації/реєстрації, як і передбачалось при проектуванні.

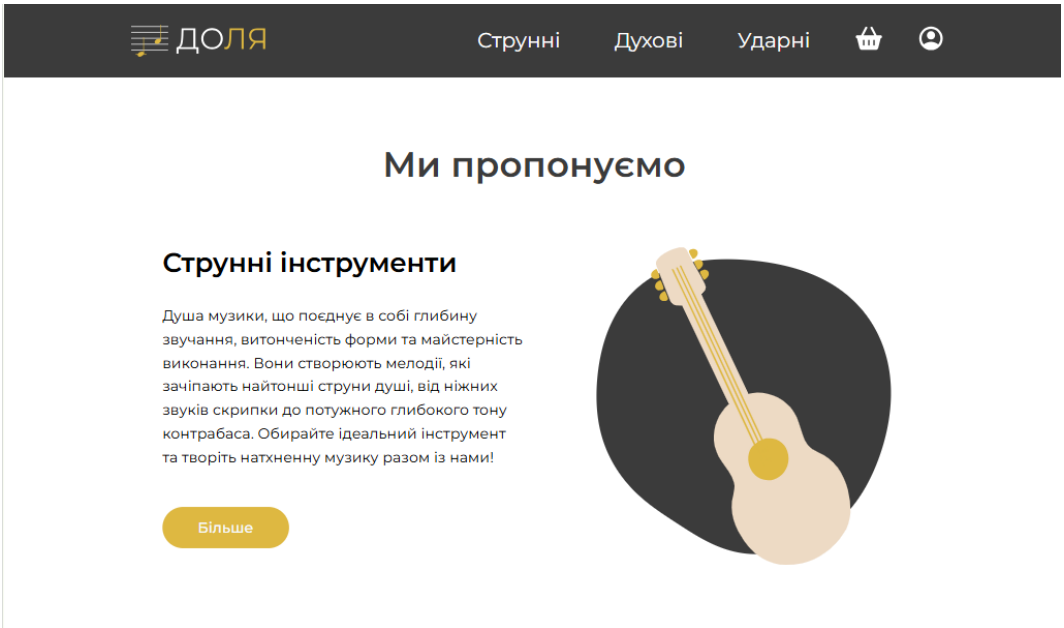

Рисунок 3.5 - Головна сторінка WEB-ресурку для підбору музичних інструментів

Натискаючи на кнопку «Більше» можна перейти до сторінки самого інструменту, де присутній короткий опис походження тих чи інших інструментів та поділення на категорії. Сторінку інструментів показано на рисунку 3.6.

Рисунок 3.6 – Сторінка струнних інструментів

Прогортавши нижче можна переглянути товари за категоріями, а саме: назву інструмента, зображення та ціну (рис. 3.7).

Кожний елемент категорії є клікабельним, іконки реагують на наведення курсору та на клік мишкою.

Товари різних категорій відображені ідентично, різні сторінки сайту виконані у схожій стилістиці, його робота представляє собою ансамбль узгоджених елементів.

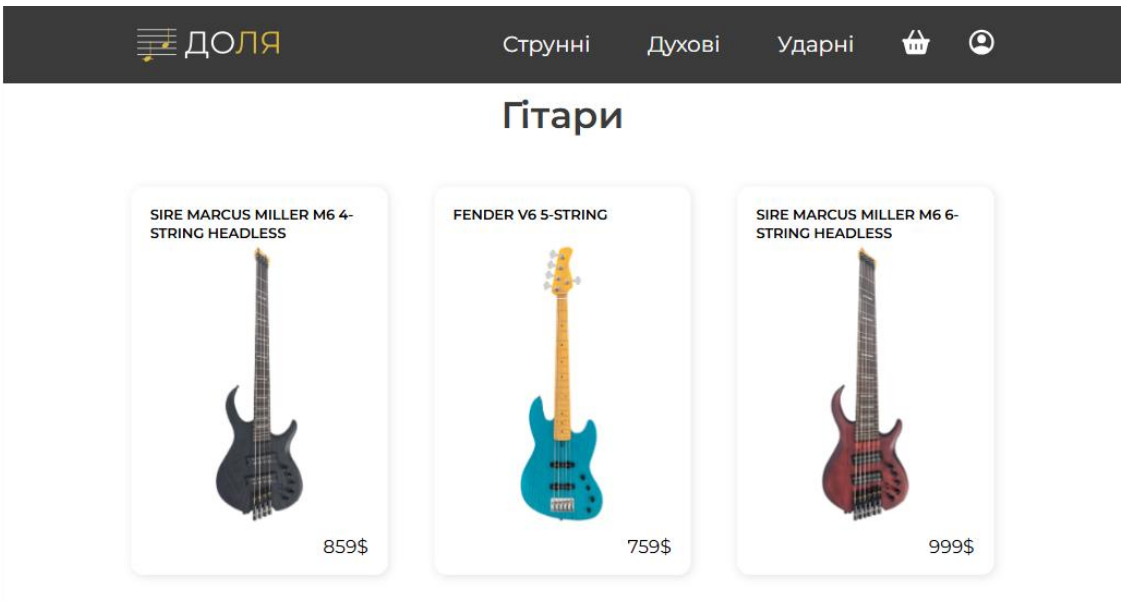

Рисунок 3.7 - Сторінка струнних інструментів та товари

Натиснувши на певний інструмент – ми переходимо до особистої сторінки товару. На ній показана карусель з фото інструмента, аудіо- та відео-матеріали, які допоможуть користувачеві у виборі. Сторінку інструмента показано на рисунку 3.8.

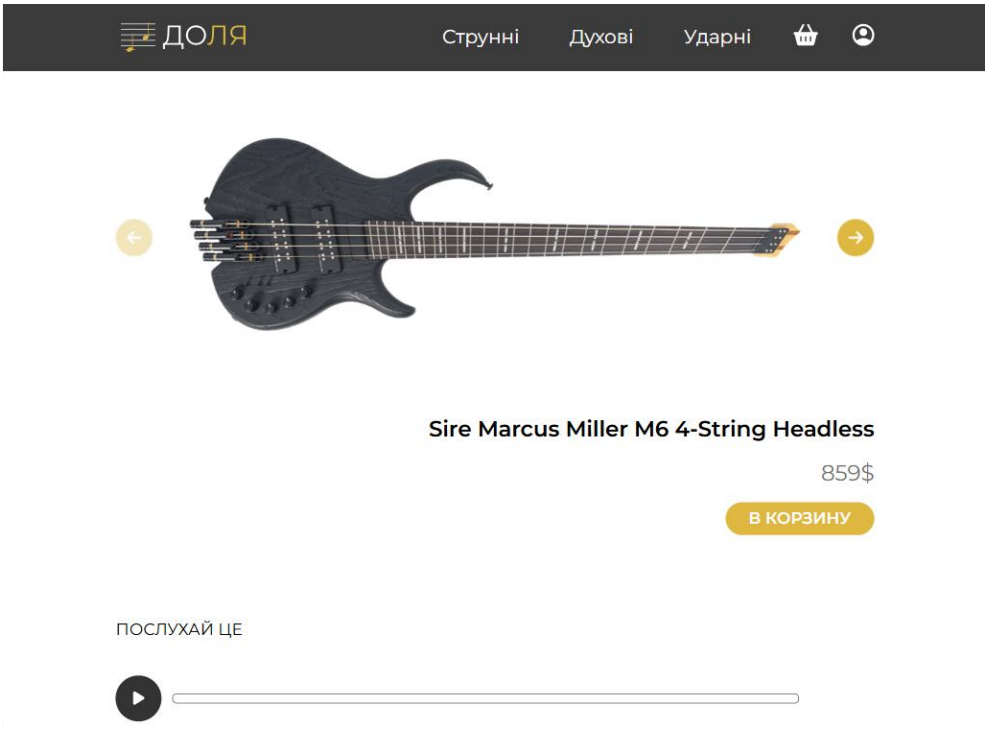

Рисунок 3.8 – Сторінка інструмента

На сторінці відображається карусель з зображенням, медіаелементи, ціна. Кнопка додавання в корзину активна.

При натисканні кнопки програвача запускається мелодія, що відповідає даному інструменту.

При натисненні кнопки програвання відео, якщо відео наявне на даній сторінці, починає відтворюватися відео. Мультимедійність працює у відповідності зі сформованим технічним завданням.

Інструмент може бути доданим до корзини лише у випадку авторизації користувача, як і передбачено технічним завданням.

У правому верхньому куточку користувач здійснює авторизацію в особистий кабінет та перехід до корзини товарів, які він попередньо додав. Вікно авторизації показано на рисунку 3.9. Користувач може увійти до свого акаунту або зареєструвати новий акаунт, якщо цього не було зроблено раніше.

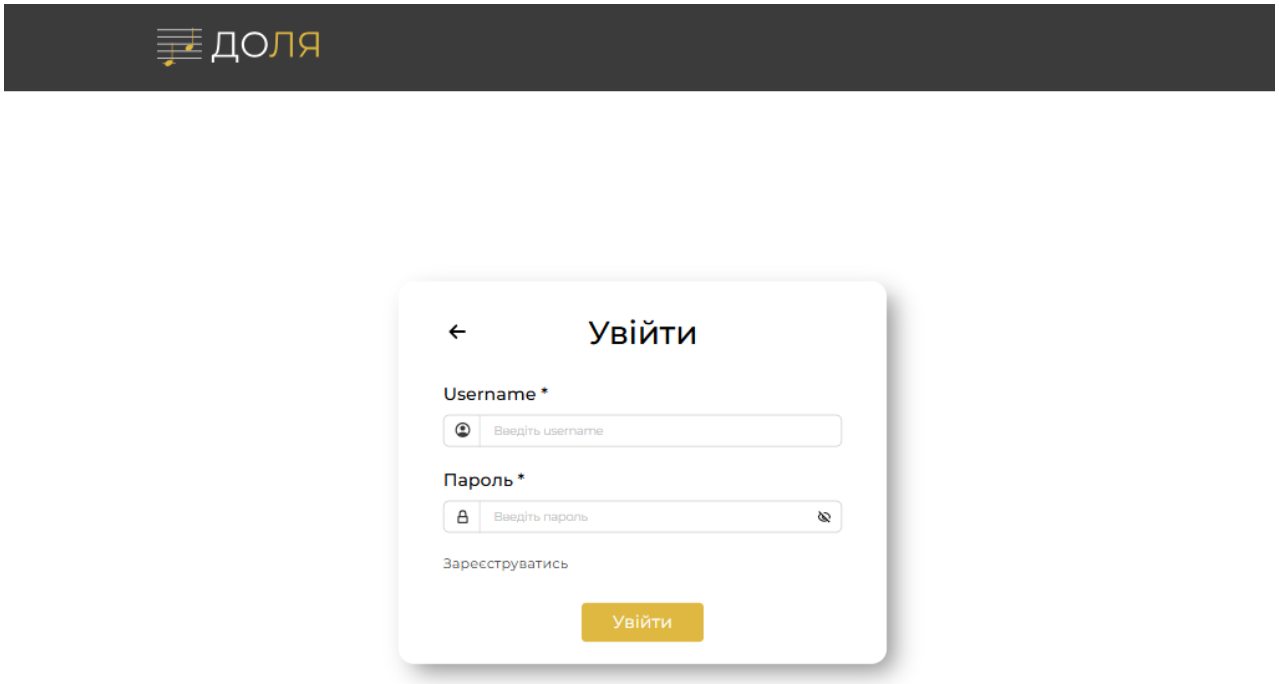

Рисунок 3.9 – Вікно авторизації

Проведемо тестування алгоритму рекомендацій. Для цього:

1. Переглянемо сторінку Товару 1 та додамо його в корзину.
2. Переглянемо сторінку Товару 2.
3. Переглянемо сторінку Товару 3 двічі.
4. Переглянемо сторінку Товару 4 двічі та додамо його в корзину.

Алгоритм повинен працювати у відповідності з описом та блок-схемою, наведеному у розділі 2.3.

Якщо товар переглянуто один раз, йому присвоюється вага 1; якщо два чи більше — вага 2. Товарам у кошику призначається більша вага — 3, що вказує на високий рівень зацікавленості. Отже, найбільша вага має бути у Товару 4, далі – Товар 1, Товар 2. Так як система рекомендацій відображає лише три товари, то Товар 3 не буде показаний. Його пріоритетність за даним алгоритмом була визначена як найменша.

Результати системи рекомендацій за алгоритмом наведено на рисунку 3.10.

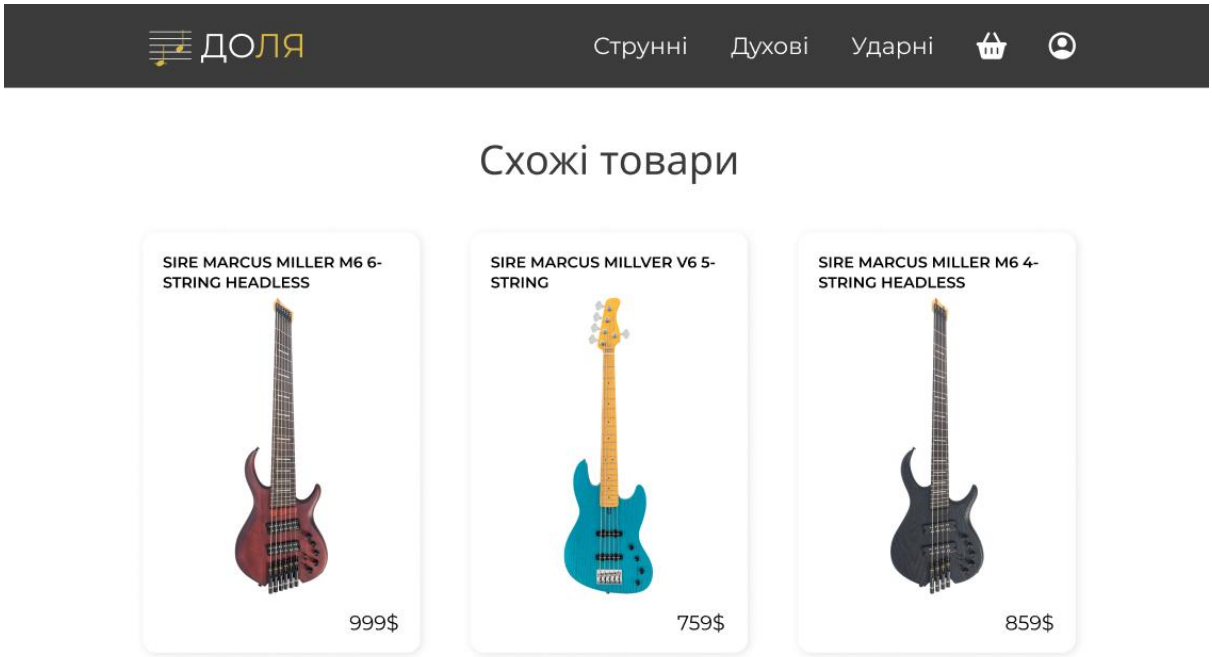

Рисунок 3.10 – Результати системи рекомендацій за алгоритмом

Товари на сторінці рекомендацій активні. Користувач може перейти на сторінку товару, додатково переглянути її і за бажанням додати до корзини. Подальші можливі дії користувача вже описані при тестуванні вище.

Користувач має можливість багатократно повторювати різні дії та динамічно взаємодіяти з розробленим програмним забезпеченням.

Таким чином, WEB-ресурсом для підбору музичних інструментів реалізовано у відповідності з поставленим завданням. Усі необідні функції реалізовано в повному обсязі.

Порівняємо розробку з програмами-аналогами.

В таблиці 3.2 подана порівняльна характеристика програм-аналогів з розробленим WEB-ресурсом.

На основі аналізу даних, поданих у таблиці, можна зробити висновок, що розроблений WEB-ресурс значно перевершує свої аналоги Musicca, 8notes, Metronom.ua та Muztorg.ua за функціональністю.

Платформа поєднує підтримку мультимедійних оглядів інструментів із безпечною системою реєстрації й авторизації, що зберігає персональні налаштування й історію переглядів, має початковий каталог ключових типів інструментів і модульну архітектуру для швидкого масштабування. Це забезпечує користувачам більш інтуїтивний, інтерактивний і персоналізований досвід.

Розроблений ресурс не передбачає наявність пошукової системи, так як це не було передбачено при проектуванні. Додавання цього компоненту є доцільним при подальшому масштабуванні системи.

Система рекомендацій працює у відповідності до визначеного алгоритму, базуючись на item-підході. Адекватність рекомендації товарів було перевірено і підтверджено при тестуванні. Наявність системи рекомендацій вигідно виділяє ресурс серед аналогів.

Таблиця 3.2 – Порівняльна характеристика програм-аналогів з розробленим WEB-ресурсом.

	Розроблений продукт	«Musicca»	«8notes»	«Metro-nom.ua»	«Muztorg.ua»
Мультиме-дійність	Так	Так	Так	Ні	Ні
Реєстрація та авторизація	Так	Так	Так	Так	Так
Пошукова система	Ні	Ні	Так	Так	Так
Типи інструментів	Кілька	Жодного	Кілька	Жодного	Всі типи
Використання сучасних веб-технологій в дизайні	Так	Ні	Так	Так	Ні

Отже, розробка нашого веб-ресурсу повністю виправдана та краща за існуючі рішення, оскільки поєднує найважливіші можливості та надає користувачам можливість здійснювати покупку товарів, спираючись на опис інструмента, медіа- та аудіо-додатки.

3.5 Висновок

Під час виконання цього розділу було проведено ґрунтовний аналіз інструментів розробки, реалізація спроектованого у другому розділі ресурсу для підбору музичних інструментів та його тестування у відповідності до поставлених задач. Таким чином, досягнуто поставленої у бакалаврській роботі мети і виконано всі задачі.

Ресурс є безкоштовним і не містить реклами, виконаний за допомогою широко відомих засобів реалізації WEB-розробки, які можуть відтворюватися сучасними браузерами, що робить його максимально доступним для широкого кола користувачів.

Порівняння результатів роботи з аналогами (Musicca, 8notes, Metronom.ua, Muztorg.ua) демонструє помітне підвищення функціональності створеного продукту: інтеграція мультимедійних оглядів інструментів, безпечна персоналізована система реєстрації з історією переглядів, наявність стартового каталогу ключових типів інструментів та модульна архітектура, що забезпечує легке масштабування. Це суттєво підвищує інтуїтивність інтерфейсу, заохочує користувачів до активної взаємодії та забезпечує комфорт при роботі з платформою.

Отже, результати аналізу та тестування не лише підтверджують досягнення поставлених цілей, але й свідчать про те, що створений WEB-ресурс є виправданим та корисним інструментом, який перевершує існуючі рішення і закладає міцну основу для подальшого розвитку й розширення функціоналу.

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи було розроблено WEB-ресурс для підбору музичних інструментів.

Усі задачі, поставлені для реалізації WEB-ресурсу для підбору музичних інструментів, були виконані у повному обсязі, а саме:

– проведено аналіз предметної області, розглянуто аналоги, описано їх переваги та недоліки та обгрунтовано актуальність розробки власного ресурсу;

– розроблено структуру програмного забезпечення WEB-ресурсу для підбору музичних інструментів.

– розроблено алгоритм для надання персональних рекомендацій з вибору музичних інструментів на основі активності користувача.

– програмно реалізовано WEB-ресурс для підбору музичних інструментів та розроблений алгоритм надання персональних рекомендацій;

– проведено тестування розробленого ресурсу;

– розроблено інструкцію користувача.

У роботі на основі аналізу існуючих рішень, було доведено актуальність розробки веб-ресурсу для підбору музичних інструментів. Крім того, було розроблено структурну схему роботи програмного забезпечення та кілька діаграм, а саме: діаграма прецедентів, діаграма класів, діаграма послідовностей. Результатом розробки є схема роботи веб-ресурсу та алгоритму рекомендацій.

Клієнтська частина реалізована з використанням React (JSX), HTML5, CSS3 та чистого JavaScript, що забезпечує компонентний підхід і динамічну взаємодію з користувачем. Серверну частину створено на Node.js із фреймворком Express, а дані зберігаються в MongoDB. Такий стек технологій гарантує швидкість, масштабованість і зручність подальшого розвитку проєкту.

Проведено тестування розробленого WEB-ресурсу. Під час тестування програми не було виявлено несправностей у системі. WEB-ресурс працює коректно та відповідає поставленим вимогам користувача.

Мета роботи, яка полягала в розширенні функціональних можливостей WEB-ресурсу для підбору музичних інструментів, досягнута. Розроблений веб-ресурс переважає аналоги у 4 з 5 характеристик, а саме: мультимедійність, реєстрація та авторизація, типи інструментів та використання сучасних веб-технологій в дизайні.

Отже, розроблений WEB-ресурс для підбору музичних інструментів виконує всі заплановані завдання і виходить за межі функціоналу програм-аналогів. При виконанні бакалаврської кваліфікаційної роботи було виконано всі задачі та досягнуто поставленої мети.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джулай А.О. Шолота В.В. Перспективи розробки WEB-ресурсу підбору музичних інструментів. *LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації.* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23155 (дата звернення: 21.05.2025).

2. Психологічні особливості впливу музики на емоційний стан. URL: https://fact-news.com.ua/psixologichni-osoblivosti-vplivu-muziki-na-emotsijnij-stan-vazhlivist-ritmu-i-melodii/ (дата звернення: 21.05.2025).

3. Лісовська Т. О. Теорія і методика музичного виховання. МНУ ім. В.О. Сухомлинського. Миколаїв: СПД Румянцева Г. В., 2022. 324 с.

4. Соловей Я. Г. Музика у духовному розвитку особистості. *Науковий вісник Мукачівського державного університету. Серія «Педагогіка та психологія».* Мукачево, 2015. № 1(1). С. 121-125.

5. Біньковська А.Б., Дудник О. Аналіз мультимедійних технологій в сучасному житті. Наукові тренди постіндустріального суспільства: матеріали VIII міжнар. наук. конф., (м. Суми, 27 вересня 2024 р.). Суми: ХНАДУ, 2024. С. 145–150.

6. Вдовичин Т.Я., Лазурчак Л.В. Проектування інформаційно-пошукових систем як засіб використання сучасних технологій. Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки. - 2022. Т. 33(72). № 4. С. 66–71. DOI https://doi.org/10.32838/2663-5941/2022.4/11

7. Musicca. URL: https://www.musicca.com/ (дата звернення: 21.05.2025).

8. 8notes. URL: https://www.8notes.com/ (дата звернення: 21.05.2025).

9. Метроном. URL: https://www.metronom.ua/ (дата звернення: 21.05.2025).

10. МузТорг. URL: https://muztorg.ua/ (дата звернення: 21.05.2025).

11. UML для бізнес-моделювання: для чого потрібні діаграми процесів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 21.05.2025).

12. Уніфікована мова моделювання (Unified Modeling Language - UML). URL: https://www.maxzosim.com/unifikovana-mova-modeluvannia/ (дата звернення: 21.05.2025).

13. Технології комп'ютерного проєктування : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / Т. О. Савчук, О. В. Ольшанська. – Вінниця : ВНТУ, 2024. – 125 с

14. L. Wang, J. Liu, X. Liu and L. Song, "A Personalized Recommendation Algorithm for E-commerce Based on User Purchase Intention," 2019 International Conference on Artificial Intelligence and Information Systems (ICAIIS), Cairo, Egypt, 2019, pp. 1-5.

15. Білошкурський В.О., Мельник І.С., Яровий А.А., Паночишин Ю.М. Інформаційна технологія надання рекомендацій щодо вибору ноутбука. *LII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації.* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/18718 (дата звернення: 21.05.2025).

16. Шептяков І.О., Левченко Н.Б., Ярова О.А., Яровий А.А. Особливості проектування WEB-орієнтованої інформаційної технології для розробки експертних систем. *LI Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації.* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15498 (дата звернення: 21.05.2025).

17. JavaScript Tutorial. URL: https://www.w3schools.com/js/default.asp (дата звернення: 21.05.2025).

18. Michael McMillan. Data Structures and Algorithms with JavaScript: Bringing classic computing approaches to the Web. — Sebastopol, CA : O'Reilly Media, 2014. — 384 с.

19. Яскевич В.О. JavaScript - бібліотека React. Навчальний посібник для студентів спеціальності Комп'ютерні науки. URL:

https://elibrary.kubg.edu.ua/id/eprint/48587/1/V_Yaskevych_tutorial_React.pdf (дата звернення: 21.05.2025).

20. Середовище розробки Visual Studio Code URL: https://code.visualstudio.com/docs (дата звернення: 21.05.2025).

21. WebStorm. The JavaScript and TypeScript IDE URL: https://www.jetbrains.com/webstorm/ (дата звернення: 21.05.2025).

22. MongoDB URL: https://www.mongodb.com/ (дата звернення: 21.05.2025).

23. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ 2023. – 54 с.

ДОДАТКИ

ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-ресурсу для підбору музичних інструментів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7,93 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Шолота В.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Джулай А.О.
(підпис) (прізвище, ініціали)

ДОДАТОК Б

Інструкція користувача

Для запуску веб-ресурсу переконайтеся, що на вашому комп’ютері встановлені Node.js та npm, а також глобально — pnpm (за потреби виконайте npm install -g pnpm).

Паралельно треба інсталювати MongoDB. Демон mongod має працювати, а для ручної роботи з базою ви можете використовувати консоль mongosh. Потім встановіть залежності (npm install) і запустіть сервер за допомогою npm run dev.

У консолі з’явиться повідомлення типу «Server running at http://192.168.0.111:3000». Необхідно запам’ятати вказаний IP та порт (рис. Г.1).

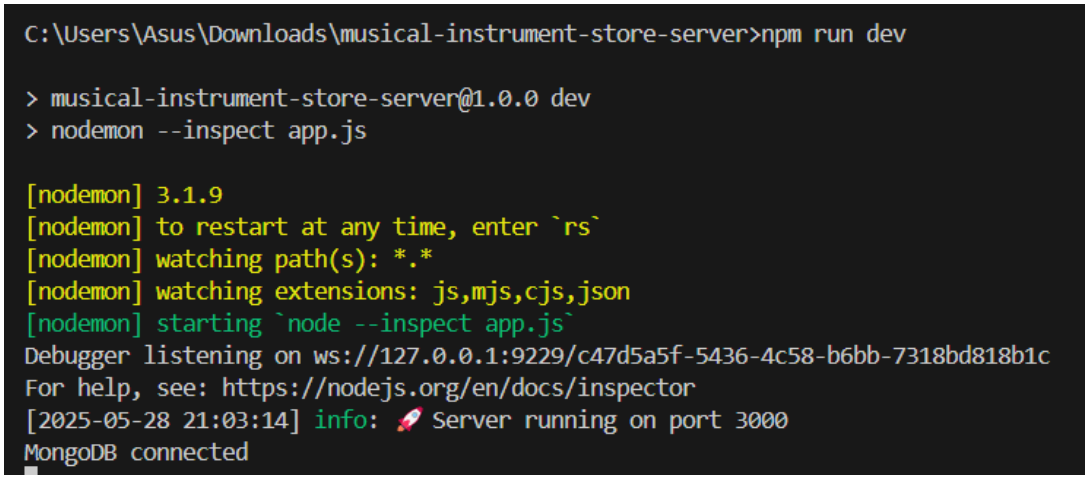

Рисунок Г.1 – Запуск сервера

Після цього необхідно у файлі .env встановити значення порта. (рис. Г.2)

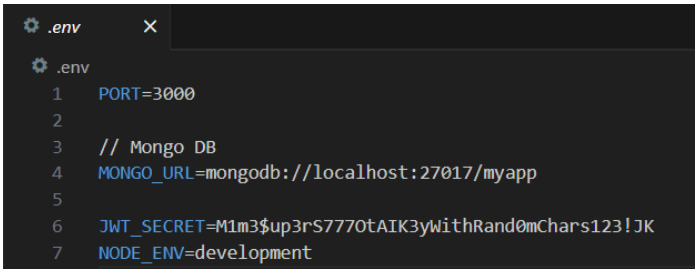

Рисунок Г.2 – Визначення порта

Запустіть клієнт командою pnpm dev. Vite покаже локальну та мережеву адреси (рис. Г.3). Відкрийте одну з них у браузері, щоб побачити інтерфейс магазину й переконатися, що всі запити до серверу коректно обробляються.

```
C:\Users\Asus\Downloads\musical-instrument-store-client>pnpm dev

> musical-instrument-store@0.0.0 dev C:\Users\Asus\Downloads\musical-instrument-store-client
> vite --host


  VITE v6.3.1  ready in 992 ms

  → Local:   http://localhost:5173/
  → Network: http://192.168.31.201:5173/
  → press h + enter to show help
```

Рисунок Г.3 – Відображення локальної та мережевої адреси

Після запуску програми з’явиться головна сторінка веб-ресурсу, яка подана на рисунку Г.4. У ній представлено меню, для легкої навігації та комфортного користування.

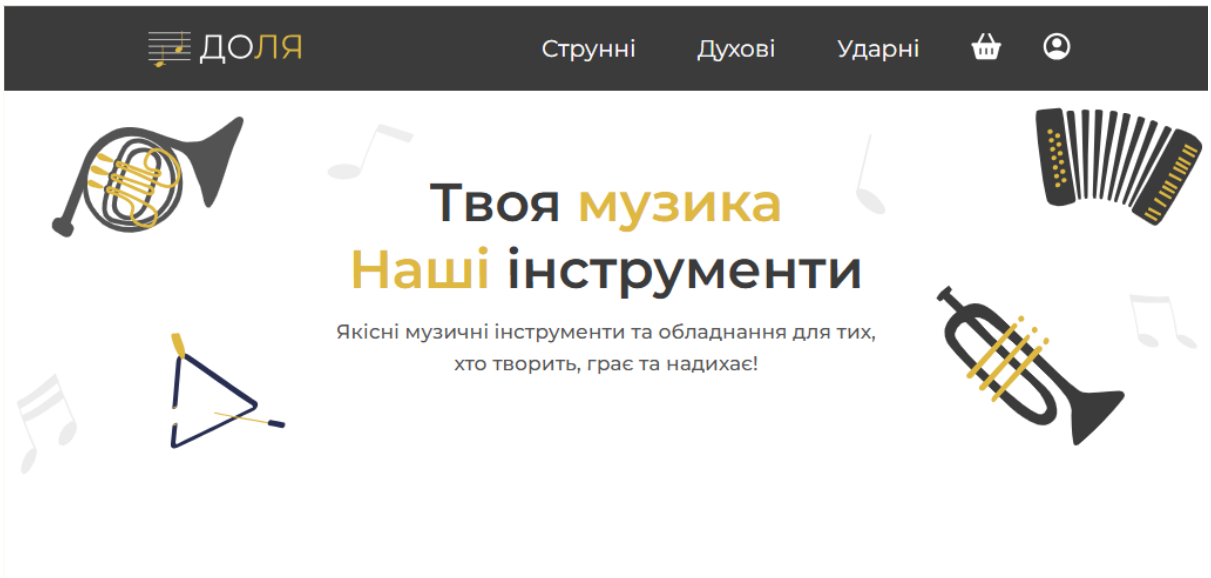

Рисунок Г.4 – Головна сторінка WEB-ресурку для підбору музичних інструментів

Також є короткий опис кожного виду інструменту, навігаційна кнопка та розділ з подіями (рис. Г.5).

Кожна з кнопок на головній сторінці є клікабельною та реагує на дії користувача.

З «шапки» головної сторінки, що залишається видимою на будь-якій іншій сторінці сайту, можна перейти до категорій товарів (струнні, духові, ударні) або

перейти до кошика чи вийти на сторінку авторизації/реєстрації, як і передбачалось при проектуванні.

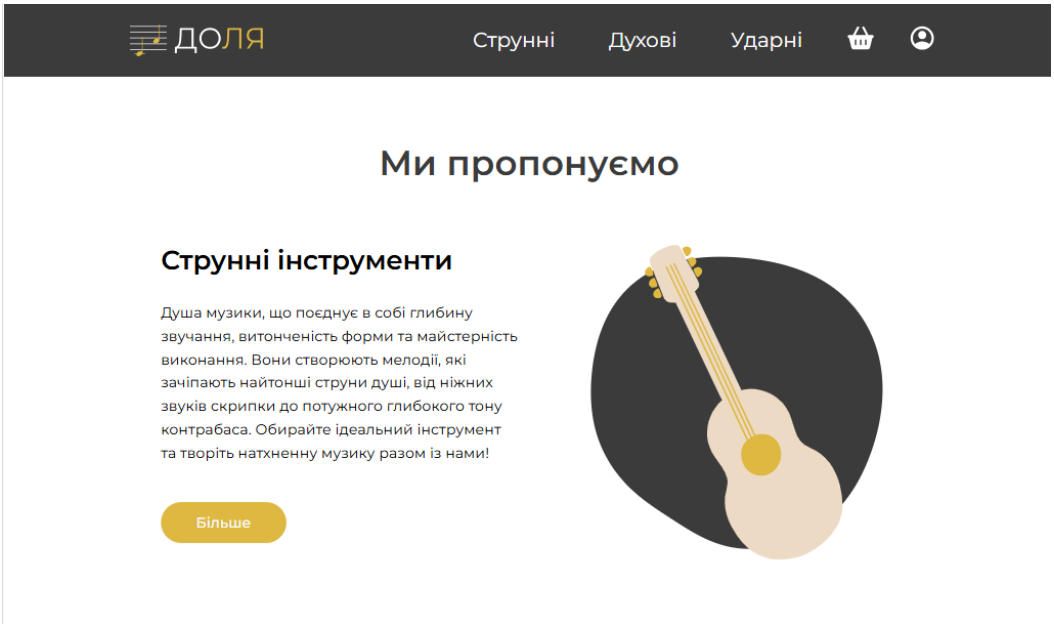

Рисунок Г.5 - Головна сторінка WEB-ресурку для підбору музичних інструментів

Натискаючи на кнопку «Більше» можна перейти до сторінки самого інструменту, де присутній короткий опис походження тих чи інших інструментів та поділення на категорії. Сторінку інструментів показано на рисунку Г.6.

Прогортавши нижче можна переглянути товари за категоріями, а саме: назву інструмента, зображення та ціну (рис. Г.7).

Натиснувши на певний інструмент – ми переходимо до особистої сторінки товару. На ній показана карусель з фото інструмента, аудіо- та відео-матеріали, які допоможуть користувачеві у виборі. Сторінку інструмента показано на рисунку Г.8.

Рисунок Г.6 – Сторінка струнних інструментів

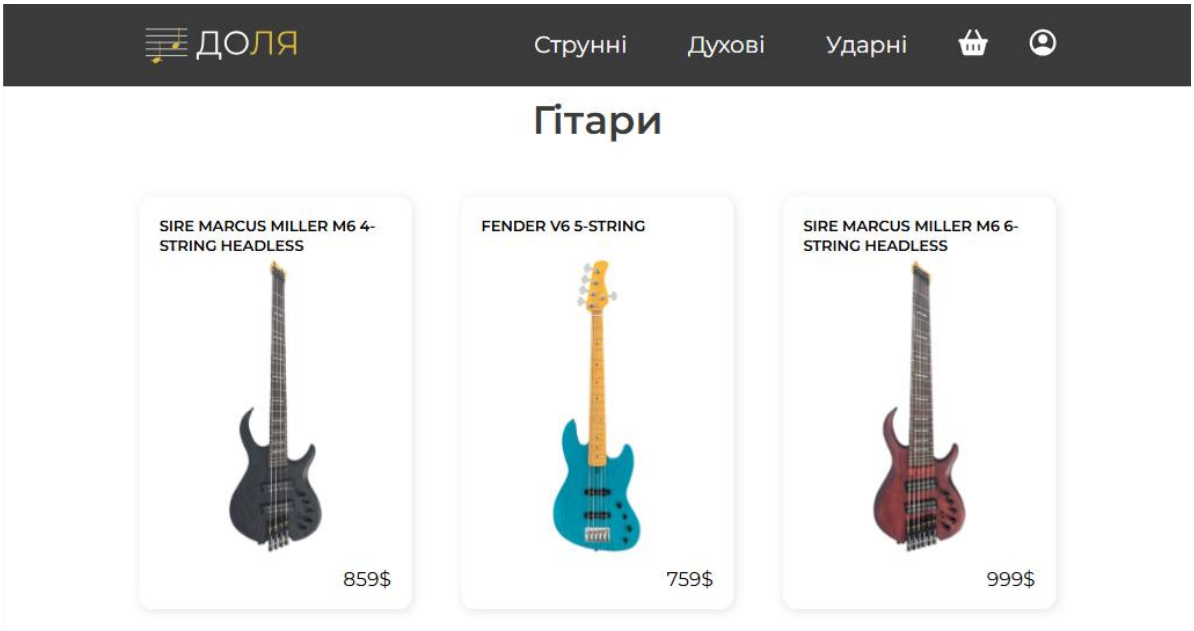

Рисунок Г.7 - Сторінка струнних інструментів та товари

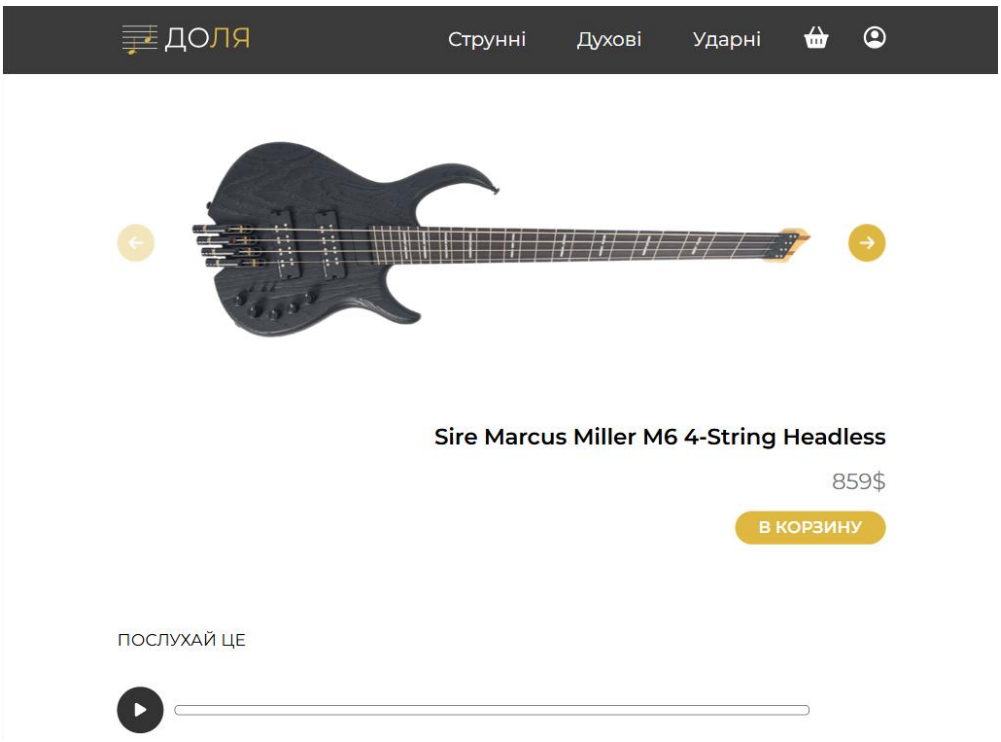

Рисунок Г.8 – Сторінка інструмента

Натиснувши на кнопку програвача почне грати мелодія, а натиснувши на відео відобразиться медіа стосовно відповідного інструмента.

У правому верхньому куточку користувач здійснює авторизацію в особистий кабінет та перехід до корзини товарів, які він попередньо додав. Вікно авторизації показано на рисунку Г.9.

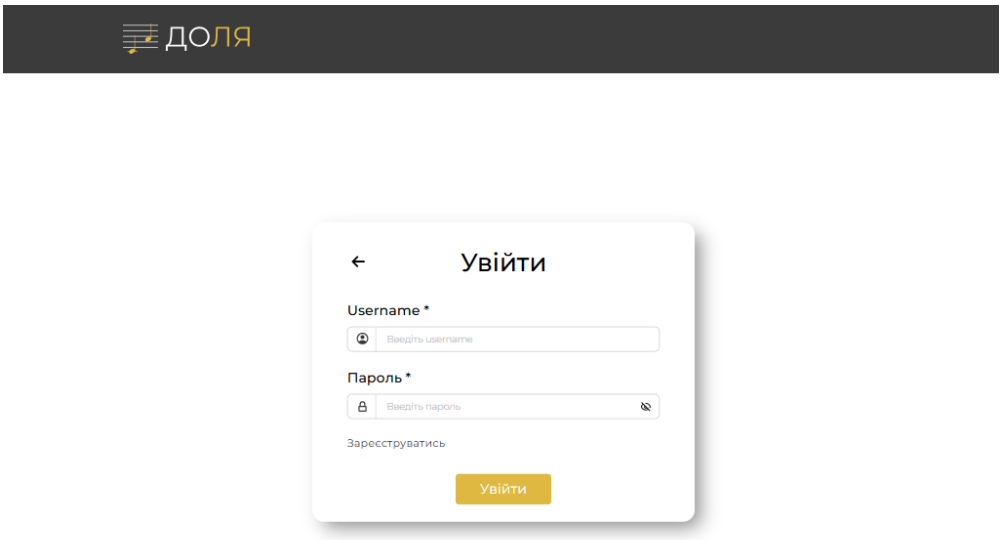

Рисунок Г.9 – Вікно авторизації

ДОДАТОК В

Лістинг програми

```
@use 'styles/resets' as *;

@use 'swiper/css' as *;
@use 'swiper/css/navigation' as *;

@font-face {
        font-family: 'Montserrat';
        src: url('assets/fonts/Montserrat-VariableFont_wght.ttf');
}

import { createAsyncThunk, createSlice, PayloadAction } from "@reduxjs/toolkit";
import axios, { AxiosError } from "axios";

interface SignUpData {
   email: string;
   username: string;
   password: string;
}

interface LoginData {
   username: string;
   password: string;
}

interface AuthError {
   message: string;
}

interface AuthState {
   loading: boolean;
   error: AuthError | null;
```

```
    isAuth: boolean;
}

const URL = import.meta.env.VITE_URL;

export const signUp = createAsyncThunk<
    void,
    SignUpData,
    { rejectValue: AuthError }
>("auth/signUp", async ({ email, password, username }, { rejectWithValue }) => {
    try {
        await axios.post(`${URL}3000/api/auth/sign-up`, {
            email,
            username,
            password,
        });
    } catch (error) {
        const err = error as AxiosError<{ message?: string }>;
        const message = err.response?.data?.message || "Помилка при реєстрації";

        return rejectWithValue({ message });
    }
});

export const login = createAsyncThunk<
    string,
    LoginData,
    { rejectValue: AuthError }
>("auth/login", async ({ username, password }, { rejectWithValue }) => {
    try {
        const response = await axios.post(`${URL}3000/api/auth/login`, {
            username,
            password,
        });
```

```
      const token = response.data.token;
      localStorage.setItem("token", token);
      return token;
    } catch (error) {
      const err = error as AxiosError<{ message?: string }>;
      const message = err.response?.data?.message || "Помилка при вході";

      return rejectWithValue({ message });
    }
});

export const getUser = createAsyncThunk<void, void, { rejectValue: AuthError }>(
  "auth/getUser",
  async (_, { rejectWithValue }) => {
    try {
      const token = localStorage.getItem("token");
      if (!token) throw new Error("Немає токену");

      const response = await axios.get(`${URL}3000/api/users/profile`, {
        headers: {
          Authorization: `${token}`,
        },
      });

      console.log(response.data);
    } catch (error) {
      const err = error as AxiosError<{ message?: string }>;
      const message =
        err.response?.data?.message || "Помилка при отриманні профілю";

      return rejectWithValue({ message });
    }
  }
```

```
);

const initialState: AuthState = {
    loading: false,
    error: null,
    isAuth: false,
};

const authSlice = createSlice({
    name: "auth",
    initialState,
    reducers: {
        logout(state) {
            state.isAuth = false;
        },
        setAuth(state, action: PayloadAction<boolean>) {
            state.isAuth = action.payload;
        },
    },
    extraReducers: (builder) => {
        builder
            .addCase(signUp.pending, (state) => {
                state.loading = true;
                state.error = null;
            })
            .addCase(signUp.fulfilled, (state) => {
                state.loading = false;
            })
            .addCase(signUp.rejected, (state, action) => {
                state.loading = false;
                if (action.payload) {
                    state.error = action.payload;
                }
            })
```

```
        .addCase(login.pending, (state) => {
            state.loading = true;
            state.error = null;
            state.isAuth = false;
        })
        .addCase(login.fulfilled, (state) => {
            state.loading = false;
            state.isAuth = true;
        })
        .addCase(login.rejected, (state, action) => {
            state.loading = false;
            if (action.payload) {
                state.error = action.payload;
            }
            state.isAuth = false;
        })
        .addCase(getUser.pending, (state) => {
            state.loading = true;
            state.error = null;
        })
        .addCase(getUser.fulfilled, (state) => {
            state.loading = false;
            state.isAuth = true;
        })
        .addCase(getUser.rejected, (state, action) => {
            state.loading = false;
            if (action.payload) {
                state.error = action.payload;
            }
            state.isAuth = false;
        });
    },
});
```

```
export const { setAuth, logout } = authSlice.actions;
export default authSlice.reducer;

import {
    Action,
    configureStore,
    ThunkAction,
    combineReducers,
} from "@reduxjs/toolkit";
import storage from "redux-persist/lib/storage";
import {
    FLUSH,
    PAUSE,
    PERSIST,
    persistReducer,
    persistStore,
    PURGE,
    REGISTER,
    REHYDRATE,
} from "redux-persist";
import authReducer from "./authSlice";
import productReducer from "./productsSlice";

const rootReducer = combineReducers({
    auth: authReducer,
    product: productReducer,
});

const persistConfig = {
    key: "root",
    storage,
};

const persistedReducer = persistReducer(persistConfig, rootReducer);
```

```
export const store = configureStore({
    reducer: persistedReducer,
    middleware: (getDefaultMiddleware) =>
        getDefaultMiddleware({
            serializableCheck: false,
            ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
        }),
});
export const persistor = persistStore(store);
export type AppDispatch = typeof store.dispatch;

export type RootState = ReturnType<typeof store.getState>;

export type AppThunk<ReturnType = void> = ThunkAction<
    ReturnType,
    RootState,
    unknown,
    Action<string>
>;

import { createSlice, PayloadAction } from "@reduxjs/toolkit";

export interface Product {
    id: number;
    title: string;
    color: string;
    type: string;
    price: string;
    productPage?: {
        description: string,
        youtubeLink: string,
        audio: string,
        generalImages: string[]
```

```
    }
}

interface ProductsState {
    viewedProducts: Product[];
    selectedProducts: Product[];
}

const initialState: ProductsState = {
    viewedProducts: [],
    selectedProducts: [],
};

const productsSlice = createSlice({
    name: "products",
    initialState,
    reducers: {
        addViewedProduct(state, action: PayloadAction<Product>) {
            const product = action.payload;
            state.viewedProducts.push(product);
        },
        addSelectedProduct(state, action: PayloadAction<Product>) {
            const product = action.payload;
            if (!state.selectedProducts.find((p) => p.id === product.id)) {
                state.selectedProducts.push(product);
            }
        },
        removeSelectedProduct(state, action: PayloadAction<number>) {
            state.selectedProducts = state.selectedProducts.filter(
                (product) => product.id !== action.payload
            );
        },
        clearAllData(state) {
            state.viewedProducts = [];
```

```
            state.selectedProducts = [];
        },
    },
});

export const {
    addViewedProduct,
    addSelectedProduct,
    removeSelectedProduct,
    clearAllData,
} = productsSlice.actions;

export default productsSlice.reducer;

import { Navigate, Outlet, useLocation } from "react-router-dom";
import { Route } from "./route.enum";

const blacklist: string[] = [Route.Login, Route.Registration];

const VISITED_PATHS_KEY = "visitedPaths";

interface ProtectedRouteProps {
    redirectPath?: Route;
}

const ProtectedRoutes = ({
    redirectPath = Route.General,
}: ProtectedRouteProps) => {
    const location = useLocation();
    const currentPath = location.pathname;
    const token = localStorage.getItem("token");

    const rawVisitedPaths = localStorage.getItem(VISITED_PATHS_KEY);
    const visitedPaths: string[] = rawVisitedPaths
```

```
        ? JSON.parse(rawVisitedPaths)
        : [];

    if (!blacklist.includes(currentPath)) {
        if (visitedPaths[visitedPaths.length - 1] !== currentPath) {
            visitedPaths.push(currentPath);
            localStorage.setItem(
                VISITED_PATHS_KEY,
                JSON.stringify(visitedPaths)
            );
        }
    }

    if (!token) {
        return <Navigate to={redirectPath} replace />;
    }

    if (blacklist.includes(currentPath)) {
        const filteredPaths = visitedPaths.filter(
            (path) => !blacklist.includes(path)
        );
        const lastVisitedPath = filteredPaths.length
            ? filteredPaths[filteredPaths.length - 1]
            : redirectPath;

        return <Navigate to={lastVisitedPath} replace />;
    }

    return <Outlet />;
};

export default ProtectedRoutes;

export enum Route {
```

```
    General = "/",
    Others = "*",

    Login = "login",
    Registration = "sign-up",
    ForgotPassword = "forgot-password",
    ResetPassword = "reset-password",

    Basket = "basket",
    SimilarProducts = "similar",

    Percussion = "percussion",
    Stringed = "stringed",
    Wind = "wind",
    Product = "product",
    ProductItem = "/stringed/:id",

    // MusicalInstruments = "instruments-catalog",
    // Product = "product",
}

{
  "name": "musical-instrument-store-server",
  "type": "module",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "dev": "nodemon --inspect app.js"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcryptjs": "^3.0.2",
```

```
    "cors": "^2.8.5",
    "dotenv": "^16.4.7",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.13.1",
    "winston": "^3.17.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.9"
  }
}

import express from 'express';

import dotenv from 'dotenv';
import cors from 'cors';
import connectDB from './config/db.js';
import userRoutes from './routes/userRoutes.js';
import authRoutes from './routes/authRoutes.js';
import errorMiddleware from './middleware/errorMiddleware.js';
import requestLogger from './middleware/loggerMiddleware.js';
import logger from './utils/logger.js';

dotenv.config();
const app = express();
const port = process.env.PORT || 3000;

// Connect to MongoDB
connectDB();

// Middleware
app.use(express.json()); // Body parser
app.use(cors()); // Enable CORS for cross-origin requests
app.use(requestLogger); // Logging requests
```

```
// Routes
app.use('/api/users', userRoutes);
app.use('/api/auth', authRoutes);

// Error Handler Middleware (must be last)
app.use(errorMiddleware);

// Root Route
app.get('/', (req, res) => {
   res.send('API is running...');
});

// Start Server
app.listen(port, '0.0.0.0', () => {
   logger.info(`🚀 Server running on port ${port}`);
});

import mongoose from 'mongoose';
import dotenv from 'dotenv';

// Load environment variables
dotenv.config();

const mongoURL = process.env.MONGO_URL;

const connectDB = async () => {
  try {
    await mongoose.connect(`${mongoURL}`);
    console.log('MongoDB connected');
  } catch (error) {
    console.error('Error connecting to MongoDB:', error);
    process.exit(1); // Stop the app if the database connection fails
```

```
  }
};

export default connectDB;

import { registerUser, loginUser } from '../services/authService.js';

// @desc    Register a new user
// @route   POST /api/auth/register
// @access  Public
export const register = async (req, res) => {
  try {
    const { username, email, password } = req.body;

    if (!username || !email || !password) {
      return res.status(400).json({ message: 'Будь ласка, заповніть усі поля' });
    }

    const data = await registerUser({ username, email, password });

    res.status(201).json({ message: 'Користувач успішно зареєстрований', ...data });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

// @desc    Login user
// @route   POST /api/auth/login
// @access  Public
export const login = async (req, res) => {
  try {
    const { username, password } = req.body;

    if (!username || !password) {
```

```
      return res.status(400).json({ message: 'Будь ласка, введіть ім\'я користувача та пароль' });
    }

    const data = await loginUser({ username, password });

    res.status(200).json({ message: 'Успішний вхід в систему', ...data });
  } catch (error) {
    res.status(401).json({ message: error.message });
  }
};

import { getUserById } from '../services/userService.js';

export const getUserProfile = async (req, res) => {
  try {
    const user = await getUserById(req.user.id);
    res.status(200).json(user);
  } catch (error) {
    res.status(404).json({ message: 'Користувача не знайдено' });
  }
};

// Count how many times each product was viewed
export function getViewCounts(viewedProducts: any) {
    const viewCounts: any = {};

    viewedProducts.forEach((product: any) => {
        if (!viewCounts[product.id]) {
            viewCounts[product.id] = { product, count: 0 };
        }
        viewCounts[product.id].count++;
    });

    return viewCounts;
```

```
}

// Build a “user profile” of attribute weights based on views and cart
export function generateUserProfile(viewedProducts: any, cartProducts: any) {
    const viewCounts = getViewCounts(viewedProducts);
    const attributeWeights: any = {};

    // Give weight 1–2 for each view of a product
    for (const { product, count } of Object.values(viewCounts) as { product: any; count: number }[]) {
        const weight = 1 + Math.min(count - 1, 1);
        addAttributes(attributeWeights, product, weight);
    }

    // Give weight 3 for any product that’s in the cart
    cartProducts.forEach((product: any) => {
        addAttributes(attributeWeights, product, 3);
    });

    return attributeWeights;
}

// Helpers: for each attribute, accumulate weight
export function addAttributes(
    attributeWeights: any,
    product: any,
    weight: number
) {
    const { title, color, type, price } = product;

    attributeWeights.title = attributeWeights.title || {};
    attributeWeights.color = attributeWeights.color || {};
    attributeWeights.type  = attributeWeights.type || {};
    attributeWeights.price = attributeWeights.price || [];
```

```
    attributeWeights.title[title] =
      (attributeWeights.title[title] || 0) + weight;
    attributeWeights.color[color] =
      (attributeWeights.color[color] || 0) + weight;
    attributeWeights.type[type] =
      (attributeWeights.type[type] || 0) + weight;

    // Strip non-digits, parse price, and push with its weight
    const numericPrice = parseFloat(
      price.replace(/[^\d.]/g, "").replace(/\s/g, "")
    );
    attributeWeights.price.push({ price: numericPrice, weight });
}

// Given all products and the user profile, score and sort by similarity
export function findSimilarProducts(allProducts: any[], userProfile: any) {
    return allProducts
        .map((product) => {
            let score = 0;

            // title, color, and type: add up matching weights
            if (userProfile.title?.[product.title]) {
                score += userProfile.title[product.title];
            }
            if (userProfile.color?.[product.color]) {
                score += userProfile.color[product.color];
            }
            if (userProfile.type?.[product.type]) {
                score += userProfile.type[product.type];
            }

            // price: compute weighted average price
            const priceData = userProfile.price || [];
            const totalWeight = priceData.reduce((sum: number, p: any) => sum + p.weight, 0) || 1;
```

```
        const avgPrice = priceData.reduce(
            (sum: number, p: any) => sum + p.price * p.weight,
            0
        ) / totalWeight;
        const priceDifference = Math.abs(product.price - avgPrice);
        if (priceDifference < 50) {
            score += 2;
        }

        return { ...product, similarityScore: score };
    })
    .filter(p => p.similarityScore > 0)
    .sort((a, b) => b.similarityScore - a.similarityScore);
}
```

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)

ГРАФІЧНА ЧАСТИНА
«РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ПІДБОРУ МУЗИЧНИХ ІНСТРУМЕНТІВ»

Виконала: студентка 4 курсу, групи 3КН-21б
Спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Джулай А.О.
(прізвище та ініціали)
Керівник: асистент каф. КН
Шолота В. В.
(прізвище та ініціали)
« 03 » червня 2025 р.

Вінниця ВНТУ – 2025

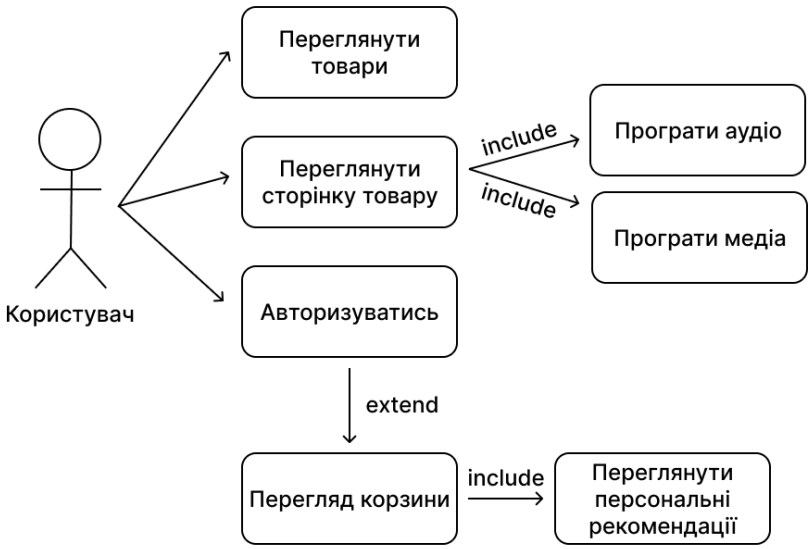

Рисунок В.1 – Діаграма прецедентів для підбору музичних інструментів

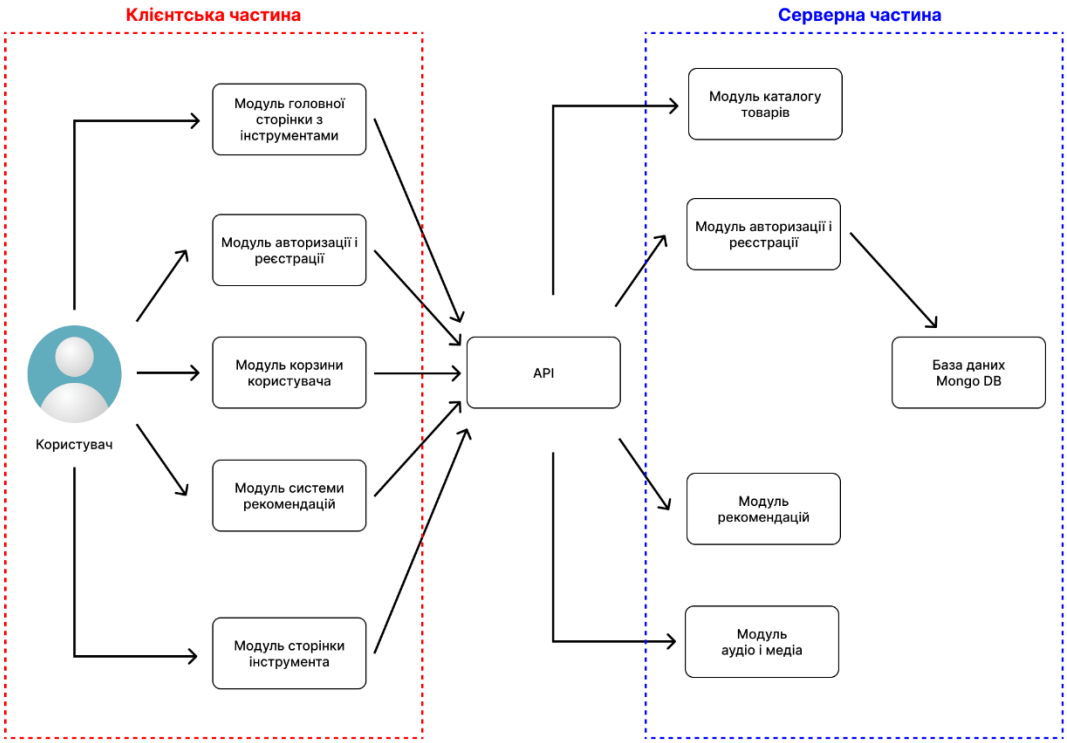

Рисунок В.2 – Загальна структурна схема компонентів програмного модуля підбору музичних інструментів

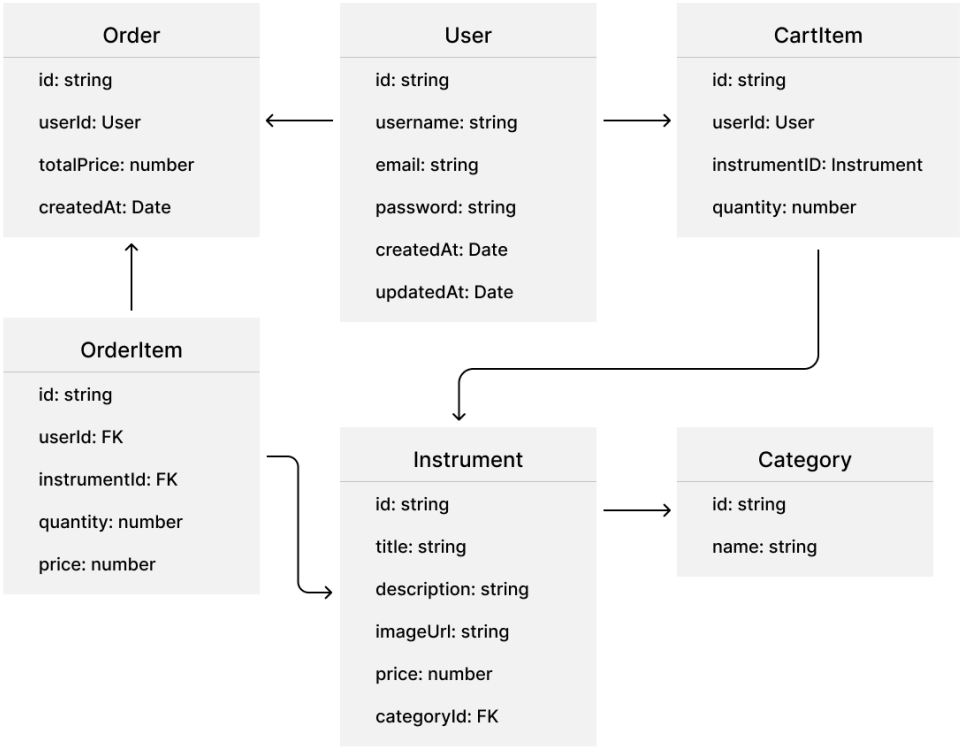

Рисунок В.3 - UML-діаграма класів клієнтської та серверної частини модуля підбору музичних інструментів

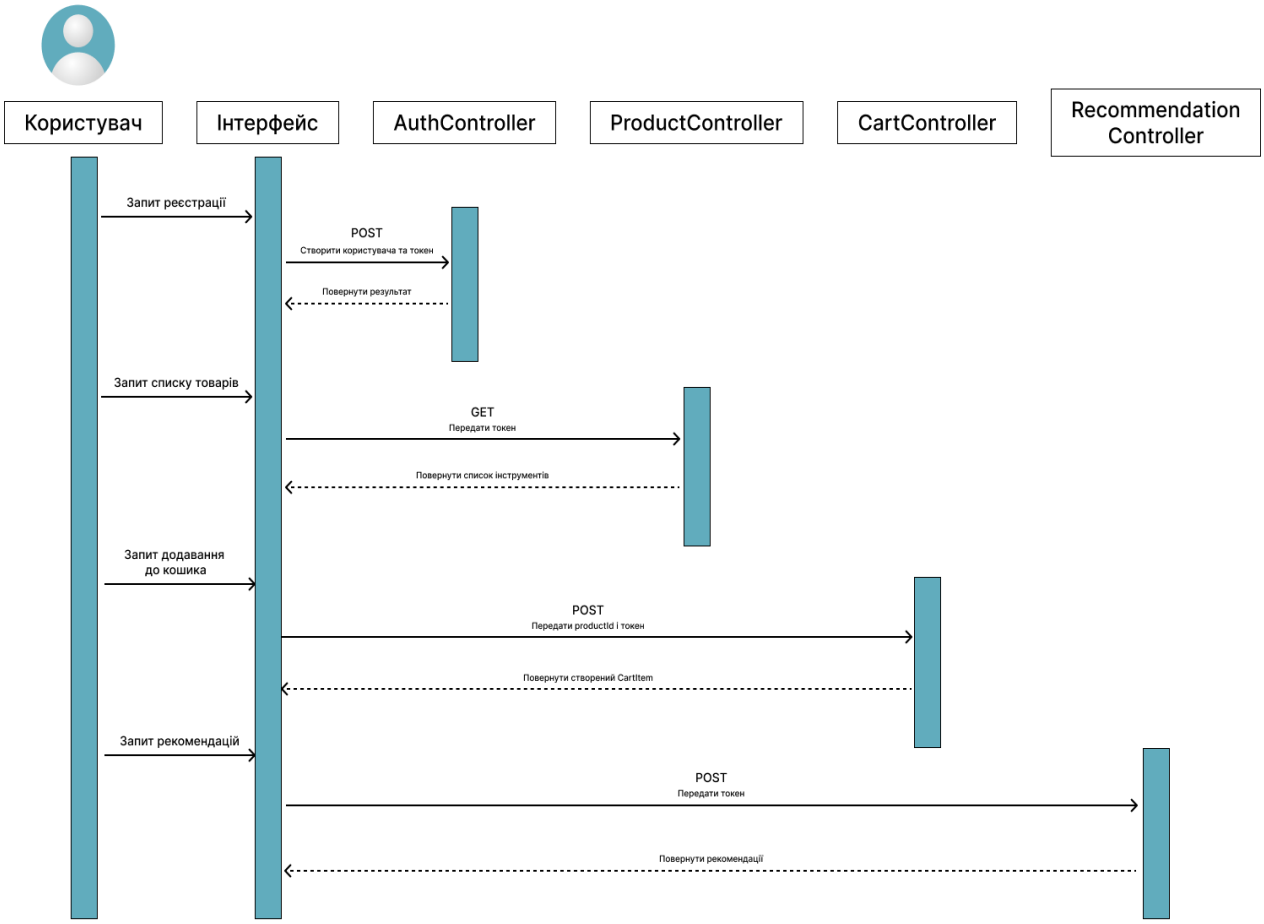

Рисунок В.4 - UML-діаграма послідовностей для підбору музичних інструментів

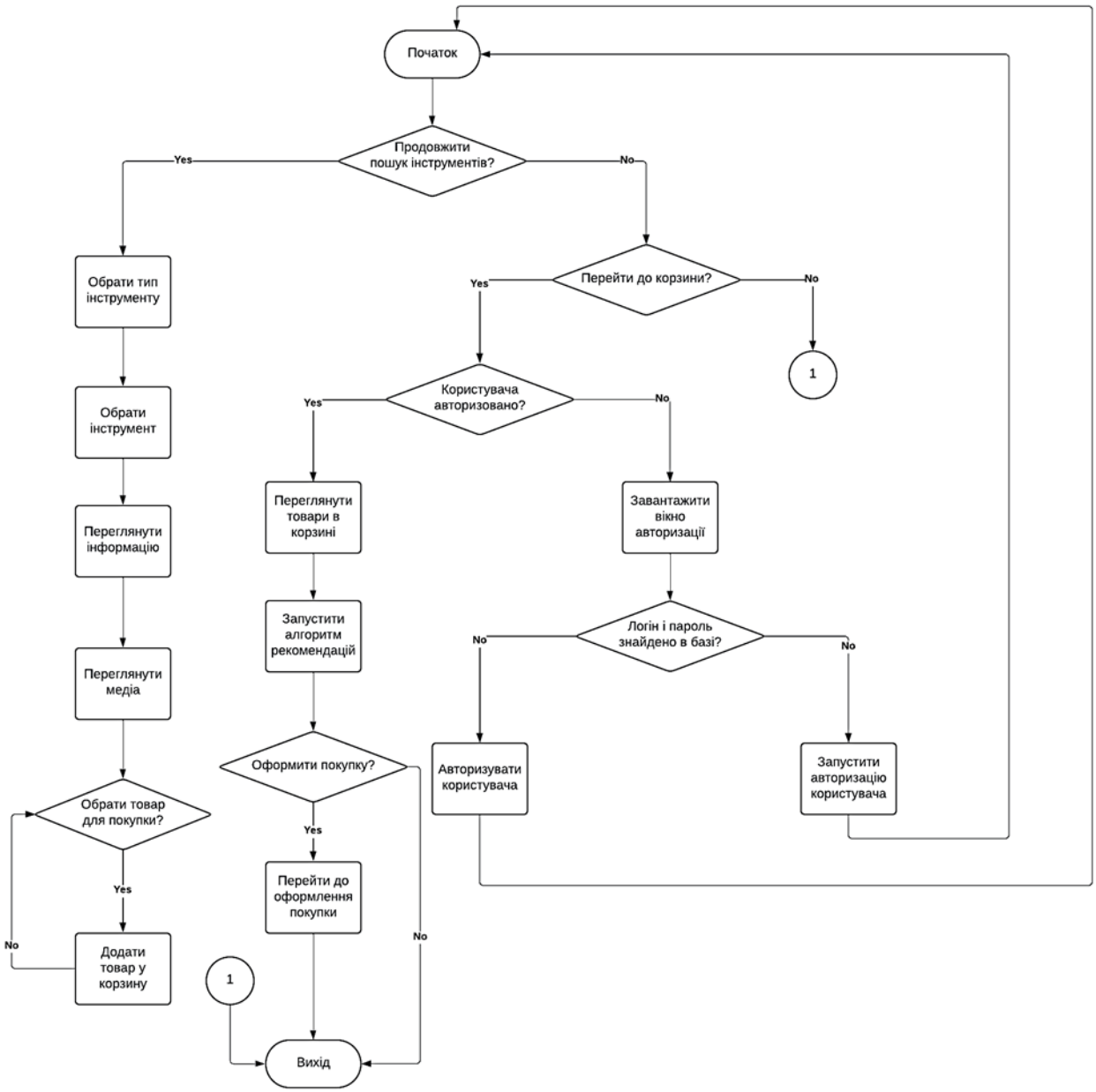

Рисунок В.5 – Блок-схема роботи веб-ресурсу для підбору музичних інструментів

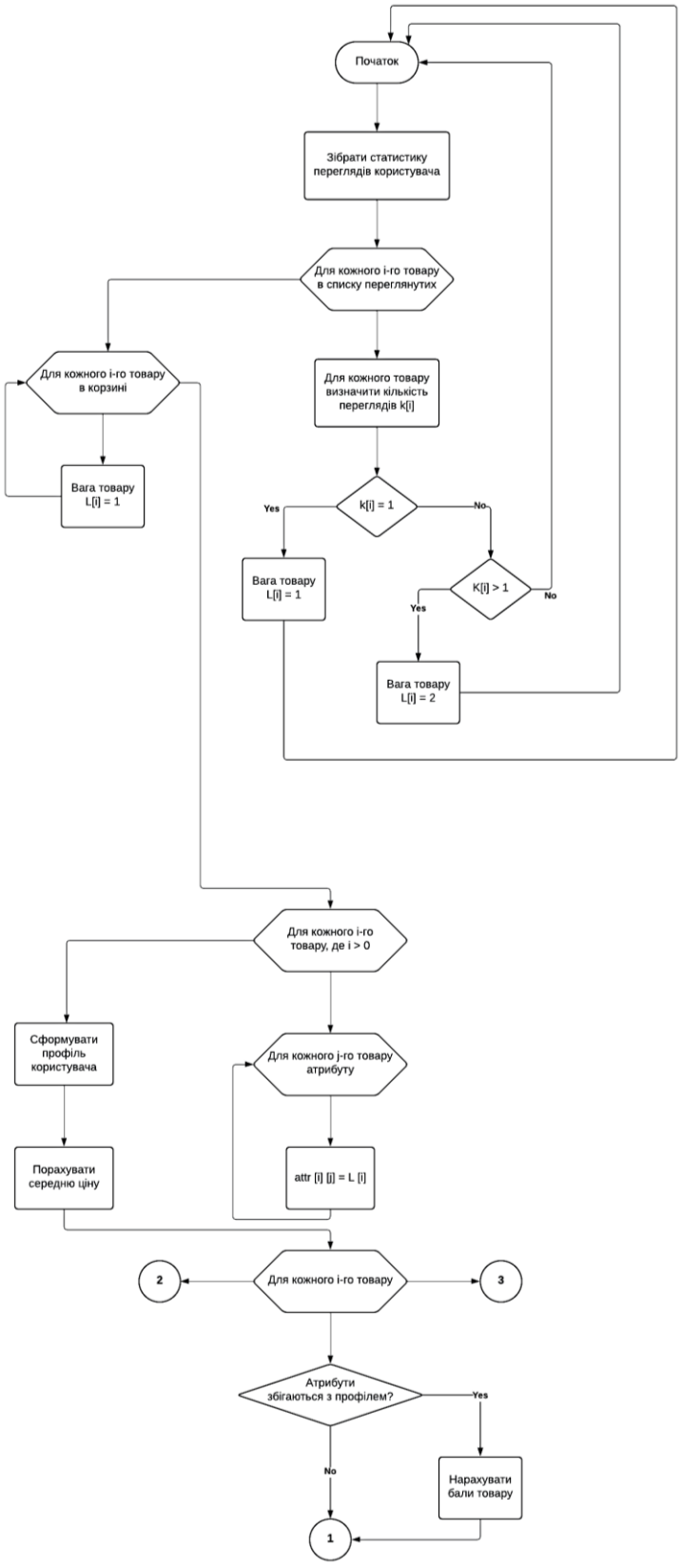

Рисунок В.6 – Блок-схема алгоритму системи рекомендацій веб-ресурсу для підбору музичних інструментів

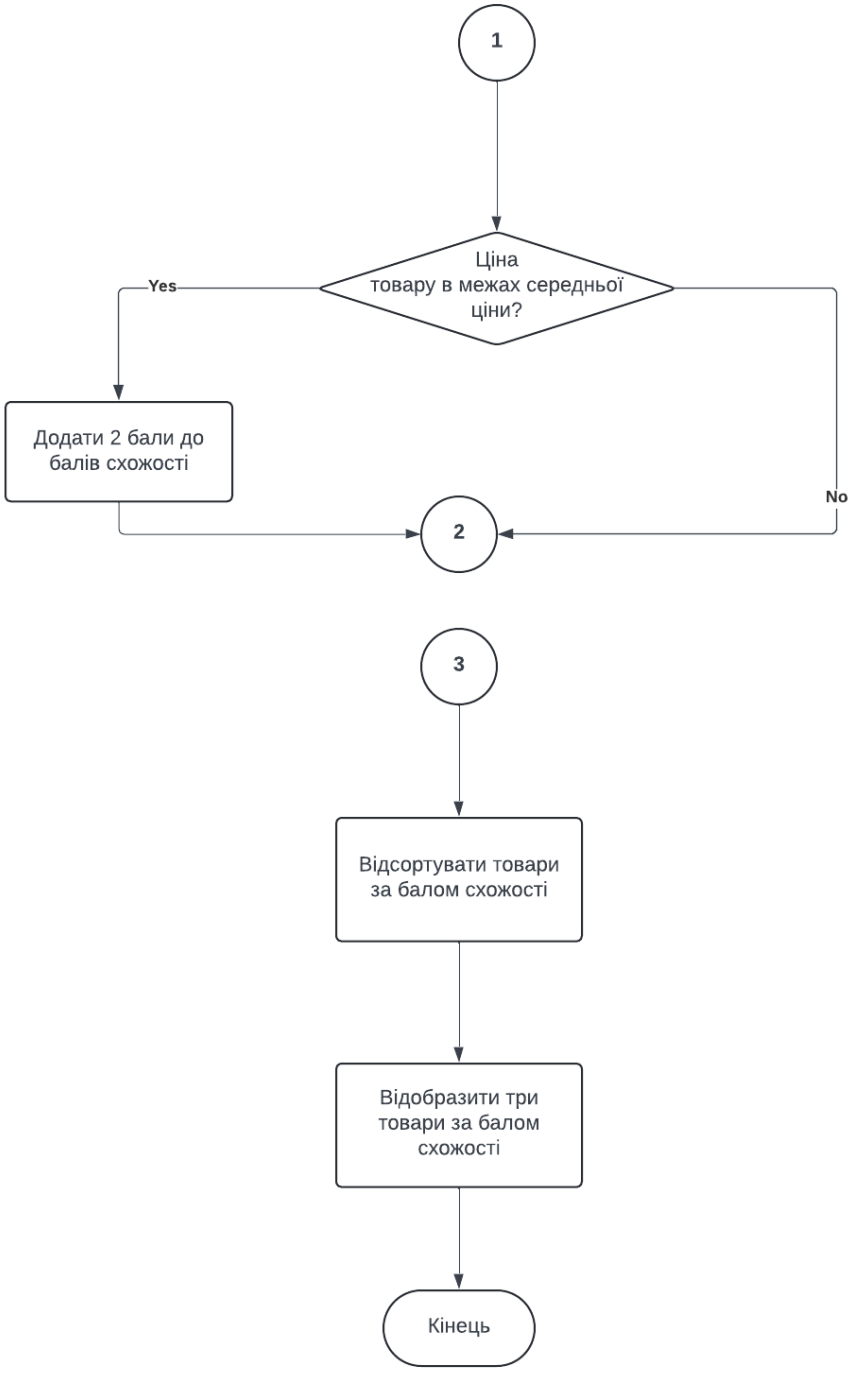

Рисунок В.6.1 – Продовження

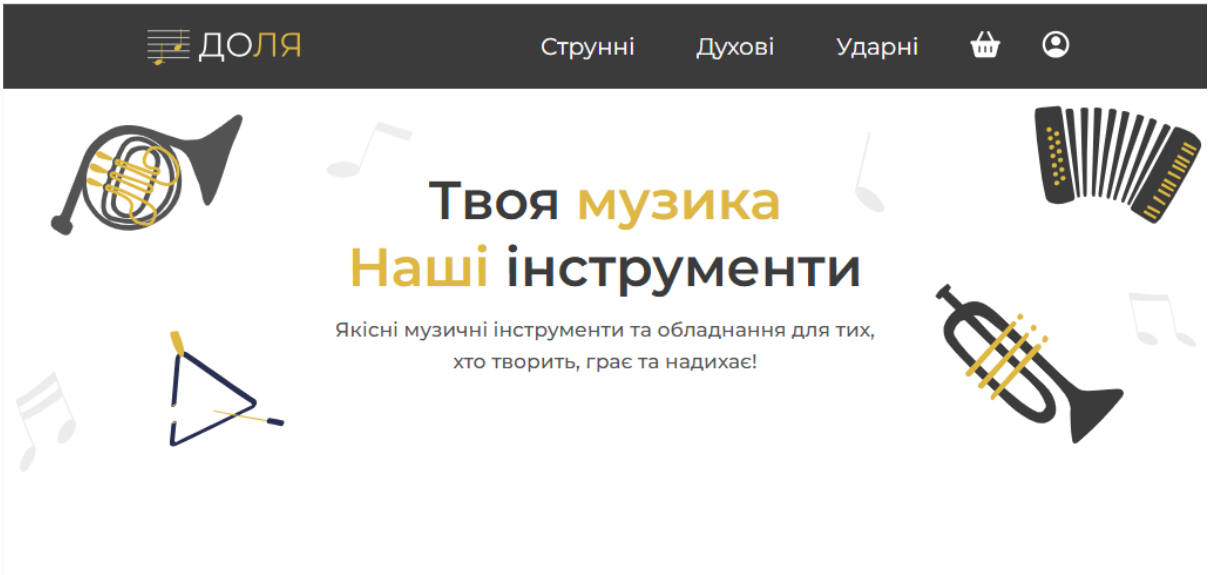

Рисунок В.7 – Головна сторінка WEB-ресурку для підбору музичних інструментів

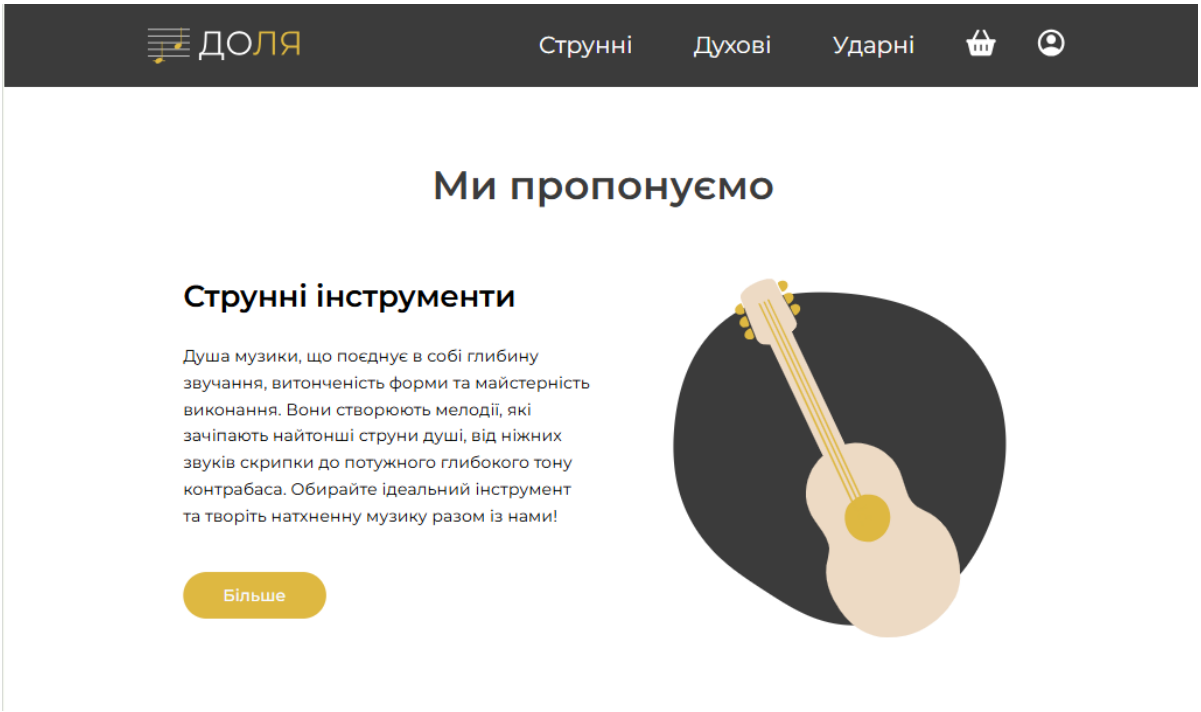

Рисунок В.8 - Головна сторінка WEB-ресурку для підбору музичних інструментів

Рисунок В.9– Сторінка струнних інструментів

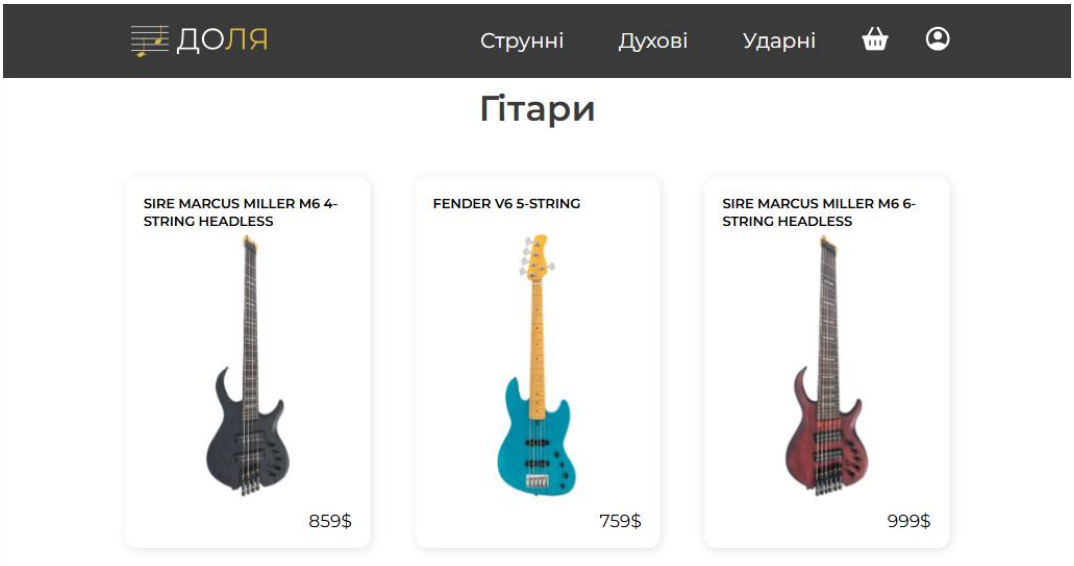

Рисунок В.10 - Сторінка струнних інструментів та товари

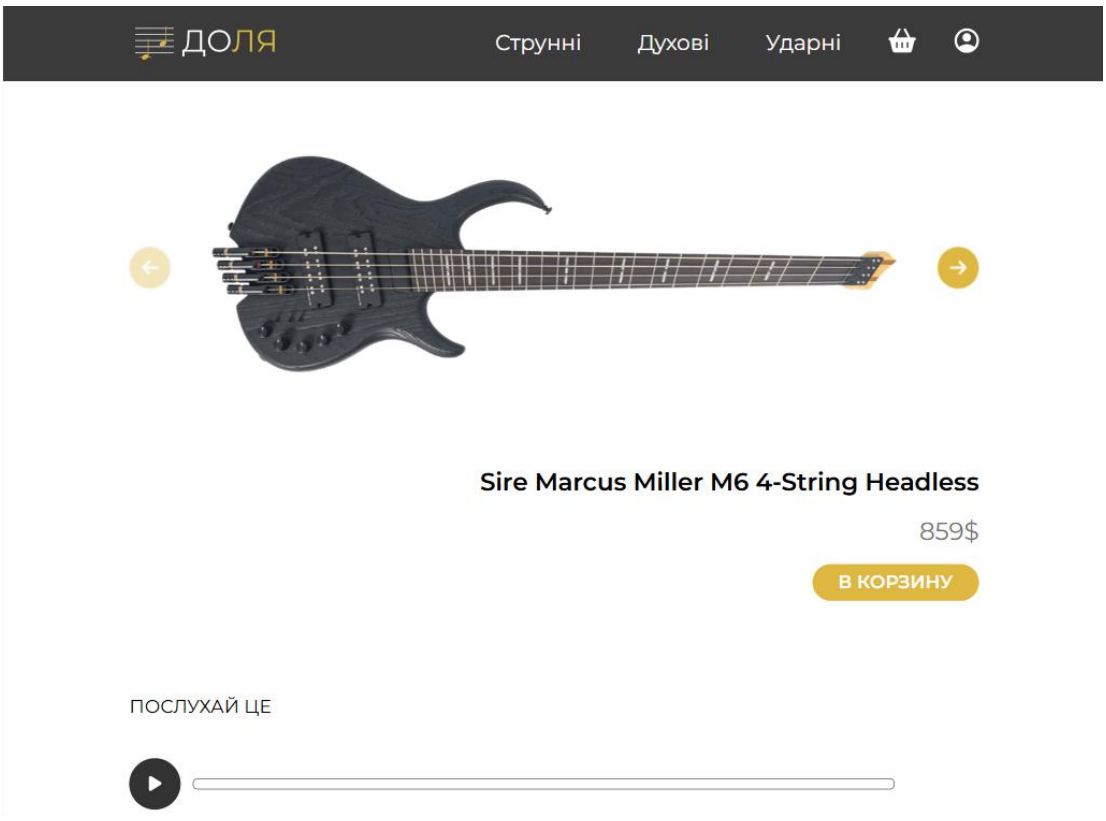

Рисунок В.11 – Сторінка інструмента

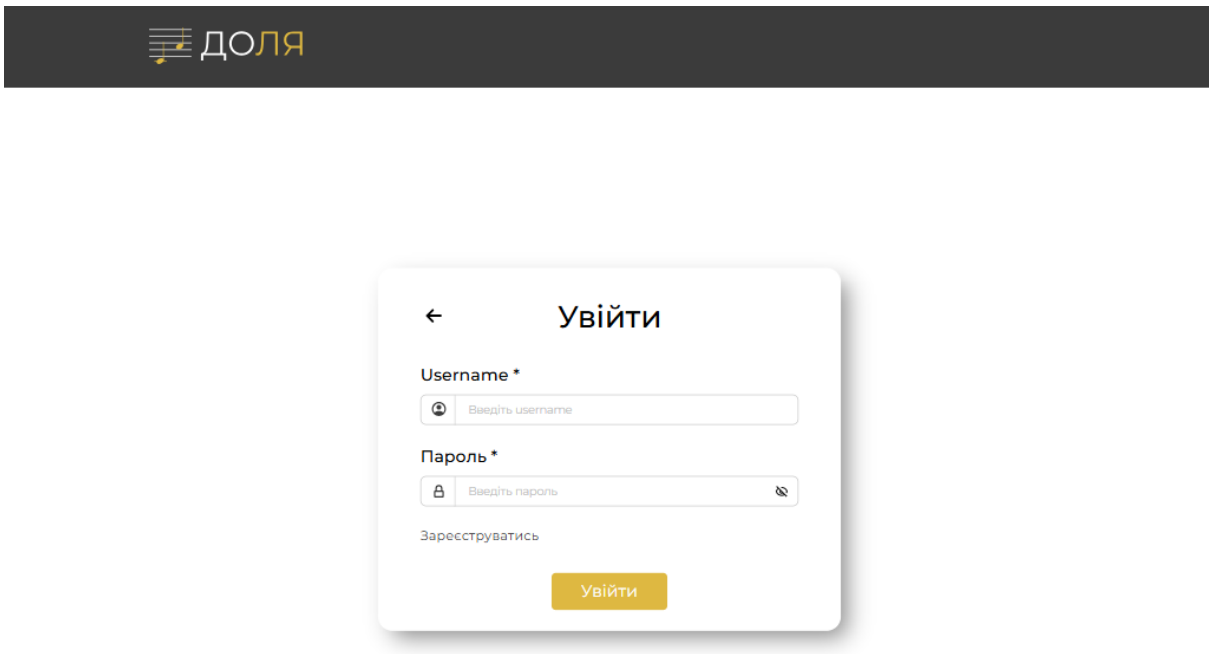

Рисунок В.12 – Вікно авторизації

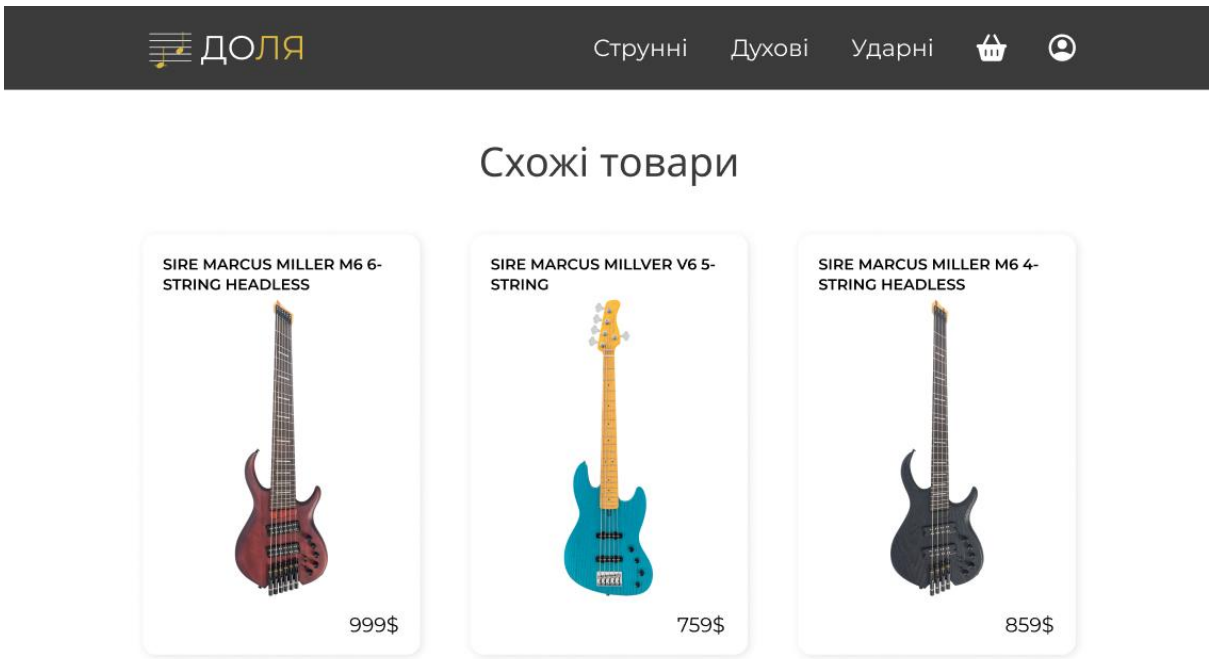

Рисунок В.13 – Результати системи рекомендацій за алгоритмом