Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Веб-застосунок для продажу
Назва теми
музичних інструментів

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190135.01.11.ПЗ

Виконав студент IV курсу, група ІПЗ-19-1 _____ М. С. Лембас
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ Г. І. Радельчук
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Г. І. Радельчук
Науковий ступінь, звання Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

5 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
02 01 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лембасу Максиму Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Веб-застосунок для продажу музичних інструментів

Керівник роботи Радельчук Галина Іванівна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5
2. Строк подання студентом роботи на кафедру 01.06.2023 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі,
Проектування веб-застосунку,
Програмна реалізація та тестування
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення у форматі А3:
UML-діаграма варіантів використання
Схема вікон інтерфейсу
UML-діаграма послідовності авторизації

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Радельчук Г. І., доцент	25.05.2023	01.06.2023
Антиплагіат	Гурман І. В., доцент	04.06.2023	04.06.2023

7. Дата видачі завдання «_02_» __січня_ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2023	
2 Проектування веб-застосунку. Розробка графічної частини	01.02 – 28.02 2023	
3 Програмна реалізація з використанням відповідних засобів розробки	01.03 – 10.04.2023	
4 Тестування програмного забезпечення	11.04 – 30.04.2023	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки, Оформлення графічної частини	01.05 – 25.05.2023	
6 Попередній захист кваліфікаційної роботи	травень 2023	
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2023	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2023	

Студент _____ М. С. Лембас
Підпис Ініціали, прізвищ

Керівник роботи _____ Г. І. Радельчук
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: Веб-застосунок для продажу музичних інструментів.

Автор: Лембас Максим Сергійович

Керівник: Радельчук Галина Іванівна

Обсяг – 67 с., 15 рис., 5 табл., 5 дод., 40 джерел.

Графічна частина: 3 креслення у форматі А3.

Мета кваліфікаційної роботи: розробка веб-застосунку для інтернет-торгівлі музичними інструментами.

У кваліфікаційній роботі проведено аналіз предметної області та сформульовано функціональні вимоги до застосунку. На основі отриманих даних було розроблено архітектуру застосунку та детальний проект. Для програмної реалізації ПЗ застосовані технології TypeScript та JavaScript, із використанням відповідних фреймворків та бібліотек.

Практичне значення одержаних результатів полягає у програмній реалізації компонентів Інтернет-платформи, яка надає можливість придбати музичні інструменти через Інтернет. Крім цього, потенційними власниками програмного забезпечення можуть бути підприємці, які бажають займатися дистрибуцією музичних інструментів через мережу Інтернет.

Результатом виконання кваліфікаційної роботи є розроблена архітектура веб-застосунка, окремі модулі ПЗ, які реалізовують функціонал інтернет-магазину для продажу музичних інструментів та пояснювальна записка з детальним описом процесу створення додатка та поясненням логіки роботи основних модулів.

01.06.2023
(дата)

(підпис)

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.190135.01.11.ПЗ	Пояснювальна записка	67		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічна частина			
4	А3	КвРІПЗ.190135.01.11.Е8	UML-діаграма варіантів	1		
			використання			
5	А3	КвРІПЗ.190135.01.11.Е8	Схема вікон інтерфейсу	1		
6	А3	КвРІПЗ.190135.01.11.Е8	UML-діаграма	1		
			послідовності			
			авторизації			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190135.01.11.ВД		
Виконав		Лембас М. С.		01.06	Веб-застосунок для продажу музичних інструментів Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Ралельчук Г. І.		01.06			
Рецензент						ХНУ, ІПЗ-19-1	
Н. контр.		Радельчук Г. І.		01.06			
Зав. каф.		Бедрацюк Л. П.		05.06			

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190135.01.11.ПЗ		
Виконав		Лембас М. С.		01.06	Веб-застосунок для продажу музичних інструментів Пояснювальна записка	Літ.	Арк. 4 / Аркушів 67
Керівник		Ралельчук Г. І.		01.06			
Рецензент						ХНУ, ІПЗ-19-1	
Н. контр.		Радельчук Г. І.		01.06			
Зав. каф.		Бедрацюк Л. П.		05.06			

ВСТУП

Станом на 2023 рік сфера веб-застосунків розвивається з високим рівнем активності, хоча і переживає певний спад після стрімкого росту під час пандемії. Такі типи веб-застосунків, як застосунки для продажу товарів та застосунки для масштабування бізнесу наразі є доволі популярним способом ведення бізнесу, відповідно розробку таких застосунків замовляють доволі часто.

Наразі, при веденні бізнесу традиційним шляхом – відкриваючи один торговий заклад, або мережу закладів, власники бізнесу наштовхуються на ряд факторів, котрі негативно впливають на співвідношення їхніх прибутків та витрат. Серед них: потреба оренди вигідної для торгівлі території, що може потребувати значних фінансових вливань; потреба найму та оплати праці робітників, котрі працюватимуть з клієнтами; обмеженість клієнтської бази лише локальними резидентами.

Інернет-застосунки, або просто сайти для продажу товарів, вирішують цілий ряд проблем для власників бізнесу, як малого, так і великого. Вони дозволяють знаходити цільову категорію покупців за допомогою реклами, масштабувати свій бізнес, дозволяють зменшити витрати на оренду торгових залів. Також це нівелює потребу клієнта відвідувати магазин, який може знаходитися у іншому місті або країні – він може отримати всю інформацію використовуючи веб-застосунок, та навіть придбати та оплатити товар – ці переваги гарантують зацікавленість власників бізнесу у купівлі послуг спеціалістів, котрі можуть створити якісну віртуальну копію їхнього магазин, що дозволить отримувати більші прибутки – у цьому і полягає актуальність тематики кваліфікаційної роботи.

Отже, розробка програмного засобу дозволить приватним або державним компаніям продавати товари у значно більших обсягах, та мінімізувати витрати на обслуговування клієнтів, що дозволить збільшити прибутки компанії.

Метою проекту є розробка програмного забезпечення, яке дозволить власниками малого та середнього бізнесу вивести об’єм торгових пропозицій на

новий рівень та збільшити рівень обсягу продажу товарів. Фінальна версія програмного продукту має виконувати функції онлайн-магазину та використовуватися з ціллю продажу товарів та комунікації між власниками бізнесу та клієнтами.

Для досягнення мети поставлено такі завдання:

– провести загальне комплексне ознайомлення зі сферою інтернет-торгівлі та веб-застосунками для продажу товарів через мережу Інтернет;

– визначити особливості предметної області;

– провести аналіз технологій, методологій, практик розробки веб-застосунків та вибір оптимальних для подальших використання;

– провести ознайомлення з сучасними архітектурними та функціональними рішеннями, вибрати ті, які будуть використані у проекті;

– розробити структуру та функціональну модель проекту;

– реалізувати проект програмним шляхом;

– провести тестування веб-застосунку;

– пілотувати документацію, що описує процес виконання проекту.

Потенційними користувачами розроблюваного програмного забезпечення є власники бізнесу, котрі займаються продажем товарів через мережі магазинів, або ті, хто хоче почати продавати товари використовуючи мережу Інтернет.

Корисність розроблюваного програмного продукту для компаній, що займаються продажем товарів полягає у можливостях збільшити базу покупців та об’єми продажів шляхом прямого використання ПЗ.

Розробка програмного забезпечення та написання супровідної документації є визначним етапом при отриманні освітнього ступеня бакалавра, робота дозволяє студентові продемонструвати та підтвердити набуті комплекси знань в сфері інженерії програмного забезпечення.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Опис предметної області кваліфікаційної роботи складається з демонстративного опису декількох важливих процесів: аналізу поточної ситуації на ринку, аналізу організаційної ситуації у сфері предметної області та аналізу технічного та програмного забезпечення, яке наразі якісно використовується в сфері предметної області. Для вдалого початку розробки програмного продукту необхідно визначити слабкі місця індустрії та звернути особливу увагу на вирішення існуючих проблем галузі. Це відбувається шляхом розробки архітектури проекту, пошуку особливостей, які необхідно враховувати при проектуванні, та розробки логіки роботи додатку шляхом опису сторінок веб-застосунку, бази даних, механізмів для взаємодії клієнта з сервером. Користувачем системи буде клієнт, котрий використовує браузер для зв’язку з базою даних та адміністрацією магазину. Усі процеси будуть детально описані у 2 та 3 розділах.

Предметну область системи описано нижче. Зазвичай користувачів не хвилює, як саме розроблена платформа, але вони дбають про безпеку власних даних на обраній платформі. Інтернет-магазин дозволяє отримати доступ до товарів, але при покупці користувач залишає свої платіжні дані, тому безпека системи є надзвичайно важливим моментом. Завдяки зручності онлайн-покупок користувачі зможуть порівнювати ціни на ідентичні товари в кількох різних магазинах. Яким би захоплюючим це не було, робити покупки лише за допомогою смартфона надзвичайно зручно.

Сьогодні користувачам може подобатися багато різних веб-сайтів для продажу товарів, і всі вони мають певну схожість архітектурних та загальних рішень. Незважаючи на те, що веб-сайти створюються з однією метою і вони мають багато спільного як в архітектурних рішеннях, так і в шаблонах, технологіях та методологіях, веб-застосунки часто використовують різні

принципи функціонування, які можуть найкращим чином забезпечити потрібну саме їм функціональність.

Тепер розглянемо типи сайтів для продажу товарів через мережу інтернет та їх архітектурні рішення. Зокрема, особливості функціонування веб-магазинів, котрі займаються саме продажем музичних інструментів.

Найпопулярнішими сайтами з продажу товарів в Інтернеті є інтернет-магазини. Цей тип платформи являє собою віртуальні копії товарів, щоб користувачі, а саме клієнти, могли ознайомитись і замовити певний товар. Зараз у кожної великої компанії є такий сайт - це зрозуміло, адже це так зручно. Що стосується архітектури веб-сайтів такого типу, зрозуміло, що нам потрібно постійно обмінюватися даними з сервером. Також варто враховувати що для покращення користувацького досвіду таким веб-застосункам необхідно забезпечити доволі швидкий відгук інтерфейсу на дії користувача – показати нову сторінку, перейти на сторінку певного товару або загрузити деталізоване фото.

Другим за поширеністю видом сайтів для продажу є сайти-візитки або, як їх неофіційно називають, лендінги. Лендінг, запозичене з англійської мови слово, корпоративний рекламний сайт. Як правило, ці веб-сайти містять інформацію про послуги, які надає компанія, або продукти, які вона продає. Відмінність цього типу від попереднього полягає в тому, що сторінки з товарами надають лише інформацію про товари, не дозволяючи їх придбати. Типовою практикою є наявність форм зворотного зв'язку на цільових сторінках. Завдяки цьому клієнт зможе надіслати повідомлення власнику сайту або, як останнім часом стало практикою, зв’язатися з ним у месенджері.

Третій тип веб-сайтів - це онлайн-каталоги. Сайти такого типу підбирають товари відразу в декількох інтернет-магазинах і дозволяють користувачам порівнювати їх, звертаючи увагу на ціну, відгуки, характеристики товарів. Хоча ці сайти не є магазинами, вони допомагають систематизувати товари інших інтернет-магазинів, дозволяючи користувачам раціонально використовувати свій час і переглядати лише найвигідніші пропозиції.

Різниця між сайтами для продажу різних товарів та музичних інструментів.

І хоча усі вищеописані типи сайтів потенційно можуть використовуватися для продажу музичних інструментів, існує ряд особливостей, які використовуються для розробки оптимальних рішень для онлайн магазинів, що продають музичні інструменти. Ось ряд особливостей:

– необхідність гарантування безпеки даних користувачів та їхніх платіжних даних, а також наявність систем, що убезпечують передання даних користувачів стороннім особам;

– необхідність наявності документації щодо усіх товарів, щоб клієнт міг бути впевненим у якості та оригінальності інструменту або різноманітних комплектуючих, для цього потрібно створити систему, яка буде демонструвати документацію;

– необхідність наявності системи, яка дозволяє користувачам отримувати придбані ними товари без безпосереднього відвідування магазину, тобто системи доставки, яку пропонує магазин.

Звичайно, розробка трьох типів веб-сайтів має багато спільного. Така ж ситуація і з логікою та функціональністю описаних сайтів. Але кожен з них має важливі відмінності. Зокрема, для онлайн-магазинів потрібно детально продумати архітектурні конструкції, оскільки на сторінках сайту буде багато складних дій і більшість даних загружаються з серверу. Як і в інтернет-магазинах і каталогах, архітектурні рішення тут вимагають глибоких знань як архітектурних принципів, так і самої програми. Оскільки онлайн-магазини часто обробляють величезні обсяги інформації та вимагають високошвидкісних алгоритмів для обробки даних, правильно підібрані архітектурні рішення та рішення для моделювання повинні забезпечити успішну роботу модулів та системи в цілому.

Попередня розробка технічного завдання та архітектури сайту дозволить уникнути неприємних ситуацій, коли під час розробки чи тестування виявляються недоліки, які не були враховані та для обробки яких не були обрані рішення. У гіршому випадку вирішення подібних завдань вимагає зміни логіки всієї програми. Тому, обираючи технології слід передбачити кілька орієнтирів.

Наприклад, у 2015 році комп’ютери були основним джерелом трафіку для більшості веб-сайтів. Значна частина сьогоднішнього трафіку надходить зі смартфонів. Тому необхідно зробити так, щоб сайт міг задовольнити потреби як користувачів смартфонів, так і користувачів ПК. Комп’ютери та смартфони мають кілька доступних операційних систем і розмірів екрана. І важливо сприяти правильному відображенню даних сайту на всіх можливих пристроях.

Для першого етапу розробки програмного забезпечення будуть використані різні додаткові інструменти - створення чорнових макетів сторінок у Figma, створення діаграм за допомогою UML, який є незамінним інструментом для розгалуженого логічного проектування системи. Використання UML допоможе створити візуальні моделі роботи системи, що збільшить і спростить розуміння принципів роботи всього проекту.

Також для візуалізації та формалізації основних процесів, які потребують використання кількох потоків даних, використано методологію функціонального моделювання та графічного опису процесів IDEF-0. Використання методології дозволяє створювати контекстні діаграми комплексних процесів. Основною перевагою використання діаграм є спрощення розуміння ієрархічного представлення об’єктів шляхом чіткого визначення зав’язків між різними складовими системи та додатковими вхідними даними.

Для створення діаграм та візуальних моделей використано безкоштовне середовище для створення діаграм, дизайну та моделей - https://creately.com/home/ .

На рисунку 1.1 показано контекстну діаграму роботи онлайн-магазину. Потоки демонструють джерела надходження даних та основну вихідну інформацію. Вхідними даними є інформація від користувачів, їх дані для реєстрації, запити до системи для отримання даних про пені товари або для отримання загальної інформації.

Також в роботі системи бере безпосередню участь адміністрація, це необхідно для контролю розміщених на платформі даних, спілкування з клієнтами, обробці замовлень. Вихідними даними також є замовлення на покупку

товарів, які залишають користувачі – саме можливість реалізації цього процесу і є причиною розробки ПЗ. Також вихідними даними є певні набори даних, котрі необхідні для створення замовлення та оптимального досвіду користувача при використанні системи.

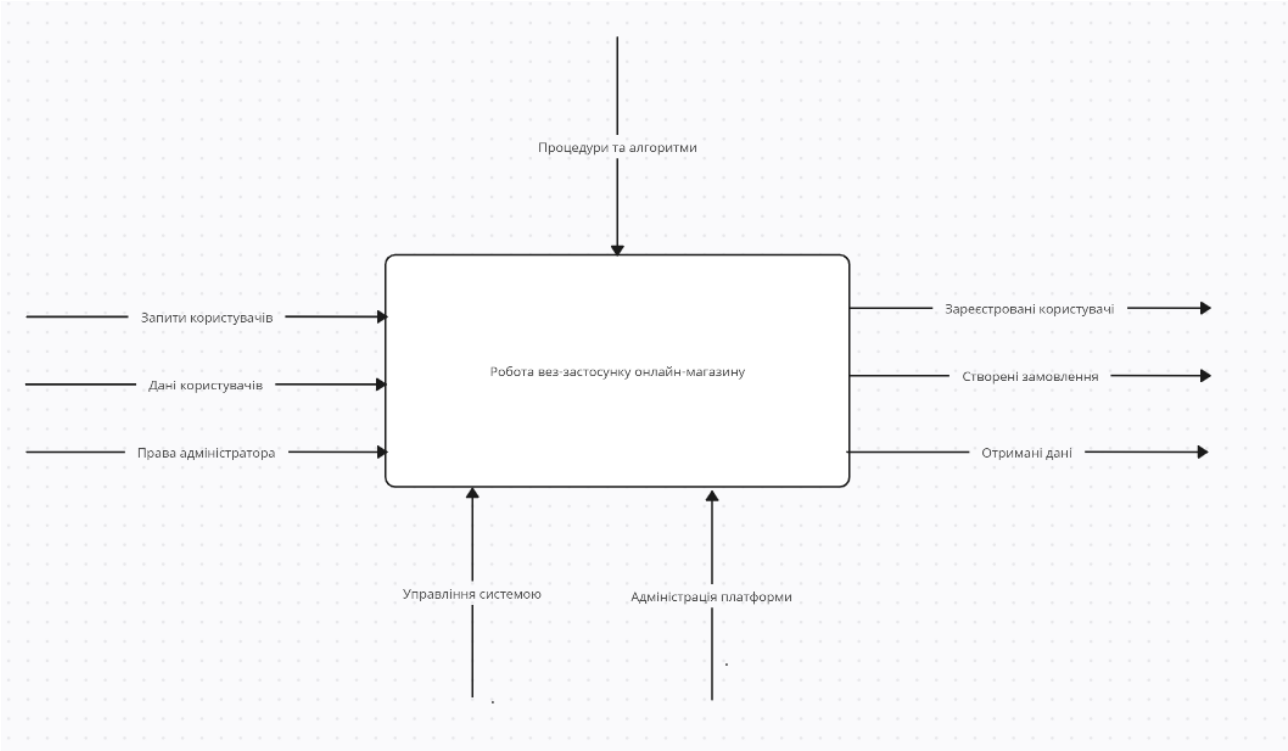


Рисунок 1.1 – Контекстна IDEF0-діаграма роботи програмного забезпечення

Для візуального зображення процесів дій системи та їх послідовності використано діаграму декомпозиції першого рівня. Це дозволяє отримати комплексне бачення послідовності головних сценаріїв використання сайту. За потреби, для покращення розуміння окремих складних процесів може бути створено діаграму декомпозиції другого рівня. Діаграму декомпозиції веб-застосунку першого рівня зображено на рисунку 1.2.

Н діаграмі відображено доступні для користувача функції – як для незареєстрованого користувача платформи (гостя), так і для користувача, який пройшов реєстрації і після цього має можливість зробити замовлення. А також продемонстровано, які додаткові джерела даних, які беруть участь у кожній фазі декомпозиції: інтерфейс сайту, дані користувача, дані про товар.

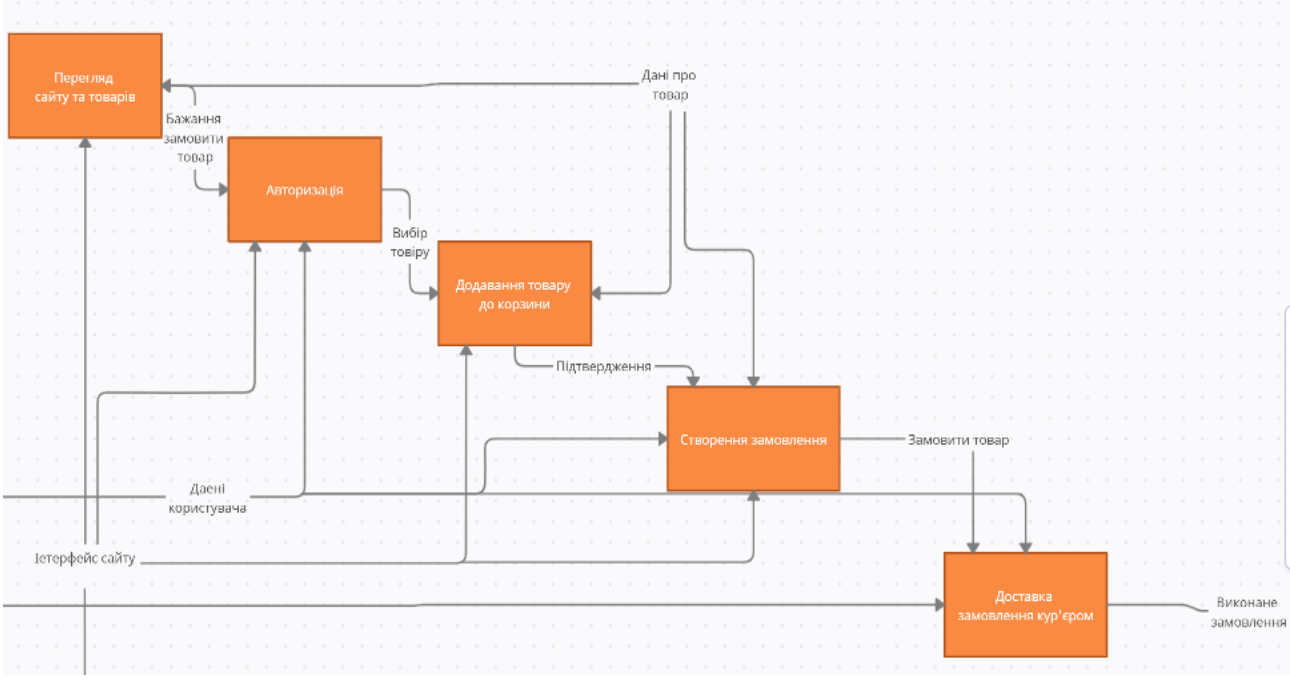


Рисунок 1.2 – IDEF0-діаграма декомпозиції першого рівня

У розробці даного програмного продукту відсутній етап спілкування із замовником, тому він починається з розробки технічного завдання. Це дозволяє уникнути несподіваних проблем під час розробки програми. Після формування технічного завдання необхідно створити прототип інтерфейсу користувача. Після розробки дизайну програмістам необхідно реалізувати функціональність сайту. Також фінальне комплексне тестування є невід'ємною частиною створення програмного продукту.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Нижче буде наведено огляд найбільш відомих інтернет-магазинів та подано характеристику їх архітектури та шаблонів. Всю інформацію можна знайти у вільному доступі. Звісно при огляді рішень враховано чи займаються магазини продажем музичних інструментів. Аналіз даних такого роду дозволить не «створювати велосипед», а взяти найкращі існуючі рішення високого рівня.

Аналіз популярних рішень розпочато з гігантів індустрії онлайн-продажів: Ebay.com та Amazon.com (рисунки 1.3 та 1.4 відповідно).

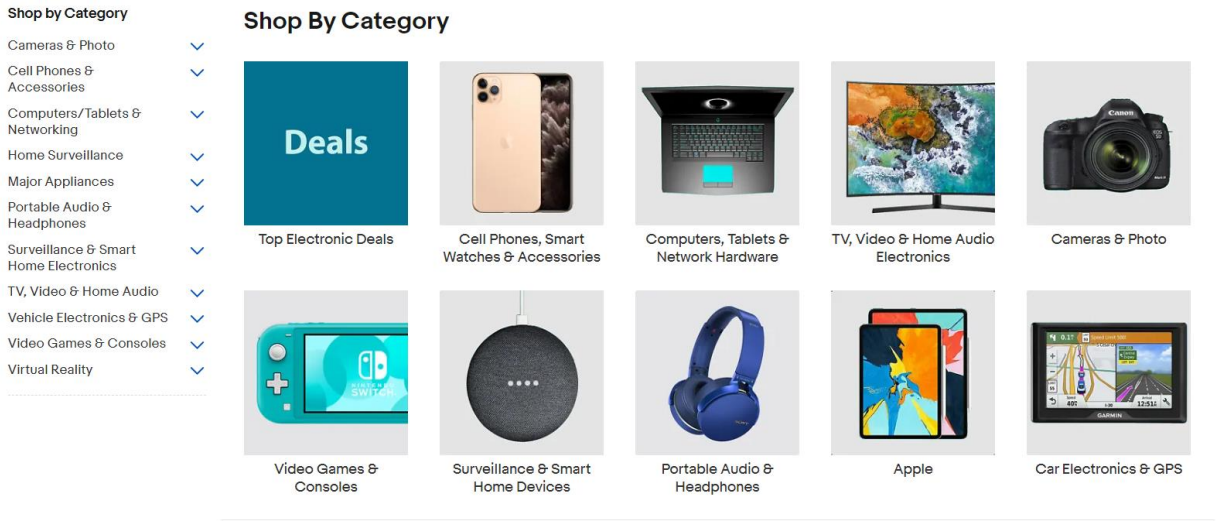


Рисунок 1.3 – Інтерфейс сайту Ebay.com [1]

Ми відразу зрозуміли, що сайт Ebay.com використовує всі можливості інтерфейсу: вони дозволяють користувачеві сприймати більше пропозицій одночасно, і в той же час динамічні елементи привертають увагу користувача.

На сайті є велика кількість елементів, які надають додаткову інформацію про товар або те, що може зацікавити клієнта. Зокрема, до них відносяться Навбари та тайпбари. Вони прискорюють пошук необхідного товару і дозволяють швидко отримати доступ до заданої категорії або країни виробництва.

Наступною платформою для аналізу є Amazon.com. Платформа надзвичайно популярна в США. На сайті є можливість придбати широкий асортимент товарів, включаючи музичні інструменти та обладнання до них. Було досліджено як загальний сектор даної платформи, так і сферу продажу музичних інструментів, котра тут реалізована особливим чином і має доволі зручні інтеграції з офіційними виробниками музичних товарів та обладнання.

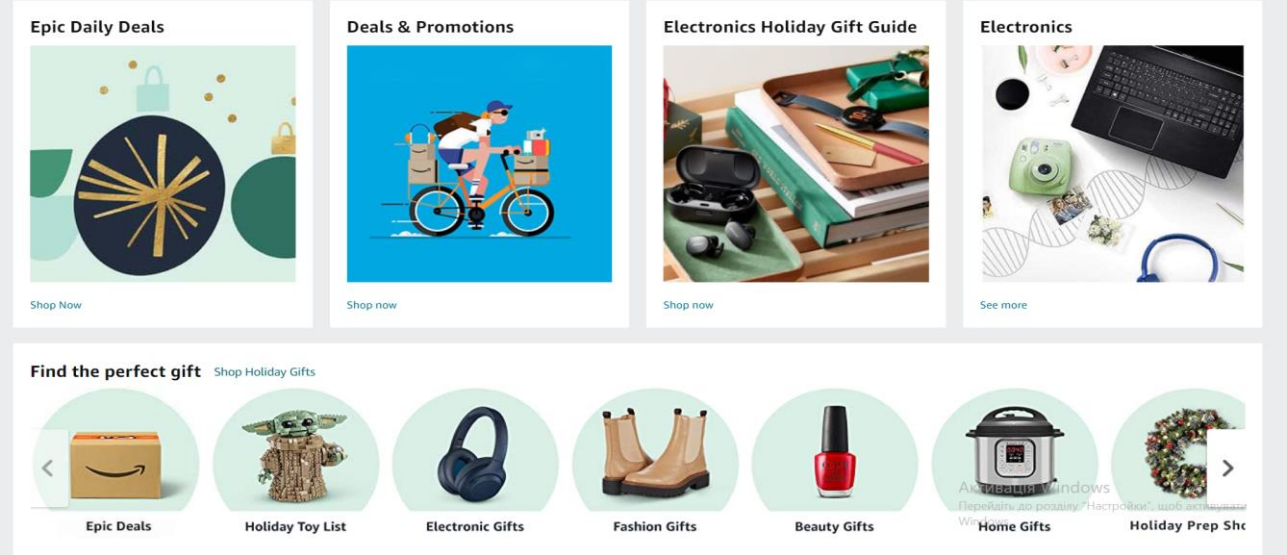


Рисунок 1.4 – Інтерфейс сайту Amazon.com [2]

Сайт досить функціональний, але можна відзначити деякі недоліки: відсутність меню для швидкої навігації та додаткових опцій. Користувачам надаються лише найпопулярніші категорії товарів. І тому абсолютній більшості користувачів варто скористатися вікном пошуку товару. А це часто важко для недосвідчених користувачів.

Однією з очевидних переваг рейтингових магазинів є сучасний і зрозумілий інтерфейс. Це приваблює користувачів. Також можна оцінювати продукти на сайті, допомагаючи іншим користувачам орієнтуватися в пропонованих продуктах.

Оскільки комп'ютерна індустрія в Україні дуже розвинена, ми провели аналіз українських інтернет-магазинів. Виявлено, що веб-сайти працюють добре, але інтерфейс часто є низької якості або застарілим. Таким чином, можна зробити висновок, що в проектах з меншим бюджетом дизайн часто не враховується, і це проблема, тому що клієнт, швидше за все, вибере веб-сайт із сучасним дизайном, що створює довіру власників сайтів.

І третім проаналізованим веб-застосунком є онлайн-магазин «gear4music.com», рисунок 1.5. Дана платформа займається виключно продажем музичних інструментів, обладнання та комплектуючих, що дозволяє скласти враження про еталонний вигляд сайтів для продажу товарів музичної індустрії.

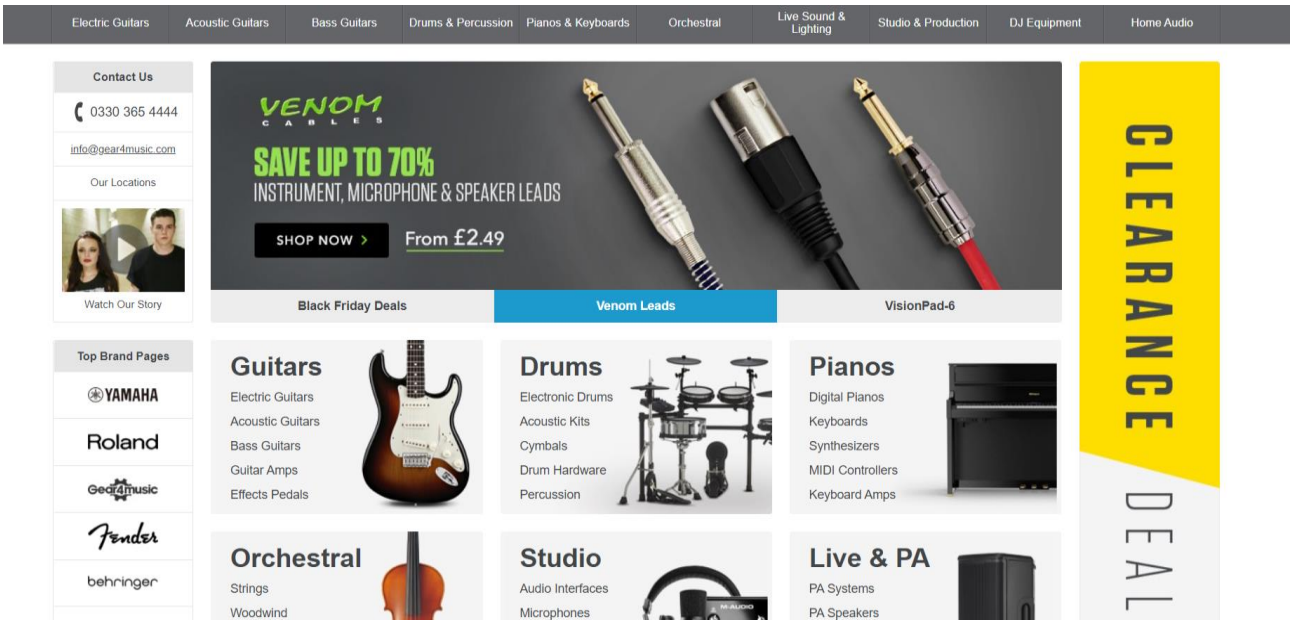


Рисунок 1.5 – Інтерфейс інтернет-магазину gear4music.com [3]

Що ж архітектурних рішень оглянутих сайтів: зрозуміло, що широко використовується клієнт-серверна архітектура адже відбувається постійна відправка запитів на сервер, їх обробка та повернення результатів. Суть клієнт-серверної архітектури полягає в поділі застосунка на дві функціональні одиниці: клієнт та сервер. І функціонування додатка відбувається при обмін даними між цими двома складовими.

Також очевидно, що розробляти такі об’ємні інформаційні системи незручно та важко без використання шаблонів, які б допомагали впорядковувати та оптимізувати код. З відкритих джерел можна дізнатися, що більшість алгоритмів, що описують логіку функціонування систем, а також, алгоритми вибору товарів для відображення та створення рекомендацій для користувачів написано на Python та Java. В свою чергу, користувацька частина сайту написана на JavaScript, а саме на широковідомому фреймворку React, з використанням додаткових бібліотек та засобів.

Для впорядкування даних, отриманих при дослідження наявних рішень зручно використовувати таблиці. Вони дозволять швидко порівняти платформи за списком важливих функцій та можливостей. Для цього було створено порівняльну таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика досліджених платформ

Платформа	Ebay.com	Amazon.com	gear4music.com
Перегляд товарів до реєстрації	Так	Так	Так
Збереження товарів до улюбленого	Так	Так	Ні
Зручність інтерфейсу та базових функцій	Інтерфейс є інтуїтивним, але не пропонує багато товарів при першому відвідуванні платформи	Інтерфейс є мінімалістичним і має відмінний деталізований поділ на категорії та поділ за усіма можливими показниками товарів	Інтерфейс є інтуїтивним, але помітні затримки при переході на сторінку товару / оплати; є багатофункціональні фільтри для пошуку оптимального товару
Швидкий зв’язок з адміністратором	Ні	Так	Так
Кросплатформність	Так	Так	Так
Зручність реєстрації та функціоналу користувача	Використовує пошту або соціальні мережі для реєстрації, пропонує користувачу кошик, історію, потенційно цікаві товари	Використовує пошту або соціальні мережі для реєстрації та має базовий функціонал користувача	Дозволяє реєструватися за номером телефону, але процес перевірки займає певний час

1.3 Визначення функціональних вимог до веб-застосунку

Варто зауважити, що детальний опис функціональності ПЗ подано у технічному завданні, а у підрозділі 1.3 подано загальну характеристику системи для розуміння її можливостей, проблем, що вона вирішує, шляхів використання для усіх потенційних користувачів.

Розроблюване програмне забезпечення повинне виконувати наступні функції: надавати можливість користувачу переглядати товари на сайті та забезпечувати безпеку даних користувача, зокрема платіжних, після його реєстрації або входу в систему. Другорядними але не менш важливими з точки зору функціонування сервісу є функції пошуку по сайту, можливість обрати кількість та специфікації обраного товару, можливість зв’язатися з адміністратором або власником сайту.

ПЗ пропонує різні варіанти використання для різних ролей користувачів – таблиця 1.2.

Таблиця 1.2 – Актори ПЗ та їх функції

Актор	Доступні шляхи використання ПЗ
Незареєстрований користувач (гість)	Перегляд товарів, пошук товарів, використання фільтрів для пошуку, реєстрація в системі для подальшого повноцінного використання
Зареєстрований користувач	Перегляд товарів, вхід в систему, пошук товарів, додавання товарів до «Улюблених», редагування персональної інформації та зміна паролю, створення замовлення та покупка товарів

Вперше перейшовши на сайт потенційний клієнт має мати можливість переглядати товари та переходити по доступних сторінках сайту для ознайомлення з додатковою інформацією. При бажанні зробити замовлення

користувач має мати можливість зареєструватись в системі або залогінитись, за умови уже створеного раніше аккаунта.

Зрозуміло, що для цього мають бути реалізовані методи взаємодії сайту з віддаленою базою даних.

Розроблюваний додаток має складатися з головної сторінки з товарами, сторінки-корзини, в яку можна додавати відразу кілька товарів. Також користувач має мати можливість власне замовити товар. Тобто відправити заявку та оплатити товар через інтернет-банкінг або одну з платіжних систем.

Веб-додаток має бути кросбраузерним та кросплатформеним.

Для зручності розуміння доступних дій для кожного користувача системи доречно створити таблиці варіантів використання ПЗ. Варіанти використання системи для незареєстрованого користувача описано у таблиці 1.3.

Таблиця 1.3 – Опис варіантів використання для незареєстрованого користувача

Актор	Варіант використання	Опис ВВ
Незареєстрований користувач (гість)	Перегляд даних, пошук	Незареєстрований в системі користувач може переглядати товари на сайті та усю інформацію про них; може шукати товари з використанням фільтрів для отримання оптимального результату
	Реєстрація	Гість може зареєструватися в системі використавши електронну пошту або соціальні мережі для подальшого входу в систему

Таблиця 1.4 – Опис варіантів використання для зареєстрованого користувача

Зареєстрований користувач	Авторизація	Зареєстрований користувач може ввійти в свій аккаунт використавши власні дані для авторизації – догін та пароль
	Перегляд та пошук товарів	Користувач може переглядати товари на сайті та усю інформацію про них; може шукати товари з використанням фільтрів для отримання оптимального результату
	Додання до Улюбленого	Користувач може додавати товари в кошик та видаляти товари з кошика
	Створення замовлення	Користувач може створити замовлення товару та оплатити товар, використавши метод електронної оплати

1.4 Висновки. Постановка завдання

Отже, у першому розділі було проведено детальний аналіз предметної області та визначено ряд особливостей при сучасних онлайн-магазинів. А саме:

– необхідність створення інтуїтивного дизайну та зручної навігації;

– необхідність створення функціоналу для різних потенційних користувачів платформи;

– необхідність створення простої логіки дій користувача від початкового етапу пошуку товару на сайті до фінальної стадії оплати товару.

Також було проведено аналіз доступних популярних рішень, а саме трьох світових та українських сайтів. Було визначено їхні основні якісні та негативні

особливості. Це дозволить покращити розроблювану систему та уникнути використання негативних практик, як для дизайну так і для логіки ПЗ. Для порівняння та спрощення даних було створено три таблиці та дві діаграми.

Крім того, у першому розділі кваліфікаційної роботи визначено функціональні та нефункціональні вимоги до розроблюваного ПЗ.

На завершення першого розділу необхідно сформулювати висновки та виділити головні завдання кваліфікаційної роботи. Згідно з усіма вище наведеними даними і для досягнення мети роботи поставлено такі завдання:

– провести загальне комплексне ознайомлення зі сферою інтернет-торгівлі та веб-застосунками для продажу товарів через мережу Інтернет;

– визначити особливості предметної області;

– провести аналіз технологій, методологій, практик розробки веб-застосунків та вибір оптимальних для подальших використання;

– провести ознайомлення з сучасними архітектурними та функціональними рішеннями, вибрати ті, які будуть використані у проекті;

– розробити структуру та функціональну модель проекту;

– реалізувати проект програмним шляхом;

– провести тестування веб-застосунку;

– пілотувати документацію, що описує процес виконання проекту.

2 ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

2.1 Архітектурне проектування

2.1.1 Аналіз використання клієнт-серверного типу архітектури в сучасних веб-застосунках

Інтернет-застосунки, що потребують постійного обміну даних між клієнтом та сервером, зазвичай, використовують клієнт-серверну архітектуру. Це означає добре налагоджений постійний обмін даними між клієнтом – браузером, який використовує користувач, та сервером – програмним (та апаратним) забезпеченням, яке обробляє отримані дані певним визначеним шляхом зберігає їх та може використовувати.

Клієнтська сторона за допомогою інтерфейсу, котрий спрощує роботу користувача, відправляє запити до сервера використовуючи мережу Інтернет, а сервер надає відповіді на запити та обробляє запити від багатьох клієнтів одночасно. Така архітектура дозволяє розділити завдання між клієнтом та сервером, зменшуючи навантаження на кожну зі сторін та покращуючи продуктивність та ефективність системи в цілому.

Клієнт-серверна архітектура є дуже популярною та широко використовується в сучасному програмному забезпеченні, так як вона дозволяє створювати більш масштабовані та надійні системи, забезпечуючи легкість та швидкість доступу до ресурсів. Крім того, вона забезпечує більш високий рівень безпеки, оскільки сервер може контролювати доступ до даних та надавати різні рівні доступу для різних користувачів.

Клієнт та сервер взаємодіють за допомогою мережевих протоколів, таких як HTTP, TCP або UDP. Клієнтські частини ПЗ можуть бути написані на різних мовах програмування, як правило, завжди використовується мова JavaScript, та технології HTML та CSS, а серверні частини додатків зазвичай розробляються з використанням мов програмування, таких як Java, Python, PHP або C #. У такій ситуації особлива увага звертається на процес передачі інформації від клієнта до сервера та навпаки.

Процес обміну даними між клієнтом та сервером є ключовим компонентом клієнт-серверної архітектури. Він складається з кількох етапів: відправки запиту від клієнта до сервера, обробки запиту на сервері та надсилання відповіді з сервера до клієнта. Протокол виконують ключову роль у забезпеченні безпечного та ефективного обміну даними між клієнтом та сервером. Вони визначають формат та механізми обміну даними, а також забезпечують безпеку передачі. Крім того, їх використання дозволяє розробникам створювати програмне забезпечення, яке може працювати на різних пристроях та платформах.

Для обміну даними можуть використовуватись різні протоколи. Наразі, для обміну даними між клієнтом та сервером найчастіше використовують HTTP (Hypertext Transfer Protocol). Цей протокол використовується для передачі гіпертекстових документів, є простим у використанні та підтримує більшість форматів даних.

Ще однією обов'язковою для використання технологією є API. Application Programming Interface - це набір серверних протоколів, інструментів та стандартів, які дозволяють програмним продуктам обмінюватися даними, що значно спрощує обмін даними у 2023 році. API дозволяє створювати програмне забезпечення, яке може взаємодіяти з іншими програмними продуктами та сервісами. API можна порівняти з інтерфейсом користувача, який дозволяє людям взаємодіяти з програмними продуктами. Але взаємодію з програмним продуктом здійснюють не люди, а інші програмні продукти. Наприклад, веб-додатки можуть використовувати API соціальних мереж для авторизації користувачів, відображення їхніх даних або додавання коментарів до записів.

API стали популярними не так давно, але вже є незамінними для спрощення процесу розробки та поліпшення обміну даними. Крім того, для зручності прийнято використовувати REST (Representational State Transfer). REST це архітектурний стиль, який використовується для розробки веб-додатків та API. Його основними принципами є використання HTTP-протоколу для передачі даних та ресурсів, що представляють собою стан додатку на певний момент часу.

REST дозволяє створювати легкі та швидкі API, що можуть бути використані різними додатками та сервісами. Використання стандартних HTTP-методів, таких як GET, POST, PUT та DELETE, дозволяє зменшити складність розробки та спростити взаємодію між додатками. Ще однією важливою перевагою REST є те, що він дозволяє використовувати кешування для збереження даних та ресурсів, що зменшує навантаження на сервер та забезпечує більш швидку відповідь.

2.1.2 Аналіз підходів до проектування серверної архітектури

Оскільки саме серверна частина ПЗ відповідає за обробку даних та надсилання коректних відповідей користувачу, то потрібно з особливою увагою обирати технології та архітектуру серверної сторони додатку. На щастя, для цього є достатньо хороших та надійних підходів.

Серверна архітектура - це організація та розподіл ресурсів, що забезпечують функціонування серверного програмного забезпечення.

Одним з найпоширеніших підходів є класична серверна архітектура, яка використовується для багатьох веб-додатків, також її називають монолітною. Вона базується на технології трьохрівневої архітектури, яка розділяє серверну архітектуру на три рівні – окремі модулі. Презентаційний рівень, рівень бізнес логіки та рівень даних. Кожен рівень відповідає за свої завдання і забезпечує певну функціональність, це дозволяє логічно спростити розробку та покращити процес подальшої підтримки додатку, а також тестування.

Першим рівнем монолітної серверної архітектури - це рівень представлення, який відповідає за взаємодію з користувачами та відображення вмісту. Це візуальний інтерфейс сайту, також на цьому рівні знаходяться шаблонізатори, стилі та скрипти, які відповідають за відображення вмісту на стороні клієнта.

Другий рівень - це бізнес-логіка. Цей рівень відповідає за обробку подій.

Логіка відповідає за виконання операцій, які пов'язані з бізнес-процесами. На цьому рівні розміщуються компоненти, такі як контролери, сервіси та репозиторії, які забезпечують обробку запитів користувачів та взаємодію з базою даних веб-застосунку.

Третім рівнем є рівень доступу до даних, який відповідає за зберігання даних та забезпечує або обмежує доступ до даних. На цьому рівні зазвичай знаходяться база даних та ORM (об'єктно-реляційне відображення), які забезпечують доступ до даних.

Також можна використовувати четвертий, необов’язковий рівень – рівень інтеграції. Він використовується для подальших інтеграцій системи з іншими видами ПЗ і тому не є обов’язковим у кожному додатку.

Головною перевагою монолітної архітектури є простота розробки та розгортання додатку. Оскільки всі компоненти додатку знаходяться в одному монолітному сервісі, це зменшує складність взаємодії між ними та спрощує тестування та налагодження. Крім того, розгортання монолітного додатку на сервері зазвичай є досить простим та швидким процесом.

Іншою перевагою монолітної архітектури є її ефективність. Монолітний додаток зазвичай працює швидше та з меншою кількістю затримок, оскільки всі компоненти додатку працюють в одному процесі та взаємодіють один з одним напряму. Також монолітна архітектура забезпечує простоту управління додатком. Оскільки всі компоненти знаходяться в одному монолітному сервісі, це забезпечує зручність моніторингу та керування додатком, зокрема при подальших внесеннях змін в роботу системи.

Хоча монолітна серверна архітектура є досить стандартним підходом до розробки програмного забезпечення, вона має свої недоліки. Один з основних недоліків полягає в тому, що великий моноліт може бути складним для розуміння та підтримки. Для роботи з великими проектами вона не є оптимальним рішенням через складність внесення подальших змін, оскільки часто частини коду будуть зв’язані з іншими, шо ускладнює внесення змін. Але це не стосується невеликих проектів.

Інший підхід - це мікросервісна архітектура, в такому випадку система розділяється на невеликі незалежні компоненти, які взаємодіють один з одним.

Взаємодія відбувається через інтерфейси програмного забезпечення (API). Кожен компонент може бути розроблений, тестований та впроваджений незалежно від інших компонентів. Цей підхід дозволяє збільшити масштабованість, доступність та стійкість системи.

Одна з головних переваг мікросервісної архітектури полягає у гнучкості та масштабованості додатку. Кожен сервіс може бути розроблений та розгорнутий окремо, що дозволяє ефективно використовувати ресурси сервера та масштабувати додаток в залежності від потреб. Модулі мікросервісної архітектури значно простіше підтримувати та змінювати, тому що вони є невеликими та відповідають за певний конкретний процес, хоча розробка таких модулів може займати більше часу. Ще однією перевагою є можливість подальшого повторного використання окремих модулів для нових завдань. Крім того, мікросервісна архітектура дозволяє розробникам використовувати різні технології та мови програмування для кожного сервісу, що дозволяє підібрати оптимальні технічні рішення для кожної задачі та складності сервісу.

Однак, мікросервісна архітектура має свої недоліки. Наприклад, вона може бути складною для розгортання та налагодження, оскільки потрібно керувати кількома сервісами та забезпечувати їх взаємодію між собою. Також, мікросервісна архітектура може збільшувати навантаження на мережу та базу даних, оскільки кожен сервіс потребує звернення до цих компонентів.

Загалом, мікросервісна архітектура є вигідною для складних та масштабних додатків, які потребують гнучкості та масштабованості. Однак, для невеликих проектів може бути доцільнішою монолітна архітектура, яка має свої переваги в простоті та ефективності.

Інші підходи включають серверну архітектуру на основі контейнерів, де кожен компонент системи запускається у власному контейнері, що забезпечує більш високий рівень ізоляції та незалежності компонентів. Також існують підходи на основі serverless архітектури, де компоненти запускаються тільки тоді,

коли вони потрібні та автоматично вимикаються, коли завдання виконано.

З огляду не переваги та недоліки розглянутих технологій проектування можна вважати, що монолітна архітектура буде оптимальним рішенням для використання при розробці онлайн-магазину. Додаток не має бути занадто об’ємним і сервер не повинен виконувати жодних надзвичайно важких операцій з даними або складних операцій для підтримки роботи системи. Основним завданням сервера буде надсилання даних з бази даних на сторону клієнта – з цим монолітна архітектура впорається оптимальним шляхом; наступною важливою функцією сервера буде обробка даних отриманих від клієнта – завдяки поділу архітектури на рівні ці операції також буде доволі зручно реалізувати та отримати максимально коректну та безперебійну функціональність.

2.2 Детальне проектування

2.2.1 Аналіз та огляд доступних рішень

Наразі платформа Node.js пропонує багато доступних продуктивних рішень для створення серверних додатків з орієнтацією на певні завдання. Оскільки написання серверної сторони на «чистому» JavaScript є ресурсозатратним рішенням, було розглянуто допоміжні варіанти.

Зокрема фреймворк Express.js. Це легкий та гнучкий веб-фреймворк для Node.js, який дозволяє швидко та якісно створювати серверні веб-додатки. До переваг Express.js відносять просту у використанні та гнучкість. Він дозволяє швидко створювати серверні додатки без необхідності заглиблення у деталі роботи платформи, при цьому Express.js дозволяє розширювати функціональність сервера за допомогою плагінів та необхідної кількості middleware, що дозволяє працювати з будь-якою базою даних та взаємодіяти з будь-якими іншими веб-сервісами. Також перевагою фреймворку є швидкість. Express.js є одним з найшвидших сучасних веб-фреймворків для Node.js. Він має мінімальну кількість внутрішніх залежностей та швидку обробку запитів.

Це також дозволяє легко додавати нову функціональність до серверного додатка і створювати складні веб-додатки, які легко масштабувати. Ці переваги дозволять реалізувати оптимальний функціонал платформи.

Фреймворк Express часто використовується разом з додатковими ORM для покращення практики розробки. Для обробки запитів та подій буде використовуватись Event Loop. Петля подій - це механізм асинхронної обробки подій, який широко використовується в браузерах та середовищах виконання JavaScript, а саме Node.js. Основна ідея Event Loop полягає в тому, що програма продовжує виконуватись поки в черзі існують події, що очікують обробки. Це дозволяє програмі не блокувати свій потік виконання та виконувати інші задачі в той час, коли виконання поточної задачі призупинено через очікування події.

Event Loop дозволяє зменшити час очікування користувача і є дуже корисним механізмом для обробки багатьох асинхронних операцій, таких як мережеві запити, операції з файлами, введення та виведення даних, таймери та багато іншого. Крім того, Event Loop дозволяє програмі зберігати ресурси та ефективно використовувати процесорний час, тому що виконання поточної задачі не блокує виконання інших задач, що можуть бути оброблені у той же час.

Загалом, Event Loop є потужним інструментом для обробки асинхронних операцій та забезпечення ефективної роботи програм. Він дозволяє програмі взаємодіяти з іншими системами та ресурсами, не блокуючи виконання інших задач, що може дуже суттєво підвищити продуктивність та ефективність додатку.

В об’єднанні з монолітною архітектурою ми отримаємо наступний схематичний вигляд архітектури системи – рисунок 2.1.

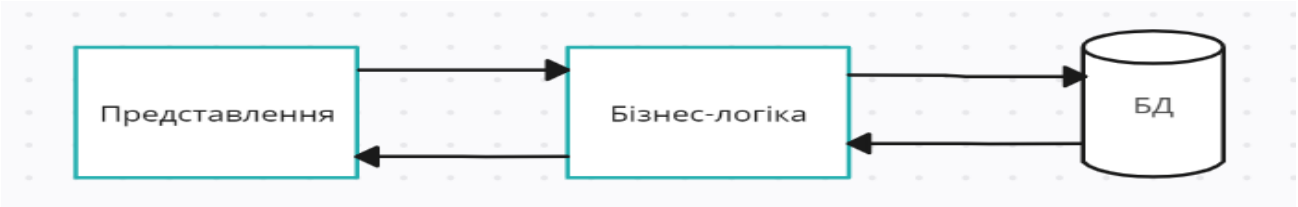


Рисунок 2.1 – Монолітна архітектура ПЗ

Відповідно до рисунка 2.1, Event Loop використовується при русі даних між користувачем та сервером та між сервером та базою даних і забезпечує швидкий обмін даними з використанням певних засобів, найчастіше для асинхронних запитів використовуються AJAX та Fetch, але власне процес обміну даними детальніше описано у розділі 3 – Розробка програмного забезпечення.

Від рівня Представлення будуть надсилатись запити на рівень Бізнес-логіки, для подальшої обробки. Запити використовуватимуть систему REST для створення оптимального зв'язку складових ПЗ. Після обробки даних на бізнес рівні REST-API надсилатиме відповідь з сервера на рівень представлення, часто у цьому процесі запити також надсилатимуться і до БД для запису даних, або отримання певного набору даних, необхідного клієнтській стороні. Така організація логіки є доволі зрозумілою і зручною при редагуванні.

2.2.2 Аналіз та вибір типу бази даних

Вибір типу бази даних є відповідальним етапом створення ПЗ. Потрібно обрати оптимальний варіант саме для зберігання наборів даних онлайн-магазину. Сьогодні існує багато варіантів баз даних і кожен з них має свої переваги та способи використання. Наразі для проекту оптимальними здаються три типи баз даних, розглянемо їх.

Реляційні бази даних: це найбільш поширений тип баз даних, який використовує таблиці для зберігання даних. Кожна таблиця складається з рядків та стовпців, і кожен рядок відповідає за один запис даних. Реляційні бази даних можна використовувати як для розробки бізнес додатків, так і для освітніх та інших галузей. Реляційні бази даних займають почесне місце серед усіх БД і на це є ряд причин:

– використання стандартизованої мови запитів SQL (Structured Query Language), що дозволяє легко взаємодіяти з різними базами даних. Це забезпечує сумісність між різними реляційними СУБД та дозволяє легко переносити дані.

– забезпечення можливості зберігати та організовувати дані у вигляді таблиць, що дає можливість легко змінювати та оновлювати дані без необхідності вносити значні зміни до БД.

– забезпечення можливості проводити складні запити та аналізувати великі об'єми даних. Запити можуть бути проведені шляхом використання команд SQL, що дозволяє швидко та ефективно знайти та відобразити потрібну інформацію.

– забезпечення механізмами безпеки, які дозволяють обмежувати доступ до даних залежно від рівня користувача. Це дає можливість захищати конфіденційну інформацію та забезпечувати відповідність різноманітним стандартам безпеки.

Іншим підходящим типом є доволі популярні зараз NoSQL бази даних: цей тип БД не використовує реляційну структуру таблиць, і використовує інші формати для зберігання даних: ключ-значення, документи, графи, колонки. NoSQL бази даних часто використовуються для обробки великих об'ємів даних та інших завдань, що вимагають високої продуктивності. До таких БД належать широковідомі в останні роки MongoDB та Firebase.

NoSQL бази даних мають ряд переваг і тому широко використовуються у сучасних проектах. Вони забезпечують більшу гнучкість, оскільки дані можна зберігати без фіксованої структури, дозволяючи легко додавати нові поля та оновлювати структуру даних без необхідності змінювати базу даних. Крім того, NoSQL бази даних дозволяють легко масштабувати базу даних, як горизонтально (додаванням нових серверів у кластер), так і вертикально (збільшенням ресурсів на одному сервері), що дозволяє пристосовуватись до зростаючих вимог щодо обсягу даних та навантаження на систему.

Крім того, NoSQL бази даних дозволяють ефективно зберігати та обробляти великі об'єми даних, що дозволяє отримувати швидкі результати з запитів. Наявність розподіленої структури дозволяє зберігати дані на декількох серверах, що забезпечує більшу стійкість до відмов та зменшує час відновлення системи після відмови.

Також NoSQL бази даних підтримують графи, документи, ключ-значення та інші, що дозволяє використовувати базу даних для різноманітних вимог. Таким чином, NoSQL бази даних забезпечують більшу гнучкість, масштабованість, швидкість та різноманітність типів даних порівняно з реляційними базами даних.

Третім типом є ієрархічні бази даних зберігають дані у вигляді дерева, де кожен вузол має дочірні вузли. Ці бази даних часто застосовуються в додатках для керування даними, де ієрархічна структура даних проста та зрозуміла, наприклад, в бібліотеках або додатках для управління проектами.

У підсумок варто сказати, що як реляційні так і NoSQL БД мають свої переваги і широко використовуються у сучасних проектах. І хоча NoSQL пропонують високу гнучкість та швидкість роботи системи, у даному випадку буде зручніше використати реляційну БД з можливістю повноцінного використання таблиць для розміщення інформації про товари.

2.2.3 Вибір системи керування базами даних

Після вибору реляційного типу бази даних для реалізації зберігання даних в програмного продукту необхідно вибрати оптимальну систему керування базами даних (СКБД). Очевидними є три системи: PostgreSQL, MySQL та MS SQL Server. Для вибору оптимального варіанту потрібно розглянути перваги та недоліки кожної СКБД.

PostgreSQL - це потужна та надійна система управління реляційними базами даних з відкритим вихідним кодом, яка має безліч переваг для використання в різних проектах. Одна з найважливіших переваг СКБД полягає у її розширюваності та масштабованості. База даних може працювати на багатьох серверах, що забезпечує можливість розподіленої обробки даних та підтримку великої кількості одночасних користувачів, що необхідно для онлайн-магазину.

СКБД також має широкі можливості для забезпечення безпеки даних, включаючи механізми шифрування, автентифікації та контролю доступу.

База даних підтримує створення власних типів даних та функцій, що дозволяє розширювати можливості бази даних у відповідності з потребами конкретного проекту. Також важливою перевагою PostgreSQL є підтримка повнотекстового пошуку, що дозволяє швидко та ефективно шукати та аналізувати великі об'єми текстової інформації.

В цілому, PostgreSQL є потужним та гнучким рішенням для зберігання та управління реляційними даними, яке має безліч переваг для використання в різних проектах та є зручною для роботи зі статичними наборами даних.

Наступною популярною реляційною СКБД є MySQL. По-перше, це дуже швидке та ефективне рішення, що дозволяє обробляти великі обсяги даних за короткий час. Вона також має широкий набір інструментів для оптимізації продуктивності. По-друге, MySQL має велику спільноту користувачів та розробників, що забезпечує широку підтримку, надійність та постійний розвиток системи. По-третє, СКБД має велику гнучкість та можливості розширення. Вона підтримує різні типи даних, включаючи текстові файли, зображення та відео, що дозволяє зберігати різноманітні дані в одній базі даних і використовує безкоштовну ліцензію, що робить її доступною та безкоштовною для використання для всіх користувачів.

Третьою системою для розгляду є MS SQL Server. Дана СКБД широко використовується на великих та малих підприємствах і має ряд переваг. Перш за все, MS SQL Server має високий рівень безпеки, що дозволяє зберігати дані в безпечному середовищі. Вона забезпечує захист даних на різних рівнях: від автентифікації користувача до шифрування даних та захисту від SQL-ін'єкцій. Також MS SQL Server добре масштабується, СКБД можна легко масштабувати залежно від потреб. Для цього вона має ряд інструментів, що дозволяють виконувати операції з розширенням та зменшенням обсягу даних. Крім того, MS SQL Server має можливість реплікації даних, що дозволяє створювати дублікати баз даних та забезпечувати доступ до них з різних місць.

З огляду на переваги кожної СКБД варто обрати PostgreSQL. Дана СКБД є об’єктно-реляційною та дозволяє працювати з даними різних форматів.

Для роботи можна використовувати популярний формат JSON та набір програм, наприклад Postman та Beekeeper Studio, які дозволять якісно відслідковувати роботу системи в цілому та шляхи виконання запитів. Зважаючи, що у проекті точно буде використовуватись API, це є доволі важливим чинником спрощення процесу перевірки функціональності системи.

2.2.4 Проектування функціональної архітектури

Розробка функціональної архітектури є незамінним етапом проектування системи для подальшої успішності розробки. Саме функціональна архітектура описує як функціональні боки будуть взаємодіяти між собою для забезпечення функціональності системи в цілому. Також це оптимальний тип опису архітектури для проектів такого розміру, адже архітектура модулів інтернет-магазину є доволі зрозумілою і її розробка не є необхідною, а от взаємодія функціональних блоків системи може спричинити значні проблеми на етапі кодування за відсутності чіткого попереднього уявлення про це.

Функціональна архітектура тісно пов’язана з функціональними вимогами до системи і повинна затвердити шляхи реалізації комплексних функціональних вимог. Крім опису доречно використовувати діаграми для поліпшення розуміння та візуалізації моделей.

Для поліпшення загального уявлення про систему створено функціональну діаграму – рисунок 2.2. Складним з функціональної точки зору можна вважати процес входу та реєстрації в системі. Комплексно можна використовувати термін «Аутентифікація». Аутентифікація - це процес перевірки та підтвердження ідентифікаційних даних користувача, щоб підтвердити його право на доступ до об'єкта або ресурсу.

У контексті інформаційної безпеки, аутентифікація забезпечує захист від несанкціонованого доступу до конфіденційної інформації та ресурсів. Аутентифікація може виконуватися за допомогою різних методів, таких як

введення пароля, використання біометричних даних, використання токенів та інших ідентифікаторів.

Тільки після аутентифікації користувачу буде надано доступ до повної функціональності застосунку, а саме до замовлення товарів. Для забезпечення високого рівня безпеки, аутентифікація повинна бути поєднана з авторизацією, тобто при наявності ролей клієнта та адміністратора функції, доступ до яких отримає користувач залежать від його рівня авторизації.

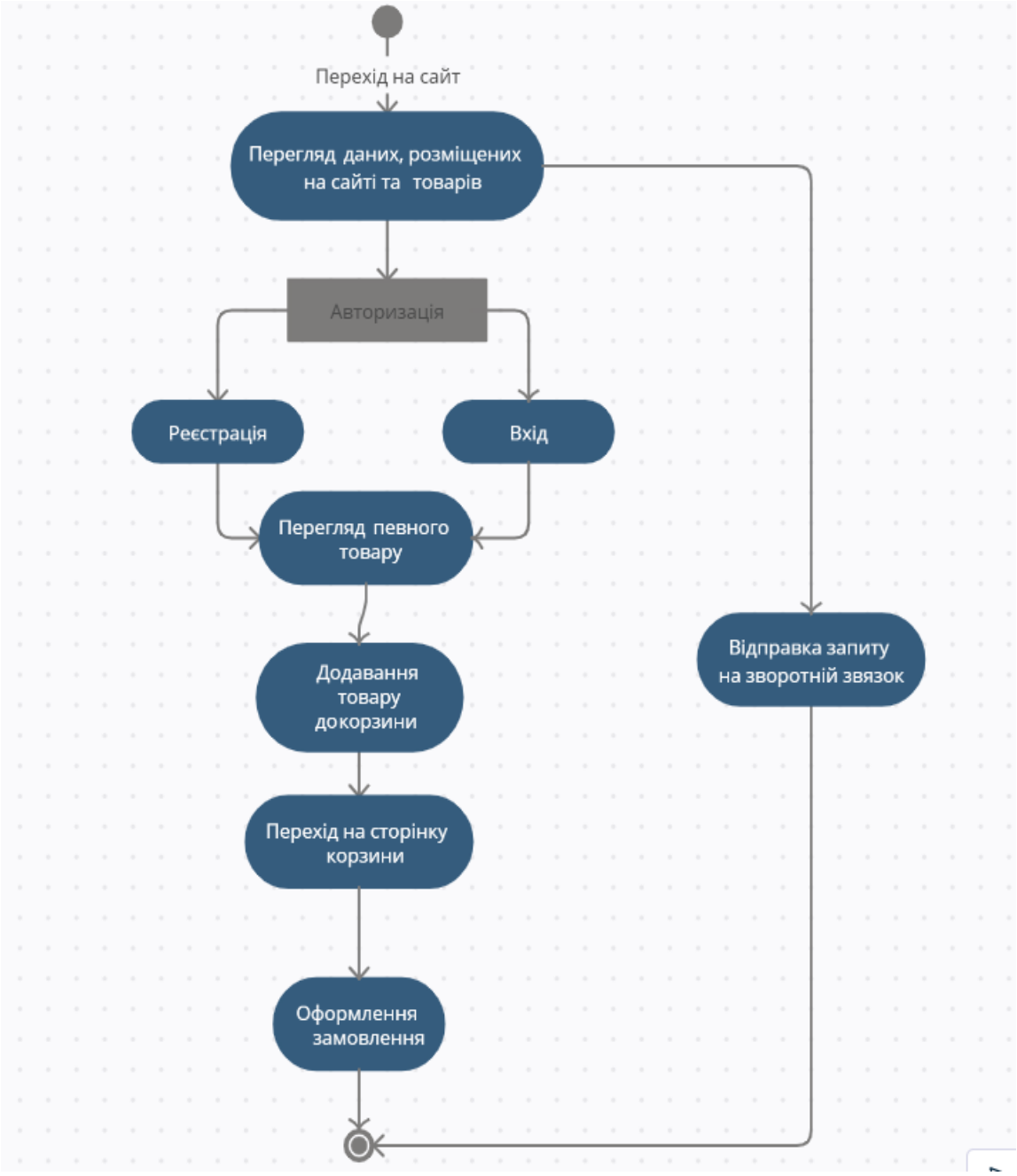


Рисунок 2.2 – Функціональна діаграма системи

На сьогодні, оптимальним рішенням для аутентифікації користувачів у системі є токени доступу – набори даних, які місять у собі зашифровану інформацію і складаються з латинських символів та цифр. Це забезпечує безпеку передачі даних та мінімізує ризики отримання персональних даних користувачів третіми особами, навіть персоналом або власниками ПЗ.

Процес авторизації в системі є комплексним, тому для покращення розуміння було створено діаграму декомпозиції другого рівня (рисунок 2.3).

Рисунок 2.3 - Діаграма другого рівня декомпозиції

Найпоширенішою практикою у сучасних додатках є використання JSON Web Token – JWT. Це стандарт створення токенів для аутентифікації та авторизації для веб-додатків та API. JWT утворюється з трьох складових: заголовку (header), тіла (payload) і підпису (signature). У заголовку зазначається тип токену та тип алгоритму шифрування для підпису. У тілі містяться дані для аутентифікації та авторизації, такі як ідентифікатор користувача, роль та термін дії токену. Підпис забезпечується за допомогою секретного ключа, що зберігається на сервері.

JWT використовується наступним чином. Коли користувач системи заходить в систему використовуючи логін та пароль (або інші дані) сервер аутентифікації створює JWT и відправляє токен користувачу-клієнту. Володіючи токеном, користувач при кожному запиті до API відправляє також сам токен. І оскільки сервер ПЗ має секретний ключ (підпис) токена, то при отримані запиту

від клієнта, він перевіряє ключ на ідентичність і таким чином перевіряє власника токена.

Також JWT має алгоритм продовження сесії користувача автоматично. Тобто токен, який користувач отримує при аутентифікації не буде дійсним протягом усього сеансу – це було б потенційною загрозою безпеці. При використанні системи та відправленню запитів сервер аутентифікації самостійно згенерує новий токен та відправить його клієнтові. Це відбудеться через певний короткий період часу, який можна визначити залежно від цілей. Саме такий алгоритм дій буде використано у веб-застосунку для реалізації аутентифікації та авторизації. Крім того, база даних та сервер застосунку майже не беруть участі у підтримці автентифікації користувача, це зменшує навантаження на сервер.

2.3 Проектування інтерфейсу користувача

Інтерфейс має бути мінімалістичним і при цьому відображати всю головну інформацію про товари. Також має використовуватись приємна для сприйняття кольорова гамма та округлі форми елементів. Перейдемо до функціональності. Для початку доречно навести діаграму варіантів використання системи для клієнта (рисунок 2.4), це допоможе визначити основні сторінки системи.

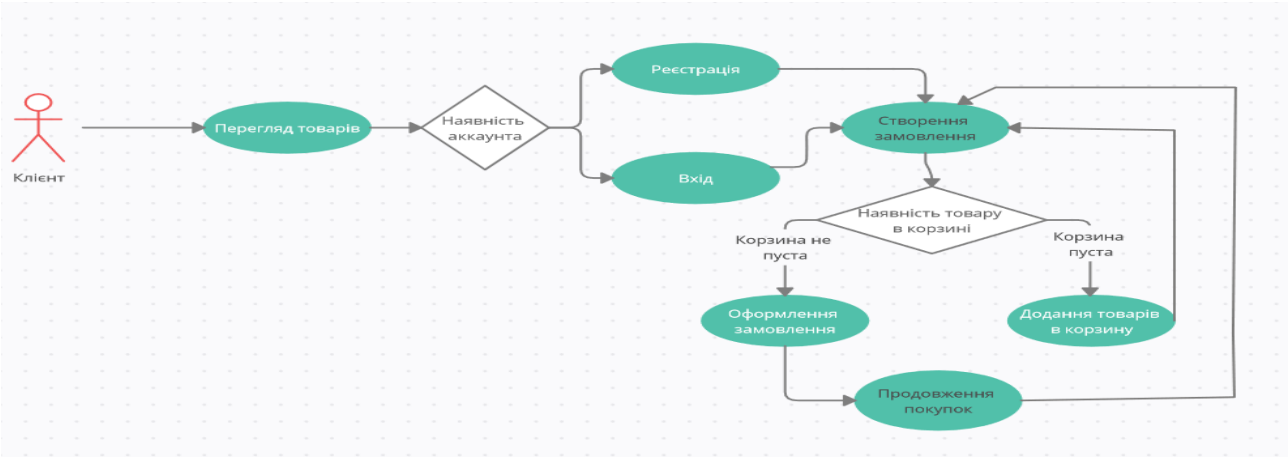


Рисунок 2.4 – Діаграма варіантів використання для клієнта для придбання товару

Діаграма відображає послідовність подій для виконання покупки клієнтом. Крім показаних на діаграмі процесів, які потребують певного вікна інтерфейсу, ПЗ має забезпечувати ряд додаткових функцій, для яких теж необхідно сотворити сторінки інтерфейсу. Для точного відображення усіх сторінок системи було створено візуальне відображення структури інтерфейсу ПЗ (рисунок 2.5). Кожний прямокутник описує окреме вікно програмного продукту.

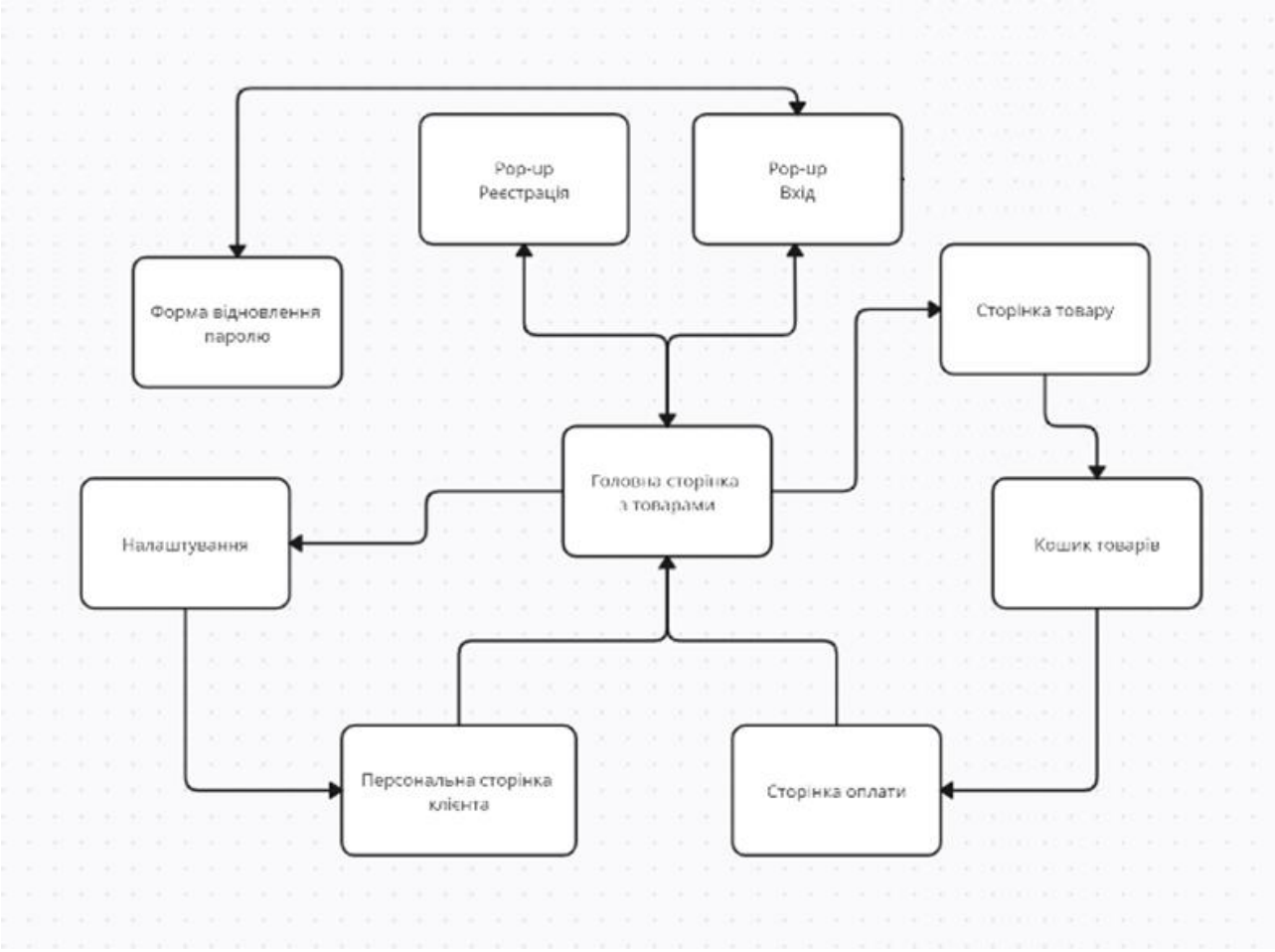


Рисунок 2.5 – Структура інтерфейсу веб-застосунку

Як показано на рисунку, основним елементом є головна сторінка онлайн-магазину. Саме на ній користувач може бачити товари, здійснювати пошук товарів, використовувати фільтри та навігацію по категоріях. Також на цій сторінці передбачено створення навбару та тайрбару – елементів які дозволять зручно перемішуватися між категоріями товарів та брендами-виробниками.

Також на головній та інших сторінках у секції футер має бути розміщено спеціальну секцію з інформацію про магазин та контакти для прямого з'єднання з працівником магазину – для консультації або уточнення деталей, адже часто користувачам потрібно впевнитись у специфікаціях товару.

Іншими важливими сторінками є сторінка корзини вподобаних товарів та сторінка перегляду певного товару. На сторінці огляду певного товару має бути розміщено детальні дані про товар, його фото, опис, особливості. З цієї сторінки товар можна додати до Вподобаного для подальшого придбання. На сторінці «Кошик» користувач системи зможе бачити вподобані товари, обирати їх особливості, кількість та створювати замовлення. Власне для оплати товарів електронним способом користувач буде перенаправлений на іншу сторінку.

Також на рисунку 2.3 продемонстровано ряд інших сторінок інтерфейсу, але потреба у їх детальному описі відсутня, адже вони наслідують стиль головної сторінки та не вміщують особливих функціональних елементів, котрі потребують детального опису можливостей або шляхів взаємодії з іншими.

Усі сторінки платформи мають бути уніфіковані та виконані в одному стилі. Для цього було обрано створений корпорацією Google Material Design. Це мінімалістичний та добре знайомий користувачам візуальний вигляд елементів сторінки. Він використовує заокруглені форми об'єктів – кнопок, елементів списку товарів, навігаційних панелей, полів вводу тексту. Елементи дизайну не повинні бути занадто малими або занадто великими, потрібно використовувати оптимальні розміри, форми, кольори елементів, які підвищать зручність використання застосунку. Переважаючими є м'які та теплі кольори, але має бути збережена контрастність для зручності сприймання інформації та рекомендовано не використовувати більше 4 кольорів – два основних та два додаткових для створення відтінків. Така реалізація дизайну відразу створює позитивне враження у користувача та не потребує розробки складних дизайнерських рішень. Оскільки елементи введення даних для авторизації реалізовано за технологією pop-up, то варто передбачити затемнення основної сторінки під час відкриття даних елементів.

Іншою важливою складовою є панель адміністратора. Сторінка має бути реалізована у такому ж стилі, але мати ряд функціональних можливостей та полів для вводу текстових даних під час роботи з БД. Це дозволить звичайному адміністратору магазину працювати з базою даних шляхом вводу тексту в звичайні текстові поля. Варто передбачити кілька варіантів – створення товару, редагування, видалення. Це необхідно, адже навіть ціна товару може змінюватися з часом і неправильно не надати адміністратору можливості змінити її власноруч.

2.4 Аналіз та вибір технологій для розробки клієнтської частини веб-застосунку

Клієнтська частина додатку є інтерфейсом користувача, за допомогою якого звичайна людина може взаємодіяти з сервером та базою даних використовуючи лише добре зрозумілі для неї елементи. Для розробки фронтенд частини додатку при йняти використовувати JavaScript та один із популярних фреймворків, що дозволяють пришвидшити розробку інтерфейсу за допомогою мови розмітки гіпертексту, каскадних таблиць стилів та власне функціональності, яку розробляють на JavaScript.

Зважаючи на вимоги до сучасних веб-застосунків та сучасні тенденції у сфері веб-розробки вирішено використовувати не звичайний JavaScript, а розширену його версію, тобто TypeScript. TypeScript має ряд переваг, які допомагають створювати сучасні досконалі застосунки та уникати певних помилок. Найбільшим плюсом використання TypeScript є те, що такий застосунок матиме статичну типізацію. Це дозволить ще на етапі опису моделей уникнути подальших проблем з типами даних або об’єктів. Крім того, TypeScript має багато спільного з ООП і такий код значно легше читати та підтримувати.

Сьогодні найпопулярнішими фреймворками для розробки інтерфейсів користувачів (UI) є React та Angular. Кожна з технологій має свої переваги і опціональні способи використання.

React широко використовується світовими гігантами медіа-індустрії – Facebook, Instagram, Netflix. Фреймворк побудований на основі компонентної структури, що дозволяє легко розбивати UI на самодостатні компоненти, що виконують тільки одну певну функцію. Це дозволяє повторно використовувати більшість елементів інтерфейсу, котрі повторюються. Ще однією якісною особливістю фреймворку є використання віртуальної моделі DOM, що дозволяє оптимізувати процес рендерингу та забезпечує оптимальну швидкість. React.js є компонентною бібліотекою, що дозволяє використовувати багато готових компонентів з відкритих джерел та легко масштабувати додатки і зберігати їх ефективність навіть при зростанні обсягів даних та функціональності.

Іншим популярним потужним інструментом для розробки інтерфейсу веб-додатків є Angular. Цей варіант забезпечує модульну архітектуру і дозволяє розробникам зосередитися на окремих частинах додатку, при цьому зменшуючи залежності між ними. Це робить код більш організованим і дозволяє використовувати модулі повторно. Також Angular має вбудовану систему валідації форм, що значно полегшує процес валідації для різних елементів форм і робить його обов’язковим для усіх елементів.

Зважаючи на сьогоднішню популярність, швидшу роботу та вражаючі можливості повторного використання елементів у React.js доцільно використати саме його при розробці веб-застосунку. Крім того, довгий час проблемою додатків, фронтенд яких був розроблений на даному фреймворку був рендиринг, який відбувався виключно на стороні клієнта, що значно погіршувало можливості оптимізації та SEO. Наразі ж, за допомогою фреймворку Next.JS є можливість реалізовувати рендеринг компонентів на стороні сервера – так званий SSR (server side rendering).

Також відразу можна визначити бібліотеки, які дозволять покращити фінальний вигляд клієнтської частини та спростити процес розробки. Серед них є популярна бібліотека React-Icons, вона пропонує на вибір широкий список візуальних елементів для дизайну. Наприклад, елемент сердечка для вподобаних товарів, або елемент, візуально схожий на кошик – для корзини товарів.

2.5 Висновки

Підсумком другого розділу є проведений аналіз рішень та технологій клієнт-серверної архітектури у сучасних веб-додатках. У результаті було виявлено, що усі технології використовують з метою забезпечення безпеки даних та підвищення стабільності роботи додатків, разом зі швидкістю.

У результаті детального проектування було визначено ключові елементи системи, зв’язки між ними та шляхи взаємодії між елементами.

У результаті функціонального проектування було визначено вимоги до ПЗ та визначено шляхи реалізації комплексних процесів роботи системи та шляхи і технології взаємодії складових системи для забезпечення безпеки та коректності функціонування.

При аналізі та виборі технологій розробки було визначено технології, які дозволять оптимально реалізувати функціонал додатку та описано якими шляхами буде відбуватися взаємодія між різними модулями та обмін даними.

Для покращення розуміння та візуалізації було створено ряд графічних матеріалів – функціональні діаграми, діаграми декомпозиції.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Створення бази даних програмної системи

Для онлайн-магазинів, тим більше для магазинів, які пропонують віртуальні копії товарів, музичних інструментів в конкретному випадку, необхідна база даних для забезпечення двох функцій. Перше завдання - зберігання даних про товари та іншого контенту, що має бути відображеним на сайті. Друге завдання – зберігання даних користувачів у зашифрованому вигляді: логінів та паролів.

Для роботи БД було установлено СКБД PostgreSQL, для перевірки взаємодії БД з бекендом – Beekeeper Studio Ultimate та OPM Prisma для роботи з БД через віртуальну копію бази даних.

Оскільки процес роботи з базами даних є доволі незручним, для його пришвидшення було використано об’єктно-реляційне відображення Prisma. Ця серверна бібліотека полегшує запис та читання даних у базі даних і дозволяє працювати з БД шляхом описання моделей відразу у середовищі розробки у файлі schema.prisma.

Спочатку було створено модель користувача системи – User. Обов’язковими полями моделі користувача є ID, Email, Password. Окрім них, було додано рядки для збереження даних про дату реєстрації користувача в системі та можливість для користувача створити свій псевдонім саме для цього сайту. І далі було додано сутності моделі, які відображають набори даних – замовлення користувача, вподобані товари. Код моделі користувача – User, для наочності наведено нижче.

```
model User {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  email     String @unique
```

```
  password String
  name       String @unique
  avatarPath String @default("/uploads/default-avatar.png") @map("avatar_path")
  phone String @default("")
  orders    Order[]
  favorites Product[]
}
```

Як видно із фрагменту коду, Prisma дозволяє зручним способом описати поля БД, при цьому задавши тип та особливості кожного з них.

Іншою важливою сутністю переданою ОРМ до бази даних є Product. Дана модель описує усі поля товарів, котрі можуть бути запропоновані на сторінках онлайн-магазину музичних інструментів. Найважливішими є ID, назва, зображення, опис, ціна та належність товару до певної категорії товарів. Код моделі наведено нижче.

```
model Product {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  name        String   @unique
  slug        String   @unique
  description String
  price       Int
  images      String[]
  orderItems OrderItem[]
  reviews    Review[]
  category   Category? @relation(fields: [categoryId], references: [id])
  categoryId Int?      @map("category_id")
  user   User? @relation(fields: [userId], references: [id])
  userId Int?  @map("user_id")
}
```

Останньою важливою моделлю є замовлення. Тут описано ID, користувача, який створив замовлення, товари замовлення та суму.

```
model Order {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  status EnumOrderStatus @default(PENDING)
  items OrderItem[]
  total Int
  user   User @relation(fields: [userId], references: [id])
  userId Int  @map("user_id")
}
```

Також для поля EnumOrderStatus було визначено можливі стадії.

```
enum EnumOrderStatus {
  PENDING
  PAYED
  SHIPPED
  DELIVERED
}
```

Окремою важливою складовою зв’язку ПЗ з БД є підключення до бази даних. Було вирішено винести рядок підключення з файлу schema.prisma до окремого файлу для збереження змінних. Власне рядок підключення має наступний вигляд.

```
DATABASE_URL="postgresql://postgres:123456@localhost:5432/postgres?schema=public"
```

Тут вказано користувача БД, пароль, хост та порт для підключення, а також назву бази даних для підключення. Саме використання даної ОРМ дозволило перенести простий синтаксис з файлу schema.prisma у базу даних та створити

відповідні поля у БД. Для цього було використано базові команди управління користувацьким терміналом або консоллю.

3.2 Реалізація серверної частини програмної системи

Основою будь-якого веб-застосунку для продажу товарів є можливість користувача авторизуватися в системі. Якісною практикою вважається першочергова реалізація можливості авторизації. Для цього створено три файли – auth.controller.ts, auth.module.ts, auth.service.ts. Кожний файл відповідає за обробку певних даних, генерацію і надсилання відповіді клієнту. Такий розподіл логіки є типовим і доволі зручним. Чітке розмежування дозволяє зручно працювати з запитами та логікою, оскільки вони розміщені у різних файлах і, при цьому, взаємодіють через залежності та прошарки – Pipes(). У контролері зберігаються можливі сценарії взаємодії, а у файлі сервісу було розроблено логіку функціонування системи для реєстрації та авторизації користувачів.

Для зберігання та обміну даних про авторизацію користувача використано JSON Web Token – JWT. Технологія містить усі необхідні засоби забезпечення приватності даних користувачів системи. Спочатку було реалізовано реєстрацію та авторизацію в системі. Тут доречно навести код реєстрації.

```
async register(dto: AuthDto) {
    const oldUser = await this.prisma.user.findUnique({
      where: {
        email: dto.email
      }
    })

    if (oldUser) throw new BadRequestException('Такий користувач уже існує')
    const user = await this.prisma.user.create({
      data: {
        email: dto.email,
        name: name.firstName(''),
        avatarPath: image.avatar(),
        phone: phone.number(''),
        password: await hash(dto.password)
      }
```

```
      })
    const tokens = await this.issueTokens(user.id)

    return {
      user: this.returnUserFields(user),
      ...tokens
    }
  }
```

Після перевірки пошти користувача на унікальність, за допомогою Prisma, дані користувача вносяться до БД. Відразу ж користувач отримує токен для подальшого використання, це відбувається непомітно для користувача, але він надсилає отриманий токен при кожному новому запиті до сервера, щоб сервер, маючи унікальний ключ, міг аутентифікувати користувача. Варто відмітити, що для безпеки даних використовується хешування паролю користувача.

Процес логіну користувача на сайті відбувається звичайним пошуком відповідних даних в БД та перевірці співпадіння логіна та пароля. Оскільки технологія JSON Web Token дозволяє налаштовувати період дійсності токенів, було вирішено, що токени доступу та оновлення будуть залишатися протягом однієї години. Тобто, якщо користувач відволочеться на невеликий проміжок часу, йому не буде потрібно знову аворизовуватись в системі онлайн-магазину.

Також важливим етапом реєстрації нового користувача в системі є перевірка введених ним даних, або валідація даних. Nest.js має зручний механізм реалізації валідації – у окремому фалі один раз потрібно прописати певні умови, і потім ці умови можна повторно використовувати у інших файлах, просто зробивши імпорт необхідного файлу і використавши клас pipe. Нижче наведено код валідації при реєстрації нового користувача.

```
export class AuthDto {
  @IsEmail()
  email: string
  @MinLength(6, {
    message: 'Пароль має містити не менше 6 символів'
  })
  @IsString()
  password: string
}
```

Далі було розроблено модулі користувача та товару.

Для зручності у модулі користувача окрім інформації про особу було додано і дані про його активність – вподобані товари. Це дозволило не створювати окремий модуль вподобаних товарів, а зробити вподобані інструменти частиною профілю користувача. Реалізовано це було наступним чином.

```
async byId(id: number, selectObject: Prisma.UserSelect = {}) {
    const user = await this.prisma.user.findUnique({
      where: {
        id
      },
      select: {
        ...returnUserObject,
        favorites: {
          select: {
            id: true,
            name: true,
            price: true,
            images: true,
            slug: true,
            category: {
              select: {
                slug: true
              }
            },
          }
        },
        ...selectObject
      }
    })

    if (!user) {
      throw new Error('Користувача не знайдено!')
    }

    return user
  }
```

Nest.js дозволяє зручно використовувати і об’єкти і властивості в одній функції, як це і зроблено у наведеному фрагменті коду програми. Також, обов’язковою є обробка помилок. Це дозволить уникнути неприємностей при можливих збоях системи, або таких неуважних діях користувачів, як помилка у використанні сторінок сайту.

Наступним етапом розробки веб-застосунку була реалізація товарів, тобто музичних інструментів. Було описано поля та реалізовано належність кожного інструменту до певної категорії музичних інструментів – духові, клавішні, струнні.

Маючи усі основні елементи для роботи системи було реалізовано ряд додаткових елементів, котрі покращують досвід користування користувача системи. До них належать поле пошуку товару, навігаційна панель, категорії товарів для швидкого переміщення по сайту, сортування товарів за ціною, популярністю або новизною.

Такі елементи пошуку є обов'язковими на усіх сайтах-онлайн-магазинах, і їхня реалізація не є складною, адже це просте звернення до бази даних з запитом на отримання товарів за певними критеріями. Для наглядності реалізації сортування інструментів нижче наведено код сортування товарів. Користувачу надано чотири можливих ідеї сортування: найновіші, найстарші, найдорожчі, найдешевші.

```
async getAll(dto: GetAllProductDto = {}) {
  const { sort, searchTerm } = dto

  const prismaSort: Prisma.ProductOrderByWithRelationInput[] = []

  if (sort === EnumProductSort.LOW_PRICE) {
    prismaSort.push({ price: 'asc' })
  } else if (sort === EnumProductSort.HIGH_PRICE) {
    prismaSort.push({ price: 'desc' })
  } else if (sort === EnumProductSort.OLDEST) {
    prismaSort.push({ createdAt: 'asc' })
  } else {
    prismaSort.push({ createdAt: 'desc' })
  }
```

Іншим необхідним програмним елементом є пагінація, тобто виведення лише певної кількості товарів на поточну сторінку веб-сайту. Вирішено відображати на головній сторінці лише двадцять товарів. Це дозолятиме користувачу зручно оглядати фото інструментів та відразу мати можливість порівняти інструмент з іншими. Для реалізації пагінації було використано

звичайний алгоритм – усі отримані товари розділятимуться на двадцять, і це формуватиме кількість сторінок доступних користувачу для перегляду.

3.3 Реалізація клієнтської частини програмної системи

Клієнтська сторона веб-застосунку є не менш важливою за інші, а для звичайного користувача навіть може здаватися найважливішою, оскільки користувачі взаємодіють виключно з інтерфейсом. Тому важливо реалізувати зручний та зрозумілий інтерфейс. Крім того, уже обрані технології мають забезпечити оптимальну швидкість відгуку застосунку на дії користувача.

Спершу потрібно реалізувати функціонал запитів, котрі потім буде відсилати клієнт, і тільки після цього можна починати розробку візуальної клієнтської частини. Важливим для розуміння є те, що Сервіси (Services) у Next.js це набір класів або об’єктів, які відповідають таким же класам або об’єктам на стороні сервера, тобто створеним на базі фреймворку Nest.js. Ця особливість Next.js дозволяє не прописувати запити щоразу у кожному новому компоненті, а просто імпортувати та викликати необхідний запит, оскільки всі вони зберігаються в одному файлі.

Після реалізації запитів клієнтської сторони необхідно реалізувати візуальний інтерфейс та створити зв’язки стану між різними елементами, що знаходяться на одній сторінці (або між різними сторінками).

Як було визначено раніше, спеціальний дизайн для сайту не розроблявся, але за основу дизайну клієнтської сторони було взято популярний сьогодні Material Design. Його основою є простота форм, невелика кількість кольорів та відсутність різких форм елементів.

Спочатку було обрано кольорову палітру дизайну та збережено коди кольорів у змінних, для зручності подальшого використання. Після цього було розроблено головну сторінку онлайн-магазину музичних інструментів, саме на цю сторінку користувач переходитиме, вперше відкривши сайт.

Головна сторінка вміщує багато елементів і складається з хедера – верхньої частини сторінки, на якій розміщено корисні утиліти: логотип магазину, який дозволяє повернутися на головну, поле для пошуку товарів, навігаційний елемент для перегляду вподобаних товарів, навігаційний елемент Корзина – для перегляду товарів, доданих до корзини для подальшого замовлення. Зліва розміщено навігаційний елемент навбар – категорії інструментів для швидкого руху та пошуку. Останнім елементом для покращення досвіду користувачів є Select у правій стороні сторінки, розроблений для сортування товарів на новизною, або ж ціною. Вигляд головної сторінки зображено на рисунку 3.1.

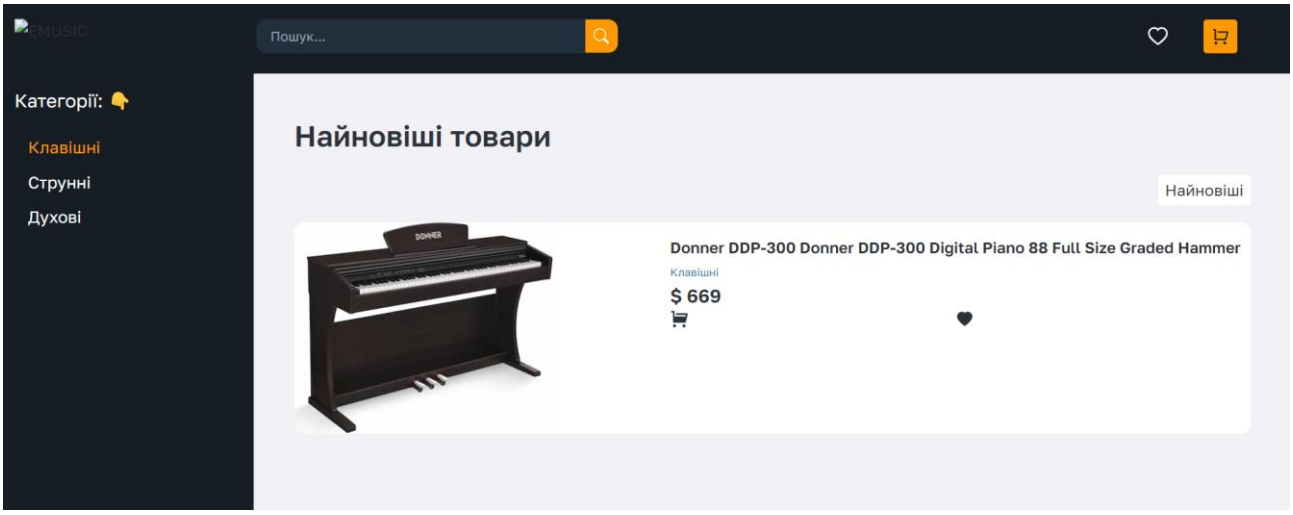


Рисунок 3.1 – Інтерфейс головної сторінки

Для розробки було використано звичайну HTML-розмітку шляхом використання React.JS та CSS-стилі шляхом використання Tailwind CSS, але для зручності та підвищення продуктивності було використано фреймворк Next.js.

Додатковими елементами, які було розроблено на клієнтській стороні додатку є індикатор завантаження сторінки, тобто анімація при очікувані завантаження веб-сторінки, а також спеціальну сторінку – повідомлення при помилку, оскільки стандартні повідомлення про помилку часто нічим не допомагають користувачу системи.

Для отримання доступу до головної сторінки користувачу не потрібно проходити авторизацію в системі.

Отже, наступним кроком була реалізація клієнтської частини сторінки реєстрації та входу. Це окрема сторінка з власним шляхом у системі React Router Dom. Залежно від поточного стану сторінки було реалізовано можливість переході від реєстрації до входу в систему і навпаки. Було використано хуки Реакту для зручної роботи з поточним станом Нижче наведено частину коду для кращого розуміння процесу.

```
const { isLoading } = useAuth()
  const { login, register } = useActions()
  const [type, setType] = useState<'Логін' | 'Реєстрація'>('Логін')
  const {
    register: formRegister,
    handleSubmit,
    formState: { errors },
    reset
  } = useForm<IEmailPassword>({
    mode: 'onChange'
  })

  const onSubmit: SubmitHandler<IEmailPassword> = data => {
    if (type === 'Логін') login(data)
    else register(data)

    reset()
  }


<section className='flex h-screen'>
        <form
          onSubmit={handleSubmit(onSubmit)}
          className='rounded-lg bg-white shadow-sm p-8 m-auto'
        >
          <Heading className='capitalize text-center mb-4'>{type}</Heading>

          {isLoading ? (
            <Loader />
          ) : (
            <>
              <Field
                {...formRegister('email', {
                  required: 'Email is required',
                  pattern: {
                    value: validEmail,
                    message: 'Введіть справжній емейл'
                  }
                })}
```

```
          placeholder='Емейл'
          error={errors.email?.message}
        />
        <Field
          {...formRegister('password', {
            required: 'Введіть пароль',
            minLength: {
              value: 6,
            message: 'Мінімальна довжина паролю 6 символів'
            }
          })}
          type='password'
          placeholder='Пароль'
          error={errors.password?.message}
        />
        <Button type='submit' variant='orange'>
          Продовжити!
        </Button>{' '}
        <div>
          <button
            type='button'
            className='inline-block opacity-20 mt-3 text-sm'
            onClick={() =>
           setType(type === 'Логін' ? 'Реєстрація' : 'Логін')
            }
          >
            {type === 'Логін' ? 'Реєстрація' : 'Логгін'}
          </button>
        </div>
      </>
    )}
  </form>
</section>
```

Після додавання шляху «/auth» до файлу React Router Dom перехід на сторінку авторизації став можливим. Візуальний вигляд сторінки зображено на рисунку 3.2. Сторінка реалізована у вигляді pop-up вікна, але це лише візуально. Насправді це зовсім інша сторінка сайту.

Як реєстрація, так і логін знаходяться на одній і тій же сторінці, за тим самим шляхом, але перехід від реєстрації до логіну реалізовано з використанням тернарного оператора «?» та хука бібліотеки React UseState(). Тобто, спочатку відсліджено поточний стан сторінки, і відносно нього, пропонується перейти на протилежний стан, тобто з входу до реєстрації і навпаки. При цьому змінюється лише логіка, а самі поля вводу та кнопка залишаються незмінними.

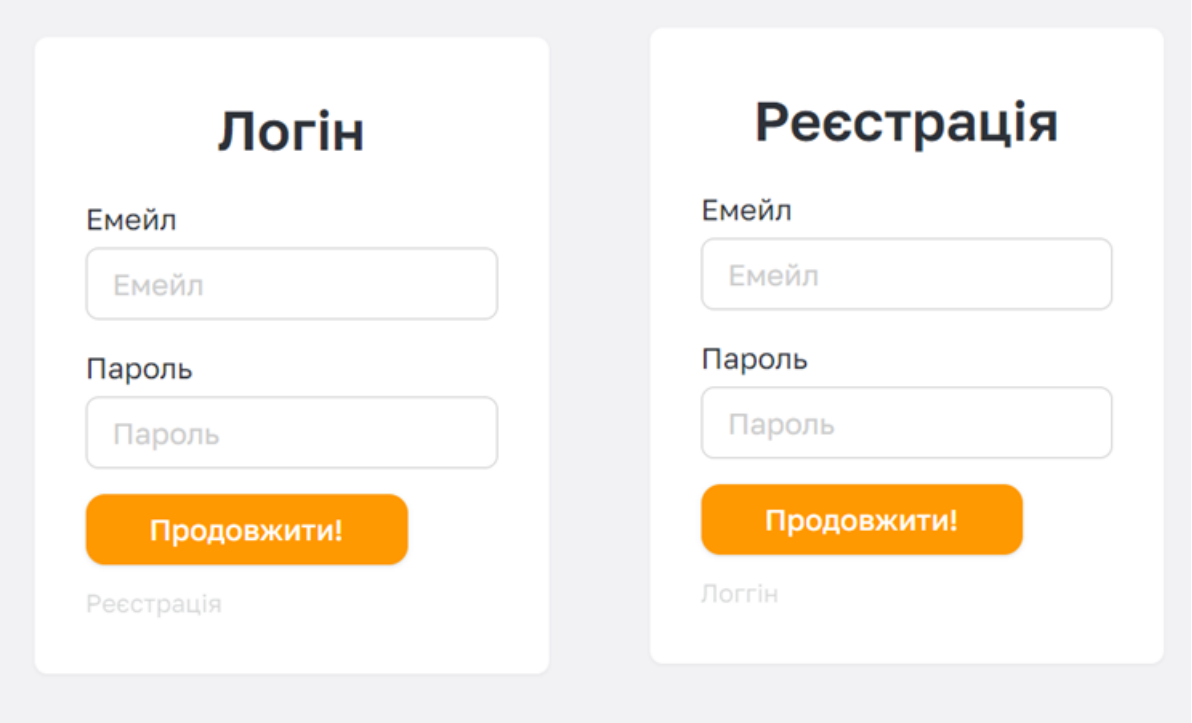


Рисунок 3.2 – Інтерфейс сторінок реєстрації та входу

На цій сторінці також використано можливості валідації. Встановлено перевірку на реальність електронної пошти, тобто на наявність символу «@». При перевірці паролю на безпечність було визначено мінімальну кількість символів, а саме 6 знаків. При невідповідності хоча б одного поля користувач не зможе продовжити і буде бачити повідомлення про помилку та правильний шлях реєстрації, або входу в систему.

Після успішного входу в систему користувач буде бачити окремий елемент в лівому нижньому кутку екрану для виходу з аккаунту і завершення поточної сесії, рисунок 3.3. Якщо користувач захоче завершити свою сесію, то йому потрібно натиснути на даний елемент. Після цього користувач буде переправлений на головну сторінку веб-системи, а його токени доступу будуть анульовані.

Вихід з аккаунту реалізовано з використанням стандартних методів JSON Web Token та засобів Next.JS. Було враховано, що після виходу з аккаунту так має бути очищено кеш браузера клієнта.

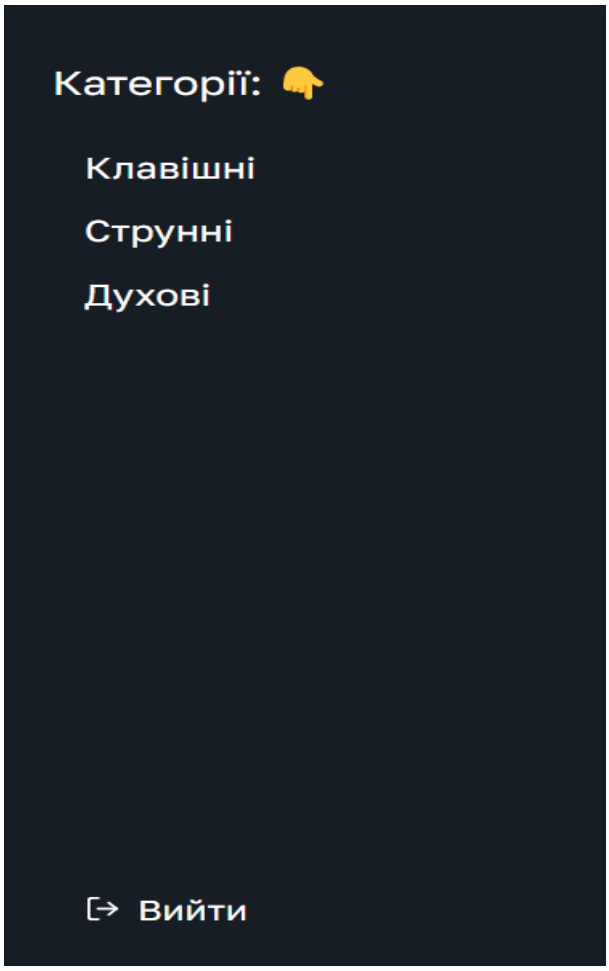


Рисунок 3.3 – Інтерфейс сторінки з елементом для виходу з аккаунту

Наступним елементом для реалізації було обрано сторінку товарів, котрі користувач додав до корзини для подальшого придбання. Зважаючи, що в таких магазинах клієнти переважно купляють лише один товар за раз, було реалізовано невелике вікно корзини. Це зручно та практично. Для реалізації було використано передання ID товару до моделі замовлення, і звідти передання до БД. Таким чином, просто зробивши елемент корзини активним, користувач відправляє запит до серверної частини і звідти отримує список відповідних товарів. Це зручний і швидкий спосіб реалізації таких процесів.

Для наглядності на рисунку 3.4 наведено візуальний інтерфейс пустої корзини. Користувач бачитиме пояснення про те, що спочатку до корзини потрібно додати товар.

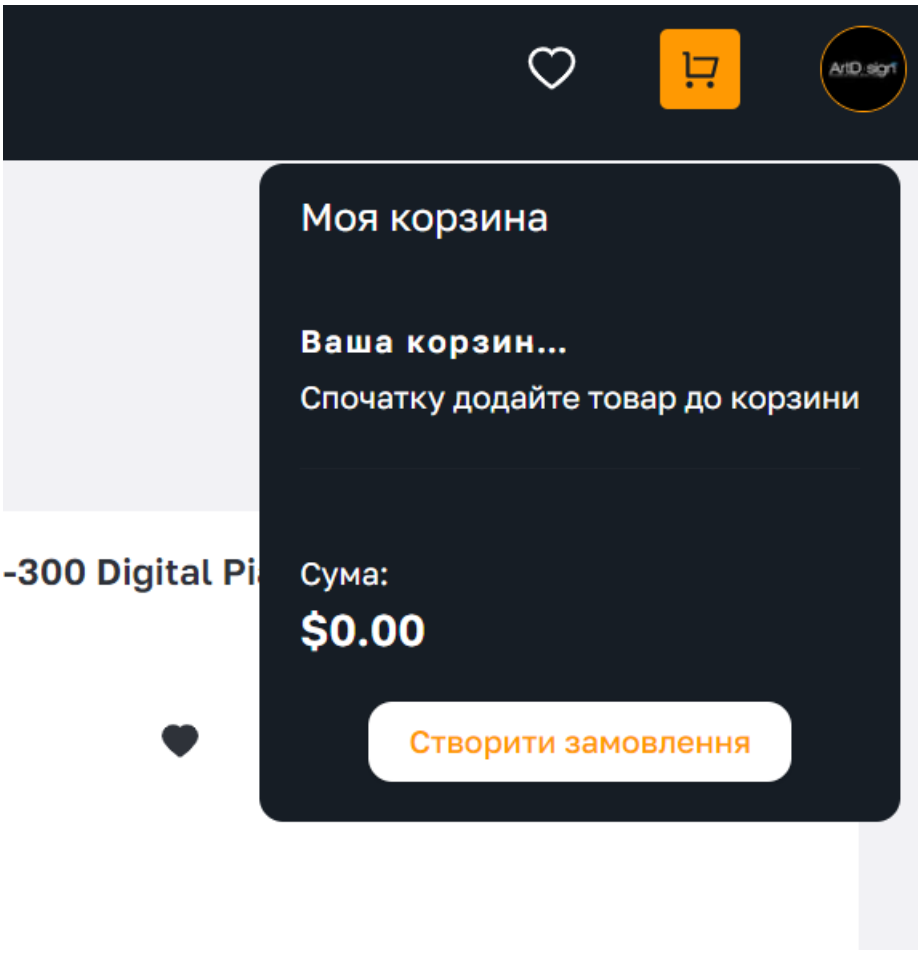


Рисунок 3.4 – Інтерфейс сторінки з елементом пустої корзини

Нижче наведено код клієнтської сторони корзини покупок. Як і раніше, для створення інтерфейсу було використано HTML-розмітку шляхом використання React.JS та CSS-стилі шляхом використання Tailwind CSS, але для зручності та підвищення продуктивності було використано фреймворк Next.js.

```
return (
    <div className='relative' ref={ref}>
      <SquareButton
        Icon={RiShoppingCartLine}
        onClick={() => {
          setIsShow(!isShow)
        }}
        number={items.length}
      />

      <div
        className={cn(
          'absolute top-[4.2rem] w-80 -left-[12.5rem] bg-secondary rounded-xl px-5 py-3 text-sm
menu z-20 text-white',
```

```
          isShow ? 'open-menu' : 'close-menu'
        )}
      >
        <div className='font-normal text-lg mb-5'>Моя корзина</div>

        <div className={styles.cart}>

        <div className={styles.item}>
       </div>

          {items.length ? (
            items.map(item => <CartItem item={item} key={item.id} />)
          ) : (
            <div className='font-light'>Корзина порожня!</div>
          )}

        <div className={styles.footer}>
          <div>{convertPrice(total)}</div>
        </div>
        <div className='text-center'>
          <Button
            variant='white'
            size='sm'
            className='btn-link mt-5 mb-2'
            onClick={() => mutate()}
          >
            Створити замовлення
          </Button>
        </div>
      </div>
    </div>
  )
```

Після цього до корзини було додано товар еклектичного фортепіано. Вигляд корзини після додавання до неї товару для подальшого придбання зображено на рисунку 3.5.

Після цього користувач зможе власне оплатити товар і придбати товар, тобто головна функція веб-застосунку буде виконаною. Після покупки товару користувач зможе повернутися на головну сторінку та продовжити огляд або покупки інших музичних інструментів.

Повний код програми наведено у додатку Б – Лістинг коду застосунку.

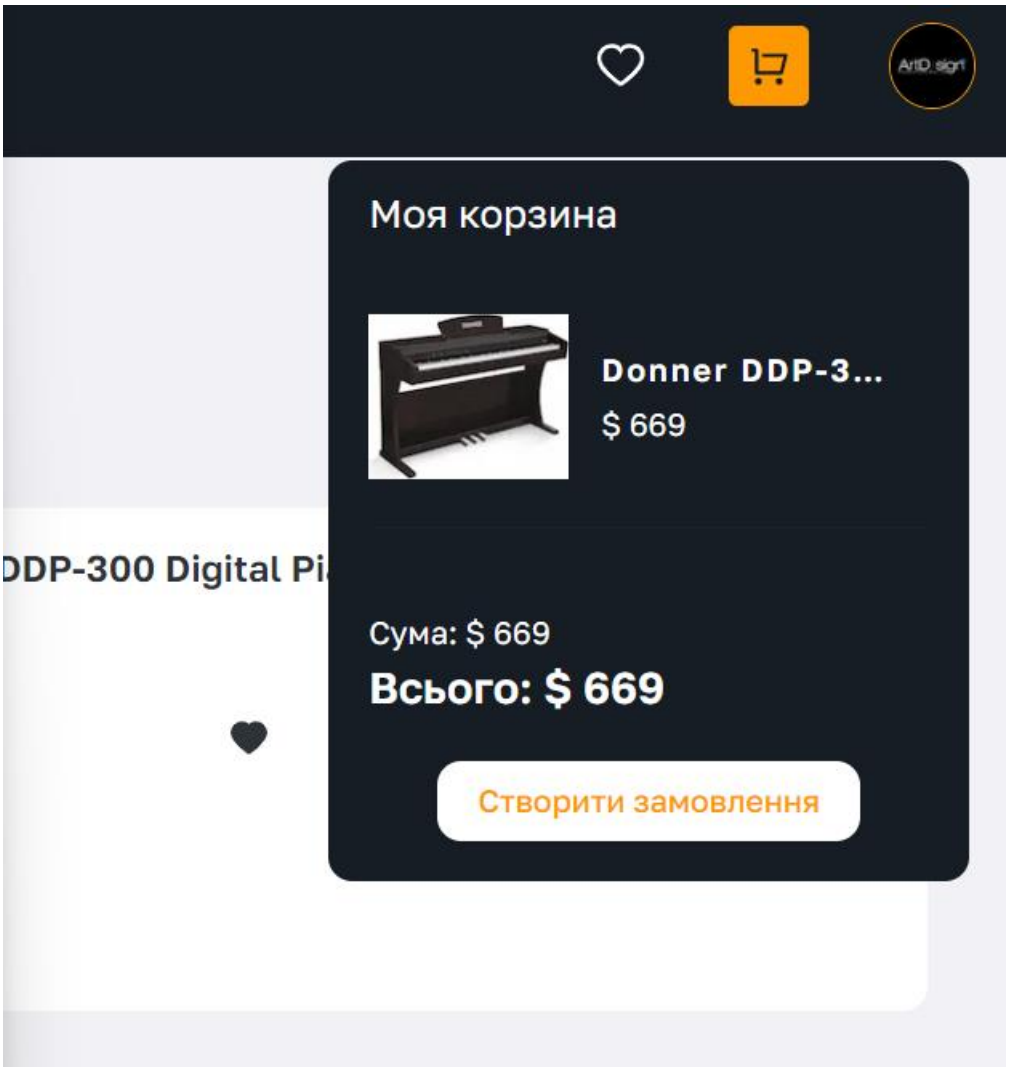


Рисунок 3.5 – Інтерфейс сторінки з елементом корзини з товаром

3.4 Інструкція користувача

Наразі веб-застосунок є безкоштовним та не містить реклами. І відвідати та переглянути вміст онлайн-магазину може кожний бажаючий. Сайт не розміщено на віртуальному хостингу, а просто на локальному сервері, але це не містить суттєвих відмінностей з випадком, якби сайту було повністю розміщено на певній платформі.

Отже, щоб розпочати роботу з сайтом користувачу необхідний комп'ютер або смартфон та доступ до інтернету. Знайшовши сайту у пошуковику або за веб-адресою користувач перейде на головну сторінку. Зайшовши на сайт можна переглядати товари, тобто музичні інструменти.

При потребі зробити замовлення потрібно зареєструватись або увійти в систему, оскільки неавторизований користувач не зможе вподобати товар, або додати його до корзини покупок. Після авторизації є можливість додати товар в корзину. Після цього можна обрати специфікації товару та власне зробити замовлення то провести оплату. Якщо в ході роботи з сайтом виникнуть проблеми, то користувач отримає кастомне повідомлення про помилку, яке повідомить шляхи уникнення неправильних дій зі сторони користувача, також клієнт може зв’язатись з адміністратором та повідомити йому про незручності.

Структура та інтерфейс сайту розроблені з урахуванням усіх типових елементів для веб-застосунків для продажу музичних інструментів, тому навіть не досвідченому користувачу буде легко орієнтуватися між сторінками веб-застосунку та замовлення музичного інструменту не має викликати суттєвого нерозуміння алгоритмів необхідних для цього дій.

3.5 Вимоги до програмно-технічних засобів середовища функціонування програмного продукту

Для коректної роботи сайту потрібно мати наступні мінімальні вимоги для технічних характеристик, якщо ви користуєтесь комп’ютером, на сьогодні більшість пристроїв відповідають даним вимогам :

- 32 (x86) чи 64(x64) -розрядний процесор з тактовою частотою від 1ГГЦ, з підтримкою інструкцій SSE2.
- 4 гігабайти оперативної пам’яті;
- 4 ГБ вільного місця на жорсткому диску;
- Графічна карта з підтримкою DirectX 10;
- Система Windows 7 або новіше;
- Стабільне з’єднання з інтернетом.

Для смартфонів:

- Система Android 6+;
- Система IOS 10+;

3.6 Тестування веб-застосунку

Тестування є обов’язковим етапом для успішного завершення розробки ПЗ. На сьогодні існує багато методологій тестування програмних продуктів, зокрема веб-застосунків. Популярним є виконання тестування за допомогою написання спеціальних автоматизованих тестів. Такий спосіб дозволяє провести якісне тестування та отримати впорядковані результати якості роботи розробленого ПЗ. Перевіреним підходом автоматизованого тестування є написання тестових сценаріїв та їх подальша реалізація. Фактично, спочатку пишеться сценарій тестування, кожний новий сценарій описує певну послідовність дій для досягнення певного результату на веб-сторінці. Після написання, кожний сценарій виконується і чіткій відповідності до описаної послідовності дій користувача веб-застосунку. Така перевірка працездатності ПЗ дозволяє практично перевірити, наскільки сайт добре працює, та чи працює весь функціонал застосунку.

Після визначення тестових сценаріїв можна скористатися інструментами автоматизації тестування. Наприклад, Selenium або Puppeteer, але найкраще також використати Cypress, оскільки середовище дозволяє писати код на JavaScript для тестування додатків на JavaScript. Після цього необхідно просто запустити тести, отримати результати та, за потреби, внести необхідні зміни до коду програмного продукту.

Іншим шляхом тестування є ручне тестування додатків, такий засіб підходить для невеликих програмних систем, або систем, які розробляються однією особою, тому що в такому разі розробник має достатньо знань про кожний модуль системи.

У випадку ручного тестування необхідно створити сценарії тестування та виконати їх для перевірки роботи програмної системи. Для зручності такі сценарії можна оформити у таблиці. Функціональні сценарії тестування веб-застосунку онлайн-магазину музичних інструментів наведено у таблиці 3.1.

Таблиця 3.1 – Тестові сценарії основного функціоналу веб-застосунку

Функціональна вимога	Дії користувача	Очікуваний результат
Реєстрація нового користувача в системі	Користувач переходить на сторінку реєстрації, вводить логін та пароль, нажимає кнопку «Продовжити»	Користувач автоматично повертається на головну сторінку та вже є авторизованим у системі
Додавання інструменту до «Вподобане»	Користувач натискає на позначку корзини на картці товару	Товар відображається у списку «Вподобане»
Додавання інструменту до Кошика	Користувач натискає на позначку кошика на картці товару	Товар відображається у вікні «Кошик» з усіма атрибутами (кількість, ціна, тощо)
Перегляд товару	Користувач натискає на карточку, назву, або фото певного товару	Користувач переходить на іншу сторінку – сторінку відповідного товару
Створення замовлення	Користувач натискає «Створити замовлення» на сторінці кошика товарів	Користувач автоматично переправляється на сторінку оплати товару
Вихід з аккаунту	Користувач натискає «Вийти»	Користувач перенаправляється на головну сторінку як гість

В ході тестування для усіх функціональних вимог було отримано очікуваний результат. Таким чином було протестовано функціональність головних модулів веб-застосунку. При виконанні усіх сценаріїв було отримано очікуваний результат. Отже, тестування системи завершено успішно і система не потребує внесення виправлень.

3.7 Висновки

Отже, при оформлені розділу «Програмна реалізація та тестування», у відповідності з обраними попередньо засобами розробки, було описано процес розробки серверної та клієнтської сторони інтернет-магазину музичних інструментів, а також реалізовано базу даних.

Для реалізації цих завдань було використано різноманітні сучасні технології, які є оптимальними для виконання конкретних функцій. Для роботи з базою даних було використано СКБД PostgreSQL. Для реалізації back-end частини сайту було використано наступні технології: TypeScript – для написання основного коду, фреймворк Nest.JS для ефективнішої та швидшої розробки програмних модулів, Prisma та Beekeeper Studio для зв’язку сервера з БД.

Для реалізації front-end частини магазину музичних інструментів було використано TypeScript та фреймворк React.JS, також для зручності роботи з Nest.JS було використано фреймворк Next.JS.

У розділі наведено фрагменти коду важливих функцій та елементи розробленого інтерфейсу веб-застосунку. Після розробки було проведено тестування функцій ПЗ.

ВИСНОВКИ

Відповідно до поставленої мети було реалізовано наступні завдання:

– проведено загальне комплексне ознайомлення зі сферою інтернет-торгівлі та веб-застосунками для продажу товарів через мережу Інтернет;

– визначено особливості предметної області;

– проведено аналіз технологій, методологій, практик розробки веб-застосунків та вибір оптимальних для подальших використання;

– проведено ознайомлення з сучасними архітектурними та функціональними рішеннями, вибрати ті, які будуть використані у проекті;

– розроблено структуру та функціональну модель проекту;

– реалізовано проект програмним шляхом;

– проведено тестування веб-застосунку;

– підготовлено документацію, що описує процес виконання проекту.

Також було комплексно описано процес розробки архітектури, окремих модулів веб-застосунку інтернет-магазину музичних інструментів, який можна використовувати для перегляду та замовлення музичних інструментів через інтернет.

Цінність отриманих результатів полягає у наявності досконало функціонуючого програмного продукту, який виконує всі функції онлайн-магазину музичних інструментів. Тобто створює можливість клієнтам переглядати та купувати товари через інтернет а власникам магазину дозволяє знаходити нових клієнтів та зручно взаємодіяти з ними.

Для розробки веб-застосунку було обрано платформу Node.JS. Клієнт-серверний розподіл архітектури. Мовою програмування є типізована версія JavaScript, тобто TypeScript. Також було використано фреймворки та бібліотеки для підвищення ефективності коду та збільшення зручності його написання та подальшої потенційної підтримки та доповнення. А саме Nest.JS, Prisma, React.JS, Next.JS.

Потенційними користувачами розроблюваного програмного забезпечення є власники бізнесу, котрі займаються продажем товарів через мережі магазинів, або ті, хто хоче почати продавати товари використовуючи мережу Інтернет.

Корисність розроблюваного програмного продукту для компаній, що займаються дистрибуцією музичних товарів та інструментів полягає у можливостях збільшення аудиторії покупців та об’ємів продажів шляхом масштабування свого бізнесу через інтернет.

Розробка програмного забезпечення та написання супровідної документації є визначним етапом при отриманні освітнього ступеня бакалавра, також, ця робота дозволяє студентові продемонструвати та підтвердити набуті комплекси знань в сфері інженерії програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Сайт E-Bay.com. URL: https://ebay.com/ (дата звернення 30.03.2023).

2. Сайт Amazon.com. URL: https://amazon.com/ (дата звернення 30.03.2023).

3. Сайт gear4music.com. URL: https:// gear4music.com / (дата звернення 30.03.2023).

4. Радельчук Г. І. Наскрізна практична підготовка: програма та методичні вказівки щодо її виконання студентами спеціальності 121 «Інженерія програмного забезпечення» освітнього ступеня «бакалавр : Хмельницький, 2021, 40 с.

5. JavaScript Nest Framework : веб-сайт. URL: https://docs.nestjs.com/pipes (дата звернення: 01.04. 2023).

6. JavaScript React Framework : веб-сайт. URL: https://react.dev/reference/react (дата звернення 09.04. 2023).

7. Гід по роботі з React : веб-сайт. URL: https://uk.javascript.info/modules-dynamic-imports (дата звернення: 19.04. 2023).

8. Бенкс А., Порселло Е. React: Сучасні шаблони для розробки застосунків, 2-ге видання : Фабула, 2022, 320 с.

9. Бейтс, Б., Семендюк, В. Системи керування базами даних : Центр учбової літератури, 2019, 464 с

10. Фрімен Е., Робсон Е. Head First. Програмування на JavaScript : Фабула, 2022, 508 с.

11. Прайс М. C# 7 та .NET Core. Кросс-платформна розробка для професіоналів, 3-є видання : Пітер Прес, 2018, 640 с.

12. JavaScript overview : веб-сайт. URL: https://www.tutorialspoint.com/javascript/javascript_overview.htm (дата звернення 30.04.2023).

13. Роберт Шелдон, Джоффрей Мойє. MySQL: базовий курс Beginning MySQL : Діалектика, 2007, 880 с.

14. Л. Аткінсон, З. Сураскін. PHP5. Бібліотека професіоналу : Вільямс, 2006, 368 с.

15. А Алексейчук І.С. Про технологію створення системи тестування : НМЦВД, 2000, 92с.

16. React + Redux - JWT Authentication Tutorial & Example : веб-сайт. URL: https://jasonwatmore.com/post/2017/12/07/react-redux-jwtauthentication-tutorial-example (дата звернення 12.04. 2023).

17. Офіційна сторінка платформи Techgig : веб-сайт. URL: https://www.techgig.com/practice (дата звернення 26.04. 2023).

18. Douglas Crockford's JavaScript : веб-сайт. URL: http://crockford.com/javascript/ (дата звернення 26.04. 2023).

19. The Software Quality Company : веб-сайт. URL: https://www.tiobe.com/tiobe-index/ (дата звернення 26.04. 2023).

20. Client side technologies : веб-сайт. URL: https://w3techs.com/technologies/overview/client_side_language (дата звернення 26.04. 2023).

21. Особенности NodeJs : веб-сайт. URL: https://senior.ua/articles/pochemu-node-js-osobennosti-i-preimuschestva (дата звернення 27.04. 2023).

22. HTML&CSS : веб-сайт. URL: https://www.w3.org/standards/webdesign/htmlcss (дата звернення 26.04. 2023).

23. Most popular JS stacks : веб-сайт. URL: https://medium.com/datadriveninvestor/most-popular-technology-stack-to-choose-from-full-stack-vs-mean-stack-vs-mern-stack-in-2019-d12c0a17439a (дата звернення 26.04. 2023).

24. Next.js : веб-сайт. URL: Режим доступу: https://www.tutorialspoint.com/nextjs/nextjs_overview.htm (дата звернення 30.04.2023).

25. Больe А. — Learning SQL : веб-сайт. URL: http://shop.oreilly.com/product/9780596007270.do (дата звернення 26.04. 2023).

26. Кейт Джонс. DOM Scripting : Web Design with JavaScript and the Document Object Model. / К. Джонс — Перше, 2005. — 368 с.

27. What is REST? : веб-сайт. URL: http://www.restapitutorial.com/lessons/whatisrest.html. (дата звернення 29.04. 2023).

28. Фаулер, М. Рефакторинг баз даних : Видавничий дім "Київський університет", 2019, 320 с.

29. Д Angular vs React vs Vue : веб-сайт. URL: https://merehead.com/ru/blog/angular-vs-react-vs-vue-2021/ (дата звернення 30.04.2023).

30. Макконнелл С. Ефективна робота з програмним кодом : Київський університет, 2018, 928 с.

31. SQL Keywords Reference : веб-сайт. URL: https://www.w3schools.com/sql/sql_references_keywords.asp (дата звернення 14.04. 2023).

32. JSON + Nest Instruction. : веб-сайт. URL: https://www.w3schools.com/js/js_json_intro.asp (дата звернення 16.04. 2023).

33. JavaScript data types and data structures : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures (дата звернення 17.04. 2023).

34. . JavaScript data types and objects : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#objects (дата звернення 22.04. 2023).

35. JavaScript client-side frameworks : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks (дата звернення 22.04. 2023).

36. Accessability in React : веб-сайт. URL : https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_accessibility (дата звернення 22.04. 2023).

37. Responsible JavaScript: Part III : веб-сайт. URL: https://alistapart.com/article/responsible-javascript-part-3/ (дата звернення 22.04. 2023).

38. React useCallback function. : веб-сайт. URL: https://react.dev/reference/react/useCallback (дата звернення 27.04. 2023).

39. JWT Authentication Tutorial & Example. : веб-сайт. URL: https://jasonwatmore.com/post/jwtauthenticatione (дата звернення 27.04. 2023).

40. Next.js : веб-сайт. URL: Режим доступу: https://www.tutorialspoint.com/nextjs/nextjs_overview.htm (дата звернення 30.04.2023).

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується згідно з процесом розробки веб-додатку «EMusics».

1 Підстава для розробки

Підставою для розробки даного веб-застосунку є «Завдання на кваліфікаційну роботу», затверджене завідувачем кафедри програмного забезпечення.

Найменування розробки: «Веб-застосунок для продажі музичних інструментів».

2 Призначення розробки

2.1 Функціональне призначення

Функціональне призначення – надати клієнтам можливість перегляду та придбання музичних інструментів та обладнання за допомогою персонального комп'ютера чи смартфону. Додаток надасть користувачам можливість оплачувати товари в електронному вигляді та виступатиме посередником між клієнтами та працівниками магазину.

2.2 Експлуатаційне призначення

Веб застосунок має використовуватись звичайними людьми без жодної професійної підготовки за допомогою браузера на ПК, ноутбуках, планшетах та смартфонах за операційними системами Windows, Mac Android, IOS.

3 Вимоги до програмного продукту

3.1 Вимоги до функціональних характеристик

Система має надавати користувачам ряд можливих шляхів використання:

– можливість перегляду товарів та детальної інформації про кожний товар;
– можливість реєстрації та подальшого входу та виходу з системи;
– можливість додання товарів в корзину;
– можливість зв'язку з працівниками магазину при виникненні проблем.

3.2 Вимоги до надійності

Веб-застосунок повинен захищати дані користувачів і передавати особисті дані клієнтів і інформацію про їх спосіб оплати лише в зашифрованому вигляді. Система повинна мати спосіб обробки помилки - відобразити повідомлення про помилку та перенаправити користувача на належно функціонуючу сторінку.

3.3 Вимоги до складу та параметрів технічних засобів та програмної сумісності

Для коректної роботи сайту визначено мінімальні вимоги для технічних характеристик комп'ютера :

- 32 (x86) чи 64(x64) -розрядний процесор;
- 4 гігабайти RAM;
- 4 ГБ вільного місця на диску;
- Відеокарта, що підтримує технологію DirectX 10;
- Windows 7+;
- Інтернет.

Вимоги до смартфонів :

- Android 4;
- IOS 10;
- 2 гігабайти RAM.

3.4 Вимоги до транспортування та зберігання

Програмний продукт може поставлятися через хмарні сховища та фізичні носії. Документація надається у двох варіантах – електронному та паперовому.

Умови експлуатації не виходять за межі умов експлуатації технічних засобів.

3.5 Спеціальні вимоги

Для коректної функціональності ПЗ спеціальні вимоги відсутні.

4 Вимоги до програмної документації

Замовнику має бути надано текст та опис програми, технічне завдання. Також замовнику має бути надано власне програма з інструкцією користувача.

5 Техніко-економічне обґрунтування

Продуктивний економічний ефект при використанні ПЗ полягає у оптимізації наступних процесів:

– зменшення кількості працівників магазину, що працюють з клієнтами;

– можливість знаходити цільову аудиторію не значно більшій території, що дозволить масштабувати бізнес;

– спрощенні ведення документації та оплати товарів.

6 Стадії розробки

Стадія та період	Етап	Зміст
Технічне завдання, січень	Обґрунтування необхідності розробки	Вибір теми ДП, коротка характеристика ПЗ, вимоги до ПЗ, стадії розробки ПЗ
Ескізний проект, січень-лютий	Створення ескізного проекту	Початкова розробка структури системи, вибір середовища та технологій
Технічний проект, лютий-березень	Створення технічного проекту	Уточнення структури та вибір остаточних технологій розробки, проектування дизайну ПЗ
Робочий проект, квітень	Створення робочого проекту	Програмна реалізація ПЗ, створення застосунку, що відповідає усім нормам
Розробка документації, травень	Створення документації	Створення супровідної документації для проекту, розробка ілюстративних матеріалів
Тестування, травень,	Проведення тестування	Проведення тестування модулів та системи цілком, виправлення виявлених невідповідностей ПЗ
Впровадження, червень	Отримання документів від курівників	Підготовка документації, необхідної для затвердження роботи: відгуки, рецензії.
Захист, червень	Розробка документації	Підготовка до захисту та захист КвР

7 Порядок контролю та приймання

Контроль і приймання здійснюється юзерами веб-застосунку та керівниками проекту. При цьому має бути проведено перевірку функціональності ПЗ та перевірку відповідності наданої документації.

ДОДАТОК Б

(обов'язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

Код класу `AuthController`

```
import {
   Body,
   Controller,
   HttpCode,
   Post,
   UsePipes,
   ValidationPipe
} from '@nestjs/common'
import { AuthService } from './auth.service'
import { AuthDto } from './dto/auth.dto'
import { RefreshTokenDto } from './dto/refresh-token.dto'

@Controller('auth')
export class AuthController {
   constructor(private readonly authService: AuthService) {}

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Post('login')
   async login(@Body() dto: AuthDto) {
      return this.authService.login(dto)
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Post('login/access-token')
   async getNewTokens(@Body() dto: RefreshTokenDto) {
      return this.authService.getNewTokens(dto.refreshToken)
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Post('register')
   async register(@Body() dto: AuthDto) {
      return this.authService.register(dto)
   }
}
```

Код класу `AuthService`

```
import { faker } from '@faker-js/faker'
import {
```

```
   BadRequestException,
   Injectable,
   NotFoundException,
   UnauthorizedException
} from '@nestjs/common'
import { JwtService } from '@nestjs/jwt'
import { User } from '@prisma/client'
import { hash, verify } from 'argon2'
import { PrismaService } from 'src/prisma.service'
import { AuthDto } from './dto/auth.dto'

@Injectable()
export class AuthService {
   constructor(private prisma: PrismaService, private jwt: JwtService) {}

   async login(dto: AuthDto) {
      const user = await this.validateUser(dto)
      const tokens = await this.issueTokens(user.id)

      return {
         user: this.returnUserFields(user),
         ...tokens
      }
   }

   async getNewTokens(refreshToken: string) {
      const result = await this.jwt.verifyAsync(refreshToken)
      if (!result) throw new UnauthorizedException('Invalid refresh token')

      const user = await this.prisma.user.findUnique({
         where: {
            id: result.id
         }
      })

      const tokens = await this.issueTokens(user.id)

      return {
         user: this.returnUserFields(user),
         ...tokens
      }
   }

   async register(dto: AuthDto) {
      const oldUser = await this.prisma.user.findUnique({
         where: {
            email: dto.email
         }
      })

      if (oldUser) throw new BadRequestException('User already exists')
```

```
    const user = await this.prisma.user.create({
      data: {
        email: dto.email,
        name: faker.name.firstName(),
        avatarPath: faker.image.avatar(),
        phone: faker.phone.number('+7 (###) ###-##-##'),
        password: await hash(dto.password)
      }
    })

    const tokens = await this.issueTokens(user.id)

    return {
      user: this.returnUserFields(user),
      ...tokens
    }
  }

  private async issueTokens(userId: number) {
    const data = { id: userId }

    const accessToken = this.jwt.sign(data, {
      expiresIn: '1h'
    })

    const refreshToken = this.jwt.sign(data, {
      expiresIn: '7d'
    })

    return { accessToken, refreshToken }
  }

  private returnUserFields(user: User) {
    return {
      id: user.id,
      email: user.email
    }
  }

  private async validateUser(dto: AuthDto) {
    const user = await this.prisma.user.findUnique({
      where: {
        email: dto.email
      }
    })

    if (!user) throw new NotFoundException('User not found')

    const isValid = await verify(user.password, dto.password)

    if (!isValid) throw new UnauthorizedException('Invalid password')
```

```
      return user
   }
}
```

Код класу `CategoryController`

```
import {
   Body,
   Controller,
   Delete,
   Get,
   HttpCode,
   Param,
   Post,
   Put,
   UsePipes,
   ValidationPipe
} from '@nestjs/common'
import { Auth } from 'src/auth/decorators/auth.decorator'
import { CategoryDto } from './category.dto'
import { CategoryService } from './category.service'

@Controller('categories')
export class CategoryController {
   constructor(private readonly categoryService: CategoryService) {}

   @Get()
   async getAll() {
      return this.categoryService.getAll()
   }

   @Get('by-slug/:slug')
   async getBySlug(@Param('slug') slug: string) {
      return this.categoryService.bySlug(slug)
   }

   @Get(':id')
   @Auth()
   async getById(@Param('id') id: string) {
      return this.categoryService.byId(+id)
   }

   @HttpCode(200)
   @Auth()
   @Post()
   async create() {
      return this.categoryService.create()
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Auth()
```

```
   @Put(':id')
   async update(@Param('id') id: string, @Body() dto: CategoryDto) {
      return this.categoryService.update(+id, dto)
   }

   @HttpCode(200)
   @Auth()
   @Delete(':id')
   async delete(@Param('id') id: string) {
      return this.categoryService.delete(+id)
   }
}
```

Код класу `CategoryService`

```
import { Injectable, NotFoundException } from '@nestjs/common'
import { PrismaService } from 'src/prisma.service'
import { generateSlug } from 'src/utils/generate-slug'
import { CategoryDto } from './category.dto'
import { returnCategoryObject } from './return-category.object'

@Injectable()
export class CategoryService {
   constructor(private prisma: PrismaService) {}

   async byId(id: number) {
      const category = await this.prisma.category.findUnique({
         where: {
            id
         },
         select: returnCategoryObject
      })

      if (!category) {
         throw new Error('Category not found')
      }

      return category
   }

   async bySlug(slug: string) {
      const category = await this.prisma.category.findUnique({
         where: {
            slug
         },
         select: returnCategoryObject
      })

      if (!category) {
         throw new NotFoundException('Category not found')
      }
```

```
      return category
   }

   async getAll() {
      return this.prisma.category.findMany({
         select: returnCategoryObject
      })
   }

   async create() {
      return this.prisma.category.create({
         data: {
            name: '',
            slug: ''
         }
      })
   }

   async update(id: number, dto: CategoryDto) {
      return this.prisma.category.update({
         where: {
            id
         },
         data: {
            name: dto.name,
            slug: generateSlug(dto.name)
         }
      })
   }

   async delete(id: number) {
      return this.prisma.category.delete({
         where: {
            id
         }
      })
   }
}
```

Код класу `OrderController`

```
import {
   Body,
   Controller,
   Get,
   HttpCode,
   Post,
   UsePipes,
   ValidationPipe
} from '@nestjs/common'
import { Auth } from 'src/auth/decorators/auth.decorator'
import { CurrentUser } from 'src/auth/decorators/user.decorator'
```

```
import { OrderDto } from './order.dto'
import { OrderService } from './order.service'
import { PaymentStatusDto } from './payment-status.dto'

@Controller('orders')
export class OrderController {
   constructor(private readonly orderService: OrderService) {}

   @Get()
   @Auth()
   getAll(@CurrentUser('id') userId: number) {
      return this.orderService.getAll(userId)
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Post()
   @Auth()
   placeOrder(@Body() dto: OrderDto, @CurrentUser('id') userId: number) {
      return this.orderService.placeOrder(dto, userId)
   }

   @HttpCode(200)
   @Post('status')
   async updateStatus(@Body() dto: PaymentStatusDto) {
      return this.orderService.updateStatus(dto)
   }
}
```

Код класу `OrderService`

```
import { Injectable } from '@nestjs/common'
import { EnumOrderStatus } from '@prisma/client'
import { PrismaService } from 'src/prisma.service'
import { productReturnObject } from 'src/product/return-product.object'
import * as YooKassa from 'yookassa'
import { OrderDto } from './order.dto'
import { PaymentStatusDto } from './payment-status.dto'

const yooKassa = new YooKassa({
   shopId: process.env['SHOP_ID'],
   secretKey: process.env['PAYMENT_TOKEN']
})

@Injectable()
export class OrderService {
   constructor(private prisma: PrismaService) {}

   async getAll(userId: number) {
      return this.prisma.order.findMany({
         where: {
            userId
```

```
      },
      orderBy: {
         createdAt: 'desc'
      },
      include: {
         items: {
            include: {
               product: {
                  select: productReturnObject
               }
            }
         }
      }
   })
}

async placeOrder(dto: OrderDto, userId: number) {
   const total = dto.items.reduce((acc, item) => {
      return acc + item.price * item.quantity
   }, 0)

   const order = await this.prisma.order.create({
      data: {
         status: dto.status,
         items: {
            create: dto.items
         },
         total,
         user: {
            connect: {
               id: userId
            }
         }
      }
   })

   const payment = await yooKassa.createPayment({
      amount: {
         value: total.toFixed(2),
         currency: 'RUB'
      },
      payment_method_data: {
         /* CHANGE */
         type: 'bank_card'
      },
      confirmation: {
         type: 'redirect',
         /* CHANGE */
         return_url: 'http://localhost:3000/thanks'
      },
      /* CHANGE */
      description: `Order #${order.id}`
```

```
      })

      return payment
   }

   async updateStatus(dto: PaymentStatusDto) {
      if (dto.event === 'payment.waiting_for_capture') {
         const payment = await yooKassa.capturePayment(dto.object.id)
         return payment
      }

      if (dto.event === 'payment.succeeded') {
         const orderId = Number(dto.object.description.split('#')[1])

         await this.prisma.order.update({
            where: {
               id: orderId
            },
            data: {
               status: EnumOrderStatus.PAYED
            }
         })

         return true
      }

      return true
   }
}
```

Код класу `ProductController`

```
import {
   Body,
   Controller,
   Delete,
   Get,
   HttpCode,
   Param,
   Post,
   Put,
   Query,
   UsePipes,
   ValidationPipe
} from '@nestjs/common'
import { Auth } from 'src/auth/decorators/auth.decorator'
import { GetAllProductDto } from './dto/get-all.product.dto'
import { ProductDto } from './dto/product.dto'
import { ProductService } from './product.service'

@Controller('products')
```

```
export class ProductController {
   constructor(private readonly productService: ProductService) {}

   @UsePipes(new ValidationPipe())
   @Get()
   async getAll(@Query() queryDto: GetAllProductDto) {
      return this.productService.getAll(queryDto)
   }

   @Get('similar/:id')
   async getSimilar(@Param('id') id: string) {
      return this.productService.getSimilar(+id)
   }

   @Get('by-slug/:slug')
   async getProductBySlug(@Param('slug') slug: string) {
      return this.productService.bySlug(slug)
   }

   @Get('by-category/:categorySlug')
   async getProductsByCategory(@Param('categorySlug') categorySlug: string) {
      return this.productService.byCategory(categorySlug)
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Auth()
   @Post()
   async createProduct() {
      return this.productService.create()
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Put(':id')
   @Auth()
   async updateProduct(@Param('id') id: string, @Body() dto: ProductDto) {
      return this.productService.update(+id, dto)
   }

   @HttpCode(200)
   @Delete(':id')
   @Auth()
   async deleteProduct(@Param('id') id: string) {
      return this.productService.delete(+id)
   }

   @Get(':id')
   @Auth()
   async getProduct(@Param('id') id: string) {
      return this.productService.byId(+id)
   }
```

```
}
```

Код класу ProductService

```
import { Injectable, NotFoundException } from '@nestjs/common'
import { Prisma } from '@prisma/client'
import { PaginationService } from 'src/pagination/pagination.service'
import { PrismaService } from 'src/prisma.service'
import { generateSlug } from 'src/utils/generate-slug'
import { EnumProductSort, GetAllProductDto } from './dto/get-all.product.dto'
import { ProductDto } from './dto/product.dto'
import {
   productReturnObject,
   productReturnObjectFullest
} from './return-product.object'

@Injectable()
export class ProductService {
   constructor(
      private prisma: PrismaService,
      private paginationService: PaginationService
   ) {}

   async getAll(dto: GetAllProductDto = {}) {
      const { sort, searchTerm } = dto

      const prismaSort: Prisma.ProductOrderByWithRelationInput[] = []

      if (sort === EnumProductSort.LOW_PRICE) {
         prismaSort.push({ price: 'asc' })
      } else if (sort === EnumProductSort.HIGH_PRICE) {
         prismaSort.push({ price: 'desc' })
      } else if (sort === EnumProductSort.OLDEST) {
         prismaSort.push({ createdAt: 'asc' })
      } else {
         prismaSort.push({ createdAt: 'desc' })
      }

      const prismaSearchTermFilter: Prisma.ProductWhereInput = searchTerm
         ? {
               OR: [
                  {
                     category: {
                        name: {
                           contains: searchTerm,
                           mode: 'insensitive'
                        }
                     }
                  },
                  {
                     name: {
                        contains: searchTerm,
```

```
                    mode: 'insensitive'
                }
            },
            {
                description: {
                    contains: searchTerm,
                    mode: 'insensitive'
                }
            }
        ]
    }
    : {}

  const { perPage, skip } = this.paginationService.getPagination(dto)

  const products = await this.prisma.product.findMany({
    where: prismaSearchTermFilter,
    orderBy: prismaSort,
    skip,
    take: perPage,
    select: productReturnObject
  })

  return {
    products,
    length: await this.prisma.product.count({
      where: prismaSearchTermFilter
    })
  }
}

async byId(id: number) {
  const product = await this.prisma.product.findUnique({
    where: {
      id
    },
    select: productReturnObjectFullest
  })

  if (!product) throw new NotFoundException('Product not found!')
  return product
}

async bySlug(slug: string) {
  const product = await this.prisma.product.findUnique({
    where: {
      slug
    },
    select: productReturnObjectFullest
  })

  if (!product) throw new NotFoundException('Product not found!')
```

```
    return product
  }

  async byCategory(categorySlug: string) {
    const products = await this.prisma.product.findMany({
      where: {
        category: {
          slug: categorySlug
        }
      },
      select: productReturnObjectFullest
    })

    if (!products) throw new NotFoundException('Products not found!')
    return products
  }

  async getSimilar(id: number) {
    const currentProduct = await this.byId(id)

    if (!currentProduct)
      throw new NotFoundException('Current product not found!')

    const products = await this.prisma.product.findMany({
      where: {
        category: {
          name: currentProduct.category.name
        },
        NOT: {
          id: currentProduct.id
        }
      },
      orderBy: {
        createdAt: 'desc'
      },
      select: productReturnObject
    })

    return products
  }

  async create() {
    const product = await this.prisma.product.create({
      data: {
        description: '',
        name: '',
        price: 0,
        slug: ''
      }
    })

    return product.id
```

```
   }

   async update(id: number, dto: ProductDto) {
      const { description, images, price, name, categoryId } = dto

      return this.prisma.product.update({
         where: {
            id
         },
         data: {
            description,
            images,
            price,
            name,
            slug: generateSlug(name),
            category: {
               connect: {
                  id: categoryId
               }
            }
         }
      })
   }

   async delete(id: number) {
      return this.prisma.product.delete({ where: { id } })
   }
}
```

Код класу `UserController`

```
import {
   Body,
   Controller,
   Get,
   HttpCode,
   Param,
   Patch,
   Put,
   UsePipes,
   ValidationPipe
} from '@nestjs/common'
import { Auth } from 'src/auth/decorators/auth.decorator'
import { CurrentUser } from 'src/auth/decorators/user.decorator'
import { UserDto } from './user.dto'
import { UserService } from './user.service'

@Controller('users')
export class UserController {
   constructor(private readonly userService: UserService) {}

   @Get('profile')
```

```
   @Auth()
   async getProfile(@CurrentUser('id') id: number) {
      return this.userService.byId(id)
   }

   @UsePipes(new ValidationPipe())
   @HttpCode(200)
   @Auth()
   @Put('profile')
   async getNewTokens(@CurrentUser('id') id: number, @Body() dto: UserDto) {
      return this.userService.updateProfile(id, dto)
   }

   @HttpCode(200)
   @Auth()
   @Patch('profile/favorites/:productId')
   async toggleFavorite(
      @CurrentUser('id') id: number,
      @Param('productId') productId: string
   ) {
      return this.userService.toggleFavorite(id, +productId)
   }
}
```

Код класу `UserService`

```
import {
   BadRequestException,
   Injectable,
   NotFoundException
} from '@nestjs/common'
import { Prisma } from '@prisma/client'
import { hash } from 'argon2'
import { PrismaService } from 'src/prisma.service'
import { returnUserObject } from './return-user.object'
import { UserDto } from './user.dto'

@Injectable()
export class UserService {
   constructor(private prisma: PrismaService) {}

   async byId(id: number, selectObject: Prisma.UserSelect = {}) {
      const user = await this.prisma.user.findUnique({
         where: {
            id
         },
         select: {
            ...returnUserObject,
            favorites: {
               select: {
                  id: true,
```

```
                name: true,
                price: true,
                images: true,
                slug: true,
                category: {
                   select: {
                      slug: true
                   }
                },
                reviews: true
             }
          },
          ...selectObject
       }
    })

    if (!user) {
       throw new Error('User not found')
    }

    return user
 }

 async updateProfile(id: number, dto: UserDto) {
    const isSameUser = await this.prisma.user.findUnique({
       where: { email: dto.email }
    })

    if (isSameUser && id !== isSameUser.id)
       throw new BadRequestException('Email already in use')

    const user = await this.byId(id)

    return this.prisma.user.update({
       where: {
          id
       },
       data: {
          email: dto.email,
          name: dto.name,
          avatarPath: dto.avatarPath,
          phone: dto.phone,
          password: dto.password ? await hash(dto.password) : user.password
       }
    })
 }

 async toggleFavorite(userId: number, productId: number) {
    const user = await this.byId(userId)

    if (!user) throw new NotFoundException('User not found!')
```

```
      const isExists = user.favorites.some(product => product.id === productId)

      await this.prisma.user.update({
         where: {
            id: user.id
         },
         data: {
            favorites: {
               [isExists ? 'disconnect' : 'connect']: {
                  id: productId
               }
            }
         }
      })

      return { message: 'Success' }
   }
}
```

Код класу `api. interceptors`

```
import axios from 'axios'

import { errorCatch, getContentType } from './api.helper'
import { getAccessToken, removeFromStorage } from '@/services/auth/auth.helper'
import { AuthService } from '@/services/auth/auth.service'

const axiosOptions = {
   baseURL: process.env.SERVER_URL,
   headers: getContentType()
}

export const axiosClassic = axios.create(axiosOptions)

export const instance = axios.create(axiosOptions)

instance.interceptors.request.use(config => {
   const accessToken = getAccessToken()

   if (config && config.headers && accessToken)
      config.headers.Authorization = `Bearer ${accessToken}`

   return config
})

instance.interceptors.response.use(
   config => config,
   async error => {
      const originalRequest = error.config

      if (
         (error?.response?.status === 401 ||
```

```
            errorCatch(error) === 'jwt expired' ||
            errorCatch(error) === 'jwt must be provided') &&
         error.config &&
         !error.config._isRetry
      ) {
         originalRequest._isRetry = true
         try {
            await AuthService.getNewTokens()
            return instance.request(originalRequest)
         } catch (error) {
            if (errorCatch(error) === 'jwt expired') removeFromStorage()
         }
      }

      throw error
   }
)
```

Код класу `Auth`

```
import { FC, useState } from 'react'
import { SubmitHandler, useForm } from 'react-hook-form'

import Heading from '@/ui/Heading'
import Loader from '@/ui/Loader'
import Meta from '@/ui/Meta'
import Button from '@/ui/button/Button'
import Field from '@/ui/input/Field'

import { IEmailPassword } from '@/store/user/user.interface'

import { useActions } from '@/hooks/useActions'
import { useAuth } from '@/hooks/useAuth'

import { useAuthRedirect } from './useAuthRedirect'
import { validEmail } from './valid-email'

const Auth: FC = () => {
   useAuthRedirect()

   const { isLoading } = useAuth()

   const { login, register } = useActions()

   const [type, setType] = useState<'Логін' | 'Реєстрація'>('Логін')

   const {
      register: formRegister,
      handleSubmit,
      formState: { errors },
      reset
   } = useForm<IEmailPassword>({
```

```
   mode: 'onChange'
})

const onSubmit: SubmitHandler<IEmailPassword> = data => {
   if (type === 'Логін') login(data)
   else register(data)

   reset()
}

return (
   <Meta title='Auth'>
      <section className='flex h-screen'>
         <form
            onSubmit={handleSubmit(onSubmit)}
            className='rounded-lg bg-white shadow-sm p-8 m-auto'
         >
            <Heading className='capitalize text-center mb-4'>{type}</Heading>

            {isLoading ? (
               <Loader />
            ) : (
               <>
                  <Field
                     {...formRegister('email', {
                        required: 'Email is required',
                        pattern: {
                           value: validEmail,
                           message: 'Введіть справжній емейл'
                        }
                     })}
                     placeholder='Емейл'
                     error={errors.email?.message}
                  />
                  <Field
                     {...formRegister('password', {
                        required: 'Password is required',
                        minLength: {
                           value: 6,
                           message: 'Мінімальна довжина пароля 6 символів'
                        }
                     })}
                     type='password'
                     placeholder='Пароль'
                     error={errors.password?.message}
                  />
                  <Button type='submit' variant='orange'>
                     Продовжити!
                  </Button>{' '}
                  <div>
                     <button
                        type='button'
```

```
                        className='inline-block opacity-20 mt-3 text-sm'
                        onClick={() =>
                           setType(type === 'Логін' ? 'Реєстрація' : 'Логін')
                        }
                     >
                        {type === 'Логін' ? 'Реєстрація' : 'Логін'}
                     </button>
                  </div>
               </>
            )}
         </form>
      </section>
   </Meta>
  )
}

export default Auth
```

Код класу `AuthProvider`

```
import Cookies from 'js-cookie'
import dynamic from 'next/dynamic'
import { useRouter } from 'next/router'
import { FC, PropsWithChildren, useEffect } from 'react'

import { useActions } from '@/hooks/useActions'
import { useAuth } from '@/hooks/useAuth'

import { TypeComponentAuthFields } from './auth-page.types'
import { getAccessToken } from '@/services/auth/auth.helper'

const DynamicCheckRole = dynamic(() => import('./CheckRole'), { ssr: false })

const AuthProvider: FC<PropsWithChildren<TypeComponentAuthFields>> = ({
  Component: { isOnlyUser },
  children
}) => {
  const { user } = useAuth()
  const { checkAuth, logout } = useActions()

  const { pathname } = useRouter()

  useEffect(() => {
    const accessToken = getAccessToken()
    if (accessToken) checkAuth()
  }, [])

  useEffect(() => {
    const refreshToken = Cookies.get('refreshToken')
    if (!refreshToken && user) logout()
  }, [pathname])
```

```
   return isOnlyUser ? (
      <DynamicCheckRole Component={{ isOnlyUser }} children={children} />
   ) : (
      <>{children}</>
   )
}

export default AuthProvider
```

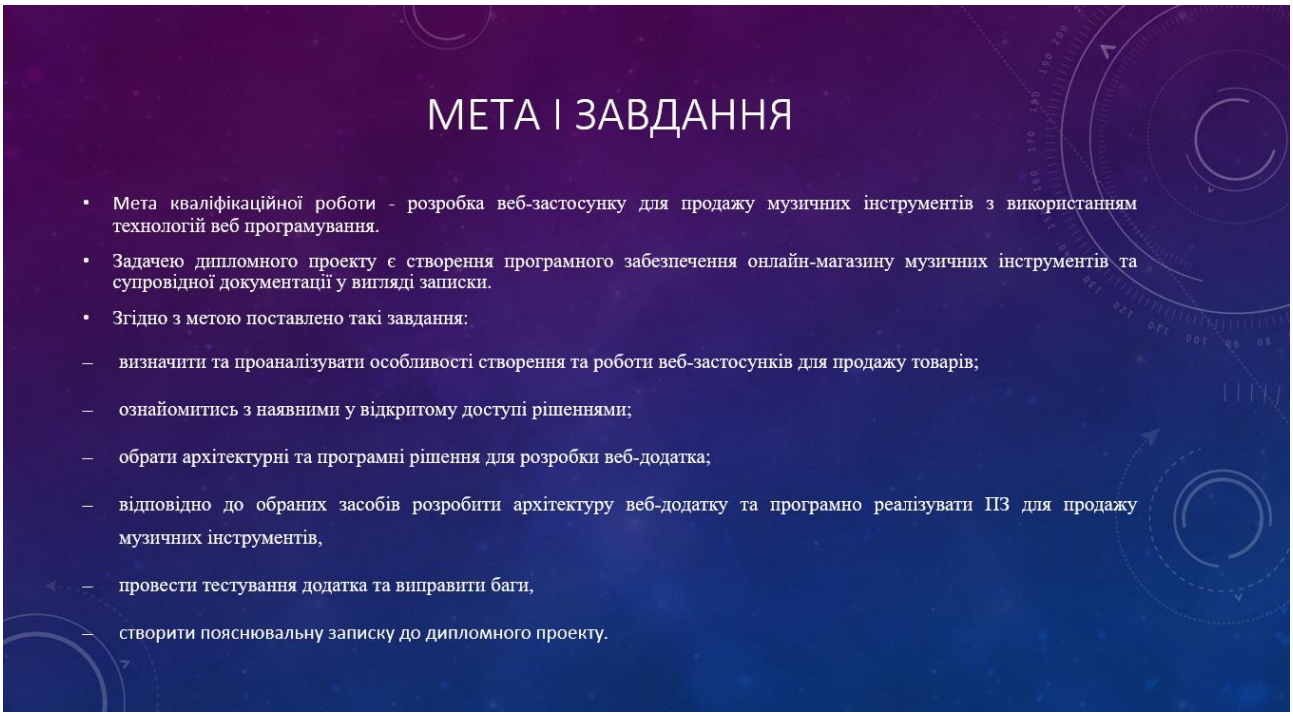
Код класу `cartSlice`

```
import { PayloadAction, createSlice } from '@reduxjs/toolkit'

import {
   IAddToCartPayload,
   ICartInitialState,
   IChangeQuantityPayload
} from './cart.types'

const initialState: ICartInitialState = {
   items: []
}

export const cartSlice = createSlice({
   name: 'cart',
   initialState,
   reducers: {
      addToCart: (state, action: PayloadAction<IAddToCartPayload>) => {
         const isExist = state.items.some(
            item => item.product.id === action.payload.product.id
         )

         if (!isExist)
            state.items.push({ ...action.payload, id: state.items.length })
      },
      removeFromCart: (state, action: PayloadAction<{ id: number }>) => {
         state.items = state.items.filter(item => item.id !== action.payload.id)
      },
      changeQuantity: (state, action: PayloadAction<IChangeQuantityPayload>) => {
         const { id, type } = action.payload
         const item = state.items.find(item => item.id === id)
         if (item) type === 'plus' ? item.quantity++ : item.quantity--
      },
      reset: state => {
         state.items = []
      }
   }
})
```

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

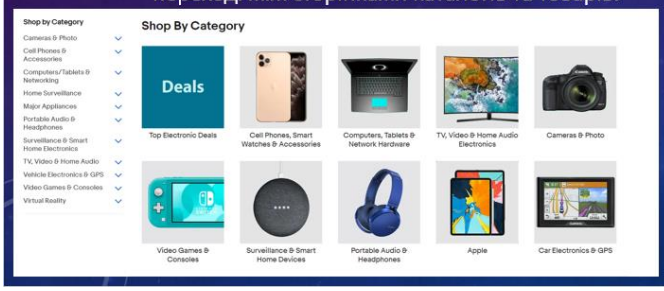


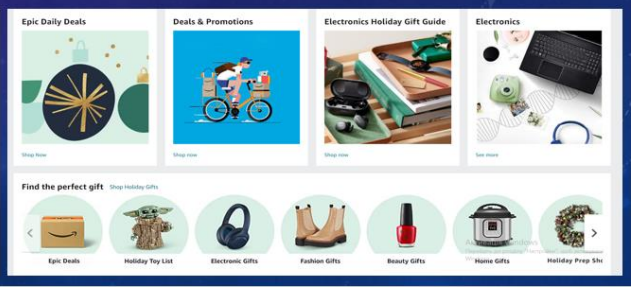
АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

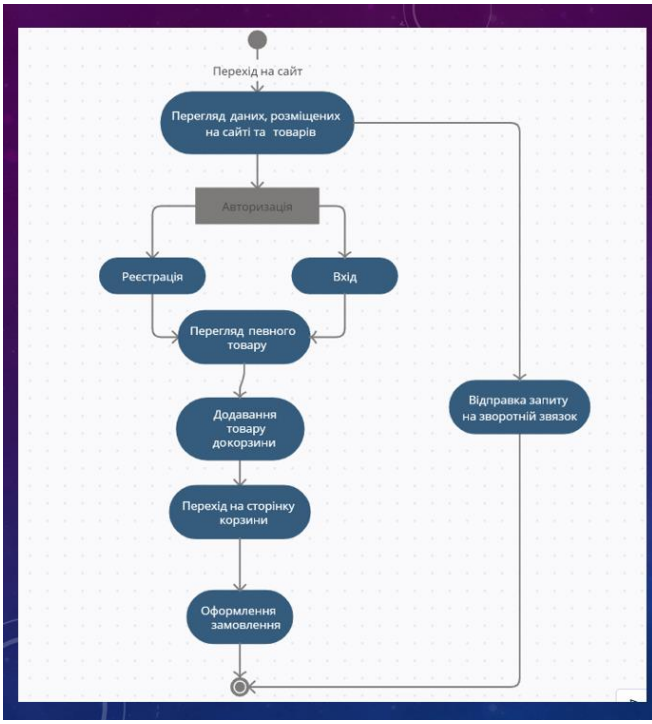
- Факторами актуальності роботи є :
- Станом на сьогодні реальним магазинам необхідно мати інтернет-версію – це дозволить збільшити об’єм продажу товарів та привабити нових клієнтів;
- Постійна потреба магазинів збільшувати рівень продажу товарів – для цього вони можуть використати розроблений онлайн-магазин;
- Зручність введення власного магазину на онлайн-платформі – це економить час як продавців так і клієнтів, крім того, дозволяє власникам збільшити прибутки

ОГЛЯД РІШЕНЬ

- В ході підготовки до проектування та розробки програмного продукту було проведено огляд наявних у вільному доступі рішень. Було розглянуто наймастабніший онлайн магазин Amazon та український магазин музичних інструментів та обладнання GearForMusic.
- Було визначено переваги платформи Amazon: інтуєтивний інтерфейс, пошук товарів за багатьма фільтрами, яскраве виділення основних характеристик товарів.
- Було виділено якісні переваги платформи GearForMusic: чітке розмежування інструментів за типами та виробниками, зручну для користувача корзину покупок.
- Також було виділено недолік українського онлайн-магазину – доволі помітну затримку при переході між сторінками каталогів та товарів.



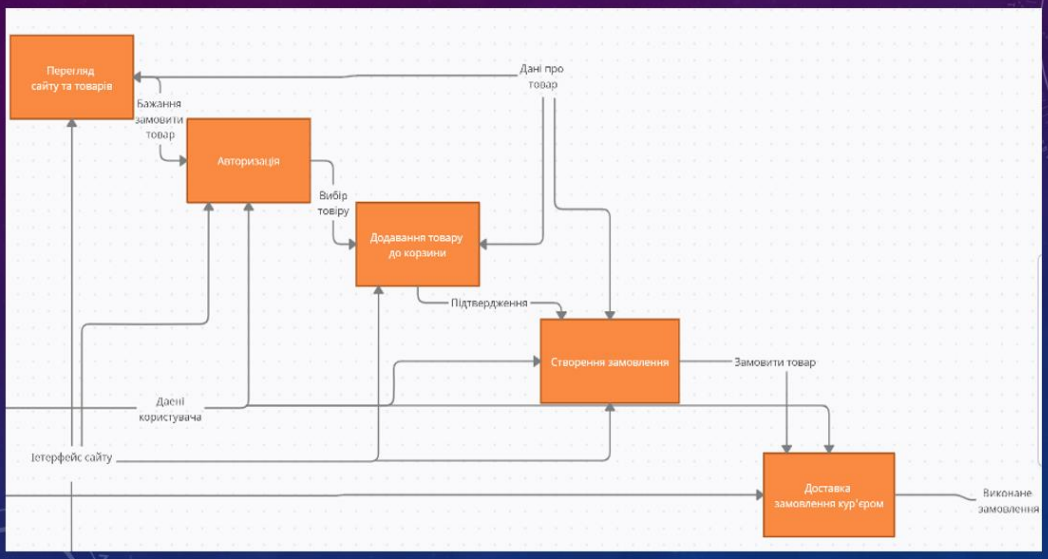




ФУНКЦІОНАЛЬНА ДІАГРАМА СИСТЕМИ

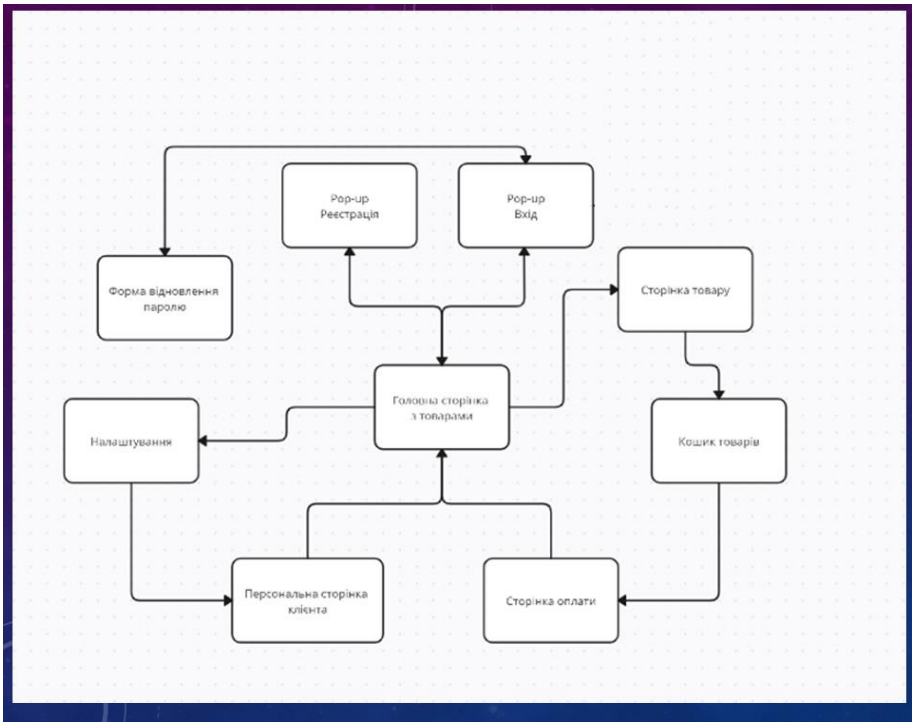
- Було розроблено ряд діаграм, зокрема функціональну діаграму. На рисунку продемонстровано основний сценарій використання веб-застосунку користувачем. Кожний елемент діаграми демонструє процес, необхідний для покупки товару у розробленому застосунку, а також послідовність процесів. Кінцевим елементом і головним сценарієм є успішне придбання товару клієнтом.

ДІАГРАМА ДЕКОМПОЗИЦІЇ СИСТЕМИ



ДІАГРАМА ДЕКОМПОЗИЦІЇ АВТОРИЗАЦІЇ

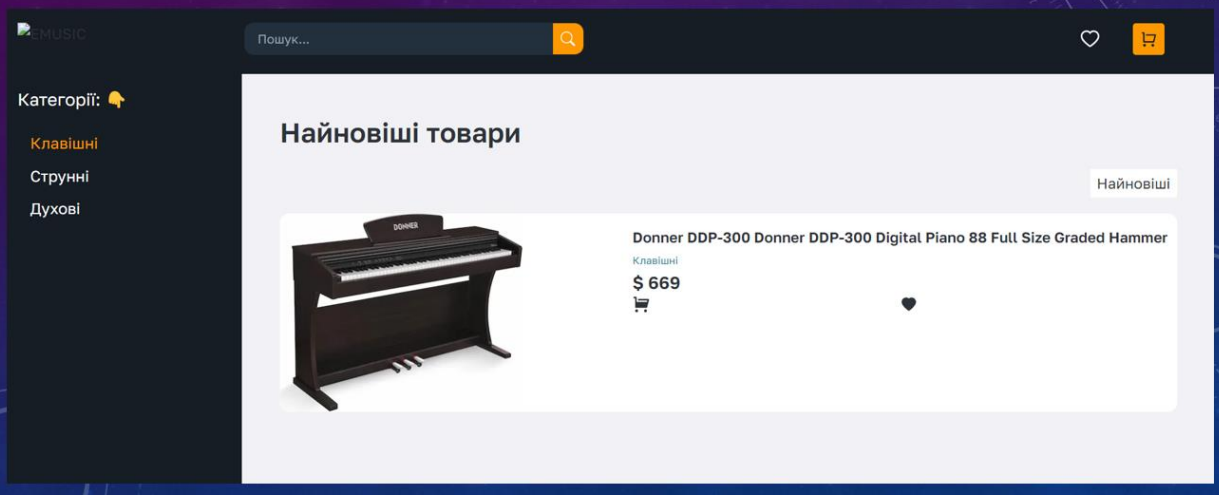
- Безпека даних клієнта в системі та досконалість авторизації є особливо важливими.
- Для реалізації авторизації було обрано JWT - JSON Web Token. Це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519, який використовується для передачі даних для аутентифікації в клієнт-серверних застосунках.
- З використанням технологій Nest.JS було налагоджено зв'язок з базою даних таким чином, щоб користувач мав можливість управляти сласними даними для авторизації.



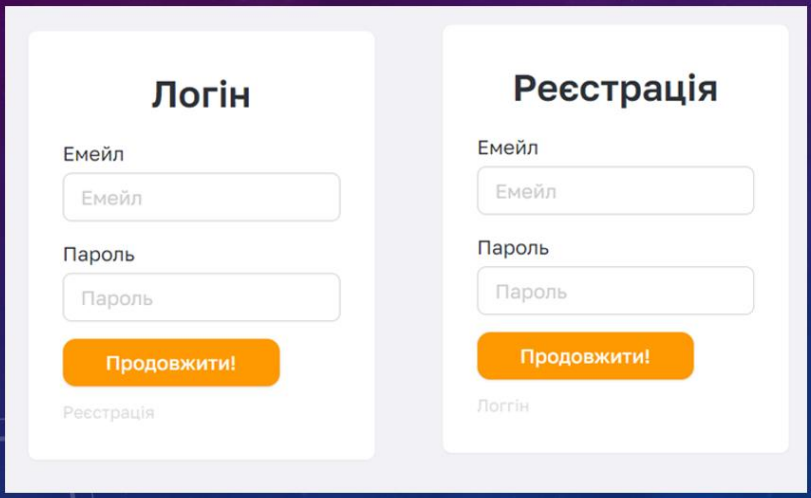
СТРУКТУРА ІНТЕРФЕЙСУ ВЕБ-ЗАСТОСУНКУ

- Головним елементом є головна сторінка, на якій користувач може бачити інструменти. З використанням технологій Nest.JS та React Router Dom було реалізовано функціонал для переходу з головної сторінки на будь-яку іншу сторінку веб-системи.

- Розробка ПЗ проходила шляхом створення окремих модулів (сторінок сайту) та написання логіки їх взаємодії. Основна логіка серверної частини написана на TypeScript з використанням фреймворку NestJS.
- Клієнтська частина застосунку розроблена з використанням технологій ReactJS та Next.JS.
- Для роботи з базою даних використовується СКБД PosgreSQL та Prisma.



АВТОРИЗАЦІЯ КОРИСТУВАЧА



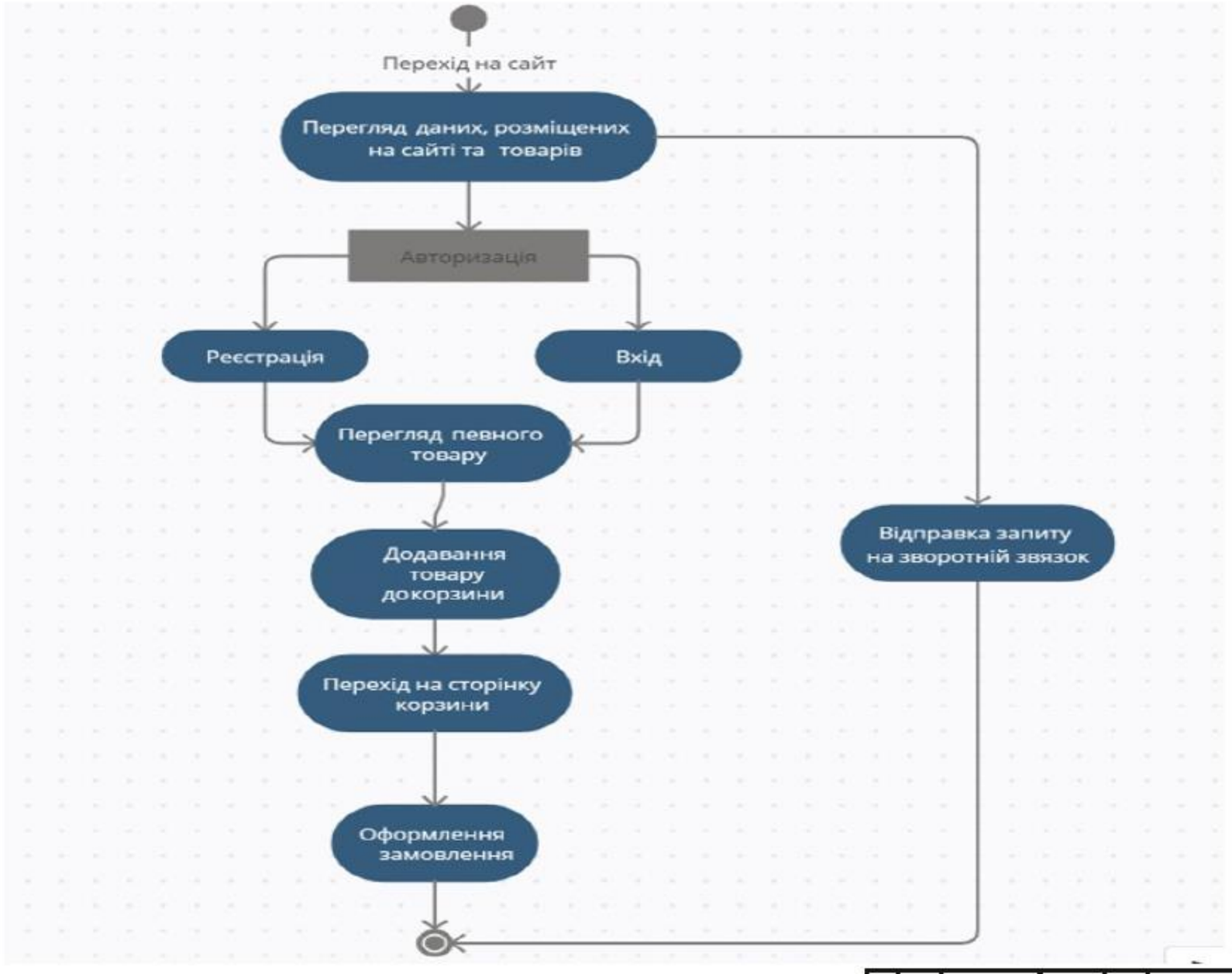
- Для реалізації авторизації використано JSON Web Token. JWT використовується наступним чином. Коли користувач системи заходить в систему використовуючи логін та пароль (або інші дані) сервер аутентифікації створює JWT и відправляє токен користувачу-клієнту. Володіючи токеном, користувач при кожному запиті до API відправляє також сам токен. І оскільки сервер ПЗ має секретний ключ (підпис) токена, то при отримані запиту від клієнта, він перевіряє ключ на ідентичність і таким чином перевіряє власника токена. Також JWT має алгоритм продовження сесії користувача автоматично.

ВИСНОВКИ

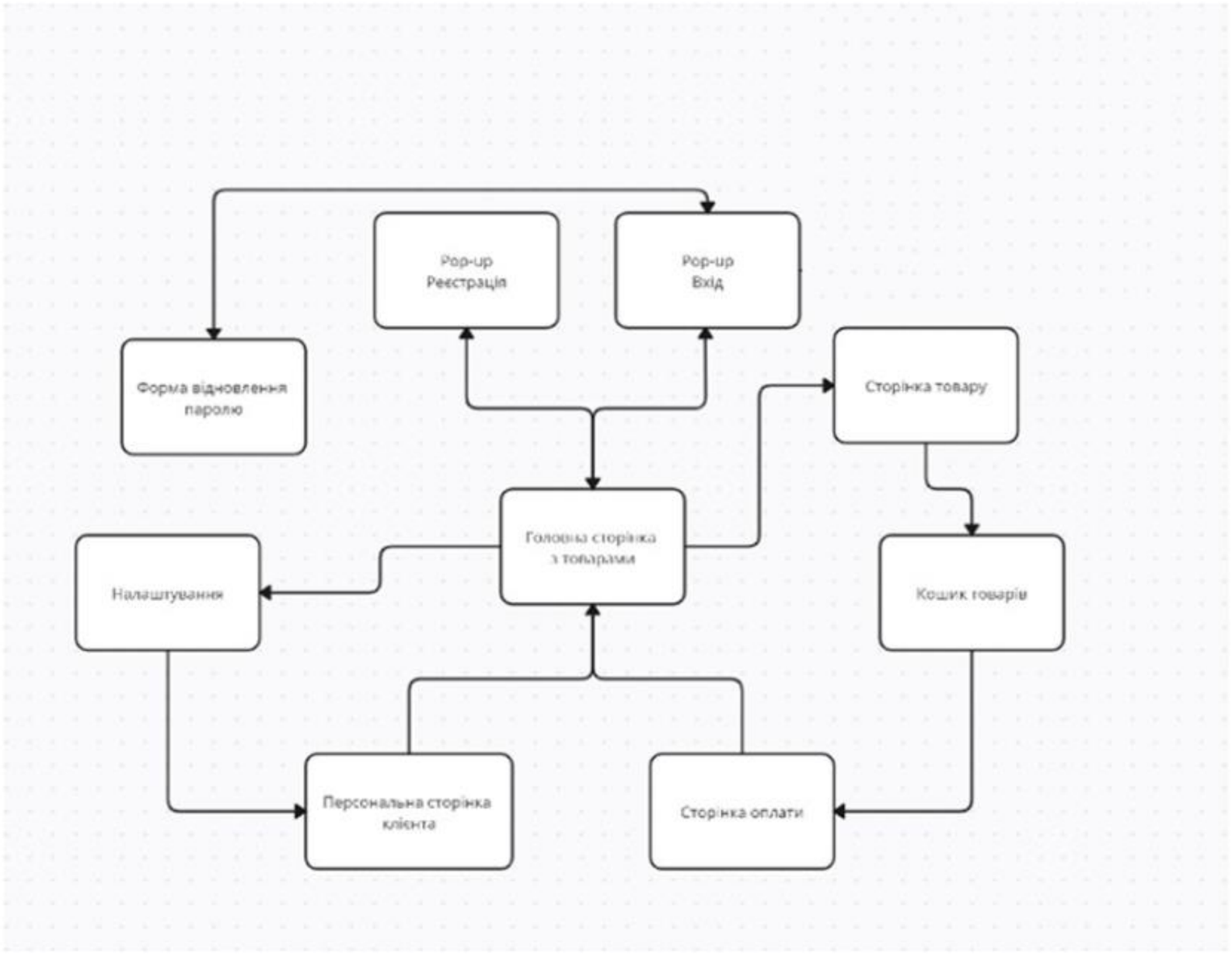
- При виконанні кваліфікаційної роботи на тему «Веб-додаток для продажу музичних інструментів» було розроблено архітектуру веб-застосунок після чого спроектовано та реалізовано функціонал веб-застосунку, що виконує функції інтернет магазину.
- Середовище розробки Microsoft Visual Studio 2019. Архітектура – клієнт-серверна. Платформа –Node JS. Мова – JavaScript, TypeScript . Фреймворки – Nest.JS, React.JS, Next.JS.
- Корисність розроблюваного програмного продукту для компаній, що займаються дистрибуцією музичних товарів та інструментів полягає у можливостях збільшення аудиторії покупців та об'ємів продажів шляхом масштабування свого бізнесу через інтернет.
- При виконанні роботи було створено веб-застосунок інтернет-магазин музичних інструментів, пояснювальну записку, три креслення у форматі А3.
- Розроблений веб-застосунок відповідає поставленим завданням, виконує усі необхідні функції та не має функціональних недоліків.

ДЯКУЮ ЗА УВАГУ

ГРАФІЧНА ЧАСТИНА



Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190135.01.11.Е8		
Розробив		Лембас М. С.			Веб-застосунок для продажу музичних інструментів	Літера	Маса	Масштаб
Керівник		Радельчук Г. І						
Консульт.					UML-діаграма варіантів використання	Аркуш 1	Аркушів 3	
Н. Контр.		Радельчук Г. І						
Зав. Каф.		Бедратюк Л.П						



Зм.	Арк.	№ докум.	Підпис	Дата
Розробив		Лембас М. С.		
Керівник		Радельчук Г. І		
Консульт.				
Н. Контр.		Радельчук Г. І		
Зав. Каф.		Бедратюк Л.П		

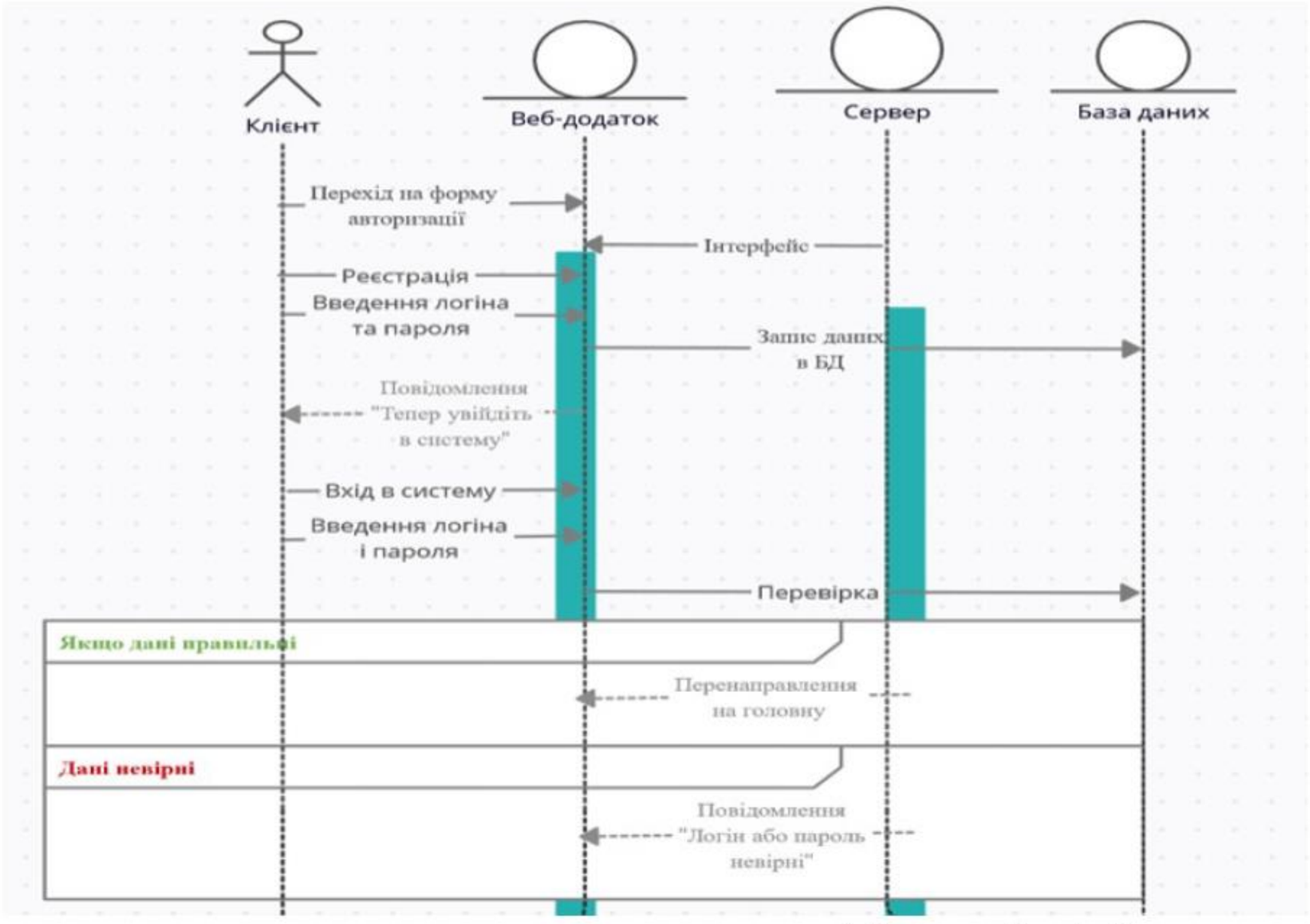
КвРІПЗ.190135.01.11.Е8

Веб-застосунок для продажу музичних інструментів

Діаграма вікон інтерфейсу

Літера	Маса	Масштаб

Аркуш 2	Аркушів 3



					КвРІПЗ.190135.01.11.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Веб-застосунок для продажу музичних інструментів UML-діаграма послідовності авторизації	Літера / Маса / Масштаб
Розробив		Лембас М. С.				
Керівник		Радельчук Г. І				Аркуш 3 / Аркушів 3
Консульт.						
Н. Контр.		Радельчук Г. І				
Зав. Каф.		Бедратюк Л.П				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Лембаса М. С.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.06.2023
дата

підпис

Sun Jun 04 11:57:37 EEST 2023, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114640 Назва: БКР Веб-застосунок для продажу музичних інструментів Додано в БД: 2023-06-04 Автора: Лембас М. С. Керівники: Радельчук Г. І. к.т.н. доц. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	85063	705	5149 (6%)	49 (7%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

Дата перевірки:
04.06.2023 12:56:43 EEST

Дата звіту:
04.06.2023 12:59:32 EEST

ID перевірки:
1015411194

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100005589

Назва документа: **Кваліфікаційна робота Лембас**

Кількість сторінок: 66 Кількість слів: 12819 Кількість символів: 101340 Розмір файлу: 3.99 MB ID файлу: 1015074426

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

5.62%
Схожість

Найбільша схожість: **1.41%** з джерелом з Бібліотеки (ID файлу: **1015045634**)

4.01% Джерела з Інтернету	387	Сторінка 68
3.21% Джерела з Бібліотеки	81	Сторінка 70

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування	10 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Лембас Максим Сергійович

Тема Веб-застосунок для продажу музичних інструментів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень три; кількість сторінок записки 67

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі проведено та описано дослідження комплексне предметної області онлайн магазинів музичних інструментів. Дослідження відбувалося шляхом огляду наявних рішень та технологій реалізації. Згідно з отриманими даними було виділено вимоги до розроблюваного програмного забезпечення та доведено актуальність його розробки. Було спроектовано веб-додаток та власне програмно реалізовано онлайн-магазин музичних інструментів. Також було проведено тестування на наявність критичних недоліків системи і таким чином доведено функціональність програмного продукту.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі до кваліфікаційної роботи описано тематику, виділено мету та актуальність роботи, а також сформульовано завдання на кваліфікаційну роботу. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних веб-застосунків та популярних типів архітектури, розглянуто їх переваги і недоліки та визначено, що оптимальною архітектурою для системи є клієнт-серверна архітектура. У третьому розділі описано створення серверних та клієнтських модулів веб-застосунку та подальше тестування системи. Усі розділи проілюстровано рисунками та доповнено таблицями.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки сфера інтернет-торгівлі активно розвивалася протягом останніх чотирьох років і така тенденція має перспективу росту, отже є велика кількість, як потенційних замовників такого програмного забезпечення, так і потенційних користувачів такого веб-застосунку.

5. Негативні сторони роботи У кваліфікаційній роботі було розроблено додаток, який не повністю реалізовує потенціал пошуку за фільтрами та детальними особливостями кожного типу музичних інструментів, це могло б сподобатися користувачам застосунку, які чітко знають, який товар вони бажають придбати.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення подано у вигляді ілюстративних рисунків, зокрема UML-діаграм та зображень описуваних елементів. Оформлення пояснювальної записки відповідає вимогам чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки подано доступно та зрозуміло, це дозволяє комплексно зрозуміти викладений матеріал та виділити головні досягнення описані у роботі. Також, графічний матеріал дає можливість візуалізувати деталі та принципи функціонування системи та краще їх зрозуміти.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Мартинюк Валерій Володимирович, доктор технічних наук, професор, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки ХНУ

"05" червня 2023 р.

(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Веб-застосунок для продажу музичних інструментів»
Автор: Лембас Максим Сергійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Радельчук Галина Іванівна, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 5.62% і адресується до 387 джерел з Інтернету та 81 джерела з бібліотеки, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата _____

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Галина РАДЕЛЬЧУК