МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна веб-орієнтована система агенції ритуальних послуг»
здобувачки групи ІН-94-1 Листопад Каріни Едуардівни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Каріна ЛИСТОПАД
(підпис)

Керівник,
доцент, кандидат фізико-математичних наук
Сергій ШАПОВАЛОВ _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-94-1 Листопад Каріни Едуардівни

1. Тема роботи: «Інформаційна веб-орієнтована система агенції ритуальних послуг» затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються при створенні інтернет-магазинів.
3) Розробка клієнтської та адміністративної частини сайту.
4) Представлення отриманих результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технології, що використовуються для реалізації інформаційної системи агенції ритуальних послуг*		
3	*Розробка інформаційної системи агенції ритуальних послуг*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 47 стр., 20 рис., 1 додаток, 14 використаних джерел.

Обґрунтування актуальності теми роботи – тема даної кваліфікаційної роботи є актуальною, так як відповідає потребам сучасного цифрового світу і може покращити доступ до інформації, оптимізувати робочі процеси та підвищити якість обслуговування для клієнтів.

Об’єкт дослідження — система агенції ритуальних послуг.

Мета роботи — розробка веб-орієнтованої системи агенції ритуальних послуг з використанням технології Java Spring Boot.

Методи дослідження — ознайомлення з можливими способами реалізації, створення БД, прототипування

Результати — створено інформаційну систему, що має необхідний набір функцій для ознайомлення з асортиментом товарів та послуг магазину з інтерфейсом для користувача та адміністратора.

JAVA SPRING BOOT, HTML, БАЗИ ДАНИХ, POSTGRE SQL, ВЕБ-ЗАСТОСУНКИ

ЗМІСТ

ВСТУП

Актуальність теми дослідження. У сучасному світі, що швидко оцифровується, веб-ресурси стали невід'ємною частиною нашого повсякденного життя. Можна з упевненістю сказати, що вони є дуже важливими в сучасному світі. Інтернет зробив революцію в тому, як люди отримують доступ до інформації, спілкуються, співпрацюють та навчаються.

Популярність веб-додатків зростає з кожним роком. Вони надають користувачам зручність доступу до послуг та інформації через Інтернет. Застосування веб-технологій дозволяють додаткам бути кросплатформеними і працювати на різних пристроях. Багато користувачів віддають перевагу веб-додаткам через їх простоту використання і можливість оновлення без необхідності завантажувати нову версію. Розвиток мобільних технологій і широка доступність до Інтернету додатково підтримують популярність веб-додатків.

Веб-сайти використовуються для різних цілей, включаючи комерційні, освітні, розважальні, новинні, соціальні та багато інших. Їх структура, дизайн та функціональність можуть значно варіюватися в залежності від потреб власників та цільової аудиторії.

Кожен розуміє наскільки важливо мати сильну присутність в Інтернеті для бізнесу, в тому числі для магазинів.

Розробка веб-сайту для магазину слугує декільком цілям і надає численні переваги: більша видимість – веб-сайт дозволяє магазину охопити ширшу аудиторію за межами його фізичного розташування. Веб-сайт дозволяє потенційним клієнтам відвідати магазин, дізнатися про нього, ознайомитися з товарами та послугами, а також здійснювати покупки в будь-який час і в будь-якому місці; доступність 24/7 – на відміну від фізичних магазинів з обмеженими годинами роботи, веб-сайти надають доступ до магазину 24/7; залучення та взаємодія з клієнтами: веб-сайти надають покупцям платформу для взаємодії з магазинами за допомогою таких

функцій, як контактні форми, чат і розділи зворотного зв'язку. Це сприяє підвищенню лояльності клієнтів, налагодженню взаємовідносин і допомагає магазинам збирати цінну інформацію для вдосконалення.

Об’єктом дослідження є безпосередньо сама веб-орієнтована інформаційна система агенції ритуальних послуг.

Мета роботи. Розробка веб-орієнтованої системи агенції ритуальних послуг, це відповідає актуальності теми. Для реалізації необхідно буде поставити основні вимоги, ознайомитися з методами реалізації, створити Бази Даних, а також прописати дизайн.

Предмет дослідження. Процес створення інформаційної веб-орієнтованої системи агенції ритуальних послуг та особливостей реалізації.

Новизна. Java Spring Boot є потужним та інноваційним фреймворком, який дозволяє швидко розробляти веб-сайти та додатки з високою продуктивністю та гнучкістю. Його функціональність та можливості роблять його гарним вибором для розробки сучасних веб-проектів.

Структура. Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Популярність та використання веб-додатків у магазинних сферах.

Власники бізнесу завжди шукають способи розвитку своїх компаній. Завдяки прогресу технологій, створення веб-додатків стає привабливим варіантом для розширення сфери впливу вашого бізнесу. Веб-додатки можуть надавати цінні послуги, доступні цілодобово і з будь-якого місця. Це означає, що ви можете пропонувати переваги своїх товарів або послуг, навіть якщо у вас немає фізичних точок продажу в інших країнах, штатах або містах.

Це також сприяє збереженню клієнтів, оскільки людям завжди потрібно займатися ділами на своєму комп'ютері, і вони можуть мати доступ до веб-додатка в будь-який зручний для них час. Загалом, кількість користувачів Інтернету зростає середньорічно на 10%, що підтверджується даними на Рис. 1.1. Це важлива статистика, яка добре підкреслює актуальність веб-додатків.

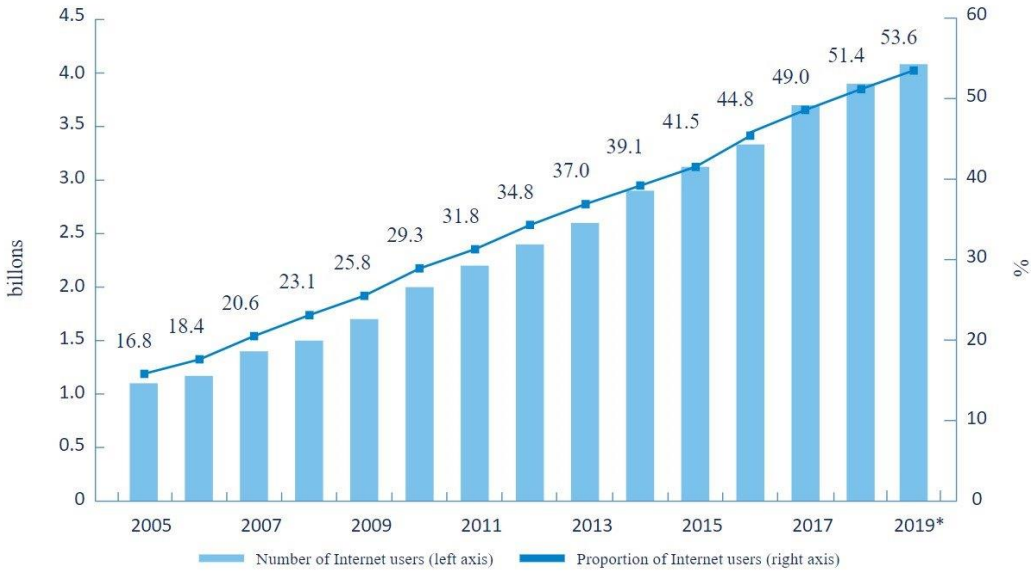

Рис. 1.1 — Статистика користувачів інтернету

Веб-додаток є програмним забезпеченням або програмою, яка може бути відкрита у будь-якому веб-браузері. Зовнішній інтерфейс веб-додатку

розроблений за допомогою мов програмування, таких як HTML, CSS та JavaScript, які підтримуються різними браузерами, такими як Opera, Chrome, Mozilla та Yandex. Серверна частина (backend) може бути написана на будь-якій іншій мові програмування або фреймворку, такій як Python, PHP, Ruby або Java.

В цілому, веб-додатки продовжують здобувати популярність, оскільки вони надають зручний доступ до функцій та даних через веб-браузер, а також забезпечують гнучкість та легкість масштабування, сприяють спільній роботі та співпраці. З розвитком нових технологій, таких як штучний інтелект та блокчейн, можна очікувати подальше розширення та розвиток веб-додатків у майбутньому. [1]

1.2. Переваги та недоліки веб-додатків

Розвиток веб-додатків продовжує активно прогресувати, розкриваючи нові можливості для різних сфер діяльності, включаючи бізнес, освіту, комунікації та інші. Розуміння переваг і недоліків веб-додатків допомагає розробникам та користувачам приймати обґрунтовані рішення щодо вибору та використання відповідних інструментів і ресурсів для своїх завдань і потреб.

Переваги веб-додатків:

1. Економічність: одним з найбільш привабливих аспектів створення веб-додатків є вартість. Послуги з веб-розробки, необхідні для створення веб-додатків, є значно доступнішими, ніж інші види розробки програмного забезпечення. Процес створення посилань між URL-адресою та програмою є відносно простим, що дозволяє економити час. Таким чином, створення веб-додатків є вигідним з фінансової точки зору для їх власників. Легкість налаштування також сприяє економії. Завдяки простоті налаштування веб-інтерфейсу багатьом розробникам вдається швидко вносити зміни до програми, що зменшує час і зусилля, що

витрачаються і призводить до ефективнішого використання ресурсів.

2. Постійно оновлюється: веб-додатки не потребують частого оновлення, як це зазвичай відбувається з традиційними програмами. Сам веб-сайт або URL-адреса, пов'язана з програмою, оновлюється до останньої версії. Так як усі користувачі мають доступ до однієї версії веб-додатку через одну і ту саму URL-адресу, всі вони завжди використовують найновішу та однакову версію.

3. Не потребує завантаження: завдяки можливості безпосередньо взаємодіяти з програмою через веб-браузер, веб-додатки не потребують окремого встановлення або завантаження з різних платформ, наприклад, Google Play або Apple App Store. Це також забезпечує економію коштів, оскільки немає необхідності витрачати гроші на пряме посилання до веб-додатка. Крім того, до веб-додатків можна отримати доступ через різні браузери і працювати на різних платформах, таких як ноутбуки, настільні комп'ютери або мобільні телефони.

4. Легка робота: веб-додатки розроблені з такими функціями веб-дизайну, що дозволяють їм працювати на будь-якій операційній системі. Їхній інтерфейс може легко адаптуватися до різних розмірів екранів, таких як iOS, Android або Windows, при умові, що встановлено веб-браузер.

Недоліки веб-додатків

1. Ненадійність Інтернету: незважаючи на широке поширення Інтернету, втрата з'єднання з мережею є відносно поширеною проблемою. Однак, веб-додаток може продовжувати працювати незалежно від доступу до Інтернету, тому його функціонал не буде повністю обмежений.

2. Залежність від доступності веб-сайту: веб-додаток побудований на основі веб-сайту, тому його функціонування може бути під впливом доступності самого сайту. Якщо веб-сайт відмовляє або працює нестабільно, це може вплинути на роботу веб-додатка. Тому важливо мати надійний та

стабільний веб-сайт для успішної роботи веб-додатка.

3. Знижена швидкість: веб-додатки часто працюють трохи повільніше порівняно з програмами, що працюють на локальному сервері. Це може зумовлено залежністю від швидкості Інтернет-з'єднання та розміром програми. Однак, технології постійно вдосконалюються, що допомагає зменшити цю різницю і поліпшує продуктивність веб-додатків.
4. Хоча застосування протоколу SSL може знизити ризик порушення даних, веб-додатки зазвичай не мають вбудованої функції контролю якості, що призводить до меншої безпеки і створює загрози для важливої та конфіденційної інформації.
5. Веб-програми мають обмежену функціональність порівняно з рідною технологією. Через те, що вони не є нативними, іноді вони не можуть ефективно співпрацювати з усім обладнанням і операційними системами конкретних пристроїв, які ви використовуєте..[2-3]

1.3 Постановка задачі

В якості задач даної роботи можна виділити такі пункти:

1. Дослідження Java Spring Boot: вивчення основних концепцій, принципів роботи та можливостей фреймворку.
2. Побудова діаграм Use-case та DFD
3. Вибір СУБД та створення Бази Даних
4. Прототипування сайту
5. Розробка сайту
6. Використання
7. Функціональне тестування сайту

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Дослідження Java Spring Boot: вивчення основних концепцій, принципів роботи та можливостей фреймворку

Java Spring Boot є відкритою інструментальною платформою, яка спрощує створення мікросервісів та веб-додатків на основі Java-фреймворків. Вона поєднує у собі переваги Java, однієї з найпопулярніших мов програмування, та функціональні можливості Spring Framework.

Spring Framework забезпечує функціонал ін'єкції залежностей, що дозволяє об'єктам визначати свої залежності, які потім автоматично інтегруються контейнером Spring. Це дозволяє розробникам створювати модульні додатки зі слабко зв'язаними компонентами, що ідеально підходить для мікросервісів та розподілених додатків.

Spring Framework також надає вбудовану підтримку для типових завдань, таких як зв'язування даних, перетворення типів, обробка винятків, керування ресурсами та подіями, інтернаціоналізація та інші. Він також інтегрується з різними технологіями Java EE, що дозволяє створювати міжплатформні додатки Java EE, які працюють у будь-якому середовищі. [4]

Як і будь-яка інша технологія Java Spring Boot має свої сильні та слабкі сторони, які слід враховувати під час ухвалення рішення про його використання.

До переваг Spring boot можна віднести:

1. Автоматичне налаштування: Spring має вбудовану функцію автоконфігурації, яка дозволяє розробникам спростити процес розробки додатків. Замість того, щоб вручну налаштовувати кожну складову частину додатку, Spring Boot автоматично визначає конфігурацію на основі включених залежностей та стандартних уявлень про те, як повинен працювати додаток. Це дозволяє розробникам економити час, який би витратили на ручне налаштування, і зосередитися на самому розробці

функціональності.

2. Вбудований сервер: Spring Boot має вбудований веб-сервер за замовчуванням, який називається Tomcat. Це означає, що для запуску додатків не потрібно окремо налаштовувати та встановлювати контейнер сервлетів, так як сервер Tomcat вже вбудований у фреймворк. Крім того, Spring Boot також підтримує інші сервери, такі як Undertow і Jetty, що дає розробникам можливість вибрати сервер, який найкраще відповідає їхнім потребам.

3. Економія пам'яті: Spring Boot використовує техніку початкового зв'язування (lazy initialization), що допомагає зменшити споживання пам'яті та прискорити завантаження додатка. Замість того, щоб завантажувати всі компоненти додатка одразу, Spring Boot завантажує їх лише при необхідності. Це дозволяє зменшити використання пам'яті, особливо для великих додатків з багатьма компонентами, і покращити час відгуку додатка.

4. Конфігурація анотацій або XML: Spring Boot надає розробникам можливість використовувати як анотації, так і конфігураційні файли у форматі XML для налаштування додатка. Це означає, що розробники можуть вибрати спосіб конфігурації, з яким вони найбільше комфортні. Використання анотацій дозволяє зробити конфігурацію більш зрозумілою та зручною для використання в коді, тоді як XML-конфігурація може бути корисною для більш складних або об'ємних налаштувань.

5. Використання легких JAR-файлів: Spring Boot підтримує використання JAR-файлів для розповсюдження та виконання додатків. JAR (Java Archive) - це компактний формат файлу, який містить весь код та залежності додатка. Використання JAR-файлів спрощує процес розгортання додатків, оскільки вони можуть бути легко перенесені та виконані на будь-якій платформі, що підтримує Java. Крім того, Spring Boot надає можливість збирати виконувані JAR-файли, які містять в собі

весь необхідний середовища запуску, що робить розгортання ще простішим.

6. Активна спільнота розробників: Spring Boot є популярним фреймворком з відкритим кодом, що має велику та активну спільноту розробників. Це означає, що ви можете знайти підтримку, навчальні матеріали та приклади коду для вирішення проблем або отримання кращих практик. Спільнота розробників постійно внесює внески до розвитку Spring Boot, публікує оновлення та нові функції, що дозволяє вам бути в курсі останніх трендів та можливостей фреймворку.

Серед недоліків Spring boot виділяють:

1. Без контролю: Один з недоліків Spring Boot полягає у тому, що розробники не мають повного контролю над розміром файлу розробки. Це пов'язано з його дизайном та конструкцією завантажувача Spring, який може додавати непотрібні залежності до проекту. Багато з цих залежностей не використовуються в конкретному проекті, але все одно додаються до файлу, збільшуючи його розмір. Це може стати проблемою, особливо для проектів з обмеженими ресурсами.

2. Підтримка великомасштабних проектів: Spring Boot спеціалізується на розробці мікросервісних архітектур і надає зручність для розгортання та управління декількома незалежними сервісами. Однак, для великих монолітних проектів, де усі компоненти тісно зв'язані, використання Spring Boot може стати менш зручним і навіть призвести до певних труднощів при управлінні великим обсягом коду.

3. Витрати часу: Перетворення проекту на Spring Boot може вимагати певного часу та зусиль. Це пов'язано з інтеграцією залежностей, налаштуванням конфігурації та перенесенням існуючого коду в формат, зрозумілий для Spring Boot. Чим складніший та розгалужений проект, тим більше часу може зайняти цей процес. Крім того, внесення змін у вже

завантажений проект Spring Boot може бути складним, особливо для розробників, які не мають глибоких знань про внутрішню структуру та історію Spring.

4. Модифікації: Розробка змін та виправлення неполадок під час використання Spring Boot може бути складнішою порівняно зі стандартним використанням Spring. Якщо розробник не має достатнього розуміння системи та внутрішніх механізмів Spring, внесення змін може бути проблематичним. Це може відбитися на швидкості розробки та затримках у вирішенні проблем.

5. Використання ресурсів: Функція автоконфігурації Spring Boot, яка дозволяє автоматично налаштовувати додаток, може призвести до використання непотрібних ресурсів. Навіть якщо деякі функції не потрібні у конкретному проекті, вони все одно можуть бути активовані та споживати ресурси комп'ютера, збільшуючи загальну обсягову споживання ресурсів.

6. Відсутність інструментів: Spring Boot поставляється з базовим набором інструментів, але може бути обмеженим для деяких специфічних завдань. Наприклад, для перетворення застарілого коду, який використовує Spring, можуть знадобитися додаткові інструменти, які не входять до стандартного комплекту Spring Boot. Розробникам може бути необхідно самостійно шукати та інтегрувати такі інструменти для вирішення конкретних завдань. [5]

2.2 Побудова діаграм Use-case, DFD та ERD

Use case діаграма є графічною представленням варіантів використання (use case) в системі або програмному забезпеченні. Вона використовується для визначення функціональних вимог до системи та моделювання очікуваної поведінки системи з точки зору кінцевого користувача.

Варіанти використання вказують на різні сценарії взаємодії між

акторами (користувачами) та системою. Кожен варіант використання описує специфічну функцію або функціональну вимогу, яку система повинна задовольнити для користувача. Вони фокусуються на "що" система повинна робити, а не на "як" це повинно бути здійснено.

Одним з ключових принципів використання діаграм варіантів використання є спрощення проектування системи, концентруючись на потребах та очікуваннях кінцевих користувачів. Це дозволяє розробникам системи зосередитися на визначенні функцій та функціональних можливостей, які важливі для користувачів.

Діаграма варіантів використання допомагає визначити зовнішньо видиму поведінку системи, тобто як система спілкується з акторами та які результати вона повинна надати. Вона може включати в себе такі елементи, як актори, варіанти використання, асоціації між ними, пріоритети та залежності між варіантами використання.

Загальний результат використання діаграми варіантів використання полягає в тому, що вона допомагає уточнити та узгодити вимоги до системи між розробниками та зацікавленими сторонами. Вона слугує важливим інструментом для забезпечення спільного розуміння функціональних потреб системи та допомагає визначити правильну архітектуру та проектування системи. [6]

Відповідна до завдання діаграма Use-case зображена на Рис. 2.1.

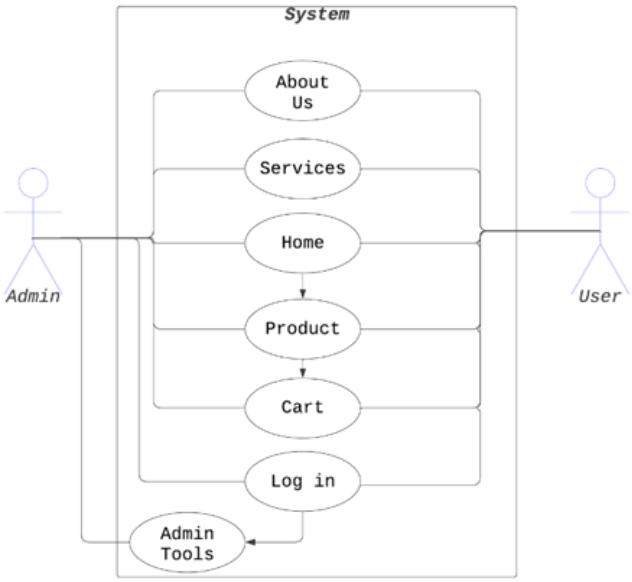

Рис. 2.1 — Use-case діаграма

Діаграма потоку даних (DFD) є ефективним засобом візуалізації та аналізу потоків даних в системах або організаціях. Вона дозволяє детально розглянути переміщення інформації з одного місця до іншого і представити цю інформацію як послідовність процесів.

Основною метою DFD є моделювання потоку даних із точки зору взаємодії між різними компонентами системи. Діаграма використовує нотацію, що включає процеси, вхідні та вихідні дані, а також потоки даних, що їх з'єднує. Процеси на діаграмі представляють обробку або маніпулювання даними, а потоки даних вказують напрямок руху інформації.

DFD може бути використана для різних цілей. Вона може служити як інструмент для моделювання бізнес-процесів, де вона відображає взаємодію різних відділів та підрозділів організації. Це допомагає встановити чіткі зв'язки та взаємозалежності між різними частинами організації та вирішувати проблеми, пов'язані з координацією та спільною роботою. [7]

Елементи даних, які є основними складовими нотації DFD, можуть бути описані наступним чином:

1. Процес: Це компонент, який здійснює перетворення вхідних даних у вихідні результати. Процес приймає вхідні дані з потоків даних і виконує певні обробки, операції або маніпуляції над цими даними, щоб згенерувати вихідні дані. Він може бути представлений у вигляді функцій, операцій, модулів або послідовностей кроків, необхідних для виконання певної задачі;

2. Потік даних: Це шлях, по якому дані переміщуються всередині системи. Потоки даних вказують на переміщення інформації між процесами, зовнішніми сутностями та сховищами даних. Вони представляють характер даних, що передаються або обробляються в системі. Кожен потік даних має свій унікальний ідентифікатор і може мати певну назву, що вказує на тип даних, що передаються;

3. Сховище даних (Datastore): Це компонент, який представляє місце для зберігання даних, які в системі не переміщуються або не обробляються в даний момент. Сховища даних можуть включати бази даних, файли, таблиці або будь-які інші форми зберігання інформації. Вони використовуються для зберігання даних для подальшого використання або обміну між різними процесами;

4. Зовнішня сутність (External Entity): Це зовнішні джерела або призначення інформації, які знаходяться за межами описуваної системи. Зовнішні сутності можуть постачати дані системі або отримувати дані від процесів. Вони можуть бути представлені як інші системи, користувачі, додатки, сенсори або будь-які інші джерела або приймачі інформації. Зовнішні сутності зазвичай розташовуються на краях діаграми DFD і вказують на границі системи.

За даними, представленими на Рис. 2.2., можна побачити послідовність процесів та взаємозв'язки між ними у системі обробки даних. Вона надає уявлення про рух інформації в системі, починаючи з початкових джерел даних і пролягаючи через різні етапи обробки.

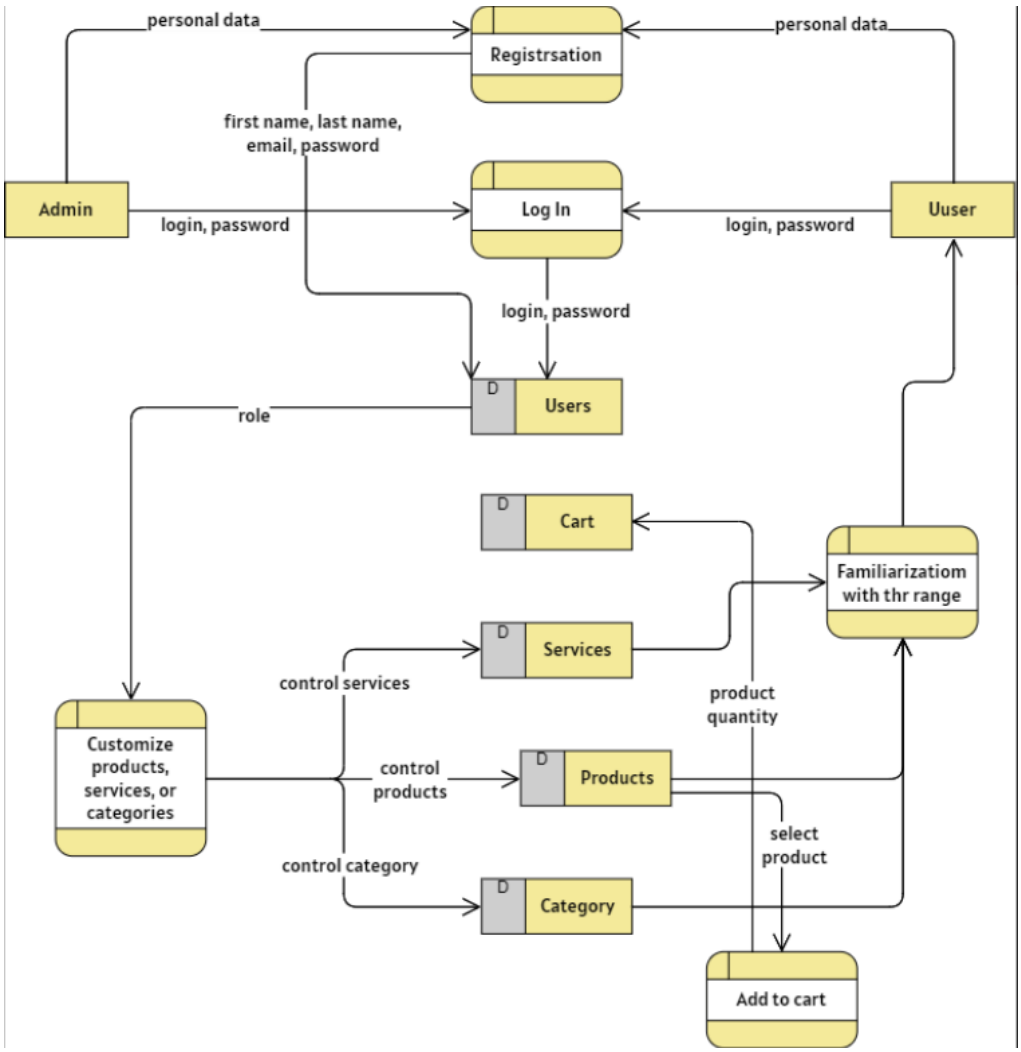

Рис. 2.2 — Data Fiew Diagrams

ER-модель, що розшифровується як Entity Relationship Model (модель "Сутність-Зв'язок"), є схемою концептуальної моделі даних високого рівня. Ця модель графічно відображає сутності, які існують у реальному світі, та зв'язки між ними.

Зв'язки в моделі ER вказують на взаємозв'язки між сутностями. Наприклад, відношення "Студент бере участь у Курсі" вказує на зв'язок між сутностями "Студент" і "Курс". Зв'язки можуть мати також атрибути, які деталізують цей взаємозв'язок.

Модель ER допомагає систематично аналізувати вимоги до даних та визначати структуру бази даних. Вона дозволяє розуміти взаємозв'язки між сутностями, виявляти ключові атрибути, визначати обмеження цілісності

даних та проектувати ефективні схеми баз даних. (Рис. 2.3).

В основному, ERD є необхідним для побудови, оскільки вона виконує наступні функції:

1. Допомагає у визначенні термінології, пов'язаної з моделюванням зв'язків між сутностями. ERD дозволяє чітко встановити та узгодити терміни, які використовуються для опису сутностей, атрибутів та зв'язків у контексті конкретної системи або бізнес-домену.

2. Надає попередній перегляд того, як таблиці пов'язуються одна з одною та які поля мають бути присутніми в кожній таблиці. ERD дозволяє візуалізувати зв'язки між таблицями бази даних та відображати структуру бази даних в зручній формі. Це допомагає проектувальникам та розробникам отримати загальне уявлення про організацію даних і встановити правильні зв'язки між ними.

3. Допомагає описати сутності, атрибути та відносини. ERD надає зручний спосіб описувати різні сутності, які існують у системі, та їх характеристики у вигляді атрибутів. Крім того, вона відображає взаємозв'язки між сутностями, що допомагає зрозуміти, як вони пов'язані одна з одною та які відношення між ними існують.

4. Надає краще розуміння інформації, що міститься в базі даних, за допомогою ERD-діаграм. ERD дає можливість візуалізувати структуру бази даних, показує, як дані організовані та взаємодіють між собою. Це допомагає користувачам та зацікавленим сторонам краще розуміти, як інформація зберігається та організована в системі бази даних.[13]

Розглядаючи всі особливості ER-діаграм, можна сказати, що даній діаграмі властиві простота, ефективність, зрозумілість, інтегрованість, гнучкість, маштабованість, візуальне представлення.

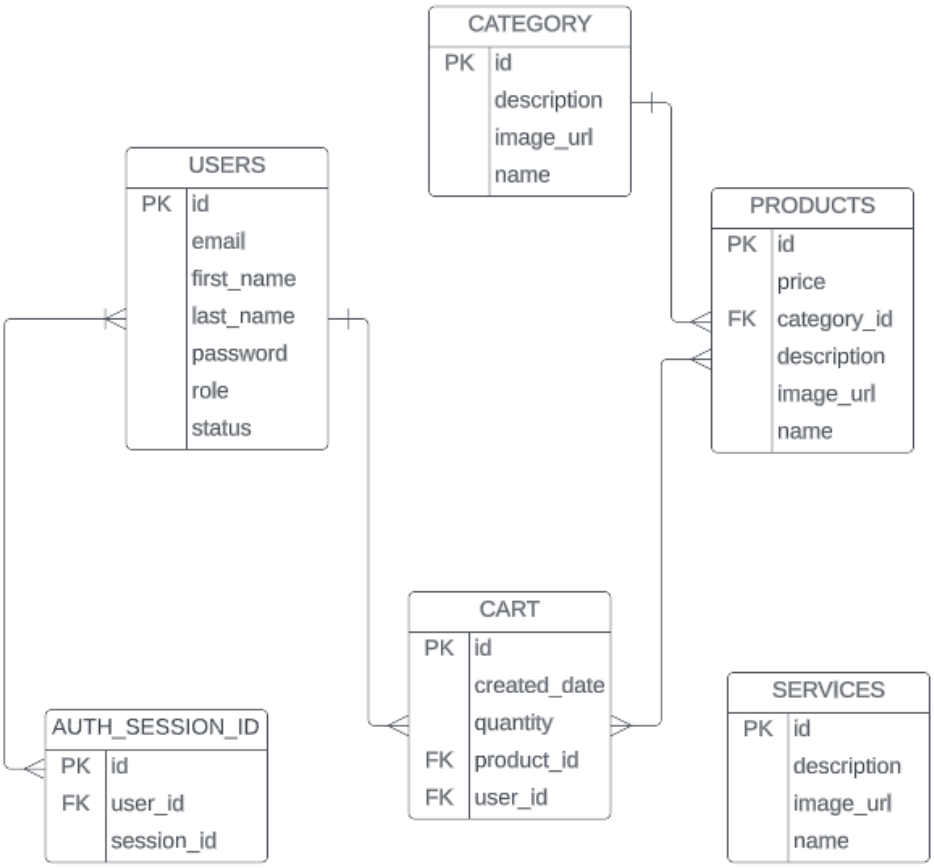

Рис. 2.3 — ER діаграма

2.3 Формування Баз Даних

2.3.1 Вибір СУБД

Було прийнято рішення використовувати реляційні бази даних через їх низку переваг:

1. Реляційні бази даних мають просту модель, що спрощує роботу з ними і не вимагає складних запитів. Це робить їх легкими у використанні.
2. Вони надають можливість швидкого доступу до інформації за допомогою мови SQL, що робить процес отримання даних ефективним та зручним.
3. Реляційні бази даних забезпечують точність і цілісність даних, оскільки вони дозволяють уникнути дублювання даних і підтримують узгодженість між таблицями.

4. Застосування нормалізації в реляційних базах даних гарантує структурну цілісність та збереження цілісності даних під час їх використання.
5. Реляційні бази даних дозволяють кільком користувачам одночасний доступ до бази даних, сприяючи співпраці та обміну інформацією.
6. Вони також забезпечують безпеку даних, обмежуючи доступ неавторизованих користувачів до бази даних. [8]

В цілому, використання реляційних баз даних є доцільним через їх простоту, ефективність, здатність забезпечувати цілісність та безпеку даних, а також сприяти спільній роботі та обміну інформацією.

Після ознайомлення з відповідною літературою на тему реляційних СУБД було вирішено використовувати PostgreSQL.

PostgreSQL є відкритою системою керування базами даних корпоративного класу. Вона надає підтримку як SQL, так і JSON для реляційних і нереляційних запитів, що дозволяє забезпечити розширюваність та відповідність стандартам SQL.

Однією з ключових особливостей PostgreSQL є його підтримка розширених типів даних і функцій оптимізації продуктивності. Це означає, що PostgreSQL надає можливості, які зазвичай доступні лише в дорогих комерційних базах даних, таких як Oracle і SQL Server. Це включає в себе розширені типи даних, такі як географічні об'єкти, бітові рядки, JSON-документи та інші, а також оптимізовані алгоритми для покращення продуктивності запитів.

Поставляючи з відкритим кодом, PostgreSQL також має активну спільноту розробників, яка постійно працює над покращенням системи, виправленням помилок і розробкою нових функцій. Це дозволяє забезпечити стабільну та надійну роботу бази даних, а також швидкість реагування на змінні потреби користувачів.

Переваги PostgreSQL:

1. PostgreSQL забезпечує можливість створювати інтерактивні та динамічні веб-додатки, так як дозволяє запускати динамічні сайти та програми.
2. PostgreSQL надзвичайно стійкий до відмов. Він дозволяє зберігати журнали транзакцій перед їх фактичним застосуванням до БД, і забезпечує можливість відновлення даних в непередбачуваних ситуаціях.
3. PostgreSQL має відкритий вихідний код, що дає волю використовувати, змінювати та розповсюджувати його. Це надає користувачам велику свободу у використанні та адаптації PostgreSQL під їхні потреби, а також сприяє активному розвитку та покращенню системи завдяки внесенню змін спільнотою розробників.
4. PostgreSQL підтримує роботу з географічними об'єктами, що робить його цікавим для розробки додатків, пов'язаних з розташуванням та географічною інформацією. Це дає можливість зберігати, опрацьовувати та аналізувати географічні дані в базі даних.
5. Вивчення PostgreSQL вважається досить простим порівняно з іншими системами баз даних. Він має логічну структуру та зрозумілі концепції, що полегшує його освоєння для нових користувачів і розвиток в навичках роботи з базами даних.
6. PostgreSQL має низькі вимоги до обслуговування та адміністрування, що робить його привабливим для використання як вбудованої системи БД, так і для корпоративних застосунків. Його ефективність і надійність дозволяють зменшити необхідність в постійному втручанні та знизити затрати на обслуговування та підтримку бази даних.

Недоліки PostgreSQL:

1. Відсутність централізованої організації, яка підтримує PostgreSQL, може впливати на ступінь визнання та прийняття цієї системи.

2. Внесення змін для покращення продуктивності PostgreSQL може вимагати більше зусиль порівняно з MySQL.
3. Деякі програми з відкритим кодом можуть підтримувати MySQL, але не володіти підтримкою для PostgreSQL.
4. PostgreSQL може бути повільнішим, ніж MySQL, з точки зору продуктивності. [9]

Після аналізу сильних та слабких сторін було вирішено, що позитивні аспекти PostgreSQL повністю перекривають мінуси цієї СУБД. До того ж наявні обмеження для даного проекту не критичні.

2.3.2 Створення БД

В рамках даної роботи була створена база даних, яка складається з декількох таблиць. Кожна відображає конкретну сутність та має відповідні поля. В таблиці 2.1 наведений опис кожної таблиці. Більш детально про поля таблиць можна дізнатися в Таблиці 2.2

Таблиця 2.1Опис таблиць

Назва таблиці	Опис таблиці
USERS	зберігає всю інформацію про користувача, яку вводить під час реєстрація на сайті
CATEGORY	таблиця з категоріями товарів
PRODUCTS	описує продукти для продажу
SERVICES	містить дані з послугами, які надає магазин
CART	утримує інформацію про те, хто і що додав у кошик
AUTH_SESSION_ID	в таблиці зберігаються дані про те, хто і коли авторизувався

Таблиця 2.2 Опис полів таблиць

Таблиця	Поле	Зміст	Тип	Ключі
users	id	Унікальний ідентифікатор користувача	integer	PK
	email	Електронна пошта	varchar(255)	
	first_name	Ім’я користувача	varchar(255)	

Таблиця	Поле	Зміст	Тип	Ключі
	last_name	Прізвище користувача	varchar(255)	
	password	Пароль	varchar(255)	
	role	Роль користувача	varchar(255)	
	status	Статус, який визначає активний обліковий запис чи ні	varchar(255)	
categoty	id	унікальний ідентифікатор категорії	integer	PK
	description	Опис категорії	varchar(255)	
	image_url	Посилання на зображення	varchar(255)	
	name	Назва категорії	varchar(255)	
products	id	унікальний ідентифікатор продукту	integer	PK
	price	Ціна продукту	double	
	category_id	Ідентифікатор категорії	integer	FK
	description	Опис продукту	varchar(255)	
	imageurl	Посилання на зображення продукту	varchar(255)	
	name	Назва продукту	varchar(255)	
services	id	унікальний ідентифікатор сервісу	integer	PK
	description	Опис сервісу	varchar(255)	
	imageurl	Посилання на зображення	varchar(255)	
	name	Назва сервісу	varchar(255)	
cart	id	унікальний ідентифікатор замовлення	integer	PK
	created_date	Дата додавання у кошик	timestamp	
	quantity	Кількість	integer	
	product_id	Ідентифікатор продукту	integer	FK
	user_id	Ідентифікатор користувача	integer	FK
auth_session_id	id	Номер відвідування	integer	PK
	user_id	Ідентифікатор відвідувача	integer	FK
	session_id	унікальний ідентифікатор відвідування	varchar(255)	

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Архітектура сайту

Сайт складатиметься з двох частин: адміністративної та користувацької. Різниця полягає в тому, що у адміністратора є можливість додавання, редагування та видалення вмісту категорій, продуктів та послуг.

Сам сайт буде складатися з таких сторінок:

- HOME
- ABOUT US
- CART
- SERVICES
- LOGIN
- REGISTRATION

У адміністратора крім цього будуть наступні сторінки:

- ADMIN TOOLS
- ADMIN PAGE OF PRODUCT CATEGORIES
- ADMIN PAGE OF PRODUCT
- ADMIN PAGE OF SERVICES

Архітектура веб-сайту визначає ієрархічну організацію сторінок вашого веб-сайту, яка відображається через внутрішні зв'язки. Метою цієї структури є полегшення користувачам пошуку інформації та сприяння зрозумінню зв’язку між різними сторінками для пошукових систем.[10]

Звичайна структура веб-сайту можна уявити як деревоподібний графік, де головна сторінка виступає коренем. Сторінки, до яких є посилання з головної сторінки, представляють собою гілки, і з кожної з них можуть виходити додаткові гілки. Поступово ці гілки з'єднуються одна з одною.

Відповідно до задачі даної практики на Рис. 3.1. зображена структура веб-сайту.

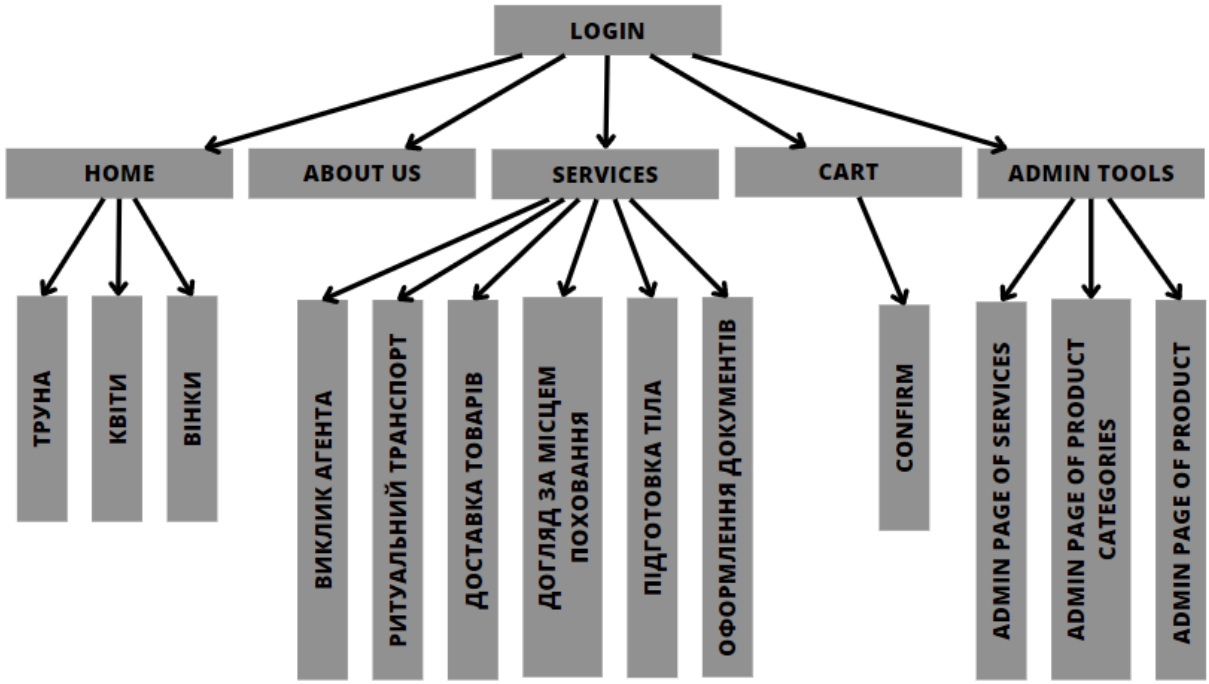

Рис. 3.1 — Структура веб-сайту

Але треба зазначити, що даний малюнок є дійсний для інтерфейсу адміністратора. У звичайного користувача не буде відвітлення “ADMIN TOOLS”.

3.2. Прототипування сайту

Прототипи веб-сайтів - це інтерактивні демонстраційні версії веб-сайту, що використовуються для збору відгуків від зацікавлених сторін на початкових етапах проекту, перед його фінальною розробкою. Створення прототипу веб-сайту є важливою частиною дослідження проекту.

Прототип веб-сайту може представляти собою будь-який макет або демонстрацію зовнішнього вигляду та функціональності веб-сайту після його запуску. Це може бути що завгодно - від простого паперового ескізу до інтерактивного прототипу на основі HTML, з яким можна взаємодіяти.

Прототип веб-сайту використовується для того, щоб отримати

зворотний зв'язок від користувачів, клієнтів або інших зацікавлених сторін проекту. Вони можуть пройти через прототип, переходячи по сторінкам, виконуючи дії та оцінюючи функціональність, щоб отримати реалістичне уявлення про те, як веб-сайт буде працювати у реальному середовищі. Це дозволяє зробити необхідні виправлення та покращення до того, як розпочати фінальну розробку веб-сайту.

Створення прототипу веб-сайту є важливою складовою процесу дослідження проекту, оскільки допомагає знизити ризики та непорозуміння, залучаючи зацікавлені сторони у визначення функціональних вимог та дизайну веб-сайту. [11]

Переваги прототипування:

1. Покращення розуміння вимог: Прототипи сприяють кращому розумінню вимог до функціональності та дизайну веб-сайту, допомагаючи уточнити потреби користувачів і зацікавлених сторін.
2. Візуалізація концепції: Прототипи веб-сайту дозволяють візуалізувати концепцію та структуру сайту, надаючи розробникам, дизайнерам та клієнтам чітке уявлення про зовнішній вигляд та роботу сайту.
3. Виявлення проблем: Прототипи допомагають виявити потенційні проблеми або недоліки на ранніх етапах розробки, що дозволяє їх виправити до фактичної реалізації веб-сайту.
4. Зменшення ризиків та витрат: Прототипування допомагає знизити ризики, пов'язані з розробкою веб-сайту, оскільки дозволяє виявити та вирішити проблеми на початкових етапах проекту, що забезпечує економію часу та ресурсів.

Недоліки прототипування:

1. Обмежена функціональність: Прототипи можуть мати обмежену функціональність у порівнянні з остаточним продуктом, оскільки їх основна мета полягає у візуалізації та перевірці концепції.

2. Додаткові зусилля: Розробка прототипу вимагає додаткових зусиль та часу порівняно з безпосередньою розробкою веб-сайту, оскільки потрібно створити модель зовнішнього вигляду та взаємодії.
3. Потенційні ризики неправильного розуміння: Існує ризик, що зацікавлені сторони можуть неправильно інтерпретувати або неправильно зрозуміти деякі аспекти прототипу, що може призвести до недорозумінь та неправильних рішень. [12]

Відповідні прототипи декількох основних сторінок можна побачити на Рис. 3.1. – Рис. 3.3.

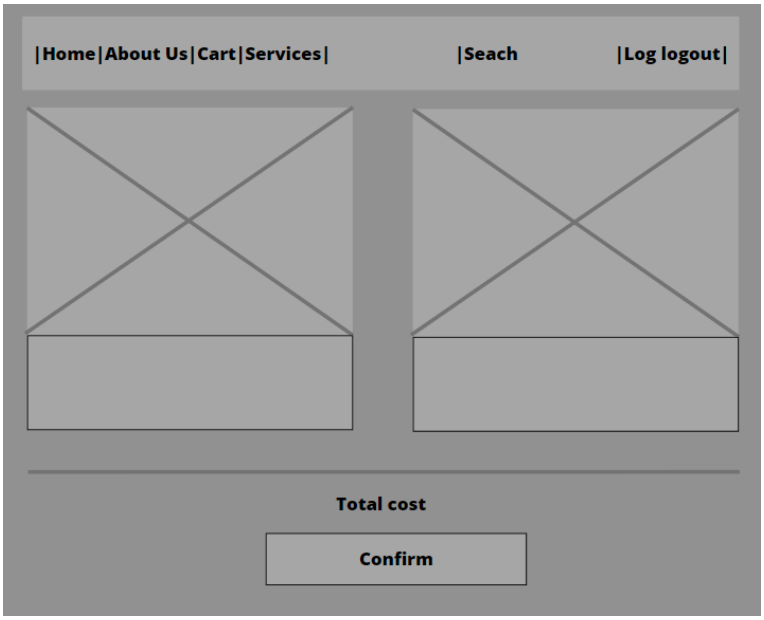

Рис. 3.1 — Прототип сторінки Cart

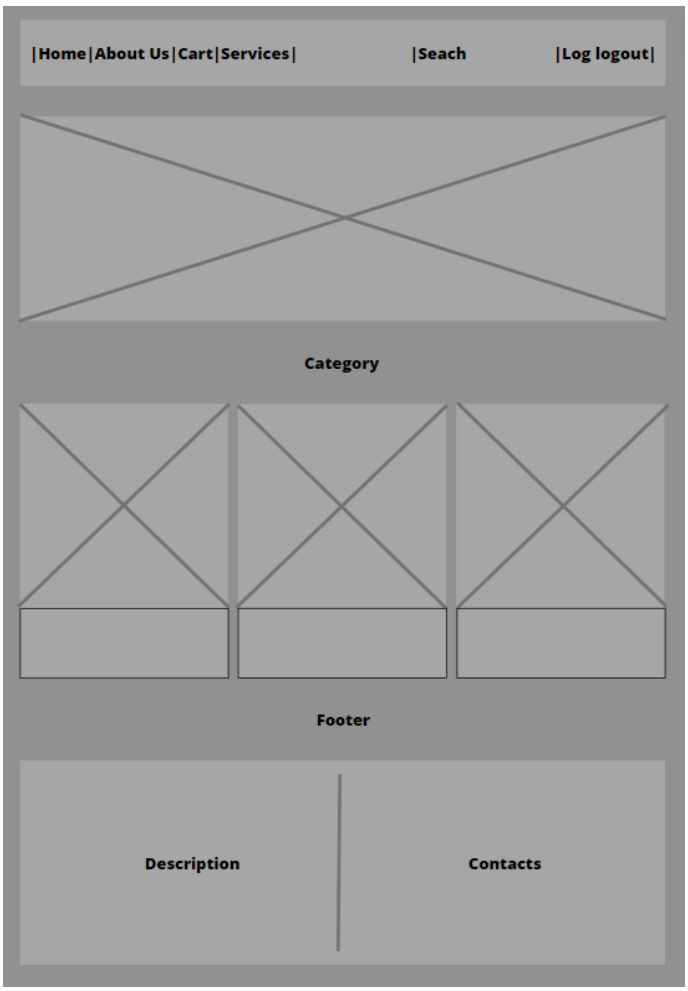

Рис. 3.2 — Прототип сторінки Home

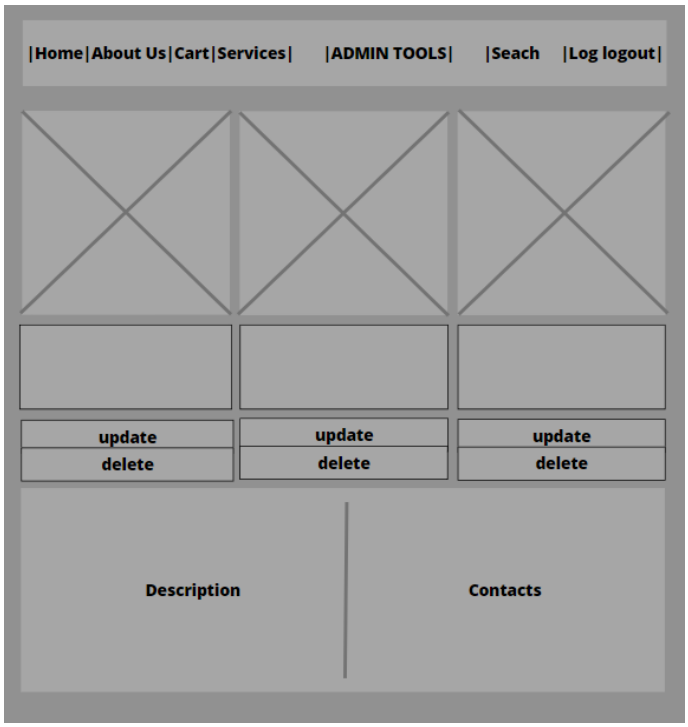

Рис. 3.3 — Прототип сторінки адміністратора

3.3. Створення да редагування категорій товарів

Одна з основних можливостей для адміністратора – це створення/редагування/видалення послуг, категорій товарів та, власно кажучи, самих товарів. Як було зазначено раніше, при авторизації від імені адміністратора з'являється вкладка, якої не бачить звичайний користувач (Рис. 3.4.).

Рис. 3.4 — Admin Tools

При переході на цю сторінку(Рис. 3.6) можна обрати один з трьох варіантів:

- Редагувати сторінку категорій
- Редагувати сторінку продуктів
- Редагувати сторінку послуг

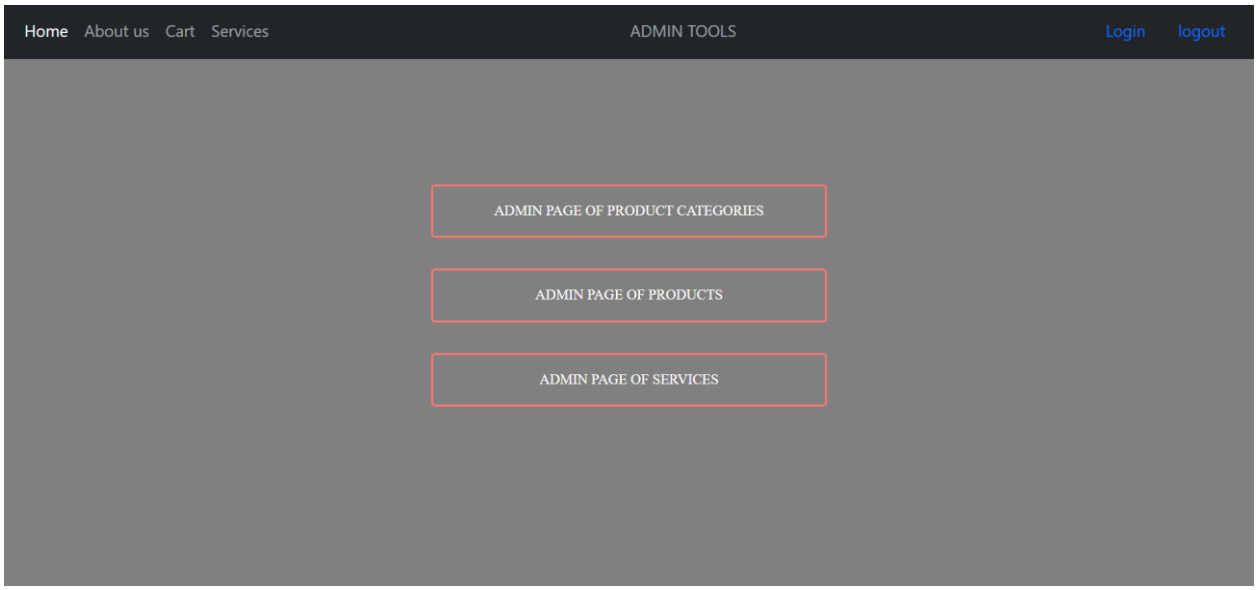

Рис. 3.6 — Сторінка Admin Tools

На даному етапі необхідно обрати сторінку в залежності від потреб. Наприклад, на сторінку ADMIN PAGE OF PRODUCTS (Рис. 3.7). Вигляд сторінок ADMIN PAGE OF PRODUCT CATEGORIES та ADMIN PAGE OF

SERVICES майже ідентична, як і інструкція редагування даних. Вгорі на даній сторінці можна побачити кнопку, для створення нового продукту, а під кожним товаром є кнопки Update та Delete, які знатні змінити або відповідно видалити дані.

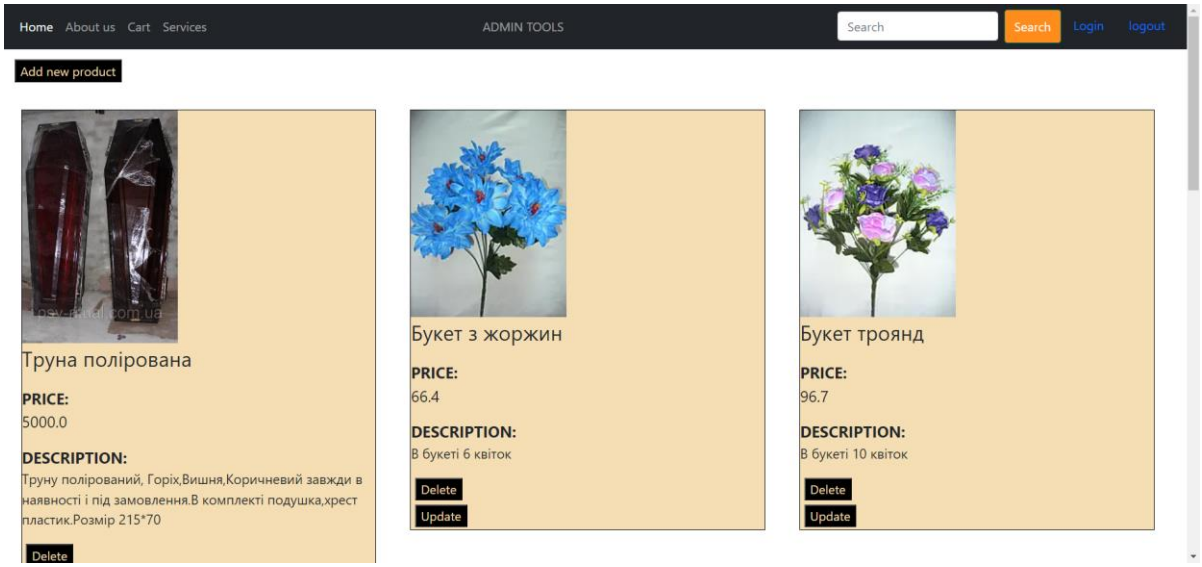

Рис. 3.7 — Admin Page Of Products

Якщо натиснути Add або Update, з’явиться форма (Рис. 3.8.) лише з тою відмінністю, що при створенні поля будуть пусті. Як можна було помітити в даній формі необхідно заповнити такі поля: ім’я, ціну, опис, URL зображення, а також номер категорії. Усі поля мають бути обов’язково заповнені, інакше з’явиться попередження з проханням вписати інформацію.

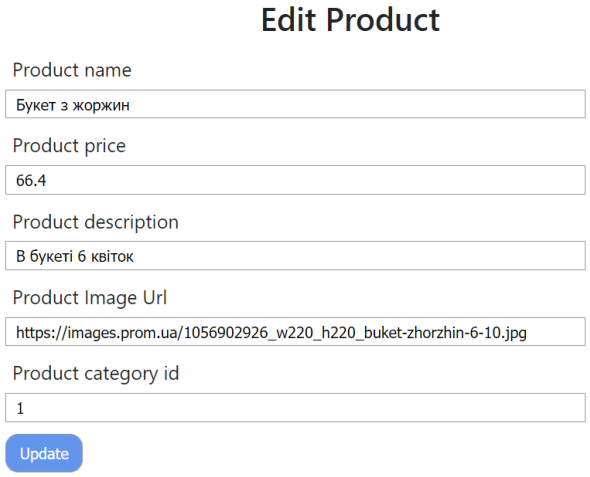

Рис. 3.8 — Форма заповнення даних про товар

У випадку з ADMIN PAGE OF PRODUCT CATEGORIES та ADMIN PAGE OF SERVICES необхідно буде заповнити такі поля:

- Назва
- Опис
- URL зображення

3.4. Користувацький інтерфейс

Інформаційна веб-орієнтована система агенції ритуальних послуг, що описується в даній роботі, розроблена з метою забезпечити користувачам комфортну взаємодію з продуктами та послугами. Вона надає такі функції, як вхід, реєстрація, головна сторінка з категоріями товарів, сторінка з конкретними товарами, кошик, послуги та сторінка "про нас".

Сторінка входу початкова для всіх користувачів. Тут вони можуть ввести свої облікові дані для доступу до своїх облікових записів (Рис. 3.9). Для створення нового облікового запису на сайті надається сторінка реєстрації (Рис. 3.10), де користувачі можуть вказати свої персональні дані та створити унікальні облікові записи.

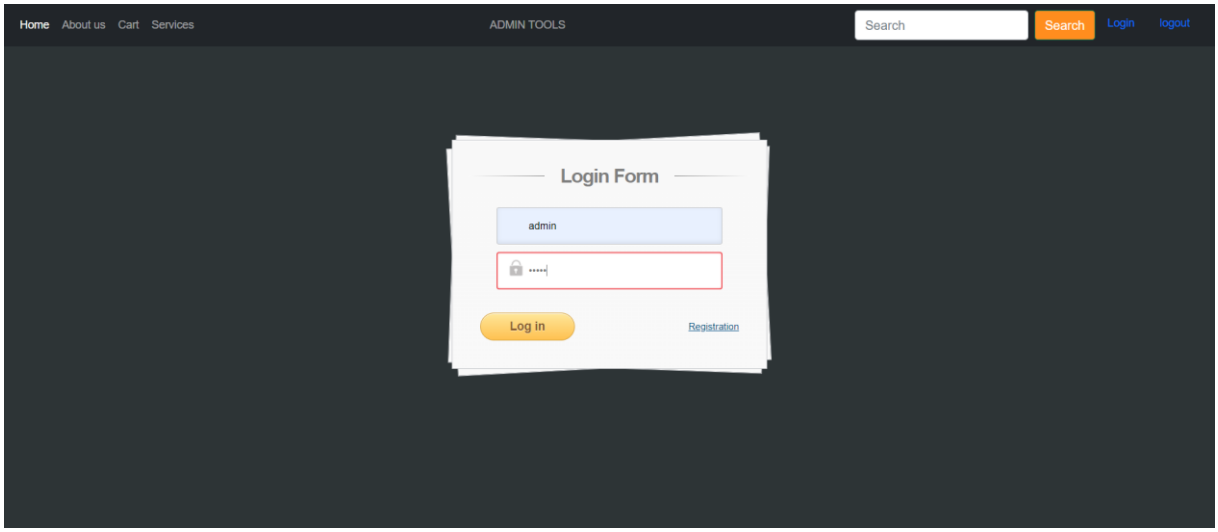

Рис. 3.9 — Сторінка входу

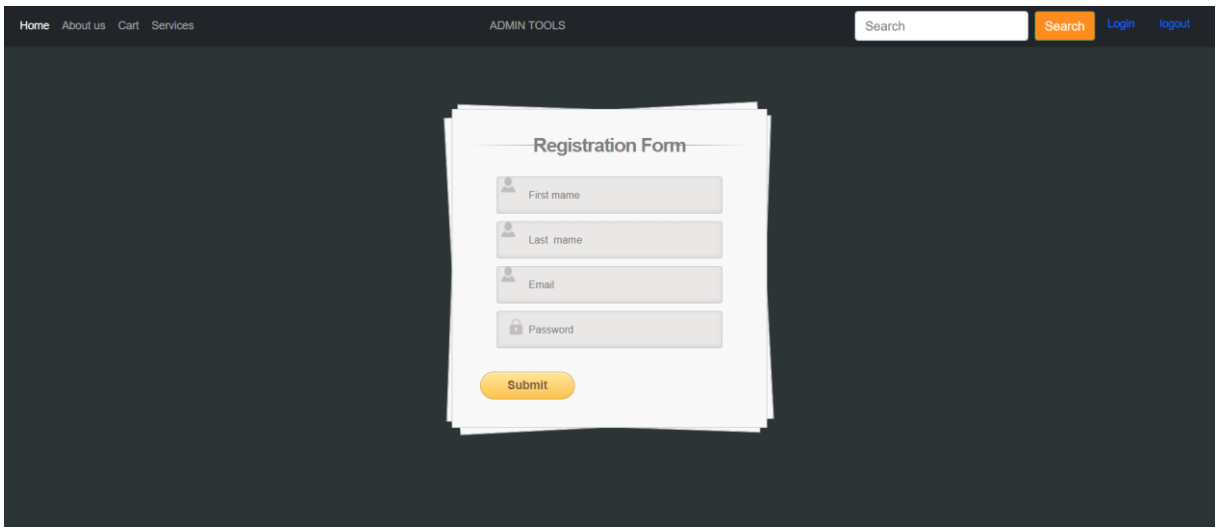

Рис. 3.10 — Сторінка реєстрації

Головна сторінка (Рис. 3.11) є центром уваги сайту та пропонує зручну навігацію за категоріями товарів. Кожна категорія відображається як привабливе зображення, пов'язане з відповідною тематикою.

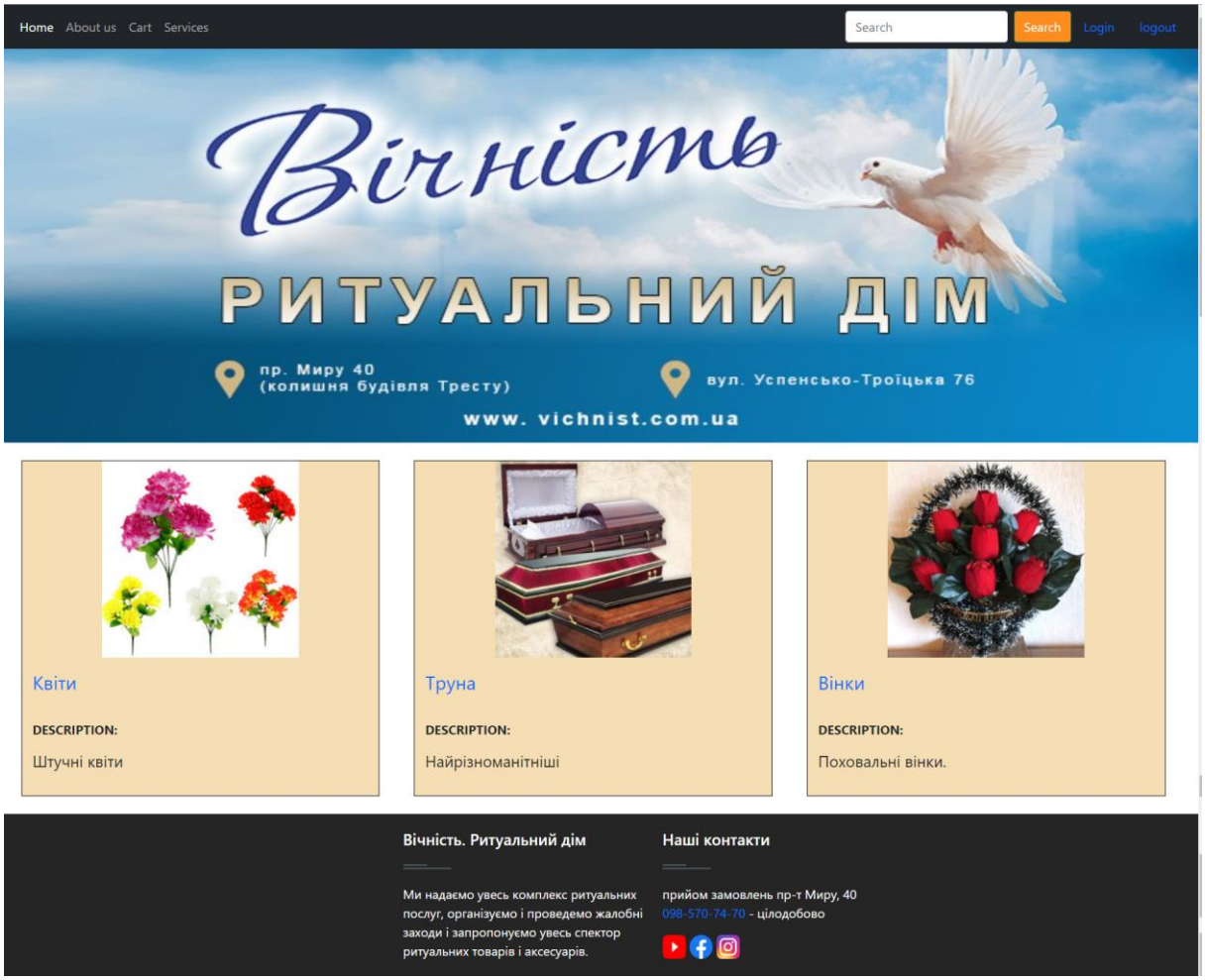

Рис. 3.11 — Головна сторінка

Після вибору конкретної категорії користувач перенаправляється на сторінку з товарами (Рис. 3.12.) в цій категорії. Тут відображаються різні продукти з відповідними описами, цінами та зображеннями.

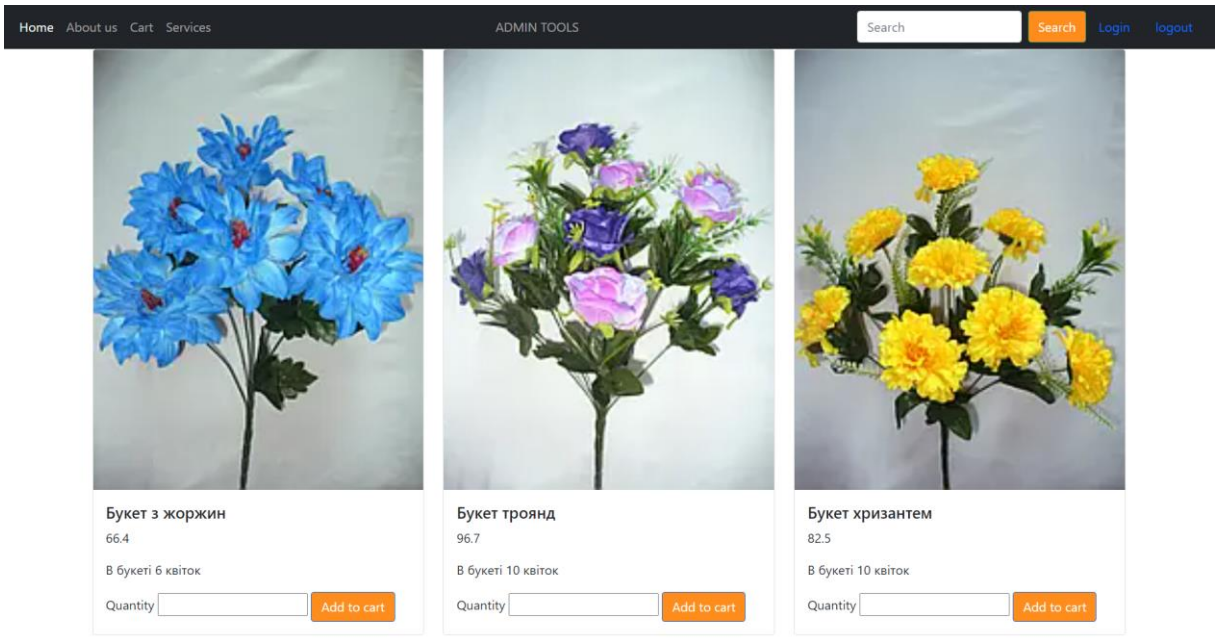

Рис. 3.12 — Товари

Кошик – це важливий елемент сайту, що дозволяє користувачам вибрані товари зібрати в одному місці для подальшої покупки. У кошику відображається список вибраних товарів. Обрані товари можна видаляти, а нижче, під переліком продукції, є інформація стосовно суми замовлення (Рис. 3.14).

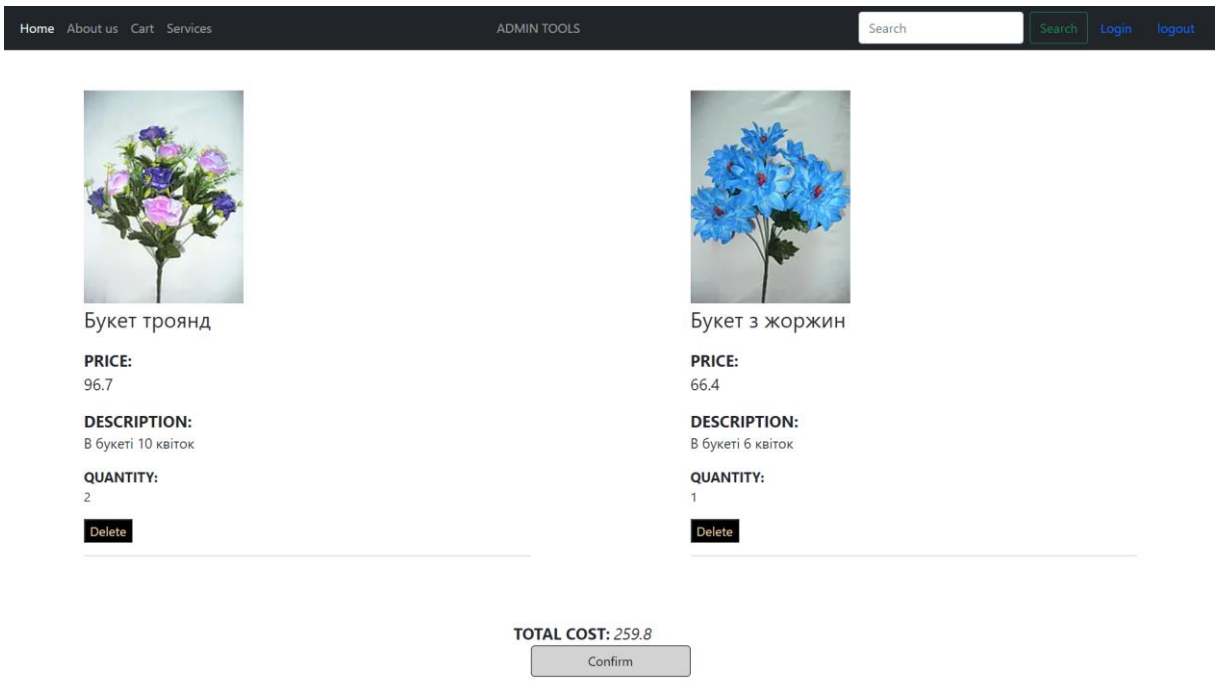

Рис. 3.13 — Кошик

Рис. 3.14 — Сума замовлення

Сервіси – це розділ, який пропонує додаткові послуги чи можливості користувачам (Рис. 3.15). Тут розміщені інформація про доставку, повернення, гарантії або інші додаткові сервіси, які пропонує сайт.

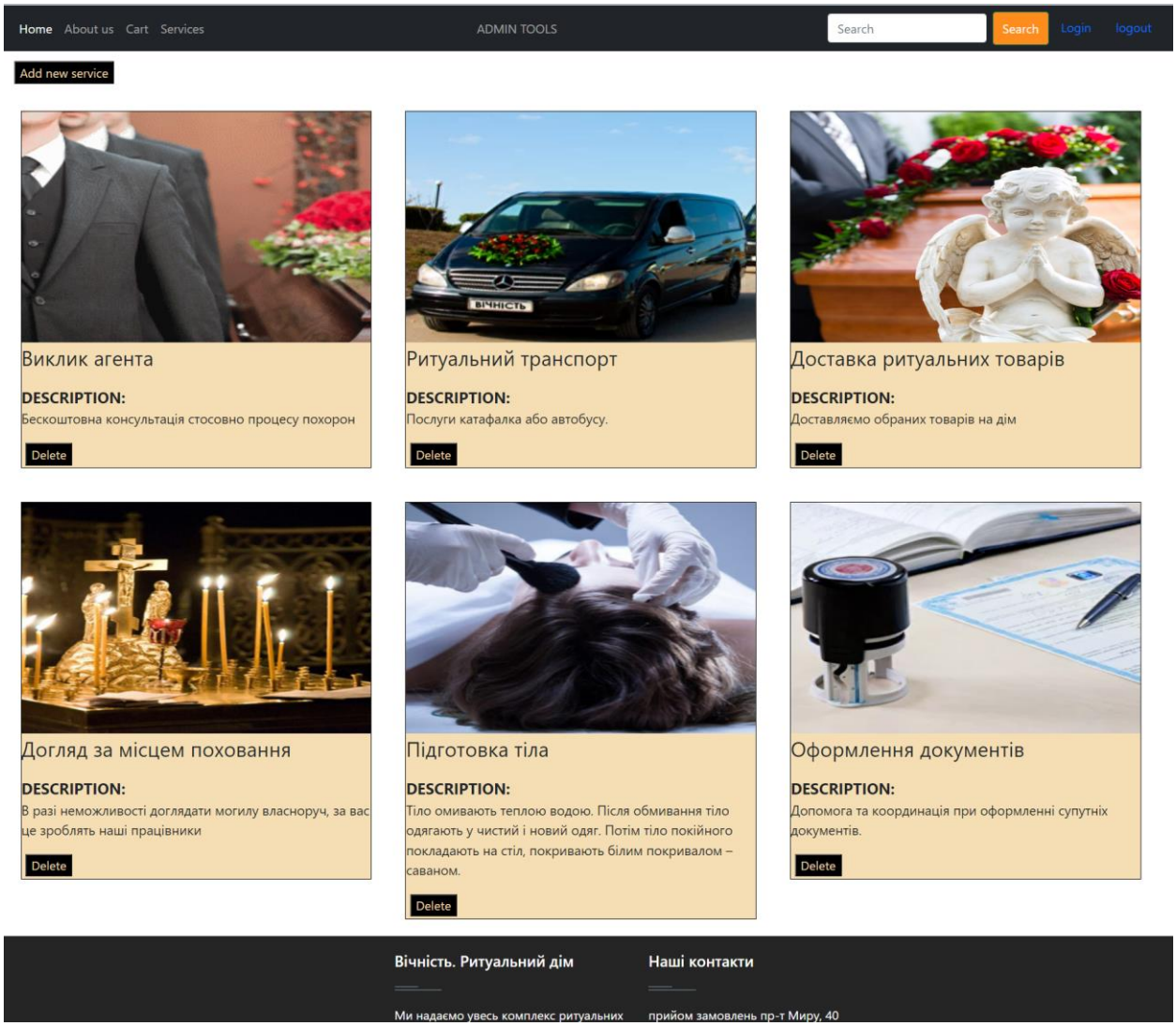

Рис. 3.15. — Сервіси

Сторінка "Про нас" (Рис. 3.16.) призначена для представлення загальної інформації. Тут є візитівка, основні пункти послуг, адреса магазину та контакти.

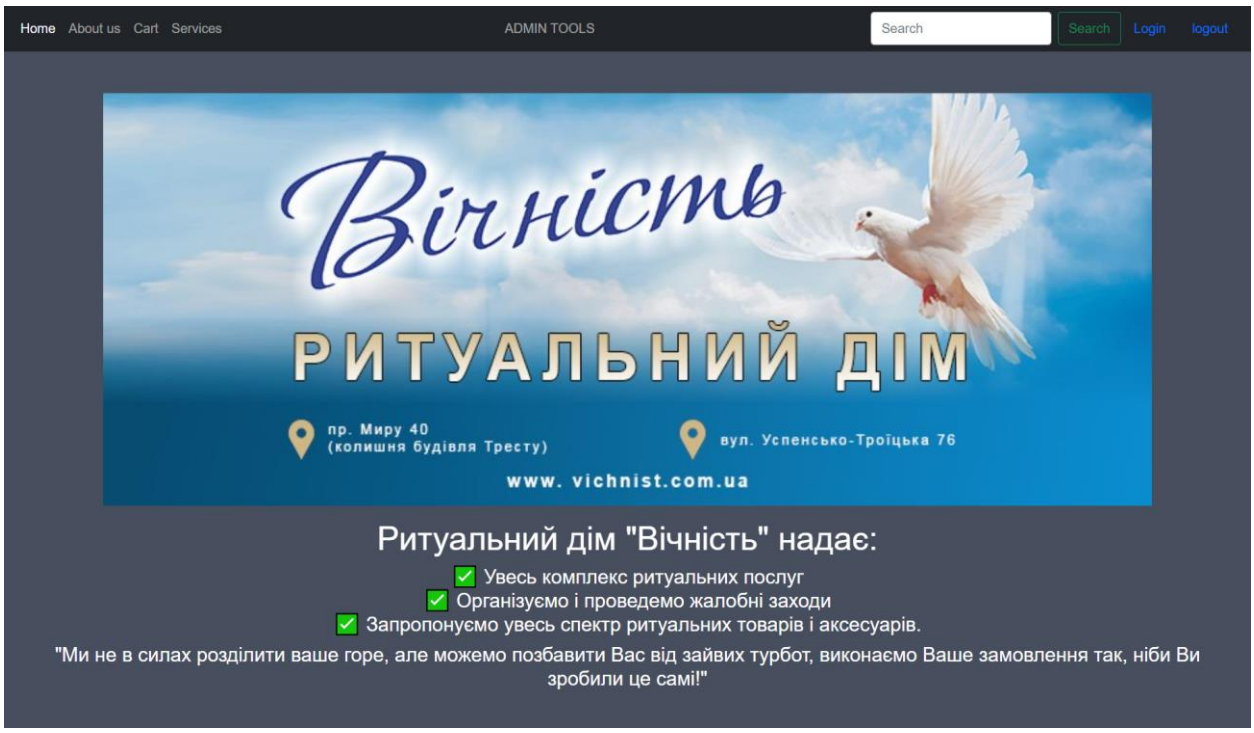

Рис. 3.16 — Про нас

3.5. Функціональне тестування

У рамках тестування програмного забезпечення функціональне тестування — це спосіб переконатися, що система функціонує відповідно до функцій, визначених бізнесом.

Важливо, щоб кожна функція програми працювала відповідно до вимог програмного забезпечення та того, як це програмне забезпечення призначене для кінцевого користувача.

Функціональне тестування має на меті перевірити зручність використання, умови помилок (чи відображаються відповідні повідомлення про помилки) і доступність системи для користувача. Це зручно та дозволяє QA виконувати як ручне, так і автоматичне тестування. [14]

Під час тестування було перевірено:

- Візуальну частину на відображення блоків, складових інтерфейсу та масштабування.
- Клікабельні елементи та коректний перехід за посиленнями

- Чи змінюється курсор при наведенні на клікабельні елементи.
- Коректність роботи головних функцій
- Користувацькі поля
- Правильність роботи кошика
- Особливості реєстрації та авторизації
- Додавання, видалення, зміна інформації на сайті
- Чи вбудована перевірка заповнюваних форм

ВИСНОВКИ

В представленій кваліфікаційній роботі створено інформаційну систему для агенції ритуальних послуг. Всі завдання, що потребували розвязання в постановці роботи виконані. Результатом роботи є інформаційне та програмне забезпечення, що відповідає вимогам, заданих в постановці завдання до проекту. Для досягнення поставлених цілей був проведений огляд інформаційного середовища та розглянуті основні переваги та недоліки, пов'язані з розробкою системи. Цей аналіз допоміг визначити методи та підходи, які будуть використовуватися в роботі.

У першу чергу, були обрані засоби реалізації системи. Для визначення функціональних вимог та взаємодії з користувачами були побудовані діаграми Use-case, що дозволили описати основні сценарії взаємодії. Для моделювання структури системи були використані діаграми DFD (Data Flow Diagram) та ERD (Entity-Relationship Diagram). На основі останньої були створені відповідні таблиці в базі даних для зберігання інформації.

Також було проведено процес прототипування, яке полягало у візуалізації та валідації концепції системи. Прототипи могли включати будь-які форми представлення, починаючи від паперових ескізів та закінчуючи інтерактивними прототипами, розробленими з використанням HTML та CSS. Після створення прототипу системи була розроблена архітектура, що включала в себе розподіл функцій та модулів системи, опис взаємодії між ними та визначення необхідних технологій для реалізації. На завершення було проведено перевірку коректності роботи системи та функціональне тестування. Це дозволило виявити та усунути можливі помилки та недоліки в ранній стадії розробки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Що таке веб додаток? | Блог WEBCASE. Webcase. URL: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/#f1 (дата звернення: 23.05.2023).

2. URL: https://www.itrobes.com/what-are-the-advantages-and-disadvantages-of-web-applications/ (дата звернення: 29.05.2023).

3. Роль технологій у сучасному веб-дизайні. ranktracker. URL: https://www.ranktracker.com/uk/blog/the-role-of-technology-in-modern-web-design/. (date of access: 29.05.2023).

4. What is Java Spring Boot? | IBM. IBM - Deutschland | IBM. URL: https://www.ibm.com/topics/java-spring-boot (date of access: 23.05.2023).

5. 6 Advantages and Disadvantages of Spring boot | Limitations & Benefits of Spring boot. HitechWhizz - The Ultimate Tech Experience. URL: https://www.hitechwhizz.com/2022/08/6-advantages-and-disadvantages-limitations-benefits-of-spring-boot.html (date of access: 23.05.2023).

6. UML Use Case Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (date of access: 23.05.2023).

7. Bilyk V. Data Flow Diagrams (DFD) Explained. ArtofBA. URL: https://www.artofba.com/post/data-flow-diagrams-dfd-explained (date of access: 23.05.2023).

8. Relational Database Benefits and Limitations (Advantages & Disadvantages) - DatabaseTown. DatabaseTown. URL: https://databasetown.com/relational-database-benefits-and-limitations/#:~:text=The%20main%20benefits%20of%20using,issue%20of%20speed%20can%20arise (date of access: 23.05.2023).

9. What is PostgreSQL? Introduction, Advantages & Disadvantages. Guru99. URL: https://www.guru99.com/introduction-postgresql.html (date of access: 23.05.2023).

10. Chi C. What is Website Architecture? 8 Easy Ways to Improve Your Site Structuring. HubSpot Blog | Marketing, Sales, Agency, and Customer Success Content. URL: https://blog.hubspot.com/marketing/website-architecture (date of access: 29.05.2023).

11. What is a website prototype?. experienceux. URL: https://www.experienceux.co.uk/faqs/what-is-a-website-prototype/#:~:text=Website%20prototypes%20are%20typically%20interactive,,%20including%20the%20end-users.

12. The Advantages & Disadvantages of Prototyping - Rapids Reproductions. Rapids Reproductions. URL: https://rapidsrepro.com/advantages-disadvantages-prototyping/ (date of access: 29.05.2023).

13. Entity Relationship (ER) Diagram Model with DBMS Example. *Guru99*. URL: https://www.guru99.com/er-diagram-tutorial-dbms.html#4 (date of access: 06.06.2023).

14. The Basics of Website Functionality Testing. *Leapwork*. URL: https://www.leapwork.com/blog/website-functionality-testing (date of access: 06.06.2023).

ДОДАТКИ
Лістинг програмного коду

```
package com.example.springmarket.service;
import com.example.springmarket.model.user.User;
import com.example.springmarket.repository.UserRepository;
import com.example.springmarket.response.ApiResponse;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    @Autowired
    BCryptPasswordEncoder bCryptPasswordEncoder;

    public ApiResponse signUp(User user) {
        ApiResponse response = new ApiResponse(true, "success
response");
        return response;
    }
    public List<User> getAll(){return userRepository.findAll();}

    public User findByEmail(String userEmail) throws Exception {
        Optional<User> optionalUser =
userRepository.findByEmail(userEmail);
        if (!optionalUser.isPresent()) {
            throw new Exception("user not present");
        }
        User user = optionalUser.get();
        return user;
    }

    public void save(User user){

user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()
));
        userRepository.save(user);
    }

    public boolean getUsersFromDB(User user) {
        Optional<User> optionalUser =
userRepository.findByEmail(user.getEmail());
        if(optionalUser.isPresent()){
```

```
            return true;
        }
        return false;
    }
}
package com.example.springmarket.service;

import com.example.springmarket.Dto.ProductDto;
import com.example.springmarket.model.Category;
import com.example.springmarket.model.Product;
import com.example.springmarket.repository.ProductRepository;
import lombok.AllArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
@AllArgsConstructor
public class ProductService {

    @Autowired
    private  ProductRepository productRepository;

    public void createProduct(ProductDto productDto, Category
category) {
        Product product = new Product();
        product.setName(productDto.getName());
        product.setPrice(productDto.getPrice());
        product.setImageURL(productDto.getImageURL());
        product.setCategory(category);
        product.setDescription(productDto.getDescription());
        productRepository.save(product);
    }

    public ProductDto getProductDto(Product product) {
        ProductDto productDto = new ProductDto();
        productDto.setDescription(product.getDescription());
        productDto.setImageURL(product.getImageURL());
        productDto.setName(product.getName());
        productDto.setCategoryId(product.getCategory().getId());
        productDto.setPrice(product.getPrice());
        productDto.setId(product.getId());
        return productDto;
    }

    public List<ProductDto> getAllProducts() {
        List<Product> allProducts = productRepository.findAll();

        List<ProductDto> productDtos = new ArrayList<>();
        for(Product product: allProducts) {
```

```
            productDtos.add(getProductDto(product));
        }
        return productDtos;
    }

    public void updateProduct(ProductDto productDto, int
productId) throws Exception {
        Optional<Product> optionalProduct =
productRepository.findById(productId);
        // throw an exception if product does not exists
        if (!optionalProduct.isPresent()) {
            throw new Exception("product not present");
        }
        Product product = optionalProduct.get();
        product.setDescription(productDto.getDescription());
        product.setImageURL(productDto.getImageURL());
        product.setName(productDto.getName());
        product.setPrice(productDto.getPrice());
        productRepository.save(product);
    }

    public Product findById(Integer productId) throws Exception
{
        Optional<Product> optionalProduct =
productRepository.findById(productId);
        if (!optionalProduct.isPresent()) {
            throw new Exception("product does not exist");
        }
        Product product = optionalProduct.get();
        return product;
    }

    public List<ProductDto> getAllProductsFromOneCategory(int
category_id) {
        List<Product> allProducts =
productRepository.getAllByCategory_Id(category_id);

        List<ProductDto> productDtos = new ArrayList<>();
        for(Product product: allProducts) {
            productDtos.add(getProductDto(product));
        }
        return productDtos;
    }

    public void deleteProduct(int id){
        productRepository.deleteById(id);
    }


}
package com.example.springmarket.service;
```

```
import com.example.springmarket.model.Category;
import com.example.springmarket.model.Product;
import com.example.springmarket.repository.CategoryRepository;
import lombok.AllArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@AllArgsConstructor
public class CategoryService {
    @Autowired
    private CategoryRepository categoryRepository;

    public List<Category> getAll(){return
categoryRepository.findAll();}
    public Category getById(int id){
        return categoryRepository.getById(id);
    }
    public Category save(Category category){
        return categoryRepository.save(category);
    }
    public void deleteById(int id){
        categoryRepository.deleteById(id);
    }

    public void updateById(int id, Category category){
        Category category1 = categoryRepository.getById(id);
        category1.setId(category.getId());
        category1.setName(category.getName());
        category1.setDescription(category.getDescription());
        category1.setImageUrl(category.getImageUrl());
        categoryRepository.save(category1);

    }


    public boolean findById(int categoryId) {
        return
categoryRepository.findById(categoryId).isPresent();
    }
}
package com.example.springmarket.service;

import com.example.springmarket.Dto.cart.AddToCartDto;
import com.example.springmarket.Dto.cart.CartDto;
import com.example.springmarket.Dto.cart.CartItemDto;
import com.example.springmarket.exceptions.CustomException;
import com.example.springmarket.model.Cart;
import com.example.springmarket.model.Product;
import com.example.springmarket.model.user.User;
import com.example.springmarket.repository.CartRepository;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Optional;

@Service
public class CartService {

    @Autowired
    ProductService productService;

    @Autowired
    CartRepository cartRepository;

    public void addToCart(AddToCartDto addToCartDto, User user)
throws Exception {

        // validate if the product id is valid
        Product product =
productService.findById(addToCartDto.getProductId());

        Cart cart = new Cart();
        cart.setProduct(product);
        cart.setUser(user);
        cart.setQuantity(addToCartDto.getQuantity());
        cart.setCreatedDate(new Date());


        // save the cart
        cartRepository.save(cart);

    }


    public CartDto listCartItems(User user) {
        List<Cart> cartList =
cartRepository.findAllByUserOrderByCreatedDateDesc(user);

        List<CartItemDto> cartItems = new ArrayList<>();
        double totalCost = 0;
        for (Cart cart: cartList) {
            CartItemDto cartItemDto = new CartItemDto(cart);
            totalCost += cartItemDto.getQuantity() *
cart.getProduct().getPrice();
            cartItems.add(cartItemDto);
        }

        CartDto cartDto = new CartDto();
        cartDto.setTotalCost(totalCost);
        cartDto.setCartItems(cartItems);
```

```
        return  cartDto;
    }



    public void deleteCartItem(Integer cartItemId, User user) {
        // the item id belongs to user

        Optional<Cart> optionalCart =
cartRepository.findById(cartItemId);

        if (optionalCart.isEmpty()) {
            throw new CustomException("cart item id is invalid:
" + cartItemId);
        }

        Cart cart = optionalCart.get();

        if (cart.getUser() != user) {
            throw  new CustomException("cart item does not
belong to user: " +cartItemId);
        }

        cartRepository.delete(cart);


    }
}
package com.example.springmarket.service;


import com.example.springmarket.model.user.AuthSessionId;
import com.example.springmarket.model.user.User;
import
com.example.springmarket.repository.AuthSessionIdRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.servlet.http.HttpSession;

@Service
public class AuthSessionIdService {

    @Autowired
    AuthSessionIdRepository authSessionIdRepository;

    @Autowired
    UserService userService;

    public User getUser(String session) throws Exception {
         AuthSessionId authSessionId =
authSessionIdRepository.findBySessionId(session);
         User user = authSessionId.getUser();
```

```
            if(user.equals(null)){
                throw new Exception("User is null");
            }
            return user;
        }

        public void save(User user, String sessionId) {
            AuthSessionId authSessionId = new AuthSessionId();
            authSessionId.setSessionId(sessionId);
            authSessionId.setUser(user);
            authSessionIdRepository.save(authSessionId);
        }

        public boolean findBySessionId(String sessionId) {
            AuthSessionId authSessionId =
    authSessionIdRepository.findBySessionId(sessionId);
            if(authSessionId == null){
                return false;
            }
            return true;
        }
    }
```