МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
(підпис)
грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія керування активністю онлайн-покупця з використанням моделі персоналізованих рекомендацій»
здобувача групи ІН.м - 32 Самсоненка Павла Євгеновича

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

Павло САМСОНЕНКО
(підпис)

Керівник,
в.о. завідувача кафедри,
кандидат технічних наук, доцент Ігор ШЕЛЕХОВ
(підпис)

Суми – 2024

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

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-32 Самсоненка Павла Євгеновича

1. Тема роботи: «Інформаційна технологія керування активністю онлайн-покупця з використанням моделі персоналізованих рекомендацій»
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи до 6 грудня 2024 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Аналітична частина 3) Практична частина 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 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	*Аналітична частина*		
3	*Практична частина*		
4	*Аналіз результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

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

АНОТАЦІЯ

Записка: 66 стор., 38 рис., 5 табл., 1 додаток, 22 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена дослідженню та аналізу сучасних розробок в галузі онлайн-купівлі, створення сучасного та функціонального веб-додатку, який буде поєднувати найкращі практики в розробці та задовільнить вимоги та потреби користувачів

Об'єкт дослідження — процес керування активністю онлайн-покупця.

Мета роботи — розробка інформаційної керування активністю онлайн-покупця з використанням моделі персоналізованих рекомендацій.

Методи дослідження —методи проєктування, тестування та оптмізації веб-додатків, методи аналізу активності онлайн-покупця.

Результати — Було виконано аналіз актуальності та аналіз актуальності інформаційної технології для онлайн-шопінгу, після чого було проаналізована та обрано тип необхідного веб-додатку для створення. Після цього було визначена структура веб-додатку та потреби його користувачів. Окремо від цього було проаналізовано можливі для додатку інтеграції та обрано потрібні.

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, ІНТЕГРАЦІЇ ЗОВНІШНІХ СИСТЕМ,
ВЕБ-ДОДАТОК ДЛЯ ОНЛАЙН-ШОППІНГУ, ВЕБ.

ЗМІСТ

ВСТУП

Інтернет-магазини для купівлі онлайн в наш час стали невід'ємною частиною сучасного бізнесу та споживчої культури. Чим більше розвиваються технології та інтернет, тим більшу можливість мають покупці у здійсненній онлайн покупок де б вони не знаходилися. При цьому існує велика кількість різноманітних ресурсів для кожного типу товару та послуги. Сфера онлайн-торгівлі наразі швидко розвивається і потребує оновлення чи створення сучасних веб-додатків.

Магістерська робота буде присвячена дослідженню та розробці сучасного веб-додатку для онлайн купівлі товарів, який дозволить покупцю здійснювати покупки з максимальною зручністю та ефективністю. Буде розглянуто різні аспекти, такі як проектування, розробка та інтеграція таких веб-додатків. Особлива увага буде приділена до сучасних технологій, користувацькому досвіду та забезпеченню безпеки даних користувача.

Об'єктом дослідження даної магістерської роботи є сучасний та багатофункціональний веб-додаток для онлайн закупівлі товарів, який буде давати можливість клієнтам з будь-якої точки країни замовляти та отримувати необхідні для них продукти. Цей додаток буде містити можливість адміністрацією сайту створювати товару, та покупцю замовляти та купувати його в онлайн режимі.

Предметом дослідження даної магістерської роботи є дослідження сучасних методів та підходів до розробки веб-додатків. Під час роботи буде досліджено сучасні тенденції та підходи до веб-розробки, разом з оглядом використаних мов програмування та інших технологій. Увага буде приділятися стороннім ресурсам, які можливо інтегрувати до веб-додатку задля покращення його функціоналу та полегшення юзер інтерфейсу. Окрім цього буде обрано тип товару який цей додаток буде поставляти.

Метою даної магістерської роботи є дослідження та аналіз сучасних розробок в галузі онлайн-купівлі, створення сучасного та функціонального веб-додатку, який буде поєднувати найкращі практики в розробці та задовільнить вимоги та потреби користувачів. Оцім цього буде досліджено практики продуктів аналогів разом з їх плюсами та мінусами.

Виконання мети розуміє під собою декілька послідовних кроків. Першим з цих кроків є дослідження актуальності даного продукту, разом з вибором тип продукту який буде продаватися на сайті. Другим кроком є аналіз аналогічних сайтів, знаходження їх плюсів та мінусів, висновки щодо того , що має бути присутнім у моєму додатку. Третім кроком є постановка задачі, яка включає в себе конкретні кроки необхідні для розробки цього продукту.

Окрім цього необхідно виділити окремо аналіз потрібних для додатку інтеграцій, безпеки додатку та проектування логіки сайту. Буде приведено теоретична інформація щодо використаних технологій та інтеграцій та проаналізовано можливі потреби користувачів та адміністрації сайту в майбутньому.

Виконання цих кроків веде до створення веб-додатку для онлайн-купівлі обраного товару, яким можливо буде користуватися і який буде дозволяти чітко адмініструвати усі надані послуги. Цей додаток повинен мати можливість розширятися та доповнюватися в майбутньому, бути простим у використанні та адмініструванні та використати останні технології, що повисить степінь його безпеки.

Ця магістерська робота є комбінацією теоретичних і практичних досліджень в галузі веб-розробки та електронної комерції. Вона розглядає як актуальні теми та тенденції в галузі онлайн-покупок, так і застосування сучасних технологій та методів для створення інноваційних веб-додатків.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Дослідження актуальності

Першим кроком при розробці будь-якого продукту є дослідження актуальності вибраної теми. Недостатньо лише створити гарний сайт, необхідно щоб в його функціоналі була нагальна потреба. Цей розділ присвячений дослідженню актуальності веб-додатків для онлайн-купівель, структурам таких сайтів та найголовніших переваг цих сайтів.

В сучасному світі розвиток технологій відбувається шаленими темпами, цей розвиток торкається більшості сфер повсякденного життя. З’являється тенденція спрощувати усі можливі складнощі за допомогою технологій та інтернету, люди створюють різноманітні сайти які мають змогу показувати відео-контент, давати онлайн-консультація та навіть заміняють походи до магазину.[1]

Сайти онлайн-купівлі стали невід’ємною частиною життя сучасної людини, навіщо ходити до магазину де ти можеш не знайти потрібний товар? Наразі існує безліч сайтів які спеціалізуються як на конкретних товарах, так і на будь-яких взагалі. Ці сайти дозволяють за допомогою кількох натискань миші купити будь-який товар і отримати його за декілька днів.

З кожним роком кількість людей які купують онлайн зростає і на сьогодні вона варується від 20 до 35%(рис. 1.1). Звісно повністю замінити звичайні магазини сайти ще не здатні, проте більшість специфічних товарів наразі купуються зазвичай онлайн.[2]

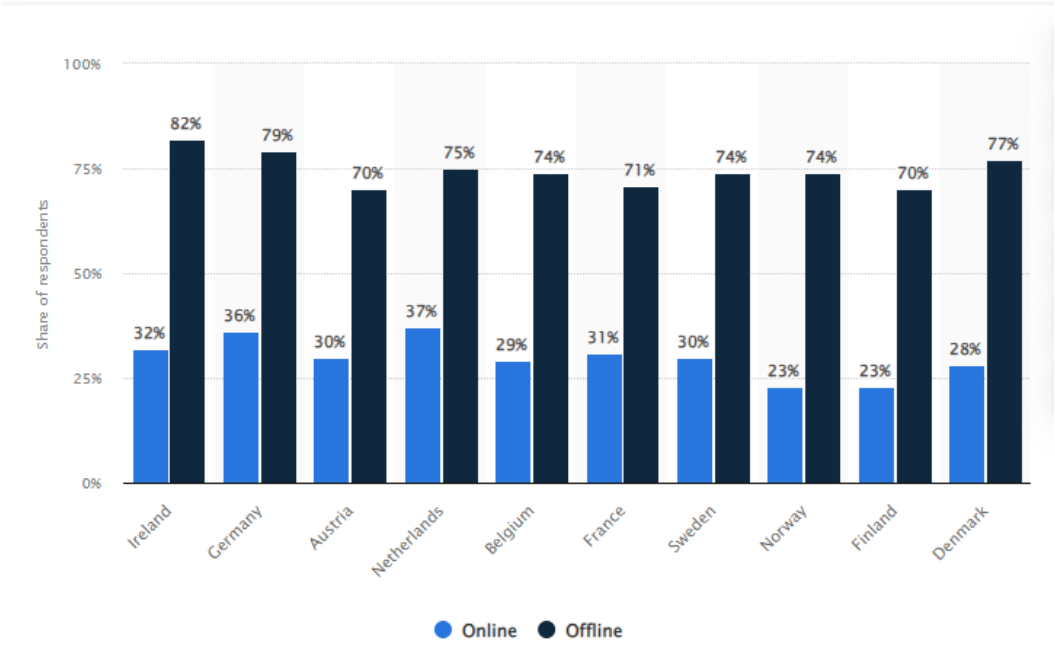

Рисунок 1.1 – Співвідношення людей які купують товари онлайн до звичайних магазинів

Онлайн магазини мають спільну структуру, що дозволяє користувачу замовляти на різних сайтах не звикаючи кожен раз до нового веб-сайту.

Основними спільними рисами сайтів є:

- Можливість зареєструватися на сайті, щоб спростити процес купівлі в майбутньому.
- Можливість сортування товару відносно потреб користувача (ціна, вага, бренд, тощо)
- Можливість передивлятися інформацію та ціну обраного товару з метою розуміння чи підходить він покупцю.
- Можливість передивлятися корзину з покупками з метою її зміни чи купівлі обраного товару.
- Можливість купівлі товару, яка включає в себе оплату та доставку(чи інформацію про отримання) товару.

Існує велика кількість причин чому люди поступово переходять до онлайн-купівлі, для розуміння чи актуальною є такі додатки на сьогоднішній день необхідно ознайомитися з найголовнішими з цих причин[3]:

1. **Зручність та ефективність**: Сайти для онлайн-покупок надають клієнтам можливість купувати без необхідності приходити до магазину. Це заощаджує час та зусилля, дозволяючи покупцю купувати товари та послуги зі свого будинку чи місця роботи.
2. **Доступність 24/7**: Онлайн-магазини доступні для покупців цілодобово. Це особливо важливо для тих, хто має нестандартні графіки роботи або має потребу в покупках у зручний для них час.
3. **Економія часу та Доставка**: Багато сайтів пропонують послуги доставки, що дозволяє покупцям заощадити час, необхідний для поїздки в магазин і назад. Окрім цього за допомогою фільтрації, клієнт може швидко знайти потрібний йому товар.
4. **Світовий Ринок**: Онлайн-покупки відкривають доступ до світового ринку, що дозволяє клієнтам купувати товари із-за кордону та взаємодіяти зі світовими брендами.
5. **Безпека та задоволення потреб**: Багато онлайн-магазинів забезпечують високий рівень безпеки транзакцій та захист особистих даних клієнтів. Це сприяє довірі покупців та виключає можливі ризи пов’язані з онлайн-купівлею.
6. **Великий асортимент**: Онлайн-магазини зазвичай пропонують величезний вибір товарів, включаючи продукцію з різних країн та велику кількість різноманітних брендів. Покупці можуть знаходити унікальні товари, які недоступні у їхньому регіоні.

З урахуванням цих аспектів стає очевидним, чому сайти для онлайн-покупок є актуальними та затребуваними в сучасному світі. Вони надають зручність, доступність, вибір та безпеку, що відповідає потребам та очікуванням сучасних споживачів.

1.2 Дослідження актуальних типів товарів

Окрім розуміння актуальності продукту, необхідно обрати тип товару, який буде продаватися на веб-додатку. Найочевиднішим рішенням здається розробка сайту, який буде мати можливість продавати декілька різних товарів, проте такі веб-додатки мають велику кількість недоліків. [4]

По-перше – ці сайти потребують величезну кількість людей для їх обслуговування, разом із великою кількість поставників товарів. Усі ці люди мають бути гарно ознайомлені з специфікацією товарів які вони будуть продавати, та мати достатньо навиків, аби адмініструвати ці товари на сайті. Така велика кількість людей може призвести до хаосу, що в свою чергу може погіршити реакцію клієнтів та збільшити час потрібний на пакування та відправку потрібного товару.[5]

По-другу – такі сайти не є зручними для клієнта, який хоче купити щось пов’язане з конкретною категорією товару. Така людина має зробити багато зайвих кроків для знаходження потрібного товару, що порушує декілька з головних причин актуальності сайтів для онлайн-купівель, а саме – зручність та економія часу.

Звісно існують виключення з цих правил, проте зазвичай це сайти на яких продавцем виступає не господар сайту, а інші люди які мають цей товар і збираються його продати. Такі сайти зазвичай є монополістами і розробка сайту який міг би конкурувати з ними дуже ускладняється цим фактом.[6]

Тому мною було обрано створення веб-додатку який спеціалізується на одному товарі і в якому продавцем виступає сам господар сайту – людина яка має певний товар (чи можливість його купівлі) і має бажання його продавати іншим людям.

Для вибору товару який буде продаватися на сайті було ознайомлено з найпопулярнішими товарами які продаються більш за все(рис. 1.2).

Рисунок 1.2 – Статистика найпопулярніших товарів

Як видно з рисунку 1.2 чотирма найбільш популярними категоріями є одяг, доставка їжі, меблі та косметика. Найпопулярнішим товаром є продаж одягу, проте саме через це вибір буде з трьох інших, адже більшість покупок виконуються на офіційних сайтах брендів цього одягу, що робить непотрібним розробку веб-додатку, адже покупець може замовити одяг з сайту самого бренду

Доставка їжі з ресторану містить під собою також велику кількість проблем, зазвичай ресторани мають свій сайт на якому можливо замовити їжу додому, а аналогічні сайти (наприклад Glovo) які спеціалізуються на доставці є дуже відомими і популярними сайтами, тому в цьому напрямі не має великої необхідності в створенні веб-додатку.[7]

Із двох останніх категорій я вирішив обрати косметику, через те що меблі хоча і замовляються онлайн, проте потребують велику кількість догляду під час перевезення, та спеціалістів які зможуть допомогти клієнту її зібрати, що є дуже проблемним, якщо ти не є керівником магазину, який має доступ до спеціалістів.

Косметичні товари є низкою з багатьох різноманітних категорій – від кремів для обличчя, до спеціалізованих предметів для догляду за собою. Однією з найпопулярніших категорій, потреба в якій існує в усіх людей, не зважаючи на їх стать чи вік є парфуми. Мною було обрано розробка веб-додатку для продажу, який буде спеціалізуватися на продажі парфумів, адже ця категорія є актуальною на даний момент, не має монополістів і потребує найменшу кількість людей не пов’язаних із сайтом.

1.3 Аналіз аналогів

Після дослідження актуальності та обрання типу товарів, що будуть продаватися я виконав аналіз аналогів. Цей аналіз включає в себе виділення окремих плюсів та мінусів сайтів-аналогів, порівняння їх функціональності та зручності в користуванні разом з висновками щодо цих даних.

Для аналізу я виділив 3 сайти, 2 з яких є суто українськими і один є загально-європейським. Аналіз цих ресурсів дозволить виділити головні недоліки та плюси які потребуються для веб-додатка.

Сайти які я проаналізував:

- Notino.ua
- Parfums.ua
- Fragrance.kiev.ua

Notino.ua

Notino – це компанія заснована в 2004 році і з того часу продає парфуми. Їх сайт має звичайний для такого типу сайтів вигляд, на головній сторінці можливо одразу побачити акції, лідерів продажу та інші панелі призначені для зацікавлення клієнтів(рис 1.3).

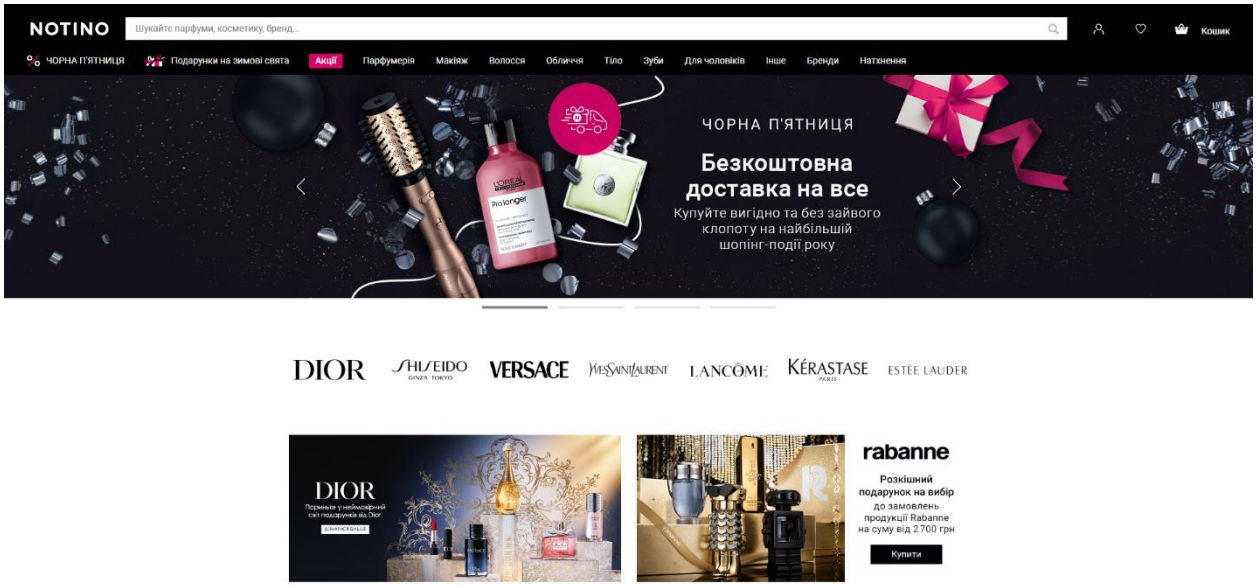

Рисунок 1.3 – головна сторінка сайту Notino

З точки зору звичайного юзера, сайт містить занадто багато інформації та є дуже перевантаженим, звичайний юзер який захоче купити парфуми буде шукати потрібні йому кнопки, оскільки вони ніяк не виділені і є занадто малими. Окрім панелей на головній сторінці, внизу сайту є футер в якому можливо побачити мапу сайту, контакти та важливу інформацію.

Для вибору парфумів на сайті існує велика кількість фільтрів які допоможуть швидко знайти потрібний людині товар(рис 1.4). Після цього людина має можливість подивитися про нього інформацію та замовити цей товар. Для замовлення потрібно зареєструватися.

Сайт виконаний в чорно-білих мінімалістичних тонах, що покращує його зовнішній вигляд, та полегшує його розробку та оновлення.

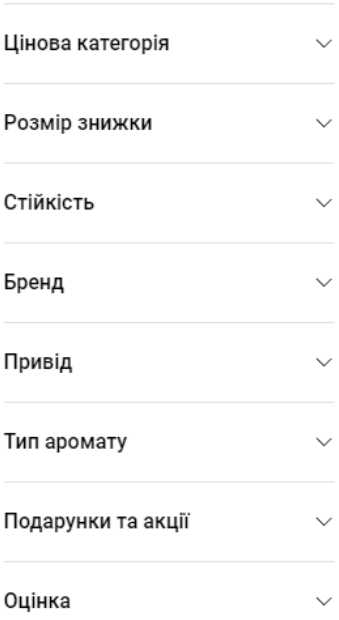

Рисунок 1.4 – Типи фільтрів на сайті

Окрім цього на сайті є кошик, особистий кабінет, та секція улюблених товарів, куди людина має можливість додавати товар, та потім його переглядати.

Плюсами цього сайту є:

- Великий функціонал – на сайті є все що потрібно для онлайн-магазину, від фільтрації та особистого кабінету, до окремих сторінок з акціями, кошику, тощо.
- Мінімалістичний та гарно оформлений дизайн – сайт виконаний в приємних для очей тонах, що в свою чергу може привабити покупців своїм зовнішнім виглядом.

Мінусами

- Сайт містить занадто багато агресивної реклами на першій сторінці, що може завадити людям у пошуку потрібної для них продукції.
- Дуже велика кількість функціоналу знаходиться у футері, що є поганою практикою і повинно бути розташованим на самому тілі сайту

Parfums.ua

Parfums.ua – це український сайт для купівлі парфумів. Виконаний він також в більш мінімалістичному стилі. На головній сторінці можливо одразу побачити увесь потрібний клієнту функціонал – особистий кабінет, фільтрацію товару, пошук по назві та можливість змінювати мову сайту(рис 1.6).

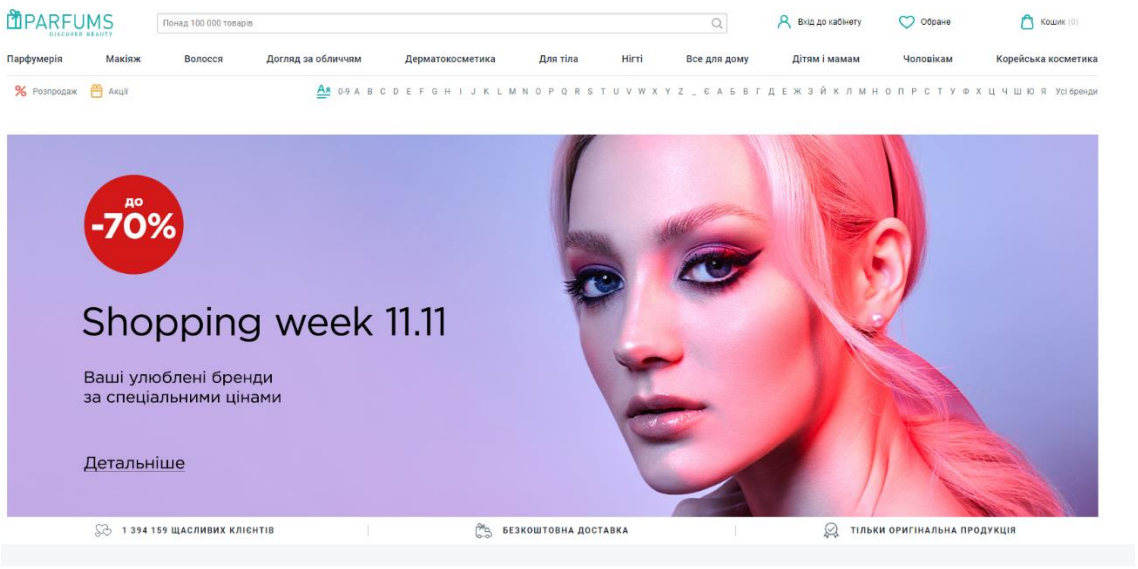

Рисунок 1.5 – Головна сторінка Parfums.ua

Аналогічно з попереднім конкурентом, цей сайт містить звичайний для такого типу веб-додатку набір функціоналу – особистий кабінет, фільтрація та пошук товару, кошик, обране та можливість замовляти. На відміну від Notino, цей сайт не містить великої кількості реклами на головній сторінці, та усі потрібні для клієнта функції виділені та збільшені у розмірі, що може значно покращити юзер досвіду.

Сайт виконано в більш теплих та світлих тонах, без можливості змінити його до більш темних кольорів, що є мінусом, адже темний дизайн є більш зручним та популярним у даний момент часу.

Аналогічно з попереднім сайтом цей сайт містить набір звичайних фільтрацій для парфумів, та окрім цього містить цікаво оформлений пошук по

брендам за першою літерою у назві, що може допомогти клієнту знайти потрібний йому бренд з головного екрану(рис 1.6).

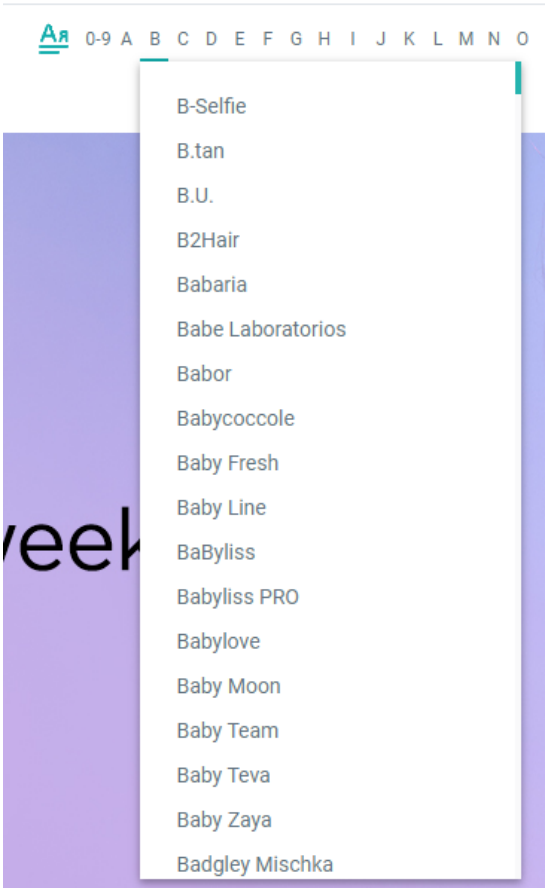

Рисунок 1.6 – Головна сторінка Parfums.ua

Плюсами цього сайту є:

- Зручний та зрозумілий інтерфейс – на сайті дуже легко орієнтуватися навіть менш досвідченим юзерам, що може сприяти захвату більшого кола аудиторії
- Наявність усього потрібного функціоналу – на сайті присутня можливість швидко знайти та обрати потрібний товар, разом з можливість швидко його купити та слідкувати за інформацією про замовлення.

Мінусами

- Сайт виконаний у занадто світлих тонах, що призводить до швидкого перевтомлення очей і погіршенню досвіду користування.

Fragrance.kiev.ua

Fragrance.kiev.ua – це сайт київської компанії, що займається парфумерією. Цей сайт оформлений в застарілому дизайні, та має занадто перевантажену головну сторінку на яку намагалися помістити майже весь функціонал. Окрім цього вітрини та акції оформлені в дуже незручному форматі – вони розташовані занадто щільно, що погіршує юзер-досвід.

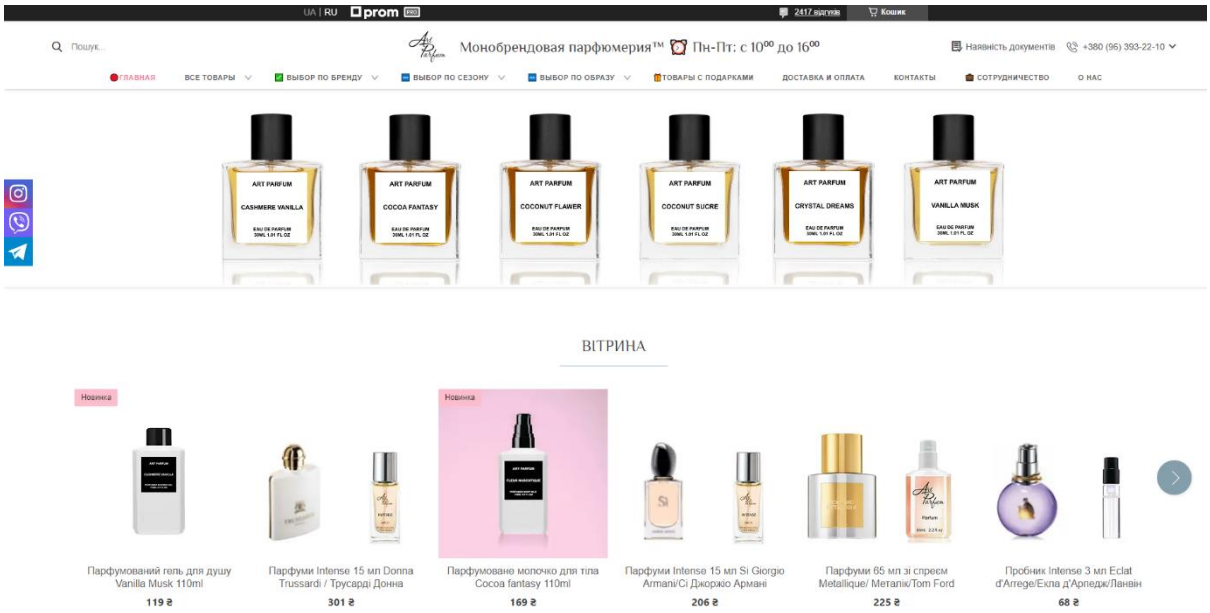

Рисунок 1.7 – Головна сторінка Fragrance.kiev.ua

З боку функціоналу тут також звичайний набір як і в усіх попередніх сайтів. Окремо потрібно виділити можливість замовляти товари без реєстрації – дана можливість спрощує процес купівлі товару.

Плюсами цього сайту є:

- Можливість купування товару без реєстрації.

- Можливість передивлятися відгуки до сайту та товарів, що може допомогти отримати довіру користувача.

Мінусами

- Погано оформлений дизайн – занадто багато функціоналу винесено на головну сторінку, усі елементи розташовані дуже щільно, що дуже погіршує досвід користування.

Виконавши аналіз трьох найбільш популярних сайтів, я зробив такі висновки:

1. Усі сайти для купівлі парфумів мають обов’язковий спільний функціонал – вітрина товарів на головній сторінці, можливість фільтрувати та шукати потрібний товар, можливість додавати потрібний товар до кошику та замовляти його.
2. Обов’язково потрібно мати можливість передивлятися інформацію щодо товару, контакти адміністрації, та можливість перевіряти статус замовлення.
3. Дуже важливим для сайту є його дизайн – погано оформлені сайти є менш популярними, та можуть дуже сильно ускладнити користування сайтом.
4. Окрім дизайну сайт повинен бути легким та зрозумілим у використанні – усі елементи мають бути чітко видимими, та головний функціонал має бути доступним одразу з головної сторінки.

Висновки які було зроблено при аналізі конкурентів допоможуть при розробці додатку, та зможуть покращити його якість.

1.4 Постановка задачі

Виконані в попередніх пунктах аналізи актуальності разом з аналізом конкурентів дозволили прояснити потрібні кроки для планування та розробки веб-додатку для купівлі парфумів.

Для розробки програмного продукту необхідно поставити та виконати такі задачі:

1. Визначитися з структурою роботи додатку, разом з потребами його користувачів.
2. Проаналізувати потрібні для додатку інтеграції, які зможуть задовільнити потреби користувачів та спростити розробку програмного коду
3. Створити опис логіки роботи веб-додатку – необхідно чітко розібрати та описати як має працювати і що має містити веб-додаток.
4. Зробити аналіз останніх технологій, які використовуються при розробці веб-додатків.
5. Створити серверну частину додатку, яка відповідає за виконання логіки сайту.
6. Створити клієнтську частину додатку, яка має відображати отримані з серверної частини дані, разом з усіма потрібними для користувача функціями.
7. Проаналізувати подальший розвиток сайту, його можливі проблеми та способи їх виправлення.

2 АНАЛІТИЧНА ЧАСТИНА

2.1 Аналіз структури додатку та потреб користувачів

Перше з чим потрібно визначитися перед розробкою додатку – це потреби користувачів та структура сайту. Для цього було використано аналіз та висновки з попередньої частини.[8]

Спочатку потрібно визначити користувачів веб-додатку. Користувачів існує два типи –

- Продавець – господар сайту, який хочу продавати товари, ця людина потребує можливості автоматизації процесу створення та продажу потрібних їх товарів.
- Покупець – людина яка має потребу в купівлі товару, вона хочу мати можливість швидко знайти потрібний для неї товар, та швидко його замовити

Ці два типи користувачів мають кардинально різні потреби, а отже і сайт має реалізовувати кожну з них. Для цього необхідно окремо виділити кожну потребу цих користувачів, для подальшого розуміння, який саме функціонал має бути присутнім на сайті.

Потребами продавця є:

- Можливість створити товар з повним його описом – для цього він потребує окремий акаунт та сторінку на якій лише цей акаунт зможе створювати новий товар.
- Можливість редагувати та видаляти створений товар – продавець потребує можливості зміни інформації щодо товару при потребі, для цього також буде використано окрему сторінку з його аккаунтом.
- Можливість дивитися інформацію щодо замовлень – продавцю потрібна окрема сторінка з переліком оформлених замовлень

Наступними необхідно розглянути потреби покупця:

- Користувач потребує можливості швидко знаходити потрібний йому товар.
- Користувач потребує можливості передивлятися більш детальну інформації щодо товару з його описом та детальним характеристиками
- Користувач потребує можливості додавання декількох товарів у кошик, задля подальшої покупки.
- Користувач потребує можливості оформити замовлення та передивлятися інформацію щодо нього

Після визначення потреб необхідно визначитися зі структурою додатку. Для цього необхідно виділити потрібні сторінки разом з їх наповненням та логікою. Для вибору потрібних сторінок було використано пункт з аналізом аналогів та обрано найбільш потрібні сторінки. Перелік потрібних для сайту сторінок разом з їх описом наведено далі.[9]

Головна сторінка – сторінка на яку в перше попадає користувач. Вона має містити можливість переходу до інших сторінок, рекламу продукту який продається, можливість для покупця чи продавця зайти до свого акаунту. Окрім цього має бути чітко відокремленим та одразу доступною можливість передивлятися свій кошик, можливість перейти до сторінки з пошуком потрібного товару та сторінки з контактами.

Сторінка реєстрації – сторінка яка дозволить покупцю створити його аккаунт, який в майбутньому буде використовуватися для авторизації та для спрощення оформлення купівлі. Ця сторінка має містити поля з потрібною для реєстрації інформацією, а саме імейл, ім’я та прізвище та пароль. Процес реєстрації має бути швидким та максимально зрозумілим для звичайного юзера. Окрім цього має бути має бути можливість перейти до сторінки з логіном для зареєстрованих користувачів.

Сторінка авторизації – сторінка на якій користувач має можливість авторизуватися на сайті. Для авторизації необхідно ввести лише імейл та пароль. Окрім цього на цій сторінці має бути можливість перейти на сторінку з реєстрацією, та можливість для юзера нагадати йому його пароль.

Сторінка з особистим кабінетом – сторінка на яку користувач чи адміністратор зможуть зайти після авторизації. Вона має містити різний функціонал відповідно до типу користувача. Звичайний користувач має мати можливість передивлятися та змінювати інформацію про себе та передивлятися інформацію щодо його замовлень. В свою чергу продавець повинен мати можливість додавати новий товар, передивлятися список усіх товарів, замовлень та юзерів з потрібною інформацією.

Сторінка з товарами – сторінка на який покупцю буде відображено весь товар. На цій сторінці обов'язково має бути присутня фільтрація товарів за категоріями та пошук. Після обрання потрібних фільтрів чи пошуком за назвою користувач має мати можливість додати товар до кошика, або перейти на сторінку з інформацією про товар.

Сторінка з детальною інформацією про товар – сторінка яка має містити інформацію щодо потрібного товару, на ній користувач має побачити детальний опис товару, його характеристику та фото. Окрім цього користувач може залишити коментар щодо товару разом з оцінкою.

Сторінка з контактами – остання сторінка, яка містить контактну інформацію необхідну для вирішення питань. Окрім цього в ній має бути інформація щодо часу роботи магазину.

Перелік цих сторінок з потребами користувача дозволяє виділити головну логіку яку потрібно описати для сайту, разом з можливістю описати структуру бази даних, яка буде використана.

Перше що було спроектовано – це чітка структура бази даних, яку буде в майбутньому використовуватися для збереження даних(рис 1.8)

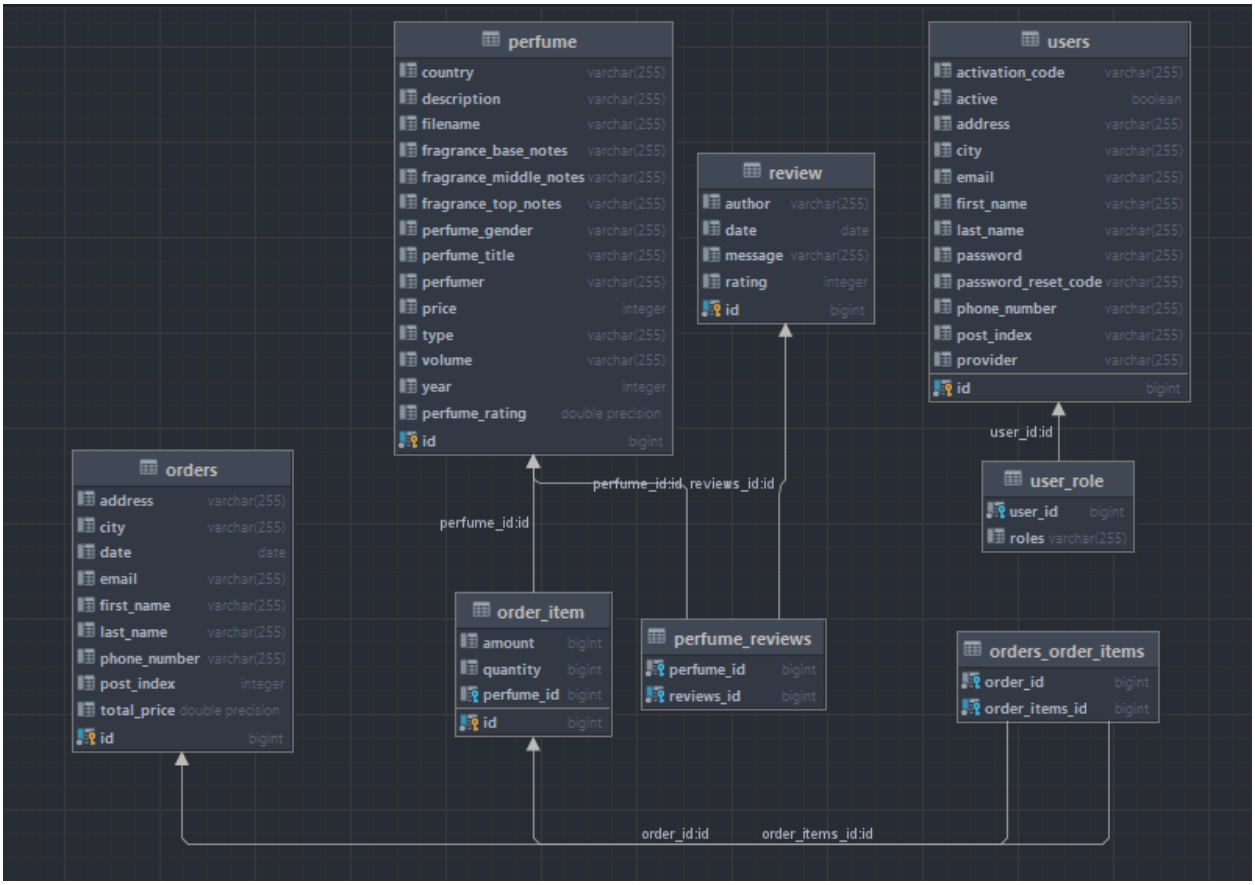

Рисунок 1.8 – Структура бази даних

Як можна побачити зі схеми структури бази даних, необхідно мати 5 головних таблиць а саме user, perfume, orders, order_item, review. Окрім цього необхідно проміжні таблиці perfume_reviewers, user_role та orders_order_items які будуть пов’язувати ці таблиці чи зберігати роль юзеру. Детально розберемо створену структуру далі.

Перша створена мною таблиця є user та user_role, ці таблиці потрібні для зберігання інформації щодо юзера(таблиця 2.1) та його ролі відповідно. Для збереження інформація про ролі використовується окрема таблиця, пов’язана з таблицею user по його ідентифікатору, в якій зберігається ідентифікатор юзера та назва його ролі.

Таблиця 2.1 – Структура таблиці user

Назва	**Опис**
activation_code	Код, який потрібен при активації акаунту
active	Інформація чи є аккаунт активним
address	Адреса
city	Місто
email	Імейл
first_name	Ім’я
last_name	Прізвище
password	Зашифрований пароль
password_reset_code	Код потрібний для скидання пароля
phone_number	Телефонний номер
post_index	Поштовий індекс
provider	Тип авторизації

Після цього було створено таблицю perfume(таблиця 2.2), яка містить інформацію щодо товару який буде продаватися. Ця інформація може бути використана для відображення, чи при створенні нового замовлення.

Таблиця 2.2 – Структура таблиці perfume

Назва	**Опис**
country	Країна виробник
description	Опис парфуму
filename	Назва файлу (для картинки)
fragrance_base_note	Базова нота парфуму
fragrance_middle_note	Середня нота парфуму
fragrance_top_note	Верхня нота парфуму
perfume_gender	Стать яка використовує парфум
perfume_title	Головна інформація про парфум
perfumer	Парфумер
price	Ціна
type	Поштовий індекс
volume	Тип авторизації
year	Рік виробництва
perfume_rating	Оцінка парфуму

І останньою великою таблицею яка була зроблена – це таблиця orders, яка використовується для збереження інформації щодо замовлень. Її структуру наведено в таблиці далі(Таблиця 2.3).

Таблиця 2.3 – Структура таблиці orders

Назва	Опис
address	Адреса
city	Місто
date	Дата
email	Імейл
first_name	Ім’я
last_name	Прізвище
phone_number	Телефонний номер
post_index	Поштовий індекс
total_price	Загальна ціна замовлення

Як видно з цієї таблиці, я не беру інформації з таблиці з юзерами, адже юзер зможе купувати парфуми навіть без реєстрації.

Ще важливими таблицями є review та order_item, які використовуються для зберігань даних відгуків та окремого предмету замовлення(таблиці 2.4 та 2.5 відповідно). Окрім цього ще буде згенеровано проміжні таблиці аби відобразити зв’язки один до багатьох і багато до багатьох. Ці таблиці містять лише інформації про ідентифікатор потрібних таблиць і не потребують окремого роз’яснення.

Таблиця 2.4 – Структура таблиці review

Назва	Опис
author	Автор
date	Дата
message	Відгук
rating	Рейтинг

Таблиця 2.5 – Структура таблиці order_item

Назва	Опис
amount	Загальна кількість парфумів
quantity	Кількість замовлених
perfume_id	Ідентифікатор парфуму

Створена структуру роботи додатку, разом з описом сторінок та користувачів допоможе при розробці додатку. Розроблені і описані таблиці в базі даних дозволять швидко створити коректну для збереження і отримання потрібних даних та дозволить уникнути проблем при розробці.

2.2 Аналіз технологій

Для розробки веб-додатку для початку необхідно ознайомитися з таким поняттям як web.[10]

Веб чи **World Wide Web**, ще відомий як «Всесвітнє павутиння», є ключова технологія інформаційної епохи. Це ресурс являє собою величезне сховище даних та інформації, доступ до якого може отримати кожен через інтернет. Веб було створено з метою обміну інформацією та забезпечення зв'язку між людьми. У цьому аналізі буде розглянуто ключові аспекти, такі як історія, архітектура, принцип роботи та вплив на сучасне суспільство.[11]

Історія Всесвітнього Павутиння налічує кілька десятиліть. Важливі етапи розвитку вебу включають:

- **Створення HTTP та HTML**: На початку 1990-х років Тім Бернерс-Лі розробив протокол HTTP та мову розмітки HTML, що поклало основу для створення веб-сторінок.[12]
- **Перший Веб-сервер та Веб-браузер**: Перший веб-сервер та веб-браузер були розроблені також в 1990 році, що дозволило користувачам переглядати веб-сторінки та передавати інформацію через Всесвітню мережу.
- **Зростання Популярності**: В 2000 роках веб став популярним серед науковців та освітніх закладів. Це призвело до зростання кількості сторінок разом з веб-сайтами. Ці сторінки використовувалися людьми з метою пошуку та обміну корисною інформацією
- **Пошук та Соціальні медіа**: Поява пошукових систем (наприклад, Google який став популярним в 2000 роках) та соціальних медіа (наприклад, Facebook який став популярним в 2006 роках) значно збільшила доступність та обмін інформацією в мережі. Ця інформація почала використовуватися людьми в повсякденному житті і сучасна людина не може уявити своє життя без цього.

Загальні принципи роботи всесвітнього павутиння включають у себе:

- **Децентралізація**: Веб не має центрального контролю та зберігається на серверах розташованих по всьому світі. Ніяка країна чи людина не має повного контролю над всією інформацією.
- **Інтертекстуальність**: Веб-сторінки можуть містити гіперпосилання на інші сторінки, створюючи інтертекстуальні зв'язки. Це дозволяє пов’язувати сторінки зі схожою тематикою і спрощує знаходження потрібної інформації(Гарним прикладом є Вікіпедія).
- **Відкритість**: Всі можуть створювати та публікувати веб-сторінки. Все що потрібно це сервер, де буде розташована твоя інформація. Для доступу до сторінки через пошукові системи необхідно відправити запит на індексування цієї сторінки.[13]

Окрім цього важливо знати загальну архітектуру вебу, яка включає в себе такі елементи:[14]

- **Веб-сервери**: Це спеціалізовані комп'ютери, які зберігають веб-сторінки та обслуговують запити від клієнтів.
- **Протокол HTTP**(Hypertext Transfer Protocol): це протокол який забезпечує передачу даних між клієнтами та серверами, що дозволяє завантажувати веб-сторінки та отримувати потрібні дані.
- **Мова розмітки HTML**(Hypertext Markup Language): це мова, яка використовується для створення структури та відображення веб-сторінок. На даний час є єдиною і не має ніяких альтернатив.
- **URL** (Uniform Resource Locator): це адреси веб-сторінок, які дозволяють користувачам знаходити та переходити до потрібних сторінок.
- **Клієнти (Веб-браузери)**: Веб-браузери, такі як Chrome, Firefox та Safari, дозволяють користувачам переглядати веб-сторінки та взаємодіяти з веб-серверами.

Після ознайомлення з поняттям веб, необхідно ознайомитися з архітектурою Rest, яка наразі є найбільш популярною при розробці веб-додатків.

REST (Representational State Transfer) - це архітектурний стиль, який є основою для створення безлічі сучасних веб-сервісів і веб-додатків. Для більшого розуміння далі буде ознайомлено з його походженням, архітектурними принципами та показана його роль в розробці веб-додатків.[15]

Історія REST тісно пов’язана з розвитком вебу. Цей стиль вперше був описаний Роєм Філдінгом у його дисертації "Архітектурні стилі та дизайн мережевого програмного забезпечення" яка була опублікована в 2000 році.

REST розроблявся як архітектурний стиль для мережевих систем, який виділяв важливість використання стандартних принципів та протоколів мережи(таких як HTTP). Його ідея була в створенні абстрактної моделі взаємодії, в якій кожен ресурс мав унікальну адресу, а користувачі взаємодіяли з цим ресурсів за допомогою HTTP запитів.[16]

На даний момент REST є невід’ємним компонентом розробки веб-сервісів та додатків. Його просто та зручність зробили його найпопулярнішим вибором, а архітектурні принципи зробили його надійним та ефективним, що і призвело до впровадження цієї архітектури в усіх галузях.

REST оперує такими ключовими принципами:

- **Ресурси** – REST працює лише з ресурсами(такими як веб-сторінки, зображення, тощо), які мають унікальну URL-адресу. Ці ресурси можуть бути представлені різними форматами, як правило JSON чи XML, вибір між якими здійснюється клієнтом.
- **Операції HTTP** – REST використовує стандартні HTTP методи, такі як **GET**(для отримання даних) **POST**(для створення нових даних) **PUT**(для доповнення даних) **DELETE**(для видалення даних).

- **Безпека**: RESTful взаємодії є безстановими, що значить що сервер не зберігає інформацію про стан клієнта, що забезпечую більший рівень безпеки.
- **Єдиний інтерфейс**: REST підтримує єдиний інтерфейс, що полегшує взаємодію та розуміння загальної архітектури і спрощує взаємодію. Цей інтерфейс припускає єдину можливість взаємодіяти з ресурсом незважаючи на тип девайсу чи додатку.[17]

Після ознайомлення з загальною інформацією, з'являється нагальна потреба у виборі методології розробки програмного продукту. Ці методології дозволяють полегшити процес розробки та зменшити затрати часу. Кожна з цих методологій має свої різноманітні принципи, які дозволяють виділити підходящий під конкретний продукт принцип розробки.

На даний момент трьома найбільш популярним методологіями є **каскадна**, **ітеративна та інкрементна модель** та **Agile.** Кожна з них буде проаналізована та обрана найбільш підходяща для розробки.[18]

Каскадна методологія(**Waterfall**) – це більш традиційний, послідовний підхід до розробки продукту. Процес розробки ділиться на фази, які ідуть одна за одною у визначеному порядку. Кожна наступна фаза наступає лише після завершення попередньої. Основними фазами є збір вимог, проектування, реалізація, тестування та супровід. Цей метод краще за все підходить до проекту з чіткими вимогами та з малою ймовірністю змін. Ця методологія може забезпечити високий рівень документації, проте є дуже чутливою до необхідності внесення змін під час розробки. [19]

Ітеративна та інкрементна методологія (Iterative and Incremental Development) – це методологія розробки, яка передбачає поділ проекту на невеликі ітерації, кожна з яких є повним циклом розробки. В рамках кожної ітерація в програмний продукт або додається новий функціонал, або покращується старий. Такий підхід дозволяє швидко реагувати на зміни в розробці та покращує зворотній зв’язок. Ця методологія є ідеальною для проектів де вимоги можуть змінювати, або не визначені із самого початку.

І остання модель **Agile -** ітеративна та інкрементна система управління проектами, орієнтованою на задоволення потреб замовника та швидку реакцію на зміни у процесі розробки. Дана система модель є покращеною ітеративною та інкрементною методологією, яка ідеально підходить для одиночного розробника. Окрім цього ідеї цієї методології дозволяють вносити зміни під час розробки, та розділити процес на ітерації, що ідеально підходить для мого проекту.

Для одиночного розробника Agile надає наступні переваги:

- **Ітеративність**: Agile дозволяє розробляти продукт у невеликих ітераціях (спринтах). Це дозволяє швидко створювати робочі прототипи та поступово покращувати їх.[20]
- **Клієнтська орієнтованість**: Agile ставить фокус на задоволення потреб клієнта. Ви можете безперервно збирати зворотний зв'язок та вносити корективи у процес розробки.
- **Адаптивність**: Agile дозволяє швидко реагувати на зміни та нові вимоги. Це особливо корисно, якщо ви створюєте онлайн магазин і хочете вносити зміни у відповідь на ринкові вимоги.

2.3 Аналіз можливих інтеграцій до веб-додатку

Інтеграції для веб-програм можуть значно розширити його функціональність і можливості. Ось деякі з найбільш популярних інтеграцій, які можна розглянути для веб-застосунку:[21]

- Але Платіжні системи: Інтеграція з платіжними системами, такими як PayPal, Stripe, Square або Braintree, дозволить вашим користувачам здійснювати онлайн-платежі за товари та послуги безпосередньо на вашому веб-сайті. Інтеграції такого типу дозволяють спрощувати виконання оплати за товар.
- Соціальні медіа: Можливість авторизації через облікові записи в соціальних мережах, а також інтеграція з платформами, такими як Facebook, Twitter, Instagram і LinkedIn, дозволить користувачам легко ділитися контентом та взаємодіяти з вашим додатком. Окрім цього це може допомогти спростити процес реєстрації чи авторизації додатку.
- API сторонніх сервісів: Інтеграція з API сторонніх сервісів, таких як Google Maps для геолокації, OpenWeatherMap для прогнозу погоди, або Amazon Web Services (AWS) для хмарного сховища, дозволить використовувати дані та функціональність цих сервісів у вашому додатку. Зазвичай такі інтеграції несуть функціональний характер, та допомагають додати до сайту новий функціонал не витрачаючи на нього багато часу.
- Електронна пошта та сповіщення: Інтеграція з поштовими сервісами, такими як Gmail або MailChimp, дозволить вам надсилати електронні листи та повідомлення користувачам про різні події у вашому додатку. Такі листування також можуть допомогти при необхідності активувати акаунт, згадати пароль чи інші проблеми пов’язані з аккаунтом.

- Аналітика та моніторинг: Використання аналітичних інструментів, таких як Google Analytics або Mixpanel, допоможе вам відстежувати поведінку користувачів, збирати потрібні для аналізу дані, які потім можливо використовувати для покращення якості сайту.
- Чат та зворотній зв'язок: Інтеграція з чат-сервісами, такими як Intercom або Zendesk, дозволить забезпечити зворотний зв'язок з користувачами. Зазвичай такі чат-сервіси використовуються в готових продуктах для додавання можливості онлайн-консультацій.
- Інтеграція з базами даних: Взаємодія з різними базами даних, такими як MySQL, PostgreSQL, MongoDB або Firebase Realtime Database допоможе зберігати і оновлювати дані в реальному часі. Для спрощення роботи існує безліч онлайн сервісів, які мають можливість розвернути бази даних на їх серверах.[22]
- Інтеграція з CMS: системами керування контентом (CMS) дозволяють інтегрувати такі додатки як WordPress або Drupal, які дозволять спростити процеси розробки, керування та оновлення цього контенту.
- Системи аналізу даних: Інтеграція з інструментами для аналізу даних, такими як Tableau, Power BI або Google Data Studio, дозволяю аналізувати отримані з програми дані, робити статистику, слідкувати за життєвим циклом програми та багато іншого.

Дані інтеграції дозволяють пришвидшити розробку та покращити досвід юзера на сайті, мною було обрано використання електронної пошти – а саме використання **Gmail**, інтеграція **AWS** для зберігання картинок та інтеграції бази даних PostgreSQL, яка була розвернена на безкоштовному онлайн хостингу **ElephantSQL** та **reCAPTCHA**.

Перед початком практичної частини необхідно ознайомитися з деталями цих інтеграцій необхідно більш детально про них дізнатися.

Спершу розглянемо інтеграцію **Gmail**, ця інтеграція може мати низку переваг, та є обов'язковою для реєстрації користувача. При реєстрації нового користувача необхідно переконатися що він зможе якось керувати своїм аккаунтом при загубленні своїх даних. Для цього використовується сервіс імейлів, які будуть відправляти йому на пошту інструкції як відновити його аккаунт.

Окрім цього інтеграція імейлу дозволить зменшити ризик спам атаки, адже для активації акаунту можливо додати необхідність переходу за спеціальним посиланням, яке буде активувати аккаунт, що в свою чергу може запобігти створенню великої кількості аккаунтів зловмисником.

В майбутньому ця інтеграція може дозволити створювати спеціальні розсилки, які будуть містити цікаву інформацію, або рекламу нового товару. Інтеграції такого роду дозволять інформувати клієнта не зважаючи на те, що він не заходить на сайт, та можуть допомогти продавцю в поширенні його товару.

Наступна використана мною інтеграція це **AWS**, а саме його можливість зберігати файли в хмарному сховищі. При створенні інформації про товар, необхідно надати користувачу можливість візуально побачити його, для цього необхідно зберігати його фотографію. Основною проблемою при збереженні фотографій є те, що їх дуже складно зберегти у базах даних, вони мають велику кількість обмежень і можуть дуже сильно сповільнити програму. Для уникнення таких ситуацій зазвичай використовують одне з двох рішень – зберігання фалів локально на сервері, або використання хмарного сховища.

В моєму випадку було обрано використання хмарного сховища, адже таке рішення по-перше дозволить зекономити на сервері (не потрібно брати зайве місце для картинок), та дозволить швидко зберегти ці фотографії в AWS, звідки їх буде зручно отримувати.

Третьою важливою інтеграцією є використання бази даних PostgreSQL , а саме розміщення її на онлайн- хостингу.

Розміщення баз даних можливо і локально, проте аналогічно із попереднім пунктом, воно потребує окремо виділених ресурсів і може збільшити складність керування цими даними. Використання онлайн сервісів зможе допомогти зекономити ці ресурси та вони зазвичай містять більш розширений функціонал, який дозволить швидко візуалізувати дані, створити бекапи та багато інших потрібних речей.

Мною було обрано **ElephantSQL** як безкоштовну альтернативу, цей хостинг не є кращим вибором збоку його ефективності чи швидкості, проте він є безкоштовний і в майбутньому може бути замінений на платний аналог з більшим функціоналом. Даний сервіс буде використовуватися для керуванням моєю базою даних, та саме в ньому буде зберігатися мої дані, що може допомогти уникнути проблем у випадку несправності на сервері з хостингом веб-додатку.

Останньою потрібною інтеграцією на сайті є будь-яка капча. Вона дозволяю зменшити ризик спам атак на сайт під час важливих операцій. Капча створює спеціальні завдання які можуть пройти лише люди і які дуже складно вирішити комп’ютеру, що дозволить мені додати її під час реєстрації та запобігти створенню акаунтів за допомогою простих скриптів.

Мною було обрана остання технологія розроблена гуглом, а саме – **reCAPTCHA**. Вона дуже сильно відрізняється від звичайної капчі, яка зазвичай створювала необхідність введення символів з картинки чи вибору картинок співпадаючих з текстом. Ця технологія на перший погляд є дуже простою, адже все що необхідно – це натиснути на чекбокс «я не робот», а сервіс сам проаналізує отримані дані(такі як швидкість курсору, його шлях, тощо) та зробить висновок чи була це реальна людина. Такий підхід дозволяє ефективніше відокремлювати роботів від людей, адже комп’ютери вже здатні розпізнавати символи чи картинки краще за людей.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Аналіз задачі

Для розробки програмного коду необхідно виконати декілька задач по ініціалізації інтеграцій. Ці задачі мають створити потрібні інстанції, які потім будуть використовуватися при розробці програмного продукту.

Ці задачі включають у себе:

1. Ініціалізація екземпляру бази даних в ElephantSQL.
2. Ініціалізації нового кошику в AWS.
3. Ініціалізація та налаштування імейлу для розсилки.
4. Підготовка усього необхідного для використання reCAPTCHA**.**

Виконання цих кроків дозволить перейти до розробки серверної та клієнтської частини, які разом мають створити потрібний веб-додаток.

Розробка серверної частини включає у себе:

1. Розробка потрібних для юзера контролерів.
2. Підключення усіх інтеграцій.
3. Розробка логіки програмної реалізації.

Після чого необхідно розробити клієнтську частину, яка має містити графічний інтерфейс для роботи з додатком та логіку роботи з серверною частиною

3.2 Ініціалізація інтеграцій

Першим було ініціалізовано базу даних в ElephantSQL. Для цього необхідно спочатку зареєструвати акаунт, а потім натиснути на кнопку створити новий екземпляр (рис 3.1)

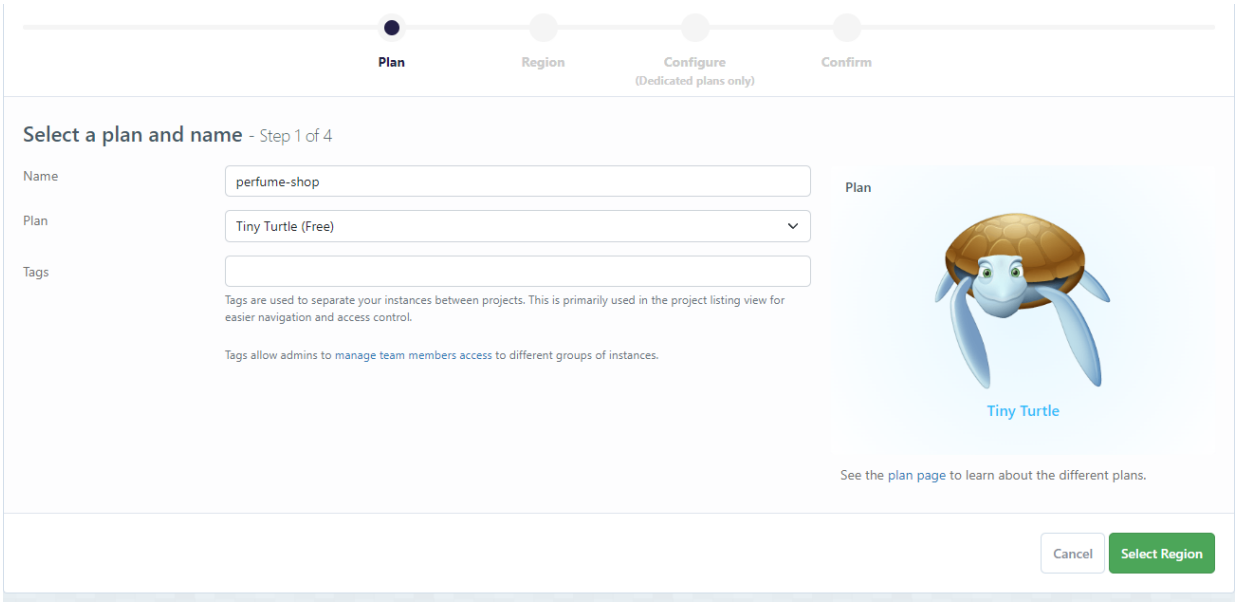

Рисунок 3.1 – Створення нового екземпляру

Після чого необхідно обрати регіон та створити цей екземпляр(рис 3.2). Правильно обраний регіон дозволить зменшити затримку при роботі з цією базою даних.

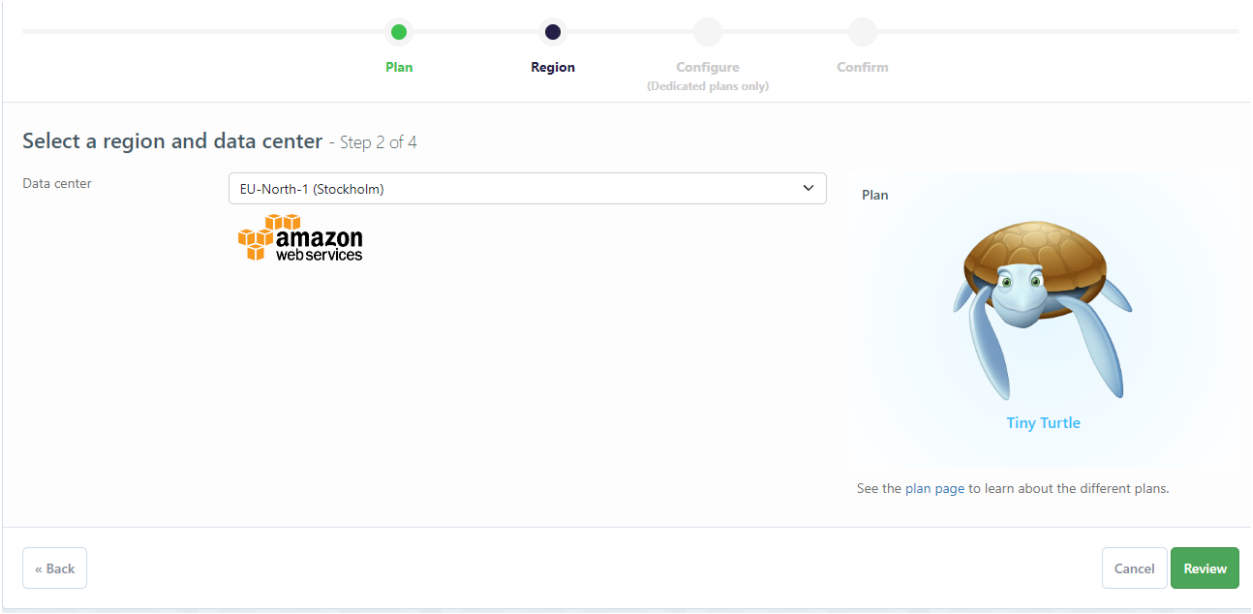

Рисунок 3.2 – Вибір регіону

Результатом цих дій має стати створений екземпляр бази даних, в якому можливо отримати необхідні дані для підключення(рис 3.3). Ці данні будуть використані при розробці серверної частини для зберігання потрібної інформації.

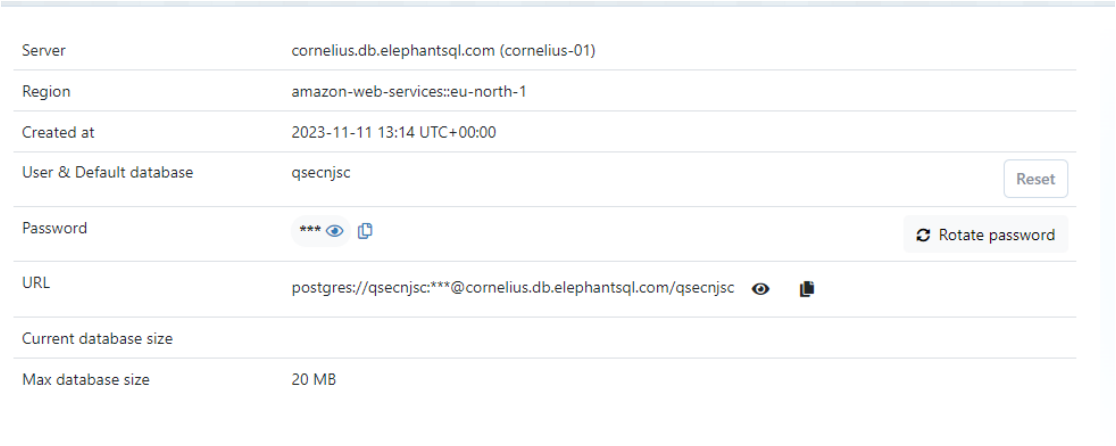

Рисунок 3.3 – Створений новий екземпляр

Наступним кроком необхідно ініціалізувати кошик в AWS. Для цього спочатку необхідна зареєструватися а потім авторизуватися на офіційному сайті AWS. Після чого в пошуку необхідно ввести S3 та обрати відповідний пункт(рис 3.4).

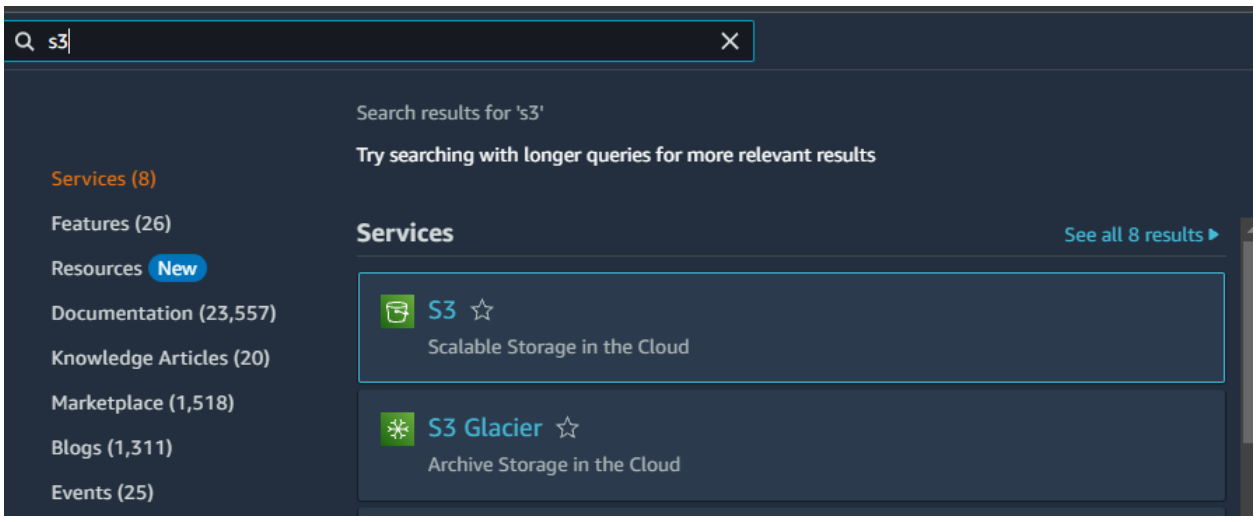

Рисунок 3.4 – Вибір потрібного пункту.

Після чого необхідно обрати пункт створення кошику (create bucket) – рис. 3.5. Обрання цього пункту перекине до меню з створення кошику в якому необхідно обрати потрібну конфігурацію (рис. 3.6 та 3.7 відповідно). Такі опції дозволять нам програмно завантажувати файли до цього кошику, та отримувати доступ до цих файлів.

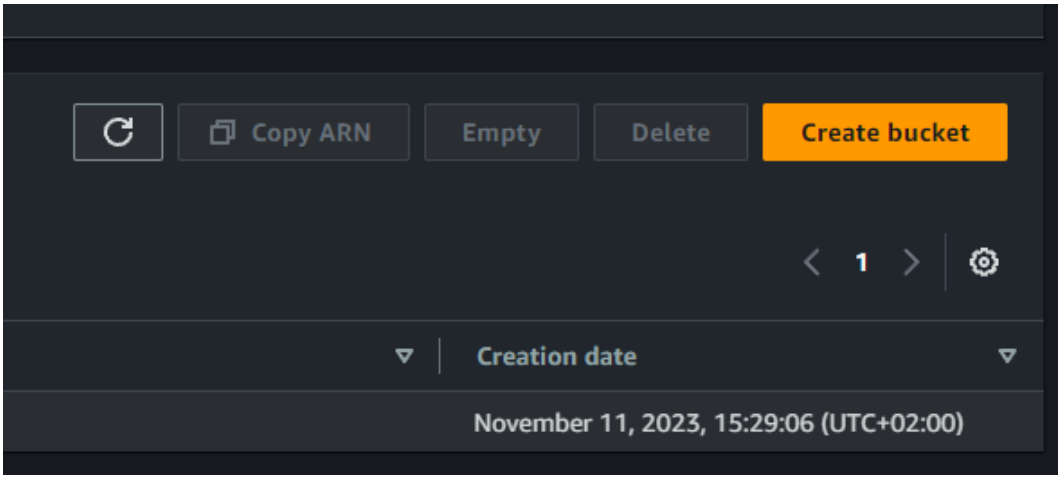

Рисунок 3.5 – Створення кошику

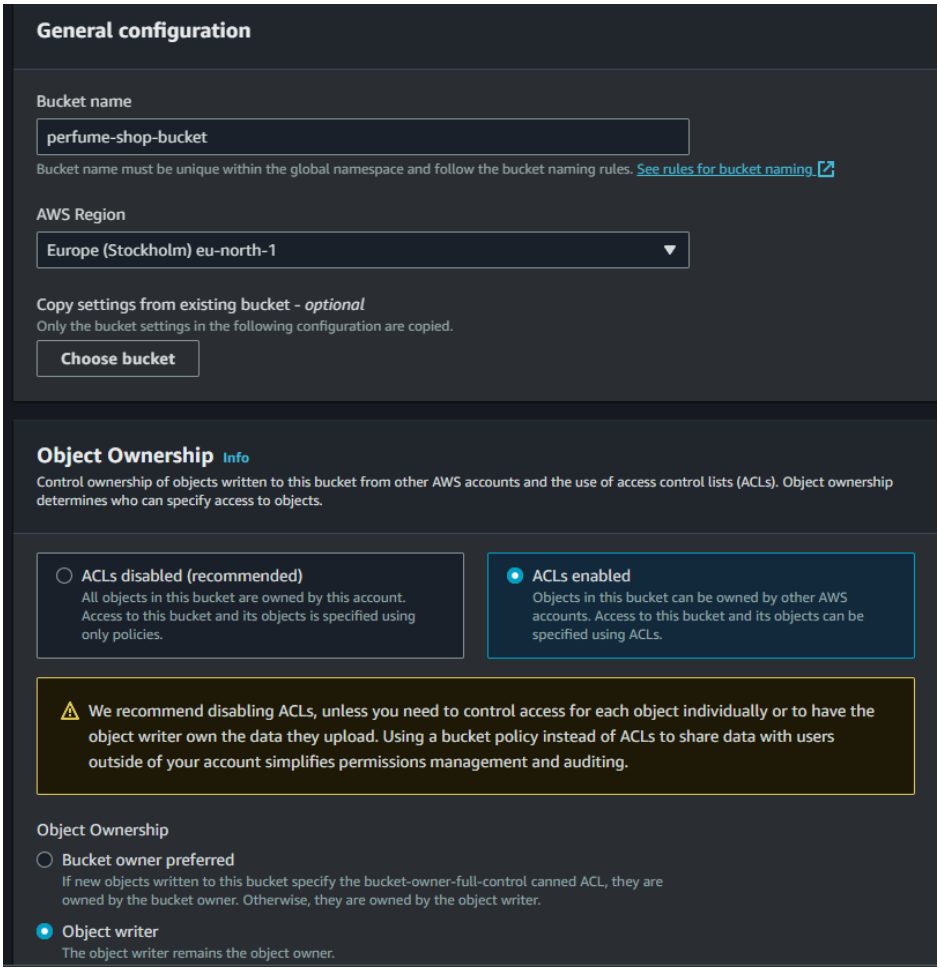

Рисунок 3.6 – Конфігурація кошику

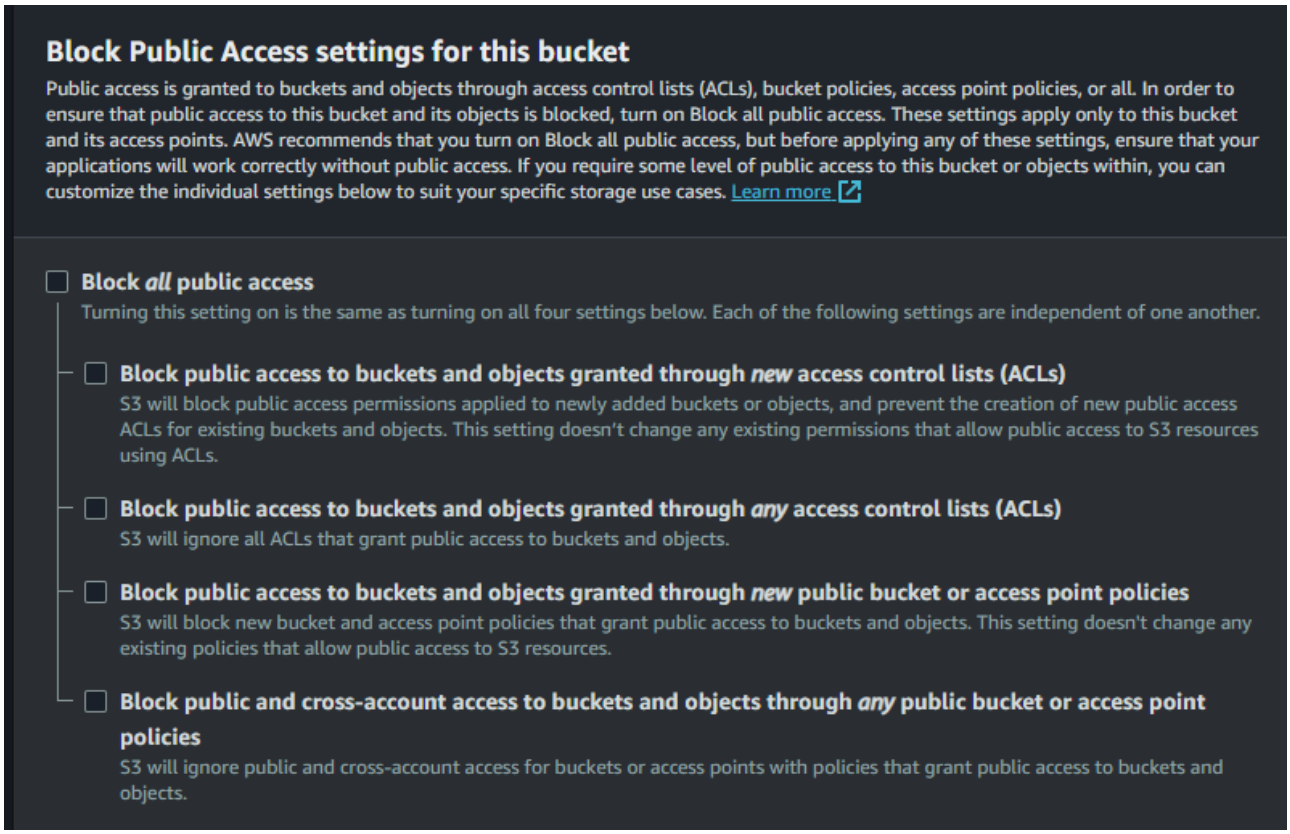

Рисунок 3.7 – Відкриття публічного доступу до кошику

Результатом таких конфігурацій має стати створений новий кошик до якого можливо зайти, подивитися усі створені файли, конфігурувати потрібну опції, тощо(рис 3.8).

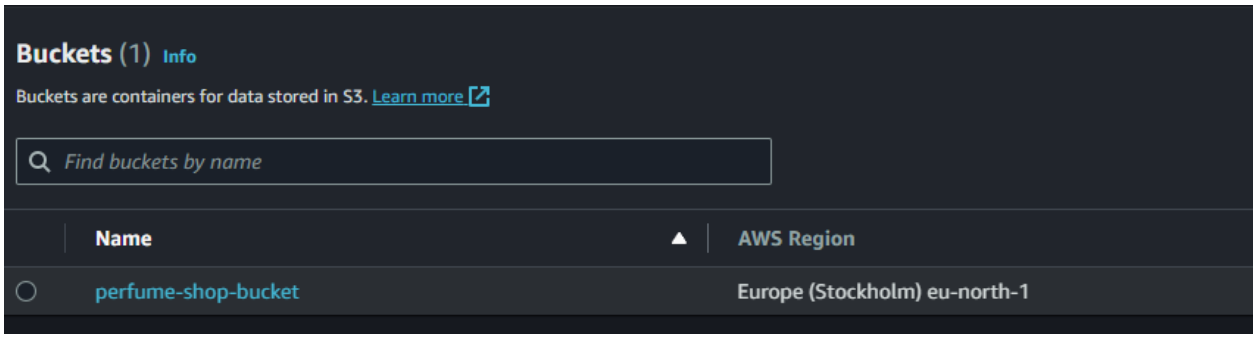

Рисунок 3.8 – Створений новий кошик

Третьою інтеграціє є імейл, для неї необхідно створити чи використовувати вже створений імейл з увімкненою MFA, що дозволить отримати пароль додатку. Для цього в пункті з двох етапною аутентифікацією необхідно обрати пункт паролі додатків(рис 3.9).

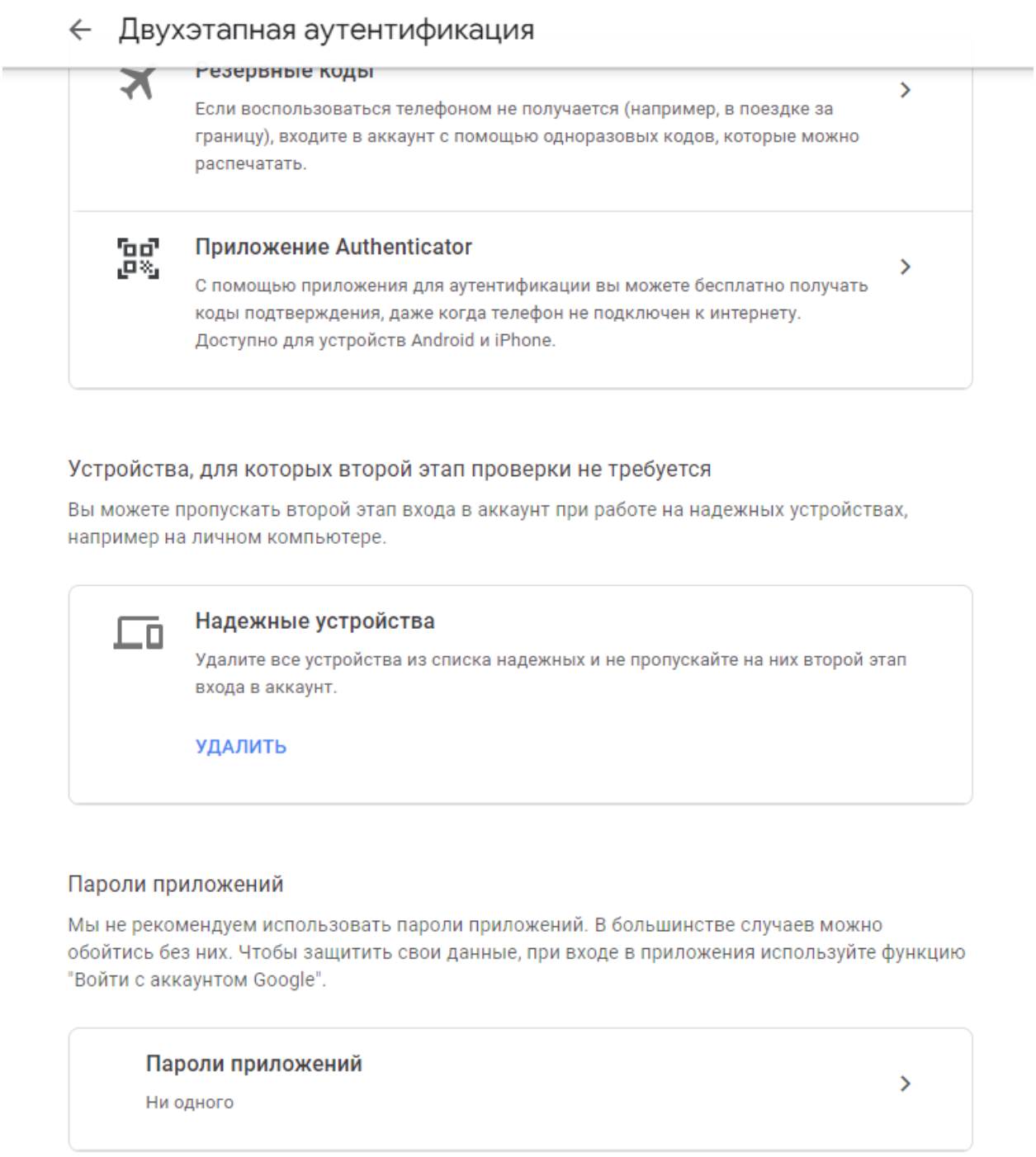

Рисунок 3.9 – Створення паролів додатків

Після обрання цього пункту необхідно додати ім’я додатку, що створить спеціальні паролі, які можливо використовувати програмно.

Останнім пунктом необхідно підготувати reCAPTCHA, для якої необхідно спочатку створити новий проект в гугл консолі(рис 3.10).

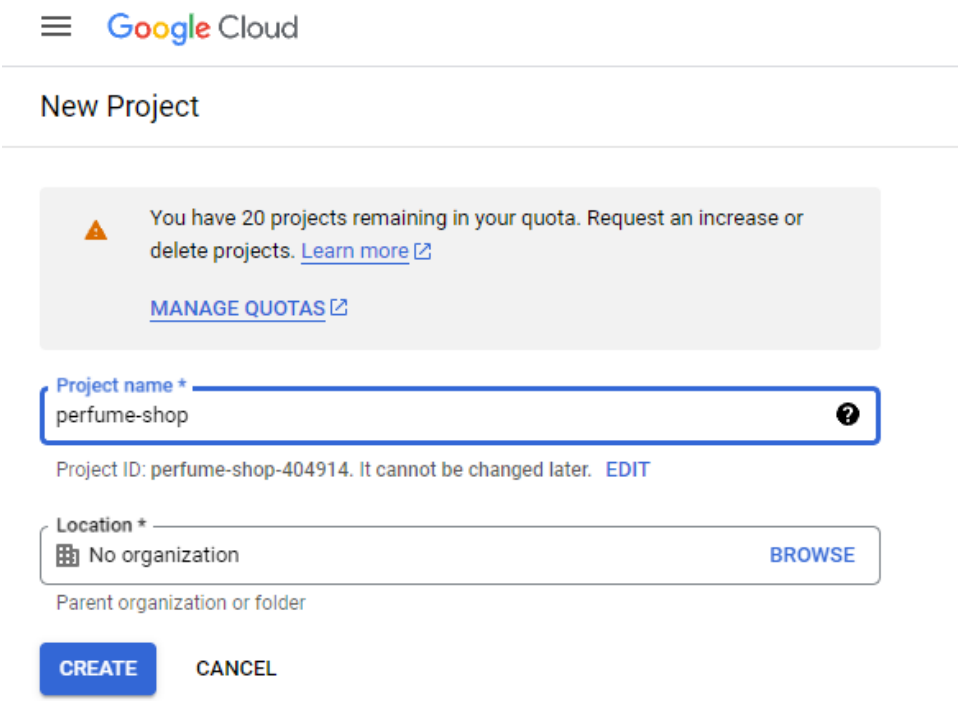

Рисунок 3.10 – Створення нового проекту

Після чого необхідно увімкнути капчу на цьому проекті.(рис 3.11)

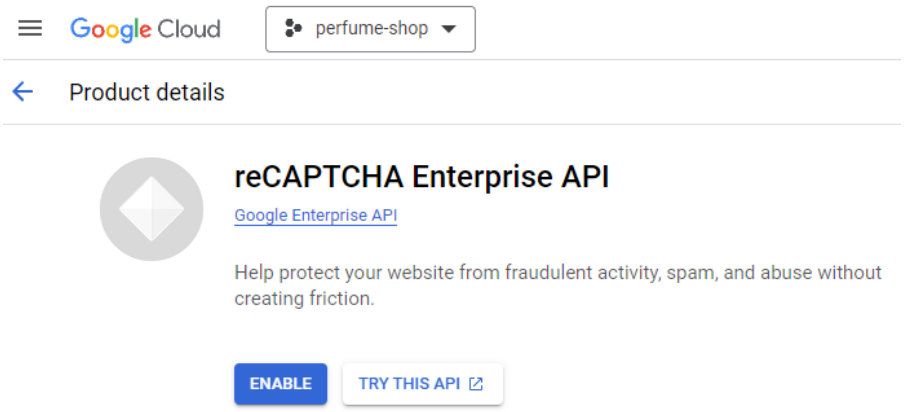

Рисунок 3.11 – Підключення reCAPTCHA до проекту

Після чого з’являється можливість налаштувати потрібну платформу та домен(рис 3.12). Для проекту було використано домен локалхост, проте при переміщенні додатку існує можливість обрати потрібний домен.

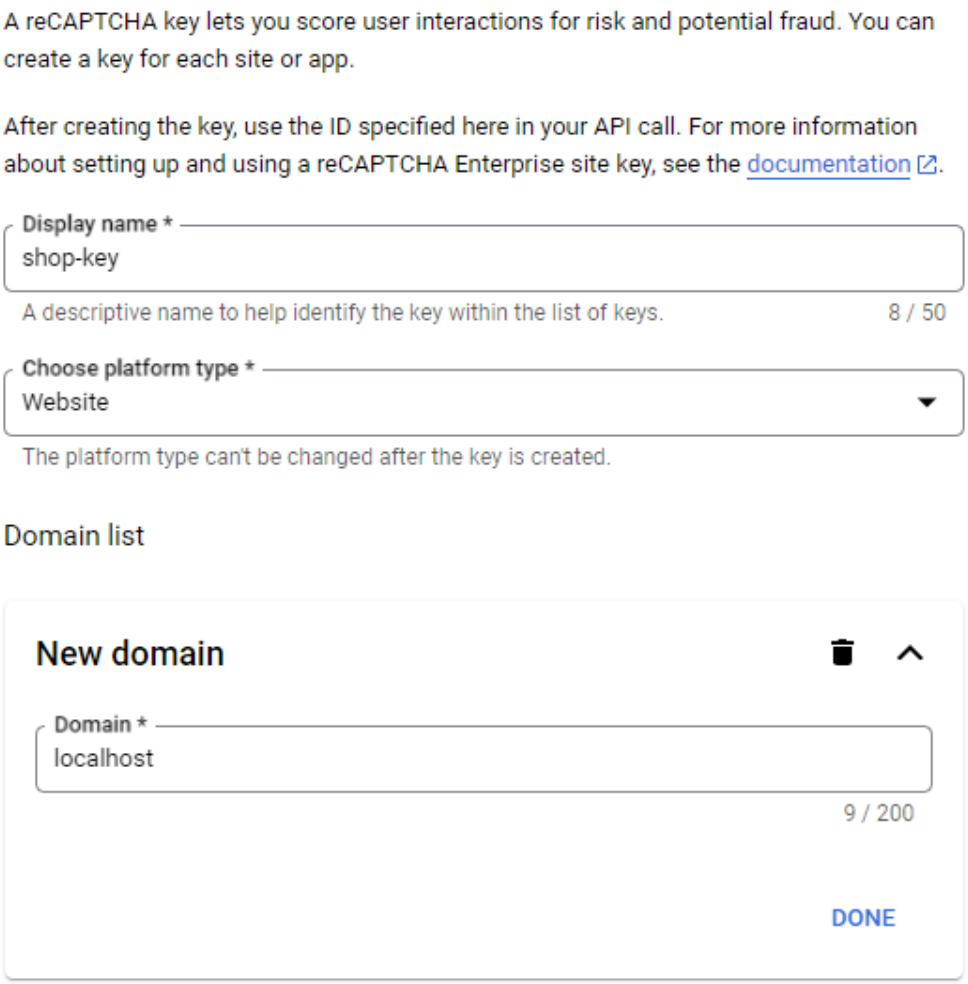

Рисунок 3.12 – Конфігурація reCAPTCHA

Результатом цих дій став приватний ключ, який буде в подальшому використовуватися в програмному коді.

3.3 Розробка коду серверної та клієнтської частини

Для розробки потрібного сайту необхідно спочатку створити проекти. Після їх створення та початкової конфігурації, необхідно перейти до безпосередньої розробки програмного коду.

Ця розробка була умовна розділена на декілька частин – підключення інтеграцій, створення потрібних сторінок та їх зовнішнього виду та створення необхідної клієнтської частини.

Першим було виконано підключення необхідних інтеграцій до серверної частини. Для цього спочатку у файлі конфігурації спочатку були вказані усі потрібні для роботи ключі(рис 3.13).

```
##Database keys
spring.datasource.url=jdbc:postgresql://cornelius.db.elephantsql.com:5432/qsecnjsc
spring.datasource.username=qsecnjsc
spring.datasource.password=DomCKKwosl4M8wSHbgV0WOPn5EctUpUl
spring.jpa.generate-ddl=false
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=validate
spring.flyway.locations=classpath:db/migration
##Email keys
spring.mail.host=smtp.gmail.com
spring.mail.username=anteikuprojectsender@gmail.com
spring.mail.password=vwtfrjbacudpjmqu
spring.mail.port=465
spring.mail.protocol=smtps
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
mail.debug=false
##Amazon keys
amazon.aws.access-key=AKIASMKMIKMNGP7OVIWA
amazon.aws.secret-key=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
amazon.s3.bucket.name=testbucketperf
##reCaptcha keys
recaptcha.secret=6Ld2mAwpAAAAAL1q28ubBJzyM-4AHIEhZ9B47cQ4
recaptcha.url=https://www.google.com/recaptcha/api/siteverify?secret=%s&response=%s
```

Рисунок 3.13 – Конфігурування ключів інтеграцій

Після чого було конфігуровано та підключено ці данні в окремих файлах, що дозволило використовувати функціонал цих інтеграцій програмно(рис 3.14 та 3.15 відповідно).

```
1 usage
@Value("${amazon.aws.access-key}")
private String awsAccessKey;

1 usage
@Value("${amazon.aws.secret-key}")
private String awsAccessSecret;

@Bean
public PasswordEncoder getPasswordEncoder() { return new BCryptPasswordEncoder( strength: 8); }

@Bean
public AmazonS3 s3Client() {
    AWSCredentials credentials = new BasicAWSCredentials(awsAccessKey, awsAccessSecret);
    return AmazonS3ClientBuilder.standard()
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .withRegion("eu-north-1")
            .build();
}
```

Рисунок 3.14 – Файл конфігурації AWS

```
1 usage
@Value("${spring.mail.properties.mail.smtp.auth}")
private String auth;

1 usage
@Value("${spring.mail.properties.mail.smtp.starttls.enable}")
private String enable;

1 usage
@Value("${mail.debug}")
private String debug;

@Bean
public JavaMailSender getMailSender() {
    JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
    mailSender.setHost(host);
    mailSender.setPort(port);
    mailSender.setUsername(username);
    mailSender.setPassword(password);
    Properties mailProperties = mailSender.getJavaMailProperties();
    mailProperties.setProperty("mail.transport.protocol", protocol);
    mailProperties.setProperty("mail.debug", debug);
    mailProperties.setProperty("mail.smtp.auth", auth);
    mailProperties.setProperty("mail.smtp.starttls.enable", enable);
    return mailSender;
}
```

Рисунок 3.15 – Файл конфігурації Gmail

reCAPTCHA та бази даних не потребують подальшої конфігурації, адже перша має бути використана на клієнтській частині, а роботою з базами даних займається спеціальна бібліотека.

Наступним кроком є створення фільтрів на серверній частині, які будуть використовуватися при авторизації. Ці фільтри спрацьовують перед кожним запитом на певні ендпоінти та перевіряють чи передається токен(рис 3.16).. Якщо він передається, то він валідується та залежно від результату користувач або отримує потрібну інформацію, або помилку авторизації(рис 3.17).

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors() CorsConfigurer<HttpSecurity>
        .and() HttpSecurity
        .csrf() CsrfConfigurer<HttpSecurity>
        .disable() HttpSecurity
        .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
        .and() HttpSecurity
        .sessionManagement() SessionManagementConfigurer<HttpSecurity>
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and() HttpSecurity
        .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
        .antMatchers(
            ...antPatterns: "/api/v1/auth/**",
            "/api/v1/auth/login",
            "/api/v1/registration/**",
            "/api/v1/perfumes/**",
            "/api/v1/users/cart",
            "/api/v1/order/**",
            "/api/v1/review/**",
            "/websocket",
            "/websocket/**",
            "/img/**",
            "/static/**") ExpressionUrlAuthorizationConfigurer<...>.AuthorizedUrl
        .permitAll() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
        .antMatchers( ...antPatterns: "/auth/**", "/oauth2/**", "/**/*swagger*/**", "/v2/api-docs") Ex
        .permitAll() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
        .anyRequest() ExpressionUrlAuthorizationConfigurer<...>.AuthorizedUrl
        .authenticated() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
        .and() HttpSecurity
        .apply(jwtConfigurer);
```

Рисунок 3.16 – Конфігурування ендпоінтів

```
public class JwtFilter extends GenericFilterBean {

    3 usages
    private final JwtProvider jwtProvider;

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        String token = jwtProvider.resolveToken((HttpServletRequest) servletRequest);

        try {
            if (token != null && jwtProvider.validateToken(token)) {
                Authentication authentication = jwtProvider.getAuthentication(token);

                if (authentication != null) {
                    SecurityContextHolder.getContext().setAuthentication(authentication);
                }
            }
        } catch (JwtAuthenticationException e) {
            SecurityContextHolder.clearContext();
            ((HttpServletResponse) servletResponse).sendError(e.getHttpStatus().value());
            throw new JwtAuthenticationException(INVALID_JWT_TOKEN);
        }
        filterChain.doFilter(servletRequest, servletResponse);
    }
}
```

Рисунок 3.17 – Конфігурація фільтру токену

Наступним кроком є задання структури клієнтської частини, для цього в відповідному файлі необхідно задати шлях, за яким користувачу необхідно перейти для доступу до певних сторінок(рис 3.18). Для більшої зручності усі ці путі було винесено в окремий файл з текстовими константами.

```
<NavBar />
<Switch>
    <Route exact path={BASE} component={Home} />
    <Route exact path={LOGIN} component={Login} />
    <Route exact path={REGISTRATION} component={Registration} />
    <Route exact path={FORGOT} component={ForgotPassword} />
    <Route exact path={`${RESET}/:code`} component={ResetPassword} />
    <Route exact path={`${ACTIVATE}/:code`} component={Login} />
    <Route exact path={MENU} component={Menu} />
    <Route exact path={`${PRODUCT}/:id`} component={Product} />
    <Route exact path={CONTACTS} component={Contacts} />
    <Route exact path={CART} component={Cart} />
    <Route exact path={ORDER} component={Order} />
    <Route exact path={ORDER_FINALIZE} component={OrderFinalize} />
    <Route path={OAUTH2_REDIRECT} component={OAuth2RedirectHandler} />
    <Route
        path={ACCOUNT}
        render={() =>
            localStorage.getItem("token") ? <Route component={Account} /> : <Route component={Home} />
        }
    />
    <Route path="*" component={Home} />
</Switch>
<Footer />
<BackTop />
```

Рисунок 3.18 – Конфігурація структури запитів

Виконання усіх цих кроків дозволило перейти до розробки відповідних сторінок разом з їх інтерфейсом. Розробка сторінок виконувалася паралельно – спочатку розроблявся графічний інтерфейс з дизайном, після чого на серверній частині створювався необхідний для роботи сторінки код.

Для розробки графічного інтерфейсу використовувалася пара React та TypeScript. Для створення сторінки спочатку необхідно було створити окрему компоненти сторінки, в яких необхідно окремо задати логіку цієї сторінки та її зовнішній вигляд.

Приклад розробки такої сторінки можливо побачити на рисунку 3.19

```
    const dispatch = useDispatch();
    const users = useSelector(selectAdminStateUsers);
    const isLoading = useSelector(selectIsAdminStateLoading);
    const totalElements = useSelector(selectTotalElements);
    const handleTableChange = useTablePagination<BaseUserResponse, number>(fetchAllUsers);

    useEffect(() => {
        dispatch(fetchAllUsers(0));

        return () => {
            dispatch(resetAdminState(LoadingStatus.LOADING));
        };
    }, []);

    return (
        <div>
            <ContentTitle title={"List of all users"} titleLevel={4} icon={<TeamOutlined />} />
            <Table
                rowKey={"id"}
                onChange={handleTableChange}
                loading={isLoading}
                pagination={{
                    total: totalElements,
                    position: ["bottomRight", "topRight"]
                }}
                dataSource={users}
                columns={[
```

Рисунок 3.19 – Приклад вигляду сторінки

Першою було розроблено головну сторінку. На цю сторінку юзер попадає під час переходу на сайт. Вона містить кнопки з можливістю перейти до усіх парфумів, авторизації чи реєстрації акаунту, можливість оглядати кошик та рекламні інтеграції.

З серверної частини для роботи цієї сторінки необхідно було розробити спеціальній сервіс та контролер, які будуть отримувати з бази даних парфуми та передавати їх деталі до клієнтської частини. Окрім цього необхідно реалізувати функціонал кошика, до якого можливо додавати парфуми, для цього аналогічно потрібно робити запити на серверну частину та зберігати нову інформацію.

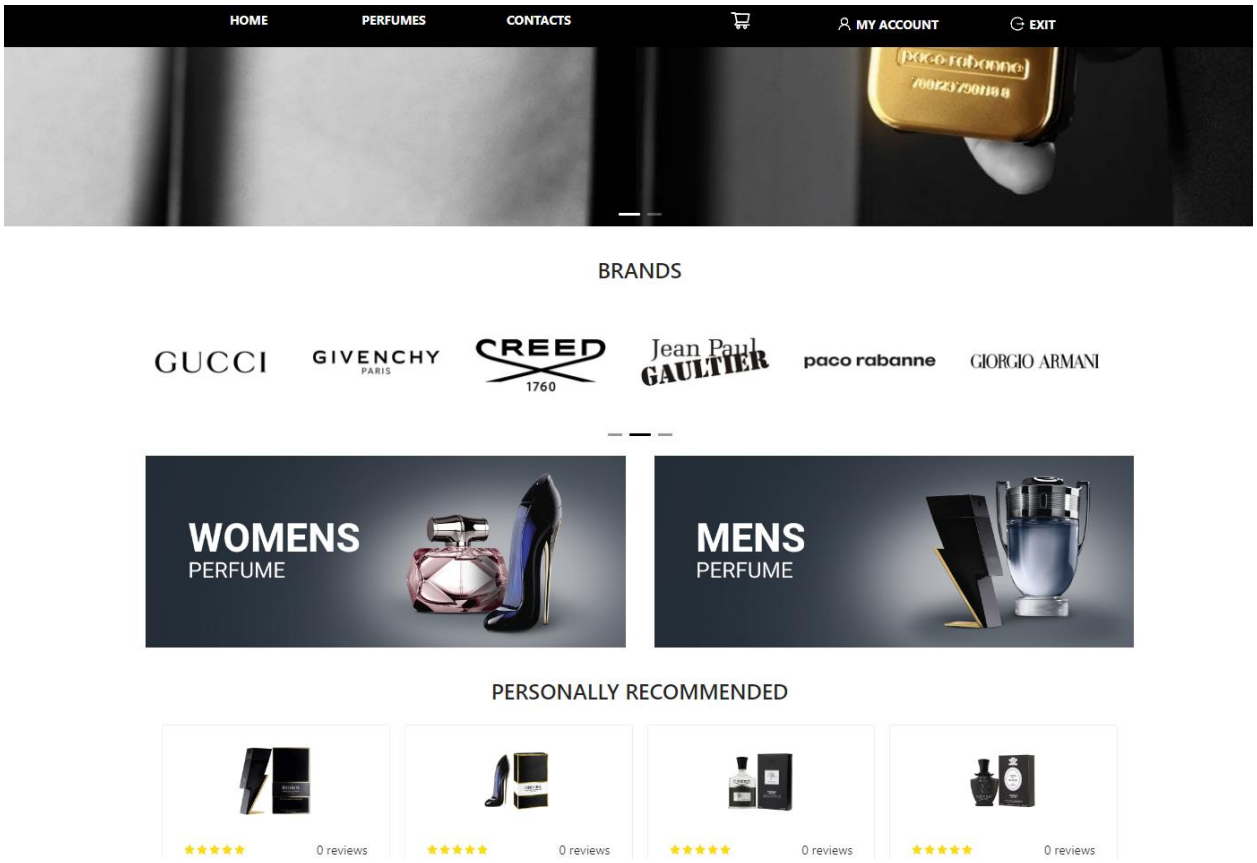

Рисунок 3.20 – Зовнішній вигляд головної сторінки

Наступними сторінками які було розроблено стали сторінки реєстрації та авторизації. Першим було створено їх дизайн(рисунки 3.21 та 3.22 відповідно), який має в собі вводити дані при реєстрації(імейл, ім’я, прізвище та пароль) та необхідні для авторизації імейл та пароль.

Після чого було створення необхідні сервіси які зберігають інформації про юзера до бази даних чи отримують її та передають на клієнтську частину. Окремо необхідно виділити інтеграції reCAPTCHA яку можливо побачити на рисунку 3.21 при реєстрації. При натисканню на відповідний чекбокс до серверної частини передається потрібна інформація, яка потім передається до гугла для аналізу.

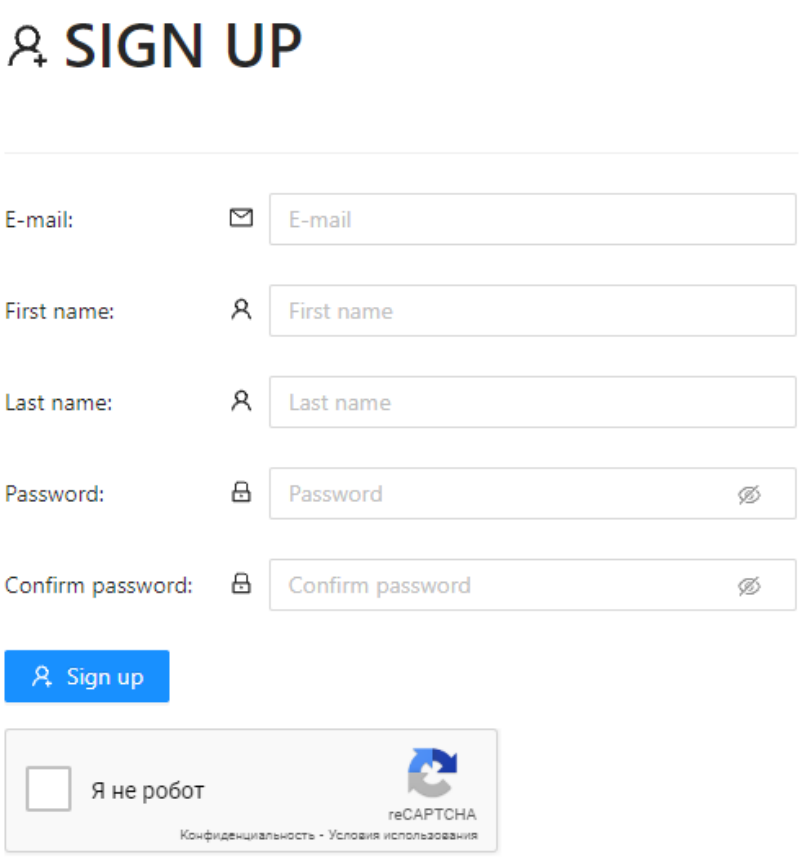

Рисунок 3.21 – Сторінка реєстрації

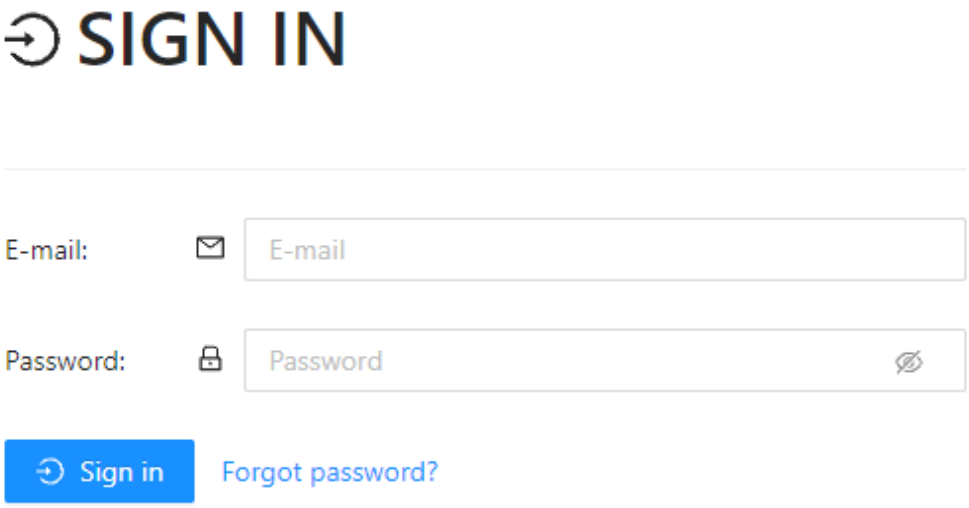

Рисунок 3.22 – Сторінка авторизації

Наступним кроком було створення сторінки особистого кабінету , який залежно від типу акаунту містить різний функціонал. Звичайний юзер має можливість переглядати свої замовлення та інформації про себе. Адміністратор же має можливість переглядати усі замовлення, створювати нові парфуми, змінювати їх інформацію та шукати їх інформацію(рисунки 3.23, 3.24, 3.25 та 3.26 відповідно).

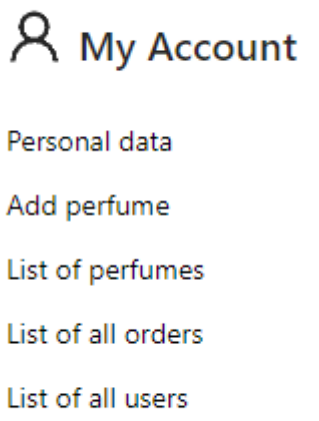

Рисунок 3.23 – Сторінка особистого кабінету

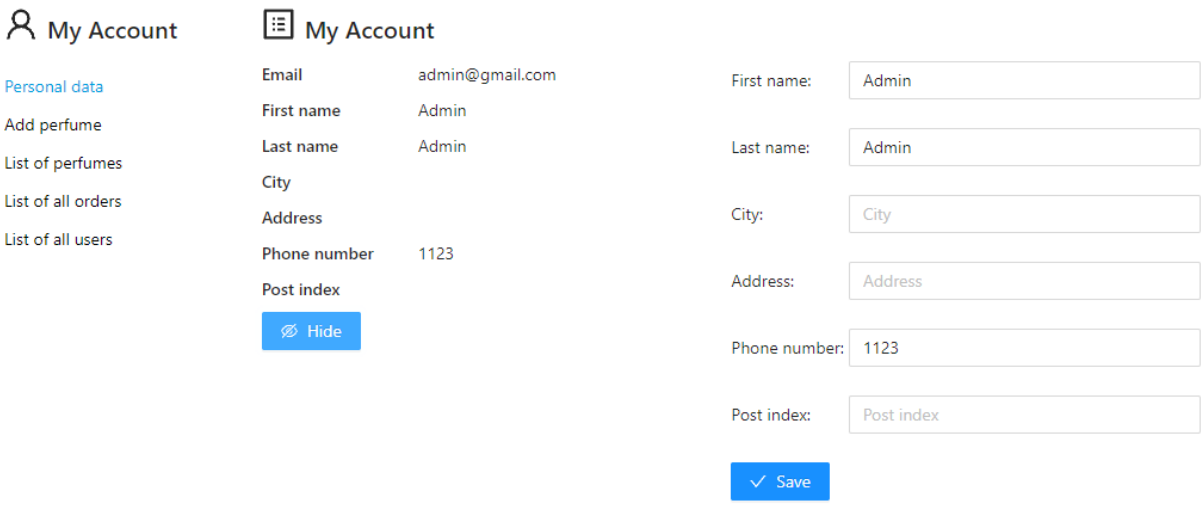

Рисунок 3.23 – Зміна та інформація про особисті дані

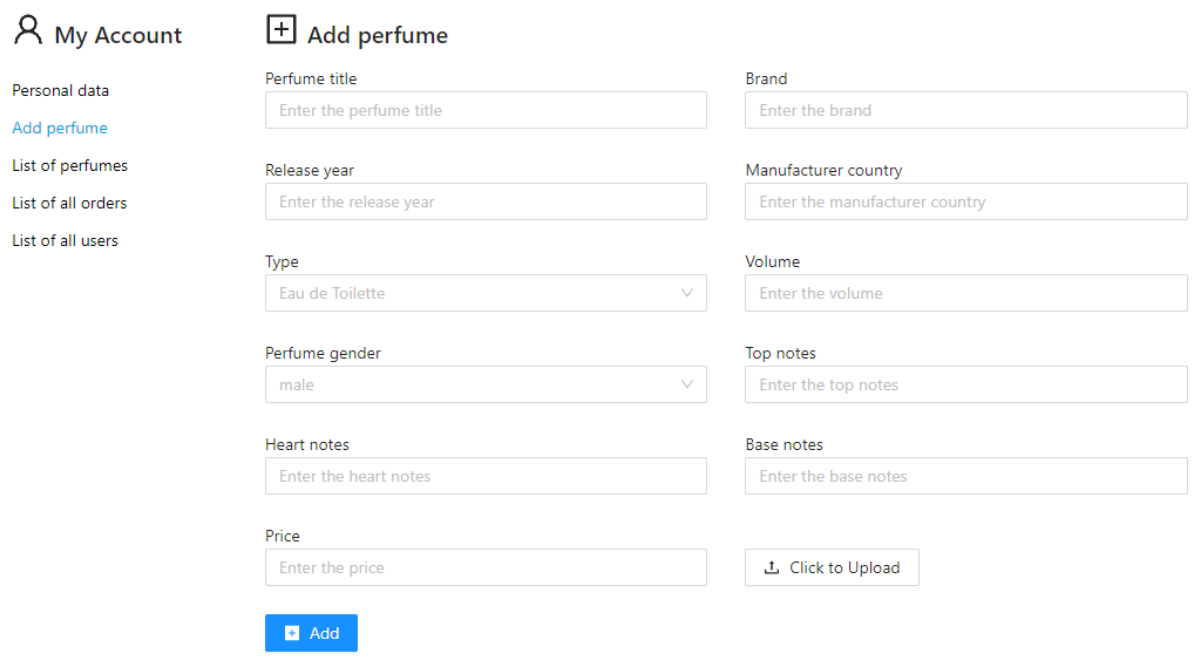

Рисунок 3.23 – Додавання парфумів

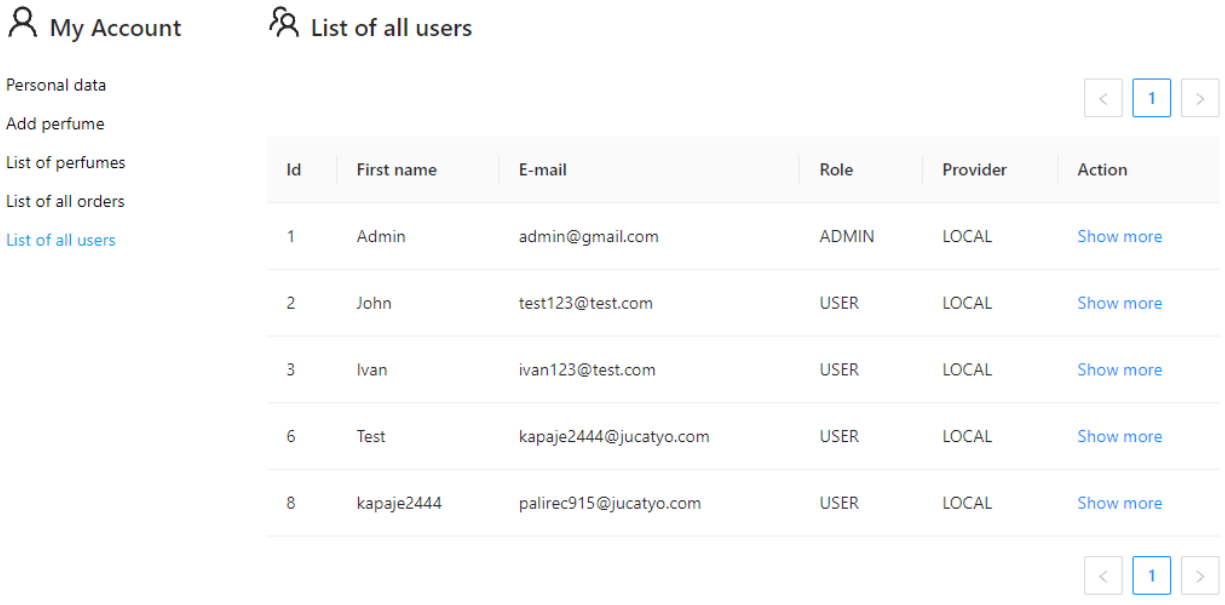

Рисунок 3.23 – Перелік усіх юзерів

Ця сторінка є найбільш складною відносно серверної та клієнтської реалізації. Для її роботи спочатку необхідно відправити запит на серверну частину та дізнатися чи є юзер адміністратором, потім відповідно від цих даних відобразити потрібні елементи.

Після чого необхідно на серверній стороні розробити контролери та сервіси які будуть створювати чи передавати інформацію про парфуми по їх ідентифікатору чи взагалі усі існуючі, аналогічно для таблиці з юзерами та замовленнями. Окрім цього усі ці дані необхідно розділити на сторінки та передавати на клієнтську частиною з можливістю отримати наступну сторінку. На рисунку 3.23 можливо побачити, що при створенні парфуму необхідно зберігати картинку, ця картинка зберігається до корзини в AWS, яку білу інтегровано раніше. Після чого до бази даних зберігається посилання до цієї картинки яку потім передається на клієнтську частину та відображається на сайті.

Окрім цього була інтегрована можливість адміністратора видаляти чи редагувати інформацію усіх створених парфумів(рис 3.24). Для цього було створено відповідні ендпоінти на серверній частині. Карточки які використовувалися будуть після цього використані при пошуку парфуму для звичайного юзера.

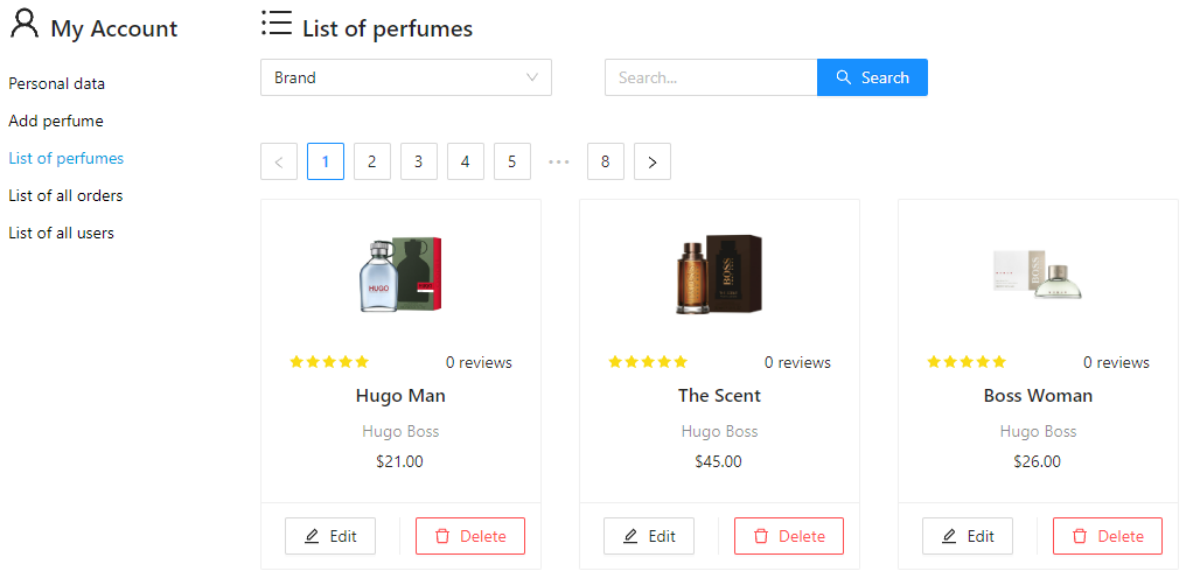

Рисунок 3.23 – Перелік усіх парфумів та їх редагування

Найпростішою сторінкою є сторінка з інформацією та контактами, вона містить інформацію щодо часів роботи та доставки товарів. Усе що необхідно це створити дизайн та відобразити цю сторінку клієнту(рис 3.24)

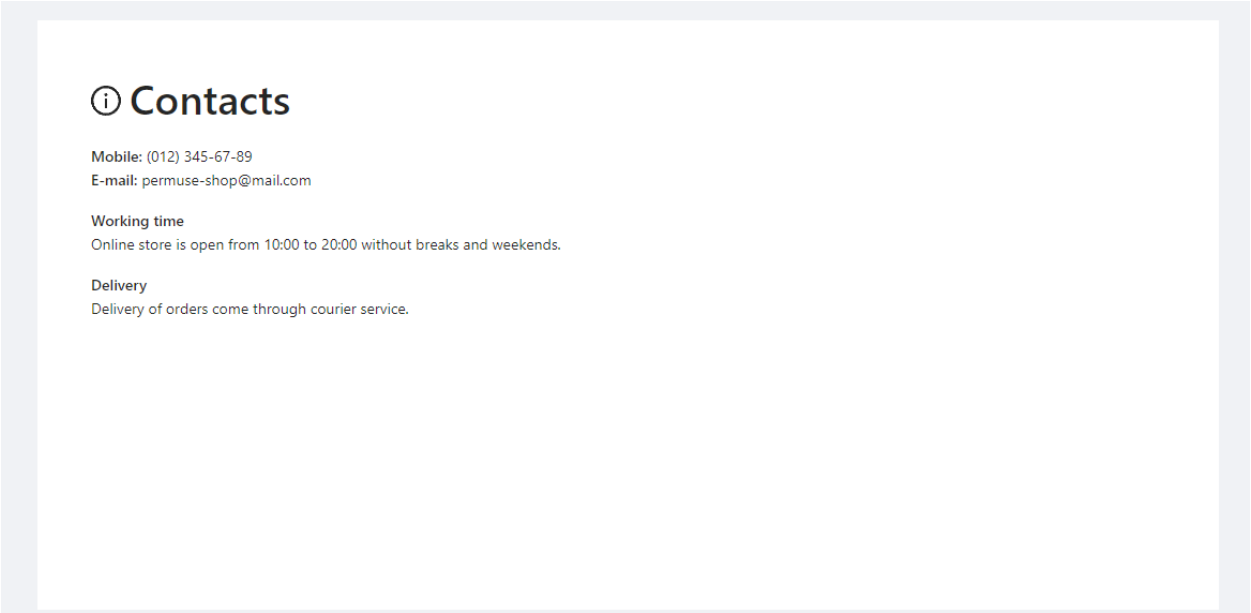

Рисунок 3.24 – Сторінка контактів

Окрім цього було створено сторінку з корзиною замовлених товарів. Для її роботи необхідно на клієнтській частині отримувати та зберігати інформацію щодо замовлення. Її дизайн можливо побачити на рисунку 3.25.

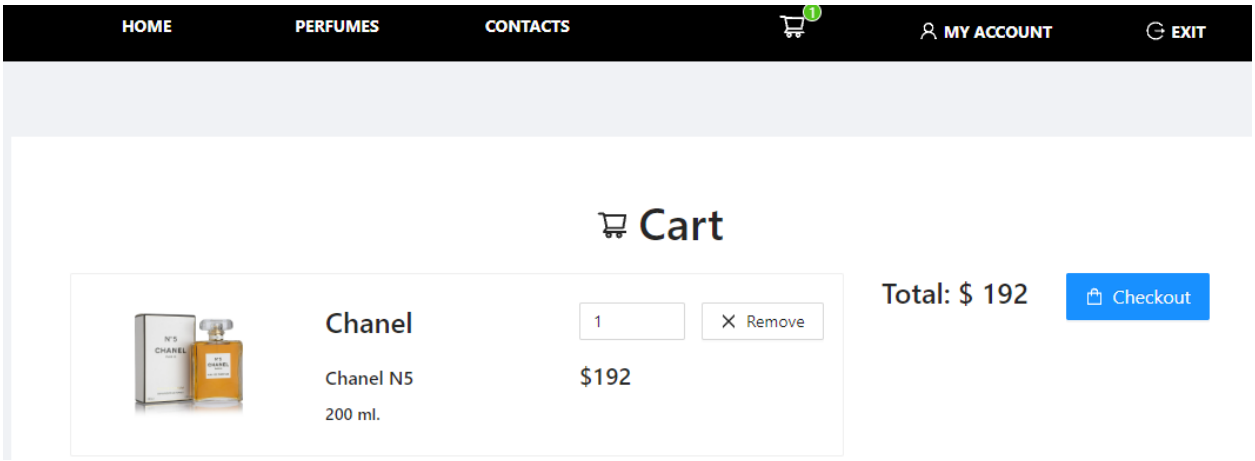

Рисунок 3.25 – Сторінка кошику

Останньою сторінкою яку було створено стала сторінка з переліком та пошуком парфумів яка має можливість додавати їх до кошику. Для відображення парфумів було використано карточки з особистого кабінету адмін акаунту. Окрім цього було додана можливість сортування за ціною, фільтрації по брендам, статтю та ціною разом з можливістю пошуку по назві(рис 3.26).

Для створення цієї сторінки на серверній частині було використані вже створені раніше сервіси для роботи з парфумами. Окрім цього було створено нові ендпоінти, які відповідають за фільтрації чи пошук товару по назві.

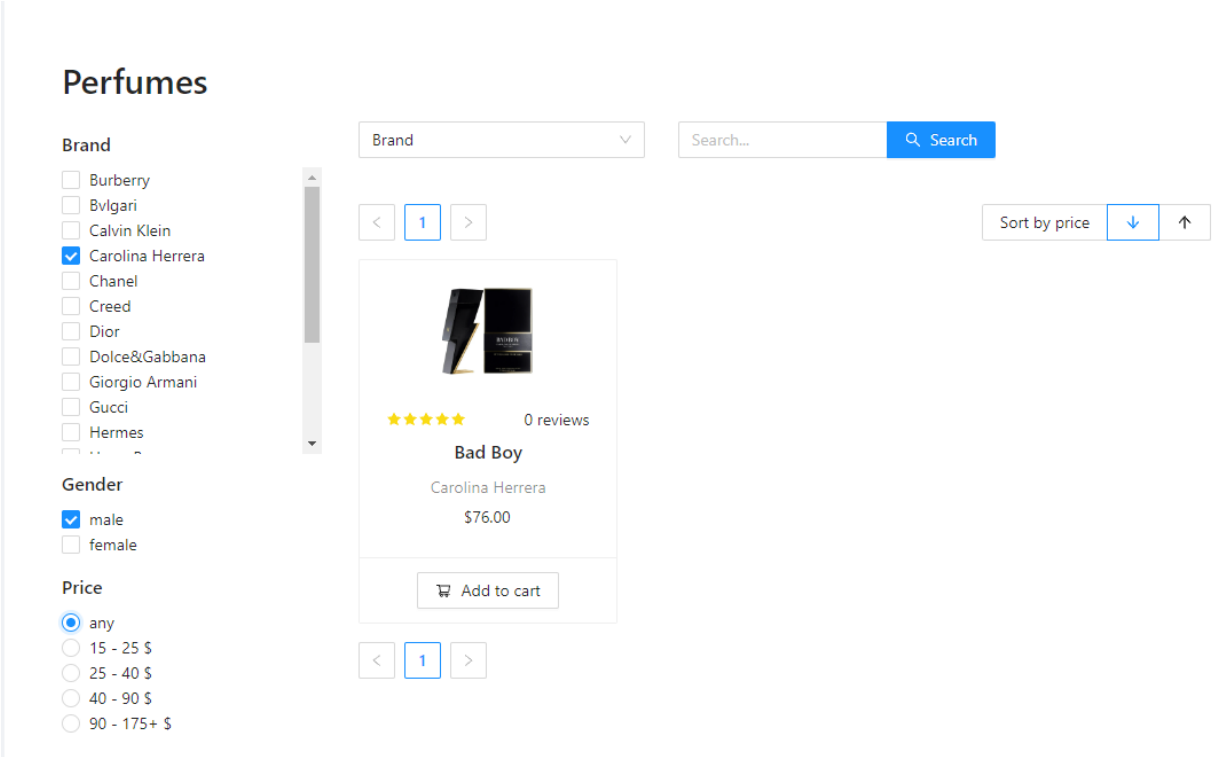

Рисунок 3.26 – Сторінка переліку та пошуку парфуму

Окремо необхідно розповісти про інтеграцію імейлу до сайту. На серверній частині було розроблено спеціальні шаблони для виправлення імейлів. Ці імейли використовуються для активації акаунту, зміні паролю та для отримання інформації щодо замовлення.

ВИСНОВОК

Під час проведення роботи було виконано аналіз актуальності та аналіз актуальності інформаційної технології для онлайн-шопінгу, після чого було проаналізована та обрано тип необхідного веб-додатку для створення.

Після цього було визначена структура веб-додатку та потреби його користувачів. Окремо від цього було проаналізовано можливі для додатку інтеграції та обрано потрібні. Було створено та описано логіку роботу сайту яка описувала як він має працювати та що потрібен містити. В наступних пунктах було проаналізовано останні сучасні технології та обрано необхідну для розробки методологію.

Результатом цих дій стало створений додаток для онлайн-шопінгу який містить весь потрібний для користувачів функціонал та є актуальним на даний час. Для його розробки було створено окрему сервісну частину з використанням Java разом з Spring та клієнтську частину з використанням React та TypeScript.

Розроблений додаток має можливість розширюватися в майбутньому та може бути доповненим відносно нових потреб користувачів. За допомогою акаунту адміністрації, його адмініструванням має можливість займатися людина не дуже досвідчена з технологіями.

Можливими покращеннями додатку є інтеграції можливості реєстрації за допомогою інших додатків(технологія Oauth), яка дозволить ще більше спростити процес реєстрації. Окрім цього можливим є додавання тестів та аналітики які будуть перевіряти діє спроможність проекту після його змін та його стан під час праці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Novikov, I., & Karamshuk, D. (2018). E-Commerce Optimization Using External Systems Integration. In Optimization of Computer Networks (pp. 187-203). Springer, Cham.
2. Zhou, X., Hong, J. M., & Lu, Y. (2016). Study on Integration Optimization of E-Commerce System Based on Consumer Experience under Big Data Background. Advances in Computer Science Research, 63, 287-292.
3. Sun, M., & Mathew, J. (2017). Integration of Internal and External Systems for Retail Supply Chain Optimization. International Journal of Business Intelligence and Data Mining, 12(3), 227-244.
4. Kuang, C. H., Huang, Y. T., Li, H. J., & Dai, J. G. (2015). Integration Optimization of E-Commerce's Outer System. Applied Mechanics and Materials, 496, 1363-1366.
5. Chen, H. Y., & Li, W. G. (2014). The Optimization of Market Platform E-commerce System Integration. Advanced Materials Research, 954-959, 3381-3384.
6. Wu, Z., & Wang, Y. (2013). Integration Optimization of E-commerce System Based on SOA. In 2013 6th International Conference on Information Management, Innovation Management and Industrial Engineering (pp. 712-715). IEEE.
7. Liu, Y., & Liu, X. (2012). Research on Optimization of E-commerce System: Application of Business Intelligence. Journal of Chemical and Pharmaceutical Research, 4(12), 5418-5423.
8. Baranov, A. V., Konovalov, O. V., Vorokhobko, V. V., & Khohlov, D. V. (2011). Integration Optimization of E-commerce Applications Based on ERP Systems. In Proceedings of the 19th International Conference on Computer Graphics and Vision (GraphiCon) (pp. 155-159). ACM.

9. Chen, H., Zhu, Q., Liu, Q., & Chen, J. (2010). On Optimization and Design of Intranet-Extranet-Internet Integration in E-commerce System. In Proceedings of the WRI Global Congress on Intelligent Systems (pp. 591-594). IEEE.
10. Zhang, W., Zhang, M., Niu, J., & Zhang, J. (2009). Development and Optimization of E-commerce Information System. Journal of Theoretical and Applied Information Technology, 7(1), 63-66.
11. Shneiderman, B. (1998). "Designing the User Interface: Strategies for Effective Human-Computer Interaction." Addison-Wesley.
12. Cooper, A., Reimann, R., & Cronin, D. (2007). "About Face 3: The Essentials of Interaction Design." Wiley.
13. Tondreau, B. (2014). "Layout Essentials: 100 Design Principles for Using Grids." Rockport Publishers.
14. Turban, E., Lee, J., King, D., & Chung, H. M. (2015). "Electronic Commerce 2016: A Managerial and Social Networks Perspective." Springer.
15. Kalakota, R., & Whinston, A. B. (1997). "Electronic Commerce: A Manager's Guide." Addison-Wesley.
16. Laudon, K. C., & Traver, C. G. (2016). "E-commerce 2016: Business, Technology, Society." Pearson.
17. Reddy, A., & Agarwal, R. (2017). "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions." Addison-Wesley.
18. Hohpe, G., & Woolf, B. (2004). "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions." Addison-Wesley.
19. McFarland, D. (2014). "JavaScript & jQuery: The Missing Manual." O'Reilly Media.
20. Krug, S. (2014). "Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability." New Riders.

21. Duckett, J. (2014). "HTML and CSS: Design and Build Websites." Wiley.
22. Norman, D. A. (2013). "The Design of Everyday Things." Basic Books.

ДОДАТОК

```
@PostMapping(ADD)
public ResponseEntity<FullPerfumeResponse> addPerfume(@RequestPart(name = "file", required = false) MultipartFile file,
                                                      @RequestPart("perfume") @Valid PerfumeRequest perfume,
                                                      BindingResult bindingResult) {
        return ResponseEntity.ok(perfumeMapper.savePerfume(perfume, file, bindingResult));
}

@PostMapping(EDIT)
public ResponseEntity<FullPerfumeResponse> updatePerfume(@RequestPart(name = "file", required = false) MultipartFile file,
                                                         @RequestPart("perfume") @Valid PerfumeRequest perfume,
                                                         BindingResult bindingResult) {
    return ResponseEntity.ok(perfumeMapper.savePerfume(perfume, file, bindingResult));
}

@DeleteMapping(DELETE_BY_PERFUME_ID)
public ResponseEntity<String> deletePerfume(@PathVariable Long perfumeId) {
    return ResponseEntity.ok(perfumeMapper.deletePerfume(perfumeId));
}

@GetMapping(ORDERS)
public ResponseEntity<List<OrderResponse>> getAllOrders(@PageableDefault(size = 10) Pageable pageable) {
    HeaderResponse<OrderResponse> response = orderMapper.getAllOrders(pageable);
    return ResponseEntity.ok().headers(response.getHeaders()).body(response.getItems());
}

@GetMapping(ORDER_BY_EMAIL)
public ResponseEntity<List<OrderResponse>> getUserOrdersByEmail(@PathVariable String userEmail,
                                                                @PageableDefault(size = 10) Pageable pageable) {
    HeaderResponse<OrderResponse> response = orderMapper.getUserOrders(userEmail, pageable);
    return ResponseEntity.ok().headers(response.getHeaders()).body(response.getItems());
}

@DeleteMapping(ORDER_DELETE)
public ResponseEntity<String> deleteOrder(@PathVariable Long orderId) {
    return ResponseEntity.ok(orderMapper.deleteOrder(orderId));
}
```

```
public class AuthenticationController {

    5 usages
    private final AuthenticationMapper authenticationMapper;

    @PostMapping(LOGIN)
    public ResponseEntity<AuthenticationResponse> login(@RequestBody AuthenticationRequest request) {
        return ResponseEntity.ok(authenticationMapper.login(request));
    }

    @GetMapping(FORGOT_EMAIL)
    public ResponseEntity<String> forgotPassword(@PathVariable String email) {
        return ResponseEntity.ok(authenticationMapper.sendPasswordResetCode(email));
    }

    @GetMapping(RESET_CODE)
    public ResponseEntity<String> getEmailByPasswordResetCode(@PathVariable String code) {
        return ResponseEntity.ok(authenticationMapper.getEmailByPasswordResetCode(code));
    }

    @PostMapping(RESET)
    public ResponseEntity<String> passwordReset(@RequestBody PasswordResetRequest passwordReset) {
        return ResponseEntity.ok(authenticationMapper.passwordReset(passwordReset.getEmail(), passwordReset));
    }

    @PutMapping(EDIT_PASSWORD)
    public ResponseEntity<String> updateUserPassword(@AuthenticationPrincipal UserPrincipal user,
                                                     @Valid @RequestBody PasswordResetRequest passwordReset,
                                                     BindingResult bindingResult) {
        return ResponseEntity.ok(authenticationMapper.passwordReset(user.getEmail(), passwordReset, bindingResult));
    }
}
```

```java
    @GetMapping
    public ResponseEntity<UserResponse> getUserInfo(@AuthenticationPrincipal UserPrincipal user) {
        return ResponseEntity.ok(userMapper.getUserInfo(user.getEmail()));
    }

    @PutMapping
    public ResponseEntity<UserResponse> updateUserInfo(@AuthenticationPrincipal UserPrincipal user,
                                                       @Valid @RequestBody UpdateUserRequest request,
                                                       BindingResult bindingResult) {
        return ResponseEntity.ok(userMapper.updateUserInfo(user.getEmail(), request, bindingResult));
    }

    @PostMapping(CART)
    public ResponseEntity<List<PerfumeResponse>> getCart(@RequestBody List<Long> perfumesIds) {
        return ResponseEntity.ok(userMapper.getCart(perfumesIds));
    }

    @PostMapping(GRAPHQL)
    public ResponseEntity<ExecutionResult> getUserInfoByQuery(@RequestBody GraphQLRequest request) {
        return ResponseEntity.ok(graphQLProvider.getGraphQL().execute(request.getQuery()));
    }
}
```

```java
4 usages
@Service
@RequiredArgsConstructor
public class MailSender {

    2 usages
    private final JavaMailSender mailSender;
    1 usage
    private final SpringTemplateEngine thymeleafTemplateEngine;

    1 usage
    @Value("${spring.mail.username}")
    private String username;

    2 usages
    @SneakyThrows
    public void sendMessageHtml(String to, String subject, String template, Map<String, Object> attributes) throws MessagingException
        Context thymeleafContext = new Context();
        thymeleafContext.setVariables(attributes);
        String htmlBody = thymeleafTemplateEngine.process(template, thymeleafContext);
        MimeMessage message = mailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(message, multipart: true, encoding: "UTF-8");
        helper.setFrom(username);
        helper.setTo(to);
        helper.setSubject(subject);
        helper.setText(htmlBody, html: true);
        mailSender.send(message);
    }
}
```

```java
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    6 usages
    private final UserRepository userRepository;
    1 usage
    private final PerfumeRepository perfumeRepository;

    1 usage
    @Override
    public User getUserById(Long userId) {
        return userRepository.findById(userId)
                .orElseThrow(() -> new ApiRequestException(USER_NOT_FOUND, HttpStatus.NOT_FOUND));
    }

    1 usage
    @Override
    public User getUserInfo(String email) {
        return userRepository.findByEmail(email)
                .orElseThrow(() -> new ApiRequestException(EMAIL_NOT_FOUND, HttpStatus.NOT_FOUND));
    }

    1 usage
    @Override
    public Page<User> getAllUsers(Pageable pageable) {
        return userRepository.findAllByOrderByIdAsc(pageable);
    }

    1 usage
    @Override
    public List<Perfume> getCart(List<Long> perfumeIds) {
        return perfumeRepository.findByIdIn(perfumeIds);
    }
```

```
    return (
        <>
            <NavBar />
            <Switch>
                <Route exact path={BASE} component={Home} />
                <Route exact path={LOGIN} component={Login} />
                <Route exact path={REGISTRATION} component={Registration} />
                <Route exact path={FORGOT} component={ForgotPassword} />
                <Route exact path={`${RESET}/:code`} component={ResetPassword} />
                <Route exact path={`${ACTIVATE}/:code`} component={Login} />
                <Route exact path={MENU} component={Menu} />
                <Route exact path={`${PRODUCT}/:id`} component={Product} />
                <Route exact path={CONTACTS} component={Contacts} />
                <Route exact path={CART} component={Cart} />
                <Route exact path={ORDER} component={Order} />
                <Route exact path={ORDER_FINALIZE} component={OrderFinalize} />
                <Route path={OAUTH2_REDIRECT} component={OAuth2RedirectHandler} />
                <Route
                    path={ACCOUNT}
                    render={() =>
                        localStorage.getItem("token") ? <Route component={Account} /> : <Route component={Home} />
                    }
                />
                <Route path="*" component={Home} />
            </Switch>
            <Footer />
            <BackTop />
        </>
    );
};

export default App;
```

```
    return (
        <ContentWrapper>
            <ContentTitle icon={<LoginOutlined />} title={"SIGN IN"} />
            <Row gutter={32}>
                <Col span={12}>
                    <Form onFinish={onClickSignIn}>
                        <Divider />
                        {errorMessage && <Alert type="error" message={errorMessage} />}
                        {successMessage && <Alert type="success" message={successMessage} />}
                        <FormInput
                            title={"E-mail:"}
                            icon={<MailOutlined />}
                            titleSpan={6}
                            wrapperSpan={18}
                            name={"email"}
                            placeholder={"E-mail"}
                        />
                        <FormInput
                            title={"Password:"}
                            icon={<LockOutlined />}
                            titleSpan={6}
                            wrapperSpan={18}
                            name={"password"}
                            placeholder={"Password"}
                            inputPassword
                        />
                        <Space align={"baseline"} size={13}>
                            <IconButton title={"Sign in"} icon={<LoginOutlined />} />
                            <Link to={FORGOT}>Forgot password?</Link>
                        </Space>
                    </Form>
                </Col>
            </Row>
        </ContentWrapper>
    );
};
```

```tsx
const Order: FC = (): ReactElement => {
    const dispatch = useDispatch();
    const history = useHistory();
    const [form] = Form.useForm();
    const usersData = useSelector(selectUserFromUserState);
    const perfumes = useSelector(selectCartItems);
    const totalPrice = useSelector(selectTotalPrice);
    const errors = useSelector(selectOrderErrors);
    const isOrderLoading = useSelector(selectIsOrderLoading);
    const [perfumesFromLocalStorage, setPerfumesFromLocalStorage] = useState<Map<number, number>>(new Map());

    useEffect(() => {
        const perfumesFromLocalStorage: Map<number, number> = new Map(
            JSON.parse(localStorage.getItem("perfumes") as string)
        );
        setPerfumesFromLocalStorage(perfumesFromLocalStorage);
        dispatch(setOrderLoadingState(LoadingStatus.LOADED));
        dispatch(fetchCart(Array.from(perfumesFromLocalStorage.keys())));

        if (usersData) {
            form.setFieldsValue(usersData);
        }

        return () => {
            dispatch(resetOrderState());
            dispatch(resetCartState());
        };
    }, []);

    const onFormSubmit = (order: OrderFormData): void => {
        const perfumesId = Object.fromEntries(new Map(JSON.parse(localStorage.getItem("perfumes") as string)));
        dispatch(addOrder({ order: { ...order, perfumesId, totalPrice }, history }));
    };
```

```tsx
    }, [userData]);

    return (
        <>
            {isUserLoading ? (
                <Spinner />
            ) : (
                <>
                    <ContentTitle title={`User: ${firstName} ${lastName}`} titleLevel={4} icon={<UserOutlined />} />
                    <Row>
                        <Col span={24}>
                            <Card>
                                <Row gutter={24}>
                                    <Col span={12}>
                                        <AccountDataItem title={"User id"} text={id} />
                                        <AccountDataItem title={"Email"} text={email} />
                                        <AccountDataItem title={"Role"} text={roles} />
                                        <AccountDataItem title={"First name"} text={firstName} />
                                        <AccountDataItem title={"Last name"} text={lastName} />
                                    </Col>
                                    <Col span={8}>
                                        <AccountDataItem title={"Provider"} text={provider} />
                                        <AccountDataItem title={"City"} text={city} />
                                        <AccountDataItem title={"Address"} text={address} />
                                        <AccountDataItem title={"Phone number"} text={phoneNumber} />
                                        <AccountDataItem title={"Post index"} text={postIndex} />
                                    </Col>
                                </Row>
                            </Card>
                            <Row style={{ marginTop: 16 }}>
                                <Col span={24}>
                                    {userOrders.length === 0 ? (
                                        <div style={{ textAlign: "center" }}>
                                            <ContentTitle title={"No orders"} titleLevel={4} />
                                        </div>
```

```
const Product: FC = (): ReactElement => {
    const dispatch = useDispatch();
    const [form] = Form.useForm();
    const params = useParams<{ id: string }>();
    const perfume = useSelector(selectPerfume);
    const reviews = useSelector(selectReviews);
    const isPerfumeLoading = useSelector(selectIsPerfumeLoading);
    const isPerfumeLoaded = useSelector(selectIsPerfumeLoaded);
    const isPerfumeError = useSelector(selectPerfumeError);
    const errorMessage = useSelector(selectPerfumeErrorMessage);
    const reviewErrors = useSelector(selectReviewErrors);
    const isReviewAdded = useSelector(selectIsReviewAdded);
    const { addToCart } = useCart(perfume?.id!);

    useEffect(() => {
        // GraphQL example
        // dispatch(fetchPerfumeByQuery(params.id));
        dispatch(fetchPerfume(params.id));
        dispatch(resetInputForm());
        window.scrollTo(0, 0);
        stompClient = Stomp.over(() => new SockJS(WEBSOCKET_URL));
        stompClient.connect({}, () => {
            stompClient?.subscribe("/topic/reviews/" + params.id, (response: any) => {
                dispatch(setReview(JSON.parse(response.body)));
            });
        });

        return () => {
            stompClient?.disconnect();
            dispatch(resetPerfumeState());
        };
    }, []);

    useEffect(() => {
        if (isPerfumeLoaded) {
            dispatch(fetchReviewsByPerfumeId(params.id));
        }
    }, [isPerfumeLoaded]);

    useEffect(() => {
        form.resetFields();
    }, [isReviewAdded]);
```