ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти бакалавр

на тему: «**Розроблення клієнт-серверного програмного забезпечення підтримки діяльності інтернет-супермаркету**»

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
освітнього ступеня бакалавр
групи 126ІСТ_бд_41
Омеляненко А.Ю.
Керівник: Одарущенко О.М.
Рецензент: Ковальчук С.Б.

Полтава – 2024 року

ВСТУП

Актуальність теми. Інтернет-торгівля продуктами харчування – це динамічно розвивається сектор, який має великий [1] потенціал для малого та середнього бізнесу, хоч і на ньому присутні великі гравці вони не здатні задовільнити всі потреби клієнтів. Існуючий бізнес не забезпечує всі потреби клієнтів в інтернет торгівлі: 53% покупців повернулися до фізичних магазинів [2]. Це свідчить про незадоволення клієнтів. Однак цих клієнтів ще можна отримати реалізуючи можливості які доступні в фізичних магазинах та створюючи унікальний досвіт [3]. Існують SaaS рішення для швидкого створення інтернет магазину або мобільних застосунків для продажів. Однак це не надає можливостей задовільнити специфічні потреби клієнтів. Створення власного інтернет застосунку наддасть бізнесу повний контроль над даними клієнтів та можливість побудувати особливий сервіс.

Сучасний світ змінюється швидше ніж компанії. Інформаційні технології стають все доступнішими а інструменти для вирішення базових задач простішими. У кінці 20го століття світ почав змінюватися під впливом інтернету. Почали створювати перші комерційні веб сайти та згодом це стало масовим. Однак доступним для більшості підприємств ведення бізнесу в інтернеті став лише останні 10 – 15 років. Ще багато підприємств не є повністю чи навіть частково оцифровані. Цифровізацію бізнесу та представлення його в інтернеті на сьогодні є не просто необхідністю а вимогою від держави. Закон України про ведення комерційної діяльності зобов'язує вести облік товарів та повідомляти податкову про реалізацію товару практично у реальному часі. Цифровізація бізнесу надає можливості обробляти дані клієнтів, передбачувати клієнтів та навіть направляти клієнтів. Кількість інформації що дня зростає, багато підприємств ще не здатні її обробити.

Більшість підприємств представлені, тим чи інакшим чином, в інтернеті. Зараз кожен може створити акаунт у соціальній мережі та використовувати його як точку продажу. Це надає можливість мінімально задовольнити клієнтів та

бути у їх полі зору. З недоліків такого підходу варто зазначити відсутність системності. Навіть маючи найсвіжішу інформацію про клієнта: знання про його місцеперебування, вподобання та історію розмов такий підхід не дає можливості аналізувати цю інформацію у повному обсязі.

Підприємства, які владуються більше у свою цифровізацію використовують різноманітні сервіси, такі як Shopify та WordPress. Такий підхід вже задовольняє більшу частину клієнтів та надає бізнесу простір для розширення. Клієнт отримує зручний перегляд товару, відстежування замовлення без необхідності очікувати відповіді від менеджера. Однак бізнес теж ще не має можливості обробляти та отримувати більшу кількість даних про клієнта. Дані системи є простими у розширені та дешеві в розробці.

Підприємства з великою кількістю даних та складними процесами використовують SaaS рішення для керування даними та менеджменту відносинами з клієнтами. Такі рішення надають великий функціонал та є легко розширювальними, знайти розробників для покращення доволі легко. Однак в рамках таких систем не завжди можливо реалізувати весь не обхідний функціонал для бізнесу, через їх структуру чи складність певних рішень.

Деякі підприємства, для оптимізації бізнес процесів, створюють власні ІС та інтригують їх з вже існуючими. Наприклад компанія ToBeModern створила систему для менеджменту працівників лише на виробництві. В системі не має інформації про заробітну платню чи переписки з клієнтами дана система створена лише для оптимізації процесів виробництва. Дана система надала змогу гнучко розподіляти час працівників базуючись на завантаженості працівників на різних етапах виробництва.

Кількість вже існуючих комерційних підприємств з існуючою головною ІС яким необхідне розширення у вигляді додаткової ІС для задоволення потреб інтернет торгівлі є великим. Перші підприємства, які реалізували торгівлю в інтернеті вже є мастодонтами, Сільпо, АТБ, Метро, які можуть керувати ринком. Однак клієнтський попит ще досі є не є задоволеним повністю. Багатьом клієнтам потрібен унікальний досвід взаємодії, який не можуть задовольнити

великий бізнес. Це ще надає можливість малому та середньому бізнесу реалізуватися в інтернет торгівлі.

Мета роботи. Дослідити існуючі рішення інтернет торгівлі великих та малих бізнесів.

Для досягнення поставленої мети потрібно вирішити наступні завдання:

– Сформувати набір функціональних вимог базуючись на проведеному аналізі.

– Розробити архітектуру програмного забезпечення та реалізувати його.

Об'єкт дослідження кваліфікаційної роботи. Процеси розроблення та модифікації вебзастосунків для їх комерційного застосування.

Предмет дослідження кваліфікаційної роботи. Архітектура та функціонал веборієнтованих систем.

Методи дослідження. системний аналіз, аналітичний аналіз, статичний аналіз, використання Agile моделі розробки (розділи 1, 2), моделювання (розділи 2, 3).

Практична значущість. Розроблено та впроваджено з існуючим підприємством програмне забезпечення для діяльності інтернет-супермаркету. Веб застосунок розміщено на домені svitovochiv.com.ua.

Робота складається зі вступу, трьох розділів, висновків та списку літератури одного додатку. Обсяг роботи становить 58 сторінок, 3 таблиці, 12 рисунків.

РОЗДІЛ 1
АНАЛІЗ ТА ОБРАННЯ ТЕХНОЛОГІЙ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд розвитку клієнт-серверного програмного забезпечення

Розвиток клієнт-серверного програмного забезпечення відбувався паралельно з еволюцією комп'ютерних мереж та інтернету, відображаючи зміни в технологіях та потребах користувачів. Початкові етапи цього розвитку можна простежити назад до ранніх днів обчислень, коли мережеві взаємодії були обмежені простим обміном файлами між системами.

Ранні роки в 1960-х та 1970-х роках, з появою перших комп'ютерних мереж, таких як ARPANET, основні концепції мережевих взаємодій почали формуватися. Ці мережі дозволяли обмінюватися даними між віддаленими комп'ютерами, що можна вважати примітивною формою клієнт-серверної архітектури. У цей час головним завданням було забезпечити спільний доступ до дорогих обчислювальних ресурсів та програмного забезпечення.

1980-ті роки: поширення персональних комп'ютерів з появою персональних комп'ютерів та локальних мереж (LAN) у 1980-х, клієнт-серверні архітектури почали набувати популярності як спосіб спрощення обміну даними і розподілу програмного забезпечення в офісних умовах. Це був час, коли бази даних, електронна пошта та інші додатки стали використовуватися в рамках клієнт-серверних взаємодій.

1990-ті роки: з розвитком Інтернету та широким поширенням всесвітньої павутини в 1990-х, клієнт-серверна архітектура стала основою для багатьох онлайн-сервісів та веб-додатків. У цей період було презентовано рішення які зробили вагомі внески в браузери. 1990 рік: Тім Бернерс-Лі розробив перший веб сервер та графічний браузер з назвою WorldWideWeb, для об’єднання комп’ютерів у CERN. Під час розробки цієї системи було створено три фундаментальні технології:

– HTML: мова розмітки (форматування);

– URI: уніфікований ідентифікатор ресурс;

– HTTP: протокол передачі гіпертексту. Дозволяє отримувати пов'язані ресурси з Інтернету.

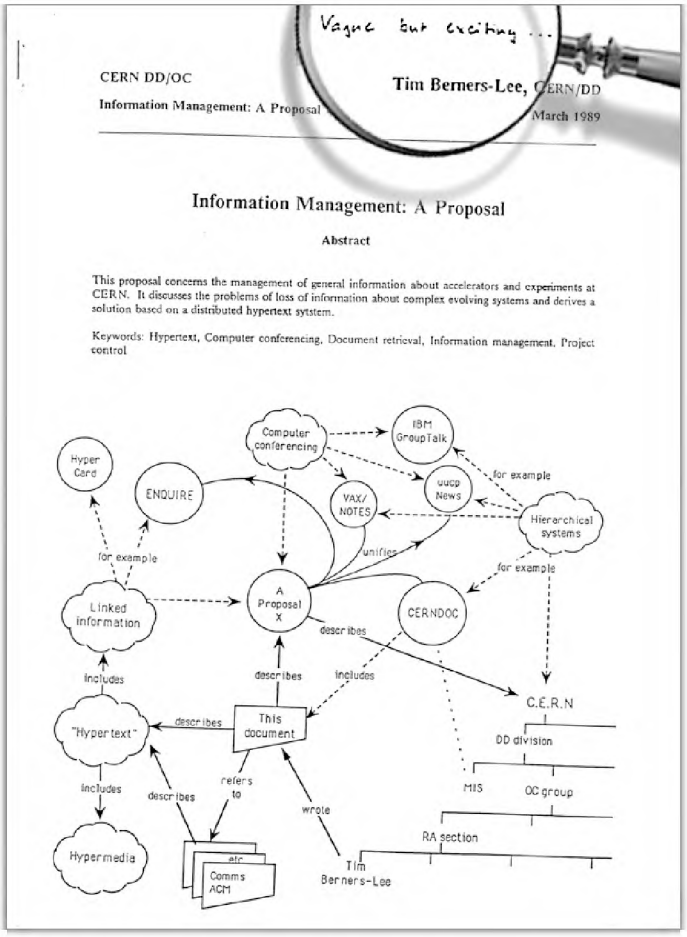


Рисунок 1.1 – Пропозиція Тіма Бернерса-Лі по створенню ІС інформаційного менеджменту

У квітні 1993го року код браузеру розроблений в CERN став загально доступним. Це надало можливість іншим компаніям розробляти свої браузери та продавати їх компаніям. Протягом двох років було створено три основних браузери NCSA Mosaic, Netscape Navigator та Internet Explorer. Через це виникла проблема несумісності браузерів між собою, навіть попри те, що у 1993 році було засновано W3C (консорціум з питань розвитку браузерів та інтернету). Netscape створив і випустив JavaScript, який надав веб-сайтам потужні обчислювальні

можливості, яких вони ніколи раніше не мали. Microsoft відповіла створенням каскадних таблиць стилів (CSS), які стали стандартом для розробки веб сторінок. Не обійшлося і без судових справ. Вже тоді компанії розуміли, що домінування одного продукту на ринку може призвести до значних набуття значного впливу на розвиток відкритої технології. Тому, коли Microsoft досягла майже повного контролю над ринком інтернет-браузерів, це викликало занепокоєння щодо здорової конкуренції та вільного вибору для споживачів. Саме ці опасіння підштовхнули до судових позовів звинувачення в порушенні антимонопольного законодавства. Це усвідомлення нагальної потреби в розширенні варіативності на ринку інтернет-браузерів спонукало Netscape відповісти не лише судовим шляхом, але й стратегічно, шляхом відкриття коду свого браузера. Такий крок мав на меті залучення ширшої спільноти розробників до поліпшення та розвитку програмного забезпечення, що могло б стати вагомою альтернативою продуктам Microsoft. Створення Mozilla Foundation та випуск Firefox стали втіленням цієї ідеї, спрямовані на відродження конкуренції та інновацій у сфері веб-перегляду.

Ранні розробники інтернету започаткували низку інноваційних концепцій, які тепер знайшли застосування далеко за межами сфери технологій:

– децентралізація: щоб опублікувати матеріал в Інтернеті, не потрібно отримувати дозвіл від жодної централізованої влади, оскільки відсутній єдиний центр управління, а отже – відсутня і єдина точка збою або "вимикач". Це забезпечує захист від довільної цензури та спостереження;

– відсутність дискримінації: коли дві особи сплачують за інтернет-підключення однакової або кращої якості, вони мають можливість спілкуватися на рівних умовах. Цей принцип рівноправності також відомий як принцип мережевого нейтралітету;

– розробка за методикою знизу вгору: на відміну від ситуації, коли код пишеться та керується обмеженою кількістю експертів, розробка велася в умовах відкритості, стимулюючи активну участь та інноваційні експерименти;

– універсальність: для забезпечення можливості публікації будь-чого будь-ким в інтернеті, всі пристрої, що беруть участь у комунікації, повинні

"говорити" між собою на одній мові, незалежно від технічних характеристик, географічного розташування чи культурно-політичних відмінностей користувачів. Мережа таким чином знищує ізоляцію, дозволяючи розквітати різноманіттю;

– консенсус: для функціонування універсальних стандартів необхідно було досягти загальної згоди щодо їх використання. Тім Бернерс-Лі та інші піонери досягли цього консенсусу, надавши всім можливість внести свій вклад у розробку стандартів через відкритий і прозорий процес участі в W3C.

2000-ті роки: на початку 21 століття з появою хмарних обчислень та мобільних технологій клієнт-серверні архітектури переживали черговий етап еволюції. Хмарні платформи дозволили розміщувати серверні компоненти в інтернеті, забезпечуючи високу доступність, масштабованість та гнучкість. Мобільні пристрої, з іншого боку, стали новим типом клієнтів, які потребують постійної синхронізації даних та доступу до онлайн-сервісів.

Окремо варто розглянути дві поширених [4] мови програмування клієнт-серверних систем. Java та JavaScript. Ці мови створювалися для вирішення однієї задачі, однак їх створення, еволюція та поточний стан доволі різний.

Створення Java, інша назва Oak, визначається своєю революційністю. Створюючи цю мову прагнули подолали головну проблему розробки програмного забезпечення на початку 1990х років – код написаний під одну платформу не міг працювати на іншій. Це відбувалося через використання різних інструкцій на різних процесорах. Java використовує проміжний етап – компіляцію в байт код. Інтерпретатор читає байт-кодові інструкції по одній, декодує їх і виконує безпосередньо. Цей процес не передбачає компіляції у машинний код, що означає, що він може запускатися швидко, але працює повільніше, ніж нативний код, оскільки кожна байт-кодова інструкція інтерпретується індивідуально під час виконання.

Розвиток інтернету в 1995 році задовольняв багатьох користувачів. Статичні веб-сайти з посиланнями на інші сторінки дозволяли швидко шукати інформацію. Для обчислення використовувався CGI [5]. Через невелику

пропускну здатність інтернету та невелику потужність серверів, користувачі могли чекати десятки секунд для отримання результату. Відсутність інтерактивності не дозволяло будувати повноцінні клієнт-серверні системи. В той час використовували розробку під комп'ютерні платформи, що несло в собі певні вади такі як: великі часові витрати на оновлення клієнтського застосунку на багатьох пристроях, підтримка ПЗ на різноманітних платформах. Для адаптації Java, тоді ще Oak, для виконання коду переданого через інтернет мережу потрібно було створити лише браузер, цим браузером став WebRunner. Програмами які передавалися через інтернет та запускалися в браузері назвали Applet. Таким чином Java стала провідною мовою, яка першою надала подібний досвід в браузері.

Для реалізації інтерактивності в браузері Netscape Navigator розглядалася й інтеграція Applet, однак було вирішено створити спеціальну, більш просту у використанні[6]. Сьогодні найбільш розповсюдженою мовою програмування в браузері є сценарна мова JavaScript [7], перша назва Mocha. Вона була розроблена після мови Java (Oak) компанією Netscape. За 10 днів було розроблено перший прототип та презентовано бета версію. У 1998 році Microsoft випустили свою мову сценаріїв для браузерів JScript. Вона була дуже подібна до JavaScript синктантично, однак дуже відрізнялася в використанні. Це призвело до ускладнення підтримки цих двох мов веб-сайтами [8]. Складність підтримки JScript та JavaScript можна проілюструвати їхніми відмінностями та унікальними можливостями, які вони пропонують. Наприклад, JScript, який є реалізацією стандарту ECMAScript від Microsoft, підтримує деякі синтаксичні особливості, яких немає в JavaScript. Однією з таких особливостей є присвоєння елементів у масив за індексом. Реалізація в JScript: f(x) = y, та в JavaScript f[x] = y. Деякі веб-сайти працювали тільки з однією мовою, інші мали спеціальну логіку для передачі сторінки з певною мовою. У кінці 1996 Netscape передало свою мову організації ECMA [9], метою було створення єдиного стандарту для мови JavaScript. Microsoft долучилася до розробки та впровадження єдиного стандарту та приймала участь до 2004 року, до поки браузер Internet Explorer не набув 95%

ринку [10]. Монополізація мови JavaScript припинила активний розвиток. Браузер Firefox та сервіси Google порушило монополію Microsoft, використовуючи нові технології. Google застосував технологію AJAX [11] у Gmail та Google maps, що дозволяло оновлювати дані без перезавантаження веб сторінки. Наступним великим кроком для JavaScript став новий двигун, реалізований в браузері Chrome, V8 на основі якого Раян Даль створив автономну систему виконання JavaScript – NodeJS. Можливість запускати JavaScript практично будьте, створило з нього повноцінного конкурента Java та іншим мовам. Знаючи лише одну мову програмування розробник може створювати клієнт серверні системи, що зменшило вартість розробки.

1.2 Аналіз існуючих рішень в інтернет торгівлі

На зламі тисячоліть, разом зі стрімким розвитком інтернет-технологій, клієнт-серверні системи зазнали значних змін, які проклали шлях для експансії інтернет-комерції. Вебсервери стали більш потужними та здатними обслуговувати мільйони запитів від користувачів з усього світу, що стало важливим поштовхом для зростання електронної комерції. Клієнт-серверні архітектури забезпечили можливість створення персоналізованого досвіду покупок для мільйонів користувачів, зберігаючи при цьому високу продуктивність, гнучкість та швидкість роботи. Розділяючи систему на клієнтську та серверну частину компанії можуть створювати централізовані сховища даних, надаючи однакову інформацію для багатьох клієнтів.

Безпека у таких системах зараз є критично важливою, зі створенням SSL (Secure Sockets Layer) та його наступника TLS (Transport Layer Security) клієнт-серверні системи стали забезпечувати безпечний обмін даними між браузерами користувачів та серверами, що стало основою для безпечних онлайн-платежів. Інтеграція з третіми сторонами: клієнт-серверні технології спростили інтеграцію інтернет-магазинів з різноманітними сервісами та платформами, такими як

системи електронних платежів, CRM-системи, логістичні служби, що дозволило створювати комплексні рішення для е-комерції. Мобільна комерція: з появою смартфонів та розвитком мобільного інтернету клієнт-серверні системи адаптувалися до нових вимог, забезпечуючи користувачам зручний доступ до інтернет-магазинів з будь-якого пристрою. Мобільні додатки для е-комерції стали невід'ємною частиною ринкової стратегії багатьох компаній. я галузі клієнт-серверних технологій продовжує формувати майбутнє інтернет-комерції. З використанням штучного інтелекту для аналізу даних користувачів та автоматизації персоналізованих рекомендацій, інтеграцією з Інтернетом речей (IoT) для створення інноваційних споживчих досвідів, до розвитку блокчейн-технологій для забезпечення більшої прозорості та безпеки транзакцій, клієнт-серверні архітектури продовжують бути в авангарді технологічних інновацій у сфері електронної комерції.

У перспективі розвитку цифрових медіа та інтернет-технологій проглядається тенденція до зміщення акценту з гіпертекстової структури на розширення інтерактивних можливостей, починаючи з покращення споживацького досвіду, використовуючи сучасні технології передачі даних закінчуючи сферою віртуальної реальності. Вже кілька десятиліть ідея віртуальної реальності залишається в полі зору наукових та масових досліджень, що ілюстровано, зокрема, у фільмі «Перший гравець приготуватися» 2018 року випуску та сучасними масовими VR шоломами. Останнім часом, завдяки стрімкому прогресу в розвитку інтернет-інфраструктури, стала можлива практична реалізація віртуальної реальності на широкому ринку. Сучасні веб-браузери, як наприклад Firefox, інтегрували підтримку WebVR та A-Frame, що забезпечує розробникам високоефективні інструменти для створення імерсійного веб-контенту у сфері віртуальної реальності. Ці технології вже у найближчому майбутньому дозволять з легкістю трансформувати стандартний веб-контент у віртуальні сцени, що взаємодіють з користувачем на новому рівні інтерактивності.

1.3 Актуальний стан хмарних технологій

Основна ідея клієнт-серверного програмного забезпечення полягає у зберіганні даних та їх основній обробці на сервері, щоб мати єдине джерело правди. У контексті хмарних технологій це означає використання віддалених серверів, доступних через Інтернет, для обробки та зберігання даних, що дозволяє компаніям оптимізувати витрати та підвищити ефективність. Наразі на ринку хмарних обчислень присутня велика конкуренція між різними компаніями, що надає можливість отримувати оптимальні потужності за найкращою ціною [12]. Для прикладу, платформи як Amazon Web Services (AWS) та DigitalOcean пропонують різні сервіси хмарних послуг, що надаючи гнучкі хмарні технології для потреб різних бізнесів. Лише AWS надає більше ніж 300 сервісів [13]. Основні характеристики хмарних обчислень:

1. Самообслуговування: користувачі можуть автоматично налаштовувати обчислювальні можливості, такі як час сервера і мережеве сховище, без втручання провайдера послуг. Відповідно користувачі можуть розпоряджатися використовувати лише VPS (AWS ec2 або droplet від DigitalOcean) та розгортати на цих серверах необхідні системи.

2. Широкий географічний доступ: хмарні ресурси доступні через мережу і широко розповсюджені по всьому світові. Це дозволяє надавати швидкий доступ до серверу чи інших ресурсів максимально швидко.

3. Швидка еластичність: ресурси можуть бути швидко масштабовані вгору чи вниз, забезпечуючи враження необмеженої пропускної спроможності. Це може бути зроблено в ручну так і за допомогою автоматизації балансування навантаження. Більшість провайдерів пропонують запускати сервер лише за потреби, тобто якщо певна дія має відбуватися рідко, то її можна виконувати за допомогою сервісів таких як AWS Lambda.

4. Оптимізація витрат: використовуючи хмарні сервіси, бізнес оптимізує витрати на багатьох аспектах, на налаштовуванні, розміщенні, підтримці

фізичних серверів, на самих потужностях серверів, задопомогою еластичності хмарних обчислювань.

Витоки даних у хмарних середовищах можуть відбуватися через різноманітні вектори атак, включаючи конфігураційні помилки, несанкціоноване використання API або через шкідливі програми. Зокрема, неправильно налаштовані хмарні сховища можуть стати легкою ціллю для зловмисників, що шукають легкий доступ до чутливих даних. На додаток, оскільки дані у хмарі часто розподіляються між різними географічними локаціями, це ускладнює та одночасно спрощує їх захист від фізичних атак або природних катастроф. Від втрати даних при подібних атаках може допомогти дублювання даних у інших регіонах світу.

Не менш значущою є проблема несанкціонованого доступу, що є проблемою не тільки хмарних середовищ а в цілому. Хакери можуть використовувати слабкі паролі, вразливості в програмному забезпеченні або техніки соціальної інженерії для отримання доступу до облікових записів різних рівнів та відповідно викрасти дані або використати ресурси серверу на власну користь. Людський фактор можна подолати задопомогою MFA та постійних тренувань робітників.

Для стартапів та малих підприємств, які розглядають можливість інтеграції хмарних технологій у свої бізнес-процеси, отримання хмарних кредитів від провайдерів може стати значущим фактором у виборі платформи. Хмарні кредити дозволяють знизити витрати на початкові етапи впровадження хмарних технологій, надаючи компаніям можливість експериментувати та оптимізувати свої хмарні ресурси без значного фінансового навантаження. Хмарні кредити часто пропонуються як частина стимулюючих програм від провайдерів таких як AWS, Google Cloud і Microsoft Azure. Ці кредити можуть бути використані для покриття витрат на обчислення, зберігання даних, передачу даних і інші хмарні сервіси. Наприклад, Amazon Web Services надає різні програми для стартапів, освітніх інституцій та некомерційних організацій, які включають безкоштовні кредити, здатні значно знизити вартість впровадження та експлуатації хмарних

рішень. AWS вже видав більше 6 мільярдів доларів у вигляді кредитів на свої ресурси [14] кредитів для стартапів через їхню програму AWS Activate, яка спеціалізується на підтримці нових бізнесів з потенціалом швидкого зростання. Такі програми не тільки сприяють зростанню бізнесу за рахунок зниження початкових витрат, але й дозволяють стартапам експериментувати з різними технологічними рішеннями без ризику значних фінансових втрат. За інформацією з Argonaut [15], крім прямої економії коштів, хмарні кредити також відкривають доступ до преміум-функцій і підтримки, які можуть бути недоступні за стандартними планами. Це включає розширену підтримку, консультації з безпеки, та інші послуги, які можуть підвищити ефективність використання хмарних ресурсів та сприяти більш успішній інтеграції технологій у бізнес-процеси компанії.

Загалом хмарні технології є чудовим вибором для малих та середніх за розміром ІС. Вони надають великий спектр вже готових рішень: автоматичне розгортання поетапне коду на всі сервери, легкі у налаштуванні мережі та брандмауери для забезпечення від певних типів зламів.

1.4 Огляд технологій розробки клієнт-серверного сучасного програмного забезпечення

У сучасному світі розробка клієнт-серверних програмних систем вимагає застосування надійних та ефективних технологій, особливо коли мова йде про реалізацію масштабованих корпоративних (enterprise) рішень. Більшість бізнесів використовують мову програмування Java (90% компаній з Fortune 500 використовують Java як основну мову програмування [16]). Java відіграє ключову роль у цьому процесі. Її широке використання пояснюється низкою факторів: стабільність, переносимість, зворотна сумісність і велика кількість наявних бібліотек та фреймворків. Java здатна підтримувати високу продуктивність у розподілених системах, що є вирішальним для корпоративних

застосунків, де надійність та безпека є критично важливими. Багато великих компаній, такі як Google, Amazon, та Netflix, обирають Java для розробки складних серверних систем та мікросервісів завдяки її здатності оптимізувати використання ресурсів та управління пам'яттю. Фреймворки, такі як Spring і Hibernate, забезпечують потужні інструменти для створення модульних, легко тестованих та легко масштабованих застосунків. Крім того, віртуальна машина Java (JVM) дозволяє додаткам виконуватися на будь-якій платформі, що підтримує JVM, без необхідності переписування коду для кожної конкретної системи, що робить Java ідеальним вибором для компаній, що прагнуть до високого рівня переносимості своїх рішень. Перевагу віртуальної машини зараз демонструє легка міграція на сервери з процесорами на новій, для серверів, архітектурі ARM. Ці особливості роблять Java вибором номер один для корпоративних систем, що вимагають рішень, здатних витримувати великі обсяги транзакцій та забезпечувати високий рівень послуг у режимі реального часу.

Незважаючи на переваги та популярність Java, у роботі з нею є недоліки, які можуть впливати на вибір технологій, особливо коли порівнювати її з JavaScript та NodeJS як середовище виконання. Головна перевага JavaScript це можливість розробляти програмне забезпечення для клієнтської частини (браузер) та серверної частини використовуючи одну мову програмування. Використовуючи Java теж можна створити клієнтську частину у вигляді застосунку, однак це ускладнить підтримку серверної частини через підтримку різних версій клієнтів. Зважаючи на те що інтернет-супермаркет має веб сайт для споживачів, за для оптимізації часових витрат та перевикористання коду, краще розробити й клієнтську частину для менеджерів у браузері. NodeJs є більш оптимальним вибором для систем у яких є велика кількість подій [17], це реалізовано задопомогою event-loop у двигуні V8 [18].

Бази даних в клієнт-серверній архітектурі можуть мати різні ролі у залежності від архітектури системи. У контексті розробки клієнт-серверних програмних забезпечень, таких як інтернет-супермаркети, важливо розуміти

відмінності та застосування різних типів баз даних. Існують три головних типи баз даних реляційні, документо-орієнтовані, OLAP. Кожен тип вирішує певний спектр проблем.

Документо-орієнтовані бази даних надають можливості дозволяє легко модифікувати структуру даних, є більш оптимізованими для запису та читання даних у порівнянні з SQL базами даних. Відсутність чітких схем даних є недоліком, який може заважати використовувати даний тип баз даних як основну, однак це є і плюсом для застосування у ролі бази даних для читання.

Реляційні бази даних, такі як PostgreSQL, MySQL, та Oracle, побудовані на принципах строгого дотримання схем даних та забезпечення транзакційної цілісності, що робить їх ідеальними для застосунків, де потрібна висока надійність та передбачуваність даних. Ці системи підтримують ACID-принципи (Atomicity, Consistency, Isolation, Durability), гарантуючи надійне виконання транзакцій та збереження консистентності даних, що є критично важливим для фінансових операцій, управління запасами, та інших ділових додатків. Перевага реляційних баз даних включає їхню здатність до ефективного виконання складних запитів через SQL – могутню та універсальну мову, яка дозволяє детально керувати даними і проводити глибокий аналіз. Стандартизація SQL сприяє легкості навчання, широкій підтримці з боку розробників, та високому рівню інтеграції з численними додатками та сервісами. Однак, реляційні бази даних мають свої недоліки. Основним обмеженням є складність горизонтального масштабування; розширення ресурсів шляхом додавання більше серверів часто вимагає складних і дорогих рішень.

OLAP (Online Analytical Processing) системи є спеціалізованими рішеннями, які оптимізовані для швидкого аналізу та обробки великих обсягів даних. Ці системи надзвичайно важливі для організацій, які потребують глибокого аналітичного інсайту для прийняття рішень на основі даних. OLAP дозволяє ефективно виконувати складні запити, не впливаючи на продуктивність транзакційних систем. Переваги OLAP включають їхню здатність підтримувати багатовимірні запити, що дозволяє користувачам швидко дивитися на дані під

різними кутами. Наприклад, аналітик може легко порівняти продажі по різних регіонах, продуктах, та часових періодах з декількома рівнями агрегації. Це особливо корисно для виконання складних аналітичних задач, таких як часові серії, та мульти-вимірний аналіз витрат і доходів. Мінуси OLAP полягають у тому, що вони зазвичай вимагають значних ресурсів для зберігання даних через необхідність підтримки великих обсягів історичних даних, які повинні бути доступні для аналізу в режимі реального часу. Це може призводити до високих витрат на зберігання та управління даними. Крім того, OLAP системи можуть бути складними у налаштуванні та обслуговуванні, вимагаючи спеціалізованих знань для ефективного використання їхнього потенціалу. Один з аспектів OLAP баз даних, це складність у створені та оновлені даних, дані можуть бути не додані, або додані пізніше. Всі ці проблеми існують через максимальну оптимізацію OLAP для обробки даних.

РОЗДІЛ 2
АРХІТЕКТУРА ТА ФУНКЦІОНАЛ КЛІЄНТ-СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вимоги до клієнт-серверного програмного забезпечення

Необхідно розробити клієнт-сервере програмне забезпечення для діяльності інтернет-супермаркету. Має бути два вебклієнти, перший для менеджерів, другий для споживачів.

Функціональні вимоги для веб-клієнту менеджерів:

1. Менеджер має можливість увійти в систему задопомогою паролю та електронної пошти.

2. Менеджер має можливість оновити товар за допомогою excel файлу з певною структурою на сторінці прайс. Приклад структури таблиці додаток А.

3. При завантажені нового прайсу старі товари не видаляються а позначаються як не активні.

4. При завантаженні нового прайсу файл перевіряється на дублікати, у разі дублікатів операція завершується та показує помилку про наявність дублікатів та місце їх розташування.

5. Менеджер має можливість переглядати активні товари на сторінці прайс у вигляді таблиці з колонками: назва, тип, ціна.

6. Менеджер має можливість переглядати замовлення клієнтів у вигляді таблиці, яка містить основну інформацію про замовлення: Ім’я та прізвище, Адреса доставки, сума замовлення, статус, дата створення. Має бути можливість фільтрування за статусами замовлення та сортування за часом створення. За замовчуванням має бути сортування від нових до старих та показувати за всіма статусами.

7. Кожен рядок у таблиці замовлення клієнтів має відкриватися та показувати детальну інформацію про замовлення. Зовнішній вигляд таблиці може бути мінімалістичним.

8. Сторінка детальна інформація про замовлення має містити адреса доставки, інформація про отримувача, дати створення та оновлення, спосіб оплати, сума замовлення, статус замовлення, таблицю з замовленими товарами. Статус замовлення може бути змінений менеджером з новий на доставлено.

9. Таблиця замовлені товари на сторінці детальна інформація про замовлення клієнта має містити такі колонки: назва товару, ціна за тип, кількість, сума. Можливість розгорнути поле таблиці для перегляду детальної інформації. Зовнішній вигляд таблиць не є критичний.

10. Менеджер має можливість вийти з акаунту.

Функціональні вимоги для вебклієнту споживачів:

1. Вебклієнт для споживачів має бути розроблений з урахуванням розмірів та іншими особливостями мобільних пристроїв.

2. Вебклієнт має містити в собі певні перевикористовувані секції:

2.1. Верхня секція 1: кнопка для відкриття меню, назва сайту, поле для пошуку товарів за назвою.

2.2. Верхня секція 2: кнопка для відкриття меню, назва сайту або логотип.

2.3. Нижня секція 1: час доставки, сума для мінімального замовлення, індикатор для візуалізації скільки ще не вистачає до мінімального замовлення, кнопка для навігації користувача у кошик та суму товарів у кошик.

2.4. Нижня секція 2 суму товарів у кошику та : кнопка замовити для навігації до сторінки оформлення даних у замовлені.

2.5. Нижня секція 3: кнопка підвередити замовлення для підтвердження замовлення та навігації до головної сторінки.

3. Головний екран має містити секції: нижня секція 1 та верхня секція 1. Має відображати список активних товарів у вигляді карток. Споживач має можливість найти бажаний товар увівши неповну назву у полі пошуку. Список товарів має кнопки для зміни сторінки та показує поточну сторінку, скільки сторінок всього.

4. Картка товару на головному екрані містить: назву товару, ціну за певну одиницю товару, інтерактивні кнопки для зміни кількості товарів у кошику.

Споживач може змінити кількість товарів у кошику натискаючи кнопку плюс, мінус або увести число в поле кількості товару.

4.1. Користувач не може увести не валідну кількість товарів додаток. Значення кількості певного товару у кошику не може бути від'ємним, у випадку з не подільними типами товарів як штуки, упаковки, пляшка кількість не може бути не цілим числом.

4.2. При зменшенні кількості до 0, товар має бути видаленим з кошика.

4.3. При оформленні замовлення кошик має очищуватися та показувати користувачу інформаційне повідомлення.

4.4. Споживач може змінювати кількість товарів у кошику.

4.5. Коли товар стає не доступним для покупки, він має позначатися певним чином в кошику.

5. Бокове меню містить навігаційні або інтерактивні кнопки: замовлення – навігація до попередніх замовлень, про нас – навігація до сторінки про компанію, увійти – кнопка для входу споживача у систему задопомогою гугл, вийти – кнопка для виходу користувача з системи.

6. Сторінка кошик містить секції: верхня секція 2 та нижня секція 2. На сторінці показується список доданих товарів у кошик, якщо товар став не активний показується інформація що товар на разі не доступний.

7. Веб-клієнт для споживачів має бути оптимізованим під пошуковий двигун від Google.

8. Програмне забезпечення має витримувати для споживачів має витримувати навантаження в 200 – 2000 користувачів одночасно. Опрацьовувати 10 замовлень на хвилину.

Додаткові вимоги:

1. Низька вартість розробки.

2. Низька вартість обслуговування.

3. Короткий час на реалізацію.

4. Доступність веб-сайту для користувачів 99% часу.

5. Легкість розвитку в майбутньому.

2.2 Окреслення етапів проектування архітектури для клієнт-серверного програмного забезпечення

Розробка архітектури починається з перекладу бізнес-термінів функціональних вимог та аналізу інформації. Бізнес надає інформацію про важливі для нього аспекти, такі як низька собівартість, гнучкість та функціональні вимоги, які мають бути розшифровані з точки зору архітектури програмного забезпечення. Важливо зазначити, що архітектура не повинна бути переукладеною для досягнення вимог, але має забезпечувати їх виконання.

Перший крок у розробці архітектури – створення єдиного словника. Це забезпечує спільне розуміння термінів та понять серед усіх учасників проекту, уникаючи непорозумінь та сприяючи ефективній комунікації між розробниками, бізнес-аналітиками та іншими зацікавленими сторонами.

Другий етап – визначення базових абстракцій, які є критично важливими для проектування архітектури. Цей етап включає ідентифікацію основних компонентів системи, їх взаємодій та ролей. Визначення базових абстракцій допомагає створити основу для побудови більш детальних архітектурних рішень і забезпечує гнучкість та масштабованість системи.

Третій етап – описання загальної архітектури. Цей етап передбачає створення високорівневої моделі системи, яка включає основні компоненти, їх взаємодію та інтеграцію з іншими системами. Загальна архітектура має забезпечити чітке бачення структури системи та її основних функцій.

Четвертий етап – створення архітектури та принципів розробки серверної частини. Цей етап включає визначення технологій, інструментів та підходів, які будуть використовуватись для розробки серверної частини системи. Основні компоненти серверної частини, їх взаємодія та інтеграція з іншими системами визначаються на цьому етапі.

П'ятий етап – створення архітектури та принципів розробки клієнтської частини. Цей етап передбачає визначення технологій, інструментів та підходів, які будуть використовуватись для розробки клієнтської частини системи.

Визначаються основні компоненти клієнтської частини, їх взаємодія з серверною частиною та користувачами.

2.3 Створення єдиного словника

Доцільність у створені єдиного словника є у створенні контексту системи. Люди з різним досвідом можуть називати однакові сутності різними іменами, певні сутності можуть мати схоже написання але різні значення. Кожне слово має певне значення та всі залучені сторони знають це слово лише у цьому значенні.

Споживач – особа або організація, яка використовує або має намір купувати товари або послуги через систему.

Працівник – особа, задіяна в операціях інтернет-магазину, яка може мати різні ролі, такі як адміністратор, менеджер замовлень, консультант з підтримки клієнтів тощо.

Товар – фізичний або цифровий об'єкт, який пропонується для продажу в інтернет-магазині.

Тип товару – категорія, яка описує одиницю виміру товару для продажу (наприклад, кілограм, грам, штука, упаковка, пляшка тощо).

Прайс – перелік усіх товарів, доступних для покупки, із зазначенням цін та інших відповідних деталей, таких як тип товару або доступні варіанти.

Кошик – тимчасове місце зберігання в системі, де споживачі можуть зберігати товари, які вони планують купити.

Товар у кошику – окрема одиниця товару, що була додана споживачем до кошика, перш ніж відбудеться оформлення замовлення.

Замовлення – запит споживача на купівлю та доставку вибраних товарів, що містить усю відповідну інформацію, таку як перелік товарів у кошику, спосіб доставки, адресу, та платіжні дані.

Замовлений товар – конкретний товар, який був вибраний споживачем та включений до замовлення.

Статус замовлення – опис стану замовлення на різних етапах його обробки. Наприклад: «нове», «в обробці», «відправлене», «доставлене», «скасоване».

Одиниця зберігання (SKU) – унікальний ідентифікатор, який дозволяє точно ідентифікувати кожний товар в системі, включаючи його розмір, колір та інші характеристики.

Споживацький профіль – дані споживача, збережені в системі, що містять інформацію про замовлення, преференції покупок, історію покупок та особисті налаштування.

2.4 Визначення базових абстракцій моделей

Для побудови правильної архітектури потрібно визначити абстракції моделей які є в системі. Головний принцип – кожна архітектурна модель є наближенням до реальності. Іншими словами, вона лише частково точна і частково повна.

Абстракція клієнт – ця сутність має найменше прав доступу в системі, може взаємодіяти з іншими моделями такі як: кошик, прайс, товар та замовлення.

Атрибути:

– Унікальний ідентифікатор, який дозволяє системі ідентифікувати клієнта;

– Ім’я;

– Прізвище;

– Електрона адреса;

– Телефонний номер.

Операції які може робити клієнт:

– Реєстрація: створення нового акаунта клієнта в системі;

– Авторизація: вхід в систем;

– Оновлення профілю: зміна даних профілю клієнта, таких як адреса, телефон, ім’я та прізвище;

– Перегляд історію замовлень: Дозволяє клієнту переглядати свої попередні покупки;

– Додавати товари до кошику;

– Видаляти товари з кошику;

– Змінювати кількість товарів у кошику;

– Оформлювати замовлення.

Обмеження клієнта:

– Клієнти не можуть змінювати електронну адресу після реєстрації.

Абстракція товар – є основною одиницею продажу в інтернет-магазині. Кожен товар характеризується унікальним ідентифікатором та набором атрибутів, які описують його характеристики.

Атрибути:

– Унікальний ідентифікатор (SKU), який дозволяє системі ідентифікувати товар.

– Назва товару, яка використовується для його визначення та відображення в каталозі;

– Вартість товару, зазвичай вказується в місцевій валюті;

– Стан товару – визначає стан наявності товару для продажу.

Операції:

– Додавання нового товару: Створення нового запису товару в системі;

– Оновлення інформації про товар: Модифікація існуючих даних товару, таких як ціна, опис, доступність;

– Видалення запису товару з системи.

Зв'язки:

– Товар може бути доданий до кошика клієнта;

– Товар може бути частиною замовлення.

Обмеження:

– Мінімальна кількість: певні товари можуть вимагати замовлення мінімальної кількості одиниць;

– Вікові обмеження: певні товари можуть мати вікові обмеження для покупки (наприклад, алкогольні напої);

– Видалення товару з системи є небажаної операцією.

Абстракція прайс – Прайс формується автоматично включаючи в себе товари доступні для покупки. Не має фізичного відображення в системі для спрощення.

Абстракція кошик – є тимчасовим сховищем для товарів, які клієнт планує придбати. Він дозволяє користувачам зберігати вибрані товари, переглядати вартість сумарного замовлення, робити зміни у виборі товарів перед остаточним підтвердженням покупки.

Атрибути:

– Унікальний ідентифікатор кошика, що асоціює його з конкретним користувачем;

– Зв'язок з клієнтом, якому належить кошик;

– Список товарів, що включає ID товару, назву, кількість, ціну за одиницю та загальну вартість за товар;

– Сумарна вартість усіх товарів у кошику.

Операції:

– Додавання товару до кошика;

– Видалення товару з кошика;

– Зміна кількості товару;

– Очищення кошика;

– Підтвердження замовлення.

Зв’язки:

– Після підтвердження кошика, товари з кошика можуть бути переведені в замовлення.

Обмеження:

– Товари в кошику не гарантують їхню резервацію; наявність товарів перевіряється під час підтвердження покупки.

Абстракція замовлення – об’єкт який був створений з кошика. Містить інформацію про замовлені товари та інформацію про доставку

Атрибути:

– Унікальний ідентифікатор, який дозволяє системі ідентифікувати конкретне замовлення;

– Зв'язок з клієнтом, який оформив замовлення;

– Деталі товарів у замовленні, включно з ID товару, назвою, кількістю та ціною;

– Дата та час створення замовлення;

– Поточний статус замовлення (наприклад, «нове», «в обробці», «відправлене», «доставлене", «скасоване»);

– Сумарна вартість усіх товарів у замовленні;

– Метод оплати, обраний клієнтом.

– Місце доставки замовлення;

Операції:

– Перетворення кошика у фінальне замовлення та його збереження в системі;

– Зміна статусу замовлення на наступний етап обробки;

– Скасування замовлення до моменту його відправки.

Зв'язки:

– Замовлення належить конкретному клієнту;

– Замовлення містить один або декілька товарів;

– Замовлення інтегроване з платіжною системою для обробки оплати.

Обмеження:

– Термін дійсності замовлення: Якщо замовлення не підтверджене або не оплачене протягом визначеного часу, воно може бути автоматично скасоване;

– Зміни в замовленні: Після переходу замовлення у статус «в обробці», зміни в складі товарів можуть бути обмежені.

Візуалізація абстракцій за допомогою концептуальної діаграми для загального розуміння сутностей в системі рисунок 2.1.

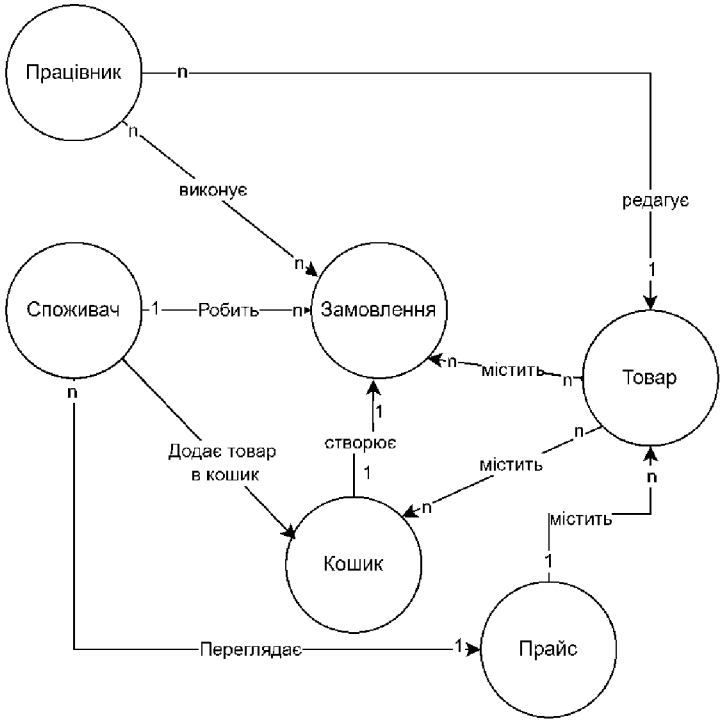


Рисунок 2.1 – Візуалізація концептуальної діаграми

Дана діаграма візуалізує абстракції та їх зв’язки, її можна використовувати для створення моделі бази даних або при створенні коду.

2.5 Обґрунтування вибору клієнт-серверної архітектури

Вибір архітектури базується на багатьох параметрах з таб. 2.1 [19] такий вибір дозволяє обрати оптимальну архітектуру. Обираючи архітектуру варто оцінювати ключові функціональні вимоги та в першу чергу можливості. Оскільки за вимогами це має бути мінімально життєздатне програмне забезпечення та бізнес може виділити мінімальну можливу кількість ресурсів на

розробку варто розглядати лише архітектурні підходи з найбільшими параметрами ціни та простоти. Однак не варто нехтувати легкістю в підтримці та масштабуванню.

Розглядаючи серверну архітектуру варто зосередитися на простоті в реалізації щоб задовільнити «Низька ціна розробки, низька ціна підтримки, короткий проміжок розробки». Найменшу ціну надають архітектури: шарова, модульна та мікрокернел. Розглядаючи ці три архітектури видно що мікрокернел має багато переваг над іншими, однак повноцінна реалізація вимагає більше витрат, через більші вимоги до аналізу доменної сфери та конфігурацій релізу. Найкращий варіант буде гібридна архітектура або не повноцінна мікрокернел архітектура. Спочатку архітектура буде модульним монолітом з можливістю стати мікрокернел. За потреби в оптимізації деякий функціонал може бути винесеним в окремі спеціалізовані сервіси.

Таблиця 2.1 – Рейтинг характеристик архітектур [19]

Характеристика	Шарова	Модульний моноліт	Мікросервіси	Подієво-орієнтована
Гнучкість	★	★★★★	★★★★	★★★★
Абстракція	★	★★	★★★	★★★★
Вартість	★★★	★★	★★★★	★★★★
Домена частина	★	★★★★	★★★★	★★★★
Еластичність	★	★	★★★★	★★★★
Відмовостійкість	★	★★★	★★★★	★★★★
Продуктивність	★★	★★★★	★★★★	★★★★
Масштабованість	★	★★	★★★★	★★★★
Простота	★★★	★★★★	★★★★	★★★★
Тестованість	★★	★★★★	★★★★	★★★★

Розробка клієнт-серверного ПЗ розділяє архітектури та дає можливість гнучко обертати оптимальні підходи для кожної частини системи. Ці частини обмінюються даними задопомогою http протоколу 1.1 та архітектурного стиля

Restful [21]. Відповідно це мінімізує залежності частин один від одного, спрощує розробку та реліз частин системи.

Архітектура клієнтської частини розділена має бути максимально простою та заточеною під вимоги. Вимогу ‘оптимізований вебсайт’ для клієнтів та вебсайт для працівників важко задовільнити створюючи один веб застосунок. Клієнтський вебсайт має бути максимально оптимізований для пошуковиків відповідно до функціональної вимоги «Оцінка в Google page speed» та «SEO оптимізація під пошуковик Google». Ці вимогу будуть задоволені створенням двох спеціалізованих веб застосунків. Кожен з цих застосунків буде направлений виключно на вирішення своїх задач, що дозволить обирати оптимальні підходи та технології.

2.6 Проєктування контролерів та команд

У серверну частину програмного забезпечення мають бути закладені принципи які дозволять змінити більш легко змінити архітектуру з часом, як це було зазначено у виборі архітектури. Звичайна шарова архітектура або модульний моноліт передбачає пряме звернення від контролера до сервісу для виконання команд, що створює складнощі при масштабуванні та декомпозиції логіки на окремі сервіси чи створення мікрокернел архітектури. Перший базовий принцип створюваної архітектури це команди. Команди виконують певну дію, контролер не знає про існування сервісів і до якого сервісу звертається команда. Це дозволить різним контролерам, сервісам чи іншим командам використовувати однакові команди для отримання чи надсилання даних. На рисунку 2.2 зображено підхід оновлення товарів в кошику. Товари можуть оновлюватися з багатьох причин. Це може бути дія споживача при додаванні товару в кошик або це може бути подія зміни наявних товарів і відповідно потрібно ввести зміни до кошика.

Команди розширюють функціонал сервісу виконуючи додаткові дії чи етапи при їх виклику, тим самим дотримується принцип єдиної відповідальності.

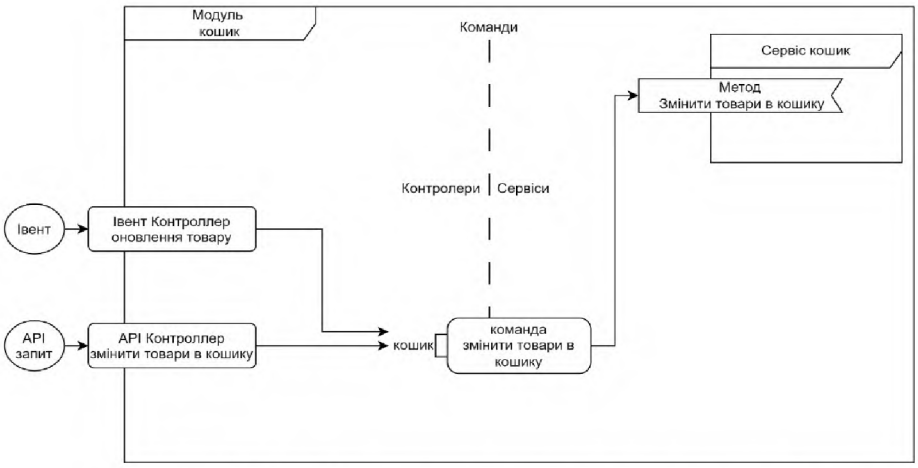


Рисунок 2.2 – Використання команди контролерами

На рисунку 2.3 зображено команду «змінити товар в кошику». Ця команда змінює товар в кошику та створює подію що повідомляє що товар було змінено відповідно цю подію може спожити обробник подій та оброблений вже за їх логікою. Таким чином сервіс «кошик» виконує лише свій функціонал.

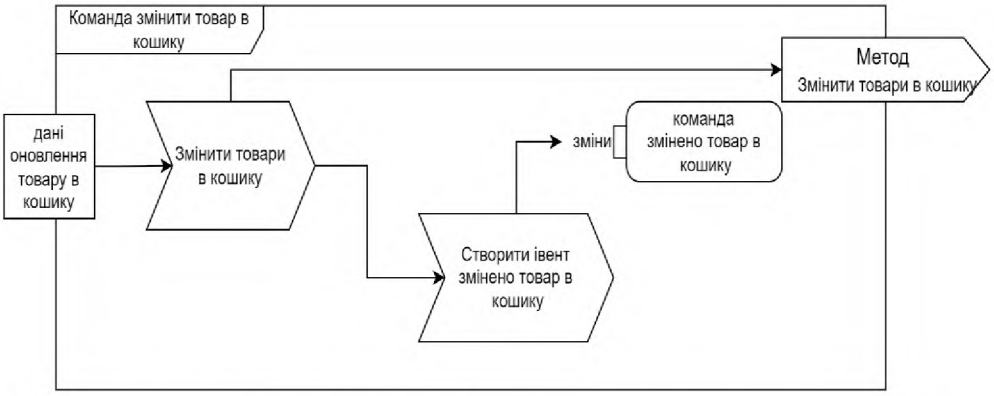


Рисунок 2.3 – Використання команд для створення сценаріїв

Обробка запитів до бази даних має відбуватися через спеціальні об’єкти такі як репозиторії та бути не прив’язаними до реалізації підключення та передачі даних до бази даних.

Балансування серверів має бути динамічним для розподілу вхідних запитів між декількома серверами. Збереження сесій в централізованому сховищі:

Використання зовнішніх сесійних сховищ, для управління станом користувача. Щоб не залежати від конкретного веб-сервера і взаємодіяти з будь-яким сервером без втрати даних сесії.

2.7 Розробка нефункціональних вимог до клієнтської частини

Створення архітектури та принципів розробки клієнтської частини для клієнт-серверного ПЗ для інтернет-супермаркету вимагає особливої уваги до користувацького досвіду, продуктивності та сумісності на різних пристроях. З метою покриття як найбільшої кількості користувачів застосовуються браузерні технології, які дозволять запускати клієнти на найрозповсюдженіших пристроях.

Клієнтська частина повинна бути розроблена з використанням модульного підходу. Кожен компонент (наприклад, корзина покупок, список товарів, користувацький профіль) має бути незалежним, що дозволяє легко змінювати або оновлювати окремі частини без впливу на решту системи.

Клієнтська частина для споживачів має бути максимально швидкою та адаптивною. Варто відмовитися від існуючих компонентних фреймворків для зменшення розміру клієнта. Застосування SSR (Server side rendering) технологій для оптимізованої доставки сторінок. Мінімальний розмір клієнту та швидкість відкриття першої сторінки підвисить якість SEO оптимізацію. Для ще більшого підвищення якості SEO оптимізації застосовуються спеціальні теги, які вказують на призначення компонента та описують вміст для пошуковиків [22].

Клієнтська частина для споживачів не має зберігати дані локально, додатково їх форматувати та містити мінімум необхідних даних для виконання функцій. Такого підходу можна досягти використовуючи рішення для кешування даних отриманих через АПІ. Таким чином мінімізуватиметься фактор дуплікації даних та зменшиться ризик розсинхронізації та виникнення інших помилок.

РОЗДІЛ 3
РОЗРОБЛЕННЯ
КЛІЄНТ-СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструментів розробки

За для ефективної розробки ПЗ були обрані технології які оптимізують людські ресурси та надають більше гнучкості в розробці.

GitHub – середовище версіонування ПЗ. Має GitHub actions – інструмент для автоматизації запуску функцій на певні події (наприклад запуск тестів перед об'єднанням гілки з розробки до основної гілки).

Windows 11 з WSL – операційна система з вбудованим ядром Linux. Дозволяє використовувати консольні команди Linux та покращує роботи Docker контейнерів.

WebStorm – редактор коду, оптимізований під велику кількість файлів. Має інтеграції з GitHub та один з найкращих статичних аналізаторів коду. У порівнянні з конкурентами VS code та Visual Studio code має такі переваги як: швидкий аналіз коду. Автоматична інтеграція з бібліотеками, більш зручний перегляд git.

DataGrip – інтегроване середовище розробки (IDE) для роботи з базами даних. Він надає зручний та ефективний спосіб перегляду, редагування та управління вашими базами даних. Однією з його ключових переваг є широкий спектр підтримуваних баз даних, включаючи MySQL, PostgreSQL, SQLite, Oracle, SQL Server та інші. Це робить DataGrip ідеальним інструментом для розробників та адміністраторів баз даних, які працюють з різними системами у своїй роботі. Додатково, DataGrip надає розширені можливості для роботи з SQL, включаючи автодоповнення коду, візуальні засоби для аналізу запитів та підтримку роботи з проектами баз даних будь-якої складності.

Ngrok – це інструмент для роботи з веб-додатками, який дозволяє локальним серверам та сервісам знаходитися за кордоном веб-інтернету. Він створює безпечні тунелі до вашого локального сервера, що дозволяє вам швидко

та легко демонструвати ваш веб-додаток клієнтам або співробітникам, навіть якщо вони знаходяться за файрволами або в мережах з обмеженим доступом. Це дозволяє ефективно вести роботу з додатками та швидко надавати їм доступ до широкої аудиторі.

GitBash – це термінальна програма для роботи з Git та іншими командами командного рядка в середовищі Windows. Вона забезпечує зручний доступ до всіх функцій Git, таких як ініціалізація репозиторію, додавання та комітування змін, створення гілок та злиття, а також перегляд історії комітів. Крім того, GitBash надає можливість виконання різних команд командного рядка, що робить його потужним інструментом для автоматизації рутинних завдань в процесі розробки програмного забезпечення.

3.2 Розробка серверної частини

Для реалізації серверної частини клієнт-серверного ПЗ інтернет-магазину важливо ретельно підібрати технології, що спроможні задовольнити описану в розділі 2 архітектуру. Серверна частина є головною частиною ПЗ з точки зору обробки інформації. Технології мають забезпечувати високу швидкість розробки, обробку великої кількості запитів та забезпечувати стабільність. Цей розділ описує вибір та конфігурацію ключових технологій, що були впроваджені в серверній частині проекту, починаючи з мови програмування та закінчуючи допоміжними інструментами, які забезпечують високу продуктивність та надійність системи.

Розглядаючи технології для реалізації серверних застосунків з урахуванням архітектурних вимог з 2го розділу, було обрано NodeJS (JavaScript) як ядро застосунку. Для повноцінного задоволення архітектурним вимогам потрібно обрати технології: фреймворк, збереження даних, обробки черг та івентів.

Для вибору оптимального фреймворку для серверної частини інтернет-магазину, необхідно визначити низку ключових вимог, які відіграють роль у забезпеченні ефективності, надійності та масштабування системи:

– підтримувати модульність та легке інтегрування з іншими сервісами та базами даних;

– має використовувати мову програмування, що забезпечує строгу типізацію;

– мати добре описану документацію та активною спільнотою;

– обробляти великі обсяги запитів та даних без значних затримок;

– підтримка вертикального та горизонтального масштабування;

– вбудовані механізми захисту від загальновідомих вразливостей.

Технологія NodeJS має невелику кількість фреймворків: NestJs, Koa.js, Sails.js та Metarhia. Кожен з них має свої особливості:

NestJS використовує TypeScript, що забезпечує строгу типізацію та об'єктно-орієнтовані підходи. Його архітектура базується на Angular, що робить його ідеальним для великих масштабованих проектів зі складною бізнес-логікою.

Koa.js, створений командою, яка розробила Express.js, пропонує більшу гнучкість і сучасні функції для обробки HTTP запитів. Він легкий і забезпечує більший контроль над обробкою запитів завдяки використанню асинхронних функцій.

Sails.js підходить для швидкої розробки застосунків, оскільки має вбудовані шаблони для API та великий набір інструментів для роботи з даними. Однак, він може бути важким для дуже великих проектів.

Metarhia це український фреймворк, який включає асинхронні шаблони та підходи до мікросервісів. Він підходить для проектів, які потребують високої продуктивності та масштабування, але має менш розвинену спільноту та ресурси порівняно з іншими фреймворками.

Зважаючи на переваги, що їх надає NestJS, включаючи його архітектуру і підтримку модулів та мікросервісних застосунків, цей фреймворк було обрано як основний для розробки серверної частини. Його можливість масштабування, а також активна спільнота та добре розроблена документація є ключовими чинниками, які вплинули на цей вибір.

Серверна частина розділена на модулі, які задопомогою методу зворотної залежності об’єднуються у застосунок [23]. Перевагами такого підходу є незалежність інтерфейсів від реалізації. Такий підхід неможливий з використанням лише самого JavaScript, він можливий з використанням TypeScript та менеджеру залежностей від фреймворку NestJs.

Кожен модуль складається зі таких компонентів (див. додаток А):

– ValueObject – найменший компонент. Використовується для валідації бізнес параметрів. Наприклад параметр тип товару є простим рядком з певними можливими значеннями, для перевірки валідності використовується клас Quantity як ValueObject; у додатку (див. додаток А);

– DTO – використовується для передачі даних між різними частинами системи. Основною метою DTO є забезпечення ефективного обміну даними, зменшення кількості викликів методів і скорочення витрат на серіалізацію та десеріалізацію даних (див. додаток А);

– Repository – відповідає за доступ до об'єктів доменної моделі з бази даних або іншого сховища. Репозиторії інкапсулюють логіку роботи з базою даних і забезпечують інтерфейс для отримання, збереження та видалення доменних об'єктів (див. додаток А);

– Service – це компонент, що містить бізнес-логіку, яка не може бути легко розміщена в доменних моделях або агрегатах. Сервіси взаємодіяють з репозиторіями або іншими сервісами (див. додаток А);

– Controller – отримує запити від клієнтів серверу, їх обробку та повернення відповідей. Контролери зазвичай знаходяться на рівні представлення (Presentation Layer) і взаємодіють із сервісами для виконання бізнес-логіки. (див. додаток А).

Серверна частина має такі модулі: авторизація, корзина, замовлення, продукти, фінансовий калькулятор, користувач. Далі буде розібрано основі моменти реалізації цих модулів.

Авторизацію реалізовано з використанням окремо створеного застосунку з використанням сервісу Supertokens [24], за для отримання більшої гнучкості в масштабуванні додатку та запобіганню надлишкової витрати обмежених ресурсів. Застосунок являє собою сервіс налаштований в docker compose який містить в собі власну логіку авторизації та верифікації користувачів. Архітектура SuperTokens включає веб SDK, яке автоматично оброблює аутентифікації та керує токенами сесій, та серверне SDK, яке додає API аутентифікації до серверної частини, і SuperTokens Core Service – HTTP-сервіс, що спілкується з базою даних і містить основну логіку аутентифікації. Така архітектура забезпечує безпеку та спрощує керування аутентифікацією у додатках.

Налаштування Supertokens для клієнт-серверного програмного забезпечення описано у додатку А. Потрібно підключити Supertokens до бази даних PostgresQl, це може бути як основна база так і спеціалізована база даних. Для даного ПЗ було обрано підключення до основної бази даних.

Підключення самого серверного застосунку до Supertokens виконується шляхом використання бібліотеки supertokens-node. Задопомогою вбудованих функцій можливо швидко підключити сторонні сервіси авторизації, такі як Google у даному проекті.

Коректна обробка помилок та логування забезпечує розуміння стану серверу: завантаженість та проблемні місця. Помилки в звичайному їх розумінні важко оброблювати та є не бажані при розробці з JavaScript: викидаючи помилки з методів потрібно постійно ставити обробники помилок, та код має їх коректно опрацьовувати, що практично неможливо при викиданні неструктурованих помилок. Патерн ExceptionFilter [25] структурує помилки перед відправкою на клієнт або логуванням. Клас помилки SeveralProductsWithTheSameNameError форматує помилку для подальшого відображення на боці клієнта (див. додаток А).

Для логування та моніторингу стану серверу використовується Grafana для відображення даних, Loki для збору логів та Prometheus для збору метрик [26]. Підключення Loki реалізоване за допомогою створення нового транспорту для WinstonLogger. Логування відбувається за допомогою сервісу LoggerService (див. додаток Б). Логування стає більш ефективним, коли є можливість відслідкувати шлях виклику коду від моменту запиту. Реалізація відслідковування складається з 2 елементів: створення traceId для кожного запиту та можливості фільтрування логи за певним текстом. Параметр traceId зберігається для кожного запиту в AsyncLocalStorage в NodeJs (див. додаток А).

Відображення логів та метрик відбуваються у Grafana (код налаштування Grafana див. додаток А). Grafana запущено на окремому сервері для отримання доступу до даних під час максимального навантаження головного серверу. Grafana це сервіс для відображення даних за допомогою спеціальних мов запитів, кожен сервіс для збору даних має власну мову запитів. Для збору даних у даному випадку обрано Prometheus з мовою PromQl та Loggy з мовою LogQl.

Розглянемо приклад запиту PromQl для моніторингу завантаження процесора, зокрема системних процесів (kernel space) [27], у відсотках:

```
irate(process_cpu_user_seconds_total{instance=~"$instance"}[15s]) * 100.
```

Компоненти запиту:

– process_cpu_system_seconds_total – метрика відображає загальний час, витрачений на системні процеси з моменту запуску системи. Вона збирається системою Prometheus;

– instance=~"$instance" – фільтр, що обмежує метрику конкретним екземпляром (інстансом). Символ ~ використовується для підтримки регулярних виразів, що дозволяє динамічно підставляти значення серверу з якого брати параметр;

– 15s – інтервал часу визначає період, за який обчислюється миттєва швидкість зміни метрики. У даному випадку, 15s означає 15 секунд. Використання невеликого інтервалу часу дозволяє отримати більш точну оцінку поточного навантаження;

– irate() – обчислює миттєву швидкість зміни значень лічильника за вказаний інтервал часу. Це дозволяє отримати значення швидкості зміни метрики за останні 15 секунд, що є критичним для оперативного моніторингу;

– 100 – результат обчислення миттєвої швидкості зміни множиться на 100 для перетворення значення з часток у секунду на відсотки. Параметр process_cpu_system_seconds_total надає значення в секундах, тому для отримання відсоткового навантаження процесора необхідно виконати це перетворення.

Діаграми метрик серверу зображені на рисунку 3.1, мають показувати дані про поточний стан серверу та мати можливість фільтрувати за часом. З важливих параметрів мають відображатися: навантаження на сервер у відсотках, затримка подієвого циклу (NodeJs EventLoop) у мілісекундах, додаткові діаграми. Метрика затримки подієвого циклу [28] важлива тому, що якщо цикл подій блокується на значний час, це може призвести до того, що додаток стає нечутливим і починає повільно реагувати на запити користувачів. Затримка понад 100 мілісекунд може призвести до значного зниження продуктивності і збільшення часу обробки операцій.



Рисунок 3.1 – Діаграми завантаженості серверу

Для виведення логів потрібно створити список з можливостю фільтрувати за текстом для пошуку логів за traceId та за рівнем логу. Запит для показу та фільтрації логів наступний:

```
{job="$app"} |= `$search` | logfmt | level =~ `$level`
```

Компоненти цього запиту:

– job – вибирає логи, які мають мітку job, що відповідає значенню змінної $app. Параметр $app підставляється на головній сторінці дошки;

– $search – цей блок реалізує пошук за текстом. Оператор |= шукає логи, які містять вказаний текст. $search – змінна, що підставляється на головній сторінці дошки;

– level – фільтрує за рівнем логів. Оператор =~ використовується для фільтрації логів за значенням поля level, яке відповідає регулярному виразу. $level підставляється на головній дошці. Дозволяє фільтрувати логи за рівнями.

На рисунку 3.2 зображена дошка з логами та можливістю фільтрування. Є можливість фільтрувати за рівнем помилки та за текстом в помилки.

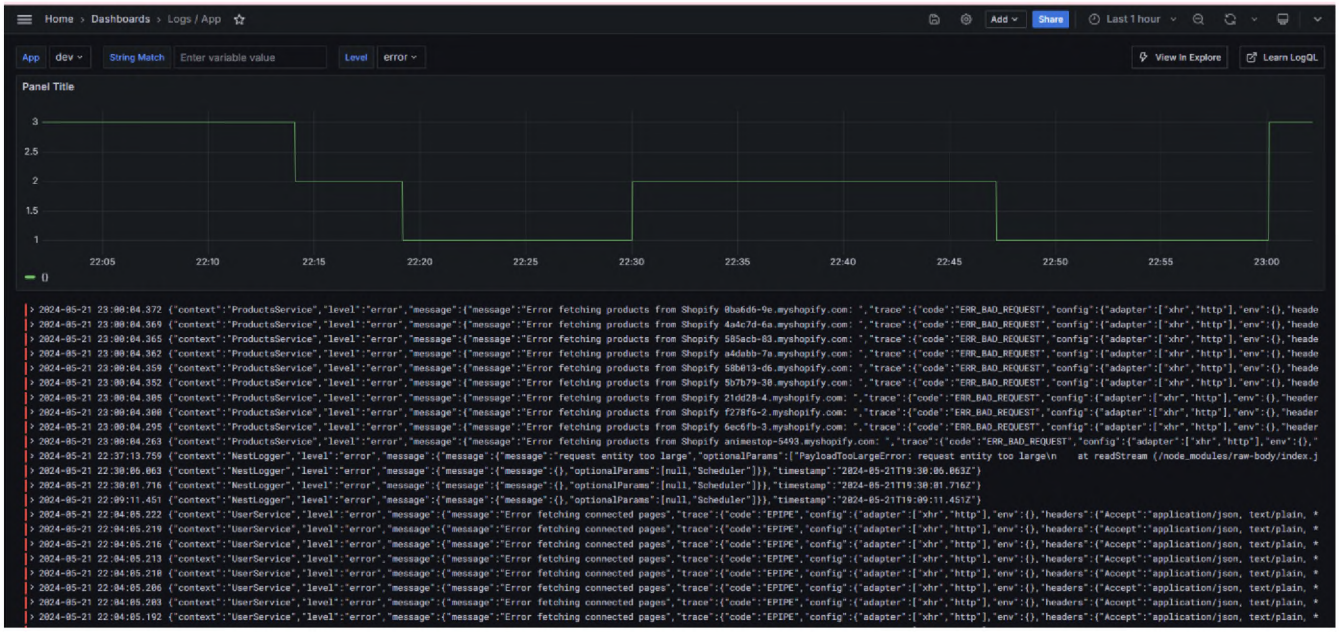


Рисунок 3.2 – Дошка з фільтруванням логів за рівнем та текстом

Розроблений функціонал надає можливість розуміти стан серверу та впроваджувати необхідні дії. Аналізуючи логи з рисунку 3.2 стає зрозуміло що є

певні проблеми з Facebook інтеграцією. Додатково можна налаштувати сповіщення на пошту чи месенджер про певні події на сервері.

Обробляючи велику кількість даних потрібно брати до уваги розподільності даних та оптимізацію виконання коду. При розробці функціоналу який буде часто використовуватися та не потребує миттєвої повної відповіді. Функціонал додавання зображень до товару може бути оптимізованим з використанням черг.

Черги реалізовані задопомогою Redis Bull та партерну команд. Для звернення до черги використовується enum ImagesProcess.

Параметр name – назва черги, параметр addImages – є назвою обробника. Для обробки черги потрібно створити обробник черги. Цей обробник черги використовується для додавання зображень до продуктів у системі. Він впроваджує сервіс ImagesService, який містить логіку додавання зображень, та має метод addImages, який отримує масив об'єктів із зображеннями (кодованими у форматі base64) та ідентифікаторами продуктів. ImagesQueue приймає завдання з черги та передає їх в ImagesService для виконання (дивю додаток А).

Для додавання зображення в чергу використовується клас команда AddImagesCommand. Завдяки цьому підходу дана команда може бути викликана з будь-якого сервісу де це необхідно (див. додаток А).

Розглянемо реєстрацію черги. Імпортуючи модулі ImagesModule та ProductModuel надається доступ до сервісів ProductService та ImagesModule. В BullModule задаються параметри черги. Параметр limiter обмежує кількість прийнятих задач до обробки за 1000 мілісекунд. Параметр max обмежує максимальну кількість виконуваних задач. Це значення 4 бо обробка зображення є не важливим в плані терміновості функціонал.

Після цих налаштувань обробка зображень буде розподілена в часі та навіть якщо працівники будуть додавати 100 зображень за раз сервер витримає це навантаження.

Кошик є важливим елементом інтернет-супермаркетом. Поставлені вимоги потребують максимально швидкого та коректної обробки запитів від

споживачів. З боку серверної частини ці вимоги будуть виконанні реалізацією гранульованих API[29]. Розглянемо API запити для маніпуляції з корзинами:

– PUT /product – оновлення продукту в кошику;

– GET /products – передає всю інформацію про товари у кошику: назву, кількість одиниць, суму для кожного продукту, ідентифікатор товару;

– GET /minimal-info – отримання мінімальної інформації про кошик. А саме суму кожного замовленого товару, та ідентифікатор товару в системі. Це необхідно для відображення суми вже обраних продуктів у списку продуктів;

– GET /sum – відправляє лише суму поточного кошика.

Кожен API запит використовується для виконання лише свого функціоналу. Бо екран який відображає лише суму кошику не потребує інформацію про всю інформацію про кошик, це зменшить час на всіх етапах обробки запиту: зменшивши складність SQL запиту до бази даних.

3.3 Розробка клієнтської частини

При розробці архітектури було закладено два окремих веб-застосунки для максимальної реалізації вимог. Розробляючи застосунки окремо отримуємо такі переваги:

– гнучкість у вибору технологій;

– гнучкість підходів розробки та оптимізації: адміністраційна частина не має вимог, бути такаю ж оптимізованою, як клієнтська частина, однак має бути більш захищеною. Однак попри гнучкість у виборі технологій, вони мають бути максимально подібними. Головні технології використані для розробки споживацького та менеджерського веб застосунку;

– Typescript – строго типізована надмножина JavaScript, яка компілюється у звичайний JavaScript [30]. Використовується для покращення досвіду розробки та зменшення кількості помилок;

– React – JavaScript бібліотека для побудови користувацьких інтерфейсів, створена компанією Facebook. Основною метою React є спрощення розробки складних інтерфейсів за допомогою компоненто-орієнтованого підходу;

– менеджери API запитів – інструменти, що спрощують взаємодію з сервером, управління даними та кешування у веб-додатках. Вони забезпечують автоматичне кешування, синхронізацію стану компонентів з серверними даними, обробку помилок і повторні спроби, оптимістичне оновлення інтерфейсу, а також підтримку паралельних і залежних запитів. У менеджерському веб-застоснку використовується RTK Query, У споживатьсякому веб-застосунку використовується React query. Вони виконують однакову функцію, але з різними підходами. RTK Query забезпечує безшовну інтеграцію з Redux що надає ефективний спосіб обробки даних на боці веб-застосунки для відображення, відправки або отримання даних. React Query надає більш ефективний та простий спосіб у керуванні керуванням API запитів, зручний у використанні при відсутності складної обробки даних;

– SCSS – це синтаксичний діалект мови стилів Sass (Syntactically Awesome Stylesheets), який розширює можливості стандартного CSS, надаючи інструменти для написання більш організованих, зручних і багаторазових стилів. SCSS підтримує вкладеність, що дозволяє структуровано описувати стилі. Міксіни (mixins) у SCSS дають можливість створювати групи стилів, які можна багаторазово використовувати в різних частинах коду, що підвищує його модульність і повторне використання. Крім того, SCSS підтримує вкладені медіа-запити, що полегшує створення адаптивного дизайну, та функції для виконання обчислень і маніпуляцій зі значеннями.

Для кращого задоволення архітектурних вимог можна було б обрати Angular [31] для розробки менеджерського веб застосунку: Angular завдяки своїй повноцінній інтеграції з Typescript та модульності краще задовольняє потребу у передбачуваності поведінки коду. Однак таке ускладнення збільшує кількість технічних знань. Обрані технології створюють однакову базу для розробки обох застосунків: що прискорить розробку.

Архітектура веб-застосунків має бути простою, тож обрано осучаснену версію партерну Container-Components [32] назва Container-View, з використанням нових можливостей React API.

Головні аспекти розробку адміністративної частини клієнт-серверного програмного забезпечення є швидкість розробки та якість готового веб-застосунку. Ці вимоги задоволені максимальним використанням бібліотек для реалізації функціоналу. Цей підхід є оптимальним, бо не потрібно писати код який вже створений і зазвичай логіка закладена в бібліотеках є високоякісною; з недоліків: збільшення розміру веб-застосунку. Збільшений розмір є не проблемою, бо це внутрішній веб-застосунок, відповідно до вимог час завантаження не є пріоритетним.

Головні використані бібліотеки:

– MUI – бібліотека компонентів для React, яка реалізує дизайн-систему Material Design. Прискорює розробку компонентів: містить основні компоненти, кнопки, поля вводу, списки, тощо. Дозволяє легко модифікувати стилі використовуючи варіанти для компонентів. Значно збільшує розмір компільованого додатку;

– @tanstack/react-table – бібліотека для створення високопродуктивних та гнучких таблиць у React-додатках. У поєднанні з віртуалізацією надає можливість відображати сотні рядків;

Розглянемо обробку додавання нових продуктів до системи. Продукти додаються у вигляді структурованого excel файлу. Веб-застосунок не робить валідацію структуру файлу. Вимоги що до обробки excel файлу з продуктами:

1. Можливість обрати файл з локального диску пристрою.

2. Відправити на сервер.

3. Відобразити результат додавання продуктів: помилку або успішне додавання на екрані.

Почнемо розбір з обробки API запиту. Функція для обробки API запиту useUploadProductsMutation створюється за допомоги об’єкту baseApi та функції injectEndpoints: в рамках цієї функції реалізуються яким чином оброблювати дані

перед відправкою та API адреса для відправки. Об'єкт baseApi містить в собі базові налаштування: базову адресу серверу та необхідні заголовки.

Присвоєння інтерфейсу вхідних та вихідних даних. Оскільки вихідні дані можуть бути відповідь від серверу з пустим значенням або з помилкою, то можна знехтувати типом вихідних даних. Вхідні дані задаються інтерфейсом IUploadProductsFileRequest, який містить в собі необхідний файл до відправки.

Присвоєння параметрів для відправки API запиту. У параметрі invalidatesTags задаються ключі інших API запитів кеш яких має бути оновлений після виклику uploadProducts запиту.

Об'єкт productApi надає функцію useUploadProductsMutation, хук (hook) [33], яка є обробником API запиту: містить різні стани запиту, надає можливість викликати запит або відмінити.

Функція useUploadProductsMutation використовується в хукові useUploadProducts, цей хук використовується в ролі сервісу та містить необхідну логіку для завантаження файлу з продуктами на сервер та відображення процесу обробки запиту.

Функція useUploadProductsMutation повертає кортеж. Перший елемент кортежу функція uploadProducts для виклику API запиту. Другий елемент кортежу об'єкт зі станами виконання запиту (див додаток А).

Таким чином логіка обробки файлу з продуктами є неприв'язана до жодного компоненту. Інтерфейс інтегрується з логікою обробки відправки продуктів у компоненті React UploadProductsForm. В патерні Container-View цей компонент є контейнером, може ініціалізувати різні хуки та комбінувати їх для передачі параметрів в компонент View (див. додаток А).

Саме UploadFileForm реалізує користувацький інтерфейс для відправки будь-яких файлів, оскільки всі параметри задаються зовні (див. додаток А).

Таким чином побудований кожен функціонал в менеджерському та споживацькому веб-застосунку. Кожен компонент може використовувати інші компоненти. Таким чином збирається весь застосунок.

В попередньому розділі був розглянутий патерн розробки Container-view, цей підхід використовується і в споживатьсякому застосункові. Відмінності споживацького застосунку в оптимізації доставки застосунку до споживачів. Це досягається завдяки не використанню складних візуальних бібліотек та грануляцією функціоналу для відправки на веб-клієнт. Кожна сторінка та певні елементи сторінки підготовлюються на боці серверу перед відправкою на веб-клієнт.

Підготовка сторінок до відправки втілена за допомоги фреймворку NextJs, надбудова над React: відповідає за SSR. Задопомогою спеціальних інструкцій React компоненти оброблюються в HTML на сервері, що дозволяє використовувати DNS кешування [34]. Розмір скомпільованого менеджерського веб-застосунку в JavaScript та CSS файли 360.6 кілобайт; Розмір головної сторінки споживацького веб-застосунку 182 кілобайти JavaScript коду та 13.6 кілобайти HTML і два файли CSS 3 та 6.3 кілобайти. Такий підхід дозволяє:

– при першому відкритті, відображати головну сторінку зі списком продуктів за 300мс при швидкості інтернету в 10,240 кб/с рис. 3.3;

– кешувати CSS та HTML файли на боці споживача, що ще більше пришвидшить відкриття сторінки при повторному відвідуванні.

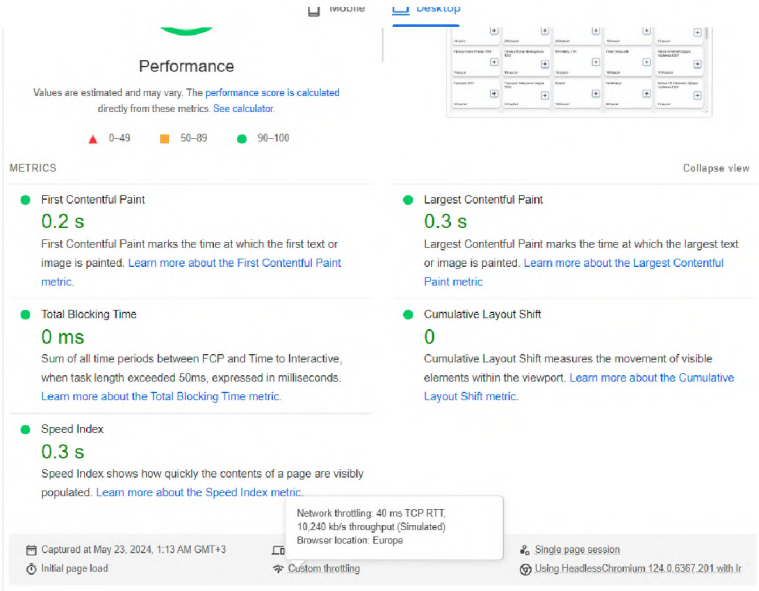


Рисунок 3.3 – Швидкість завантаження споживацького веб-застосунку

У функціональних вимогах прописано оптимізованість до пошукових двигунів. Для досягнення цієї вимоги окрім оптимізованої доставки сторінок до веб браузеру потрібно використовувати спеціальну розмітку Schema.org [22]. Відповідно до тесту PageSpeed Insights, результати зображено на рисунку 3.4, було досягнуто 100% балу. Звісно цей показник лише говорить про застосовані методи та про оптимізацію до пошукових двигунів. Максимальний бал не гарантує вищі позиції при пошуку веб-сайту.

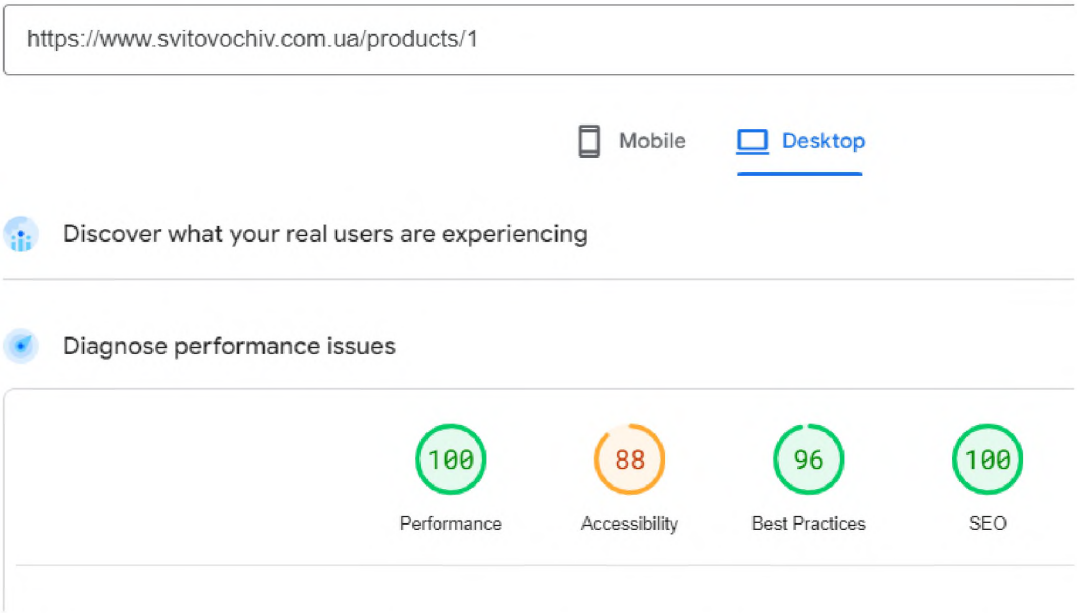


Рисунок 3.4 – Результати оцінки SEO оптимізації споживацького веб-застосунку

Розглянемо на прикладі основні принципи та техніки застосовані при розробці. Основний патерн як і в менеджерському веб-застосунку є Container-View. Головна відмінність між застосунками це методи компіляції та підготовка файлів до відправки.

Як написано вище споживацький застосунок використовує SSR для відправки веб-застосунку до браузеру. SSR накладає певні обмеження та правила до написання коду. Компоненти які рендяряться на сервері на можуть містити якогось стану всередині себе: тобто заборонено використовувати React хуки для

збереження стану. Для обробки стану можуть використовуватися функції які імпортуються у файл, стан може передаватися через параметри компоненту. У коді нижче компонент сторінки продукту приймає параметрами інтерфейс ProductsPageParamsProps. Який містить в собі параметри з пошукової стрічки: pageNumber – номер сторінки, withName – текст шуканого продукту.

Функція getProducts є асинхронною та звертається до серверу для отримання продуктів. Таким чином веб-браузер споживача одразу отримає сторінку зі списком продуктів без додаткових запитів (див. додаток А).

Розміщення тегів Shema.org відбувається за певними правилами. Розглянемо принципи розміщення тегів на прикладі продуктової карточки у додатку А.

– itemScope – використовується для оголошення нового елемента Schema.org. Він вказує, що даний HTML-елемент містить семантичну розмітку, яка представляє один об'єкт або сутність;

– itemType – використовується разом з itemScope для визначення типу об'єкта, який описується. Значенням цього атрибута є URL-адреса типу Schema.org;

– itemProp – використовується для позначення властивостей об'єкта, описаного елементом з атрибутом itemScope.

З використанням вище продемонстрованих підходів, створена кожна сторінка та елемент споживацького веб-застосунку.

Тестування ПЗ поділяється на різні типи та має різні методи. Для даного проекту обрано тестування білого та чорного ящика, unit тестування [35].

Unit тестування – це метод тестування, який передбачає перевірку окремих модулів або компонентів програмного забезпечення на рівні коду. У поєднанні з підходом розробки TDD [36] (Test driven development) дозволяє виявляти та виправляти помилки на ранніх етапах розробки, що значно знижує витрати на їх усунення на пізніших етапах життєвого циклу ПЗ. У контексті даного проекту,

unit тестування було використано для перевірки ключових функцій серверної частини ПЗ.

Головні інструменти для розробки unit тестів серверної частини є фреймворк для написання unit тестів Jest. Розберемо приклад тесту для сервісу корзини у додатку А наведено код. Перший етап Unit тестування це визначення найкритичніших і в водночас найменших елементів для тестування. Потім підготовка даних до тестування, не рекомендується використовувати дані з бази даних або не випадкові дані при проведені unit тестуванні. При використанні jest методи отримання даних можуть бути замінені за допомогою функції обгортки spyOn, таким чином можна створити не прив'язаний до бази даних тест.

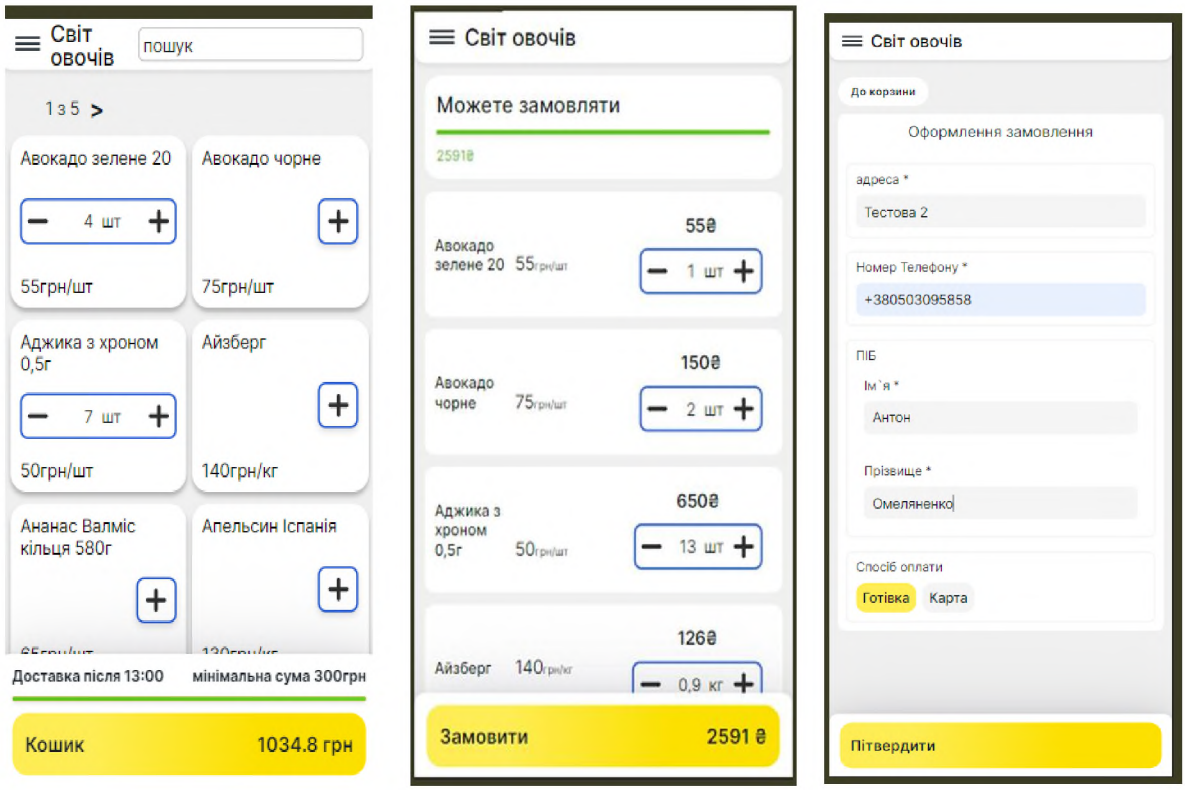
Для демонстрації тесту змінено логіку методу getOrderedProductsSum на неправильну. Очікувано що тест не пройде та отримаємо інформацію про те який тест не пройшов та чому, на рисунку 3.5 зображено результат запуску тестів.

```
Test Suites: 2 failed, 2 passed, 4 total
Tests:       2 failed, 3 passed, 5 total
Snapshots:   0 total
Time:        9.719 s
Ran all test suites.
```

Рисунок 3.5 – Результат запуску тестів з некоректної логікою в сервісі BasketService метод getOrderedProductsSum

При обмеженій кількості ресурсів та швидкозмінності ПЗ на ранніх стадіях unit тести варто створювати для найкритичніших компонентів системи. Також варто зауважити, що розробка подібних тестів є не тільки дорогою, а й саме використання має бути обережним. Запуск 100 тестів на віддаленій машині може призвести до надмірних витрат. Тож обрано запуск тестів лише при об'єднанні нових функціональних гілок до головної гілки.

Перший етап взаємодії споживача з ПЗ це відкриття головної сторінки вебзастосунку (рис.3.6, а). На головній сторінці застосунку споживач може переглянути наявні товари, шукати їх, додати до корзини.



а) б) в)

Рисунок 3.6 – Інтерфейс вебзастосуноку: а) головна сторінка, б) сторінка корзини, в) сторінка підтвердження замовлення

Обравши необхідний товар, споживач може перейти до корзини та зробити замовлення (рис. 3.6, б). Сторінка корзини перевикористовує елементи карточок товарів, містить інформацію про обраний товар та суму замовлення.

Натиснувши кнопку замовлення споживач переходить на сторінку підтвердження рис. 3.6, в. на якій заповнює інформацію необхідну для доставки товару.

Після підтвердження замовлення корзина споживача очищується обрані товари переходять у замовлення. Споживач бачить повідомлення про успішне прийняття замовлення в обробку рис 3.7.

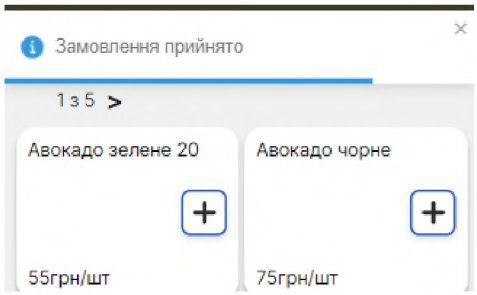


Рисунок 3.7 – Повідомлення про успішне прийняття замовлення

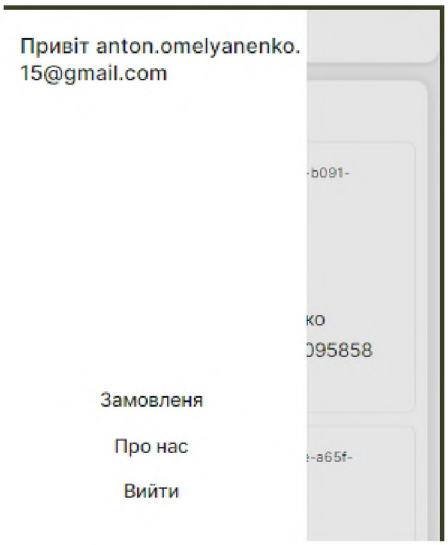


Рисунок 3.8 – Бокове меню споживацького вебзастосунку

У правому боковому меню споживач бачить навігаційні кнопки для переходу на сторінки: замовлення, про нас та кнопку виходу з акаунту. У верхній частині бокового меню розміщене привітання до споживача.

3.4 Вартість розробки

Вартість роботи різних спеціалістів взято з веб-порталу Djinni [38], який надає інформацію про середню заробітну плату спеціалістів в Україні. Використання даних з Djinni дозволяє спиратися на актуальну та перевірену інформацію, що відображає ринкові реалії. Таблиця 3.1 показує середню вартість години спеціалістів різного рівня.

Спеціаліст початкового рівня – здатний самостійно виконати більшість задач на проекті. Досвід роботи від року до двох років.

Спеціаліст середнього рівня – здатний самостійно виконати всі задачі на проектів. Досвід від двох до чотирьох років.

Спеціаліст високого рівня – здатний високоякісно виконати всі поставлені задачі та закласти масштабовану архітектуру. Досвід від 4ьох років.

Таблиця 3.1 – Вартість години спеціалістів різного рівня

Спеціальність	Вартість за годину відповідно до рівня досвіду, грн		
	Початковий	Середній	Професійний
Javascript Full-stack програміст	170	450	915
Дизайнер	137	295	550
Бізнес аналітик	268	460	660

Розглянемо мінімальну можливу кількість годин, необхідних для розробки, з урахуванням залучення спеціалістів різного рівня. Для простішого розрахунку вартості встановимо умовність, що спеціалісти різного рівня виконають завдання за однаковий час, звісно з різною якістю. Це дозволить стандартизувати процес розрахунку та уникнути зайвих складнощів, пов'язаних з оцінкою продуктивності спеціалістів різного досвіду та часових витрат на завдання. Такий підхід також сприяє більш прозорому та зрозумілому визначенню бюджету проекту. Обираючи спеціалістів для MVP (Minimum viable product) розробки даного варто звернути увагу на рекомендовану вартість спеціаліста, особливо при розробці програмного забезпечення, за для уникнення складнощів у подальшому розвитку. Також можливо розглянути мінімальну вартість розробки при частковому залучені спеціаліста високого рівня, який буде курувати основні аспекти розробки, архітектури та перевіряти код. При наявності зацікавленого високо кваліфікованого спеціаліста цей підхід може бути оптимальним. За для уникнення найму додаткового спеціаліста, тестування та бізнес-аналіз може відбуватися замовником та розробниками проекту.

Таблиця 3.2 – Аналіз витрат на розробку з урахуванням мінімальної кількості годин.

Спеціальність	Складова розробки	Витрачено годин	Вартість виконаного завдання спеціалістом відповідно до ціни та вартості, грн			Рекомендована вартість, грн
			Початківець	Середній	Професіонал	
Javascript Full-stack програміст	Розробка архітектури	25	4250	11250	22875	22875
Дизайнер	Розробка дизайну вебзастосунку	15	2055	4425	8250	3000
Бізнес аналітик	Функціональний аналіз	30	8040	13800	19800	10500
Javascript Full-stack програміст	Розробка простих елементів веб-застосунку	30	5100	13500	27450	5100
Javascript Full-stack програміст	Розробка складних елементів веб-застосунку	35	5950	15750	32025	16100
Javascript Full-stack програміст	Розробка простих елементів серверної частини	30	5100	13500	27450	12000
Javascript Full-stack програміст	Розробка складних елементів серверної частини	60	10200	27000	54900	54900
Сума		225	40695	99225	192750	124475

Рекомендована вартість розробки є 125 тисяч гривень.

ВИСНОВКИ

Підчас виконання кваліфікаційної роботи було проведено огляд ринку інтернет-супермаркетів, огляд клієнт-серверних архітектур та технологій, задопомогою яких вони будуються. Було проаналізовано хмарні провайдери та способи заощадження коштів, при використанні їх послуг.

Було закладено функціональні вимоги до клієнт-серверного програмного забезпечення та закладено архітектуру для подальшої розробки програм. Перед розробкою архітектури, було проведено дослідження доменної бізнес логіки. Для вибору архітектури був проведений аналіз можливостей з точки зору безпеки, масштабування, складності реалізації.

Використовуючи архітектурні та функціональні вимоги було обрано технології для розробки клієнт-серверного програмного забезпечення для підтримки діяльності інтернет-супермаркету. Коротко оглянуто основні аспекти розробки та створення допоміжних сервісів, таких як Grafana. Надано використання приклад використання автоматизованих тестів ПЗ їх переваги для даного проекту. Та продемонстровано робочій варіант інтернет-супермаркету. Проведено розрахунок вартості виконаних робіт.