МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
"_____"_____ 2021 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Технологія розробки вебдодатків на базі фреймворку *Vue.js*

Виконавець: _____Машталер Б.В.

Керівник: _____ доц. к.ф.м.-н., Кучерява О.М.

Нормоконтролер: _____Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Литвиненко О.Є.
“_____”_____ 2021 р.

З А В Д А Н Н Я
на виконання дипломної роботи (проекту)

Машталера Богдана Васильовича
(прізвище, ім’я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проекту): Технологія розробки вебдодатків на базі фреймворку *Vue.js*

затверджена наказом ректора від "04" лютого 2021 року № 135/ст.

2. Термін виконання роботи (проекту): з 17.05.2021 до 20.06.2021

3. Вихідні дані до роботи (проекту): мова програмування *JavaScript*, фреймоворк *JavaScript – Vue*, мова розмітки гіпертексту – *HTML*, динамічна мова стилів – *LESS*, база даних *FireBase*, середовище розробки *Visual Studio Code*

4. Зміст пояснювальної записки:
1) Автоматизація розробки веб додатків
2) Застосування веб-додатків в електронній комерції
3) Розробка інтернет магазину на базі фреймворку *Vue.js*

5. Перелік обов’язкового графічного (ілюстративного) матеріалу:

1) Структурна схема роботи *AJAX* запитів
2) Схема динамічного рендеру сторінки.
3) Блок схема роботи *SPA* додатків
4) Блок схема безсерверної архітектури
5) Структурна схема формату *JSON*

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі дипломного проектування.	17.05.2021 – 18.05.2021	
2	Вивчити спеціальну літературу і технічну документацію.	18.05.2021 – 20.05.2021	
3	Проаналізувати *e-commerce* сферу, тонкощі роботи із фреймворком *Vue*.	20.05.2021 – 21.05.2021	
4	Написати розділ 1.	20.05.2021 – 21.05.2021	
5	Провести проектування програмного забезпечення для створення *e-commerce* продукту.	21.05.2021 – 22.05.2021	
6	Написати розділ 2.	21.05.2021 – 22.05.2021	
7	Провести розробку веб-додатку.	23.05.2021 – 25.05.2021	
8	Написати розділ 3.	26.05.2021 – 28.05.2021	
9	Оформити пояснювальну записку.	28.05.2021 – 30.05.2021	
10	Підготувати графічний демонстраційний матеріал та доповідь.	08.06.2021 – 10.06.2021	

7. Дата видачі завдання: "17" травня 2021 р.

Керівник дипломної роботи (проекту) _____ Кучерява О.М.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Машталер Б.В.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Технологія розробки веб-додатків на базі фреймворку *Vue.js*»: 51 с., 12 рис., 14 літературних джерел.

ВЕБ-ДОДАТОК, МОВА ПРОГРАМУВАННЯ *JAVASCRIPT*, СЕРЕДОВИЩЕ РОЗРОБКИ *VISUAL STUDIO CODE*, ВЕБ-СЕРВЕР *FIREBASE*.

Об’єкт дипломного проектування – веб-додаток в сфері *e-commerce*.

Предмет дипломного проектування – розробка веб-додатку для інтернет користувачів.

Мета дипломного проекту – розробка веб-додатку інтернет магазину для взаємодії користувачів з інтернет магазином.

Методи проектування – методи сучасної розробки веб-додатків з фреймворками *JavaScript* зокрема *Vue*. Концепція *Serverless* архітектури, мова програмування *JavaScript*, середовище розробки *Visual Studio Code*, веб-сервер *Firebase*.

Результатом виконання дипломного проекту є розроблений додаток, який призначений для роботи інтернет магазину. Додаток забезпечує реальну взаємодію користувачів з магазином для звершення покупок.

ЗМІСТ

ВСТУП

Інтернет магазини все більше і більше завойовують популярність серед громадян. Перевага в основному в тому, що онлайн магазини торгують товарами через мережу, що не примушує людей фізично відкривати двері для пошуку недоступних раніше асортименту товару.

Всі покупки в онлайн магазинах можна здійснювати не виходячи з дому або ж це здійснювати при поїздці в транспорті або ж на відпочинок. Інтернет магазини стали невід'ємною частиною нашого сучасного життя.

Що ж потрібно для того щоб зробити покупку чого або в онлайн магазинах? Ну, по-перше, потрібно мати доступ до глобальної мережі інтернет. Таким чином, в браузері кожен покупець може подивитися на фото товару, що дасть можливість мати краще уявлення про продукт, потім прочитавши опис і порівнявши відгуки ви можете придбати цей товар за допомогою замовлення і доставки поштою. З оплатою теж все просто, ви можете розрахуватися як безготівковим платежем і електронними грошима, так і готівковим при отриманні товару в руки. Але яким чином працює інтернет магазини ?

Одним з популярних способів реалізувати інтернет магазин за допомогою *SPA*-додатку. Головною перевагою цього підходу є те, що завдяки динамічному оновленню за допомогою *JavaScript*, під час використання не потрібно перезавантажувати або довантажувати додаткові сторінки. На практиці це означає, що користувач бачить в браузері весь основний контент, а під час переміщення або переходах на інші сторінки, замість повного перезавантаження потрібні елементи просто завантажуються.

РОЗДІЛ 1
АВТОМАТИЗАЦІЯ РОЗРОБКИ ВЕБ-ДОДАТКІВ

1.1. Різниця між веб-сайтом та веб-додатком

Для звичайного користувача може здатися, що концепції веб-сайту та веб-програми повністю перекривають одна одну і можуть використовуватися як взаємозамінні без значного впливу. Плутанина випливає з того, що обидві концепції мають спільну мову і звучать дуже схоже одна на одну. У загадці веб-сайтів проти веб-додатків важливо пролити світло на ключові диференціатори та характеристики, які роблять кожну концепцію унікальною.

Веб-сайт – це колекція загальнодоступних, взаємопов'язаних веб-сторінок, що мають єдине доменне ім'я. Веб-сайти можуть створюватися та підтримуватися окремою особою, групою, бізнесом чи організацією для різноманітних цілей.

Разом усі загальнодоступні веб-сайти становлять Всесвітню павутину.

Зазначимо, що веб-сайт – це колекція вмісту, який публікується під одним доменним іменем та принаймні на одному веб-сервері. Веб-сайти порівняно просто визначити, оскільки більшість із нас стикалися з веб-сайтами, але коли справа доходить до визначення веб-сайту та веб-додатків, важко розрізнити їх. Це нормально, оскільки навіть деякі досвідчені програмісти борються з чітким визначенням того, що відрізняє ці концепції.

Класичні приклади веб-сайтів включають ділові веб-сторінки, урядові веб-сайти, блоги, освітні та новинні сайти.

Основними елементами веб-сайту є мова розмітки гіпертексту (*HTML*)[7], каскадні таблиці стилів (*CSS*) та *JavaScript*. Веб-сайти не потребують мов програмування чи баз даних.

Там, де існують веб-сайти для інформування, існують веб-програми для взаємодії.

Веб-програми слугують настільними програмами, які є динамічними, залежно від взаємодії користувача для досягнення своєї мети, чи то шляхом обміну вмістом, чи збору інформації.

Враховуючи свою природу, веб-додатки набагато більш чутливі до дій користувачів. Вони інтерактивні та надають користувачам можливість маніпулювати даними та запитувати різні результати.

Класичні приклади веб-додатків включають програми веб-пошти, веб-сайти роздрібної торгівлі, текстові процесори, офісні інструменти, відеоредактори тощо.

Основними елементами веб-програми є *HTML, CSS* та *JavaScript*.

Ми говорили про інтерактивність, оскільки це вирішальний фактор, який відрізняє веб-сайт від веб-програми. Цей ключовий диференціатор – це те, що принципово відрізняє веб-сайти та веб-програми. За допомогою веб-додатків користувачі можуть отримувати доступ до вмісту та маніпулювати ним. Як правило, взаємодія – це діалог, коли користувач виконує певну дію, наприклад, натискання на елемент, а сторінка надає відповідь, наприклад, завантаження документа.

Частина плутанини пов’язана з тим, що сучасні веб-сайти розвиваються щодня, і в результаті ви отримуєте більше натяків на інтерактивність. Але ми повинні пам’ятати, що справжній веб-сайт базується на інформаційному вмісті, який ви можете читати, переглядати або слухати, тоді як основна функціональність справжньої веб-програми ґрунтується виключно на високих рівнях взаємодії.

Веб-сайти та веб-програми поділяють подібність, яку обидві можна інтегрувати з іншим програмним забезпеченням. З огляду на це, набагато частіше спостерігається інтеграція з веб-програмами, оскільки їх функціональність є більш складною та надійною, тому часто потрібна зовнішня взаємодія сторонніх систем.

Як правило, інтеграції призначені для зміцнення системи та полегшення функціональних аспектів веб-програми, таких як збір даних, зберігання даних, обробка даних, аналіз даних тощо.

Наприклад, веб-сайтам рекомендується інтегруватися із Системою управління вмістом (*CMS*), щоб ви могли швидко та легко регулярно оновлювати інформацію та дані. Для веб-програми електронної комерції дуже часто можна побачити, що вона інтегрована із системою управління взаємовідносинами з клієнтами (*CRM*) для обробки даних про клієнтів, інформації про замовлення та покращення спроб продажів. Загалом інтеграції дозволяють веб-програмам та веб-сайтам забезпечувати чудовий рівень досвіду для кінцевих користувачів і можуть допомогти забезпечити більш персоналізований вміст.

Зрештою, і веб-сайти, і веб-програми можуть інтегруватися із зовнішнім програмним забезпеченням на основі Інтернету, але на відміну від веб-програм, де це частина їхньої основної функціональності, інтеграція для веб-сайтів є необов'язковою.

Аутентифікація – це ще одна важлива відмінність у веб-сайті від сфери веб-додатків. Як ви вже здогадалися, автентифікація не завжди потрібна для веб-сайтів, але з веб-програмами це інший випадок, оскільки досить стандартно вимагати облікові дані користувачів для доступу до веб-програми.

Наприклад, веб-програма соціальної мережі вимагає особистої інформації, щоб отримати доступ до даних, пов'язаних з унікальним профілем. Оскільки веб-програми мають вищий ступінь складності, рівні безпеки є більш надійними, ніж з інформативним веб-сайтом. Крім того, автентифіковані користувачі з певними дозволами та налаштуваннями мають доступ до налаштованих даних та інших способів взаємодії з сайтом.

Підводячи підсумок, як веб-сайти, так і веб-програми можуть вимагати автентифікації, але цей показник частіше зустрічається у веб-програмах, враховуючи ступінь їхньої інтерактивності, безпеки та складності.

1.2. Поняття *Single Page Application*

Single Page Application (*SPA*) – це одна сторінка (звідси і назва), де багато інформації залишається незмінним, і лише кілька частин потрібно оновлювати одночасно.[6]

Наприклад, під час перегляду електронної пошти ви помітите, що під час навігації не змінюється багато – бічна панель та заголовок залишаються незайманими, коли ви переглядаєте вхідні.

SPA відправляє лише те, що вам потрібно, при кожному натисканні, і ваш браузер надає цю інформацію. Це відрізняється від традиційного завантаження сторінки, коли сервер при кожному натисканні клавіші рендерить повну сторінку та надсилає її у ваш браузер.

Цей поштучний метод на стороні клієнта пришвидшує час завантаження для користувачів і робить обсяг інформації, яку повинен надсилати сервер, значно меншим та набагато економічним. Безпрограшний.

SPA – це веб-додаток або веб-сайт, який взаємодіє з користувачем шляхом динамічного переписування поточної сторінки, а не завантаження цілих нових сторінок із сервера.

Цей підхід призводить до скасування переривання користувальницької роботи між послідовними сторінками, змушуючи програму поводитися більше як настільна програма. На більшості веб-сайтів багато повторюваного вмісту.

Деякі з них залишаються незмінними незалежно від того, куди переходить користувач (колонтитули, колонтитули, логотипи, панель навігації тощо), деякі з них є постійними лише у певному розділі (смужки фільтрів, банери), і є багато повторюваних макетів та шаблонів (блоги, самообслуговування, налаштування пошти *google*, згадані вище).

Традиційні багатосторінкові веб-сайти відображають для вас всю сторінку на сервері та надсилають її у ваш браузер.

SPA – надає можливість не підвантжувати раз за разом повторювані елементи на вашій сторінці. Односторінковий додаток дозволяє змінювати зміст

сторінки не перезавантажуючи сторінки і не відправляючи запит на сервер отримати дані. Дана технологія називається *AJAX*.

AJAX (на англ. *Asynchronous Javascript and XML*) – технологія, що дозволяє взаємодіяти з сервером без перезавантаження сторінки. [2]

При використанні *AJAX* немає необхідності користувачу оновлювати всю сторінку а лише окрему частину. Таким чином час отримання даних і відображенням їх на сторінці значно зменшується.

Тобто *AJAX*:

– Має можливість створення зручного *Web*-інтерфейсу
– Активно взаємодіє з користувачем
– Має можливість часткової перезавантаження сторінки

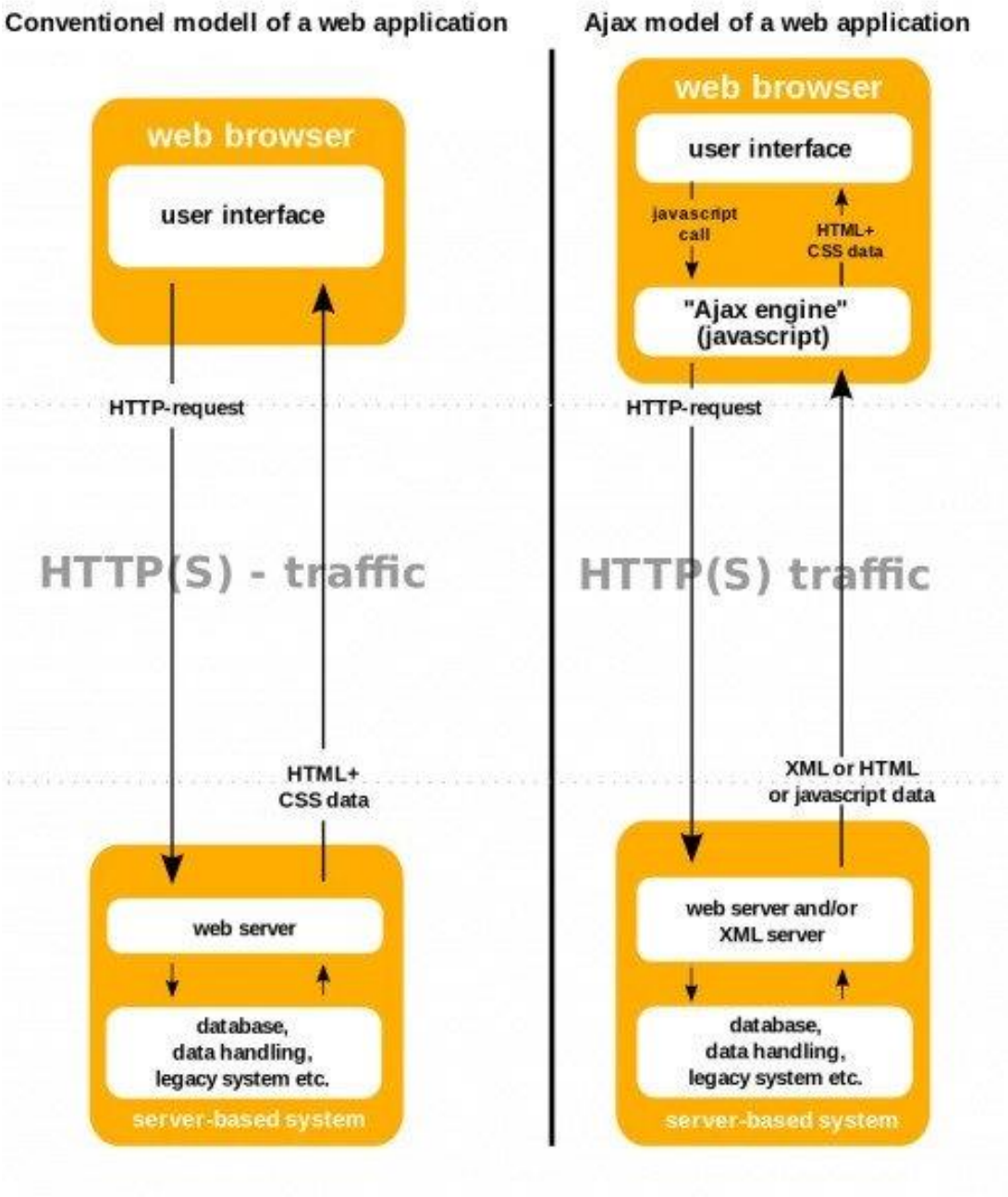

Рис. 1.1. Структурна схема роботи *AJAX* запитів

Рішення *SPA* мають багато переваг, таких як покращена продуктивність та послідовність програм, а також скорочення часу на розробку та витрат на інфраструктуру.

Відділяючи презентацію від вмісту та даних, команди розробників можуть працювати з різною швидкістю, в той же час інтегруючись для загального рішення. *SPA* добре підходить для адаптивного дизайну для мобільних пристроїв, настільних ПК та планшетів.

SPA після початкового завантаження сторінки (*HTML*, *CSS*, *JS*), сервер більше не надсилає вам *HTML* – ви завантажуєте його все на початку.

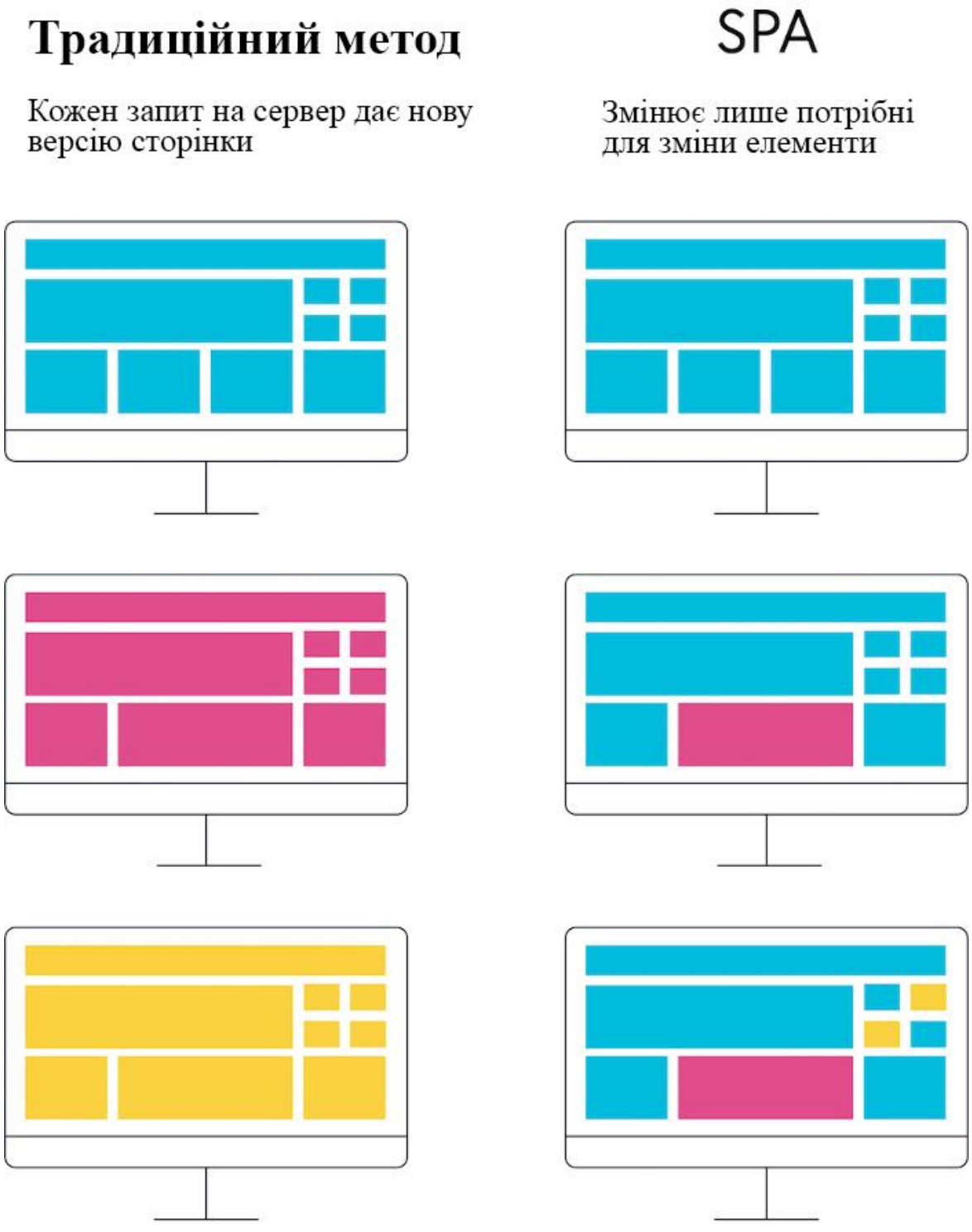

Рис. 1.2. Схема динамічного рендеру сторінки.

Сервер надсилає вам сторінку, а ваш браузер відображає інтерфейс користувача (*UI*).

Потім, коли ви натискаєте, *SPA* надсилає назад запити на дані та розмітку, сервер надсилає необхідну інформацію, а ваш браузер бере її та робить оновлений інтерфейс – обмінюються шматками, ніколи не потребуючи оновлення повної сторінки.

Ця швидка взаємозамінність робить *SPA* надзвичайно корисними на сторінках, які мають високу навігацію та використовують повторювані шаблони.

Наступною великою перевагою є те що серверу не потрібно витрачати час і енергію на повне малювання, *SPA*-центри загалом знижують вплив на ваші сервери – це означає, що ви можете заощадити гроші, використовуючи менше серверів для однакової кількості трафіку.

Разом із більш швидким часом роботи, описаним вище, *SPA* також дозволяють розробникам набагато швидше будувати інтерфейс.

Це пов'язано з роз'єднаною архітектурою *SPA* або розділенням серверних сервісів та інтерфейсу.

Багато важливих для бізнесу функціональних можливостей не так сильно змінюються на задньому плані.

Хоча те, як ваші клієнти входять, реєструються, купують та відстежують замовлення, час від часу може змінювати їх «вигляд» або презентацію, логіка та організація даних, що стоять за цим, досить постійна і ви не хочете ризикувати зіпсувати це.

Подібним чином, ваш необроблений вміст і дані можуть залишатися незмінними, але те, як ви хочете їх відображати, відрізняється.

Від’єднуючи цю внутрішню логіку та дані від того, як вони представлені, ви перетворюєте їх на «послугу», і розробники можуть створити безліч різних інтерфейсних способів показати та використовувати цю послугу.

Завдяки розв’язаній установці розробники можуть створювати, розгортати та експериментувати з інтерфейсом повністю незалежно від базової технології інтерфейсу.

Вони розробляють, як вони хочуть, щоб користувальницький досвід виглядав і відчував, а потім втягують вміст, дані та функціональність через ці служби.

Це робиться за допомогою *API*, які є стандартним набором правил між програмами щодо того, як вони структуруватимуть, обмінюватимуться та збиратимуть дані.

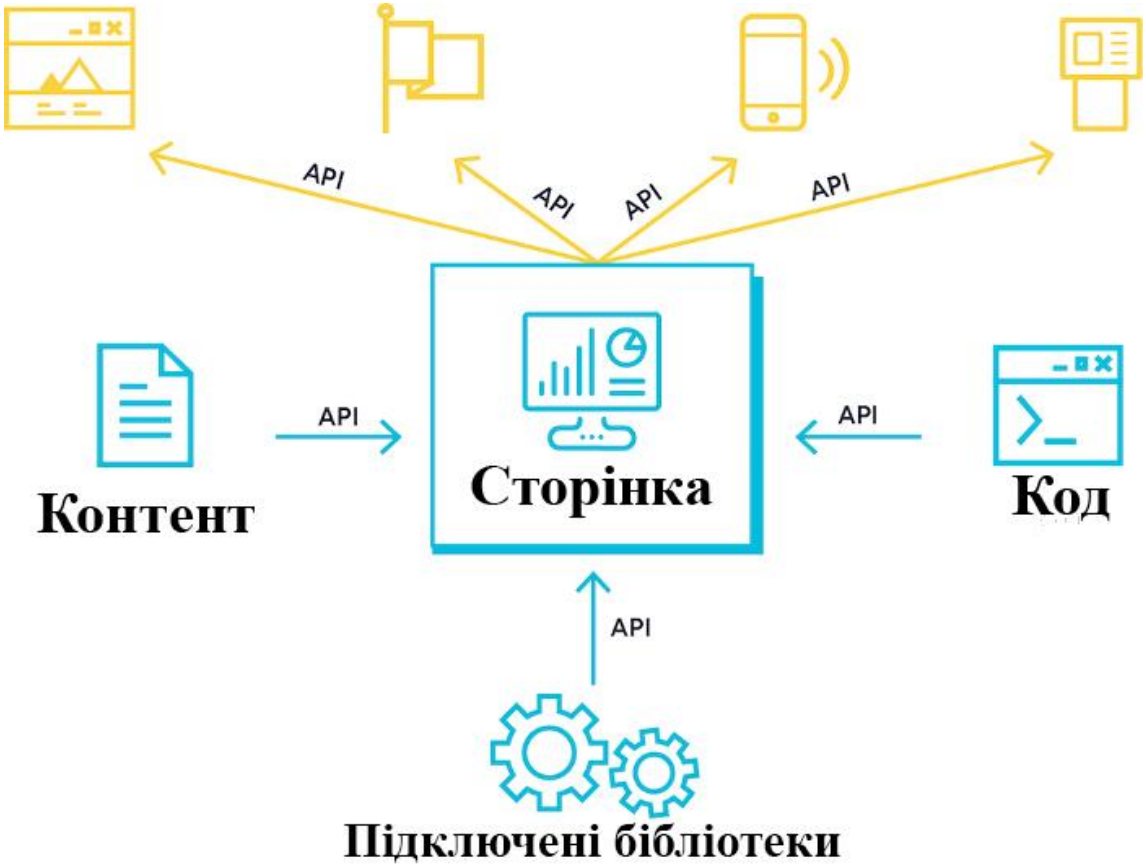

Рис. 1.6. Блок схема роботи *SPA* додатків.

Це налаштування *API* дозволяє розробникам швидко працювати над інтерфейсом користувача, не ризикуючи для критично важливих для бізнесу технологій.

Оскільки все більше і більше функціональних можливостей будуються як модульні сервіси (архітектура мікросервісу), які можна оновлювати самостійно, стає простіше експериментувати з тим, як вони відображаються та використовуються.

1.3. *UI* та *UX* в розробці веб-додатків

UI означає інтерфейс користувача в дизайні інтерфейсу означає «користувальницький інтерфейс». Інтерфейс користувача – це графічний макет програми. Він складається з кнопок, на які натискають користувачі, тексту, який вони читають, зображень, повзунків, полів для введення тексту та всіх інших елементів, з якими взаємодіє користувач. Це включає макет екрана, переходи, анімацію інтерфейсу та кожну окрему мікро-взаємодію. Будь-який візуальний елемент, взаємодія чи анімація повинні бути розроблені.[1]

UX означає взаємодія з користувачем. Взаємодія користувача з додатком визначається тим, як вони з ним взаємодіють. Це досвід гладкий та інтуїтивний чи незграбний та заплутаний? Взаємодія з користувачем визначається тим, наскільки легко або складно взаємодіяти з елементами інтерфейсу користувача, створеними дизайнерами інтерфейсу.

Ці два поняття дуже важливі при розробці будь якого додатку. Тому що від цього залежить дуже багато – починаючи від першого враження користувача до становлення користувача вашим «лідом»[11]

Який би не був функціональний ваш сайт, користувачі будуть надвати перевагу веб-вебсторінкам якими: по-перше, зручно користуватися; по-друге, їм

приємно користуватись.

1.4. *CRUD* в розробці веб-додатку

CRUD – це абревіатура, що стосується чотирьох функцій, які вважаються необхідними для реалізації постійної програми зберігання: створення, читання, оновлення та видалення. Оскільки кожен повноційний додаток працює з данними, який отримує шляхами: від бази данних, від серверу або через деяке *API*.

База даних складається з таблиць з рядками та стовпцями. У базі даних кожен рядок таблиці відомий як кортеж або запис. Кожен стовпець таблиці представляє певний атрибут або поле. Чотири функції *CRUD* можуть бути викликані користувачами для виконання різних типів операцій з вибраними

даними в базі даних. Це може бути здійснено за допомогою коду або за допомогою графічного інтерфейсу користувача. Давайте розглянемо кожен із чотирьох компонентів.

Функція *create* дозволяє користувачам створювати новий запис у базі даних. Пам'ятайте, що запис – це рядок, а стовпці називаються атрибутами. Користувач може створити новий рядок і заповнити його даними, що відповідають кожному атрибуту, але лише адміністратор може додати нові атрибути до самої таблиці.

Функція *read* дозволяє користувачам шукати та отримувати певні записи в таблиці та читати їх значення. Користувачі можуть знайти бажані записи, використовуючи ключові слова, або фільтруючи дані на основі індивідуальних критеріїв.

Функція *update* дозволяє користувачам модифікувати існуючі записи, що існують у базі даних. Щоб повністю змінити запис, користувачам, можливо, доведеться змінити інформацію в декількох полях.

Функція *delete* дозволяє користувачам видаляти записи з бази даних, яка більше не потрібна. Деякі баз даних можуть дозволити користувачам виконувати як жорстке, так і м’яке видалення. Жорстке видалення назавжди видаляє записи з бази даних, тоді як м’яке видалення може просто оновити статус рядка, щоб вказати, що воно було видалено, залишаючи дані наявними та недоторканими.

Операції *CRUD* широко використовуються у багатьох додатках, які підтримуються базовими реляційними базами даних. Ці чотири основні функції *CRUD* неймовірно універсальні в тому, як вони можуть підтримувати різноманітні важливі функції в різних бізнес-моделях та галузевих вертикалях.

Давайте розглянемо приклад того, як реалізується *CRUD*.

Read – користувач може переглядати різні товари в інтернет магазині.

Create – адмінстратор може додавати товари в магазин. Користувач може створювати замовлення.

Update – адмінстратор може змінювати різні поля деяких товарів.

Delete – адмінстратор може видаляти товари яких вже немає в наявності.

1.5. *Firebase* в розробці веб-додатку

Firebase – це платформа для розробки програмного забезпечення. Заснована як база даних у режимі реального часу. Вся платформа - це рішення *Backend-as-a-a-service* як для мобільних, так і для веб-додатків, яке включає послуги зі створення, тестування та управління додатками.

Рішення *BaaS* дозволяють усунути необхідність в управлінні базовими базами даних та отриманні відповідного обладнання. Натомість ви можете підключити їх до свого додатку через спеціальні *API* для кожної окремої служби. У випадку з *Firebase* їх існує 7, які охоплюють весь спектр внутрішніх технологій для програми. Список платформ, з якими інтегрується *Firebase*, включає *Android, iOS, Web* та *Unity*.

Firebase включає різні сервіси для роботи з керованою серверною базою. У наступному розділі ми надамо загальний огляд кожної послуги, доступної на платформі *Firebase*.

Firebase Realtime Database найстабільний сервіс на всій платформі. Реальна база даних – це, по суті, хмарне сховище *NoSQL*, яке можна підключити до програми, щоб забезпечити доступ до даних у реальному часі на різних платформах. Однією з переваг є те, що база даних може працювати в автономному режимі, кешуючи дані в пам’яті пристрою та після повторного підключення до Інтернету, синхронізуючи їх.

Дані зберігаються у форматі *JSON* і можуть запитуватися користувачами. З точки зору безпеки, база даних у реальному часі забезпечує доступ до даних на основі дозволів. Це можна зробити за допомогою автентифікації *Firebase* та надання дозволів за ідентифікацією користувача або правилами безпеки.

Cloud Firestore – це ще одна база даних *NoSQL* в режимі реального часу, розміщена у хмарі. На відміну від бази даних *Firebase Realtime, Cloud Firestore* призначений для корпоративного використання, що передбачає масштабованість, складні моделі даних та розширені параметри запитів. Консоль *Firebase* може

використовуватися для перегляду даних в обох базах даних. Іншим спільним моментом є те, що існують *SDK* для роботи з кодом на стороні сервера обох баз даних. Вони доступні для *Python, Node.js, Golang, Ruby, PHP, Java, NET* та *C#.*

Cloud Storage – це в основному *Google Cloud* для вмісту, створеного користувачем фото, аудіо чи відеофайли.

Authentication Firebase – це функція автентифікації *Google*, розроблена для додатків, що використовують *Firebase*. Це дозволяє використовувати заздалегідь створений або створювати користувацький інтерфейс для автентифікації користувачів та входити в систему за допомогою користувацьких облікових даних, електронних листів або соціальних мереж.

Якщо ви створюєте веб-додаток, прогресивний веб-додаток або мобільну цільову сторінку, вам точно знадобиться хостинг. *Firebase* пропонує статичний веб-хостинг для програм, побудованих за допомогою *HTML, CSS* та *JavaScript*. З точки зору безпеки, він використовує стандартні протоколи *HTTPS* та *SSL* для доставки файлів та інших типів даних.

Serverless applications – це ще одна інтеграція існуючого продукту *Google* у *Firebase*. Це інструмент для запуску внутрішнього коду з хмари на основі подій. Те, як *Cloud Functions* пропонує запустити ваш додаток, називається безсерверною архітектурою. Цей тип архітектури означає побудову додатків як набору окремих функцій, ізольованих у хмарі та пов’язаних між собою за допомогою *API*. Цей тип архітектури додатків став дедалі популярнішим кілька років тому, тому ви можете розглядати *Google* як одного з основних безсерверних постачальників.

Можливості бази даних хоча це завжди залежить від вашого бюджету, *Google* пропонує досить надійні бази даних для використання з вашими додатками. Як *Realtime*, так і *Firestore* можна масштабувати за розміром, пропонуючи повністю безпечне кероване рішення, яке все одно забезпечує легкий доступ до ваших даних через консоль *Firebase*. Оновлення даних та доступ в автономному режимі роблять бази даних придатними для застосування в режимі реального часу, а також синхронізації декількох БД.

Firebase пропонує безліч продуктів, щоб ваша програма працювала. Ви можете вибирати між двома базами даних (база даних у реальному часі та *Firestore*), зберігати носії у хмарі та навіть створювати безсерверні програми за допомогою інтегрованих хмарних функцій.

На початку роботи *Firebase* не вимагатиме оплати більшості своїх послуг і буде абсолютно безкоштовним, коли ви почнете з нею. Це дозволить зрозуміти, чи відповідає воно вашому додатку, та зрозуміти всі особливості. Досягнувши певного обсягу пам'яті бази даних або потребуючи певної послуги, ви завжди можете вибрати між планами. Сторінка цін містить калькулятор цін, який можна регулювати різними параметрами, оскільки це звичайна практика для хмарних служб.

У більшості випадків *Firebase* вимагає мінімальних знань мови програмування та пропонує інтеграцію через користувальницький інтерфейс. Хоча хтось може назвати це мінусом для гнучкості, з іншого боку, це виключає необхідність у складних конфігураціях, тому майже кожен може налаштувати програму.

1.6. Принципи *Serverless* архітектури

Безсерверна – це хмарна модель розробки, яка дозволяє розробникам створювати та запускати додатки без необхідності управління серверами.

Сервери все ще існують у безсерверних, але вони абстрагуються від розробки додатків. Постачальник хмарних послуг займається рутинною роботою з надання, обслуговування та масштабування серверної інфраструктури. Розробники можуть просто упакувати свій код у контейнери для розгортання.

Після розгортання безсерверні програми реагують на попит і автоматично масштабуються вгору та вниз за потреби.

Безсерверність відрізняється від інших моделей хмарних обчислень тим, що хмарний провайдер відповідає за управління хмарною інфраструктурою та

масштабуванням програм. Безсерверні програми розгортаються в контейнерах, які автоматично запускаються на вимогу при виклику.

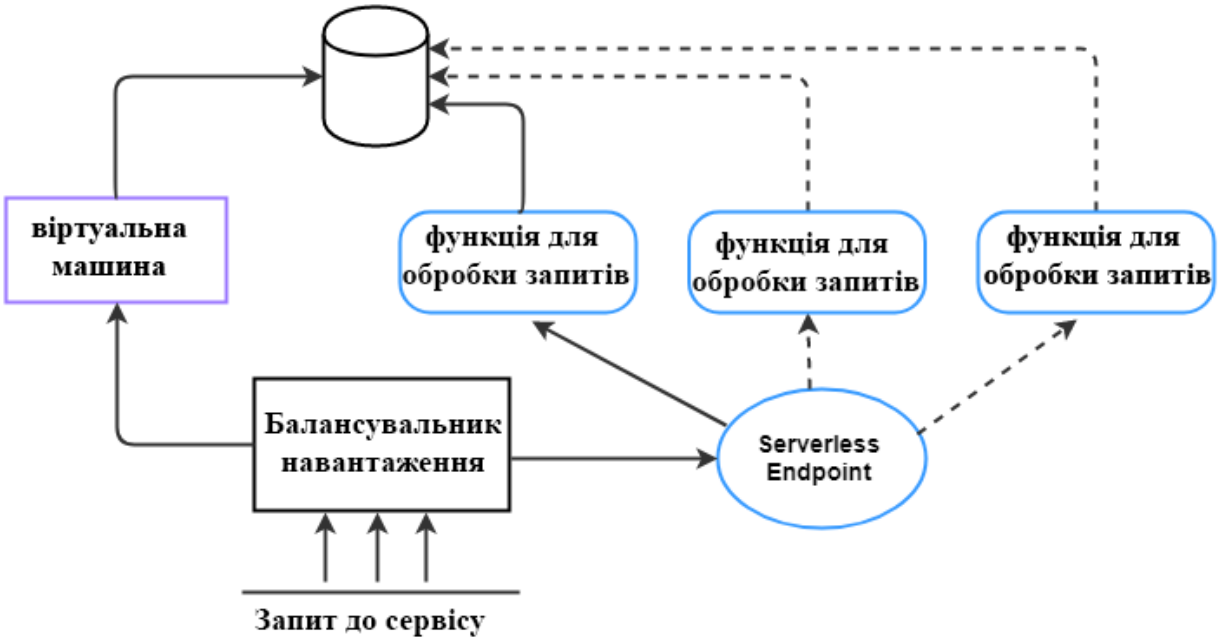

Рис. 1.6. Блок схема безсерверної архітектури.

Відповідно до стандартної моделі хмарних обчислень "Інфраструктура як послуга" (*IaaS*), користувачі попередньо купують одиниці ємності, тобто ви платите загальнодоступному хмарному провайдеру за постійні серверні компоненти для запуску ваших програм. Користувач несе відповідальність за збільшення пропускної спроможності сервера під час високого попиту та за зменшення обсягу, коли ця пропускна здатність більше не потрібна. Хмарна інфраструктура, необхідна для запуску програми, активна, навіть коли вона не використовується.

На відміну від безсерверної архітектури, програми запускаються лише за потреби. Коли подія викликає запуск коду програми, публічний хмарний провайдер динамічно розподіляє ресурси для цього коду. Користувач припиняє платити, коли код закінчує виконуватися. На додаток до переваг вартості та

ефективності, безсерверність звільняє розробників від рутинних та дрібних завдань, пов'язаних із масштабуванням програми та наданням серверів.

Завдяки безсерверним рутинним завданням, таким як управління операційною системою та файловою системою, виправлення безпеки, балансування навантаження, управління пропускною спроможністю, масштабування, реєстрація та моніторинг перевантажуються постачальнику хмарних послуг.

Можна створити повністю безсерверну програму або програму, що складається з частково безсерверних та частково традиційних компонентів мікросервісів.

Безсерверна архітектура ідеально підходить для асинхронних додатків без стану, які можна запускати миттєво. Так само безсерверність добре підходить для випадків використання, які бачать рідкісні, непередбачувані сплески попиту.

Подумайте про таке завдання, як пакетна обробка вхідних файлів зображень, яка може працювати нечасто, але також повинна бути готова, коли велика партія зображень надходить одразу. Або таке завдання, як відстеження вхідних змін до бази даних, а потім застосування низки функцій, таких як перевірка змін відповідно до стандартів якості або їх автоматичний переклад.

Безсерверні додатки також добре підходять для випадків використання, які включають вхідні потоки даних, ботів чату, заплановані завдання або бізнес-логіку. Деякі інші поширені випадки використання без серверів – це інтерфейси *API* та веб-додатки, автоматизація бізнес-процесів, безсерверні веб-сайти та інтеграція між різними системами.

1.7. Роль формату *JSON* в розробці веб-додатків

JSON – це формат, який зберігає структуровану інформацію і в основному використовується для передачі даних між сервером і клієнтом. Файл *JSON* являє

собою більш просту і легку альтернативу розширенню з аналогічними функціями *XML* (*Extensive Markup Language*).

JSON – це загальний формат даних з мінімальною кількістю типів значень: рядки, числа, логічні значення, списки, об'єкти та *null*. Хоча позначення є підмножиною *JavaScript*, [3] ці типи представлені у всіх поширених мовах програмування, що робить *JSON* хорошим кандидатом для передачі даних через мовні прогалини.

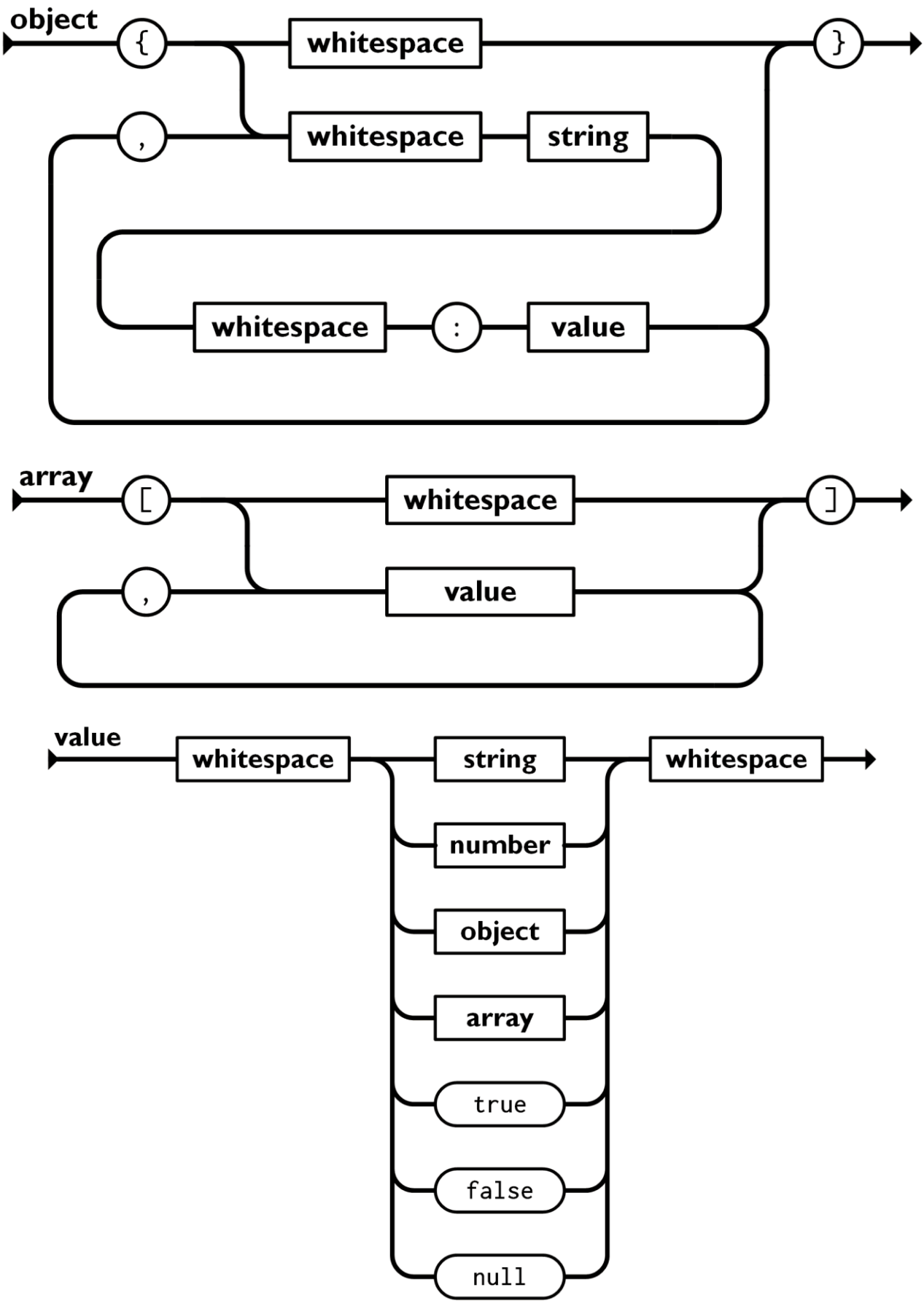

Рис. 1.7 Структурна схема формату *JSON*

1.8. *Vue*, як клієнт веб-додакту.

Vue.js – це прогресивна платформа *JavaScript* з відкритим кодом, яка використовується для розробки інтерактивних веб-інтерфейсів. *Vue.js* фокусується на рівні представлення, пропонує багато функціональних можливостей для рівня перегляду і може бути використаний для створення потужних односторінкових веб-програм. [2]

Vue.js використовує віртуальний *DOM*. Віртуальний *DOM*, по суті, є копією основного елемента *DOM*, поглинає всі зміни, призначені для *DOM*, і присутній у вигляді структур даних *JavaScript*. Зміни, внесені до структур даних *JavaScript*, порівнюються із початковою структурою даних.

І лише остаточні зміни відображаються на реальному *DOM*, який буде видимим для користувачів. Це винахідливий метод, менш дорогий, і зміни можна вносити швидше.

Функція прив'язки даних допомагає маніпулювати або призначати значення атрибутам *HTML*, змінювати стиль, призначати класи за допомогою директиви прив'язки, що називається *v-bind*, доступна у *VueJS*.

Vue.js надає кілька методів для застосування переходу до елементів *HTML*, коли вони додаються, оновлюються або видаляються з *DOM* [8]. *Vue.js* має вбудований компонент переходу, який обгортає елемент, відповідальний за повернення ефекту переходу. Розробники можуть без проблем додати сторонні бібліотеки анімації та зробити взаємодію з користувачем більш інтерактивною.

Як я вже згадував вище, *Vue.js* пропонує шаблони на основі *HTML*, які пов'язують *DOM* з даними екземпляра *Vue.js*. *Vue.js* компілює шаблони у функції віртуального *DOM Render*. Веб-розробник може використовувати шаблон функцій візуалізації та може замінити шаблон функцією візуалізації.

Це одна з важливих особливостей *Vue.js*. Це допомагає прослуховувати зміни, внесені до елементів інтерфейсу, та виконує необхідні обчислення. Для цього не потрібно додаткове кодування.

Спостерігачі застосовуються до даних, які змінюються. Наприклад, елементи введення форми. Тут розробнику не потрібно додавати додаткові події. *Watcher* піклується про будь-які зміни даних, роблячи код простим і швидким.

Методи використовуються, коли ми хочемо змінити стан компонента або коли сталася подія, яка не обов'язково пов'язана з мутованими даними екземпляра. Методи можуть приймати аргументи, але не відстежувати жодних залежностей.

Метод зазвичай створює деякий побічний ефект всередині компонента. Методи запускатимуться щоразу, коли компонент перезавантажується.

Обчислювані властивості використовуйте обчислювану властивість, коли ми хочемо мутувати властивість, яка залежить від іншої властивості, що змінюється. Обчислювані властивості, як правило, залежать від інших властивостей даних.

Будь-яка зміна залежних властивостей спричинить логіку для обчислюваної властивості. Обчислювані властивості кешуються на основі їх залежностей, тому вони будуть повторно запущені, лише якщо залежність зміниться.

Використовуйте спостерігачів, коли нам потрібно виконати певну логіку в результаті зміни, яка відбулася в певній властивості даних. Переглянуті властивості діють лише на одну властивість. Це найбільш корисно, коли ви хочете виконувати асинхронні або дорогі операції у відповідь на зміну даних.

Vue.js є гнучким та масштабованим. На практиці це означає, що його можна використовувати для величезного модульного *SPA* а також для побудови невеликих, інтерактивних деталей, що інтегруються за допомогою іншої технології.

Іншими словами, це може бути все, що вам потрібно: просто бібліотека у вашому проекті або повнофункціональний фреймворк, який використовується для створення цілого продукту.

Написання типового коду – це витрата часу та ресурсів. *Vue.js* допомагає уникнути такої напруженої роботи, надаючи безліч вбудованих рішень та застосовуючи конвенції про економію зусиль.

Деякі з них включають рідну підтримку таких речей, як управління державою, анімація та компонування компонентів. Хоча це може здатися дуже технічним, винос простий: навіщо витрачати час на те, щоб винаходити колесо, коли ви можете просто використовувати *Vue*?

Технічне обслуговування – це часто забута частина розробки програмного забезпечення.

Після того, як ви створили та розгорнули свій додаток, ви також повинні бути в курсі виправлень помилок, нових функцій та інших удосконалень. *Vue.js* полегшує оновлення. Нові випуски, включаючи великі оновлення, зроблені таким чином, щоб зберегти якомога більше зворотної сумісності.

Крім того, проект на основі *Vue* не потребуватиме рефактору швидко, звільняючи ваші дорогоцінні ресурси для розробки функцій, а не того, що в основному переробляється з точки зору бізнесу.

Розробники люблять *Vue.js* не лише за те, що це чудова технологія, а й за те, що вона створена з урахуванням них.

По-перше, існує *vue-cli* (*CLI* означає «інтерфейс командного рядка»), зручний інструмент, який полегшує запуск (а також налаштування, запуск, аналіз та тестування) нового проекту за допомогою вибраних вами інструментів. *Vue-cli* набагато гнучкіший у порівнянні з аналогічними пропозиціями конкурентів і надає багато попередньо налаштованих налаштувань.

Також доступний графічний інтерфейс (*Vue GUI*), який може допомогти вам розпочати свій проект, не вводячи загадкові команди в термінал.

Інші приклади корисних інструментів *Vue* – або наборів інструментів, враховуючи їх широкі функціональні можливості – включають *nuxt.js* («метафреймворк для універсальних додатків»), *Vue Storefront* («наступне покоління *PWA* для електронної комерції) та *Vuetify* (« Матеріальний дизайн компонентний фреймворк ”).

Вам не потрібно багато часу, щоб ознайомитись із цією технологією, крім того, немає необхідності з’ясовувати тонни інструментів. Якщо ви все-таки

застрягли, просто зверніться до спільноти – це дійсно надійно, тому ви зазвичай отримуєте відповідь швидко.

Vue.js насправді робить найкраще з будь-яких ресурсів, які ви йому виділяєте.

По-перше, програми *Vue* мають менший розмір (тому вони швидше завантажуються і використовують меншу пропускну здатність) і, як правило, ефективніші, ніж еквівалентні програми, вбудовані в інші фреймворки.

Це не питання думки – для підтвердження цього твердження існують фактичні еталони.

Поки ми говоримо про продуктивність, *Vue* самостійно піклується про велику кількість оптимізацій, тому розробникам не потрібно турбуватися про налаштування програми в міру її зростання.

Вони можуть зосередитися на додаванні до нього нових речей, орієнтованих на цінність. Це також означає, що програми *Vue* є масштабованими: порівняно легко перенести їх із простої односторінкової програми на вдосконалену систему.

Якщо у вас є проект, який потребує підвищення продуктивності, переписування його у *Vue* може бути срібною кулею.

Завдяки своєму прогресивному характеру, фреймворк можна впроваджувати у вашу кодову базу поступово немає необхідності переробляти все це одним махом. Переходьте компонент за компонентом, щоб замість цього зробити цілі зусилля більш керованими.

1.9. *MVVM* архітектура.

Патерн *MVVM* (*Model-View-ViewModel*) дозволяє відокремити логіку додатки від візуальної частини (подання). Даний патерн є архітектурним, тобто він задає загальну архітектуру програми.

Даний патерн був представлений Джоном Госсманом (*John Gossman*) в 2005 році як модифікація шаблону *Presentation Model* і був спочатку націлений на розробку додатків в *WPF*. І хоча зараз цей патерн вийшов за межі *WPF* і

застосовується в самих різних технологіях, в тому числі при розробці під *Android*, *iOS*, проте *WPF* є досить показовою технологією, яка розкриває можливості даного патерну.

MVVM складається з трьох компонентів: моделі (*Model*), моделі подання (*ViewModel*) і уявлення (*View*).

Модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

Нерідко модель реалізує інтерфейси які дозволяють повідомляти систему про зміни властивостей моделі. Завдяки цьому полегшується прив'язка до подання, хоча знову ж пряму взаємодію між моделлю і представленням відсутня.

View або подання визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до *WPF* уявлення – це код в *xaml*, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

Хоча вікно (клас *Window*) в *WPF* може містити як інтерфейс в *xaml*, так і прив'язаний до нього код *C #*, проте в ідеалі код *C #* не повинен містити якийсь логіки, крім хіба що конструктора, який викликає метод *InitializeComponent* і виконує початкову ініціалізацію вікна . Вся ж основна логіка додатки виноситься в компонент *ViewModel*.

Однак іноді в файлі пов'язаного коду все може знаходитися певна логіка, яку важко реалізувати в рамках паттерна *MVVM* у *ViewModel*.

Подання і не виконує жодних події за рідкісним винятком, а виконує дії в основному за допомогою команд.

ViewModel або модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу *INotifyPropertyChanged* автоматично йде зміна

відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані.

ViewModel також містить логіку по отриманню даних з моделі, які потім передаються в уявлення. І також *VewModel* визначає логіку по оновленню даних в моделі.

Оскільки елементи уявлення, тобто візуальні компоненти типу кнопок, не використовують події, то уявлення взаємодіє з *ViewModel* за допомогою команд.

Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду під *ViewModel*. А *ViewModel* вже отримує передані дані і відповідно до них оновлює модель.

Підсумком застосування патерну *MVVM* є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати.

1.10. Висновки до розділу.

Веб-програма (веб-програма) – це прикладна програма, яка зберігається на віддаленому сервері та доставляється через Інтернет через інтерфейс браузера.

Веб-програми можуть бути розроблені для найрізноманітніших потреб, і ними може користуватися кожен; від організації до приватної особи з багатьох причин.

Отже у веб-додатку можуть використовуватись різноманітні технолгоії, підходи, патерни проектування, бази данних і фреймворки.

РОЗДІЛ 2
ЗАСТОВУВАННЯ ВЕБ-ДОДАТКІВ В ЕЛЕКТРОННІЙ КОМЕРЦІЇ

2.1. Поняття елктронної комерції.

Електронна комерція, або електронна комерція, стосується транзакцій, що здійснюються через Інтернет. Щоразу, коли приватні особи та компанії купують або продають товари та послуги в Інтернеті, вони беруть участь у електронній комерції. Термін електронна комерція також охоплює інші види діяльності, включаючи Інтернет-аукціони, Інтернет-банкінг, платіжні шлюзи та Інтернет-квитки.

2.2. Класифікація електронної комерції.

Існує багато способів класифікації веб-сайтів електронної комерції. Ви можете класифікувати їх за продуктами чи послугами, які вони продають, сторонами, з якими вони укладають операції, або навіть платформами, на яких вони працюють.

Це інтернет-магазини. Одяг, меблі, інструменти та аксесуари – все це приклади фізичних товарів. Покупці можуть купувати фізичні товари через Інтернет-магазини, відвідуючи веб-сайти магазинів, додаючи товари в кошик для покупок та здійснюючи покупку.

Після того, як покупець зробив покупку, магазин доставляє товари безпосередньо біля їх порогу. Є також Інтернет-магазини, де клієнти можуть здійснити покупку в Інтернеті, але самі зайти до магазину, щоб забрати продукти.

Окрім продуктів, послуги також можна придбати через Інтернет. Щоразу, коли ви наймаєте викладачів, фрілансерів та консультантів через Інтернет-платформи, ви ведете бізнес із сервісними електронними магазинами.

Процес купівлі послуг залежить від продавця. Деякі можуть дозволити вам придбати їх послуги відразу з їх веб-сайту або платформи.

Деякі постачальники послуг, навпаки, вимагають від вас першого зв'язку (тобто замовлення консультації), щоб визначити ваші потреби.

Транзакції електронної комерції здійснюються через Інтернет, тому в області електронної комерції продукти зазвичай називають «електронними товарами». Термін цифрові товари позначає всі предмети, які перебувають у цифровому форматі, включаючи електронні книги, онлайн-курси, програмне забезпечення, графіку та віртуальні товари.

2.3. Класифікація електронної комерції за злученими сторонами.

Як випливає з назви, модель електронної комерції *B2C* являє собою операцію між бізнесом та приватними особами. Електронна комерція *B2C* є найпоширенішою бізнес-моделлю як серед фізичних, так і через інтернет-магазини.

У моделі електронної комерції *B2B* обидві сторони є бізнесом. При такому виді операцій один бізнес надає іншому товари та / або послуги.

Slack, платформа для спілкування між віддаленими підприємствами, та *Xero*, хмарне програмне забезпечення для бухгалтерського обліку для бізнесу, є прикладами компаній *B2B*.

Бізнес-модель *C2B* являє собою транзакцію, в якій фізичні особи створюють цінність для бізнесу, на відміну від традиційної моделі від бізнесу до споживача, коли компанії забезпечують цінність. Споживачі забезпечують компанії продуктами та / або послугами, співпрацюють над проектами і, зрештою, допомагають бізнесу збільшувати свій прибуток.

Електронна комерція *C2C* трапляється, коли дві залучені сторони є споживачами, які торгують між собою. *eBay* та *Craigslist* – це приклади інтернет-ринків, де люди купують та продають товари один одному.

Моделі електронної комерції *G2B* трапляються, коли уряд забезпечує компанії товарами та послугами. Державні закупівлі, центри обробки даних та електронне навчання – все це приклади електронної комерції *G2B*.

Модель *B2G* стосується компаній та підприємств, які надають товари та послуги для уряду. Наприклад, *OpenGov* – це компанія, яка пропонує урядові хмарні платформи для спілкування, звітності та бюджетування.

Щоразу, коли споживачі сплачують податки, медичне страхування, електронні рахунки або запитують інформацію щодо державного сектору, вони беруть участь у *C2G*.

2.4. Як працює електронна комерція?

Подорож фірми електронної комерції починається із створення веб-сайту електронної комерції.

Після цього товари відображаються разом із необхідними деталями, такими як опис товару та цінники.

Для зручності клієнтів біля товарів розміщується кнопка заклику до дії, як-от "Купити зараз". Якщо вони хочуть придбати товар, вони можуть просто натиснути кнопку та здійснити платіж.

Клієнтів зазвичай просять розмістити замовлення, заповнивши форму. Там вони повинні надати всі необхідні дані, наприклад адресу доставки. У галузі електронної комерції використовуються різні способи оплати.

Покупці можуть здійснити платіж через платіжний шлюз, або використовувати свої кредитні картки, або накладеним платежем. Цьому переважно віддають перевагу старші клієнти.

У цьому варіанті покупець здійснює платіж, коли замовлений товар доставляється на його особисту адресу. Оплату можна здійснити готівкою або за допомогою карток або іншими зручними способами.

Якщо ви не задоволені якістю товару або помилково вам доставляють інший товар, ви можете використовувати зворотну логістичну систему. Це просто повернення речей та повернення грошей.

Вам потрібно повідомити, що ви хочете повернути замовлений товар. До вас прийде представник компанії та забере товар назад. Гроші будуть переведені на ваш рахунок.

Просування товару в електронній комерції здійснюється також цифровим способом. Два найбільш відомі методи – це цифрова реклама та маркетинг електронною поштою.

Оголошення розміщуються на соціальних медіа-платформах, які є найбільш переважними для цільових груп. Усіх відвідувачів просять вказати свої електронні адреси.

На основі цього складається список електронної пошти. Після цього надсилаються особисті електронні листи. Іноді особисті сповіщення також надсилаються в акаунти соціальних мереж про костюми потенційних клієнтів.

2.5. Юридичні вимоги до веб-сайту електронної комерції?

Перш ніж запустити веб-сайт електронної комерції, потрібно переконатися, що він юридично відповідає законодавству про конфіденційність даних, яке застосовується до регіону вашої аудиторії. *GDPR* та *CCPA* є двома першими нормативними актами, які безпосередньо впливають на збір, використання та зберігання даних у широкому масштабі.

Політики, які потрібно встановити на веб-сайті електронної комерції: політика конфіденційності, правила та умови.

Застереження (пов'язані з вашим товаром, послугою, вмістом або доходом афілійованих осіб)

Це лише деякі з основних правил, необхідних для чіткого спілкування з відвідувачами.

2.6. Висновки до розділу.

Електронна комерція зробила світ значно меншим, завдяки Інтернету. Це зручно, швидко і легко.

Кількість людей, які звертаються до електронних покупок, з часом лише зростатиме. Тож змусіть свій бізнес виходити в Інтернет для електронної комерції та створіть безладний і простий у користуванні веб-сайт електронної комерції, і спостерігайте, як збільшиться прибуток.

РОЗДІЛ 3
РОЗРОБКА ІНТЕРНЕТ МАГАЗИНУ НА БАЗІ ФРЕЙМВОРКУ *VUE.JS*

3.1. Логіка роботи

Веб додаток дозволяє оглядати товари інтернет магазину, формувати кошик замовлення, обирати кількість замовляємого товару, оформлення замовлення.

Логіка зі сторони користувача:

– Користувач заходить на сторінку каталогу товарів;

– Перед відображенням товарів додаток робить *http* запит – *get* для отримання товарів в форматі *JSON*;

– Користувач може переглядати товари на сторінці обраного товару, додавати до кошику товар;

– Після додавання товару до кошика передається *id* товару до компоненту кошику і в масив товарів у кошику додається обраний;

– По кліку на елемент кошику в «навбарі» сайту користувач переходить на сторінку кошику і бачить додані ним товарів. В користувача є можливість збільшити кількість товарів, видалити товар;

– Для замовлення товару потрібно заповнити форму ввести ім'я, прізвище, номер телефону, електронну пошту.

Логіка зі сторони адміністратора:

– Сторінка входу для адміністраторів.
– Змінений інтерфейс магазину з підвищеними можливостями.
– Можливість додавати товари.
– Можливість змінювати товари і видаляти товари.
– Можливість переглядати всі замовлення що були зроблені.

3.2. Програмне забезпечення

Для реалізації проекту необхідно встановити наступні пакети.

Редактор коду з підтримкою терміналу:

– *Visual Studio Code*.

Головний *JavaScript* рушій:

– *Node.js*.

Для реалізації проекту необхідно встановити наступні бібліотеки:

– *Npm*;
– *Vue.js*;
– *Vue cli*;
– *Less.css;*
– *Axios;*
– *Firebase;*
– *Vuex;*
– *Router;*
– *Ant design vue;*
– *Moment;*
– *Eslint;*
– *Fetch.*

3.3. Встановлення бібліотек

Встановити будь-який текстовий редактор в якому зручно писти код в моєму випадку *Visual Studio Code*.

Встановити з офіційного сайту *Node.js*

Node.js це – виконавча *JavaScript*. Що ж це означає, і як працює?

Оточення *Node.js* включає все, що вам потрібно для виконання програми, написаної на *JavaScript*.[4]

Раніше ви могли запустити *JavaScript* тільки в браузері, але одного разу розробники розширили його, і тепер ви можете запускати *JavaScript* на своєму комп'ютері в якості окремого додатка. Так з'явився *Node.js*.

Тепер ви можете зробити набагато більше з *JavaScript*, ніж просто інтерактивні веб-сайти.

Тепер у *JavaScript* є можливість робити те, що можуть робити інші скриптові мови програмування, такі як *Python*.

Обидва – браузерні *JavaScript* і *Node.js* запускаються в середовищі виконання *V*8. Цей движок використовує ваш *JS* код, і перетворює його в більш швидкий машинний код. Машинний – низько

При створенні нового проекту скористаємося *Vue-cli*. Так можна створювати проекти з використанням офіційних шаблонних проектів *Vue*, або одного з безлічі шаблонних проектів з відкритим вихідним кодом, і, звичайно ж, ви можете створити свій власний і використовувати його в будь-якому місці.

VueJS може бути встановлений з командного рядка за допомогою інтерфейсу командного рядка *vue-cli*. Він допомагає швидко створювати і компілювати проект за допомогою *vue-cli*. Отже, для початку встановимо *vue-cli* в якості глобального пакету: *npm install -g vue-cli*

Створимо проєкт інтернет магазину: *vue create ishop*

Встановимо бібліотеку *Axios*: *npm install axios*

Axios – це *JavaScript* бібліотека для виконання або *HTTP*-запитів в *Node.js*, або *XMLHttpRequests* в браузері. Вона підтримує Проміс – новинку *ES*6. Одна з особливостей, яка робить її краще *fetch* () – автоматичні перетворення *JSON*-даних.

Встановимо бібліотеку *Firebase*: *npm install firebase*

3.4. Структура проєкту

Після встановлення всіх потрібних бібліотек розберемо структуру проєкту.

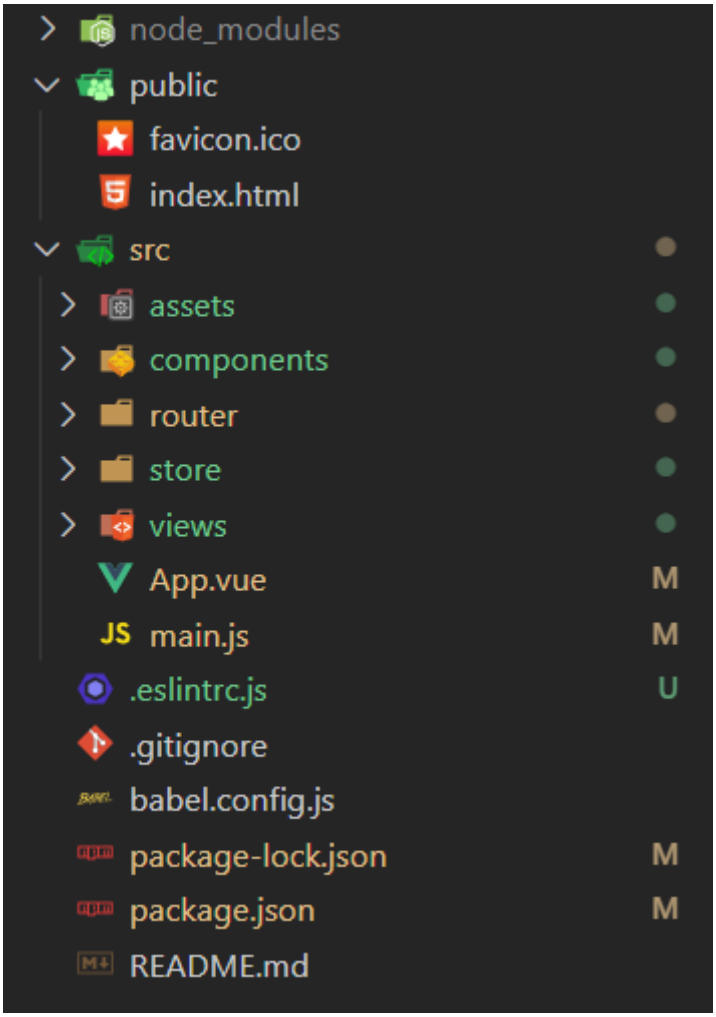

Рис. 3.1. Структура проєкту

Main.js – зазвичай це файл *JavaScript*, який ініціює цей кореневий компонент елементом на вашій сторінці. Він також відповідає за налаштування плагінів та сторонніх компонентів, які ви можете використовувати у своєму проєкті.

Index.html – ця сторінка надає точку входу в *html*, забезпечуючи елемент для завантаження *Vue*-файлів та імпорт *main.js* для ініціалізації вашого додатка.

Package.json – цей файл містить різні метадані, що стосуються проекту. Цей файл використовується для надання інформації *npm*, яка дозволяє йому ідентифікувати проект, а також обробляти залежності проекту. Він також може містити інші метадані, такі як опис проекту, версія проекту в певному дистрибутиві, інформація про ліцензію, навіть дані конфігурації - все це може бути життєво важливим як для *npm*, так і для кінцевих користувачів пакету.

App.vue – це головний файловий компонент, який є батьківськми для всіх інших компонентів. Він містить 3 фрагменти коду: *HTML, CSS* та *JavaScript*.

Views – папка, яка містить компоненти, які прив'язані до маршрутизатора.

Components – папка, яка містить компоненти, які багаторазово використовується.

Store – папка, яка містить файли *Vuex*. *Vuex* - це шаблон управління станом плюс бібліотека для розробки програм *Vue.js*. Централізовано сховище даних через яке компоненти можуть «спілкуватися» між сбою.

Router – папка, яка містить файли *Vue Router*. *Vue Router* допомагає зв’язати *URL*-адресу / історію браузера та компоненти *Vue*, дозволяючи певним шляхам відображати будь-яке представлення, пов’язане з ним.

Assets – папка що містить загальні стилі проєкту.

Node modules – ви можете уявити папку *node_modules* як кеш для зовнішніх модулів, від яких залежить ваш проект. Коли ви встановлюєте їх у *npm*, вони завантажуються з Інтернету та копіюються у папку *node_modules*, і *nodejs* навчається шукати їх там, коли ви їх імпортуєте (без певного шляху).

3.5. Розробка програмного засобу

Зареєструемося веб-додатку *firebase*. Створимо базу даних і підключимо авторизацію для адміністраторів інтрнет магазину. Підключимо до нашого *main.js* файлу налаштування конфігу *firebase*.

Створимо в базі даних одного користувача, який і буде нашим адміністратором.

Створимо форму входу для адміністраторів, приховаємо цю сторінку від звичайних користувачі магазину по особливому *url*.

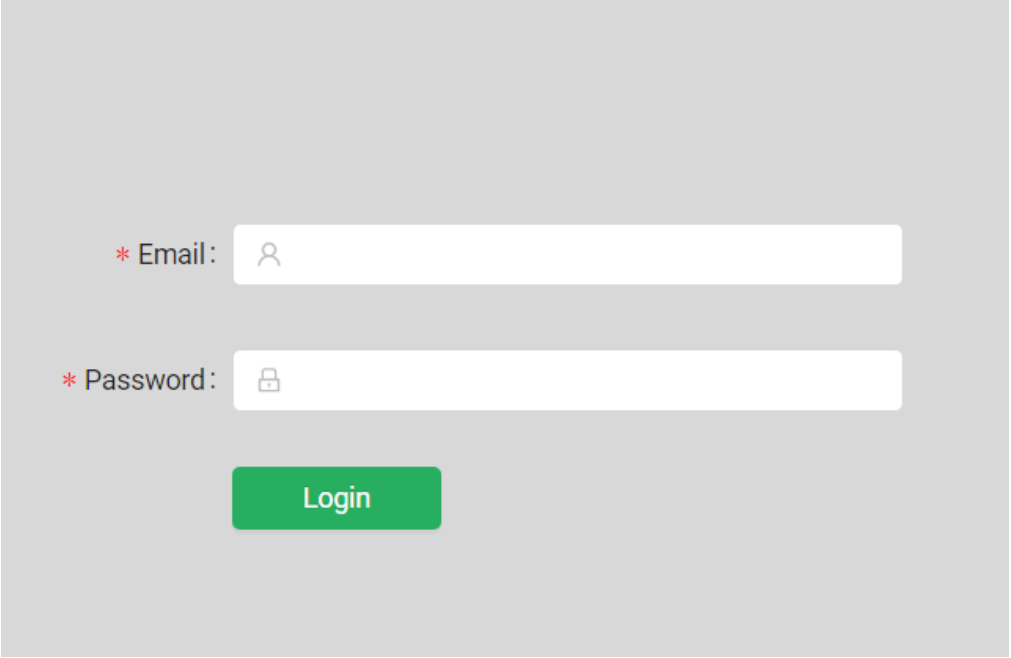

Рис. 3.2. Форма авторизації адміністратора

Після авторизації адмінстратора запишемо інформацію про данного адміністратора до *Vuex* і надамо більше функціоналу на сайті. Тільки для адміністратора буде видно доданий розділ *Admin*, а також можливості в ньому.

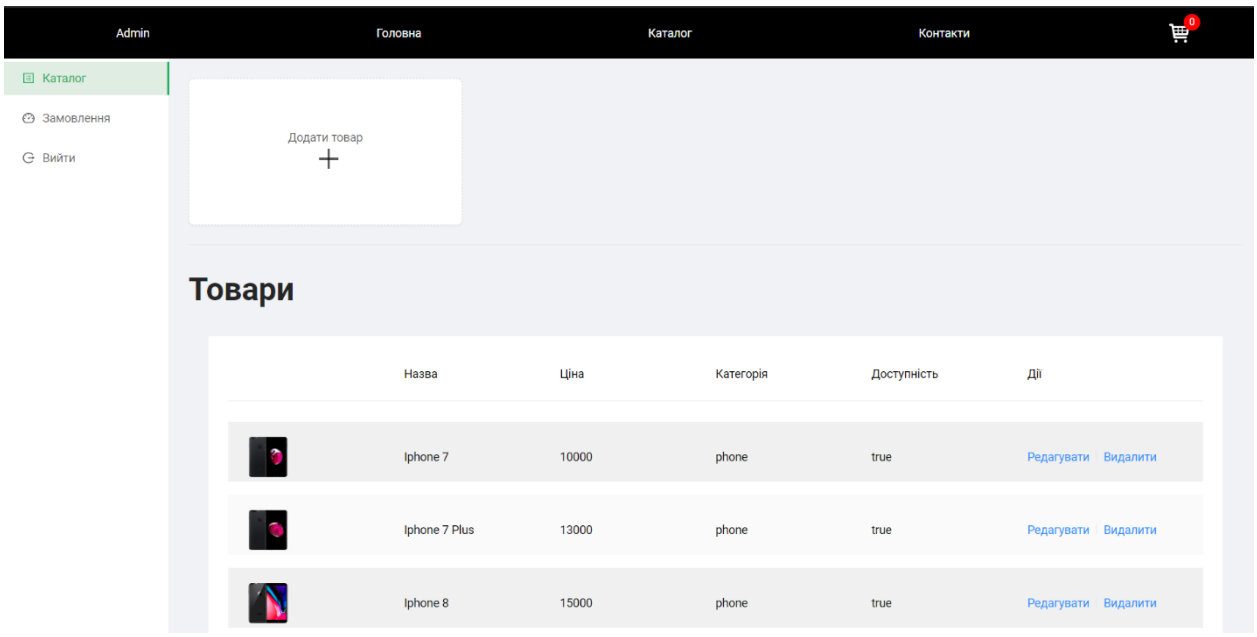

Рис. 3.3. Вікно панелі адміністратора

Адміністраторська панель виглядає наступним чином:

– Розділ товарів в каталозі;
– Розділ замовлень, зроблених користувачем;
– Розділ для додавання товарів;
– Для кожного товару є дії редагувати/видалити;
– Завершити сеанс адміністратора.

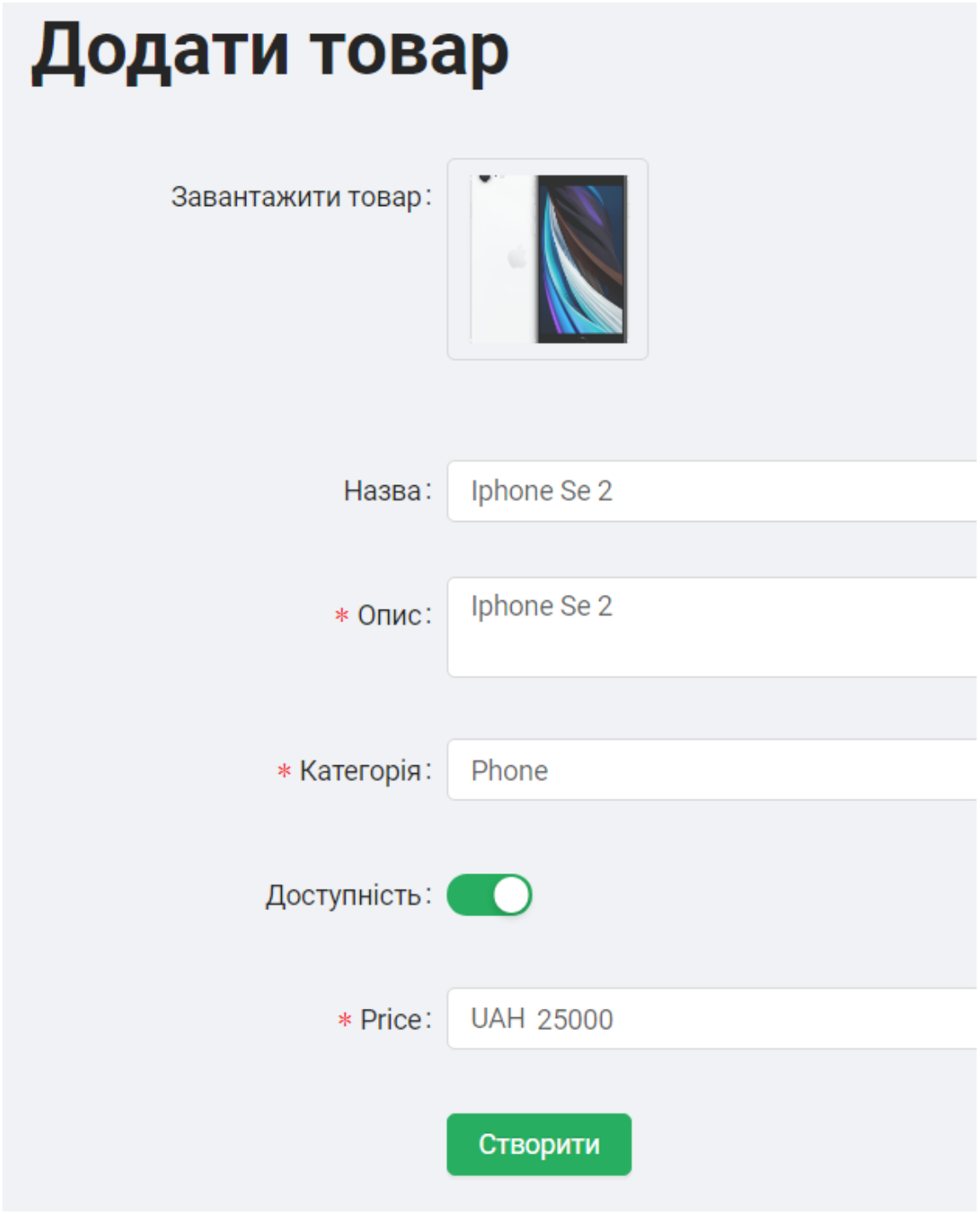

Рис. 3.4. Форма створення товару

Розробимо форму для створення товарів до каталогу. Звяжемо заповненні дані в об’єкт який будемо відправляти з запитом до бази даних *firebase*.

```
const newProduct = new Product(
        payload.image,
        payload.name,
        payload.desc,
        payload.category,
        payload.avaliable,
        payload.price,
        getters.user.id
       );
const product = await firebase.database().ref("products").push(newProduct);
```

По аналогії реалізуємо змінення товару:

```
await firebase.database().ref("products").child(id).update({ ...fields });
```

Видалення з бази даних товару:

```
await firebase.storage().refFromURL(imageUrl).delete();
```

Видалення з бази даних зображення товару:

```
await firebase.database().ref("products").child(id).remove();
```

Після цього налаштуємо відображення наших товарів в публічному каталозі для всіх користувачів.

```
const product = await firebase.database().ref("products").once("value");
```

Стилізуємо вигляд наших товарів на сайті. Додамо кнопку для додавання товарів до кошику по *id* товару. Додамо посилання на товар по натисканню на зображенню товарів.

Наш каталог має наступний вигляд:

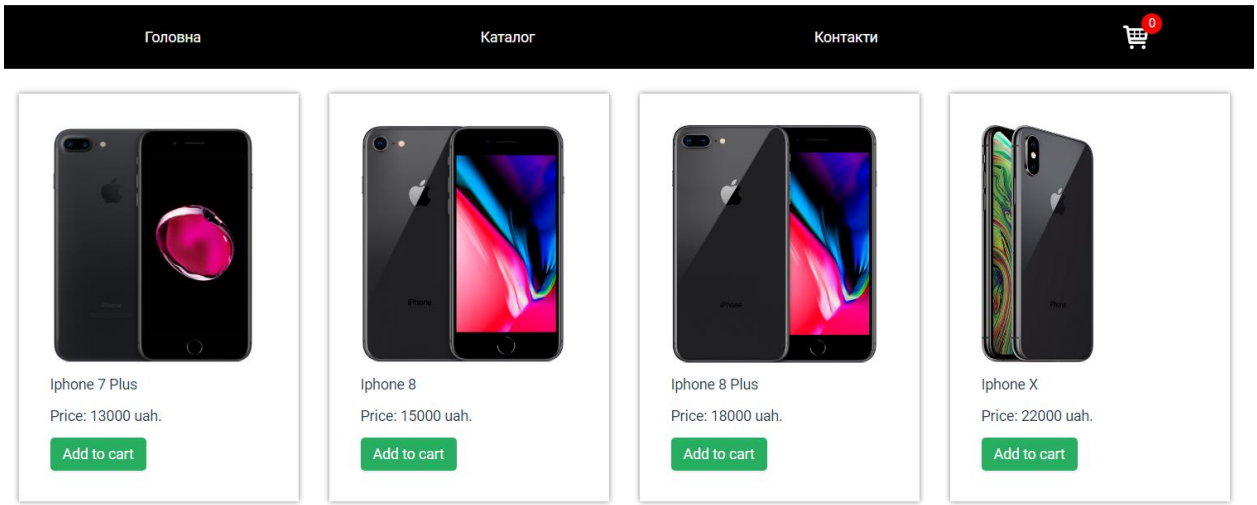

Рис. 3.5 Вікно каталогу

По натисканню на зображення налаштуємо динамічний роутінг по кожному товару. А також сторінку для детальної інформації про товар.

Рис. 3.6. Вікно інформації про товар

Url даної сторінка має вигляд:

/#/catalog-item-view/-MaVVcGPGOmC9qf8nAbJ

Кожен *url* генерується через *id* товару з бази. Тому не прийдеться всім товарам прописуватити роути вручну.

Налаштуємо функціонал для додавання товарів до кошика. При додаванні товару більше ніж один, додається не товар а його кількість.

Розробимо функціонал для відправлення товарів в кошику а також персональних даних.

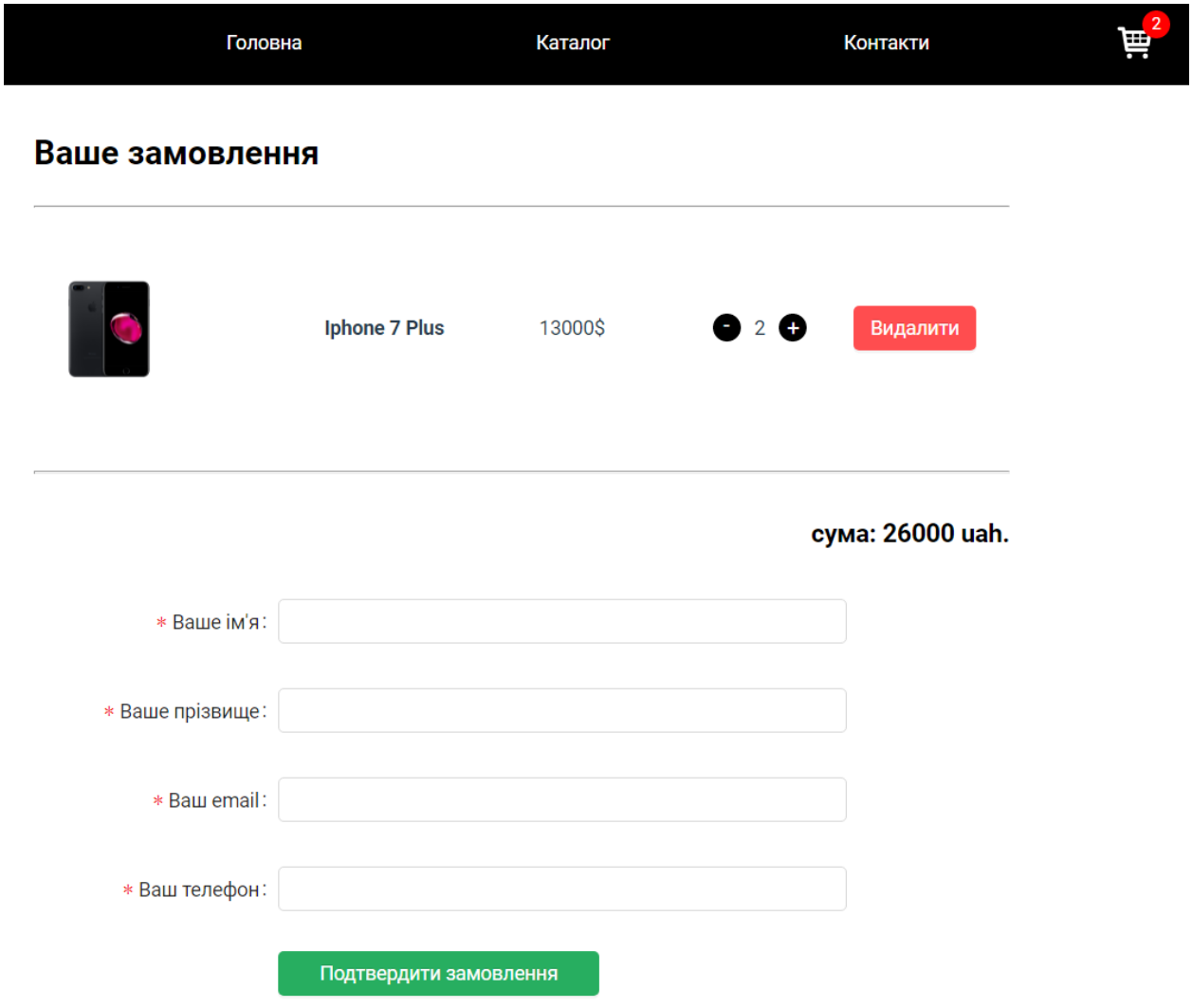

Рис. 3.7. Вікно кошика магазину

Налаштуємо додавання замовлення до бази даних.

```
newOrder = {
        name: currentOrder.name,
        lastname: currentOrder.lastname,
        tel: currentOrder.tel,
        email: currentOrder.email,
        total: currentOrder.cartTotalCost,
        date: currentOrder.date,
        order: currentOrder.order,
       };
const order = await firebase.database().ref("orders").push(newOrder);
```

Після підтвердження замовлення. Замовлення додається до бази даних і відображається в адмін панелі.

3.6. Висновки до розділу

В третьому розділі на основі компонентного підходу і безсерверної архітектури створили аналог роботи інтернет магазину з адмін панеллю.

Було спроектовано функції додатку та розроблено програмний інтерфейс програмної реалізації електронного цифрового підпису.

В третьому розділі реалізовано.

Описано інтерфейс адміністратора. Було продемонстровано основні елементи, які допомагають створити товари, замовлення. Було розглянуто їх основні властивості.

ВИСНОВКИ

Під час виконання дипломного проекту було проаналізовано такі теми: автоматизація розробки веб додатків, застосування веб-додатків в електронній комерції, розробка інтернет магазину на базі фреймворку *Vue.js*.

Зараз в електронна комерція моєї країни швидко розвивається, але існує як багато позитивних сторін так і негативних з точки зору зручності в використанні, функціоналу тощо.

Важко зараз уявити життя без інтернет магазинів. Вони спрощують життя мілiонам людей щодня. Оскільки на сьогоднішній день гарно розвнинена логістика, що дозволяє економити дуже багато часу населенню.

Зараз не потрібно ходити і обирати товар, створювати черги – цього і дозволяють уникнути інтернет магазини. Але існує багато проблем з ними такі, як політика користувача, процедура повернення коштів в разі повернення товарів, не якісного ПО інтернет магазинів, надійність інтернет транзакцій, через маловідомі платіжні системи, безпека персональних даних.

Не всі компанії можуть собі дозволити якісну кібербезпеку. Це дуже важлива деталь *e-commerce* ринку.

Зараз є лва основних шляхи для створення інтернет магазину: за допомогою *CMS*, за допомогою *SPA*-додатків.

На прикладі створення інтернет магазину було розглянуто як створювати *SPA*-додатки за допомогою фреймворка *Vue*. Продемонстровано, що являє собою *SPA*-додаток буквально одна сторінка, яка постійно взаємодіє з користувачем, динамічно переписуючи поточну сторінку, а не завантажуючи цілі нові сторінки з сервера. Існує багато прикладів односторінкових додатків, які ми бачимо кожен день: *Trello, Facebook, Gmail, Twitter*. При завантаженні нових модулів в *SPA* контент на них оновлюється тільки частково, так як немає необхідності повторно завантажувати незмінні елементи. Це збільшує швидкість відповіді і скорочує передається обсяг даних між браузером і сервером. Це було і продемострировано,

що при додаванні товару в адмін панелі він одразу з’являвся в каталозі, при зміні змісту товару він одразу змінюється в усіх частинах магазину, як і в адмін панелі, так і в каталозі.

У дипломному прокті було описано якими способами можна відправляти запити, та отримувати дані в *JSON* форматі, як працює компонентне проеткування і серверна архітектура, як компоненти взаємодіють між собою, і все це дозволяє робити фреймворк *Vue*.

Також в дипломному проєкті було розглянуто більшість основних можливостей *Vue*, а саме: робота з *props*, яка дозволяє робити багаторазові компоненти; *router*, який дозволяє гнучко налашовувати *url*-адреси, *vuex* -загальне сховище даних для всіх компонент; *v-for* – цикли, які дозволяють відображати *n*-кількість елементів зроблених в одному стилі; *v-if* – условні оператори які мають можливість приховувати елементи на сторінці. Головним завданням даних можливостей є створення додатків різної складності, один із видів було продемонстровано *SPA* додаток з можливостями *CRUD*.

Дипломний проект було виконано з дотриманням діючих стандартів та положень [9], [10].

СПИСОК БІБІЛОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація *Vue.js – https://vuejs.org/* (дата звернення 17.05.2021)
2. Сучасний підручник *JavaScript – https://learn.javascript.ru/* (дата звернення 17.05.2021)
3. Офіційна документація *Node.js – https://nodejs.org/uk/*
4. Розробка односторінкових веб-додатків. 2017.
5. *Html book – http://htmlbook.ru/* (дата звернення 19.05.2021)
6. *w3 school – https://www.w3schools.com/html/* (дата звернення 21.05.2021)
7. Офіційна документація *LESS – https://lesscss.org* (дата звернення 20.05.2021)
8. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86с.
9. Слободян О. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: Видавництво НАУ, 2017. – 63с.
10. UX-дизайн. Практичний посібник з проектування досвіду взаємодії. Унгер Р., Чендлер К. (дата звернення 21.05.2021)