МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ВАСИЛІВ Святослав Васильович

Інтелектуалізована веб-система "Конструктор логотипів" / Intellectualized Web System "Logos Designer"

спеціальність: 121 - Інженерія програмного забезпечення
освітньо-професійна програма - Інженерія програмного забезпечення

Кваліфікаційна робота

Виконав студент групи ІПЗм-21
С. В. Василів

_____

Науковий керівник:
к.т.н., доцент, В. І. Манжула

_____

Кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2022

ЗМІСТ

ВСТУП

Актуальність проекту визначається постійним збільшенням темпів автоматизації в усіх сферах життя і стрімко зростаючими можливостями обчислювальних систем. У світлі появи в останні десятиліття технічних можливостей, все частіше обслуговування користувачів здійснюється за допомогою даних систем. особливо зростає потреба в програмних продуктах, що дозволяють користувачеві виробляти самообслуговування - таких як системи електронних платежів, покупки квитків і замовлення їжі. Перераховані вище сервіси набирають все більшої популярності, так як це є зручним способом зробити все швидко і без використання третіх осіб. У тому числі стало актуальним створення веб-сайту, який би дозволив вдало поєднати зазначені вище сервіси в одному комунікаційному додатку.

На даний момент вже накопичено певний досвід розробок в даній області, але від цього коло завдань тільки розширюється. Є багато сервісів, які здійснюють розробку логотипів, замовлення та розроблення проекту, проте кожен з них має свої переваги і недоліки. Точніше кажучи, більшість створених програмних комплексів орієнтовані на конкретні завдання і певні умови.

Об’єктом дослідження є створення веб-сайту рекламного агенства «Fantazər». Предмет дослідження – методи та програмні засоби створення веб-сайту рекламного агенства «Fantazər».

Мета роботи - розробка клієнтського веб-додатку для комплексних рішень по створенню і розробки брендів і ребрендингу онлайн .

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Коротка характеристика об’єкту управління

Рекламне агентство «Fantazər» розміщений у центральному парку м.Київ.Організаційно-правова форма - юридична форма, в якій здійснюється реєстрація й діяльність юридичної особи. Рекламне агентство «Fantazər» є товариством з обмеженою відповідальністю – заснований декількома особа - ми товариство, статутний капітал якого розділений на частки визначених установчими документами розмірів; учасники товариства з обмеженою відповідальністю невідповідають за його зобов'язаннями і несуть ризик збитків, пов'язаних з діяльністю товариства в межах вартості внесених ними вкладів.

Статутний капітал товариства з обмеженою відповідальністю складається з вартості вкладів його учасників. Ця організаційно-правова форма поширена серед дрібних і середніх підприємств.

Місія і цілі рекламного агентства «Fantazər». Місія організації визначає місце, роль і становище в суспільстві, її суспільний статус. Іноді це поняття замінюють таким виразом, як «девіз організації». Місія організації - це вираженесловесно, основне соціально значуще, функціональне призначення організації в довгостроковому періоді. Як правило, організація при розробці своєї місії підкреслює саме соціальний характер свого призначення для суспільства. Місія рекламного агентства «Fantazər» - це задоволення потреб сучасної людини в комплексних рішеннях по створенню та розробці брендів.

Місія реалізує цілі розвитку організації, які по суті визначають перспективні напрями. У залежності від значимості, цілі поділяються на головну і додаткові цілі, що забезпечують досягнення головної мети. Далі вони поділяються до рівня завдань.

Місією рекламного агентства є створення проектів і опрацювання до ключових деталей , проводження тестування повних створюваних продуктів. Люди хочуть бачити творчі фантазії у своєму бренду у першу чергу для естетичного задоволення, таким чином основними цілями рекламного агентства «Fantazər» є:

○ Вміння грамотно поєднувати ідеї та технології.

○ У майбутньому стати однією з кращих рекламних агентств Києва.

○ Докласти зусиль для підтримки і зміцнення здоров'я, задоволеності працею і фінансовим добробутом своїх співробітників.

Портрет цільового користувача рекламного агентства «Fantazər». Цільовий користувач представляє собою групу людей зі схожими потребами щодо конкретного товару або послуги, достатніми ресурсами, а також готовністю і можливістю купувати. Ця група, перш за все, визначається тим, який саме соціального прошарку призначений товар або послуга, і тим, як цінові та інші характеристики товару її визначають. Ці умови, як можна бачити, також представляють собою єдине логічне ціле. Що стосується портрета користувача рекламного агентства - це різні люди. Дуже багато молодих, 20-40-літніх креативних чоловіків та жінок , у яких є хороша робота, цілі та креативні ідеї щодо їхнього бренду приходять до нас щоб втілити їх у життя, світські люди, які знаходяться в увазі громадськості - все це люди, готові заплатити за якісне комплексне рішення та розробку бренду або ребрендингу.

1.2. Опис предметної області

Відповідно до зростання популярності веб-додатків, все більше і більше підприємців переносять свої бізнеси у Інтернет. Майже кожна мережа закладів, компаній тощо у будь-якому місті має свій вебсайт. Якщо компанія приймає цифрове замовлення, веб-сайт стає його обличчям.

Все більше і більше сучасні веб-сайти створені з використанням SPAпідходу. Single page application (SPA) - це підхід до розробки веб-сайтів, де вміст кожної нової сторінки подається не з завантаження нових HTMLсторінок, а згенерований динамічно завдяки можливості JavaScript маніпулювати елементами DOM (Document Object Model) на самій існуючій сторінці.

У більш традиційній архітектурі веб-сторінок сторінка index.html може посилатися на інші HTML-сторінки на сервері, які браузер завантажуватиме та відображатиме з нуля.

SPA-підхід дозволяє користувачеві продовжувати користуватися сторінкою та взаємодіяти з нею, коли нові елементи оновлюються чи отримуються, і це значно пришвидшує взаємодію та перезавантаження контенту [2].

Крім того, HTML дозволяє нам змінювати URL-адресу сторінки без перезавантаження сторінки, дозволяючи нам створювати окремі URLадреси для різних представлень даних.

Потрапивши всередину SPA, програма може динамічно отримувати вміст із сервера через AJAX-запити або веб-сокети. Це дозволяє браузеру тримати відкриту поточну сторінку під час подання запитів на сервер у фоновому режимі, щоб отримати додатковий вміст або нові "сторінки" взагалі.

Прикладом динамічного запиту можуть бути проміжні результати, які з'являються під формою ведення пошукового запиту під час введення тексту. Він відбувається у фоновому режимі та оновлює DOM елементи.

Серверні запити можуть отримувати будь-які дані. Слід розглянути використання SPA-підходу у наступних випадках:

1) коли взаємодія між користувачем і вашим додатком повинна бути насиченою. Такі програми, як Карти Google, широко використовують цей підхід, щоб забезпечити зміни в режимі реального часу під час прокрутки з

одного місця на інше або натискання на маркери місць, щоб переглянути фотографії певного місця.

2) коли необхідно надати оновлення в режимі реального часу на сторінці, наприклад сповіщення, потокова передача даних та графіки в режимі реального часу вимагають використання такого підходу.

Уникати використання SPA варто тоді, коли вміст веб-сайту - статичний. У такому випадку введення SPA погіршує час завантаження для користувача, вимагаючи від користувача завантажити та виконати JavaScript-скрипти, перш ніж мати можливість переглядати будь-який вміст. Більшість веб-додатків використовують архітектурний підхід REST API.

REST API - це програмний архітектурний стиль, який визначає набір правил, які використовуються для створення веб-сервісів. Веб-сервіси, що відповідають архітектурному стилю REST, відомі як RESTful веб-сервіси. Це дозволяє робити запити до системи доступу та керувати веб-ресурсами, використовуючи єдиний і заздалегідь визначений набір правил. Взаємодія в системах на базі REST відбувається через протокол передачі гіпертексту в Інтернеті (HTTP).

Важливо створити API REST відповідно до галузевих стандартів, що призводить до полегшення розробки та збільшення кількості клієнтів. Є шість API RESTful архітектурних обмежень:

1) Уніфікований інтерфейс;

2) Stateless;

3) Cacheable;

4) Client-Server;

5) Layered System;

Єдиним необов'язковим обмеженням архітектури REST є код на запит. Якщо система порушує будь-які інші обмеження, її не можна чітко називати RESTful.

1) Уніфікований інтерфейс. Це ключове обмеження, яке розрізняє REST API і не-REST API. Це дозволяє припустити, що має існувати єдиний спосіб взаємодії з певним сервером незалежно від пристрою чи типу програми (веб-сайт, мобільний додаток).

Існує три принципи Уніфікованого інтерфейсу:

- індивідуальні ресурси визначаються у запитах. Наприклад: API / користувачі.

- клієнт має представлення ресурсу, і він містить достатньо інформації для зміни або видалення ресурсу на сервері за умови, що він має дозвіл на це. Наприклад, зазвичай користувач отримує ідентифікатор користувача, коли користувач запитує список користувачів, а потім використовує його для видалення або зміни цього конкретного користувача.

- кожне повідомлення містить достатньо інформації, щоб описати, як обробити повідомлення, щоб сервер міг легко аналізувати запит.

2) Stateless. Це означає, що необхідний стан для обробки запиту міститься в самому запиті, і сервер не зберігатиме нічого, пов’язаного з сеансом. У програмі REST клієнт повинен включити всю інформацію для сервера для виконання запиту, як частину параметрів, заголовків або URI запитів. Stateless забезпечує більшу доступність, оскільки сервер не повинен підтримувати, оновлювати та повідомляти цей стан сеансу. Є недолік, коли клієнту потрібно надсилати занадто багато даних на сервер, щоб зменшити сферу оптимізації мережі та вимагати більшої пропускної здатності.

3) Cacheable. Кожна відповідь повинна містити, відповідь є кешованою чи ні, і скільки тривалості відповідей може кешуватися на стороні клієнта. Клієнт поверне дані зі свого кешу для будь-якого наступного запиту, і не потрібно буде знову надсилати запит на сервер. Добре керований кешування частково або повністю виключає деякі взаємодії клієнт-сервер, ще більше покращуючи доступність та продуктивність. Але колись є шанси, що користувач може отримати несвіжі дані.

4) Client-Server. Програма REST повинна мати архітектуру клієнт-сервер. Клієнт – це той, хто запитує ресурси та не переймається збереженням даних, який залишається внутрішнім для кожного сервера, а сервер - це той, хто володіє ресурсами і не стосується інтерфейсу користувача або стану користувача.

Вони можуть еволюціонувати самостійно. Клієнту не потрібно нічого знати про бізнес-логіку, а сервер не повинен нічого знати про інтерфейс інтерфейсу.

5) Layered System. Архітектура додатків повинна складатися з декількох шарів. Кожен шар не знає нічого про будь-який шар, окрім шару безпосередньо, і між клієнтом та кінцевим сервером може бути багато проміжних серверів. Посередницькі сервери можуть покращити доступність системи, включивши балансування навантаження та надавши спільні кеші.

Найбільш важливим та необхідним у створення сучасного веб-додатку - це дизайн. Веб-дизайн важливий, оскільки впливає на те, як аудиторія сприймає бренд. Враження, яке справляється на них, може або змусити їх залишитися на сторінці та дізнатися про компанію, або залишити сторінку та звернутися до конкурента. Хороший веб-дизайн допомагає зберігати потенційних клієнтів на веб-сайті.

Існує неймовірно величезна кількість веб-сайтів різних компанійрізної спеціалізації, які використовують найкращі та найефективніші практики створення та розробки веб-додатків. Однак не всі з них поєднують все в собі. Розглянемо найбільш потрібні та важливі аспекти та ознаки, які сучасному веб-сайту слід мати:

1) Адаптивный дизайн та мобільність. Адаптивний веб-дизайн - це підхід, я якому дизайн та розробка повинні відповідати поведінці та оточенню користувача на основі розміру екрана, платформи та орієнтації девайса. Іншими словами, адаптивний веб-сайт - це веб-сайт, який добре

відтворюється на екранах різного розширення та різних пристроях (персональний комп'ютер, телефон, планшет).

Це важливо, тому що на смартфони зараз припадає понад 51% всього інтернет-трафіку, а планшети прийшли трохи більше 12%. І ця кількість зростає. Наприклад всесвітньо-відома компанія “Google” акцентує увагу на мобільні пристрої.

2) Висока швидкість завантаження веб-сайту У всіх веб-сайтів швидкість завантаження дуже різна, але все менше і менше залишаються повільних, тому що сучасна людина - нетерпляча та не стане очікувати довго поки веб-додаток завантажиться, тому становиться дуже важливим швидкість завантаження веб-сайту. Згідно статистики, майже половина користувачів Інтернету очікує, що сайт завантажиться за 2 секунди або менше, і вони, як правило, відмовляються від сайту, який не завантажується протягом 3 секунд! Приблизно 79% покупців в Інтернеті, які мають проблеми з роботою веб-сайтів, кажуть, що вони не повернуться на сайт, щоб придбати знову, і близько 44% з них повідомили б другу, якщо у них поганий досвід покупок в Інтернеті.

Цю проблему вирішують по-різному, існує декілька шляхів поліпшити швидкість завантаження веб-сайту, наприклад кешування веб-переглядача та спочатку завантаження вмісту, що знаходиться зверху (вміст, який ви бачите, не прокручуючи сторінку вниз). Розглянемо основні проблеми, вирішення яких можуть покращити швидкість завантаження веб-сайту.

- велика кількість зображень;
- великі файли;
- непотрібний код;
- надмірне використання CSS та JavaScript;
- Flash;

Однак одним із найпростіших способів є оптимізація та стискання зображень. При цьому вони повинні залишатись у високій якості.

3) Оптимізація пошукових систем (SEO). SEO означає оптимізацію веб-сайту для відображення в таких пошукових системах, як Google. Правильно налаштована оптимізація може притягувати тисячі трафіку на сайт щомісяця без зайвих зусиль. І навпаки зроблено погано, жоден користувач не знайде веб-додаток в Google. Слід оптимізовувати заголовки та метатеги. Теги заголовків - це те, як сканери пошукової системи оцінюють релевантність сторінки. Мета опис - це те, що відвідувачі бачать у результатах пошуку.

Як теги заголовка, так і метатеги повинні містити ключові слова, щоб збільшити рейтинг сторінки в результатах пошуку. Кожна сторінка повинна мати свій унікальний заголовок та метатег.

4) Забезпечення сайту шифруванням SSL. Зелений замок у адресному рядку поруч із веб-сайтом - це ознака того, що веб-сайт має SSL-шифруванням. Google надає зашифрованим сайтам невеликий стимул для SEO. Але важливіше те, що це фактор довіри користувача.

Це особливо актуально, якщо на веб-сайті здійснюється купівля чи продаж будь-яких товарів чи послуг. Люди хочуть, щоб їх інформація була безпечною ще до того, як вони відкриють свій гаманець.

5) Google Analytics. Аналітика Google може допомогти зрозуміти, звідки рухається трафік на веб-сайт.

6) Можливість здійснювати оплату онлайн. Часто веб-додатки представляют можливість робити замовлення чи купівлю товару. Це, звісно, є великою перевагою та зручністю для сучасного користувача.

7) Ефективна навігація. Хороша навігація - один з найважливіших аспектів зручності використання веб-сайту. Прості меню HTML або JavaScript, як правило, працюють найкраще і виглядають послідовними у всіх браузерах та платформах.

Слід максимально обмежити кількість пунктів. Випадаюче меню або допоміжна навігація можуть працювати краще на великому сайті з багатьма розділами та сторінками.

Навігація більше, ніж меню. Ось деякі інші аспекти, які слід врахувати:

- хороша функція пошуку;
- сторінка помилок (наприклад 404);
- інформаційний заголовок і колонтитул;

8) Правильне відловлювання помилок. Правильна обробка помилок та опис екранних повідомлень дуже важливі для хорошої зручності використання. Однак це часто не помічають. Правильне поводження з помилками на рівні коду гарантує надійність вебсайту. Відображення коректного відповідного повідомлення про помилку покращує користувацький досвід та загальну зручність використання.

9) Зручні форми. Форми є дуже важливим елементом на ділових веб-сайтах або на звичайних сторінках авторизації чи реєстрації.

Щоб отримати максимальну користь від веб-сайту, важливо забезпечити, щоб форми були простими у використанні та доступними для всіх.

Слід відзначити кілька корисних порад:

- використовування правильних міток для всіх полів;
- дотримання принципів дизайну хорошої форми;
- зведення до мінімуму кількість полів;
- підказки та пропозиції;
- відображення на екрані повідомлення про завершення;
- правильна валідація полів;

На жаль, багато з них не оптимізовані, не дозволяють користувачам легко розібратись у дизайні та знайти саме те, що вони шукають, не адаптовані під різні девайси та браузери, містять застарілу інформацію.

1.3. Огляд і аналіз існуючих аналогів, що реалізують функції предметної області

Для ефективного створення якісної та потрібної для застосування програмної системи необхідно також провести порівняння вже існуючих аналогів програмних систем для даної області застосування.

Перед розробкою веб-сайту або веб-додатку потрібно зробити попередній аналіз сайтів для ресторанів. [8]

1. Аналіз конкурентів.
2. Виписати кожну особливість сайту.
3. Зробити аналіз ЦА (цільової аудиторії).
4. Виявлення плюсів та мінусів проекту.

На основі зібраних даних формується чітка логіка проекту. Щоб реалізувати цей проект було проведено аналіз конкурента.

На рисунку 1.1. представлено головну сторінку рекламного агентства «PeaR Team»

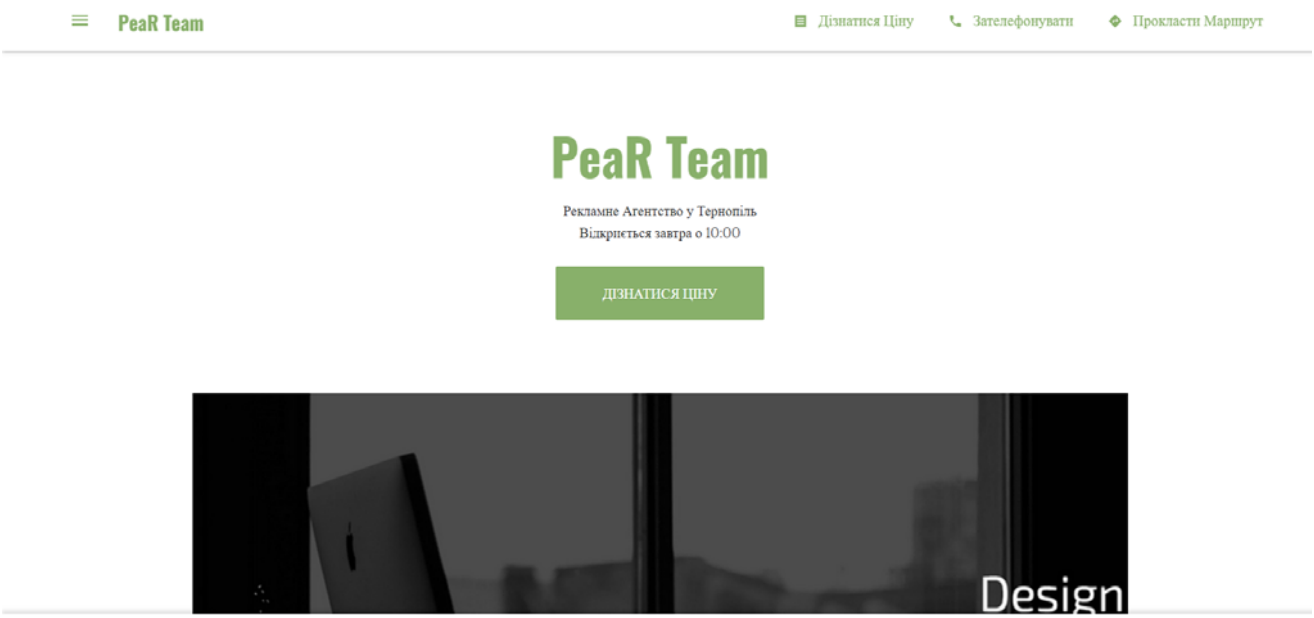

Рис. 1.1. Головна сторінка рекламного агентства «PeaR Team» у м. Тернопіль.

На рисунку 1.2 зображено сторінку із оновленнями, фірмовими виробами та із можливістю здійснення онлайн замовлення .

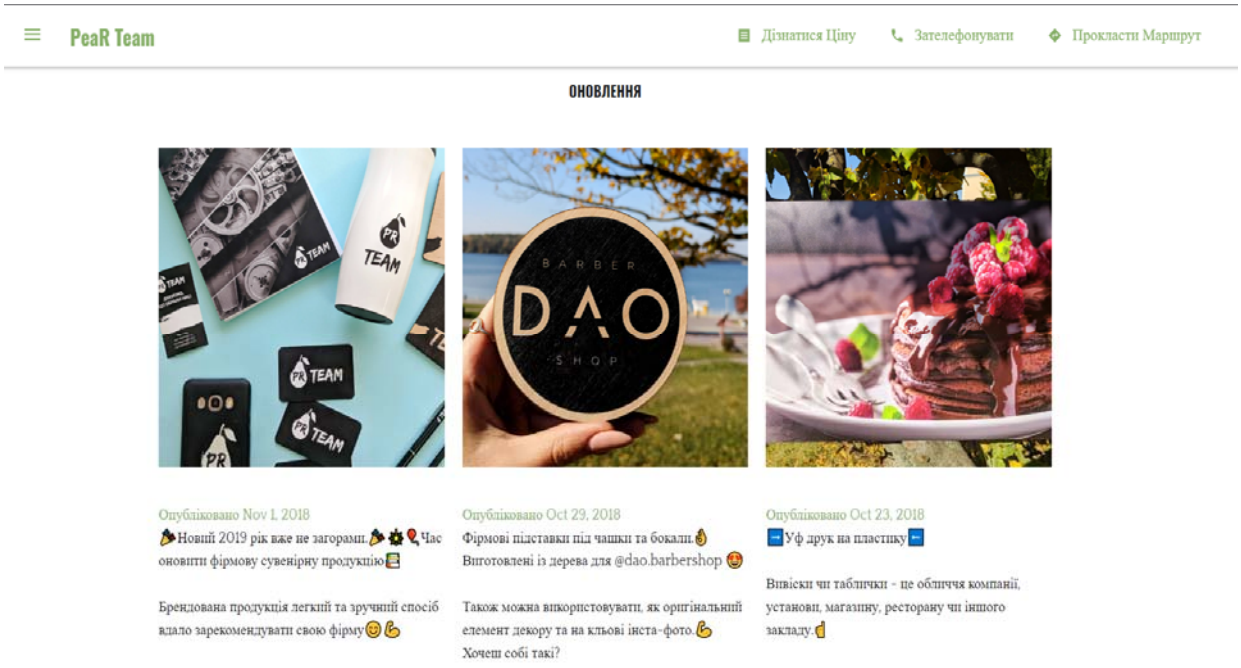

Рис. 1.2. Головна сторінка рекламного агентства «PeaR Team» у м. Тернопіль.

На рисунку 1.3 зображено головну сторінку сайту “ PeaR Team ” конкурента по бізнесі.

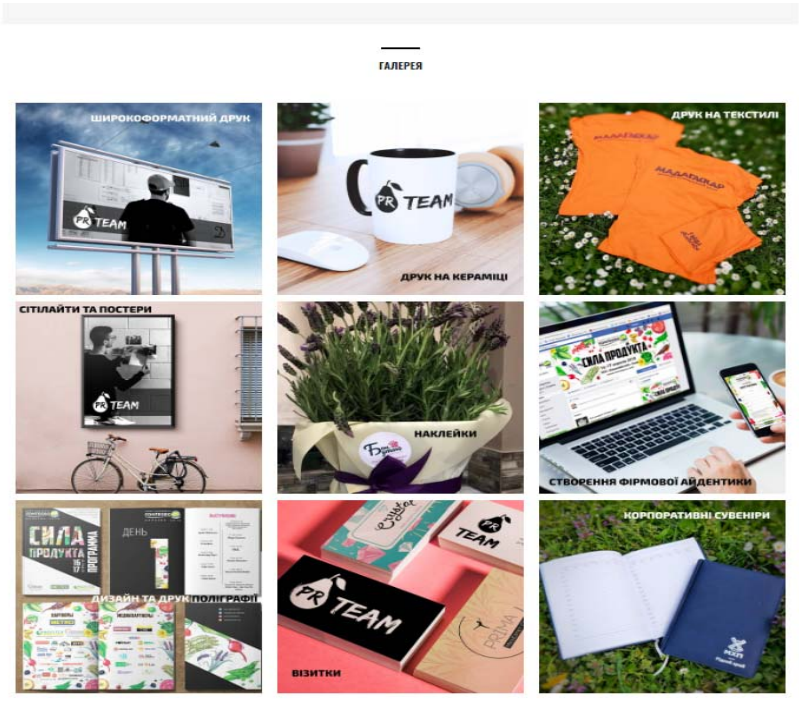

Рис. 1.3. Головна сторінка рекламного агентства «PeaR Team» у м. Тернопіль.

Рис. 1.4. Сервісні послуги рекламного агентства «PeaR Team» у м. Тернопіль.

Було проведено первинний аналіз (табл. 1.1) сайтів-конкурентів, які видавалися пошуковою системою Google по ключовим запитам «Рекламні агентства Тернопіль» найпершими. Аналіз дозволив виявити гарні сторони та проблеми, які мають бути уникнути при створені магістерського проекту.

Таблиця 1.1.

Аналіз розглянутих комерційних сайтів для рекламних агентств

Критерії	Web-сайт агентства "CityLife"	Web-сайт агентства "GoMedia"	Web-сайт агентства "Sprint"	Web-сайт агентства "Далі"
Відсутність непотрібної реклами	+	+	-	+
Застарілий дизайн	+	+	-	-
Можливість зворотного зв'язку	+	+	+	+
Компактність елементів контексту	-	+	+	+
Зручна панель навігації	-	+	-	-
Швидке завантаження сторінок	+	+	-	-
Відгуки	+	+	-	-
Цінова політика	+	+	+	+

Серед основних недоліків існуючих аналогів слід відзначити найбільш загальні:

− недопрацьований контент;

− застарілий дизайн;

− погана навігація.

До переваг сайтів-аналогів можна віднести:

− швидке завантаження сторінок;

− компактність елементів контексту;

− сайт містить Відгуки;

− Відсутність непотрібної реклами;

Саме ці переваги будемо намагатися реалізувати у власному проекті.

1.4. Специфікація вимог до системи

Специфікація вимог до програмної системи – це специфікація окремого програмного продукту, програми або набору програм, які виконують деякі дії в деякому середовищі. Тобто – це повний опис поведінки системи що розробляється [3].

В загальному випадку специфікація включає наступне:

- глосарій проекту;
- опис варіантів використання.

Наведемо список основних термінів та понять в області розробки веб-сайту для рекламного агентства «Fantazər».

Глосарій – список понять в специфічній області знання з їх визначеннями [3]. Ці поняття та визначення подано у таблиці 1.2.

Таблиця 1.2

Глосарій проекту

Термін	Опис терміну
1. Основні поняття та категорії предметної області та проекту	
Веб-сайт	сукупність веб-сторінок та залежного вмісту, доступних у мережі Інтернет, які об'єднані як за змістом, так і за навігацією під єдиним доменним ім'ям. Фізично сайт може розміщуватися як на одному, так і на кількох серверах.
Рекламне агентство	тип творчих або медійних підприємств , де здійснюються комплексні рішення по створенню та розробці брендів або ж ребрендингу , а також із допомогою засобів масової інформації надається послуги по приверненні додаткової уваги до бренду
Програмний продукт	програмний засіб, програмне забезпечення, які призначені для постачання користувачів (покупцеві, замовникові) [3].
Бізнес-процес	будь-яка діяльність, що має вхідний продукт, додає вартість до нього, та забезпечує вихідний продукт для внутрішнього або зовнішнього споживача [3].
2. Користувачі системи	
Користувач	Людина, яка користується послугами обчислювальної техніки для отримання інформації чи розв’язання різноманітних задач.
3. Вхідні та вихідні документи	
Список рекламних агентств	документ, який містить інформацію про наявні рекламні агентства в конкретному місці
Рейтинг рекламних агентств	документ, який забезпечує сортування рекламних агентств за певним критерієм

Діаграма варіантів використання (Use Case Diagram) є важливою частиною планування проекту. Вона є найпростішою з поведінкових діаграм

та демонструє результат взаємодії між акторами та прецедентами. Створена діаграма дозволяє отримати інформацію щодо майбутнього функціоналу системи, а також показує яким чином актор може діяти в відповідній системі.

Для створення діаграми використовуються такі актори:

User Customer - той хто обирає певну систему та збирається взаємодіяти з нею. Після відбору всіх потенційних користувачів системи формується перелік можливих варіантів взаємодії з нею. Варіанти обираються відповідно до вимог, яким відповідає веб - ресурс.

Діаграма Use Case була розроблена на основі інформації про користувачів веб - додатку, а також можливих варіантів взаємодії з ним.

Фіолетовим кольором визначається загальні дії. Решта відповідно до кольору кожного юзера.

Варіанти використання web-сайту:

- реєстрація;
- авторизація;
- перегляд інформації;
- перегляд історії
- залишати відгуки;
- пошук по сайту;
- оплата онлайн;
- підтримка юзерів
- підтримка сайту;
- налаштування профілю в особистому кабінеті

На основі сформованих даних про акторів та всі можливі варіанти використання веб-сайту, була розроблена Use Case діаграма. Вона представлена на рисунку 1.5.

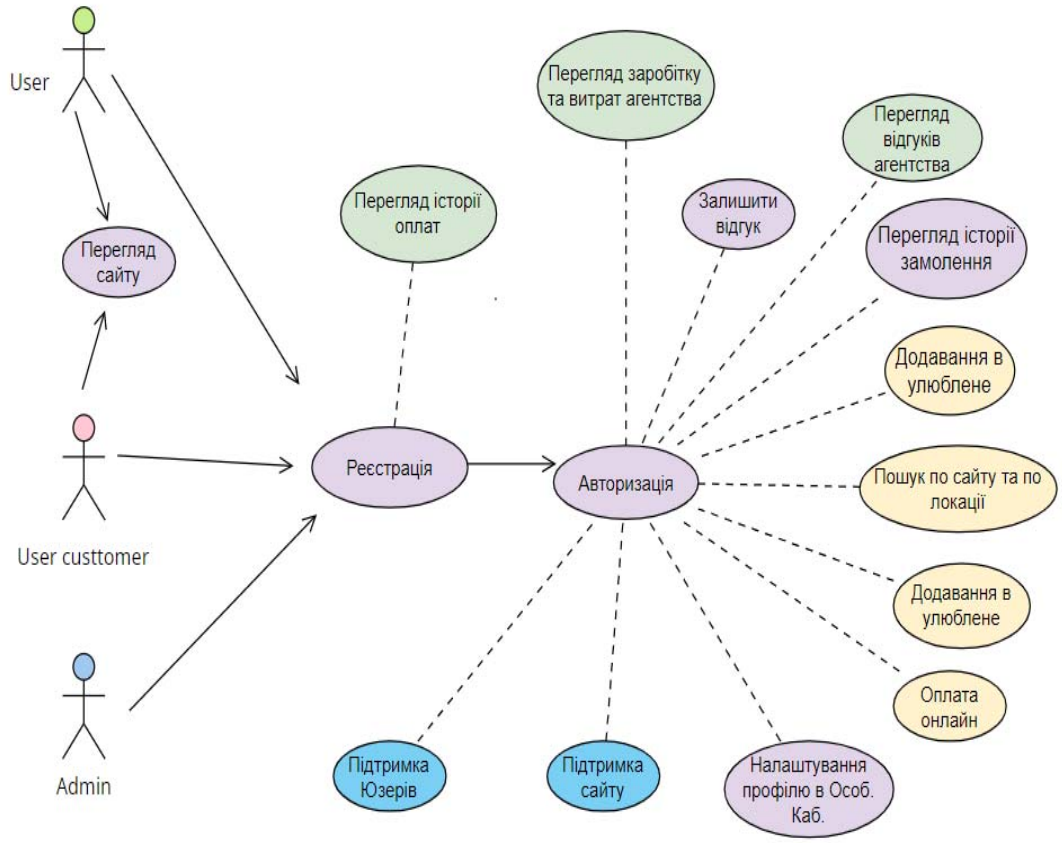

Рис. 1.5. Діаграма прецедентів взаємодії користувача з програмою

Наступним кроком у специфікації вимог до програмного продукту є розробка діаграми послідовності. Ця діаграма дозволяє виділити деякі об’єкти в системі та відобразити події основного сценарію програмного продукту, у вигляді послідовності повідомлень, якими обмінюються об’єкти. Діаграма послідовності зображена на рисунку 1.6.

Як видно з рисунку 1.6 користувач може виконати наступне:

1) реєстрація;
2) авторизація;
3) пошук;
4) завантаження інформації про рекламне агентство;

5) перегляд інформації “Наші роботи”;

6) замовлення власного бренду;

7) оплата онлайн;

8) вихід з системи.

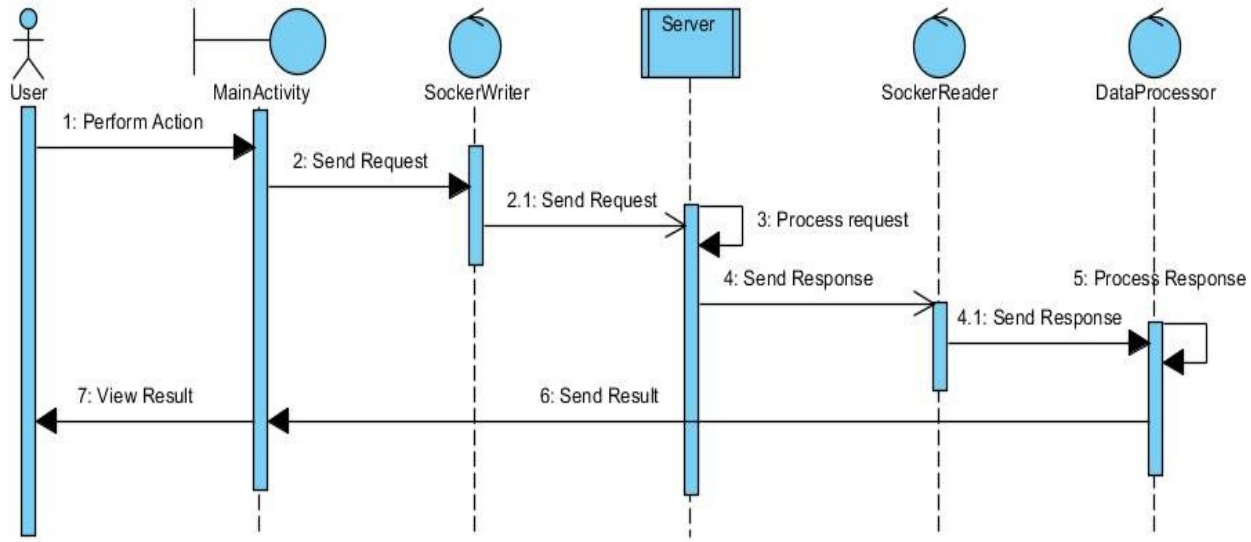

Рис. 1.6 – Діаграма послідовності

Розглянемо детальніше основні варіанти використання веб-сайту, які описані у наступних таблицях (таблиця 1.3-1.8).

Таблиця 1.3

Варіант використання «Реєстрація»

Контекст викор.	Реєстрація в системі
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Ввести url-адресу сайту рекламного агентства 2. Дочекатися з’єднання з сервером 3. Форма реєстрації 4. Заповнення необхідної інформації 5. Підтвердження реєстрації
Постумова	Відображається форма реєстрації в системі

Таблиця 1.4

Варіант використання «Авторизація»

Контекст викор.	Вхід в систему
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Форма авторизації
Постумова	Відображається форма входу в систему

Таблиця 1.5

Варіант використання «Пошук»

Контекст використання	Відображення основної інформації відносно діяльності ресторану
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Натиснути кнопку «Пошук» 4. Відображення результатів пошуку відповідно до сформованого пошукового запиту
Постумова	Відображення результатів пошуку

Таблиця 1.6

Варіант використання «Завантаження інформації про рекламне агентство»

Контекст використання	Завантаження вибраного агентства
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Натиснути кнопку «Переглянути інформацію про рекламне агентство» 4. Відображення інформації про ресторан
Постумова	Відображення інформації про рекламне агентстов

Таблиця 1.7

Варіант використання «Перегляд брендів»

Контекст використання	Перегляд брендів
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Натиснути кнопку «Переглянути бренди» 4. Відображення сторінки з брендами
Постумова	Відображення брендів

Таблиця 1.8

Варіант використання «Замовлення власного бренду»

Контекст використання	Перегляд інформації про замовлення власного бренду
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Натиснути кнопку «Замовлення бренду» 4. Відображення брендів
Постумова	Відображення альтернативних брендів

Таблиця 1.9

Варіант використання «Бронювання того чи іншого бренду»

Контекст використання	Бронювання бренду
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Натиснути кнопку «Забронювати бренд» 4. Відображення бренду 5. Забронювати бренд
Постумова	Відображення підходящого бренду

Таблиця 1.10

Варіант використання «Замовлення бренду»

Контекст використання	Замовлення бренду в рекламному агентстві
Дійові особи	Користувач
Передумова	Активне підключення до мережі Інтернет та успішне встановлення з’єднання з сервером
Тригер	Немає
Сценарій	1. Запустити браузер із введеною адресою веб-сайту 2. Дочекатися з’єднання з сервером 3. Натиснути кнопку «Переглянути бренди» 4. Відображення брендів 5. Замовити обраний бренд
Постумова	Відображення відображення брендів

Таблиця 1.11

Варіант використання «Вихід з системи»

Контекст використання	Вихід з ситеми для авторизованого користувача
Дійові особи	Користувач
Передумова	Успішне налаштування системи
Тригер	Немає
Сценарій	1. Відкрита сторінка з сайтом 2. Натиснути кнопку «Вихід» 3. Повернення на головну сторінку сайту
Постумова	Повернення на головну сторінку

Для створення розкадровки використаємо інструмент Balsamiq

Mockups – зручний програмний засіб для створення ескізів екранних форм. На рисунках 1.7 – 1.8 наведено ескізи основних екранних форм відповідно до сценаріїв використання.

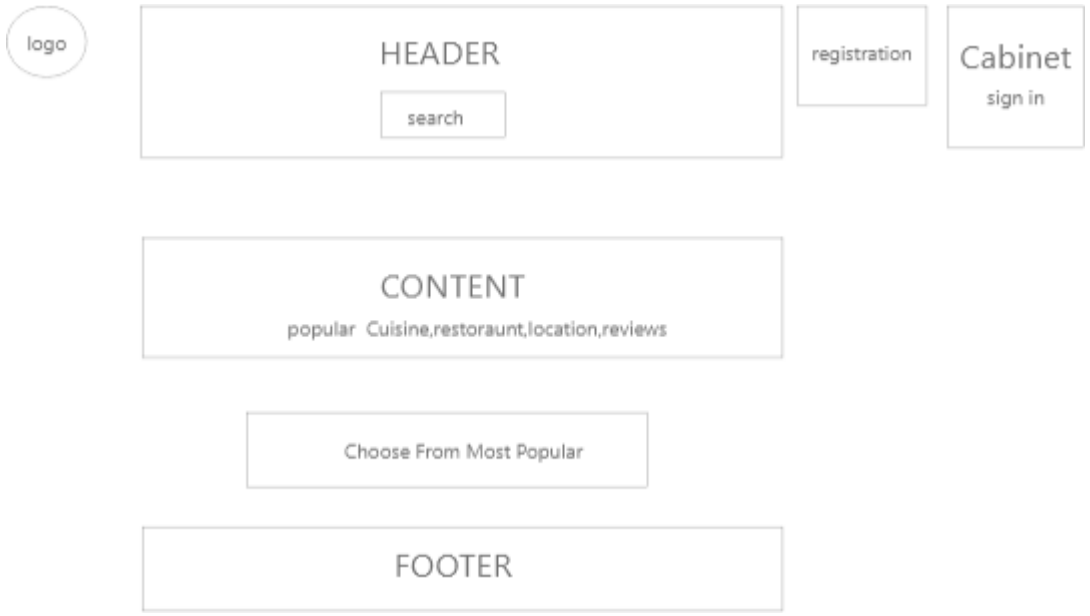

Рис. 1.7. Ескіз екранної форми головної сторінки

Веб-додаток має бути реалізована у вигляді сайту, доступного в мережі Інтернет. Сайт повинен складатися із взаємозалежних розділів із чітко розділеними функціями.

Адміністратору для підтримки сайту й експлуатації веб-інтерфейсу системи керування сайтом від персоналу не повинно вимагатися спеціальних технічних навичок, знання технологій або програмних продуктів, за винятком загальних навичок роботи з персональним комп'ютером і стандартним веб-браузером (наприклад, MS Internet Explorer 8.0, Google Chrome, Opera та інші).

Веб-додаток повинен складатися з наступних розділів:

− сторінка Restoraunt – містить інформацію про ресторан та

− сторінка PricePlan – містить цінову політику сайту.

− сторінка How it works – містить інформацію як це працює;

− сторінка Blog – містить контент сайту

Рис. 1.8. Ескіз екранної форми із розміщення головних інформаційних блоків головної сторінки ресторану

− сторінка FAQ – містить правила сайту та відповіді на питання;
− сторінка Contact – містить контакти сайту;
− особистий кабінет клієнта
− особистий кабінет ресторатора

Специфікація функціональних і нефункціональних вимог наведено утаблиці 1.12, 1.13

Таблиця 1.12

Специфікація функціональних вимог

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог		
		Пріоритет	Складність	Контакт
1.	Пошук	Обов’язкова	Низька	Користувач
2.	Завантаження інформації про рекламне агентство	Обов’язкова	Низька	Користувач
3.	Перегляд інформації про бренди	Обов’язкова	Середня	Користувач
4.	Бронювання бренда	Обов’язкова	Середня	Користувач
5.	Замовлення бренда	Обов’язкова	Середня	Користувач
6.	Перегляд історії замовлень	Обов’язкова	Низька	Користувач
7.	Перегляд відгуків	Обов’язкова	Низька	Користувач
8.	Зворотній зв’язок	Обов’язкова	Середня	Користувач
9.	Особистий кабінет	Обов’язкова	Середня	Користувач
10.	Додавати в улюблене	Обов’язкова	Середня	Користувач

Специфікація не функціональних вимог описана у таблиці 1.13.

Таблиця 1.13

Специфікація нефункціональних вимог

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Складність	Контакт
1. Застосовність				
1.1	Час, необхідний для навчання звичайних і досвідчених користувачів	Рекомендована	Низька	Користувач
1.2	Основні вимоги застосовності нової системи відносно інших систем, які знають користувачі	Рекомендована	Низька	Користувач
1.3	Вимоги по відповідності загальним стандартам застосовності та стандартам графічного інтерфейсу користувача	Обов’язкова	Низька	Користувач
2. Надійність				
2.1	Доступність	Обов’язкова	Середня	Користувач
2.2	Середній час безвідмовної роботи	Рекомендована	Середня	Користувач
3. Робочі характеристики				
3.1	Використання ресурсів	Рекомендована	Середня	Користувач

В даному веб-додатку має бути чітка та зручна для звичайного юзера навігація по сайту, перед тим як програмувати та створювати веб-сайт. По-перше необхідно створити такий дизайн, який враховує особливості сприйняття звичайного користувача, та виконати тестування розробки на людях які не пов’язані з IT-сферою. Таким чином буде краще добитися зручної та якісної навігації.

Значення нефункціональних вимог:

- час, необхідний для навчання звичайних користувачів – 30хвилин;
- час, необхідний для навчання досвідчених користувачів – 15 хвилин;
- основні вимоги застосовності нової системи відносно інших систем, які знають користувачі – всі функції системи є легкими у виконанні, а структура програми не відрізняється від існуючих аналогів;
- вимоги по відповідності загальним стандартам застосовності та стандартам графічного інтерфейсу користувача – система повинна працювати у всіх найпопулярніших браузерах ;
- доступність – час, що витрачається на обслуговування системи не повинен перевищувати 3% від загального часу роботи;
- середній час безвідмовної роботи – 3години;
- використання ресурсів – мінімальні системні вимоги:
  - 256 Мб пам’яті;
  - 10Мб вільного дискового простору;
  - Процесор з тактовою частотою 600МГц;
  - HTML, CSS та JavaScript;
  - Angular;
  - Node.js;

Висновки до розділу 1

Здійснено опис предметної області, напрями діяльності. Визначено склад функцій, що входять до бізнес-процесу на основі яких розроблено схему управління бізнес-процесом. Проведено аналіз відомих програмних систем. Здійснено аналіз вимог до програмної системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ВЕБ-САЙТУ

2.1. Розроблення архітектури програмної системи

З метою проектування архітектури веб-сайту рекламного агентства «Fantazər», здійснено загальне функціональне моделювання бізнес-процесів в закладах досліджуваного типу, де буде використовуватись автоматизована проектована система, відповідно до вимог стандарту IDEF0 [7].

Мета моделювання (Purpose): Швидко та якісно обслужити користувача сайту ;

Точка зору (ViewPoint): адміністратор сайту;

Визначення моделі (Definition): Це модель, яка описує бізнес-процес на сайті, де буде використовуватись автоматизована проектована система.

Область дії (Scope): аналіз, проектування, розробка, тестування та впровадження продукту. На рисунку 2.1 наведено контекстну діаграму процесу обслуговування користувача сайту.

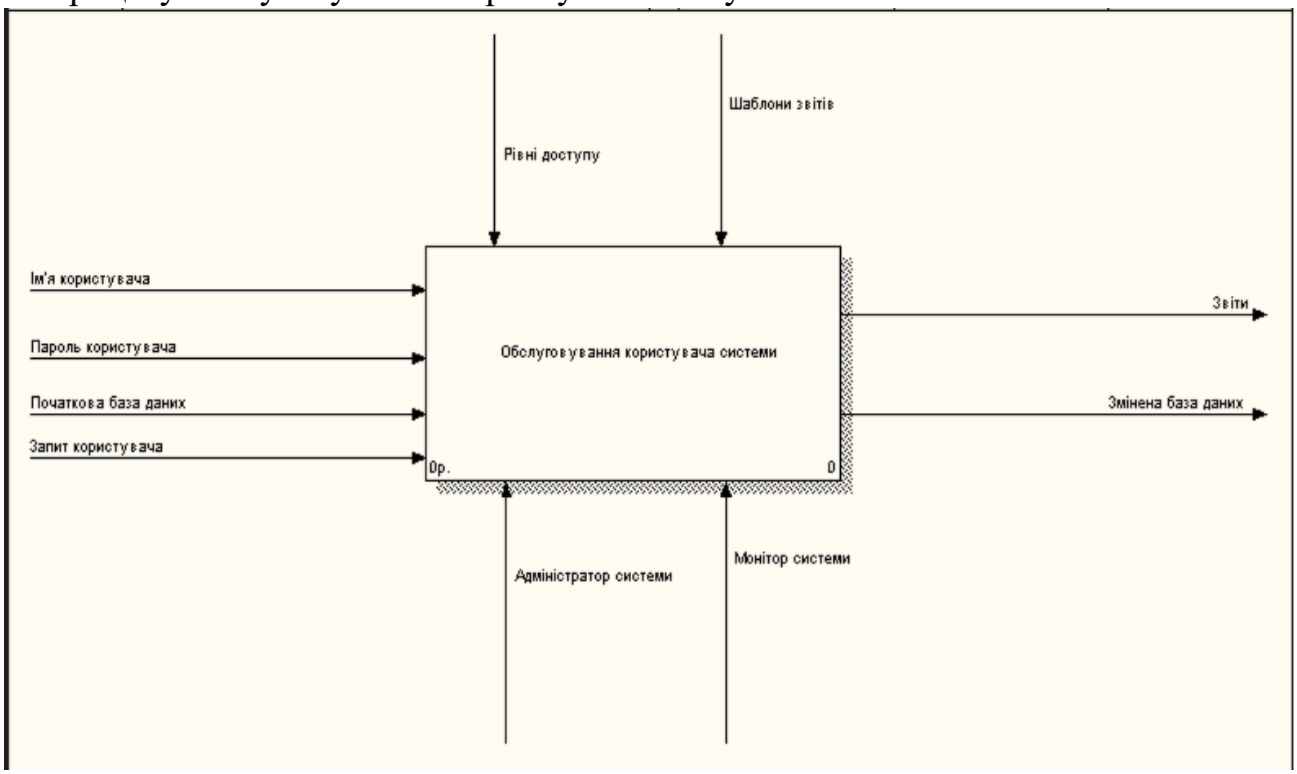

Рис. 2.1. Контекстна діаграма процесу обслуговування користувача рекламного агентства

Головний процес має зв'язок з зовнішніми сутностями. Стандарти якості обслуговування визначають основні вимоги та правила обслуговування користувачів. Автоматизована система обслуговування – це основний механізм в обслуговування відвідувачів сайту. Виконане замовлення є результатом виконання головного процесу.

На рисунку 2.2 наведено результат декомпозиції головного процесу.

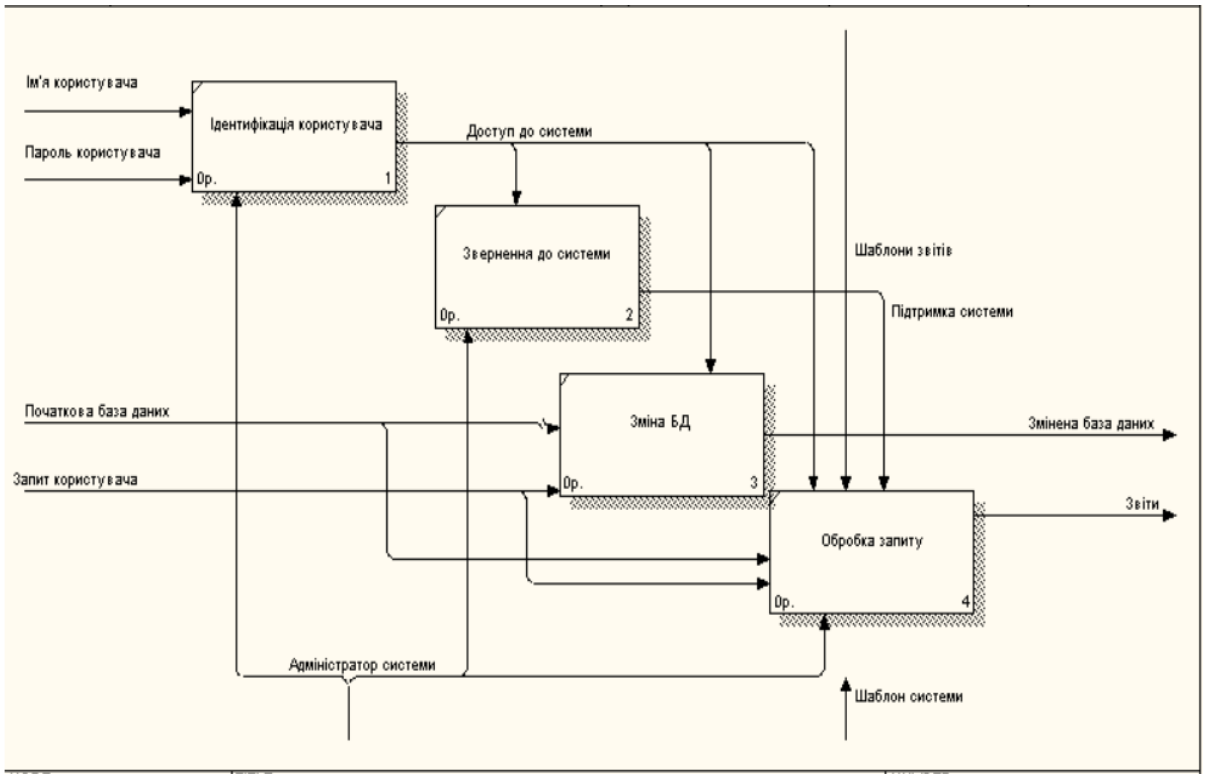

Рис. 2.2. Контекстна діаграма головного процесу

На рисунку 2.3 наведено результат декомпозиції процесу подання бренду. Він складається з таких процесів: перевірка на наявність ПЗ на пристрої, завантаження ПЗ, оновлення даних з серверу, перегляд сайту, формування та відправка замовлення на сервер.

Результатом процесу оновлення даних є оновлене сайту з серверу, що є вхідними даними для процесу перегляд сайту. Результатом процесу перегляду є бренди та логотипи , які користувач бажає замовити, що і є вхідними

даними до процесу формування і відправка замовлення на сервер. Вихідними даними є замовлення, сформоване клієнтом.

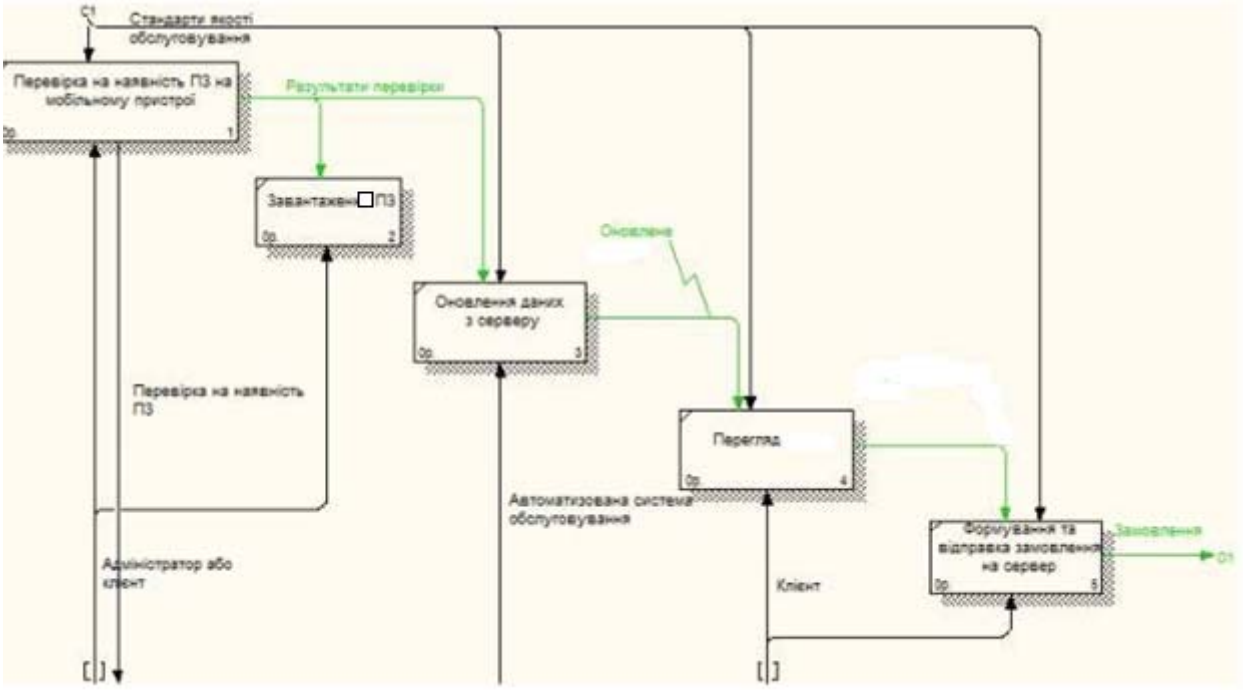

Рис. 2.3. Декомпозиція процесу «Подання Бренду»

На рисунку 2.4 наведено результат декомпозиції процесу обробки замовлення сервером. Він включає такі процеси: аналіз запиту сервером, генерація відповіді, передача замовлення на веб-клієнт.

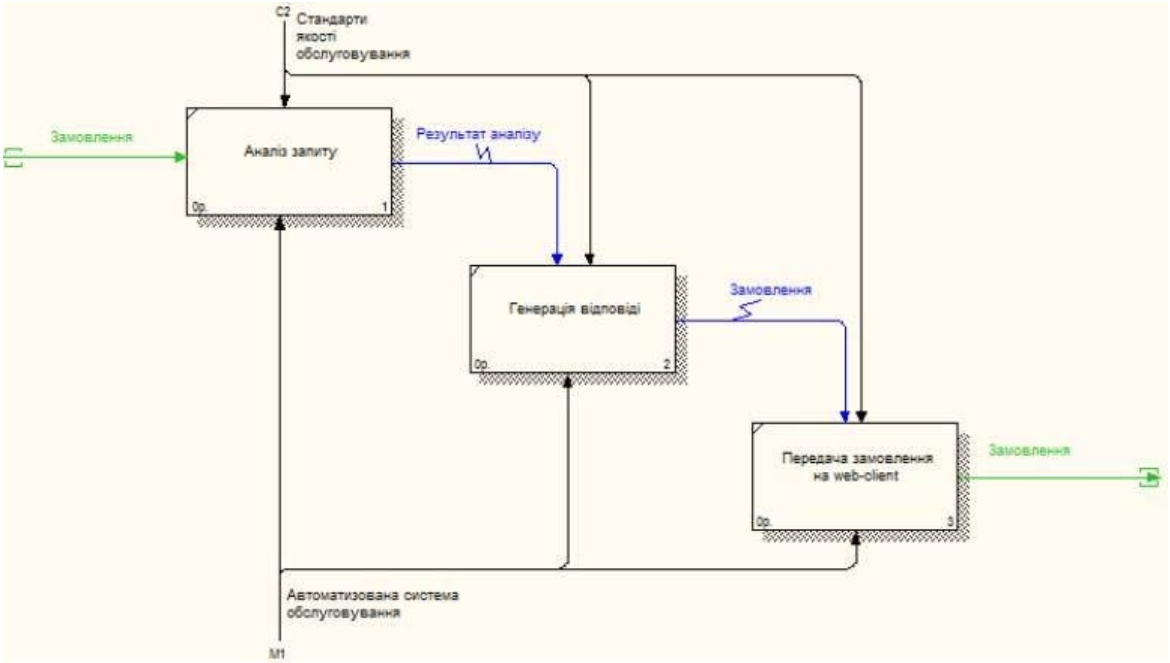

Рис. 2.4. Декомпозиція процесу обробки замовлення сервером

На рисунку 2.5 представлено результат декомпозиції процесу аналізу замовлення веб-клієнтом, який включає такі процеси: прийняття запиту з сервера, додання даних в БД, відображення даних на екрані адміністратора. Під час виконання процесу прийняття запиту з серверу, отримуються дані, які зберігаються у базі даних веб-клієнта, і відображаються на екрані монітора під час процесу відображення. Кінцем результатом процесу є замовлення користувача, готове для аналізу аналітиками сайту.

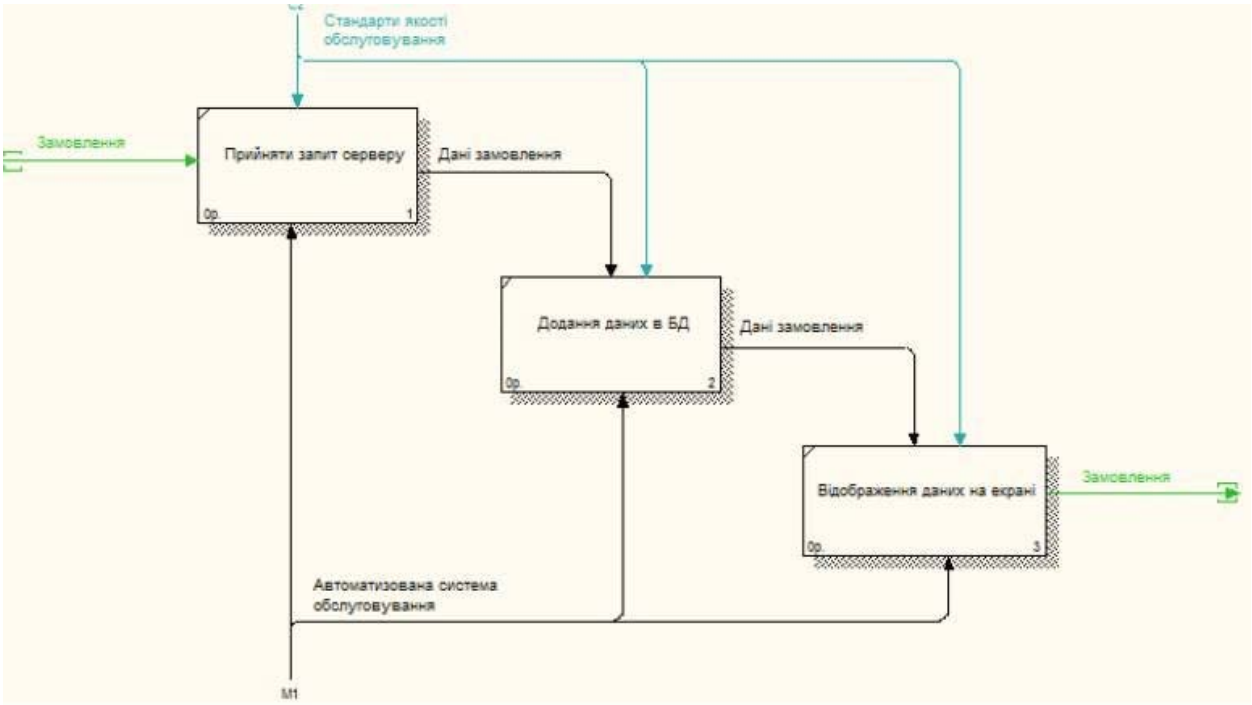

Рис. 2.5. Декомпозиція процесу аналізу замовлення веб-клієнтом

На рисунку 2.6 представлено узагальнену систему реєстрації нового клієнта через веб-сайт.

Пакет – це набір класів, який описує одну логічну одиницю системи. Оскільки проектування системи проводиться на високому рівні деталізації, то діаграми абстракцій можна вважати діаграмами реальних елементів [10]. Так діаграма абстракцій на рівні пакетів може бути використана для зображення діаграми пакетів системи.

Діаграма містить такі пакети як Activities, Adapters, DAOLayer, Layouts, QueriesToDB, QueriesToServer і ParserLayer.

Activities – даний пакет містить класи, які реалізують логіку взаємодії з користувачем, представляють веб-інтерфейс. Дані класи представляють сторінки програми: MainActivity – головна сторінка системи, SectionBrand – сторінка для відображення брендів, DishActivity – сторінка інформації про конкретний бренд, OrderActivity – сторінка для

відображення/редагування, PayOrderActivity – сторінка для оплати замовлення користувачем сайту.

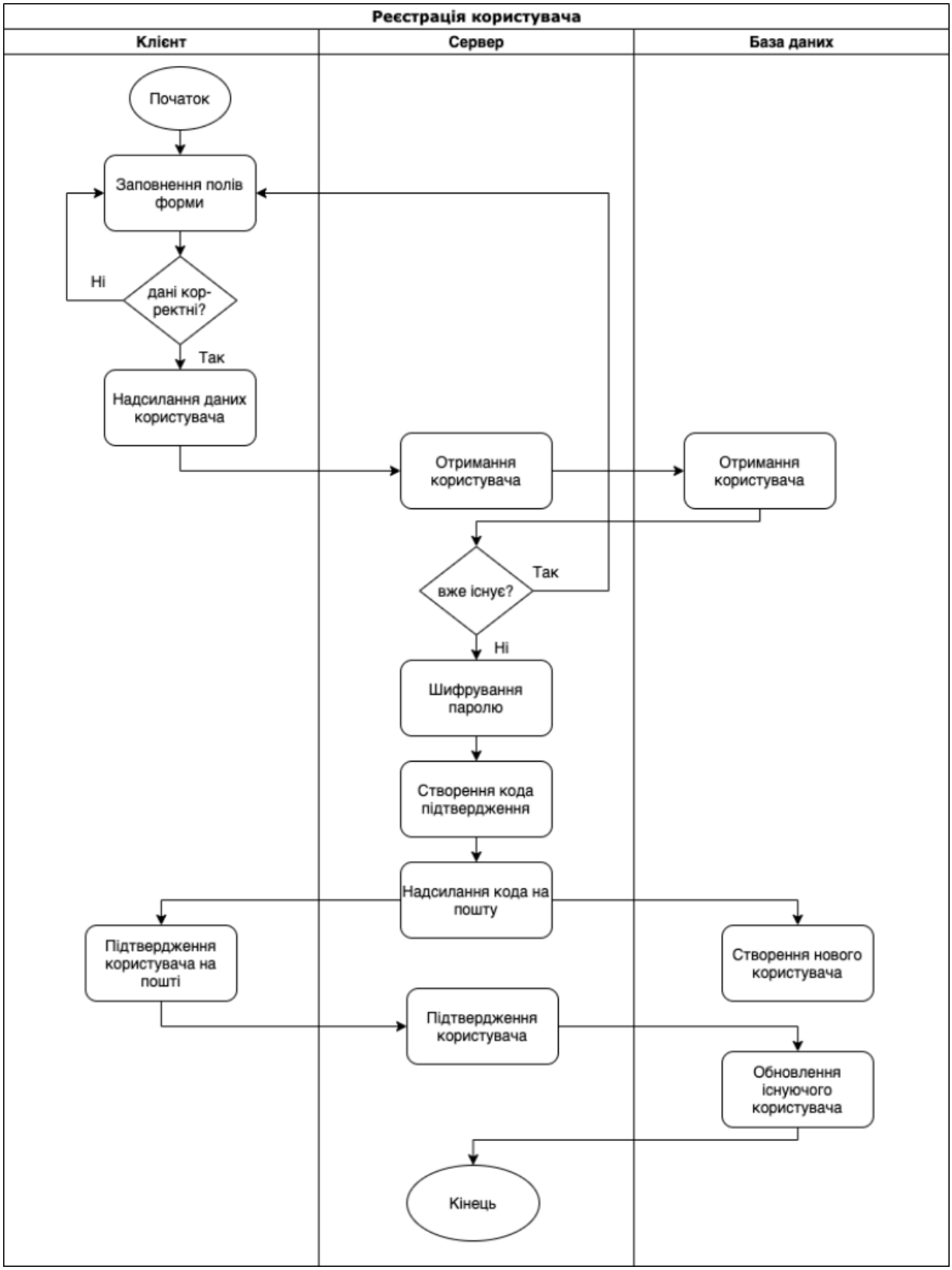

Рис. 2.6. Схема реєстрації нового клієнта через веб-сайт

Adapters – даний пакет містить класи-адаптери, які використовуються для зв’язку інтерфейсу та даних.

Layouts – даний пакет містить класи, які представляють сторінки графічного інтерфейсу користувача.

QueriesToServer – містить класи, які реалізують зв’язок з сервером, представляють різні запити, метою виконання яких є завантаження даних – клас, оновлення - клас GetData, відправка замовлення клієнта - SendBrand.

Клас CheckVersionData – запит, який здійснюється для перевірки актуальності даних на клієнті, тобто перевіряє версію даних. QueriesToDB – містить класи, які реалізують з’єднання з відповідною базою даних та виконання різних запитів, які стосуються вибірки чи оновлення.

ParserLayer – даний пакет містить класи, які виконують аналіз результатів виконання запитів з сервером, виконують парсинг даних.

DAOLayer – Data Access Layer – даний пакет містить класи, які здійснюють зв’яз ок з запитами до БД та логікою програми. Повертає вибрані набори даних з БД у потрібних структурах даних.

Далі розглянемо діаграму ключових абстракцій (рисунок 2.7). З діаграми видно, що ключовими абстракціями є замовлення, клієнт, категорії брендів, бренд, пункт замовлення. На діаграмі відображені різні зв’язки між абстракціями: асоціація, агрегація строга та нестрога.

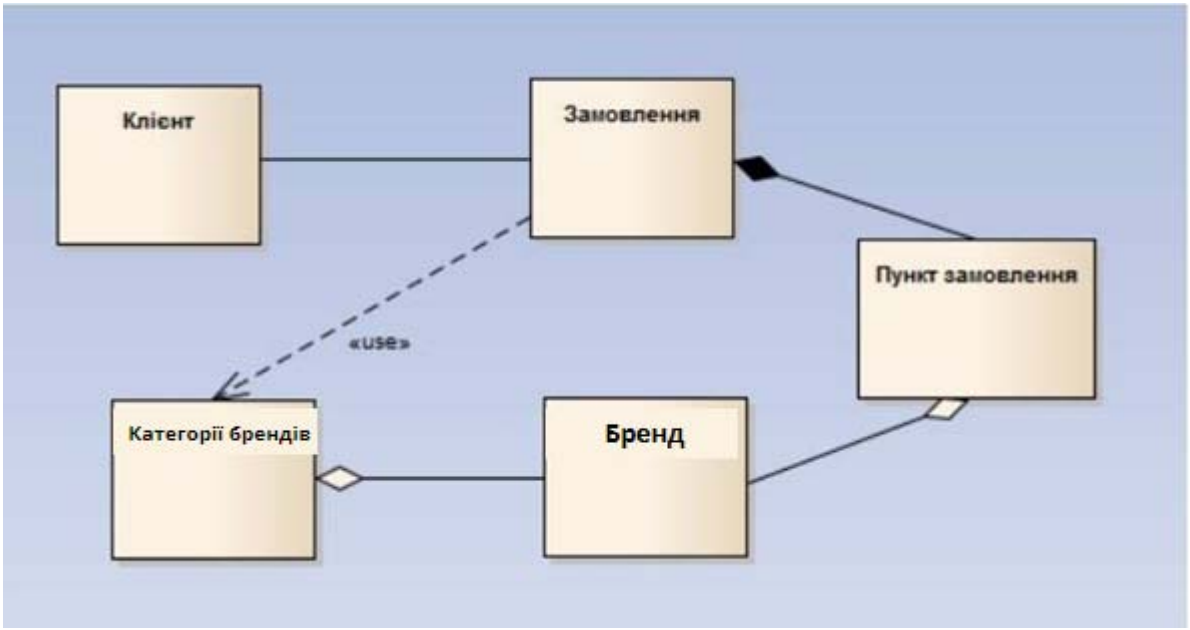

Рис. 2.7. Діаграма ключових абстракції

Між абстракціями «Замовлення» та «Пункт замовлення» присутня строга агрегація, яка означає, що «Замовлення» містить в собі «Пункт замовлення».

Між сутністю «Категорії брендів» та «Бренд» присутня нестрога агрегація, яка означає, що «Категорії брендів» використовує об’єкти сутності «Бренд». Основною відмінністю між строгою та нестрогою агрегацією є те, що якщо видалити об’єкти «Замовлення», то одночасно і видаляться об’єкти

«Пункт замовлення», у випадку нестрогої агрегації, якщо видалити «Категорії брендів», то категорія «Бренд» залишиться.

На рисунку 2.8 представлено діаграму послідовності варіанту використання «Складання замовлення». З діаграми видно, що користувач взаємодіє через різні Activities – сторінки сайту. Для відображення даних на екрані, клас з пакету Activities дістає відповідний йому графічний інтерфейс з класу Layouts, звертається для отримання даних, визиває адаптер для зв’язку даних з графічним інтерфейсом, після чого відображає дані на екрані.

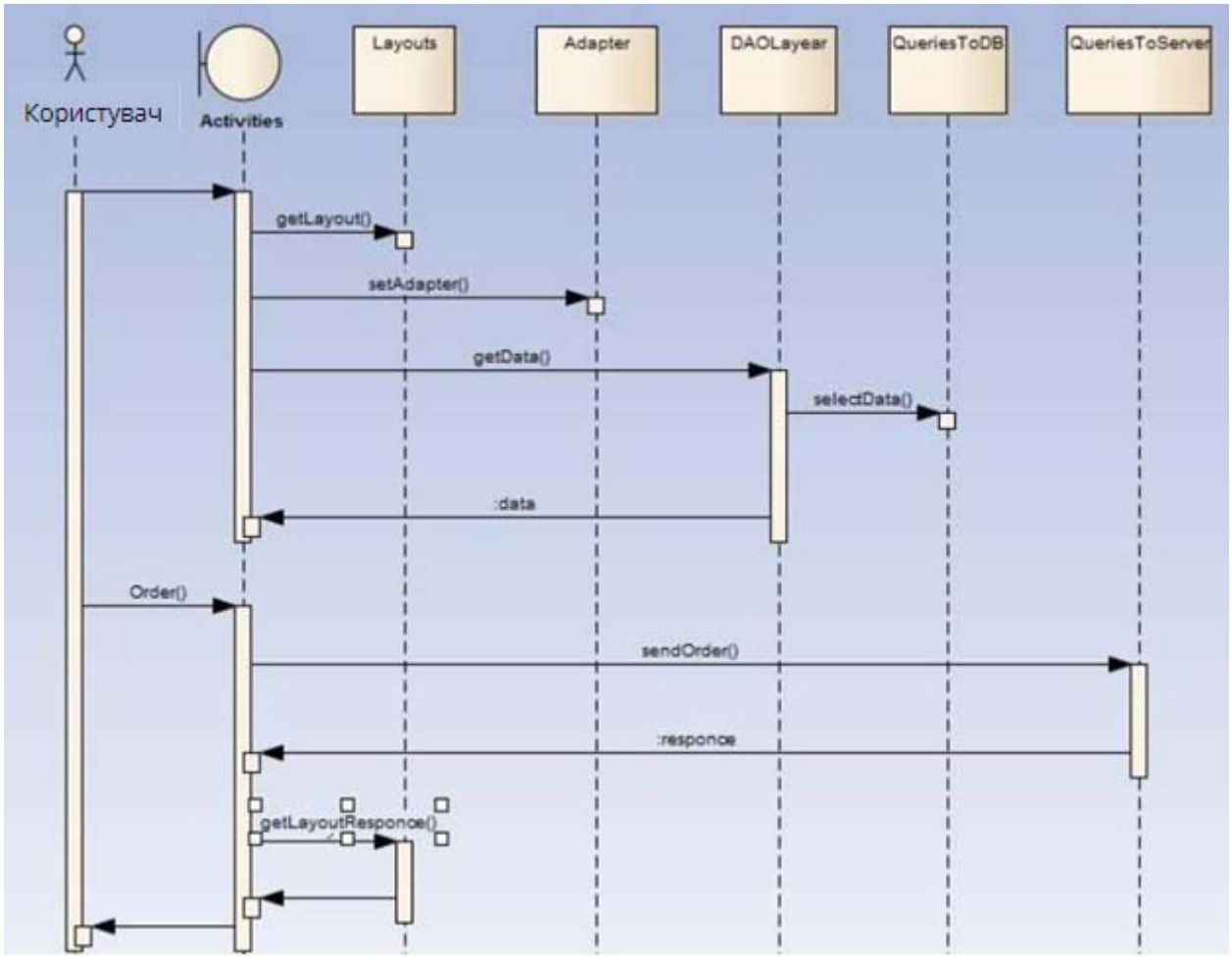

Рис. 2.8. Діаграма послідовності варіанту використання «Складання замовлення»

Далі, взаємодіючи через активіті, клієнт складає замовлення. Клас з пакету Activities викликає клас з пакету QueriesToServer, для передачі замовлення на сервер. В залежності від отриманої відповіді з серверу, вибирається відповідний діалог для відображення повідомлення з пакету Layouts. Після того як користувач побачить відповідь про прийняття його замовлення, варіант використання завершується.

На рисунку 2.9 наведено алгоритм для варіанту використання «Відновлення пароля».

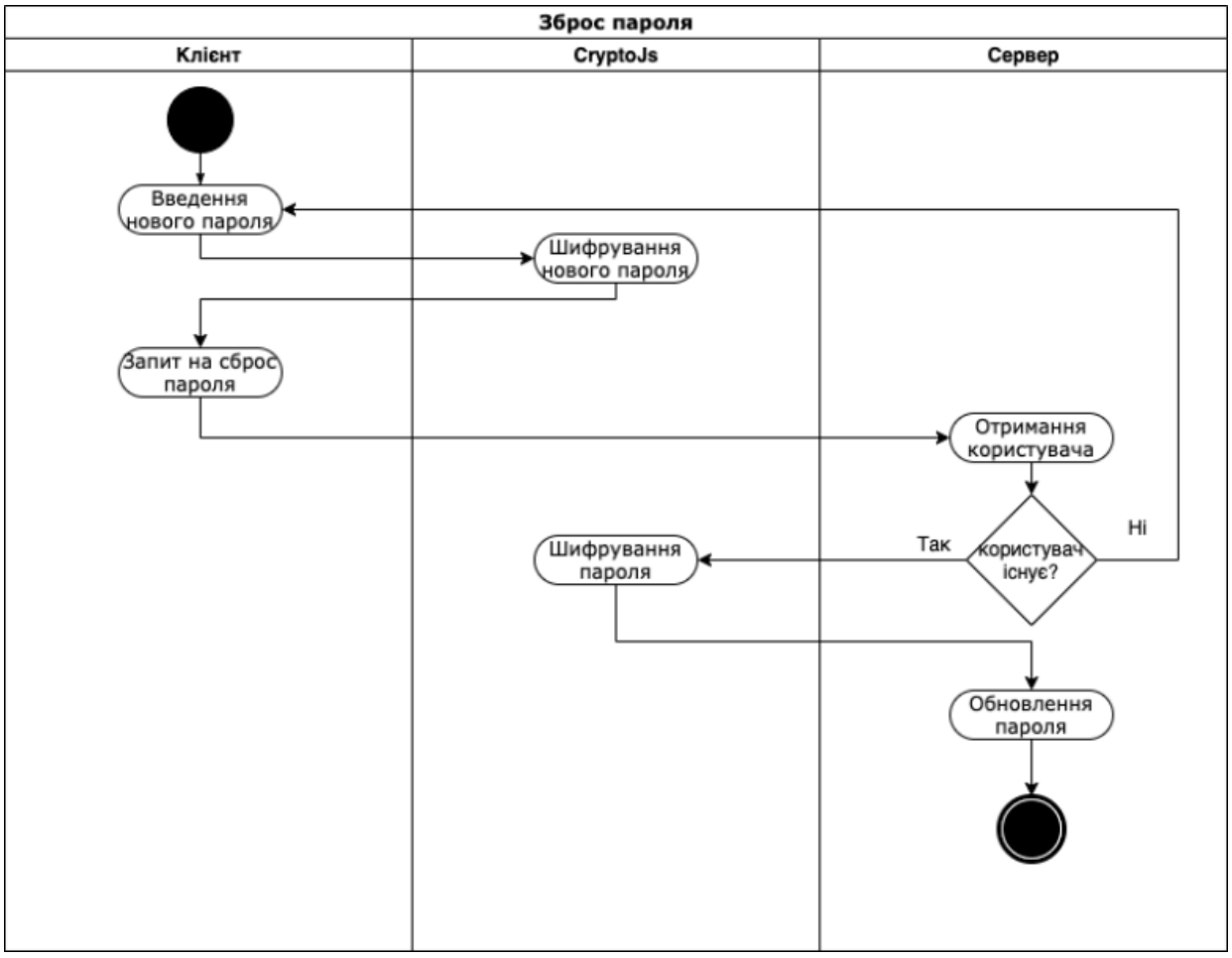

Рис. 2.9. Алгоритм встановлення нового пароля

2.2. Особливості реалізації джерела даних

Дані веб-додатку необхідно систематизовано зберігати і обробляти, для чого і використовується база даних. Перш ніж вибрати систему управління базою даних (СУБД), слід розглянути передумови для вибору такої.

З огляду на бізнес-правила, наприклад, що при статусі замовлення "затверджений", його заборонено редагувати і особливості БД, сама часто

використовувана операція при використанні СУБД - читання даних, потім - додавання.

Так як типи продуктів мають ієрархічну структуру і документи можуть кілька відрізнятися один від одного деякими атрибутами, в залежності від типу, а також що вимоги можуть бути змінені, тягти за собою можливу зміну атрибутів, було прийнято рішення про вибір СУБД, де є підтримка неструктурованої моделі даних - NoSQL [10].

На підставі передумов і вимог, було зроблено висновок, що необхідна СУБД, яка пропонує:

- Підтримку NoSQL.
- Можливість горизонтальної масштабованості, зокрема реплікації БД.

Плюсом буде підтримка даних типу JSON, так як JSON обраний в якості формату обміну даними між клієнтом і API. А також поширеність і документація.

На підставі сказаного, як оптимальне рішення була обрана СУБД MongoDB. MongoDB - документно-орієнтована СУБД [11]. Зберігає дані в форматі BSON, що являє собою бінарну форму Представлення JSON. Таким чином, запис в MongoDB є аналогом об'єктного представлення у форматі JSON.

MongoDB має інтерфейс для виконання операцій на мові JavaScript, можливості асинхронної реплікації, підтримує необхідні типи даних, які можуть знадобитися при проектуванні БД згідно предметної області, наприклад, масив, дата, об’єкт та інші. Раціонально підходить для зберігання ієрархічних даних [12].

2.3. Загальне представлення веб-додатку

Додаток використовує трирівневу архітектуру. Комунікація клієнтського додатка, сервера додатків і СУБД, здійснюється текстовими

структурованими даними в форматі JSON, рисунок 2.10. Логіка клієнтської і серверної частини реалізуються на мові JavaScript.

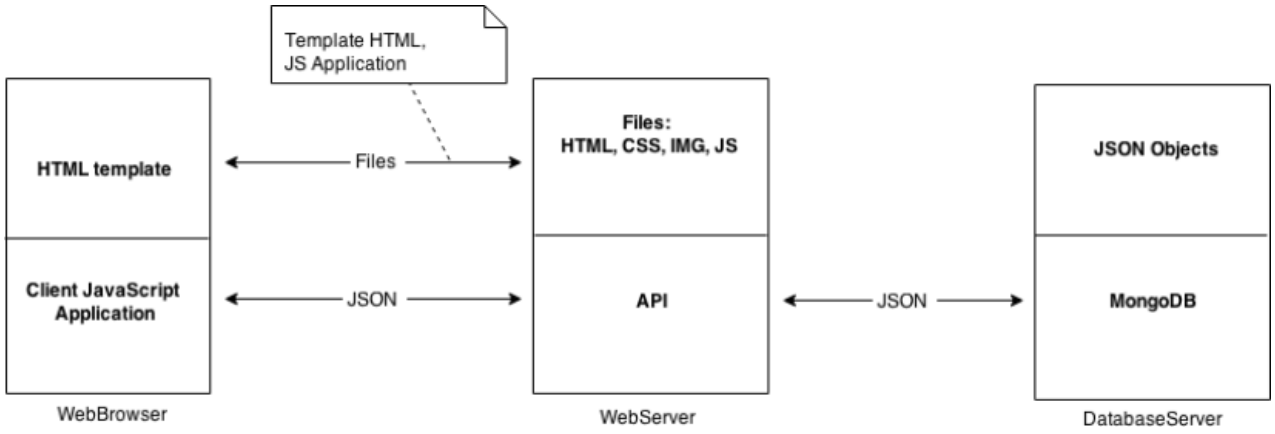

Рис. 2.10 - Обмін даними в форматі JSON в трирівневій архітектурі

Додатки кожного рівня використовують об'єктне представлення JSON. Примірник сутності предметної області на всіх трьох рівнях є JSON-об'єкт, приклад наведено в таблиці 2.1.

Таблиця 2.1.

Використання JSON-об'єкта на трьох рівнях

Клієнтський додаток	{ "document_type" : "Brand", "file" : { "name" : "Winter brand.pdf", "hash" : "9b142be8b7707a2bfb8efd3acf452877" }, "products" : [{ "product_type" : "Brand and Orders", "product_subtype" : "Orders", "product" : "Single Orders" }] }
Серверний додаток	
База даних	

Протокол транспортного рівня отримує дані від представницького рівня, відповідно до моделі OSI, в текстовому (бінарному) форматі, саме

тому при отриманні цих даних їх необхідно привести до об'єктного, щоб програми використовували їх як об'єкти JSON. Щоб отримати об'єктний вид досить скористатися JavaScript-функціями перетворення з строкового виду до об'єктного, - операція десеріалізації.

І серверна частина, і клієнтська частина реалізують бізнес-логіку, однак дана відповідальність була розділена за принципом звернення до БД. У процесі отримання бренду, це опорні точки які передбачають виконання запитів до серверної частини, зокрема до API:

- Надіслати критерії пошуку, щоб отримати список страв.
- Надіслати запит на замовлення.
- Надіслати запит на уточнення статусу.
- Надіслати дані замовлення.

Вся інша логіка при отриманні документа-бренду реалізується на клієнтській частині. Перший запит клієнта до сервера є не що інше, як отримання клієнтського додатку. Після чого він виконується на клієнті, при необхідності запитуючи дані у API серверної частини. API має стандартизований інтерфейс для запитів відповідно до REST. API відображає модель предметної області.

В принципі, всі дані документів можуть бути завантажені разом з першим запитом. Це дозволить не виконувати запит на пошук документів на сервері, пропустивши першу опорну точку, а отримати ці дані з локального сховища клієнта. Однак такий підхід викличе довге очікування завантаження даних відповідно до призначеного для користувача сприйняття, що створить проблему UX. Тому цей варіант прийнятий не був. Для зменшення часу відгуку серверної частини, було вирішено визначити відповідальність за надання статичних файлів клієнтської частини проксі-сервера, рисунок 2.11. Нагадаємо, що в якості проксі-сервера виступає nginx і оптимальним чином виконує цю роботу.

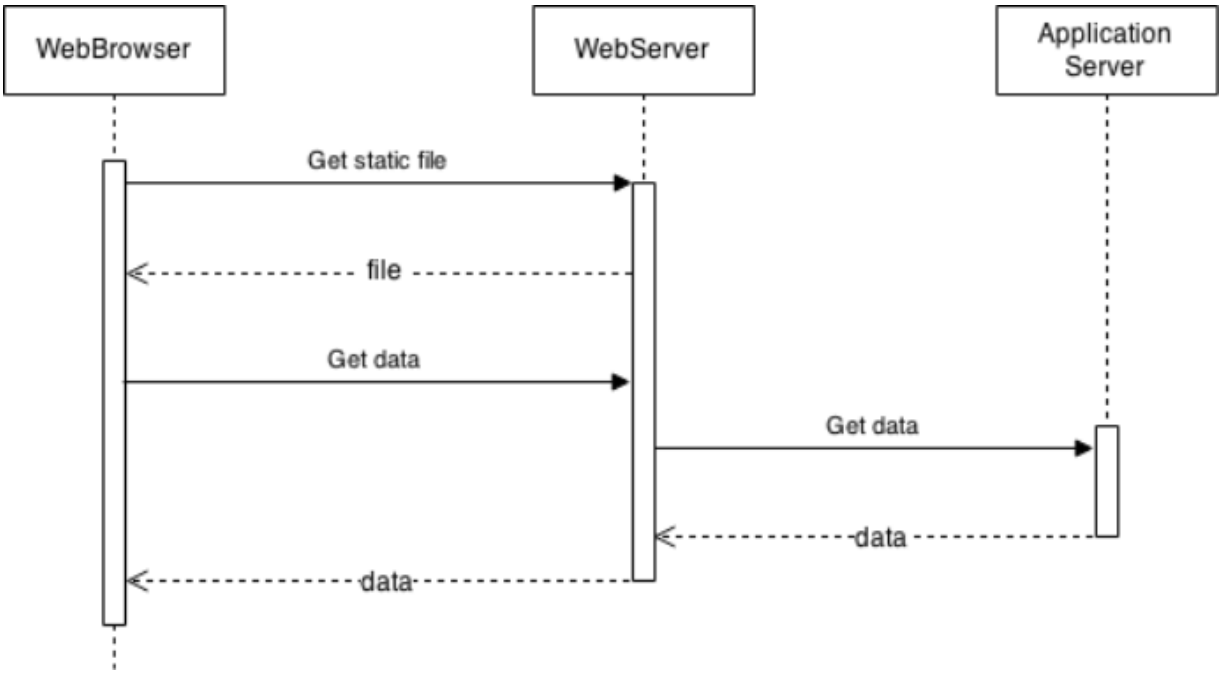

Рис. 2.11 - Опрацювання запитів проксі-сервером

При запиті статичного файлу, завдання виконується проксі-сервером, в іншому випадку запит делегується серверу додатку. Нижче наведена частина коду клієнтського додатку, в якій описаний запит до серверної частини, де створюється об’єкт замовлення.

Клієнтська частина:

```
var order = new Order({
address : $scope.cart.address,
items : items,
user_name: $scope.user_name,
user_id: $scope.user_id
})
order.$save({}, function success(data) {
$scope.cart.clear();
$location.path('/order/' + data.id);
}, function err() {
flash.error = getErrorMessageOrderSave();
})
```

Серверна частина:

```
APIRouter.route('/order')
.post(function(req, res, next) {
var order = new Order(req.body);
order.save(function(err) {
if (err) {
next(err);
} else {
order.sendEmail();
res.json({id: order._id});
}
})
})
OrderSchema.methods.sendEmail = function() {
if (this.address.email) {
var email = new OrderEmail({order: this});
email.send();
}
}
```

На рисунку 2.12, представлено розташування фізичних компонентів системи.

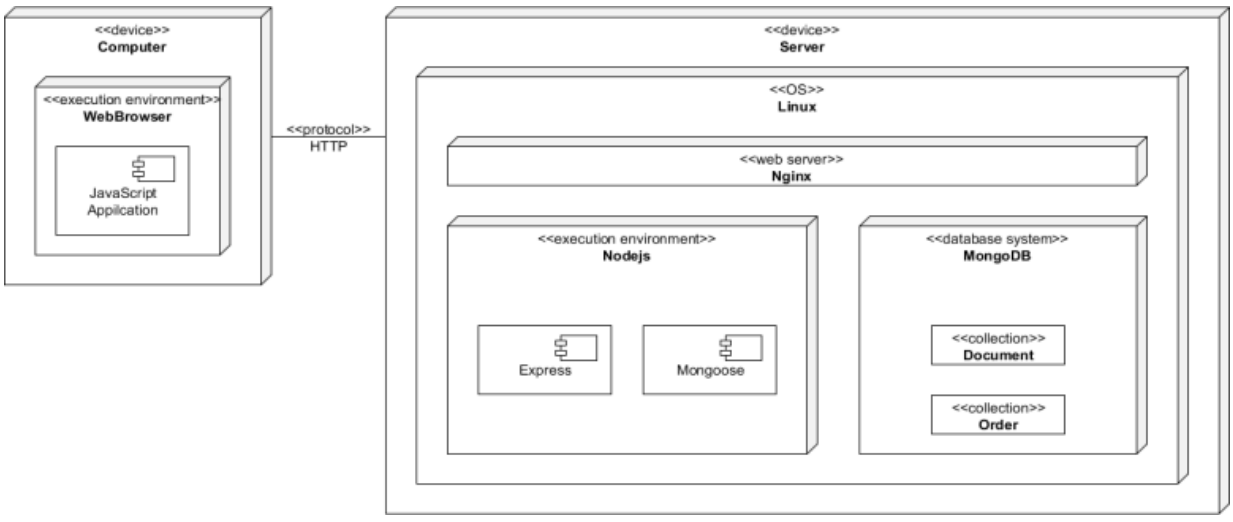

Рис. 2.12. Діаграма розгортання, фізичні компоненти веб-додатку

Висновки до розділу 2

У цьому розділі розроблено архітектуру програмного додатку, що дозволить краще зрозуміти функції основних його частин. Створено та описано структурну схему, основними компонентами якої є: рівень клієнта, рівень бізнес-логіки та рівень даних. Описано функціональну структуру системи та її основних елементів – модулів обробки даних, спроектовано структуру бази даних.

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-САЙТУ ДЛЯ РЕКЛАМНОГО АГЕНТСТВА «Fantazər»

3.1. Особливості програмної реалізація

Як і більшість сучасних веб-сайтів, веб-додаток побудований як Singlepage Application (SPA) - це додаток, якому не потрібно перезавантажувати сторінку під час її використання та працює в браузері (Facebook, Карти Google, Gmail, Twitter, Google Drive або GitHub).

Основними перевагами SPA-додатків є:

- швидкість та адаптивність;
- можливість кешування;
- лінійний досвід користувачів;
- налагодження за допомогою Chrome;

SPA-архітектура досить проста - вона складається з клієнтських технологій (у даному випадку React.js та Angular) та серверних технологій (Node.js).

Щоб створити односторінковий додаток, необхідні AJAX та HTML5 для створення адаптивних сторінок, в той час як Angular, React відповідають за обробку даних на клієнтській базі SPA.

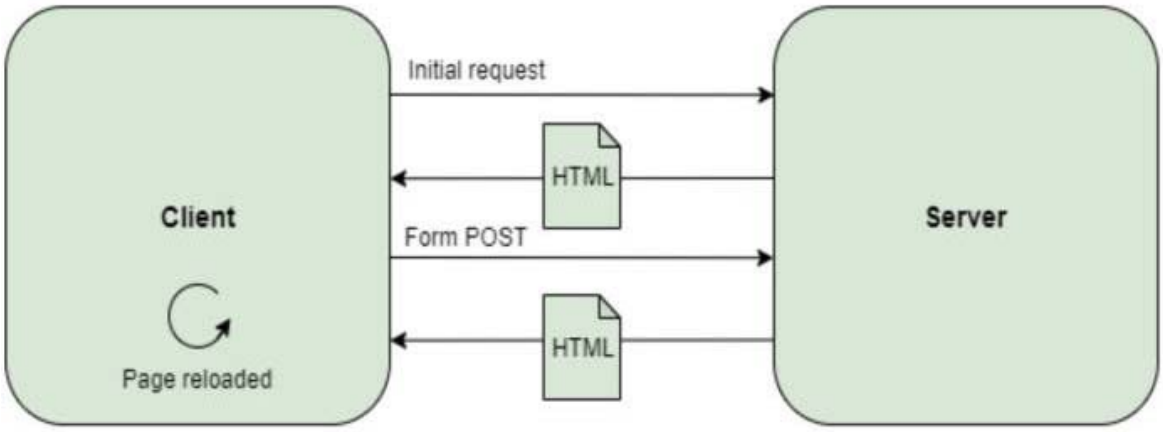

Рис. 3.1. Single Page Application (SPA)

Як тільки користувач отримує доступ до сторінки та виконує будь-які дії на цій сторінці, сторінка перезавантажується змінами, які були зроблені на стороні сервера і користувач отримує майже миттєву реакцію зі сторінки.

В основі у веб-додатку використаний REST API архітектурний підхід.

Перевагами цього підходу є [3]:

1) Розмежування між клієнтом і сервером. Протокол REST повністю відокремлює інтерфейс користувача від сервера та зберігання даних. Це має деякі переваги при розробці. Наприклад, це покращує портативність інтерфейсу для інших типів платформ, збільшує масштабованість проектів і дозволяє самостійно розвиватися різні компоненти розробок. Це особливо важливо, так як веб-додаток має два окремих клієнт сервісів: для звичайних користувачів та для адміністратора. Оба клієнта повинні вміти працювати з сервером і по однаковому інтерфейсу.

2) Надійність та масштабованість. Розмежування між клієнтом і сервером має одну очевидну перевагу, і це те, що кожна команда розробників може масштабувати продукт без особливих проблем. Вони можуть мігрувати на інші сервери або вносити всі види змін у базу даних за умови коректного надсилання даних із кожного запиту. Розділення полегшує наявність клієнтської та серверної сторін на різних сервісах, а це робить додатки більш гнучкими для роботи.

3) API REST завжди не залежить від типу платформи або мов API REST завжди адаптується до типу синтаксису або платформ, що використовуються, що дає значну свободу при зміні або тестуванні нових середовищ в рамках розробки. З API REST ви можете мати сервери PHP, Java, Python або Node.js. Єдине, що обов'язково, щоб відповіді на запити завжди відбувалися мовою, що використовується для обміну інформацією, як правило, XML або JSON.

Система REST складається з:

1) клієнт, який запитує ресурси;

2) адмін-клієнт, який запитує ресурси;

3) сервер, який має ресурси.

Існує декілька HTTP-методів для виконання відповідних дій, а саме: GET, PUT, POST, DELETE, UPDATE, PATCH.

Клієнтська частина використовує HTTP-методи для зміни ресурсів, які обробляються на серверній частині. У веб-додатку використано наступні методи:

- GET - використовується для запиту даних із визначеного ресурсу.

- POST, PUT - використовується для надсилання даних на сервер для створення / оновлення ресурсу.

- DELETE - використовується для видалення вказаного ресурсу.

PUT використовується в основному для оновлення записів. POST - це спосіб подання даних, пов'язаних із заданим URI.

Клієнт. Оскільки веб-додаток має CSR, в ньому використано наступні основні найпопулярніші технології:

- Мова розмітки гіпертексту (HTML) та каскадні таблиці стилів (CSS). HTML повідомляє браузеру, як відображати вміст вебсторінок, а CSS - це вміст.

- JavaScript. JS робить веб-сторінки інтерактивними.

Але чисту мову Javascript (vanilla Javascript) все рідше і рідше використовують при створенні сучасних веб-сайтів. Причиною цьому - швидкий розвиток фреймворків для цієї мови, таких як Angular, React, Vue, jQuery, Backbone, Ember та інші. Кожен з них має свої переваги. Найбільш популярними на сьогоднішній день являються Angular, React та Vue. Всі три бібліотеки є взаємозамінними. У клієнт-додатку використано найбільш популярний фреймворк, створений компанією Facebook - React.

Facebook, Uber, Instagram та WhatsApp були створені за допомогою цього фреймворку. Розглянемо основні переваги React.js [4]:

1) Легкий загальний процес написання компонентів. JSX - необов'язкове розширення синтаксису до JavaScript, що значно спрощує написання власних компонентів. Він приймає котирування HTML і полегшує візуалізацію підкомпонентів.

2) Висока продуктивність і легке подальше обслуговування. React має можливість повторного використання системних компонентів. Повторне використання забезпечує послідовний вигляд програми та полегшує підтримку та зростання бази даних коду.

3) Швидка візуалізація. Команда розробників Facebook представила Virtual DOM - на даний момент одна з переваг використання React для важких та динамічних програмних рішень. Це віртуальне представлення об'єктної моделі документа, тому всі зміни спершу застосовуються до віртуальної DOM, а потім, використовуючи алгоритм diff, обчислюється мінімальний обсяг необхідних DOM-операцій, після чого реальне дерево DOM оновлюється відповідно, забезпечуючи мінімальний витрачений час. Цей метод гарантує кращу роботу користувачів та більш високу продуктивність програми.

4) Стабільний код. React використовує лише низхідний потік даних. Змінюючи об’єкт, розробники просто змінюють його стан, вносять зміни, і після цього будуть оновлюватися лише окремі компоненти. Ця структура прив'язки даних забезпечує стабільність коду та постійну продуктивність програми.

5) Корисний набір інструментів для розробників. React Developer Tools - це розширення для браузера, доступне для різних браузерів (Chrome, Firefox). Це дає можливість розробникам спостерігати ієрархії реактивних компонентів, виявляти дочірні та батьківські компоненти та перевіряти їх поточний стан та реквізити.

6) Велике ком'юніті та розвинена екосистема. React GitHub репозиторії налічує понад 1100 учасників, користувачі можуть задавати свої запитання

щодо переповнення стека, форуму обговорень, платформ соціальних медіа та багатьох інших.

Щоб встановлювати, видаляти та керувати всіма інструментами, пакетами, фреймворками та бібліотеками в веб-додатку, потрібно мати менеджер пакунків. Існує багато їх видів, але виділимо два найбільш популярні: npm та yarn. Вони встановлюють пакети, які використовуються додатку та надають корисний інтерфейс для роботи з ними. Розглянемо їх відмінності [5]:

1) Файл блокування (.lock-file). npm: генерується файл "package-lock.json". Файл package-lock.json є трохи складнішим ніж yarn.lock. Завдяки цій складності, пакетний замок буде генерувати ту саму папку node_modules для різних версій npm. Кожна залежність матиме точний номер версії, пов’язаний з нею у файлі блокування пакунків.

- yarn: генерується файл "yarn.блок". Файли блокування yarn допомагають легко зливатися. Злиття передбачені також через дизайн файлу блокування.

2) Вихідний журнал:

- npm install: npm створює масивні журнали виводу команд npm.

- yarn add: журнали виводу чисті, візуально відрізняються та короткі. Вони також відсортовані у вигляді дерева.

3) “why”- команда

- npm: npm ще не має вбудованої функціональності "why".

- yarn: yarn поставляється з командою "why", яка говорить про те, чому залежність присутня в проекті. Наприклад, це залежність, власний модуль або проектна залежність.

4) Перевірка ліцензій:

- npm: npm ще не має перевірки ліцензій, який може дати зручний опис всіх ліцензій, з якими пов'язаний проект, через встановлені залежності.

- yarn: yarn має акуратну перевірку ліцензій. Щоб переглянути їх, введіть команду: yarn licenses list

5) Отримання пакетів:

- npm: npm вибирає залежності з реєстру npm під час кожної команди 'npm install'.

- yarn: yarn зберігає залежності локально і витягується з диска під час команди «yarn add» (припускаючи, що залежність присутня локально).

6) Процедура установки:

- npm: npm встановлюється за допомогою Node автоматично.

- yarn: для встановлення yarn npm повинен бути встановлений.

Виконайте команду npm install yarn --global

Незважаючи на переваги yarn пакетного менеджера над npm, у використанні вони дуже схожі та не мають суттєвих відмінностей. Для встановлення та керування пакетами обрано пакетний менеджер - yarn.

React побудований так, щоб компоненти могли внутрішньо керувати своїм станом без необхідності зовнішньої бібліотеки чи інструменту. Це добре для додатків з малою кількістю компонентів, але в міру того, як програма збільшується, управління станами, що поділяються між компонентами, стає все більш складним та неможливим. Для вирішення проблеми керування внутрішнього стану компонент, використано контейнер стану Redux.

Redux - це єдине місце у додатку, яке вміщує весь стан програми. Кожен компонент може отримати доступ до збереженого стану без необхідності надсилати дані з одного компонента в інший.

Розберемо роботу Redux [6]:

1) У будь-якому місці програми (включаючи компоненти) викликається певна дія: store.dispatch(action), де action - це звичайний об'єкт, що описує те, що сталося.

2) Redux викликає відповідну функцію редьюсер, який отримує два аргументи: поточне стан додатку та action.

3) Кореневий редьюсер може поєднувати вихід декількох редьюсерів в один за допомогою combineReducers-метода.

4) Redux зберігає повне дерево стану, повернене кореневим редьюсером.

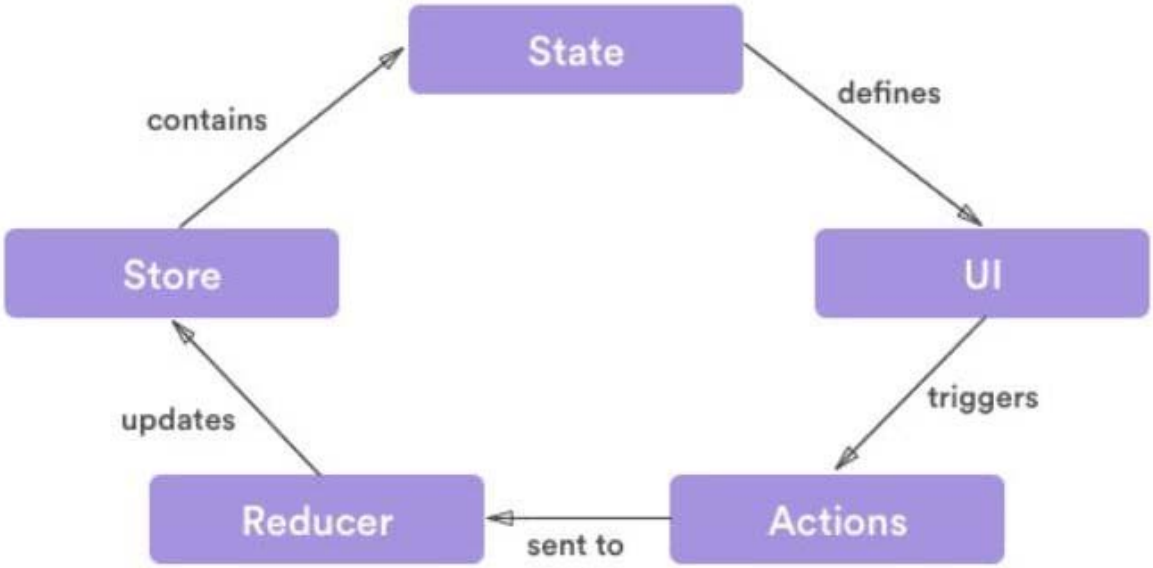

Рис. 3.2. Життєвий цикл Redux

Redux - не єдиний механізм керування стану додатка. Альтернативами являється React.Context, який також використаний у додатку та забезпечує спосіб передачі даних через дерево компонентів без необхідності передавати дані вниз вручну на кожному рівні.

Контекст в основному використовується, коли деякі дані мають бути доступними багатьма компонентами на різних рівнях, що водночас може ускладнювати повторне використання компонентів. Розглянемо основний підхід використання контексту:

1) Створюється об'єкт контекст: React.createContext(defaultValue)

2) Компонент, який передає дані на нищі рівні та має об'єкт контексту постачає його за допомогою компонента Provider, який дозволяє споживачам компонентам підписуватися на зміну контексту.:

```
<MyContext.Provider value={/* some value */}>
```

3) React-компонент, який підписується на зміну контексту використовує Consumer:

```
<MyContext.Consumer>
{value => /* render something based on the context value */}
</MyContext.Consumer>
```

Також у екосистемі React існує багато додаткових фреймворків для спрощення та прискорення розробки компонентів. Дані бібліотеки містять попередньо вбудовані компоненти, які розробники можуть використати у процесі розробки. Всі ці фреймворки дуже схожі, та не мають сильних принципових відмінностей, тому це вибір смаку. Для використання вбудованих компонент було обрано популярний та простий Semantic UI React фреймворк.

Semantic UI - це компонентна структура інтерфейсу для тематичних вебсайтів. Semantic UI дозволяє розробникам створювати веб-сайти з швидким і стислим HTML

Адмін-клієнт. Клієнтський веб-додаток для адміністратора також має CSR та являється окремим сервісом. Використовуються HTML, CSS та JavaScript. Для адміндодатка використано інший популярний фреймворк, створений компанією Google - Angular. Розглянемо основні переваги Angular [7]:

1) Архітектура MVC

За допомогою архітектури MVC (Model-View-Controller) можна ізолювати логіку програми від рівня інтерфейсу та підтримувати розділення коду. Controller отримує всі запити на додаток і працює з Model, щоб підготувати будь-які дані, необхідні для перегляду. The view використовує дані, підготовлені Controller-ом, і відображає остаточну візуальну відповідь.

2) Розширена Design Architecture. Великі веб-додатки містять безліч компонентів. Angular спрощує спосіб управління цими компонентами. Архітектура побудована таким чином, що допомагає програмісту легко знаходити та розробляти код.

3) Модулі. Модуль - це механізм, який об'єднує пов'язані директиви, компоненти та сервіси таким чином, що їх можна поєднувати з іншими модулями для створення програми. Angular-додаток може розглядатися як пазли, де кожен модуль є необхідною складовою, щоб мати можливість побачити повну картину. Існує ряд способів додавання різних елементів до модуля. Angular вирішує проблему глобальної експлуатації функцій, обмежуючи сферу застосування всіх функцій модулем, в якому вона була визначена та використовується.

4) Services and Dependency Injection (DI). Сервісу або компоненту іноді можуть знадобитися інші залежні сервіси для виконання завдання. Для виконання цих залежностей використовується шаблон Services and Dependency Injection (DI). Він розділяє завдання між різними сервісами. Клієнтський сервіс не створить залежний об'єкт, скоріше він буде створений і введений Angular-інжектором. Angular-інжектор відповідає за створення службових примірників та введення їх у такі класи, як компоненти та сервіси.

5) Спеціальні директиви. Спеціальні директиви покращують функціональність HTML і підходять для динамічних додатків. Всі вони починаються з префікса ng, щоб HTML міг їх ідентифікувати. Деякі з них:

- ngModel: забезпечує двостороння прив'язка даних до елементів форми HTML.

- ngClass: видаляє та додає набір класів CSS.

- ngStyle: додає та видаляє набір стилів HTML.

6) TypeScript. Angular пишеться за допомогою TypeScript, що є суперсетом JavaScript. Він повністю відповідає JavaScript, а також допомагає виявити і усунути поширені помилки під час кодування. Незважаючи на те,

що невеликі проекти JavaScript не потребують такого вдосконалення, для корпоративних програм потрібні розробники, щоб зробити свій код чистішим і частіше перевіряти якість.

7) Комьюніті. Angular з'явився набагато раніше ніж деякі фреймворки (Vue), тому комьюніті забезпечила достатньо навчальних матеріалів, дискусій та сторонніх інструментів, щоб приступити до використання даного фреймворку, а також знайти рішення майже кожного виникаючого питання.

До недоліків цього фреймворку можно віднести:

1) Обмежені можливості SEO та недостатня доступність для сканерів пошукових систем.

2) Для написання функціоналу, потрібно написати більше програмного коду ніж на інших фреймворках (React, Vue).

3) Складніше зрозуміти та важче вивчити ніж інші фреймворки (React, Vue) Angular має свої недоліки, але оскільки вони не є суттєвими, його було обрано при створенні клієнтської частини для адміністратора.

Для встановлення та керування пакетами обрано пакетний менеджер - yarn. Оскільки, адмін-частина значно менша ніж клієнтська частина для звичайних користувачів, не було проблеми з керуванням внутрішнього стану компонент.

Так як у звичайній клієнтській частині, написаній на React, було використано додатковий фреймворк для спрощення та прискорення розробки компонентів, у admin-клієнтській частині написаній на Angular використано популярний фреймворк Angular Material.

Angular Material - це бібліотека компонентів інтерфейсу для Angular розробників. Компоненти Angular матеріалу допомагають створювати привабливі, послідовні та функціональні веб-сторінки та веб-додатки, дотримуючись сучасних принципів веб-дизайну, таких як портативність браузера, незалежність пристрою.

Сервер. Користувач не бачить сторону сервера, але вона предоставляє API для використання клієнтом. Для розробки серверного додатку використано технологію Node.js.

Node.js полягає в тому, що це середовище виконання Javascript, яке допомагає виконувати JavaScript-код на сервері. Це крос-платформний JavaScript з відкритим кодом, який допомагає розвивати мережевий додаток у режимі реального часу.

Розглянемо основні переваги та недоліки використання Node.js на стороні сервера [8]:

1) Легка масштабованість. Однією з ключових переваг Node.js є те, що розробникам легко масштабувати програми в горизонтальному та вертикальному напрямках. Програми можна масштабувати горизонтально шляхом додавання додаткових вузлів до існуючої системи. Крім того, Node.js також дає можливість додавання додаткових ресурсів до окремих вузлів під час вертикального масштабування програми.

2) Легкий у застосуванні. Оскільки на стороні клієнта використано мову програмування JavaScript, використання цієї ж мови на сервері дійсно пришвидшує та спрощує розробку додатку.

3) Підтримка великого та активного комьюніті. Node.js має велику та активну спільноту розробників, які постійно вносять свій внесок у його подальший розвиток та вдосконалення.

4) Обробка запитів одночасно. Оскільки Node.js надає можливість не блокувати системи вводу / виводу, це допомагає одночасно обробляти кілька запитів. Система може обробляти паралельний запит ефективно краще, ніж інші, включаючи Ruby або Python. Вхідні запити виконуються швидко та систематично.

5) JSON-форматом. Оскільки, клієнтська частина написана за допомогою мови Javascript, це дуже зручно працювати з JSON-форматом,

щоб забезпечити обмін даними між веб-сервером і клієнтом. Цьому також сприяють вбудовані API для розробки серверів HTTP, TCP та DNS тощо.

Серед недоліків Node.js - платформи можно відзначити наступне:

- нестабільний API;

- відсутність сильної системи підтримки бібліотек;

- asynchronous programming model;

Для встановлення та керування пакетами обрано пакетний менеджер - yarn. Також потрібен додатковий фреймворк, який би дозволяв структурувати веб-додаток для обробки декількох різних запитів http за певною URLадресою. Для цих цілей обрано Express.js. Express - це мінімальний та простий, open-source гнучкий фреймворк для Node.js веб-додатків, створений для того, щоб значно спростити розробку веб-сайтів та API [9]. Розглянемо основні переваги даної бібліотеки [10]:

1) Простий у налаштуванні.

2) На основі методів URL та HTTP можна визначити маршрути існуючої програми.

3) Доступна інтеграція з різними двигунами шаблонів, включаючи EJS, Vash, Jade та інші (у server side rendering випадку).

4) Ресурси та статичні файли програми легко обслуговувати.

5) Створення сервера API REST.

6) Можливість скористатися з'єднанням з такими базами даних, як MySQL, Redis та MongoDB.

У додатку А наведеного лістинг основних модулів додатку.

3.2. Програмна реалізація бізнес-логіки веб-сайту

Для того щоб створити SPA-додаток, написаний за допомогою React.js бібліотеки, використано набір інструментів Create React App (рисунок 3.3).

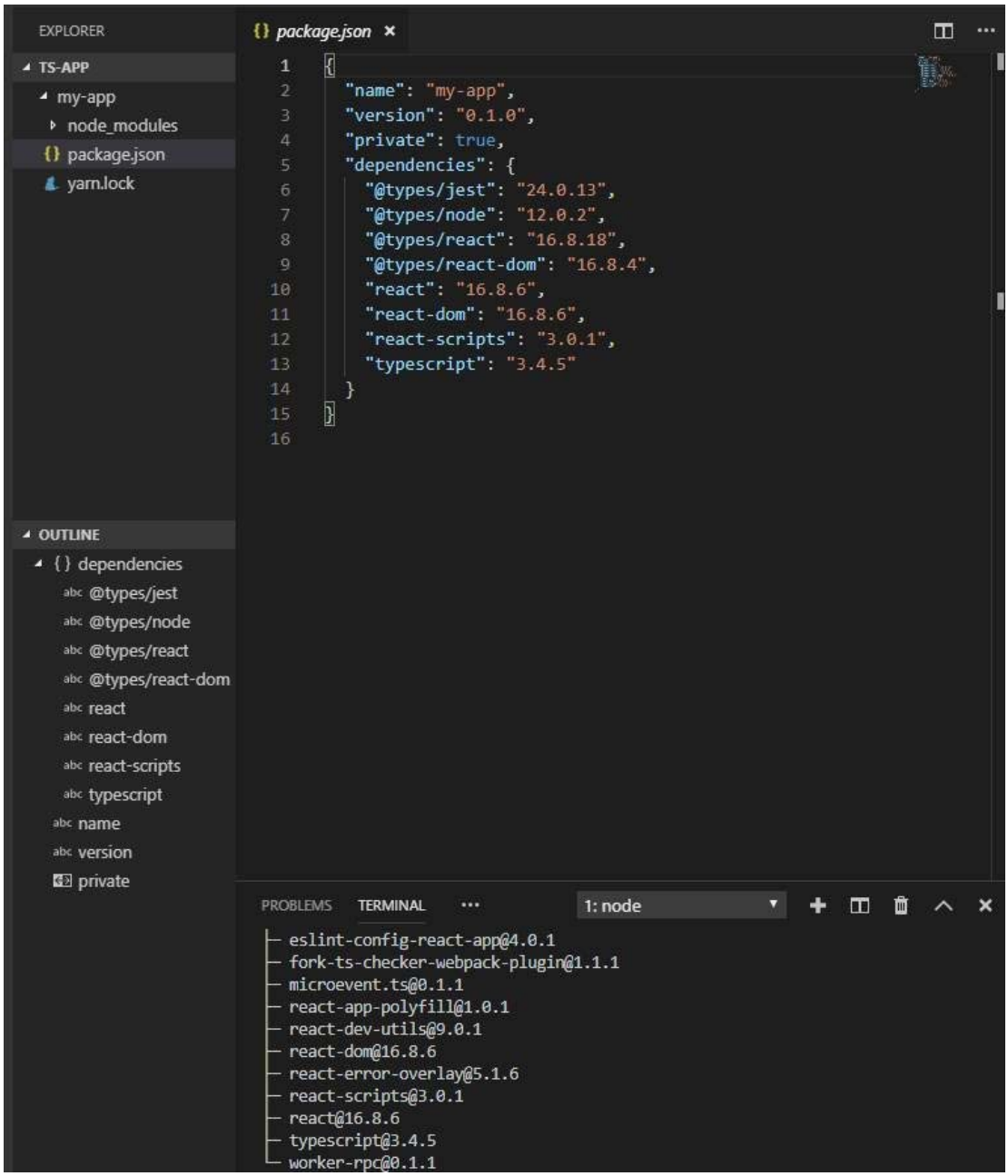

Рис. 3.3. Create React App

Create React App - зручне середовище для вивчення фреймворка React.

Цей прилад налаштовує нові можливості Javascript, оптимізовує додаток для продакшена.

Для того щоб створити проект, виконано команду:

npx create-react-app my-app

Для розробки клієнтської частини веб-додатку необхідні наступні популярні пакети:

- axios - Promise HTTP-клієнт для браузера та node.js

- redux - контейнер стану для додатків JavaScript.

- react-router - предоставляє основний функціонал маршрутизації для React Router;

- crypto-js - Javascript бібліотека для шифрування;

- lodash - сучасна бібліотека утиліт JavaScript, яка забезпечує модульність та продуктивність;

- semantic-ui - це фреймворк інтерфейсу, призначений для тематизації.

У веб-додатку здійснення HTTP-запитів для отримання або збереження даних реалізовано за допомогою бібліотеки axios [12]. Наступний фрагмент коду створює екземпляр HTTP-клієнта, налаштовує за умовчанням та встановлює інтерцептори (дозволяють запускати або змінювати запит до того, як він відправиться та досягне місця призначення) для запиту та відповіді:

```
import axios from 'axios';
import { store } from 'redux/store/configureStore';
const instance = axios.create({
headers: { 'Content-Type': 'application/json' },
baseURL: 'https://eatngo.herokuapp.com/',
responseType: 'json'
});
instance.interceptors.request.use(
(config) => {
const AUTH_TOKEN = JSON.stringify(store.getState().auth);
if (AUTH_TOKEN) {
config.headers['Authorization'] = AUTH_TOKEN;
}
return config;
}, error => Promise.reject(error)
);
instance.interceptors.response.use(
response => response,
({ response = { status: 500 } }) => {
```

```
const { status } = response;
const redirect = redirects[status];
if (redirect) document.location = redirect;
return Promise.reject(response);
}
);
```

Управління станом додатку реалізовано за допомогою бібліотеки redux, redux-thunk для здійснення HTTP-запитів у redux та redux-persist для збереження об’єкта стану Redux у сховищі Persistent. Наступний фрагмент коду:

- створює конфігурацію для persistent-сховища
- створює головний редьюсер, який задає, як змінюється стан програми у відповідь на дії, надіслані в сховище;
- створює сховище;
- створює persistent-сховище;

```
import { createStore, applyMiddleware, compose } from 'redux';
import { persistStore, persistReducer } from 'redux-persist';
import logger from 'redux-logger';
import thunk from 'redux-thunk';
import storage from 'redux-persist/lib/storage';
import autoMergeLevel2 from 'redux-persist/lib/stateReconciler/autoMergeLevel2';
import rootReducer from 'redux/reducers';
const persistConfig = {
key: 'root',
storage: storage,
whitelist: ['auth'],
stateReconciler: autoMergeLevel2
};
const pReducer = persistReducer(persistConfig, rootReducer);
export const storeFactory = () => createStore(pReducer,
composeEnhancers(applyMiddleware(thunk), applyMiddleware(logger)));
```

```
export const store = storeFactory();
export const persistor = persistStore(store);
```

У веб-додатку реалізована авторизація та реєстрація. Кожен користувач хоче бути впевненим, що його персональні дані залишаються засекреченими.

Для цих цілей, потрібно шифрувати дані користувача такі як пароль. Для цього використано бібліотеку шифрування crypto-js. Наступний фрагмент коду шифрує дані на основі стандарту кодування двійкових даних за допомогою тільки 64 символів ASCII, коду автентифікації повідомлення на основі хеша та секретного слова “secret” [13]:

```
import CryptoJS from 'crypto-js';
export const crypting = (value) => {
return CryptoJS.enc.Base64.stringify(CryptoJS.HmacSHA256(value, 'secret'));
};
```

Сервер. На серверній частині веб-сайту використано невеликий, open-source та гнучкий фреймворк веб-додатків Node.js - Express.js. Даний фреймворк як і React предоставляє зручний інструмент для швидкого створення "каркасу" веб-додатка. Для того щоб скористатись ним створити проект необхідно виконати наступні команди:

- npm install express-generator –g

express my-app

Для розробки серверної частини веб-додатку необхідні наступні популярні пакети:

- express;
- mongoose - інструмент моделювання об'єктів MongoDB, призначений для роботи в асинхронному середовищі
- nodemailer - предоставляє функціонал для надсилання електронних повідомлень на пошту на Node.js
- crypto-js - Javascript бібліотека для шифрування.

Відправлені запити з клієнтського додатку повинні бути оброблені маршрутизацією. Маршрутизація визначає, як додаток відповідає на

клієнтський запит до конкретної адреси (URI). Маршрутизація розбиває API endpoints на певні роути такі як: /users, /dishes, /places, /offers, /order та інші.

Кожен з цих роутів має додаткові шляхи або параметризовані адреси та відповідні обробники. У наступному фрагменті коду наведений приклад userroute та його обробників на різні параметри та HTTP-методи:

```
const user = require('../controllers/usersController')
module.exports = app => {
app
.route('/registration')
.post(user.userRegistration);
app
.route('/authenticate')
.post(user.authenticate);
app
.route('/users')
.get(user.getUserInfo)
.put(user.updateUserInfo)
app
.route('/verify/:code')
.get(user.verification)
app
.route('/forgotPassword')
.post(user.sendEmailToResetPassw)
app
.route('/reset')

.post(user.resetPassword)
};
```

Усі розділені маршрути комбінуються в один корінний:

```
const routeSpecialOffers = require('./specialOffersRouters');
const routesOrder = require('./orderRoutes');
const routesTables = require('./tablesRoutes');
const routesUsers = require('./usersRoutes');
```

```
const routesUserOrders = require('./userOrdersRoutes');
const routesAdmin = require('./adminRoutes');
const routesDishes = require('./dishesRoutes');
const routesPlaces = require('./placesRoutes');
module.exports = app => {
routeSpecialOffers(app);
routesOrder(app);
routesTables(app);
routesUserOrders(app);
routesDishes(app);
routesPlaces(app);
routesUsers(app);
routesAdmin(app);
}
```

У головному файлі index.js налаштовано маршрутизацію:

```
const routes = require('./src/routes/setUpRoutes');
const app = express();
routes(app);
```

На кожен маршрутний роут є відповідний обробник. Наступний фрагмент коду містить усі контролери до /user маршруту:

```
const mongoose = require('mongoose');
const User = mongoose.model('User');
const userService = require('../services/userService');
const queryWrapper = require('../utils/queryWrapper');
exports.getUserInfo = async (req, res) => {
queryWrapper(req, res, userService.getUserInfo, userService.errHandler);
};


.post(user.resetPassword)
};
```

Усі розділені маршрути комбінуються в один корінний:

```
const routeSpecialOffers = require('./specialOffersRouters');
const routesOrder = require('./orderRoutes');
```

```
const routesTables = require('./tablesRoutes');
const routesUsers = require('./usersRoutes');
const routesUserOrders = require('./userOrdersRoutes');
const routesAdmin = require('./adminRoutes');
const routesDishes = require('./dishesRoutes');
const routesPlaces = require('./placesRoutes');
module.exports = app => {
routeSpecialOffers(app);
routesOrder(app);
routesTables(app);
routesUserOrders(app);
routesDishes(app);
routesPlaces(app);
routesUsers(app);
routesAdmin(app);
}
```

У головному файлі index.js налаштовано маршрутизацію:

```
const routes = require('./src/routes/setUpRoutes');
const app = express();
routes(app);
```

На кожен маршрутний роут є відповідний обробник. Наступний фрагмент коду містить усі контролери до /user маршруту:

```
const mongoose = require('mongoose');
const User = mongoose.model('User');
const userService = require('../services/userService');
const queryWrapper = require('../utils/queryWrapper');
exports.getUserInfo = async (req, res) => {
queryWrapper(req, res, userService.getUserInfo, userService.errHandler);
};


default: ""
},
email: {
```

```
type: String,
required: 'email cannot be blank'
},
confirm: {
type: Boolean,
default: false
},
password: {
type: String,
required: 'password cannot be blank'
},
phoneNumber : {
type: String,
default: ""
},
userImage: {
type:  String,
default: ""
}
},
{ collection: 'users' }
);
module.exports = mongoose.model('User', usersSchema);
```

Оскільки у користувача є можливість авторизуватись та реєструватись на клієнтській частині, слід обробляти, зашифровувати і розшифровувати дані. Для цих цілей використовується така ж бібліотека шифрування як і на клієнтській частині - crypto-js. Наступний фрагмент коду створює функції шифрування та розшифрування даних:

```
const CryptoJS = require("crypto-js");
exports.secret = 'newSecret';
exports.deCryptPassword = (password, secret = secret) => {
const bytes = CryptoJS.AES.decrypt(password.toString(), secret);
const plainText = bytes.toString(CryptoJS.enc.Utf8);
```

```
return plainText;
};
exports.enCryptPassword = (password, secret = secret) => {
const cipherText = CryptoJS.AES.encrypt(password, secret);
return cipherText;
};
```

Приклад їх використання у авторизації адміністратора:

```
authentication = async adminData => {
const admin = await dbService.getOneElementByField
(Admin, {name: adminData.name});
if(!admin) throw new Error('Admin name or password is incorrect');
const decryptedPassword = cryptor.deCryptPassword
(admin.password, cryptor.secret);
if(decryptedPassword === adminData.password){
return true;
} else {
throw new Error('Admin name or password is incorrect');
}
}
```

Наступний фрагмент коду виконує підключення до бази даних:

```
exports.connectToDB = () => {
return new Promise((resolve, reject) => {
mongoose.connect(url, { useNewUrlParser: true,
useFindAndModify: false, dbName:
dbName }, (err) => {
err ? reject(err) : resolve();
});
})
};
```

де url - адреса до сервісу бази даних, а dbName - назва бази даних

API серверу. Розглянемо основні API endpoints нашого серверу, які представлено в таблиці 3.1.:

Таблиця 3.1

API endpoints

url	method	Description
/brands	GET	отримати усі брендів
/brands	POST	створити бренд
/brands/:brandId	GET	отримати окремий бренд
/brands/:brandId	PUT	оновити окремий бренд
/brands/: brandId	DELETE	видалити окремий бренд
/brands/:filter/:count/:category	GET	отримати список відфільтрованих брендів
/places	GET	отримати всі
/places	POST	створити власний бренд
/places/:placeId	GET	отримати окремий бренд
/places/:placeId	PUT	оновити окремий бренд
/places/:placeId	DELETE	видалити окремий бренд
/offers	GET	отримати усі спеціальні акції
/offers	POST	створити акцію
/offers/:specialOffersId	GET	отримати окрему акцію
/offers/:specialOffersId	PUT	оновити окрему акцію
/offers/:specialOffersId	DELETE	видалити окрему акцію
/slogans	GET	отримати список всіх слоганів
/slogans/:restId	GET	отримати список слоганів окремого бренду
/registration	POST	зареєструвати користувача
/authenticate	POST	аутентифікувати користувача
/users	GET	отримати інформацію користувача
/users	PUT	оновити інформацію користувача
/verify/:code	GET	верифікувати користувача
/forgotPassword	POST	відправити повідомлення на пошту для відновлення паролю
/reset	POST	скинути пароль користувача
/myorders	POST	отримати замовлення користувача
/rating	POST	додати оцінку користувача на страву
/order	POST	створити замовлення
/order	GET	отримати усі замовлення
/order/:orderId	PUT	оновити замовлення

З даним API повинні вміти працювати клієнти. Список API-endpoints може легко масштабуватись та включати нові.

3.3. Програмна реалізація бази даних

Базу даних потрібно розмістити на окремому сервері. Для цих цілей використано безкоштовний хостинг баз даних на https://cloud.mongodb.com/.

Дана система надає безкоштовний керований сервер, оптимізований для роботи бази даних. Підключення до бази даних описані на серверній частині веб-додатку.

У MongoDB база даних складається з колекцій. Як правило, колекція відображає сутність предметної області. Колекція складається з "документів" (далі запис). Запис є JSON-об’єкт з обов’язковою наявністю ідентифікатора ObjectId, який MongoDB задає автоматично і контролює його унікальність, аж до унікальності в усьому світі [2]. ObjectId є типом даних. У порівнянні з реляційними СУБД, колекція є таблицею, а запис є рядком в таблиці.

Тип збережених в запису даних - довільний, з перерахованих нижче:

- String;
- Array (структура даних з доступом по цілочисельному індексу);
- Boolean;
- Date;
- Integer;
- Null.
- Object (структура даних, що надає доступ до елементів по ключу).

Цілісність зв'язків в MongoDB не підтримується. Це завдання вирішується на рівні коду сервера додатків, що і було реалізовано за допомогою інструменту ODM, - Mongoose.

Для того щоб мати представлення відношення між сутностями, була розроблена схема БД, на якій відображені псевдо-зв’язки, рисунок 3.8, нотація Гордона Евересту [21].

Доповнимо до схеми, що найменування колекцій і псевдо-зв'язки можуть представляти дещо інший сенс, ніж таблиці в реляційних БД, так як існують вкладені об’єкти. Прикладом є колекція з найменуванням product_type.

Типи псевдо-відношень документа з іншими сутностями, враховуючи не структурованість даних:

● HasOne: має один тип документа.

● HasOne: має одну мову.

● ManyToMany: належить до одного або множини продуктів (BelongsTo), так як є загальним. З іншого боку один товар має множину документів на різних мовах (HasMany).

● HasMany: має багато змін.

● BelongsTo: належить до одного скачування, відправлення по email або замовлення.

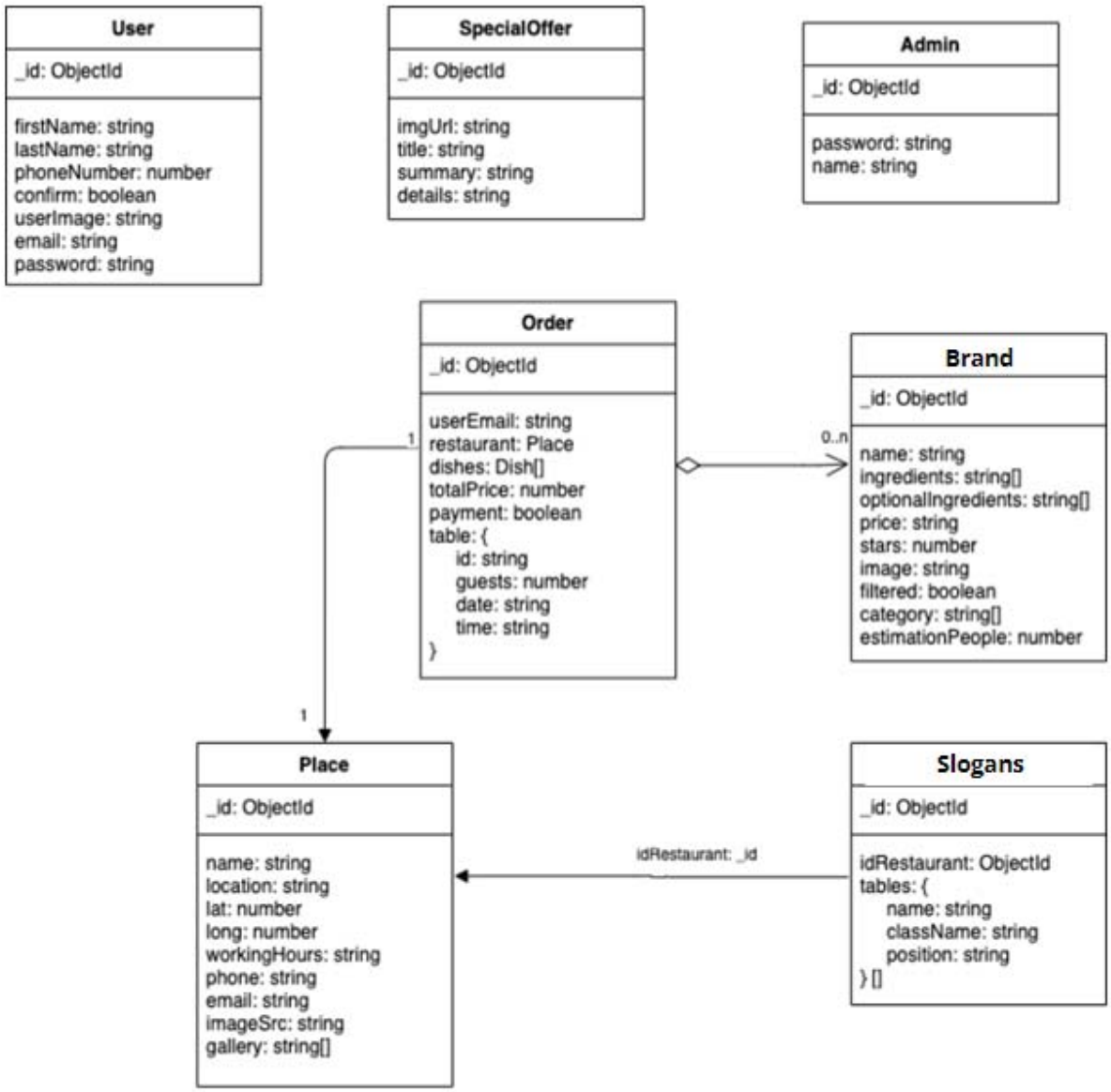

Рис. 3.4. Схема бази даних з позначенням псевдо-зв’язків

Кожна з колекцій містить відповідні документи. Розглянемо вид user документа:

1) _id - автоматично створюваний ідентифікатор для документа

2) firstName - ім'я користувача

3) lastName - прізвище користувача

4) confirm - чи верифікований користувач

5) phoneNumber - номер телефону користувача

6) userImage - посилання на фото користувача, якщо він завантажив його у персональному кабінеті

7) email - електронна пошта користувача

8) password - зашифрований пароль користувача

Вид документів в базі даних повністю співпадає з Mongoose-схемами, які використовуються на серверній частині.

Для опису особливостей проектування схем в MongoDB, були використані такі поняття:

● Вкладення - повне утримання об’єкта (ів) однієї сутності в іншій.

● Посилання - унікальний ідентифікатор ObjectId зовнішньої сутності.

На рисунку 3.5, представлений приклад вкладення одного об’єкта в інший в колекції product_type.

Використання інструменту Mongoose дозволяє автоматично генерувати ObjectId при вставці вкладеного об'єкта. Наприклад, при додаванні продукту в тип продукту. При проектуванні були визначені деякі особливості проектування схеми БД в MongoDB, які полягають в тому, що:

● При складових псевдо-ключах різко втрачається гнучкість з даними.

● Зберігання тільки ідентифікатора об’єкта зовнішньої сутності зажадає виконати додатковий запит для отримання всіх даних зовнішньої сутності.

● Вкладення об’єкта сутності має сенс тоді, коли вкладений об'єкт буде зустрічатися разом з батьківським.

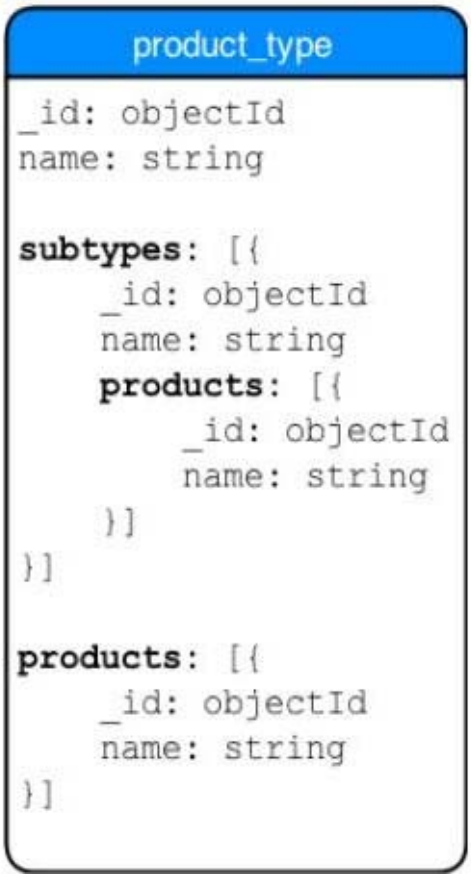

Рисунок 3.9 - Вкладення об’єкта в колекції

● Вкладення об'єкта сутності відображає в реляційній схемі зв’язок один до багатьох.

● Посилання забезпечують більшу гнучкість при частих змінах вкладеної сутності.

● Масив посилань об'єктів сутності відображає в реляційній схемі зв'язок багато до багатьох. Атрибути основної сутності документ, представлені в таблиці 3.3.

При запиті бренд відбувається пошук по атрибутам: products, valid_from_date, status. Результат повинен повернути бренд з відповідними атрибутами.

Таблиця 3.3.

Атрибути основної сутності brand

Найменування атрибуту	Тип	Опис
number	string	Номер версії бренда
language (localized)	string	Унікальне іменування мови.
document_type	string	Унікальне іменування типу документа.
countries	array	Унікальне іменування країн(и).
owner	string	Ім'я та прізвище укладача бренду.
file	object	Найменування файлу, що складається з імені файлу, що згенерованого на підставі його хеша для унікальності в файловій системі.
orders	object array	Замовлення для яких доступний бренд.
valid_from_date	date	Дата початку дії
reminder_date	date	Дата нагадування актуальності бренду.
status	string	Статус бренду відкритий, активний,застарілий.

Висновки до розділу 3

У даному розділі обґрунтовано технологію, мову програмування та розроблено програмну систему. Обґрунтовано засоби розробки додатку та здійснено опис основних програмних модулів системи.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ

4.1. Тестування веб- сайту

Існує кілька методів тестування програмного забезпечення, розглянемо і зробимо їх аналіз, для вибору найбільш оптимального. Ручне тестування - сутність методу полягає в перегляді вихідних текстів програми і правильності алгоритму роботи. Для великого обсягу вихідного коду і через його складність використання цього методу не дуже ефективне.

Автоматизоване тестування, яке діляться на метод «білого ящика» і метод «чорного ящика».

Метод «білого ящика» - фактично перевіряє відповідність програми розробленим раніше алгоритмам. Цей метод не виявляє алгоритмічні помилки і тому використовується тільки як доповнення до методу «чорного ящика». Вихідною інформацією для цього методу є граф-схема алгоритму, яка потім перетвориться в граф управління. Далі розробляється набір текстів, з тим або іншим ступенем повноти покривають множинк дуг або маршрутів в графі управління. Оскільки для складної програмної системи керуючий граф також виходить вельми складним, областю застосування цього методу фактично є модулі і невеликі підсистеми.

Метод «чорного ящика» - реалізує тестування програмного комплексу по входу/виходу. Метод перевіряє відповідність функціонування системи початкової специфікації на програмний засію. Він може виявляти помилки будь-яких класів і придатний на практиці: для тестування великих систем.

Розглянемо процедуру тестування розроблюваного додатку для вивчення документа за допопогою Mocha і Chai для його тестування.

Mocha - це javascript фреймворк для Node.js (рисунок 4.1), який дозволяє проводити асинхронне тестування. Він створює середовище, в якому ми можемо використовувати свої улюблені assert бібліотеки.

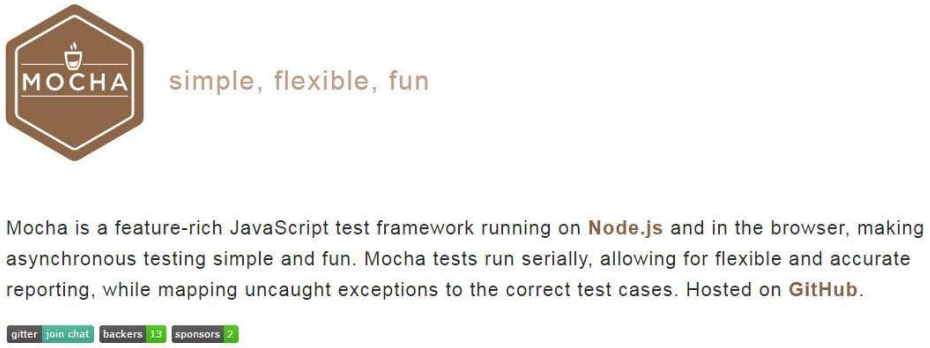

Рис. 4.1. Середовище тестування Mocha

Mocha поставляється з величезною кількістю можливостей. На сайті їх величезний список. Найбільше нам занадобилося наступне:

- проста підтримка асинхронності, включаючи Promise;
- підтримка таймаутів асинхронного виконання before, after, before each, after each (дуже корисно для очищення середовища перед тестами);
- використання будь-якої assertion бібліотеки, яку ви знаходите (в нашому випадку Chai).

Chai: assertion бібліотека (рисунок 4.2). Отже, з Mocha у нас з’явилося середовище для виконання наших тестів.

Chai дає нам можливість вибору інтерфейсу: "should", "expect", "assert". В нашому проекті використано should. У Chai є плагін Chai HTTP, який дозволяє без труднощів тестувати HTTP запити.

Тестуємо / GET. Chai виконує GET запит і перевіряє, що змінна res задовольняє першому параметру (твердження) блоку it "it should GET all the

Documents". А саме, для даного порожнього завдання відповідь повинна бути наступною:

- Статус 200;
- Результат повинен бути масивом.

Так як база порожня, ми очікуємо що розмір дорівнюватиме 0.

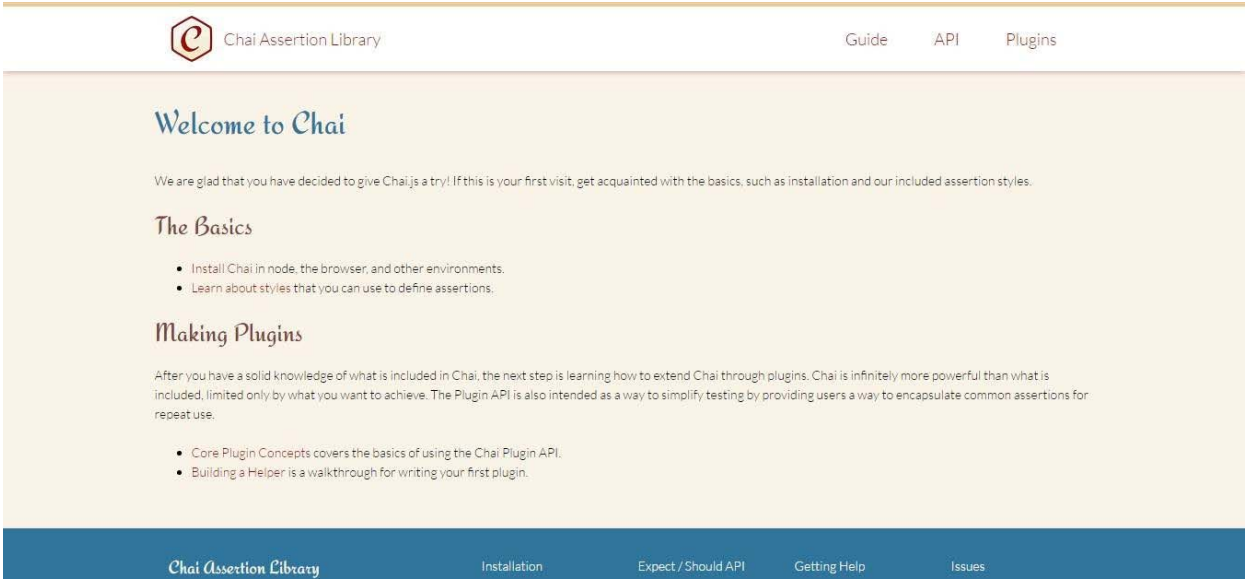

Рис. 4.2. Assertion бібліотека Chai

Синтаксис should інтуїтивний і дуже схожий на розмовну мову. В командному рядку виконуємо команду:

```
npm test
```

Результат виконання команди представлено на рисунку 4.3.

Рис. 4.3. Результат виконання команди з використанням Chai

Тест пройшов і результат відображає структуру, яку ми описали за допомогою блоків describe.

Тестуємо / POST. На наступному етапі перевірено наскільки хороший наш API. Припустимо ми намагаємося додати реквізиту без поля `describe: сервер не повинен повернути відповідну помилку. Для цього додано код в кінець блоку describe ('Documents'):

```
      describe('/POST Document', () => {
            it('it should not POST a Document without describe field', (done)
=> {
              let Document  =  {
                  title: "Array",
                  author: "Document",
                  year: 2018
              }
              chai.request(server)
                  .post('/Document')
                  .send(Document)
                  .end((err, res) => {
                      res.should.have.status(200);
                      res.body.should.be.a('object');
                      res.body.should.have.property('errors');
                      res.body.errors.should.have.property('pages');

res.body.errors.pages.should.have.property('kind').eql('required');
                        done();
                  });
            });
        });
```

Тестуємо/ GET /: ідентифікатор. Тепер створюємо реквізиту, зберігаємо її в базу та використовуємо id для виконання GET запросу. Добавляємо наступний блок:

```
      describe('/GET/:id Document', () => {
            it('it should GET a Document by the given id', (done) => {
              let  Document  =  new  Document({  title:  "Max  array",  Client:
"Najko", date: 05.2018, value: 1064 });
              Document.save((err, Document) => {
                  chai.request(server)
                  .get('/Document/' + Document.id)
                  .send(Document)
                  .end((err, res) => {
                      res.should.have.status(200);
```

```
                res.body.should.be.a('object');
                res.body.should.have.property('title');
                res.body.should.have.property('Client');
                res.body.should.have.property('date');
                res.body.should.have.property('value');
                res.body.should.have.property('_id').eql(Document.id);
              done();
            });
        });
    });
});
```

Через asserts ми переконалися, що сервер повернув всі поля і потрібну реквізиту (id у відповіді від сервера збігається з запитаним) (рисунок 4.4):

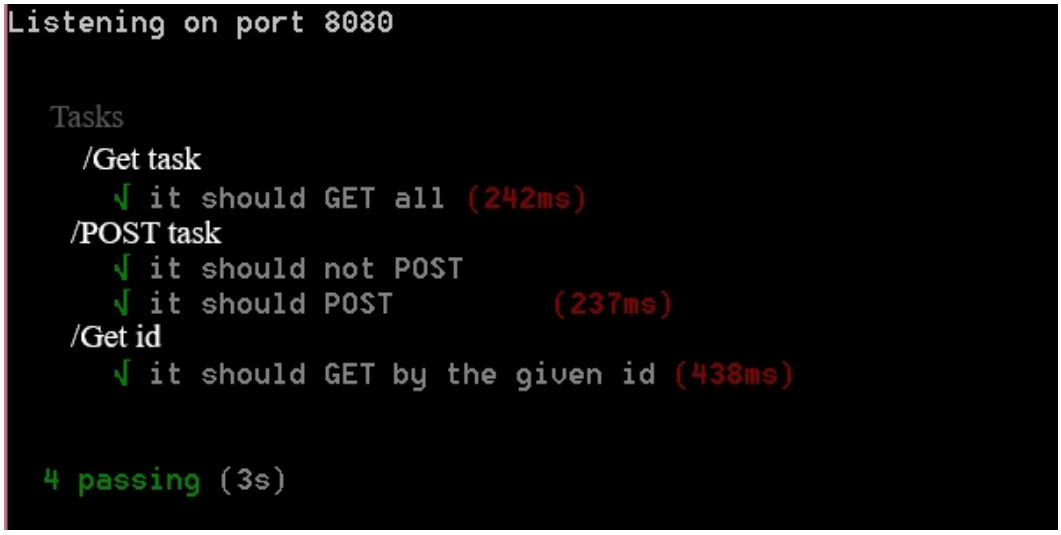

Рис. 4.4. Результат виконання команди з використанням Chai

Тестування окремих маршрутів всередині незалежних блоків дозволило отримали хороший вивід, ми написали кілька тестів, які можна повторити за допомогою однієї команди.

Для проходження етапу тестування було вирішено розгорнути розроблюваний веб-додаток на хмарному сервісі. Такі сервіси як Heroku і Windows Azure дають можливість безкоштовного використання деяких хмарних ресурсів для своїх потреб. Windows Azure не дає безкоштовного використання NoSQL бази даних MongoDB, тому розгортання проходило на хмарному сервісі Heroku.

Веб-додаток пройшов також наступні види тестування:

- функціональне тестування;

- тестування інтерфейсу;

- юзабіліті тестування.

4.2. Розгортання додатку Node.js та MongoDB в хмарі

Для розгортання розробленого веб-сайту використовується контейнер Node.js та MongoDB від Bitnami, які присутні на gcloud.

Спочатку треба зареєструватись на cloud.google.com натиснувши кнопку "Free Trial". Також для користування навіть тріал-версіями продуктів, необхідно прив’язати платіжну картку (в процесі реєстрації чи пізніше через Menu - Billing консолі управління за адресою console.cloud.google.com) (рисунок 4.5).

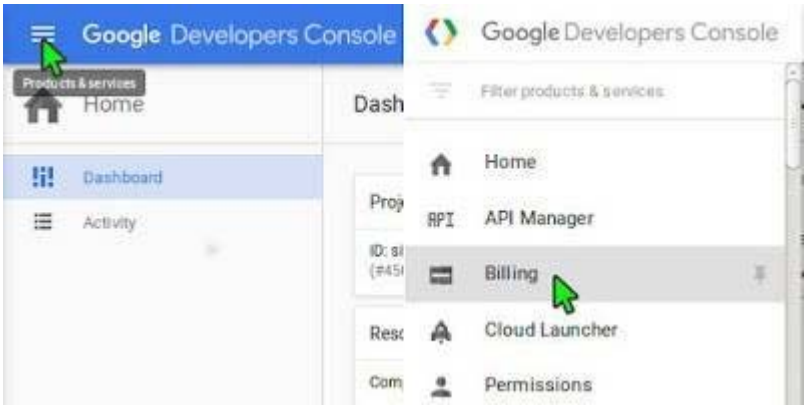

Рис. 4.5. Процедура реєстрації на на cloud.google.com

Відразу після реєстрації, буде запропоновано створити новий проект. Якщо натиснути "Edit" можна підібрати цікаве ім’я для проекту, інакше воно буде згенероване автоматично (рисунок 4.6):

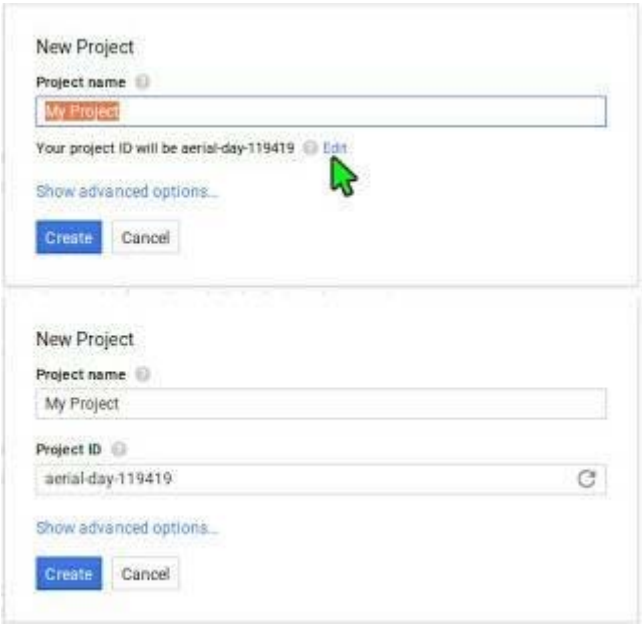

Рис. 4.6. Створення нового проекту

Саме це ім'я буде фігурувати в адресі https://<ім'я проекту>.appspot.com, за якою буде доступний додаток. Можна переглянути "Advanced options", але на момент написання, керовані додатки можливо створювати лише в зоні "us-central".

Меню створення та керування проектами буде доступним після кліку на назві поточного проекту (права верхня частина тулбару консолі управління, поруч з полем пошуку) (рисунок 4.7):

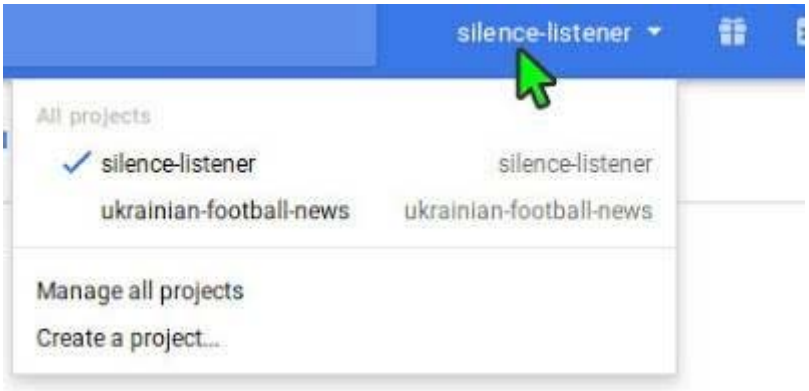

Рис. 4.7. Сторінка управління проектами

Порожній проект Google Cloud створено. Консоль керування додатками знаходиться за адресою console.cloud.google.com.

Cloud Launcher - зручний інструмент підключення модулів хмарного додатку. Саме за його допомогою підключимо до створеного проекту базу даних MongoDB від Bitnami. Для цього перейдемо до меню та виберемо відповідний пункт (рисунок 4.8):

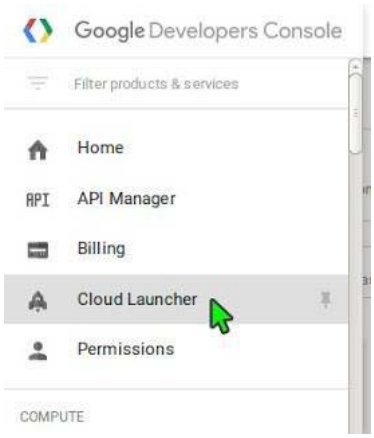

Рис. 4.8. Підключення модулів хмарного додатку

За допомогою рядка пошуку, знаходимо необхідний пакет - MongoDB від Bitnami, що представлено на рисунку 4.9.

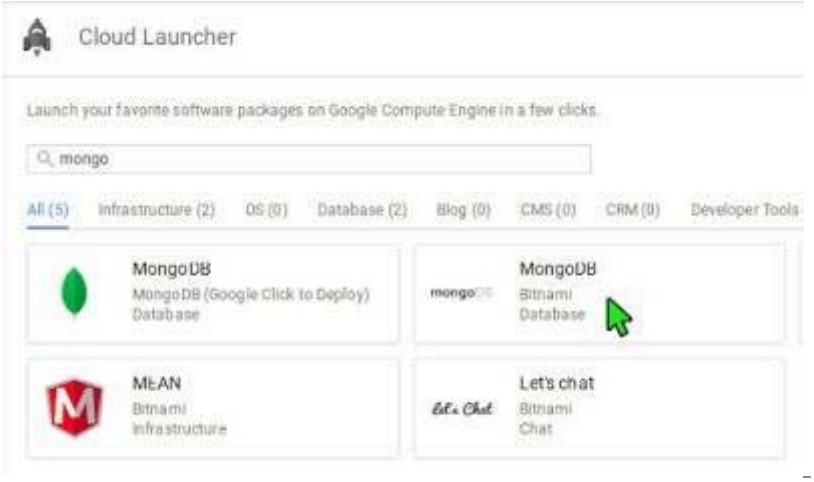

Рис. 4.9. Підключення MongoDB

В наступному меню натиснемо кнопку "Launch on Compute Engine". Далі, оберемо параметри віртуального контейнера, на якому буде розміщено базу даних (рисунок 4.10):

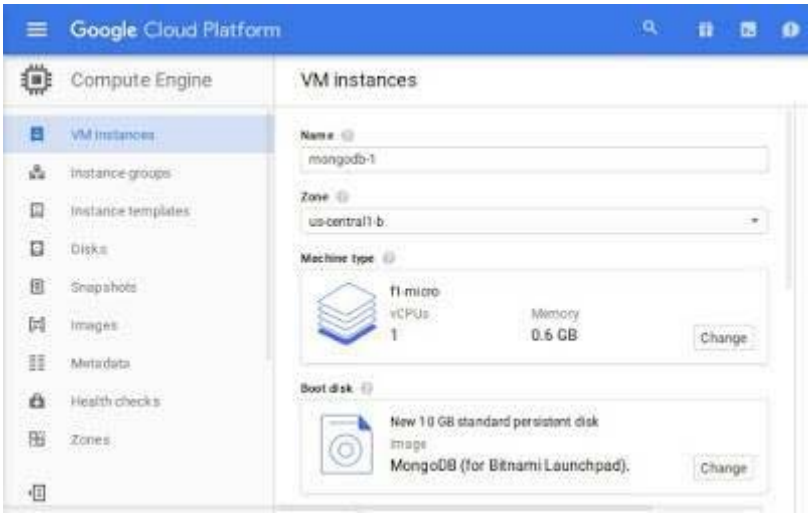

Рис. 4.10. Підключення MongoDB

Параметрів конфігурації достатньо: обрано відповідно до потреб додатку. Зверніть увагу на територіальне розміщення сервера - логічно розміщувати його поближче до споживачів. В цьому прикладі описанов все за змовчуванням, для створення необхідно натиснути кнопку "Create".

Встановлення завершиться через кілька хвилин з деталями створеного віртуального ресурсу. Увагу слід звернути, в першу чергу, на ім'я створеного ресурсу.

Переходимо на вкладку меню "VM instances" та знайдемо створений віртуальний ресурс за ім’ям (рисунок 4.11):

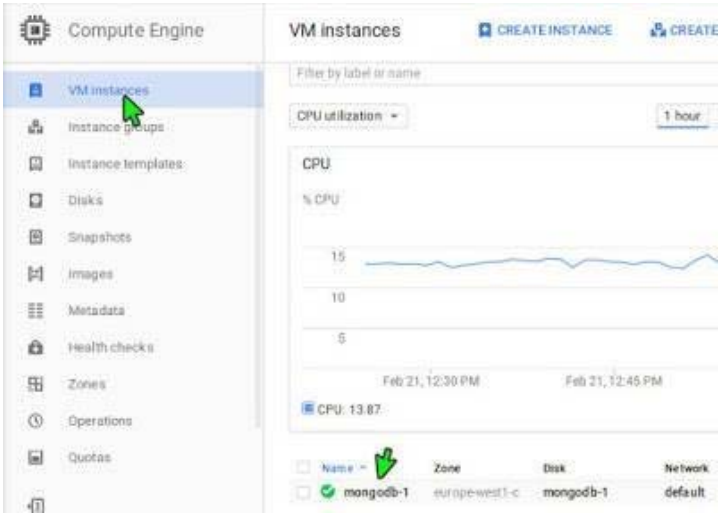

Рис. 4.11. Створений віртуальний ресурс

Перейшовши за посиланням, потрапляємо на сторінку кевування віртуальним ресурсом. На цій сторінці, серед різноманіття віджетів управління та контролю, також зібрані дані, що необхідні для доступу до віртуального сервера (рисунок 4.12):

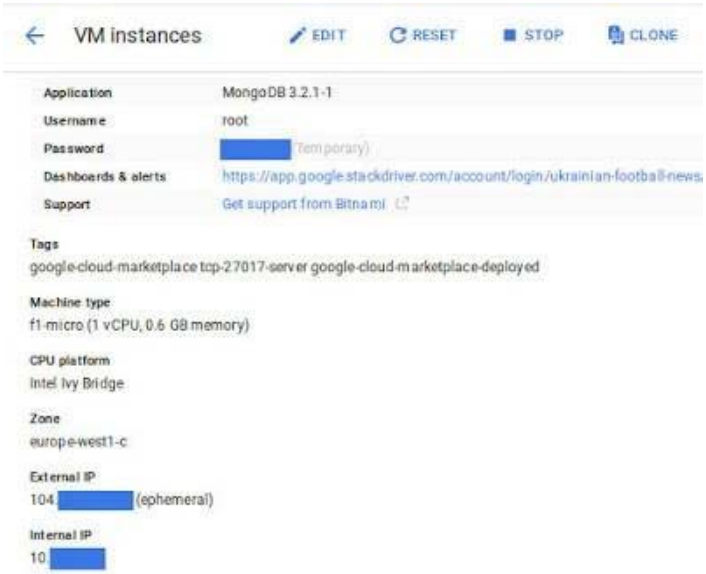

Рис. 4.12. Управління віртуальним сервером

Перевіримо з’єднання з базою даних. Для цього на вашому комп’ютері має бути встановленим MongoDB та, якщо ви працюєте на Windows, більш-менш зручна консоль (термінал), наприклад, GitBash. Відкриваємо термінал та набираємо команду для з’єднання:

```
mongo --username root -p --host xxx.xxx.xxx.xxx --port 27017
```

де xxx.xxx.xxx.xxx - IP адреса віртуального ресурсу (External IP з попереднього рисунка). В результаті цієї команди повинен з’явитись запит на введення паролю. Після введення правильного паролю - запрошення до введення команд у вигляді ">", та, можливо, деяка службова інформація.

Якщо з’днання встановлено, необхідно налаштувати базу даних. Коротко налаштування зводиться до трьох пунктів:

- зміна паролю адміністратора баз даних;
- створення окремого користувача та бази даних для додатку;
- налаштування бази даних на прийом команд лише з адреси додатку.

Для зміни паролю адміністратора необхідно виконати команди (при встановленому підключенні):

```
db = db.getSiblingDB('admin')
db.changeUserPassword("root", "new_password")
```

де new_password - новий пароль адміністратора.

Щоб створити базу даних та користувача для додатку, потрібно виконати наступні команди:

```
db = db.getSiblingDB('db_name')
db.createUser( { user: "db_user", pwd: "db_password", roles: [ "readWrite", "dbAdmin" ] } )
```

де: db_name - ім’я бази даних, яку буде використовувати додаток, db_user - ім’я користувача бази даних, db_password - пароль для доступу до бази даних.

Щоб закрити поточне з’єднання, натискаємо Ctrl + C або вводимо команду "exit". Протестуємо з’єдняння створеного користувача з базою даних:

```
mongo xxx.xxx.xxx.xxx:27017/db_name -u db_user -p db_password
```

Залишилось налаштувати ще один параметр бази даних - IP-адресу додатку, з яким потрібно взаємодіяти. Використовуємо налаштування за замовчуванням - база доступна з будь-якої адреси.

Спробуємо підключитись до створеної бази даних з додатку Node.js, що розміщений на локальному комп’ютері. Розгортаємо додаток Node.js:

```
npm init
```

Цей додаток спочатку просто підключатиметься до створеної бази даних та повідомлятиме про вдале підключення. Найкраще роботу додатку пояснює зміст файлу index.js:

```
var MongoClient = require('mongodb').MongoClient;
MongoClient.connect(process.env.DB_URI, function (err, db) {
    if (err) {
        return console.error('Can\'t connect to DB!');
    }
    console.log('Connection established');
});
```

Встановлюємо необхідний модуль mongodb (драйвер підключення до БД):

```
npm install --save mongodb
```

Задаємо змінну середовища DB_URI та запускаємо додаток:

```
DB_URI=mongodb://db_user:db_password@xxx.xxx.xxx.xxx/db_name nodeindex.js
```

де db_user, db_password, xxx.xxx.xxx.xxx, db_name - ім’я користувача бази даних, пароль, IP-адреса, ім’я бази даних відповідно. Якщо в результаті роботи додатку отримали повідомлення "Connection established" - значить все пройшло вдало.

Щоб ми змогли перевірити роботу додатку за допомогою браузера, Файл index.js повинен мати наступний вигляд:

```
var MongoClient = require('mongodb').MongoClient,
    express = require('express');
MongoClient.connect(process.env.DB_URI, function (err, db) {
    if (err) {
        return console.error('Can\'t connect to DB!');
    }
    var app = express();
    app.get('/', function (req, res) {
        res.json({ message: 'ok' });
    });
    app.listen(process.env.PORT || 3000, function () {
        console.log('Server started');
    });
});
```

Також встановлюємо необхідний модуль express:

```
npm install --save express
```

Для зручності, щоб не набирати щоразу команду запуску додатку, вказуємо їй у розділі "scripts" файлу package.json для випадку тестової та production версій:

```
{
   ...
   ...
   "scripts": {
       "test": "DB_URI=mongodb://db_user:db_password@xxx.xxx.xxx.xxx/db_name
node index.js",
       "start": "node index.js"
   },
   ...
   ...

}
```

Тепер появляється можливість запустити додаток командою

```
npm test
```

Відкриємо браузер на сторінці localhost:3000, повинні спостерігати

```
{ "message" : "ok" }
```

Виконаємо команду для ініціалізації gcloud:

```
gcloud init
```

В процесі ініціалізації потрібно авторизуватись та надати відповідні дозволи для додатку (відкриється веб-браузер). Після надання доступу у браузері, в консолі необхідно буде ввести відповіді на кілька запитань, наприклад, зона розміщення сервера, тощо - на перший раз можна залишити за змовчуванням. Також треба буде вказати ідентифікатор поточного проекту:

```
Enter project id would you like to use:
```

Після цього буде запропоновано зклонувати поточний проект:

```
Generally projects have a repository named [default]. Would you like to try
clone it? (Y/n)?
```

Проект буде склоновано у поточну директорію під ім’ям "default", що може створити деяку путаницю. Склонувати проект можна командою:

```
gcloud source repos clone default YOUR_DIR
```

де YOUR_DIR - директорія, у яку буде склоновано проект, розміщувати цю директорію в папці проекту з ім’ям "release" або "dist", та копіювати в неї лише production версії файлів, необхідних для роботи додатку.

Скопіюємо в директорію YOUR_DIR файли нашого проекту index.js та package.json, а також створимо файл app.yaml з наступним вмістом:

```
runtime: nodejs
vm: true
env_variables:
```

```
  DB_URI: 'mongodb://ufnprsr:News0FfoToUA16zx@104.155.55.150:27017/ufn'
skip_files:  - ^node_modules
```

Створений файл – конфігураційний, в нашому випадку - вказано тип середовища, в якому працює додаток, що для нього потрібно створити окремий віртуальний ресурс, визначено змінну середовища та файли, які не потрібно копіювати на сервер.

Щоб запустити проект на сервері, потрібно виконати команду:

```
gcloud preview app deploy app.yaml --promote
```

підтвердити свої наміри в консолі (Y/n) та дочекатись завершення процесу встановлення, що може тривати кілька хвилин. Потім перейти за посиланням у браузері, що має вигляд https://<ім'я проекту>.appspot.com.

В результаті розгортання додатку nodejs на сервері, буде створено новий віртуальний сервер, що керується "google app engine" (його назва починатиметься з "gae"). Побачити його можна на вкладці "VM instances" (рисунок 4.13):

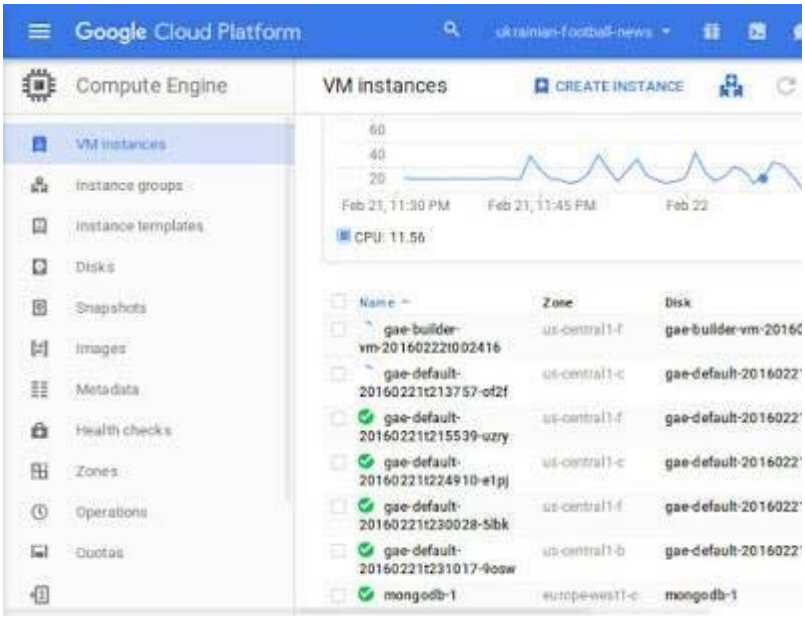

Рис. 4.13. Результат розгортання додатку

Також можливо змінити поточну активну версію додатку (Make Default) рисунок 4.14):

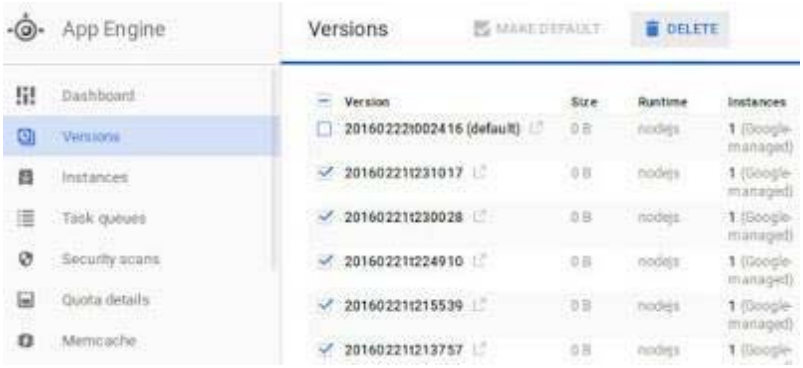

Рис. 4.14. Управління версіями додатку

На цьому завершується процедура розгортання веб-дадатку в Google Cloud. В результаті описаних дій опубліковано додаток Node.js + MongoDB. Додаток працює на двох окремих обчислювальних нодах (compute engine VM instances), але в межах одного проекту. Звичайно, можливі й інші варани розгортання додатків такого типу в залежності від необхідних ресурсів.

4.3. Інструкція користувача

Щоб відкрити веб-додаток, необхідно перейти на адресу у браузері, де розміщений клієнтська частина. Переходячи за адресою, користувач бачить головну сторінку веб-сайту, яка складається з:

1) шапки сайту (рисунок 4.15), яка містить навігацію по додатку у розділи такі як: послуги, портфоліо, контакти, блог, тарифний план, про нас та особистий кабінет.

2) Основної сторінки, на якій знаходиться візуал сайту та додаткова навігація по веб-сайту (рисунок 4.16 , 4.17).

3) секції найпопулярніших страв з короткою інформацією (назва, ціна, рейтинг, фото) про них та кнопки “Show More” за допомогою якої здійснюється редирект на сторінку страв (рисунок 4.18).

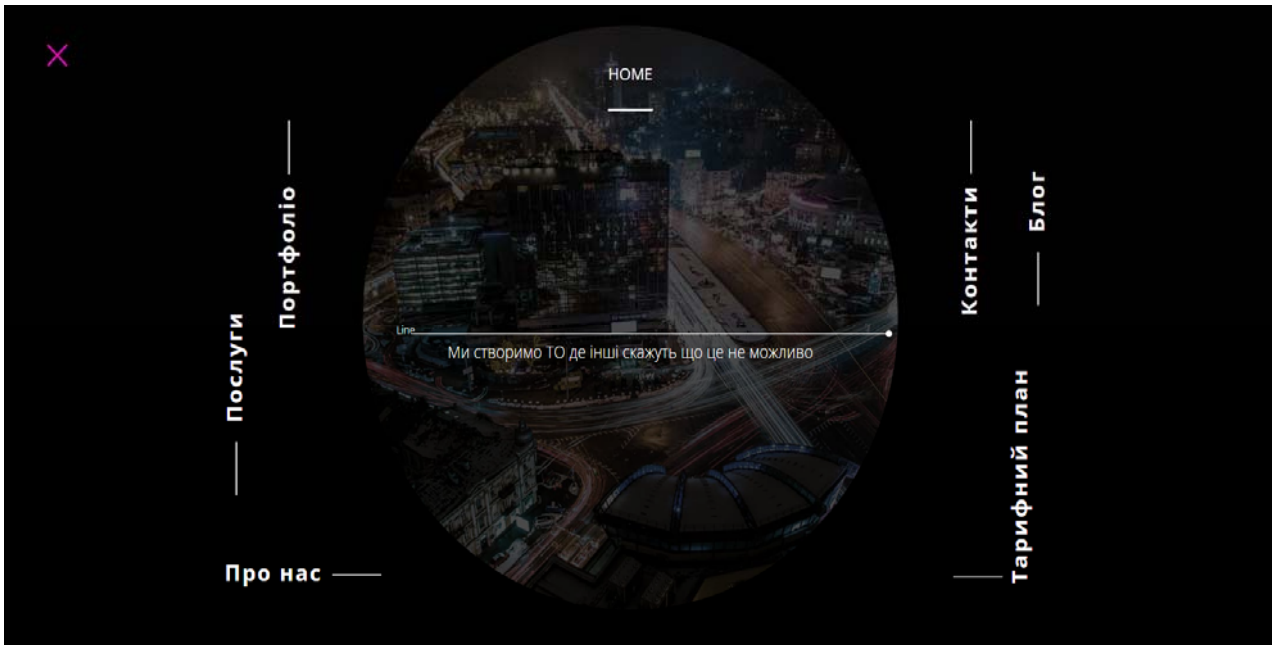

Рис. 4.15. Головна сторінка сайту рекламного агентства «Fantazər»

Рис. 4.16. Шапка сайту рекламного агентства «Fantazər»

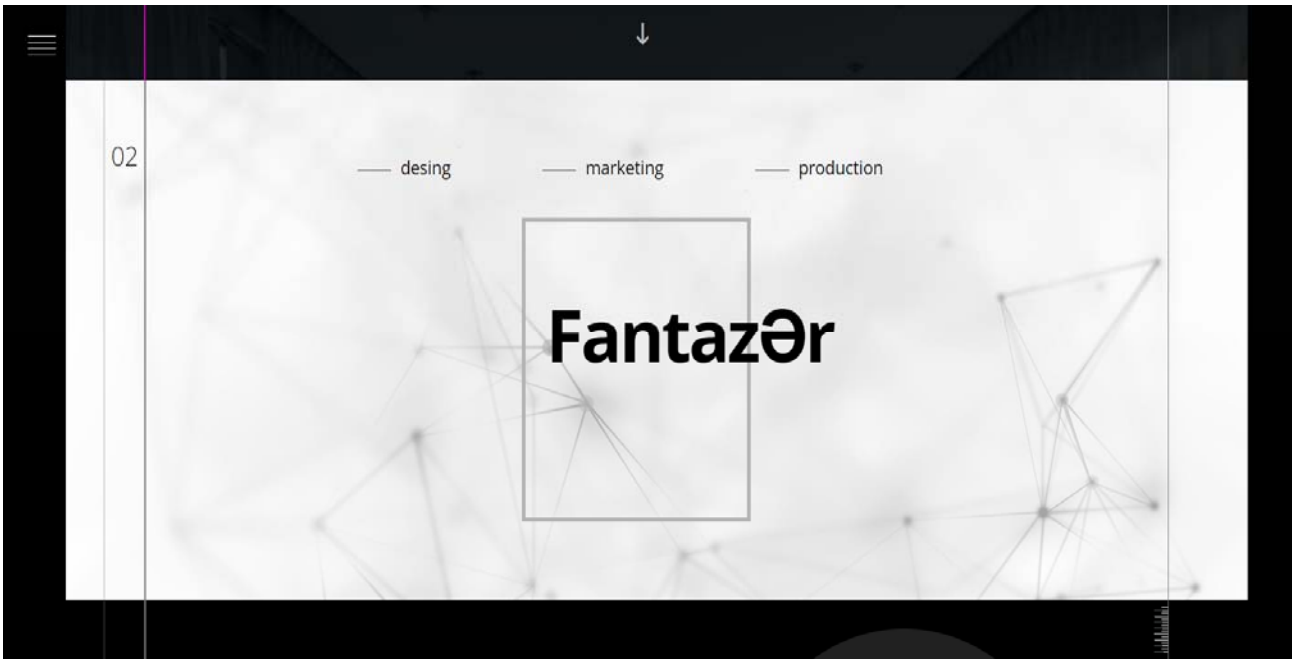

Рис. 4.17. Шапка сайту рекламного агентства «Fantazər»

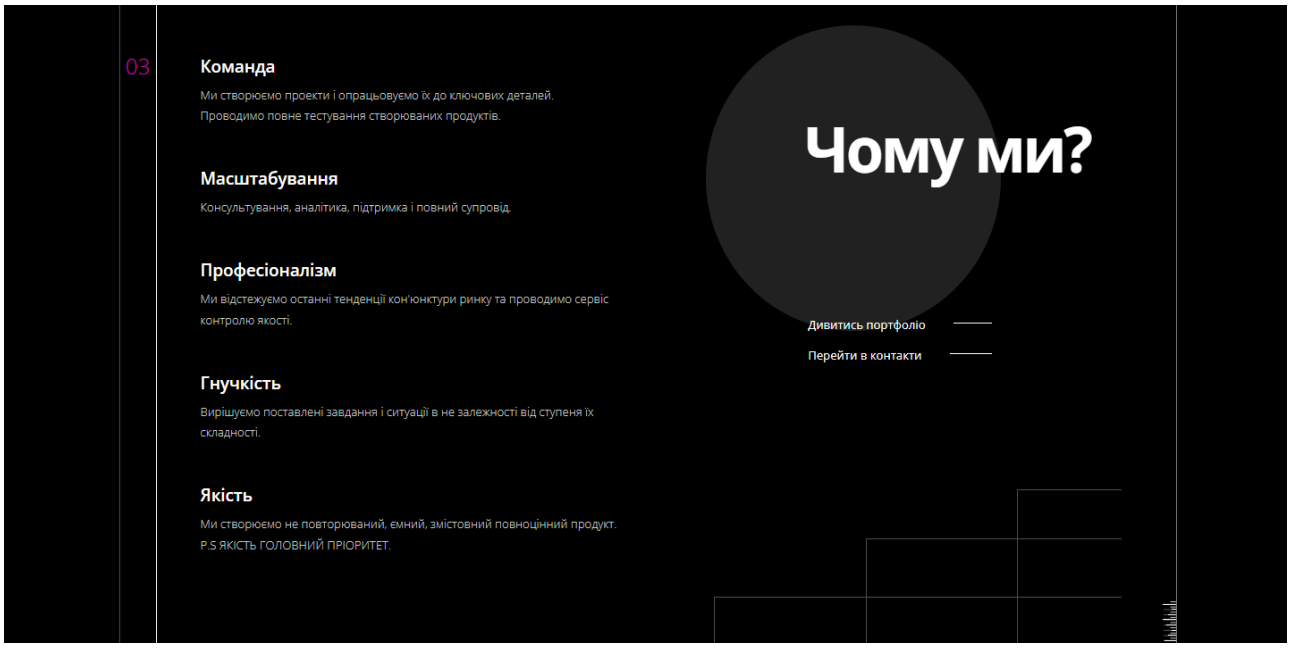

Рис. 4.18. Сторінка сайту “Чому ми?” рекламного агентства «Fantazər»

На даній сторінці користувач може ознайомитись з роботою команди та функціоналом , гнучкістю та якістю які очікують клієнта зображено на рисунку 4.18, також натиснувши на кнопку “Дивитись портфоліо” користувача перекидує на сторінку з роботами рекламного агентства «Fantazər» , а ще можна натиснути кнопку “Перейти в контакти” для зворотнього зв’язку з сайтом.

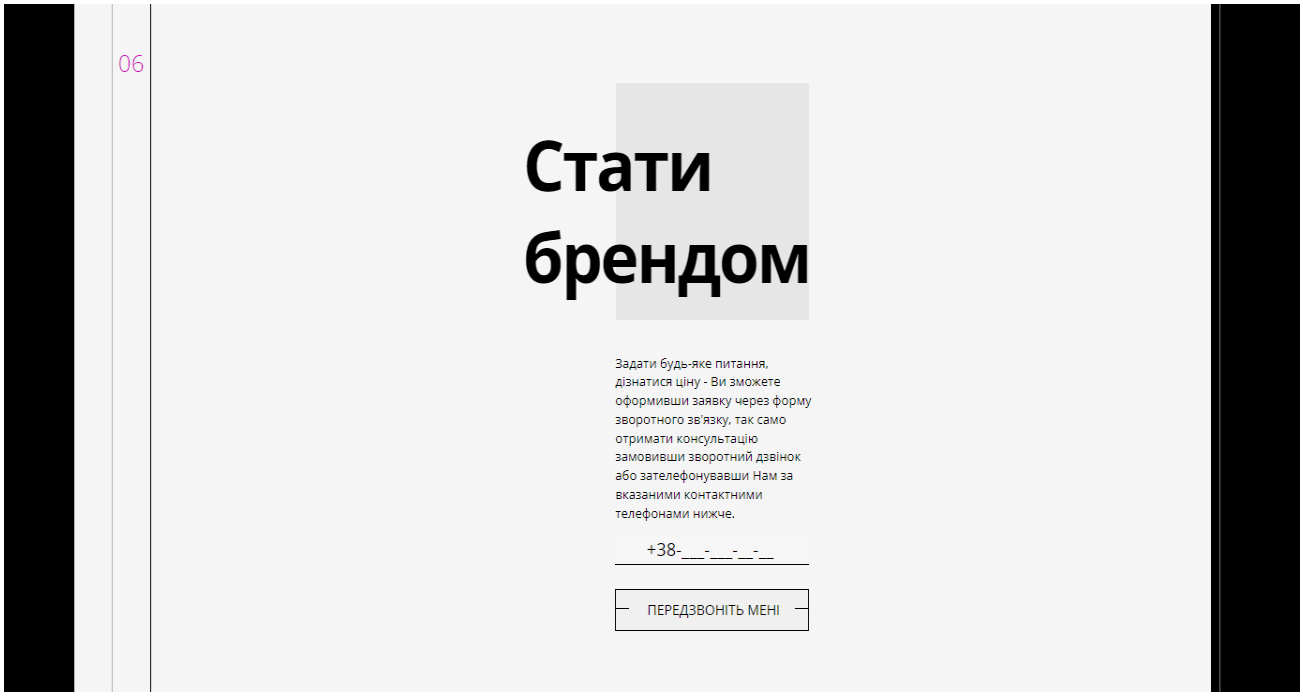

Рис. 4.19. Сторінка “Стати брендом” рекламного агентства «Fantazər»

На цій сторінці зображеній на рисунку 4.19 користувач ознайомлюється з формою зворотного зв’язку, отримавши консультацію замовивши зворотній дзвінок або зателефонувати за вказаним номером, також, на сторінці присутня кнопка “ПЕРЕДЗВОНІТЬ МЕНІ” натиснувши яку , користувач автоматично встає в чергу та очікує дзвінок від адміністратора сайту для подальшої консультації.

На рисунку 4.20 зображено інформацію про компанію «Fantazər»

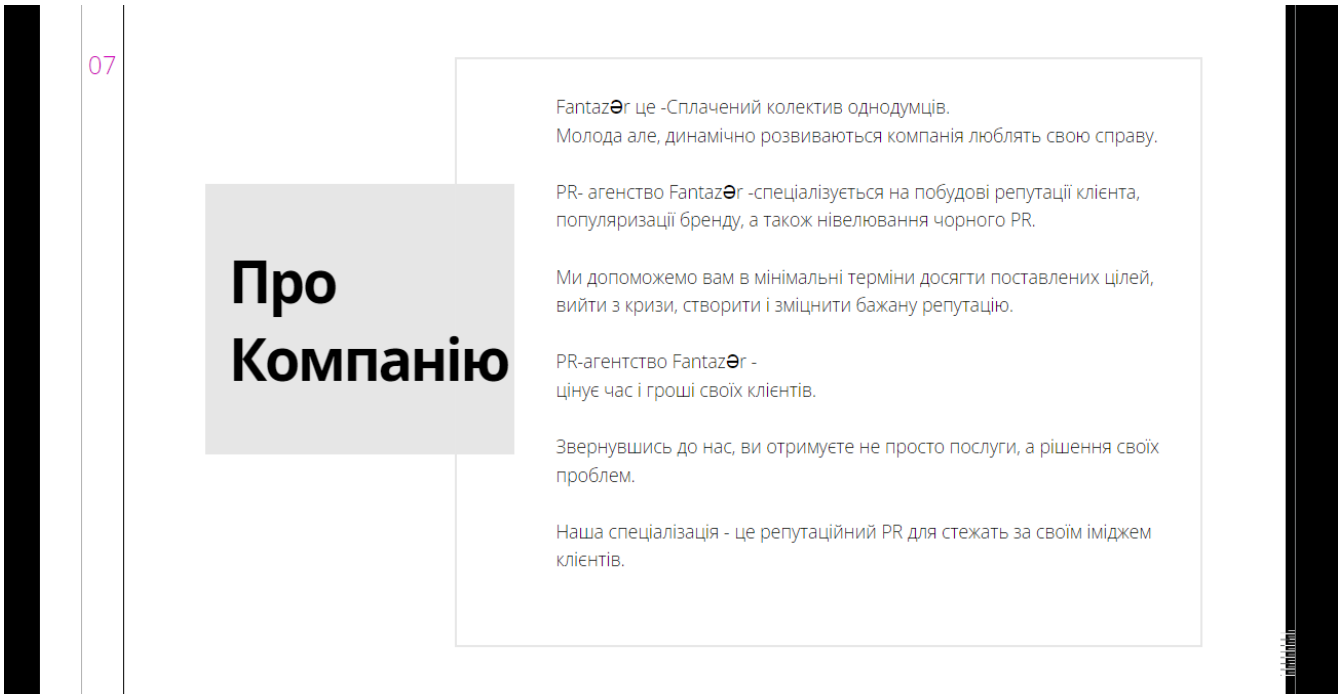

Рис. 4.20. Загальна інформація про компанію «Fantazər»

Наступний крок для клієнта зображений на рисунку 4.21 , де демонструються роботи , рекламного агентства «Fantazər», де користувач ознайомлюється з роботами , також на сторінці є можливість переключатись між брендами за допомогою кнопок “Назад” та “Вперед” .

Рис. 4.21. Сторінка робіт

Щоб підтвердити оформлення бренду, користувачеві потрібно ввести своє ім’я та e-mail адрес та натиснути кнопку відправити для зворотного зв’язку , процес зображений на рисунку 4.22 .

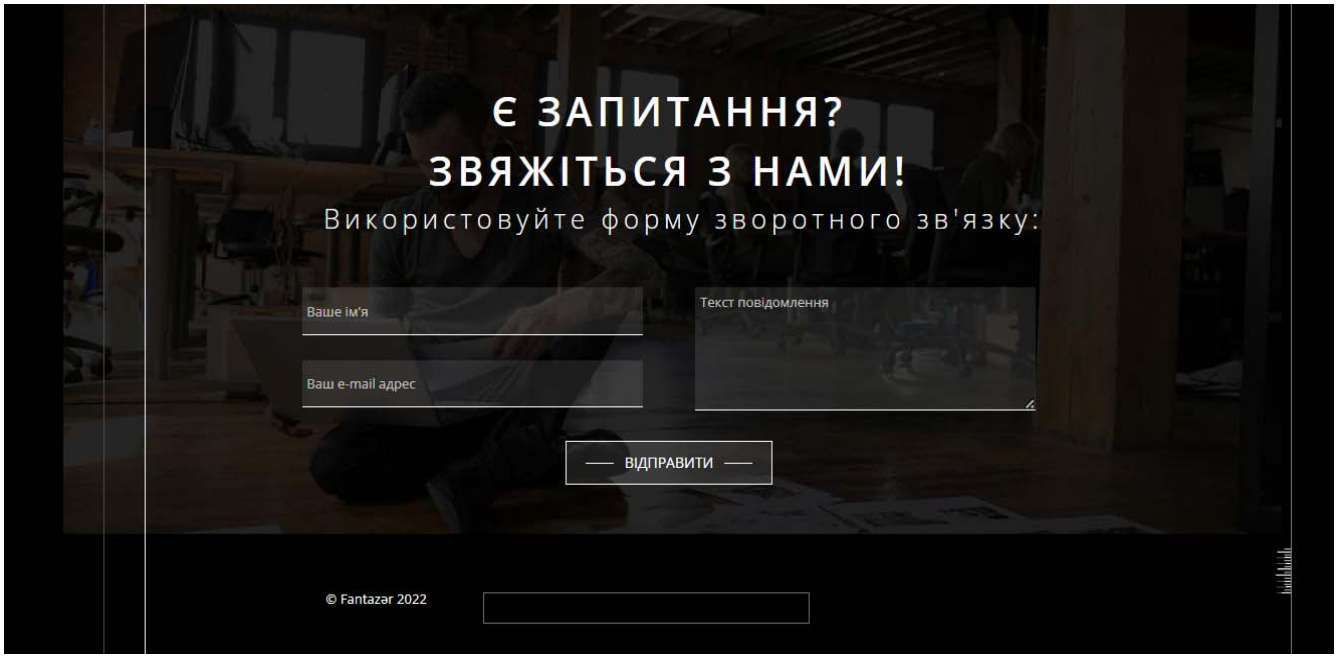

Рис. 4.22. Реєстрація на сайті «Fantazər»

В особистому кабінеті користувач бачить свою коротку інформацію (ім'я, прізвище, номер телефону та пошта), яку він може змінити разом із паролем.

Висновки до розділу 4

В даному розділі було продемонстровано та описано інструкції користування для звичайного користувача та адміністратора.

Інтерфейс звичайного користувача розроблений максимально лаконічно та конструктивно для користувача. Кожна секція, дія, яку необхідно виконати інтуїтивно зрозуміла та проста. Веб-сайт коректно працює на пристроях різного розширення та у різних браузерах. В дизайні веб-додатка було використано різні анімації, плавні переходи, зображення високої якості та можливість переходу на сторінки за допомогою шапки веб-сайту.

ВИСНОВКИ

Метою даної магістерської роботи є розробка веб-додатку для керування рекламним агентство «Fantazər». Розроблений веб-додаток являє собою сучасний веб- сайт, який містить інформацію про діяльність рекламного агентства з можливістю створення власного бренду online для користувача. Відповідно до сучасних стандартів, було розроблено зручний, конструктивний та зрозумілий дизайн системи.

Веб-додаток складається з чотирьох основних частин:

1) Серверна частина;

2) База даних;

3) Клієнтська частина;

4) Admin клієнтська частина;

Для розробки звичайного клієнта було використано React фреймворк та Redux для управління станом додатку. Комунікація з сервером здійснюється за допомогою axios. Для додаткової, спрощеної стилізації та тематизації вебдодатка використано semantic-ui.

Для розробки admin клієнта було використано Angular фреймворк. Комунікація з сервером здійснюється за допомогою вбудованого інструмента HttpClient з пакету @angular/common/http. Для додаткової, спрощеної стилізації та тематизації веб-додатка використано @angular/material.

При розробці серверної частини додатку було детально описані усі API endpoints, через які клієнтські частини можуть працювати з сервером. Серверна частина розроблена за допомогою програмної платформи Node.js та Express (фреймворк веб-додатків для Node.js). Для шифрування інформації використано crypto-js.

Як систему управління базою даних було обрано MongoDB. Основні сутності бази даних: users, places (ресторани), dishes, specialOffers, orders, table.

Інтерфейс звичайного користувача розроблений максимально лаконічно та конструктивно для користувача. Кожна секція, дія, яку

необхідно виконати інтуїтивно зрозуміла та проста. Веб-сайт коректно працює на пристроях різного розширення та у різних браузерах.

Інтерфейс адміністратора розроблений з мінімальною кількістю анімації, зображень для спрощення дизайну та фокусуванням на інструментах керування звичайним клієнтом. В ході виконання магістерського проектування були вивченні та покращені основні теоретичні та практичні знання розробки сучасних веб-сайтів.