НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____Сергій СТІРЕНКО

«___»_____2020 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Веб-додаток для керування системою закладів громадського харчування»

Виконав:
студент IV курсу, групи ІП-62
Олександр Шевчук _____

Керівник:
доцент Олександр Долголенко _____

Консультант з нормоконтролю:

проф. Валерій Сімоненко _____

Рецензент:
доцент Андрій Писаренко _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____*Сергій СТІРЕНКО*
(підпис)
"___"_____2020 р.

ЗАВДАННЯ

на дипломний проект студента

Шевчука Олександра Сергійовича

1. Тема проекту «*Веб-додаток для керування системою закладів громадського харчування»*

керівник проекту Долголенко Олександр Миколайович, доцент, затверджені наказом по університету від «*07*» *травня* 2020р.
№__1081-с__

2. Термін здачі студентом закінченої роботи_____*2020р.*

3. Вихідні дані до проекту *технічне завдання, теоретичні дані.*

4. Зміст пояснювальної записки: *опис предметної області, дослідження архітектури сучасних веб-сайтів, веб-додаток для керування системою закладів громадського харчування.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

6. Дата видачі завдання__*01.09 .2019 року*_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	*Затвердження теми роботи*	*01.09.2019 - 15.12.2019*	
2.	*Вивчення та аналіз завдання*	*15.12. 2019 - 15.03.2020*	
3.	*Розробка архітектури та загальної структури систем*	*15.03. 2020 - 25.03.2020*	
4.	*Розробка модулів системи*	*25.03 2020 - 5.04.2020*	
5.	*Програмна реалізація системи*	*5.04.2020-15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04. 2020 - 20.05.2020*	
7.	*Передзахист*	*29.05.2020*	
8.	*Захист*	*19.06.2020*	

Студент Шевчук Олександр _____

(підпис)

Керівник роботи Долголенко Олександр_____

(підпис)

Анотація

В бакалаврській *дипломній* роботі реалізовано Веб-додаток для керування системою закладів громадського харчування.

Програма дозволяє користувачу робити замовлення у ресторанах свого міста, обираючи свої найулюбленіші страви, стіл, годину, дату та кількість гостей з можливістю оплати замовлення. Адміністратор додатка має можливість редагувати, видаляти, додавати страви та спеціальні акції у додатку. Програмний продукт був створений з застосуванням мови програмування Javascript. Серверна частина додатку була створена на платформі Node.js. Для створення користувацького інтерфейсу був використаний React.js та Angular 8 для інтерфейсу адміністратора.

Аннотация

В бакалаврской дипломной работе реализовано Веб-приложение для управления системой заведений общественного питания.

Программа позволяет пользователю делать заказ в ресторанах своего города, выбирая свои любимые блюда, стол, время, дату и количество гостей с возможностью оплаты заказа. Администратор приложения имеет возможность редактировать, удалять, добавлять блюда и специальные акции в приложении. Программный продукт был создан с применением языка программирования Javascript. Серверная часть приложения была создана на платформе Node.js. Для создания пользовательского интерфейса был использован React.js и Angular 8 для интерфейса администратора.

Annotation

In this work for a Bachelor's Degree, a Web Application for managing catering catering establishments was implemented.

The program allows the user to make an order in the restaurants of his city, choosing his favorite dishes, table, hour, date and number of guests with the possibility of paying for the order. The application administrator has the ability to edit, delete, add dishes and special promotions in the application. The software product was created using the Javascript programming language. The server part of the application was created on the Node.js platform. React.js was used to create the user interface and Angular 8 for the admin interface.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
	А 4		Завдання на дипломний проект	2	
	А4	ІАЛЦ.467200.001 ОП	Опис проекту	1	
	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	3	
	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	66	
	А3	ІАЛЦ.467200.004 Д1	Схема моделі даних	1	
	А4	ІАЛЦ.467200.005 Д2	Схема алгоритму реєстрації	1	
	А3	ІАЛЦ.467200.006 Д3	Схема алгоритму встановлення нового пароля	1	
	А4	ІАЛЦ.467200.007 Д4	Лістинг	6	

Технічне завдання

до дипломного проекту

на тему: «Веб-додаток для керування системою закладів громадського харчування»

Київ – 2020

Зміст

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-додаток для керування системою закладів громадського харчування

Область застосування: управління та адміністрування у ресторанному бізнесі.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ

Метою даного проекту є розробка Веб-додатку для керування системою закладів громадського харчування.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, довідники, публікації в Інтернеті по опису архітектури і принципу побудови та розробки веб-додатків, а саме: клієнтської та серверної частин.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмної моделі

Програмна система має мати наступний функціонал:

- Можливість реєстрації у веб-додатку та мати особистий кабінет
- Можливість зробити замовлення
- Можливість управляти контентом веб-додатка

5.2. Вимоги до програмного забезпечення

- Операційна система Linux/Windows/MacOS;
- Node.js, React.js, MongoDB, Angular;

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium G840 і вище;
- більше 4 ГБ оперативної пам’яті;
- більше 8 ГБ вільного простору на диску ;
- Підключення до Інтернету;

6. **ЕТАПИ РОЗРОБКИ**

Назва етапу	Дата
Вивчення джерел за тематикою роботи	10.01.2020
Розроблення і узгодження технічного завдання	20.02.2020
Моделювання структури програмного забезпечення	03.03.2020
Розробка програмного забезпечення	25.03.2020
Тестування системи	01.05.2020
Виправлення помилок	15.05.2020
Оформлення документації дипломної роботи	25.05.2020

Пояснювальна записка

до дипломного проекту

на тему: «Веб-додаток для керування системою закладів громадського харчування»

Київ – 2020

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

HTTP (Hyper Text Transfer Protocol) - поширений протокол передачі даних

HTTPS (Hyper Text Transfer Protocol Secured) - розширення протоколу HTTP для підтримки шифрування з метою підвищення безпеки документів.

API (Application Programming Interface) - код, який дозволяє двом програмним програмам спілкуватися один з одним

REST API (Representational State Transfer) - архітектурний стиль взаємодії компонентів розподіленого додатка в мережі.

SPA (Single page Application) - додаток, який працює всередині браузера і не потребує перезавантаження сторінки під час використання

API endpoint - це точка, в якій інтерфейс прикладної програми (API) підключається до програмного забезпечення.

CSR (Client Side Rendering) - вид візуалізації, що дозволяє робити веб-сайти повністю виведеними в браузері за допомогою JavaScript.

JS (Javascript) - мова програмування

JSON (Javascript Object Notation) - це легкий формат для зберігання та транспортування даних

npm (Node package manager) - це програмне забезпечення, яке дозволяє встановлювати та керувати залежностями проекту.

yarn (Yet Another Resource Negotiator) - це альтернатива до npm.

ВСТУП

Метою даної дипломної роботи є розробка веб-додатку для керування системою закладів громадського харчування. Проаналізувавши існуючі аналоги, був розроблений відповідний, сучасний та зрозумілий дизайн веб-додатка.

Розроблений веб-додаток являє собою веб-сайт, який містить заклади громадського харчування певного міста з можливістю створення замовлень online для користувача. На веб-сайті описана актуальна коротка інформація про заклади громадського харчування, така як:

1) Місцезнаходження
2) Меню страв
3) Спеціальні акції та знижки

Веб-додаток можна легко масштабувати, підлаштовувати під конкретний бізнес або мережу закладів громадського харчування, що дійсно допоможе підприємцям у керуванні ними, являється правильною та сучасною маркетинговою стратегією, швидко принесе додатковий наплив клієнтів та, звісно, прибуток.

В ході виконання дипломної роботи були поставлені наступні завдання:

1) Аналіз існуючих аналогів у мережі Інтернет.
2) Створення відповідного сучасного дизайну веб-додатка.
3) Проектування структури та архітектури веб-сайту.
4) Вибір технологій та додаткових бібліотек для розробки веб-сайту.
5) Розробка веб-додатку: створення серверної та клієнтських частин для звичайного користувача та адміністратора, заповнення та створення бази даних, розміщення в Інтернеті.
6) Тестування функціональності, перевірка коректності роботи в різних браузерах та на різних девайсах, усунення виявлених неполадок.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ ВЕБ-ДОДАТКІВ

1.1 Визначення веб-додатку

Веб-додаток - клієнт-серверний додаток, в якому клієнт взаємодіє з веб-сервером за допомогою браузера.

Іншими словами, веб-додаток - це веб-сайт.

Для перегляду веб-сайту нам потрібен браузер (Chrome, Internet Explorer, Edge, Safari, Firefox та інші). Ви можете відкрити веб-сайт, ввівши URL-адресу в адресному рядку. Наприклад, набравши "https://www.youtube.com", відкриється домашня сторінка Youtube. Якщо ви не знаєте URL веб-додатку, який хочете відвідати, ви можете скористатися пошуковою системою, щоб знайти веб-сайт в Інтернеті.

Практично будь-який інтернет-ресурс - це веб-додаток. Це пошукові системи, відео сервіси типу youtube, соцмережі, будь-які веб-сайти з функціями аутентифікації користувача, покупки, замовлення, бронювання.

Сьогодні в Інтернеті є мільярди веб-сайтів, які можна розділити на декілька типів категорій: блог, архів, бізнес-сайт та корпоративний веб-сайт, інформаційний веб-сайт, ігровий веб-сайт, довідковий веб-сайт, веб-сайт Q&A, веб-сайт для обміну медіа, веб-сайт новин, персональний веб-сайт, навчальний веб-сайт, пошукові системи, соціальні мережі, веб-пошти та багато інших.

Деякі веб-сайти можна віднести до більш ніж однієї з категорій. Наприклад, веб-сайт також може бути форумом, веб-поштою, блогом або пошуковою системою. Спектр використання веб-додатків дуже широкий.

Також є поняття веб-сторінки. Веб-сайт посилається на центральне місце, яке містить більше однієї веб-сторінки або серії веб-сторінок. Наприклад, Youtube вважається веб-сайтом, який містить мільйони різних веб-сторінок.

1.2 Структура веб-додатку

Сторінки у браузері можуть бути статичними і динамічними.

Статична web-сторінка відображається для всіх відвідувачів однаково. Як це працює:

1) Людина вводить в адресному рядку запит або адреса сторінки.
2) Браузер відправляє його на веб-сервер.
3) Той аналізує запит, визначає, що ніяких особливих ознак і інструкцій немає.
4) Відправляє веб-сторінку браузеру без зміни будь-яких даних на ній. Наприклад, це новинний матеріал, загальна стандартна інформація.

У випадку з динамічними сторінками схема виглядає так:

1) Браузер відправив запит на веб-сервер. Наприклад, при цьому надійшла інформація, що у цього користувача є набір ознак, при наявності яких для нього потрібно показувати певну інформацію, значить сторінка буде динамічною.
2) Веб-сервер пересилає її на сервер додатків, де спеціальне програмне забезпечення застосує правила та інструкції для додавання особливих змінних. Наприклад, користувач авторизований в системі. Йому може показатись сторінка з ПІБ та іншою релевантною інформацією.
3) Сервер забирає готову веб-сторінку, віддає браузеру, який показує її відвідувачеві, що створив запит.

Розглянемо приклад посилання на веб-додаток - https://www.test.com/page1.html. URL-адреси веб-сайт є “test.com”, а веб-сторінка - “page1.html”. Слід також зауважити, для веб-сторінки не обов'язково потрібне розширення файлу, наприклад .htm або .html, щоб бути веб-сторінкою. Багато сайтів розроблені налаштовані так, щоб не було розширень файлів або мають сторінки за замовчуванням у каталозі (наприклад, index.html).

Інформація на веб-сторінці відображається в Інтернеті за допомогою веб-браузера (наприклад Chrome), який з'єднується з сервером, на якому

розміщується вміст веб-сайту через протокол передачі гіпертексту (HTTP). Наприклад, розглянемо будь-яку URL-адресу веб-сторінки, вона матиме префікс "http" або “https”, який повідомляє браузеру, який протокол використовувати для виконання конкретного запиту URL-адреси.

Для того, щоб побудувати веб-сайт потрібно не лише програмний продукт, а і доменне ім'я та хостинг веб-сайту.

Доменне ім’я - це ім'я та адреса веб-сайту. Цю адресу відвідувачі використовують, коли намагаються знайти сайт через веб-браузери. Приклад доменного імені - youtube.com. Доменне ім'я зазвичай платне, але є безкоштовні імена, які надають різні хостингові сервіси наприклад heroku, firebase та інші, при цьому адреса на безкоштовному хостинг-сервісі буде видозмінена.

Є два типи розширень доменних імен: загальні та локальні. Слід використовувати "загальні" розширення доменного імені, наприклад .com, .net або .org, якщо вашою метою є міжнародні відвідувачі. А використовувати "локальні" розширення доменного імені, наприклад .de, .fr або .ru, якщо вашою метою є відвідувачі, орієнтовані на конкретну країну.

Крім доменного імені, також необхідним є веб-хостинг.

Веб-хостинг - це послуга, яка розміщує та зберігає файли веб-додатку на захищеному сервері, який постійно працює. Без веб-хостингу сайт не буде доступний для читання та перегляду для інших.

Варто піклуватись про додаткові особливості, наприклад доменне ім’я з SSL (для безпеки). Основна причина, по якій використовується SSL - це зберігання конфіденційної інформації, що надсилається через Інтернет, зашифрованою, щоб отримати доступ до неї лише призначений одержувач. Коли використовується сертифікат SSL, інформація стає нечитабельною для всіх, крім сервера, на який ви надсилаєте інформацію.

Веб-додатки часто розділяють на клієнтську частину (фронтенд) та серверну (бекенд).

Клієнтська частина веб-сайту - це те, що бачить користувач у браузері. Це включає все: від тексту та кольорів до кнопок, зображень та меню навігації.

Найбільш відомі та популярні технології, які використовуються при створенні будь-якого веб-сайту - це HTML, CSS, Javascript.

Вміст кожної веб-сторінки зазвичай представлений у форматі HTML, що дозволяє легко структурувати інформацію та швидко читати веб-браузером клієнта. За допомогою CSS (Cascading Style Sheets) дизайнери можуть точно контролювати зовнішній вигляд веб-сторінки, що стосується макета, кольорової гами та навігації. Інструкції CSS можуть бути або вбудовані у веб-сторінку HTML, або можуть бути включені в окремий зовнішній файл.

Серверна частиною веб-сайту (бекенд) - те, що користувач не бачить. Сервер несе відповідальність за зберігання та впорядкування даних та забезпечення того. Сервер зв’язується з клієнтською частиною, надсилаючи та отримуючи інформацію для відображення у вигляді веб-сторінки. Щоразу, коли користувач заповнює контактну форму, вводить веб-адресу або здійснює покупку (будь-яка взаємодія користувача на стороні клієнта), браузер надсилає запит на сторону сервера, який повертає інформацію у вигляді коду фронтенда, який браузер може інтерпретувати та відображати.

Є веб-додатки, які не мають серверної частини. Вони є прикладом статичних веб-сайтів - вміст яких насправді не змінюється. Для статичних веб-сайтів вся необхідна інформація, яка визначає, що знаходиться на веб-сторінці, міститься в самому коді клієнтської частини. Статичні веб-сайти можна використовувати для показу таких речей, як бізнес, ресторани, портфелі або професійні профілі. Але якщо потрібно перетворити сайт у щось, з чим можуть взаємодіяти користувачі, нам буде необхідний сервер.

Також веб-додатку потрібна база даних для управління всією інформацією про клієнта та товар. База даних зберігає вміст веб-сайту у такій структурі, яка дозволяє легко отримувати, упорядковувати, редагувати

та зберігати дані. Існує багато різних систем управління базами даних, які широко використовуються, такі як Oracle, MySQL, SQL Server, PostgresSQL та MongoDB.

З базою даних буде “спілкуватись” наша серверна частина додатку. Бекенд частину слід створити за допомогою мови, яку може розпізнати база даних. Деякі поширені мови запуску - Ruby, PHP, Java, .Net та Python. Ці мови програмування часто працюють разом з фреймворками, що спрощують процес веб-розробки. Rails, наприклад, є фреймворком, написаною в Ruby. Звідси названа Ruby on Rails - популярна технологія для створення динамічних веб-додатків, що робить процес набагато швидшим.

1.3 Популярність та необхідність у сучасному світі

Щоб наглядно показати наскільки веб-додатки популярні на сьогоднішній день приведемо деякі дані з статистики [1]:

1) Станом на січень 2020 року в Інтернеті було понад 1,74 мільярда веб-сайтів. info.cern.ch був першим в історії веб-сайтом в Інтернеті, опублікованим 6 серпня 1991 року.
2) Станом на 5 січня 2020 року в Інтернеті було 4,437,215,927 (4+ мільярдів) користувачів Інтернету.
3) Зараз Google обробляє понад 7 мільярдів пошукових запитів на день у всьому світі (хоча деякі кажуть, що це може бути до 10 мільярдів на день). 15% цих запитів раніше не шукали в Google
4) У III кварталі 2019 року було зареєстровано 359,8 млн. доменних імен.
5) Очікується, що цифрові медіа принесуть 51% або понад 240 мільярдів доларів усіх рекламних грошей, витрачених у світі в 2019 році.
6) Приблизно 40% населення світу мали інтернет-зв’язок у 2018 році. У 1995 році він становив менше 1%.

7) Пошукова реклама є найпопулярнішим видом цифрової реклами на даний момент. У 2018 році витрати на рекламу становили 113 мільярдів доларів.
8) В Інтернеті щодня публікується понад 5 760 000 публікацій блогів.
9) Загалом у світі існує приблизно 333,8 млн. реєстрацій доменних імен, і це число постійно збільшується приблизно на 1,0% щороку.
10) У 2020 році у світі налічується понад 600 мільйонів блогів. У США
11) Загалом у світі існує приблизно 333,8 млн. реєстрацій доменних імен, і це число постійно збільшується приблизно на 1,0% щороку.
12) У в 2018 році в Інтернеті купували 47,3% світового населення.
13) Зараз у Facebook є 2,27 мільярда користувачів.
14) Люди проводять в середньому 2 години 15 хвилин на день у мережах соціальних мереж.

Дивлячись на ці цифри та статистику, стає зрозумілим та очевидним, що Інтернет та веб-додатки стали невід'ємною частиною нашого життя. Ми використовуємо веб-сайти для розваг, навчання, роботи, бізнесу, здійснення купівлі товарів, оплати будь-яких послуг та інше.

Відповідно, веб-додатки стають все більше і більше необхідними та постійно змінюються. Це означає, що тип контенту, маркетингові стратегії, спосіб взаємодії людей зі своїми мобільними пристроями, те, як споживачі здійснюють покупки на веб-сайтах та способи пошуку інформації, також постійно змінюються.

1.4 Аналіз існуючих аналогів

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

Все більше і більше сучасні веб-сайти створені з використанням SPA-підходу. Single page application (SPA) - це підхід до розробки веб-сайтів, де вміст кожної нової сторінки подається не з завантаження нових HTML-сторінок, 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 URL-адреси 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 закладів громадського харчування, 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 меню. 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 веб-сайту. 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 про заклад таку як: меню страв, години роботи, контакти, адресу, фото закладів внутрі та інше.

1.5 Короткий огляд проекту

Проект являє собою веб-сайт закладів громадського харчування. Даний веб-додаток надає повну інформацію про ресторани та їхні меню, можливість здійснювати онлайн замовлення, створення особистого кабінету для користувача з відслідковуванням попередніх зроблених замовлень та налаштуванням профілю та керування змісту сайту адміністратору. При створенні веб-додатку використані та реалізовані найважливіші аспекти і підходи сучасних веб-сайтів, для того щоб він відповідав сучасним стандартам та був конкурентоспроможним у глобальній мережі Інтернет.

До веб-сайту розроблений відповідний веб-дизайн, який є ясним, зручним з точки зору юзабіліті.

Веб-додаток не обмежується одним закладом громадського харчування, на ньому можуть бути розташовані цілі мережі закладів.

У процесі аналізу цілей сайту і його структури було визначено наступні вимоги до верстки сайту:

- мінімальне розширення 1024 × 768 пікселів;
- верстка сайту оптимізована на коректну роботу популярних браузерів (Opera, Mozilla FireFox, Chrome);

Веб-сайт розроблений з використанням сучасного SPA-підходу (Single Page Application) та використовує наступні технології:

- Javascript

- HTML
- SCSS
- React.js
- Angular 8
- Node.js
- MongoDB
- Express.js

Для розміщення веб-сайту використано безкоштовні хостинги, які надають firebase.google.com та heroku.com.

У зв'язку з неймовірною популярністю соціальних мереж та електронних пошт, додана можливість авторизації через “Gmail” та “Facebook”.

Висновок до розділу 1

В даному розділі було розглянуто структуру веб-додатку. Досліджено принципи його роботи, було розглянуто існуючі аналоги, їх переваги та недоліки та назначені основні вимоги по технологіям та функціоналу до веб-сайту.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Архітектура веб-додатку

Як і більшість сучасних веб-сайтів, веб-додаток побудований як Single-page 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.

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.

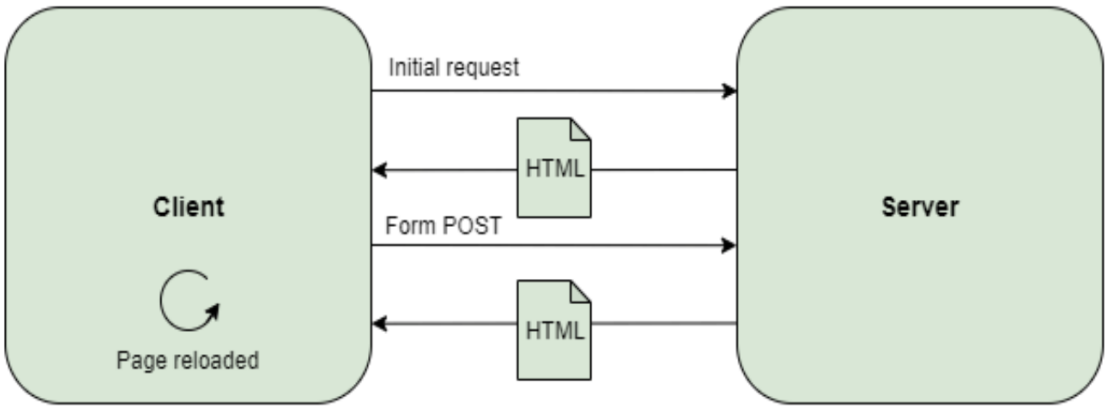


Рисунок 2.1 Single Page Application (SPA)

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.

2.2 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 веб-сторінок, 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

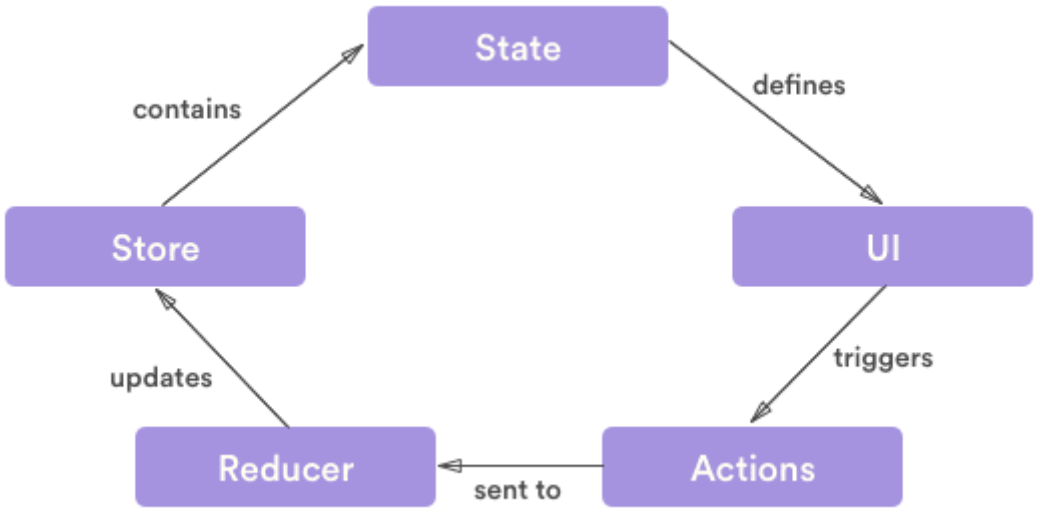


Рисунок 2.2 Життєвий цикл Redux

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 веб-сайтів. Semantic UI дозволяє розробникам створювати веб-сайти з швидким і стислим HTML

2.3 Адмін-клієнт

Клієнтський веб-додаток для адміністратора також має CSR та являється окремим сервісом. Використовуються HTML, CSS та JavaScript. Для адмін-додатка 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.

2.4 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 URL-адресою. 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.

2.5 База даних

Веб-додатку потрібне місце для зберігання даних, а саме: інформація про страви, ресторани, користувачів, замовлення, спеціальні акції. Існує два типи баз даних: реляційні та нереляційні, кожна з яких має свої плюси і мінуси.

Оскільки, як клієнт так і сервер побудований за допомогою мови програмування Javascript, використання документоорієнтованого зберігання даних (дані зберігаються у вигляді документів у стилі JSON) спростить та пришвидшить розробку веб-додатку. Для цих цілей обрано нереляційну базу даних MongoDB. Розглянемо інші основні переваги даної системи управління базами даних [11]:

- Зрозуміла структура одного об’єкта
- Відсутність складних join-запитів
- Легке налаштування та масштабування
- Не потрібне перетворення об'єктів програми в об'єкти бази даних
- Використання внутрішньої пам'яті для зберігання (віконного) робочого набору, що забезпечує швидший доступ до даних
- Глибокий запити. MongoDB підтримує динамічні запити щодо документів, використовуючи мову запитів на основі документа, яка майже така ж потужна, як і SQL.
- Відсутність схем (одна колекція містить різні документи). Кількість полів, вміст і розмір документа можуть відрізнятися від одного документа до іншого.

Висновок до розділу 2

В даному розділі визначено архітектуру веб-додатку та API сервера, було розглянуто основні технології для розробки клієнтських частин для звичайного користувача та адміністратора, серверної частини та бази даних. Обґрунтовано вибір технологій, основуючись на їх основних перевагах та недоліках.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Клієнт

Для того щоб створити SPA-додаток, написаний за допомогою React.js бібліотеки, використано набір інструментів 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'));
};
```

3.2 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 user-route 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);
 };
```

```
exports.updateUserInfo = async (req, res) => {
 queryWrapper(req, res, userService.updateUserInfo, userService.errHandler);
};
exports.userRegistration = async (req, res) => {
 queryWrapper(req, res, () => userService.registerNewUser(User, req),
userService.errorHandler);
};
exports.authenticate = async (req, res) => {
 queryWrapper(req, res, () => userService.authUser(req), userService.errorHandler);
};
exports.verification = async (req, res) => {
 queryWrapper(req, res, () => userService.verificationEmail(req, res));
};
exports.sendEmailToResetPassw = async (req, res) => {
 queryWrapper(req, res, () => userService.sendEmailToResetPassw(req),
userService.errorHandler);
};
exports.resetPassword = async (req, res) => {
 queryWrapper(req, res, () => userService.resetPassword(req, res),
userService.errorHandler);
};
```

Для зручної роботи з нереляційною базу даних MongoDB використовується бібліотека Mongoose, яка управляє зв'язками між даними, забезпечує перевірку схеми і використовується для перекладу між об'єктами в коді та представлення цих об'єктів у базі даних. До кожної сущності у базі даних є відповідна схема. Наступний фрагмент коду демонструє схему документа користувача:

```
const mongoose = require('mongoose');
const { Schema } = mongoose;
const usersSchema = new Schema(
 {
   firstName: {
     type: String,
     default: ""
   },
   lastName: {
     type: String,
```

```
      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.

3.3 API серверу

Розглянемо основні API endpoints нашого серверу:

Таблиця 3.1 API endpoints

url	method	description

/dishes	GET	отримати усі страви
/dishes	POST	створити страву
/dishes/:dishId	GET	отримати окрему страву
/dishes/:dishId	PUT	оновити окрему страву
/dishes/:dishId	DELETE	видалити окрему страву
/dishes/:filter/:count/:category	GET	отримати список відфільтрованих страв
/places	GET	отримати всі ресторани
/places	POST	створити ресторан
/places/:placeId	GET	отримати окремий ресторан
/places/:placeId	PUT	оновити окремий ресторан
/places/:placeId	DELETE	видалити окремий ресторан
/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
/tables	GET	отримати список всіх столів

Продовження таблиці 3.1 API endpoints

url	method	description
/tables/:restId	GET	отримати список столів окремого ресторану
/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.

3.4 База 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.

Створено єдину базу даних, яка містить декілька колекцій:

- users
- places (ресторани)
- dishes
- specialOffers
- orders
- tables

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.

3.5 Admin клієнт

Для керування та встановлення пакунків та бібліотек на Admin-клієнті використано yarn (як на стороні звичайного клієнта). Angular предоставляє альтернативний інструмент до Create React App, за допомогою якого можно легко створювати SPA-додаток - Angular CLI.

Щоб створити каркас проекту, слід виконати наступні команди:

- `npm install -g @angular/cli`
- `ng new my-first-project`

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

- @angular/material - інфраструктура компонентів інтерфейсу та компонент Material Design для Angular веб-додатків.
- crypto-js - Javascript бібліотека для шифрування
- @angular/router - предоставляє основний функціонал маршрутизації для Angular

На відміну від звичайного клієнта, Admin-клієнт здійснює HTTP-запити для отримання або збереження даних реалізовано за допомогою бібліотеки нативного HTTP-клієнта. Наступний фрагмент коду містить функцію, яка робить запит на сервер, щоб аутентифікувати адміністратора:

```
import {HttpClient} from '@angular/common/http';
authenticateAdmin(name, password) {
   const admin = { name, password };
   return this.http.post(`${this.url}admin`, admin);
 }
```

де this.http містить екземпляр HTTP-клієнта, а this.url - адреса сервера.

Шифрування інформації адміністратора відбувається так як на звичайному клієнті:

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

Висновок до розділу 3

В даному розділі було продемонстровано та описано імплементації звичайного і admin клієнтських, сервеної систем, хостинг бази даних на удаленному сервері, додаткові бібліотеки, фреймворки та інші залежності.

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 веб-додатка 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. Оскільки admin клієнт значно менший від звичайного клієнта, не було використано додаткового інструмента для управління станом додатку.

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, tables

РОЗДІЛ 4
ІНСТРУКЦІЯ КОРИСТУВАЧІВ

4.1 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, яка містить навігацію по додатку у розділи такі як: меню, усі ресторани, замовлення, акції, авторизація та особистий кабінет

Рисунок 4.1 Шапка сайту

2) рекламного банера, на якому знаходяться спеціальні пропозиції від закладів та додаткова навігація по веб-сайту

Рисунок 4.2 Рекламний баннер сайту

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.

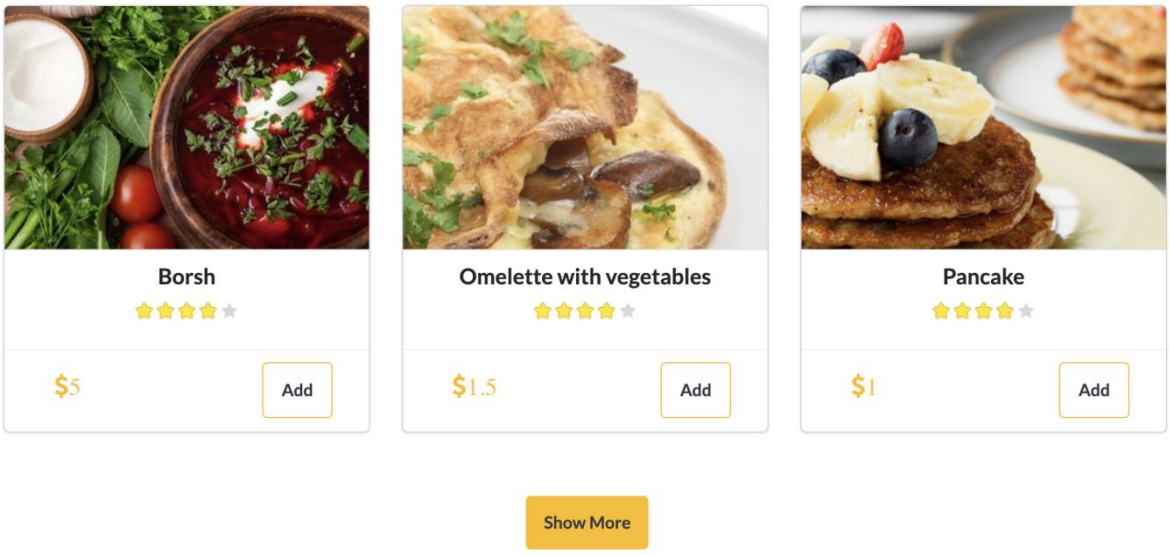


Рисунок 4.3 Найпопулярніші страви

4) секції ресторанів з короткою інформацію (адреса, фото, робочі години) про них та кнопки “View on map” за допомогою якої здійснюється редирект на сторінку ресторанів.

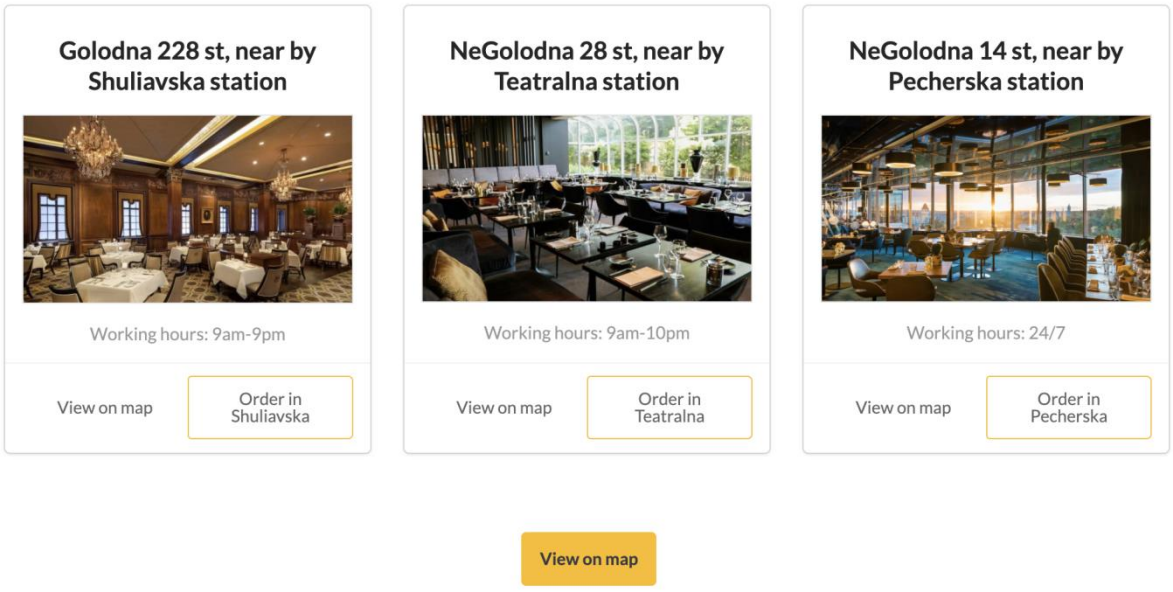


Рисунок 4.4 Ресторани

Для здійснення замовлення, користувач може натиснути “Book a table” у шапці сайту або у секції ресторанів на головній сторінці обрати один з них та натиснути кнопку “Order in ...”. У першому випадку, процес замовлення

почнеться з обиранням ресторану. Для того щоб перейти на наступний крок, слід натиснути “Next”.

Рисунок 4.5 Переключатель кроків замовлення

Наступним кроком являється вибір дати, години, кількості відвідувачів та столика ресторану.

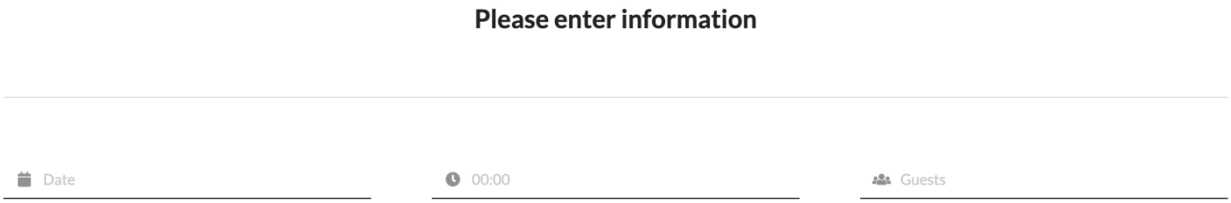


Рисунок 4.6 Вибір дати, години, кількості відвідувачів

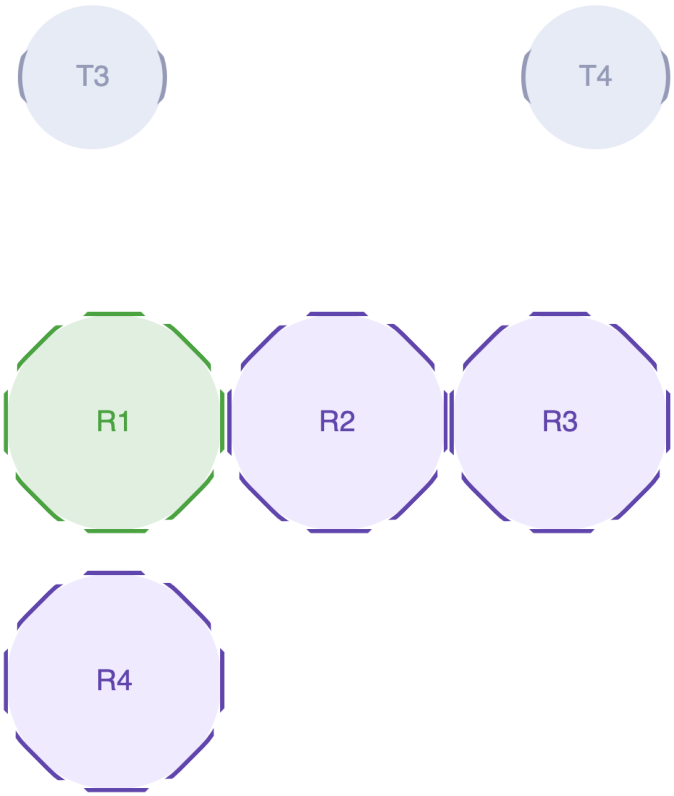


Рисунок 4.7 Вибір столика ресторану

Наступним кроком являється вибір страв. Користувач бачить вже раніше замовлені ним страви, оскільки жодної страви не було замовлено, слід натиснути кнопку “Add dishes!", після чого його буде перенаправлено на сторінку страв. На цій сторінці, користувач обирає страви або видаляє їх. Також для зручності пошуку необхідних страв реалізовано їх фільтрування за абеткою та ціною, роздрібненість по категоріям. Щоб завершити процес вибору страв, слід натиснути іконку “Корзина” в лівому верхньому углі.

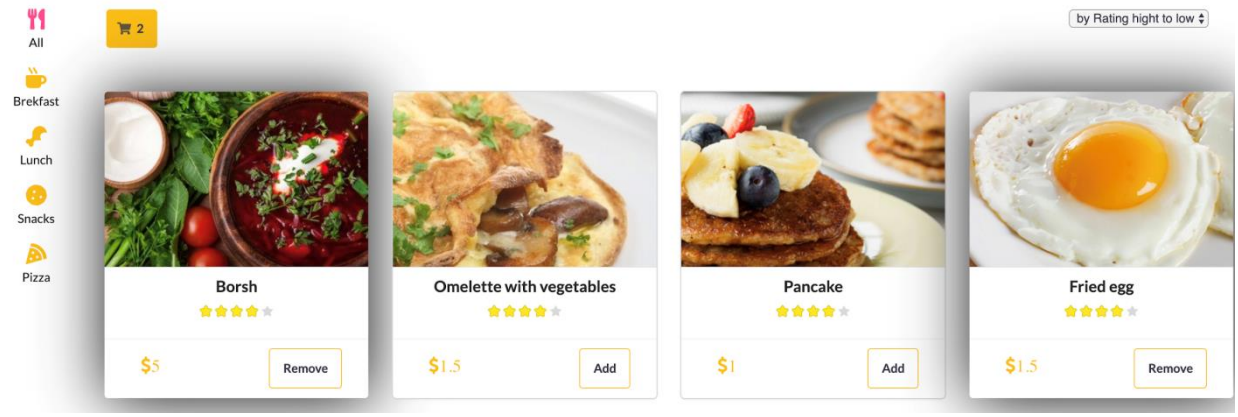


Рисунок 4.8 Сторінка страв

Повернувшись до замовлення на крок з меню, користувач бачить замовлені щойно ним страви, які він може редагувати, змінюючи їхню кількість та видалити.

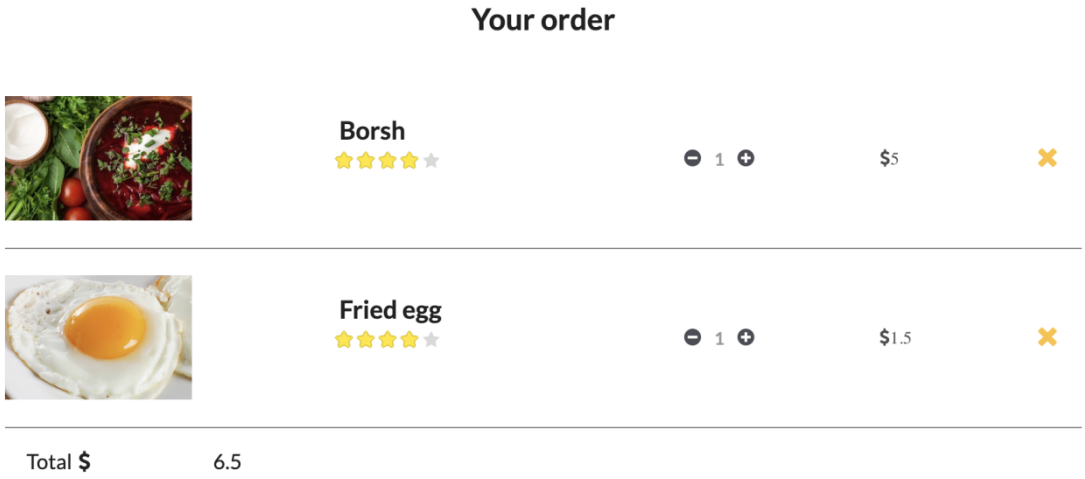


Рисунок 4.9 Крок замовлення страв

Наступний крок для клієнта - чек з усією інформацією про замовлення (час, дата, замовленні страви, ціна) та оплата.

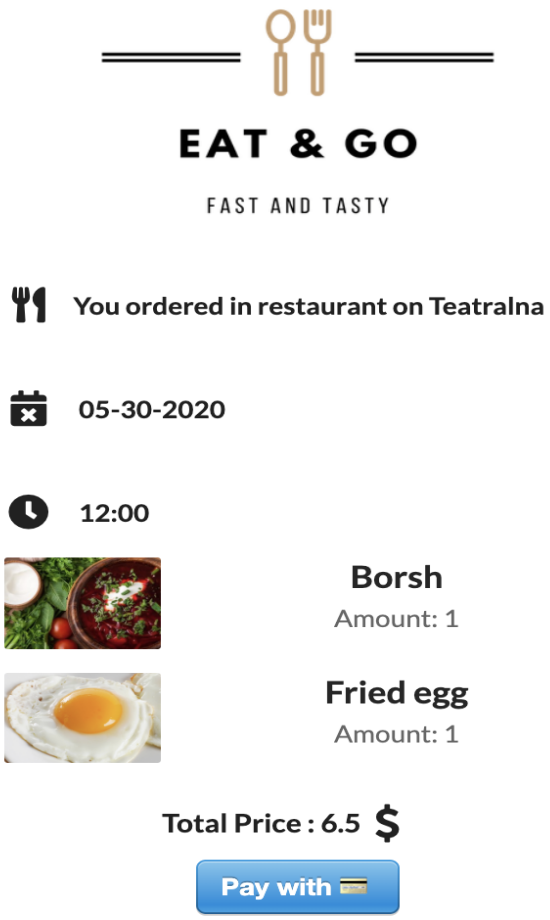


Рисунок 4.10 Чек замовлення

Щоб підтвердити замовлення, потрібно оплатити його, натиснувши “Pay with”. Для здійснення оплати з'являється нове модальне вікно, в якому заповнюється дані карти та здійснюється оплата за допомогою кнопки “Pay”.

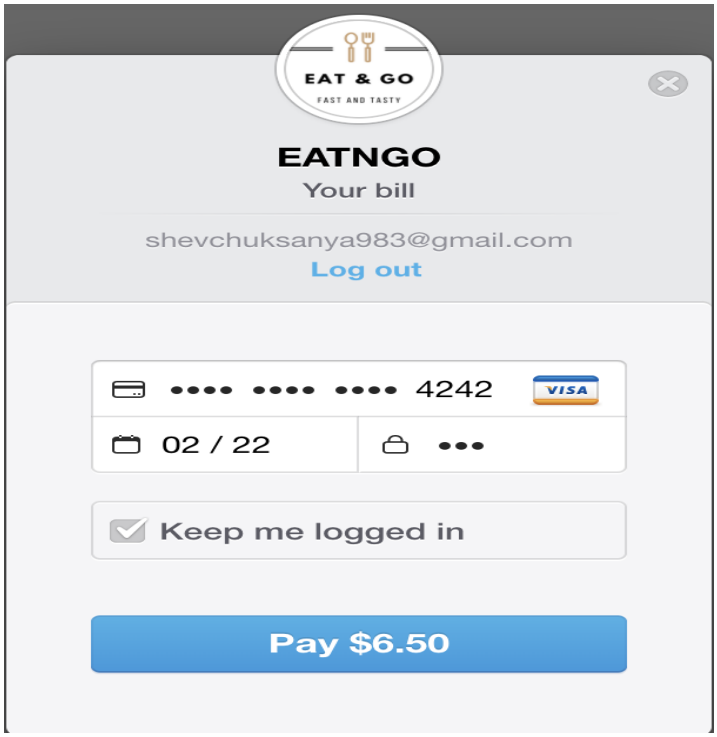


Рисунок 4.11 Форма оплати

Дане замовлення було виконано не зареєстрованим користувачем, що позбавляє його додаткових можливостей сервісу. Зареєстрований користувач має можливість створення власного кабінету з інформацією про себе та спеціальні знижки, які пропонують заклади. Для реєстрації та авторизації слід натиснути “Log-in” у шапці сайту, що перенаправить на Log-in сторінку.

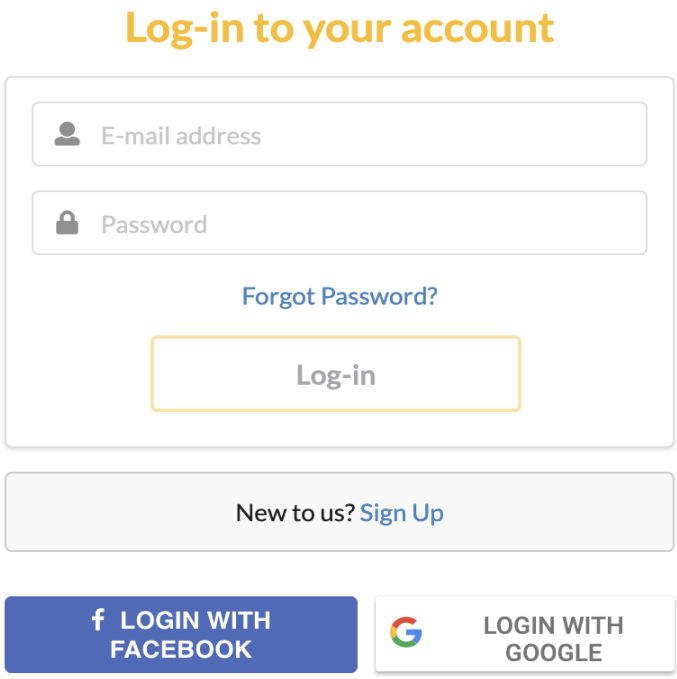


Рисунок 4.12 Log-in форма

Оскільки користувач не має зареєстрованого аккаунта, він може авторизуватись через відому соціальну мережу “Facebook”, або “Google” пошту, або створити новий аккаунт натиснувши кнопку “Sign Up”. Зареєструємо нового користувача.

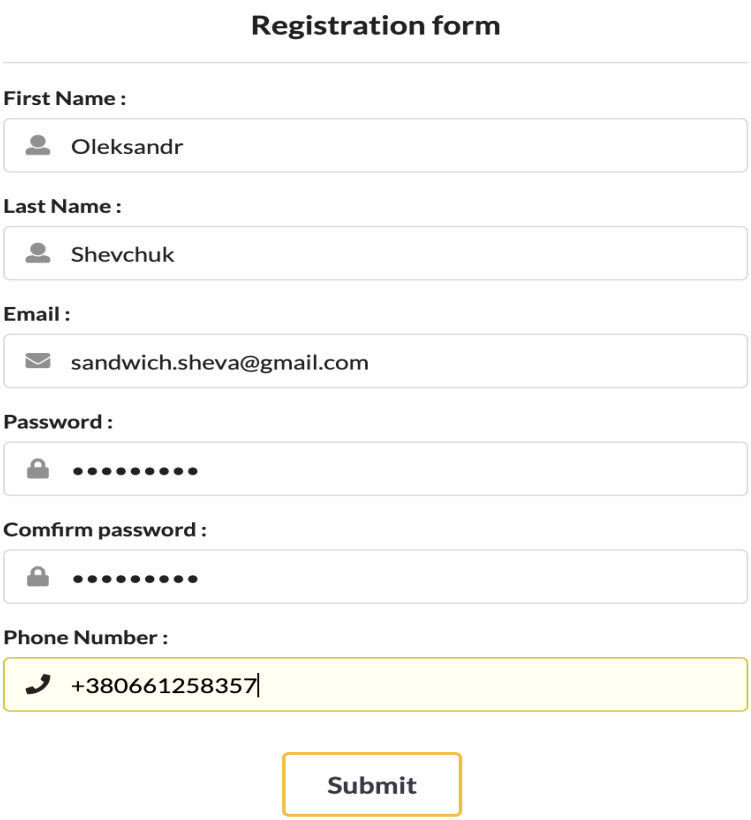


Рисунок 4.13 Sign-up форма

Після успішної реєстрації, створюється особистий кабінет користувача, який можна знайти у шапці веб-сайту.

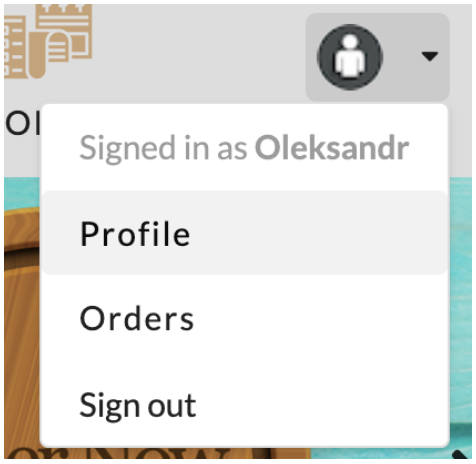


Рисунок 4.14 Юзер дропдаун

Особистий кабінет користувача виглядає наступним чином.

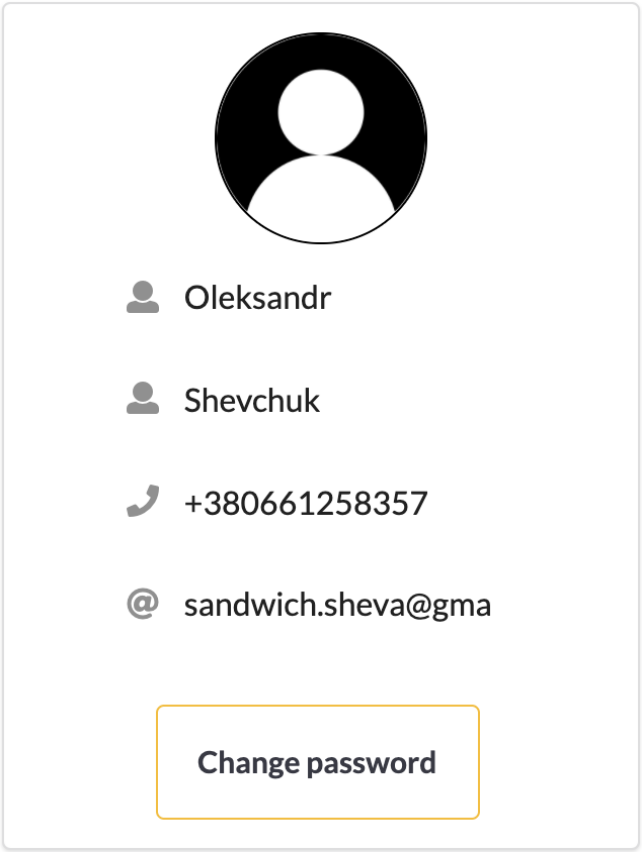


Рисунок 4.15 Особистий кабінет користувача

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

4.2 Інструкція адміністратора

Щоб відкрити веб-додаток для адміністратора, необхідно перейти на адресу у браузері, де розміщений адмін-клієнтська частина. Переходячи за адресою, користувач бачить Log-in форму.

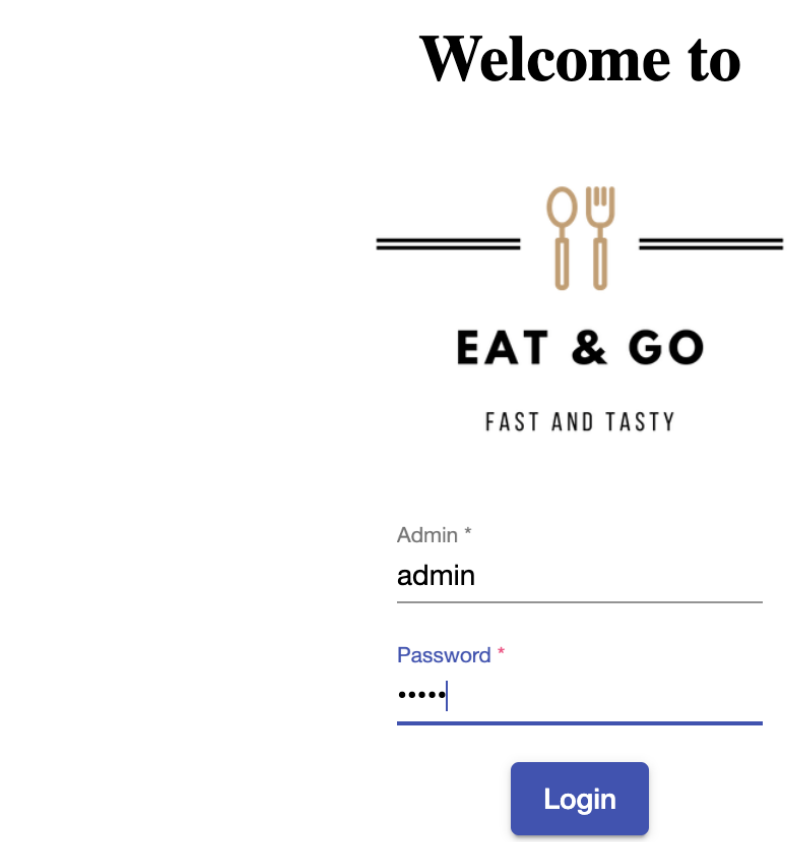


Рисунок 4.16 Admin Log-in форма

Після коректного ведення логіна та пароля адміністратора, користувача перенаправляє на головну сторінку, в якій є лише шапка веб-сайту.

Рисунок 4.16 Admin шапка веб-сайту

Розглянемо редагування меню страв. Адміністратор має можливість додавати, видаляти та редагувати страви.

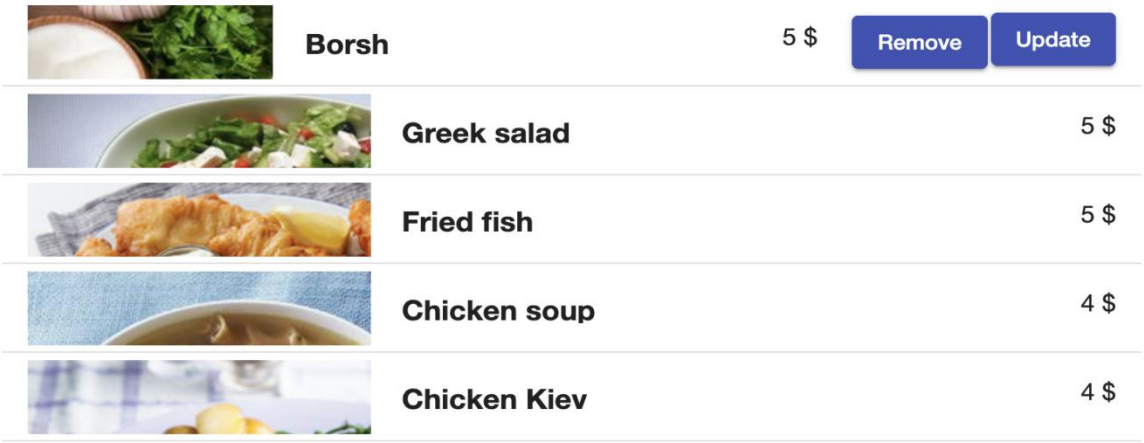


Рисунок 4.17 Список страв

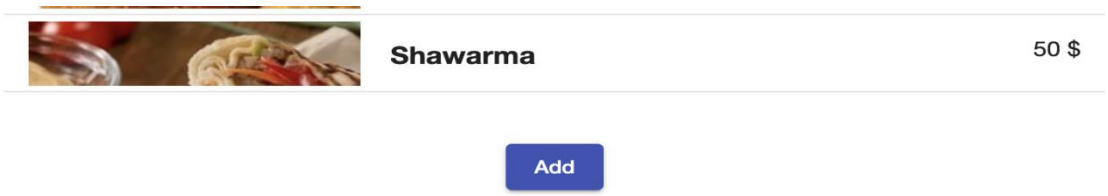


Рисунок 4.18 Кнопка для додавання страв

При додаванні або редагуванні страви, адміністратор повинен заповнити інформацію про страву.

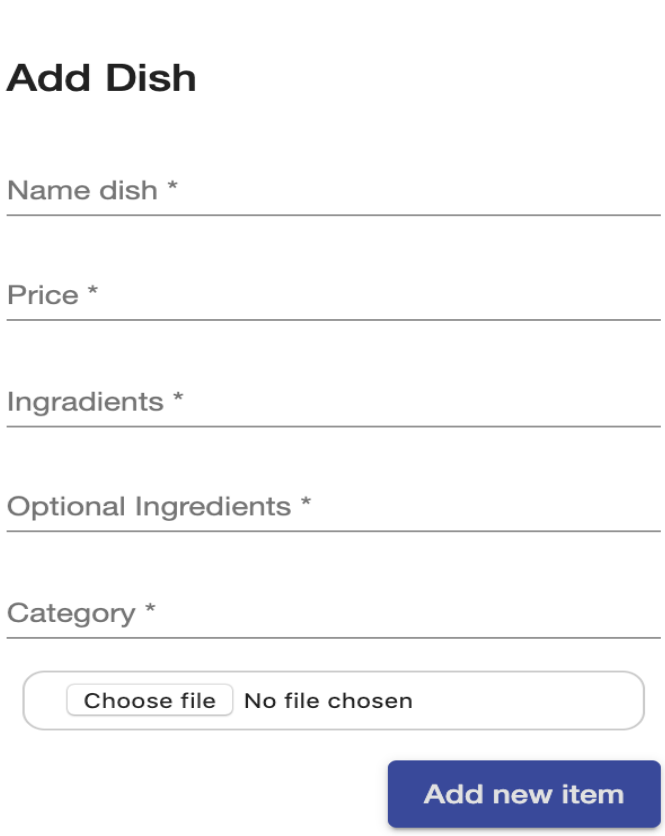


Рисунок 4.19 Форма додавання страв

Такі ж самі операції адміністратор може виконувати над спеціальними акціями ресторанів.

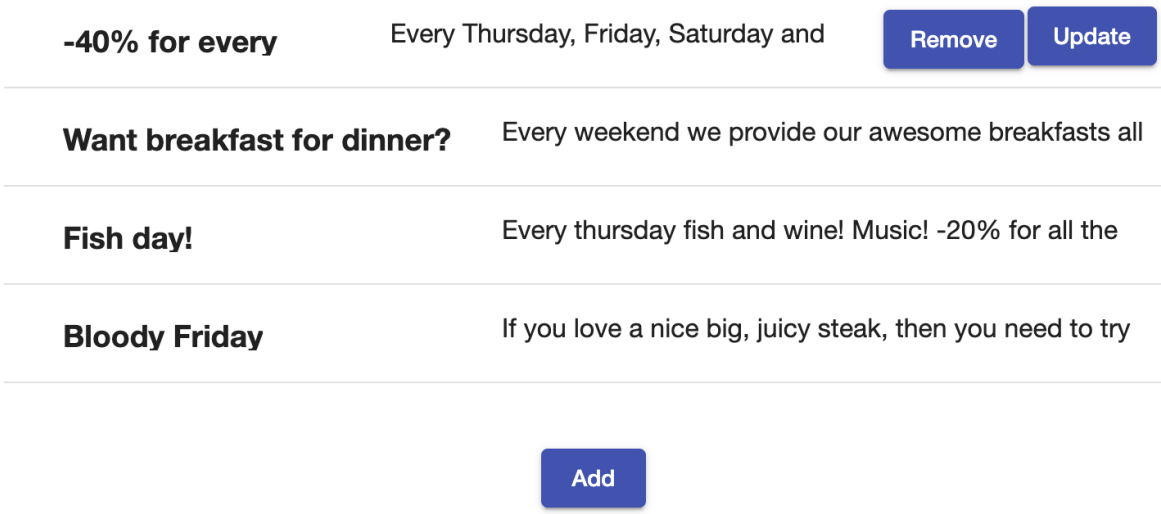


Рисунок 4.20 Список спеціальних акцій

Висновок 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. Для зареєстрованого користувача додан особистий кабінет, в якому він може заповнювати інформацію про себе, змінювати пошту, пароль, завантажувати власне фото. Також зареєстрований користувач має додаткові бонуси, спеціальні акції у ресторанах та можливість перегляду попередніх замовлень, які були здійснені в додатку. Веб-додаток містить інформацію про страви, спеціальні акції, розташування, фото та робочі години закладів громадського харчування для того, щоб надати користувачеві достатньо необхідної інформації про заклад.

Інтерфейс адміністратора розроблений з мінімальною кількістю анімації, зображень для спрощення дизайну та фокусуванням на інструментах керування звичайним клієнтом.

ЗАГАЛЬНІ ВИСНОВКИ

Метою даної дипломної роботи є розробка веб-додатку для керування системою закладів громадського харчування.

Розроблений веб-додаток являє собою сучасний веб-сайт, який містить заклади громадського харчування певного міста з можливістю створення замовлень 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 веб-додатка 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 дипломного проектування були вивченні та покращені основні теоретичні та практичні знання розробки сучасних веб-сайтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 100+ Internet statistics and facts for 2020 https://www.websitehostingrating.com/internet-statistics-facts/
2. Стаття про SPA https://wezom.com.ua/blog/chto-takoe-spa-prilozheniya
3. Стаття про REST API https://habr.com/ru/post/38730/
4. Переваги використання React.js https://webformyself.com/react-js-jsx/
5. npm vs yarn https://www.sitepoint.com/yarn-vs-npm/
6. Принцип роботи Redux https://habr.com/ru/post/439104/
7. Особливості роботи Angular https://dzudzylo.com/javascript/osoblyvosti-frejmvorku-angularjs.html
8. Переваги Node.js https://techrocks.ru/2019/01/20/why-do-you-need-node-js/
9. Node.js та Express.js https://habr.com/ru/company/piter/blog/265649/
10. Переваги використання Express.js https://www.impressico.com/2015/10/06/advantages-of-using-express-js/
11. Переваги використання MongoDB https://echo.lviv.ua/dev/9693
12. Налаштування бібліотеки axios https://www.npmjs.com/package/axios
13. Шифрування за допомогою бібліотеки cryptoJs https://cryptojs.gitbook.io/docs/

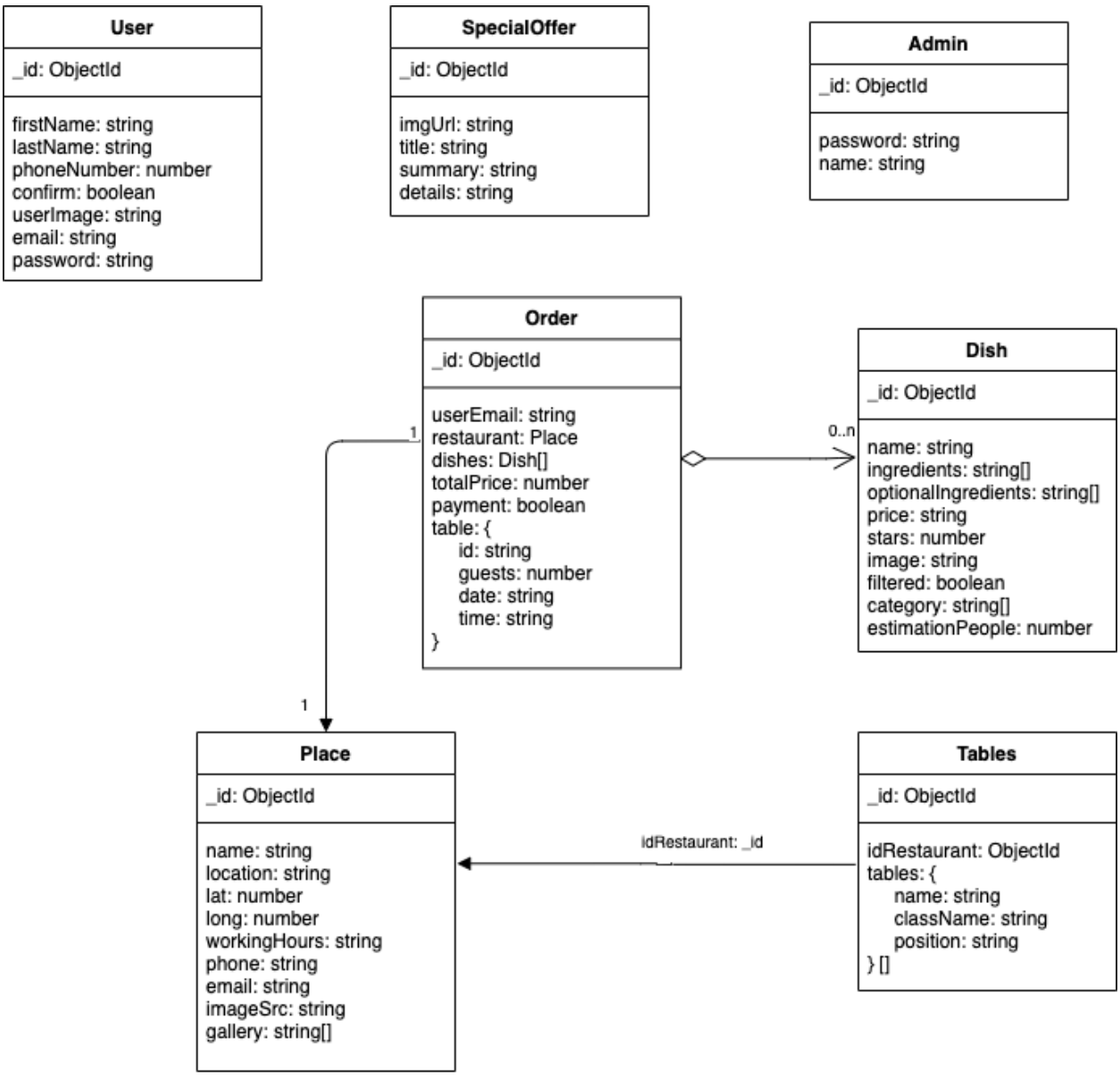
ДОДАТОК 1

Веб-додаток для керування системою закладів громадського харчування

СХЕМА МОДЕЛІ ДАНИХ

АРКУШІВ 1

Київ – 2020



					ІАЛЦ 467800.003 ПЗ		
Зм.	*Арк.*	*Прізвище*	*Підпис*	*Дата*			
Розроб.		*Шевчук О.С.*			Веб-додаток для керування системою закладів громадського харчування. *Пояснювальна записка*	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перевірив.		*Долголенко О.М.*					
Н. кон.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, ІП-62*	
Затв.		*Стіренко С.Г.*					

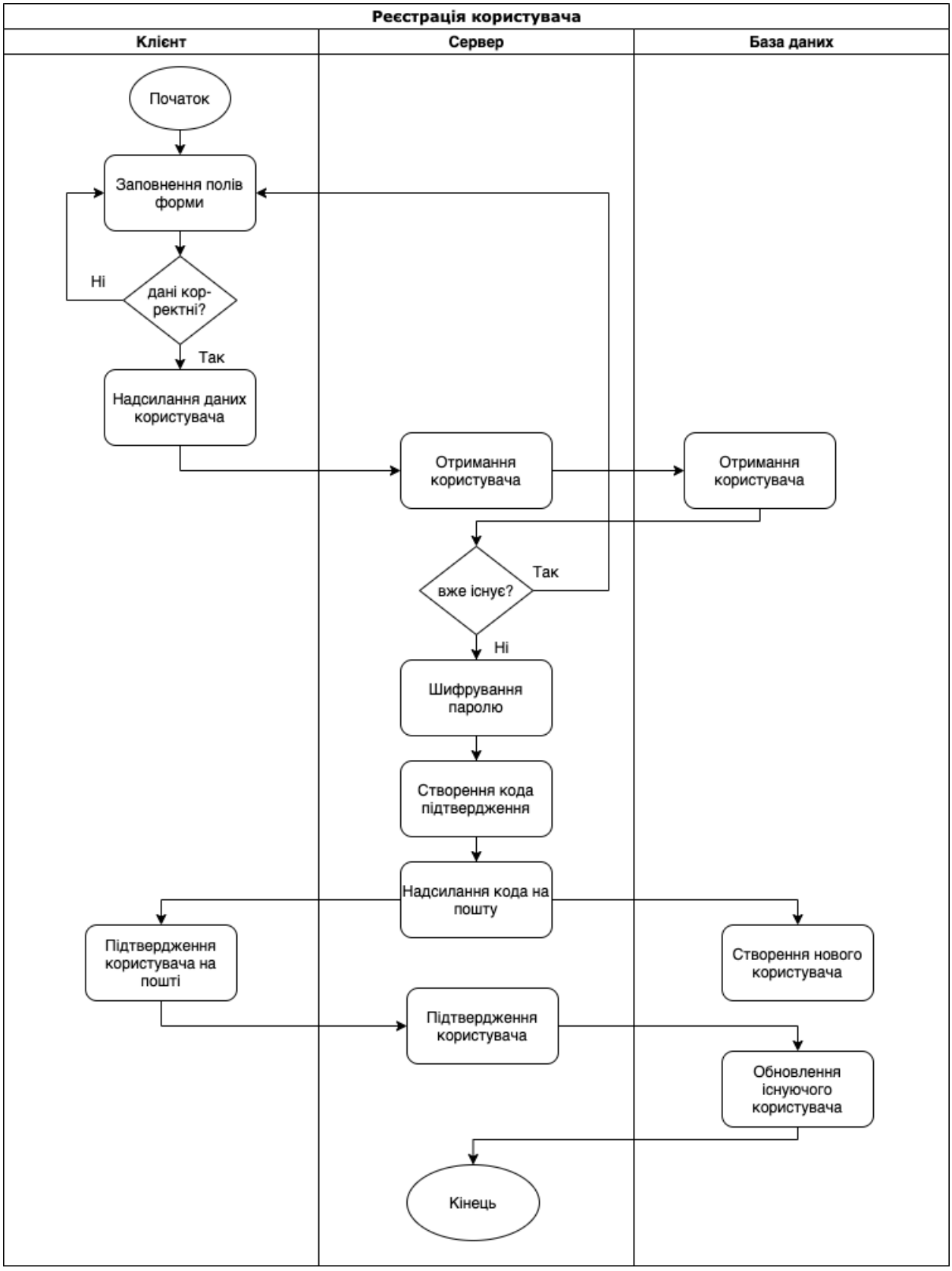
ДОДАТОК 2

Веб-додаток для керування системою закладів громадського харчування

СХЕМА АЛГОРИТМУ РЕЄСТРАЦІЇ

АРКУШІВ 1

Київ – 2020



					ІАЛЦ 467800.003 ПЗ		
Зм.	*Арк.*	*Прізвище*	*Підпис*	*Дата*			
Розроб.		*Шевчук О.С.*			Веб-додаток для керування системою закладів громадського харчування. *Пояснювальна записка*	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перевірив.		*Долголенко О.М.*					
Н. кон.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, ІП-62*	
Затв.		*Стіренко С.Г.*					

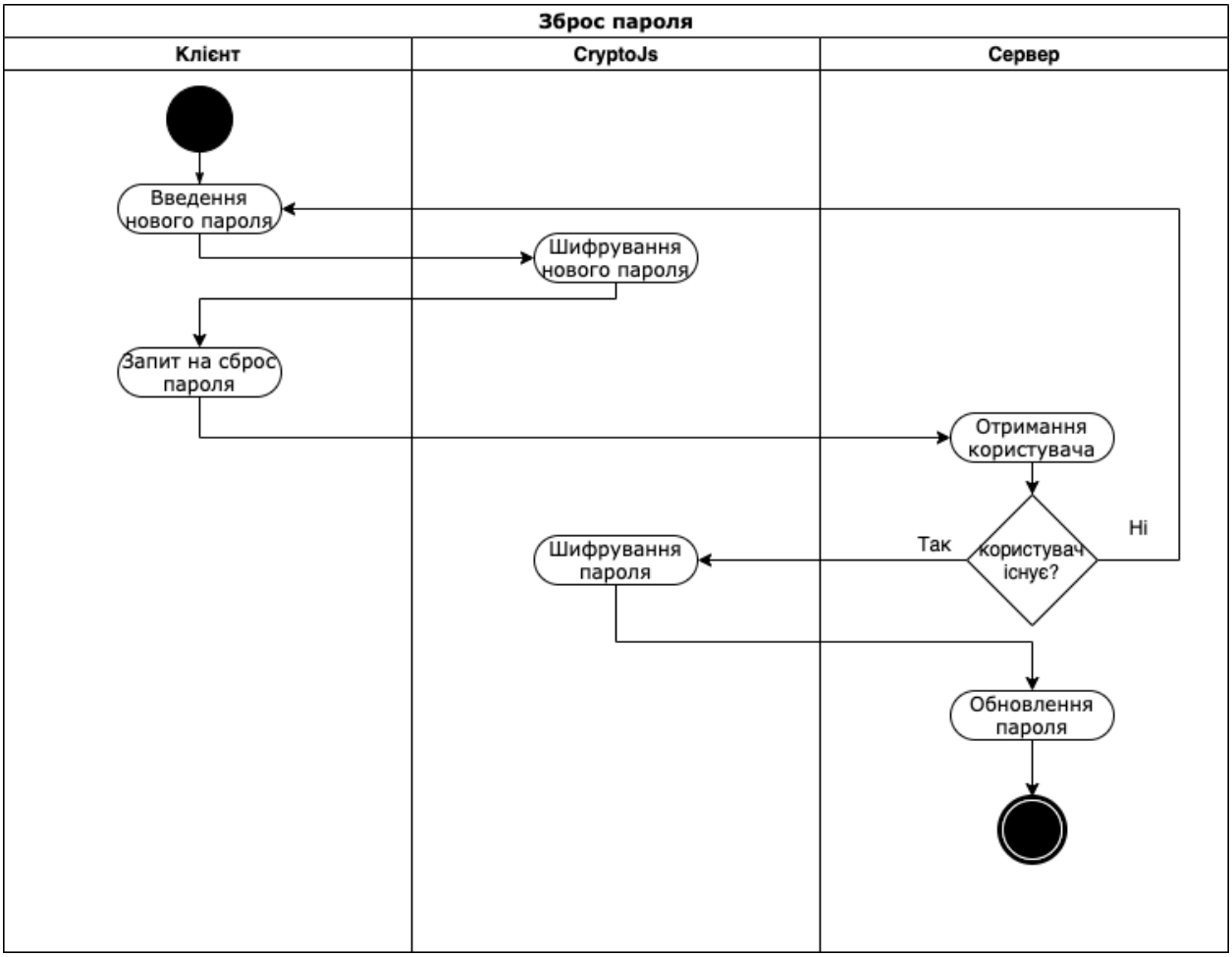
ДОДАТОК 3

Веб-додаток для керування системою закладів громадського харчування

АЛГОРИТМ ВСТАНОВЛЕННЯ НОВОГО ПАРОЛЯ

АРКУШІВ 1

Київ – 2020



					ІАЛЦ 467800.003 ПЗ		
Зм.	*Арк.*	*Прізвище*	*Підпис*	*Дата*			
Розроб.		*Шевчук О.С.*			Веб-додаток для керування системою закладів громадського харчування. *Пояснювальна записка*	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перевірив.		*Долголенко О.М.*					
Н. кон.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, ІП-62*	
Затв.		*Стіренко С.Г.*					

ДОДАТОК 4

Веб-додаток для керування системою закладів громадського харчування

Лістинг програми

АРКУШІВ 6

Київ – 2020

Client Front-end

EngMainPage.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import EngHeader from 'components/UITable/EngHeader';
import EngFooter from 'components/UITable/EngFooter';
import EngMainPageContent from 'components/EngMainPageContent';
import EngOrder from 'pages/EngOrder';
import EngRegistrationForm from 'components/EngRegistrationForm';
import EngLoginForm from 'components/EngLoginForm';
import EngResetPassword from 'components/EngResetPassword';
import EngRestaurants from 'pages/EngRestaurants';
import EngSpecialOffers from 'pages/EngSpecialOffers';
import EngMenu from 'pages/EngMenu';
import EngProfile from 'pages/EngProfile';
import EngUserOrders from 'components/EngUserOrders';

import {authRoute} from 'services/http-client/authService';
import './styles.scss';

export function EngMainPage() {
  return (
    <Router>
      <div className='main-container'>
        <EngHeader className="Header" />
        <main>
          <Switch>
            <Route exact path='/' component={EngMainPageContent} />
            <Route exact path='/registration' component={EngRegistrationForm} />
            <Route path='/login' component={EngLoginForm} />
            <Route path={authRoute.reset} component={EngResetPassword} />
            <Route path='/restaurants' component={EngRestaurants} />
            <Route path='/menu' component={EngMenu} />
            <Route path='/offers' component={EngSpecialOffers} />
            <Route path='/profile' component={EngProfile} />
            <Route exact path='/order' component={EngOrder} />
            <Route exact path='/myorders' component={EngUserOrders} />
          </Switch>
        </main>
        <EngFooter />
      </div>
    </Router>
  );
}
```

EngOrderContext.js

```
import React, {useState} from 'react';

export const EngOrderContext = React.createContext([ {}, () => {}]);

export const initialContextState = {
 restaurants: {},
 tables: {},
 dishes: [],
 payment: false,
 totalPrice: 0,
 currentStep: 'restaurants'
};

export const EngOrderProvider = props => {
```

```
 const [state, setState] = useState(initialContextState);

 return (
  <EngOrderContext.Provider value={[state, setState]}>
    {props.children}
  </EngOrderContext.Provider>);
};
```

users-registration.js

```
import { post } from './index';

export const usersRegistration = (url, data) => {
   return post(url, data);
};
```

Admin Front-end

auth.service.ts

```
import { Injectable } from '@angular/core';
import CryptoJS from 'crypto-js';
import {HttpClient} from '@angular/common/http';
import {Router} from '@angular/router';

@Injectable({
 providedIn: 'root'
})
export class AuthService {
 private url = 'https://eatngo.herokuapp.com/';
 private _isLoggedIn = false;
 private _errorMessage: string;

 get isLoggedIn() {
  return this._isLoggedIn;
 }
 get errorMessage() {
  return this._errorMessage;
 }

 changeStatus(isLoggedIn: boolean) {
  if(!isLoggedIn){
    this._errorMessage = 'Admin name or password was wrong!!!';
  } else {
    this._errorMessage = '';
    this.router.navigate(['/admin'])
  }
  this._isLoggedIn = isLoggedIn;
 }

 constructor(private http: HttpClient, private router: Router) { }

 authenticateAdmin(name, password) {
  const admin = {
    name,
    password
  };
  return this.http.post(`${this.url}admin`, admin);
```

```
 }

 login(name, password) {
  if(name && password){
    const cryptedPassword = this.crypting(password);
    return this.authenticateAdmin(name, cryptedPassword)
      .subscribe(() => {
         this.changeStatus(true);
       }, () => {
         this.changeStatus(false);
       }
      );
  }
 }

 crypting = (value) => {
  return CryptoJS.enc.Base64.stringify(CryptoJS.HmacSHA256(value, 'secret'));
 }
}
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import {EngLoginComponent} from './components/eng-login/eng-login.component';
import {EngMainPageComponent} from './components/eng-main-page/eng-main-page.component';
import { EngListComponent } from './components/eng-list/eng-list.component';

const routes: Routes = [
 { path: 'login', component: EngLoginComponent},
 { path: '',   redirectTo: '/login', pathMatch: 'full' },
 { path: 'admin',
  component: EngMainPageComponent,
  children: [
   { path: ':type', component: EngListComponent}
  ]
 }
];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule { }
```

interfaces.ts

```
import * as mongoose from 'mongoose';

export interface IAdminInterface {
  name: string,
  password: string
}

export interface IDish {
  _id?: mongoose.Schema.ObjectId,
  name: string,
  ingradients?: string[],
  price?: number,
```

```
  isDeleted?: boolean,
  stars?:number,
  image: string,
  category?: string[],
  estimationPeople?: number
}

export interface ISpecialOffer {
  _id?: mongoose.Schema.ObjectId;
  imgUrl: string,
  title?: string,
  summary?: string,
  details?: string
}
```

Back-end

placesModel.js

```
const mongoose = require('mongoose');
const {Schema} = mongoose;

const PlaceSchema = new Schema(
  {
    name: {
      type: String,
      required: 'location must be'
    },
    location: {
      type: String,
      required: 'location must be'
    },
    workingHours: {
      type: String,
      required: 'have to type working hours for example 8am-10pm'
    },
    imageSrc: {
      type: String,
      required: 'image cannot be blank'
    }
  },
  { collection: 'places' }
);

module.exports = mongoose.model('Place', PlaceSchema);
```

placesController.js

```
const mongoose = require('mongoose');
const Place = mongoose.model('Place');
const dbService = require('../services/dBService');
const queryWrapper = require('../utils/queryWrapper')

exports.listAllPlaces = async (req, res) => {
 queryWrapper(req, res, () => dbService.getAllDataFromCollection(Place));
};

exports.createPlace = async (req, res) => {
 queryWrapper(req, res, () => dbService.createElement(Place, req.body))
};

exports.readPlace = async (req, res) => {
 queryWrapper(req, res, () => dbService.getOneElementById(Place, req.params.placeId))
```

```
};

exports.updatePlace = async (req, res) => {
 queryWrapper(req, res, () => dbService.updateOneElement(Place, req.params.placeId, req.body))
};

exports.deletePlace = async (req, res) => {
 queryWrapper(req, res, () => dbService.deleteOneElement(Place, req.params.placeId))
}
```

placesRoutes.js

```
const placeBuilder = require('../controllers/placesController');

module.exports = app => {
 app
  .route('/places')
  .get(placeBuilder.listAllPlaces)
  .post(placeBuilder.createPlace);

 app
  .route('/places/:placeId')
  .get(placeBuilder.readPlace)
  .put(placeBuilder.updatePlace)
  .delete(placeBuilder.deletePlace);
};
```

dbService.js

```
const mongoose = require('mongoose');
const { uri, dbName } = require('./dBParameters');

exports.connectToDB = () => {
 return new Promise((resolve, reject) => {
  mongoose.connect(uri, { useNewUrlParser: true, useFindAndModify: false, dbName: dbName }, (err) => {
   err ? reject(err) : resolve();
  });
 })
};
exports.getAllDataFromCollection = (model) => {
 return new Promise((resolve, reject) => {
  model.find((err, dataArray) => {
   err ? reject(err) : resolve(dataArray);
  });
 });
};
exports.getAllDataFromCollectionItem = (model, field) => {
 return new Promise((resolve, reject) => {
  model.find({ '_id': { $in: field } }, (err, arrId) => {
   err ? reject(err) : resolve(arrId);
  });
 });
};
exports.createElement = (model, params) => {
 return new Promise((resolve, reject) => {
  const newElement = new model(params);
  newElement.save((err, element) => {
   (err) ? reject(err) : resolve(element);
  });
 })
};
exports.getOneElementById = (model, elementId) => {
 return new Promise((resolve, reject) => {
  model.findById(elementId, (err, element) => {
```

```
    err ? reject(err) : resolve(element);
  });
 });
};
exports.getOneElementByField = (model, field) => {
 return new Promise((resolve, reject) => {
  model.findOne(field, (err, element) => {
    err ? reject(err) : resolve(element);
  });
 });
};
exports.getAllElementByField = (model, field) => {
 return new Promise((resolve, reject) => {
  model.find(field, (err, element) => {
    err ? reject(err) : resolve(element);
  });
 });
};

exports.findByIdAndUpdate = (model, field, params) => {
 return new Promise((resolve, reject) => {
  model.findByIdAndUpdate(
    field,
    params,
    { new: true },
    (err, element) => {
      err ? reject(err) : resolve(element);
    });
 });
}
exports.updateElement = (model, field, params) => {
 return new Promise((resolve, reject) => {
  model.findOneAndUpdate(
    field,
    params,
    { new: true },
    (err, element) => {
      err ? reject(err) : resolve(element);
    });
 });
};
exports.deleteOneElement = (model, _id) => {
 return new Promise((resolve, reject) => {
  model.deleteOne(
    { '_id': _id },
    err => {
      err ? reject(err) : resolve('User successfully deleted');
    });
 });
```