КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Веб-застосунок підтримки діяльності закладу громадського харчування»

Виконав _____ _____
(Підпис)
Васильєв Олександр Олександрович
(прізвище, ім'я, по батькові)

Унікальність тексту 93,37%

Автор __ _____ Васильєв О.О.
(Підпис) (Прізвище, ініціали)

Керівник _Бойко Юлія Петрівна
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри ___ _ Плескач В.Л.
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2022

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2022	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2022	Виконано
9.	Подання роботи у першому варіанті	28.04.2022	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	23.05.2022	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	Виконано
14.	Захист кваліфікаційної роботи бакалавра	22.06.2022	Виконано

Здобувач вищої освіти _____ _____

(підпис)

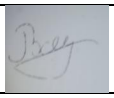
Керівник 

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

Назва теми: «Веб-застосунок підтримки діяльності закладу громадського харчування»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Васильєв Олександр Олександрович	

Назва роботи українською та англійською мовами:

Веб-застосунок підтримки діяльності закладу громадського харчування
Web application to support the activities of catering establishments

Мета бакалаврської роботи: підвищення ефективності роботи закладу громадського харчування шляхом розробки веб-застосунку
План роботи: 1. Огляд сучасного стану проблеми та постановка завдання 2. Аналіз програмних засобів та підходів до розробки 3. Програмна реалізація та тестування веб-застосунку

ПІБ, ступінь, звання наукового керівника роботи: к.т.н., доцент Бойко Ю.П.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

JSON - JavaScript Object Notation

DB – Database (база даних)

JS - JavaScript

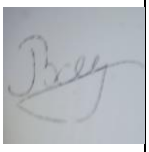
MEAN – MongoDB, Express JS, Angular, Node JS

SQL - Structured query language

JWT – JSON Web token

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	2
Анотація (іноземною мовою-англійською)	2
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
Розділ 1	6
Розділ 2	16
Розділ 3	12
Висновки	1
Перелік використаних джерел	2
Додатки	4

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розроб н.</td><td>Васильєв О.О.</td><td>  </td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Бойко Ю.П.</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Базилюк А.М.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Кваліфікаційна робота: 53 с., 23 рис., 11 табл., 24 джерел, 2 дод.

Ця робота присвячена розробці веб-застосунку для забезпечення діяльності закладу громадського харчування.

Метою роботи є підвищення ефективності роботи закладу громадського харчування шляхом розробки веб-застосунку

Завдання роботи:

- Дослідити теоретичні аспекти запровадження електронних систем у закладах громадського харчування;
- Дослідити архітектуру побудови веб-застосунку підтримки діяльності закладу громадського харчування;
- Розробити та протестувати веб-застосунок підтримки діяльності закладу громадського харчування.

Об'єктом дослідження є процеси автоматизації діяльності закладу громадського харчування.

Предметом дослідження є засоби та принципи розробки веб-застосунків для забезпечення ефективності роботи закладу громадського харчування.

*Методи дослідження***:**

Методи теоретичного рівня: абстрагування, ідеалізація, формалізація, аналіз і синтез, аксіоматика, узагальнення – використовуються для проведення логічного дослідження наявної інформації щодо веб-розробки, вироблення понять, думок, виведення висновків. Методи емпіричного рівня: спостереження, порівняння, розрахунок, тести – використовуються для формулювання результатів розробки; метод проб і помилок – у процесі розробки.

Практичне значення одержаних результатів полягає у можливості використовувати результати як методичний матеріал або як приклад для майбутніх розробок проектів схожого призначення.

Структура роботи: бакалаврська робота складається зі вступу, трьох розділів, розподілених на підрозділи, висновку, додатків та списку використаних джерел.

Ключові слова: веб-застосунок, клієнт-сервер, MEAN-стек, громадське харчування, тестування.

ANNOTATION

Qualification work: 53 pages., 23 images., 11 tables., 24 sources, 2 additions.

This work is devoted to the development of a web application to support the activities of the catering establishment.

The aim of the work is to increase the efficiency of the catering institution by developing a web application.

Tasks of work:

● Investigate the theoretical aspects of the introduction of electronic systems in catering establishments;

● Investigate the architecture of building a web application to support the activities of catering establishments;

● Develop and test a web application to support the activities of the catering establishment;

The object of research is the processes of automation of public catering establishments.

The subject of the study is the means and principles of developing web applications to ensure the efficiency of the catering establishment.

*Research methods***:**

Methods of theoretical level: abstraction, idealization, formalization, analysis and synthesis, axiomatics, generalization - are used to conduct a logical study of available information on web development, development of concepts, opinions, conclusions. Empirical level methods: observation, comparison, calculation, tests - are used to formulate the results of development; trial and error method - in the process of development.

The practical significance of the results obtained is the ability to use the results as a methodological material or as an example for future development of projects for similar purposes.

Structure of the work: the bachelor's thesis consists of an introduction, three sections divided into sections, conclusion, appendices and a list of sources used.

Keywords: web application, client-server, MEAN-stack, catering, testing.

ЗМІСТ

ВСТУП

Актуальність дослідження. На сьогодні сфера громадського харчування є важливою частиною повсякденного життя. Однак, через високий рівень конкуренції власникам закладів необхідно працювати над всіма доступними шляхами підвищення ефективності роботи, які забезпечать комфортні робочі умови для персоналу та покращать враження клієнтів від швидкості та якості обслуговування. Створення застосунку є важливою складовою для виконання даного завдання.

Метою роботи підвищення ефективності роботи закладу громадського харчування шляхом розробки веб-застосунку

Завдання роботи:

- Дослідити теоретичні аспекти запровадження електронних систем у закладах громадського харчування;
- Дослідити архітектуру побудови веб-застосунку підтримки діяльності закладу громадського харчування;
- Розробити та протестувати веб-застосунок підтримки діяльності закладу громадського харчування.

Об’єктом дослідження є процеси автоматизації діяльності закладу громадського харчування.

Предметом дослідження є засоби та принципи розробки веб-застосунків для забезпечення ефективності роботи закладу громадського харчування.

Методи дослідження:

Методи теоретичного рівня: абстрагування, ідеалізація, формалізація, аналіз і синтез, аксіоматика, узагальнення – використовуються для проведення логічного дослідження наявної інформації щодо веб-розробки, вироблення понять, думок, виведення висновків. Методи емпіричного рівня: спостереження, порівняння, розрахунок, тести – використовуються для формулювання результатів розробки; метод проб і помилок – у процесі розробки.

Практичне значення одержаних результатів полягає у можливості використовувати результати як методичний матеріал або як приклад для майбутніх розробок проектів схожого призначення.

Структура роботи: бакалаврська робота складається зі вступу, трьох розділів, розподілених на підрозділи, висновку, додатків та списку використаних джерел.

РОЗДІЛ 1
ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Поняття закладу громадського харчування

Заклад громадського харчування - підприємство, призначене для виробництва кулінарної продукції, борошняних, кондитерських і булочних виробів, їх реалізації та/або організації їх споживання[1].

Хоч заклади громадського харчування і виконують одну й ту саму функцію, вони поділяються на різні типи відповідно до специфіки організації процесів, особливостей обслуговування, асортименту, місця розташування.

Можна виділити наступні категорії:

- ресторан - заклад, який відрізняється широким та різноманітним асортиментом, якісним приготуванням страв, наявністю фірмових страв, високим рівнем обслуговування та підготовки персоналу. Може надавати розважальні послуги, приймати індивідуальні замовлення.

- кафе - підприємство харчування, яке пропонує гостям обмежений асортимент страв і напоїв у поєднанні з відпочинком і розвагами.

- бар - заклад громадського харчування, що спеціалізується на продажі алкогольних напоїв, які споживаються на місці. Від кафе або ресторану відрізняється асортиментом та відсутністю обов’язкової необхідності для приготування страв.

- буфет - заклад ресторанного господарства з обмеженим асортиментом страв нескладного приготування, булочних і кондитерських виробів, закупних товарів. Буфети призначені передусім для швидкого обслуговування споживачів, буфети отримують продукцію від ресторану або сторонніх виробників кулінарних виробів.

- закусочна;

- їдальня;

- кафетерій - заклад ресторанного господарства, який відрізняється невеликими розмірами і обмеженим асортиментом страв. Відмінною особливістю є робота по системі самообслуговування.

- фабрика-заготівельня;

- фабрика-кухня[2]

1.2 Запровадження електронних систем у закладах громадського харчування

На сьогоднішній день у багатьох закладах громадського харчування використовуються електронні системи для їх керування. Це пояснюється широким вибором вже готових розробок, які є універсальними (можуть використовуватися у різних видах закладів), простими у користуванні (мають усе необхідне, щоб не витрачати багато часу для навчання працівника), відносно недорогими (у порівнянні до потенційної вартості подібних систем при індивідуальній розробці), та містять функціонал, що може значно спростити роботу закладу на будь-якому його етапі. З метою аналізу вимог до майбутньої розробки у рамках даної роботи було проаналізовано функціонал та особливості деяких систем для підтримки діяльності закладів громадського харчування, які є реально використовуваними на ринку: системи Poster, Fusion POS та Quick Resto.

Інтерфейс системи Poster наведено на рисунку 1.1.

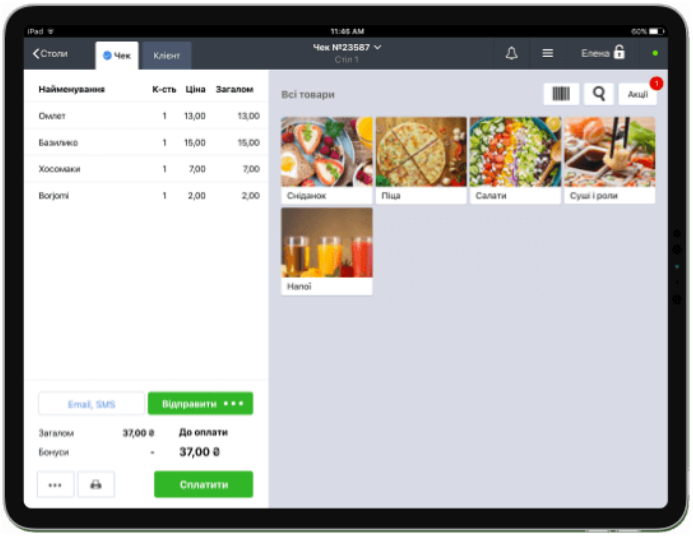
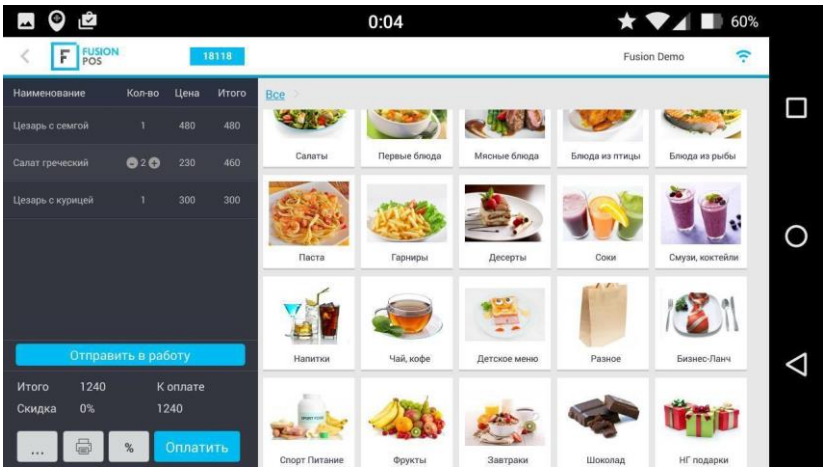


Рисунок 1.1 – Інтерфейс Poster[3]

Інтерфейс системи Fusion наведено на рисунку 1.2.



1.2 – Інтерфейс системи Fusion

Інтерфейс системи Quick Resto наведено на рисунку 1.3.

Рисунок 1.3 – Інтерфейс системи Quick Resto

На основі отриманої інформації у таблиці 1.1 було сформовано перелік деяких загальних аспектів роботи, які успішно були перенесені у електронний вигляд як функції у проаналізованих системах автоматизації.

Таблиця 1.1 – Основні функції систем автоматизації ресторанів

Функція	Опис
Електронне меню	У персоналу завжди під рукою є актуальний перелік усіх доступних позицій

Продовження таблиці 1.1

Функція	Опис
Формування рахунку	Автоматична генерація та друк рахунку по замовленим клієнтом позиціях
Електронне табло на кухні	Кухарям не потрібно запам’ятовувати або записувати перелік страв, які очікують приготування
Прийом замовлень офіціантом	Офіціанту не потрібно запам’ятовувати або записувати замовлення клієнта, для прийому та відправки запиту на кухню достатньо натиснути кнопку
Ведення обліку документів	Застосунки для керування рестораном дозволяють вести автоматичний облік для виконання вимог законодавства, зокрема – інтеграція POS-терміналу
Керування закупівлею та зберіганням продуктів	Облік даних у електронному вигляді про стан продуктового забезпечення закладу

Слід враховувати, що вище наведений перелік не є вичерпним, оскільки завжди є шляхи щодо вдосконалення тієї чи іншої частини роботи закладу, а також все може залежати від специфіки його роботи.

1.3 Технічні особливості систем

Для формування вимог щодо розробки власного застосунку для підтримки діяльності закладу громадського харчування є необхідним визначити основні технічні особливості та очікування в умовах специфіки роботи. Більшість

розглянутих систем є кросплатформеними. Це означає, що ними можна користуватися як на телефоні, так і на комп'ютері – з підтримкою різних операційних систем. У таблиці 1.2 наведено підтримку платформу розглянутих системах управління ресторанами.

Таблиця 1.2 – Стан кросплатформеності у проаналізованих системах

Назва	Windows	Android	IOS	Mac	Web
Poster	+	+	+	+	+
Fusion POS	+	+	+		+
Quick resto					+

Однак, найбільш універсальним рішенням з урахуванням даної характеристики є використання Web-застосунку, оскільки доступ до нього є з будь-якого пристрою, який підтримує браузер, а також немає необхідності завантажувати та встановлювати окремий застосунок на пристрій. Очевидний недоліком є недоступність до програми при відсутності підключення до мережі Інтернет, однак, при таких умовах повноцінне використання застосунків цієї сфери застосування у будь-якому випадку є фактично неможливим.

Наступною важливою технічною характеристикою застосунків, призначених для управління закладами громадського харчування, є можливість оновлення даних у режимі реального часу. Така необхідність пояснюється тим, що робітник не має можливості кожної секунди проводити оновлення вручну, перевіряючи, чи не з'явилося нове замовлення.

1.4 Постановка завдання

Відповідно до розглянутих у попередніх підрозділах особливостей електронних систем управління закладами громадського харчування, бюджету, часу та кваліфікації розробників, встановлено очікування від розробленого під час виконання даної роботи застосунку. В якості закладу, який вимагає запровадження застосунку, виступатиме кафе невеликих розмірів. Цей вибір перш за все впливає на встановлення початкового набору функцій, які повинні

бути реалізованими, а також їх кількість. У даному випадку це значить, що функціонал буде значно меншим, ніж в уже існуючих рішеннях, оскільки не вимагається забезпечення діяльності, не властивій обраному типу закладів. Це дасть можливість зосередитися на найбільш необхідних деталях. Однак потрібно пам’ятати, що можливість подальшого розширення у майбутньому є обов’язковою частиною кожної розробки.

Встановлений набір вимог до функціоналу:

- Авторизація користувачів з обмеженою реєстрацією для забезпечення захисту від доступу до даних осіб, які не є працівниками закладу.
- Можливість графічного представлення стану залу для розуміння, на якій стадії обслуговування перебувають клієнти та їх замовлення.
- Можливість додавання позицій у замовлення шляхом вибору їх у онлайн-меню.
- Можливість прив’язувати замовлення до окремих відвідувачів (роздільний чек).
- Комунікація між офіціантами та кухарями
- Формування рахунків
- Оновлення даних у реальному часі
- Зручний для користування інтерфейс
- Підтримка великих (монітор) та середніх розмірів екранів (планшетів)
- Швидкість роботи

1.5 Висновок до розділу

Після проведення аналізу стану впровадження додатків у роботу закладів громадського харчування можна зробити висновки, що воно є досить розвинутим та затребуваним. Тим не менш, шляхи до досягнення заявленої мети не обмежуються використанням вже розроблених технологій і завжди залишається можливість створення окремих програмних рішень, оскільки власна розробка дає більше можливостей для забезпечення специфічних потреб власного закладу, які не передбачені представленими на ринку рішеннями або не відповідають окремим запитам.

РОЗДІЛ 2. АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ ТА ПІДХОДІВ ДО РОЗРОБКИ

2.1 Поняття та архітектура веб-застосунку

Перед вибором підходу до практичної реалізації необхідно розуміти, що собою являє “веб-застосунок”, та чим він відрізняється від веб-сайту. Хоч ці поняття є досить схожими, однак веб-сайти являють собою лише набір доступних до перегляду веб-сторінок, які, хоч і можуть бути динамічними і виконувати певні функції, але не мають у собі складної логіки. Веб-застосунок у свою чергу – це повноцінна програма, призначена для тривалої роботи, яка виконується у середовищі браузері та може бути частиною веб-сайту або окремим додатком, Тобто, різниця полягає у першу чергу в призначенні – веб-сайт виконує інформаційну функцію, а веб-застосунок – функцію взаємодії з користувачем.

Веб-застосунок є розподіленою системою, тобто складається з клієнтської та серверної частини. Серверною частиною є веб-сервер, клієнтською – браузер[4]. Таке розподілення дає змогу в майбутньому легко перенести клієнт на іншу платформу, такі як Windows або Android, не потребуючи виконання жодної роботи для переналаштування сервера, тим самим перетворивши застосунок на кросплатформений.

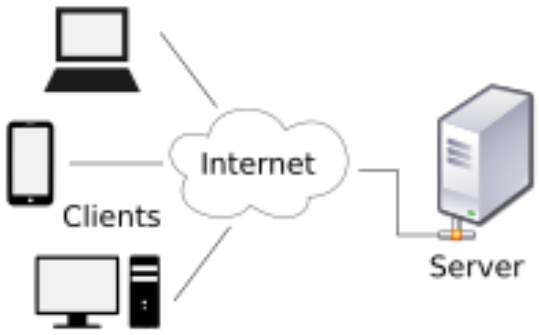


Рисунок 2.1 – Клієнт-серверна архітектура

Функції, що виконуються сервером

- обробка запитів від клієнту

- відправлення відповідей клієнту після обробки запитів.
- зберігання даних, резервне копіювання, управління доступом, їх захист.

Функції, які виконує клієнт:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправка;
- отримання результатів запиту, обробка, виведення на користувацький інтерфейс

Клієнт-серверна архітектура може бути дворівневою та трьохрівневою.

Дворівнева архітектура складається з таких частин:

- сервер, який отримує та відправляє запити без використання сторонніх ресурсів
- клієнт, який формує та відправляє запити.

Трирівнева архітектура складається з:

- Рівня представлення даних – призначений для користувача інтерфейс;
- Рівня логіки – сервер самого додатку;
- Рівня даних – сервер бази даних.

Трирівнева архітектура відрізняється розділенням серверної частини на рівень логіки та рівень даних. В результаті запит від клієнта оброблюють декілька серверів, що дозволяє знизити навантаження. Саме такий вид архітектури буде застосовуватися у даній роботі. Схема трирівневої архітектури зображена на рисунку 2.2.

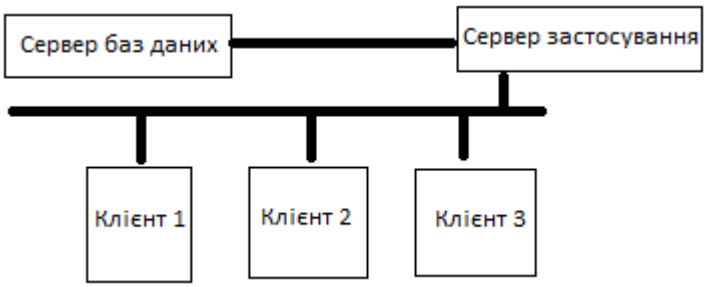


Рисунок 2.2 – Трирівнева архітектура

2.2 Підходи до розробки клієнтської частини

Для веб-застосунку клієнтом є браузер. Існує велика кількість підходів та інструментів для реалізації застосунків у цьому середовищі, тому важливо визначитися з тим, що саме буде використано. Але перед усім потрібно розуміти, що під час розробки неможливо обійтися без низки основних елементів, що використовуються у цьому середовищі – HTML, CSS та JavaScript.

HTML — це мова розмітки (мова тегів), засобами якої здійснюється розмітка майже усіх веб-сторінок та веб-застосунків. Існують й інші, більш специфічні варіанти мов розмітки, але HTML є найпоширенішим.

CSS - мова стилю, що використовується для опису зовнішнього вигляду сторінок[5]. З функціональної точки зору використання CSS є необов'язковим, однак стандартної поведінки HTML-елементів без стилізації недостатньо для досягнення достатніх показників зручності та привабливості інтерфейсу, а також втрачається можливість кастомізації.

JavaScript – скриптова мова програмування, яка відповідає за надання динамічності веб-сторінкам[6]. Якщо для звичайного веб-сайту, призначеного лише для отримання інформації, JavaScript може бути потрібен лише для створення складних візуальних елементів і можна обійтися лише засобами HTML+CSS, то для веб-застосунку, побудованого на клієнт-серверній архітектурі, використання цієї мови потрібно для забезпечення функціональності, складної логіки та формування запитів до серверу.

Конфігурація з чистих HTML+CSS+JS є повністю достатньою для створення клієнтської частини веб-застосунку, однак, враховуючи наявність на сьогоднішній день великої кількості надбудов, які дозволять максимально спростити та ефективізувати процес розробки, слід виділити деякі причини, чому варто розглянути використання деяких із них. Розробка з використанням лише чистих HTML+CSS+JS призводить до складності побудови архітектури та розділення коду, необхідності з нуля створювати базові програмні рішення, дублювання коду.

Саме тому для розробки клієнтської частини є популярним використання веб-фреймворків. Фреймворк - це готовий до використання комплекс програмних рішень, включаючи дизайн, логіку та базову функціональність системи або підсистеми. Може містити в собі також допоміжні програми, деякі бібліотеки, скрипти та загалом все, що полегшує створення та поєднання різних компонентів великого програмного забезпечення чи швидке створення готового і не обов'язково об'ємного програмного продукту.

Найбільш популярними фронтенд-фреймворками/бібліотеками на даний момент є React, Angular, Vue[7][8][9]. Для того, щоб обрати один із них, потрібно розглянути кожен із них відповідно до переваг та недоліків. Через відсутність досвіду роботи з Vue, даний варіант не буде розглядатися з метою уникнення витрати часу на вивчення.

React – розроблена компанією Facebook бібліотека для розробки користувацьких інтерфейсів. Незважаючи на те, що іноді вона позиціонується як фреймворк, насправді це лише бібліотека, оскільки відповідає лише рівню представлення, і сама не є достатньо комплексним рішенням для відповідності поняттю ‘фреймворк’. Однак на практиці вона доповнюється низкою інших бібліотек, що дозволяє ліквідувати даний недолік. Використовується у веб-застосунках компаній:

- Facebook
- Netflix
- Yahoo
- Airbnb
- Sony
- Atlassian

Angular — написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється у компанії Google, а також спільнотою приватних розробників та корпорацій. Використовується у веб-застосунках компаній:

- Google
- PayPay

- Upwork
- Expedia
- Lego
- Adidas

Переваги Angular:

- Використовується завжди у зв’язці з TypeScript (надбудовою над мовою JavaScript, яка дозволяє позбавитися проблеми відсутності статичної типізації та підвищити надійність коду[10]).
- Наявність Angular CLI – інструменту для автоматичної генерації необхідних компонентів застосунку
- Підтримка як односторонньої, так двосторонньої прив’язки даних
- Наявність системи впровадження залежностей (Dependency Injection)
- Структура та архітектура, спеціально створені для великої масштабованості проекту
- Велика кількість інструментів, які доступні одразу без необхідності встановлення сторонніх бібліотек
- Вбудована бібліотека RXJS, що дозволяє реалізовувати реактивний стиль програмування

Недоліки Angular:

- Підвищена порівняно з React складність для вивчення через наявність великої кількості структур (Injectables, Components, Pipes, Modules).
- Дещо нижча продуктивність та швидкість роботи

Переваги React:

- Легкість вивчення
- Більш ефективні механізми оптимізаціїї рендерингу, що робить її більш швидкою, ніж Angular
- Потужна підтримка рендерингу на стороні сервера та PWA (Progressive Web App).

- Підтримка TypeScript (використання якого не є обов’язковим)
- Наявність лише односторонньої прив’язки даних (дана особливість може сприйматися як перевага і недолік одночасно – дозволяє позбутися небажаних побічних ефектів, однак вимагає більше роботи, ніж при розробці пов’язаних з цією механікою місць застосунку у Angular)
- Використання парадигми функціонального програмування, що дозволяє створювати більш простий у тестуванні код

Недоліки React:

- Відсутність жодних чітких правил по реалізації архітектури
- Відсутність єдиного підходу до написання стилізації
- Поступовий відхід від класових компонентів (що є недоліком для розробників, які надають перевагу об’єктно орієнтованому стилю програмування)
- Відсутність комплексних рішень у базовій конфігурації (у даному підрозділі наведено аргументацію того, що React не є фреймворком)

З огляду усіх недоліків та переваг стає зрозуміло, що обидва варіанти є повністю прийнятними для розробки веб-застосунку закладу громадського харчування, тому вибір буде базуватися виключно на особистих перевагах та більшому особистому досвіді використання до одного із них. Саме з цих міркувань було обрано Angular.

2.3 Управління станами у клієнтській частині

За своєю структурою фреймворк Angular є набором компонентів, які відповідають за певні частини застосунку. При таких умовах постає проблема необхідності забезпечення усіх цих компонентів даними із централізованого джерела. Наприклад, перелік замовлень клієнта може бути потрібним як і офіціанту, так і на кухні, так і на видачі – тобто, в абсолютно різних частинах застосунку. При зміні списку у всіх пов’язаних із ними частинах застосунку має автоматично відбуватися оновлення та новий рендеринг відповідно до нових даних. Стандартним рішенням є використання ‘stateful services’ – сервісів (класів), які зберігають у собі дані, та підписка на ці сервіси за допомогою RXJS,

що дозволяє реалізувати патерн Observer. Принцип цього патерну полягає у тому, що є Subject (певне джерело даних) та Observer (спостерігач), який підписується на оновлення джерела[23]. При оновленні Subject сповіщає про це всіх підписників. Схематично реалізацію патерну Observer наведено на малюнку 2.3.

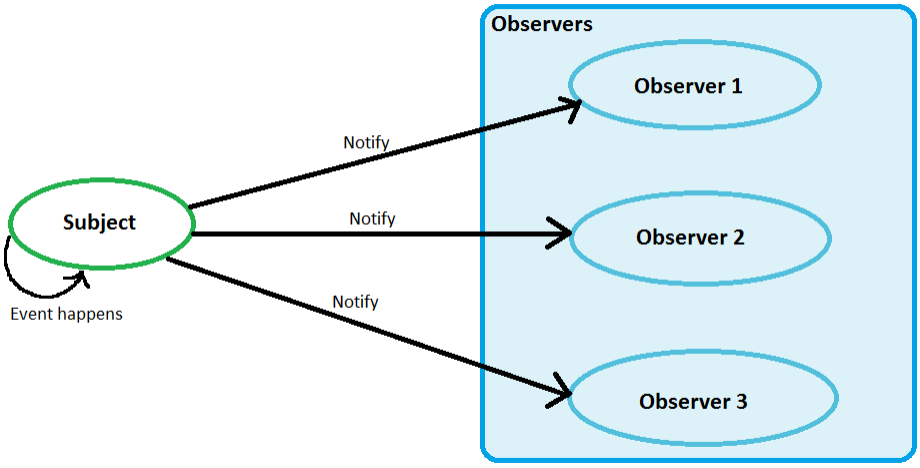


Рисунок 2.3 – Схема патерну Observer

Хоча використання stateful сервісів є прийнятним рішенням, при розширенні застосунку виникатимуть наступні недоліки:

- Дублювання коду
- Складні залежності між сервісами
- Складність рефакторингу
- Відсутність централізованості

Тому альтернативою є використання дещо іншого підходу – централізованого сховища (state), яке зберігає у собі усі спільні дані, потрібні для застосунку. Для цієї мети також можуть підійти сервіси, зпроєктовані таким чином, щоб зберігати усі дані в одному місці, однак є більш доречні рішення – спеціальні бібліотеки, призначені для управління станом застосунку. Найпопулярнішим є Redux – бібліотека, яка доступна як для React, так і для Angular. У Redux існують основні сутності: Store, Actions, Reducers[11].

- Store – сховище, що містить у собі стан додатку та надає можливість підписатися до нього усім частинам застосунку.

- Action – певна дія – певний запит до сховища на модифікацію даних, оскільки напряму це робити не можна, адже воно доступне лише для читання. Надсилається за допомогою використання Dispatch з можливістю прикріплення даних.
- Reducer – обробник дій (actions), який виконує зміну даних у сховищі відповідно від типу запиту та прикріплених до нього даних.

Відповідно, принцип роботи Redux можна описати наступним чином:

1. View відправляє action (дію), у який вказано тип дії та дані, які необхідні для виконання (наприклад – замовлення, яке потрібно додати до загального списку)
2. Reducer обробляє дію, та змінює дані у сховищі
3. Усі підписані на сховище частини застосунку отримують оновлені дані та перелаштовуються відповідно до них

Схематично цей процес наведено на малюнку 2.4.

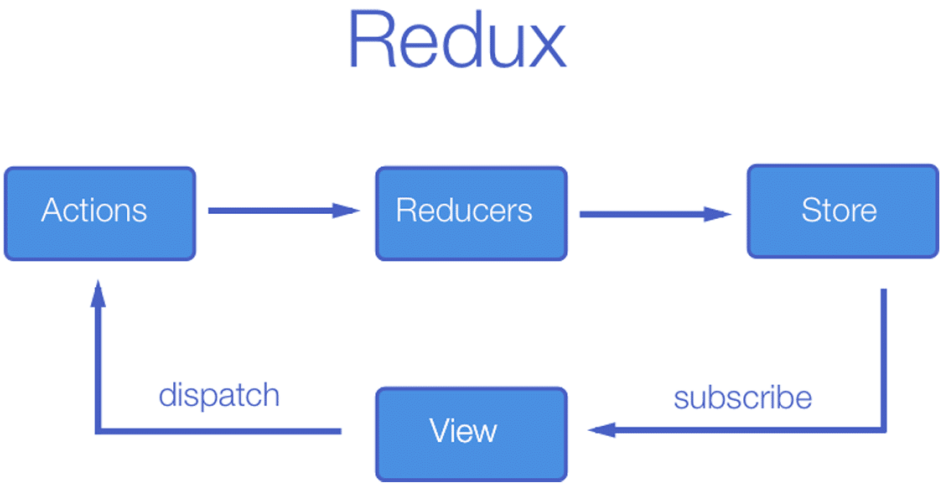


Рисунок 2.4 – Схема роботи Redux

Реалізація Redux для фреймворку Angular представлена декількома бібліотеками: NGRX, NGXS[12]. Обидві із них дозволяють вирішити проблему керування станами у застосунку, однак в даному випадку обрано було саме NGXS з наступних причин:

- Пов’язаний з ним код займає набагато менший об’єм
- Є більш легким для зрозуміння

- Концептуально більше пов’язаний з Angular, на відміну від NGRX, що своїм корінням виходить з React
- Перевага використанню ООП стилю

2.4 Підходи до розробки серверної частини

Розробка серверної частини може відбуватися за допомогою великої кількості різних серверних мов: PHP, Ruby, Java, C, Python, Perl, C#, NodeJS та їх фреймворків. У таблиці 2.1 наведено найпопулярніші фреймворки для розробки серверної частини.

Таблиця 2.1 – Найпопулярніші серверні фреймворки

Мова	Фреймворки
PHP	Laravel, Yii
Python	Django, Flask
Java	Spring Boot
Node JS	Express
Ruby	Ruby on Rails

В даній роботі для створення серверної частини було обрано мову Node JS та фреймворк Express. Вибір було зроблено з огляду на принцип “JS everywhere”, що означає використання однією і тієї ж мови як на клієнті (JavaScript), так і на сервері (Node JS, що синтаксично повністю відповідає JavaScript та базується на ньому). Цей принцип значно спрощує роботу у випадку, коли обидві частини застосунку розробляються однією людиною, адже не вимагає базових знань з одразу двох різних мов і дозволяє швидко перемикати увагу між серверною та клієнтською частинами в разі необхідності. Фреймворк Express було обрано з огляду на його найбільшу популярність в середовищі Node JS, де-факто він являється стандартним у ньому.

Наступним питанням є вибір бази даних. Бази даних поділяються на реляційні (також називаються SQL) та нереляційні (також називаються NoSQL).

Реляційні бази даних є більш структурованими, ніж нереляційні. Дані розміщуються в таблицях, а структура ретельно розробляється. Бази даних NoSQL мають масштабований, високопродуктивний та гнучкий характер, що в основному корисно для роботи з величезними наборами розподілених даних.

Порівняння особливостей SQL та NoSQL наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння SQL та NoSQL

SQL	NoSQL
Вертикально масштабовані	Горизонтально масштабовані
Складаються з таблиць у формі рядків і стовпців, повинні суворо дотримуватися стандартних схем	Базуються на документах, парах ключ-значення, графіках або стовпцях
Заздалегідь визначені та нормалізовані схеми	Динамічні та денормалізовані схеми
Дорого в масштабі	Дешевший масштаб у порівнянні з реляційними базами даних
Підходять для складних запитів	Не підходять для складних запитів,
Не підходять для ієрархічного зберігання даних.	Підходять для ієрархічного зберігання даних
Підходять для високого рівня транзакцій	Погано підходять для високого рівня транзакцій
Не підходять для ієрархічного зберігання даних.	Підходять для ієрархічного зберігання даних

Було обрано нереляційну базу даних MongoDB. Вона є частиною стекою MEAN (абревіатура від MongoDB, Express, Angular, Node JS) – популярного набору технологій розробки веб-застосунків повного циклу.

2.5 Взаємодія клієнта та сервера

В розроблюваному застосунку необхідна реалізація оновлення даних в режимі реального часу, відповідно, звичайні HTTP запити формату ‘запит-відповідь’ не відповідають даній вимозі, оскільки клієнти не знають, у який момент відбулась зміна і потрібно надіслати новий запит. Для вирішення цього завдання існує декілька різних механізмів.

Перший та найбільш очевидний – автоматична відправка запитів на оновлення даних кожен визначений невеликий проміжок часу. Такий підхід має назву Polling. Схематично підхід зображено на рис. 2.6.

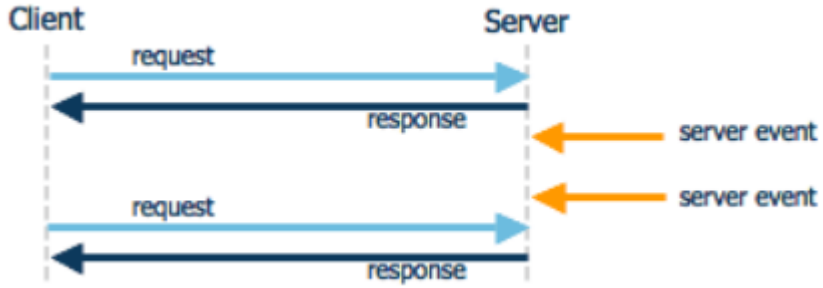


Рисунок 2.6 – Polling

Однак, використання цього способу призводить до великого навантаження на сервер, який вимушений постійно отримувати та оброблювати запити від кожного користувача без необхідності. А також в цьому випадку наявна затримка, яка у найгіршому випадку дорівнює проміжку часу між запитами.

Наступний варіантом є покращена версія Polling, яка має назву Long Polling[14]. Клієнт відправляє запит на сервер, сервер тримає відкритим з'єднання, поки не прийдуть дані або клієнт не відключиться самостійно. Як тільки дані прийшли - надсилається відповідь, з'єднання закривається і відкривається нове. Схему роботи Long Polling зображено на рис. 2.7.

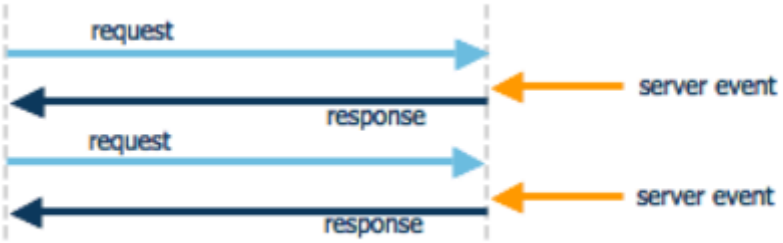


Рисунок 2.7 – Long Polling

Такий варіант є допустимим, однак є не найкращим у випадку великої частоти оновлення даних.

Третій варіант – відмова від HTTP запитів і натомість використання спеціального протоколу Websockets. WebSocket - це протокол зв'язку, що забезпечує можливість обміну даними між браузером та сервером через постійне TCP-з'єднання. Дані передаються по ньому в обох напрямках у вигляді «пакетів», без розриву з'єднання та додаткових HTTP-запитів. Схему роботи WebSockets зображено на рис. 2.8.

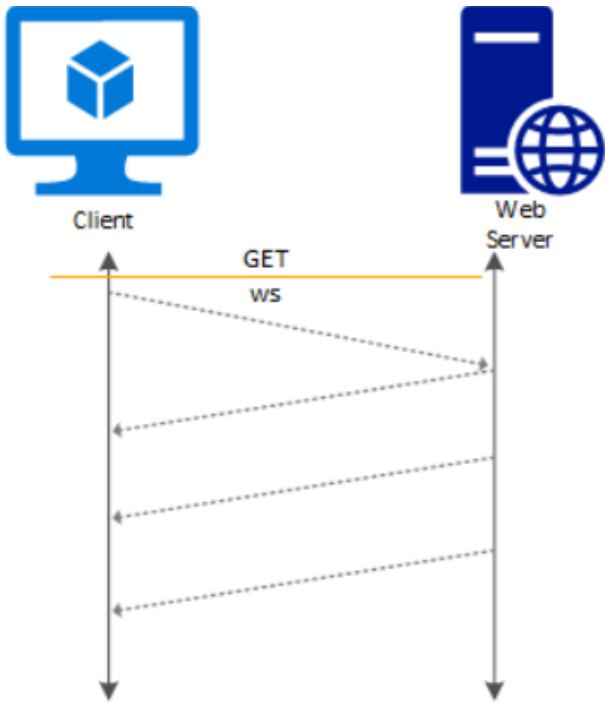


Рисунок 2.8 – Схема роботи Websockets

Рішенням, яке дозволяє розширити можливості Websockets, є бібліотека Socket IO. Socket.IO — бібліотека JavaScript, заснована на веб-сокетах та інших

технологіях[13]. Вона використовує веб-сокети, коли вони доступні, або інші методи оновлення даних в режимі реального часу, якщо ні.

Переваги Socket.IO перед Websockets:

- Можливість надсилати повідомлення до всіх підключених клієнтів
- Підтримка проксування та балансувальників навантаження.
- Підтримка поступової деградації.
- Підтримка автоматичного перепідключення при розриві з'єднання.
- Легкість роботи

2.6 Авторизація та захист даних

Застосунок призначений для внутрішнього користування працівниками закладу, тому необхідно забезпечити обмеження доступу до нього для сторонніх осіб. У веб-застосунках з цією метою широко поширене використання авторизації на основі jwt-токена.

JSON Web Token (JWT) - це стандарт токена доступу на основі JSON, що використовується для передачі даних при аутентифікації в клієнт-серверних програмах. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи. Токен JWT складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису або даних шифрування. Перші два елементи – це JSON об'єкти певної структури. Третій елемент обчислюється на основі перших і залежить від обраного алгоритму[20].

Генерація токена відбувається на стороні сервера, коли користувач відправляє запит для авторизації з правильними даними логін-пароль. У payload (корисне завантаження) записуються деякі необхідні для ідентифікації дані (наприклад, ім’я, роль користувача). Приклад наведено на рис. 2.9.

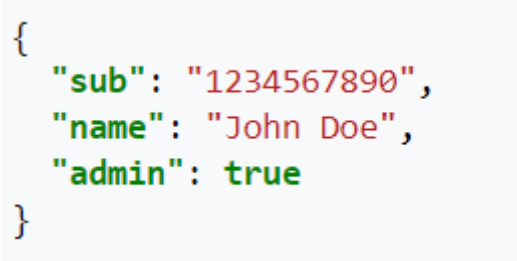


Рисунок 2.9 – Корисне навантаження у JWT-токені

Важливо розуміти, що токен можливо розшифрувати та прочитати payload, тому у ньому не повинні зберігатися чутливі дані (такі як пароль). Однак токен підписується підписом стандарту JSON Web Signature, що не дозволяє модифікацію (підміну) зашифрованих даних.

Після створення токен відправляється на клієнт, зберігається на ньому у локальному сховищі або cookies та використовується у всіх подальших захищених запитах. Для нього виставляється певний час дії, тому у випадку заволодінням ним зловмисниками через деякий час він автоматично стає неактуальним і доступ втрачається. Для бібліотеки Socket IO також передбачено даний механізм захисту, який зручно реалізується бібліотекою socketio-jwt. Вона надає проміжну перевірку (middleware) даних, з якими відбулося підключення, та відмовляє у доступі в разі некоректності/відсутності JWT токена.

Іншим механізмом захисту, який є обов’язковим для впровадження, є шифрування паролів користувачів. У випаду зламу бази даних зловмисники можуть отримати усі дані з неї, в тому числі і паролі, тим самим отримавши повний доступ до кожного існуючого акаунту. На сьогоднішній день широко розповсюдженою практикою для запобігання цій проблемі є збереження у базі даних не самого паролю у відкритому вигляді, а згенерований хеш по секретному ключу (зерну). Хешування є незворотним процесом, тобто неможливо відновити оригінальний пароль користувача. Саме з цієї причини у більшості сервісів функція ‘Забув пароль’ не нагадує існуючий пароль, а пропонує створити новий. Для того, щоб підтвердити вірність введеного паролю при авторизації, відбувається лише порівняння хешів після проходження хеш-функції. Для цієї мети не рекомендується використання власних хеш функцій,

оскільки великий ризик створення ненадійного хешу, який буде реально розшифрувати. Для даний цілей у Node JS існує бібліотека Bcrypt, яка надає необхідні функції для хешування для порівняння хешів. На рис. 2.10 наведено приклад того, як можуть виглядати хеші порівняно з оригінальними паролями.

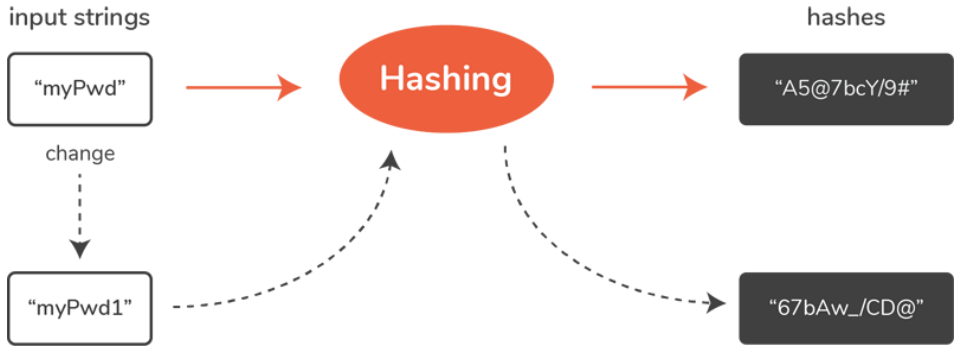


Рисунок 2.10 – Вигляд оригінальних даних та хешованих

2.7 Висновок до розділу

Для того, щоб створити веб-застосунок, існує велика кількість підходів, інструментів, архітектурних рішень. Безумовно, більшість із них є можливими цілком робочими варіантами для досягнення мети. Однак, щоб мінімізувати час, складність, потенційні проблеми при розробці, слід заздалегідь оцінити наявний досвід застосування або можливість швидкого опанування потенційного варіанту, відповідність специфічних особливостей кожного із них заявленій меті. Вибір стеку технологій (який випав на стек MEAN) та архітектурних варіантів в окремо взятих частинах повністю базується на названих критеріях і дозволить максимально якісно і швидко виконати розробку застосунку.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1 Структура клієнтської частини

Angular надає можливість розбивати застосунок на окремі компоненти, кожен з яких містить свою частину HTML-розмітки, CSS-стилізації, та логіки заповнення. Це дозволяє створювати модульну структуру частин застосунку, які не залежать від інших та заново використовувати їх декілька разів без необхідності дублювання, змінюючи лише за необхідності вхідні дані.

Перелік компонентів у застосунку наведено у таблиці 3.1.

Таблиця 3.1 – Перелік компонентів

Назва	Призначення
CafeComponent	Кореневий компонент розділу ‘Зал’
FoodListComponent	Компонент списку страв, які додано до замовлення
KitchenComponent	Кореневий компонент розділу ‘Кухня’
LoginComponent	Форма входу в застосунок
MenuComponent	Навігаційна панель застосунку
ReceiptsPopupComponent	Спливаюче вікно керування чеками
ReceiptPrintPopup	Спливаюче вікно для друку чеку

Правильним підходом у розробці є необхідність дотримуватися принципу ‘Складний сервіс та простий компонент’. Це означає, що потрібно прагнути до зменшення логіки у компонентах та перенесення її до сервісів.

Для досягнення відповідної мети було створено ряд сервісів, перелік яких наведено у таблиці 3.2.

Таблиця 3.2 – Перелік сервісів

Назва	Призначення
FoodListService	Виконання дій, пов’язаних з додаванням, зміною статусу, видаленням позицій у замовленні
ReceiptsService	Виконання дій, пов’язаних з створенням, редагуванням, зміною статусу, закриттям чеків
AuthService	Виконання дій, пов’язаних з авторизацією

Залежності компонентів від сервісів наведено на рисунку 3.1.

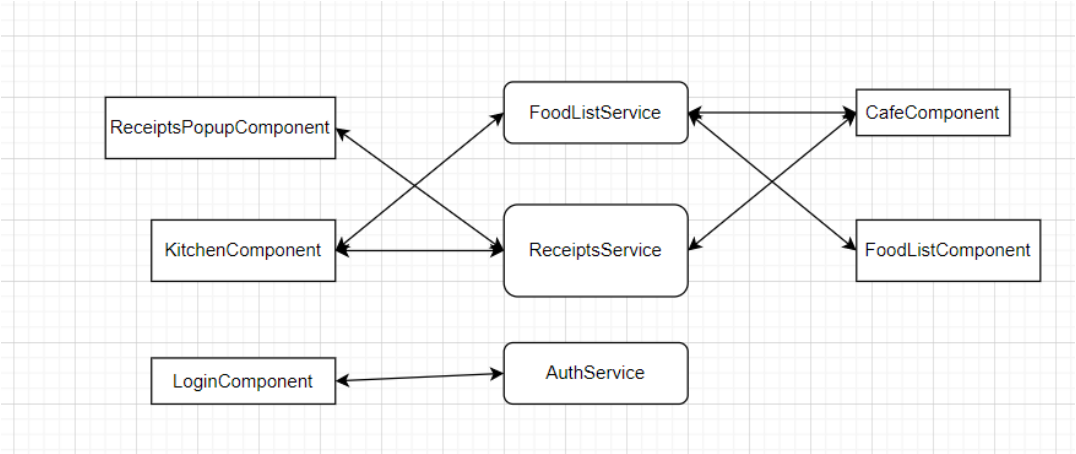


Рисунок 3.1 – Взаємодія компонентів та сервісів

3.2 Опис бази даних

База даних складається з декількох документів (моделей).

Перша модель – FoodItem, схема наведено у таблиці 3.3.

Таблиця 3.3 – Модель FoodItem

Поле	Тип даних
_id	ObjectId
menu_id	ObjectId

Продовження таблиці 3.3

Поле	Тип даних
receipt_id	ObjectId
tableName	String
status	String

Наступна модель – Menu, у якій зберігається інформація про доступні позиції у меню – назва, ціна, зображення. Структуру наведено у таблиці 3.4.

Таблиця 3.4 – Модель Menu

Поле	Тип даних
_id	ObjectId
title	String
price	Number
photo	String

Модель Receipt, у якій зберігається інформація про відкриті на даний момент – їх статус, та до якого стола вони прив’язані. Структуру наведено у таблиці 3.5.

Таблиця 3.5 – Модель Receipt

Поле	Тип даних
_id	ObjectId
table_id	String
status	String

Модель Tables, де зберігаються усі наявні столи та їх назви. Структуру наведено у таблиці 3.6.

Таблиця 3.6 – Модель Tables

Поле	Тип даних
_id	ObjectId
name	String

На рисунку 3.2 зображена схема взаємодій документів.

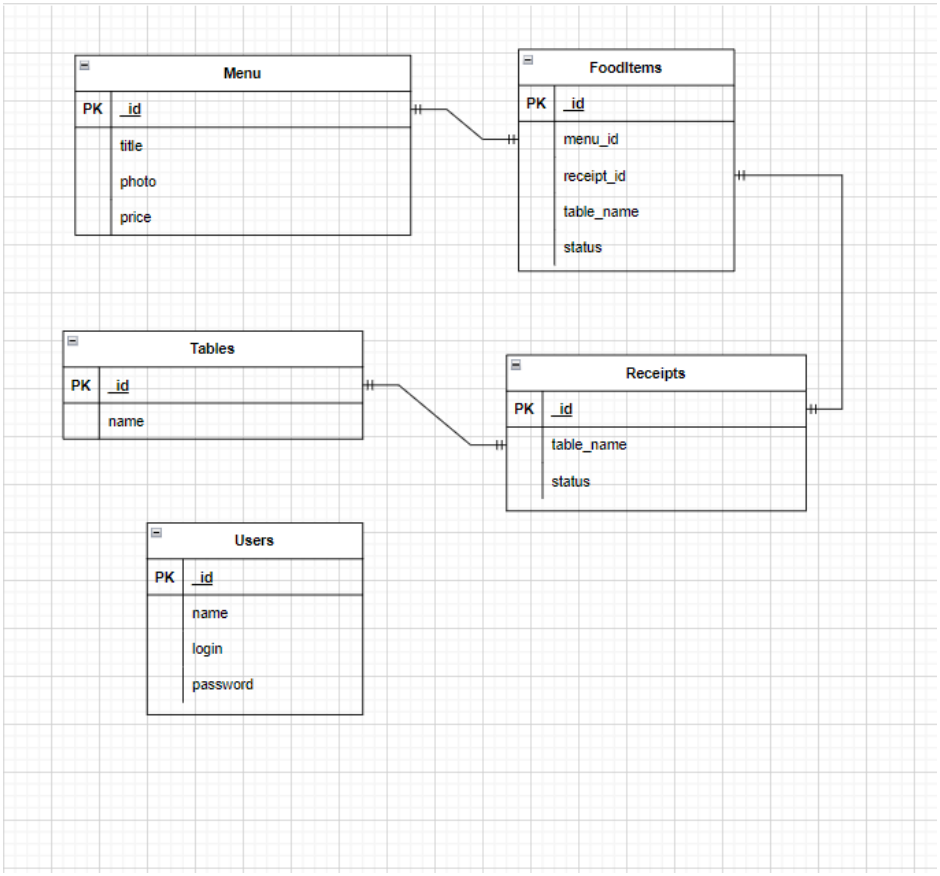


Рисунок 3.2 – Схема бази даних

3.3 Структура серверної частини

Серверна частина поділена на декілька частин:

- Точка входу (app.js) – виконує ініціалізацію застосунку
- Сервісна частина (services.js) – запити до бази даних
- Обробка подій (sockets.js) – обробка подій відповідно до отриманої події Socket IO.
- Моделі бази даних (models/..) – моделі, наведені у розділі 3.2.

Обрана система взаємодії з клієнтом передбачає обробку подій, які надходять від клієнта за допомогою Socket IO. Події складаються з певної унікальної назви (типу події) та даних, що додаються до неї (наприклад, id рахунку для закриття). Загальна схема обробника події. Загальний вигляд обробника події наведено на рисунку 3.3.

```
socket.on("details", (...args) => {
  // ...
});
```

Рисунок 3.3 – Обробник подій

Перелік подій у застосунку наведено у таблиці 3.7.

Таблиця 3.7 – Підтримувані події у застосунку

Тип події	Призначення
addTable	Створити стіл
createReceipt	Створити рахунок
addFoodToReceipt	Додати позицію в рахунок
moveFoodToKitchen	Перевести позицію на кухню
moveFoodToGiven	Перевести позицію в статус ‘Передано клієнту’
moveFoodToReadyForGive	Перевести позицію в статус ‘Готово до видачі’
getFoodItems	Отримати перелік позицій
removeFood	Видалити позицію
getMenu	Отримати меню

3.3 Інструкція для користування

В застосунку передбачається робота двух осіб – працівників залу (офіціантів) та працівників кухні (кухарів/комплектувальників). Діаграму зображено на рисунку 3.4.

Рисунок 3.4 – Діаграма прецендентів

Застосунок призначений лише для працівників закладу, відповідно перед початком роботи акаунт має бути зареєстрованим заздалегідь вручну адміністратором. При відкритті застосунку необхідно ввести отримані логін та пароль, після чого відбудеться авторизація.

У лівій частині екрану зображено ім’я користувача та меню, у якому доступні пункти:

- Кухня: розділ для перегляду замовлень, які знаходяться на видачі або очікують приготування.
- Зал: розділ для перегляду стану столів у залі, створення та управління чеками.

Зображення меню наведено на рисунку 3.5.

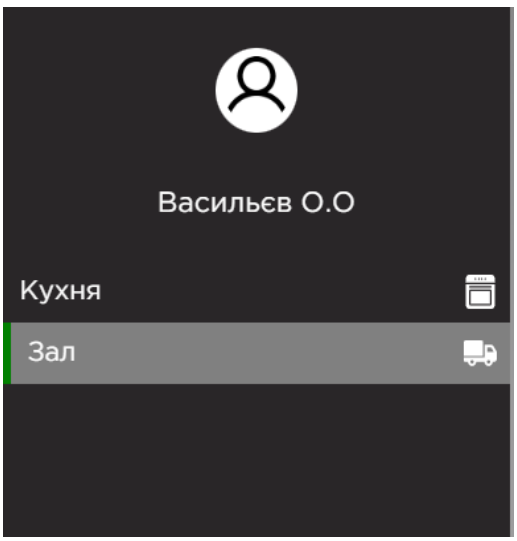


Рисунок 3.5 – Меню застосунку

Розділ “Зал” містить у собі зображення усіх столів, які відкриті для обслуговування, та список відкритих чеків на кожному із них. Зображення розділу наведено на рисунку 3.6.

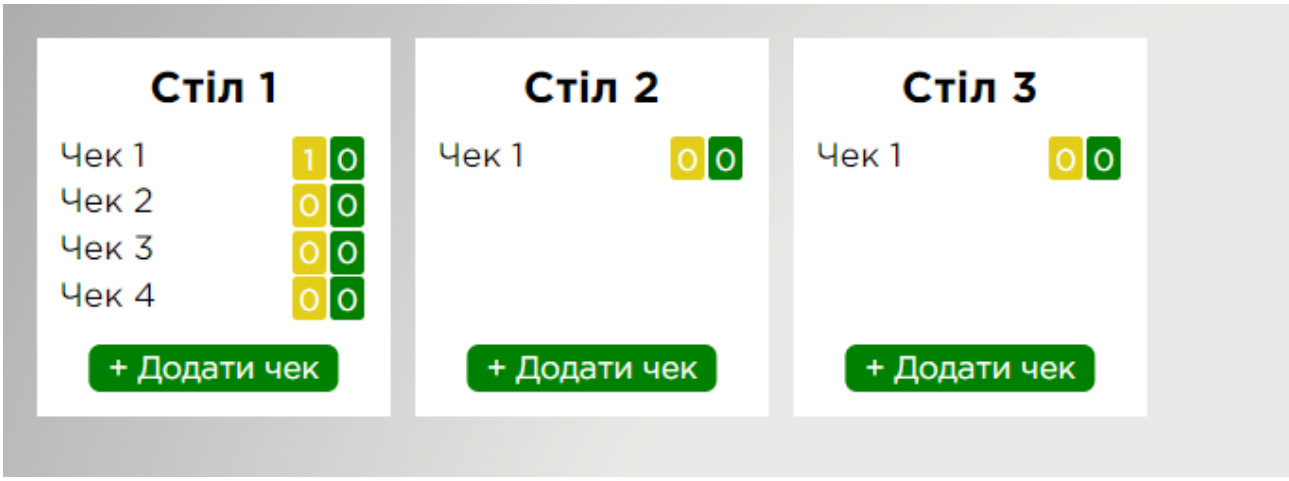


Рисунок 3.6 – Столи з закріпленими чеками

Напроти кожного чеку зображено кількість позицій у замовленні, які перебувають у процесі приготування (жовтий колір), та кількість позицій, які доступні на видачі (зелений колір).

Для того, щоб відкрити новий чек, потрібно натиснути відповідну кнопку ‘Додати чек’ знизу від картки стола. Зображення відкритого вікна наведено на рисунку 3.7.

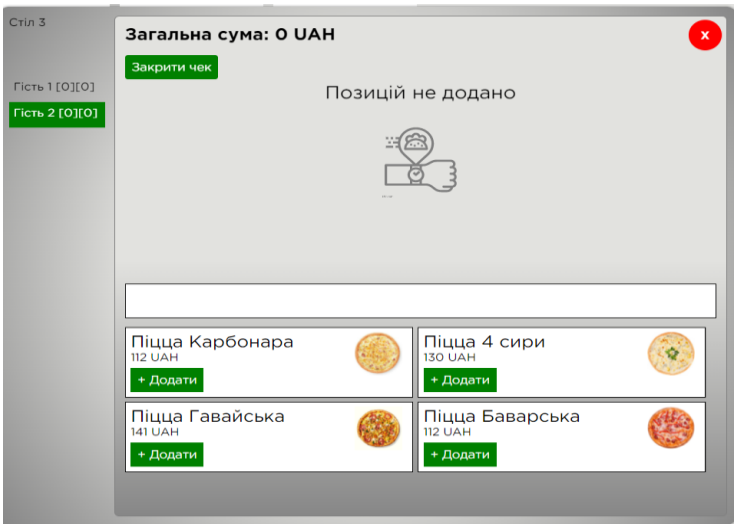


Рисунок 3.7 – Спливаюче вікно управління чеками

Ліва панель дає змогу швидко перемикатися між відкритими чеками одного стола. Права (основна) панель містить у собі верхню частину, у якій відображається інформація про замовлення, додані до чеку, їх ціну, та елементи керування. Для додавання нового елементу до замовлення необхідно натиснути кнопку ‘Додати’ (попередньо за необхідністю скориставшись полем пошуку).

Зображення доданої позиції меню наведено на рисунку 3.8.

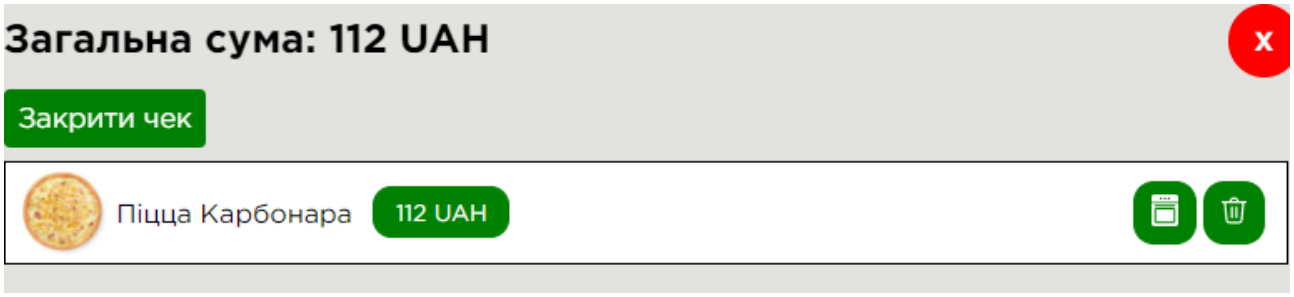


Рисунок 3.8 – Додана в чек позиція

Після цього позиція додається до замовлення. На даному етапі її можна скасувати, натиснувши на праву кнопку керування, або відправити її на кухню, натиснувши ліву кнопку.

Пісня цього статус позиції змінюється на ‘Готується’, і для подальшої роботи слід перейти у розділ ‘Кухня’ > ‘Очікує приготування’. Це дошка для кухарів, на якій відображаються назви та зображення усіх замовлень, які необхідно приготувати.

Зображення дошки наведено на рисунку 3.9.

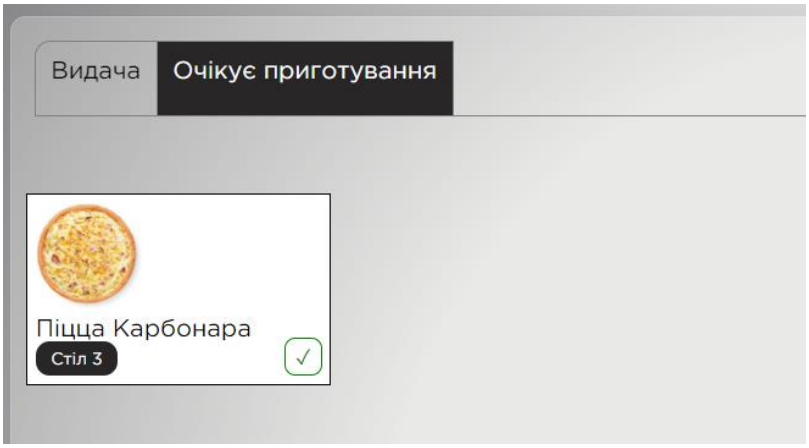


Рисунок 3.9 – Дошка очікуючих приготування страв

Коли замовлення готове, слід натиснути на кнопку підтвердження, і замовлення переходить у статус ‘Готове до видачі’. Усі замовлення з даним статусом відображені у сусідній вкладці розділу ‘Кухня’.

Зображення дошки наведено на рисунку 3.10.

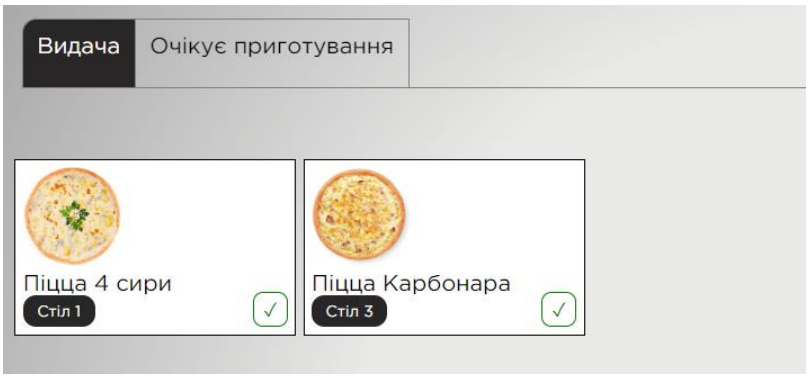


Рисунок 3.10 – Дошка готових до видачі страв

Офіціант може підтвердити видачу замовлення у даній вкладці, або у вікні керування чеком.

Зображення кнопки підтвердження наведено на рисунку 3.11.

Рисунок 3.11 – Кнопка підтвердження видачі

Після підтвердження замовлення переходить у статус ‘Завершено’. Якщо всі позиції мають такий статус, чек можна закрити, натиснувши на кнопку “Закрити чек’. Відбудеться формування рахунку, який буде доступним для друку.

3.4 Тестування застосунку

Тестування програмного забезпечення – це процес перевірки продукту на відповідність усім поставленим вимогам, який здійснюється шляхом спостереження за роботою у певних визначених ситуаціях і порівнянням результату з очікуваним.

Для перевірки роботи застосунку буде використано два види тестування: мануальне (ручне) та модульне (unit-testing).

При мануальному тестуванні тестувальник виконує роль користувача, перевіряючи, що програма працює коректно при усіх діях користувача, в тому числі і нестандартних. Для такого виду тестування використовуються тест-кейси - сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції або її частини, що тестується. Структура тест кейсу може бути різною, однак деякі поля є спільними для всіх випадків:

- Унікальний ідентифікатор тест-кейсу
- Назва
- Передумови
- Кроки
- Очікуваний результат

На рисунку 3.12 наведено приклад одного з тест-кейсів, що були створені для ручного тестування.

Ідендифікатор	CF-01	
Назва	Додання чеку	
Передумови	Користувач авторизований; Відкрита сторінка ‘/cafe’; У столі немає відкритих чеків	
Кроки	Очікуваний результат	Статус
Натиснути кнопку ‘Додати чек’	З’являється новий чек з номером 1	Пройдено

Рисунок 3.12 – Приклад тест-кейсу

Модульне (unit) тестування — це спосіб тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У випадку тестування Angular, під цю процедуру потрапляють компоненти, сервіси, функції, поля з окремих файлів. Unit-тести створюються у процесі розробки або відразу після неї, щоб перевірити коректність функціонування коду.

Для unit-тестування в додатках Angular використовується фреймворк Jasmine. Тестові файли мають формат <назва тестованого файлу>.spec.ts та запускаються за допомогою команди npm run test[24].

Загальна структура тесту окремої частини коду наведена на рисунку 3.13

```
import {...} from '...';
...

describe('related tests group', () => {
    beforeEach(() => {...});

    it('test description', async(() => {
        expect(...).toBe(...);
    }));

    ...
});
```

Рисунок 3.13 – Структура юніт-тесту

Результат проходження тестів наведено на рисунку 3.14.

```
✓ Browser application bundle generation complete.
Chrome 101.0.4951.67 (Windows 10): Executed 12 of 27 (skipped 15) SUCCESS (0.131 secs / 0.083 secs)
TOTAL: 12 SUCCESS
```

Рисунок 3.14 – Успішно пройдені тести

3.5 Висновок до розділу

В межах розділу було створено веб-застосунок для підтримки діяльності закладу громадського харчування. Виконано мануальне та модульне тестування з використанням Jasmine. Наведено вичерпну інструкцію для користувача. Розроблений застосунок відповідає поставленому завданню, має необхідний базовий функціонал, адаптивний та зручний інтерфейс, працює швидко та надійно, успішно пройшов тестування.

ВИСНОВОК

В процесі виконання роботи було розглянуто стан та впровадження застосунків для автоматизації закладів громадського харчування. Було встановлено, що використання застосунків у закладаї є дуже поширеним, що дало можливість визначити особливості рішень-аналогів і використати результати у роботі.

Проаналізовано основні підходи до реалізації веб-застосунків, зокрема програмні засоби реалізації клієнтської та серверої частин, забезпечення їх взаємодії, архітектурні рішення. Відповідно до переваг та недоліків обрано стек MEAN, до якого входять Node JS (фреймворк Express JS) для серверної частини, Angular для клієнтської частини. Взаємодія у режимі реального часу забезпечена бібліотекою Socket IO.

Створено веб-застосунок для підтримки діяльності закладу громадського харчування. Виконано мануальне та модульне тестування з використанням Jasmine. Розроблений застосунок є відкритим для подальшого розширення функціоналу та перенесення на інші платформи. Відповідає поставленому завданню, має необхідний базовий функціонал, адаптивний та зручний інтерфейс, працює швидко та надійно. Недоліком є наявність лише мінімального набору функцій та відсутність універсальності. Однак, після проведення необхідних базових налаштуваннь та доробок під особливості закладу та виконання інтеграції з POS-системами може бути використаним у закладі громадського харчування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. П'ятницька Н. О. Організація обслуговування у підприємствах ресторанного господарств. Київ : КНТЕУ, 2005. 584 с.
2. Архіпов В. Організація ресторанного господарства. Київ : Центр учб. літ., Фірма «Інкос, 2007. 335 с.
3. Автоматизація ресторану від Poster. URL: https://joinposter.com/ua/business/restaurant (дата звернення: 19.05.2022).
4. Клієнт-серверна архітектура – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура (дата звернення: 19.05.2022).
5. Основи CSS | MDN. MDN Web Docs. URL: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics (дата звернення: 19.05.2022).
6. JavaScript | MDN. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата звернення: 19.05.2022).
7. React Documentation. URL: https://uk.reactjs.org/docs/getting-started.html (дата звернення: 19.05.2022).
8. Angular Documentation. URL: https://angular.io/ (дата звернення: 19.05.2022).
9. Vue.js - the progressive javascript framework. URL: https://vuejs.org/ (дата звернення: 19.05.2022).
10. Handbook - The TypeScript Handbook. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 19.05.2022).
11. Redux JS Faq. URL: https://redux.js.org/faq (дата звернення: 19.05.2022).
12. Introduction - NGXS. URL: https://www.ngxs.io/ (дата звернення: 19.05.2022).
13. Introduction | socket.io. URL: https://socket.io/docs/v4/ (дата звернення: 19.05.2022).

14. Long polling. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/long-polling (date of access: 19.05.2022).
15. MongoDB: the application data platform. URL: https://www.mongodb.com/ (дата звернення: 19.05.2022).
16. Timms S., Antani V., Mantyla D. JavaScript: functional programming for javascript developers. Packt Publishing, 2016. 646 c.
17. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/ (дата звернення: 19.05.2022).
18. WebSocket. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/websocket (дата звершення: 19.05.2022).
19. WebSocket: lightweight client-server communications. Andrew Lombardi : O'Reilly Media, Inc, 2015. 144 c.
20. JSON Web Tokens. URL: https://jwt.io/ (дата звернення: 19.05.2022).
21. NgRx docs. URL: https://ngrx.io/ (дата звернення: 19.05.2022).
22. Node.js docs. URL: https://nodejs.org/uk/docs/ (дата звернення: 19.05.2022).
23. RxJS. URL: https://rxjs.dev/ (дата звернення: 19.05.2022).
24. Jasmine documentation. URL: https://jasmine.github.io/ (дата звернення: 19.05.2022).

ДОДАТОК А
Приклади тест-кейсів

Ідендифікатор	CF-02	
Назва	Переведення позиції на кухню	
Передумови	Користувач авторизований; Відкрито вікно керування чеками; У чеку немає позицій	
Кроки	Очікуваний результат	Статус
Натиснути кнопку ‘Додати’ на одому з пунктів меню	Пункт меню з’являється у списку у верхній частині вікна; Загальна вартість збільшилася на вартість позиції; Доступні кнопки переведення на кухню та видалення	Пройдено
Натиснути на кнопку переведення на кухню	Кнопки керування зникають; Замість кнопок керування висвітлюється ‘Готується’; Кількість позицій у лічильнику ‘Готується’ збільшилася на 1	Пройдено
Перейти у розділ ‘Кухня’, обрати підрозділ ‘Очікують приготування’	У списку відображається переведена позиція та відповідний номер стола	Пройдено

Ідендифікатор	CF-03	
Назва	Подтвердження готовності замовлення	
Передумови	Користувач авторизований; Відкрита сторінка ‘/kitchen’; Обраний розділ ‘Очікують приготування’; У розділі наявна позиція	
Кроки	Очікуваний результат	Статус
Натиснути кнопку підтвердження на позиції	Позиція зникає зі списку	Пройдено
Перейти на вкладку ‘Готові до видачі’	У списку наявне підтверджене замовлення	

ДОДАТОК В
Лістинг

1. Файл роботи зі State

```
@State<StateModel>({
  name: 'cafe',
  defaults: {
    user: null,
    tables: [],
    currentTableIndex: null,
    currentReceiptId: null,
    currentReceiptItems: [],
    menu: [],
  },
})
@Injectable()
export class CafeState {
  @Selector()
  static user(state: StateModel): User | null {
    return state.user;
  }
  @Selector()
  static tables(state: StateModel): Table[] {
    return state.tables;
  }
  @Selector()
  static currentTableIndex(state: StateModel): number | null {
    return state.currentTableIndex;
  }
  @Selector()
  static menu(state: StateModel): Menu[] {
    return state.menu;
  }
  @Selector()
  static currentReceiptId(state: StateModel): string | null {
    return state.currentReceiptId;
  }
  @Selector()
  static currentReceiptItems(state: StateModel): ReceiptItem[] {
    return state.currentReceiptItems;
  }
  @Action(SetTables)
  setTables(ctx: StateContext<StateModel>, action: SetTables) {
    const state = ctx.getState();
    ctx.setState({
      ...state,
      tables: action.tables,
    });
  }
```

```
  @Action(SetCurrentTableIndex)
  setCurrentTableIndex(
    ctx: StateContext<StateModel>,
    action: SetCurrentTableIndex
  ) {
    const state = ctx.getState();
    ctx.setState({
      ...state,
      currentTableIndex: action.currentTableIndex,
    });
  }
  @Action(SetMenu)
  setMenu(ctx: StateContext<StateModel>, action: SetMenu) {
    const state = ctx.getState();
    ctx.setState({
      ...state,
      menu: action.menu,
    });
  }
  @Action(SetCurrentReceiptId)
  setCurrentReceiptId(
    ctx: StateContext<StateModel>,
    action: SetCurrentReceiptId
  ) {
    const state = ctx.getState();
    ctx.setState({
      ...state,
      currentReceiptId: action.id,
    });
  }
  @Action(SetCurrentReceiptItems)
  setCurrentReceiptItems(
    ctx: StateContext<StateModel>,
    action: SetCurrentReceiptItems
  ) {
    const state = ctx.getState();
    ctx.setState({
      ...state,
      currentReceiptItems: action.items,
    });
  }
}
```

2. CafeComponent.html

```
<div class="container">
  <div class="tables-area">
    <div class="table" *ngFor="let table of tables$ | async; let i = index">
      <h3>{{ table.name }}</h3>
      <ng-container *ngIf="table.receipts?.length; else noReceipts">
        <div
          class="table-row"
```

```
          *ngFor="let receipt of table.receipts; let j = index"
          (click)="openTable(i + 1, receipt._id)"
        >
          <span class="receipt-number">Чек {{ j + 1}}</span>
          <div>
            <span class="receipt-pending-count"> {{ getPendingCount(receipt._id) }}
</span
            ><span class="receipt-given-count">{{ getGivenCount(receipt._id
)}}</span>
          </div>
        </div>
      </ng-container>
      <ng-template #noReceipts> <div class="free-table"><img src="assets/free-
table.svg" alt=""></div></ng-template>
      <button class="add-button" (click)="addReceipt(table._id, table.name)">
        + Додати чек
      </button>
    </div>
  </div>
</div>
```

3. app.js

```
const express = require('express');
const path = require('path');
const cors = require('cors');
const app = express();
const http = require('http').Server(app);
const mongoose = require('mongoose');

const io = require('socket.io')(http, {cors: {
    origin: '*',
  }});


app.use(cors());
app.use(express.json());

require('./sockets')(io);

const start = async () => {
  try {
    await mongoose.connect(env.CONN_STRING, {
      useNewUrlParser: true, useUnifiedTopology: true,
    });
    app.listen(8080);
    console.log('Started');
  } catch (err) {
    console.error(`Error on server startup: ${err.message}`);
  }
};
```

```
start();


http.listen(4444, () => {
    console.log('Listening on port 4444');
  });
```