НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено

Завідувач кафедри

_____Стіренко С.Г.

«___»_____2021 р.

Дипломний проєкт

на здобуття освітнього ступеня бакалавра

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

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Прогресивний веб-додаток для закладів швидкого харчування»

Виконав:
студент IV курсу, групи ІП-74
Скригун Владислав
Олександрович _____

Керівник:
Долголенко Олександр
Миколайович _____

Консультант з нормоконтролю:
професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
_____
_____ _____

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

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені 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
_____Стіренко С.Г.
«___»_____2021 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Скригуну Владиславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту «Прогресивний веб-додаток для закладів швидкого харчування», керівник проєкту Долголенко Олександр Миколайович, кандидат технічних наук, доцент, затвердженінаказом по університету від 11.05. 2021 р. № 1139-с

2. Термін здачі студентом закінченого проєкту 20 травня 2021 р.

3. Вихідні дані до проєкту: технічна документація, теоретичні та статистичні дані, ресурси мережі Інтернет.

4. Зміст пояснювальної записки: аналіз предметної області та огляд існуючих веб-додатків,вибір інструментів розробки, розробка веб-додатку та аналіз розробленого веб-додатку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): схема структури бази даних, лістинг програми.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В. П., професор		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	10.12.2021-15.12.2021	
2	Вивчення та аналіз завдання	15.12.2021-15.03.2021	
3	Розробка архітектури та загальної структури системи	15.03.2021-25.03.2021	
4	Розробка структур окремих підсистем	25.03.2021-5.04.2021	
5	Програмна реалізація системи	5.04.2021-15.04.2021	
6	Оформлення пояснювальної записки	15.04.2021-20.05.2021	
7	Захист програмного продукту	25.04.2021- 22.05.2021	
8	Попередній захист	23.05.2021	
9	Захист	17.06.2021	

Студент-дипломник _____ Скригун В.О.
(підпис)

Керівник роботи _____Долголенко О.М.
(підпис)

АНОТАЦІЯ

У дипломному проєкті було детально розглянуто існуючі вебдодатки для закладів швидкого харчування, а також їх принцип роботи. Було проаналізовано їхні слабкі та сильні сторони, а також принципи їхньої роботи. З урахуванням існуючих рішень було запропоновано створити новий вебдодаток для закладів швидкого харчування, який вирішував би основні недоліки існуючих вебдодатків. Розроблена програма дозволяє користувачеві робити замовлення у закладах швидкого харчування. Адміністратор додатка має змогу редагувати, видаляти, додавати страви та спеціальні акції у додатку. Програмний продукт було розроблено із застосуванням мови програмування Javascript. Серверна частина додатку була створена на платформі Node.js. Задля створення користувацького інтерфейсу був використаний React.js та Angular 8 для інтерфейсу адміністратора.

В результаті роботи було розроблено веб-додаток для закладів швидкого харчування, що є не гірший за існуючі аналоги та усуває їхні недоліки.

ANNOTATION

The diploma project examined in detail the existing web applications for fast food restaurants, as well as their principle of operation. Their strengths and weaknesses were analyzed, as well as the principles of their work. Based on existing solutions, it was proposed to create a new web application for fast food restaurants, which would address the main shortcomings of existing web applications. The developed program allows the user to place orders in fast food restaurants. The administrator of the application can edit, delete, add dishes and special promotions in the application. The software product was developed using the Javascript programming language. The server part of the application was created on the Node.js platform. React.js for the administrator interface was used to create the user interface.

As a result, a web application for fast food restaurants was developed, which is not worse than existing analogues and eliminates their shortcomings.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1.	А 4		Завдання на дипломний проєкт	2	
2.	А4	ІАЛЦ.467200.001 ОП	Опис проєкту	1	
3.	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
4.	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	52	
5.	А4	ІАЛЦ.467200.004 Д1	Схема системи	1	
6.	А4	ІАЛЦ.467200.005 Д2	Діаграма класів	1	
7.	А4	ІАЛЦ.467200.006 Д3	Алгоритм створення замволення	1	
8.	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	18	

				ІАЛЦ.467200.001 ВП		
	ПІБ	Підп.	Дата			
Розроб.	Скригун В.О.			Відомість дипломного проєкту	Арк.	Аркушів
Перев.	Долголенко О.М.				1	5
Реценз.					НТУУ «КПІ» ФІОТ ІП-74	
Н. контр.	Сімоненко В.П.					
Затверд.	Стіренко С.Г.					

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Прогресивний веб-додаток для закладів швидкого харчування»

Київ – 2021

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Скригун В.О.			Прогресивний веб-додаток для закладів швидкого харчування Технічне завдання	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Долголенко О.М.					
Реценз.						НТУУ КПІ ФІОТ ІП-74	
Н. Контр.		Сімоненко В.П.					
Затвердив		Стіренко С. Г.					

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Назва розробки: Прогресивний веб-додаток для закладів швидкого харчування.

Область застосування: управління та адміністрування у закладах швидкого харчування.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проєкту, що його затверджено кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» імені Ігоря Сікорського.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення простого на зручного прогресивного веб-додатку для закладів швидкого харчування.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література за темою проєкту, монографії, публікації в періодичних виданнях та мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється.

- можливість реєстрації у вебдодатку та мати особистий кабінет;
- можливість зробити замовлення;
- можливість управляти контентом вебдодатка.

5.2. Вимоги до програмного забезпечення

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

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

	Дата
Вивчення літератури	20.12.2020
Розроблення і узгодження технічного завдання	03.02.2021
Моделювання структури програмного забезпечення	16.03.2021
Розробка програмного забезпечення	11.05.2021
Доопрацювання і налагодження програми	01.06.2021
Оформлення документації дипломного проєкту	09.06.2021

Пояснювальна записка
до дипломного проєкту
на тему: «Прогресивний веб-додаток для закладів швидкого харчування»

Київ – 2021 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

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

HTTPS (Hyper Text Transfer Protocol Secured) - розширення протоколу HTTP для підтримання шифрування задля 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 у мережі.

PWA - прогресивний веб-додаток

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) - легкий формат для зберігання і транспортування даних.

npm (Node package manager) - програмне забезпечення, яке дозволяє встановлювати і керувати залежностями проєкту.

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

ВСТУП

З кожним роком темп життя людей збільшується. Однією з проблем, з якою стикаються багато, стає можливість швидко перекусити в кафе або ресторані в обід. Дана проблема найбільш актуальна для офісних працівників, учнів та студентів різних навчальних закладів, у яких час на обідню перерву досить обмежений і швидкість приготування страв є найважливішим критерієм вибору закладу для обіду. Однак через великі черг і довгого очікування приготування страв існує ризик не вкластися за часом.

Рішенням даної проблеми може стати можливість попереднього замовлення їжі, що дозволить отримати вже приготовану страву без довгого очікування. Тому все більше закладів швидкого харчування впроваджують в свої бізнес-процеси можливість створення замовлення онлайн. Такий спосіб передусім дозволяє користувачам, з одного боку, уникати черг і економити час. З іншого - заощадити час закладів на приймання замовлень та інформування клієнтів.

Автоматизація процесу онлайн-замовлення їжі в закладах швидкого харчування має на увазі розроблення інформаційної системи, яка дозволить клієнтам створювати замовлення і відслідковувати його готовність; співробітникам ресторану обробляти замовлення; а власникам (центральний офіс, head-office) вести статистику і управляти торговими точками (їх асортиментом, категоріями, користувачами і т.п.).

Даний проєкт представляє собою частину загальної інформаційної системи онлайн-замовлення їжі в закладах швидкого харчування і її метою є розроблення недорогого вебдодатку для head-офісу, яке дозволить управляти торговими точками, категоріями, товарами, а також переглядати статистику по замовленнях.

Наразі існує безліч інформаційних систем для автоматизації процесів центрального офісу закладів швидкого харчування. Дані системи мають свої переваги і недоліки, але більшість з них мають високу вартість і закритий нерозширюваний код, що накладає певні обмеження їх застосування та налаштування під конкретні заклади, що, в свою чергу, і визначає актуальність роботи.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз ринку готових рішень;
- виявити функціональні можливості;
- скласти набір функціональні вимог;
- спроектувати призначений для користувача інтерфейс;
- вибрати стек технологій;
- розробити веб-додаток.

Автоматизація head-офісу закладів швидкого харчування дозволить збільшити прибуток і скоротити витрати, підвищити рівень якості обслуговування клієнтів, вести контроль ефективності роботи, поводити аналіз поточної діяльності і приймати рішення по її оптимізації, а також планувати подальший розвиток.

Тому тема даного дослідження є актуальною, а отримані результати мають теоретичну та практичну цінність.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ ВЕБДОДАТКІВ

1.1. Визначення вебдодатку

Останніми рокам Web-додатки стрімко розвиваються, поступово витісняючи настільні рішення і стаючи найважливішим компонентом бізнесу в сучасному світі. Все частіше компанії вдаються до послуг розробки веб-додатків (Web-application), аби ефективно вирішувати широкий спектр бізнес-завдань.

Вебдодаток – це клієнт-серверний додаток, основна частина якого міститься на вилученому сервері, а призначений для користувача інтерфейс (UI) відображається в браузері у вигляді вебсторінок.

Для запуску вебдодатку користувачеві не потрібно встановлювати ніяких додаткових програм, воно запускається на будь-якому пристрої з браузером і з доступом в інтернет.

Робота клієнта не залежить від операційної системи, що стоїть на комп'ютері користувача, тому при розробці веб-додатків немає необхідності писати окремі версії для Windows, Linux, Mac OS і інших операційних систем.

Для створення серверної частини веб-додатків використовуються такі мови програмування, як: PHP, ASP, ASP.NET, Perl, C / C ++, Java, Python, Ruby, NodeJS.

Для реалізації клієнтської частини використовують HTML, CSS, JavaScript, Ajax.

WEB-додаток - це вебсайт, з частково розробленими сторінками, кінцевий вміст яких визначається за запитом користувача. Перевагою є його кросплатформенність, тобто не важливо, яка операційна система встановлена на пристрої.

Перша сторінка, на яку потрапляє відвідувач після входу в браузер називається статичною, тобто її вміст завжди стабільно, незмінно.

Більшість інших сторінок, на які користувач потрапляє, натиснувши на кнопку або зайшовши у вкладку - динамічні, тому що формуються під певний запит користувача.

Вебдодатки використовують найрізноманітніші компанії. Наприклад, AMAZON, Microsoft, новинний сайт CNN, електронний журнал The Economist.

Переваги web-додатків.

1. Доступ з будь-якого пристрою. З веб-додатком можна працювати в будь-якій точці світу з комп'ютера, планшета або смартфона, підключеного до Інтернету.

2. Економія. Веб-додатки працюють на всіх платформах і виключають необхідність розроблення програми окремо для Android і iOS.

3. Адаптивність. Якщо для нативних додатків потрібні певні ОС, то для роботи з веб-додатком підходить будь-яка ой операційною системою (Windows, MAC, Linux і т.д.) і будь-який браузер (Internet Explorer, Opera, FireFox, Google Chrome і т.д.).

4. Відсутність клієнтського програмного забезпечення. Дешевша і простіша установка, обслуговування та модернізація клієнтського інтерфейсу. Оновлення до останньої версії відбувається при черговому завантаженні сторінки.

5. Мережева безпека. Вебсистема має єдину точку входу, захистити і налаштувати безпеку якої можна централізовано.

6. Масштабованість. Зі зростанням навантаження на систему не треба нарощувати потужність клієнтських місць. Веб-додаток дозволяє обробляти більшу кількість даних, як правило, тільки силами апаратних ресурсів, без переписування коду і зміни архітектури.

7. Захист від втрати даних. Дані користувачів зберігаються в «хмарі», за цілісність якого відповідають хостинг-провайдери, і захищені від втрати при пошкодженні жорсткого диска комп'ютера.

1.2. Структура вебдодатку

Вебдодатки можна розділити на кілька типів залежно від різних поєднань їх основних складових.

Backend (бекенд або серверна частина програми) працює на віддаленому комп'ютері, який може знаходитися де завгодно. Вона може бути написана на різних мовах програмування: PHP, Python, Ruby, C# та інших. Якщо створювати додаток використовуючи тільки серверну частину, то в результаті будь-яких переходів між розділами, відправок форм, оновлення даних, сервером буде генеруватися новий HTML-файл і сторінка в браузері перезавантажуватиметься.

Frontend (фронтенд або клієнтська частина програми) виконується в браузері користувача. Ця частина написана на мові програмування Javascript. Додаток може складатися тільки з клієнтської частини. Це можуть бути, наприклад, фоторедактори або прості іграшки.

Single page application (SPA або односторінкове додаток). Більш цікавий варіант, коли використовуються і бекенд і фронтенд. За допомогою їх взаємодії можна створити додаток, який буде працювати зовсім без перезавантажень сторінки в браузері. Або в спрощеному варіанті, коли переходи між розділами викликають перезавантаження, але будь-які дії в розділі обходяться без них.

Прогресивний вебдодаток (PWA) [9] - це вебдодаток, який може бути встановленим на систему користувача. Він працює оффлайн, коли немає підключення до інтернету, по максимуму використовуючи дані, закешовані під час останньої роботи користувача з додатком. Якщо користувач заходить на сайт з Chrome на робочому столі, і у нього включені відповідні прапори, то з’являється прохання встановити додаток. PWA або Progressive Web App - це наступний крок в зручних для користувача додатках, які поєднують в собі зручність і зовнішній вигляд програми.

PWA - це гібрид звичайної веб-сторінки і мобільного додатку. Він поєднує в собі функції більшості сучасних браузерів з перевагами мобільних додатків і створюється з використанням стандартних веб-технологій, включаючи HTML, CSS і JavaScript. Функціональні можливості включають роботу в автономному режимі, push-повідомлення та доступ до апаратури, що створює аналогічний нативним додаткам досвід роботи для користувача.

Прогресивні програми можна назвати адаптивними сайтами, тому що вони підлаштовуються під можливості браузера користувача. Вони автоматично можуть покращувати вбудовані функції браузера, аби робота з сайтом була схожа на роботу з нативним веб-додатком. Базові компоненти PWA:

- маніфест вебдодатку: для надання нативних функцій, таких як іконка програми на робочому столі;
- технологія Service Workers: для фонових завдань і роботи в оффлайн-режимі;
- архітектура application shell (оболонка додатку): для швидкого завантаження з Service Workers.

Найпопулярніші приклади використання PWA - це сайти Alibaba, Forbes, The Weather Channel, MakeMyTrip, Twitter, Instagram, Uber, Pinterest.

1.3. Популярність та необхідність у сучасному світі

Популярність вебдодатків щороку зростає. Компанія DataReportal спільно з We Are Social і Hootsuite випустила підсумковий звіт Digital 2021 [4], який показує в якій мірі цифрові, мобільні і соціальні медіа стали частиною повсякденного життя людей у всьому світі за 2020 рік. У звіт були включені і дані по Україні.

На основі зазначеного звіту наведемо актуальну інформацію про користувачів інтернету, зміни в їхній поведінці в соціальних мережах, їх нових пошукових звички, статистикою e-commerce і вплив пандемії на цифрову сферу.

Основні статистичні дані і тренди по цифровій галузі в світі на січень 2021 року такі [5]:

1. Глобальне населення: за станом на початок 2021 року чисельність населення світу становила 7,83 мільярда осіб. За даними ООН, ця цифра сьогодні зростає на 1% в рік. Це означає, що з початку 2020 року населення світу збільшилася більш ніж на 80 мільйонів осіб.

2. Мобільні пристрої: сьогодні мобільним телефоном користуються 5,22 мільярда осіб - 66,6% світового населення. З січня 2020 роки кількість унікальних мобільних користувачів виросло на 1,8% (93 мільйони), в той час як загальна кількість мобільних підключень збільшилася на 72 мільйони (0,9%) і досягла 8,02 мільярда до початку 2021 року.

3. Інтернет: в січні 2021 року інтернетом користуються 4,66 мільярда людей у всьому світі, що на 316 мільйонів (7,3%) більше, ніж в минулому році. Рівень проникнення інтернету зараз становить 59,5%. Однак COVID-19 значно вплинув на збір даних про кількість користувачів інтернету, тому фактичні цифри можуть бути вищими.

4. Соціальні мережі: зараз в світі налічується 4,20 мільярда користувачів соціальних мереж. За останні 12 місяців ця цифра зросла на 490 мільйонів, що означає зростання більш ніж на 13%. Соціальними мережами в 2021 році користується 53,6% світового населення. В середньому кожен день протягом 2020 року створювали більше 1,3 мільйона нових акаунтів, що становить приблизно 15,5 нових користувачів в секунду.

9 із 10 користувачів Інтернету стверджують, що виходять в Інтернет через смартфон, але дві третини також заявляють, що використовують ноутбук або настільний комп'ютер для доступу до Інтернету. Зараз мобільні телефони є найбільш широко використовуваним Інтернет-пристроєм у всіх країнах, але розрив між мобільними телефонами та комп’ютерами часто є досить незначним.

Мобільні телефони стали пріоритетним пристроєм. Дві третини світового населення користуються мобільними телефонами кожен день.

Таким чином, за 12 місяців 2020 року користувачі Android провели в телефонах більше 3,5 трильйона годин. Як повідомляє GWI, Baby Boomers частіше підключаються до Інтернету через комп'ютери та планшети, ніж через телефони, причому мобільні пристрої становлять менше 50% загального часу в Інтернеті серед користувачів віком від 45 років [5].

Більше 40% веб-сторінок, що обслуговуються в грудні 2020 року, були запитані веб-браузерами, що працюють на ноутбуках та настільних пристроях, хоча загальна частка цих пристроїв дещо зменшилася порівняно з груднем 2019 року.

Однією з найбільш примітних тенденцій 2020 року стало посилення електронної комерції, коли пандемія COVID-19 підштовхнула споживачів у всьому світі до покупок в інтернеті. Майже 77% користувачів інтернету з усього світу у віці від 16 до 64 років здійснюють покупки онлайн кожного місяця.

У січні 2021 року в Україні було 29,47 млн. користувачів Інтернету. Кількість користувачів Інтернету в Україні зросла на 2,0 мільйона (+7,3%) між 2020 і 2021 роками. Проникність Інтернету в Україні становила 67,6% у січні 2021 року [5].

У січні 2021 року в Україні було 25,70 мільйона користувачів соціальних мереж. Кількість користувачів соціальних медіа в Україні зросла на 3,5 мільйона (+16%) між 2020 і 2021 роками. Кількість користувачів соціальних медіа в Україні становила 58,9% від загальної кількості населення в січні 2021 року [5].

У січні 2021 року в Україні було 60,78 млн. мобільних зв'язків. Кількість абонентів мобільного зв'язку в Україні зменшилась на 406 тис. (-0,7%) між січнем 2020 та січнем 2021 р. Кількість мобільних зв'язків в Україні в січні 2021 року становила 139,4% від загальної кількості населення [5]. Зауважимо, що у багатьох людей є більше одного мобільного номера, тому показники мобільних зв'язків можуть перевищувати 100% від загальної кількості населення.

Аналіз наведених статистичних даних показав, що Інтернет та веб-додатки вже стали невід'ємною та важливою частиною нашого повсякдення. Громадяни використовують веб-сайти для розваг, роботи, бізнесу, навчання, замовлення товарів та послуг, оплати тощо.

З урахуванням зазначеного робимо висновок, що вебдодатки набувають особливої актуальності та постійно змінюються. А це означає, що маркетингові стратегії фірми, тип контенту, методи взаємодії людей зі своїми мобільними пристроями, а також особливості здійснення споживачами покупок на вебсайтах та способи пошуку інформації теж постійно змінюються.

1.4. Порівняльний аналіз наявних додатків

Зважаючи на зростання популярності вебдодатків все більша кількість підприємців переносять свої бізнеси у Інтернет. Практично кожна мережа закладів швидкого харчування має свій вебсайт.

Аналіз показав, що існують такі вебдодатки на Android [10, 13].

1. Your Restaurant App

Your Restaurant (Ваш Ресторан) App це шаблон, виконаний із застосуванням Material Design. У додатку передбачені цікаві функції, необхідні в будь-якому додатку для закладу, такі як меню, галерея зображень, новини, місце розташування, соціальні мережі, можливість резервування та push-повідомлення, створені на основі Firebase для взаємодії з користувачем. У ньому також є можливість адміністративної роботи на стороні бекенд для управління додатком.

2. Restaurant Finder. Restaurant Finder - це шаблон додатку під Android, який призначений для пошуку найближчого закладу. У додатку є функція з рейтингів, за відгуками користувачів, для їжі і за цінами, а так само можливість зробити галерею закладів харчування. Користувачі також можуть відстежувати свої улюблені місця для відвідування. Можуть знайти дорогу до ресторану, за допомогою карт, створених із застосуванням Google Maps. Цей додаток має можливість адміністративних налаштувань на основі PHP і можливістю інтеграції з AdMob. Також є можливість зателефонувати в заклад прямо з програми.

Додатки для iOS

1. Restaurant Finder. Цей шаблон додатків під платформу iOS. За допомогою нього користувачі легко зможуть знайти ресторан і зарезервувати столик за дзвінком, по електронній пошті або SMS. **У** шаблоні також є такі функції, як функція пошуку ресторану із запитом за поштовим індексом, назвою ресторану або подається їжі. Для зручності пошуку маршруту є інтеграція з картами, а також можливість створення профілю користувача, і можливість залишати огляди. Також існує можливість фільтрації ресторанів по кухнях. У додатку є адміністративна вебпанель, яка поставляється разом з шаблонами, де адміністратор може додавати нові ресторани, нові види страв, спеціальні пропозиції та багато іншого.

2. Restaurateur iOS. У цьому додатку під iOS є безліч корисних функцій і можливість гнучкого використання як для одного ресторану, так і для декількох. Що робить це додаток унікальним, так це наявність кошика замовлень і можливість оплати, що дозволяє користувачам купувати їжу онлайн на винос або для доставки додому. У цьому додатку теж є функція пошуку найближчих ресторанів, запити, замовлення, профілі користувачів і підтримка служби Apple Push Notification. Цей шаблон додатку також включає в себе панель управління системою. У нього приємний та простий інтерфейс користувача і зручна навігаційна система.

3. Food Delivery System for Restaurant

Цей шаблон підійде для одного ресторану. Його функціонал включає можливість пошуку і замовлення окремих страв, можливість управління замовленням, можливість резервування, нагадування про резервування і інтеграція з Admob. Для шаблону передбачена адміністративна панель на PHP, для налаштування програми під конкретний ресторан. Він сумісний з усіма версіями iOS включаючи 9. А також на CodeCanyon доступна Android версія шаблону Food Delivery System [12].

4. Restaurant IOS Template. Цей шаблон додатків сумісний із засобами розробки під iOS 8 та 9, але за запитом доступне оновлення до версії iOS 10 та

Swift 3. Серед особливостей наявність меню, описи та фотографії для кожного елемента, а також галерея для ресторану, інтеграція в соціальні мережі та контактна інформація. Тут відсутня адміністративна панель для бекенду - настройка здійснюється офлайн та за допомогою конфігураційних фалів XML. У додатку передбачена інтеграція з Google Analytics. Це простий шаблон для iOS, який легко встановити і яким легко користуватися [6].

Крос-платформні шаблони для додатків.

1. Restaurant App Template - React Native [8]. Це шаблон додатків написаний на React Native і який працює на обох платформах Android and IOS. У додатку унікальний та привабливий користувальницький інтерфейс і деякі примітні особливості: підтримка декількох мов, управління замовленнями, push-повідомлення для взаємодії з користувачами, інтеграція PayPal, і адміністративна панель для управління, зроблена за підтримки Firebase.

2. Restaurant Ionic. Це крос-платформенний додаток, написаний на Ionic для одного ресторану. Серед його цікавих особливостей - колірні схеми, які зручні при брендингу, інтеграція бекенд на основі Firebase, меню з категоріями, система пошуку ресторанів з вбудованими картами, наявність спеціальних пропозицій і призначеної для користувача кошика. Для взаємодії з користувачами можна посилати push-повідомлення, а також в програмі є інтеграція з соціальними мережами, так що у користувачів є можливість взаємодіяти з профілем ресторану в соціальних мережах.

3. Crunchy - Restaurant Booking. Цей шаблон додатку, написаний на фреймворкі PhoneGap та Ionic, містить функціональність, яка підійде для будь-якого ресторанного додатку. У цього крос-платформного шаблону є можливість інтеграції з платіжними системами PayPal та Stripe, можливість входу через соціальні мережі Facebook або Google+, а також бекенд підтримка створена на основі фреймворку CodeIgniter PHP.

4. Caetano

Це дуже "прозорий" і легкий для використання шаблон ресторанного додатку, написаний на Ionic 2. Серед особливостей - заставка, меню, кошик покупок та профіль користувача.

У цьому шаблоні немає будь-яких готових варіантів для адміністративного управління, але є можливість організувати цей сервіс, використовуючи платформу на вибір користувача. Шаблон нескладно налаштовувати.

1.5. Стислий огляд проєкту

Даний дипломний проєкт має на меті оптимізацію черг в закладах швидкого харчування. Складається з адміністративною панелі, клієнтської частини та серверу. Використовуючи адміністративну панель, що може бути кастомізована (видозмінена) для кожного закладу індивідуально, можна створювати, редагувати та видаляти замовлення. Для створення замовлення необхідно ввести такі дані: прізвище та ім’я користувача, його номер замовлення та орієнтовний час очікування. Час очікування можна вводити довільний або скористатися наявними кнопками введення часу (5, 10,15 чи 20 хвилин).

Після створення замовлення, що являє собою збереження вище вказаних даних в нереляційній (NoSQL) базі даних, відбувається відображення інформації про замовлення як на адміністративній панелі так і на клієнтській частини. Слід зауважити, що відображення залишкового часу до приготування замовлення на клієнтській частини, відбувається в реальному часі й змінюється що хвилини, шляхом ререндингу відповідних елементів. Після приготування замовлення використовуючи адмін-панель можна змінити статус замовлення на «Готовий», що слугуватиме тригером для відправлення Push Notification сервером клієнту. Після того, як клієнт забрав своє замовлення, на адміністративній панелі наявна функція повного видалення інформації про замовлення з бази даних. Відбудеться ререндеринг наявної інформації про всі замовлення, що відобажаються як на клієнтській частині так і на адмін-панелі, в реальному часі.

Дана функція можлива за рахунок використання графової мови запитів з відкритим кодом – GraphQL [16] та механізму, що надає GrapQL – підписки (subscriptions).

Використання даного проєкту не обмежується тільки одним закладом, мінімальними змінами в коді, можна легко перенести його функціонал на будь-які інші мережі закладів.

Цей застосунок відповідає таким вимогам:

- Адаптивна верстка
- Коректна робота на популярних браузерах, а саме Google Chrome, Opera та Mozilla FireFox

Технології, що були використанні в процесі розробки:

- JavaScript
- CSS
- HTML
- React.js
- Mongoose.js
- Node.js
- MongoDB
- Apollo
- GraphQL

Сайт розміщенно на хостинговій платформі VPS.ua.

Висновки до розділу 1

В даному розділі було здійснено огляд існуючих вебдодатків для закладів швидкого харчування. Зокрема, розглянуто сутність та структуру таких вебдодатків, досліджено принципи їх роботи. Також були досліджені причини зростання популярності таких вебдодатків.

Встановлено, що мобільний вебдодаток для закладів швидкого харчування дозволяє підприємцю оптимізувати роботу персоналу, знизити витрати на рекламу, збільшити середній чек і налагодити комунікацію з клієнтом. Зокрема, можна інформувати клієнтів про акції та пропонувати їм дисконтні купони, а також надати їм зручний сервіс для замовлення їжі. Відвідувач може зробити замовлення їжі, отримати рахунок та оплатити його прямо з мобільного пристрою. Це дозволяє збільшити число лояльних клієнтів і як наслідок, підвищити прибуток закладу.

Ефективність використання вебдодатків пояснюється: економією часу відвідувача на здійснення замовлення, розрахунок і оплату; оптимізацією роботи по обслуговуванню клієнтів; зростанням лояльності відвідувачів до закладу.

Вебдодаток, що пропонується в дипломному проекті, має на меті оптимізацію черг в закладах швидкого харчування. Складається з адміністративною панелі, клієнтської частини та серверу. Використовуючи адміністративну панель, що може бути кастомізована (видозмінена) для кожного закладу індивідуально, можна створювати, редагувати та видаляти замовлення. Його використання не обмежується тільки одним закладом, за допомогою мінімальних змінами в коді, можна легко перенести його функціонал на будь-які інші мережі закладів, що надає йому додаткові конкурентні переваги.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Архітектура вебдодатку

Вебдодаток побудований за технологією PWA, що візуально й функціонально трансформує сайт додатку в мобільний застосунок браузера. PWA є гібридним рішення, що дозволяє відкрити застосунок за допомогою браузера при повному збереженні нативного застосунку, а саме:

1. відправка push-notification (крім пристроїв під керування IOS);
2. робота в режимі офлайн;
3. доступ до апаратного забезпечення;
4. встановлення ярлика на робочий стіл мобільного пристрою, що візуально не відрізняється від ярлика нативного застосунку.

Для роботи з PWA необхідно, аби сайт відповідав таким вимогам:

1. Наявність Service Worker.
2. Наявність Web App Manifest.
3. SLL-сертифікат для підтримки HTTPS.

Service Worker – це JavaScript-файл, що запускається у фоновому режимі як автономний сервіс, він не пов’язаний з DOM (Document Object Model) чи вебсторінками, працює на іншому потоці і отримує доступ до DOM за допомогою API postMessage [14].

Web App Manifest надає інформацію щодо програми у текстовому JSON-файлі. Необхідно, аби вебдодаток було завантажено і візуально він відображався для користувача аналогічно до нативного додатка.

Для взаємодії з сервером використовується GraphQL, що дозволяє клієнтам задавати структуру необхідних даних від сервера.

Адміністративна панель представляє собою добре відому SPA single-page application) - вебдодаток, що взаємодіє з користувачем шляхом динамічних змін поточної вебсторінки [3].

Сервер, back-end, реалізовано за допомогою мови програмування NodeJS, використовуючи NoSQL базу даних MongoDB. Архітектура сервера реалізована з дотриманням DDD (domain drive design), що забезпечує наявність чітко відокремленої бізнес-логіки [11]. Для взаємодії з базою даних використовується ODM Mongoose, що дозволяє співставляти об'єкти класів і документи колекцій з бази даних.

2.2. Клієнт

Для побудови клієнтської частини вебзастосунка було використано найсучасніші технології, а саме:

- Typescript
- React.js
- Graphql
- HTML
- CSS

З розвитком мови програмування JavaScript можна помітити тенденцію зниження частоти використання «чистого» JavaScript (vanilla JavaScript). Цьому сприяє розвиток таких взаємозамінних бібліотек як React, Angular та Vue [16].

Щоб додавати, видаляти та оновлювати пакети, необхідно використовувати пакетний менеджер. В даній роботі було використано пакетний менеджер від Facebook – yarn. Слід зауважити, що для встановлення yarn необхідна наявність іншого пакетного менеджера npm. Yarn був створений з метою заміни собою пакетного менеджера npm за рахунок пришвидшення роботи, але після виходу npm версії 5, в якій було пришвидшено роботу, переваги yarn майже нівельовані.

Маючи на меті зафіксувати всі версії пакетів, що були використанні на даному проєкті, слід ініціювати створення файлу, що буде зберігати всі версії пакетів.

Для можливості використання статичної типізації на клієнті, використовуючи бібліотеку React на клієнті, наявні такі інструменти:

- Flow
- TypeScript

Інструменти для статичної типізації дозволяють знаходити більшу частину помилок ще до виконання коду. Також вони суттєво поліпшують процеси розробки, додаючи автодоповнення та інші можливості.

Flow - це бібліотека для статичної типізації JavaScript, розроблена в Facebook і часто застосовується в зв'язці з React. Flow розширює можливості JavaScript, додаючи анотації типів для змінних, функцій і React-компонентів [6].

TypeScript - це мова програмування, розроблена в Microsoft. TypeScript є надмножиною JavaScript, має статичну систему типів і власний компілятор. Статична типізація дозволяє відловлювати помилки і баги під час компіляції, ще до запуску програми.

Щоб використовувати TypeScript, потрібно:

- додати TypeScript в проект як залежність.
- налаштувати компілятор.
- використовувати правильні розширення файлів.
- встановити файли оголошень для використовуваних бібліотек.

Під час обміну даними між клієнтом та сервером задля отримання можливості роботи з оновленнями в реальному часі (subscriptions) і наявності єдиного end-point на проєкті, як і на бекенд частині так і на клієнті, використовується GraphQL.

Для полегшення верстки сайту, створення mobile-first application та пришвидшення розроблення сайту було використано бібліотеку React-Bootstrap.

Також з метою створення PWA було реалізований наступний функціонал:

- Створення Service Worker
- Додавання SSL сертифікату для HTTPS

- Application Shell
- Web App manifest.

Серце PWA - Service Worker. Це проксуючий шар між Front-end і Back-end, що знаходиться в браузері. Всі запити браузера йдуть через нього.

Цей поділ на два незалежних шари дозволяє зробити перехід звичайного вебсайту в PWA максимально простим. Зі сховищ у Service Worker'а є доступ до Cache Storage для вебресурсів і IndexDB для даних [9]. Але, найголовніше, повна свобода для реалізації бізнес-логіки.

З програмної точки зору Service Worker є javascript файл, що підключається в html коді сторінки.

App shell – це скелет графічного інтерфейсу, шаблон. Суть в тому, що app shell зберігається на клієнті і завантажується при запуску програми, а потім вже в нього вантажиться з мережі динамічна інформація.

Web App manifest – це JSON файл, декларативно визначає для браузера назву програми, іконку, як буде виглядати PWA (fullscreen, standalone і ін.) та деякі інші параметри. Дозволяє встановити PWA як окремий додаток на домашній екран смартфона.

2.3. Admin-клієнт

Адміністративна частина вебзастосунка представляє собою SPA (Single Page App), що його створено з використанням бібліотеки для мови програмування JavaScript – React.js. Для комунікації з веб-сервером використано GraphQL.

Особливість бібліотеки React.js полягає в такому [11]:

- наявність віртуального DOM;
- використання pure function для написання компонентів;
- однонаправлена прив’язка даних завдяки Flux архітектурі;
- іммутабельність стану компонента.

Бібліотека React змогла серйозно знизити гостроту проблеми неконтрольованих мутацій завдяки використанню архітектури Flux. Замість

того, аби приєднувати до довільної кількості довільних об'єктів (моделей) обробник подій, що викликають поновлення DOM, бібліотека React дала розробникам єдиний спосіб управління станом компонента. Це - диспетчеризація дій, що впливають на сховище даних. Коли змінюється стан компонента, система пропонує компоненту повторної візуалізації.

Використання розширення jsx, що дозволяє декларативно створювати компоненти користувацького інтерфейсу, має такі можливості:

- застосування декларативної розмітки;
- код розмітки розташований в одному місці з кодом компонента;
- розподілення відповідальності;
- абстраговане керування DOM.

Певні особливості, які потрібно враховувати під час роботи з файлами тупу jsx [12, 15]:

- відмінність підходу іменування атрибутів елементів від того, що застосовується в HTML;
- наявність унікального атрибута в кожного елементу списку, що потребує рендерингу.

Також ще однією особливістю бібліотеки React.js є життєвий цикл її компонентів.

Концепція життєвого циклу React-компонентів орієнтована на захист стану компонента. Стан компонента не повинен змінюватися в процесі його виведення на екран. Це досягається завдяки наступній схемі роботи: компонент набуває певного стану і візуалізується. Потім, завдяки подіям життєвого циклу, наявна можливість застосування до нього ефектів, що здані впливати на його стан [13].

В життєвому циклі React-компонентів можна виділити 3 ключові стани.

1. Монтування (mount)
2. Оновлення (update)
3. Unmount

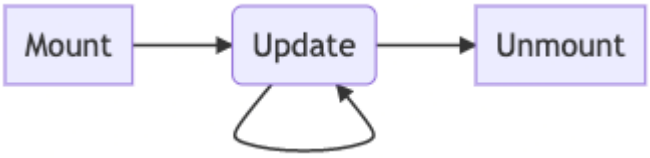


Рис. 2.1. Ключові стани життєвого циклу React-компонентів

Етап оновлення можна розділити на 3 частини: рендеринг, підготовка до внесення змін у DOM дерево, внесення змін у DOM дерево.

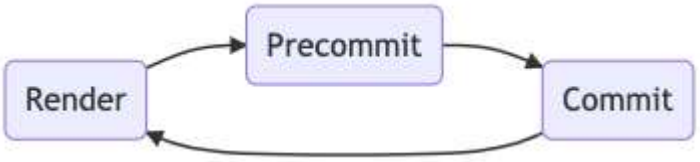


Рис. 2.2. Складові етапу оновлення

Деталізуємо етапи життєвого циклу.

Render – на цьому етапі життєвого циклу компонента проводиться його рендеринг. Метод компонента render () повинен представляти собою детерміновану функцію, яка не має побічних ефектів. Цю функцію варто розглядати як чисту функцію, яка одержує дані з вхідних параметрів компонента і повертає JSX.

Precommit – на цьому етапі можна прочитати дані з DOM, користуючись методом життєвого циклу компонента getSnapShotBeforeUpdate.

Commit – на цій фазі життєвого циклу компонента React оновлює DOM.

2.4. Сервер

Для розроблення серверної частини було обрано мультипарадигмену мову програмування JavaScript, з середовищем виконання NodeJS. NodeJS написана на C/C++ і представляє собою середовище виконання JavaScript використовуючи V8 для трансляції JavaScript в машинний код. В основі NodeJS лежить подіє-орієнтоване та асинхронне програмування з неблокуючим введенням/виведенням.

Основні переваги NodeJS

1. Асинхронність

Оскільки NodeJS надає змогу не блокувати систему input/output, це

допомагає обробляти декілька запитів одночасно значно ефективніше, ніж Ruby чи Python.

Розглянемо основні способи роботи з асинхронністю в NodeJS.

Функції колбеки. Коллбек - це звичайна функція, яка передається, як значення, інший функції. Викликана вона буде тільки в тому випадку, коли відбудеться якась подія. В JavaScript реалізована концепція функцій першого класу. Такі функції можна призначати змінним і передавати іншим функціям (так званим функціями вищого порядку) [18].

Обробка помилок під час роботи з колбеками. Існує одна поширена стратегія обробки подібних помилок, яка застосовується і в Node.js. Вона полягає в тому, що першим параметром будь-якої функції зворотного виклику роблять об'єкт помилки. При відсутності помилок в цей параметр буде записано значення null. В іншому випадку тут буде об'єкт помилки, що містить її опис і додаткові відомості про неї.

Коллбек-функціями зручно користуватися в простих ситуаціях. Однак, кожен коллбек - це додатковий рівень вкладеності коду. Якщо використовується кілька вкладених коллбек-функцій, це швидко призводить до значного ускладнення структури коду.

Promise та async/await.

Проміс (promise-об'єкти) - це один із способів роботи з асинхронними програмними конструкціями в JavaScript, який, в цілому, дозволяє скоротити використання коллбеків [12].

Після виклику Promise він переходить в стан очікування (pending). Це означає, що функція, яка викликала Promise, залишиться активним, при цьому в Promise виконуються якісь обчислення, по завершенні яких Promise повідомляє про це. Якщо операція, яку виконує Promise, завершується успішно, то Promise переводиться у стан «виконано» (fulfilled). Про таий Promise кажуть, що він успішно виконаний. Якщо операція завершується з помилкою, Promise переводиться у стан «відхилено» (rejected).

Async/await

Конструкція async / await є сучасним підходом до асинхронного програмування, спрощуючи його. Асинхронні функції можна представити у вигляді комбінації Promise і генераторів, і, в цілому, ця конструкція представляє собою абстракцію над Promise.

Конструкція async / await дозволяє зменшити обсяг шаблонного коду, що його доводиться писати при роботі з Promise. Коли Promise з'явилися в стандарті ES2015, то вони були спрямовані на вирішення проблеми створення асинхронного коду. З цим завданням вони впоралися, але за два роки, що розділяють вихід стандартів ES2015 і ES2017, стало зрозуміло, що вважати їх остаточним вирішенням проблеми не можна.

Однією з проблем, яку вирішували Promise, було знамените «пекло коллбек-функцій», але вони, вирішуючи цю проблему, створили власні проблеми схожого характеру.

Проміс представляли собою прості конструкції, навколо яких можна було б побудувати щось, що володіє більш простим синтаксисом. В результаті, коли прийшов час, з'явилася конструкція async / await. Її використання дозволяє писати код, який виглядає як синхронний, але при цьому є асинхронним, зокрема, не блокує головний потік.

2. Наявність багатої екосистеми пакетів.

Використовуючи NPM (Node Package Manager), можна значно пришвидшити розробку застосунка за допомогою використання сторонніх пакетів з відкритим кодом [10].

Під час виконання команди npm init, де відбувається заповнення базової інформації про проєкт, також створюється файл package.json, який містить секцію, в якій зберігаються версії всіх пакетів, що їх було використано під час розробки.

3. Легке масштабування застосунку.

Розглянемо основні способи масштабування.

Найпростіше зробити масштабування великого додатка – це клонувати його кілька разів, і кожен клонований екземпляр оброблятиме частину робочого

навантаження (наприклад, з балансувальником навантаження). Це не потребує багато часу з точки зору розробки, і це дуже ефективно. Ця стратегія є базовою під час масштабування, і Node.js має вбудований модуль, кластер, щоб полегшити реалізацію стратегії клонування на одному сервері [8].

Декомпозиція додатка.

Можна масштабувати додаток, розкладаючи його на основі функціональних можливостей та послуг. Це означає наявність декількох різних додатків з різними базами коду, а іноді і з власними виділеними базами даних та інтерфейсами користувача.

Ця стратегія зазвичай асоціюється з терміном мікросервіс, де мікро вказує на те, що ці послуги повинні бути якомога меншими, та насправді розмір послуги не є важливим, а скоріше є забезпеченням вільного зв'язку та високої згуртованості між службами. Впровадження цієї стратегії часто буває непростим і може призвести до довгострокових несподіваних проблем, але при правильному виконанні переваги великі.

Недолік NodeJS: падіння продуктивності при високому навантаженні на CPU. Під час виконання задачі, що потребує значних ресурсів CPU, можливе сповільнення роботи всього застосунка [3].

Для взаємодії з GraphQL було використано бібліотеку Apollo-server, що значно полегшує та пришвидшує розробку та допомагає запускати застосунок як на сервері так і в lambda функціях від AWS. Також слід зазначити легкість логування системи (в даному застосунку всі логи попадають в output), відмовостійкість та підтримку механізму взаємодії з клієнтом в реальному часі, використовуючи механізм subscriptions. Для забезпечення строгої типізації було використано TypeScript. Як пакетний менеджер на проекті використовується yarn.

Yarn - це альтернативний npm-клієнт для роботи в якості пакетного менеджера JavaScript, спільно створений Facebook, Google, Exponent і Tilde. Цей менеджер пакетів прискорює збір пакетів і робить його безпечнішим.

NPM у файлі package.json фіксує не конкретну версію використовуваних пакетів, а їх діапазон. Таким чином, стратегія стандартного клієнта npm може

сприяти тому, що на основі ідентичних файлів package.json у різний час будуть встановлені різні версії пакетів. Yarn вирішує цю проблему, так як дозволяє точно зафіксувати залежність у файлі yarn.lock

Послідовна установка пакетів через npm сповільнює роботу. Yarn підтримує паралельну установку, яка зазвичай у декілька разів швидше.

Ключовою особливістю GraphQL є опис схеми, що включає в собі три основні типи [15]:

1. Mutations. Використовуються для створення, оновлення чи видалення сутностей.
2. Query - використовуються для отримання інформації з серверу.
3. Subscriptions - для взаємодії в реальному часі.

Описуючи GraphQL схему зникає необхідність написання API документації для розробників клієнтської частини застосунку, оскільки вся необхідна інформація (структура даних, що надсилаються та її тип) наявна в даній схемі

Для взаємодії з NoSQL базою даних – MongoDB, використовується ODM Mongoose.

Архітектура серверної частини побудована за принципами DDD (domain driven design).

Основні концепції: Context, Domain, Model, Ubiquitous Language.

2.5. База даних

Для зберігання даних про замовлення, його статус, інформації про власника замовлення та перелік всіх адміністраторів, що можуть змінювати стан замовлення, необхідно зберігати дані в певній базі даних. Існують два типи баз даних: SQL (реляційні) та NoSql (нереляційні) бази даних. Кожна з них має свої переваги та недоліки. Розглянемо ключові особливості як SQL та NoSql баз даних.

Особливості SQL баз даних [10, 14-16, 20]:

- Структура даних, що зберігається

Наявна таблиця з чіткою структурою, що містить в собі рядки та стовбці. В

кожному рядку міститься певна інформація про конкретну одиницю. Наприклад, є таблиця Users, кожен рядок це інформація про користувача. Кожен стовбець таблиці - це інформація про певний параметр певною одиниці.

- Робота з даними, що зберігаються.

Певні таблиці в базі даних можуть мати відносини з іншими таблицями (one-to-many, many-to-many). Завдяки цим відносинам та механізму join можна будувати складні вибірки для відображення даних зі сховища.

- Маштабування

Найрозповсюдженіший спосіб масштабування SQL баз даних - вертикальне масштабування, шляхом збільшення потужностей сервера, де розташована база даних.

Особливості NoSQL баз даних:

- Структура даних що зберігається

Якщо в SQL використовуються таблиці, то в NoSQL використовуються колекція, що можуть різнитися між собою. Замість рядків використовуються документи, що можуть мати складну структура даних, де значенням ключа може виступати як об’єкт так і масив.

- Робота з даними, що зберігаються.

Відсутність складних join через спосіб зберігання даних.

- Масштабування. Можна використовувати як горизонтальне так і вертикальне масштабування.

Для розроблення даного веб-додатку було використано NoSQL базу даних MongoDB.

Система підтримує ad-hoc-запити: вони можуть повертати конкретні поля документів і призначені для користувача JavaScript-функції. Підтримується пошук за регулярними виразами. Також можна налаштувати запит на повернення випадкового набору результатів [4].

Є підтримка індексів.

Система може працювати з набором реплік [7], тобто містити дві або більше копії даних на різних вузлах. Кожен екземпляр набору реплік може в будь-який

момент виступати в ролі основної або допоміжної репліки. Всі операції запису і читання за замовчуванням здійснюються з основною реплікою. Допоміжні репліки підтримують в актуальному стані копії даних. У разі, коли основна репліка дає збій, набір реплік проводить вибір, яка з реплік повинна стати основною. Другорядні репліки можуть додатково бути джерелом для операцій читання.

Система масштабується горизонтально, використовуючи техніку сегментування (англ. Sharding) об'єктів баз даних - розподіл їх частин з різних вузлів кластера. Адміністратор вибирає ключ сегментування, який визначає за яким критерієм дані будуть рознесені по вузлах (в залежності від значень хеша ключа сегментування). Завдяки тому, що кожен вузол кластера може приймати запити, забезпечується балансування навантаження.

Система може бути використана в якості файлового сховища з балансуванням навантаження і реплікацією даних (функція Grid File System [4]; поставляється разом з драйверами MongoDB). Надаються програмні засоби для роботи з файлами і їх вмістом. GridFS використовується в плагінах для Nginx і lighttpd. GridFS розділяє файл на частини і зберігає кожну частину як окремий документ [4].

Може працювати відповідно до парадигми MapReduce. У фреймворку для агрегації є аналог SQL-вирази GROUP BY. Оператори агрегації можуть бути пов'язані в конвеєр подібно UNIX-Конвейр. Фреймворк так само має оператор $ lookup для зв'язки документів при вивантаженні і статистичні операції – такі, як середньоквадратичне відхилення.

Підтримується JavaScript в запитах, функціях агрегації (наприклад, у MapReduce).

Висновки до розділу 2

В даному розділі було створено архітектуру веб-додатку на прикладі клієнтської частини, адмін-частини та серверу, створено API застосунку. Розглянуто такі підходи до створення вебдодатків, як SPA (Single page application) та PWA (progressive web application), досліджено спосіб комунікації між сервером та клієнтом за допомогою GraphQL. Наведено огляд SQL та NoSQL баз даних. Наведено обґрунтування вибору потрібних технологій задля задоволення бізнес-потреб.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Клієнт

Для створення темплейту для роботи з React.js було використано команду Create React App, що дозволяє швидко створити весь необхідний шаблонний код для подальшої роботи над застосунком [10]. Для створення PWA необхідно було написати два ключові файли - manifest.json та serviceWorker.js.

У файлі manifest.json зосереджено певні елементи аплікації, такі як назва, шляхи до іконок, початкове посилання та формат відображення.

Скорочений вигляд файлу наведено нижче

```
{
  "short_name": "eateasy",
  "name": "eateasy",
  "icons": [
    {
    "src": "\/icons\/android-icon-192x192.png",
    "sizes": "192x192",
    "type": "image\/png",
    "density": "4.0"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Для підключення файлу serviceWorker.js доброї практикою вважається перевірка наявності ‘serviceWorker’ в об’єкті navigator та в разі наявності під час завантаження сторінки зареєструвати файл, що буде містити всю необхідну логіку для коректної роботи service worker.

Необхідно в кореневому файлі index.html підключити маніфест наступним чином

```
<link rel="manifest" href="/manifest.json">
```

та описти скрипт реєстраації serviceWorker

```
<script>
 if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
   navigator.serviceWorker.register('/sw.js').then(
    function(registration) {
     // Registration was successful
     console.log('ServiceWorker registration successful with scope: ',
registration.scope); },
    function(err) {
     // registration failed :(
     console.log('ServiceWorker registration failed: ', err);
    });
  });
 }
</script>
```

Також для коректної роботи всього клієнтського застосунка необхідно передбачити можливість вимкненого JavaScript на стороні кінцевого користувача наступним кодом

```
<noscript>You need to enable JavaScript to run this app.</noscript>
```

Для коректного функціонування клієнтської частини веб-додатку були використані такі основні пакети:

Bootstrap - для поліпшення візуальної частини сайту та полегшення роботи з

css.

Graphql - для можливості взаємодії з бекенд частиною використовуючи GraphQL.

У клієнтській частині веб-додатку для отримання інформації з серверної частини було використано пакет node-fetch.

Нижче наведено приклад коду для отримання списку всіх замовлень.

```
import fetch from 'node-fetch';
export const fetchGetOrders = async (companyName:string) => {
  const graphql = JSON.stringify({
    query: `
    query MyQuery($variable: ListOrdersInput!) {
       listOrders(input: $variable) {
       orders {
         _id
         orderNumber
         cookingTime
         firstName
         lastName
         status
         company
         createdAt
         updatedAt
    }
    metadata {
         skip
         limit
         count
         totalCount
    }
    }
```

```
}`,
    variables: { variable: { limit: 30, find: { company: companyName } } }
  })
  const res = await fetch(process.env.REACT_APP_URL as string, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: graphql,
  })
  const ret = await res.json();
  return ret.data?.listOrders?.orders;
}
```

3.2. Сервер

Для створення серверу було використано Node.js та open-source фреймворк Apollo Server для роботи з GraphQL.

Основні переваги, що надає Apollo Server [9]:

1. Швидкий та легкий початок проекту
2. Легке розширення наявного функціоналу
3. Універсальну сумісність

Для початку необхідно зробити опис GraphQL схеми у файлі з розширенням .graphql, де слід описати схему з трьох основних компонентів, а саме: query, mutation та subscription.

Далі слід описати конфігурації для Apollo, що являє собою об’єкт з наступними полями: typeDefs, resolvers, cors, methods, context, formatError, subscriptions.

- typeDefs - це є вміст файлу розширення .graphql опрацьований функцією gql, що береться з пакету apollo-server.

  Оскільки застосунок не може почати свою роботу без вмісту цього файлу, його зчитування відбувається синхронно.

const typeDefs = gql(fs. readFileSync(__dirname.concat('/schema.graphql'), 'utf8'));

- resolvers - тут зосереджена логіка обробки кожного запиту на сервер.

const resolvers = {

...Mutation,

...Query,

...Subscription,

} as IResolvers;

- cors - налаштування політики CORS для серверної частини
- methods - перелік всіх доступних HTTP методів запиту на сервер. Оскільки специфіка GraphQL дозволяє працювати лише з запитами POST/GET, то значення поля methods є масивом з двома елементами типу String.

methods: [

'POST',

'GET',

],

- context - метод, що відповідає за логіку обробки наявного контексту, зазвичай використовується для встановлення з'єднання з Базою Даних.
- formatError - метод, що відповідає за опрацювання помилок, що виникають під час роботи серверу. Використовуючи змінні середовища можна налаштовувати різну логіку обробки помилок в development, test та production режимах.
- subscriptions - об'єкт, що відповідає за встановлення з'єднання для подій в реальному часу, має два поля onConnect та onDisconnect.

Для коректної роботи серверної частини було використано наступні пакети:

- bcrypt - для хешування паролів користувачів (адмінів закладу) та порівняння хешу під час авторизації.
- typescript - для можливості використання всіх переваг TypeScript, строгої типізації зокрема.

- apollo-server - використовується для створення серверу та для роботи з GraphQL.
- dotenv - пакет для роботи зі змінними середовища. В змінних середовищах зазвичай передається дані для підключення до бази даних (логін/пароль), режим роботи (development, test чи production).

Основні Mutations, що використовуються для роботи:

```
type Mutation {
  createOrder(input: CreateOrderInput!): OrderType!
  updateOrder(input: UpdateOrderInput!): OrderType!
  deleteOrder(id: ID!): Boolean
  login(input: LoginInput!): LoginDataType!
}
```

Основі Query, що використовуються для роботи:

```
type Query {
  listOrders(input: ListOrdersInput!): OrderListType!
}
```

Слід звернути увагу на змінні типу input. Це дані, що розраховує отримати сервер під час взаємодії з Mutatios чи Query.

Для взаємодії з базою даних використовується ОДМ Mongoose. З'єднання встановлюється за допомогою змінної типу String, в якій зі змінних середовища підставляються дані для входу, а саме назву бази даних, логін та пароль.

```
const connectionUrl: string =
`mongodb+srv://${dbUser}:${dbPassword}@firstcluster.uq8y3.mongodb.net/${dbName}?retryWrites=true&w=majority`;
```

3.3. API серверу

Оскільки для взаємодії з сервером використовується GraphQL, то все API зосереджено в одному файлі schema.graphql.

```
type Mutation {
```

createOrder(input: CreateOrderInput!): OrderType!

updateOrder(input: UpdateOrderInput!): OrderType!

deleteOrder(id: ID!): Boolean

login(input: LoginInput!): LoginDataType!

}

createOrder - мутація, що використовується для створення замовлення, адміністратором закладу. На вхід очікується набір даних, що задовольняє наступні структуру даних

input CreateOrderInput {

orderNumber: String!

cookingTime: String!

firstName: String!

lastName: String!

}

updateOrder - мутація, що використовується для оновлення статусу замовлення. Від клієнтської частини застосунку очікується наступний набір даних:

input UpdateOrderInput {

id: ID!

status: OrderStatus!

}

deleteOrder - мутація, що використовується для видалення замовлення після його опрацювання використовуючи унікальний ідентифікатор замовлення [10].

login - мутація, яка отримує логін та пароль адміністратора закладу, для отримання доступу до ресурсу [12].

Тип Subscriptions використовується для можливості оновлення всіх даних в реальному часі.

Повний перелік Graphql.schema

schema {

query: Query

```
  mutation: Mutation
  subscription: Subscription
}
type Mutation {
  ### ORDER
  createOrder(input: CreateOrderInput!): OrderType!
  updateOrder(input: UpdateOrderInput!): OrderType!
  deleteOrder(id: ID!): Boolean
  login(input: LoginInput!): LoginDataType!
}
type Query {
  hello: String!
  listOrders(input: ListOrdersInput!): OrderListType!
}
type Subscription {
  createOrder: OrderType!
  companyOrders(input: CompanyOrdersInput!): OrderListType!
  updateOrder: OrderType!
}
##### ENUMS
enum OrderStatus {
  COOKING
  COOKED
  ACCEPT
}
##### TYPES
type TokenTypes {
  accessToken: String!
  refreshToken: String!
}
```

```
type LoginDataType {
  tokens: TokenTypes!
  profile: UserProfileType!
}
type UserProfileType {
  _id: ID!
  email: String!
  firstName: String!
  lastName: String!
  company: String!
  createdAt: String!
  updatedAt: String!
}
type OrderType {
  _id: ID!
  orderNumber: String!
  cookingTime: String!
  status: OrderStatus!
  company: String!
  createdAt: String!
  firstName: String!
  lastName: String!
  updatedAt: String!
}
type MetaDataListType {
  limit: Int
  skip: Int
  count: Int
  totalCount: Int
}
```

```
type OrderListType {
  orders: [OrderType]
  metadata: MetaDataListType
}
##### INPUT
input CreateOrderInput {
  orderNumber: String!
  cookingTime: String!
  firstName: String!
  lastName: String!
  # company: String!
}
input LoginInput {
  password: String!
  login: String!
}
input ListOrdersInput {
  skip: Int
  limit: Int
  find: findListOrdersInputProperty
}
input findListOrdersInputProperty {
  company: String!
}
input CompanyOrdersInput {
  company: String!
  skip: Int
  limit: Int
}
input UpdateOrderInput {
```

```
  id: ID!
  status: OrderStatus!
}
```

3.4. База даних

Для зменшення навантаження на сервер, що містить back-end частину застосунку та для збільшення відмовостійкості база даних була розміщення на окремому сервері - https://cloud.mongodb.com .

Підключення до бази даних описані на серверній частині вебдодатку.

Було створено базу даних, що містить дві колекції:

1. Users - що містить інформацію про всіх адміністраторів закладів.
2. Orders - що містить інформацію про всі наявні замовлення для певного закладу.

Кожна колекція містить відповідні документи. Розглянемо детальніше структуру кожного.

Users:

1. _id - автоматично створюваний ідентифікатор для документа. Використовується MongoID.
2. email - електронна адреса адміністратора
3. password - хешований пароль користувача
4. firstName - ім’я адміністратора
5. lastName - прізвище адміністратора закладу
6. company - назва компанії, де працює адміністратор.
7. createdAt - дату створення адміністратора
8. updatedAt - дату оновлення інформації про адміністратора.

Orders:

1. _id - автоматично створюваний ідентифікатор для документа. Використовується MongoID.
2. orderNumber - номер замовлення

3. firstName - ім’я власника замовлення
4. lastName - прізвище власника замовлення
5. cookingTime - очікуваний час приготування замовлення
6. status - статус замовлення
7. company - назва закладу, в якому було зроблено замовлення.
8. createdAt - дату створення замовлення
9. updatedAt - дату оновлення інформації про замовлення.

3.5. Admin-клієнт

Для встановлення залежностей на адмін панелі використовується пакетний менеджер yarn. Адміні панель створена як Single Page Aplication, за допомогою команди npx create-react-app. Комунікація з серверною частиною відбувається за допомогою GraphQL.

Щоб отримати можливість керувати замовленнями, а саме створювати, редагувати та видаляти, адміністратору необхідно пройти авторизацію. Нижче наведено приклад коду, що за допомогою React-hooks дозволяє отримати http-only token для можливості комунікації з сервером.

```
const [login] = useMutation(LOGIN_MUTATION, {
  variables: {
    "variable" : {
      "login": formState.email,
      "password": formState.password
    }
  },
  onCompleted: ({ login }) => {
    history.push({
      pathname: '/about',
      state: login
    });
```

```
  },
  onError: err => {
    console.log(err.graphQLErrors.length)
    if (err.graphQLErrors.length > 0) {
      setErrors((prevState) => {
        return {
          ...prevState,
          email: 'WRONG_EMAIL',
          password: 'WRONG_PASSWORD'
        }
      });
    }
  }
});
```

React-hooks є альтернативою класовим компонентам, що дозволяє зменшити зусилля на написання коду та поліпшити його якість, було прийнято рішення використовувати саме їх під час розроблення адміністративної панелі [19].

Пакети, що використовуються:

1. apollo/client - для роботи з graphQL
2. apollo/react-hooks - для роботи з React-hooks.
3. apollo-link-ws - для встановлення з’єднання з сервером через websocket

Висновки до розділу 3

В даному розділі був наведений опис імплементації адмін-частини, клієнтської частини та серверної частини, використовуючи NodeJS та React.js. Також була наведена структура NoSQL бази даних. Продемонстровано базові вимоги до створення PWA. Створено механізм взаємодії в реальному часі за допомогою GraphQL механізму Subscriptions. Наведений список основних пакетів, що були використані для розробки та їх короткий опис.

РОЗДІЛ 4
ІНСТРУКЦІЯ КОРИСТУВАЧІВ

4.1. Інструкція користувача

Використання веб-додатку починається з того, що відвідувач закладу сканує QR-code в якому міститься посилання на клієнтську частину додатку. Після внесення користувацьких даних адміністратором закладу гість може бачити перелік всіх замовлень, час приготування та статус замовлення. Перейшовши за посилання користувач може побачити наступні частини веб-додатку.

1. Шапку додатку, яка містить логотип закладу, адресу та пошуковий рядок (рис. 4.1).

Рис. 4.1. Шапка додатку

2. Перелік всіх замовлень та інформація про них (рис. 4.2).

Рис. 4.2. Перелік всіх замовлень та інформації щодо них

3. Кнопку, що дозволяє переглянути всі замовлення в даному закладі (рис. 4.3).

Рис. 4.3. Вигляд кнопки що дозволяє переглянути всі замовлення в даному закладі.

Після кліку на кнопку “Your Order” користувач зможе переглянути інформацію з приводу всіх актуальних замовлень в даному закладі (рис. 4.4).

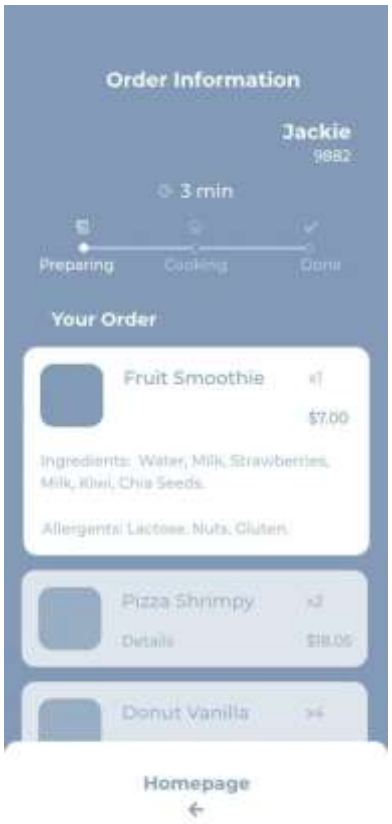


Рис. 4.4. Опція можливості перегляду всіх актуальних замовлень у закладі

На даному екрані містить більш детальна інформація з приводу замовлень, а саме їх кількість, ціна, назву, склад, статус замовлення та орієнтовний час приготування

4.2. Інструкція адміністратора

Після того, як адміністратор закладу перейшов за посиланням для авторизації, йому необхідно ввести свої дані, електронну адресу та пароль.

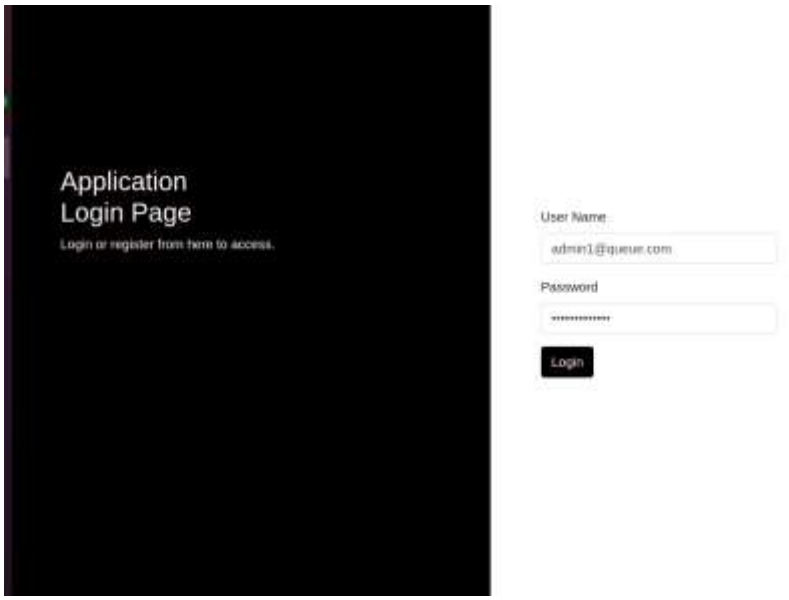


Рис. 4.5. Сторінка входу

Після введення валідних даних адміністратор переходить на сторінку керування замовленнями, що складається з шапки сайту, блоку, що дозволяє створювати замовлення та переліком всіх замовлень.

Рис. 4.6. Шапка сайту

Рис. 4.7 Блок створення

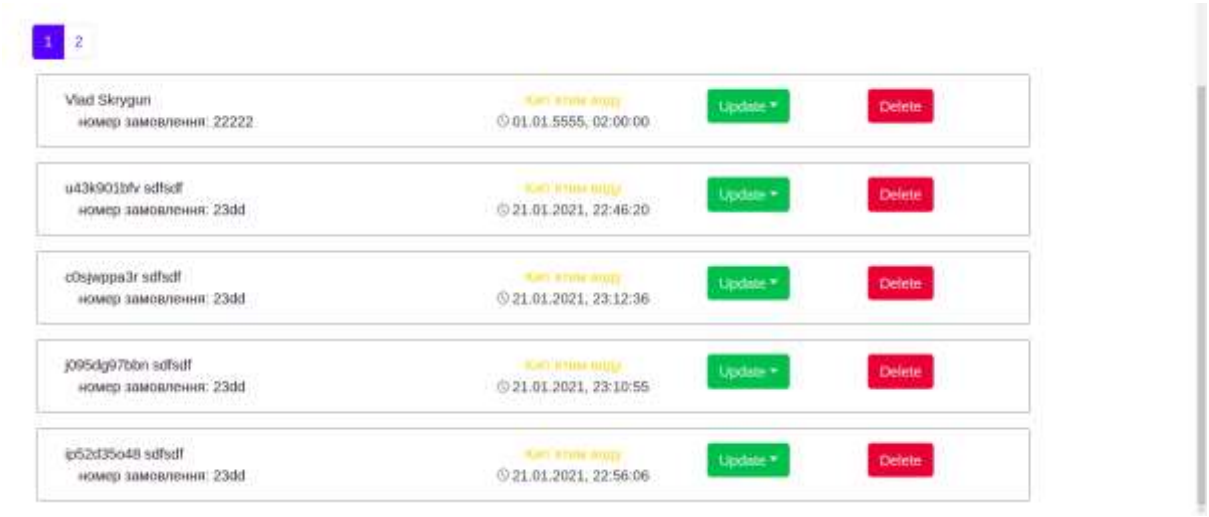


Рис. 4.8. Перелік всіх замовлень

Висновки до розділу 4

У цьому розділі описано та продемонстровано інструкції використання розробленого веб-додатку для звичайних користувачів та адміністраторів, наведено приклад використання клієнтської частини веб-додатку та адміністративної частини. Використовуючи клієнтську частину відвідувачі закладу мають змогу слідкувати за станом свого замовлення, часом приготуванням та отримувати повідомлення про його приготування. За допомогою адміністративної панелі працівники закладу можуть реєструвати замовлення в системі, змінювати його статус та видаляти замовлення.

ЗАГАЛЬНІ ВИСНОВКИ

В даному дипломному проєкті було створено прогресивний вебдодаток (PWA - Progressive Web App), котрий дозволяє оптимізувати систему черг в закладах швидкого харчування та мінімізувати скупчення людей біля закладу, що є актуальним в часи пандемії.

Під час розробки даного веб-додатку було використано:

1) Середовище виконання мови програмування JavaScript - NodeJS. Мову запитів GraphQL, для оптимізації даних, що передаються на клієнт. NoSQL базу даних для зберігання даних про адміністраторів та користувацьких замовлень. Також NoSQL даних дозволяє легко масштабуватися як вертикально (шляхом збільшення потужностей серверу), так і горизонтально (створюючи ще один сервер з базою даних).

Для зручності написання серверної частини було використано фреймворк Apollo Server.

2) React.js для створення як адміністративної панелі SPA (Single Page Application), так і для клієнтської частини, що представляє собою PWA.

Під час розроблення вебдодатку були враховані усі вимоги, які були зазначені в технічному завданні. Мета дипломного проєкту досягнута - розроблено недорогий вебдодаток, котрий дозволяє головному офісу: управляти торговими точками, категоріями, товарами, переглядати статистику по замовленнях в кожній окремій торговій точці, а клієнтам надає зручний сервіс для замовлення їжі в торгових точках зі свого мобільного телефону.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alex R. Young, Bradley Meck, Mike Cantelon. Node.js in Action, Second Edition. Manning Publications, 2017. 420 p.
2. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps 2nd Edition. O'Reilly Media. 2020. 310 p.
3. Buttigieg Stefan, Jevdjenic Milorad. Learning Node.js for Mobile Application Development: Make use of Node.js to learn the development of a simple yet scalable cross-platform mobile application. Packt Publishing, 2015. 248 p.
4. Copeland Rick. MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database. O'Reilly Media, 2013. 160 p.
5. Digital 2021: global overview report. URL: https://datareportal.com/reports/digital-2021-global-overview-report (дата звернення 22.05.2021)
6. Doglio F. REST API Development with Node.js. Apress, 2018. 331 p.
7. Guerrero S. Microservices in SAP HANA XSA: A Guide to REST APIs Using Node.js. Apress Media LLC., 2020. 229 p.
8. Guttman David. Fullstack Node.js: The Complete Guide to Building Production Apps with Node.js. Fullstack.io, 2020. 258 p.
9. Herron D. Node.js Web Development. 5th ed. Packt Publishing, 2020. 711 p.
10. Hibbard J., Kolce J., White L., Wilken J., Holmes S. 9 Practical Node.js Projects. SitePoint, 2018. 232 p.
11. Hunter II Thomas. Distributed Systems with Node.js Building Enterprise-Ready Backend Services. O'Reilly Media, Inc., 2020. 377 p.
12. Magolan Greg, Bell Jay, Guijarro David, Peretti Adrien de, Housley Patrick. Nest.js: A Progressive Node.js Framework. Bleeding Egle Press, 2018. 313 p.
13. Resende Diogo. Hands-On Microservices with Node.js: Build, test, and deploy robust microservices in JavaScript. Packt Publishing Ltd., 2018. 304 p.

14. Vickler Andy. Javascript: Javascript Back End Programming. Independently published, 2021. — 177 p.

15. Аквино Крис, Ганди Тодд. Front-end. Клиентская разработка для профессионалов. Node.js, ES6, REST. СПб.: Питер, 2017. 512 с.

16. Бэнкс А., Порселло Е. GraphQL. Язык запросов для современных веб-приложений. СПб, Питер. 2019. 240 с.

17. Бэнкс А., Порселло Е. React и redux. Функциональная веб-разработка. СПб, Питер. 2018. 336 с.

18. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. СПб, Питер. 2016. 640 с.

19. Крокфорд Д. JavaScript: сильные стороны. СПб.: Питер, 2012. 176 с

20. M. Casciaro, Mammino L. Node.js: Design Patterns. Master a series of patterns and techniques to create modular, scalable, and efficient applications. 3rd ed. Packt, 2020. 661 p.

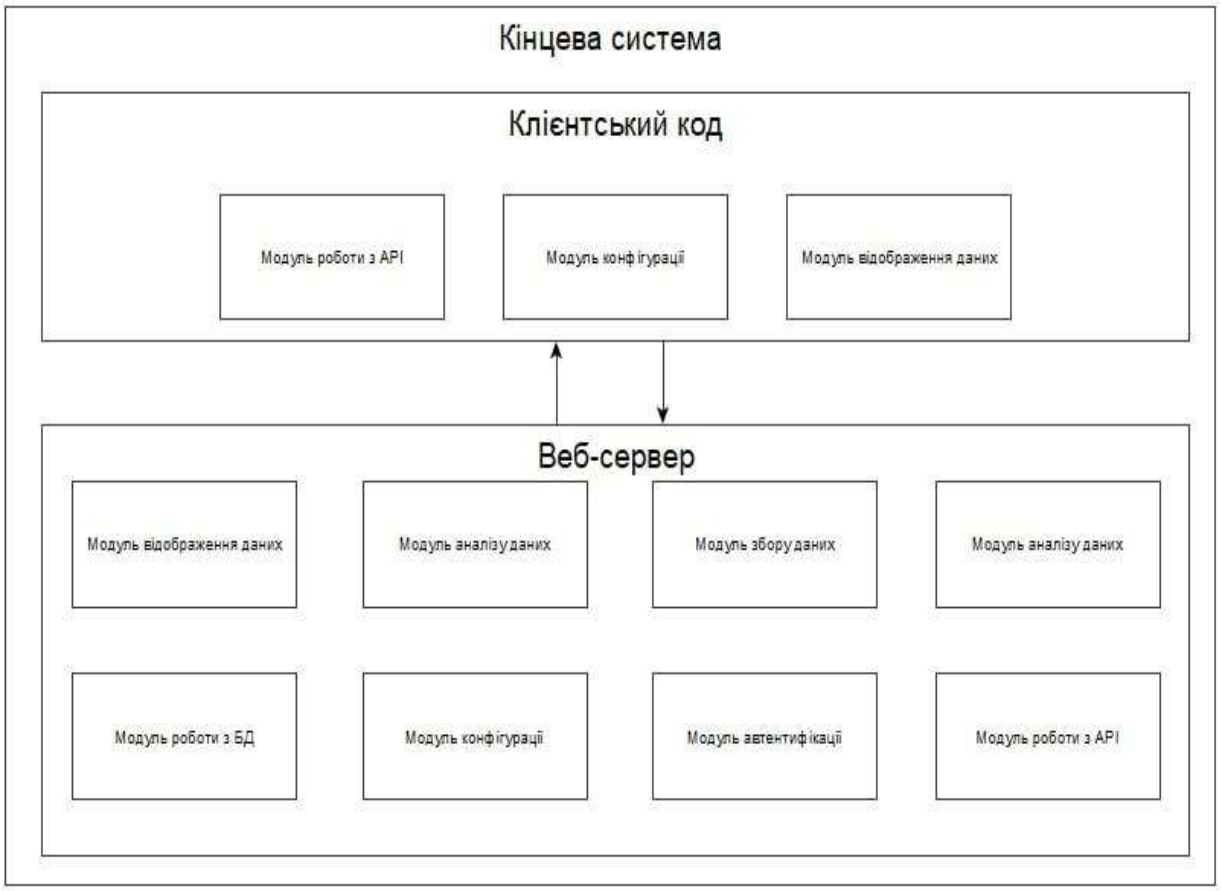
21. Флэнаган Д. JavaScript. Подробное руководство. 6-е издание. СПб, Символ-Плюс, 2012. – 1080 с.

ДОДАТОК А
Прогресивний веб-додаток для закладів швидкого харчування

СХЕМА СИСТЕМИ

Аркушів 1

Київ 2021 р.


Кінцева система
Клієнтський код
Модуль роботи з API
Модуль конфігурації
Модуль відображення даних
Веб-сервер
Модуль відображення даних
Модуль аналізу даних
Модуль збору даних
Модуль аналізу даних
Модуль роботи з БД
Модуль конфігурації
Модуль автентифікації
Модуль роботи з API

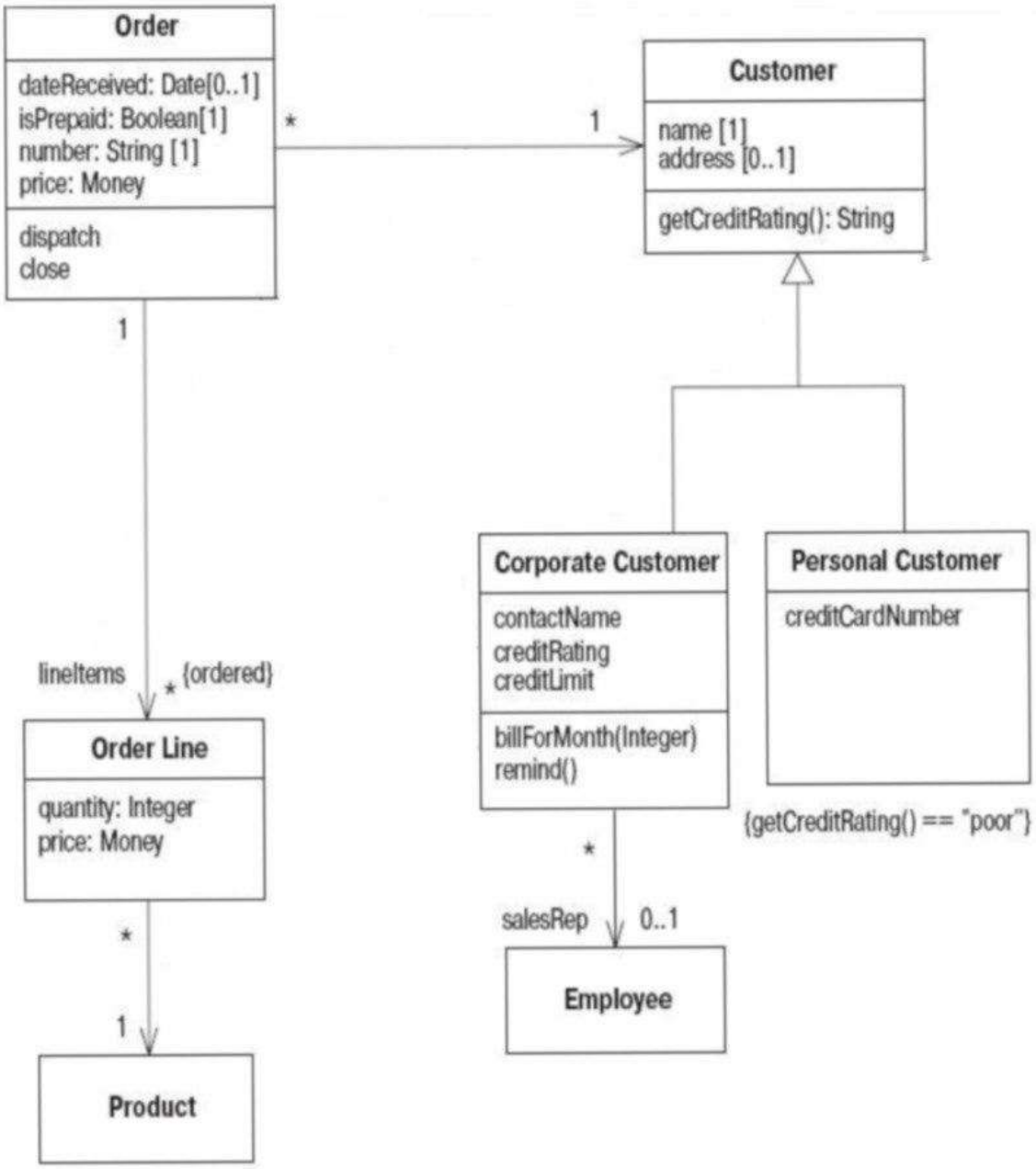
ДОДАТОК Б

Прогресивний веб-додаток для закладів швидкого харчування

ДІАГРАМА КЛАСІВ

Аркушів 1

Київ – 2021 р.


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String [1]
price: Money
dispatch
close
*
1
Customer
name [1]
address [0..1]
getCreditRating(): String
1
lineItems
* {ordered}
Order Line
quantity: Integer
price: Money
*
1
Product
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}
*
salesRep
0..1
Employee

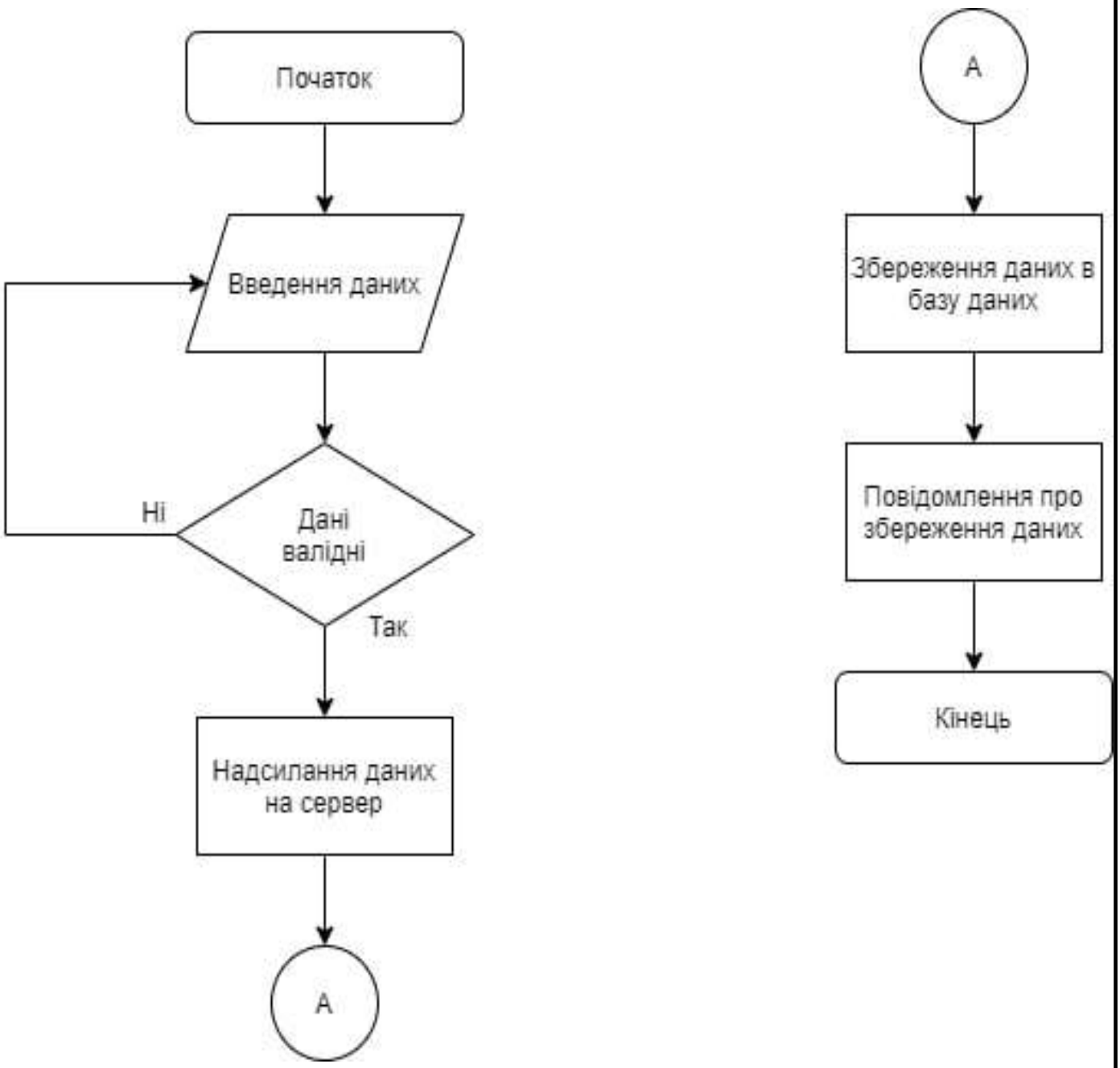
ДОДАТОК В

Прогресивний веб-додаток для закладів швидкого харчування

АЛГОРИТМ СТВОРЕННЯ ЗАМОВЛЕННЯ

Аркушів 1

Київ – 2021 р.


Початок
Введення даних
Дані валідні
Ні
Так
Надсилання даних на сервер
A
A
Збереження даних в базу даних
Повідомлення про збереження даних
Кінець

ДОДАТОК Г
Прогресивний веб-додаток для закладів швидкого харчування

ЛІСТИНГ ПРОГРАМИ

Аркушів 18

Київ 2021 р.

Back-end
schema.graphql

```
schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}

type Mutation {
  ### ORDER
  createOrder(input: CreateOrderInput!): OrderType!
  updateOrder(input: UpdateOrderInput!): OrderType!
  deleteOrder(id: ID!): Boolean

  login(input: LoginInput!): LoginDataType!
}

type Query {
  hello: String!
  listOrders(input: ListOrdersInput!): OrderListType!
}

type Subscription {
  createOrder: OrderType!
  companyOrders(input: CompanyOrdersInput!): OrderListType!
  updateOrder: OrderType!
}



##### ENUMS

enum OrderStatus {
  COOKING
  COOKED
  ACCEPT
}

##### TYPES

type TokenTypes {
  accessToken: String!
  refreshToken: String!
}

type LoginDataType {
  tokens: TokenTypes!
  profile: UserProfileType!
}

type UserProfileType {
  _id: ID!
  email: String!
```

```
  firstName: String!
  lastName: String!
  company: String!
  createdAt: String!
  updatedAt: String!
}

type OrderType {
  _id: ID!
  orderNumber: String!
  cookingTime: String!
  status: OrderStatus!
  company: String!
  createdAt: String!
  firstName: String!
  lastName: String!
  updatedAt: String!
}


type MetaDataListType {
  limit: Int
  skip: Int
  count: Int
  totalCount: Int
}

type OrderListType {
  orders: [OrderType]
  metadata: MetaDataListType
}

##### INPUT

input CreateOrderInput {
  orderNumber: String!
  cookingTime: String!
  firstName: String!
  lastName: String!
  # company: String!
}


input LoginInput {
  password: String!
  login: String!
}

input ListOrdersInput {
  skip: Int
  limit: Int
  find: findListOrdersInputProperty
}

input findListOrdersInputProperty {
```

```
  company: String!
}

input CompanyOrdersInput {
  company: String!
  skip: Int
  limit: Int
}

input UpdateOrderInput {
  id: ID!
  status: OrderStatus!
}
```

index.ts

```
import apolloServer from './lib/apolloServer';
import mongoose     from 'mongoose';
import Users from './lib/models/Users';
import * as dotenv  from 'dotenv';
import path         from 'path';

dotenv.config({ path: path.join(__dirname, '../', '.env') });

const dbUser = process.env.DB_USER;
const dbPassword = process.env.DB_PASSWORD;
const dbName = process.env.DB_NAME;

const connectionUrl: string =
`mongodb+srv://${dbUser}:${dbPassword}@firstcluster.uq8y3.mongodb.net/${dbName}?retryWrites=true&w
=majority`;



async function run(): Promise<void> {

  await mongoose.connect(connectionUrl, { useNewUrlParser: true });
  mongoose.set('useFindAndModify', false);

  apolloServer.listen().then(({ url }) => {

    console.log(`\n🚀 Local Server ready at ${url} and subpath ${apolloServer.subscriptionsPath}`);
  });
}

run();

process.on('SIGINT', () => {
  apolloServer.stop();
  mongoose.connection.close(() => {
    console.log('Mongoose disconnected on app termination');
```

```
    process.exit(0);
  });
});
```

apolloseServer.ts

```
import { ApolloServer, gql, IResolvers } from 'apollo-server';
import fs                          from 'fs';
import getContext                  from './helpers/getContext';
import { corsOrigin }              from './helpers/getEnv';
import * as Query                  from './query';
import * as Mutation               from './mutation';
import * as Subscription           from './subscription';
import apolloServerPluginHttpHeaders          from 'apollo-server-plugin-http-headers';

const typeDefs = gql(fs.readFileSync(__dirname.concat('/schema.graphql'), 'utf8'));

const resolvers = {
  ...Mutation,
  ...Query,
  ...Subscription,
} as IResolvers;
const configApollo = {
  typeDefs, resolvers,
  cors: {
    origin: corsOrigin.split(','),
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    preflightContinue: false,
    optionsSuccessStatus: 204,
    credentials: true,
  },
  methods: [
    'POST',
    'GET',
  ],
  plugins: [apolloServerPluginHttpHeaders],
  context: async (value) => {
    return  await getContext(value);
  },
  formatError: (err) => {
    // Don't give the specific errors to the client.
    console.log(JSON.stringify(err, null, 2));

    return err;
  },
};

const apolloServer = new ApolloServer(configApollo);

export default apolloServer;
```

mutation/index.ts

```
import * as orders from './orders';
import * as session from './session';
```

```
// tslint:disable-next-line: variable-name
export const Mutation = {
  ...orders,
  ...session,
};

                                mutations/orders.ts
import mongoose           from 'mongoose';

import CreateOrder        from '../domainModel/order/CreateOrder';
import UpdateOrder         from '../domainModel/order/UpdateOrder';
import DeleteOrder        from '../domainModel/order/DeleteOrder';
import { CreateOrderDTO }  from '../../../types/dto/orders/CreateOrder';
import CustomError         from '../helpers/errors/CustomError';
import { CREATE_ORDER, UPDATE_ORDER, DELETE_ORDER }          from '../constants';

function checkContext(context): void {
  if (!context.user) {
    const error = {
      fields: {},
      errorMessage: 'PERMISSION_DENIED',
    };

    throw new CustomError(error);
  }
}

async function createOrder(_, args, context): Promise<CreateOrderDTO> {
  checkContext(context);
  const session = await mongoose.startSession();
  session.startTransaction();

  try {
    console.log(context.user);

    const params = {
      ...args.input,
      company: context.user.company,
    };

    const createOrderHelper: CreateOrder = new CreateOrder(params, session);
    const result:CreateOrderDTO = await createOrderHelper.createOrder();

    await session.commitTransaction();
    context.pubsub.publish(CREATE_ORDER, result);
    return result;

  } catch (error) {
    await session.abortTransaction();
    throw error;
  } finally {
    session.endSession();
  }

}
```

```
async function updateOrder(_, args, context): Promise<CreateOrderDTO> {
  checkContext(context);
  const session = await mongoose.startSession();
  session.startTransaction();

  try {
    const updateOrderHelper = new UpdateOrder(args.input, session);
    const result:CreateOrderDTO = await updateOrderHelper.updateOrder();

    await session.commitTransaction();
    context.pubsub.publish(UPDATE_ORDER, result);
    return result;

  } catch (error) {

    await session.abortTransaction();
    throw error;

  } finally {

    session.endSession();

  }
}

async function deleteOrder(_, args, context): Promise<Boolean> {
  checkContext(context);
  const session = await mongoose.startSession();
  session.startTransaction();

  try {
    const updateOrderHelper = new DeleteOrder(args, session);

    const result:Boolean =  await updateOrderHelper.deleteOrder();

    await session.commitTransaction();
    context.pubsub.publish(DELETE_ORDER, result);
    return result;

  } catch (error) {

    await session.abortTransaction();
    throw error;

  } finally {

    session.endSession();

  }
}

export {
    createOrder,
    updateOrder,
```

```
    deleteOrder,
};
```

mutations/session.ts

```
import Login   from '../domainModel/session/Login';
import { CreateSessionDTO } from '../../../types/dto/session/CreateSession';

async function login(_, args, context): Promise<CreateSessionDTO> {
  const loginHelper = new Login(args.input);

  const { setCookies } = context;

  const { tokens, profile } = await loginHelper.login();
  setCookies.push({
    name: 'accessToken',
    value: tokens.accessToken,
    options: {
      httpOnly: true,
      secure: false,
    },
  });

  return { tokens, profile };
}

export {
    login,
};
```

query/index.ts

```
import * as orders from './orders';

// tslint:disable-next-line: variable-name
export const Query = {
  ...orders,
};
```

query/orders.ts

```
import ListOrders         from '../domainModel/order/ListOrders';
import { ListOrdersDTO }  from '../../../types/dto/orders/ListOrders';

async function listOrders(_, args): Promise<ListOrdersDTO> {

  const listOrdersHelper = new ListOrders(args.input);
  const orders:ListOrdersDTO = await listOrdersHelper.listOrders();

  return orders;
}

export {
    listOrders,
};
```

models/Orders.ts

```
import mongoose from 'mongoose';
```

```
import { CreateOrderDTO } from '../../../types/dto/orders/CreateOrder';
const { Schema } = mongoose;

const userSchema = new Schema({
  id: Schema.Types.ObjectId,
  orderNumber: { type: String, required: true },
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  cookingTime: { type: String, required: true },
  status: { type: String, required: true, default: 'COOKING' },
  company: { type: String, required: true },
},
  // tslint:disable-next-line: align
  {
    timestamps: true,
  });

// tslint:disable-next-line: variable-name
const Orders = mongoose.model< CreateOrderDTO & mongoose.Document>('Orders', userSchema);

export default Orders;
```

domainModel/order/CreateOrder.ts

```
import Orders from '../../models/Orders';
import { CreateOrderDTO }  from '../../../../types/dto/orders/CreateOrder';

export default class CreateOrder {
  private params;
  private session;
  constructor(args, transaction) {
    this.params = args;
    this.session = transaction;
  }

  async createOrder() : Promise<CreateOrderDTO> {
    console.log(this.params);
    const order = await Orders
    .create([this.params], { session: this.session });

    return order[0];
  }
}
```

Client
index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
// import './index.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import App from './components/app/app.component';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
```

```
  document.getElementById('root')
);
```

components/app/app.component.tsx

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import { ApolloProvider } from '@apollo/client';
import client from '../../common/apollo-client';
import SwipeableView from '../swipeableView/swipeableView.component'


import '../../common/styles';
import './app.component';

const App: React.FC = () => {
   return (
      <>
         <ApolloProvider client={client()}>
            <Router>
               <Switch>
                  <Route path="/:companyName" component={SwipeableView} />
               </Switch>
            </Router>
         </ApolloProvider>
      </>
   )
}
export default App;
```

swipeableView/swipeableView.component.tsx

```
import React from 'react';
import SwipeableViews from 'react-swipeable-views';

import Header from '../header/header.component';
import About from '../../pages/about/about.page'
import Home from '../../pages/home/home.page'

import '../../common/styles';
import './swipeableView.component';


const SwipeableView: React.FC = ({ match }:any) => {

   let { companyName } = match.params;

   return (
     <>
       <Header companyName={companyName} />
       <SwipeableViews>
         <Home companyName={companyName}/>
         <About/>
       </SwipeableViews>
     </>
   )
```

```
}
export default SwipeableView;
```

orders-grid.component.tsx

```
import React from 'react';
import { Order } from '../../../common/interfaces/order.interface';

import OrdersGridItem from './orders-grid-item/orders-grid-item.component'

interface OrdersGridProps {
   orders: Order[]
}
const OrdersGrid: React.FC<OrdersGridProps> = ({ orders }:OrdersGridProps) => {
   return (
      <div className="orders-grid">
         {orders.map(order => (
            <div key={order._id}>
               <OrdersGridItem order={order} />
            </div>
         ))}
      </div>
   );
}

export default OrdersGrid;
```

Admin

index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bootstrap/dist/css/bootstrap.min.css';
import App from './components/app/app.component'
import { ApolloProvider } from '@apollo/client';
import client from './api/client';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
  <BrowserRouter>
   <ApolloProvider client={client()}>
     <App/>
   </ApolloProvider>
  </BrowserRouter>,
 document.getElementById('root')
);
```

components/login.componet.tsx

```
import React, { useState } from 'react';
import { gql, useMutation } from '@apollo/client';
import { useHistory } from 'react-router';
import './login.component.css'

const LOGIN_MUTATION = gql`
  mutation LoginMutation($variable: LoginInput!) {
```

```
    login(input: $variable) {
    profile {
      _id
      firstName
      lastName
      email
      company
    }
    }
  }
`;

const Login = () => {
  const history = useHistory();
  const [errors, setErrors] = useState({
    email: '',
    password: ''
  });
  const [formState, setFormState] = useState({
    login: true,
    email: 'admin1@queue.com',
    password: 'Password@#333',
  });

  const [login] = useMutation(LOGIN_MUTATION, {
    variables: {
      "variable" : {
        "login": formState.email,
        "password": formState.password
      }
    },
    onCompleted: ({ login }) => {
      // localStorage.setItem('x-api-key', login.tokens.accessToken);
      // localStorage.setItem('refreshToken', login.tokens.refreshToken);
      history.push({
        pathname: '/about',
        state: login
      });
      // <Redirect to="/about" />
    },
    onError: err => {
      if (err.graphQLErrors.length > 0) {
        setErrors((prevState) => {
          return {
              ...prevState,
              email: 'WRONG_EMAIL',
              password: 'WRONG_PASSWORD'
          }
        });
      }
    }
  });

  const loginHandler = (event:React.MouseEvent<HTMLElement>) => {
    login()
```

```
  event.preventDefault()
}

const loginChange = (event:React.ChangeEvent<HTMLInputElement>) => {
  console.log(event.target.name)
  setFormState((prevState) => {
    return {
      ...prevState,
     [event.target.name]: event.target.value
    }
  })
}

return (
  <div>
    <div className="sidenav">
      <div className="login-main-text">
        <h2>Application<br/> Login Page</h2>
        <p>Login or register from here to access.</p>
      </div>
    </div>
    <div className="main">
      <div className="col-md-6 col-sm-12">
        <div className="login-form">
          <form>
            <div className="form-group">
              <label>User Name</label>
              <input
              type="text" className="form-control" placeholder="User Name"
              name="email"
              onChange={loginChange}
              value={formState.email}
              ></input>
            </div>
            <div className="invalid-email">
             {errors.email}
            </div>
            <div className="form-group">
              <label>Password</label>
              <input type="password" className="form-control" placeholder="Password"
                name="password"
                onChange={loginChange}
                value={formState.password}
              >
              </input>
            </div>

            <div className="invalid-password">
             {errors.password}
            </div>

            <button
             type="submit" className="btn btn-black"
             onClick={loginHandler}
            >
```

```
                Login
                </button>
              {/* <button type="submit" className="btn btn-secondary">Register</button> */}
            </form>
          </div>
        </div>
      </div>
    </div>

  );
};

export default Login;
```

login/login.copmponents.css

```
.main-head{
    height: 150px;
    background: #FFF;

}

.sidenav {
    height: 100%;
    background-color: #000;
    overflow-x: hidden;
    padding-top: 20px;
}

.invalid-email {
    color: red;
    margin-bottom: 1em;

}

.invalid-password {
    color: red;
    margin-bottom: 1em;
}


.main {
    padding: 0px 10px;
}

@media screen and (max-height: 450px) {
    .sidenav {padding-top: 15px;}
}

@media screen and (max-width: 450px) {
    .login-form{
        margin-top: 10%;
    }

    .register-form{
```

```
        margin-top: 10%;
    }
}

@media screen and (min-width: 768px){
    .main{
        margin-left: 40%;
    }

    .sidenav{
        width: 40%;
        position: fixed;
        z-index: 1;
        top: 0;
        left: 0;
    }

    .login-form{
        margin-top: 80%;
    }

    .register-form{
        margin-top: 20%;
    }
}


.login-main-text{
    margin-top: 20%;
    padding: 60px;
    color: #fff;
}

.login-main-text h2{
    font-weight: 300;
}

.btn-black{
    background-color: #000 !important;
    color: #fff;
}
```

orders/grid/create.tsx

```
import React, { useState } from 'react';
import { Col } from 'react-bootstrap';
import Form    from 'react-bootstrap/Form'
import Button  from 'react-bootstrap/Button'
import './create.css'


function setTime(time:string) {
  const now = new Date()

  // const timeZoneOffset = now.getTimezoneOffset() * 60 * 1000;
```

```
  const until = new Date(now.getTime() -
   (now.getTimezoneOffset() * 60 * 1000))
    .toISOString();

    console.log({until})

  const nowMinutes = new Date().getMinutes();
  const newTime = new Date(now.setMinutes(nowMinutes + +time)).toISOString();

  console.log({time, newTime})

  return newTime
}

const CreateOrder: React.FC<{createOrder: any, listOrders: any}> = ({createOrder, listOrders}: { createOrder:
any, listOrders: any }) => {

    const [orderInfo, setOrderInfo] = useState({
      firstName: Math.random().toString(36).slice(2),
      lastName: 'sdfsdf',
      orderNumber: '23dd',
    });

    const [cookingTime, setCookingTime] = useState('')
    const [displayCookingTime, setDisplayCookingTime] = useState('');

    const onCookingTimeInput = (event: React.ChangeEvent<HTMLInputElement>) => {

      const targetValue:string = event.target.value;

      if(targetValue.charAt(0) === '0') {
        return;
      } else {
        console.log(targetValue)

        const newTime = setTime(targetValue);

        setCookingTime(newTime)
        setDisplayCookingTime(targetValue)

        console.log('AFTER',cookingTime)
      }

    }

    const onInput = (event: React.ChangeEvent<HTMLInputElement>) => {

      const targetName = event.target.name;
      const targetValue = event.target.value;

      setOrderInfo((prevState: any) => {
        return {
          ...prevState,
          [targetName]: targetValue
        }
```

```
  })
}

const selectTimeBtn = (time: string) => {
  const newTime = setTime(time);

  setCookingTime(newTime)
  setDisplayCookingTime(time)

  console.log('AFTER',cookingTime)
}

const formSubmit = (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault();

  console.log({...orderInfo, cookingTime})

  createOrder({...orderInfo, cookingTime})

  setOrderInfo({
    firstName: Math.random().toString(36).slice(2),
    lastName: 'sdfsdf',
    orderNumber: '23dd',
  })

  // createOrder({...orderInfo, cookingTime}).then(() => {
    // setOrderInfo({
    //   firstName: Math.random().toString(36).slice(2),
    //   lastName: 'sdfsdf',
    //   orderNumber: '23dd',
    // })
  // })
}

return (
    <Form onSubmit={formSubmit}>
    <Form.Row>
      <Form.Group as={Col} md="4" controlId="validationCustom01">
        <Form.Label>First name</Form.Label>
        <Form.Control
          required
          type="text"
          name="firstName"
          placeholder="First name"
          value={orderInfo.firstName}
          onChange={onInput}
        />
        <Form.Control.Feedback>Looks good!</Form.Control.Feedback>
      </Form.Group>

      <Form.Group as={Col} md="4" controlId="validationCustom02">
        <Form.Label>Last name</Form.Label>
        <Form.Control
          required
          type="text"
```

```
    placeholder="Last name"
    name="lastName"
    value={orderInfo.lastName}
    onChange={onInput}
   />
   <Form.Control.Feedback>Looks good!</Form.Control.Feedback>
  </Form.Group>


  <Form.Group as={Col} md="4" controlId="validationCustom03">
   <Form.Label>Order Number</Form.Label>
   <Form.Control
    required
    type="text"
    placeholder="Order number"
    name="orderNumber"
    value={orderInfo.orderNumber}
    onChange={onInput}
   />
   <Form.Control.Feedback>Looks good!</Form.Control.Feedback>
  </Form.Group>

  <Form.Group as={Col} md="4" controlId="validationCustom03">
   <Form.Label>Cooking Time</Form.Label>
   <Form.Control
    required
    type="number"
    placeholder="Cooking time"
    name="cookingTime"
    value={displayCookingTime}
    onChange={onCookingTimeInput}
   />

   <Button variant="outline-secondary" className="time-button"
    onClick={() => selectTimeBtn('5')}
   >05</Button>{' '}
   <Button variant="outline-secondary" className="time-button"
    onClick={() => selectTimeBtn('10')}
   >10</Button>{' '}
   <Button variant="outline-secondary" className="time-button"
    onClick={() => selectTimeBtn('15')}
   >15</Button>{' '}
   <Button variant="outline-secondary" className="time-button"
    onClick={() => selectTimeBtn('30')}
   >30</Button>{' '}

   <Form.Control.Feedback>Looks good!</Form.Control.Feedback>

  </Form.Group>

 </Form.Row>
 <Form.Group>

 </Form.Group>
 <Form.Group>
```

```
        </Form.Group>

        <Button type="submit">Submit form</Button>



      </Form>
    )
}

export default CreateOrder
```