МІНІ МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра економічної кібернетики та інженерії програмного забезпечення

ДО ЗАХИСТУ
ДОПУЩЕНИЙ

Зав. кафедрою _____

(підпис)

д.е.н., доц. Левицький С.І.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Система управління замовленнями для закладів масового харчування

Виконав

ст.гр. ІПЗ-129 _____ Тринц Б.Л.

(підпис)

Керівник

Ст. Викл. _____ Дереза К.В.

Запоріжжя

2023

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

ЗАТВЕРДЖУЮ

Зав. кафедри д.е.н., доц.

Левицький С.І. _____

«__» _____ 2023 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ ДИПЛОМНУ РОБОТУ

Студенту гр. *ІПЗ-129*, спеціальності: *«Інженерія програмного забезпечення»*

Тринц Богдану Леонідовичу

(прізвище, ім'я, по батькові)

1. Тема: *Система управління замовленнями для закладів масового харчування*

затверджена наказом інституту: №_____ від

2. Термін здачі студентом закінченої роботи:

3. Перелік питань, що підлягають розробці:

1. Вивчити літературу присвячену данній тематиці;

2. Виконати аналіз сучасних CRM-систем;

3. Розглянути принцип роботи фреймворків Nest.js та React Native;

4. Розглянути принципи побудови запитів GraphQL;

5. Створити додатки для офіціанта та для клієнта;

6.Оформити результати у вигляді звіту.

Календарний план роботи:

№ п/н	Назва розділів та етапи виконання	Термін виконання	Готовність по графіку (%),	Підпис керівника про готовність етапу, дата
1	Формування теми магістерської роботи	**16.01.23-11.02.23**		
2	**І атестація** І розділ магістерської дипломної роботи	**27.03.23-01.04.23**		
3	Виконання розділу 1	**24.04.23-29.04.23**		
4	**ІІ атестація** ІІ розділ магістерської дипломної роботи	**22.05.23-27.05.23**		
5	**ІІІ атестація** ІІІ розділ магістерської дипломної роботи, висновки, додатки, реферат, перевірка програмою «Антиплагіат»	**15.05.23-12.06.23**		
6	Доопрацювання магістерської дипломної роботи, підготовка презентації, отримання відгуку керівника та рецензії	**29.05.23-12.06.23**		
7	Попередній захист магістерської дипломної роботи	**12.06.23-18.06.23**		
8	Подача магістерської дипломної роботи на кафедру	**за 3 дні до захисту**		
9	Захист магістерської дипломної роботи	**19.06.23-24.06.23**		

Керівник_____(ПІБ) «_____»_____2023р.

Студент_____(ПІБ) «_____»_____2023р.

РЕФЕРАТ

Магістерська дипломна робота складається з: сторінок - 101, рисунків – 31, таблиць – 4, формул – 11, додатків – 5, схем - 1.

Об’єкт дослідження: комплекс з трьох програмних продуктів для організації роботи ресторану-коворкінгу.

Мета роботи: розробка і створення програмного продукту для організації роботи ресторану-коворкінгу.

Результатом роботи став комплекс з трьох програмних продуктів для організації роботи ресторану-коворкінгу

При розробці системи буде використана мова програмування TypeScript, також наступні інструменти та бібліотеки: Nest.js, React Native, GraphQL та Expo.

JAVASCRIPT, TYPESCRIPT, NODE.JS, REACT, REACT NATIVE, NEST.JS

ЗМІСТ

ВСТУП 7

1 РОЗДІЛ 1 9

1.1. Громадське харчування та коворкінг 9

1.2. CRM системи у ресторанному бізнесі 12

1.3. Ресторани-коворкінги Impact Hub Zürich 13

1.4. Аналогічні CRM та особливості 15

1.5. Актуальність та задачі розробки проєкту 16

1.6. Висновки до першого розділу 17

2 РОЗДІЛ 18

2.1. Серверна частина 18

2.1.1. Node.js 18

2.1.2. Nest.js 19

2.1.3. PostgresSQL 21

2.1.4. GraphQL 23

2.2. Фронт частина 24

2.2.1. React Native 24

2.2.2. Expo 28

2.2.3. Apollo Client 29

2.3. Висновки до другого розділу 30

3 РОЗДІЛ 2 31

3.1. Початок роботи. QR-сканер. 31

3.2. Меню ресторану та створення замовлення 34

3.2.1. Програмний код меню 36

3.3. Сторінка замовлення 37

3.3.1. Програмний код сторінки замовлення 38

3.4. Оплата замовлень 40

3.4.1. Програмний код сторінки оплати замовлень 43

3.4.2. Життєвий цикл замовлення 48

3.5. Виклик офіціанта 49

3.6. Додаток офіціанта. Вхід 51

3.7. Сторінка подій 52

3.7.1. Програмна частина функціоналу подій 53

3.8. Сторінка статистики 55

3.8.1. Серверна частина сторінки статистики 56

3.9. Сторінка замовлень 57

3.9.1. Серверна частина роботи з замовленнями 59

3.10. Сторінка чайових 62

3.11. Серверна частина функціоналу чайових 62

3.12. Сторінка імпорту продуктів 63

3.12.1. Серверна частина імпорту 64

3.12.2. Серверна частина експорту 66

4 РОЗДІЛ ..... **Ошибка! Закладка не определена.**

4.1. Визначення трудомісткості робіт **Ошибка! Закладка не определена.**

4.2. Розрахунок чисельності персоналу **Ошибка! Закладка не определена.**

4.3. Загальновиробничі витрати **Ошибка! Закладка не определена.**

4.4. Обґрунтування економічної**Ошибка! Закладка не определена.**

4.5. Висновок до четвертий розділу**Ошибка! Закладка не определена.**

ВИСНОВОК..... 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Слово/словосполучення	Скорочення	Умова використання
A		
Active Server Pages	ASP	В тексті
Application Programming Interface	API	В тексті
C		
Система управління базами да́них	СУБД	В тексті
Customer Relationship Management	CRM	В тексті
Comma-Separated Values	CSV	В тексті
J		
JavaScript Object Notation	JSON	В тексті
Q		
Quick Response	QR	В тексті

S		
Structured Query Language	SQL	В тексті

ВСТУП

Сучасний бізнес у сфері гастрономії та коворкінгу все більше прагне досягти успіху, залучаючи і утримуючи клієнтів. Одним із ключових факторів в цьому процесі є ефективне управління взаємодією з клієнтами. Для досягнення цієї мети дедалі більше підприємств обертаються до використання CRM систем (Customer Relationship Management - систем управління взаємовідносинами з клієнтами).

Ця дипломна робота присвячена дослідженню та розробці CRM системи спеціально для кафе-коворкінгу. Кафе-коворкінги є особливим типом закладів, де клієнти можуть не лише насолоджуватись смачною їжею та напоями, але й виконувати свої робочі завдання в комфортному та сприятливому середовищі.

Наша CRM система буде спеціально розроблена з урахуванням унікальних потреб та вимог кафе-коворкінгів. Вона надасть можливість власникам та персоналу таких закладів ефективно керувати всіма аспектами взаємодії з клієнтами - від прийому та обробки замовлень до управління резерваціями робочих місць та столиків.

Метою цієї дипломної роботи є розробка та реалізація інноваційної CRM системи, яка допоможе кафе-коворкінгам забезпечити високий рівень обслуговування клієнтів, підвищити лояльність та залучити нових клієнтів, а також покращити ефективність управління всіма аспектами їх діяльності. Дослідження буде базуватися на аналізі сучасних підходів до CRM, а також специфічних потреб та вимог кафе-коворкінгів.

Очікується, що результати цієї роботи допоможуть покращити ефективність та конкурентоспроможність кафе-коворкінгів шляхом впровадження спеціалізованої CRM системи.

1 РОЗДІЛ
АНАЛІЗ ТА ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ДИПЛОМНОГО ПРОЄКТУ. РЕСТОРАНИ КОВОРКІНГИ – ГРОМАДСЬКЕ ХАРЧУВАННЯ

1.1. Громадське харчування та коворкінг

Підприємствами громадського харчування називаються організації, які надають населенню послуги у вигляді виробництва, реалізації кулінарної продукції, організації харчування.

Усі підприємства громадського харчування мають чітку градацію залежно від послуг чи особливостей кулінарної продукції. Відмінності дозволяють виділяти типи підприємств із властивими кожному визначальними чинниками.

За асортиментом послуг підприємства громадського харчування прийнято підрозділяти на такі види послуг, що надають споживачам:

- Виробництво кулінарної продукції;
- Реалізація;
- Організація питання;
- Обслуговування;
- Організація дозвілля;
- Інформаційні послуги;
- Консультативні послуги.

Загальні вимоги до підприємств громадського харчування включають обов'язкове дотримання регламентованих правил, включаючи санітарні, пожежні та інші, які забезпечують якість і безпеку послуг.

Основні типи підприємств громадського харчування включають столові, закусочні, кафе, ресторани та бари. Заготівельні підприємства, такі як кулінарні фабрики, комбінати напівфабрикатів та заготівельні фабрики, є особливою

категорією. Вони здійснюють механізоване виробництво з метою постачання іншим підприємствам громадського харчування. Комбінати харчування та фабрики-кухні спрямовані на великі обсяги виробництва. Роздрібна реалізація кулінарних продуктів здійснюється іншими підприємствами, такими як магазини кулінарії, буфети та інші заклади.

Коворкінг - це комфортне простір для спільної роботи або колективного офісу, де люди можуть орендувати робочі місця на короткий або довгий період, користуватися інтернетом та офісною технікою, перекусити та знайти однодумців.

Іншими словами, це щось середнє між офісом та кафе. У коворкінгу є свій робочий стіл, переговорні кімнати та зони відпочинку, ви можете користуватися ноутбуком та офісною технікою. Але він не має чіткого графіка роботи, дрес-коду або суворих правил. На рисунку 1.1 зображено план типового коворкінгу.

Рис. 1.1 – План типового коворкінгу

Концепцію коворкінгу можна розглядати з кількох аспектів. По-перше, він надає можливість заощадити на окремому офісі, що особливо корисно для фрілансерів та підприємців. По-друге, це місце, де можуть працювати тимчасово люди, що знаходяться в командировках або не можуть працювати у своїх постійних офісах через конфлікти або війну. По-третє, це простір, де фахівці

різних професій можуть спілкуватися та обмінюватися досвідом, що часто не вистачає, коли працюєш з дому.

Іноді коворкінги називають "гнучкими офісами". Ці поняття схожі, але мають певні відмінності. "Гнучкі офіси" більше орієнтовані на корпоративних клієнтів, які мають окремі приміщення, які можна адаптувати під їхні потреби. У коворкінгах відвідувачі орендують робоче місце в відкритому просторі (Open Space) і не можуть перебудовувати його за своїм бажанням.

Хто скористується послугами коворкінгів:

- Початківці-підприємці - вони можуть орендувати робочі місця, конференц-зали та переговорні кімнати, що дозволяє їм знизити витрати порівняно з постійною орендою офісу і змінювати кількість місць, що потрібно орендувати;
- Фрілансери та віддалені працівники - вони можуть орендувати робочі місця та офісне обладнання, а також скористатися додатковими послугами, такими як обіди, тренажерні зали, навчальні курси та семінари;
- Компанії - вони можуть орендувати приміщення для проведення семінарів та конференцій. В коворкінгах також можуть працювати співробітники невеликих фірм, яких тимчасово наймають для роботи над проектами;
- Подорожуючі - вони можуть орендувати робоче місце на час своїх поїздок;
- Батьки з дітьми, які працюють - деякі коворкінги надають ігрові зони та послуги нагляду за дітьми, що дозволяє батькам працювати в сусідніх приміщеннях, поки малята займаються своїми справами;
- Люди, у яких вдома відсутність електроенергії та інтернету - в умовах аварій, війни або планових відключень електропостачання, коворкінги стають прихистком для тих, хто намагається знайти робоче місце та доступ до інтернету.

1.2. CRM системи у ресторанному бізнесі

Здається, що CRM системи в ресторанному бізнесі можуть здатися непотрібними, оскільки менеджери можуть контролювати багато процесів вручну. Проте, така система дозволяє спростити та автоматизувати багато завдань, що полегшує контроль, поліпшує зрозумілість та прозорість. Вона допомагає забезпечити швидкий сервіс клієнтів, якісне обслуговування та внутрішній контроль, а також вирішує завдання з прийому замовлень від клієнтів, ведення бази даних клієнтів, маркетингу та автоматизації розсилок, а також контролю роботи персоналу. Використання CRM систем дозволяє підвищити ефективність бізнесу та забезпечити найкращий сервіс для клієнтів.

Програмне забезпечення допомагає компаніям в висококонкурентному бізнесі не тільки утримувати свої позиції, але й покращувати їх. CRM система вирішує три основні завдання: швидкий сервіс клієнтів, якісне обслуговування та внутрішній контроль. Проте, це лише декілька здібностей, які CRM системи для ресторанів можуть виконувати. Ось деякі завдання, що можуть бути вирішені за допомогою CRM систем для ресторанів:

- Прийом замовлень від клієнтів: це особливо важливо для закладів з доставкою, оскільки потрібно організувати ефективну взаємодію між менеджером, кухнею, кур'єром та клієнтом з мінімальним ризиком помилок.
- Ведення бази даних клієнтів: у конкурентному середовищі необхідно не тільки залучати клієнтів, але й утримувати їх, пропонувати персоналізовані умови та збільшувати їх лояльність. CRM система для ресторанів може досягати цих цілей шляхом якісного ведення бази даних, в яку включаються ключові відомості. Клієнтів можна сегментувати, керувати роботою менеджера та залучати нових користувачів.
- Вирішення маркетингових завдань: завдяки сегментації клієнтів та розширеним звітам, проведення маркетингових кампаній стає простішим і

ефективнішим. Можна створювати персоналізовані умови, контролювати продажі, поліпшувати якість реклами та збільшувати активний трафік.

- Автоматизація розсилок: за допомогою CRM можна легко автоматизувати розсилки зі спеціальними пропозиціями, акціями та рекламою. Це зменшує роботу менеджерів та покращує якість розсилок, оскільки їх можна здійснювати точно вибраним аудиторіям.
- Контроль роботи персоналу: зберігання дзвінків, ведення історії взаємодії з клієнтами. CRM для ресторанів дозволяє створювати звіти про ефективність роботи співробітників, складати графіки роботи та проводити фінансові розрахунки. CRM системи для ресторанів є потужними інструментами, які можуть піднести бізнес на новий рівень та здатні перевершити конкурентів, надаючи клієнтам найкращий сервіс. Використання правильних технологій дозволить автоматизувати процеси, вирішувати багато бізнес-завдань та збільшувати дохід.

1.3. Ресторани-коворкінги Impact Hub Zürich

Компанія, для якої розробляється система називається Impact Hub Zürich. Impact Hub Zürich - це швейцарська компанія, яка володіє мережею ресторанів-коворкінгів. На даний момент в мережі є три заклади, які знаходяться в місті Цюріху, а саме:

1. Impact Hub Café Auer&Co;
2. Impact Hub Colab;
3. Impact Hub Kraftwerk.

Всі ресторани мережі зображено нижче на рисунках 1.2, 1.3 та 1.4.

Рис. 1.2 – Головний зал Impact Hub Colab

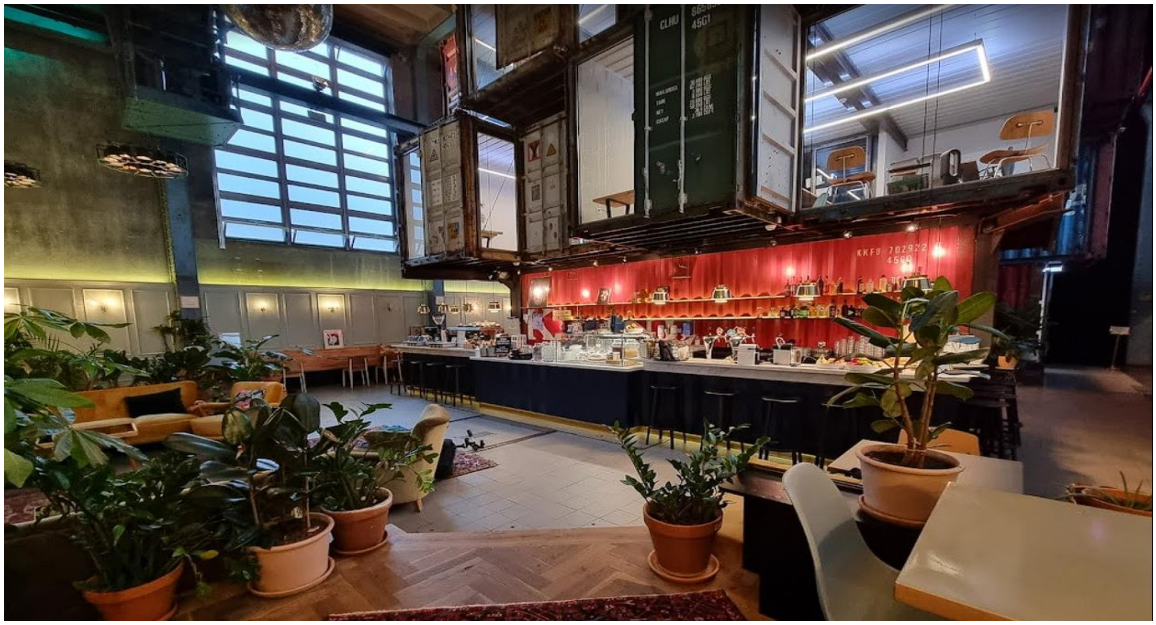


Рис. 1.3 – Головний зал Impact Hub Kraftwerk

Рис. 1.4 – Тераса Impact Hub Café Auer&Co

Стимулом для замовлення CRM системи став карантин (розробка почалась в кінці 2021 року) - Impact Hub Zürich потрібна легка та доступна система, що дозволить мінімізувати фізичний контакт персоналу та клієнта.

1.4. Аналогічні CRM та особливості

CRM-систем для ресторанів в світі розроблена велика кількість. Їх настільки багато, що навіть чисто українських розробок можна знайти не менше 100. Але в системі, розробленій для Impact Hub Zürich є особливості, які виділяють її на фоні будь-якої іншої системи і не дають провести звичайного порівняння.

По-перше, по бізнес-логікі дана система ближче до інтернет магазинів - життєвий цикл замовлення, наприклад, був наполовину скопійований з життєвого циклу замовлень Shopify. Онлайн-оплата замовлень та надання чеків по електронній пошті також перекочували напряму з бізнес-процесу інтернет-магазина.

По-друге, в розробленій системі є компоненти і для менеджерського персоналу, і для звичайних працівників, і для клієнтів. Менеджер має доступ

до керування правами користувачів, контенту та іншого. Офіціант обмежений редагуванням та закриттям замовлень, імпортом та експортом продуктів та доступом до конвеєру активностей. Клієнт може лиш відривати замовлення та оплачувати їх.

По-третє, в угоду легкості були дуже сильно обрізані (точніше взагалі не реалізовані) можливості для збору та перегляду статистики. На відміну від більшості подібних систем, тут можливості для статистики доволі мізерні – статистика продажів по категоріям товарів, кількість замовлень та загальні фінанси.

По-четверте, завдяки жертвам в вигляді обрізаної статистики, дана система не потребує спеціального хостингу або обладнання. По системним вимогам розроблена система стоїть на одному рівні з звичайними сайтами або веб-додатками. З клієнтського боку ситуація аналогічна – веб-компоненти системи звичайні веб-додатки, тому працюють на будь-якому пристрої з браузером.

По-п’яте, життєвий цикл замовлень та інші компоненти системи були оптимізовані для роботи в умовах коворкінгу, де клієнт може знаходитись не одну годину, створити не одне замовлення та використовувати різні способи оплати.

1.5. Актуальність та задачі розробки проєкту

Станом на початок 2023 року можна рахувати, що основна хвиля коронавірусові інфекції пройшла. Але в більшості країн Європи, в тому числі Швейцарії, карантин був дуже послаблений або взагалі знятий, ковід все ще дуже впливає на життя та бізнес. Люди тепер надають перевагу електронним комунікаціям, мінімізуючи прямий контакт між один одним. Також в суспільстві спостерігається страх перед потенційною загрозою в вигляді особливого штаму коронавірусу або взагалі новим вірусом. Тому бізнеси повинні підлаштовуватись під нові реалії. Розроблена система (проект для

дипломної роботи), як раз є відповіддю на такий виклик – при наявності у клієнта смартфону та банківського рахунку можна взагалі не розмовляти з персоналом ресторану.

Отже, отримавши всю інформацію та проаналізувавши її, поставлені наступні задачі:

- Зібрати інформацію за тематикою дипломного проекту;
- Дати аналіз існуючим або неіснуючим аналогам подібних систем;
- Розробити програмний продукт по заданому технічному завданню;
- При розробці задіяти всі обрані інструменти розробки та описати їх у відповідному розділі звіту;
- Представити керівництво користувача з боку всіх потенційних користувачів;
- Представити обрані частини вихідного коду програмного продукту;
- По всій представленій інформації зробити відповідні висновки та представити до захисту.

1.6. Висновки до першого розділу

В першому розділі бакалаврської дипломної роботи було розглянуто та описано:

1. Сферу, для якої розроблено додаток, а саме сфера закладів масового харчуванні;
2. Поняття коворкінгу для кращого розуміння причин, які змусили розробити систему саме в такому вигляді та саме з таким набором функцій;
3. Мережа ресторанів, для якої створено систему;
4. Актуальність даної розробки як для роботи так і для дипломної роботи;
5. Описано особливості системи, які виділяють її серед подібного програмного забезпечення.’

2 РОЗДІЛ
ІНСТРУМЕНТАРІЙ РОЗРОБКИ ДИПЛОМНОГО ПРОЄКТУ

2.1. Серверна частина

Серверна частина виконана на платформі Node.JS з використанням фреймворку Nest.js. Використана СУБД – PostgresSQL. Додаткові технології – GraphQL.

2.1.1. Node.js

Node.js є середовищем виконання коду JavaScript, побудованим на базі движка JavaScript Chrome V8. Воно дозволяє перетворювати виклики JavaScript на машинний код. Основне призначення Node.js - створення серверних програм на мові JavaScript. Також існують проекти, які використовують Node.js для написання десктопних додатків (наприклад, Electron) і навіть для створення коду для мікроконтролерів. Нижче наведено зображення логотипу.

Рис. 2.1 – Логотип Node.js

Node.js найчастіше використовується як веб-сервер, і саме в цій ролі він проявляє переваги подієво-орієнтованої архітектури, яка не блокує введення/виведення. Завдяки здатності розподіляти ресурси сервера в залежності від активності та бездіяльності, разом з вбудованою бібліотекою Libuv, Node.js перетворює один потік JavaScript на нескінченний цикл вирішення завдань. Це робить Node.js унікальним рішенням.

2.1.2. Nest.js

NestJS - це фреймворк, призначений для розробки серверних програм на Node.js. Він надає просте та чітке середовище з великим набором можливостей, і його користувачна база постійно зростає з кожним релізом.

NestJS розроблений для створення складних та унікальних систем. Він має вбудовані компоненти, необхідні для швидкого запуску проектів, але також забезпечує інтеграцію з будь-якими компонентами Express.js, бібліотеками та модулями TypeScript та JavaScript, а також багатьма іншими. Нижче наведено зображення логотипу.

Рис. 2.2 – Логотип Nest.js

Завдяки зручному, ASP-подібному синтаксису, компоненти виходять структурованими та зручними для розуміння. Приклад використання синтаксису наведено нижче. Це один з резолверів проекту – модуль який обробляє запити з додатку для офіціанта.

Лістинг коду 2.1 – Admin Resolver

```
import {Args, Mutation, Query, Resolver} from '@nestjs/graphql';
import {Ctx, RequestContext} from '@vendure/core'
import {CustomOrderService} from "../services/customOrder.service";
import {TipsService} from "../services/tips.service";
import {ActionService} from "../services/action.service";
import {ActionEntity} from "../entities/action.entity";

@Resolver()
export class AdminResolver {

  constructor(private customOrderService: CustomOrderService,
              private tipsService: TipsService,
              private actionService: ActionService) {}

  @Query()
  async getActions(@Ctx() ctx: RequestContext) {
    try {
      return await this.actionService.getActions(ctx)
    } catch (e) {
      throw new Error(e)
    }
  }

  @Query()
  async getTips(@Ctx() ctx: RequestContext) {
    try {
      return await this.tipsService.getAll(ctx)
    } catch (e) {
      return []
    }
  }

  @Query()
  async getGatewayDataById(@Ctx() ctx: RequestContext, @Args() args: any) {
    try {
      const {transaction: {data} } = await
this.customOrderService.getPaymentsData(ctx, args.gatewayId)
      return {
        id: data.id,
        status: data.status,
        amount: data.amount,
        time: data.time,
      }
    } catch (e) {
      console.error(e)
    }
  }

  @Query()
  async getOpenOrders(@Ctx() ctx: RequestContext, @Args() args: any) {
```

```
    try {
      return await this.customOrderService.getOpenOrders(ctx, args.tableId)
    } catch (e) {
      throw new Error(e)
    }
  }

  @Mutation()
  async setAction(@Ctx() ctx: RequestContext, @Args() args:
Partial<ActionEntity>) {
    return await this.actionService.update(ctx, args)
  }

  @Mutation()
  async setVisitAllActions(@Ctx() ctx: RequestContext, @Args() args:
Partial<ActionEntity>) {
    return await this.actionService.setVisitedForAllActions(ctx)
  }

  @Mutation()
  async setVisitedAction(@Ctx() ctx: RequestContext, @Args() args: {id:
number}) {
    try {
      const action = await this.actionService.getActionByOrderId(args.id)
      if(!action) {
        throw new Error('Action not found')
      }
      return await this.actionService.update(ctx, {id: action.id, visible:
false})
    } catch (e) {
      throw new Error(e)
    }
  }

  @Mutation()
  async modifyPayLaterOrder(@Ctx() ctx: RequestContext, @Args() args: {id:
number}) {
    console.log('resolver')
    return await this.customOrderService.modifyPayLaterOrder(ctx, args.id)
  }

}
```

2.1.3. PostgresSQL

PostgreSQL є однією з найбільш популярних систем управління базами даних, і сам проект PostgreSQL поступово розвивався з іншого проекту, відомого як Ingres. Фактично, розвиток PostgreSQL розпочався ще у 1986 році під назвою POSTGRES. Але у 1996 році проект був перейменований на PostgreSQL, що вказувало на більший акцент на мову запитів SQL. 8 липня 1996 року відбувся

перший реліз даного продукту. З того часу вийшло безліч версій PostgreSQL. Нижче зображено логотип.

Рис. 2.3 – Логотип PostgresSQL

PostgresSQL використовує синтаксис SQL для виконання запитів. Приклад одного з запитів, а саме запит на створення таблиці замовлень, наведено нижче.

Лістинг коду 2.2 – Створення таблиці

```
CREATE TABLE public."order" (
    "createdAt" timestamp without time zone DEFAULT now() NOT NULL,
    "updatedAt" timestamp without time zone DEFAULT now() NOT NULL,
    code character varying NOT NULL,
    state character varying NOT NULL,
    active boolean DEFAULT true NOT NULL,
    "orderPlacedAt" timestamp without time zone,
    "couponCodes" text NOT NULL,
    "shippingAddress" text NOT NULL,
    "billingAddress" text NOT NULL,
    "currencyCode" character varying NOT NULL,
    "subTotal" integer NOT NULL,
    "subTotalWithTax" integer NOT NULL,
    shipping integer DEFAULT 0 NOT NULL,
    "shippingWithTax" integer DEFAULT 0 NOT NULL,
    id integer NOT NULL,
    "taxZoneId" integer,
    "customerId" integer
);
```

2.1.4. GraphQL

GraphQL - мова запитів даних та мова маніпулювання даними з відкритим вихідним кодом для побудови веб-орієнтованих програмних інтерфейсів. GraphQL була розроблена як внутрішній проект компанії Facebook у 2012 році, а пізніше у 2015 році була випущена публічно. За допомогою GraphQL можна отримати дані з API та передати їх у програму (від сервера до клієнта). Логотип зображено нижче.

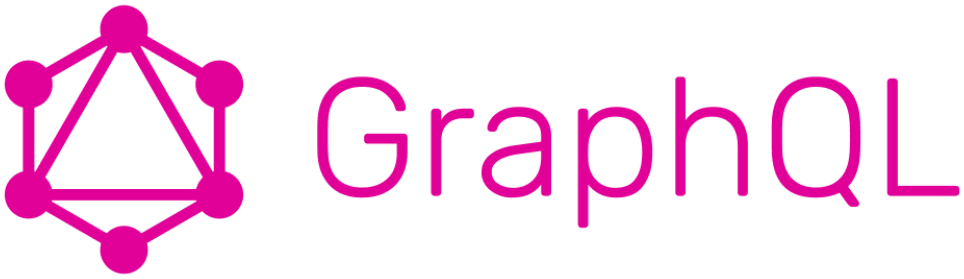


Рис. 2.4 – Логотип GraphQL

Синтаксис GraphQL дуже зручний та зрозумілий, особливо якщо дані, які отримуються або відправляються запитом, вміло структуровані та підготовлені. Приклад використання GraphQL наведено нижче.

Лістинг коду 2.3 – Шема деяких запитів та мутацій

```
extend type Query {
  getOpenOrders(tableId: String!): [Order!]
}

extend type Mutation {
  setVisitedAction(id: Int!): Action!
}

extend type Mutation {
  setAction(id: Int!, visible: Boolean): Action!
}
```

```
extend type Mutation {
  setPaymentAfterSuccess(orderIdsList: [String!], method: String!, tip: Int!,
metadata: String!): [Order]
}

extend type Mutation {
  setStateAndActiveOrderList(orderIdsList: [String!], state: String!, active:
Boolean!): [Order]
}

extend type Query {
  getGatewayDataById(gatewayId: String): PaymentData!
}
```

2.2. Фронт частина

Веб-додаток написано на React Native з використанням білдеру Expo. Додаткові технології – Apollo Client.

2.2.1. React Native

React Native (також відомий як RN) - це популярна платформа мобільних програм на основі JavaScript, яка дозволяє створювати мобільні програми з власним інтерфейсом для iOS і Android. Фреймворк дозволяє створювати програми для різних платформ, використовуючи ту саму кодову базу. Логотип React Native зображено нижче.

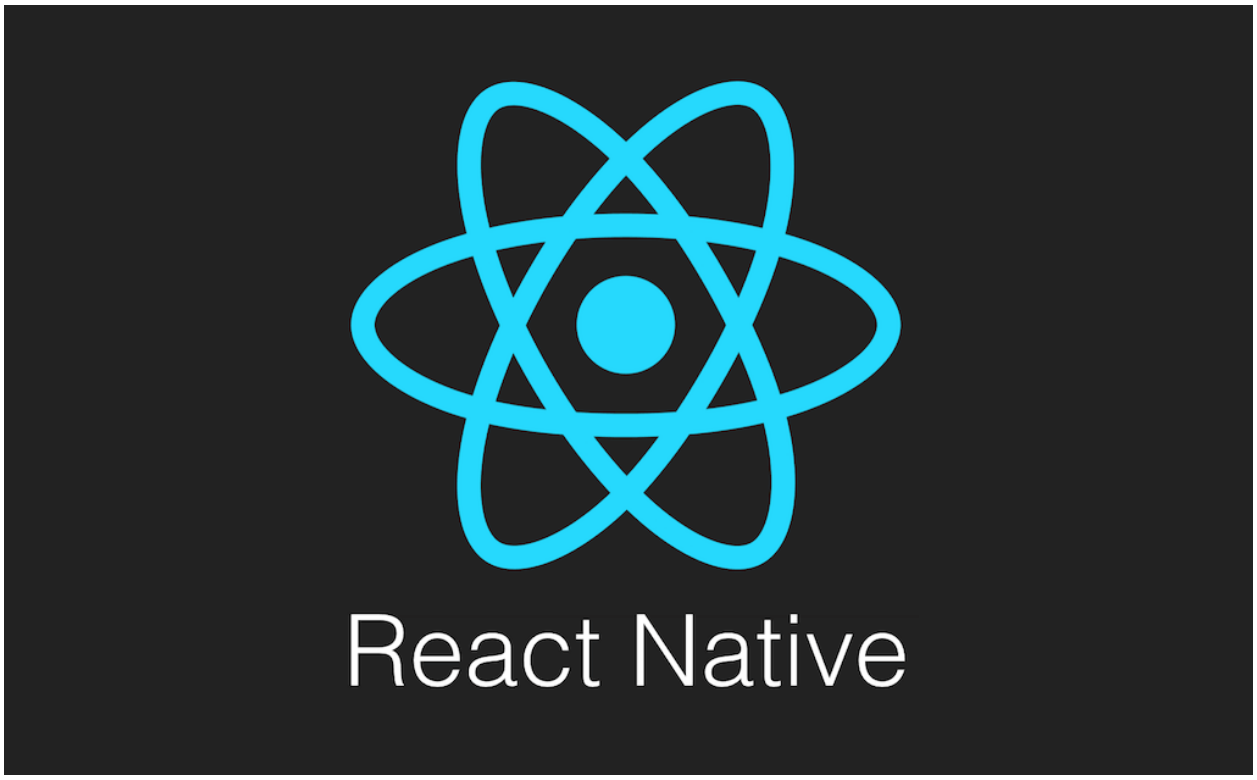


Рис. 2.5 – Логотип React Native

У 2015 році Facebook випустив React Native як проект з відкритим вихідним кодом. За декілька років він швидко став одним з найпопулярніших інструментів для мобільної розробки. React Native використовується для створення деяких найвідоміших мобільних додатків у світі, включаючи Instagram, Facebook і Skype.

Існує кілька причин глобального успіху React Native.

По-перше, React Native дозволяє компаніям створювати код один раз і використовувати його для розробки програм як для iOS, так і для Android. Це призводить до значного збереження часу та ресурсів.

По-друге, React Native заснований на бібліотеці JavaScript під назвою React, яка вже була надзвичайно популярною на момент випуску мобільної платформи.

По-третє, цей фреймворк дозволяє розробникам інтерфейсів, які раніше могли працювати лише з веб-технологіями, створювати надійні програми для мобільних платформ, готові до роботи.

Коли Facebook першість вирішив зробити свій сервіс доступним на мобільних пристроях, вони обрали шлях запуску мобільної веб-сторінки, замість

розробки власної програми, як багато інших провідних технологічних компаній того часу. Це рішення базувалося на використанні HTML5. Проте, цей підхід не виявився досить стійким на протязі часу, і він залишив багато простору для поліпшення користувацького інтерфейсу та продуктивності.

Незабаром після цього, в 2013 році, розробник Facebook Джордан Волк зробив революційне відкриття - він знайшов метод створення елементів UI для програм iOS за допомогою JavaScript . Це викликало фурор і був організований спеціальний хакатон, щоб ще більше дізнатися, як багато мобільної розробки можна зробити з використанням традиційних веб-рішень JavaScript .

Ось так і народився React Native. Спочатку розроблений тільки для iOS, Facebook швидко продовжив його за допомогою Android, перш ніж опублікувати фреймворк в 2015 році.

Синтаксис React Native подібний до такого у React, але має свої особливості. Приклад компоненту, а саме модальне вікно для оплати замовлення готівкою, наведено нижче.

Лістинг коду 2.4 – Компонент PayCashModal

```
import { Image, Text, View } from 'react-native'
import { StyleService, useStyleSheet } from '@ui-kitten/components'
import Button from '../../ui/primitive/components/Button'
import React from 'react'
import hand from '../../assets/icons/successHand.png'

const PayCashModal = ({ handler }) => {
  const styles = useStyleSheet(themedStyles)

  const okHandler = () => {
    handler()
  }

  return (
    <View style={styles.wrapper}>
      <View style={styles.backdrop} />
      <View style={styles.card}>
        <Text style={styles.header}>Waiter notified</Text>
        <Text style={styles.subHeader}>
          Waiter will come to your table so you can pay in cash. :)
        </Text>
        <Image source={hand} style={styles.image}></Image>
        <Button type={'blue'} style={styles.button} onPress={okHandler}>
          <Text style={styles.buttonText}>Ok, Thank you</Text>
        </Button>
      </View>
    </View>
```

```
  )
}

export default PayCashModal

const themedStyles = StyleService.create({
  wrapper: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    height: '100%',
    overflow: 'hidden',
    position: 'absolute',
    top: 0,
    right: 0,
    zIndex: 20,
  },
  image: {
    width: 81,
    height: 72,
    marginHorizontal: 'auto',
    marginBottom: 32,
  },
  backdrop: {
    width: '100%',
    height: '100%',
    backgroundColor: '#999999',
    position: 'absolute',
    top: 0,
    left: 0,
  },
  card: {
    width: '80%',
    backgroundColor: '#FFFFFF',
    borderRadius: 8,
    padding: 24,
    display: 'flex',
  },
  header: {
    color: '#333333',
    textTransform: 'uppercase',
    fontSize: 28,
    fontFamily: 'oswald-bold',
    marginBottom: 12,
  },
  subHeader: {
    color: '#333333',
    fontFamily: 'inter-regular',
    fontSize: 14,
    marginBottom: 32,
  },
  button: {
    height: 48,
    width: '100%',
    borderRadius: 28,
    marginBottom: 16,
  },
  buttonText: {
    color: '#FFFFFF',
    fontFamily: 'inter-bold',
    fontSize: 12,
    letterSpacing: 2,
    textTransform: 'uppercase',
```

```
  },
})
```

2.2.2. Expo

Expo — це платформа з відкритим кодом для створення універсальних React Native програм, які працюють на Android, iOS і в Інтернеті. Він включає в себе універсальне середовище виконання та бібліотеки, які дозволяють створювати нативні програми, створюючи React і JavaScript.

Expo Application Services (EAS) — це платформа розміщених служб, яка глибоко інтегрована з відкритими інструментами Expo. EAS допомагає створювати, надсилати та повторювати програму окремо чи командою. Логотип Expo зображено нижче.

Рис. 2.6 – Логотип Expo

2.2.3. Apollo Client

Apollo Client є розширеною бібліотекою управління станом для JavaScript, що забезпечує можливість ефективно контролювати як локальні, так і віддалені дані з використанням GraphQL. Нижче наведено зображення логотипу.

Рис. 2.7 – Логотип Apollo

Клієнт Apollo допомагає структурувати код економним, передбачуваним і декларативним способом, що відповідає сучасній практиці розробки. Основна @apollo/client бібліотека забезпечує вбудовану інтеграцію з React.

Apollo використовує синтаксис GraphQL та React для зручної роботи з бєк-частиною проекту. Приклад одного з хуків, а саме хук реєстрації нового користувача, наведено нижче.

Лістинг коду 2.5 – Хук для реєстрації користувача

```
import { gql, useMutation } from '@apollo/client'
import { GET_ACTIVE_CUSTOMER } from './useActiveCustomer'
import { Register, RegisterVariables } from './types/Register'
export const REGISTRATION = gql`
  mutation Register(
    $email: String!
    $firstName: String
    $lastName: String
    $password: String!
  ) {
    registerCustomerAccount(
      input: {
        emailAddress: $email
        firstName: $firstName
        lastName: $lastName
```

```
        password: $password
      }
    ) {
      ... on Success {
        success
      }
    }
  }
`

const useRegistration = () => {
  const [register, data] = useMutation<Register, RegisterVariables>(
    REGISTRATION,
    {
      refetchQueries: [GET_ACTIVE_CUSTOMER],
      awaitRefetchQueries: true,
    },
  )

  const handler = async (variables: RegisterVariables) => {
    const { data } = await register({ variables })

    if (data.registerCustomerAccount.__typename === 'MissingPasswordError') {
      throw new Error('Incorrect password')
    }

    if (data.registerCustomerAccount.__typename ===
'NativeAuthStrategyError') {
      throw new Error('Ooops')
    }

    return data.registerCustomerAccount
  }
  return [handler, data] as const
}

export default useRegistration
```

2.3. Висновки до другого розділу

В другому розділі бакалаврської дипломної роботи було розглянуто та описано:

1. Інструменти розробки веб-частини бакалаврської дипломної роботи;
2. Інструменти розробки серверної частини бакалаврської дипломної роботи;
3. СУБД бакалаврської дипломної роботи;
4. Додаткові інструменти розробки бакалаврської дипломної роботи;
5. Збирач веб-частини бакалаврської дипломної роботи.

3 РОЗДІЛ
ОГЛЯД КОДУ ТА ФУНКЦІОНАЛУ ДИПЛОМНОГО ПРОЄКТУ

3.1. Початок роботи. QR-сканер.

Першою сторінкою додатку клієнта, тобто відвідувача, є сканер QR-коду, завдяки якому користувач може закріпити за собою столик або місце в закладі.

Користуватися сканером легко – просто навести камеру пристрою на QR-код в ресторані. В разі читання некоректного коду додаток не перейде на іншу сторінку та покаже помилку користувача.

Привітальна форма зображена на рисунку 3.1. Роботу сканера зображено на рисунку 3.2. Помилку при читанні некоректного коду зображено на рисунку 3.3.

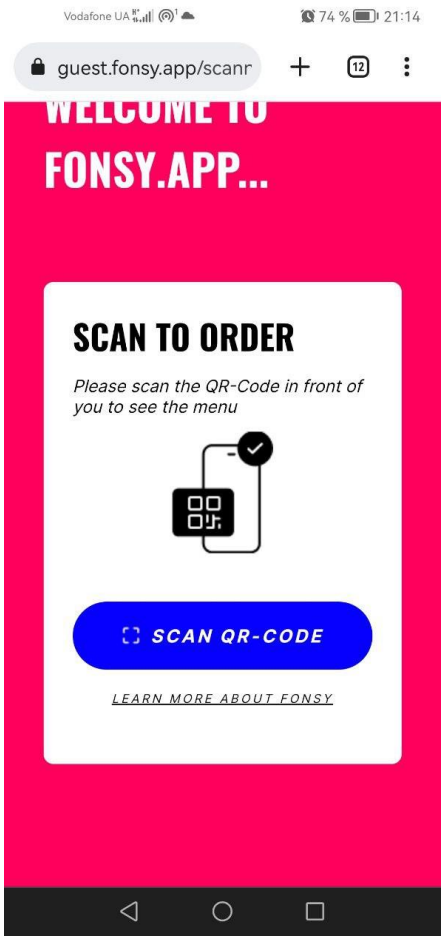


Рис. 3.1 – Стартова форма додатку

Рис. 3.2 – Робота сканеру

Рис. 3.3 – Помилка при читанні невалідного коду

Програмний код сканеру доволі простий – це звичаний компонент, який сканує та розшифровує код, після чого через браузерне API переходить по посиланню на сторінку меню закладу.

Лістинг коду 3.1 – Компонент сканеру

```
const Scanner = ({closeScanner}) => {
  const styles = useStyleSheet(themedStyles)
  const [error, setError] = useState(false)

  const readQRHandler = useCallback(
    async (data) => {
      try {
        const parsedUrl = urlParse(data)
        if (parsedUrl && parsedUrl.tableId && parsedUrl.channel) {
          const newUrl = new URL(window.location.href)
          newUrl.pathname = 'collections/'
          newUrl.searchParams.set('tableId', parsedUrl.tableId)
          newUrl.searchParams.set('channel', parsedUrl.channel)
          window.location.href = newUrl.pathname + newUrl.search
        } else {
          throw 'QR code does not contain required data '
        }
      } catch (e) {
        console.error(e)
        setError(true)
      }
    },
    [setError],
  )

  return (
    <View style={styles.container}>
      <View style={styles.scannerContainer}>
        <QrReader
          constraints={{facingMode: 'environment', height: 200, width: 200}}
          onResult={(result, error) => {
            if (!!result) {
              readQRHandler(result.getText())
            }
            if (!!error) {
              console.info(error)
            }
          }}
        />
      </View>
      <View style={styles.errorContainer}>
```

```
        {error && <Text style={styles.errorText}>restaurant not found</Text>}
      </View>
      <Link onPress={closeScanner} style={styles.link}>
        <Text style={styles.linkText}>Back</Text>
      </Link>
    </View>
  )
}
```

3.2. Меню ресторану та створення замовлення

Після переходу по посилання ю в QR-коді, користувач потряпляє на сторінку меню ресторану, в якому він просканував цей код. Дизайн меню мінімалістичний, представляє собою розбиті по категоріям товари, які можуть мати варіанти. На рисунку 3.4 зображено меню, одна розкрита категорія товарів та варіанти одного з продуктів.

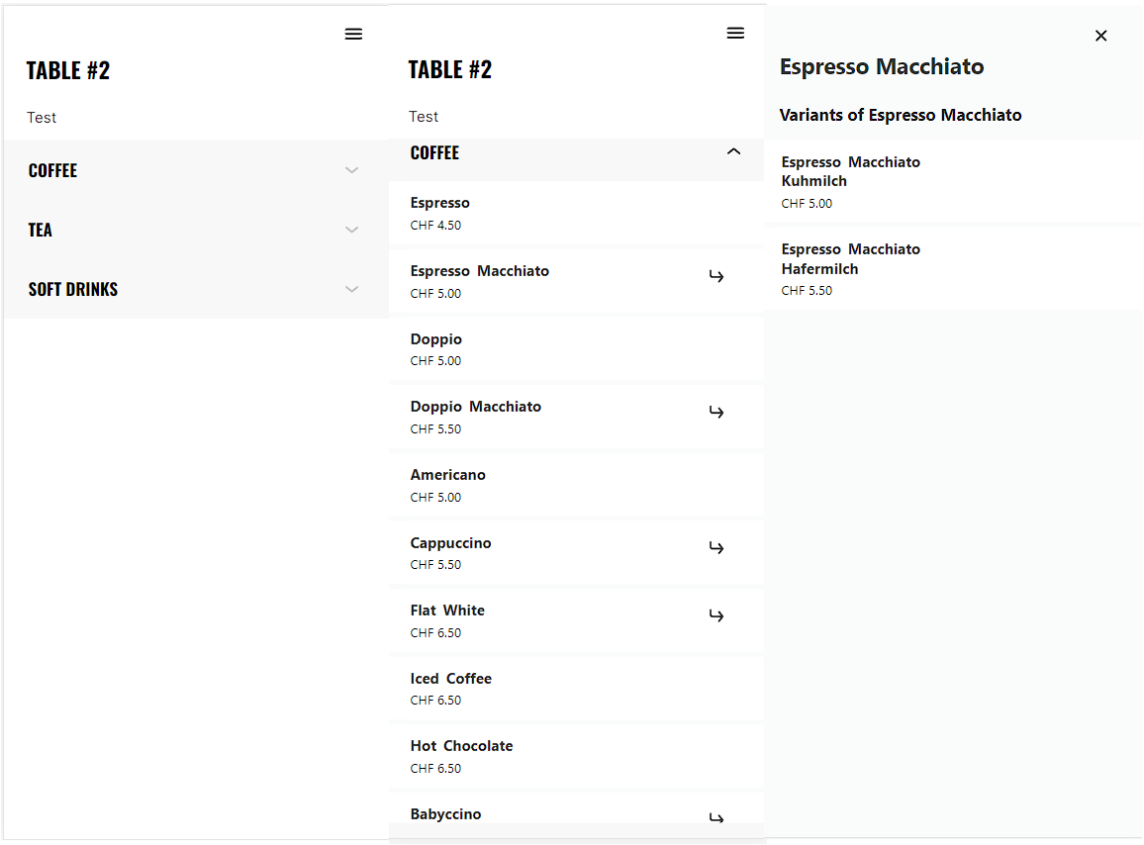


Рис. 3.4 – Меню додатку відвідувача

Створення замовлення відбувається при виборі першого продукту з меню. На рисунку 3.5 зображено вид меню при створеному замовленні.

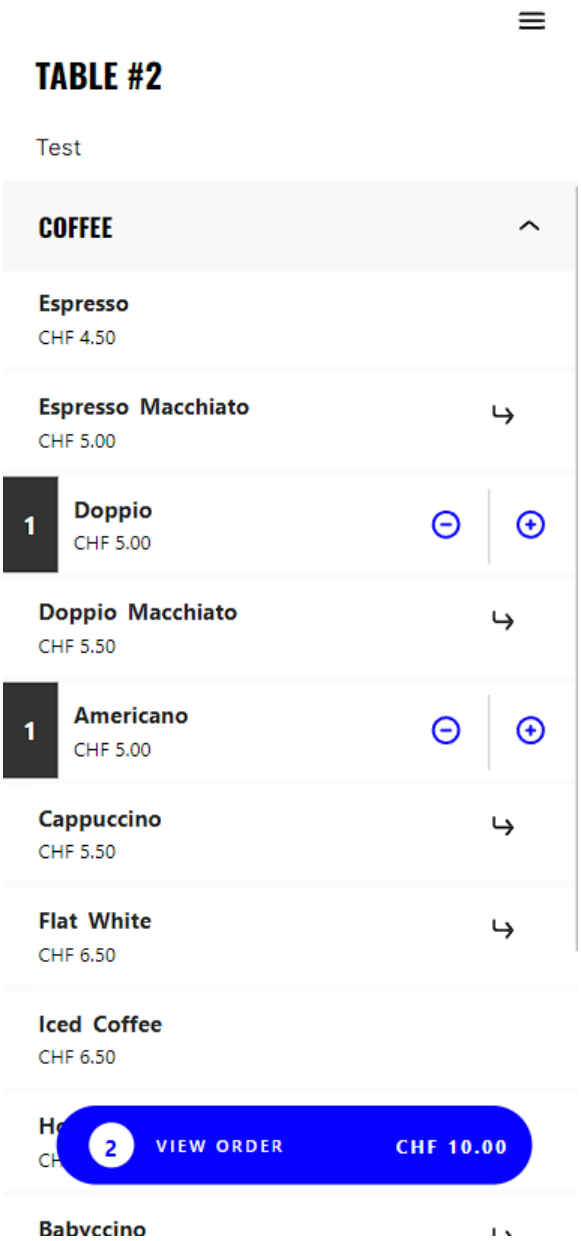


Рис. 3.5 – Меню з замовленням

При видалення всіх товарів з замовлення воно видаляється.

3.2.1. Програмний код меню

Меню складається з 12 компонентів та використовує 7 хуків, зв'язуючих додаток з серверною частиною. Основним компонентом меню є компонент продукту.

Лістинг коду 3.2 – Компонент продукту

```
addItemToOrderMutation({
          variables: {productVariantId, quantity},
        })
      }
      }
    },
```

Даний хук створює або редагує вже створене замовлення.

Мутація, яка приймає та обробляє запит приведена нижче.

Лістинг коду 3.4 – Мутація замовлення

```
    @Mutation()
    @Allow(Permission.UpdateOrder, Permission.Owner)
    async addItemToOrder(
        @Ctx() ctx: RequestContext,
        @Args() args: MutationAddItemToOrderArgs & ActiveOrderArgs,
    ): Promise<ErrorResultUnion<UpdateOrderItemsResult, Order>> {
        const order = await this.activeOrderService.getActiveOrder(
            ctx,
            args[ACTIVE_ORDER_INPUT_FIELD_NAME],
            true,
        );
        return this.orderService.addItemToOrder(
            ctx,
            order.id,
            args.productVariantId,
            args.quantity,
            (args as any).customFields,
        );
    }
```

Дана мутація викликає методи сервісу замовлень та передає в метод аргументи. Сервіс єе викликає інші сервіси, які вже створюють чи редагують замовлення, позиціях в замовленні, його загальну ціну та інші деталі, після чого викликається подія про створення чи редагування замовлення.

3.3. Сторінка замовлення

На сторінці замовлення користувач може редагувати замовлення, вказати ім’я, по якому офіціант зможе звертатись до клієнта та обрати спосіб оплати. Нижче представлено стани сторінки замовлення – стартовий, з помилкою та з заповненим полем імені користувача.

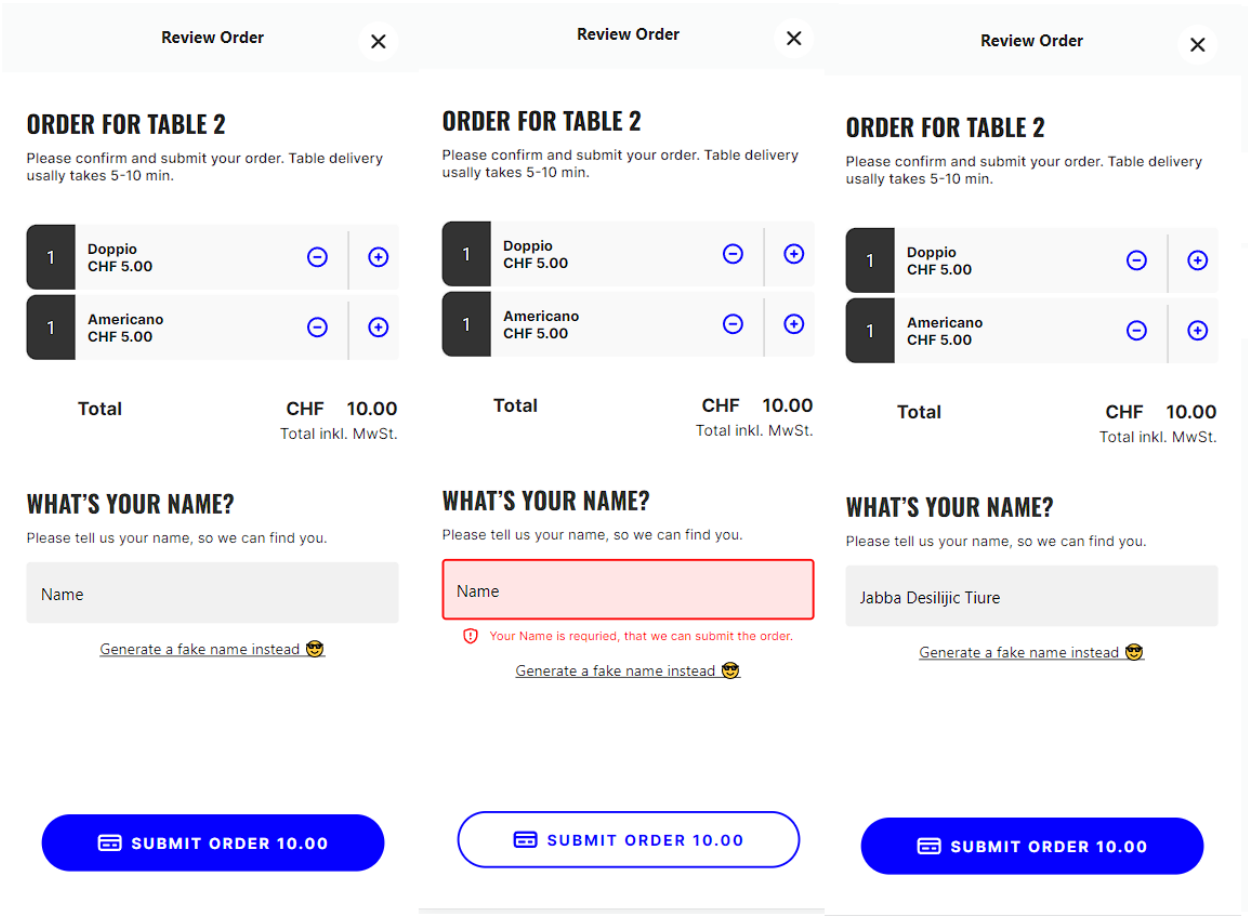


Рис. 3.6 – Сторінка замовлення

3.3.1. Програмний код сторінки замовлення

Сторінка замовлення складається з чотирьох компонентів та використовує два хуки. Основний компонент, ReviewOrder, великий тому нижче буде приведено лиш скорочений фрагмент коду. Даний компонент відображає активне замовлення, яке на даний момент формує користувач.

Лістинг коду 3.5 – Компонент сторінки замовлення

```
const ReviewOrder: FC<IReviewOrder> = ({visible, onClose}) => {

  useEffect(() => {
    getTableId().then(setTableId)
    AsyncStorage.getItem('@client_name').then((clientName) => {
      if (clientName) {
        setIsError(false)
        setFirstVisit(false)
        setName(clientName)
      }
    })
  }, [setFirstVisit])

  useEffect(() => {
    if (id) AsyncStorage.setItem('@last_active_order_id', id)
    if (!lastTotal) {
      setLastTotal(totalWithTax)
    }
  }, [id, lastTotal, totalWithTax])

  }

  //after PLACE ORDER button
  const afterWait = useCallback(() => {
    closeAll()
    setSplash(true)
  }, [closeAll, setSplash])

    const config: Config = {
      dictionaries: [starWars],
    }
    inputChange(uniqueNamesGenerator(config))
  }, [inputChange])

  const confirmedCloseHandler = useCallback(() => {
    onClose()
```

```
    setOrderConfirmed(false)
  }, [onClose, setOrderConfirmed])

  return (
    <Modal visible={visible} animationType={'slide'}>
      <WaitingScreen
        visible={waitOpen}
        placeOrder={placeOrder}
        goBack={closeAll}
        afterWait={afterWait}
      />        <CircleButton
          style={styles.closeButton}
          iconElement={<MyIcon name="close-outline" fill="#333333" />}
          onClick={onClose}
        />
        <View style={styles.titleWrapper}>
          <Text style={styles.title}>Review Order</Text>
        </View>
        <View style={styles.container}>
          <View style={styles.contentTitleWrapper}>
            <Text style={styles.containerTitle}>order for Table {tableId}</Text>
            <Text style={styles.containerSubTitle}>              </View>
              errorText={'Your Name is requried, that we can submit the order.'}
            />
            <Pressable onPress={generateName}>
              <Text style={styles.generateName}>
                Generate a fake name instead 😇
              </Text>
            </Pressable>
          </View>
        </View>
      </ScrollView>
      <View style={[styles.buttonContainer, {bottom: 40}]}>
        <PaymentButton
          text={'Submit order ' + priceTransform(totalWithTax)}
          disabled={errorStatus}
          onClick={completeOrder}
        />
      </View>
    </Modal>
  )
}
```

3.4. Оплата замовлень

Додаток має функціонал оплати декількох замовлень за раз, тому на сторінці оплати потрібно обрати які саме замовлення будуть оплачені. Також тут можна обрати кількість чайових. Сторінка оплати замовлень зображена нижче.

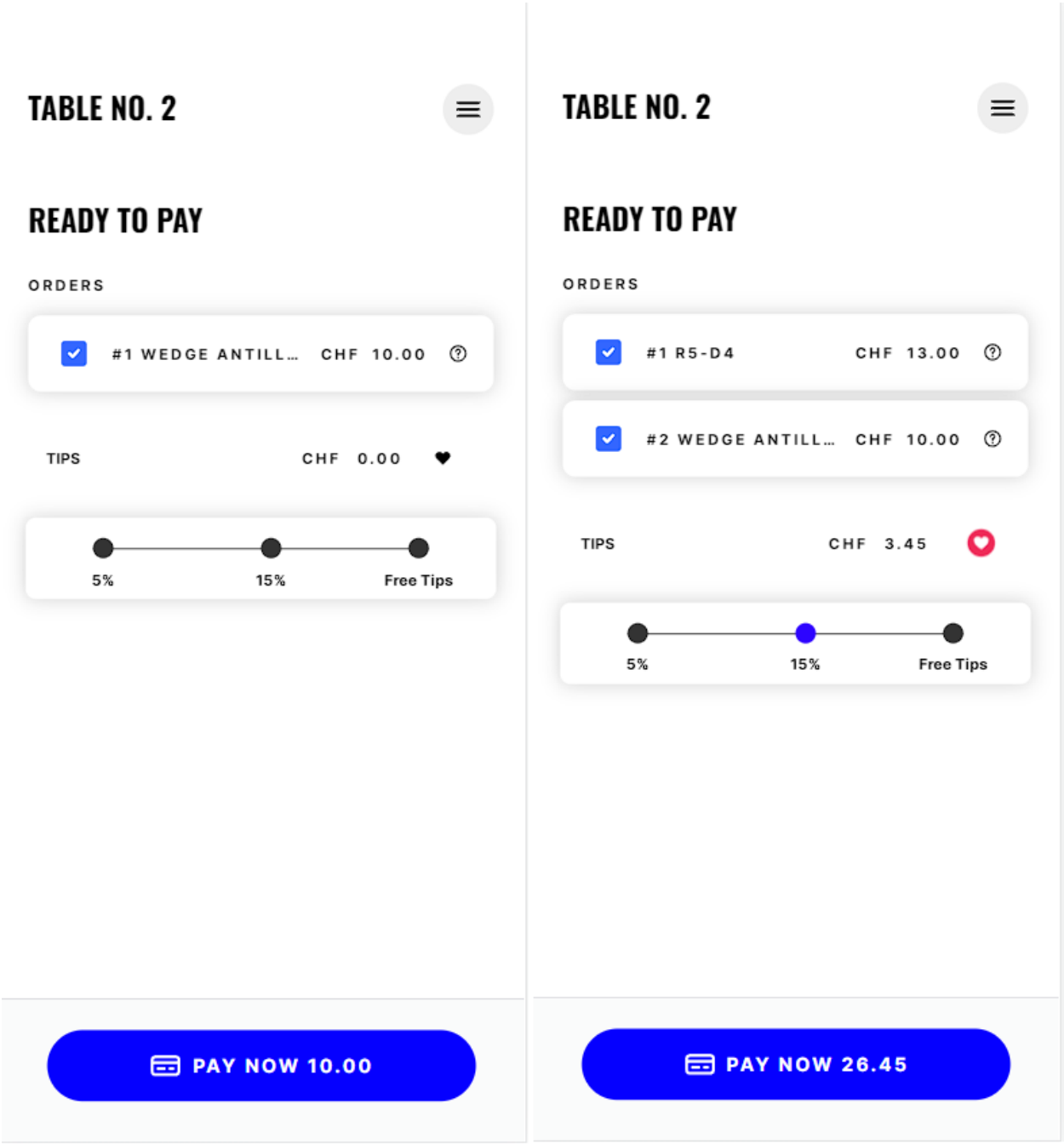


Рис. 3.7 – Сторінка оплати замовлень

На вибір користувача є два способи оплати – готівкою та електронним платежем. При виборі електронного способу потрібно ще обрати метод оплати з наведених. Вибір способу оплати зображено нижче.

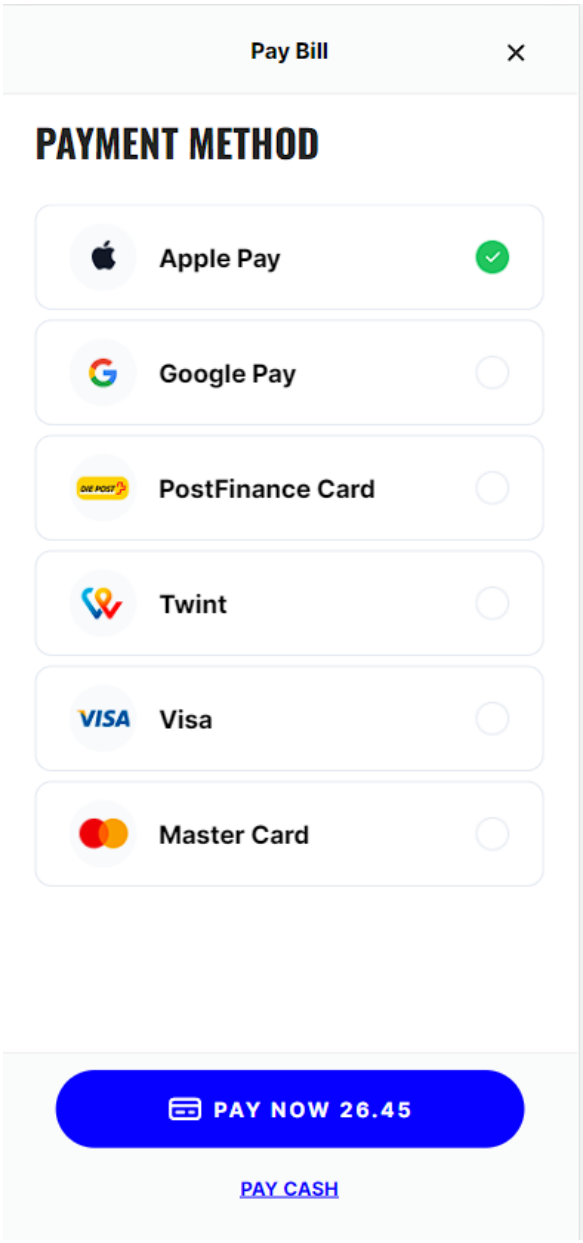


Рис. 3.8 – Вибір способу оплати

При електронному платежі користувач буде переправлений на сторонній сервіс оплати, а саме Payrexx. При успішній оплаті користувач перейде на сторінку успішної оплати, в іншому випадку – на сторінку помилки. Сторінка успішної оплати зображена нижче.

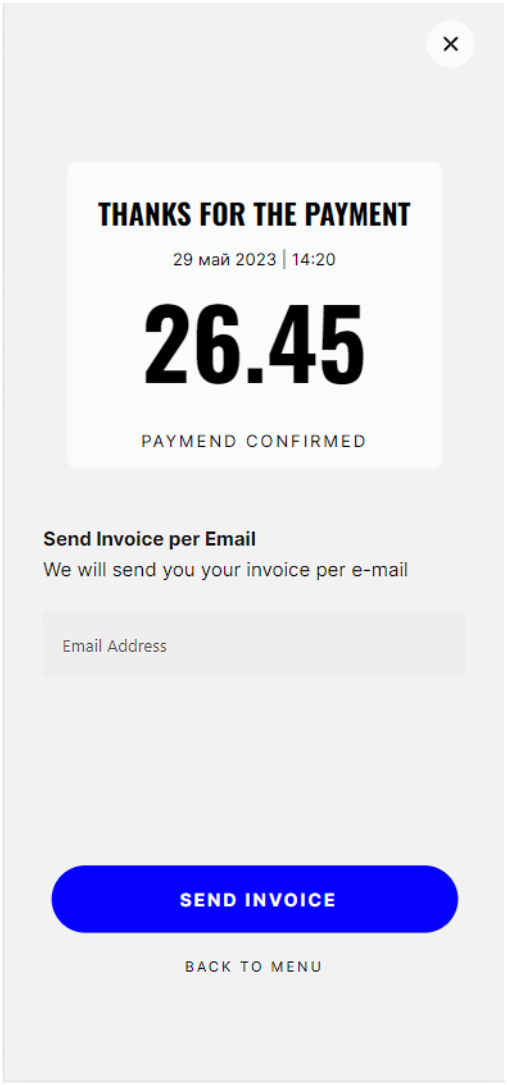
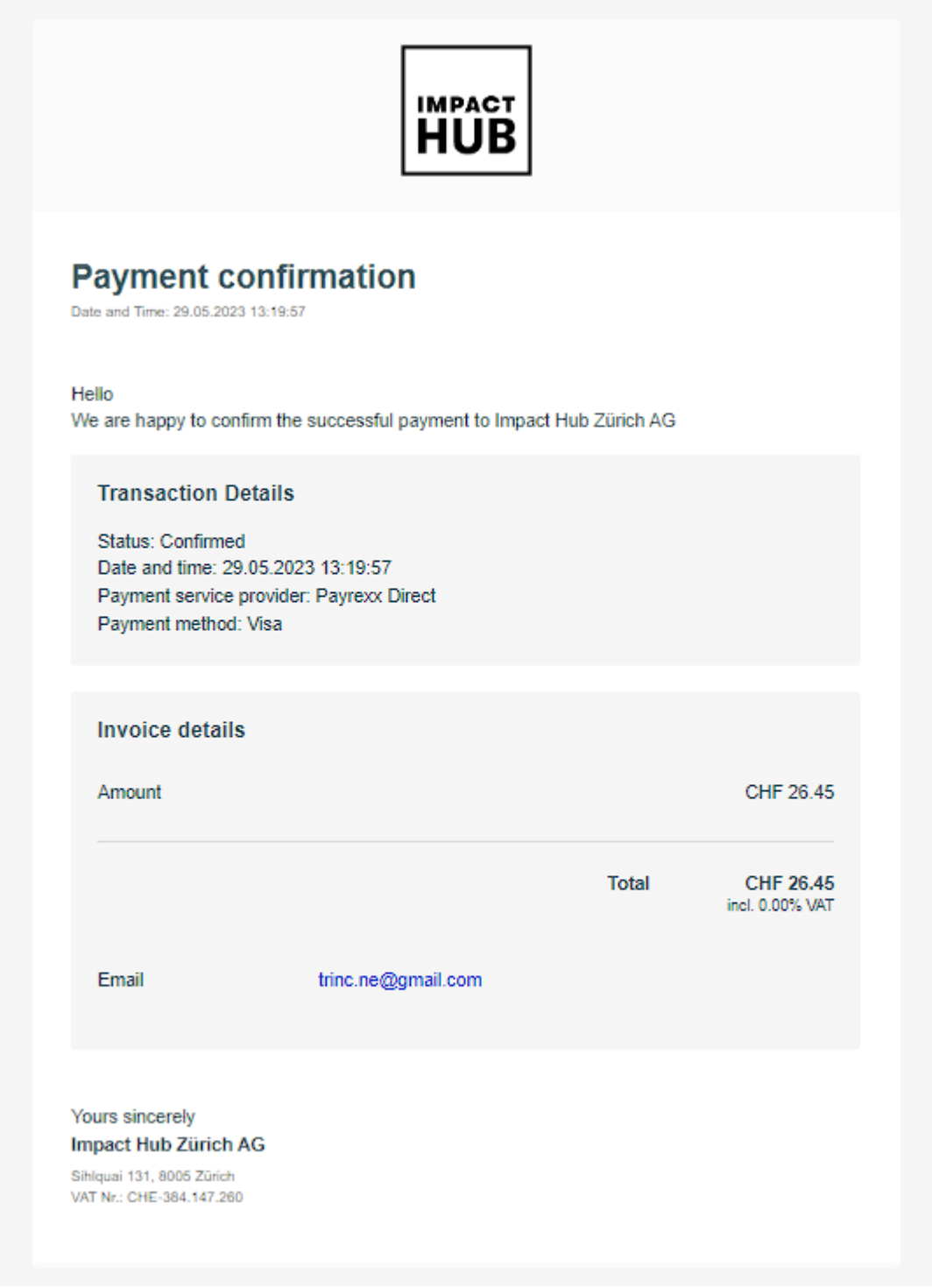


Рис. 3.9 – Сторінка успішної оплати

На сторінці успішної оплати можна відправити чек на введений емєйл. Приклад такого чека наведено нижче.



Payment confirmation

Date and Time: 29.05.2023 13:19:57

Hello
We are happy to confirm the successful payment to Impact Hub Zürich AG

Transaction Details

Status: Confirmed
Date and time: 29.05.2023 13:19:57
Payment service provider: Payrexx Direct
Payment method: Visa

Invoice details

Amount CHF 26.45

Total CHF 26.45
incl. 0.00% VAT

Email trinc.ne@gmail.com

Yours sincerely
Impact Hub Zürich AG
Sihlquai 131, 8005 Zürich
VAT Nr.: CHE-384.147.260

Рис. 3.10 – Чек оплати

3.4.1. Програмний код сторінки оплати замовлень

Сторінка оплати замовлень складається з 24 компонентів та використовує 10 хуків. Основою всієї сторінки є кореневий елемент та контекст, в якому

знаходиться вся важлива для оплати інформація. Код компоненту кореневого елементу та контексту наведено нижче.

Лістинг коду 3.6 – Контекст та кореневий елемент

```
const OrderProvider: FC = ({children}) => {
  const setCheckOrdersIds = (status: boolean, id: string) => {
    const obj = Object.assign({}, checkOrdersIds)
    obj[id] = status
    setIds(obj)
  }

  return (
    <OrderContext.Provider
      value={{
        tips,
        totalPrice,
        setTotalPrice,
        setTips,
        setError,
        error,
        checkOrdersIds,
        setCheckOrdersIds,
        orderCustomerName,
        setOrderCustomerName,
      }}
    >
      {children}
    </OrderContext.Provider>
  )
}

const Orders: FC = () => {
  const styles = useStyleSheet(themedStyles)
  const [tableId, setTableId] = useState('')
  const [sideMenuOpen, setSideMenuOpen] = useState(false)

  useEffect(() => {
    getTableId().then(setTableId)
  }, [])

  return (
    <OrderProvider>
      <SideMenu
        modalVisible={sideMenuOpen}
        onClose={() => {
          setSideMenuOpen(false)
        }}
```

```
      />
      <View style={[styles.wrapper, {height: windowHeight}]}>
        <ScrollView>
          <View style={styles.headerConatiner}>
            <Text style={styles.header}>Table No. {tableId ?? 1}</Text>
            <CircleButton
              onClick={() => {
                setSideMenuOpen(true)
              }}
              style={styles.menuButton}
              iconElement={
                <Icon style={styles.icon} name="menu-outline" fill="#000000" />
              }
            />
          </View>

          <Text style={styles.subHeader}>Ready to Pay</Text>
          <OrdersList />
          <Tips />
        </ScrollView>
        <Buttons />
      </View>
    </OrderProvider>
  )
}
```

Серверна частина, яка відповідає за оплату, код якої представлено в додатку А.

Даний фрагмент виконує функцію setPaymentAfterSuccess. Ця функція виконує дії, пов'язані з обробкою платежу після успішної оплати замовлення.

Спочатку функція перевіряє, чи передані всі необхідні параметри, такі як metadata. Якщо metadata відсутній, викидається виключення з повідомленням "Payment is undefined". Якщо metadata присутній, він розбирається з формату JSON за допомогою JSON.parse.

Далі функція викликає метод payrexxApiCreate для створення об'єкту payrexx, який представляє інтеграцію з платіжною системою. Потім виконується запит до payrexx.retrieveGateway з використанням metadataObj для отримання інформації про платіжний шлюз.

Якщо отриманий об'єкт gateway не має властивості data, викидається виключення з повідомленням "Error: Payment is invalid". Якщо статус платежу (gateway.data.status) не є "confirmed", викидається виключення з повідомленням "Error: Payment is unpaid".

Далі визначаються змінні notificationParams і createdTip. Якщо переданий параметр tip має значення, викликається метод this.tipService.create для створення об'єкту createdTip з вказаним значенням tip.

Далі замовлення отримуються з бази даних за допомогою this.connection.getRepository(ctx, Order).findByIds(orderIdsList). Якщо замовлення не знайдено або їх довжина дорівнює нулю, викидається виключення з повідомленням "Error: Order is undefined".

Розраховується загальна сума замовлень totalAmount за допомогою методу reduce для підсумовування значень totalWithTax кожного замовлення.

Далі виконується пошук платіжного методу (paymentMethodService.findAll) за вказаним кодом method. Якщо метод не знайдено, викидається виключення з повідомленням "Error: Method is undefined".

Потім виконується цикл for, який обробляє кожне замовлення. Якщо замовлення не пусте і його стан не є "PayLater" або "ArrangingPayment", викидається виключення з повідомленням "Error: This order payed".

Значення параметра tableId встановлюється відповідно до поля tableId в кастомних полях замовлення. Далі виконується створення або оновлення клієнта (customer) з даними "guest". Якщо виникає конфлікт електронної адреси, викидається виключення з повідомленням "Error: Email Address Conflict". Якщо клієнт успішно створений або оновлений, викликається метод this.orderService.addCustomerToOrder для додавання клієнта до замовлення.

Якщо передано значення tipNumber, викликається метод this.orderService.updateCustomFields для оновлення поля tippId замовлення.

Потім викликаються методи this.orderService.transitionToState для зміни стану замовлення на "ArrangingPayment" і this.paymentService.createPayment для створення платежу. Отриманий платіж додається до замовлення, яке оновлюється

в базі даних за допомогою this.connection.getRepository(ctx, Order).save. Після цього замовлення переходить до стану "PaymentSettled".

На останок, функція перевіряє, чи totalAmount більше нуля, і якщо так, викликає метод this.notificationService.paymentSuccessNotification для сповіщення про успішну оплату.

Якщо під час виконання будь-якого кроку виникає помилка, викидається виключення з повідомленням про помилку.

Для відправки електронних листів використовується MailgunAPI. Шаблон написаний на шаблонізаторі Handlebars. В шаблоні листа використовується набір хелперів, який дозволяє рендерити в листі товари та замовлення. Код методу обробки шаблону та сам шаблон приведено в додатку Б. Основна функція getInvoiceTemplate є асинхронною і отримує об'єкт vars в якості параметра.

У цьому фрагменті спочатку викликається функція registerHelper бібліотеки Handlebars для створення двох допоміжних функцій: ordersRender і orderProductsRender. Ці функції використовуються для відображення даних про замовлення та продукти в шаблоні рахунку.

Функція ordersRender приймає рядок ordersString, який містить JSON-подібні дані про замовлення. Вона розбирає цей рядок у масив об'єктів orders і для кожного об'єкта виконує певні дії. Значення замовлення (такі як id, orderCustomerName, orderTotal і products) передаються до блока шаблону block.data. Після цього блок шаблону обробляється за допомогою block.fn(this), а результат додається до змінної accum. На кінці циклу функція повертає накопичений результат.

Функція orderProductsRender працює подібним чином. Вона приймає рядок productsString, який містить JSON-подібні дані про продукти в замовленні. Рядок розбирається у масив об'єктів lines, і для кожного об'єкта значення (наприклад, name, quantity і linePrice) передаються до блока шаблону block.data. Потім блок шаблону обробляється, результат додається до змінної accum, і на кінці циклу накопичений результат повертається.

У кінці функції компілюється шаблон emailTemplate за допомогою handlebars.compile, і він застосовується до об'єкта vars. Результат компіляції шаблону повертається як вихід з функції getInvoiceTemplate.

3.4.2. Життєвий цикл замовлення

Створення та оплата замовлення є частинами його життєвого циклу. Схема життєвого циклу замовлення зображена нижче.

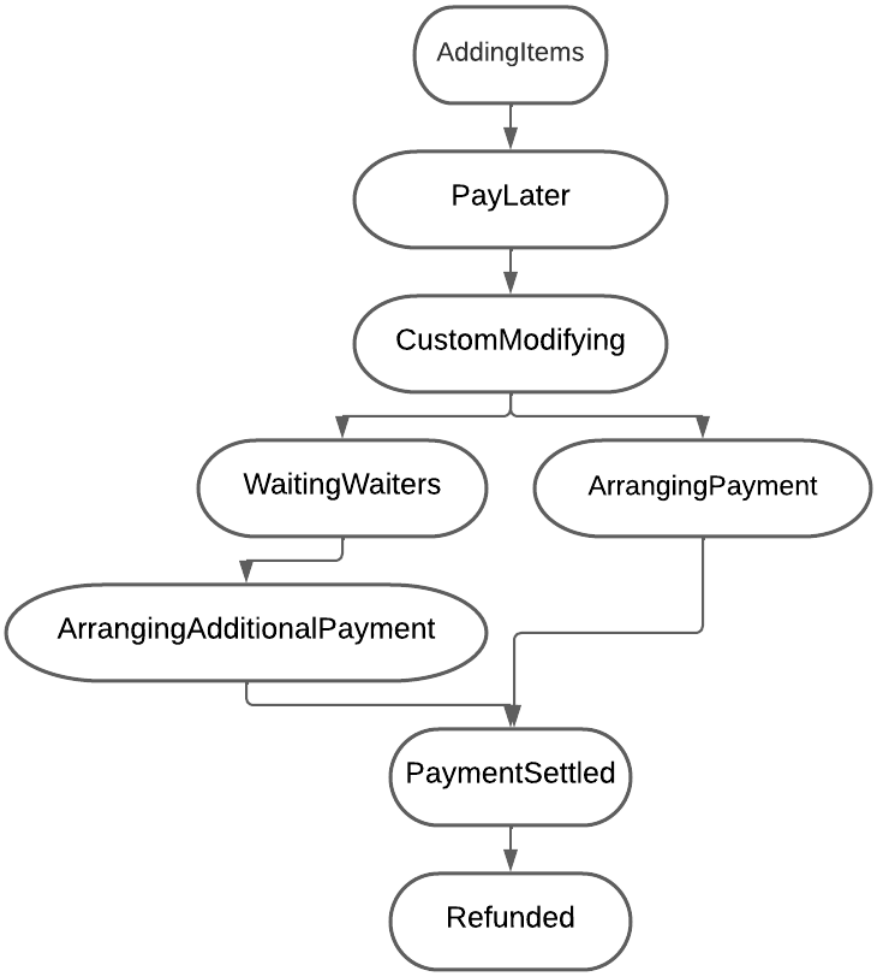


Рис. 3.11 – Життєвий цикл замовлення

1) AddingItems – момент створення замовлення. На даному етапі користувач має можливість змінювати товари в замовленні;

2) PayLater – замовлення відправлене офіціанту. На даному етапі користувач не має маожливості змінювати товари в замовленні;
3) CustomModifying – замовлення редагується. Офіціант може перевести замовлення в цей стан та змінити його за проханням клієнта;
4) WaitingWaiters – замовлення чекає підтвердження офіціантом оплати готівкою;
5) ArrangingAdditionalPayment – оплата готівкою;
6) ArrangingPayment – оплата електронним платежем в процесі. В даний стан замовлення переходить при ініціалізації процесу онлайн оплати. В разі невдачі замовлення повертається в стан PayLater;
7) PaymentSettled – Замовлення оплачене та ніяким модифікаціям не підлягає;
8) Refunded – в даний стан замовлення переходить в разі повернення грошей клієнту.

3.5. Виклик офіціанта

У клієнта є можливість викликати офіціанта до столику. Кнопка для виклику офіціанта знаходиться в додатковому меню додатку. Воно зображене нижче.

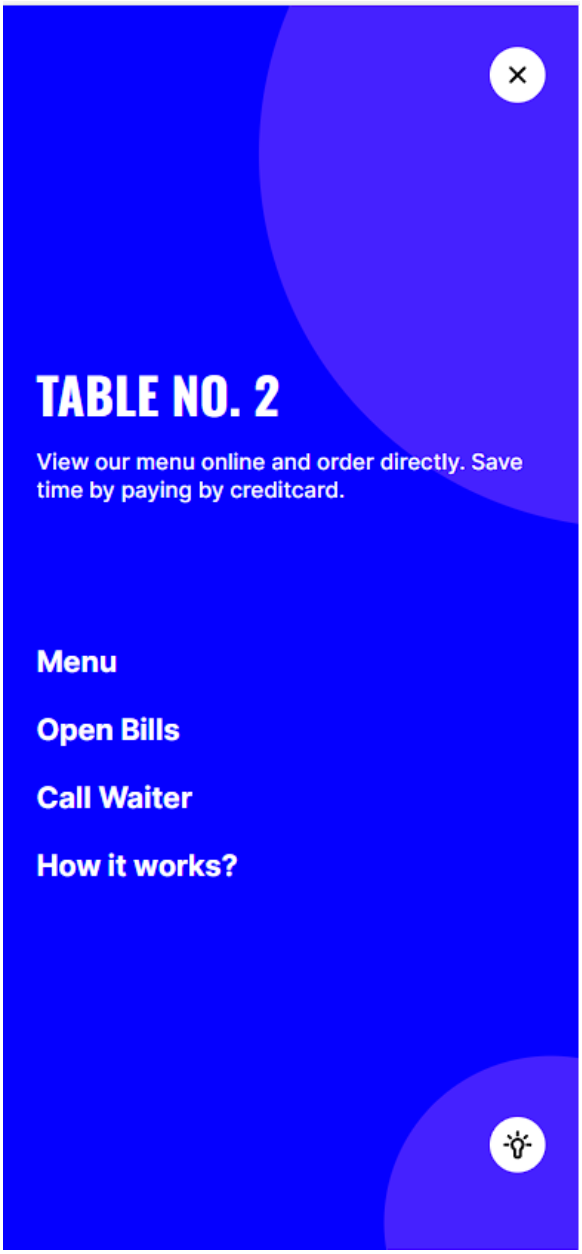


Рис. 3.12 – Додаткове меню

При натисканні на кнопку Call Waiter офіціанту прийде повідомлення що хтось його викликає. Код створення події виклику офіціанта в серверній частині додатку наведено в додатку В до звіту.

Даний фрагмент коду містить функцію createCallWaiterAction. Ця функція виконує дії, пов'язані з створенням дії "виклик офіціанта" у системі.

Спочатку виконується отримання токена з заголовків запиту. Заголовок з токеном зберігається в змінній ctxToken. Якщо значення ctxToken є масивом, то

токен визначається як перший елемент цього масиву, інакше токен визначається як єдиний елемент ctxToken.

Далі отримується channelId за допомогою методу getChannelFromToken сервісу channelService з використанням отриманого токена.

Потім виконується перевірка наявності значення tableId в заголовках запиту. Якщо tableId відсутній, викидається помилка з повідомленням "TableId is not defined".

Після цього виконується створення нового запису в базі даних з вказаними параметрами, такими як tableId, status, visible, timeStamp і channelId. Для цього викликається метод create з action-сервісу.

Якщо всі дії виконані успішно, функція повертає рядок "Success". У разі виникнення помилки, функція повертає рядок "Error".

3.6. Додаток офіціанта. Вхід

Форма входу офіціанта проста та зручна. Немає сенсу робити їй унікальний дизайн бо окрім співробітників закладу ніхто її не побачить. Форма входу зображена на рисунку 3.13.

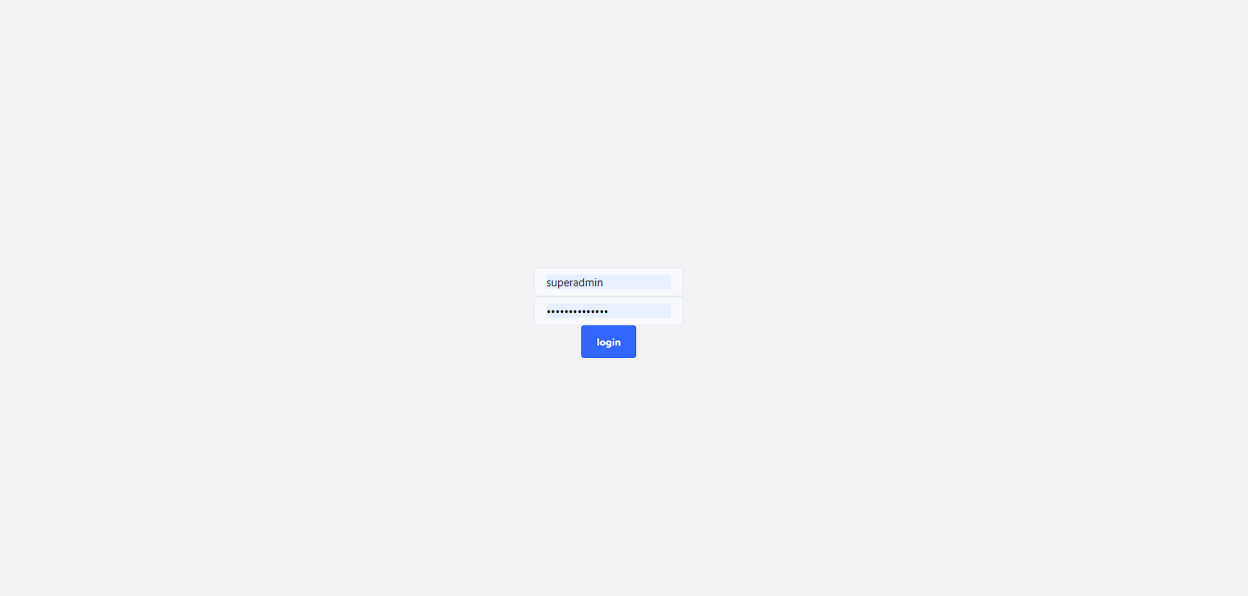


Рис. 3.13 – Форма входу до додатку офіціанта

Після аутентифікації офіціант переходить до сторінки подій.

3.7. Сторінка подій

Система подій є одною з особливостей додатку офіціанта. Завдяки даному функціоналу офіціант може дізнаватись де його чекають на оплату, на редагування замовлення чи просто викликають до столу. Сторінка подій зображена нижче.

Рис. 3.14 – Сторінка подій

При натисканні на подію відкривається форма прив’язаного до неї замовлення. В разі натискання на подію виклику офіціанта буде показане невелике спливаюче вікно. Форма і вікно зображені на наведеному нижче рисунку.

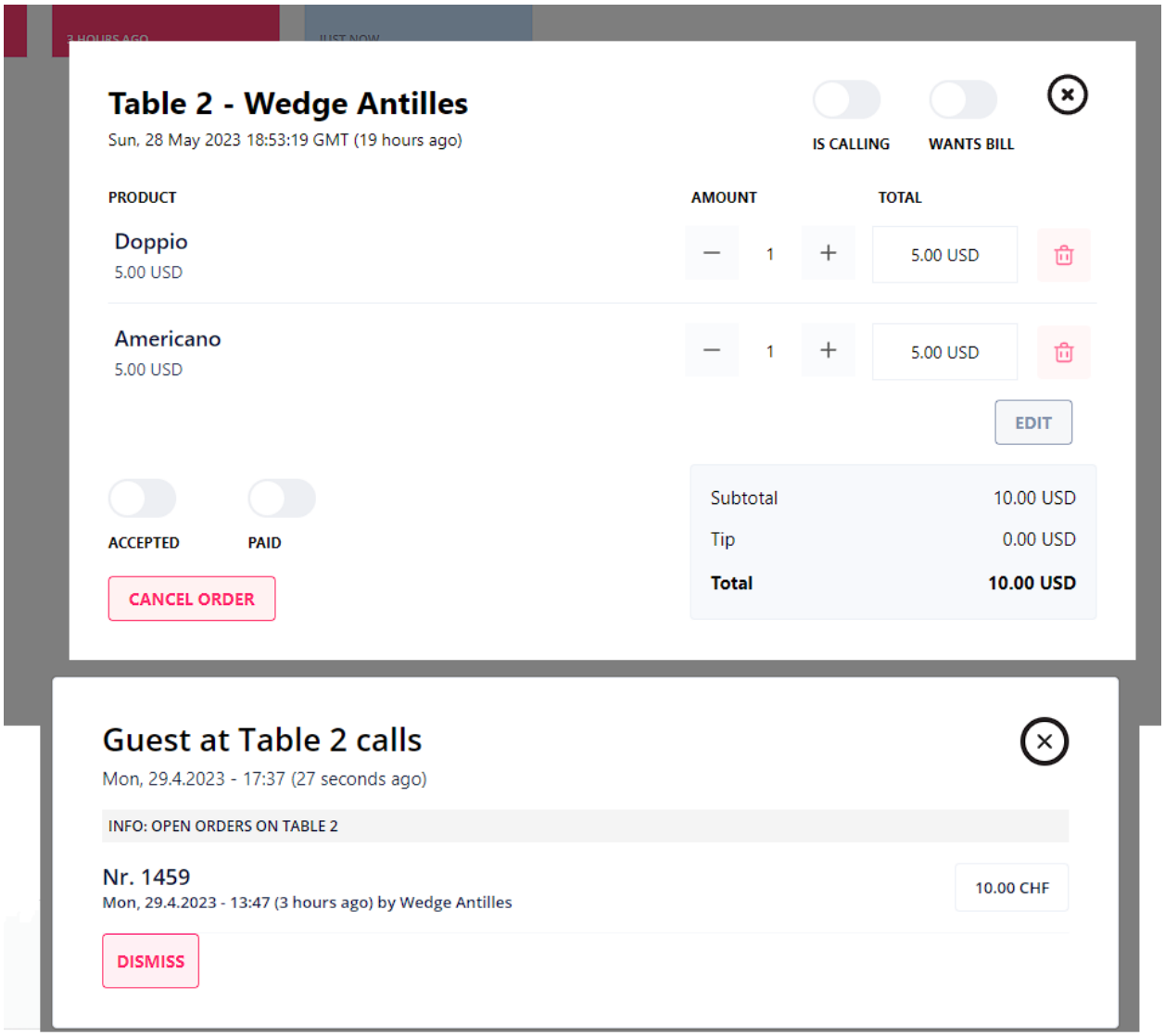


Рис. 3.15 – Вікно виклику та форма замовлення

3.7.1. Програмна частина функціоналу подій

Сторінка подій складається з шести компонентів та використовує чотири хуки.

В серверній частині додатку основою системи подій є функція підписання на зміни стану замовлень - subscribeToOrder. Дана функція наведена в додатку Г.

subscribeToOrder підписується на подію OrderStateTransitionEvent в екземплярі eventBus.

При отриманні події, функція перевіряє, чи новий стан замовлення не є "AddingItems" і чи існує замовлення з відповідним ідентифікатором. Якщо ці умови не виконуються, функція повертається без виконання подальших дій.

Потім отримується поточний час у вигляді рядка. Далі отримується channelId за допомогою методу getChannelByOrderId, використовуючи ідентифікатор замовлення.

Після цього виконується перевірка значення event.toState у внутрішньому switch-блоку. В залежності від значення event.toState виконуються різні дії:

- Якщо event.toState дорівнює "PayLater", перевіряється наявність дії замовлення з типом "PayLater" за допомогою методу getActionByOrderId. Якщо дія не знайдена, створюється нова дія з параметрами, включаючи tableId, orderId, status, visible, timeStamp та channelId;
- Якщо event.toState дорівнює "PaymentSettled", спочатку перевіряється наявність дії замовлення з типом "WaitingWaiters". Якщо дія не знайдена, викидається помилка "Action not found". Потім оновлюється видимість дії на false за допомогою методу update. Якщо event.fromState не дорівнює "ArrangingAdditionalPayment", створюється нова дія з параметрами, аналогічними попередньому випадку;
- Якщо event.toState дорівнює "WaitingWaiters", створюється нова дія з параметрами, аналогічними попереднім випадкам;
- Якщо event.toState дорівнює "Cancelled", спочатку перевіряється наявність дії замовлення з типом "WaitingWaiters". Якщо дія не знайдена, викидається помилка "Action not found". Потім оновлюється видимість дії на false за допомогою методу update;
- Якщо значення event.toState не відповідає жодному з вказаних випадків, функція повертається без виконання подальших дій.

У разі виникнення помилки в будь-якому з випадків, викидається помилка з повідомленням, яке містить початок "IN ACTIONS: " та повідомлення про помилку.

Створення події виклику офіціанта проходить після запиту на створення даної події.

3.8. Сторінка статистики

У офіціанта є доступ до статистики. Сторінка статистики зображена нижче.

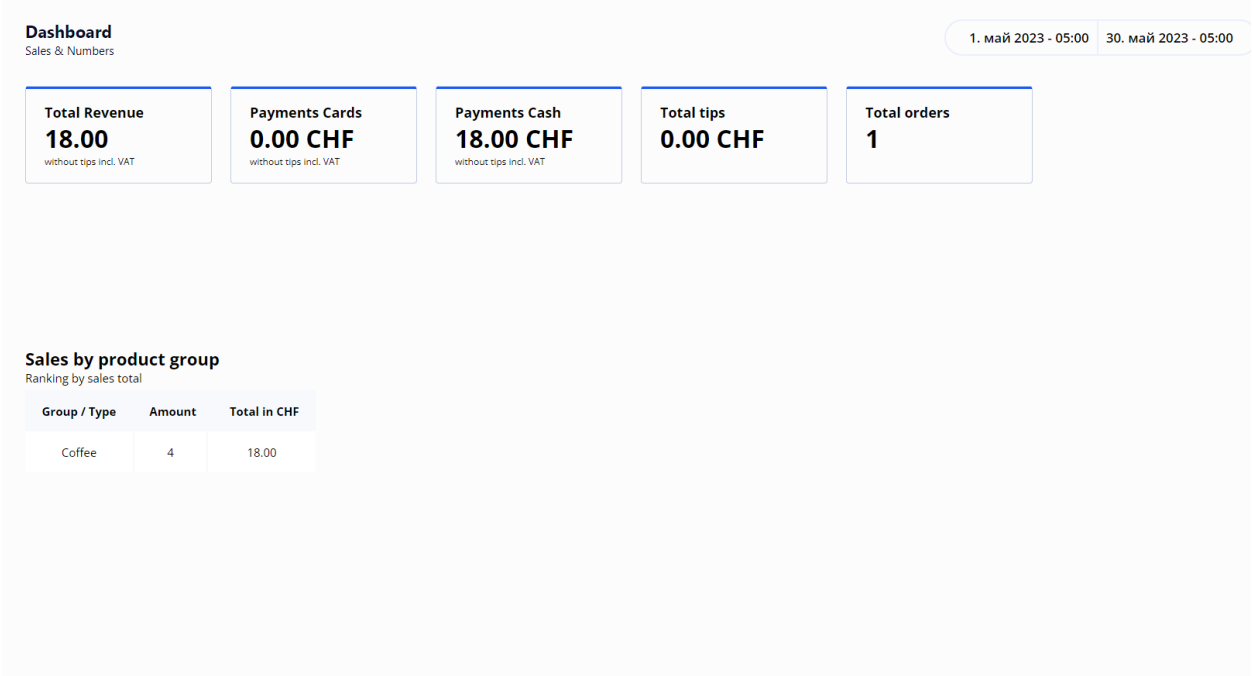


Рис. 3.16 - Сторінка статистики

Тут офіціант може обрати період, за який хоче подивитись статистику продажів. У відображенні сторінкb статистики рендеряться п'ять карточок (DashboardCard). Кожна карточка має свій заголовок (title), підзаголовок (subContent) та вміст (content), які використовують відповідні значення з formattedData для відображення загального доходу (Total Revenue), платежів картами (Payments Cards), платежів готівкою (Payments Cash), загальних чайових (Total tips) та загальної кількості замовлень (Total orders).

Нижче рендериться таблиця продажів по категоріям.

3.8.1. Серверна частина сторінки статистики

Для отримання даних використовується дві функціях – отримання даних для карточок та отримання даних до таблиці продаж. Код представлено в додатку Д.

Отримання даних для карточок.

Спочатку у функції змінна data отримує дані замовлень (orders), які мають статус "PaymentSettled" (оплачені замовлення). У разі, коли початкова дата та кінцева дата dataRange співпадають, вибираються всі оплачені замовлення. У протилежному випадку, вибираються оплачені замовлення, які були створені в межах вказаного діапазону дат.

Далі у функції отримуються дані про чайові (tips) за вказаний діапазон дат.

У циклі перебираються отримані замовлення (data.items) і обчислюються сумарні значення для загального доходу (totalRevenue), суми оплати карткою (totalCardsAmount), суми оплати готівкою (totalCashAmount) та загальних чайових (totalTips). Залежно від способу оплати, відповідна сума додається до відповідної змінної.

На основі обчислених значень створюється об'єкт, який містить дані для відображення в карточках у панелі керування. Повертається цей об'єкт як результат виконання функції.

У разі виникнення помилки під час виконання функції, виводиться повідомлення про помилку у консоль, і функція повертає помилку.

У функції використовується ctx (RequestContext) для отримання даних з контексту. Функція отримує також dataRange (діапазон дат), який використовується для фільтрації даних.

Отримання даних до таблиці продаж

Спочатку у функції змінна data отримує дані замовлень (orders), які мають статус "PaymentSettled" (оплачені замовлення). У разі, коли початкова дата та кінцева дата dataRange співпадають, вибираються всі оплачені замовлення. У протилежному випадку, вибираються оплачені замовлення, які були створені в межах вказаного діапазону дат.

Далі у змінну allOrdersLines збираються всі рядки замовлень з отриманих даних. Це дозволяє отримати загальний список продуктів у всіх замовленнях.

У циклі перебираються всі рядки замовлень (allOrdersLines). Для кожного рядка отримується повний варіант продукту (fullVariant) з використанням його ідентифікатора. Якщо повний варіант не знайдено, генерується помилка.

Далі перевіряється, чи існує вже запис для цього продукту у зібраних даних (compiledData). Якщо запису не існує, створюється новий запис з даними про продукт, його кількість (amount), загальну ціну (total) та ідентифікатор колекції (collectionId). Якщо запис вже існує, оновлюються дані про кількість та загальну ціну для відповідного запису.

Далі отримуються всі колекції (collections).

У циклі перебираються всі колекції, окрім кореневої колекції. Для кожної колекції перевіряється, чи є валідні (відповідні до колекції) продукти у зібраних даних. Якщо є валідні продукти, створюється об'єкт collectionSales з ім'ям колекції, загальною кількістю (amount) та загальною сумою (total). Потім для кожного валідного продукту обновлюються значення кількості та загальної суми в об'єкті collectionSales. Згенерований об'єкт collectionSales додається до масиву collectionsSales.

На кінці функція повертає масив collectionsSales, який містить дані про продажі по групах товарів.

3.9. Сторінка замовлень

На сторінці замовлень знаходиться таблиця замовлень, в якій згідно заданим фільтрам відображаються замовлення. Сторінка представлена нижче.

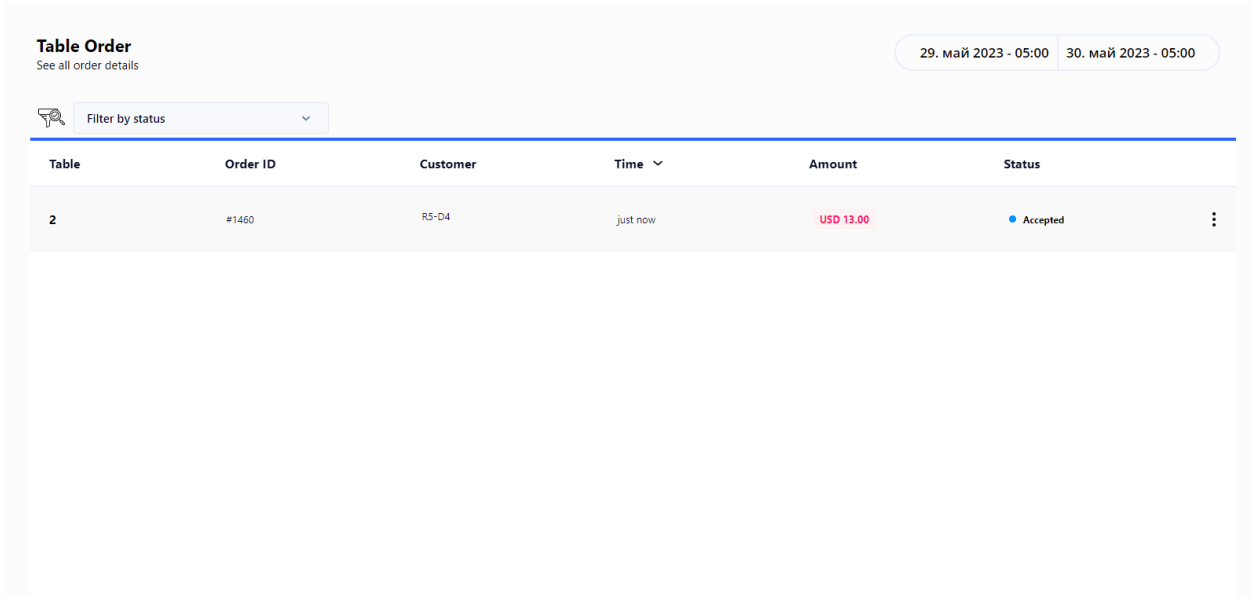


Рис. 3.17 – Сторінка замовлень

Сторінка підтримує фільтрацію за часом створення замовлення та за його статусом

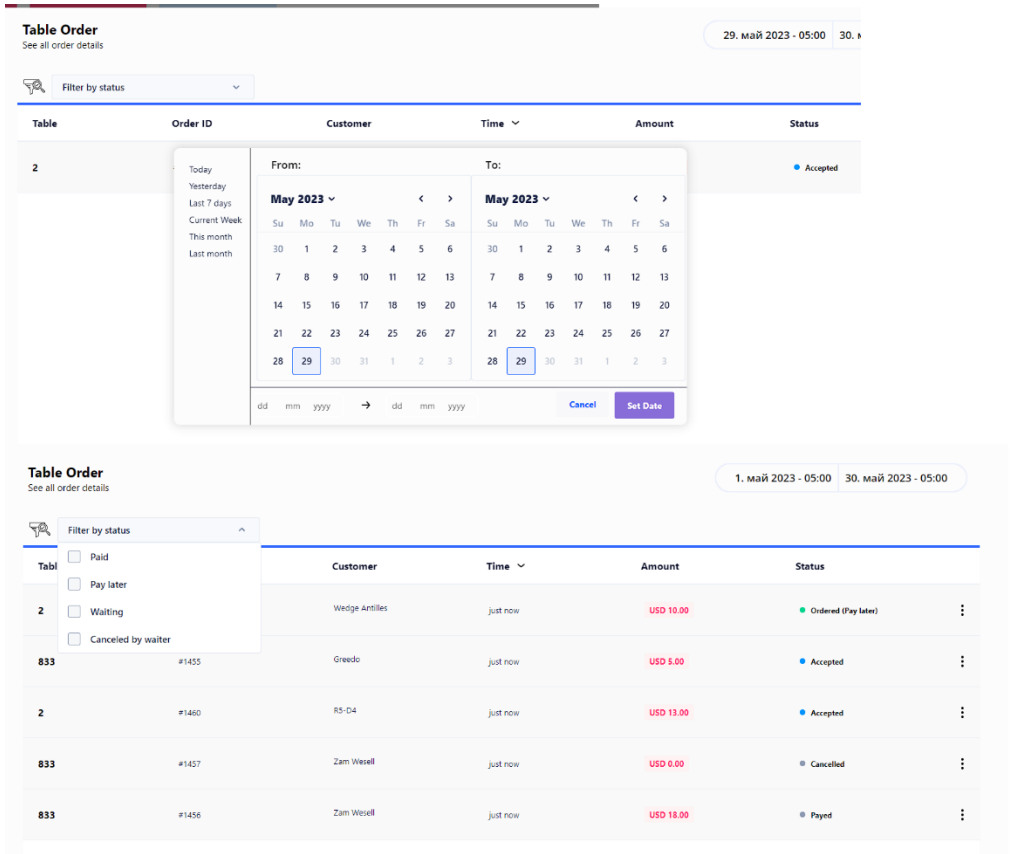


Рис. 3.18 – Фільтрація таблиці замовлень

Також доступне сортування по значенню колонок.

3.9.1. Серверна частина роботи з замовленнями

Отримання замовлень не має особливостей та нічим не відрізняється від отримання подій. Тому в даному підрозділі буде розглянуто робота з замовленнями, а саме оплата готівкою, модифікування замовлення та повернення коштів.

Повернення коштів.

У функції передаються контекст ctx (RequestContext) та ідентифікатор замовлення orderId.

У тілі функції спочатку отримується замовлення за його ідентифікатором, використовуючи сервіс orderService. Якщо замовлення не знайдено, генерується помилка "Order not found".

Далі отримується екземпляр платіжного шлюзу (gateway) з використанням сервісу payrexxApiCreate. Поле orderPaymentGateway в полі customFields замовлення містить дані про платіжний шлюз, які розпаковуються у змінну gatewayData. Використовуючи payrexx.retrieveGateway, отримується інформація про шлюз.

Далі виконується повернення платежу (refundPayment) за допомогою payrexx.refundPayment. У цьому випадку використовується перший транзакційний ідентифікатор з інформації про шлюз, а також сума платежу замовлення з податками (order.totalWithTax).

Після успішного повернення платежу оновлюється статус замовлення на "Refunded" за допомогою orderService.transitionToState.

На кінці функція повертає оновлене замовлення (updatedOrder).

У разі виникнення помилки під час виконання функції, генерується новий об'єкт помилки з повідомленням про помилку, який потім може бути перехоплений та оброблений вищим рівнем коду.

Модифікування замовлення.

Асинхронна функція modifyPayLaterOrder, яка здійснює модифікацію замовлення, що має статус "PayLater".

У функцію передаються контекст ctx (RequestContext), ідентифікатор замовлення id і масив змін changes (Array<IChange>), які потрібно внести до замовлення.

У тілі функції спочатку отримується замовлення за його ідентифікатором, використовуючи метод getOrderById, який повертає перше знайдене замовлення. Якщо замовлення не знайдено, генерується помилка "Order not found".

Далі виконується перехід замовлення до стану "AddingItems" за допомогою методу transitionToState сервісу orderService.

Потім проходить ітерація по змінам (changes), де кожна зміна розглядається окремо. Якщо кількість (quantity) зміни більше або дорівнює 1, викликається метод addItemToOrder сервісу orderService, щоб додати елемент до замовлення.

У разі, якщо кількість (quantity) зміни менше 1, спочатку отримується варіант товару (productVariant) за його ідентифікатором. Якщо варіант не знайдено, генерується помилка "Product variant not found".

Далі отримується лінія замовлення (line), використовуючи метод getRepository сервісу connection, який виконує запит до бази даних з певними обмеженнями. Якщо лінію не знайдено, генерується помилка "Order line not found".

Обчислюється загальна кількість товарів (itemsQuantity) у лінії замовлення, які не були скасовані.

Далі виконується коригування кількості товарів (adjustQuantity) у лінії замовлення з урахуванням зміни. Якщо після коригування кількість стає рівною 0, викликається метод removeItemFromOrder сервісу orderService для видалення елемента з замовлення. Якщо кількість стає більшою або дорівнює 1, викликається метод adjustOrderLine сервісу orderService для коригування лінії замовлення. У протилежному випадку генерується помилка "Invalid quantity".

Після проходження всіх змін повертається замовлення (order).

У разі виникнення помилки під час виконання функції, вона виводиться в консоль за допомогою console.log, а потім генерується новий об'єкт помилки з повідомленням про помилку, який може бути перехоплений та оброблений вищим рівнем коду.

В кінці, незалежно від того, чи виникла помилка, виконується перехід замовлення до стану "PayLater" за допомогою методу transitionToState сервісу orderService.

Оплата готівкою.

Функція addCashManualPayment, яка додає ручний спосіб оплати замовлення.

У функцію передаються контекст ctx (RequestContext) та ідентифікатор замовлення orderId.

У тілі функції спочатку виконується перехід замовлення до стану "ArrangingAdditionalPayment" за допомогою методу transitionToState сервісу orderService. Цей перехід означає, що замовлення потребує додаткової оплати.

Потім викликається метод addManualPaymentToOrder сервісу orderService, який додає ручний спосіб оплати до замовлення. У даному випадку, передається об'єкт з параметрами оплати, включаючи ідентифікатор замовлення (orderId), метод оплати (method) - 'manual-payment' і метадані (metadata), які можуть бути порожніми об'єктом {}.

Після цього виконується другий перехід замовлення до стану "PaymentSettled" за допомогою методу transitionToState сервісу orderService. Цей перехід означає, що оплата замовлення була виконана.

Якщо виникає помилка під час виконання будь-якого з кроків, вона виводиться в консоль за допомогою console.log.

У кінці функція повертає результат третього переходу до стану "PaymentSettled" за допомогою return await this.orderService.transitionToState(ctx, orderId, 'PaymentSettled').

3.10. Сторінка чайових

На сторінці чайових офіціант може побачити список чайових. Таблиця чайових зображена нижче.

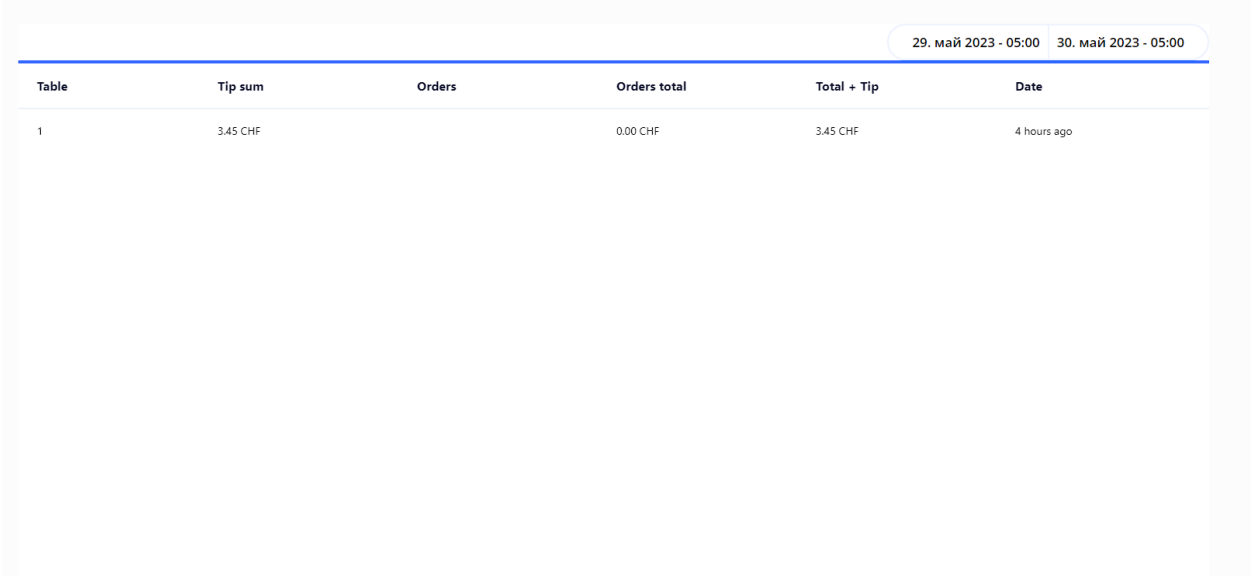


Рис. 3.19 – Сторінка чайових

Так же як і на сторінці замовлень, чайові можна сортувати та фільтрувати.

3.11. Серверна частина функціоналу чайових

В даному підрозділі буде описано сервіс чайових.

Даний сервіс TipsService надає функціональність для роботи з чайовими (tips). Він містить різні методи, які дозволяють отримувати та зберігати інформацію про чайові.

Основні методи сервісу:

1) getOrdersByTipID: Цей метод приймає ідентифікатор чайового (id) та повертає список замовлень (Order[]), які були пов'язані з цим чайовим. Використовується метод getRepository для отримання доступу до репозиторію

2) TipEntity та Order. Запит до бази даних використовує createQueryBuilder для вибору замовлень, що відповідають заданим умовам.
3) getTips: Цей метод приймає діапазон дат (dateRange) і повертає список чайових (TipEntity[]), які були створені протягом цього діапазону дат. Крім того, для кожного чайового включається також список замовлень, пов'язаних з цим чайовим. Використовується метод getRepository для отримання доступу до репозиторію TipEntity та Order. Запит до бази даних використовує createQueryBuilder для вибору чайових та їх пов'язаних замовлень.
4) getTipsByDataRange: Цей метод є подібним до попереднього методу getTips, але повертає лише список чайових без пов'язаних замовлень. Використовується метод getRepository для отримання доступу до репозиторію TipEntity. Запит до бази даних використовує createQueryBuilder для вибору чайових.
5) getTipById: Цей метод приймає ідентифікатор чайового (id) і повертає об'єкт TipEntity або null, якщо чайовий не знайдений.
6) create: Цей метод створює новий запис чайового (TipEntity). Приймає контекст (ctx) та вхідні дані (input) і зберігає новий чайовий у базі даних.
7) update: Цей метод оновлює існуючий запис про чайовий. Приймає контекст (ctx) та вхідні дані (input), які містять оновлені значення для чайового. Оновлений запис зберігається у базі даних.

Для взаємодії з базою даних використовується об'єкт connection типу TransactionalConnection, який надає методи доступу до репозиторіїв та виконання запитів до бази даних. Контекст (ctx) використовується для ідентифікації та автентифікації користувача, що виконує запити.

3.12. Сторінка імпорту продуктів

На сторінці імпорту продуктів та колекцій користувач, тобто офіціант, може змінити меню, якщо, звісно, йому надали такі права. Також є можливість

експортувати продукти з меню, створивши новий файл для подальшого використання. Сторінка імпорту зображена нижче.

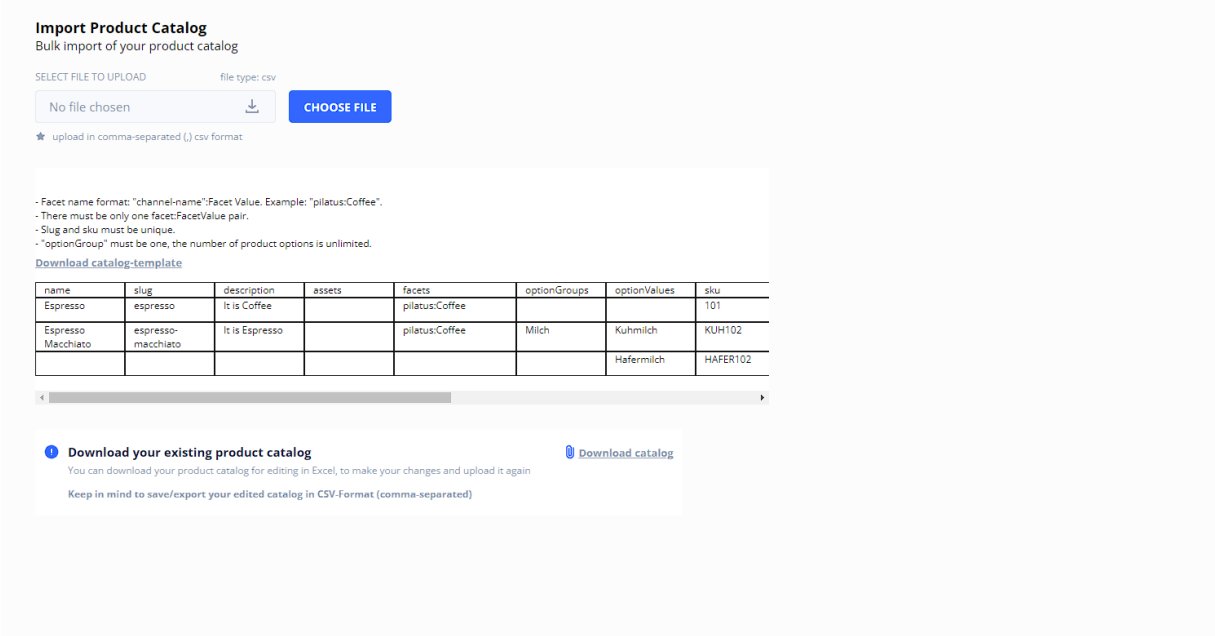


Рис. 3.20 – Сторінка імпорту та експорту

На даній сторінці є поле для завантаження файлу колекцій, приклад таблиці, посилання для завантаження прикладу та кнопка завантаження меню.

3.12.1. Серверна частина імпорту

При відправці файлу, його приймає мутація importCollection. Цей метод приймає контекст (ctx) та аргументи (args) і виконує наступні дії:

1. Перевіряє, чи існують контекст сесії та активний користувач (ctx.session та ctx.activeUserId). Якщо ні, викидається помилка "UNAUTHORIZED".
2. Перевіряє, чи існує канал (ctx.channel). Якщо ні, викидається помилка "Channel not found".
3. Перевіряє, чи існує файл (file). Якщо ні, викидається помилка "File not found".

4. Перевіряє, чи тип файлу є "text/csv". Якщо ні, викидається помилка "File is not csv".
5. Конвертує файл CSV у масив об'єктів за допомогою this.importService.convertCsvToObject(file).
6. Перевіряє, чи перший рядок CSV є дійсним продуктом за допомогою this.importService.isCsvLine(readed[0]). Якщо ні, викидається помилка "Product is invalid".
7. Викликає методи this.importService.setInactiveExistingProducts(ctx, readed), this.importService.creteCollectionsList(ctx, readed) та this.importService.createProductList(ctx, readed) для імпортування даних.
8. Повертає об'єкт з повідомленням "Uploaded" та статусом "DONE" у разі успішного завершення виконання, або об'єкт з повідомленням про помилку та статусом "ERROR" у разі виникнення помилки.

Імпорт даних реалізований в сервісі ImportService. Він має наступні методі:

- Метод getRootCollectionByChannel отримує кореневу колекцію за каналом і повертає її. Якщо колекція не знайдена, повертається значення null.
- Метод convertCsvToObject отримує об'єкт файлу CSV і повертає масив об'єктів, які представляють рядки CSV.
- Метод streamToString перетворює потік даних на рядок.
- Метод creteCollectionsList створює список колекцій на основі даних з CSV файлу. Він створює нові фасети та значення фасетів, створює або оновлює кореневу колекцію та додаткові колекції на основі фасетів.
- Метод deleteRedundantFacets видаляє непотрібні фасети, які не використовуються в CSV файлі.
- Метод deleteRedundantCollections видаляє непотрібні колекції, які не використовуються в CSV файлі.

- Метод createProductList створює список продуктів на основі даних з CSV файлу. Він створює нові продукти, встановлює їх атрибути, включаючи фасети, опції і варіанти продукту.
- Метод setInactiveExistingProducts встановлює статус "неактивний" для існуючих продуктів та варіантів продуктів, які не містяться в CSV файлі.
- Метод createOrUpdateProductVariant створює або оновлює варіант продукту на основі вхідних даних. Він перевіряє наявність варіанта за SKU і оновлює його, якщо він вже існує, або створює новий варіант.
- Метод splitAndFormat розбиває рядок фасету на назву і значення, які форматуються.
- Метод formatName форматує рядок, видаляючи зайві символи та перетворюючи його на URL-сумісний формат.

3.12.2. Серверна частина експорту

Метод exportProducts є обробником HTTP GET запитів, який експортує товари в CSV файл і дозволяє користувачеві його завантажити. Ось опис того, що робить цей метод:

1) Метод має декоратор @Get('export-products'), що означає, що він оброблятиме GET запити за шляхом /export-products.
2) Метод приймає два параметри: ctx типу RequestContext і res типу Response. Ці параметри представляють контекст запиту і об'єкт відповіді відповідно.
3) В межах блоку try-catch, метод перевіряє наявність властивості ctx.channel. Якщо вона не існує, це означає, що канал не знайдено, і метод повертає JSON відповідь зі статусом 404 і повідомленням "Канал не знайдено".
4) Метод визначає змінну path, яка вказує шлях до файлу, де буде збережений CSV файл. Вона використовує функцію join з модуля path для об'єднання поточного каталогу (__dirname) з іменем файлу (exported.csv).

5) Створюється об'єкт csvWriter за допомогою функції createCsvWriter, який приймає шлях до файлу і об'єкт заголовків для CSV файлу. Заголовки визначають поля, які будуть присутні в CSV файлі.
6) Викликається метод productsExport сервісу exportService з параметром ctx для отримання рядків, які потрібно експортувати.
7) Якщо отримано не пустий масив рядків, то вони записуються у CSV файл за допомогою методу writeRecords об'єкта csvWriter. Після цього метод повертає відповідь на завантаження файлу за шляхом path за допомогою res.download.
8) Якщо масив рядків порожній, то метод повертає JSON відповідь зі статусом 400 і повідомленням "Порожній список товарів".
9) У випадку виникнення помилки, метод повертає JSON відповідь зі статусом 400 і повідомленням про помилку, якщо воно є, або "Помилка при експорті" в іншому випадку.

Сервіс ExportService є відповідальним за експорт товарів в CSV формат. Ось опис його функціоналу:

1. Метод productsExport виконує експорт товарів у форматі CSV. Він приймає контекст ctx типу RequestContext і повертає масив об'єктів типу CsvLine, які представляють рядки CSV файлу.
2. У методі спочатку отримуються всі товари, використовуючи productService.findAll. Застосовується фільтрація за властивістю enabled: true. Отримані товари містять пов'язані варіанти, значення фасетів, фасети та групи опцій.
3. Перевіряється, чи існують товари та перший товар у масиві. Якщо ні, викидається помилка з повідомленням про помилку та кодом каналу.
4. Ініціалізується порожній масив lines для збереження рядків CSV файлу.
5. Для кожного товару виконується наступне:
    - Створюється змінна facet, яка представляє значення фасету для каналу.

- Ініціалізується об'єкт line типу CsvLine зі значеннями полів, що відповідають заголовкам CSV файлу. Поки що поля містять порожні значення, крім поля facets, яке містить значення фасету.
- Якщо кількість варіантів товару дорівнює 1, то отримується повний об'єкт варіанта за його ідентифікатором за допомогою productVariantService.findOne. Якщо варіант існує, то значення полів line заповнюються значеннями варіанта, отриманими з об'єкта variant. Потім line додається до масиву lines.
- У випадку, якщо кількість варіантів товару більше 1, отримуються переклади назв опцій групи, які використовуються для всіх варіантів товару. Потім викликається метод variantsLineCreator, який створює рядки для кожного варіанта товару і повертає масив variantsLines. Отримані рядки додаються до масиву lines.

6. На завершення метод повертає масив lines, який містить всі рядки для CSV файлу.
7. Метод variantsLineCreator використовується для створення рядків CSV файлу для кожного варіанта товару. Він приймає контекст ctx, масив варіантів variants і рядок товару productLine, який містить загальну інформацію про товар.
8. У методі створюється порожній масив lines для збереження рядків.
9. Для кожного варіанта товару виконується наступне:
    - Ініціалізується змінна line типу CsvLine. Якщо індекс варіанта дорівнює 0, то line приймає значення productLine, інакше створюється новий об'єкт line з порожніми значеннями полів, за винятком поля facets, яке містить значення з productLine.facets.
    - Отримуються повні дані про варіант за його ідентифікатором з використанням productVariantService.findOne. Якщо варіант не існує, викидається помилка "Варіант не визначений".
    - Значення полів line заповнюються значеннями варіанта.

- Переклади значень опцій отримуються з варіанта і додаються до поля optionValues в line.
- Рядок line додається до масиву lines.

10. На завершення метод повертає масив lines, який містить рядки для CSV файлу.

ВИСНОВОК

Під час виконання магістерської дипломної роботи було зібрано та опрацьовано купу матеріалу, в основному з мережі Інтернет. Розглянуто методи та інструментарій розробки додатків на React Native та серверних програм на Node.js.

Розроблений комплекс з трьох програмних продуктів виконує основні функції, поставлені на початку розробки, а саме:

1) Створення та робота з замовленнями.
2) Оповіщення офіціанта.
3) Підведення статистики.
4) Менеджмент столів з доступом по QR-коду.
5) Онлайн-оплата замовлень.
6) Імпорт та експорт продуктів та колекцій.

Після завершення розробки було сформовано та написано звіт.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

CRM системи в ресторанному бізнесі:

1. "The Ultimate Guide to Restaurant Customer Relationship Management (CRM)" [Електроний ресурс] - https://www.punchh.com/blog-posts/the-ultimate-guide-to-restaurant-customer-relationship-management-crm/ (Дата звернення 04.02.2023)
2. "How CRM Systems Benefit the Restaurant Industry" [Електроний ресурс] - https://www.posist.com/restaurant-times/technology/how-crm-systems-benefit-the-restaurant-industry.html (Дата звернення 04.02.2023)
3. "Restaurant CRM: What It Is, Why You Need It, and How It Works" [Електроний ресурс] - https://www.qsrautomations.com/blog/restaurant-crm-what-it-is-why-you-need-it-how-it-works/ (Дата звернення 01.02.2023)
4. "CRM for Restaurants: Why It Matters and How to Choose the Right Solution" [Електроний ресурс] - https://pos.toasttab.com/blog/on-the-line/restaurant-crm-why-it-matters-how-to-choose-the-right-solution (Дата звернення 04.02.2023)

Node.js:

5. Офіційний сайт Node.js [Електроний ресурс] - https://nodejs.org/en/ (Дата звернення 05.05.2023)
6. "Node.js Documentation" [Електроний ресурс] - https://nodejs.org/docs/latest-v14.x/api/ (Дата звернення 05.05.2023)
7. Офіційний сайт Nest.js [Електроний ресурс] - https://nestjs.com/ (Дата звернення 05.05.2023)
8. "Nest.js Documentation" [Електроний ресурс] - https://docs.nestjs.com/ (Дата звернення 05.05.2023)

React.js:

9. Офіційний сайт React.js [Електроний ресурс] - https://reactjs.org/ (Дата звернення 12.05.2023)

10. "React.js Documentation" [Електроний ресурс] - https://reactjs.org/docs/getting-started.html (Дата звернення 12.05.2023)

React Native:

11. Офіційний сайт React Native [Електроний ресурс] - https://reactnative.dev/ (Дата звернення 12.05.2023)
12. "React Native Documentation" [Електроний ресурс] - https://reactnative.dev/docs/getting-started

GraphQL:

13. Офіційний сайт GraphQL [Електроний ресурс] - https://graphql.org/ (Дата звернення 30.05.2023)
14. "GraphQL Documentation" [Електроний ресурс] - https://graphql.org/learn/ (Дата звернення 30.05.2023)
15. "Apollo Documentation" https://www.apollographql.com/ (Дата звернення 01.02.2023)