

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
ОБЛІКУ ТА КЕРУВАННЯ ЗАМОВЛЕННЯМИ У
МЕРЕЖІ КАВ'ЯРЕНЬ

SOFTWARE DEVELOPMENT FOR KEEPING RECORD OF
AND MANAGING ORDERS IN COFFEE SHOPS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-130
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ЖМУЦЬКИЙ М.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ(РОБОТУ) СТУДЕНТА(КИ)

ЖМУЦЬКОГО Максима Анатолійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення для обліку та керування замовленнями у мережі кав'ярень. Software Development for Keeping Record of and Managing Orders in Coffee Shops

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання « 15 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

Підпис
(підпис)

Максим ЖМУЦЬКИЙ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Тетяна ФЕДОРОНЧАК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
159 с., 12 табл., 67 рис., 3 дод., 22 джерел.

МЕРЕЖА КАВ'ЯРЕНЬ, ОБЛІК, ЗАМОВЛЕННЯ, АВТОМАТИЗАЦІЯ,
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, REACT, NESTJS, POSTGRESQL, REST.

Об'єкт дослідження – керування мережею кав'ярень.

Предмет дослідження – програмне забезпечення для обліку та керування замовленнями у мережі кав'ярень.

Мета роботи – розробка програмного забезпечення, яке базується на клієнт-серверній архітектурі та дозволяє здійснювати операції пов'язані з обліком та керуванням замовленнями у мережі кав'ярень.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване та компонентно-орієнтоване програмування, модульна архітектура, мова програмування JavaScript, HTML, фреймворк Nest.js, бібліотека React, CSS фреймворк TailwindCSS, бібліотеку анімації Framer Motion, база даних PostgreSQL, персональний комп'ютер з процесором Intel Core i5-12500H під управлінням операційної системи Windows 11 Version 22H2.

Результати. Розроблено програмне забезпечення для автоматизації процесів обліку та керування замовленнями у мережі кав'ярень, що дозволяє здійснювати управління різними аспектами діяльності та оптимізувати бізнес-процеси закладів.

Висновки. Реалізовано програмну систему для управління, обліку та зручної взаємодії з усіма аспектами діяльності мережі закладів кав'ярень.

Галузь використання – автоматизація бізнес-процесів, оптимізація управління та обліку у мережах кав'ярень.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 159 p., 12 tables, 67 figures, 3 appendices, 22 sources.

COFFEE SHOPS, ACCOUNTING, ORDERING, AUTOMATION, SOFTWARE, REACT, NESTJS, POSTGRESQL, REST.

The object of the research is the management of a network of coffee shops.

The subject of the research is software for accounting and managing orders in a network of coffee shops.

The purpose of the work is to develop software that is based on client-server architecture and allows operations related to accounting and order management in the network of coffee shops.

Materials, methods and technical means: object-oriented and component-oriented programming, modular architecture, JavaScript programming language, HTML, Nest.js framework, React library, TailwindCSS framework, Framer Motion animation library, PostgreSQL database, personal computer a computer with an Intel Core i5-12500H processor running the Windows 11 Version 22H2 operating system.

The results. Software has been developed to automate accounting and order management processes in the network of coffee shops, which allows you to manage various aspects of activity and optimize the business processes of establishments.

Conclusions. A software system has been implemented for management, accounting and convenient interaction with all aspects of the network of coffee shops.

Field of use – automation of business processes, optimization of management and accounting in coffee shop chains.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Постановка проблеми	11
1.2 Особливості функціонування систем управління мережами кав'ярень....	12
1.3 Розподіл відповідальності у системі	13
1.4 Ідентифікація користувачів у системі	14
1.5 Аналіз аналогів	14
1.6 Висновок	18
2 Розробка архітектури програми.....	20
2.1 Вибір засобів та технологій розробки.....	20
2.2 Вибір середовища розробки	28
2.3 Вибір та обґрунтування системи керування базою даних	29
2.4 Функціональні вимоги до програми.....	30
2.5 Висновки за розділом.....	35
3 Розробка програми	37
3.1 Реалізація серверної частини	37
3.2 Реалізація клієнтської частини	57
3.3 Висновки	68
4 Експлуатація та тестування програми.....	69
4.1 Призначення програми	69
4.2 Умови використання системи	69
4.3 Експлуатація програми	70

4.4 Тестування програми	97
4.5 Висновки	101
Висновки	102
Перелік джерел посилань	103
Додаток А Технічне завдання	105
Додаток Б Текст програми	109
Додаток В Слайди презентації	151

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CRUD	– Create, Read, Update, Delete;
CSS	– Cascading Style Sheets;
ER	– Entity-Relationship;
ES6	– ECMAScript 6;
HTML	– HyperText Markup Language;
HTTP	– Hypertext Transfer Protocol;
JS	– JavaScript;
ORM	– Object-Relational Mapping;
PHP	– Hypertext Preprocessor;
POS	– Point of Sale;
REST	– Representational State Transfer;
SQL	– Structured Query Language;
БД	– база даних;
СКБД	– система керування базами даних.

ВСТУП

Класифікація кав'ярні у якості «сучасної» вбачає не лише наявність у ній великого асортименту напоїв, а й те, як вона впроваджує сучасні технології у свою роботу. Особливого значення це питання набуває в умовах бурхливого розвитку ІТ-індустрії, яка все більше закріплюється у більшості сферах людського буття.

Все більше кав'ярень впроваджують передові технології забезпечуючи покращення власних сервісів та оптимізації бізнес-процесів. Наприклад, використання спеціалізованих програмних застосунків дозволяє підвищити швидкість обслуговування клієнтів або забезпечити надійний та зручний облік продукції, що неодмінно відзначиться як на якості обслуговування, так і на інших процесах у кав'ярні.

До того ж, впровадження систем управління надає можливість здійснення фінансового контролю роботи кав'ярні, що надає можливість власникам подібних бізнесів ефективно керувати ними, впроваджуючи заходи для покращення показників.

Тенденція впровадження сучасних технологій у роботу власного бізнесу призводить до позитивних наслідків як для самого власника, так і для його клієнтів, для яких створюється комфортне середовище, у якому можна не лише насолодитися кавовим напоєм, а й відчувати усі переваги розвитку цифрових технологій.

Саме тому об'єктом дослідження стала розробка програмного забезпечення для автоматизації різних аспектів управління та обслуговування кав'ярень через вебплатформу. Такий підхід дозволить оптимізувати робочі процеси, покращити обслуговування клієнтів та підвищити конкурентоспроможність бізнесу.

Метою даної дипломної роботи є розробка програмного забезпечення для управління та обліку діяльності мережі кав'ярень.

Для досягнення поставленої мети у дипломній роботі вирішуватимуться такі завдання:

- провести огляд стану ринку кав'ярень та існуючих методів управління бізнесом у цій галузі;
- дослідити особливості роботи мережі кав'ярень та визначити ключові бізнес-процеси, які можна оптимізувати;
- розробити програмне забезпечення для автоматизації управління кав'ярнями з урахуванням потреб оптимізації бізнес-процесів та підвищення ефективності управління;
- провести експериментальні дослідження та тестування розробленого програмного забезпечення для підтвердження його ефективності та придатності для використання в практичній діяльності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка проблеми

Важливість ефективного та своєчасного контролю усіх аспектів пов'язаних з діяльністю мереж кав'ярень викликає все більшу потребу в системах обліку та управління подібними мережами. Першочергово слід розглянути основні бізнес-процеси, з якими стикаються подібні заклади під час роботи.

Перш за все, оптимальність розподілу ресурсів, серед яких персонал, інгредієнти, обладнання, між закладами мережі є надскладним завданням. Надмірність або нестача ресурсів можуть призвести до їх нераціонального використання, що є неприпустимим у будь-якому бізнесі.

Наступним постає питання керування людським ресурсом, вірно налагоджена система дозволяє не лише вчасно реагувати на критичні ситуації, пов'язані з надмірною кількістю робочих годин, але й заохочувати працівників шляхом оцінювання продуктивності праці.

Будь-яка мережа має власний перелік позицій, які вона може запропонувати своїм клієнтам, тому чіткість визначення процесів пов'язаних з приготуванням їжі та напоїв є вкрай важливою для забезпечення якості обслуговування відвідувачів.

Кожний власник бізнесу прагне мати інструмент, який може забезпечити чіткий облік доходів та витрат, дозволить стежити за різними показниками роботи, що неодмінно допоможе відслідковувати рентабельність різних послуг або закладів в цілому.

Спеціалізована система обліку та управління мережами кав'ярень має вирішити ці питання, забезпечивши належне функціонування кожної кав'ярні та підвищивши її загальну продуктивність і прибутковість.

1.2 Особливості функціонування систем управління мережами кав'ярень

Сучасні тенденції розвитку закладів швидкого харчування сприяють все більшій популяризації систем управління та обліку подібних бізнесів, масовим характером здебільшого користується програмне забезпечення спрямоване на заклади, які спеціалізуються на ресторанній справі. Але з кожним днем все більше і більше з'являється закладів у форматі «на виніс», для яких існуючі рішення є доволі перевантаженими непотрібним функціоналом, якого неможливо уникнути та який погіршує інтуїтивність та зручність систем.

Існує низка систем, які спеціалізуються виключно на кав'ярнях та дозволяють користуватися рішеннями, необхідними лише цій категорії закладів. Подібні застосунки зазвичай надають зручний та зрозумілий інтерфейс, який поєднує у собі лише основні аспекти діяльності, на кшталт, замовлень, обліку запасів та звітності.

Таким чином, слід виділити ряд основних функціональних вимог до систем, які спеціалізуються виключно на кав'ярнях:

- наявність функціоналу керування замовленнями, тобто має надаватися можливість швидкої та ефективної обробки замовлень клієнтів, забезпечуючи зручний у використанні інтерфейс для барист;
- функціонал додавання, редагування та видалення позицій меню, включаючи можливості визначення рецептури та ціни;
- засоби для ведення обліку та контролю залишків продуктів;
- можливість проведення касових операцій, включаючи готівкові та безготівкові розрахунки;
- інструменти для впровадження та керування акційними пропозиціями на певні позиції;
- засоби для керування персоналом, включаючи здійснення контролю доступності та продуктивності праці персоналу;

– функціонал для отримання статистичних показників пов'язаних з різними складовими мережі.

Таким чином, перелічені функціональні вимоги необхідні для підтримки успішної роботи кав'ярні, забезпечення оптимального управління бізнесом і задоволеності клієнтів.

1.3 Розподіл відповідальності у системі

Спеціалізовані системи управління мережами кав'ярень передбачають дві ключові ролі – бариста та адміністратор.

Бариста є фахівцем, який забезпечує приготування відповідних позицій меню, використовуючи наданий йому перелік рецептів, та якісне обслуговування клієнтів. Обов'язково володіє професійними навичками роботи з обладнанням, яке необхідне в ході його роботи, розуміє технологію роботи з різними категоріями продукції та має гарні навички комунікації та обслуговування клієнтів.

Адміністратор вбачає у собі роль особи, яка відповідальна за керування діяльністю мережі кав'ярень у всіх її проявах, серед яких, контроль за фінансовою діяльністю, провадження меню та керуванням рентабельністю ціноутворень, відслідковуванням роботи підпорядкованого персоналу.

Роль адміністратора передбачає наявність необмеженого доступу до будь-якого функціоналу системи, в межах власної мережі.

Відповідно баристі надається обмежений рівень доступності, здебільшого це операції пов'язані з переглядом різної інформації, за винятком можливості створення замовлень, оновлення кількості запасів та роботи з касою кав'ярні.

Обидві ролі в системі є ключовими у системі. Бариста надає якісне обслуговування клієнтів і виробництво високоякісної продукції, адміністратор забезпечує контроль над усіма аспектами діяльності компанії та сприяє її безперебійній роботі.

1.4 Ідентифікація користувачів у системі

Системи управління кав'ярнею передбачають наявність надійної системи для захисту від несанкціонованого доступу до даних. У подібних системах ідентифікація користувачів здійснюється на основі облікових записів.

Для входу у систему потребується наявність в користувача власної пошти та паролю.

У випадку адміністраторів мереж, інформація про обліковий запис формується з введених під час реєстрації даних. Облікові записи барист є, фактично, складової мережі, тому створення відбувається шляхом надання відповідних даних адміністратору мережі. Додатково кожному з учасників системи має надаватися можливість відновлення доступу до облікового запису.

Для можливості виконання наведених вище дій, система повинна мати у своєму складі наступні модулі:

- модуль аутентифікації для барист та адміністраторів;
- модуль реєстрації адміністраторів;
- модуль відновлення доступу до облікового запису.

Ці рішення дозволяють вберегти від несанкціонованого доступу до даних з боку користувачів відсутніх у системі.

1.5 Аналіз аналогів

1.5.1 Kavarr

Kavarr є комплексним програмним забезпеченням для управління кав'ярнями, яке спрямоване на оптимізацію процесів, пов'язаних з обслуговуванням клієнтів, обробкою замовлень та процесу інвентаризації. Система дозволяє здійснювати керування замовленнями, запасами, персональними даними та послугами, спеціалізується виключно на мережах кав'ярень, що забезпечує відсутність надмірного функціоналу [1].

На рис. 1.1 зображено інтерфейс Kavarr.



Рисунок 1.1 – Інтерфейс Kavarr [1]

Розглянемо детально переваги цієї системи:

- система Kavarr автоматизує багато рутинних завдань, пов'язаних з обробкою замовлень, управлінням запасами та керуванням роботою барист;
- застосунок дозволяє формувати звіти про продажі та загальну аналітику, які допомагають приймати раціональні рішення для вдосконалення стратегії роботи та розвитку кафе;
- програмне забезпечення надає можливість налаштування програми лояльності клієнтів.

До недоліків даної системи можна віднести:

- впровадження та налаштування Kavarr є дорогим і трудомістким процесом, особливо для невеликих кав'ярень з обмеженим бюджетом;
- інтерфейс доволі складний, тому деяким користувачам може знадобитися час для його освоєння, особливо новачкам, які не мали раніше досвіду роботи з подібними системами.

1.5.2 SmartCafe

SmartCafe є інтегрованою POS системою, яка розроблена виключно для забезпечення потреб кафе та ресторанів. SmartCafe надає можливості для

керування замовленнями, інвентарем та персоналом, забезпечуючи зручну та оптимальну роботу більшості бізнес-процесів [2].

На рис. 1.2 зображено інтерфейс SmartCafe.



Рисунок 1.2 – Інтерфейс SmartCafe [2]

Серед переваг використання цього засобу управління бізнесом можна виділити наступне:

- система забезпечує швидке та ефективне оброблення та виконання замовлень, сприяючи збільшенню швидкості обслуговування клієнтів;
- наявна можливість легкого налаштування та оновлення меню, зокрема додавання нових пунктів, змінювання ціни та встановлення акційних пропозицій;
- SmartCafe надає комплексні аналітичні звіти, зокрема з продажів, дозволяючи власникам кав'ярень приймати правильні управлінські рішення;
- система легко інтегрується з різними платіжними системами для зручності оплати замовлень;
- інтерфейс надає можливість використання різноманітних мобільних пристроїв, дозволяючи працівникам кафе бути мобільними та ефективними.

Недоліками даного програмного продукту є:

- POS SmartCafe вимагає чималих витрат на впровадження програмного забезпечення та його налаштування;
- обслуговування системи передбачає обов’язкове фізичне втручання спеціаліста, що може зробити систему дорогою та важкою в обслуговуванні;
- SmartCafe особливо схильна до помилок і збоїв, які здебільшого виникають через помилки у середині програмного забезпечення.

1.5.3 Poster

POS Poster є системою, яка забезпечує автоматизацію управління кафе та ресторанами [3].

На рис. 1.3 зображено інтерфейс Poster.

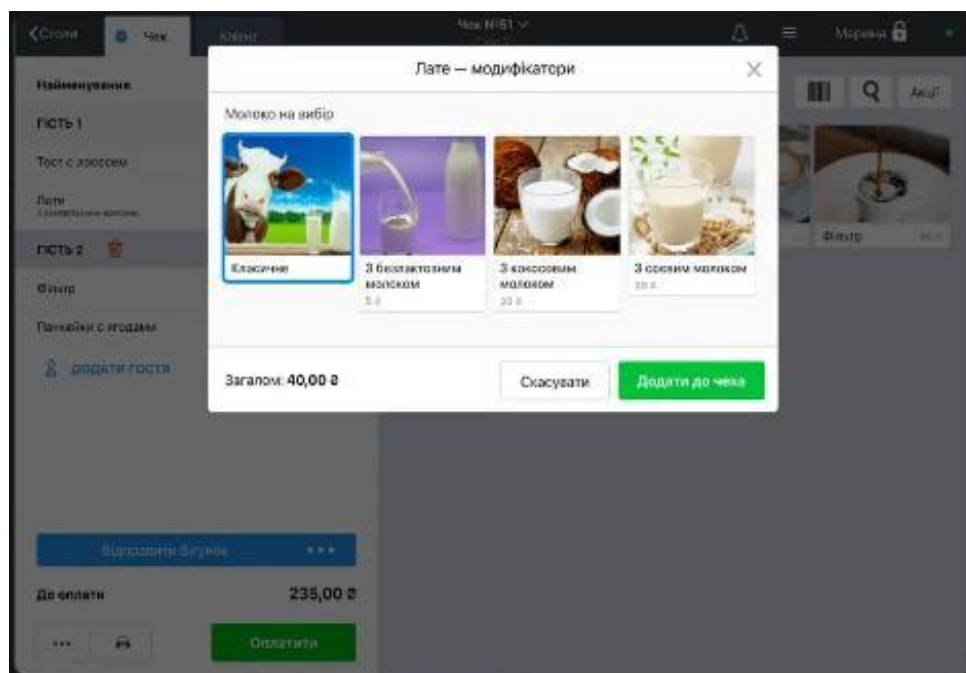


Рисунок 1.3 – Інтерфейс Poster [3]

Перш за все, виділимо переваги цього програмного рішення, а саме:

- Poster має інтуїтивно зрозумілий і зручний у використанні інтерфейс, який дозволяє користувачам швидко освоїти систему та працювати з нею без особливих зусиль;

- система має широкий набір функцій, таких як управління замовленнями, інвентаризацію, управління готівкою та отримання звітності;
- вивіски POS легко адаптуються до потреб будь-якого типу закладу, незалежно від його розміру та характеристик;
- забезпечується інтеграція з різними платіжними системами, що підвищує зручність оплати замовлень;
- Poster надає можливість створення та керування базою даних клієнтів, що дозволяє об'єднувати інформацію про постійних клієнтів і надавати їм персоналізовані пропозиції та знижки.

Серед недоліків системи можна виділити:

- впровадження та підтримка системи може бути дорогим для деяких власників кафе чи ресторанів, особливо малого бізнесу з невеликим бюджетом;
- важкість роботи з інтерфейсом вимагає часу та зусиль для навчання персоналу, що може призвести до збільшення робочого часу або тимчасових перерв у роботі.

1.6 Висновок

Таким чином, у даному розділі було досліджено предметну область програмного забезпечення для управління мережами кав'ярень, в результаті чого сформовано ряд проблемних питань, які виникають в ході керування мережами кав'ярень.

Виділено основні особливості функціонування систем спеціалізованих виключно на роботі кав'ярень.

Визначено перелік ролей у системі та розподіл їх відповідальності. Деталізовано систему ідентифікації користувачів системи, з визначенням необхідних даних для ідентифікації облікового запису та набір модулів для роботи цієї частини системи.

Здійснено аналіз аналогів розроблюваного застосунку, який показав відсутність спеціалізованих систем для кав'ярень, які дозволяють вирішувати

проблеми вузької спеціалізації, без навантаження надлишковим функціоналом. Тому розроблюваний застосунок має вирішити цю проблему, надавши увесь необхідний функціонал для управління мережами кав'ярень та забезпечивши конфіденційність даних, які зберігаються у системі.

Таким чином, розроблювана система має стати оптимальним рішенням для здійснення керування мережами кав'ярень.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів та технологій розробки

2.1.1 Вибір мови програмування

Вибір мови програмування для реалізації програмного забезпечення передбачає не лише визначення її здатності вирішувати поставлене завдання, але й аналізу її впливу на показники продуктивності, масштабованості та складності підтримки майбутнього проєкту, досягнення найкращих результатів у цих показниках забезпечується шляхом правильного підбору засобів розробки, зокрема мови програмування.

Популярність мови програмування забезпечує їй велику спільноту розробників, яка дозволяє при появі певних складнощів, у ході розробки, звернутися до неї для пошуку рішень. З іншого боку, обрання невідповідного інструменту розробки призведе до проблем масштабованості, затримок проєктування та вразливої системи безпеки. Тому важливість ретельного аналізу потреб та особливостей проєкту, перш ніж визначитися з мовою програмування, є неабиякою доцільною.

У поточній ситуації, найпопулярнішими мовами програмування для веброзробки є C#, JavaScript (включаючи TypeScript), Python та PHP. Кожній з перелічених мов притаманні свої унікальні особливості та сфери застосування, зокрема розробка вебзастосунків [4].

У табл. 2.1 наведено порівняльну характеристику перелічених мов програмування, відповідно до потреб проєкту.

Таблиця 2.1 – Порівняльна характеристика мов програмування

№	Критерій за яким порівнюються	C#	JS(TS)	Python	PHP
1	2	3	4	5	6
1	Складність розробки	Середня	Низька	Низька	Висока
2	Швидкодія	Висока	Висока	Низька	Середня

Продовження таблиці 2.1

1	2	3	4	5	6
3	Екосистема	Велика	Велика	Велика	Велика
4	Масштабованість	Добра	Добра	Добра	Погана
5	Асинхронність	Підтримує	Підтримує	Не підтримує	Не підтримує
6	Строга типізація	Є	Є	Відсутня	Відсутня
7	Спільнота та підтримка	Середня	Велика	Велика	Велика

Проаналізувавши найбільш конкуруючі мови програмування, серед наведених у порівнянні, для веброзробки, якими є JavaScript (зокрема, TypeScript) і Python, необхідно розглянути їх більш детально.

JavaScript (включаючи TypeScript) має низку переваг, перш за все, ця мова програмування є однією з найбільш широко використовуваних мов для веброзробки, що забезпечує широкий спектр фреймворків і бібліотек для створення вебзастосунків як на стороні клієнта, так і на стороні сервера. TypeScript, розширення JavaScript, додає до мови статичну типізацію та інші сучасні функції, що значно полегшує створення та підтримку великих проєктів. Крім того, JavaScript можна виконувати як на стороні клієнта, так і на стороні сервера, що забезпечує зручне створення та розгортання.

Беручи до уваги показник масштабованості, JavaScript (завдяки TypeScript) має великий потенціал. Це дозволяє розробникам легко створювати складні системи та масштабувати їх з часом завдяки гнучкості мови та широкому набору інструментів керування кодом.

З точки зору швидкості, JavaScript (особливо при використанні JIT-компіляції в сучасних браузерах і серверних середовищах) може бути досить швидким. Нові версії механізму JavaScript постійно оптимізуються для підвищення продуктивності та швидкості виконання коду.

Таким чином, JavaScript є гарним вибором для розробки вебзастосунків, які потребують швидкої реакції та високої продуктивності.

З іншого боку, Python також має свої переваги. Python, відомий своєю простотою та лаконічністю синтаксису, зручністю для початківців. Мова має великий набір фреймворків, таких як Django і Flask, які дозволяють швидко створювати ефективні вебзастосунки. Крім того, Python широко використовується в наукових і академічних сферах для розробки складних обчислювальних моделей і аналізу даних. Високий рівень абстракції та крос-платформна підтримка дозволяють розробникам створювати програмні рішення для різноманітних галузей, включаючи веброзробку. Таким чином, Python зберігає свою популярність як вибір для різноманітних завдань програмування, включаючи розробку вебзастосунків.

Враховуючи можливість використання JavaScript як на стороні клієнта, так і на стороні сервера, ця мова програмування є оптимальним вибором для створення подібного програмного забезпечення. Це дає можливість працювати однією мовою в обох частинах програми, що спрощує процес створення, підтримки та розгортання програмного забезпечення. Крім того, широкий вибір фреймворків і бібліотек, доступних для JavaScript, дозволяє створювати застосунки з високою продуктивністю та великими функціональними можливостями, забезпечуючи підтримку однорідності коду та оптимізацію робочого процесу, забезпечуючи гнучкість і ефективність розробки.

2.1.2 Вибір засобів розробки клієнтської частини застосунку

У зв'язку з тим, що основною мовою розробки є JavaScript, для реалізації клієнтської частини обрана бібліотека React завдяки низці переваг.

Перш за все, React відомий своєю високою швидкістю та ефективністю завдяки використанню віртуального DOM, що дозволяє скоротити час, необхідний для оновлення вмісту сторінки, і забезпечує більш плавну та швидку

взаємодію з користувачем, що особливо важливо у випадку вебзастосунків із великою кількістю взаємодій, як у нашому випадку [5].

До того ж, React пропонує компонентну архітектуру, яка сприяє простоті розробки коду та зручності обслуговування. Концепція повторного використання компонентів дозволяє полегшити створення та підтримку вебзастосунків. Окремі компоненти також забезпечують ясність і структуру коду, дозволяючи швидше реагувати на зміни та впроваджувати нові функції [5].

Крім того, React має активну спільноту розробників і широкий вибір сторонніх бібліотек і розширень, що робить його чудовим вибором для швидкого й ефективного створення вебзастосунків. Його велика екосистема дозволяє знаходити рішення на будь-який смак і вирішувати різноманітні завдання програмування. Тому, обираючи React, ми отримуємо потужний інструмент для створення сучасних та ефективних вебзастосунків [5].

Так як у ході роботи застосунку виникає потреба у зберіганні глобального стану застосунку, доступ до даних якого можна буде отримати на будь-якій сторінці застосунку. Для вирішення цієї потреби було використано бібліотеку Redux, так як вона має найкращу сумісність з React.

Redux – це популярна бібліотека для керування станом програм у програмах JavaScript. Одним із головних завдань великих і складних програм є ефективне управління станом програми. Redux вирішує цю проблему, надаючи єдине централізоване сховище всієї інформації про стан програми, роблячи стан керованим і передбачуваним [6].

Головним принципом Redux є єдине сховище стану, яке зберігається у вигляді дерева. Інтерфейси, які змінюють стан, називаються «діями», які надсилаються редуктору(ам). Редуктори – це функції, які обробляють дії та змінюють стан залежно від цих дій. Детальну схему роботи наведено на рис. 2.1.

Однією з ключових переваг Redux є простота використання та чистота коду. Це полегшує відстеження та керування станом вашої програми, роблячи код чистішим і зручнішим для обслуговування.

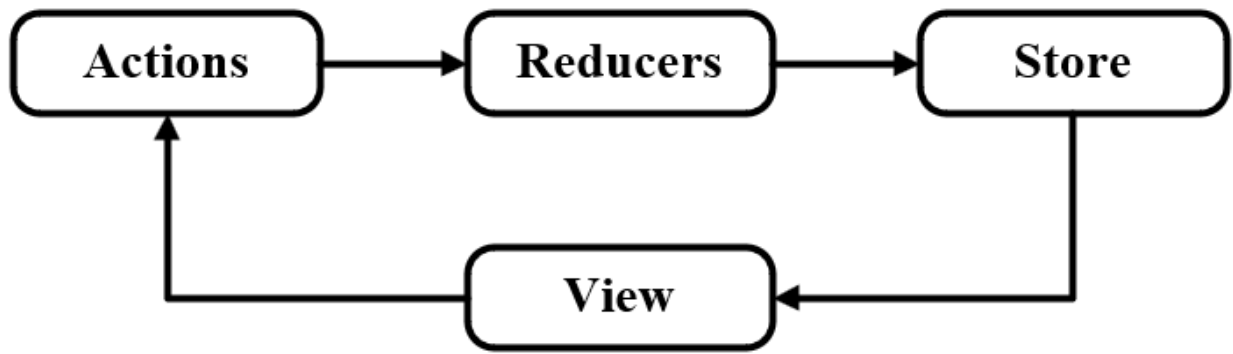


Рисунок 2.1 – Схема роботи Redux

Наступним постає питання маршрутизації в межах застосунку, для цього було обрано бібліотеку React Router.

React Router – бібліотека для маршрутизації у вебзастосунках, яка містить набір компонентів, що дозволяють організувати навігацію у системі та відображати різні компоненти залежно від URL-адреси [7].

React Router допускає декларативний підхід до визначення маршрутів, що робить його більш зрозумілим і простим у використанні. Також він має вбудовану підтримку параметрів маршруту, розширені можливості передачі та керування навігацією в програмі.

Загалом, структура клієнтської частини виглядає наступним чином:

- components – складові компоненти сторінок;
- helper – допоміжні функції;
- hooks – хуки;
- layouts – шаблони для відповідних груп маршрутів;
- pages – сторінки для маршрутів;
- router – інформація про маршрутизацію;
- services – сервіси для роботи з сутностями;
- store – інформація та функціонал глобального оброблювача стану;
- types – інформація про типи для обробки різних подій пов'язаних з сутностями.

сутностями.

2.1.3 Вибір засобів розробки серверної частини застосунку

Для реалізації серверної частини системи було обрано фреймворк Nest.js.

Nest.js є фреймворком, який використовується для створення масштабованих і ефективних серверних додатків на Node.js, і базується на принципах модульності, об'єктно-орієнтованого програмування та функціонального реактивного програмування [8].

Серед основних переваг цього фреймворку можна виділити наступні:

- Nest.js пропонує вбудовану модульну систему (див.рис. 2.2), яка дозволяє розділити функціональність програми на підсистеми та ефективно керувати їх залежностями, що спрощує розробку, тестування та обслуговування коду;

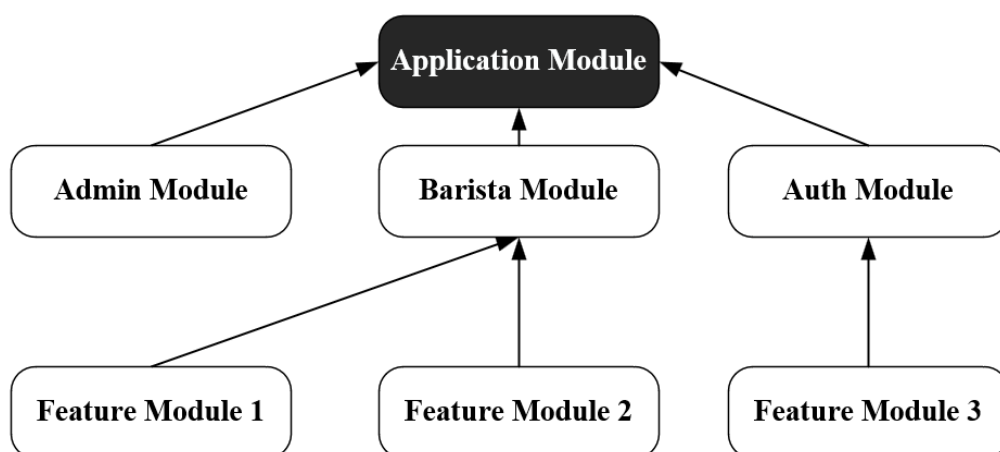


Рисунок 2.2 – Приклад схеми модульності застосунку на базі Nest.js

- фреймворк використовує об'єктно-орієнтований підхід до програмування, що робить код більш зрозумілим і простим в управлінні, що дозволяє створювати шаблони та класи, які можна повторно використовувати та розширювати;

- Nest.js повністю підтримує TypeScript, дозволяючи писати безпечний і типізований код, полегшуючи пошук і виправлення помилок під час процесу розробки;

– фреймворк пропонує велику кількість вбудованих функціональних можливостей, таких як модуль HTTP, WebSockets тощо. Крім того, Nest.js підтримує розширюваність за допомогою проміжного програмного забезпечення та плагінів, дозволяючи впроваджувати власні функції та розширювати можливості фреймворку. Гарним прикладом є «pipes» та «guards», вони виступають у ролі «middleware», які дозволяють оброблювати запити до сервера перед тим як відбудеться виклик оброблювача маршруту;

– Nest.js має активну спільноту розробників, а також добре задокументовану офіційну документацію з безліччю прикладів і ресурсів для ознайомлення, що неодмінно пришвидшує процес розробки.

Виходячи з усіх переваг, Nest.js є найкращим вибором для впровадження серверної частини застосунку.

Архітектура фреймворку Nest.js має трьох компонентну складову: моделі, контролери та служби. Кожен компонентів відіграє важливу роль в організації та розвитку програми.

Моделі визначають структуру даних і спосіб взаємодії з ними. Вони відповідають за роботу з базою даних і представлення даних у додатку. У контексті Nest.js модель може використовувати ORM для спрощення взаємодії з базою даних. Моделі дозволяють визначати структуру та типи даних, які використовуються в додатку, що полегшує роботу з ними та забезпечує їх узгодженість.

Контролери відповідають за обробку запитів клієнтів і взаємодію з моделями та службами для отримання необхідних даних і відповідей. Вони приймають HTTP-запити, виконують необхідні операції та повертають клієнтам відповіді. Контролери допомагають відокремити логіку обробки запитів від бізнес-логіки, роблячи вашу програму більш організованою та легкою для розширення.

Сервіси відповідають за впровадження бізнес-логіки та обробки даних. Вони використовуються адміністраторами для виконання операцій, не пов'язаних безпосередньо з обробкою HTTP-запитів. Сервіси можуть

взаємодіяти з моделями для доступу до даних і маніпулювання ними. Вони допомагають відокремити бізнес-логіку від логіки контролера, роблячи програму більш гнучкою та легко розширюваною.

Базуючись на концепціях фреймворку Nest.js отримуємо наступну структурну схему застосунку (див.рис. 2.3).

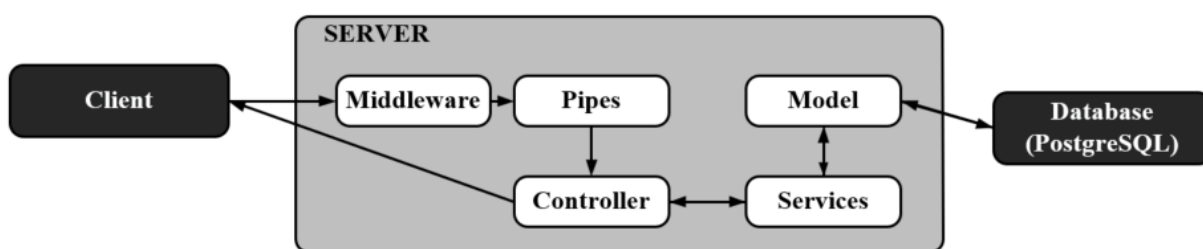


Рисунок 2.3 – Структурна схема серверу на базі Nest.js

Поняття «Модуль» узагальнює поєднання трьох складових: Controller, Model та Service.

В ході роботи системи здебільшого постає питання виконання операцій пов'язаних зі створенням, читанням, оновленням та видаленням (CRUD), які дозволяють маніпулювати інформацією в базі даних. Архітектура REST (Representational State Transfer) є одним із найпоширеніших підходів до реалізації такого типу операцій [9].

REST забезпечує простий стандартизований спосіб взаємодії між клієнтом і сервером за допомогою запитів HTTP. Цей підхід поділяє дані на ресурси, до яких можна отримати доступ за допомогою єдиного набору операцій, таких як GET (для отримання даних), POST (для створення нових даних), PUT або PATCH (для оновлення даних) і DELETE (для видалення даних). Це спрощує розробку та обслуговування системи, а також забезпечує чітку структуру та зручність у взаємодії з даними [9].

Крім того, використання архітектури REST допомагає покращити масштабованість системи та незалежність від платформи, оскільки дані надсилаються у формі стандартних HTTP-запитів і відповідей. Це робить систему більш гнучкою та готовою до майбутніх змін.

2.2 Вибір середовища розробки

У якості середовища розробки проєкту було обрано Visual Studio Code, який є чудовим вибором для створення проєктів з використанням React і Nest.js виходячи з низки важливих причин, серед яких [10]:

- VS Code має широку підтримку TypeScript і JavaScript, що робить його ідеальним середовищем для створення проєктів React, які базуються на використанні TypeScript або JavaScript. Середовище забезпечує автозаповнення, перевірку помилок, рефакторинг та інші корисні функції, які полегшують процес розробки;

- VS Code надає широкий спектр розширень для підтримки розробки з використанням React і Nest.js. Серед них, розширення React/Redux Snippets, яке пропонує ряд корисних шаблонів коду для швидкого написання компонентів React, а розширення Nest Snippets надає подібні можливості для Nest.js. Ці розширення допомагають підвищити продуктивність і швидкість програмування;

- вбудована підтримка систем контролю версій, таких як Git, що дозволяє зручно виконувати коміти, push-завантаження, пули та інші операції безпосередньо з інтерфейсу редактора;

- середовище розробки VS Code відоме своєю швидкістю і простотою використання, воно швидко запускається, використовує мало ресурсів і працює ефективно навіть на слабких пристроях. Інтерфейс інтуїтивно зрозумілий і добре організований, що робить його ідеальним засобом для розробників будь-якого рівня кваліфікації.

Тому Visual Studio Code є ідеальним середовищем для створення проєктів використовуючи бібліотеку React та фреймворк Nest.js, завдяки ряду функцій, розширенням та можливості інтеграції з іншими інструментами розробки. Це сприяє комфортній та продуктивній роботі над проєктами будь-якої складності.

2.3 Вибір та обґрунтування системи керування базою даних

Тенденція зростання обсягу даних при роботі притаманна кожній інформаційній системі, тому проблема обрання системи керування базою даних є надзвичайно важливою. Правильно підібрана СКБД забезпечує швидкість обробки даних, масштабованість та безпеку зберігання даних. Важливими чинниками є достатній функціонал, екосистема розширень та рівень підтримки від виробника.

Вибір системи керування базами даних є вирішальним кроком у розробці будь-якого програмного забезпечення, система керування мережею кав'ярень не є винятком. В умовах зростання конкуренції та зміни потреб клієнтів ефективна обробка даних, швидкий доступ до даних і надійне зберігання стають важливими аспектами успішного бізнесу в цій галузі.

На сьогодні найбільшою популярністю користуються наступні СКБД: PostgreSQL, MySQL, MongoDB та Microsoft SQL Server [11].

У табл. 2.2 наведено порівняльну характеристику наведених СКБД, відповідно до потреб проєкту.

Таблиця 2.2 – Порівняльна характеристика мов програмування

Критерій	PostgreSQL	MySQL	MongoDB	MS SQL Server
Масштабованість	+	+	+/-	+
Продуктивність при роботі з великим об'ємом даних	+	-	-	+
Надійність	+	+/-	+/-	+
Функціональність	+	+	+	+
Безкоштовна	+	+	+	-
Спільнота	+	+	+	+

Управління мережею кав'ярень вимагає комплексного підходу до обробки та аналізу даних. PostgreSQL, завдяки своїй реляційній моделі даних, має чудові можливості для забезпечення узгодженості та цілісності даних. Гнучкість і розширюваність дозволяють легко адаптувати базу даних до зростаючих потреб бізнесу.

Порівнюючи з іншими СКБД, PostgreSQL має перевагу у багатьох показниках. Їй притаманні висока продуктивність і експлуатаційна надійність, а також широкий спектр функціональних можливостей. Враховуючи складні вимоги до управління та обліку мережами кав'ярень, PostgreSQL є оптимальним вибором.

Завдяки великій та активній спільноті PostgreSQL, забезпечується підтримка та розвиток СКБД, як зараз, так і у майбутньому. Цей фактор обов'язковий для забезпечення стабільної та безпомилкової роботи програмного забезпечення в довгостроковій перспективі.

Таким чином, вибір СКБД у якості PostgreSQL, задля розробки програмного забезпечення пов'язаного з управлінням мережею кав'ярень, є не тільки логічним рішенням, але й практично виправданим. Вона забезпечує необхідну функціональність, продуктивність та надійність, які є критично важливими в сучасному бізнес-середовищі.

2.4 Функціональні вимоги до програми

В результаті аналізу предметної області, виконаного у розділі 1, було визначено наявність у системі двох ролей: адміністратор та бариста.

Обидві ролі передбачають наявність можливості авторизації, відновлення паролю та виходу з системи, а також інструментарій для роботи з сутностями.

2.4.1 Функціональні вимоги для ролі баристи

Роль баристи передбачає взаємодію з наступними сутностями: «Інгредієнти», «Замовлення», «Точка продажу», «Позиція меню», «Рецепти» та «Робочі зміни».

Виходячи з цього на рис. 2.4 представлено use-case діаграму функціональних вимог для ролі баристи.

Таким чином, роль баристи передбачає можливість виконання наступних функцій відносно зазначених сутностей:

- пошук та перегляд інгредієнтів;
- оновлення кількості інгредієнтів;
- створення замовлень;
- перегляд позицій меню;
- перегляд створених замовлень;
- зміна статусу замовлень;
- перегляд інформації про кав'ярню;
- зняття грошей з рахунку кав'ярні;
- перегляд та пошук рецептів;
- отримання статистичних даних за останній місяць;
- перегляд власної інформації;
- оновлення власної інформації;
- початок робочої зміни;
- завершення робочої зміни.

Тож, цей набір функцій дозволяє забезпечити усі основні потреби баристи під час роботи.

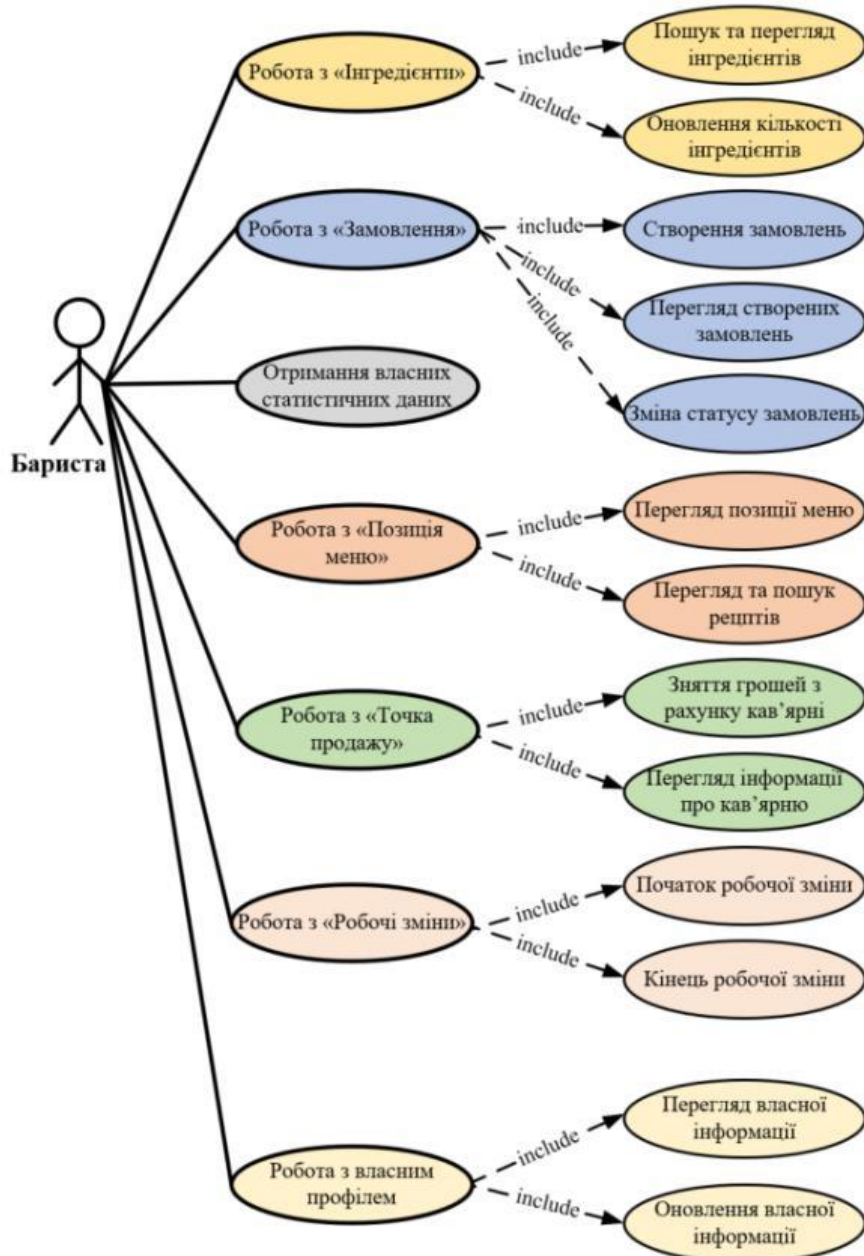


Рисунок 2.4 – Use-case діаграма для ролі баристи

2.4.2 Функціональні вимоги для ролі адміністратора

У свою чергу, роль адміністратора передбачає взаємодію з наступними сутностями: «Бариста», «Категорія», «Позиція меню», «Замовлення», «Знижки», «Робочі зміни», «Інгредієнти», «Замовлення», «Точка продажу» та «Рецепти».

Виходячи з цього на рис. 2.5 представлено use-case діаграму функціональних вимог для ролі адміністратор.

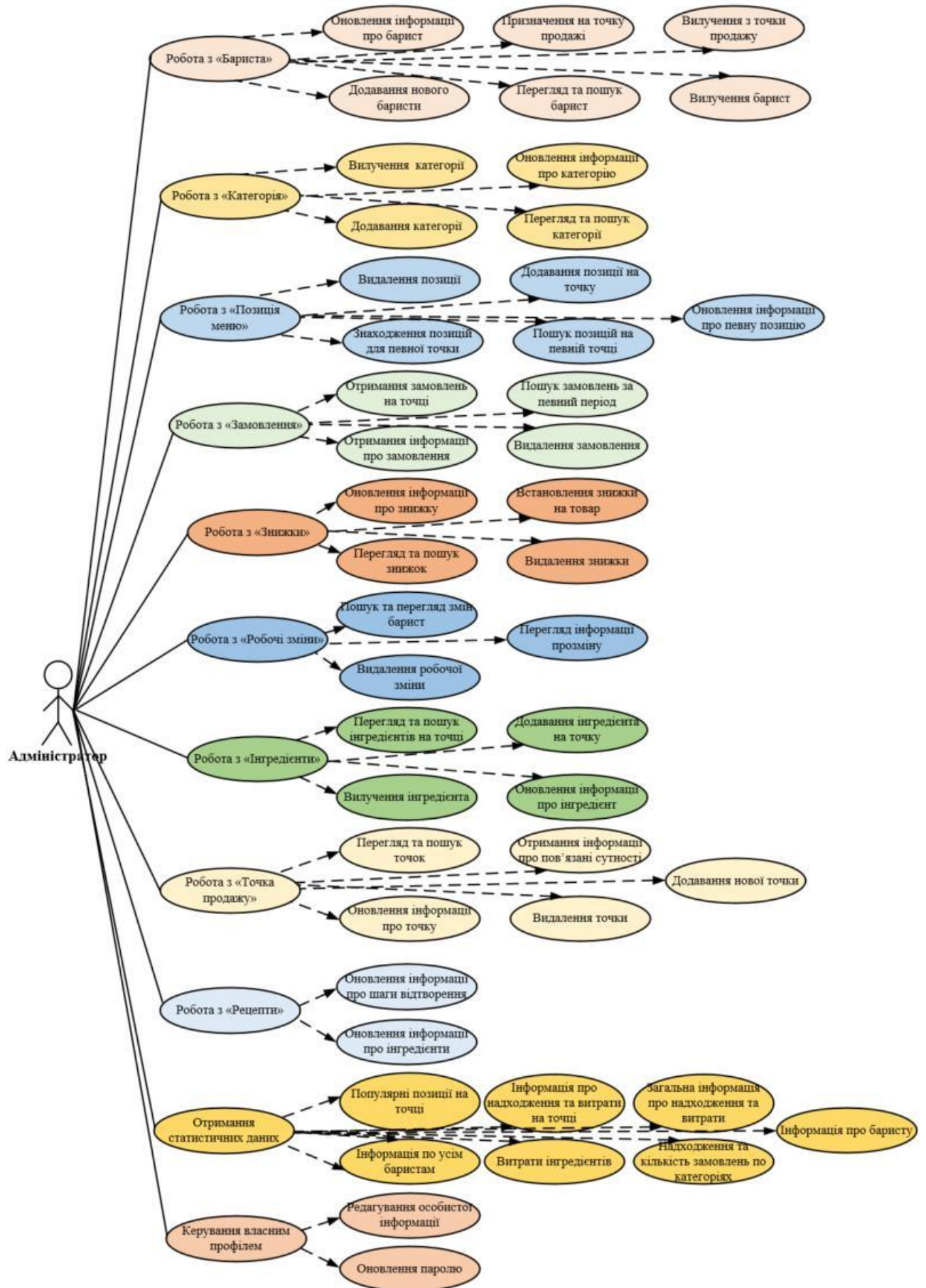


Рисунок 2.5 – Use-case діаграма для ролі адміністратор

Таким чином, роль адміністратора передбачає можливість виконання наступних функцій відносно зазначених сутностей:

- додавання нового баристи;
- перегляд та пошук барист;
- вилучення барист;
- оновлення інформації про барист;
- призначення баристи на точку продажу;
- вилучення баристи з точки продажу;
- додавання категорії;
- вилучення категорії;
- оновлення інформації про категорію;
- перегляд та пошук категорій;
- додавання позиції на точку;
- видалення позиції;
- перегляд та пошук позицій по точкам;
- оновлення інформації про певну позицію;
- отримання замовлень на точці;
- пошук замовлень за певний період;
- отримання детальної інформації про замовлення;
- видалення замовлення;
- встановлення знижки на позицію меню;
- перегляд та пошук знижок;
- оновлення інформації про знижку;
- видалення знижки;
- пошук та перегляд змін барист;
- перегляд інформації про зміну;
- видалення робочої зміни;
- перегляд та пошук інгредієнтів на точці;
- додавання інгредієнтів на точку;
- вилучення інгредієнтів з точки;

- оновлення інформації про інгредієнт;
- перегляд та пошук точок продажу;
- додавання точок продажу;
- оновлення інформації про точку продажу;
- вилучення точки продажу;
- отримання інформації про пов'язані сутності з точкою(кількість інгредієнтів, позицій та барист);
- оновлення інформації про шага відтворення рецепту позиції меню;
- оновлення інформації про інгредієнти рецепту позиції меню;
- отримання статистики про популярні позиції на точці;
- отримання детальної статистики про прибуток та витрати на точці;
- отримання статистики про загальні, по усім точкам, витрати та прибутки;
- отримання детальної статистики про барист;
- отримання статистики про усіх барист;
- отримання статистики про витрати інгредієнтів на точках;
- отримання статистики про надходження та прибутки по категоріях;
- перегляд та редагування особистої інформації;
- оновлення паролю.

Усі перелічені функції дозволяють здійснювати повноцінне керування усіма складовими мережі кав'ярень, починаючи від категорій та завершуючи точками продажів та баристами.

2.5 Висновки за розділом

Таким чином, в даному розділі було визначено основні засоби та інструменти розробки, сформовано складові структур клієнтської та серверної частини відповідно до вибраних засобів, що є необхідним для вирішення поставленої задачі.

Мовою програмування для розробки даного програмного забезпечення обрано JavaScript. Для вирішення завдань клієнтської частини буде задіяно бібліотеку React у поєднанні з Redux та React Router. Сервер буде побудовано на базі фреймворку Nest.js. Архітектурою для зв'язку між клієнтом та сервером обрано REST.

Роботу з базою даних буде забезпечено СКБД PostgreSQL.

Додатково, у розділі було окреслено функціональні вимоги для ролей бариста та адміністратор, сформовано основний набір функцій, до яких повинна мати доступ кожна з наведених ролей.

3 РОЗРОБКА ПРОГРАМИ

3.1 Реалізація серверної частини

3.1.1 Особливості функціонування системи

Програмне забезпечення для управління та обліку мережею кав'ярень є повноцінною системою, яка надає можливість створювати та керувати безліччю повністю ізольованих, одна від одної, мереж.

Розглянемо концепції реалізації ізольованості мереж кав'ярень у системі.

Унікальною сутністю є адміністратор(уособлює у собі мережу), який має доступ лише до власних барист та кав'ярень, що дозволяє об'єднувати їх в одне ціле. Додатково за адміністратором закріплюються категорії, так як вони поширюються на усю мережу, незалежно від обраної кав'ярні.

На рис. 3.1 відображено ER-діаграму відношень між сутностями у системі.

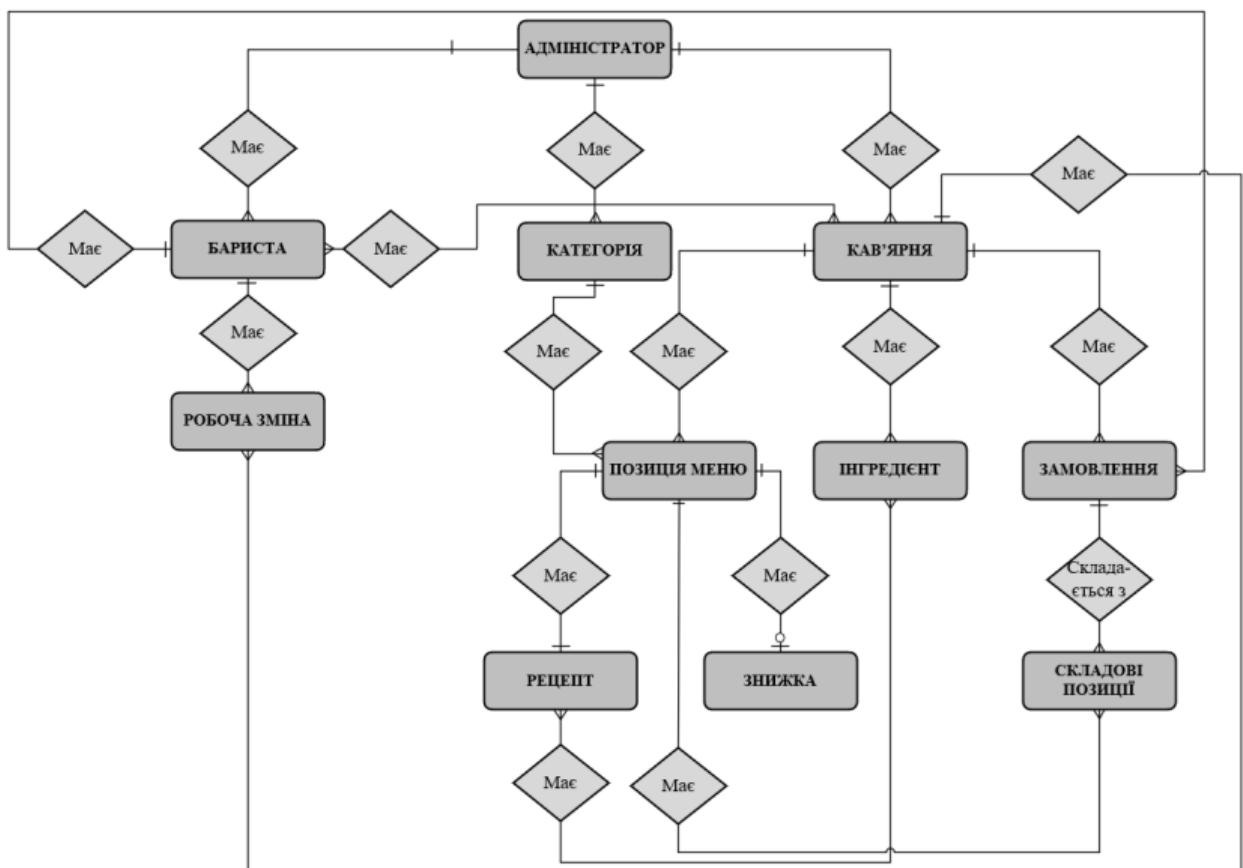


Рисунок 3.1 – ER-діаграма відношень між сутностями

За кожною кав'ярнею закріплюється перелік позицій меню, інгредієнтів та замовлень.

За кожною позицією меню закріплюються рецепт та знижка.

За кожним замовленням закріплюється перелік складових позицій.

За кожним баристою закріплюється перелік робочих змін.

Цей перелік відношень дозволяє повністю ізолювати мережу кав'ярень, виключаючи будь-які ситуації з перетином інформації між мережами.

3.1.2 Проєктування бази даних

При проєктуванні БД було впроваджено ряд відношень між сутностями, які дозволяють підтримувати можливість ізолюваності.

У табл. 3.1 наведено перелік сутностей, що наявні у системі з зазначенням полів та призначення сутності.

Таблиця 3.1 – Перелік сутностей

№	Сутність	Перелік полів	Призначення
1	2	3	4
1	admin	id, name, surname, email, password, isEmailConfirmed	Інформація про облікові записи адміністраторів
2	menu position	id, name, description, price	Інформація про позиції меню
3	category	id, title	Інформація про категорії
4	position discount	id, amount, startedAt, endAt	Інформація знижки на позиції меню
5	recipe	id, stepsToReproduce, ingredients	Інформація про рецепти позиції меню

Продовження таблиці 3.1

1	2	3	4
6	Ingredient	id, name, quantity, price	Інформація про інгредієнти
7	barista	id, name, surname, email, phoneNumber, password, fixedHourRate, percentFromEarnings, dateOfEmployment	Інформація про облікові записи барист
8	point	id, name, address, description, workingHours, pointMoney	Інформація про кав'ярні
9	order	id, createdAt, status, totalAmount, paymentMethod, receivedAmount, costOfIngredients	Інформація про замовлення
10	order position	id, menuPosition, quantity	Інформація про складові позиції замовлень
11	shift	id, status, time, baristaSalary	Інформація про робочі зміни

На рис. 3.2 представлено детальну схему БД системи з наведенням усіх відношень, що існують між сутностями та типізацією полів сутностей.

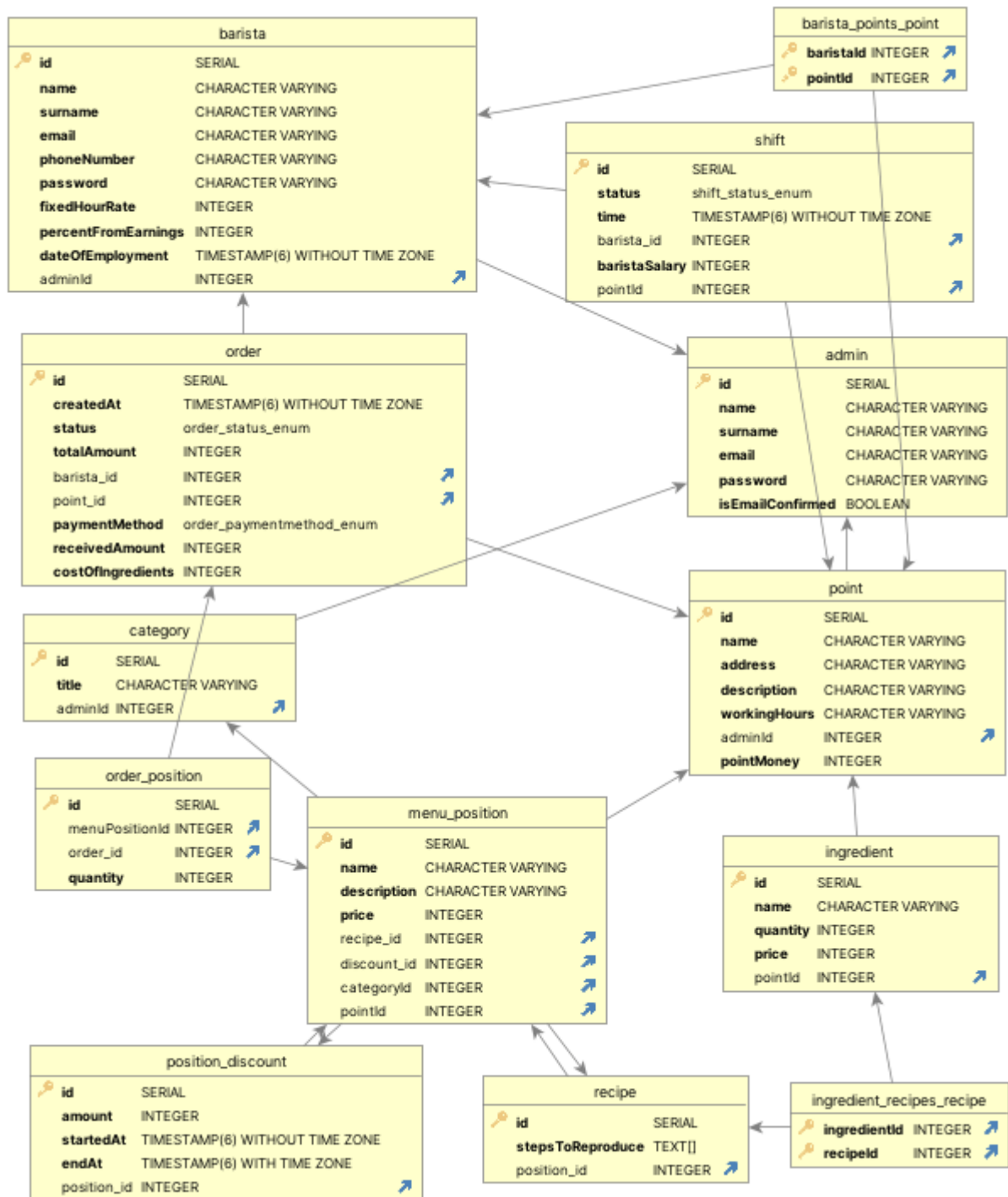


Рисунок 3.2 – Схема БД з усіма відношеннями

Як було зазначено вище, ізолюваність системи реалізовано шляхом побудови відношень між сутностями, далі їх буде описано більш детально:

– сутність «admin» має відношення 1 до багатьох з сутностями «barista», «point» та «category», тому що адміністратор може мати безліч барист, кав'ярень та категорій;

– сутність «menu position» має відношення 1 до 1 з сутностями «recipe», «position discount», тому що одній позиції меню належить один рецепт та лише одна знижка;

– сутність «category» має відношення 1 до багатьох з сутністю «menu position», тому що позиції меню можуть мати однакову категорію;

– сутність «ingredient» має відношення багато до багатьох з сутністю «recipe», тому що кожен рецепт може включати будь-який з інгредієнтів, та вони можуть повторюватися у безлічі рецептів;

– сутність «point» має відношення один до багатьох з сутностями «order», «ingredient», «shift» та «menu positions», тому що ці сутності є унікальними в межах однієї кав'ярні;

– сутність «barista» має відношення один до багатьох з сутностями «order» та «shift», бо один бариста може створити безліч замовлень та відпрацювати безліч робочих змін. Додатково, «barista» має відношення багато до багатьох із сутністю «point», так як один бариста може працювати у безлічі кав'ярень мережі, та за декількома кав'ярнями можуть бути закріплені однакові баристи;

– сутність «order position» має відношення багато до 1 з сутністю «order», тому що замовлення може складатися з безлічі різних позицій. Також, «order position» має відношення багато до багатьох з «menu position», це дозволяє вирішити питання присутності декількох однакових позицій в одному замовленні.

Для керування БД у проєкті буде використовуватися TypeORM [12].

Це потужний інструмент, який дозволяє працювати з реляційними базами даних, використовуючи об'єктно-орієнтований підхід. TypeORM надає зручний інтерфейс для взаємодії з БД, який дозволяє легко створювати моделі даних, запити та виконувати міграції.

Крім того, TypeORM підтримує обрану систему керування базами даних, а саме PostgreSQL. Цей інструмент є надійним і широко використовується в спільноті розробників для впровадження складних і ефективних програмних

рішень, які працюють з базами даних. Використання TypeORM допоможе забезпечити ефективне та надійне керування даними в системі.

Сумісність з Nest.js, дозволяє TypeORM здійснювати легку інтеграцію в архітектуру проєкту.

3.1.3 Опис системи маршрутизації

Фундаментальною основою будь-якого серверу є система маршрутизації. Маршрутизація сервера відноситься до процесу визначення та керування маршрутами клієнтських запитів до різних частин серверної програми. Це важливий елемент вебсервера, який визначає, як сервер обробляє різні URL-адреси та відповідає на них [13].

Для розроблюваного програмного забезпечення маршрутизація дозволяє здійснення CRUD операцій шляхом обробки різних типів запитів:

- GET – дозволяє здійснювати читання інформації;
- POST – дозволяє надсилати інформацію на сервер;
- PATCH – дозволяє здійснювати оновлення інформації;
- DELETE – дозволяє видаляти інформацію.

Фактично, усі розроблені маршрути можна поділити на наступні групи:

- для барист(префікс «barista»);
- для адміністраторів(префікс «admin»);
- для здійснення процесів пов'язаних з реєстрацію, аутентифікації та відновленням даних(префікс «auth»);
- для отримання статистики(префікс «statistics»).

У табл. 3.2 перераховано розроблені маршрути для барист, разом із їх функціональним призначенням та типом запиту.

Таблиця 3.2 – Розроблені маршрути серверу для барист

№	Маршрут	Тип	Призначення
1	/api/barista	PATCH	Оновлення інформації про баристу
2	/api/barista/menu	GET	Отримання меню для поточної кав'ярні баристи
3	/api/barista/orders	POST	Створення нового замовлення
5	/api/barista/orders/{orderID} /ready	GET	Встановлення статусу замовленню «Готовий»
6	/api/barista/shift/start /{pointID}	GET	Розпочати робочу зміну у кав'ярні
7	/api/barista/shift/end /{pointID}	GET	Завершити робочу зміну у кав'ярні
8	/api/barista/ingredients	GET	Отримати усі інгредієнти у поточній кав'ярні
9	/api/barista/ingredients /{ingredientID}/onPoint /{pointID}	POST	Оновити кількість інгредієнтів у поточній кав'ярні
10	/api/barista/points/{pointID} /takeMoney	POST	Отримати гроші з рахунку кав'ярні
11	/api/barista/points	GET	Отримати перелік кав'ярень де працює бариста
12	/api/barista/me	GET	Отримати інформацію про себе(для баристи)
13	/api/barista/lastpoint	GET	Отримати останню робочу кав'ярню баристи

У табл. 3.3 перераховано розроблені маршрути для адміністраторів, разом із їх функціональним призначенням та типом запиту.

Таблиця 3.3 – Розроблені маршрути серверу для адміністраторів

№	Маршрут	Тип	Призначення
1	2	3	4
1	/api/admin	PATCH	Оновити інформацію про адміністратора
2	/api/admin/categories	POST	Створити нову категорію
3	/api/admin/categories	GET	Отримати усі категорії
4	/api/admin/categories/{id}	GET	Отримати інформацію про категорію
5	/api/admin/categories/{id}	PATCH	Оновити інформацію про категорію
6	/api/admin/categories/{id}	DELETE	Видалити категорію
7	/api/admin/baristas	POST	Додати баристу
8	/api/admin/baristas	GET	Отримати список усіх барист
9	/api/admin/baristas/names	GET	Отримати перелік імен усіх барист
10	/api/admin/baristas/{id}	GET	Отримати інформацію про баристу
11	/api/admin/baristas/{id}	PATCH	Оновити інформацію про баристу
12	/api/admin/baristas/{id}	DELETE	Видалити баристу
13	/api/admin/barista /{baristaID}/setpoint /{pointID}	POST	Призначити баристу у кав'ярню
14	/api/admin/barista /{baristaID}/removePoint /{pointID}	POST	Вилучити баристу з кав'ярні
15	/api/admin/points	POST	Додати кав'ярню
16	/api/admin/points	GET	Отримати список усіх кав'ярень
17	/api/admin/points/forBarista /{id}	GET	Отримати перелік кав'ярень для баристи

Продовження таблиці 3.3

1	2	3	4
18	/api/admin/points/only	GET	Отримати інформацію виключно про кав'ярню(без супутньої інформації)
19	/api/admin/points/names	GET	Отримати перелік імен кав'ярень
20	/api/admin/points/ingredients	GET	Отримати інформацію про кав'ярні з інгредієнтами
21	/api/admin/points/short	GET	Отримати інформацію про кав'ярню з замовленнями
22	/api/admin/points/{id}	GET	Отримати повну інформацію про кав'ярню
23	/api/admin/points/{id}	PATCH	Оновити інформацію про кав'ярню
24	/api/admin/points/{id}	DELETE	Видалити кав'ярню
25	/api/admin/ingredients/{pointID}	POST	Створити новий інгредієнт
26	/api/admin/ingredients/{pointID}	GET	Отримати інформацію про усі інгредієнти у кав'ярні
27	/api/admin/ingredients	GET	Отримати перелік усіх інгредієнтів
28	/api/admin/ingredients/{id}	PATCH	Оновити інформацію про інгредієнт
29	/api/admin/ingredients/{id}	DELETE	Видалити інгредієнт
30	/api/admin/positions/{pointID}	POST	Створити позицію меню
31	/api/admin/positions/{pointID}	GET	Отримати перелік позицій меню у кав'ярні

Продовження таблиці 3.3

1	2	3	4
32	/api/admin/positions /{pointID}/recipe /{positionID}	GET	Отримати рецепт позиції меню
33	/api/admin/positions /{pointID}/update /{positionID}	PATCH	Оновити позицію меню
34	/api/admin/positions /{pointID}/delete /{positionID}	DELETE	Видалити позицію меню
35	/api/admin/shifts/point /{pointID}	GET	Отримати перелік робочих змін на точці
36	/api/admin/shifts/{shiftID}	DELETE	Видалити робочу зміну
37	/api/admin/shifts/barista /{baristaID}	GET	Отримати перелік змін для баристи
38	/api/admin/shifts /{baristaID}/state	GET	Отримати поточний стан зміни баристи(розпочата/завершена)
39	/api/admin/discounts /{positionID}	POST	Створити знижку на позицію меню
40	/api/admin/discounts /{positionID}	PATCH	Оновити інформацію про знижку
41	/api/admin/discounts /{positionID}	DELETE	Видалити знижку
42	/api/admin/discounts	GET	Отримати перелік усіх знижок
43	/api/admin/orders/{orderID} /list	GET	Отримати інформацію про склад замовлення
44	/api/admin/orders	GET	Отримати усі замовлення
45	/api/admin/orders/{pointID}	GET	Отримати перелік замовлень на точці

Продовження таблиці 3.3

1	2	3	4
46	/api/admin/orders/byBarista/{baristaID}	GET	Отримати перелік замовлення для баристи
47	/api/admin/orders/{orderID}/info	GET	Отримати інформацію про замовлення
48	/api/admin/orders/barista/{baristaID}/today	GET	Отримати перелік замовлень для баристи за поточний день
49	/api/admin/orders/{orderID}	DELETE	Видалити замовлення

У табл. 3.4 перераховано розроблені маршрути для аутентифікації, реєстрації та супутніх процесів, разом із їх функціональним призначенням та типом запиту.

Таблиця 3.4 – Розроблені маршрути серверу для групи «auth»

№	Маршрут	Тип	Призначення
1	/api/auth/registration	POST	Реєстрація нового адміністратора
2	/api/auth/confirming/{token}	GET	Підтвердження пошти адміністратора
3	/api/auth/password/restore	POST	Запит на відновлення паролю
4	/api/auth/password/update/{token}	POST	Оновлення паролю для адміністратора
5	/api/auth/login	POST	Аутентифікація
6	/api/auth/profile	GET	Отримання профілю

У табл. 3.5 перераховано розроблені маршрути для отримання статистичних показників, разом із їх функціональним призначенням та типом запиту.

Таблиця 3.5 – Розроблені маршрути серверу для отримання статистики

№	Маршрут	Тип	Призначення
1	/api/statistics/positions /{pointID}	POST	Популярність позицій у певній кав'ярні
2	/api/statistics/points /{pointID}	POST	Витрати та доходи в окремій кав'ярні
3	/api/statistics/points	POST	Витрати та доходи в усіх кав'ярнях
4	/api/statistics/baristas /{baristaID}	POST	Робочі зміни та заробітна платня баристи
5	/api/statistics/baristas	POST	Робочі зміни та заробітна платня всіх барист
6	/api/statistics/ingredients /{pointID}	POST	Витрати інгредієнтів у кав'ярні
7	/api/statistics/categories	POST	Кількість продажів та дохід за категорією по усій мережі

Велика кількість схожих маршрутів за призначенням обумовлена тим, що розроблювана система потребує наявності актуальних даних у певний момент часу, що дозволяють зробити запити, які надають максимально оптимізовану відповідь. Тому під час роботи системи користувачі завжди матимуть актуальну інформацію з найменшим часом очікування.

Таким чином, створені маршрути забезпечують гнучкість та оптимальну завантаженість системи. Їх можна легко використовувати та інтегрувати з клієнтським додатком, що забезпечує високу ефективність і швидкість взаємодії між клієнтом і сервером.

3.1.4 Опис трьох шарового захисту маршрутів

Будь-яка система має передбачати захист даних від несанкціонованого доступу, розроблювана система не є винятком.

На рис. 3.3 представлено візуалізацію шарів захисту, які дозволяють убезпечити дані від несанкціонованого доступу.

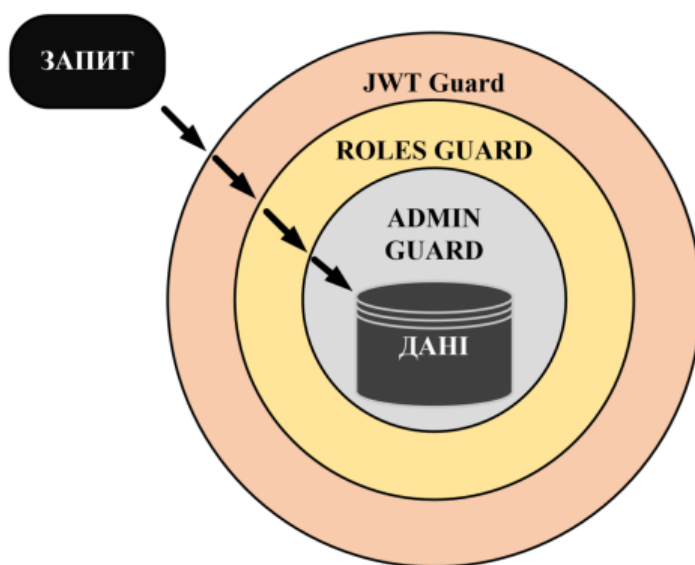


Рисунок 3.3 – Візуалізація шарів захисту застосунку

Як видно з рисунку, при надходженні запиту відбувається по етапне проходження трьох шарового захисту, лише після чого надається доступ до даних.

Першочергово, слід зазначити, що два з трьох шарів базуються на одній з концепцій Nest.js, а саме «Guards».

У Nest.js «guards» використовуються для захисту маршрутів, перевірки доступу до ресурсів або обробки запитів до сервера, перш ніж вони досягнуть контролера або обробника запитів. Guards — це тип проміжного ПЗ, який можна використовувати для фільтрації запитів на основі різних умов, таких як автентифікація, авторизація, перевірка ролей користувача тощо [14].

Створення відбувається шляхом реалізації спеціального класу з методом `canActivate`, який перевіряє певні умови доступу. Нижче представлено фрагмент коду реалізації «guards» на основі ролей користувачів.

```
@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const allowedRoles =
this.reflector.getAllAndOverride<Roles[]>(ROLES_NAME, [
      context.getHandler(),
      context.getClass(),
    ]);
    if (!allowedRoles) {
      return true;
    }
    const { user } = context.switchToHttp().getRequest();
    return !!allowedRoles.find((role) => user.role === role);
  }
}
```

Кожен з шарів має власну відповідальність у захисті:

- JWT Guard: шар захисту, заснований на «Guards», який здійснює авторизацію користувача шляхом вилучення JWT-токену [15] з переданого у запиті хедеру, який несе у собі інформацію про ідентифікатор, ім'я, роль та пошту користувача(у випадку з баристою ще ідентифікатор адміністратора) та додавання його у тіло запиту;

- Roles Guard: наступним йде шар, також заснований на «Guards», який вже вилучає з тіла інформацію про роль користувача та здійснює перевірку на доступність до обраного маршруту. Заздалегідь, до кожного маршруту у мета дані було додано інформацію про доступність відповідним ролям;

- Admin Guard: кінцевим етапом, коли доступ до маршруту надано, під час здійснення запиту до БД відбувається додаткова перевірка на наявність такого запису у межах мережі кав'ярень(до якої належить користувач),

шляхом включення інформації про ідентифікатор адміністратора(фактично мережі).

Таким чином, у разі не проходження першого етапу буде надіслано відповідь з помилкою «401 Unauthorized» та відмовлено у доступі, в протилежному випадку відбудеться розшифровка токена та додавання до запиту даних після чого відбудеться перевірка приналежності до дозволених ролей маршруту, а далі до мережі, дані з якої потребують дії.

3.1.5 Опис реалізованих сервісів

В ході опису маршрутизації застосунку, було зазначено про наявність чотирьох груп маршрутів, які відповідають чотирьом контролерам, кожен з яких здійснює маніпулювання відповідним сервісом. Лише сервіс рецептів ніяк не взаємодіє з контролерами через те, що здебільшого взаємодія відбувається через сервіс позицій меню. Далі представлено схему відношень між сервісами та контролерами (див.рис. 3.4).

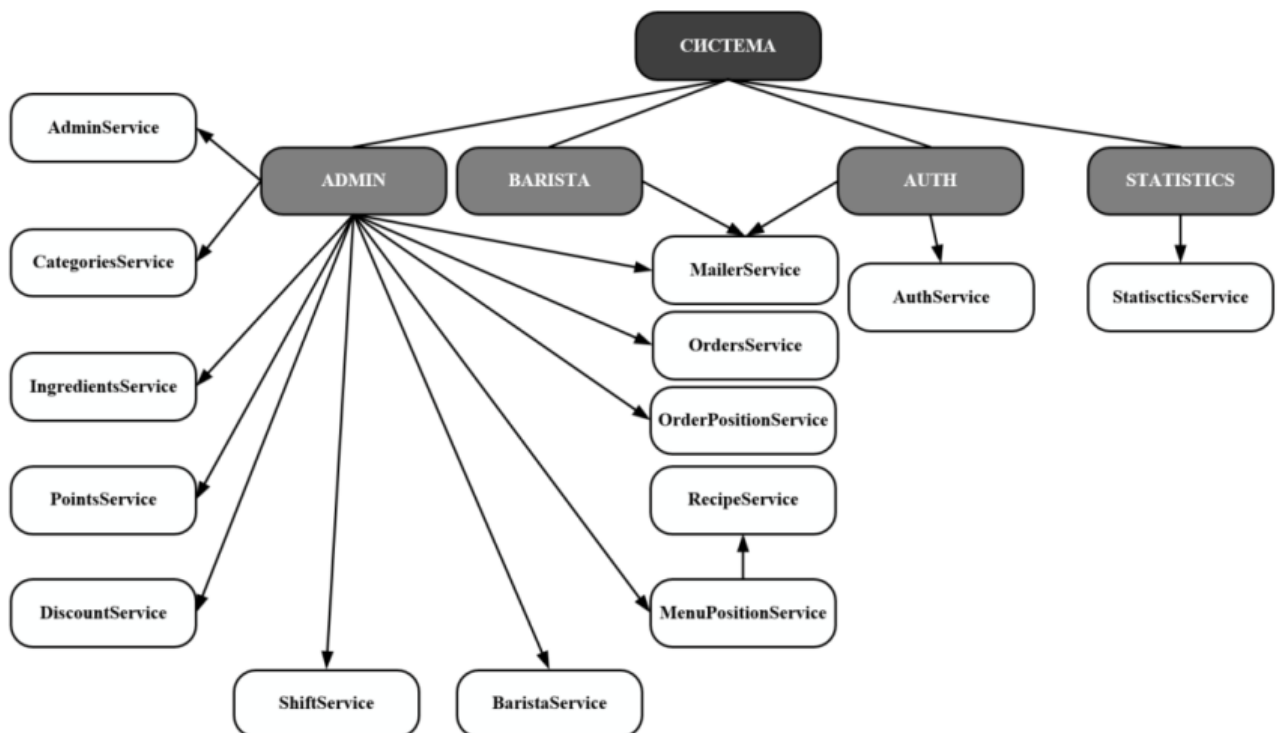


Рисунок 3.4 – Схема відношень між сервісами та контролерами

У табл. 3.6 наведено перелік сервісів з описом їх призначення.

Таблиця 3.6 – Перелік сервісів та їх призначення

№	Назва сервісу	Призначення
1	2	3
1	AuthService	Сервіс дозволяє здійснювати операції пов'язані з реєстрацією адміністраторів, підтвердженням електронної пошти, оновленням паролю та здійснення авторизації
2	PointsService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про кав'ярні
3	DiscountService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про знижки на позиції меню
4	ShiftService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про робочі зміни
5	BaristaService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про барист
6	MenuPositionService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про позиції меню
7	RecipeService	Сервіс дозволяє здійснювати операції пов'язані з створенням рецептів
8	OrderPositionService	Сервіс дозволяє здійснювати операції пов'язані з створенням та отриманням переліку позицій до замовлення

Продовження таблиці 3.6

1	2	3
9	MailerService	Сервіс дозволяє здійснювати операції пов'язані з відправку повідомлень на пошту з актуальними даними для авторизації, підтвердженням та зміною паролю
10	OrdersService	Сервіс дозволяє здійснювати операції пов'язані з створенням, отриманням(відповідно до вказаних параметрів) та видаленням замовлень
11	StatisticsService	Сервіс дозволяє здійснювати підрахунок статистичних показників за різним процесами діяльності кав'ярні або усієї мережі
12	AdminService	Сервіс дозволяє здійснювати операції пов'язані з обліковими записами адміністраторів
13	CategoriesService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про категорії
14	IngredientsService	Сервіс дозволяє здійснювати операції пов'язані з створенням, оновленням, отриманням та видаленням інформації про інгредієнти

Таким чином, створенні сервіси забезпечують реалізацію функціональних вимог до розроблюваної системи.

3.1.6 Опис алгоритмів алгоритмів функціонування

3.1.6.1 Алгоритм реєстрації

Алгоритм процесу реєстрації наведено на рис. 3.5.

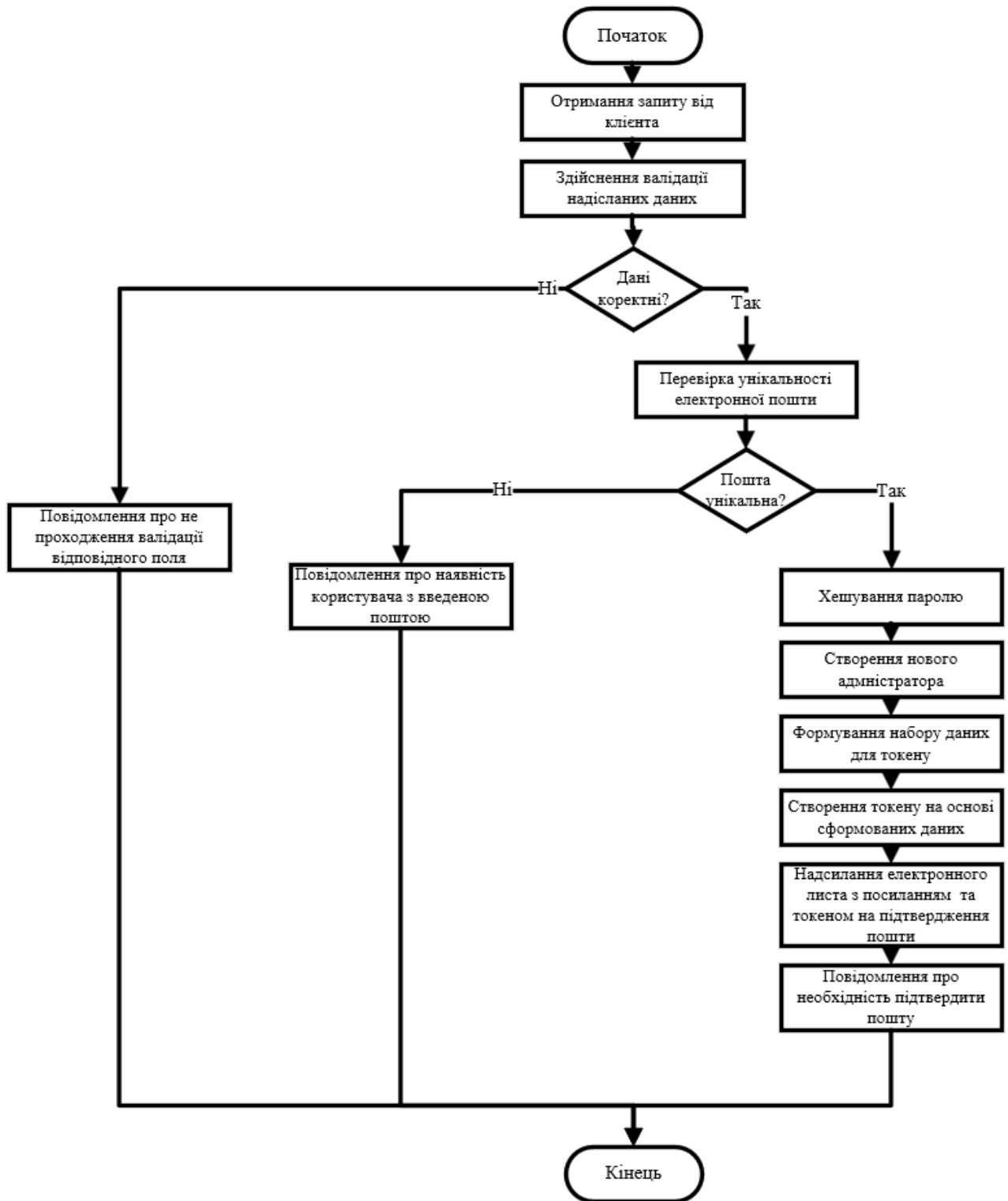


Рисунок 3.5 – Алгоритм процесу реєстрації

3.1.6.2 Алгоритм аутентифікації

Алгоритм процесу аутентифікації наведено на рис. 3.6.

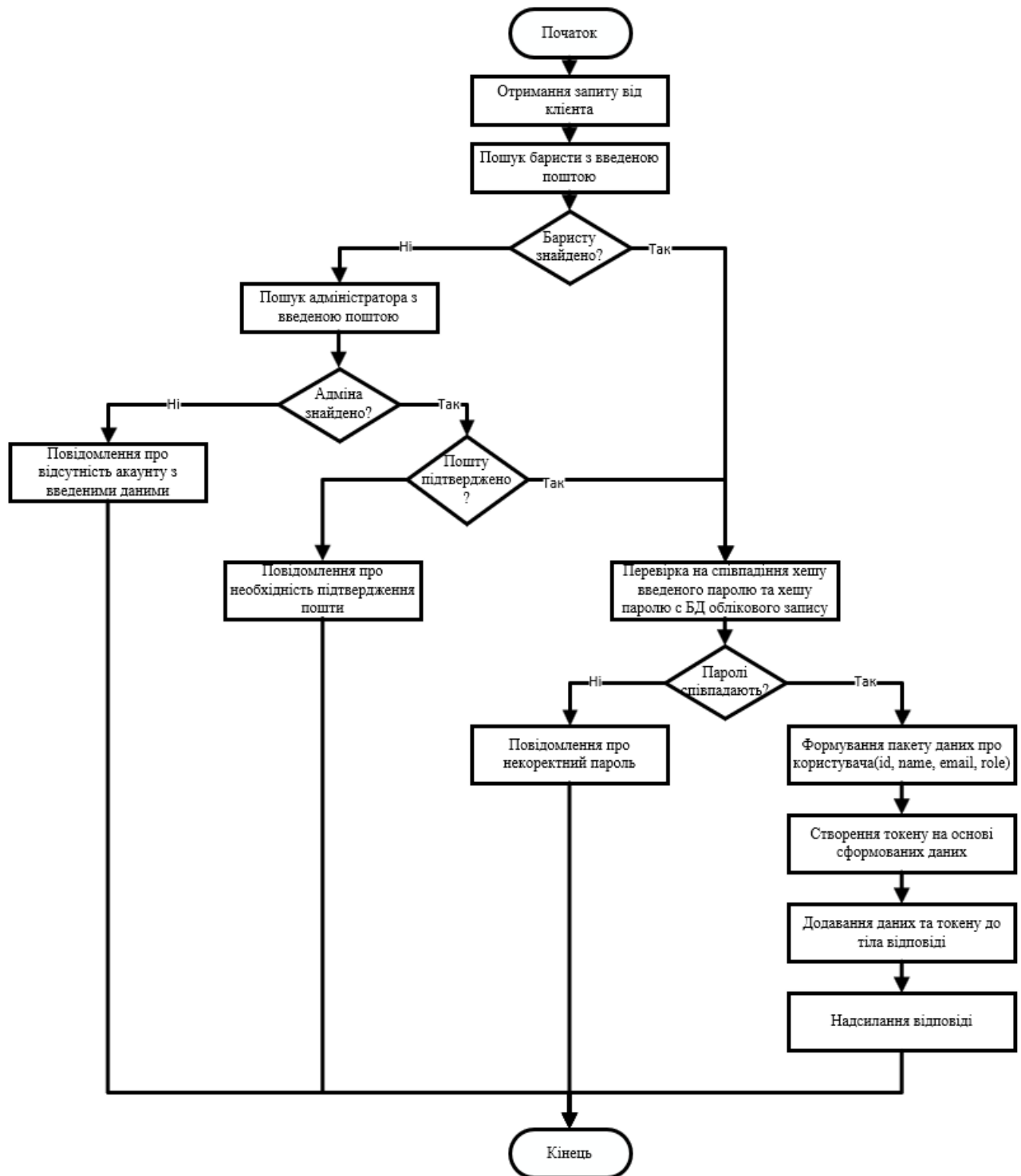


Рисунок 3.6 – Алгоритм процесу аутентифікації

3.1.6.3 Алгоритм процесів створення, оновлення та видалення сутностей

Алгоритми процесів створення, оновлення та видалення усіх сутностей, окрім адміністраторів та робочої зміни, передбачає однакову логіку за винятком набору полів для створення та ключового поля, яке має бути унікальним. На рис. 3.7 наведено процес додавання кав'ярні.

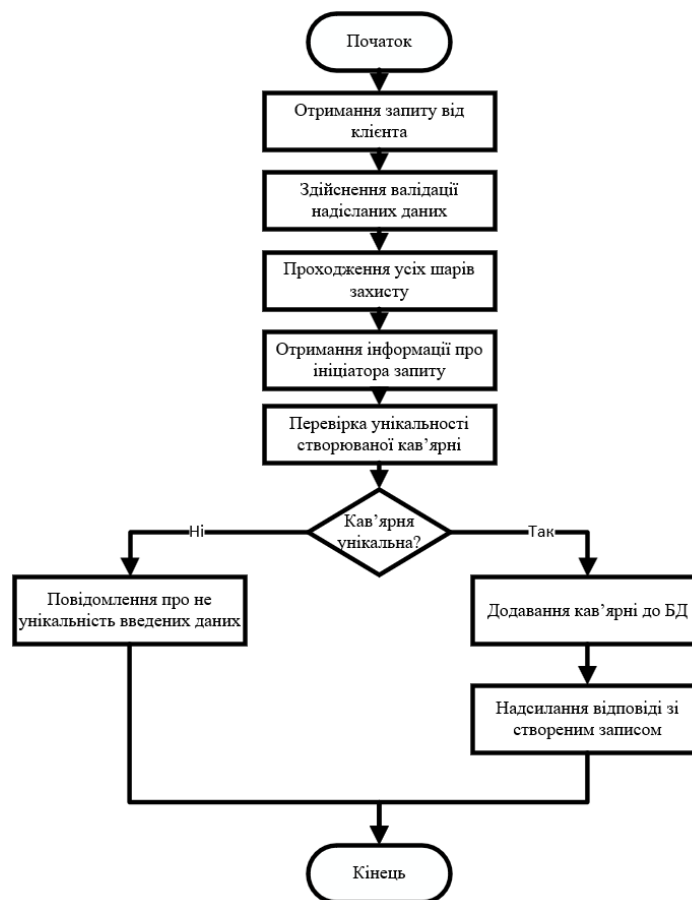


Рисунок 3.7 – Алгоритм додавання кав'ярні

3.1.6.4 Алгоритм оновлення інформації про облікові записи

Алгоритм оновлення інформації передбачає однакову логіку як для баристи, так і для адміністратора.

На рис. 3.8 наведено алгоритм оновлення інформації про обліковий запис адміністратора.

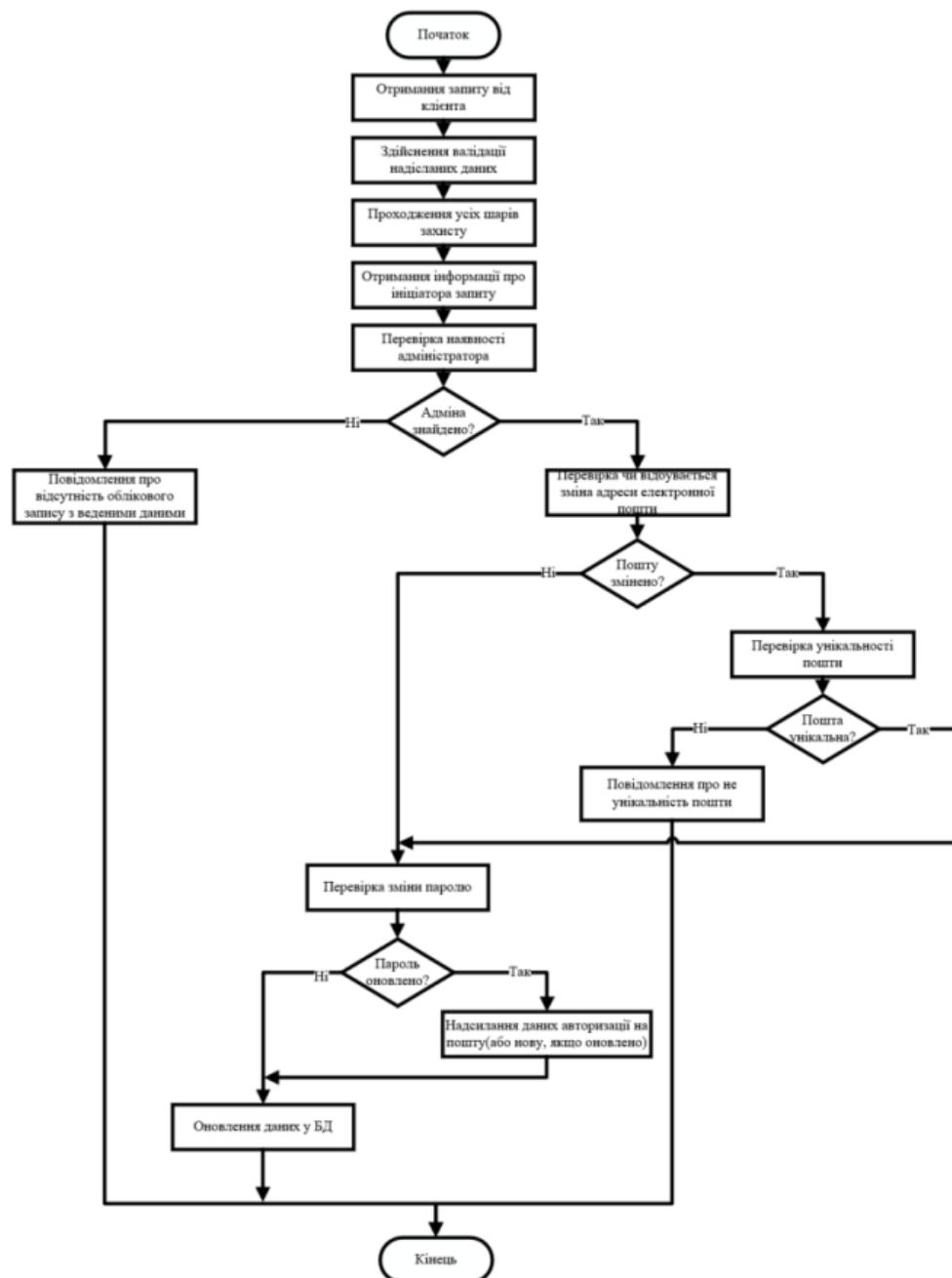


Рисунок 3.8 – Алгоритм оновлення інформації про обліковий запис адміністратора

3.2 Реалізація клієнтської частини

3.2.1 Принцип функціонування

Принцип роботи клієнтської частини програми заснований на виконанні асинхронних запитів до сервера, що дозволяє швидко і ефективно обробляти дані.

Асинхронність дозволяє уникнути блокування клієнтської програми під час очікування відповіді від сервера. Кожен запит обробляється окремо, і клієнт може продовжувати виконувати інші дії, не чекаючи завершення запиту. Це робить роботу з даними більш плавною та ефективною для кінцевого користувача.

Однією з ключових переваг асинхронного підходу є забезпечення актуальності даних. Клієнт може надсилати запити на оновлення інформації на сервер у режимі реального часу, що дозволяє користувачам завжди працювати з актуальними даними.

Приклад подібного функціонування представлено у вигляді діаграми послідовності процесу створення замовлення з розрахунком готівкою на рис. 3.9.

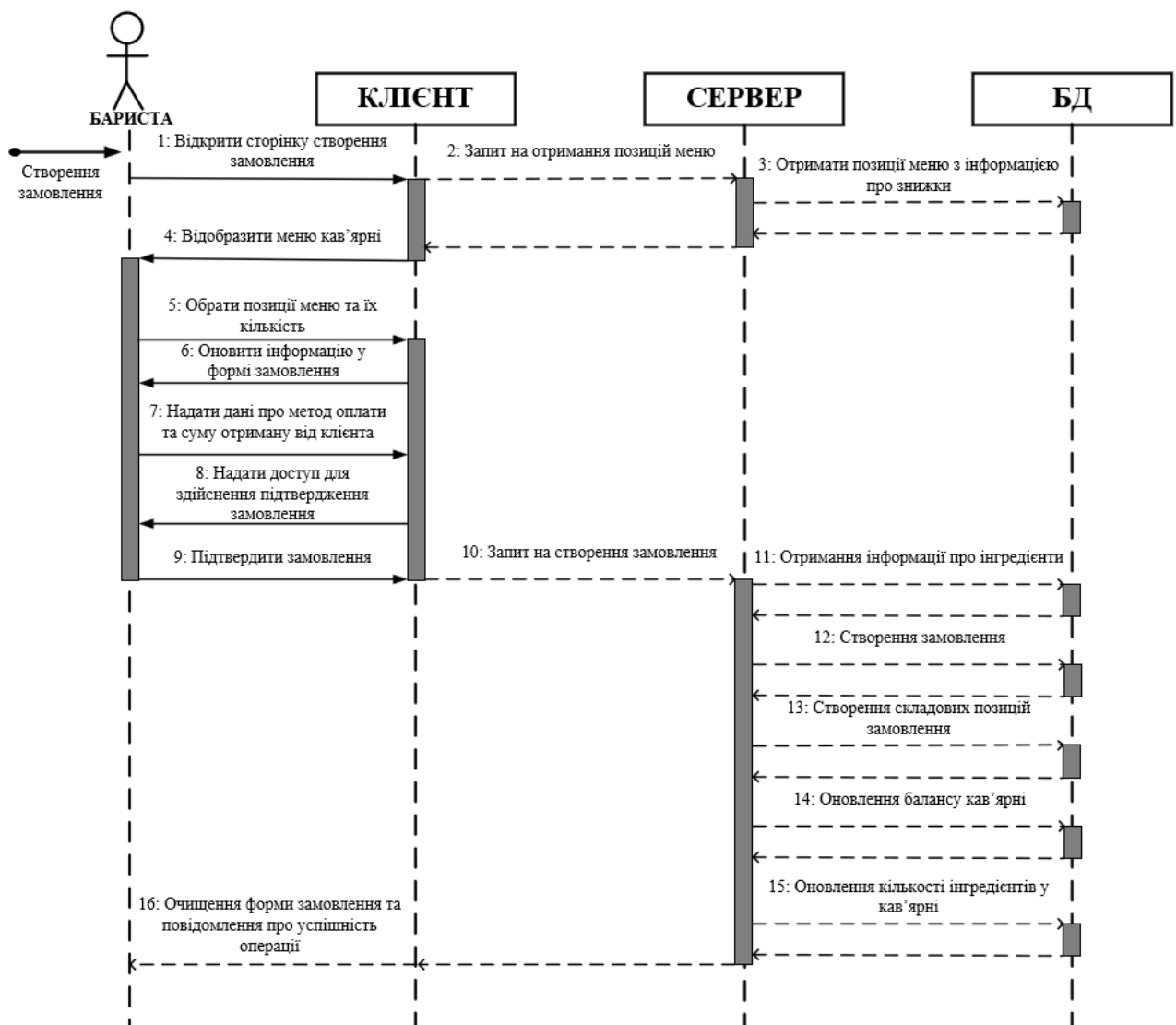


Рисунок 3.9 – Діаграма послідовності «Створення замовлення»

3.2.2 Опис системи маршрутизації

Система маршрутизації клієнської частини складається з трьох груп маршрутів, які наведено нижче:

- «admin»: відповідає за надання функціоналу авторизованим користувачам у ролі адміністратор;
- «barista»: надає доступ до функціоналу баристи користувачам з відповідною роллю;
- «auth»: надає можливість здійснення усіх процесів пов'язаних з аутентифікацією та реєстрацією.

Повний перелік шляхів маршрутизації та відповідних компонентів сторінок наведено у табл. 3.7.

Таблиця 3.7 – Шляхи маршрутизації та компоненти клієнської частини

№	Шлях	Призначення	Компонент
1	2	3	4
1	/auth/login	Сторінка аутентифікації	Login
2	/auth/registration	Сторінка реєстрації	Registration
3	/auth/confirming/:token	Сторінка підтвердження пошти	Confirming-Email
4	/change/:token	Сторінка оновлення паролю для адміністраторів	Change-Password
5	/auth/restore	Сторінка створення запиту на відновлення паролю	Restore-Password
6	/auth/select	Сторінка обрання кав'ярні для початку робочої зміни баристи	SelectPoint
7	/barista	Сторінка інформації про себе	AboutMe

Продовження таблиці 3.7

1	2	3	4
8	/barista/ingredients	Сторінка роботи з інгредієнтами	IngredientsInfo
9	/barista/createOrder	Сторінка створення замовлення	CreateOrder
10	/barista/point	Сторінка роботи з кав'ярнею	CurrentPoint-Info
11	/barista/recipes	Сторінка перегляду рецептів	Recipes
12	/admin	Сторінка інформації про себе	Admin
13	/admin/baristas	Сторінка роботи з обліковими записами барист	Baristas
14	/admin/categories	Сторінка роботи з категоріями	Categories
15	/admin/discounts	Сторінка роботи зі знижками на позиції меню	Discounts
16	/admin/ingredients	Сторінка роботи з інгредієнтами	Ingredients
17	/admin/positions	Сторінка роботи з позиціями меню	MenuPositions
18	/admin/orders	Сторінка роботи з замовленнями	Orders
19	/admin/points	Сторінка роботи з кав'ярнями	Points
20	/admin/shifts	Сторінка роботи із робочими змінами барист	Shifts

Продовження таблиці 3.7

1	2	3	4
21	/admin/statistics/popularity	Сторінка статистики про популярність позицій у певній кав'ярні	Popularity
22	/admin/statistics/expenses	Сторінка статистики про витрати та доходи в окремії кав'ярні	Expenses
23	/admin/statistics/common	Сторінка статистики про витрати та доходи в усіх кав'ярнях	Common
24	/admin/statistics/barista	Сторінка статистики про робочі зміни та заробітна платня баристи	BaristaSalary-Shifts
25	/admin/statistics/allBarista	Сторінка статистики про робочі зміни та заробітна платня всіх барист	AllBarista
26	/admin/statistics/consumption	Сторінка статистики про витрати інгредієнтів у кав'ярні	Consumption
27	/admin/statistics/categories	Сторінка статистики про кількість продажів та дохід за категорією по усій мережі	Categories-Incomes
28	/	Головна сторінка для отримання довідкової інформації про систему	MainPage

У разі спроби доступ до маршрутів, які відсутні у наведеному вищій переліку передбачено оброблювач, який виводить повідомлення про помилку та пропонує перехід на головну сторінку (див.рис. 3.10).

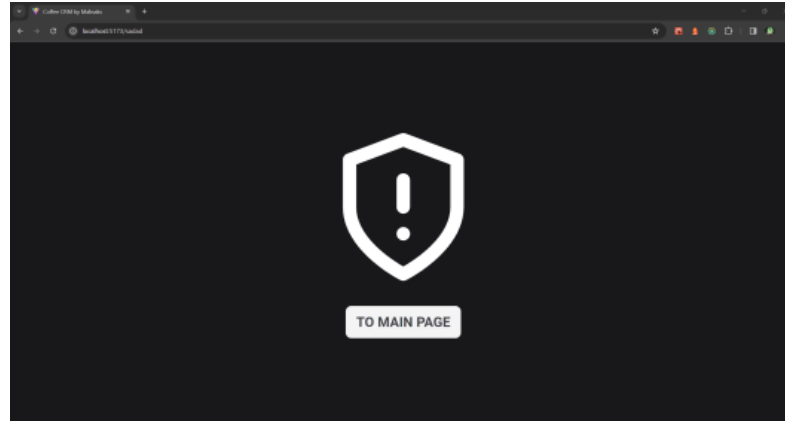


Рисунок 3.10 – Приклад спроби переходу по некоректному маршруту

Таким чином, було побудовано систему маршрутів, яка забезпечує максимальну зручність та надійність використання.

3.2.3 Розробка інтерфейсу

При розробці інтерфейсу програмного забезпечення завжди важливо враховувати різні чинники, серед яких:

- потоки взаємодії користувачів з системою(зліва-направо, згори-до низу);
- обізнаність користувача у подібних системах: система має бути інтуїтивно зрозумілою, щоб користувач з легкістю міг розібратися як працює система без сторонньої допомоги;
- кольорова схема: кольорова гама застосунку має забезпечувати мінімальне навантаження на зорові рецептори, тому кольори застосунку мають бути здебільшого у темних тонах, лише елементи керування можуть виділятися світлими [16].

Тому, виходячи з наведених вище правил організації інтерфейсів, було створено інтерфейс у темно-сірих відтінках, що забезпечує мінімізацію напруги для очей. Базовий шаблон проєктування робочих сторінок застосунку наведено на рис. 3.11.

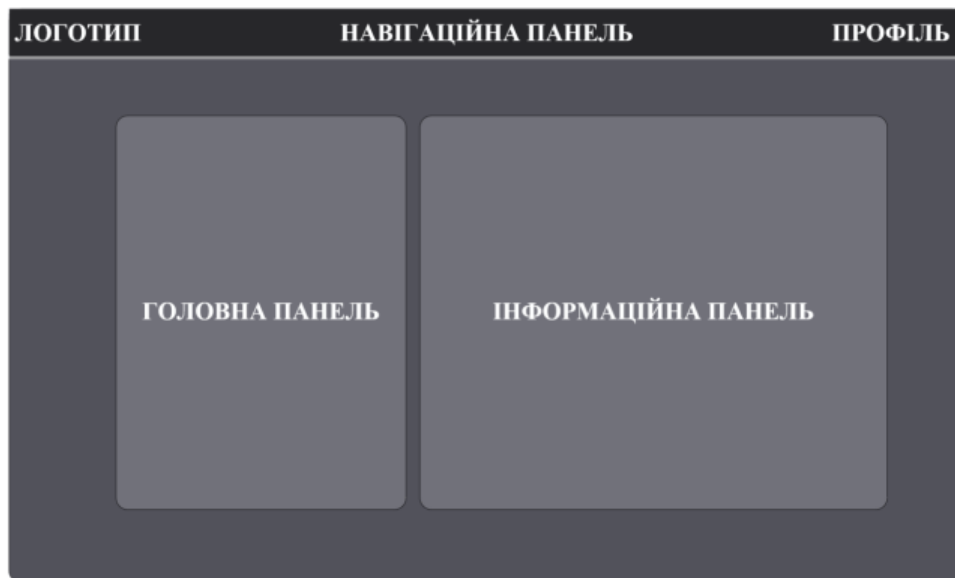


Рисунок 3.11 – Шаблон проєктування робочих сторінок

Елементи керування створено, як зазначалося вище, поєднання кольорів загальної кольорової схеми та світлих кольорів для виділення з поміж інших неактивних елементів (див.рис. 3.12).

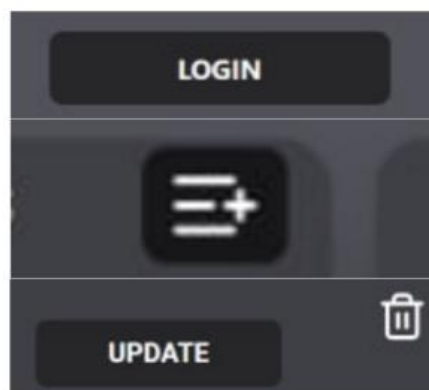


Рисунок 3.12 – Приклад реалізації елементів керування

Засновуючись на цьому шаблоні було створено ряд робочих сторінок для користувачів системи, які більш детально буде розглянуто в розділі 4.

Таким чином, в кінцевому результаті вдалося створити ряд практичних та інтуїтивно зрозумілих сторінок для усіх маршрутів клієнтської частини.

3.2.4 Опис розроблених компонентів

Під час розробки програмного забезпечення було реалізовано низку компонентів, які можна поділити на три групи: компоненти сторінок, компоненти шаблонів та допоміжні компоненти.

У таблиці 3.8 представлено опис компонентів шаблонів.

Таблиця 3.8 – Опис компонентів шаблонів

№	Назва компоненту	Опис
1	RolesRoutesProtector	Компонент вищого порядку, який отримує інформацію про поточний стан авторизації та здійснює перехід на сторінку входу у разі негативної відповіді
2	AdminLayout	Компонент вищого порядку, який зберігає навігаційну панель для адміністратора та слугує «обгорткою» для усіх сторінок адміністратора
3	BaristaLayout	Компонент вищого порядку, який зберігає навігаційну панель для баристи та слугує «обгорткою» для усіх сторінок баристи
4	AuthLayout	Компонент вищого порядку для усіх сторінок аутентифікації та реєстрації
5	AuthRoutesProtector	Компонент вищого порядку, який здійснює переадресацію після завершення процесу аутентифікації відповідно до ролі користувача
6	Statistics	Компонент вищого порядку, який слугує обгорткою для сторінок отримання статистики, надаючи можливість навігації

У таблиці 3.9 представлено опис допоміжних компонентів.

Таблиця 3.9 – Опис допоміжних компонентів

№	Назва компоненту	Опис
1	2	3
1	AdminHeader	Компонент хедеру для шаблону адміністратора
2	AdminNavigation	Компонент навігації для шаблону адміністратора
3	BaristaHeader	Компонент хедеру для шаблону баристи
4	BaristaNavigation	Компонент навігації для шаблону баристи
5	MenuItemPosition	Компонент позиції меню для сторінки створення замовлення
6	OrderPositionItem	Компонент позиції замовлення для сторінки створення замовлення
7	OrderPositionList	Компонент інформації про замовлення для сторінки створення замовлення
8	Button	Компонент кнопки
9	HeaderInfo	Компонент для здійснення виходу із системи
10	Input	Компонент поля введення
11	MessageBox	Компонент вікна повідомлення для процесів аутентифікації та авторизації
12	CreateBaristaModal	Модальне вікно для додавання баристи
13	CreateCategoryModal	Модальне вікно для створення категорії
14	CreateDiscountForPositionModal	Модальне вікно для додавання знижки на товар
15	CreateMenuItemPositionModal	Модальне вікно для створення позиції меню

Продовження таблиці 3.9

1	2	3
16	CreatingIngredientModal	Модальне вікно для створення інгредієнту
17	CreatingPointModal	Модальне вікно для додавання кав'ярні
18	OrderInformationModal	Модальне вікно для отримання інформації про замовлення
19	SetPointToBaristaModal	Модальне вікно для закріплення баристи за кав'ярнею
20	UpdateCategoryModal	Модальне вікно для оновлення категорії
21	UpdateDiscountModal	Модальне вікно для оновлення знижки на позицію меню
22	UpdateIngredientModal	Модальне вікно для оновлення інгредієнту
23	UpdateIngredientQuantityModal	Модальне вікно для оновлення кількості інгредієнту на точці
24	UpdateIngredientsModal	Модальне вікно для оновлення інгредієнтів у рецепті
25	UpdateStepsMenuPositionModal	Модальне вікно для оновлення шагів відтворення у рецепті

Стилізацію компонентів здійснювалася за допомогою фреймворку TailwindCSS [17].

TailwindCSS — це потужний інструмент для створення вебінтерфейсу користувача, який містить набір готових до використання класів CSS. Tailwind дозволяє швидко й ефективно створювати компоненти стилів, не витрачаючи час на написання власних стилів CSS.

Основна ідея Tailwind полягає в тому, щоб зосередитися на атомарних класах, які відповідають певним властивостям CSS, що надає можливість змінювати зовнішній вигляд елементів, комбінувати різні класи та створювати унікальні дизайнерські рішення без необхідності писати CSS з нуля. Tailwind також забезпечує гнучкість і налаштування, дозволяючи налаштовувати стилі відповідно до конкретних потреб проєкту.

Додавання анімації появи елементів здійснювалося за допомогою засобів бібліотеки Framer Motion [18], яка надає широкий спектр анімацій для об'єктів інтерфейсу.

Таким чином, було розроблено ряд компонентів, які дозволяють оптимізувати розробку, полегшити масштабованість у майбутньому та покращити читабельність коду.

3.2.5 Опис сервісів

Для зручної взаємодії з серверною складовою системи було впроваджено ряд сервісів, які дозволяють оптимізувати зручність роботи з сервером. Далі наведено перелік сервісів та їх призначення:

- AuthService: надає доступ до запитів пов'язаних з аутентифікацією та авторизацією;
- BaristaService: надає доступ до запитів пов'язаних з роботою баристи;
- BaristasManagmentService: надає доступ адміністратору до запитів пов'язаних з баристами;
- CategoryService: надає доступ адміністратору до запитів пов'язаних з категоріями;
- DiscountsService: надає доступ адміністратору до запитів пов'язаних з знижками на позиції меню;
- IngredientsService: надає доступ адміністратору до запитів пов'язаних з інгредієнтами;

- MenuPositionService: надає доступ адміністратору до запитів пов'язаних з позиціями меню;
- OrdersService: надає доступ адміністратору до запитів пов'язаних з замовленнями;
- PointsService: надає доступ адміністратору до запитів пов'язаних з кав'ярнями;
- ShiftsService: надає доступ адміністратору до запитів пов'язаних з робочими змінами барист;
- StatisticsService: надає доступ адміністратору до запитів пов'язаних з статистичними даними.

Таким чином, завдяки використанню сервісів вдалося значно покращити читабельність коду у компонентах, де існує потреба у звертанні до серверу.

3.3 Висновки

Таким чином, у цьому розділі було представлено основні аспекти та ключові відомості стосовно розробки серверної та клієнтської частини системи. Детально описано складові кожної частини починаючи від маршрутизації та закінчуючи сервісам.

Для клієнтської частини наведено основні принципи за якими здійснювалася побудова інтерфейсу, описано призначення кожного з компонентів.

Представлено структуру БД та перелік сутностей, з визначенням відношення між ними.

Чітко визначено особливості функціонування системи.

Деталізовано систему захисту від несанкціонованого доступу до даних під час роботи з системою.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програмний продукт, розроблений в ході роботи, надає можливості здійснення обліку та керування замовленнями у мережі кав'ярень. Передбачається наявність у системі понад однієї мережі кав'ярень, які є повністю ізольованими одна від одної. Система дозволяє здійснювати ряд операцій, які розподілені відповідно до наявних ролей:

- роль адміністратора: необмежений доступ до власної мережі(серед чого, операції пов'язані зі створення, оновленням, переглядом та видаленням сутностей: бариста, категорія, позиція меню, замовлення, знижка, робоча зміна, інгредієнт, кав'ярня, рецепт), керування власним профілем та отримання статистичних даних по роботі мережі кав'ярень;

- роль баристи: здійснення пошуку та оновлення кількості інгредієнтів, створення, перегляд та зміну статусу замовлень, отримання статистичних даних про свою роботу, перегляд та зняття коштів з рахунку кав'ярні, початок та завершення робочих змін, перегляд та оновлення інформації про себе.

Наявний функціонал дозволяє вирішувати усі потреби бізнес-процесів мережі кав'ярень.

4.2 Умови використання системи

Для коректного функціонування серверної частини застосунку, апаратна частина серверу на якому розміщується ця складова має відповідати наступним вимогам:

- процесор Intel Xeon E5630 і вище(або аналогічне рішення інших виробників);
- оперативна пам'ять 32 Гб і вище;
- постійна пам'ять 1 Тб і вище.

Що стосується клієнтської частини, то вимоги обумовлюються тим, що формування та відображення клієнтської частини здійснюється на апаратній платформі клієнта, тому до нею висуваються наступні мінімальні вимоги:

- веббраузер з підтримкою ES6 та HTML5 [19];
- процесор Intel Core i3 5-покоління і вище(або аналогічне рішення інших виробників);
- оперативна пам'ять 4 Гб і вище.

4.3 Експлуатація програми

4.3.1 Здійснення процесів реєстрації та аутентифікації

4.3.1.1 Реєстрація

Початок роботи з системою відбувається шляхом переходу за відповідним посиланням, після чого відобразиться головна сторінка, яку зображено на рис. 4.1.



Рисунок 4.1 – Головна сторінка

Далі користувач, за наявності в нього облікового запису, може здійснити перехід на сторінку аутентифікації за допомогою кнопки з відповідною іконкою (див.рис. 4.1). Або ж ознайомившись з усіма перевагами системи перейти до розділу отримання доступу до системи (див.рис. 4.2).

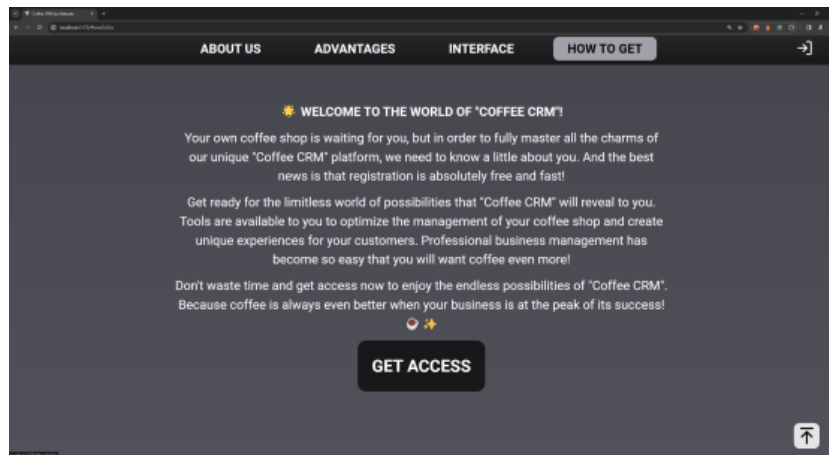


Рисунок 4.2 – Розділ отримання доступу до системи

Після переходу на сторінку отримання доступу перед користувачем відобразиться сторінка створення нового облікового запису адміністратора, фактично нової мережі кав'ярень у системі (див. рис. 4.3).

Рисунок 4.3 – Сторінка створення нового облікового запису

Усі поля форми реєстрації є обов'язковими до заповнення, у разі не заповнення хоча б одного поля форму не буде надіслано та користувачу відобразиться відповідне повідомлення. Електронна пошта обов'язково має бути унікальною, тобто належати лише користувачу, у іншому разі при введенні пошти, яка вже присутня у системі буде виведено повідомлення про помилку (див.рис. 4.4).

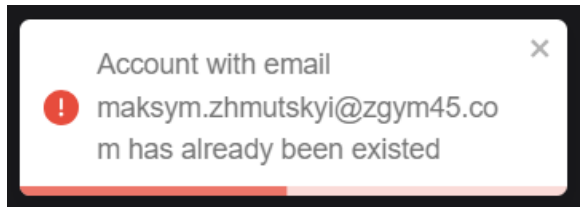


Рисунок 4.4 – Повідомлення про не унікальність пошти

При введенні не надто надійного паролю буде виведено відповідне повідомлення (див.рис. 4.5).

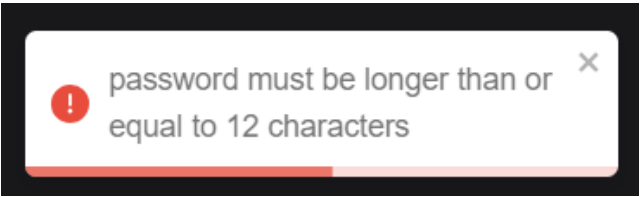


Рисунок 4.5 – Повідомлення про необхідність створення більш надійного паролю

Після успішного проходження реєстрації користувачу відобразиться повідомлення наведене на рис. 4.6, у якому зазначається про необхідність підтвердження пошти шляхом переходу за посиланням у листі, який відправлено на вказану під час реєстрації пошту.

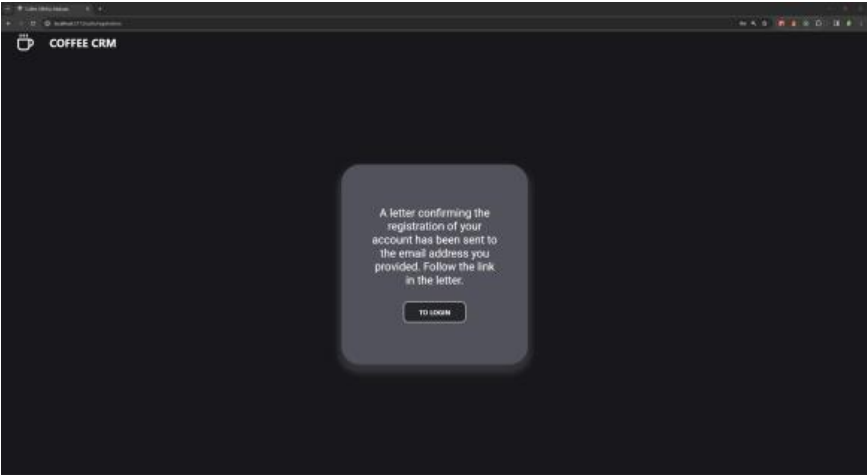


Рисунок 4.6 – Повідомлення в результаті успішної реєстрації

Після натискання на посилання у листі, користувача буде направлено на сторінку підтвердження пошти, де після натискання на елемент керування «Confirm» пошту буде успішно підтверджено та запропоновано здійснити перехід до сторінки аутентифікації (див.рис. 4.7). Додатково користувачеві буде надіслано дані для входу в обліковий запис на пошту.

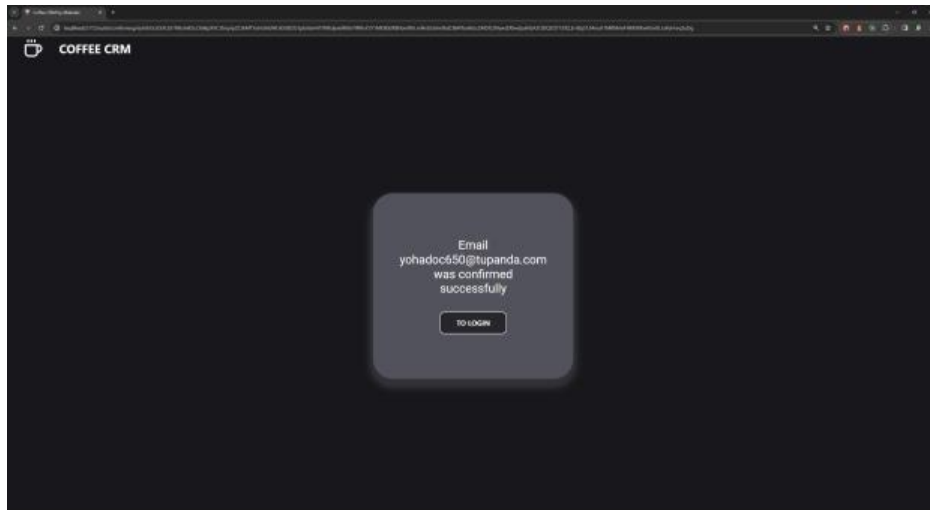


Рисунок 4.7 – Сторінка результату підтвердження пошти

Користувачам з роллю баристи можливість здійснення самостійної реєстрації не надано. Для створення облікового запису баристи адміністратору мережі необхідно здійснити процес «працевлаштування» баристи у свою мережу, шляхом додавання за допомогою відповідної форми свого робочого простору.

4.3.1.2 Аутентифікація користувача

При переході на сторінку аутентифікації перед користувачем відображається форма введення даних облікового запису користувача (див.рис. 4.8).

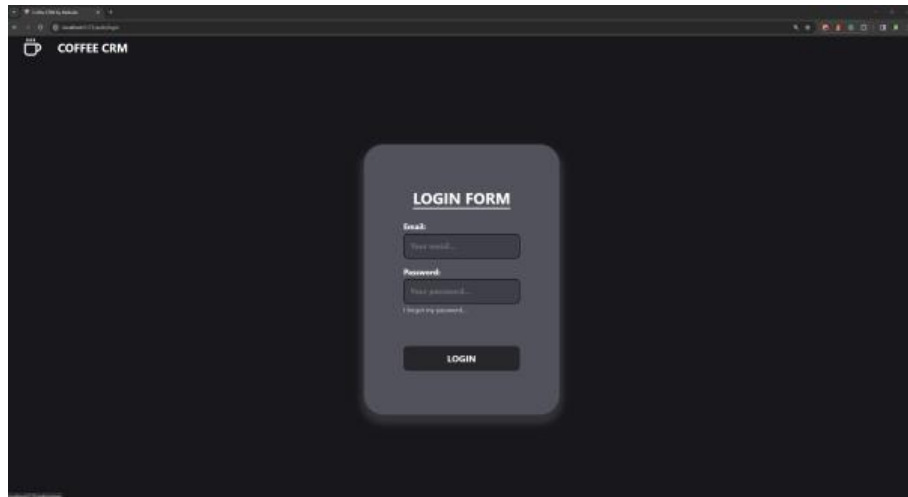


Рисунок 4.8 – Сторінка аутентифікації

Після чого користувачу необхідно ввести коректні дані для входу, кожне поле є обов'язковим, у інакшому випадку користувач може отримати помилки наведені на рис. 4.9.

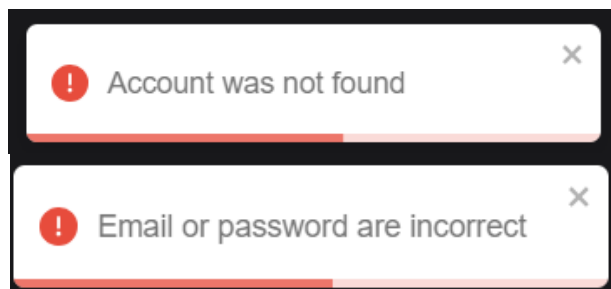


Рисунок 4.9 – Можливі помилки під час невдалої аутентифікації

Якщо дані було введено коректного, та користувача пройшов процес аутентифікації, відобразиться повідомлення про успішність дії та сторінка інформації про адміністратора, у разі якщо аутентифікацію було здійснено адміністратором (див.рис. 4.10).

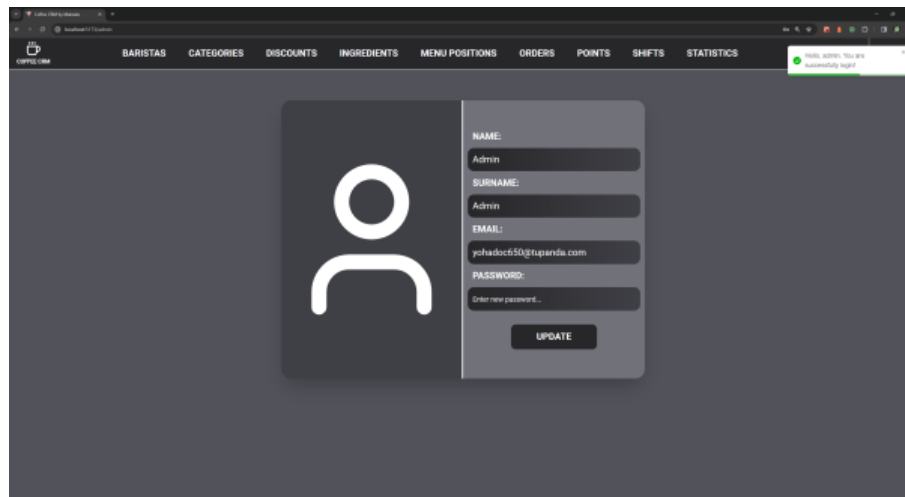


Рисунок 4.10 – Сторінка інформації про адміністратора

Якщо користувача аутентифіковано у якості баристи, йому відобразиться сторінка з вибором доступних йому кав'ярень для початку робочої зміни (або повідомлення про необхідність звернення до адміністратора з проханням про призначення на певну кав'ярню). Сторінку обрання кав'ярні наведено на рис. 4.11. Вибір необхідно здійснити шляхом вибору однієї з кав'ярень у списку та натисканням на елемент керування «Select point». У разі потреби завершення процесу користувач може здійснити перехід до сторінки аутентифікації натиснувши на відповідний елемент керування.

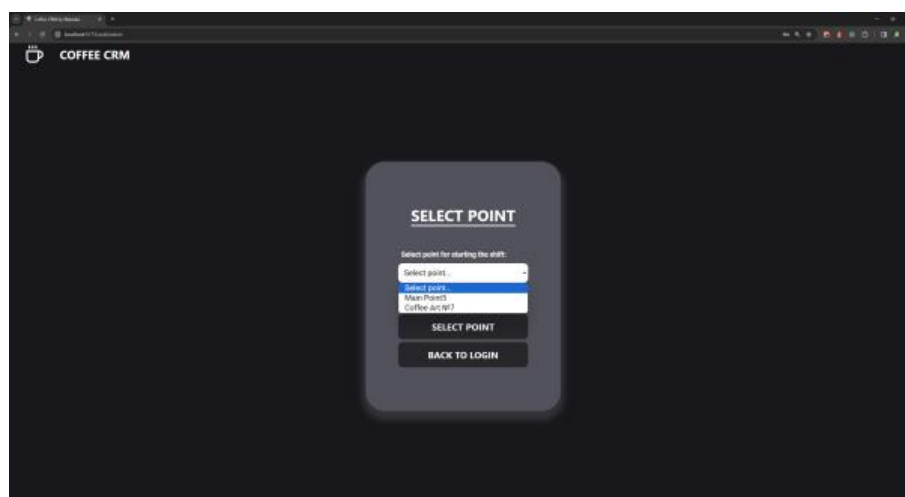


Рисунок 4.11 – Сторінка обрання кав'ярні

Після чого перед баристою(аналогічно до адміністратора) з'явиться сторінка його профілю (див.рис. 4.12), та буде здійснено запит на занесення інформації про початок робочої зміни.

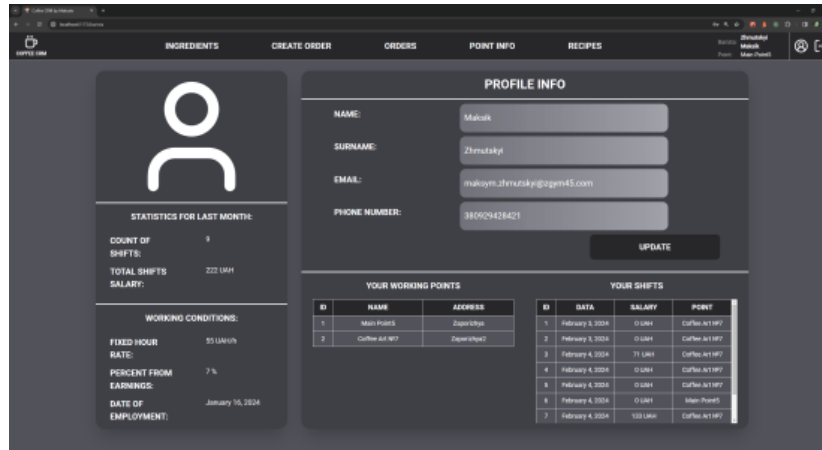


Рисунок 4.12 – Сторінка профілю баристи

4.3.1.3 Вихід з облікового запису

Після завершення робочої зміни, або за будь-якої іншої причини, користувач має здійснити вихід з системи, шляхом натискання елемента керування з відповідним зображенням у правому верхньому куті.

Якщо вихід із системи здійснено користувачем у ролі адміністратора, буде здійснено очищення усієї інформації про сеанс роботи з програмою та перехід на сторінку аутентифікації.

Аналогічні дії буде здійснено у випадку користувача з роллю баристи, але додатково здійсниться запит з інформацією про завершення робочої зміни, після чого відобразиться додаткове повідомлення з інформацією про час та заробітну платню за цю зміну (див.рис. 4.13).

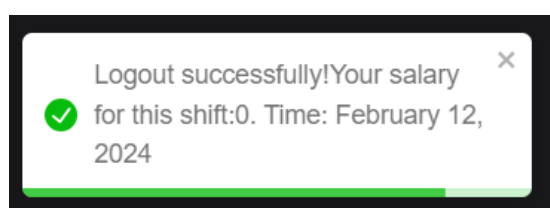


Рисунок 4.13 – Повідомлення з інформацією про завершену зміну

4.3.1.4 Відновлення даних для входу

Вході здійснення процесу аутентифікації може постати проблема пов'язана з відсутністю коректного паролю(у разі якщо користувач його не пам'ятає). Для цього у системі передбачено можливість відновлення паролю з використанням електронної пошти користувача. Для цього на сторінці аутентифікації необхідно натиснути на елемент «I forgot my password...», після чого відбудеться перехід на сторінку відновлення паролю.

Після чого необхідно ввести коректну електронну пошту користувача, який наявний у системі. Якщо обліковий запис, для якого здійснюється відновлення паролю належить користувачеві з роллю бариста, буде виведено повідомлення про те, що адміністратору його мережі буде надіслано повідомлення з проханням оновити дані для входу. Після того, як адміністратор здійснить цей процес, користувачеві на пошту буде надіслано лист з оновленими даними для входу.

Якщо відновлення паролю здійснюється адміністратором, після підтвердження запиту на відновлення йому на пошту надійде лист з посиланням на форму відновлення паролю.

Після переходу за посиланням буде відображено сторінку з формою для введення нового паролю, після заповнення та відправки якої відбудеться зміна паролю до облікового запису. Додатково на пошту буде надіслано нові дані для аутентифікації у системі.

4.3.2 Робочий простір баристи

Після успішного проходження процесу аутентифікації у якості баристи, та обрання кав'ярні, користувачу буде надано доступ до робочого простору баристи, взаємодію з яким здійснюється за допомогою навігаційної панелі, яку розташовано в горі сторінки (див.рис. 4.14).

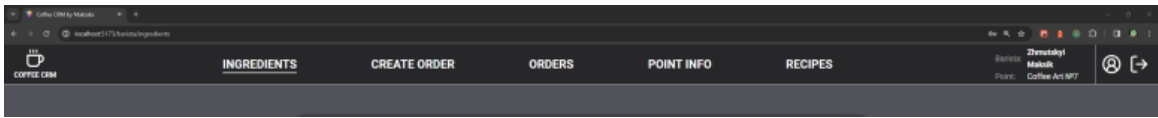


Рисунок 4.14 – Навігаційна панель робочого простору баристи

4.3.2.1 Сторінка інгредієнтів

При обранні на навігаційній панелі сторінки «Інгредієнти», перед користувачем буде відображено відповідну сторінку (див.рис. 4.15).

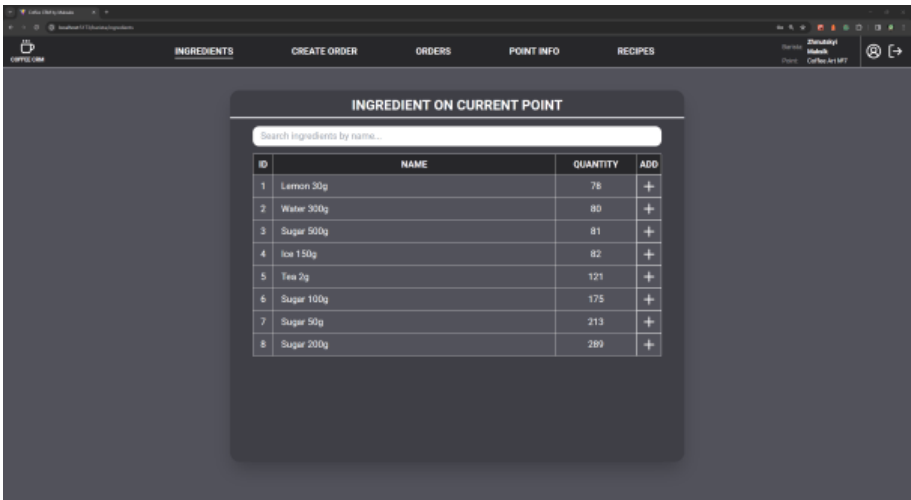


Рисунок 4.15 – Сторінка «Інгредієнти»

Функціональні можливості сторінки дозволяють здійснювати перегляд та пошук використовуючи відповідний елемент керування. У разі поставки товарів до кав'ярні, користувач може оновити кількість відповідного товару у системі шляхом натискання елемента керування з зображенням «+», після чого буде відображено модальне вікно з полем для введення кількості товару поставленого до кав'ярні та елементом керування для здійснення підтвердження дії, після чого вікно буде закрито, а дані про інгредієнт оновлено. Зображення модального вікна для оновлення кількості інгредієнтів наведено на рис. 4.16.

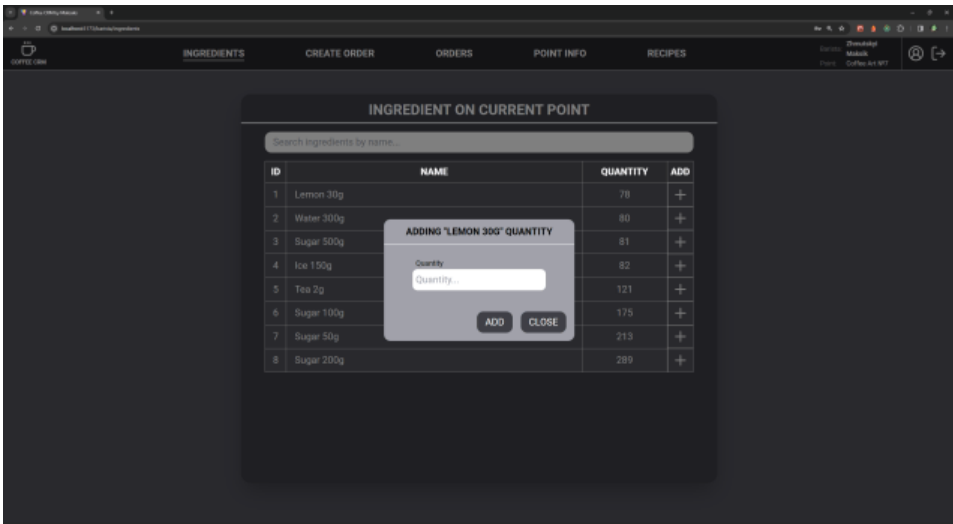


Рисунок 4.16 – Модальне вікно для оновлення кількості інгредієнтів

4.3.2.2 Сторінка створення замовлення

При обранні на навігаційній панелі сторінки «Створення замовлення», перед користувачем буде відображено відповідну сторінку (див.рис. 4.17).

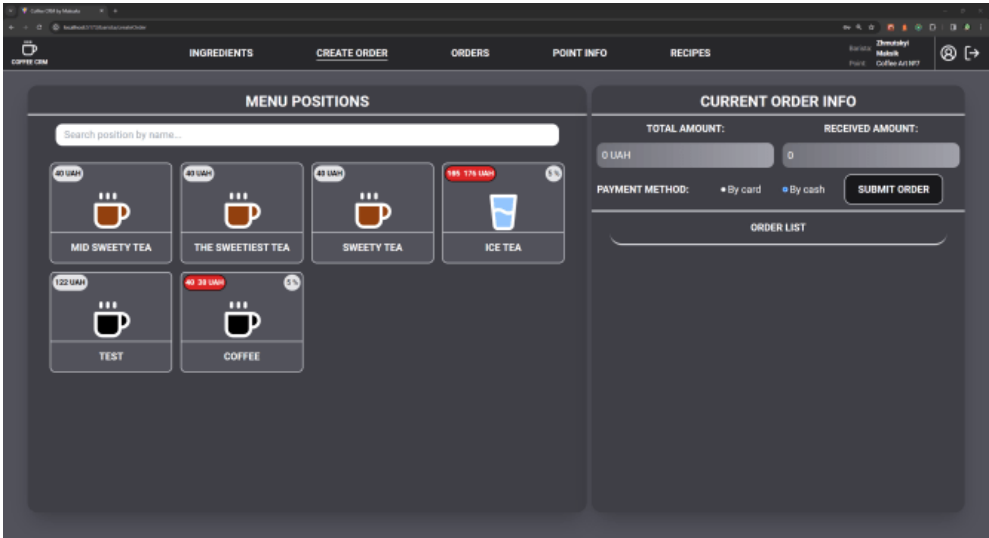


Рисунок 4.17 – Сторінка «Створення замовлення»

Сторінка складається з двох частин, ліворуч розташовано перелік позиції наявних у кав'ярні, з відображення ціни(та знижки, у разі наявності), а праворуч розташовано форму створення замовлення, після натискання на позицію меню її буде додано до замовлення. Додатково на формі створення замовлення присутня

можливість введення кількості або видалення позиції меню з замовленні шляхом натискання відповідних елементів керування. Приклад заповненої форми створення замовлення наведено на рис. 4.18.

CURRENT ORDER INFO	
TOTAL AMOUNT:	RECEIVED AMOUNT:
792 UAH	0
PAYMENT METHOD: ☒ By card ☐ By cash	
SUBMIT ORDER	
ORDER LIST	
ICE TEA	352.00 UAH
+	2
-	
SWEETY TEA	160.00 UAH
+	4
-	
THE SWEETEST TEA	160.00 UAH
+	4
-	
MID SWEETY TEA	120.00 UAH
+	3
-	

Рисунок 4.18 – Приклад заповненої форми створення замовлення

На формі присутні два інформаційних поля, перше з яких дозволяє отримати інформацію про суму замовлення, а друге ввести суму отриману від клієнта, лише у разі обраного методу оплати «Готівкою». Якщо різниця між введеною сумою отриманою від клієнта та загальною сумою замовлення є більшою за наявну суму грошей у кав'ярні, тобто бариста не зможе видати решту, відобразиться відповідне повідомлення. Якщо методом оплати замовлення обрано «Картою», необхідність введення отриманої суми відсутня.

Після підтвердження замовлення здійсниться автоматичне очищення форми створення замовлення та буде відображено модальне вікно з чеком замовлення (див.рис. 4.19).



Рисунок 4.19 – Модальне вікно з чеком замовлення

4.3.2.3 Сторінка перегляду замовлень

При обранні на навігаційній панелі сторінки «Замовлення», перед користувачем буде відображено відповідну сторінку (див.рис. 4.20).

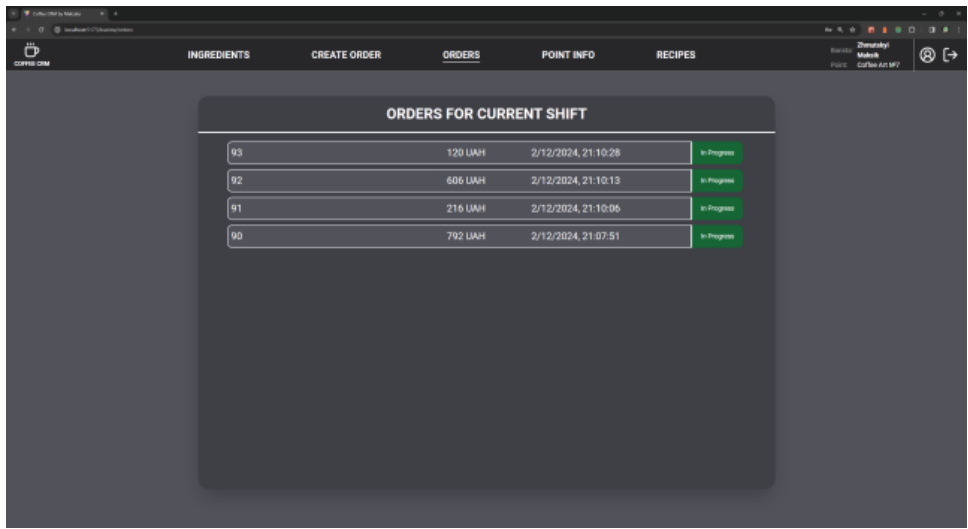


Рисунок 4.20 – Сторінка «Замовлення»

Функціональна панель цієї сторінки надає можливість перегляду замовлення користувача за поточну робочу зміну. При натисканні на відповідне замовлення перед користувачем з’явиться модальне вікно з детальною інформацією про замовлення. У разі необхідності зміни статусу замовлення

потрібно натиснути на відповідний елемент керування з надписом «In Progress», після чого статус замовлення буде оновлено та відображено відповідні повідомлення(див.рис. 4.21).

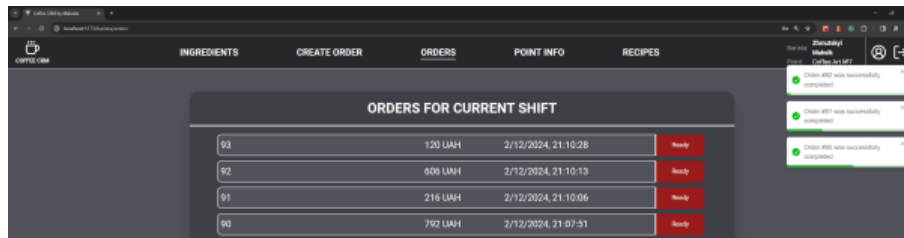


Рисунок 4.21 – Оновлені статуси замовлення

4.3.2.4 Сторінка керування поточною кав'ярнею

При обранні на навігаційній панелі сторінки «Інформація про кав'ярню», перед користувачем буде відображено відповідну сторінку (див.рис. 4.22).

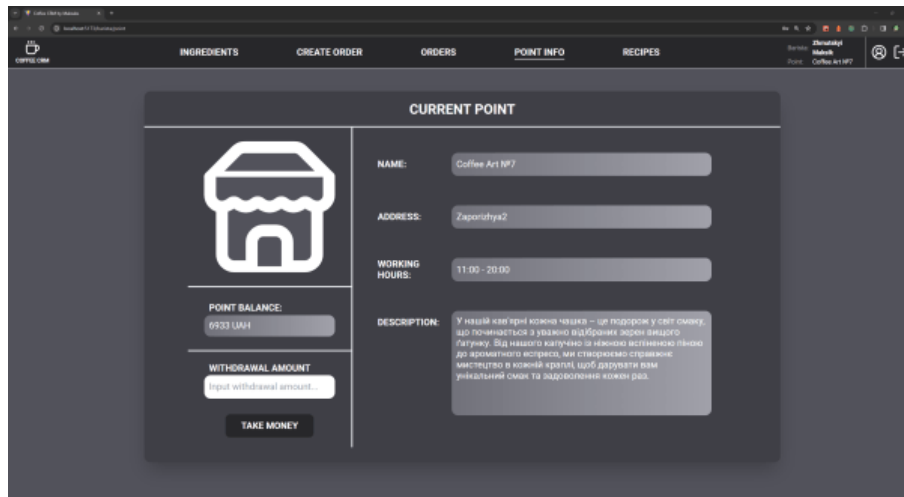


Рисунок 4.22 – Сторінка «Інформація про кав'ярню»

Ця сторінка надає можливість переглядати інформацію про поточну кав'ярню, зокрема, поточний баланс та здійснювати зняття з нього грошей(за потреби розрахунку з постачальником або щось інше).

4.3.2.5 Сторінка рецептів

При обранні на навігаційній панелі сторінки «Рецепти», перед користувачем буде відображено відповідну сторінку (див.рис. 4.23).

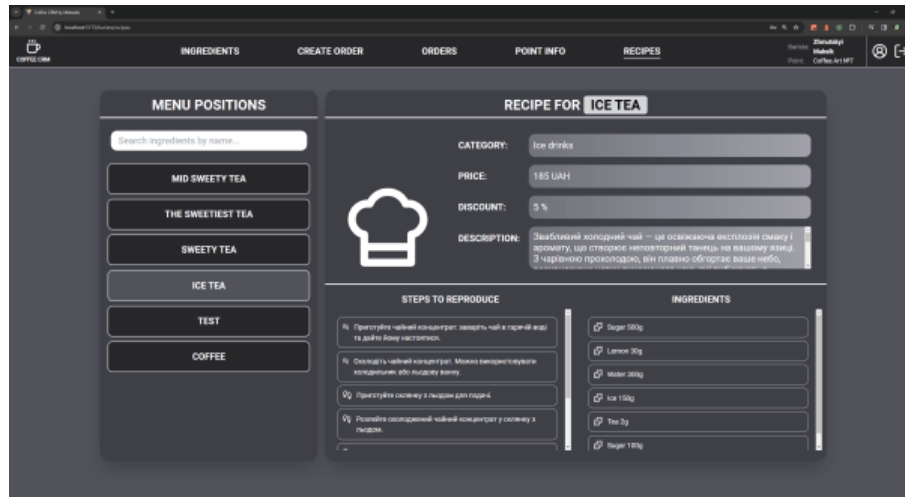


Рисунок 4.23 – Сторінка «Рецепти»

Інтерфейс сторінки складається з двох частин, ліворуч знаходиться елемент для здійснення пошуку та перегляду наявних позицій меню, після обрання якого, шляхом натискання, у частині праворуч відображаються відповідні дані про позицію меню з детальним описом шляху його відтворення та інгредієнтів, які знадобляться для приготування.

4.3.2.6 Сторінка профілю

При натисканні на навігаційній панелі елементу керування з зображенням профілю перед користувачем буде відображено сторінку інформації про нього (див.рис. 4.24).

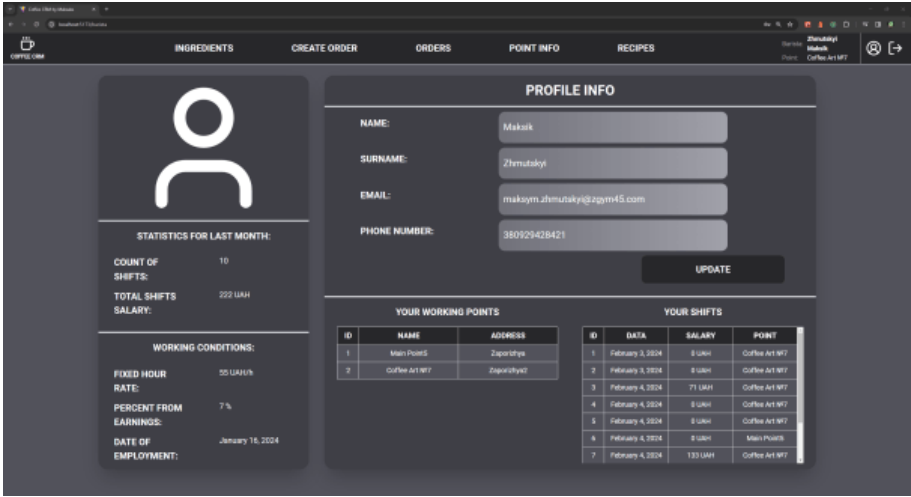


Рисунок 4.24 – Сторінка «Інформація про себе»

Інформація, яка надається про користувача складається з декількох частин.

Першою частиною слугують статистичні дані за останній місяць його роботи, які складаються з кількості робочих змін та заробітної платні за цей період. Нижче наводиться інформація про робочі умови з наведенням погодинної ставки, проценту від заробітку та дату працевлаштування.

Другою частиною є інформація про користувача, яка дозволяє здійснити оновлення ім'я, прізвища, пошти та телефонного номеру користувача шляхом оновлення відповідних полів форми. У разі зміни пошти користувачу буде надіслано повідомлення на нову пошту з даними для авторизації.

Третьою частиною інтерфейсу сторінки є інформація про кав'ярні у яких користувач є працевлаштованим та перелік змін користувача з інформацією про дату зміни та заробітну платню.

4.3.3 Робочий простір адміністратора

Після успішного проходження процесу аутентифікації у якості адміністратора, користувачу буде надано доступ до робочого простору, взаємодію з яким здійснюється за допомогою навігаційної панелі, яку розташовано в горі сторінки (див.рис. 4.25).

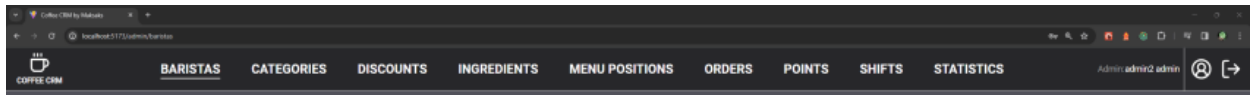


Рисунок 4.25 – Навігаційна панель робочого простору адміністратора

4.3.3.1 Керування баристами

При обранні на навігаційній панелі сторінки «Баристи», перед користувачем буде відображено відповідну сторінку (див.рис. 4.26).

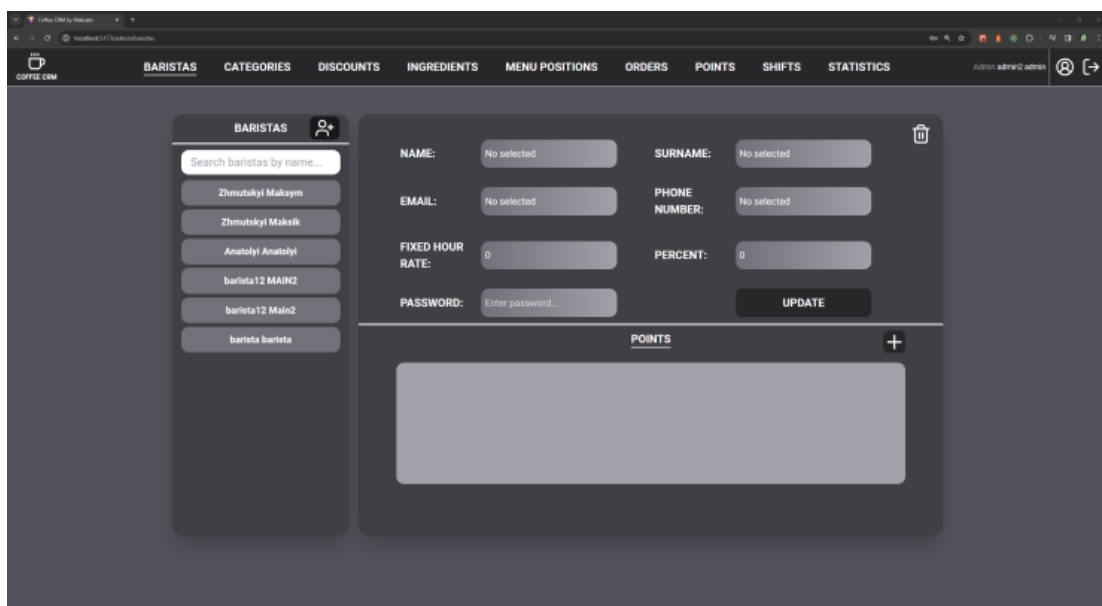


Рисунок 4.26 – Сторінка «Баристи»

Простір сторінки розділено на дві частини. Частина ліворуч дозволяє здійснювати пошук та перегляд барист. За допомогою елемента керування з відповідним зображенням здійснюється додавання нового баристи, шляхом заповнення відповідного модального вікна, яке з'явиться після натискання на елемент керування (див.рис. 4.27).

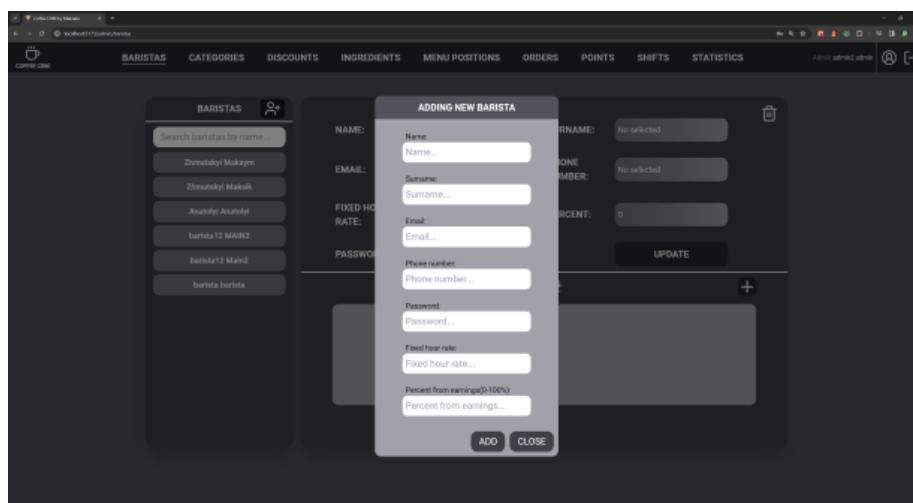


Рисунок 4.27 – Модальне вікно для додавання баристи

Після додавання нового баристи сторінку буде оновлено та автоматично обрано нового баристу для перегляду детальної інформації.

Для призначення баристи у кав'ярню необхідно використовувати елемент керування з знаком «+» у розділі «Кав'ярні» поточної сторінки, який відображає модальне вікно з списком можливих кав'ярень. Після обрання кав'ярні та підтвердження призначення, нова кав'ярня з'явиться в інформації про обраного баристу.

Додатково передбачено можливість оновлення інформації про баристу, шляхом оновлення відповідних полів форми(у разі оновлення паролю чи пошти буде надіслано відповідний лист на (нову)пошту баристі з оновленими даними для входу).

Також наявна можливість видалення баристи натиснувши на елемент керування з відповідним зображенням.

4.3.3.2 Керування категоріями та знижками

Розглянемо функціональні можливості сторінок «Категорії» та «Знижки» на прикладі першої.

При обранні на навігаційній панелі сторінки «Категорії», перед користувачем буде відображено відповідну сторінку (див.рис. 4.28).

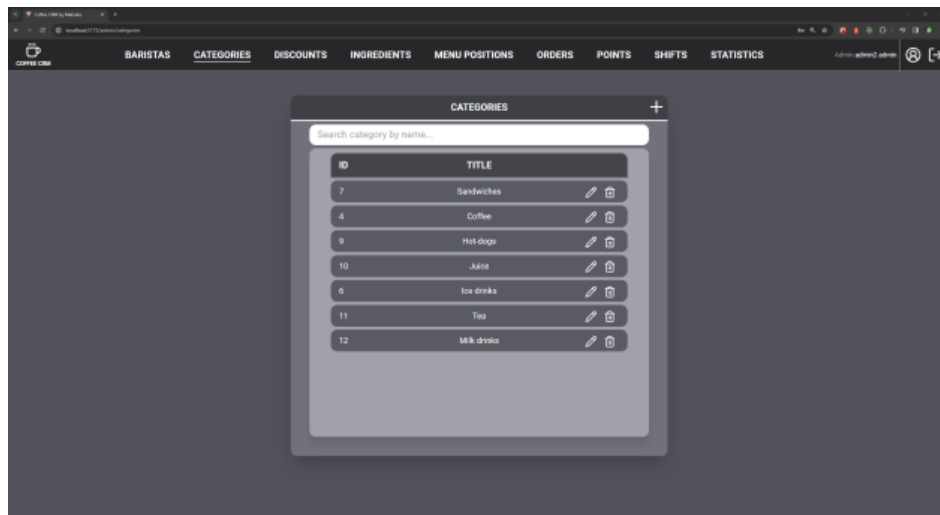


Рисунок 4.28 – Сторінка «Категорії»

На сторінці відображено елемент для здійснення пошуку та перегляду категорій наявних у системі. Для додавання нової категорії потрібно натиснути на елемент з зображенням «+», після чого буде відображено відповідне модальне вікно з полями, після заповнення яких та підтвердження створення, перелік категорій буде оновлено.

Передбачено можливість видалення категорії, шляхом натискання на відповідному записі елементу керування з відповідним зображенням.

При натисканні на елемент керування з зображенням олівця, відкривається модальне вікно та надається можливість редагування інформації про обрану категорію (див.рис. 4.29).

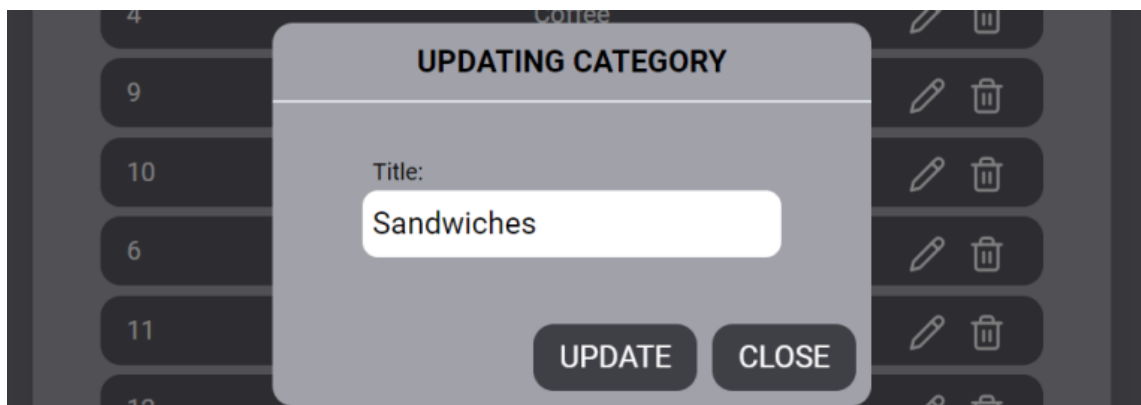


Рисунок 4.29 – Модальне вікно для оновлення категорії

Здійснення взаємодії з сторінкою «Знижки» аналогічне сторінці «Категорії» за винятком того, що відсутня можливість створення знижки, так як ця можливість надається на сторінці «Позиції меню». Сторінку «Знижки» наведено на рис. 4.30.

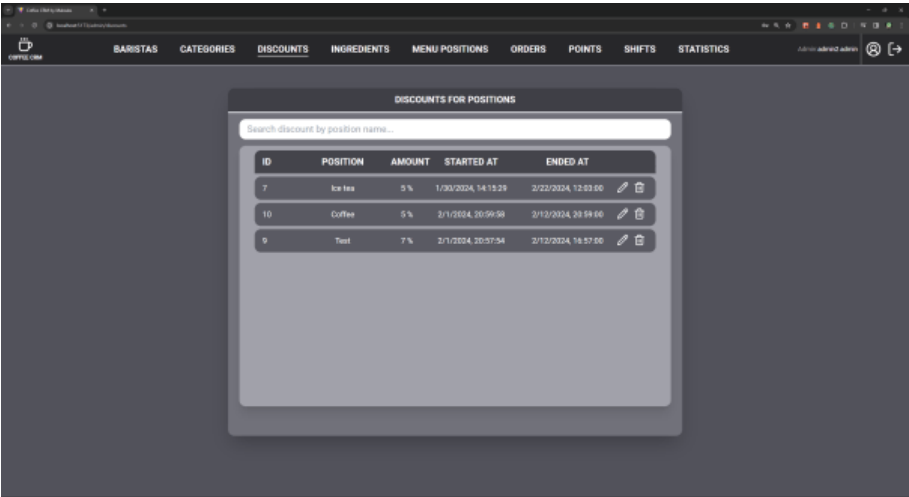


Рисунок 4.30 – Сторінка «Знижки»

4.3.3.3 Керування інгредієнтами

При обранні на навігаційній панелі сторінки «Інгредієнти», перед користувачем буде відображено відповідну сторінку (див.рис. 4.31).

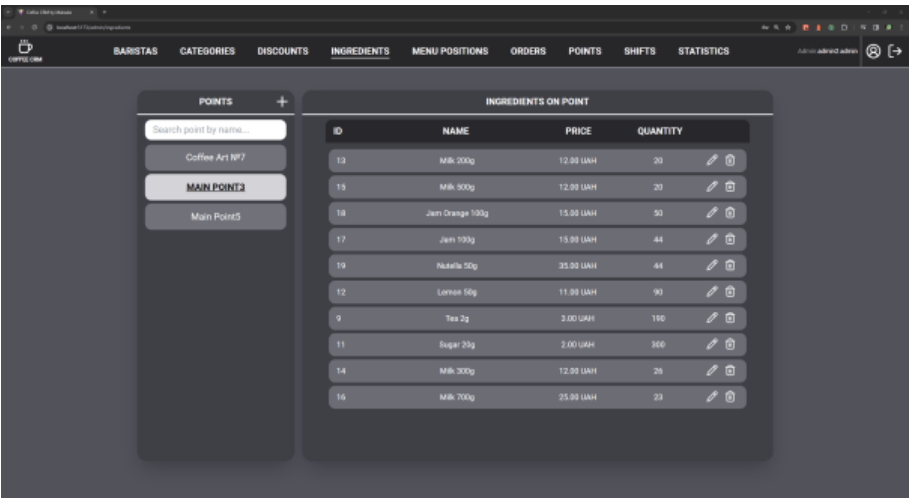


Рисунок 4.31 – Сторінка «Інгредієнти»

Сторінка складається з двох функціональних елементів. За допомогою елемента, який розташовано ліворуч, здійснюється пошук та перегляд кав'ярень. За допомогою елемента керування з відповідним зображенням здійснюється додавання нового інгредієнту до обраної кав'ярні, шляхом заповнення відповідного модального вікна, яке з'явиться після натискання на елемент керування.

При обрані кав'ярні у елементів праворуч відображається перелік інгредієнтів наявних у кав'ярні. За допомогою елементів керування на кожному записі надається можливість редагування інформації та видалення обраного.

Редагування здійснюється шляхом показу модального вікна та оновлення необхідних полів, після здійснення оновлення відбувається оновлення переліку інгредієнтів та відображення нового стану.

4.3.3.4 Керування позиціями меню

При обранні на навігаційній панелі сторінки «Позиції меню», перед користувачем буде відображено відповідну сторінку (див.рис. 4.32).

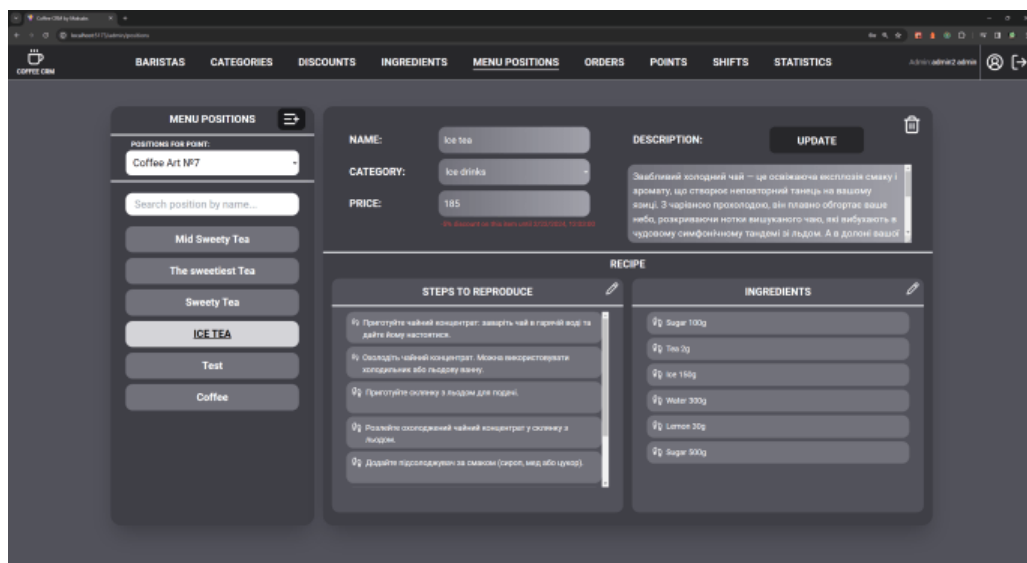


Рисунок 4.32 – Сторінка «Позиції меню»

Елемент, який розташований ліворуч на сторінці, дозволяє здійснити вибір кав'ярні на якій відбувається робота з позиціями меню та пошук позиції меню. Вгорі цього елемента знаходиться елемент керування з відповідним зображенням для додавання нової позиції до обраної кав'ярні. Після натискання на нього з'являється модальне вікно для заповнення інформації про нову позицію. Після створення відбувається відображення її у елементі праворуч.

Елемент праворуч дозволяє переглядати, оновлювати та видаляти інформацію про обрану позицію. Оновлення здійснюється шляхом редагування необхідних полів та підтвердження натисканням елемента керування «Оновити». Видалення відбувається шляхом натискання елемента з відповідним зображенням.

Для редагування кроків відтворення рецепту позиції слід натиснути на елементу керування з зображенням олівця у відповідній частині з назвою «Кроки відтворення». Після чого буде відображено модальне вікно, яке наведено на рис. 4.33.

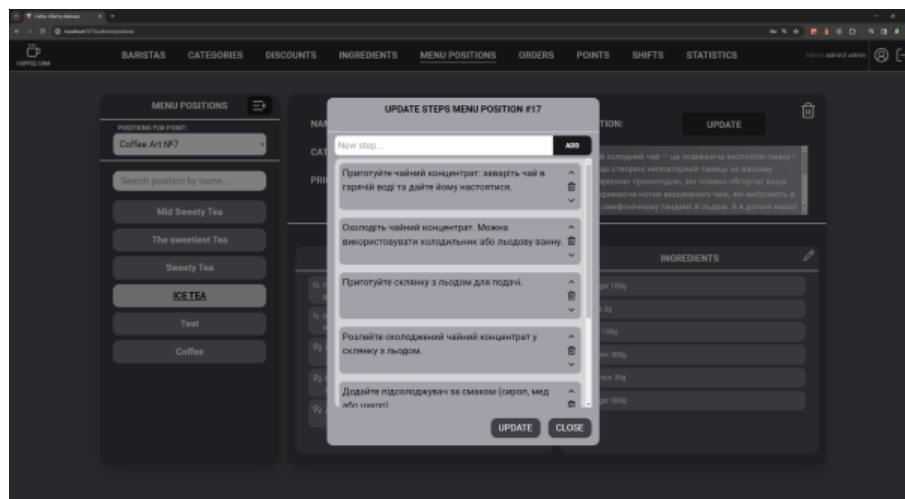


Рисунок 4.33 – Модальне вікно для оновлення кроків відтворення рецепту

Для додавання нового кроку потрібно ввести необхідну інформацію про нього у відповідне поле та натиснути на елемент з надписом «Додати», після чого відбудеться додавання кроку на останню позицію списку.

Для редагування послідовності кроків наявні елементи керування з зображенням стрілок вгору та донизу, що дозволяють переміщувати крок у відповідному напрямку. Вилучи крок з послідовності можна шляхом натискання на елемент з відповідним зображенням. Підтвердження оновлення кроків здійснюється шляхом натискання на елемент з надписом «Оновити», після чого інформацію про позицію буде оновлено.

Аналогічно до кроків відтворення, здійснюється оновлення інгредієнтів, форму, яка відображається для здійснення цього процесу, наведено на рис. 4.34.

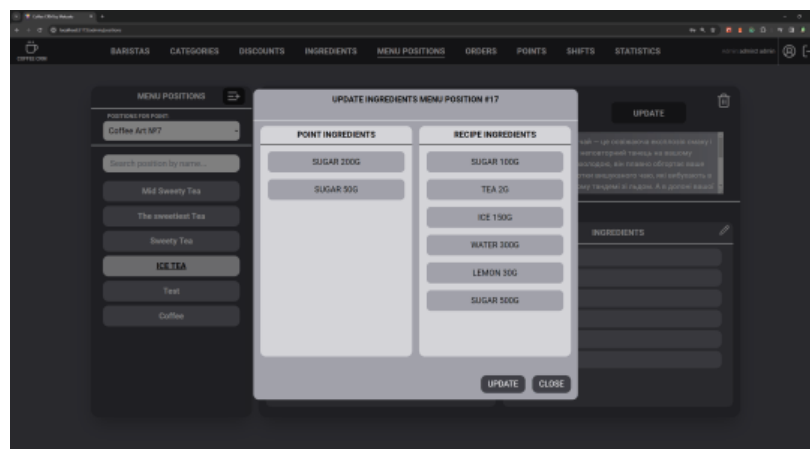


Рисунок 4.34 – Модальне вікно для оновлення інгредієнтів

У модальному вікні ліворуч розташовано перелік наявних інгредієнтів на точці, що не використовуються у обраному рецепті, праворуч наведено перелік інгредієнтів, що вже присутні у рецепті. Шляхом натискання на елементи в одному списку, здійснюється їх переміщення у інший, уникаючи дублювання.

Для елементів у яких відсутня знижка, наявна можливість її встановлення, шляхом натискання на елемент керування з зображенням «%» (див.рис. 4.35).

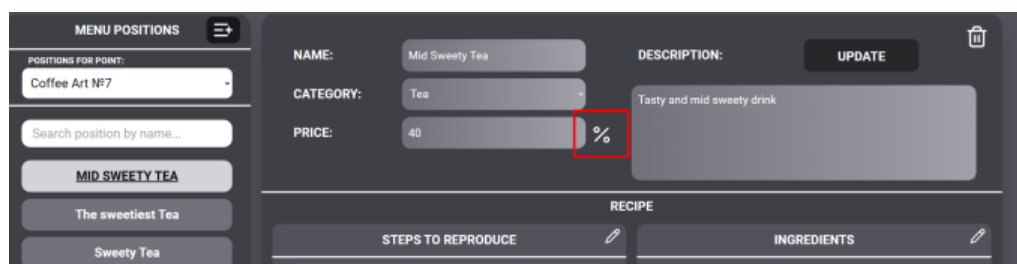


Рисунок 4.35 – Елемент керування для встановлення знижки

Після натискання на нього буде відображено модальне вікно з полями для введення розміру знижки та кінцевої дати її дії. В результаті встановлення знижки відповідна інформація відобразиться на інформаційній картці елементу (див.рис. 4.36).

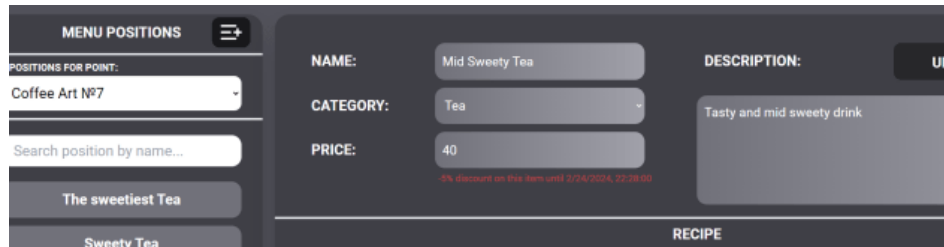


Рисунок 4.36 – Відображення інформації про знижку на позицію меню

4.3.3.5 Керування замовленнями

При обранні на навігаційній панелі сторінки «Замовлення», перед користувачем буде відображено відповідну сторінку (див.рис. 4.37).

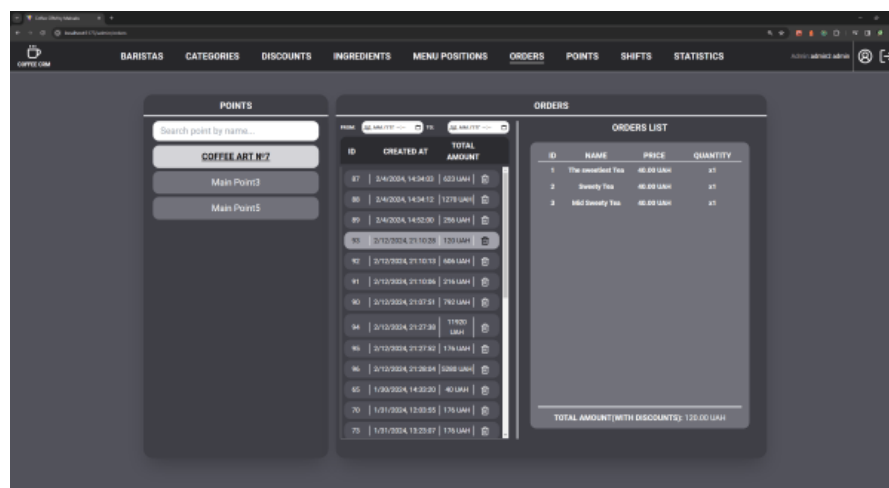


Рисунок 4.37 – Сторінка «Замовлення»

Дана сторінка надає можливість здійснювати пошук замовлень, які було створено у обраній кав'ярні. Додатково передбачається здійснення фільтрації замовлень по даті створення шляхом встановлення обмежень у відповідних елементах керування.

При натисканні на один з елементів списку замовлень відображається інформація про складові позиції, їх кількість та загальну суму замовлення.

Існує можливість видалення інформації про певне замовлення, шляхом натискання на елемент керування з відповідним зображенням.

4.3.3.6 Керування кав'ярнями

При обранні на навігаційній панелі сторінки «Кав'ярні», перед користувачем буде відображено відповідну сторінку (див.рис. 4.38).

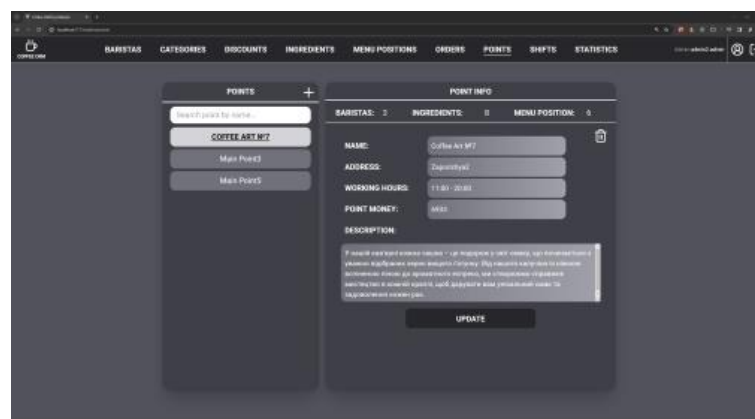


Рисунок 4.38 – Сторінка «Кав'ярні»

Обрана сторінка надає можливість пошук та перегляду інформації про кав'ярні. Інформаційна картка окрім опису кав'ярні надає статистичну інформацію про загальну кількість барист, інгредієнтів та позицій меню у кав'ярні.

При використанні елемента керування з зображенням «+» надається можливість додавання нової кав'ярні до мережі, шляхом відповідних заповнення полів модального вікна, яке з'явиться після натискання.

За необхідності оновлення інформації про кав'ярню слід здійснити редагування відповідного поля та підтвердити операцію шляхом натискання на елемент з надписом «Оновити».

Видалення здійснюється використовуючи елемент з відповідним зображенням.

4.3.3.7 Керування робочими змінами

При обранні на навігаційній панелі сторінки «Робочі зміни», перед користувачем буде відображено відповідну сторінку (див.рис. 4.39).

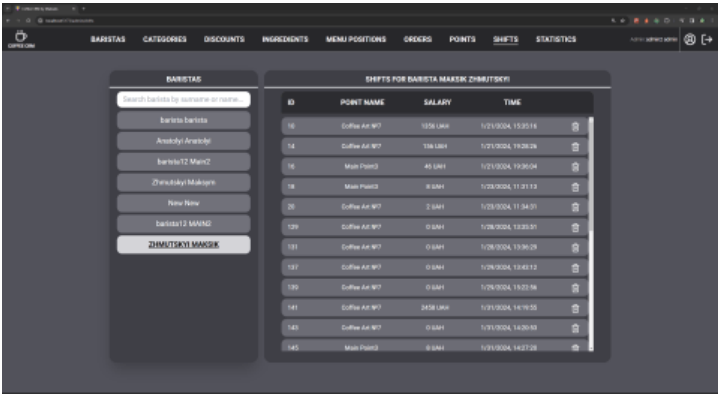


Рисунок 4.39 – Сторінка «Робочі зміни»

Елементи сторінки надають можливість здійснення пошуку та перегляду детальної інформації про зміни відповідного баристи.

Шляхом натискання на елемент з відповідним зображенням на відповідному записі, здійснюється вилучення інформації про робочу зміну.

4.3.3.8 Отримання статистичних показників

При обранні на навігаційній панелі сторінки «Статистики», перед користувачем буде відображено відповідну сторінку (див.рис. 4.40).

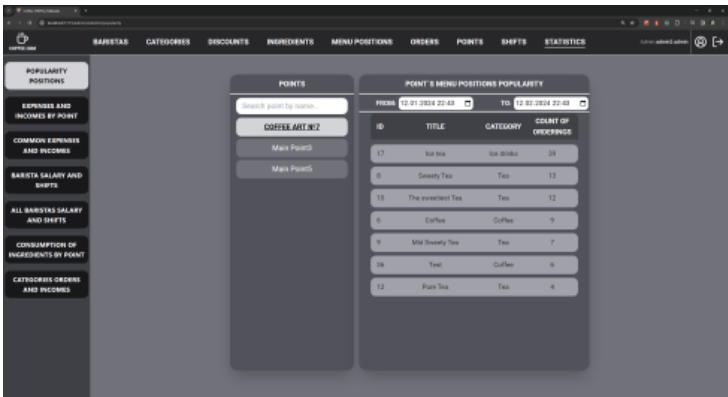


Рисунок 4.40 – Сторінка «Статистики»

Ця сторінка є групою сторінок, які надають можливість отримання статистики за відповідними показниками, серед яких:

- популярність позицій у певній кав'ярні;
- витрати та доходи в окремій кав'ярні;
- витрати та доходи в усіх кав'ярнях;
- робочі зміни та заробітна платня баристи;
- робочі зміни та заробітна платня всіх барист;
- витрати інгредієнтів у кав'ярні;
- кількість продажів та дохід за категорією по усій мережі.

Для отримання кожних статистичних показників передбачено власний пункт навігаційного меню, яке розташоване на сторінці ліворуч.

У центрі сторінки розташовані елементи, один з яких, розташований ліворуч, дозволяє обрати сутність системи, відносно якої буде отримано статистичні показники, у свою чергу, другий елемент, розташований праворуч, відображає отримані дані. У випадках коли надається узагальнена інформація, наявний лише другий елемент, який містить перелік усіх даних.

Також, існує можливість отримання показників за певний обраний період за допомогою використання відповідних елементів керування на кожній сторінці формування статистичних показників.

На рис. 4.41-4.42 представлено приклади деяких сформованих звітів.

ID	TITLE	CATEGORY	COUNT OF ORDERINGS
17	Ice Tea	Ice drinks	28
8	Energy Tea	Tea	13
10	Tea (sweetened) Tea	Tea	12
6	Coffee	Coffee	9
9	Milk (sweetened) Tea	Tea	7
30	Tea	Coffee	6
12	Pine Tea	Tea	4

Рисунок 4.41 – Статистичні дані про популярність позицій у певній кав'ярні

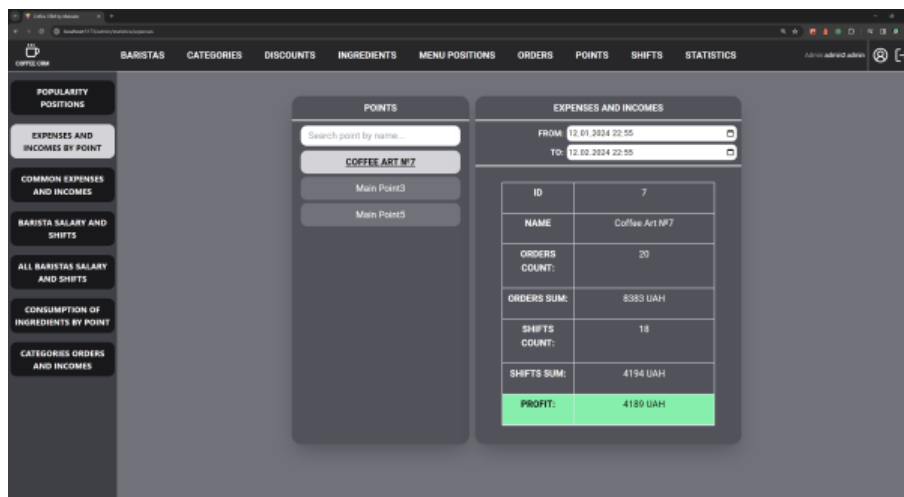


Рисунок 4.42 – Статистичні дані про витрати та доходи в окремій кав'ярні

4.3.3.9 Керування профілем

При натисканні на навігаційній панелі елементу керування з зображенням профілю перед користувачем буде відображено сторінку інформації про нього (див.рис. 4.43).

Рисунок 4.43 – Сторінка «Інформація про себе»

Сторінка надає інформацію про користувача, яка дозволяє здійснити оновлення ім'я, прізвища, пошти та паролю користувача шляхом оновлення відповідних полів форми. У разі зміни пошти або/та паролю користувачу буде надіслано повідомлення на (нову)пошту з даними для авторизації.

4.4 Тестування програми

4.4.1 Тестування серверної частини

На першому кроці тестування системи було здійснено перевірку коректної реалізації усіх функціональних вимог на боці серверу. Для цього було створено набір тестів для усіх маршрутів з використанням різних наборів даних.

Для тестування API програмного продукту було використано Postman.

Postman – популярний інструмент розробки та тестування API. Він забезпечує зручний та інтуїтивно зрозумілий інтерфейс для створення, надсилання та отримання HTTP-запитів і відповідей. Postman дозволяє зручно взаємодіяти з API, створюючи та налагоджуючи запити, перевіряючи відповіді, автоматизуючи тести та відстежуючи продуктивність API [20].

На рисунку 4.44 представлено перелік наборів тестів у Postman створених для кожного з наявних маршрутів та розділено за ролями користувачів, відповідно до їх рівня доступу до певного функціоналу.

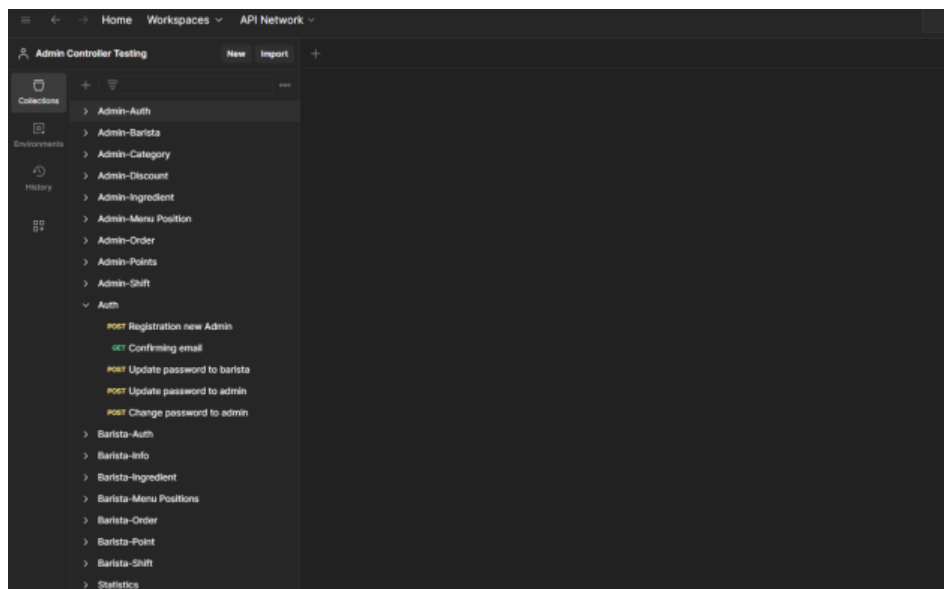


Рисунок 4.44 – Перелік наборів тестів у Postman

Результати тестування серверної складової програми представлено у вигляді чек-листу, який наведено у табл. 4.1.

Таблиця 4.1 – Чек-лист тестування серверної складової програми

№	Назва набору тестів	Роль	Результат
1	Аутентифікація	Адміністратор	Пройдено вдало
2	Керування баристами	Адміністратор	Пройдено вдало
3	Керування категоріями	Адміністратор	Пройдено вдало
4	Керування знижками	Адміністратор	Пройдено вдало
5	Керування інгредієнтами	Адміністратор	Пройдено вдало
6	Керування позиціями меню	Адміністратор	Пройдено вдало
7	Керування замовленнями	Адміністратор	Пройдено вдало
8	Керування кав'ярнями	Адміністратор	Пройдено вдало
9	Керування робочими змінами	Адміністратор	Пройдено вдало
10	Реєстрація та відновлення даних	Адміністратор , Бариста	Пройдено вдало
11	Аутентифікація	Бариста	Пройдено вдало
12	Отримання інформації про себе	Бариста	Пройдено вдало
13	Робота з інгредієнтами	Бариста	Пройдено вдало
14	Робота з позиціями меню	Бариста	Пройдено вдало
15	Робота з замовленнями	Бариста	Пройдено вдало
16	Робота з кав'ярнею	Бариста	Пройдено вдало
17	Робота з власними робочими змінами	Бариста	Пройдено вдало
18	Отримання статистичних показників	Адміністратор	Пройдено вдало

Проходження авторизації, де це вимагається, під час тестування API, відбувалося після отримання токена в ході аутентифікації та впровадження його у відповідний заголовок запиту.

Здійснення кожного запиту супроводжувалося паралельною перевіркою відповідності отриманих даних до БД.

Таким чином, було підтверджено коректну роботу усіх маршрутів серверної складової системи, відповідно до їх призначення.

4.4.2 Тестування клієнтської частини

На другому етапі перевірки системи, було здійснено тестування клієнтської частини, а саме тестування коректного відображення усіх розроблених компонентів. Цей етап тестування здійснювався за допомогою інструментарію React Testing Library.

React Testing Library – це бібліотека тестування компонентів React, яка допомагає тестувати орієнтовані на користувача компоненти. Це дозволяє писати тести, які зосереджено на тому, як компоненти взаємодіють з користувачем, симулюючи роботу у справжньому вебзастосунку [21].

Приклад фрагменту коду створеного тесту для компоненту:

```
describe('MyComponent', () => {
  it('рендериться правильно на основі id', () => {
    const { getByTestId } = render(
      <Button data-testid="testedButtonComponent" />
    );
    expect(getByTestId('myComponentId')).toBeInTheDocument();
  });
});
```

Підсумком результатів тестування коректного відображення компонентів є повністю вдале проходження перевірки, усі розроблені компоненти в ході тестування було відображено вірно.

Додатково було здійснено перевірку адаптивності інтерфейсу до різної роздільної здатності моніторів, так як використання системи може здійснюватися на різних пристроях з різними характеристиками.

На сьогоднішній день популярними характеристиками дисплеїв серед виробників є: HD(1280x720), FullHD(1920x1080) та WQHD(2560x1440) [22].

Перевірка адаптивності здійснювалася стандартними засобами будь-якого браузеру, результати наведено на рис. 4.45-4.47. В ході перевірки було доведено гарну адаптивність інтерфейсу до будь-якої роздільної здатності дисплею.

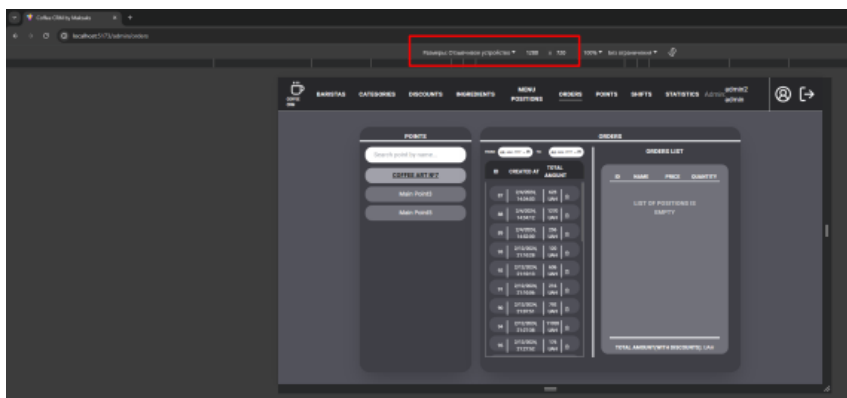


Рисунок 4.45 – Адаптивність інтерфейсу до роздільної здатності HD(1280x720)

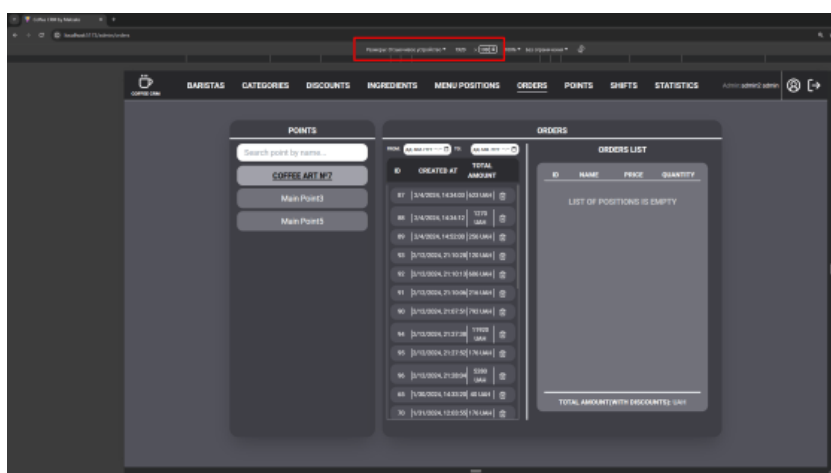


Рисунок 4.46– Адаптивність інтерфейсу до роздільної здатності FullHD(1920x1080)

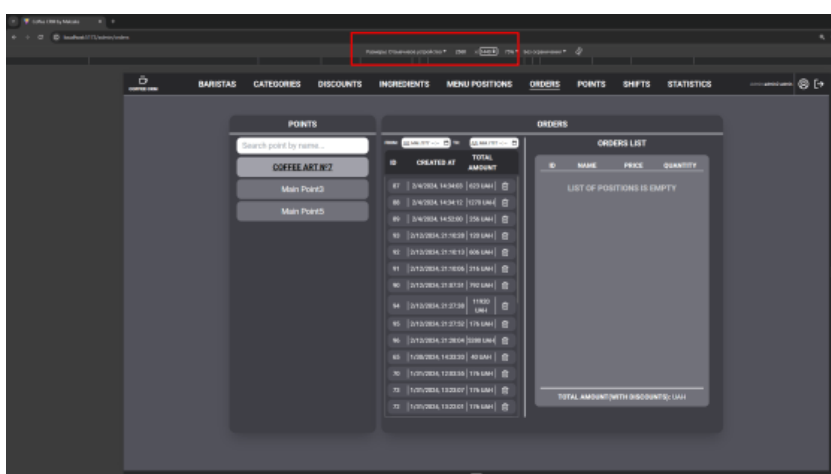


Рисунок 4.47 – Адаптивність інтерфейсу до роздільної здатності WQHD(2560x1440)

4.5 Висновки

Таким чином, програмний продукт повністю відповідає потребам бізнес-процесів мережі кав'ярень.

Система підтримує кілька мереж, які працюють незалежно одна від одної. Ролі в системі поділяються на адміністраторів та барист. Адміністратор має повний доступ до власної мережі та може керувати різними аспектами, такими як бариста, категорія, пункт меню, замовлення тощо. Бариста може керувати запасами інгредієнтів та замовленнями, отримувати статистику власної роботи та керувати профілем.

Загалом розроблене програмне забезпечення відповідає усім вимогам для забезпечення оптимального функціонування мереж кав'ярень.

Тестування продукту дозволило впевнитись у його коректному функціонуванні та стабільності за різних умов.

ВИСНОВКИ

Таким чином, у результаті виконання бакалаврської дипломної кваліфікаційної роботи було досягнуто основної мети – розроблено програмний продукт, який базується на клієнт-серверній архітектурі та дозволяє здійснювати операції пов’язані з обліком та керуванням мережею кав’ярень.

Для досягнення цієї мети було вирішено ряд завдань:

- досліджено основні проблеми, які виникають в процесі роботи мереж кав’ярень;
- проаналізовано особливості функціонування систем управління мережами кав’ярень, розподіл відповідальності та ідентифікації користувачів у системі;
- оцінено існуючі програмні рішення для подібних задач.

Реалізація системи здійснювалася з використанням мови JavaScript(TypeScript) у поєднанні з бібліотекою React(для клієнтської частини) та фреймворку Nest.js(для реалізації серверної складової). Додатково застосувалися для розробки клієнтської частини мова розмітки HTML та бібліотека стилів TailwindCSS та анімації Framer Motion. Системою керування базою даних слугувала PostgreSQL.

Розроблене програмне забезпечення дозволяє підтримувати функціонування понад однієї мережі в ізольованому, одна від одної, форматі. Відповідальність у системі розподілена між двома ролями: адміністратор та бариста. Адміністратор має повний доступ до власної мережі та може керувати різними аспектами, такими як персонал, категорії, пункти меню, замовлення тощо. Бариста може стежити за рівнем запасів і виконувати замовлення, отримувати статистику своєї роботи та керувати власним профілем.

Розроблений програмний продукт дозволяє ефективно керувати різними аспектами діяльності мереж кав’ярень та оптимізувати робочі процеси у закладах, що сприяє покращенню рівня обслуговування та підвищенню якості послуг, а також збільшенню продуктивності та зниженню витрат.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kavapp [Електрон. ресурс]. – Режим доступу: https://kavapp.com/uk/about_me.
2. POS SmartCafe [Електрон. ресурс]. – Режим доступу: <https://smartcafe.com.ua/uk/avtomatizaciya-kavjarni>.
3. Poster [Електрон. ресурс]. – Режим доступу: <https://joinposter.com/ua/business/coffeeshop>.
4. Рейтинг мов програмування 2023 [Електрон. ресурс]. – Режим доступу: <https://dou.ua/lenta/articles/language-rating-2023/>.
5. Rippon, C. Learn React with TypeScript. 2nd Edition / Rippon Carl. – Birmingham: Packt Publishing, 2023. – 474 p.
6. Redux [Electronic resource]. – Access mode: <https://redux.js.org>.
7. React Router [Electronic resource]. – Access mode: <https://reactrouter.com/en/main>.
8. Nest.js [Electronic resource]. – Access mode: <https://nestjs.com>.
9. What is REST? [Electronic resource]. – Access mode: <https://www.codecademy.com/article/what-is-rest>.
10. Visual Studio Code [Electronic resource]. – Access mode: <https://code.visualstudio.com>.
11. 5 найпопулярніших систем управління базами даних [Електрон. ресурс]. – Режим доступу: <https://www.management.com.ua/partners/2021/02/21/5-najpopulyarnishih-sistem-upravlinnya-bazami-danih-2020-roku/>.
12. TypeORM [Electronic resource]. – Access mode: <https://typeorm.io>.
13. Маршрутизація [Електрон. ресурс]. – Режим доступу: http://wiki.tneu.edu.ua/index.php?title=3_Маршрутизація.
14. Guards [Electronic resource]. – Access mode: <https://docs.nestjs.com/guards>.
15. Introduction to JSON Web Tokens [Electronic resource]. – Access mode: <https://jwt.io/introduction>.

16. Темний режим інтерфейсу: 9 підказок правильного дизайну Dark Mode [Електрон. ресурс]. – Режим доступу: <https://www.komarov.design/temnii-dizain-intierfieisu-8-pidskazok-pravilnogho-proiektuvannia/>.
17. TailwindCSS [Electronic resource]. – Access mode: <https://tailwindcss.com>.
18. Framer Motion [Electronic resource]. – Access mode: <https://www.framer.com/motion/>.
19. Frisbie, M. Professional JavaScript for Web Developers 5th Edition / Frisbie Matt. – Hoboken: John Wiley & Sons, 2023. – 1104 p.
20. Postman [Electronic resource]. – Access mode: <https://www.postman.com>.
21. Klingman, L. React Programming: The Big Nerd Ranch Guide / L. Klingman, A. Parker. – Atlanta: Big Nerd Ranch, 2023. – 420 p.
22. Що таке роздільна здатність екрана [Електрон. ресурс]. – Режим доступу: <https://samsungshop.com.ua/samsung-news/1151.html>.

ДОДАТОК А
Технічне завдання

Вступ

Використання програмного забезпечення для обліку та керування замовленнями у мережах кав'ярень є надзвичайно важливим у сучасному бізнес-середовищі. Зростаюча конкуренція та популярність кавової індустрії створюють потребу в ефективному управлінні, оптимізації процесів та якісному наданні послуг. Програмне забезпечення дозволяє автоматизувати та спростити рутинні операції, такі як облік запасів, управління персоналом, обробка замовлень та звітність, що у свою чергу допомагає підвищити ефективність бізнесу, сприяє зменшенню витрат та забезпеченню задоволеності клієнтів.

A.1 Підстава для розробки

Розробки здійснювалася на основі завдання на дипломну кваліфікаційну роботу з теми «Розробка програмного забезпечення для обліку та керування замовленнями у мережі кав'ярень», у відповідності з наказом №168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програмне забезпечення, розроблене для мережі кав'ярень, спрямоване на автоматизацію обліку та оформлення замовлень у кожному закладі мережі. Надає можливість ефективного керування такими аспектами, як персонал, меню, інвентар, замовлення та статистика продуктивності.

Користувачі програми можуть відстежувати продукти та інгредієнти, створювати й оновлювати категорії продуктів і пункти меню. Вони також можуть обробляти замовлення, отримувати статистику своєї роботи та керувати інформацією про себе.

А.3 Основні вимоги до розроблюваної програми

А.3.1 Вимоги до надійності

Програмний продукт має забезпечувати коректну обробку помилок у наступних випадках:

- коли користувач вводить недопустимі значення;
- коли користувач здійснює спробу несанкціонованого доступу до даних;
- коли можливість отримання даних з серверу відсутня.

А.3.2 Умови експлуатації

Для використання програмного забезпечення, користувачу необхідний персональний комп'ютер з доступом до мережі Інтернет та наявністю веббраузера.

Задля повноцінного користування системою, користувач повинен мати дані для входу в його обліковий запис.

Відповідно до розподілення ролей, користувач має взаємодіяти зі своїм робочим простором.

А.3.3 Вимоги до складу й параметрів технічних засобів

Для коректного функціонування серверної частини застосунку, апаратна частина серверу на якому розміщується ця складова має відповідати наступним вимогам:

- процесор Intel Xeon E5630 і вище(або аналогічне рішення інших виробників);
- оперативна пам'ять 32 Гб і вище;
- постійна пам'ять 1 Тб і вище.

Що стосується клієнтської частини, то вимоги обумовлюються тим, що формування та відображення клієнтської частини здійснюється на апаратній платформі клієнта, тому до нею висуваються наступні мінімальні вимоги:

- веббраузер з підтримкою ES6 та HTML5;
- процесор Intel Core i3 5-покоління і вище(або аналогічне рішення інших виробників);
- оперативна пам'ять 4 Гб і вище.

A.3.4 Вимоги до маркування й пакування

Програмне забезпечення може бути записане на будь-який носій інформації, пакування якого містить інформацію про назву програми – «Програмне забезпечення для обліку та керування замовленнями у мережі кав'ярень».

A.3.5 Вимоги до транспортування й зберігання

Транспортування та зберігання програмної системи має відповідати вимогам, які аналогічні вимогам до носія, на якому зберігається програма.

A.3.6 Вимоги до програмної документації

Програма повинна супроводжуватися інструкцією по експлуатації.

ДОДАТОК Б
Текст програми

Б.1 Текст модуля Order.service.ts

```
import { getTodayMidnight } from 'src/helpers/GetingDate.helper';
import { MenuPosition } from 'src/menu-position/entities/menu-position.entity';
import { OrderPositionService } from 'src/order-position/order-position.service';
import { PointsService } from 'src/points/points.service';
import { RecipeService } from 'src/recipe/recipe.service';
import { Shift, ShiftStatus } from 'src/shifts/entities/shift.entity';
import { CreateOrderDto } from '../dto/create-order.dto';
import { Order, OrderStatus, PaymentMethod } from '../entities/orders.entity';

@Injectable()
export class OrdersService {
  constructor(
    @InjectRepository(Order)
    private readonly orderRepository: Repository<Order>,
    @InjectRepository(MenuPosition)
    private readonly positionRepository: Repository<MenuPosition>,
    @InjectRepository(Shift)
    private readonly shiftRepository: Repository<Shift>,
    private readonly pointService: PointsService,
    private readonly recipeService: RecipeService,
    private readonly orderPositionService: OrderPositionService,
  ) {}

  async create(createOrderDto: CreateOrderDto, baristaID: number) {
    const currentShift = await this.shiftRepository.find({
      where: {
        barista: { id: baristaID },
        point: { id: createOrderDto.point.id },
      },
      relations: { point: true },
      order: { id: 'DESC' },
      take: 1,
    });

    if (!currentShift.length) {
      throw new BadRequestException('Shifts for this barista were not found');
    }

    if (
      currentShift[0].status.toString() === ShiftStatus.EndOfWork.toString()
    ) {
      throw new BadRequestException('First, you need to start the shift!');
    }

    const costOfIngredients = await this.recipeService.getCostOfIngredientsList(
      createOrderDto.point.id,
      createOrderDto.orderList,
    );

    const orderPositions = await this.orderPositionService.create(
      createOrderDto.orderList,
    );

    let newOrder = await this.orderRepository.save({
      ...createOrderDto,
      costOfIngredients,
```

```

        barista: { id: baristaID },
        orderList: orderPositions,
    });
    newOrder = await this.orderRepository.findOne({
        where: { id: newOrder.id },
        relations: { orderList: true, point: true, barista: true },
    });

    if (newOrder.paymentMethod.toString() === PaymentMethod.ByCash.toString()) {
        await this.pointService.putMoneyOnBalance(
            newOrder.point.id,
            baristaID,
            newOrder.totalAmount,
        );
    }
    await this.pointService.updateIngredientsOnPoint(
        newOrder.point.id,
        createOrderDto.orderList,
    );

    return newOrder;
}

async findAll(adminID: number) {
    const allOrders = await this.orderRepository.find({
        where: { point: { admin: { id: adminID } } },
        relations: { point: true, barista: true, orderList: true },
    });
    if (!allOrders.length) {
        throw new BadRequestException('Orders were not found');
    }
    return allOrders;
}

async findByPointId(pointID: number, adminID: number) {
    const existedOrderForPoint = await this.orderRepository.find({
        where: { point: { id: pointID, admin: { id: adminID } } },
    });
    if (!existedOrderForPoint.length) {
        throw new BadRequestException(
            `Orders for Point #${pointID} were not found`,
        );
    }
    return await this.orderRepository.find({
        where: { point: { id: pointID } },
        relations: { orderList: true, barista: true },
    });
}

async findOrdersByBaristaCurrentShift(baristaID: number, adminID: number) {
    const currentShift = await this.shiftRepository.find({
        where: {
            barista: { id: baristaID },
            point: { admin: { id: adminID } },
        },
    });

```

```

    },
    relations: { point: true },
    order: { id: 'DESC' },
    take: 1,
  });
  if (!currentShift.length) {
    return new BadRequestException('Shifts for this barista were not found');
  }
  if (
    currentShift[0].status.toString() === ShiftStatus.EndOfWork.toString()
  ) {
    return new BadRequestException('First, you need to start the shift!');
  }
  const currentShiftOrders = await this.orderRepository.find({
    where: {
      barista: { id: baristaID, admin: { id: adminID } },
      point: { id: currentShift[0].point.id },
      createdAt: Between(currentShift[0].time, new Date()),
    },
    relations: { orderList: true },
    order: { createdAt: 'DESC' },
  });

  if (!currentShiftOrders.length) {
    return new BadRequestException('Orders for current shift were not found');
  }
  return currentShiftOrders;
}

async findByBaristaIDAndForPeriod(
  baristaID: number,
  periodFrom: Date,
  periodTo: Date,
) {
  const existedOrdersForBarista =
    periodFrom < periodTo
      ? await this.orderRepository.find({
          where: {
            barista: { id: baristaID },
            createdAt: Between(periodFrom, periodTo),
          },
          relations: {
            orderList: true,
            point: true,
            barista: true,
          },
        })
      : await this.orderRepository.find({
          where: {
            barista: { id: baristaID },
            createdAt: Between(periodTo, periodFrom),
          },
          relations: {
            orderList: true,

```

```

        point: true,
        barista: true,
    },
    });
    if (!existedOrdersForBarista.length) {
        return new BadRequestException(
            `Any orders for barista #${baristaID} during selected period were not found`,
        );
    }
    return existedOrdersForBarista;
}

async findOne(id: number, adminID: number) {
    const existedOrder = await this.orderRepository.findOne({
        where: { id, point: { admin: { id: adminID } } },
    });
    if (!existedOrder) {
        throw new BadRequestException(`Order with #${id} was not found`);
    }
    return await this.orderRepository.findOne({
        where: { id },
        relations: { barista: true, orderList: true, point: true },
    });
}

async completeOrder(orderID: number, adminID: number) {
    const existedOrder = await this.orderRepository.findOne({
        where: { id: orderID, point: { admin: { id: adminID } } },
    });
    if (!existedOrder) {
        throw new BadRequestException(`Order with #${orderID} was not found`);
    }
    existedOrder.status = OrderStatus.Ready;
    return await this.orderRepository.save(existedOrder);
}

async calculateTotalSumOfOrdersOnTodayByBaristaID(
    baristaID: number,
    adminID: number,
) {
    const existedOrder = await this.orderRepository.find({
        where: {
            barista: { id: baristaID, admin: { id: adminID } },
            createdAt: Between(getTodayMidnight(), new Date()),
        },
    });
    if (!existedOrder.length) {
        throw new BadRequestException(
            `Orders on today by Barista #${baristaID} were not found`,
        );
    }
    return existedOrder.reduce((acc, cur) => {
        return acc + cur.totalAmount;
    }, 0);
}

```

```

    }
    async remove(id: number, adminID: number) {
      const existedOrderForPoint = await this.orderRepository.findOne({
        where: { id, barista: { admin: { id: adminID } } },
      });
      if (!existedOrderForPoint) {
        throw new BadRequestException(`Order with #${id} was not found`);
      }
      return await this.orderRepository.delete(id);
    }
  }
}

```

Б.2 Текст модуля Admin.service.ts

```

import { Barista } from 'src/barista/entities/barista.entity';
import { MailerSenderService } from 'src/mailer/mailer.service';
import { CreateAdminDto } from './dto/create-admin.dto';
import { UpdateAdminDto } from './dto/update-admin.dto';
import { Admin } from './entities/admin.entity';

@Injectable()
export class AdminService {
  constructor(
    @InjectRepository(Admin)
    private readonly adminRepository: Repository<Admin>,
    @InjectRepository(Barista)
    private readonly baristaRepository: Repository<Barista>,
    private readonly mailerService: MailerSenderService,
  ) {}

  async create(createAdminDto: CreateAdminDto) {
    await this.checkEmailUnique(createAdminDto.email);
    await this.mailerService.sendEmailWithLoginData(
      createAdminDto.surname + ' ' + createAdminDto.name,
      createAdminDto.email,
      createAdminDto.password,
      createAdminDto.email,
    );
    const newAdmin = {
      ...createAdminDto,
      password: await argon2.hash(createAdminDto.password),
    };
    return await this.adminRepository.save(newAdmin);
  }

  async findOne(id: number) {
    const existedAdmin = await this.adminRepository.findOne({
      where: { id },
    });
    if (!existedAdmin) {
      return new BadRequestException(
        `Any admin with #${id} hasn't already existed`,
      );
    }
  }
}

```

```

    }
    return existedAdmin;
}

async findOneByEmail(email: string) {
    const existedAdmin = await this.adminRepository.findOne({
        where: { email },
    });
    if (!existedAdmin) {
        return new BadRequestException(`Admin hasn't already existed`);
    }
    return existedAdmin;
}

async update(id: number, updateAdminDto: UpdateAdminDto) {
    const existedAdmin = await this.adminRepository.findOne({
        where: { id },
    });
    if (!existedAdmin) {
        throw new BadRequestException(
            `Admin with #${id} hasn't already existed`,
        );
    }

    if (updateAdminDto.email && existedAdmin.email !== updateAdminDto.email) {
        await this.checkEmailUnique(updateAdminDto.email);
        if (!updateAdminDto.password) {
            await this.mailerService.sendEmailAboutUpdatingLoginData(
                existedAdmin.surname + ' ' + existedAdmin.name,
                updateAdminDto.email,
                'Password wasn't changed. Use last one.',
                updateAdminDto.email,
            );
        }
    }

    if (updateAdminDto.password) {
        await this.mailerService.sendEmailAboutUpdatingLoginData(
            existedAdmin.surname + ' ' + existedAdmin.name,
            existedAdmin.email,
            updateAdminDto.password,
            existedAdmin.email,
        );
        updateAdminDto.password = await argon2.hash(updateAdminDto.password);
    }
    return await this.adminRepository.update(id, updateAdminDto);
}

async remove(id: number) {
    const existedAdmin = await this.adminRepository.findOne({
        where: { id },
    });
    if (!existedAdmin) {
        throw new BadRequestException(

```

```

        `Admin with #${id} hasn't already existed`,
    );
}
return await this.adminRepository.delete(id);
}

async confirmEmail(id: number) {
    const existedAdmin = await this.adminRepository.findOne({
        where: { id },
    });
    if (!existedAdmin) {
        throw new BadRequestException(
            `Admin with #${id} hasn't already existed`,
        );
    }
    existedAdmin.isEmailConfirmed = true;
    return this.adminRepository.save(existedAdmin);
}

async checkEmailUnique(email: string) {
    const existedBarista = await this.baristaRepository.findOne({
        where: { email },
    });
    const existedAdmin = await this.adminRepository.findOne({
        where: { email },
    });
    if (existedBarista || existedAdmin) {
        throw new BadRequestException(
            `Account with email ${email} has already been existed`,
        );
    }
}
}
}

```

Б.3 Текст модуля Ingredient.service.ts

```

import { Point } from 'src/points/entities/points.entity';
import { Shift, ShiftStatus } from 'src/shifts/entities/shift.entity';
import { CreateIngredientDto } from './dto/create-ingredient.dto';
import { UpdateIngredientDto } from './dto/update-ingredient.dto';
import { Ingredient } from './entities/ingredient.entity';

@Injectable()
export class IngredientsService {
    constructor(
        @InjectRepository(Ingredient)
        private readonly ingredientRepository: Repository<Ingredient>,
        @InjectRepository(Point)
        private readonly pointRepository: Repository<Point>,
        @InjectRepository(Shift)
        private readonly shiftRepository: Repository<Shift>,
    ) {}
}

```

```

async create(
  createIngredientDro: CreateIngredientDto,
  pointID: number,
  adminID: number,
) {
  const existedPoint = await this.pointRepository.findOne({
    where: {
      id: pointID,
      admin: { id: adminID },
    },
  });
  if (!existedPoint) {
    return new BadRequestException(`Any point #${[pointID]} has not existed`);
  }
  const existedIngredient = await this.ingredientRepository.findOne({
    where: {
      name: createIngredientDro.name,
      point: { id: pointID },
    },
  });
  if (existedIngredient) {
    throw new BadRequestException(
      `Ingredient with name [${createIngredientDro.name}] has already existed on Point with
    #${pointID}`,
    );
  }
  return await this.ingredientRepository.save({
    ...createIngredientDro,
    point: { id: pointID },
  });
}

async findAll(adminID: number) {
  return await this.ingredientRepository.find({
    where: { point: { admin: { id: adminID } } },
    relations: { point: true },
  });
}

async findAllOnPoint(pointID: number, adminID: number) {
  const existedIngredient = await this.ingredientRepository.find({
    where: { point: { id: pointID, admin: { id: adminID } } },
  });
  if (!existedIngredient.length) {
    throw new BadRequestException(`No ingredients on Point #${pointID}`);
  }
  return existedIngredient;
}

async findAllOnPointForBarista(baristaID: number, adminID: number) {
  let lastShiftPointID: number;
  const lastShift = await this.shiftRepository.find({
    where: {
      barista: { id: baristaID },
      point: { admin: { id: adminID } },
    },
  },

```

```

        relations: { point: true },
        order: { id: 'DESC' },
        take: 1,
    });
    if (lastShift[0].status.toString() === ShiftStatus.StartOfWork.toString()) {
        lastShiftPointID = lastShift[0].point.id;
    }
    if (!lastShiftPointID)
        throw new BadRequestException(`No ingredients on this Point`);
    const existedIngredient = await this.ingredientRepository.find({
        where: {
            point: {
                id: lastShiftPointID,
                barista: { id: baristaID },
                admin: { id: adminID },
            },
        },
        order: {
            quantity: 'ASC',
        },
    });
    if (!existedIngredient.length) {
        throw new BadRequestException(`No ingredients on this Point`);
    }
    return existedIngredient;
}

async findOneByID(id: number, adminID: number) {
    const existedIngredient = await this.ingredientRepository.findOne({
        where: {
            id,
            point: { admin: { id: adminID } },
        },
        relations: { recipes: true, point: true },
    });
    if (!existedIngredient) {
        throw new BadRequestException(`Ingredient with #${id} was not found`);
    }
    return existedIngredient;
}

async findOneByName(name: string, adminID: number) {
    const existedIngredient = await this.ingredientRepository.findOne({
        where: { name, point: { admin: { id: adminID } } },
        relations: { recipes: true, point: true },
    });
    if (!existedIngredient) {
        throw new BadRequestException(
            `Ingredients with name "${name}" was not found`,
        );
    }
    return existedIngredient;
}

async update(
    id: number,

```

```

    adminID: number,
    updateIngredientDto: UpdateIngredientDto,
  ) {
    const existedIngredient = await this.ingredientRepository.findOne({
      where: { id, point: { admin: { id: adminID } } },
    });
    if (!existedIngredient) {
      throw new BadRequestException(`Ingredient with #${id} was not found`);
    }
    if (updateIngredientDto.name) {
      const existedIngredient = await this.ingredientRepository.findOne({
        where: {
          name: updateIngredientDto.name,
          point: { admin: { id: adminID } },
        },
      });
      if (existedIngredient) {
        throw new BadRequestException(
          `Ingredient with name ${updateIngredientDto.name} has already existed`,
        );
      }
    }
    return await this.ingredientRepository.update(id, updateIngredientDto);
  }

  async remove(id: number, adminID: number) {
    const existedIngredient = await this.ingredientRepository.findOne({
      where: { id, point: { admin: { id: adminID } } },
    });
    if (!existedIngredient) {
      throw new BadRequestException(`Ingredient with #${id} was not found`);
    }
    existedIngredient.recipes = [];
    await this.ingredientRepository.save(existedIngredient);
    return await this.ingredientRepository.delete(id);
  }

  async addQuantityOfIngredientsOnPoint(
    baristaID: number,
    ingredientID: number,
    adminID: number,
    addQuantityOfIngredient,
  ) {
    const existedIngredient = await this.ingredientRepository.findOne({
      where: {
        id: ingredientID,
        point: { barista: { id: baristaID, admin: { id: adminID } } },
      },
    });
    if (!existedIngredient) {
      throw new BadRequestException(
        `Ingredient with #${ingredientID} was not found`,
      );
    }
  }

```

```

    existedIngredient.quantity += addQuantityOfIngredient;
    return await this.ingredientRepository.save(existedIngredient);
  }
}

```

Б.4 Текст модуля MenuPosition.service.ts

```

import { InjectRepository } from '@nestjs/typeorm';
import { Point } from 'src/points/entities/points.entity';
import { Recipe } from 'src/recipe/entities/recipe.entity';
import { Shift, ShiftStatus } from 'src/shifts/entities/shift.entity';
import { CreatePositionDto } from '../dto/create-position.dto';
import { UpdatePositionDto } from '../dto/update-position.dto';
import { MenuPosition } from '../entities/menu-position.entity';

@Injectable()
export class MenuPositionService {
  constructor(
    @InjectRepository(MenuPosition)
    private readonly menuPositionRepository: Repository<MenuPosition>,
    @InjectRepository(Recipe)
    private readonly recipeRepository: Repository<Recipe>,
    @InjectRepository(Point)
    private readonly pointRepository: Repository<Point>,
    @InjectRepository(Shift)
    private readonly shiftRepository: Repository<Shift>,
  ) {}

  async create(
    menuPositionCreateDto: CreatePositionDto,
    pointID: number,
    adminID: number,
  ) {
    const existedPosition = await this.menuPositionRepository.findOne({
      where: { name: menuPositionCreateDto.name, point: { id: pointID } },
    });
    if (existedPosition) {
      return new BadRequestException(
        `Menu-Position with name '${existedPosition.name}' has already existed`,
      );
    }

    const existedPoint = await this.pointRepository.findOne({
      where: {
        admin: { id: adminID },
        id: pointID,
      },
      relations: { ingredients: true },
    });
    if (!existedPoint) {
      throw new BadRequestException(`Point #${pointID} was not found`);
    }
    if (menuPositionCreateDto.ingredients) {
      for (const ingredient of menuPositionCreateDto.ingredients) {

```

```

        if (
            !existedPoint.ingredients.find((item) => item.id === ingredient.id)
        ) {
            throw new BadRequestException(
                'Not all ingredients are existed on Point',
            );
        }
    }
}

const recipe = await this.recipeRepository.save({
    ingredients:
        menuPositionCreateDto.ingredients &&
        menuPositionCreateDto.ingredients.length
        ? menuPositionCreateDto.ingredients.map((item) => {
            return { id: item.id };
        })
        : [],
    stepsToReproduce:
        menuPositionCreateDto.stepsToReproduce &&
        menuPositionCreateDto.stepsToReproduce.length
        ? menuPositionCreateDto.stepsToReproduce
        : [],
});

const newPosition = await this.menuPositionRepository.save({
    ...menuPositionCreateDto,
    point: { id: pointID },
    recipe: recipe,
});

await this.recipeRepository.save({ ...recipe, menuPosition: newPosition });
return newPosition;
}

async getMenu(pointID: number, adminID: number) {
    let existedMenuPositions = await this.menuPositionRepository.find({
        where: { point: { id: pointID, admin: { id: adminID } } },
        relations: { category: true, recipe: true, discount: true },
    });

    if (!existedMenuPositions.length) {
        throw new BadRequestException(
            `Any menu position on this point was not found`,
        );
    }

    existedMenuPositions = existedMenuPositions.map((pos) => {
        if (pos.discount && pos.discount.endAt < new Date()) {
            pos.discount = null;
        }
        return pos;
    });

    return existedMenuPositions;
}

async getMenuByBaristaID(baristaID: number, adminID: number) {

```

```

let lastShiftPointID: number;
const lastShift = await this.shiftRepository.find({
  where: {
    barista: { id: baristaID },
    point: { admin: { id: adminID } },
  },
  relations: { point: true },
  order: { id: 'DESC' },
  take: 1,
});
if (lastShift[0].status.toString() === ShiftStatus.StartOfWork.toString()) {
  lastShiftPointID = lastShift[0].point.id;
}
if (!lastShiftPointID)
  throw new BadRequestException(`No menu positions on this Point`);

let existedMenuPositions = await this.menuPositionRepository.find({
  where: { point: { id: lastShiftPointID, admin: { id: adminID } } },
  relations: { category: true, recipe: true, discount: true },
});
if (!existedMenuPositions.length) {
  throw new BadRequestException(
    `Any menu position on this point was not found`,
  );
}
existedMenuPositions = existedMenuPositions.map((pos) => {
  if (pos.discount && pos.discount.endAt < new Date()) {
    pos.discount = null;
  }
  return pos;
});
return existedMenuPositions;
}

async findOne(id: number, pointID: number, adminID: number) {
  const existedPosition = await this.menuPositionRepository.findOne({
    where: { id, point: { id: pointID, admin: { id: adminID } } },
  });
  if (!existedPosition) {
    throw new BadRequestException(`Menu Position with #${id} was not found`);
  }
  return await this.menuPositionRepository.findOne({
    where: { id },
    relations: { category: true, recipe: true, discount: true },
  });
}

async update(
  id: number,
  pointID: number,
  adminID: number,
  updatedMenuPositionDto: UpdatePositionDto,
) {
  const existedPosition = await this.menuPositionRepository.findOne({
    where: { id, point: { id: pointID, admin: { id: adminID } } },
  });

```

```

        relations: { recipe: true },
    });
    if (!existedPosition) {
        throw new BadRequestException(`Menu Position with #${id} was not found`);
    }
    if (updatedMenuPositionDto.name) {
        const existedPosition = await this.menuPositionRepository.findOne({
            where: {
                name: updatedMenuPositionDto.name,
                point: { id: pointID, admin: { id: adminID } },
            },
            relations: { recipe: true },
        });
        if (existedPosition) {
            throw new BadRequestException(
                `Menu Position with name ${updatedMenuPositionDto.name} has already existed`,
            );
        }
    }
    if (
        !updatedMenuPositionDto.ingredients &&
        !updatedMenuPositionDto.stepsToReproduce
    ) {
        return await this.menuPositionRepository.update(
            id,
            updatedMenuPositionDto,
        );
    }
    const existedPoint = await this.pointRepository.findOne({
        where: { id: pointID },
        relations: { ingredients: true },
    });
    const recipe = await this.recipeRepository.findOne({
        where: { id: existedPosition.recipe.id },
        relations: { ingredients: true },
    });

    let ingredients;
    if (updatedMenuPositionDto.ingredients) {
        if (updatedMenuPositionDto.ingredients.length) {
            for (const ingredient of updatedMenuPositionDto.ingredients) {
                if (
                    !existedPoint.ingredients.find((item) => item.id === ingredient.id)
                ) {
                    throw new BadRequestException(
                        'Not all ingredients are existed on Point',
                    );
                }
            }
            ingredients = [...updatedMenuPositionDto.ingredients];
        } else {
            ingredients = [];
        }
    } else {

```

```

    ingredients = recipe.ingredients;
  }

  const updatedRecipe = {
    ...recipe,
    ingredients: ingredients,
    stepsToReproduce: updatedMenuPositionDto.stepsToReproduce
      ? !updatedMenuPositionDto.stepsToReproduce.length
        ? []
        : updatedMenuPositionDto.stepsToReproduce
      : recipe.stepsToReproduce,
  };
  await this.recipeRepository.save(updatedRecipe);
  return existedPosition;
}

async delete(id: number, pointID: number, adminID: number) {
  const existedPosition = await this.menuPositionRepository.findOne({
    where: { id, point: { id: pointID, admin: { id: adminID } } },
    relations: { recipe: true },
  });
  if (!existedPosition) {
    throw new BadRequestException(`Menu Position with #${id} was not found`);
  }
  const recipe = await this.recipeRepository.findOne({
    where: { id: existedPosition.recipe.id },
    relations: { ingredients: true },
  });
  recipe.ingredients = [];
  await this.recipeRepository.save(recipe);
  await this.recipeRepository.delete(recipe.id);
  return await this.menuPositionRepository.delete(id);
}

async getRecipe(id: number, pointID: number, adminID: number) {
  const existedPosition = await this.menuPositionRepository.findOne({
    where: { id, point: { id: pointID, admin: { id: adminID } } },
  });
  if (!existedPosition) {
    throw new BadRequestException(`Menu Position with #${id} was not found`);
  }
  return await this.recipeRepository.findOne({
    where: { menuPosition: { id } },
    relations: { ingredients: true, menuPosition: true },
  });
}
}

```

Б.5 Текст модуля Shift.service.ts

```

import { getHourDifference } from 'src/helpers/GetHourDifference.helper';
import { Order } from 'src/orders/entities/orders.entity';
import { OrdersService } from 'src/orders/orders.service';
import { Point } from 'src/points/entities/points.entity';

```

```

import { Shift, ShiftStatus } from './entities/shift.entity';

@Injectable()
export class ShiftsService {
  constructor(
    @InjectRepository(Shift)
    private readonly shiftRepository: Repository<Shift>,
    @InjectRepository(Point)
    private readonly pointRepository: Repository<Point>,
    private readonly orderService: OrdersService,
  ) {}
  async create(
    shiftStatus: ShiftStatus,
    pointID: number,
    baristaId: number,
    adminID: number,
  ) {
    const isBaristaOnPoint = (
      await this.pointRepository.findOne({
        where: { id: pointID, admin: { id: adminID } },
        relations: { barista: true },
      })
    )?.barista.find((item) => item.id === baristaId);
    if (!isBaristaOnPoint) {
      throw new BadRequestException(
        `Barista #${baristaId} does not have access to this point #${pointID}`,
      );
    }
    if (shiftStatus.toString() === ShiftStatus.StartOfWork.toString()) {
      const lastShift = await this.shiftRepository.find({
        where: {
          barista: { id: baristaId },
          point: { admin: { id: adminID } },
        },
        relations: { point: true },
        order: { id: 'DESC' },
        take: 1,
      });

      if (
        lastShift.length &&
        lastShift[0].status === ShiftStatus.StartOfWork.toString() &&
        lastShift[0].point.id !== pointID
      ) {
        throw new BadRequestException(
          `Please end your shift at the point [${lastShift[0].point.name}]`,
        );
      }
    }
    if (
      !lastShift.length ||
      lastShift[0].status === ShiftStatus.EndOfWork.toString()
    ) {
      return await this.shiftRepository.save({
        status: shiftStatus,

```

```

        point: { id: pointID },
        baristaSalary: 0,
        barista: { id: baristaId },
    });
} else throw new BadRequestException('Last shift hasn't been ended');
} else if (shiftStatus.toString() === ShiftStatus.EndOfWork.toString()) {
    const lastShift = await this.shiftRepository.find({
        where: {
            barista: { id: baristaId },
            point: { admin: { id: adminID } },
        },
        order: { id: 'DESC' },
        take: 1,
    });
    if (!lastShift.length)
        throw new BadRequestException(
            'You can't end of shift until you start it',
        );
    if (lastShift[0].status === ShiftStatus.StartOfWork.toString()) {
        const lastDayOrders =
            (await this.orderService.findByBaristaIDAndForPeriod(
                baristaId,
                lastShift[0].time,
                new Date(),
            )) as Order[];
        if (!lastDayOrders.length) {
            return await this.shiftRepository.save({
                status: shiftStatus,
                point: { id: pointID },
                baristaSalary: 0,
                barista: { id: baristaId },
            });
        }
        const baristaSalaryFromPercent = lastDayOrders.reduce((acc, cur) => {
            const salaryFromPercent =
                (cur.totalAmount * cur.barista.percentFromEarnings) / 100;
            return acc + salaryFromPercent;
        }, 0);
        const timeStartShift = lastShift[0].time;
        const timeEndShift = new Date();
        const totalBaristaSalary =
            baristaSalaryFromPercent +
            lastDayOrders[0].barista.fixedHourRate *
                getHourDifference(timeStartShift, timeEndShift);
        return await this.shiftRepository.save({
            status: shiftStatus,
            point: { id: pointID },
            baristaSalary: Math.round(totalBaristaSalary),
            barista: { id: baristaId },
        });
    } else
        throw new BadRequestException(
            'You can't end of shift until you start it',
        );
}

```

```

    }
  }

  async deleteShiftByID(id: number, adminID: number) {
    const endShift = await this.shiftRepository.findOne({
      where: {
        id,
        barista: { admin: { id: adminID } },
        status: ShiftStatus.EndOfWork,
      },
      relations: { barista: true },
    });
    if (!endShift) {
      throw new BadRequestException(`Shift with #${id} was not found`);
    }
    const startShift = await this.shiftRepository.find({
      where: {
        barista: { id: endShift.barista.id, admin: { id: adminID } },
        status: ShiftStatus.StartOfWork,
        time: LessThan(endShift.time),
      },
      order: { id: 'DESC' },
      take: 1,
    });
    return await this.shiftRepository.remove([startShift[0], endShift]);
  }

  async getAllShiftsByBarista(id: number, adminID: number) {
    const shifts = await this.shiftRepository.find({
      where: {
        status: ShiftStatus.EndOfWork,
        barista: { id, admin: { id: adminID } },
      },
      relations: {
        point: true,
      },
    });
    if (!shifts.length) {
      return new BadRequestException(
        `Any shifts for Barista with #${id} weren't found`,
      );
    }
    return shifts;
  }

  async getAllShiftsByPoint(id: number, adminID: number) {
    const shifts = await this.shiftRepository.find({
      where: { point: { id, admin: { id: adminID } } },
      relations: {
        barista: true,
      },
    });
    if (!shifts.length) {
      throw new BadRequestException(`Shifts for Point #${id} weren't found`);
    }
  }

```

```

    return shifts;
}

async getAllShifts(adminID: number) {
    const shifts = await this.shiftRepository.find({
        where: { point: { admin: { id: adminID } } },
        relations: {
            barista: true,
            point: true,
        },
    });
    if (!shifts.length) {
        throw new BadRequestException(`Shifts were not found`);
    }
    return shifts;
}

async getCurrentStatus(baristaID: number, adminID: number) {
    const lastShift = await this.shiftRepository.find({
        where: {
            barista: { id: baristaID, admin: { id: adminID } },
        },
        order: { id: 'DESC' },
        take: 1,
        relations: { point: true },
    });
    if (!lastShift.length) {
        throw new BadRequestException(
            `First shift of barista #${baristaID} has not started yet`,
        );
    }
    if (lastShift[0].status === ShiftStatus.EndOfWork) {
        return { status: ShiftStatus.EndOfWork };
    } else {
        return { status: ShiftStatus.StartOfWork, point: lastShift[0].point };
    }
}

async getBaristaSalaryAndCountOfShiftsForPeriod(
    adminID: number,
    baristaID: number,
    periodFrom: Date,
    periodTo: Date,
) {
    let shifts: Shift[];
    if (periodFrom > periodTo) {
        shifts = await this.shiftRepository.find({
            where: {
                barista: { id: baristaID },
                point: { admin: { id: adminID } },
                status: ShiftStatus.EndOfWork,
                time: Between(periodTo, periodFrom),
            },
        });
    }
}

```

```

    } else {
      shifts = await this.shiftRepository.find({
        where: {
          barista: { id: baristaID },
          point: { admin: { id: adminID } },
          status: ShiftStatus.EndOfWork,
          time: Between(periodFrom, periodTo),
        },
      });
    }
    if (!shifts.length) {
      throw new BadRequestException(`Shifts were not found`);
    }
    const totalBaristaSalary = shifts.reduce(
      (acc, cur) => acc + cur.baristaSalary,
      0,
    );
    const countOfShifts = shifts.length;
    return { totalBaristaSalary, countOfShifts };
  }
  async getAllBaristasSalaryAndCountOfShiftsForPeriod(
    adminID: number,
    periodFrom: Date,
    periodTo: Date,
  ) {
    let shifts: Shift[];
    if (periodFrom > periodTo) {
      shifts = await this.shiftRepository.find({
        where: {
          point: { admin: { id: adminID } },
          status: ShiftStatus.EndOfWork,
          time: Between(periodTo, periodFrom),
        },
        relations: {
          barista: true,
        },
      });
    } else {
      shifts = await this.shiftRepository.find({
        where: {
          point: { admin: { id: adminID } },
          status: ShiftStatus.EndOfWork,
          time: Between(periodFrom, periodTo),
        },
        relations: {
          barista: true,
        },
      });
    }
    const baristas: BaristasData[] = shifts.map((shift) => {
      return {
        baristaID: shift.barista.id,
        baristaFullName: shift.barista.name + ' ' + shift.barista.surname,
      };
    });
  }

```

```

    });
    if (!shifts.length) {
        throw new BadRequestException(`Shifts were not found`);
    }
    const totalData = [];
    for (const barista of baristas) {
        const baristaShifts = shifts.filter(
            (shift) => shift.barista.id === barista.baristaID,
        );
        const totalBaristaSalary = baristaShifts.reduce(
            (acc, cur) => acc + cur.baristaSalary,
            0,
        );
        const countOfShifts = baristaShifts.length;
        totalData.push({
            ...barista,
            totalBaristaSalary,
            countOfShifts,
        });
    }
    return totalData;
}
}

type BaristasData = {
    baristaID: number;
    baristaFullName: string;
};

```

Б.6 Текст модуля Statistics.service.ts

```

import { Barista } from 'src/barista/entities/barista.entity';
import { Category } from 'src/categories/entities/category.entity';
import { Ingredient } from 'src/ingredients/entities/ingredient.entity';
import { MenuPosition } from 'src/menu-position/entities/menu-position.entity';
import { OrderPosition } from 'src/order-position/entities/order-position.entity';
import { Order } from 'src/orders/entities/orders.entity';
import { Point } from 'src/points/entities/points.entity';
import { Shift, ShiftStatus } from 'src/shifts/entities/shift.entity';
import { TableItem } from './interfaces/TableItem.interface';

@Injectable()
export class StatisticsService {
    constructor(
        @InjectRepository(Category)
        private readonly categoriesRepository: Repository<Category>,
        @InjectRepository(OrderPosition)
        private readonly orderPositionRepository: Repository<OrderPosition>,
        @InjectRepository(MenuPosition)
        private readonly menuPositionRepository: Repository<MenuPosition>,
        @InjectRepository(Point)
        private readonly pointRepository: Repository<Point>,

```

```

@InjectRepository(Barista)
private readonly baristaRepository: Repository<Barista>,
@InjectRepository(Ingredient)
private readonly ingredientRepository: Repository<Ingredient>,
@InjectRepository(Shift)
private readonly shiftRepository: Repository<Shift>,
@InjectRepository(Order)
private readonly orderRepository: Repository<Order>,
) {}

async getPopularityOfMenuPositionByPoint(
  pointID: number,
  adminID: number,
  from: Date,
  to: Date,
) {
  const allOrdersPositionByPoint = await this.orderPositionRepository.find({
    where: {
      order: {
        point: { id: pointID, admin: { id: adminID } },
        createdAt: from < to ? Between(from, to) : Between(to, from),
      },
    },
    relations: { menuPosition: true },
  });
  const tableOfPopularity = new Map<number, TableItem>();
  allOrdersPositionByPoint.map((item) => {
    const position = tableOfPopularity.get(item.menuPosition.id);
    if (position) {
      position.number_of_ordering += item.quantity;
      tableOfPopularity.set(item.menuPosition.id, position);
    } else {
      const key = item.menuPosition.id;
      const value: TableItem = {
        name: item.menuPosition.name,
        category: item.menuPosition.category.title,
        number_of_ordering: item.quantity,
      };
      tableOfPopularity.set(key, value);
    }
  });
  const resultList = [];
  for (const [key, value] of tableOfPopularity) {
    resultList.push({
      id: key,
      ...value,
    });
  }
  return resultList;
}

async getExpensesAndIncomesByPoint(
  pointID: number,
  adminID: number,

```

```

    from: Date,
    to: Date,
  ) {
    const ordersByPoint = await this.orderRepository.find({
      where: {
        point: { id: pointID, admin: { id: adminID } },
        createdAt: from < to ? Between(from, to) : Between(to, from),
      },
      relations: { point: true },
    });
    const incomesFromOrders = ordersByPoint.reduce((acc, cur) => {
      return acc + cur.totalAmount - cur.costOfIngredients;
    }, 0);
    const shiftsByPoint = await this.shiftRepository.find({
      where: {
        point: { id: pointID, admin: { id: adminID } },
        status: ShiftStatus.EndOfWork,
        time: from < to ? Between(from, to) : Between(to, from),
      },
    });
    const expensesOnShifts = shiftsByPoint.reduce(
      (acc, cur) => acc + cur.baristaSalary,
      0,
    );

    return {
      id: pointID,
      name: ordersByPoint.length
        ? ordersByPoint[0].point.name
        : 'Information was not found',
      orders: {
        orders_count: ordersByPoint.length,
        orders_sum: incomesFromOrders,
      },
      shifts: {
        shifts_count: shiftsByPoint.length,
        shifts_sum: expensesOnShifts,
      },
      profit: incomesFromOrders - expensesOnShifts,
    };
  }
}

async getAllExpensesAndIncomes(adminID: number, from: Date, to: Date) {
  type PointData = {
    name: string;
    incomes: number;
    expenses: number;
  };

  const totalData = new Map<number, PointData>();
  const ordersByPoint = await this.orderRepository.find({
    where: {
      point: { admin: { id: adminID } },
      createdAt: from < to ? Between(from, to) : Between(to, from),
    },
    relations: { point: true },
  });

```

```

});

ordersByPoint.map((order) => {
  const item = totalData.get(order.point.id);
  if (!item) {
    totalData.set(order.point.id, {
      name: order.point.name,
      incomes: order.totalAmount - order.costOfIngredients,
      expenses: 0,
    });
  } else {
    item.incomes += order.totalAmount - order.costOfIngredients;
    totalData.set(order.point.id, item);
  }
});

const shiftsByPoint = await this.shiftRepository.find({
  where: {
    point: { admin: { id: adminID } },
    status: ShiftStatus.EndOfWork,
    time: from < to ? Between(from, to) : Between(to, from),
  },
  relations: { point: true },
});

shiftsByPoint.map((shift) => {
  const item = totalData.get(shift.point.id);
  if (!item) {
    totalData.set(shift.point.id, {
      name: shift.point.name,
      incomes: 0,
      expenses: shift.baristaSalary,
    });
  } else {
    item.expenses += shift.baristaSalary;
    totalData.set(shift.point.id, item);
  }
});

const resultList = [];
for (const [key, value] of totalData) {
  resultList.push({
    id: key,
    ...value,
  });
}

return resultList;
}

async getBaristaInfoById(
  baristaID: number,
  adminID: number,
  from: Date,
  to: Date,
) {
  const baristaShifts = await this.shiftRepository.find({

```

```

    where: {
      barista: { id: baristaID, admin: { id: adminID } },
      status: ShiftStatus.EndOfWork,
      time: from < to ? Between(from, to) : Between(to, from),
    },
    relations: { barista: true, point: true },
  });
const shifts_count = baristaShifts.length;
const totalSalary = baristaShifts.reduce((acc, cur) => {
  return acc + cur.baristaSalary;
}, 0);

return {
  barista_name: baristaShifts[0]
    ? baristaShifts[0].barista.surname + ' ' + baristaShifts[0].barista.name
    : 'Not found',
  barista_email: baristaShifts[0]
    ? baristaShifts[0].barista.email
    : 'Not found',
  barista_phone_number: baristaShifts[0]
    ? baristaShifts[0].barista.phoneNumber
    : 'Not found',
  barista_date_of_employment: baristaShifts[0]
    ? baristaShifts[0].barista.dateOfEmployment
    : 'Not found',
  barista_hour_rate: baristaShifts[0]
    ? baristaShifts[0].barista.fixedHourRate
    : 'Not found',
  barista_percent: baristaShifts[0]
    ? baristaShifts[0].barista.percentFromEarnings
    : 'Not found',
  shifts_count,
  totalSalary,
};
}

async getAllBaristasInfo(adminID: number, from: Date, to: Date) {
  type PointData = {
    name: string;
    salary: number;
    count_of_shifts: number;
  };
  const totalData = new Map<number, PointData>();

  const baristaShifts = await this.shiftRepository.find({
    where: {
      barista: { admin: { id: adminID } },
      status: ShiftStatus.EndOfWork,
      time: from < to ? Between(from, to) : Between(to, from),
    },
    relations: { barista: true, point: true },
  });

  baristaShifts.map((shift) => {

```

```

const item = totalData.get(shift.barista.id);
if (!item) {
  totalData.set(shift.barista.id, {
    name: shift.barista.surname + ' ' + shift.barista.name,
    count_of_shifts: 1,
    salary: shift.baristaSalary,
  });
} else {
  item.count_of_shifts += 1;
  item.salary += shift.baristaSalary;
  totalData.set(shift.barista.id, item);
}
});

const resultList = [];
for (const [key, value] of totalData) {
  resultList.push({
    id: key,
    ...value,
  });
}
return resultList;
}

async getConsumptionOfIngredientsByPoint(
  pointID: number,
  adminID: number,
  from: Date,
  to: Date,
) {
  type IngredientData = {
    name: string;
    price: number;
    consumption: number;
  };
  const totalData = new Map<number, IngredientData>();
  const orderPositionsByPoint = await this.orderPositionRepository.find({
    where: {
      menuPosition: { point: { id: pointID, admin: { id: adminID } } },
      order: { createdAt: from < to ? Between(from, to) : Between(to, from) },
    },
    relations: { menuPosition: true },
  });

  orderPositionsByPoint.map((position) => {
    position.menuPosition.recipe.ingredients.map((ingredient) => {
      const data = totalData.get(ingredient.id);
      if (!data) {
        totalData.set(ingredient.id, {
          name: ingredient.name,
          price: ingredient.price,
          consumption: position.quantity,
        });
      } else {

```

```

        data.consumption += position.quantity;
        totalData.set(ingredient.id, data);
    }
});
const resultList = [];
for (const [key, value] of totalData) {
    resultList.push({
        id: key,
        ...value,
    });
}
return resultList;
}

async getOrdersCountByCategories(adminID: number, from: Date, to: Date) {
    type CategoryData = {
        title: string;
        orders_count: number;
        total_income: number;
    };
    const totalData = new Map<number, CategoryData>();
    const orderPositionsByPoint = await this.orderPositionRepository.find({
        where: {
            menuPosition: { point: { admin: { id: adminID } } },
            order: { createdAt: from < to ? Between(from, to) : Between(to, from) },
        },
        relations: { menuPosition: true, order: true },
    });

    orderPositionsByPoint.map((orderPosition) => {
        const data = totalData.get(orderPosition.menuPosition.category.id);
        if (!data) {
            totalData.set(orderPosition.menuPosition.category.id, {
                title: orderPosition.menuPosition.category.title,
                total_income:
                    (orderPosition.menuPosition.price -
                     orderPosition.menuPosition.recipe.ingredients.reduce(
                         (acc, cur) => acc + cur.price,
                         0,
                     )) *
                    orderPosition.quantity,
                orders_count: 1,
            });
        } else {
            data.orders_count += 1;
            data.total_income +=
                orderPosition.menuPosition.price -
                orderPosition.menuPosition.recipe.ingredients.reduce(
                    (acc, cur) => acc + cur.price,
                    0,
                ) *
                orderPosition.quantity;
            totalData.set(orderPosition.menuPosition.category.id, data);
        }
    });
}

```

```

    }
  });
  const resultList = [];
  for (const [key, value] of totalData) {
    resultList.push({
      id: key,
      ...value,
    });
  }
  return resultList;
}
}

```

Б.7 Текст модуля Admin.tsx

```

import Button from '../components/Button/Button'
import { getTokenFromLocalStorage } from '../helper/localstorage.helper'
import { isValidEmail } from '../helper/validate-email.helper'
import { useAppSelector } from '../store/hooks'
import { updateAdminInfo } from '../store/slice/admin.slice'
import { IUpdateAdminInfo } from '../types/IUpdateAdminInfo'

const Admin: FC = () => {
  const admin = useAppSelector(state => state.admin.admin)
  const dispatch = useDispatch()
  const [name, setName] = useState(admin?.name)
  const [surname, setSurname] = useState(admin?.surname)
  const [email, setEmail] = useState(admin?.email)
  const [password, setPassword] = useState('')

  const submitHandler = async (e: React.FormEvent<HTMLFormElement>) => {
    try {
      e.preventDefault()
      if (!admin || !email?.length || !name?.length || !surname?.length)

      if (!isValidEmail(email)) return
      let dataToUpdate: IUpdateAdminInfo = {}
      if (name !== admin.name) {
        dataToUpdate = { ...dataToUpdate, name }
      }
      if (surname !== admin.surname) {
        dataToUpdate = { ...dataToUpdate, surname }
      }
      if (email !== admin.email) {
        dataToUpdate = { ...dataToUpdate, email }
      }
      if (password.length) {
        dataToUpdate = { ...dataToUpdate, password }
      }
      if (
        dataToUpdate.email ||
        dataToUpdate.name ||

```

```

        dataToUpdate.password ||
        dataToUpdate.surname
    ) {
        await axios.patch(`http://localhost:3000/api/admin`,
dataToUpdate, {
            headers: {
                Authorization: 'Bearer ' +
getTokenFromLocalStorage(),
            },
        })
        toast.success('Data was successfully updated')
        dispatch(updateAdminInfo(dataToUpdate))
    }
} catch (err: any) {
    const error = err.response?.data.message
    toast.error(error.toString())
}

return (
    <motion.div
        initial={{ opacity: 0, y: -20 }}
        animate={{ opacity: 1, y: 0 }}
        exit={{ opacity: 0, y: -20 }}
        transition={{ duration: 0.7 }}
        className='bg-zinc-700 mx-auto mt-20 w-[40%] rounded-3xl flex items-
center h-[60vh] shadow-2xl text-white font-roboto'
    >
        <User className='w-[50%] h-full border-r-4' />
        <form
            className='w-[50%] h-full bg-zinc-500 rounded-r-3xl p-3 flex
flex-col gap-[1vh] justify-center'
            onSubmit={submitHandler}
        >
            <label className='text-[2vh] uppercase font-bold px-
3'>Name:</label>

            <input
                required
                name='name'
                value={name}
                onChange={e => setName(e.target.value)}
                className='w-full bg-gradient-to-r from-zinc-800 to-
zinc-700 p-3 rounded-2xl hover:border-2 placeholder:text-black/50 placeholder:text-lg text-[2vh]'
                placeholder='Enter your name...'
            />

            <label className='text-[2vh] uppercase font-bold px-
3'>Surname:</label>

            <input
                required
                name='surname'
                className='w-full bg-gradient-to-r from-zinc-800 to-
zinc-700 p-3 rounded-2xl hover:border-2 placeholder:text-black/50 placeholder:text-lg text-[2vh]'
                value={surname}
                onChange={e => setSurname(e.target.value)}
            />
        </form>
    </motion.div>
)

```

```

        placeholder='Enter your surname...'
      />
      <label      className='text-[2vh]      uppercase      font-bold      px-
3'>Email:</label>

      <input
        required
        name='email'
        value={email}
        onChange={e => setEmail(e.target.value)}
        className={`w-full bg-gradient-to-r from-zinc-800 to-
zinc-700 p-3 rounded-2xl hover:border-2 placeholder:text-black/50 placeholder:text-lg text-[2vh]
${email?.length && !isEmailValid(email) ? 'border-red-700 border-2' : ''}`}
        placeholder='Enter your email...'
      />
      <label      className='text-[2vh]      uppercase      font-bold      px-
3'>Password:</label>

      <input
        name='password'
        value={password}
        onChange={e => setPassword(e.target.value)}
        className={`w-full bg-gradient-to-r from-zinc-800 to-
zinc-700 p-3 rounded-2xl hover:border-2 placeholder:text-white/90 placeholder:text-lg text-[2vh]`}
        placeholder='Enter new password...'
      />
      <Button
        className='uppercase mt-[2vh] w-[50%] mx-auto'
        title='Update'
        type='submit'
      />
    </form>
  </motion.div>
)
}

export default Admin

```

Б.8 Текст модуля router.tsx

```

import AdminLayout from '../layouts/admin/AdminLayout'
import BaristaLayout from '../layouts/barista/BaristaLayout'
import Admin from '../pages/Admin/Admin'
import Baristas from '../pages/Admin/Baristas'
import Categories from '../pages/Admin/Categories'
import Discounts from '../pages/Admin/Discounts'
import Ingredients from '../pages/Admin/Ingredients'
import MenuPositions from '../pages/Admin/MenuPositions'
import Orders from '../pages/Admin/Orders'
import Points from '../pages/Admin/Points'
import RefreshPage from '../pages/Admin/RefreshPage'
import Shifts from '../pages/Admin/Shifts'
import Statistics from '../pages/Admin/Statistics'
import AllBarista from '../pages/Admin/Statistics/AllBarista'
import BaristaSalaryShifts from '../pages/Admin/Statistics/BaristaSalaryShifts'

```

```

import CategoriesIncomes from '../pages/Admin/Statistics/CategoriesIncomes'
import Common from '../pages/Admin/Statistics/Common'
import Consumption from '../pages/Admin/Statistics/Consumption'
import Expenses from '../pages/Admin/Statistics/Expenses'
import Popularity from '../pages/Admin/Statistics/Popularity'
import { allBaristasSalaryLoader } from '../pages/Admin/loaders/allBaristasSalaryLoader'
import { allCategoriesLoader } from '../pages/Admin/loaders/allCategoriesLoader'
import { baristasLoader } from '../pages/Admin/loaders/baristasLoader'
import { categoriesLoader } from '../pages/Admin/loaders/categoriesLoader'
import { commonLoader } from '../pages/Admin/loaders/commonLoader'
import { discountsLoader } from '../pages/Admin/loaders/discountsLoader'
import {
    ingredientsWithPointsLoader
} from
'../pages/Admin/loaders/ingredientsWithPointsLoader'
import { ordersPointsLoader } from '../pages/Admin/loaders/ordersLoader'
import { pointsLoader } from '../pages/Admin/loaders/pointsLoader'
import { pointsShortLoader } from '../pages/Admin/loaders/pointsShortLoader'
import { positionsLoader } from '../pages/Admin/loaders/positionsLoader'
import { shiftsLoader } from '../pages/Admin/loaders/shiftsLoader'
import AuthLayout from '../pages/Auth/AuthLayout'
import ChangePassword from '../pages/Auth/ChangePassword'
import ConfirmingEmail from '../pages/Auth/ConfirmingEmail'
import Login from '../pages/Auth/Login'
import Registration from '../pages/Auth/Registration'
import RestorePassword from '../pages/Auth/RestorePassword'
import SelectPoint from '../pages/Auth/SelectPoint'
import AboutMe from '../pages/Baritsta/AboutMe'
import CreateOrder from '../pages/Baritsta/CreateOrder'
import CurrentPointInfo from '../pages/Baritsta/CurrentPointInfo'
import IngredientsInfo from '../pages/Baritsta/IngredientsInfo'
import OrdersInfo from '../pages/Baritsta/OrdersInfo'
import Recipes from '../pages/Baritsta/Recipes'
import { aboutMeLoader } from '../pages/Baritsta/loaders/aboutMeLoader'
import { currentPointInfoLoader } from '../pages/Baritsta/loaders/currentPointInfoLoader'
import { ingredientsLoader } from '../pages/Baritsta/loaders/ingredientsLoader'
import { menuLoader } from '../pages/Baritsta/loaders/menuLoader'
import { ordersLoader } from '../pages/Baritsta/loaders/ordersLoader'
import ErrorPage from '../pages/Error/ErrorPage'
import MainPage from '../pages/MainPage/MainPage'

export const router = createBrowserRouter([
    {
        path: 'auth',
        element: <AuthLayout />,
        errorElement: <ErrorPage />,
        children: [
            {
                path: 'login',
                element: <Login />,
            },
            {
                path: 'registration',
                element: <Registration />,
            },
        ]
    }
])

```

```

        path: 'confirming/:token',
        element: <ConfirmingEmail />,
    },
    {
        path: 'change/:token',
        element: <ChangePassword />,
    },
    {
        path: 'restore',
        element: <RestorePassword />,
    },
    {
        path: 'select',
        element: <SelectPoint />,
    },
],
},
{
    path: 'barista',
    element: <BaristaLayout />,
    errorElement: <ErrorPage />,
    children: [
        {
            index: true,
            element: <AboutMe />,
            loader: aboutMeLoader,
        },
        {
            path: 'ingredients',
            element: <IngredientsInfo />,
            loader: ingredientsLoader,
        },
        {
            path: 'createOrder',
            element: <CreateOrder />,
            loader: menuLoader,
        },
        {
            path: 'orders',
            element: <OrdersInfo />,
            loader: ordersLoader,
        },
        {
            path: 'point',
            element: <CurrentPointInfo />,
            loader: currentPointInfoLoader,
        },
        {
            path: 'recipes',
            element: <Recipes />,
            loader: menuLoader,
        },
        {
            path: 'refresh',

```

```

        element: <RefreshPage />,
      },
    ],
  },
  {
    path: 'admin',
    element: <AdminLayout />,
    errorElement: <ErrorPage />,
    children: [
      {
        index: true,
        element: <Admin />,
      },
      {
        path: 'baristas',
        element: <Baristas />,
        loader: baristasLoader,
      },
      {
        path: 'categories',
        element: <Categories />,
        loader: categoriesLoader,
      },
      {
        path: 'discounts',
        element: <Discounts />,
        loader: discountsLoader,
      },
      {
        path: 'ingredients',
        element: <Ingredients />,
        loader: ingredientsWithPointsLoader,
      },
      {
        path: 'positions',
        element: <MenuPositions />,
        loader: positionsLoader,
      },
      {
        path: 'orders',
        element: <Orders />,
        loader: ordersPointsLoader,
      },
      {
        path: 'points',
        element: <Points />,
        loader: pointsLoader,
      },
      {
        path: 'shifts',
        element: <Shifts />,
        loader: shiftsLoader,
      },
    ],
  },

```

```

        path: 'statistics',
        element: <Statistics />,
        children: [
            {
                path: 'popularity',
                element: <Popularity />,
                loader: pointsShortLoader,
            },
            {
                path: 'expenses',
                element: <Expenses />,
                loader: pointsShortLoader,
            },
            { path: 'common', element: <Common />, loader:
commonLoader },
            {
                path: 'barista',
                element: <BaristaSalaryShifts />,
                loader: shiftsLoader,
            },
            {
                path: 'allBarista',
                element: <AllBarista />,
                loader: allBaristasSalaryLoader,
            },
            {
                path: 'consumption',
                element: <Consumption />,
                loader: pointsShortLoader,
            },
            {
                path: 'categories',
                element: <CategoriesIncomes />,
                loader: allCategoriesLoader,
            },
        ],
    },
    {
        path: 'refresh',
        element: <RefreshPage />,
    },
],
},
{
    index: true,
    element: <MainPage />,
    errorElement: <ErrorPage />,
},
])

```

Б.9 Текст модуля MenuPositionService.ts

```
import { getTokenFromLocalStorage } from '../../helper/localstorage.helper'
import { ICategoryData } from '../../types/ICategory'
import { ICreateDiscountData } from '../../types/ICreateDiscountData'
import { ICreateMenuPosition } from '../../types/ICreateMenuPosition'
import { IMenuPositionWithRecipeData } from '../../types/IMenuPositionWithRecipe'
import { IPointAllData } from '../../types/IPointAllData'
import {
  IUpdateRecipeIngredients,
  IUpdateRecipeSteps,
} from '../../types/IUpdateRecipe'

export const MenuPositionsService = {
  async getPointsWithPositions() {
    const pointsWithPositionsData = await axios.get<IPointAllData[]>(
      `http://localhost:3000/api/admin/points`,
      {
        headers: {
          Authorization: 'Bearer ' + getTokenFromLocalStorage(),
        },
      },
    )
    return pointsWithPositionsData.data
  },
  async getCategories() {
    const categories = await axios.get<ICategoryData[]>(
      `http://localhost:3000/api/admin/categories`,
      {
        headers: {
          Authorization: 'Bearer ' + getTokenFromLocalStorage(),
        },
      },
    )
    return categories.data
  },
  async getMenuPositionInfoByID(pointID: number, positionID: number) {
    const pointsWithPositionsData =
      await axios.get<IMenuPositionWithRecipeData>(
        `http://localhost:3000/api/admin/positions/${pointID}/info/${positionID}`,
        {
          headers: {
            Authorization: 'Bearer ' +
              getTokenFromLocalStorage(),
          },
        },
      )
    return pointsWithPositionsData.data
  },
  async deleteMenuPositionByID(pointID: number, positionID: number) {
    await axios.delete(
```

```

`http://localhost:3000/api/admin/positions/${pointID}/delete/${positionID}`,
    {
        headers: {
            Authorization: 'Bearer ' + getTokenFromLocalStorage(),
        },
    }
)
},
async createMenuPosition(pointID: number, createData: ICreateMenuPosition) {
    const pos = await axios.post<IMenuPositionWithRecipeData>(
        `http://localhost:3000/api/admin/positions/${pointID}`,
        createData,
        {
            headers: {
                Authorization: 'Bearer ' + getTokenFromLocalStorage(),
            },
        }
    )
    return pos.data
},
async updateMenuPosition(
    pointID: number,
    positionID: number,
    updateData: ICreateMenuPosition
) {
    await axios.patch(

`http://localhost:3000/api/admin/positions/${pointID}/update/${positionID}`,
    updateData,
    {
        headers: {
            Authorization: 'Bearer ' + getTokenFromLocalStorage(),
        },
    }
)
},
async updateMenuPositionSteps(
    pointID: number,
    positionID: number,
    updateData: IUpdateRecipeSteps
) {
    await axios.patch(

`http://localhost:3000/api/admin/positions/${pointID}/update/${positionID}`,
    updateData,
    {
        headers: {
            Authorization: 'Bearer ' + getTokenFromLocalStorage(),
        },
    }
)
},
async updateMenuPositionIngredients(

```

```

        pointID: number,
        positionID: number,
        updateData: IUpdateRecipeIngredients
    ) {
        await axios.patch(
            `http://localhost:3000/api/admin/positions/${pointID}/update/${positionID}`,
            updateData,
            {
                headers: {
                    Authorization: 'Bearer ' + getTokenFromLocalStorage(),
                },
            }
        )
    },
    async createDiscountForPosition(
        positionID: number,
        discountData: ICreateDiscountData
    ) {
        await axios.post(
            `http://localhost:3000/api/admin/discounts/${positionID}`,
            discountData,
            {
                headers: {
                    Authorization: 'Bearer ' + getTokenFromLocalStorage(),
                },
            }
        )
    },
}

```

Б.10 Текст модуля CreateBaristaModal.tsx

```

import { isValidEmail } from '../../../helper/validate-email.helper'
import { BaristasManagementService } from '../../../services/Admin/BaristasManagementService'
import { ICreateBaristasData } from '../../../types/ICreateBaristaData'

interface Props {
    setVisibleModal: (visible: boolean) => void
}

const CreateBaristaModal: FC<Props> = ({ setVisibleModal }) => {
    const [name, setName] = useState('')
    const [surname, setSurname] = useState('')
    const [email, setEmail] = useState('')
    const [phoneNumber, setPhoneNumber] = useState('')
    const [password, setPassword] = useState('')
    const [fixedHourRate, setFixedHourRate] = useState('')
    const [percentFromEarnings, setPercentFromEarnings] = useState('')
    const navigate = useNavigate()
    const submitHandler = async (e: React.FormEvent<HTMLFormElement>) => {
        try {

```

```

e.preventDefault()
if (!isEmailValid(email)) return
if (
    +fixedHourRate < 0 ||
    +percentFromEarnings < 0 ||
    +percentFromEarnings > 100
)
    return
const newBarista: ICreateBaristasData = {
    name,
    surname,
    email,
    phoneNumber,
    password,
    fixedHourRate: +fixedHourRate,
    percentFromEarnings: +percentFromEarnings,
}
const result = await
BaristasManagementService.createBarista(newBarista)
if (!result) return
toast.success('Barista was successfully created')
setVisibleModal(false)

navigate(`/admin/refresh`)
navigate(`/admin/baristas?id=${result.id}`)
} catch (err: any) {
    const error = err.response?.data.message
    toast.error(error.toString())
}
}
return (
    <motion.div
        initial={{ opacity: 0, y: -20 }}
        animate={{ opacity: 1, y: 0 }}
        exit={{ opacity: 0, y: -20 }}
        transition={{ duration: 0.7 }}
        className='fixed w-full h-full bg-black/50 flex justify-center items-
center z-10'
    >
        <form
            className='w-[20%] h-[77vh] bg-zinc-400 rounded-2xl flex flex-
col justify-between font-roboto gap-[1vh]'
            onSubmit={submitHandler}
        >
            <h2 className='font-bold uppercase w-full text-center text-[2vh]
border-b-2 p-3 bg-zinc-600 text-white rounded-t-2xl'>
                ADDING NEW BARISTA
            </h2>
            <div className='w-[70%] mx-auto'>
                <label htmlFor='name'>
                    <small className='text-[1.6vh] pl-
2'>Name:</small>

                    <input
                        required

```

```

        type='text'
        name='name'
        value={name}
        onChange={e => setName(e.target.value)}
        placeholder='Name...'
        className='text-[2vh] w-full p-2 rounded-
xl'

        />
    </label>
</div>
<div className='w-[70%] mx-auto'>
    <label htmlFor='surname'>
        <small                className='text-[1.6vh]                pl-
2'>Surname:</small>

        <input
            required
            type='text'
            name='surname'
            value={surname}
            onChange={e                                =>
                setSurname(e.target.value)}

            placeholder='Surname...'
            className='text-[2vh] w-full p-2 rounded-
xl'

            />
        </label>
    </div>
<div className='w-[70%] mx-auto'>
    <label htmlFor='email'>
        <small                className='text-[1.6vh]                pl-
2'>Email:</small>

        <input
            required
            type='text'
            name='email'
            value={email}
            onChange={e => setEmail(e.target.value)}
            placeholder='Email...'
            className={`text-[2vh] ${email.length &&
!isEmailValid(email) ? 'border-red-700 border-2' : ''} w-full p-2 rounded-xl`}
            />
        </label>
    </div>
<div className='w-[70%] mx-auto'>
    <label htmlFor='phoneNumber'>
        <small                className='text-[1.6vh]                pl-2'>Phone
number:</small>

        <input
            required
            type='text'
            value={phoneNumber}
            onChange={e                                =>
                setPhoneNumber(e.target.value)}

            name='phoneNumber'

```

```

placeholder='Phone number...'
className='text-[2vh] w-full p-2 rounded-

xl'

        />
    </label>
</div>
<div className='w-[70%] mx-auto'>
    <label htmlFor='password'>
        <small          className='text-[1.6vh]          pl-
2'>Password:</small>

        <input
            required
            type='password'
            name='password'
            value={password}
            onChange={e                =>

                placeholder='Password...'
                className='text-[2vh] w-full p-2 rounded-

xl'

                />
            </label>
        </div>
        <div className='w-[70%] mx-auto'>
            <label htmlFor='fixedHourRate'>
                <small className='text-[1.6vh] pl-2'>Fixed hour
rate:</small>

                <input
                    required
                    type='number'
                    name='fixedHourRate'
                    value={fixedHourRate}
                    onChange={e                =>

                        placeholder='Fixed hour rate...'
                        className='text-[2vh] w-full p-2 rounded-

xl'

                        />
                    </label>
                </div>
                <div className='w-[70%] mx-auto'>
                    <label htmlFor='percentFromEarnings'>
                        <small className='text-[1.6vh] pl-2'>
                            Percent from earnings(0-100%):
                        </small>
                        <input
                            required
                            type='number'
                            value={percentFromEarnings}
                            onChange={e                =>

                                name='percentFromEarnings'
                                placeholder='Percent from earnings...'

```

```

        className='text-[2vh] w-full p-2 rounded-
xl'
        />
      </label>
    </div>
    <div className='flex items-center justify-end gap-[1vh] w-full
p-[1vh] '>
      <button
        className={`bg-zinc-700 text-white px-5 py-2
rounded-2xl text-[2vh] hover:bg-zinc-300 hover:text-black uppercase disabled:cursor-not-allowed
disabled:hover:bg-zinc-700 disabled:hover:text-white`}
        type='submit'
      >
        Add
      </button>
      <button
        type='button'
        onClick={() => setVisibleModal(false)}
        className='bg-zinc-700 text-white px-5 py-2
rounded-2xl text-[2vh] hover:bg-zinc-300 hover:text-black uppercase'
      >
        Close
      </button>
    </div>
  </form>
</motion.div>
)
}

export default CreateBaristaModal

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів
Дипломна кваліфікаційна робота бакалавра

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБЛІКУ ТА КЕРУВАННЯ ЗАМОВЛЕННЯМИ У МЕРЕЖІ КАВ'ЯРЕНЬ

Виконав
ст.гр. КНТ-130

Максим ЖМУЦЬКИЙ

Керівник
к.т.н., доцент

Тетяна ФЕДОРОНЧАК

Рисунок В.1 – Слайд № 1

МЕТА, ОБ'ЄКТ, ПРЕДМЕТ І ЗАДАЧІ ДОСЛІДЖЕННЯ

- Мета роботи – розробка програмного забезпечення, яке базується на клієнт-серверній архітектурі та дозволяє здійснювати операції пов'язані з обліком та керуванням замовленнями у мережі кав'ярень.
- Об'єкт дослідження – керування мережею кав'ярень.
- Предмет дослідження – програмне забезпечення для обліку та керування замовленнями у мережі кав'ярень.
- Для досягнення поставленої мети буде вирішено наступні завдання:
 - ✓ огляд стану ринку кав'ярень та існуючих методів управління бізнесом у цій галузі;
 - ✓ дослідження особливостей роботи мережі кав'ярень та визначення ключових бізнес-процесів, які потрібно оптимізувати;
 - ✓ розробка програмного забезпечення для автоматизації управління кав'ярнями з урахуванням потреб оптимізації бізнес-процесів та підвищення ефективності управління;
 - ✓ здійснення експериментальних досліджень та тестувань розробленого програмного забезпечення для підтвердження його ефективності та придатності для використання в практичній діяльності.

2

Рисунок В.2 – Слайд № 2

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Розглянуто основні бізнес-процеси пов'язані з функціонуванням мереж кав'ярень

Виділено ряд функціональних вимог до систем, які спеціалізуються виключно на кав'ярнях

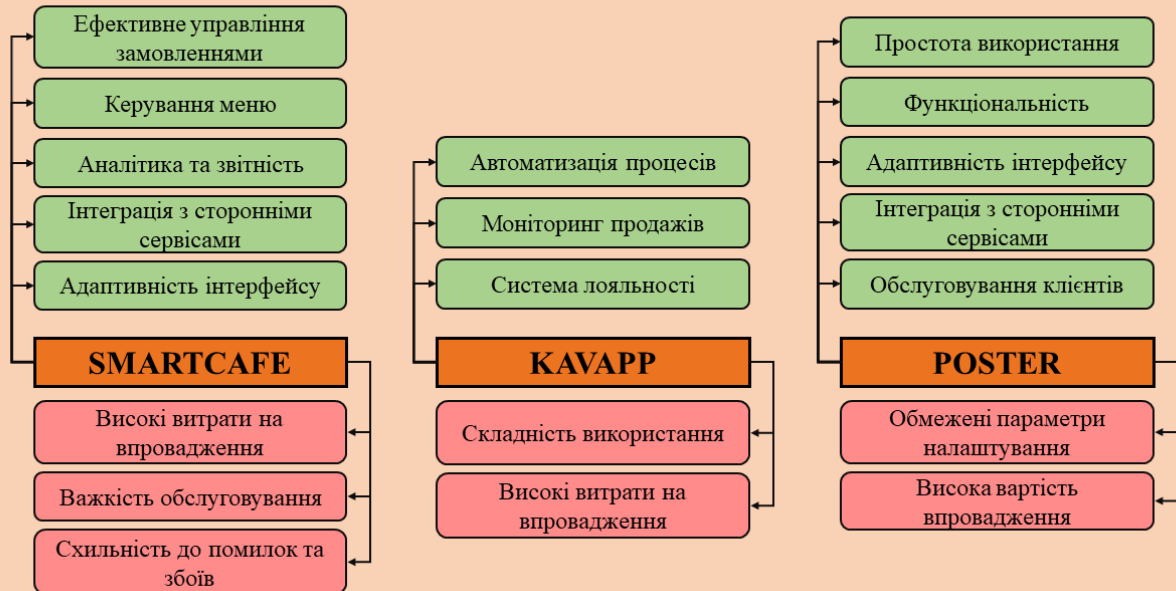
Окреслено розподіл відповідальності у системі

Досліджено питання безпечного функціонування системи

3

Рисунок В.3 – Слайд № 3

ОГЛЯД АНАЛОГІВ



4

Рисунок В.4 – Слайд № 4

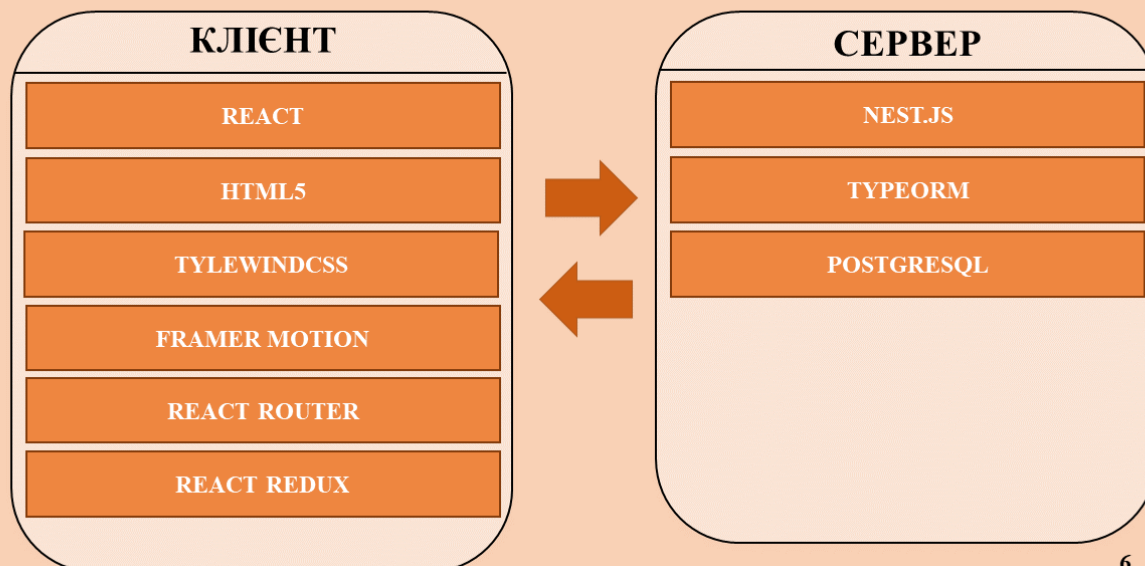
ВИБІР МОВИ ПРОГРАМУВАННЯ

№	Критерій за яким оцінюється	C#	JS{TS}	Python	PHP
1	Складність розробки	Середня	Низька	Низька	Висока
2	Швидкодія	Висока	Висока	Низька	Середня
3	Екосистема	Велика	Велика	Велика	Велика
4	Масштабованість	Добра	Добра	Добра	Погана
5	Асинхронність	Підтримує	Підтримує	Не підтримує	Не підтримує
6	Строга типізація	Є	Є	Відсутня	Відсутня
7	Спільнота та підтримка	Середня	Велика	Велика	Велика

5

Рисунок В.5 – Слайд № 5

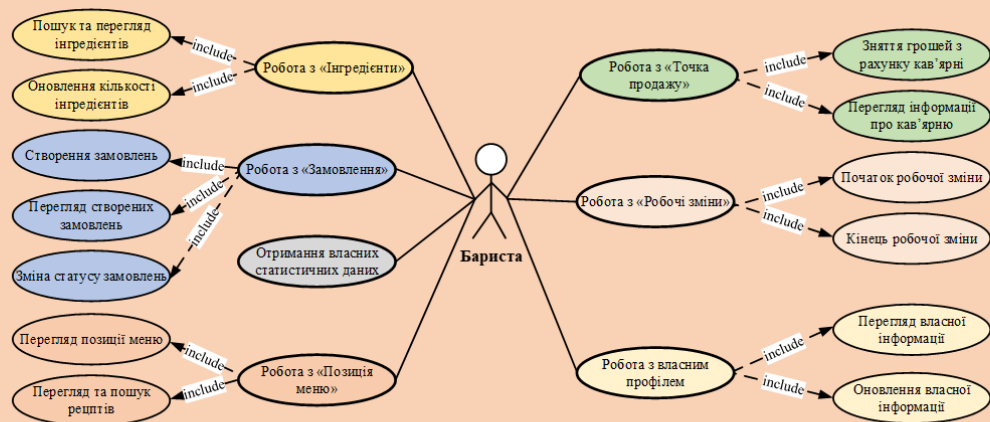
ВИБІР ІНСТРУМЕНТАРІЮ



6

Рисунок В.6 – Слайд № 6

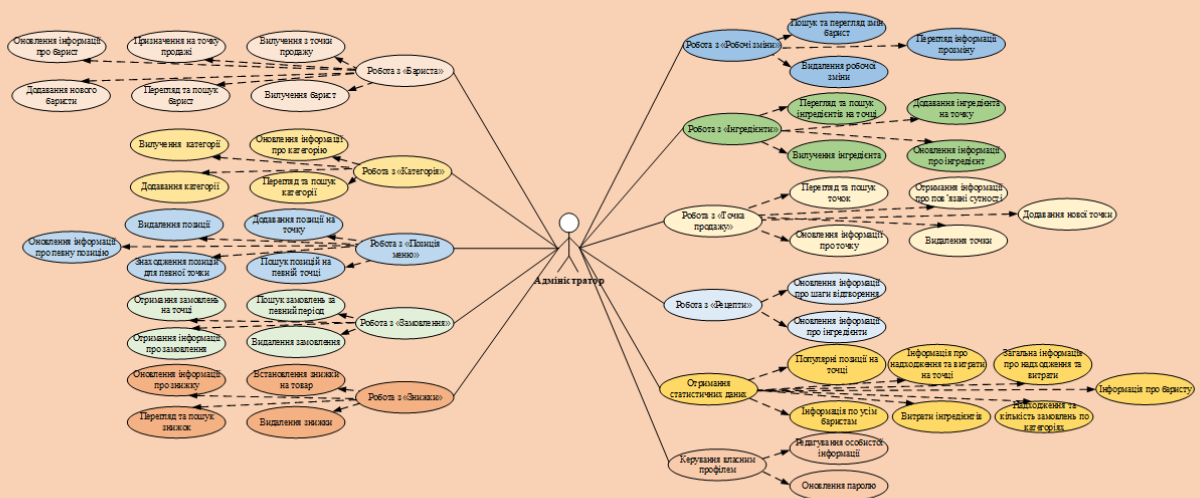
ФУНКЦІОНАЛЬНІ ВИМОГИ ДО РОЛІ «БАРИСТА»



7

Рисунок В.7 – Слайд № 7

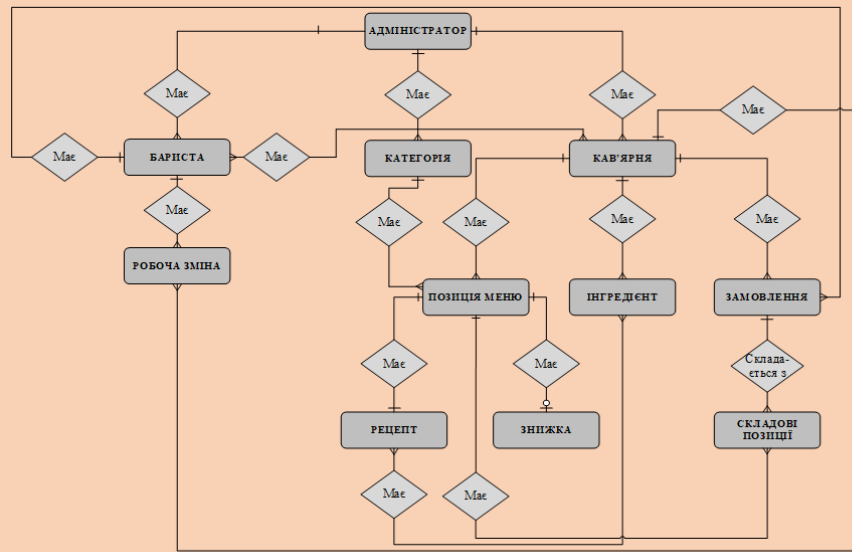
ФУНКЦІОНАЛЬНІ ВИМОГИ ДО РОЛІ «АДМІНІСТРАТОР»



8

Рисунок В.8 – Слайд № 8

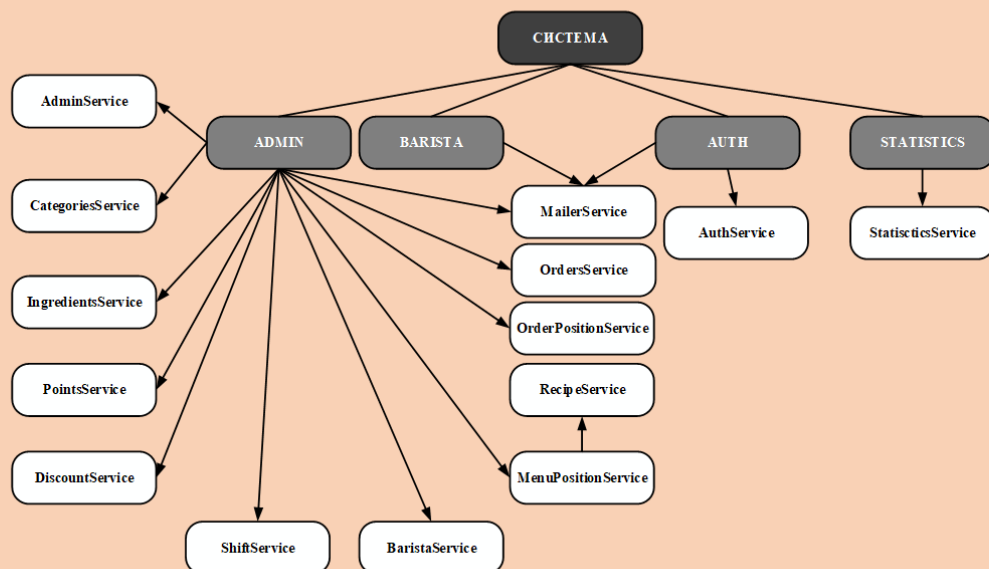
ВІДНОШЕННЯ МІЖ СУТНОСТЯМИ НАЯВНИМИ У СИСТЕМІ



9

Рисунок В.9 – Слайд № 9

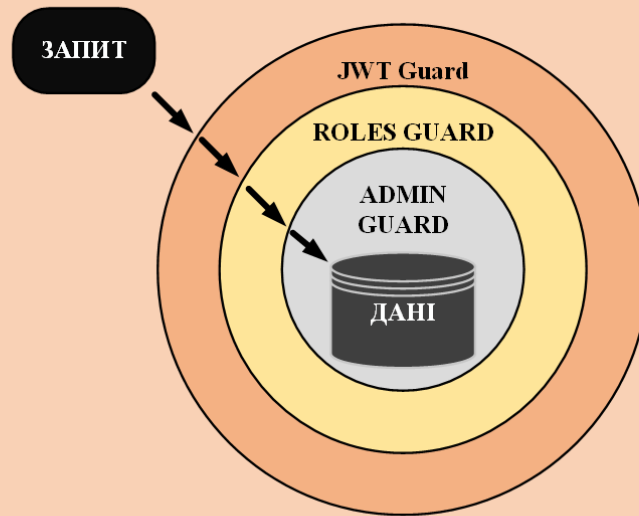
СХЕМА МОДУЛІВ СЕРВЕРУ



10

Рисунок В.10 – Слайд № 10

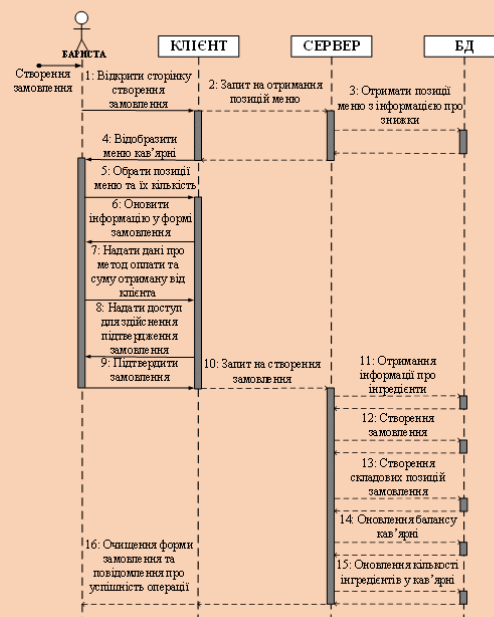
СИСТЕМА ЗАХИСТУ МАРШРУТІВ СЕРВЕРУ



11

Рисунок В.11 – Слайд № 11

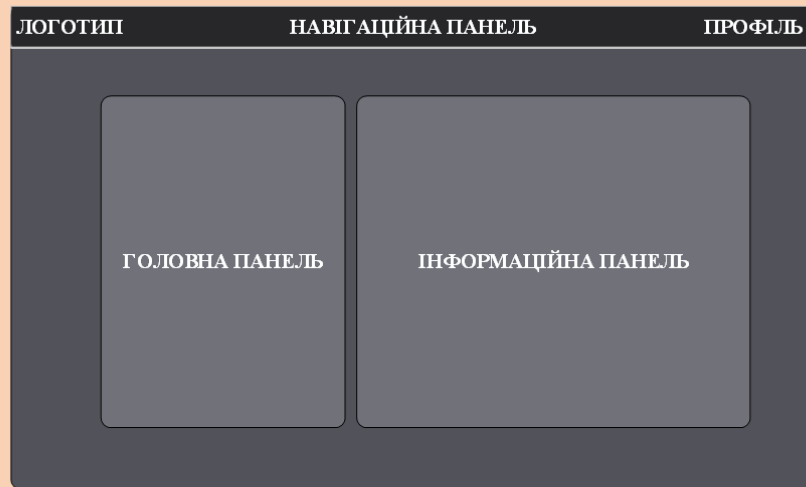
ФУНКЦІОНУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ



12

Рисунок В.12 – Слайд № 12

ШАБЛОН ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ



13

Рисунок В.13 – Слайд № 13

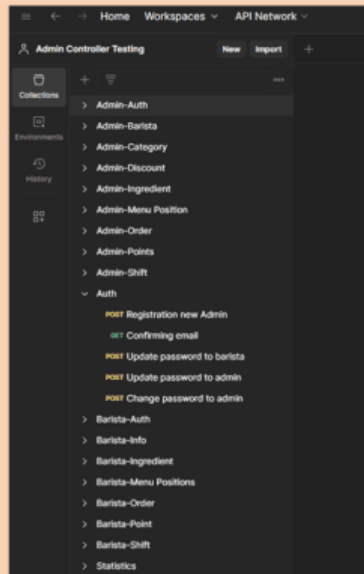
ІНТЕРФЕЙС ЗАСТОСУНКУ



14

Рисунок В.14 – Слайд № 14

ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ



№	Назва набору тестів	Роль	Результат
1	Аутентифікація	Адміністратор	Пройдено вдало
2	Керування баристами	Адміністратор	Пройдено вдало
3	Керування категоріями	Адміністратор	Пройдено вдало
4	Керування знижками	Адміністратор	Пройдено вдало
5	Керування інгредієнтами	Адміністратор	Пройдено вдало
6	Керування позиціями меню	Адміністратор	Пройдено вдало
7	Керування замовленнями	Адміністратор	Пройдено вдало
8	Керування кав'ярнями	Адміністратор	Пройдено вдало
9	Керування робочими змінами	Адміністратор	Пройдено вдало
10	Регістрація та відновлення даних	Адміністратор, Бариста	Пройдено вдало
11	Аутентифікація	Бариста	Пройдено вдало
12	Отримання інформації про себе	Бариста	Пройдено вдало
13	Робота з інгредієнтами	Бариста	Пройдено вдало
14	Робота з позиціями меню	Бариста	Пройдено вдало
15	Робота з замовленнями	Бариста	Пройдено вдало
16	Робота з кав'ярнею	Бариста	Пройдено вдало
17	Робота з власними робочими змінами	Бариста	Пройдено вдало
18	Отримання статистичних показників	Адміністратор	Пройдено вдало

15

Рисунок В.15 – Слайд № 15

ВИСНОВКИ

Таким чином, у результаті виконання бакалаврської дипломної кваліфікаційної роботи було досягнуто основної мети – розроблено програмний продукт, який базується на клієнт-серверній архітектурі та дозволяє здійснювати операції пов'язані з обліком та керуванням мережею кав'ярень.

Для досягнення цієї мети було вирішено ряд завдань:

- проаналізовано особливості функціонування систем управління мережами кав'ярень, розподіл відповідальності та ідентифікації користувачів у системі;
- оцінено існуючі програмні рішення для подібних задач;
- розроблено програмне забезпечення для автоматизації управління кав'ярнями з урахуванням потреб оптимізації бізнес-процесів та підвищення ефективності управління;
- здійснено експериментальні дослідження та тестування розробленого програмного забезпечення для підтвердження його ефективності та придатності для використання в практичній діяльності.

Розроблений програмний продукт дозволяє ефективно керувати різними аспектами діяльності мереж кав'ярень та оптимізувати робочі процеси у закладах, що сприяє покращенню рівня обслуговування та підвищенню якості послуг, а також збільшенню продуктивності та зниженню витрат.

16

Рисунок В.16 – Слайд № 16