Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка CRM-системи для ресторанного бізнесу на основі патерну SDUI-MVVM»

Виконав: студент групи К23-2М

Спеціальність 122 Комп'ютерні науки

Лєсніков М.О.
(прізвище та ініціали)

Керівник д.т.н., проф. Яковенко В.О.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Завідувач кафедри кібербезпеки та інфомармаційних технологій
(посада)

к.т.н., доц.Прокопович-Ткаченко Д.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Лєсніков М.О. Розробка CRM-системи для ресторанного бізнесу на основі патерну SDUI-MVVM

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єкт дослідження –процес автоматизації управління бізнес-процесами у ресторанному закладі.

Предмет дослідження – методи та алгоритми розробки CRM-систем для ресторанного бізнесу.

Метою роботи є розробка CRM-системи для ресторанного бізнесу на основі архітектурного патерну SDUI-MVVM, що дозволяє централізовано управляти інтерфейсом користувача через сервер, підвищуючи гнучкість та знижуючи витрати на обслуговування системи.

Дипломна робота присвячена розробці та впровадженню CRM-системи, яка автоматизує управління замовленнями, клієнтськими базами та покращує обслуговування клієнтів у ресторанному бізнесі. У роботі проведено аналіз існуючих CRM-рішень, розглянуто ключові методи проектування та реалізації програмного забезпечення. Основний акцент зроблено на використанні архітектурного патерну SDUI-MVVM, який забезпечує централізоване керування інтерфейсом користувача з використанням Angular.js для клієнтської частини та Node.js для серверної.

Практична цінність роботи полягає у створенні функціональної CRM-системи, яка може бути використана у реальних умовах для автоматизації управління замовленнями, підвищення рівня обслуговування клієнтів та ефективності бізнесу.

Ключові слова: CRM-система, SDUI-MVVM, Angular.js, Node.js.

ABSTRACT

Lesnikov M.O. Development of a CRM System for the Restaurant Business Based on the SDUI-MVVM Pattern

Diploma thesis (project) for obtaining a master's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2025.Object of the research: the process of automating business process management in a restaurant establishment.

Subject of the research: methods and algorithms for developing CRM systems for the restaurant business.

The aim of the work is to develop a CRM system for the restaurant business based on the SDUI-MVVM (Server-Driven UI with Model-View-ViewModel) architectural pattern, which enables centralized user interface management through the server, increasing flexibility and reducing system maintenance costs.

The thesis focuses on the development and implementation of a CRM system that automates order management, customer databases, and improves customer service in the restaurant business. The study includes an analysis of existing CRM solutions and a review of key methods for software design and implementation. The primary focus is on the SDUI-MVVM pattern, which ensures centralized control of the user interface using Angular.js for the client-side and Node.js for the server-side development.

Practical significance: The developed CRM system can be used in real-world conditions for order management automation, improving customer service levels, and increasing business efficiency.

Keywords: CRM system, SDUI-MVVM, Angular.js, Node.js.

ЗМІСТ

ВСТУП

У сучасних умовах автоматизації бізнес-процесів та підвищення вимог до ефективності управління клієнтськими взаємовідносинами, впровадження CRM-систем у ресторанному бізнесі стає критично важливим. CRM-системи дозволяють оптимізувати процеси обслуговування клієнтів, автоматизувати управління замовленнями, підвищити рівень персоналізації сервісу, а також ефективно управляти базами даних клієнтів. Впровадження таких систем сприяє покращенню обслуговування, підвищенню рівня задоволеності клієнтів та зростанню прибутковості бізнесу.

Інноваційність даної роботи полягає у використанні архітектурного патерну SDUI-MVVM (Server-Driven UI з Model-View-ViewModel) для побудови CRM-системи. Цей підхід передбачає централізоване управління інтерфейсом користувача через сервер, що забезпечує гнучкість у оновленні інтерфейсу без потреби внесення змін до клієнтської частини. Це сприяє спрощенню підтримки системи та зменшенню витрат на її обслуговування.

Мета роботи є – **р**озробка CRM-системи для ресторанного бізнесу на основі архітектурного патерну SDUI-MVVM.

Методи дослідження – методи проектування, розробки програмного забезпечення, метод теорії інформації, обробка та аналіз інформації.

Для досягнення поставленої мети застосовувалися методи:

1. аналіз предметної області та наукових джерел щодо CRM-систем;
2. методи та засоби програмної інженерії для розробки архітектури системи;
3. Розробити вимоги до програмного забезпечення
4. Реалізація CRM-систем;
5. Провести тесутвання.

Об'єкт дослідження – процес автоматизації управління бізнес-процесами у ресторанному закладі.

Предмет дослідження методи та алгоритми розробки CRM-систем для ресторанного бізнесу.

Практична значимість даної роботи полягає у розробці CRM-системи, яка може бути використана у ресторанному бізнесі для підвищення ефективності управління замовленнями, зменшення часу на обробку клієнтських запитів, покращення персоналізації сервісу та підвищення загальної продуктивності закладу.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 60 сторінки основного тексту, 29 рисунків та 3 таблиці.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз публікацій щодо розробки CRM-систем

CRM-система — це сучасний інструмент управління відносинами з клієнтами, що поєднує програмне забезпечення, технології та методики для ефективного обліку, обробки та зберігання даних про взаємодію з клієнтами. Завдяки їй компанії можуть автоматизувати ключові бізнес-процеси, зокрема продажі, маркетинг, обслуговування та підтримку клієнтів, що сприяє підвищенню рівня сервісу та задоволеності клієнтів [2].

CRM-системи виступають як єдиний центр координації взаємодії різних підрозділів компанії, які працюють з клієнтами, а також об'єднують усі канали комунікації, забезпечуючи прозорість обміну інформацією. Важливою перевагою є можливість надання доступу до повної інформації про клієнта всім залученим співробітникам, що допомагає персоналізувати обслуговування та краще задовольняти потреби клієнта.

Спочатку CRM-інструменти використовувалися переважно для автоматизації процесів продажів і клієнтського супроводу. Однак сучасні системи еволюціонували у комплексні платформи для управління бізнесом, які дозволяють налаштовувати внутрішні бізнес-процеси під потреби конкретної компанії, незалежно від її розміру чи галузі. Головна цінність сучасної CRM полягає у здатності покращувати внутрішні комунікації та забезпечувати злагоджену роботу всіх структурних підрозділів.

Бізнес-процес у CRM — це логічна послідовність етапів, що ведуть до досягнення конкретного результату. Його ефективність залежить від таких ключових факторів, як: чітке планування, правильна постановка завдань, моніторинг ключових точок, зворотний зв'язок між співробітниками та системне ведення звітності. Основними елементами бізнес-процесу є:

- Стійкі зв’язки між етапами — забезпечення безперервності виконання завдань.
- Визначені дії на кожному етапі — чіткий набір кроків, які виконуються на кожному етапі процесу.
- Логічне завершення — досягнення конкретного результату або цілі.

Важливою особливістю управління бізнес-процесами є чіткий розподіл відповідальності серед учасників процесу. Кожен співробітник має виконувати лише свій етап у встановлений термін, а менеджмент компанії — своєчасно оцінювати результативність та коригувати робочі процеси у разі потреби. Документування кожного етапу процесу допомагає фіксувати всі нюанси роботи, виявляти слабкі місця та покращувати ефективність як окремих етапів, так і всієї системи загалом [5].

Автоматизація бізнес-процесів за допомогою CRM є актуальною як для великих корпорацій так і для малих підприємств, оскільки дозволяє систематизувати рутинні завдання, такі як: погодження документів, оформлення замовлень, управління складськими операціями, обслуговування клієнтів тощо. Це сприяє стабільності процесів та зниженню ризиків, пов'язаних із людським фактором.

Впровадження CRM-системи дозволяє компаніям досягти низки стратегічних переваг:

- Створення єдиної IT-інфраструктури, яка об’єднує всі процеси в компанії.
- Систематизація рутинних завдань із мінімізацією впливу людського фактору.
- Підвищення прозорості бізнесу завдяки доступу до детальних звітів та історії взаємодій.
- Інтеграція клієнтів у бізнес-процеси, включаючи особисті кабінети для самостійного відстеження статусу замовлень.
- Оптимізація ресурсів через автоматизацію ключових завдань.

Таким чином, CRM-системи вже давно вийшли за межі простих інструментів для управління клієнтською базою та перетворилися на потужні платформи для комплексної оптимізації бізнесу, допомагаючи компаніям залишатися конкурентоспроможними та ефективно керувати всіма етапами роботи з клієнтами.

CRM-системи за функціональним призначенням можна розділити на три основні категорії: управління клієнтським обслуговуванням, управління продажами та управління маркетингом. Кожна з них виконує унікальні функції, спрямовані на покращення окремих аспектів бізнес-діяльності та сприяє комплексній оптимізації внутрішніх процесів компанії.

Управління клієнтським обслуговуванням (Customer Service and Support – CSS) спрямоване на забезпечення якісної підтримки клієнтів через створення централізованої бази даних, що містить усі історичні дані про взаємодію з клієнтом. Це дає змогу оперативно виявляти попередні звернення клієнта та розуміти їх контекст, забезпечуючи персоналізований підхід до кожного запиту.

Відмінною рисою CSS-систем є можливість не лише обслуговування віддалених клієнтів, але й підтримка командної співпраці з партнерами та замовниками. Інструменти для спільної роботи дозволяють надавати послуги у дистанційному режимі, що актуально для глобальних компаній із розгалуженою мережею клієнтів.

CRM-системи цієї категорії часто включають функції збереження статистики звернень, створення бази стандартних відповідей на типові запити та автоматизацію обробки повторюваних звернень, що знижує навантаження на персонал. Крім того, CSS допомагає збирати дані про потенційних клієнтів і постійно оновлювати їхню актуальність, що сприяє розширенню клієнтської бази.

Функціонал CSS також дозволяє контролювати роботу сервісних відділів, аналізувати показники обслуговування та підвищувати лояльність

клієнтів завдяки системі пріоритезації запитів. Це забезпечує індивідуальний підхід до клієнтів з урахуванням їхньої цінності для бізнесу.

Управління продажами (Sales Force Automation – SFA) орієнтоване на автоматизацію процесів продажів і управління торговими командами. Його ключова мета — систематизувати та оптимізувати всі етапи продажу, починаючи від першого контакту з клієнтом і закінчуючи укладенням угоди.

SFA-системи забезпечують актуалізацію контактної інформації, контроль історії взаємодій із клієнтом і зберігання даних про всі етапи продажу. Це дозволяє менеджерам швидко отримувати повну картину щодо кожного клієнта та планувати подальші дії.

Особливістю SFA є функція управління діяльністю торгових представників через календарні модулі, які допомагають координувати завдання окремих співробітників і цілих відділів. Це сприяє чіткому плануванню та підвищенню ефективності роботи персоналу.

Цікавою функцією є можливість прогнозування майбутніх продажів на основі зібраних маркетингових даних та аналітики попередніх угод. Це дозволяє виявляти успішні стратегії та коригувати слабкі місця у продажах.

Додатково SFA надає інструменти для аналізу прибутковості по кожному клієнту, автоматизованої генерації комерційних пропозицій та тарифних планів відповідно до оновлених даних клієнтської бази. Це допомагає підвищити ефективність продажів і сприяти зростанню доходів компанії.

Управління маркетингом (Marketing Automation – MA) охоплює планування, реалізацію та аналіз маркетингових кампаній, орієнтованих на залучення нових клієнтів і підтримку постійної взаємодії з існуючими.

MA-системи дозволяють проводити детальний аналіз цільової аудиторії, сегментувати клієнтську базу та виявляти ключові споживчі групи. Це допомагає створювати персоналізовані маркетингові кампанії, орієнтовані на специфічні потреби різних сегментів клієнтів [1].

Важливим компонентом МА є інструменти для автоматизації маркетингових процесів, зокрема створення та управління базами потенційних клієнтів, автоматизація розсилок, аналіз ефективності кампаній та управління взаємодіями з клієнтами на всіх етапах маркетингового циклу.

Система дозволяє розробляти комплексні стратегії з урахуванням індивідуальних переваг клієнтів, що сприяє збільшенню конверсій та формуванню лояльної аудиторії. МА також передбачає створення детальних звітів за результатами кампаній, що допомагає бізнесу аналізувати ефективність маркетингових зусиль і приймати обґрунтовані рішення.

Кожен із типів CRM-систем виконує унікальні функції, спрямовані на покращення різних аспектів бізнесу. В підсумку сформована таблиця 1.1, що демонструє ключові відмінності між трьома функціональними областями CRM-систем та їхню унікальну роль у бізнес-автоматизації.

Таблиця 1.1 – Функціональні області CRM-систем

Функціональна область	Управління клієнтським обслуговуванням (CSS)	Управління продажами (SFA)	Управління маркетингом (МА)
Основне призначення	Обслуговування клієнтів, забезпечення якісної підтримки та персоналізованої взаємодії	Автоматизація процесів продажу, управління командою продавців	Планування, управління та аналіз маркетингових кампаній
Ключові функції	- Централізована база даних клієнтів	- Управління контактами та історією продажів	- Аналіз цільової аудиторії
	- Історія взаємодій	- Контроль за діяльністю співробітників	- Сегментація клієнтів
	- Автоматизація обробки запитів	- Прогнозування продажів	- Автоматизація розсилок
	- Пріоритезація обслуговування	- Автоматизоване створення	- Оцінка ефективності

		комерційних пропозицій	маркетингових кампаній
Основні переваги	- Покращення обслуговування клієнтів	- Оптимізація процесу продажів	- Збільшення залученості клієнтів
	- Зниження часу на обробку звернень	- Підвищення продуктивності співробітників	- Підвищення ефективності маркетингових кампаній
	- Підвищення лояльності клієнтів	- Покращення обліку угод	- Оптимізація ресурсів
Цільова аудиторія	Відділи підтримки, сервісні центри	Відділи продажів, комерційні підрозділи	Маркетингові відділи, стратегічні аналітики
Типові функції автоматизації	- Автоматизація відповіді на типові запити	- Автоматизація планування зустрічей	- Планування маркетингових кампаній
	- Контроль якості обслуговування	- Генерація прогнозів продажів	- Генерація звітів про ефективність
	- Управління партнерськими комунікаціями	- Ведення бази клієнтів	- Управління цільовими розсилками
Взаємозв'язок	Збирає дані для аналітики маркетингових кампаній та продажів	Працює з даними CSS для управління повторними продажами	Впливає на ефективність продажів через правильне позиціонування продукту
Результати впровадження	- Покращення рівня обслуговування	- Підвищення показників закриття угод	- Ефективне планування рекламних кампаній
	- Прискорення реакції на запити клієнтів	- Оптимізація роботи відділів продажу	- Підвищення конверсійності маркетингових заходів

1.2 Аналіз існуючих систем

Для визначення вимог, функціональних можливостей та логіки створюваної CRM-системи було проведено аналіз вже існуючих рішень у цій сфері. У процесі дослідження було надано короткий опис предметної області, здійснено огляд популярних CRM-систем та проведено аналіз технологій, які можуть бути використані під час розробки власної системи. Дані були отримані шляхом пошуку аналогічних рішень у мережі Інтернет та вивчення їх функціональності.

Аналіз існуючих CRM-систем показав значну різноманітність представлених рішень, кожне з яких має як переваги, так і недоліки. Водночас більшість систем демонструють схожий підхід до інтерфейсу та набору функцій, що зумовлено їх основним призначенням — управлінням клієнтськими даними. Тому під час розробки особливий акцент буде зроблено на зручність користування системою та якість візуального оформлення інтерфейсу [2].

В рамках проведеної роботи було здійснено огляд та порівняння ключових CRM-систем з метою визначення кращих практик та оптимальних рішень для подальшого використання у власній розробці.

«Sales Creatio»

«Sales Creatio» — це комплексне рішення, призначене для автоматизації та прискорення повного циклу продажів: від обробки лідів до управління повторними замовленнями. Крім основного функціоналу, платформа пропонує конфігурації для управління маркетингом, сервісом та бізнес-процесами, що робить її універсальним інструментом для різних бізнес-задач (рис. 1.1).

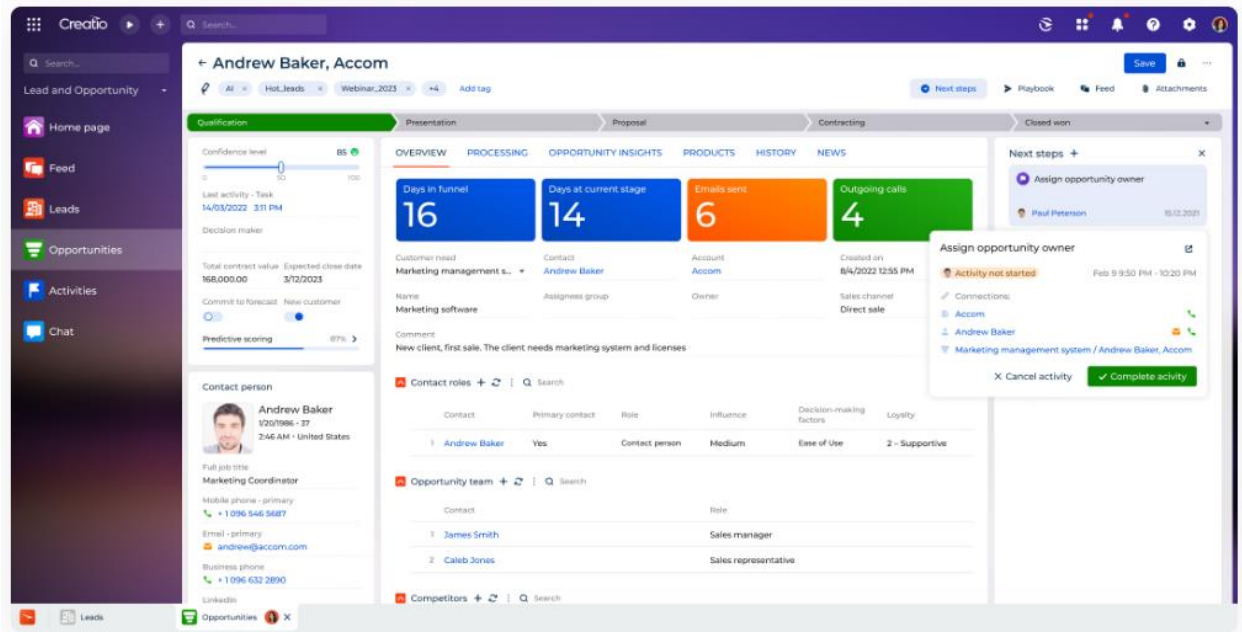

Рисунок 1.1 – Sales Creatio

На онлайн-майданчику Marketplace доступні готові модулі, галузеві рішення, конектори та шаблони, які допомагають розширити функціональність системи та автоматизувати різноманітні бізнес-процеси. Особливу увагу платформа приділяє інтеграції інтелектуальних, матеріальних та фізичних ресурсів для підвищення ефективності бізнесу та покращення впливу на навколишнє середовище.

Важливою перевагою «Sales Creatio» є можливість створення демо-версії для тестування системи, навіть незважаючи на те, що основний продукт є платним. При цьому демо-версія автоматично наповнюється демонстраційними даними, що дозволяє користувачам ознайомитися з основними функціями системи у реальному використанні [8].

Серед ключових функціональних можливостей «Sales Creatio» варто відзначити:

- Управління замовленнями та рахунками.
- Управління процесом продажів.
- Управління продуктовим портфелем.
- Управління документами.

- Планування, розклад та комунікації.
- Аналіз та планування продажів.
- Інструменти для налаштування системи.

Ці функції спрямовані на оптимізацію та прискорення процесів продажів, маркетингу, сервісу та операційних процесів, що робить платформу ефективним рішенням для компаній різного масштабу.

KeepinCRM

«KeepinCRM» — це хмарна CRM-система, спеціально розроблена для потреб малого та середнього бізнесу, яка забезпечує всебічний контроль за взаємодією з клієнтами, управління угодами, фінансами, складом та документацією. Платформа орієнтована на спрощення бізнес-процесів завдяки своєму зручному, інтуїтивно зрозумілому інтерфейсу та широкому набору функціональних можливостей (рис. 1.2).

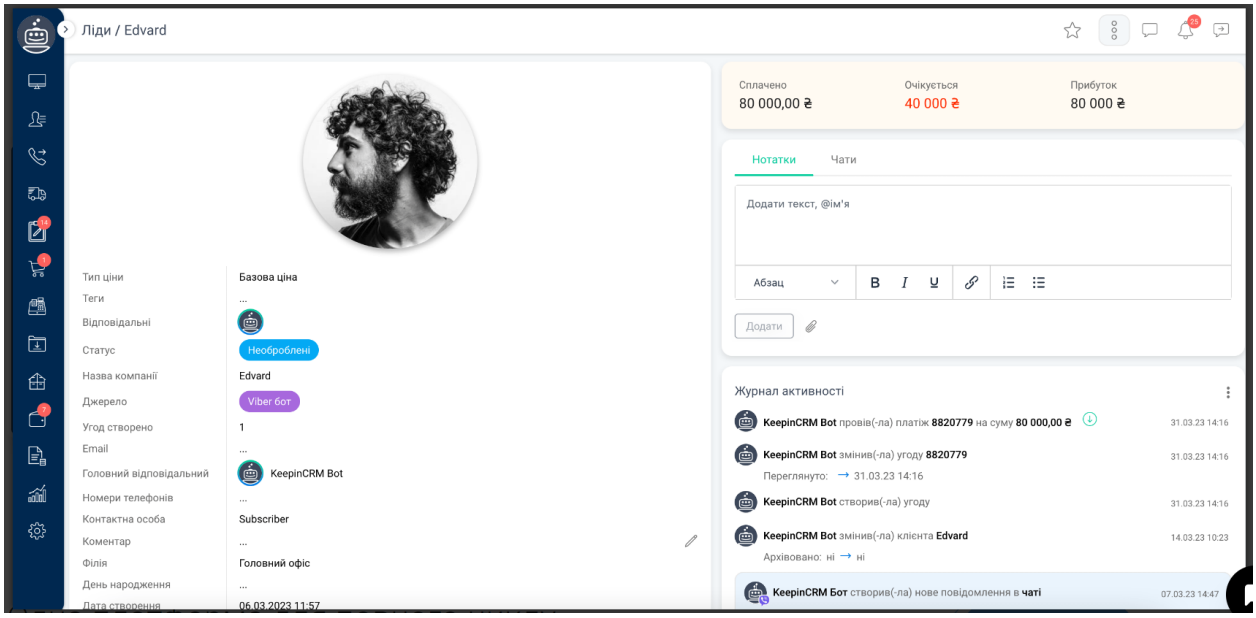

Рисунок 1.2 – KeepinCRM

Основний функціонал включає:

- Централізований журнал взаємодій із клієнтами

Всі етапи комунікації, від першого контакту до завершення угоди, фіксуються в системі з можливістю додавання відповідального менеджера, завдань та коментарів.

- Інструменти для масових комунікацій

Платформа підтримує розсилку SMS та Email для оперативного інформування клієнтів та лідів.

- Автоматизація фінансових операцій

Можливість створення рахунків, комерційних пропозицій та супровідної документації безпосередньо з системи.

- Воронка продажів

Графічне відображення етапів угод, що допомагає відстежувати прогрес угоди на кожному етапі циклу продажу.

- Інтегрований онлайн-консультант для сайту

Можливість налаштування віджета для комунікації з відвідувачами сайту в режимі реального часу.

- Аналітика фінансових результатів (P&L)

Вбудовані звіти для оцінки прибутковості та аналізу фінансових показників бізнесу.

- Система управління завданнями

Гнучке управління завданнями з контролем виконання, дедлайнів та історії змін.

- Гранульоване управління доступами

Гнучка система налаштувань дозволів для кожного користувача залежно від його ролі у компанії.

- Каталог товарів і прайс-листи

Можливість створення та редагування прайс-листів з актуальними цінами та товарними позиціями.

- Підтримка багатофілійності

Інструменти для управління кількома філіями з окремими базами даних.

- Складський облік

Вбудований модуль для контролю залишків товарів на складі та управління постачанням.

- Вбудований корпоративний чат

Можливість внутрішньої комунікації між співробітниками, включаючи створення групових чатів.

- Підтримка мультивалютності
- Календар та планування завдань
- Контроль логістичних процесів

Автоматизація процесу доставки товарів, включаючи формування товарно-транспортних накладних та моніторинг відправлень.

- Інтеграція IP-телефонії

Контроль та облік усіх дзвінків (вхідних, вихідних, пропущених) з функцією ведення статистики та запису розмов.

- Управління клієнтською базою
- Глибока аналітика продажів та продуктивності персоналу

Формування звітів за успішністю продажів, ефективністю співробітників та показниками товарообігу.

- Редагування товарів і послуг [7]

Унікальність KeepinCRM полягає в її здатності поєднувати в одному рішенні функції для управління взаємодією з клієнтами, фінансами та операційними процесами. Вона особливо підходить для компаній, які прагнуть автоматизувати ключові бізнес-задачі, спростити контроль над продажами та підвищити прозорість бізнесу. Її гнучкість та адаптивність робить систему ефективною як для невеликих команд, так і для підприємств із розгалуженою структурою.

1.3 Аналіз методів розробки CRM-систем

Сучасні інформаційні системи та програмні додатки досягли високого рівня розвитку, що робить критично важливим використання поняття

«архітектура» при їх проєктуванні. Архітектура програмного забезпечення визначає принципи побудови системи, взаємодію її компонентів, розподіл функціональності та способи управління даними. Грамотно спроєктована архітектура забезпечує стабільність, надійність, високу продуктивність системи, а також спрощує її подальше обслуговування та розвиток.

Процес планування структури веб-системи є складним організаційним завданням, яке безпосередньо впливає на ключові показники успіху проєкту: швидкість розробки, витрати, функціональність, зручність використання та естетичну привабливість інтерфейсу. Логічна структура веб-системи визначає спосіб організації інформації на сторінках, її ієрархію, доступність та зручність для користувача. Раціональне структурування контенту дозволяє оптимізувати навігацію, прискорити пошук потрібної інформації та створити позитивний користувацький досвід, що є однією з ключових умов успішного функціонування веб-платформи.

CRM-система (Customer Relationship Management) є складним веб-додатком, основним призначенням якого є автоматизація процесів управління взаємовідносинами з клієнтами. Вона дозволяє компаніям ефективно обробляти інформацію про клієнтів, зберігати історію комунікацій, автоматизувати процеси продажів, планувати маркетингові кампанії та аналізувати ефективність бізнес-процесів. Головною метою впровадження CRM є покращення якості обслуговування клієнтів, підвищення рівня продажів та створення єдиного інформаційного простору для управління бізнесом.

Методи розробки CRM-систем мають суттєве значення у процесі створення та впровадження програмного забезпечення для управління взаєминами з клієнтами. Вибір відповідного підходу до розробки забезпечує можливість ефективного управління відносинами із замовниками, зниження операційних витрат та підвищення ефективності бізнес-процесів. До основних методів розробки належать:

1. Водоспадний метод

Водоспадний метод, або каскадний підхід, є класичним способом організації процесу розробки програмного забезпечення, що передбачає лінійну послідовність етапів. Основними етапами цього методу є:

1. Аналіз вимог: Визначення та документування функціональних і нефункціональних вимог до системи.
2. Проєктування: Розробка архітектурних рішень, вибір технологічного стеку та створення детальної технічної документації.
3. Розробка: Написання коду відповідно до проєктної документації.
4. Тестування: Проведення тестів для перевірки відповідності системи заданим вимогам.
5. Впровадження: Перенесення системи у робоче середовище, налаштування та навчання користувачів.

Перевагою водоспадного методу є чіткість і структурованість процесу, що підходить для проєктів зі стабільними вимогами. Недоліком є обмежена гнучкість у разі необхідності внесення змін на пізніх етапах розробки.

2. Ітеративний метод

Ітеративний підхід передбачає реалізацію розробки у вигляді циклів, кожен з яких охоплює етапи аналізу, проєктування, реалізації та тестування. Особливостями цього методу є:

- Поступове уточнення вимог протягом життєвого циклу проєкту.
- Розробка частинами з поступовим вдосконаленням функціональності.
- Тестування на кожному етапі для забезпечення якості продукту.
- Оцінка результатів кожної ітерації для вдосконалення наступних етапів.

Цей підхід підходить для проєктів із неповністю визначеними вимогами, що потребують адаптації до змін у процесі розробки.

3. Гнучкі методології (Agile)

Agile є сукупністю гнучких методологій, спрямованих на ітеративний розвиток програмного забезпечення із пріоритетом співпраці між учасниками процесу та швидкої адаптації до змін. Основні принципи Agile включають:

1. Ітеративний розвиток: Розробка через короткі цикли (спринти).
2. Спільна робота та самоорганізація: Команда розробників має високу автономність у прийнятті рішень.
3. Гнучкість до змін: Вимоги можуть бути змінені навіть на пізніх етапах проєкту.
4. Активна взаємодія із замовником: Постійний зворотний зв’язок із клієнтом для вдосконалення продукту.

Найпоширенішими фреймворками для реалізації Agile є Scrum та Kanban, які сприяють швидкому отриманню результатів та ефективному управлінню ризиками.

4. Прототипування

Метод прототипування передбачає створення ранніх версій програмного забезпечення (прототипів) з метою перевірки функціональності та отримання зворотного зв’язку. Основні переваги цього підходу:

- Валідація ідей на ранніх етапах.
- Виявлення недоліків до початку повномасштабної розробки.
- Швидке реагування на потреби користувачів.

Прототипи можуть бути як статичними макетами, так і інтерактивними моделями, наближеними до кінцевого продукту.

5. Компонентно-орієнтована розробка

Компонентно-орієнтований підхід передбачає розподіл системи на незалежні компоненти, кожен з яких реалізує певну функціональність. Переваги цього методу включають:

- Модульність: Кожен компонент може бути розроблений та протестований окремо.
- Повторне використання коду: Компоненти можна використовувати у різних системах.

- Гнучкість розширення: Можливість швидкого додавання нових функцій.

Цей метод є ефективним для великих проєктів із високими вимогами до масштабованості та повторного використання функціональних модулів.

У процесі розробки CRM-систем ключову роль відіграє вибір відповідного методу, оскільки він визначає структуру роботи над проєктом, рівень гнучкості, а також ефективність управління ресурсами. Існують різні підходи, кожен із яких має свої переваги та недоліки, що робить їх більш або менш придатними для різних типів проєктів. Таблиця 1.2 демонструє порівняння основних методів за ключовими критеріями.

Таблиця 1.2 – Порівняння основних методів розробки

Метод	Основні характеристики	Переваги	Недоліки	Застосування
Водоспадний метод	Послідовний, лінійний підхід із фіксованими етапами	- Чітка структура	- Низька гнучкість	Стабільні вимоги, короткі проєкти
		- Простота управління	- Високий ризик невідповідності вимогам	
		- Повна документація	- Труднощі із внесенням змін	
Ітеративний метод	Циклічний процес із повторенням етапів розробки	- Гнучкість до змін	- Витрати ресурсів на кожен цикл	Довготривалі проєкти з поступовим розвитком
		- Можливість перевірки після кожної ітерації	- Можливість затягування термінів	
Agile-методології	Гнучкий підхід з короткими спринтами та постійною	- Висока адаптивність	- Складність управління великими командами	Динамічні проєкти, де вимоги можуть

	взаємодією із замовником	- Постійний зворотний зв’язок	- Вимога активної участі замовника	змінюватись
		- Прозорість процесу		
Прототипування	Створення ранніх версій системи для перевірки ідей та дизайну	- Мінімізація ризиків	- Обмежений функціонал прототипів	Перевірка концепцій, нові продукти
		- Можливість валідації вимог	- Може спричинити затримки при надто детальному прототипуванні	
		- Швидке отримання зворотного зв'язку		
Компонентно-орієнтована розробка	Поділ системи на незалежні компоненти, які можна використовувати повторно	- Висока модульність	- Складність інтеграції компонентів	Великі, масштабовані проєкти з можливістю розширення
		- Повторне використання коду	- Потребує чіткої архітектури	
		- Легке тестування		

Вибір методу розробки CRM-системи залежить від особливостей проєкту, стабільності вимог та рівня необхідної гнучкості. Водоспадний підхід підходить для чітко визначених проєктів, тоді як ітеративні та гнучкі методології забезпечують більшу адаптивність у мінливих умовах. Компонентно-орієнтована розробка та прототипування дозволяють ефективно управляти складними проєктами та зменшувати ризики помилок на ранніх етапах.

1.4 Висновок до першого розділу

У даному розділі було проведено аналіз предметної області CRM-систем для ресторанного бізнесу, зокрема їхніх функціональних можливостей та методів розробки. Було розглянуто основні принципи роботи CRM-систем, їхню роль у підвищенні ефективності управління клієнтськими відносинами та автоматизації бізнес-процесів.

Особливу увагу приділено функціональним можливостям сучасних CRM-систем, серед яких управління клієнтськими базами, автоматизація процесів продажів та маркетингових кампаній, а також інтеграція з іншими інформаційними системами підприємства. Було проаналізовано ключові підходи до розробки програмного забезпечення, включаючи каскадний, ітеративний підходи, Agile-методології та прототипування, з метою вибору найбільш ефективної моделі для розробки CRM-системи для ресторану.

Метою аналізу було визначення оптимальних інструментів та архітектурних рішень для створення ефективної CRM-системи, яка відповідатиме вимогам ресторанного бізнесу. Для досягнення цієї мети було виконано такі завдання:

1. Проведено огляд наукових джерел та аналітичних матеріалів щодо CRM-систем.
2. Визначено основні функціональні можливості та переваги CRM-систем у контексті ресторанного бізнесу.
3. Проаналізовано сучасні методи розробки CRM-систем, включаючи підходи до архітектурного проєктування.
4. Обґрунтовано вибір архітектури для подальшої розробки веб-додатку.

Отримані результати стали основою для подальшого розроблення CRM-системи, з урахуванням вимог до автоматизації процесів обслуговування клієнтів та управління замовленнями.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ CRM СИСТЕМИ

2.1 Вибір програмних засобів для реалізації проекту

Мережеві багатокористувацькі додатки можуть застосовувати два основні підходи до передачі повідомлень між користувачами: централізоване управління або розподілену синхронізацію. Централізований підхід, відомий як архітектура «клієнт-сервер», передбачає використання сервера, який виконує функції обробки клієнтських запитів, збереження інформації про стан користувачів та синхронізації між ними. Сервер отримує дані від кожного клієнта, обробляє їх та передає іншим користувачам, що забезпечує рівномірний розподіл навантаження та контроль за передачею даних.

Розподілена синхронізація, навпаки, передбачає взаємодію клієнтів між собою без залучення центрального сервера. У цьому випадку кожен учасник обмінюється даними безпосередньо з іншими, часто за принципом широкомовлення. Такий підхід сприяє зниженню ресурсних витрат, пов’язаних із підтримкою центрального вузла, проте ускладнює реалізацію клієнтської частини через необхідність обробки складніших мережевих процесів [6].

Існує також комбінований підхід, відомий як архітектура «точка-точка» (peer-to-peer), де дві сторони взаємодіють безпосередньо, динамічно змінюючи свої ролі клієнта та сервера, що підвищує гнучкість системи в процесі обміну даними.

Запропонована система для обміну повідомленнями через Інтернет доцільно базується на архітектурі клієнт-сервер, оскільки вона має низку ключових переваг. По-перше, використання сервера дозволяє знизити обчислювальне навантаження на клієнтські пристрої, оскільки основні процеси обробки даних виконуються на серверному боці. По-друге, централізоване управління усуває потребу у прямому з'єднанні між клієнтами, що сприяє підвищенню рівня безпеки та спрощенню мережевої взаємодії. По-

третє, централізована архітектура забезпечує стабільність системи, спрощує її адміністрування та технічне обслуговування, а також сприяє більш ефективному управлінню ресурсами.

Архітектура інформаційної системи — це концептуальна модель, що визначає структуру, функціональність, а також взаємозв'язки між компонентами системи. З розвитком програмного забезпечення інтеграція інформаційних систем між собою стає все більш важливою для створення єдиного інформаційного простору підприємства. Для побудови надійної архітектури та ефективної інтеграції програмних систем необхідно дотримуватися сучасних стандартів у цій галузі.

Програмні системи можна класифікувати за архітектурними підходами, серед яких виділяють:

- централізовану архітектуру;
- архітектуру типу «файл-сервер»;
- дворівневу архітектуру «клієнт-сервер»;
- багаторівневу архітектуру «клієнт-сервер»;
- архітектуру розподілених систем;
- архітектуру веб-додатків;
- сервіс-орієнтовану архітектуру.

Розроблювана CRM-система реалізована за принципом клієнт-серверної архітектури. Вона передбачає взаємодію двох незалежних процесів — клієнта та сервера, які можуть виконуватись на різних пристроях і обмінюватися даними через мережу. Така модель широко використовується для систем управління базами даних (СКБД), поштових серверів та інших інформаційних систем.

2.2 Вимоги до програмного забезпечення

Вимоги до розробки CRM-системи для ресторану визначають набір характеристик, необхідних для забезпечення ефективності, стабільності та

зручності керування замовленнями, меню та клієнтськими даними. Чітке формулювання цих вимог є критично важливим для створення якісного програмного забезпечення, яке відповідатиме потребам користувачів та технічним обмеженням.

Функціональні вимоги:

1. Реєстрація та автентифікація користувачів
2. Управління замовленнями: створення, редагування, перегляд, оновлення статусу та скасування замовлень.
3. Управління меню: додавання, редагування, видалення позицій меню з можливістю оновлення цін та складу.
4. Ведення клієнтської бази: збереження інформації про клієнтів, історія замовлень.
5. Генерація звітів про продажі, популярність страв, фінансові показники.
6. Бронювання столиків із зазначенням дати, часу та кількості осіб.
7. Відправлення автоматичних сповіщень користувачам про зміну статусу замовлення.
8. Можливість фільтрації замовлень за статусом, датою та клієнтом.
9. Ведення історії взаємодії з клієнтами для аналізу лояльності.

Нефункціональні вимоги:

1. Система не повинна втрачати у швидкодії.
2. Можливість розширення функціональності для мережі ресторанів.
3. Підтримка актуальних версій браузерів (Chrome, Firefox, Edge).
4. Авторизація через JWT.
5. Інтуїтивно зрозумілий інтерфейс з мінімальною кількістю кліків для оформлення замовлення.

Ці вимоги забезпечують ефективну реалізацію CRM-системи, спрямовану на автоматизацію бізнес-процесів ресторану, підвищення якості обслуговування клієнтів та оптимізацію управління ресурсами закладу.

2.3 Патерни проектування

Патерни проектування відіграють важливу роль у розробці веб-додатків, забезпечуючи чітку структуру, зрозумілість та можливість масштабування коду. Вони дозволяють програмістам застосовувати перевірені рішення для типових завдань, що виникають під час створення програмного забезпечення. Серед найбільш поширених патернів у веб-розробці виділяють Model-View-Controller (MVC), Model-View-ViewModel (MVVM) та VIPER. Кожен з них має свої унікальні особливості, переваги та недоліки, що робить їх важливими інструментами для розробників. Розглянемо кожен з них детальніше.

Model-View-Controller (MVC) — це патерн проектування, що використовується для організації коду в структурованих додатках, особливо у веб-розробці. Його основною ідеєю є поділ програми на три основні компоненти:

1. Model (Модель) — відповідає за управління даними додатка та бізнес-логікою.
2. View (Представлення) — відображає дані користувачеві.
3. Controller (Контролер) — координує взаємодію між Model та View, обробляючи запити користувача.

Переваги:

- Чіткий поділ обов'язків: Поділ на окремі компоненти спрощує розробку, тестування та підтримку коду.
- Модульність: Кожен компонент можна розробляти, тестувати та змінювати незалежно.
- Гнучкість: Можливість змінювати окремі частини програми без впливу на інші компоненти.
- Повторне використання коду: Модель і представлення можна використовувати повторно в різних частинах додатка або навіть у різних проєктах [12].

Model-View-ViewModel (MVVM) — патерн проектування, що забезпечує чітку структуру при створенні масштабованих додатків, особливо з використанням фреймворків на зразок WPF, Angular та React. Його основна мета — розділення інтерфейсу користувача та бізнес-логіки.

1. Model (Модель) — керує даними додатка, обробляє бізнес-логіку та надає дані ViewModel.

2. View (Представлення) — відповідає за візуальне відображення даних користувачу.

3. ViewModel — посередник між Model і View, який обробляє дані для зручного відображення у View та обробляє події від користувача.

Переваги:

- Чітке розділення обов'язків: Поділ на три компоненти спрощує розробку та обслуговування додатка.
- Модульність: Кожен компонент можна розробляти, тестувати та підтримувати окремо.
- Двостороння прив'язка даних: Зміни у ViewModel автоматично відображаються у View і навпаки, що спрощує роботу з UI.
- Повторне використання коду: Model та ViewModel можна використовувати повторно у різних частинах додатка або різних проєктах [13].

SDUI-MVVM (Server-Driven UI with Model-View-ViewModel) — це патерн проектування, який поєднує принципи серверно-орієнтованого інтерфейсу (SDUI) та архітектури Model-View-ViewModel (MVVM). Його основна мета — перенести управління інтерфейсом на сервер, тоді як клієнтська частина відповідає за рендеринг отриманих даних та їх обробку. Такий підхід використовується для створення структурованих, масштабованих веб-додатків, де інтерфейс може динамічно змінюватися без потреби у зміні клієнтського коду.

Основні компоненти SDUI-MVVM:

1. Server (Сервер) — відповідає за формування інтерфейсу та логіки його відображення, передаючи на клієнт JSON-структури або інші формати, які описують UI.

2. Model (Модель) — керує бізнес-логікою, обробляє серверні дані та забезпечує доступ до інформації.

3. View (Представлення) — відповідає за відображення даних користувачу, використовуючи інформацію, надану ViewModel.

4. ViewModel — виступає посередником між Model та View. Він отримує дані з Model, обробляє їх та передає View у зручному для відображення форматі. Також він обробляє події від View та може ініціювати нові серверні запити.

Переваги:

- Централізоване управління інтерфейсом: Сервер повністю контролює вигляд інтерфейсу, що дозволяє швидко оновлювати UI без оновлення клієнтського застосунку.
- Чітке розділення обов'язків: Поділ між сервером, моделлю, представленням та ViewModel спрощує розробку, тестування та підтримку коду.
- Модульність: Кожен компонент (Server, Model, View, ViewModel) може бути розроблений, протестований і підтримуваний незалежно, що сприяє масштабованості додатка.
- Гнучкість: Легка зміна інтерфейсу на стороні сервера без необхідності змін у клієнтському застосунку.
- Повторне використання коду: Компоненти Model та ViewModel можуть бути використані повторно у різних частинах додатка або навіть у різних проєктах [15].

VIPER (View, Interactor, Presenter, Entity, Router) - це патерн проектування, що застосовується для розробки модульних і масштабованих додатків, особливо в контексті мобільної розробки, але також може бути адаптований для веб-додатків. Основна мета VIPER - розділення

відповідальностей між різними компонентами для досягнення високої ступені модульності та тестованості коду.

1) View - відповідає за відображення даних і взаємодію з користувачем. Він отримує команди від Presenter і відображає відповідний контент.

2) Interactor - містить бізнес-логіку додатка. Він отримує запити від Presenter, обробляє їх, і повертає результат назад до Presenter.

3) Presenter - виступає як посередник між View та Interactor. Він отримує дані від Interactor і форматує їх для відображення у View. Також обробляє події з View і передає їх до Interactor.

4) Entity - містить модель даних. Це можуть бути об'єкти, що представляють дані додатка, такі як користувачі, продукти тощо.

Router - відповідає за навігацію між екранами додатка. Містить логіку переходів та навігаційних маршрутів [14].

Нижче представлена порівняльна таблиця, яка демонструє ключові відмінності між патернами проектування MVC (Model-View-Controller), MVVM (Model-View-ViewModel) та SDUI-MVVM (Server-Driven UI with Model-View-ViewModel). Ці патерни використовуються для організації архітектури програмних додатків з метою забезпечення чіткого розділення обов'язків, спрощення тестування, підтримки та масштабування коду. Кожен з них має свої переваги та сфери застосування, що робить їх придатними для різних типів веб-розробки. Таблиця 2.1 допоможе краще зрозуміти, як ці патерни працюють та в яких ситуаціях їх доцільно застосовувати.

Таблиця 2.1 – Порівняння патернів проектівання

Критерій	**MVC (Model-View-Controller)**	**MVVM (Model-View-ViewModel)**	**SDUI-MVVM (Server-Driven UI)**	**VIPER (View, Interactor, Presenter, Entity, Router)**
Призначення	Розділення логіки	Поділ інтерфейсу,	Віддалене управління	Модульний підхід для

	додатка на три компоненти для веб-додатків	логіки та даних, особливо для SPA	UI через сервер із застосуванням MVVM	масштабованих мобільних та веб-додатків
Компоненти	Model, View, Controller	Model, View, ViewModel	Server, Model, View, ViewModel	View, Interactor, Presenter, Entity, Router
Управління інтерфейсом	Клієнтська сторона	Клієнтська сторона	Серверна сторона	View керується через Presenter
Джерело управління логікою	Controller	ViewModel	Server та ViewModel	Interactor та Presenter
Двостороння прив'язка даних	Відсутня	Присутня	Присутня	Відсутня
Модульність	Висока	Висока	Висока	Висока
Гнучкість змін	Середня (зміни потребують оновлення клієнта)	Висока (двостороння прив'язка)	Висока (зміни реалізуються на сервері)	Висока (компоненти розділені)
Придатність для SPA	Менш придатний	Висока	Висока	Менш придатний
Повторне використання коду	Можливе	Високе	Високе	Високе
Відповідальність за логіку	Controller	ViewModel	Server та ViewModel	Interactor та Presenter
Основні сфери застосування	Веб-додатки з класичною структурою	SPA, Desktop-додатки, WPF	Веб-додатки з частими змінами UI	Мобільні додатки, веб-додатки зі складною навігацією
Приклади технологій	ASP.NET MVC, Ruby on Rails	WPF, Angular, React, Blazor	Next.js, Firebase, Netflix UI	iOS, Android, Swift, Kotlin, Clean Architecture

2.3 Засоби для розробки клієнтської частини

Розробка клієнтської частини CRM-системи для ресторану здійснюється із використанням фреймворку Angular.js, який є потужним інструментом для створення односторінкових веб-застосунків (SPA). Angular.js забезпечує зручність у розробці завдяки компонентному підходу, що дозволяє створювати повторно використовувані елементи інтерфейсу, спрощуючи підтримку та масштабування системи.

Однією з ключових переваг Angular.js є двостороннє зв’язування даних (two-way data binding), яке автоматично синхронізує модель даних із користувацьким інтерфейсом. Це означає, що будь-які зміни, внесені у форму введення або інший елемент інтерфейсу, миттєво відображаються у відповідній моделі даних, і навпаки. Така властивість особливо важлива для CRM-системи ресторану, оскільки дозволяє динамічно оновлювати статус замовлень, актуальність меню та інформацію про клієнтів без необхідності перезавантаження сторінки.

Інтерфейс у Angular.js побудований на основі компонентів та директив. Компоненти є незалежними блоками, які можна використовувати повторно в різних частинах застосунку. У CRM-системі для ресторану до таких компонентів можуть належати елементи для відображення списку замовлень, інформації про клієнтів чи управління меню. Angular.js також підтримує використання директив, які є спеціальними атрибутами та елементами HTML, що розширюють функціональність стандартної розмітки. Наприклад, директиви ng-repeat, ng-if та ng-show дозволяють зручно працювати зі списками замовлень, відображати лише актуальні дані та керувати видимістю елементів інтерфейсу на основі умов.

Важливим елементом розробки є використання сервісів Angular.js, які відповідають за управління бізнес-логікою та взаємодію з серверною частиною. Сервіси дозволяють централізувати логіку обробки даних, спрощуючи тестування та повторне використання коду. У CRM-системі

ресторану сервіси можуть відповідати за отримання даних про замовлення, управління інформацією про клієнтів або обробку запитів до сервера через вбудовані HTTP-клієнти, зокрема $http або HttpClient.

Для підвищення якості коду та зниження ймовірності помилок у розробці клієнтської частини системи використовується TypeScript. Це надбудова над JavaScript, яка додає підтримку статичної типізації. Завдяки TypeScript можна виявляти помилки ще на етапі компіляції, що значно підвищує стабільність великих проектів, таких як CRM для ресторану. Крім того, використання TypeScript покращує читабельність коду, оскільки дозволяє створювати інтерфейси для об’єктів, що забезпечує чітке визначення структури даних, таких як Order, Customer або MenuItem. Це мінімізує ймовірність помилок, пов’язаних із неправильним використанням об’єктів або їх властивостей [9].

2.4 Засоби для розробки серверної частини

Node.js — це сучасне серверне середовище для виконання JavaScript-коду на стороні сервера, розроблене компанією Google на основі рушія V8. Основна особливість цієї технології полягає у використанні неблокуючої архітектури, яка забезпечує ефективну обробку багатопоточних запитів у реальному часі. Це особливо актуально для систем управління ресторанами, де важливим є швидке опрацювання замовлень, оновлення меню та управління клієнтськими даними.

Node.js базується на подієво-орієнтованій моделі, де сервер працює в єдиному потоці, обробляючи асинхронні операції через подієвий цикл. Такий підхід дозволяє уникнути блокувань у процесі обробки запитів, що сприяє високій продуктивності навіть при значному навантаженні на систему. Це важливо для CRM-систем, які обробляють бронювання, замовлення та повідомлення одночасно.

Node.js також підтримує екосистему модулів через менеджер npm (Node Package Manager), що спрощує інтеграцію сторонніх бібліотек для розробки RESTful API, підключення до баз даних та забезпечення авторизації користувачів.

Переваги:

- Асинхронна архітектура – забезпечує обробку великої кількості запитів одночасно, підвищуючи швидкодію системи.
- Масштабованість – можливість обробки великої кількості з'єднань завдяки кластеризації.
- Єдиний стек технологій: Можливість використання JavaScript як для серверної, так і для клієнтської частини, що спрощує розробку.
- Велика спільнота: Широкий вибір готових модулів та активна підтримка розробників.

Недоліки:

- Однопотокова природа: Може ускладнювати обробку ресурсомістких завдань, таких як обчислювально складні операції.
- Обмежена підтримка реляційних баз: Для складних запитів до SQL-баз може знадобитися використання додаткових ORM-бібліотек.
- Чутливість до якості модулів: Велика кількість сторонніх пакетів може призводити до проблем із безпекою або продуктивністю [11].

JSON Web Token (JWT) — це стандарт відкритого формату для безпечної передачі інформації між сторонами у вигляді компактного веб-токена. Він використовується переважно для аутентифікації та авторизації у веб-застосунках, а також для обміну даними між різними системами. JWT забезпечує цілісність і перевірку достовірності переданих даних шляхом використання цифрового підпису.

JWT складається з трьох логічних компонентів: заголовка (Header), корисного навантаження (Payload) та підпису (Signature).

Заголовок (Header) містить метаінформацію про токен, зокрема вказівку на алгоритм підпису та тип токена.

Корисне навантаження (Payload) включає в себе твердження (claims) — набір заявлених даних про суб'єкта, якого стосується токен. Ці дані можуть містити ідентифікатор користувача, ім’я, електронну пошту, дату створення токена тощо. Твердження можуть бути стандартними (наприклад, час створення токена, термін його дії) або спеціально визначеними для конкретного застосунку.

Підпис (Signature) є результатом криптографічного перетворення, який гарантує цілісність токена та його захист від підробок. Він формується шляхом застосування алгоритму хешування до закодованих заголовка та корисного навантаження разом із секретним ключем.

JWT використовується у двох основних сценаріях: аутентифікація та авторизація.

- Аутентифікація,

Коли користувач вводить свої облікові дані, сервер перевіряє їх і генерує JWT. Цей токен передається користувачеві, який надалі використовує його для підтвердження своєї особи.

- Авторизація

При зверненні до захищених ресурсів користувач надсилає JWT разом із запитом. Сервер перевіряє достовірність токена та надає або відмовляє у доступі до запитуваних ресурсів.

Переваги використання JWT

1. Компактність: Токен має невеликий розмір, що спрощує його передавання через HTTP-запити, зокрема в заголовках.

2. Самодостатність: Усі необхідні дані містяться безпосередньо в токені, що дозволяє серверу працювати без постійного звернення до бази даних.

3. Міжплатформність: JWT можна використовувати в різних мовах програмування та протоколах.

4. Безстейтова природа: Серверу не потрібно зберігати стан користувача між сесіями, оскільки токен сам містить всю необхідну інформацію.

Недоліки та потенційні загрози

1. Неможливість відкликання токена
2. Вразливість до крадіжки токена.
3. Відсутність шифрування даних [10].

JSON Web Token (JWT) та платформа Node.js є потужними інструментами для створення сучасних веб-застосунків, зокрема CRM-систем для ресторанного бізнесу. JWT забезпечує компактний, автономний і міжплатформний спосіб аутентифікації та авторизації користувачів, що дозволяє безпечно передавати дані між сторонами. Водночас використання токенів потребує обережності через ризики, пов’язані з їхнім зберіганням і обробкою. Використання захищених з'єднань, обмеження терміну дії токена та впровадження механізмів оновлення сприяє підвищенню рівня безпеки.

Node.js, зі своєю асинхронною архітектурою, продуктивністю та масштабованістю, є ефективною платформою для створення CRM-систем. Вона здатна обробляти велику кількість запитів одночасно, що є важливим для автоматизації ключових бізнес-процесів у ресторані, таких як управління замовленнями, бронювання столиків та аналіз продажів. Поєднання JWT та Node.js дозволяє створити безпечний, швидкий та ефективний інструмент для оптимізації роботи ресторанного бізнесу.

2.5 Висновок до другого розділу

У другому розділі було проведено комплексний аналіз засобів для реалізації CRM-системи для ресторану. Розглянуто основні архітектурні підходи, зокрема централізовану архітектуру «клієнт-сервер», розподілену синхронізацію та комбіновану архітектуру «точка-точка». Обрано

централізовану архітектуру клієнт-сервер через її стабільність, ефективність у керуванні ресурсами та підвищений рівень безпеки даних.

Було детально досліджено патерни проектування, зокрема MVC, MVVM та SDUI-MVVM. Для реалізації CRM-системи обрано патерн SDUI-MVVM (Server-Driven UI with Model-View-ViewModel), який поєднує принципи серверно-орієнтованого інтерфейсу та архітектури MVVM. Це рішення забезпечує централізоване управління логікою відображення на сервері, що дозволяє динамічно змінювати інтерфейс користувача без оновлення клієнтського застосунку, а також покращує масштабованість та модульність системи.

Для клієнтської частини обрано Angular.js завдяки підтримці двостороннього зв’язування даних, компонентного підходу та використання TypeScript, що забезпечує надійність коду. Серверна частина реалізована на Node.js, що дозволяє ефективно обробляти асинхронні запити у реальному часі.

В результаті аналізу архітектурних підходів, патернів проектування та програмних засобів було обґрунтовано вибір патерну SDUI-MVVM у поєднанні з Angular.js та Node.js, що забезпечує стабільність, безпеку, гнучкість та зручність у подальшому розвитку CRM-системи для ресторану.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ДОДАТКУ НА ОСНОВІ СИТСТЕМИ CRM

3.1 Актуальність розробки

У сучасних умовах бізнесу важливою складовою успіху є ефективне управління взаємовідносинами з клієнтами. Це зумовлює актуальність розробки CRM (Customer Relationship Management) систем, які автоматизують процеси збору, обробки та аналізу даних про клієнтів. Проте зростаюча складність веб-додатків потребує більш гнучких архітектурних підходів для їхньої побудови. Одним із найбільш перспективних підходів є використання патерну SDUI (Server-Driven UI), що забезпечує централізоване управління інтерфейсом та підвищення ефективності розробки.

Патерн SDUI передбачає, що серверна частина відповідає не лише за бізнес-логіку, але й за управління структурою інтерфейсу, передаючи клієнтській частині опис UI-компонентів у вигляді JSON або XML. Це дозволяє динамічно змінювати користувацький інтерфейс без необхідності оновлення клієнтського коду. У контексті розробки CRM-систем це особливо актуально, оскільки такі додатки мають часто оновлювані елементи інтерфейсу, як-от списки клієнтів, статуси замовлень, показники продажів тощо.

Основна актуальність розробки веб-додатку CRM за допомогою патерну SDUI полягає в підвищенні гнучкості та швидкості внесення змін у додаток. Оскільки структура інтерфейсу визначається сервером, це дозволяє бізнесу швидко адаптувати систему під нові вимоги, оновлювати візуальні компоненти або додавати новий функціонал без необхідності повторного розгортання фронтенд-додатка. Це є критичним для CRM-рішень, де оновлення інтерфейсу може бути викликане змінами в маркетингових кампаніях або процесах обслуговування клієнтів.

Ще одним важливим аспектом актуальності є централізоване управління логікою додатка. У класичних архітектурах значна частина логіки реалізована

на клієнтській стороні, що може ускладнювати контроль за даними та призводити до дублювання коду. Використання SDUI дозволяє зосередити контроль над поведінкою інтерфейсу на сервері, що підвищує узгодженість роботи системи та зменшує ризик помилок, пов'язаних із розсинхронізацією клієнта та сервера.

Патерн SDUI також сприяє покращенню продуктивності веб-додатку, оскільки передача мінімального обсягу даних з серверу у вигляді JSON-повідомлень значно зменшує навантаження на клієнтську частину. Це дозволяє CRM-системам працювати стабільно навіть при обмеженій пропускній здатності мережі, що є важливим для систем з великою кількістю одночасних користувачів, наприклад, у великих ресторанах, де одночасно ведеться робота з клієнтськими замовленнями, постачальниками та фінансовими звітами.

Важливою перевагою використання SDUI у розробці CRM-систем є покращення безпеки. Контроль за інтерфейсом і бізнес-логікою на сервері мінімізує ризики маніпуляцій з боку клієнтської частини. Це особливо важливо для CRM-систем, де обробляються конфіденційні дані, такі як особиста інформація клієнтів, історія замовлень та фінансові транзакції.

Таким чином, актуальність розробки веб-додатків CRM-систем на основі патерну SDUI зумовлена його здатністю забезпечувати гнучкість, централізоване управління, підвищення продуктивності та покращення безпеки додатка. Цей підхід стає ключовим для сучасних бізнес-рішень, де критично важливо швидко реагувати на зміни ринкових умов, адаптувати інтерфейс для зручності користувачів та забезпечувати стабільну роботу додатка в умовах великих навантажень.

3.2 Структура проекту

Веб-додаток на основі CRM-системи, реалізований за патерном SDUI, складається з кількох ключових структурних підрозділів, які забезпечують централізоване управління як бізнес-логікою, так і інтерфейсними компонентами. Усі елементи інтерфейсу контролюються серверною частиною, що дозволяє забезпечити узгодженість даних, злагоджену роботу програмного забезпечення та ефективне управління бізнес-процесами. (рис. 3.1).

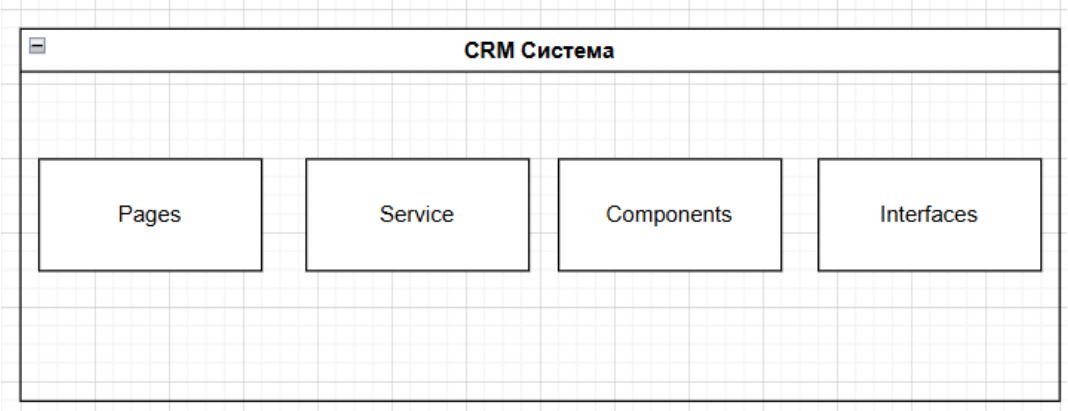

Рисунок 3.1 – Структура проекту

Спершу це Pages - фронтенд або клієнтська частина відповідає за інтерфейс користувача та взаємодію з ним. Це те, що бачить кінцевий користувач при роботі з додатком. Основні компоненти фронтенду включають панелі управління, форми для введення даних, таблиці з інформацією про клієнтів, замовлення та звіти. Для створення фронтенду використовуються технології HTML, CSS, JavaScript, а також фреймворки в даному випадку Angular.

Service - серверна частина відповідає за обробку даних, виконання бізнес-логіки та зв’язок між фронтендом і базою даних. Це ядро додатка, яке приймає запити від клієнтської частини, обробляє їх та надсилає відповідь. Основні технології для створення бекенду — Node.js, Express.js, ASP.NET

Core, Python (Django або Flask) тощо. Бекенд забезпечує аутентифікацію користувачів, обробку замовлень, розрахунки та управління ролями.

База даних є основним сховищем всієї інформації додатку, включаючи дані про клієнтів, замовлення, меню, фінансові звіти та інші бізнес-дані. Для цього використовуються об’єкти intefaces.

Components - це інтерфейсні модулі або компоненти веб-додатку є структурними елементами клієнтської частини (фронтенду), які відповідають за окремі функціональні елементи користувацького інтерфейсу. Вони забезпечують зручний, інтуїтивно зрозумілий досвід взаємодії з CRM-системою та сприяють візуальній узгодженості додатка.

Спочатку необхідно детально розглянути серверну частину проєкту, оскільки вона є основою функціонування всього додатка. Серверна частина відповідає за обробку запитів, збереження даних, управління користувачами та логіку бізнес-процесів. Вона є ключовим елементом, який забезпечує коректну роботу інтерфейсу та доступ до даних, а також виконує функції безпеки та контролю доступу.

Особливу увагу слід приділити компонентам зі структурного підрозділу Services, які відповідають за реалізацію бекенд-логіки. Вони включають обробку HTTP-запитів, взаємодію з базою даних, управління автентифікацією та авторизацією користувачів.

Перший компонент ApiService, використовується для тестування додатку. Тестування є ключовим етапом у розробці веб-додатків, оскільки воно дозволяє перевірити коректність роботи окремих компонентів та сервісів, забезпечуючи надійність і передбачуваність функціональності додатка.

У наведеному коді використовується метод describe (рис. 3.2), який є основним блоком для групування пов’язаних тестів у Jasmine — популярному фреймворку для написання тестів у JavaScript та TypeScript. Він визначає групу тестів для сервісу ApiService та виконує перевірку, чи створюється він правильно. Ключовим аспектом тестування є використання методу beforeEach. Цей метод виконується перед кожним тестом у групі та відповідає за

ініціалізацію тестового середовища. У ньому використовується TestBed.configureTestingModule({}), який створює тестовий модуль Angular, і дозволяє підготувати ізольоване оточення для кожного тесту. Це середовище дає змогу виконувати тести без залежності від інших частин застосунку, що є важливим принципом модульного тестування.

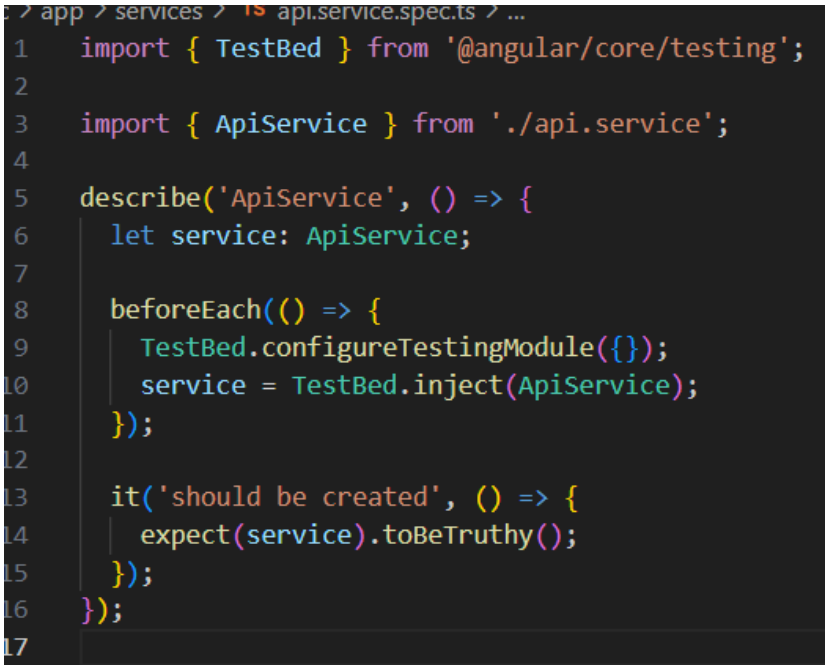

Рисунок 3.2 - Сервіс ApiService

Сервіс ApiService ініціалізується через метод TestBed.inject(ApiService), який відповідає за отримання екземпляра сервісу з DI-контейнера (Dependency Injection). Angular використовує механізм інверсії управління (IoC), що дозволяє зручно управляти залежностями між об'єктами, зокрема й у тестовому середовищі.

Основний тест перевіряє, чи успішно створюється екземпляр ApiService. Для цього використовується метод expect(service).toBeTruthy();, який перевіряє, чи об'єкт service був створений і чи він не є null або undefined. Це базова перевірка, яка гарантує, що сервіс може бути успішно ініціалізований у тестовому середовищі.

Наступний Angular-сервіс DishService (рис. 3.3), призначений для взаємодії з API ресторанної CRM-системи. Основне завдання цього сервісу — отримання списку страв із сервера за допомогою HTTP-запиту. Він демонструє використання сучасних підходів до управління залежностями та обробки HTTP-запитів у Angular, зокрема через впровадження механізму inject() і реактивного програмування за допомогою RxJS.

```
import { Injectable, inject } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { firstValueFrom } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DishService {
  http = inject(HttpClient);
```

Рисунок 3.3 - Angular-сервіс DishService

Ключовим елементом цього сервісу є декоратор @Injectable({ providedIn: 'root' }). Він вказує, що сервіс буде доступний у всьому додатку (глобально), без необхідності додаткового імпорту до окремих модулів. Це забезпечує централізоване управління сервісом і спрощує доступ до нього з будь-якого компонента Angular.

У цьому прикладі використовується новий підхід до ін'єкції залежностей через inject(), який був представлений в Angular 14. Метод inject (рис. 3.4) дозволяє напряму отримати екземпляр HttpClient без використання конструктора. HttpClient є основним механізмом для виконання HTTP-запитів у Angular та дозволяє відправляти GET, POST, PUT, DELETE-запити до зовнішніх API.

Основний метод getAllDishes (рис. 3.4) відповідає за отримання списку страв із сервера. Він використовує HttpClient.get() для виконання GET-запиту до сторінки /admin/dish. Для автентифікації використовується токен, збережений у localStorage. Значення токену зчитується через localStorage.getItem('token') і передається у заголовках (Authorization). Це стандартний підхід для авторизації через JSON Web Tokens (JWT) у RESTful API, де сервер перевіряє валідність токена перед обробкою запиту.

Рисунок 3.4 – Методи getAllDishes, inject

Метод повертає Promise<any> завдяки використанню firstValueFrom() з бібліотеки RxJS. firstValueFrom() перетворює потік значень (Observable) на Promise, що дозволяє працювати з асинхронними даними за допомогою стандартних конструкцій async/await. Це спрощує використання сервісу в компонентах, де часто потрібно отримати результат запиту одноразово.

Оголошення rootURL як окремої властивості дозволяє централізовано керувати базовим шляхом до API. Це сприяє покращенню підтримуваності коду: у разі зміни URL сервера, достатньо внести правки лише в одному місці.

Далі сервіс MenuService. Він відповідає за взаємодію з API ресторанної CRM-системи для управління меню. Основна функція цього сервісу полягає у виконанні CRUD-операцій (Create, Read, Update, Delete) через HTTP-запити,

дозволяючи отримувати, створювати, оновлювати та видаляти меню на сервері. Сервіс використовує сучасний підхід до управління залежностями за допомогою методу inject() замість класичного використання конструктора.

Сервіс позначений декоратором @Injectable({ providedIn: 'root' }), що забезпечує його глобальну доступність у всьому додатку Angular. Це означає, що сервіс може бути використаний у будь-якому компоненті без необхідності додаткового імпорту. Для виконання HTTP-запитів використовується HttpClient, який впроваджується через метод inject(HttpClient) і дозволяє надсилати асинхронні запити до API.

```
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { firstValueFrom } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class MenuService {

  private http = inject(HttpClient);
```

Рисунок 3.5 – Сервіс MenuService

Усі методи сервісу базуються на асинхронній роботі з даними та використовують функцію firstValueFrom з бібліотеки RxJS. Цей підхід дозволяє конвертувати потік даних Observable у Promise, що спрощує обробку результатів у компонентах. Запити до API включають передачу токена для автентифікації, який зчитується з localStorage. Це стандартний підхід для реалізації авторизації за допомогою JWT (JSON Web Token), де токен використовується для перевірки прав доступу до захищених ресурсів.

Метод getAllMenus(order: string) призначений для отримання всіх меню із сервера з можливістю сортування (рис. 3.6). Він приймає параметр order для визначення способу сортування даних (наприклад, за датою чи назвою меню). Токен авторизації отримується з localStorage та передається в заголовках

запиту для аутентифікації користувача. Метод використовує firstValueFrom для конвертації потоку Observable у Promise, що дозволяє зручно працювати з результатами запиту в асинхронному стилі.

Метод getMenuById(id: number) дозволяє отримати конкретне меню за унікальним ідентифікатором id (рис. 3.6). Як і попередній метод, він використовує токен авторизації, збережений у localStorage. HTTP-запит типу GET надсилається до сторінки, доповненого ідентифікатором меню, а результат повертається як Promise для асинхронної обробки.

```
async getAllMenus(order: string): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return await firstValueFrom(this.http.get<any>(`${this.rootUrl}
}

async getMenuById(id: number): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return await firstValueFrom(this.http.get<any>(`${this.rootUrl}/
}
```

Рисунок 3.6 – Методи getAllMenus, getMenuById

Метод createMenu(name: string, date: string, dishes: any[], price: number) призначений для створення нового меню. Він приймає чотири параметри: назву меню, дату, список страв (dishes) та ціну. Ці дані разом із токеном передаються у вигляді POST-запиту до API, що дозволяє зберегти нове меню у базі даних сервера. Токен авторизації включається у заголовки запиту для забезпечення захищеного доступу до API.

Метод updateMenu(id: number, name: string, date: string, dishes: any[], price: number) використовується для оновлення існуючого меню. Він приймає ті ж параметри, що й метод створення, з додатковим параметром id, який визначає конкретне меню для оновлення. Запит PUT надсилається до відповідного endpoint, включаючи нові дані меню та токен авторизації. Як і у попередніх методах, використовується firstValueFrom для конвертації Observable у Promise.

Метод getDailyMenu(date: string) повертає меню, доступне на конкретну дату (рис. 3.7). Цей метод не потребує авторизації, оскільки токен не передається у заголовках запиту. Він надсилає GET-запит до публічного endpoint з передачею параметра date у рядку запиту. Це може бути корисним для користувачів, які хочуть переглянути меню без входу в систему.

Метод deleteMenu(id: number) відповідає за видалення меню за його унікальним ідентифікатором (рис. 3.7). Він надсилає DELETE-запит до сервера, використовуючи токен для авторизації. Запит включає ідентифікатор меню в URL-адресі, а результат повертається як Promise.

```
async getDailyMenu(date: string): Promise<any> {
  const result = await firstValueFrom(this.http.get<any>(`https://
  return result;
}

async deleteMenu(id: number): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return await firstValueFrom(this.http.delete<any>(`${this.rootUr
}
```

Рисунок 3.7 – Методи getDailyMenu, deleteMenu

Усі методи сервісу мають уніфікований підхід до автентифікації, використовуючи токен із localStorage для доступу до захищених endpoint API. Методи firstValueFrom з бібліотеки RxJS дозволяють конвертувати потоки даних Observable у Promise, спрощуючи роботу з асинхронними запитами. Загалом, сервіс MenuService забезпечує базову функціональність для управління меню в CRM-системі, підтримуючи як публічні, так і захищені запити до серверної частини додатка.

Сервіс ReservationService у фреймворку Angular, який відповідає за управління бронюваннями клієнтів у ресторанній CRM-системі. Він надає функціональність для створення, отримання та оновлення статусу бронювань через HTTP-запити до зовнішнього API. Його структура побудована на сучасних підходах до ін'єкції залежностей та асинхронної обробки даних.

Метод createReservation відповідає за створення нового бронювання. Він приймає об'єкт reservation типу IReservation як параметр, який містить усі необхідні дані для створення бронювання, наприклад, ім'я клієнта, дату та час резервування. Запит відправляється методом POST до ендпоінту /reservations, а для аутентифікації передається токен, збережений у localStorage. Для асинхронної обробки результату використовується метод firstValueFrom з бібліотеки RxJS, який конвертує потік значень Observable у Promise.

Метод getReservationById (рис. 3.8) використовується для отримання бронювань за унікальним ідентифікатором клієнта (id). Він надсилає GET-запит до ендпоінту /reservations/customer, використовуючи параметр id у рядку запиту. Як і в інших методах, для захисту даних передається токен авторизації у заголовках запиту.

```
createReservation(reservation: IReservation): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(
    this.http.post<IReservation>(`${this.rootUrl}/reservations`, reserva
}

getReservationById(id: number): Promise<ICustomerReservationResponse[]>
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(
    this.http.get<ICustomerReservationResponse[]>(`${this.rootUrl}/reser
}
```

Рисунок 3.8 – Методи getReservationById, getReservationByUserId

Метод getReservationByUserId (рис. 3.8) схожий за функціональністю на попередній, але замість id клієнта використовує userId, що може бути корисним для пошуку бронювань, пов’язаних із конкретним користувачем системи. Логіка обробки запиту залишається ідентичною — використовується GET-запит з параметром у URL-рядку.

Метод getReservations (рис. 3.9) забезпечує гнучкий пошук бронювань за різними критеріями. Він приймає об'єкт reservation типу IReservationOptionalParameters, який може містити довільний набір параметрів

для фільтрації результатів (наприклад, діапазон дат, статус бронювання тощо). Для побудови URL-рядка параметри об'єкта перетворюються у рядок запиту через цикл for...in. Метод динамічно формує параметризований GET-запит до API, включаючи передані критерії пошуку.

Метод changeStatusReservation (рис. 3.9) дозволяє змінити статус конкретного бронювання. Він приймає два параметри: id бронювання та новий status. Для оновлення статусу використовується PUT-запит, де статус передається в URL-адресі. Цей метод використовується для зміни стану бронювання, наприклад, з "очікується" на "підтверджено" або "скасовано". Токен авторизації також включається у заголовки запиту.

Рисунок 3.9 – Методи getReservations, changeStatusReservation

Сервіс ReviewsService (рис. 3.10) у фреймворку Angular, який відповідає за управління відгуками користувачів у CRM-системі ресторану. Основне завдання сервісу — забезпечення операцій для відгуків через HTTP-запити до серверного API.

```
export class ReviewsService {

  private http = inject(HttpClient);

  private rootUrl = 'https://crm-restaurantapi-z9vj.onrender.com/api';


  getReviews(): Promise<any> {
    const token = localStorage.getItem('token') as string;
    const headers = { 'Authorization': token };
    return firstValueFrom(this.http.get<any>(`${this.rootUrl}/review`, { headers }));
  }
```

Рисунок 3.10 – Сервіс ReviewsService

Метод getReviews дозволяє отримати відгуки, які були створені поточним автентифікованим користувачем. Для цього виконується GET-запит до /review. Оскільки доступ до цієї інформації обмежений лише для авторизованих користувачів, метод зчитує токен з localStorage та передає його через заголовок Authorization. Такий підхід забезпечує безпеку та обмежує доступ до даних лише для відповідних користувачів.

Метод getReviewsAll використовується для отримання всіх відгуків, доступних у системі, без обмежень доступу. Цей метод може бути застосований у публічних розділах додатка для відображення всіх наявних відгуків, оскільки токен не передається у заголовках.

```
  getReviews(): Promise<any> {
    const token = localStorage.getItem('token') as string;
    const headers = { 'Authorization': token };
    return firstValueFrom(this.http.get<any>(`${this.rootUrl}/review`, { headers }));
  }

  getReviewsAll(): Promise<any> {
    return firstValueFrom(this.http.get<any>(`${this.rootUrl}/review/all`));
  }
```

Рисунок 3.11 – Методи getReviews, getReviewsAll

Метод getSomeReviews(amount: number, order: string) дозволяє отримати обмежену кількість відгуків, використовуючи параметри limit та order. Він надсилає GET-запит до API, де вказані параметри передаються у рядку запиту.

Це може бути корисним для відображення найбільш актуальних або найпопулярніших відгуків у вигляді списку на головній сторінці ресторану.

Метод createReview відповідає за створення нового відгуку. Він приймає об'єкт review як параметр і надсилає його на сервер за допомогою POST-запиту до /review. Оскільки створення відгуків дозволено лише авторизованим користувачам, у запиті також передається токен у заголовках для перевірки прав доступу.

```
getSomeReviews(amount: number, order: string): Promise<any> {
  return firstValueFrom(this.http.get<any>(`${this.rootUrl}/review/a
}

createReview(review: any): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(this.http.post<any>(`${this.rootUrl}/review`
}
```

Рисунок 3.12 – Методи getSomeReviews, createReview

Метод deleteReview (рис. 3.13) дозволяє користувачу видалити власний відгук за унікальним ідентифікатором id. Використовується DELETE-запит до ендпоінту /review/{id}. Для безпеки перевірка автентифікації здійснюється шляхом передачі токена у заголовках запиту.

Метод deleteReviewAdmin (рис. 3.13) має схожу функціональність, але призначений для адміністраторів системи. Він дозволяє видаляти будь-які відгуки в системі, використовуючи /admin/review/{id}. Це забезпечує можливість модерації контенту, наприклад, у випадках, коли відгук порушує політику компанії.

```
deleteReview(id: string): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(this.http.delete<any>(`${this.rootUrl}
}


deleteReviewAdmin(id: string): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(this.http.delete<any>(`${this.rootUrl}
}
```

Рисунок 3.13 – Методи deleteReview, deleteReviewAdmin

Таким чином, сервіс ReviewsService забезпечує комплексний функціонал для управління відгуками, включаючи створення, отримання, фільтрацію, а також видалення відгуків як користувачами, так і адміністраторами.

Далі сервіс TablesService, що відповідає за управління столиками в CRM-системі ресторану. Основне завдання цього сервісу — забезпечення базових операцій для управління столиками, включаючи отримання списку доступних столиків, створення нових, оновлення їхньої місткості та видалення.

```
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { first, firstValueFrom } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class TablesService {

  private http = inject(HttpClient);
  private rootUrl = 'https://crm-restaurantapi.onre

  getTables(): Promise<any> {
```

Рисунок 3.14 – Сервіс TablesService

Метод getTables відповідає за отримання списку всіх доступних столиків у системі. Він виконує GET-запит до API за /tables, передаючи токен для перевірки прав доступу. Повертається promis, який дозволяє асинхронно обробляти результат запиту.

Метод createTable дозволяє створити новий столик у базі даних. Він приймає об'єкт table, який містить необхідну інформацію для створення столика (наприклад, номер столика, місткість). HTTP-запит типу POST надсилається до /tables, а токен передається для авторизації. Цей метод може використовуватися адміністраторами системи або менеджерами ресторану для додавання нових столиків до системи.

Метод updateTableCapacity використовується для оновлення місткості конкретного столика. Він приймає два параметри — id столика та нове значення місткості capacity. Оновлення здійснюється через PUT-запит до /tables/{id}/capacity/{capacity}. Цей функціонал може бути корисним у випадках, коли адміністрація ресторану хоче змінити розміщення або кількість місць за столиком для покращення обслуговування клієнтів.

```
createTable(table: any): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(
    this.http.post<any>(`${this.rootUrl}/tables`, table, { headers })
  )
}

updateTableCapacity(id: number, capacity: number): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(
    this.http.put<any>(`${this.rootUrl}/tables/${id}/capacity/${capac
  )
}
```

Рисунок 3.15 – Методи createTable, updateTableCapacity

Метод deleteTable (рис. 3.16) забезпечує видалення столика з бази даних за його унікальним ідентифікатором. DELETE-запит надсилається до API за

/tables/{id}, а токен передається для перевірки прав адміністратора. Цей метод дозволяє видаляти непотрібні або зайві столики з бази даних.

Метод getFutureTables (рис. 3.16) використовується для отримання доступних столиків на певну майбутню дату. Він приймає параметр date та надсилає GET-запит до /tables/future/{date}. Це може бути корисним для перевірки доступності столиків для майбутніх бронювань або організації подій. Токен також передається у заголовках для обмеження доступу лише для авторизованих користувачів.

```
deleteTable(id: number): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(
    this.http.delete<any>(`${this.rootUrl}/tables/${id}`, { headers })
  )
}

getFutureTables(date: string): Promise<any> {
  const token = localStorage.getItem('token') as string;
  const headers = { 'Authorization': token };
  return firstValueFrom(
    this.http.get<any>(`${this.rootUrl}/tables/future/${date}`, { headers }));
}
```

Рисунок 3.16 – Методи deleteTable, getFutureTables

Далі потрібно розглянути функціонал сторінок додатку. Спочатку потрібно розглянути функціонал CreateReview. компонент CreateReviewComponent у фреймворку Angular, призначений для створення відгуків користувачами в CRM-системі ресторану. Компонент використовує реактивні форми (ReactiveFormsModule) для керування введенням даних та забезпечення валідації, а також інтегрує сервіси для роботи з API та навігації.

```
import { Component, inject } from '@angular/core';
import { FooterComponent } from "../../components/footer/footer.component";
import { HeaderComponent } from "../../components/header/header.component";
import { FormsModule, ReactiveFormsModule, FormGroup, FormControl, Validators } from '@angular/forms';
import { ReviewsService } from '../../services/reviews.service';
import { IUserResponse } from '../../interfaces/user.interfaces';
import Swal from 'sweetalert2';
import { Router } from '@angular/router';
@Component({
  selector: 'app-create-review',
  standalone: true,
  imports: [FooterComponent, HeaderComponent, FormsModule, ReactiveFormsModule],
  templateUrl: './create-review.component.html',
  styleUrl: './create-review.component.css'
})
export class CreateReviewComponent {

  reviewForm: FormGroup;

  // Inyección del servicio de reseñas y router para la navegación
```

Рисунок 3.17 – компонент CreateReviewComponent

Компонент визначений як standalone, що означає його незалежність від модульної системи Angular, завдяки чому він може імпортувати необхідні залежності напряму через параметр imports. У цьому випадку імпортуються два вкладені компоненти FooterComponent та HeaderComponent, які використовуються для відображення шапки та підвалу сторінки. Для роботи з формами додано FormsModule та ReactiveFormsModule.

В основі компонента лежить об'єкт FormGroup під назвою reviewForm, який ініціалізується у конструкторі. У формі є два поля: rating (оцінка) та comment (коментар), обидва з яких мають обов'язкові валідатори (Validators.required). Це гарантує, що користувач не зможе надіслати порожній відгук.

Метод onSubmit() викликається при надсиланні форми. Він перевіряє, чи всі поля форми заповнені коректно, і лише після цього викликає метод createReview() для обробки даних. Метод createReview() реалізований як асинхронний і відповідає за взаємодію з сервером через сервіс ReviewsService. Спочатку він оновлює значення полів форми за допомогою методу patchValue(), щоб переконатися, що форма містить актуальні дані. Потім створюється об'єкт із полями rating та comment, які передаються у метод createReview() сервісу для надсилання POST-запиту на сервер.

```
async createReview() {
  try {
    if (this.reviewForm.valid) {

      this.reviewForm.patchValue({
        rating: this.reviewForm.get('rating')?.value,
        comment: this.reviewForm.get('comment')?.value
      });

      const rating = this.reviewForm.get('rating')?.value;
      const comment = this.reviewForm.get('comment')?.value;

      const reviews = await this.reviewsService.createReview({
        comment: comment,
        rating: rating
      });
      Swal.fire({
        title: "¡Gracias por tu comentario!",
        icon: "success",
        preConfirm: () => {
          this.router.navigate(['/user']);
        }
      });

      this.reviewForm.reset();
    }
  } catch (error: any) {
```

Рисунок 3.18 – Методи onSubmit, createReview

У разі успішного створення відгуку використовується бібліотека SweetAlert2 для відображення модального вікна з повідомленням про успіх. Після підтвердження цього повідомлення користувач перенаправляється на сторінку /user за допомогою Router для зручності навігації.

Якщо під час створення відгуку виникає помилка, вона обробляється через блок catch. У випадку помилки об'єкт error.error інтерпретується як відповідь API типу IUserResponse, яка містить статус, заголовок та повідомлення про помилку. Якщо повідомлення про помилку містить текст "Need a completed reservation before making a review", воно перекладається і виводиться українською мовою: "Потрібно мати хоча б одну завершену резервацію для створення відгуку". Це покращує зручність використання додатка для кінцевих користувачів.

Наступний компонент відповідає за логіку авторизації LoginComponent. Він використовує реактивні форми (ReactiveFormsModule) для забезпечення валідованого введення даних користувачем під час входу в систему, а також включає механізми обробки помилок та маршрутизації.

Рисунок 3.19 – Компонент LoginComponent

Основним елементом компонента є форма loginForm (рис. 3.19), яка ініціалізується в конструкторі через FormGroup. Форма складається з двох полів: email та password. Для поля email використовуються валідатори required та email, які перевіряють, чи введено значення, та чи воно відповідає формату електронної пошти. Поле password також має валідатор обов'язковості (required) та мінімальної довжини (minLength(8)), що підвищує рівень безпеки паролів.

Метод onSubmit() використовується для обробки натискання кнопки входу. Він перевіряє, чи форма заповнена коректно (this.loginForm.valid), і лише після цього викликає метод loginUser() для взаємодії з сервером.

Метод loginUser (рис. 3.20) є асинхронним і відповідає за комунікацію з API через сервіс ApiService. Він зчитує введені користувачем дані (email і password) та передає їх у метод loginUser сервісу ApiService для надсилання POST-запиту на сервер. Якщо аутентифікація проходить успішно, токен авторизації, отриманий у відповіді від сервера, зберігається у localStorage для подальшого використання при взаємодії з API. Після цього відбувається додатковий запит до сервера для отримання інформації про користувача через метод getUser() сервісу ApiService. На основі ролі користувача (admin або звичайний користувач) виконується перенаправлення на різні сторінки: якщо

користувач має роль адміністратора, його перенаправляють на /dashboard, а якщо це звичайний користувач — на головну сторінку /.

```
async loginUser() {
  try {

    const email = this.loginForm.get('email')?.value;
    const password = this.loginForm.get('password')?.value;

    const user = await this.apiService.loginUser({ email, password });

    localStorage.setItem('token', user.token);

    const response = await this.apiService.getUser();

    if (response.data.role === 'admin') {
      this.router.navigate(['/dashboard']);
    } else {
      this.router.navigate(['/']);
    }
```

Рисунок 3.20 – Метод loginUser

У разі виникнення помилки аутентифікації метод loginUser обробляє її в блоці catch. Об'єкт помилки (error.error) перетворюється у відповідь типу IUserResponse, з якої зчитуються status, title та message. Для покращення зворотного зв'язку з користувачем використовується бібліотека SweetAlert2, яка виводить модальне вікно з повідомленням "Usuario y/o contraseña incorrectos" (Ім'я користувача або пароль неправильні).

Метод getPasswordErrorMessage() використовується для відображення повідомлення про помилку, якщо поле password залишено порожнім. Якщо умова required не виконана, виводиться повідомлення "Password is required". Це дозволяє підвищити зручність використання форми.

Метод checkValidation() використовується для перевірки валідності конкретного поля форми. Він повертає true, якщо поле є недійсним (invalid) і вже було торкнуте користувачем (touched).

```
  getPasswordErrorMessage(): string {
    const passwordControl = this.loginForm.get('password');
    if (passwordControl?.errors?.['required']) {
      return 'Password is required';
    }
    return '';
  }


  checkValidation(field: string) {
    return this.loginForm.get(field)?.invalid && this.loginForm.get(field)?.touched;
  }
}
```

Рисунок 3.21 – Методи getPasswordErrorMessage, checkValidation

Компонент HomeComponent у фреймворку Angular, який виконує роль головної сторінки веб-додатка для ресторанної CRM-системи. Він призначений для відображення основного контенту додатка, включаючи відгуки клієнтів, а також містить базові компоненти для навігації та структури сторінки.

Основним функціоналом компонента є отримання та відображення останніх відгуків користувачів. Для цього використовується сервіс ReviewsService, який підключається через ін'єкцію залежностей (inject(ReviewsService)), що є сучасним підходом в Angular для підвищення читабельності та зменшення використання конструктора. Сервіс зберігається у приватній властивості reviewsService.

```
import { Component, inject } from '@angular/core';
import { HeaderComponent } from "../../components/header/header.component";
import { RouterLink } from '@angular/router';
import { FooterComponent } from "../../components/footer/footer.component";
import { ReviewCardComponent } from "../../components/dashboard/review-card/revi
import { ReviewsService } from '../../services/reviews.service';
import { IUserResponse } from '../../interfaces/user.interfaces';



@Component({
  selector: 'app-home',
  standalone: true,
  imports: [HeaderComponent, RouterLink, FooterComponent, ReviewCardComponent],
  templateUrl: './home.component.html',
  styleUrl: './home.component.css'
})
export class HomeComponent {
```

Рисунок 3.22 – властивості reviewsService

Компонент містить асинхронний метод ngOnInit (рис. 3.23), який викликається автоматично при ініціалізації компонента. У цьому методі виконується звернення до сервісу ReviewsService через метод getSomeReviews(), який отримує останні три відгуки (3) у порядку спадання (desc). Результат запиту зберігається у масиві reviews, який надалі використовується для відображення даних у шаблоні компонента через ReviewCardComponent.

Важливим аспектом компонента є обробка помилок. У разі невдалої спроби отримання відгуків (наприклад, через проблеми з мережею або сервером) помилка обробляється у блоці catch. Помилка приводиться до типу IUserResponse для зручності обробки, а потім її статус, заголовок та повідомлення виводяться в консоль для діагностики.

3.3 Тестування проекту

Завантажимо проект у веб-браузері. Користувач повинен бачити стартовий екран, в якому є по центру є заголовок «Ласкаво просимо на Frontend Feast!» (рис. 3.23)

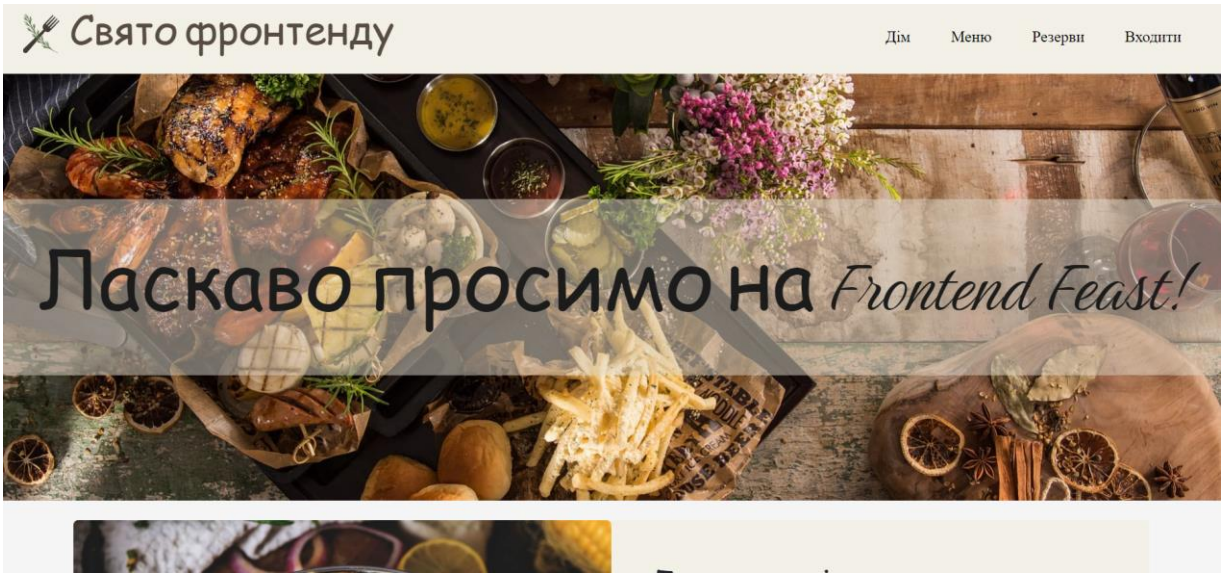

Рисунок 3.23 – Головна сторінка

На цій сторінці можна перейти до меню страв

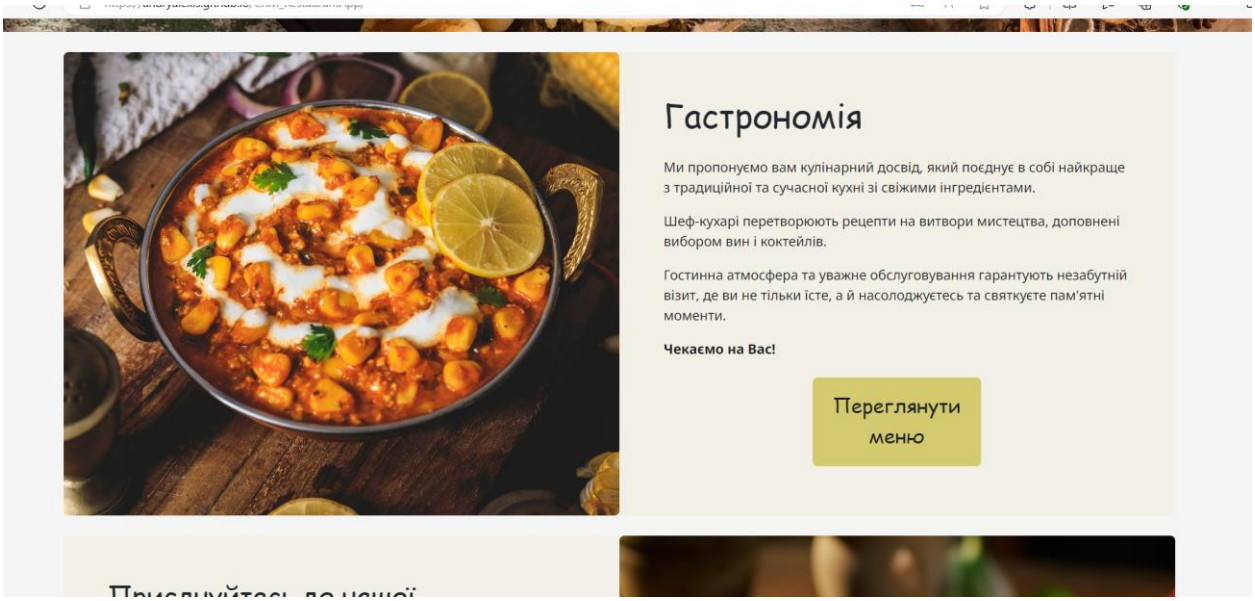

Рисунок 3.24 – Опис та кнопка переходу до меню страв

А також на головній стрінці можна отримати можливість реєтрації рис. 3.25.

Рисунок 3.25 – Перехід до реєстрації

Після натискання кнопки «Меню» користувач потрапляє на нову сторінку з меню страв рис.3.26

Рисунок 3.26 – Сторінка Меню страв

Якщо користувач забажає він може зареєструватися. Для цього йому треба перейти на сторінку Логіну та Реєстрації рис. 3.28.

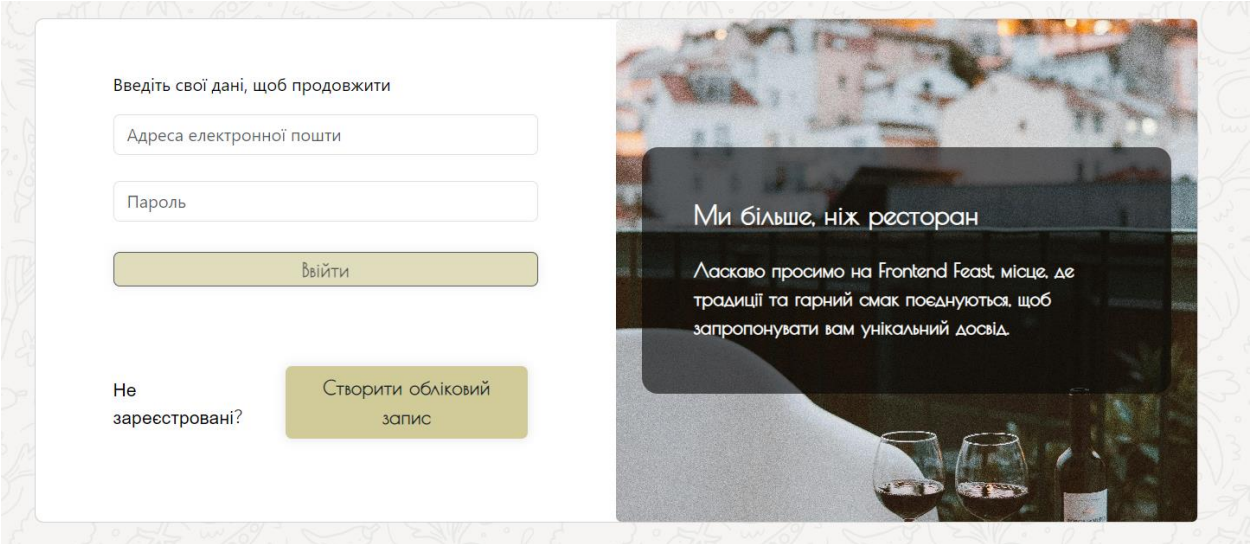

Рисунок 3.27 – Сторінка логіну та реєстрації

3.5 Висновок до третього розділу

У третьому розділі було представлено процес розробки веб-додатку для ресторанної CRM-системи. Особливу увагу приділено архітектурному патерну SDUI (Server-Driven UI), який забезпечує централізоване управління інтерфейсом користувача з боку сервера. Це рішення дозволяє динамічно змінювати компоненти інтерфейсу без необхідності оновлення клієнтської частини, що підвищує гнучкість та зручність внесення змін.

Розроблена система включає ключові елементи, такі як серверна частина для обробки запитів та управління даними, клієнтську частину, побудовану на Angular.js, та використання TypeScript для підвищення надійності коду. Важливими компонентами стали сервіси для управління меню, замовленнями та обробки автентифікації користувачів, що забезпечують стабільну роботу системи.

Таким чином, третій розділ демонструє завершення етапу розробки веб-додатку з використанням SDUI-патерну, що відповідає вимогам сучасних CRM-систем для ресторанного бізнесу.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи на тему «Розробка CRM-системи для ресторанного бізнесу» було проведено комплексне дослідження предметної області, вибір оптимальних архітектурних та програмних рішень, а також практичну реалізацію веб-додатку для автоматизації управління клієнтськими взаємовідносинами. Робота включала теоретичний аналіз сучасних CRM-систем, обґрунтування вибору патерну SDUI-MVVM, а також практичну розробку та тестування програмного продукту.

Основними завданнями, які були вирішені під час виконання роботи, стали:

1. Проведення аналізу існуючих CRM-систем, їх функціональних можливостей та архітектурних підходів.
2. Обґрунтування вибору патерну SDUI-MVVM як основи для побудови системи з централізованим управлінням інтерфейсом.
3. Розробка архітектури веб-додатку з використанням Angular для клієнтської частини та Node.js для серверної.
4. Програмна реалізація системи, включаючи ключові модулі управління меню, замовленнями та клієнтами.
5. Тестування CRM-системи для перевірки її відповідності функціональним та нефункціональним вимогам, зокрема стабільності, безпеки та зручності використання.

У ході роботи було здійснено ґрунтовний аналіз технологій та інструментів для розробки веб-додатків, включаючи Angular, Node.js, TypeScript та JWT для автентифікації. Обґрунтовано використання патерну SDUI-MVVM, що дозволив досягти централізованого управління інтерфейсом користувача та спрощення оновлення системи без необхідності зміни клієнтської частини.

Особливу увагу було приділено принципам модульності, масштабованості та забезпечення безпеки обробки даних. Система успішно

пройшла тестування, показавши стабільність у роботі та відповідність заявленим вимогам.

Таким чином, поставлена мета щодо розробки CRM-системи для ресторанного бізнесу була досягнута, а розроблений програмний продукт може бути використаний для автоматизації бізнес-процесів ресторанів, підвищення ефективності управління замовленнями та покращення якості обслуговування клієнтів. Отримані результати можуть бути основою для подальших досліджень у сфері автоматизації бізнесу та розвитку функціональності CRM-систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How to reach a million customers in one year – [Електронний ресурс] – Режим доступу: https://www.wamda.com/2016/03/how-to-reach-million-customers-in-one-yeareat#!.

2. What is CRM system? Full Guide: Definition & Benefits/ Creatio – [Електронний ресурс] – Режим доступу: https://www.creatio.com/page/what-is-crm

3. Vandermolen F. Chapter 4: Requirements - Writing Requirements / Fulton Vandermolen // Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA / Fulton Vandermolen., 2018. – (ISBN 9781351831420). – С. 89-93.

4. Teorey, T.; Lightstone, S. and Nadeau, T.(2005) Database Modeling & Design: Logical Design, 4th edition, Morgan Kaufmann Press.

5. Creatio CRM. Architecture and principles. – [Електронний ресурс] – Режим доступу: https://www.creatio.com/our-technologies/architecture-and-principles p.

6. Web Application Architecture: How the Web Works – [Електронний ресурс] – Режим доступу: https://www.altexsoft.com/blog/engineering/web-application-architecture-howthe-web-works/.

7. KeepinCRM – [Електронний ресурс] – Режим доступу: https://keepincrm.com.

8. Sales Creatio – [Електронний ресурс] – Режим доступу: https://www.terrasoft.ru/sales.

9. What is Angular? – [Електронний ресурс] – Режим доступу: https://angular.dev/overview

10. JSON – [Електронний ресурс] – Режим доступу: http://surl.li/uvjkba

11. What is Node.js (Node)? – [Електронний ресурс] – Режим доступу: https://www.techtarget.com/whatis/definition/Nodejs#:~:text=js%20(Node)%20is%20an%20Open,to%20learn%20an%20additional%20language..

12. MVC Framework Introduction – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-framework-introduction/

13. Introduction to MVVM Architecture – [Електронний ресурс] – Режим доступу: http://surl.li/yjbvta

14. Understanding VIPER Architecture – [Електронний ресурс] – Режим доступу: http://surl.li/prptzq

15. SDUI-MVVM – [Електронний ресурс] – Режим доступу: http://surl.li/ibwyzm