НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система обліку процесів для власника ресторану**»**

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми

д.е.н., професор _____ **Роман РУДЕНСЬКИЙ**

(підпис)

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ **Ірина БОРОДКІНА**

(підпис)

Виконав _____ **Богдан КИЩЕНКО**

(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Кищенку Богдану Андрійовичу

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи «Інформаційна система обліку процесів для власника ресторану» затверджена наказом від «06» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Аналіз предметної області ресторанного бізнесу; процеси обліку продажів; облік складських операцій; облік персоналу; фінансовий облік; формування звітності та аналітики; вимоги до автоматизації управлінських процесів ресторану; технології розробки інформаційних систем; сучасні засоби зберігання та обробки даних.
Перелік питань, що підлягають дослідженню:

- Аналіз предметної області та визначення функціональних вимог до інформаційної системи обліку процесів ресторану.
- Проєктування структури інформаційної системи та моделі даних.
- Розробка програмного забезпечення для автоматизації обліку продажів, складських операцій, персоналу та фінансів.
- Реалізація механізмів формування звітності та аналітичної інформації для власника ресторану.
- Тестування та оцінка ефективності функціонування розробленої системи

Дата видачі завдання «06» січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Ірина БОРОДКІНА**

Завдання взяв до виконання _____ (підпис) **Богдан КИЩЕНКО**

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних
СУБД — система управління базами даних
ІС — інформаційна система
ППЗ — прикладне програмне забезпечення
АРМ — автоматизоване робоче місце
UML — Unified Modeling Language, уніфікована мова моделювання
ER-діаграма — діаграма «сутність-зв’язок»
PK — Primary Key, первинний ключ таблиці
FK — Foreign Key, зовнішній ключ таблиці
SQL — Structured Query Language, мова структурованих запитів
CRUD — операції створення, читання, оновлення та видалення даних
ORM — Object-Relational Mapping, об’єктно-реляційне відображення
LINQ — Language Integrated Query, засіб формування запитів у C#
MVC — Model–View–Controller, архітектурний шаблон побудови вебзастосунку
ASP.NET Core MVC — технологія для розробки серверної частини вебзастосунку
EF Core — Entity Framework Core, ORM-засіб для роботи з базою даних
HTML — мова розмітки вебсторінок
CSS — мова опису стилів вебсторінок
JS — JavaScript, мова програмування для інтерактивності вебінтерфейсу
UI — User Interface, користувацький інтерфейс
SSMS — SQL Server Management Studio
AuditLogs — журнал дій користувачів системи

Вступ

Ефективність роботи сучасного ресторану значною мірою залежить від швидкості обробки інформації, організації внутрішніх процесів та контролю діяльності персоналу [6]. У процесі функціонування ресторанного закладу щоденно виконується велика кількість взаємопов’язаних операцій, серед яких прийом і супровід замовлень, контроль оплат, ведення меню, бронювання столиків, облік складських запасів та формування звітної інформації. За відсутності єдиної системи управління ці процеси часто виконуються вручну або за допомогою декількох окремих програмних рішень, що ускладнює роботу працівників, збільшує кількість помилок та негативно впливає на ефективність управління закладом.

Однією з основних проблем ресторанного бізнесу є складність централізованого контролю за діяльністю закладу. Власник ресторану повинен мати можливість оперативно отримувати інформацію про фінансові показники, стан замовлень, ефективність роботи персоналу, складські витрати та загальний стан діяльності ресторану. Водночас працівники різних категорій повинні мати доступ лише до тих функцій системи, які необхідні для виконання їхніх посадових обов’язків. Саме тому автоматизація внутрішніх процесів ресторану є важливим напрямом розвитку сучасних закладів громадського харчування [8].

Використання інформаційної системи дозволяє значно спростити організацію роботи ресторану та забезпечити централізоване управління основними процесами [1; 6]. Автоматизація дозволяє покращити контроль за замовленнями та оплатами, оптимізувати ведення складського обліку, забезпечити ефективне управління бронюванням столиків та спростити процес формування аналітичної інформації. Крім цього, використання сучасної системи управління сприяє підвищенню швидкості роботи персоналу та покращенню якості обслуговування клієнтів.

Актуальність теми бакалаврської кваліфікаційної роботи обумовлена необхідністю створення сучасної інформаційної системи, яка забезпечує автоматизацію процесів діяльності ресторану та підтримку прийняття управлінських рішень власником закладу. Впровадження такої системи дозволяє зменшити навантаження на працівників, мінімізувати ризик виникнення помилок та забезпечити більш ефективний контроль за роботою ресторану [8; 20].

Об’єктом дослідження є процеси управління діяльністю ресторану, пов’язані з обліком замовлень, оплат, бронюванням столиків, управлінням персоналом та складськими операціями. Предметом дослідження є методи та засоби автоматизації процесів ресторанного бізнесу із використанням сучасних інформаційних технологій.

Метою бакалаврської кваліфікаційної роботи є розробка інформаційної системи обліку процесів для власника ресторану, яка забезпечує централізоване управління діяльністю закладу, автоматизацію основних операцій та формування аналітичної інформації для підтримки прийняття управлінських рішень.

У процесі виконання роботи було проведено аналіз особливостей функціонування ресторанного бізнесу та визначено основні вимоги до інформаційної системи. Значну увагу приділено організації взаємодії між різними категоріями користувачів системи, оскільки робота офіціантів, кухарів, адміністраторів та власника ресторану тісно пов’язана між собою. Для кожної ролі визначено окремий набір функціональних можливостей відповідно до посадових обов’язків працівників.

У межах роботи реалізовано механізми управління меню та категоріями страв, створення і супроводу замовлень, контролю оплат, управління складськими запасами та бронювання столиків. Також у системі реалізовано модуль аналітики та звітності, який дозволяє власнику ресторану переглядати

інформацію про фінансові показники, ефективність роботи персоналу, популярність страв та результати діяльності закладу.

Для розробки інформаційної системи використано сучасні технології веброзробки та управління даними [2; 4; 5]. Серверна частина системи реалізована із використанням ASP.NET Core MVC, що забезпечує обробку запитів користувачів, виконання бізнес-логіки та взаємодію з базою даних [2; 4]. Для зберігання інформації використовується система керування базами даних Microsoft SQL Server, а взаємодія із базою даних реалізована за допомогою Entity Framework Core [5; 7]. Клієнтська частина системи створена із використанням HTML, CSS, Bootstrap та JavaScript, що дозволило сформувати сучасний та зручний вебінтерфейс користувача [21; 22; 23].

Результати бакалаврської кваліфікаційної роботи були апробовані у вигляді тез на тему «Інформаційна система обліку процесів для власника ресторану». У межах апробації було представлено основні результати дослідження, функціональні можливості розробленої системи та особливості автоматизації процесів управління ресторанним закладом.

Практичне значення роботи полягає у створенні інформаційної системи, яка може бути використана для автоматизації діяльності ресторану, покращення контролю внутрішніх процесів та підвищення ефективності управління закладом. Використання системи дозволяє оптимізувати роботу персоналу, забезпечити централізований облік інформації та підвищити якість обслуговування клієнтів [8; 20].

Пояснювальна записка бакалаврської кваліфікаційної роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено системний аналіз предметної області ресторанного бізнесу, виконано постановку завдання, розглянуто існуючі рішення та здійснено моделювання предметної області. Другий розділ присвячений інформаційному забезпеченню системи, зокрема розробці логічної моделі даних, вибору системи управління інформаційною базою та створенню інформаційної бази системи. У

третьому розділі описано прикладне програмне забезпечення системи, організаційну структуру програмного забезпечення, вибір інструментальних засобів розробки, а також алгоритмізацію та програмування програмних модулів. Четвертий розділ містить результати тестування системи, вимоги до апаратного та програмного забезпечення, а також рекомендації щодо впровадження та експлуатації системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Діяльність сучасного ресторану пов’язана з постійною обробкою значної кількості інформації, що формується під час роботи закладу. До такої інформації належать дані про працівників, меню, категорії страв, замовлення, оплати, бронювання столиків, складські залишки та звітність. У разі відсутності єдиної інформаційної системи значна частина цих процесів виконується вручну або за допомогою декількох окремих програмних засобів, що ускладнює контроль діяльності ресторану та знижує ефективність управління [8].

Основним завданням розроблюваної інформаційної системи є автоматизація обліку процесів ресторану та забезпечення власника закладу актуальною інформацією для прийняття управлінських рішень [1; 6]. Система повинна забезпечувати централізоване зберігання інформації, підтримку роботи користувачів різних ролей та формування звітно-аналітичної інформації.

У системі необхідно зберігати інформацію про працівників ресторану, ролі користувачів, меню, страви, інгредієнти, замовлення, оплати, бронювання та складські залишки [1; 3]. Для кожного користувача повинні зберігатися персональні дані, роль у системі та статус облікового запису. Рольова модель доступу має забезпечувати розмежування функціональних можливостей між власником ресторану, адміністратором, офіціантом та кухарем [4; 16].

Важливою частиною системи є автоматизація роботи із замовленнями [1]. Система повинна дозволяти створювати замовлення, додавати страви, змінювати статуси позицій та проводити оплату. Для кухаря система повинна відображати перелік страв, які необхідно приготувати, а для офіціанта — забезпечувати контроль поточного стану замовлень.

Окремої автоматизації потребує процес бронювання столиків [1]. Система повинна зберігати інформацію про клієнта, дату та час бронювання, кількість

гостей і статус бронювання. Доступність столиків має визначатися з урахуванням поточного стану столика та наявних бронювань.

Для контролю ресурсів ресторану система повинна підтримувати ведення складського обліку. Необхідно забезпечити зберігання інформації про інгредієнти, залишки продуктів, закупівлі та списання, а також зв’язок між складськими позиціями та стравами меню [1; 3].

До операцій, які необхідно автоматизувати, належать створення та редагування замовлень, проведення оплат, управління меню, формування бронювань, ведення складського обліку, управління працівниками та формування звітів. Автоматизація цих процесів дозволяє зменшити кількість ручних операцій та підвищити ефективність роботи ресторану [20].

Результатом роботи системи повинні бути звітні та аналітичні документи, які містять інформацію про замовлення, оплати, бронювання, складські залишки, персонал та результати діяльності ресторану. Адміністратор повинен мати можливість формувати звіти, а власник ресторану — переглядати аналітичну інформацію та використовувати її для оцінки ефективності діяльності закладу.

Розроблювана система належить до класу інформаційних систем управління [6]. За характером використання інформації система є обліково-аналітичною, а за ступенем автоматизації — частково автоматизованою [6]. Система призначена для використання в межах окремого ресторанного закладу.

До розроблюваної системи висуваються вимоги щодо надійності зберігання даних, швидкості обробки інформації, зручності користувацького інтерфейсу, розмежування доступу користувачів та можливості подальшого розширення функціональних можливостей [20].

1.2 Огляд інформаційних джерел та існуючих рішень

Для автоматизації роботи ресторанів сьогодні використовується велика кількість інформаційних систем, які допомагають вести облік замовлень, контролювати оплату, організовувати роботу персоналу та формувати звітність. Аналіз існуючих рішень дозволяє визначити основні підходи до автоматизації ресторанного бізнесу, а також виявити переваги й недоліки сучасних програмних продуктів [6; 8].

Одним із найбільш популярних рішень для автоматизації ресторанного бізнесу є система Poster POS. Poster POS — це хмарна система автоматизації ресторанів, кафе та закладів громадського харчування, яка забезпечує можливість ведення меню, створення замовлень, контролю оплат, складського обліку та формування аналітичної інформації.

Основними функціями системи Poster POS є:

– створення та супровід замовлень;

– ведення меню та категорій страв;

– контроль оплат;

– підтримка складського обліку;

– формування аналітики та звітів;

– підтримка роботи офіціантів та адміністраторів;

– можливість роботи через вебінтерфейс та мобільні пристрої.

До переваг системи можна віднести:

– сучасний та зручний інтерфейс користувача;

– підтримку хмарного зберігання даних;

– можливість віддаленого доступу до системи;

– підтримку аналітики та звітності;

– інтеграцію з касовим обладнанням.

Серед недоліків системи можна виділити:

– залежність від інтернет-з’єднання;

– обмежені можливості адаптації під окремий заклад;

– необхідність використання платної підписки;

– частину функцій доступно лише у платних тарифах.

Ще одним поширеним рішенням є система Syrve. Дана система використовується для автоматизації роботи ресторанів, кафе та мереж закладів громадського харчування. Syrve підтримує роботу із замовленнями, складським обліком, персоналом та фінансовою звітністю.

Основними функціями системи Syrve є:

– управління замовленнями;

– ведення складського обліку;

– контроль роботи персоналу;

– формування фінансової звітності;

– підтримка системи ролей користувачів;

– контроль продажів та аналітика.

До переваг системи належать:

– широкий функціонал;

– підтримка великої кількості користувачів;

– наявність інструментів аналітики;

– підтримка роботи мережі закладів;

– можливість інтеграції з додатковими сервісами.

Недоліками системи є:

– складність налаштування;

– висока вартість використання;

– значна кількість функцій, які можуть бути надлишковими для невеликих ресторанів;

– складніший інтерфейс користувача порівняно з іншими системами.

Також було розглянуто систему r_keeper, яка є одним із найбільш відомих рішень для автоматизації ресторанного бізнесу. Система забезпечує

автоматизацію процесів обслуговування клієнтів, ведення замовлень, контроль оплат та управління складськими операціями.

Основними функціями системи r_keeper є:

– ведення замовлень;

– підтримка роботи касових операцій;

– управління меню;

– складський облік;

– підтримка аналітики та звітності;

– інтеграція з POS-обладнанням.

До переваг системи можна віднести:

– стабільність роботи;

– підтримку інтеграції з касовим обладнанням;

– широкі можливості налаштування;

– підтримку роботи великих ресторанних закладів.

Серед недоліків системи можна виділити:

– складність інтерфейсу;

– високу вартість впровадження;

– необхідність навчання персоналу;

– складність адміністрування системи.

Окрему увагу було приділено веборієнтованим системам управління рестораном. Використання вебтехнологій дозволяє забезпечити централізований доступ до інформації через браузер, спростити адміністрування системи та забезпечити швидке оновлення даних. Крім цього, веборієнтовані системи не потребують встановлення окремого програмного забезпечення на кожному робочому місці.

Проведений аналіз показав, що сучасні системи автоматизації ресторанного бізнесу забезпечують широкий функціонал для управління закладом, проте більшість із них мають недоліки, пов’язані зі складністю використання, високою вартістю або надлишковим функціоналом для

невеликих ресторанів. Саме тому виникає потреба у створенні інформаційної системи, яка забезпечить зручне управління замовленнями, бронюванням столиків, складським обліком, персоналом та аналітикою відповідно до потреб конкретного ресторанного закладу

1.3 Моделювання предметної області

Для опису роботи інформаційної системи обліку процесів ресторану було використано комплекс моделей та UML-діаграм [14], які дозволяють відобразити структуру системи, логіку взаємодії користувачів із програмним забезпеченням, а також основні бізнес-процеси ресторанного закладу. Побудовані моделі забезпечують наочне представлення функціонування системи та спрощують процес її подальшого проєктування й реалізації.

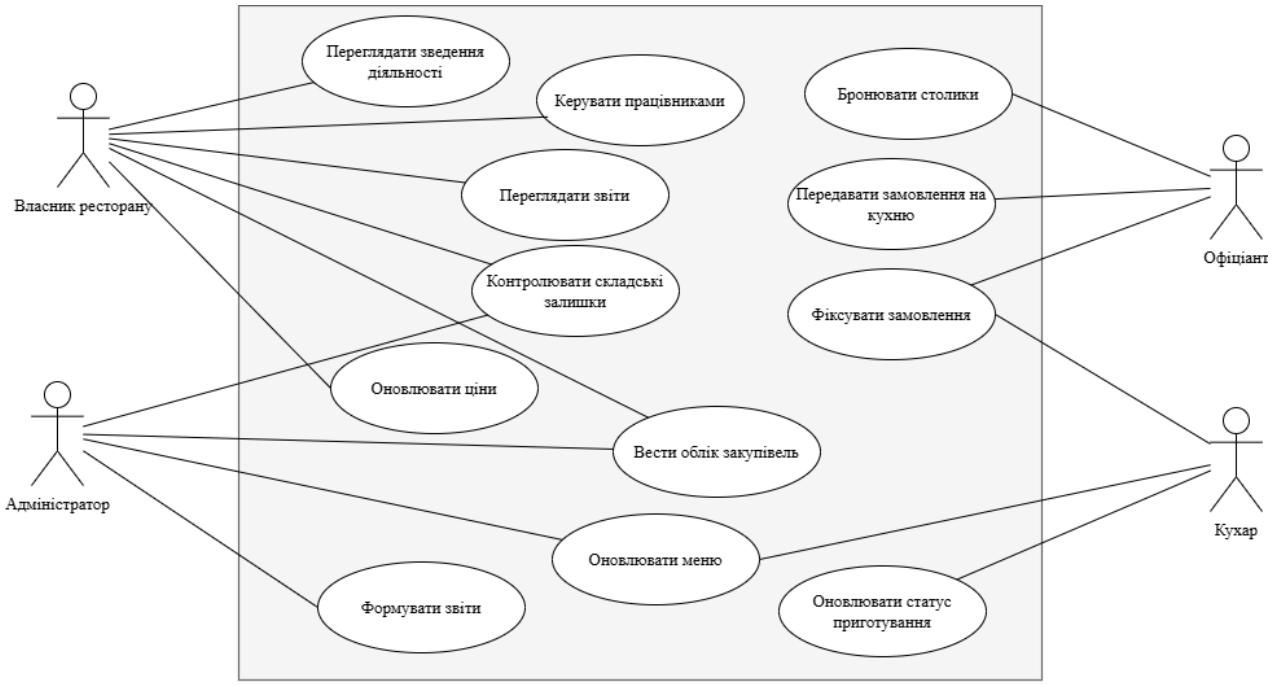


Рис. 1 Діаграма прецедентів

Діаграма прецедентів відображає взаємодію користувачів із інформаційною системою ресторану та основні функції кожної ролі. У системі

передбачено чотири типи користувачів: власник ресторану, адміністратор, офіціант та кухар [14].

Власник ресторану має доступ до перегляду аналітики, звітів, управління працівниками та контролю складських залишків. Адміністратор відповідає за оновлення меню, цін, ведення закупівель та формування звітності. Офіціант здійснює бронювання столиків, створення та передачу замовлень на кухню. Кухар переглядає замовлення та оновлює статус приготування страв [14; 16].

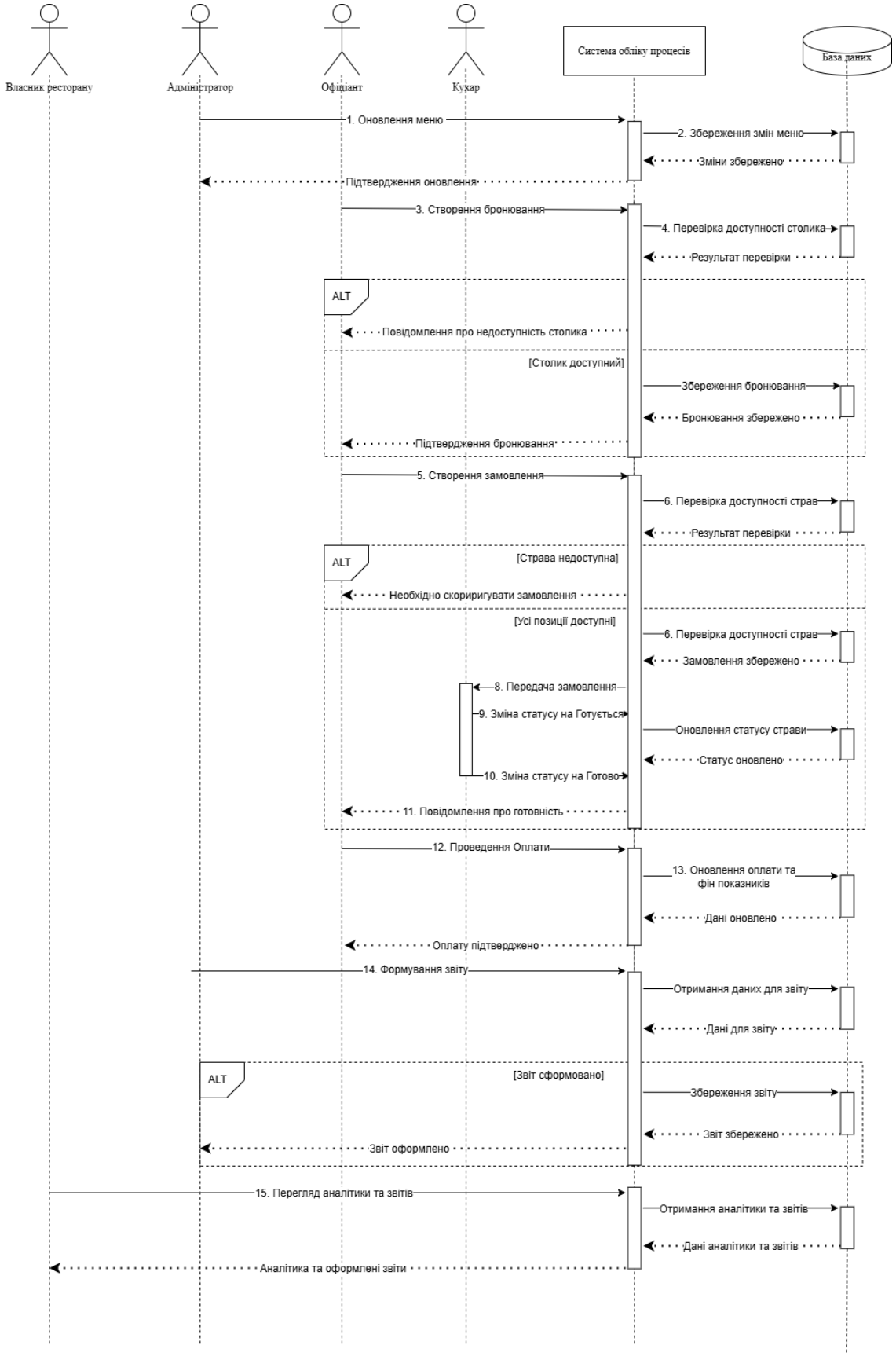


Рис. 2 Діаграма послідовності

Діаграма послідовності відображає процес взаємодії користувачів із системою обліку процесів ресторану під час бронювання столиків, створення замовлень, приготування страв, проведення оплати та формування звітності [14]. На діаграмі показано обмін повідомленнями між офіціантом, адміністратором, кухарем, власником ресторану, системою обліку процесів та базою даних. Також відображено перевірку доступності столиків і страв, оновлення статусів замовлення, збереження даних у базі даних та перегляд аналітичної інформації власником ресторану.

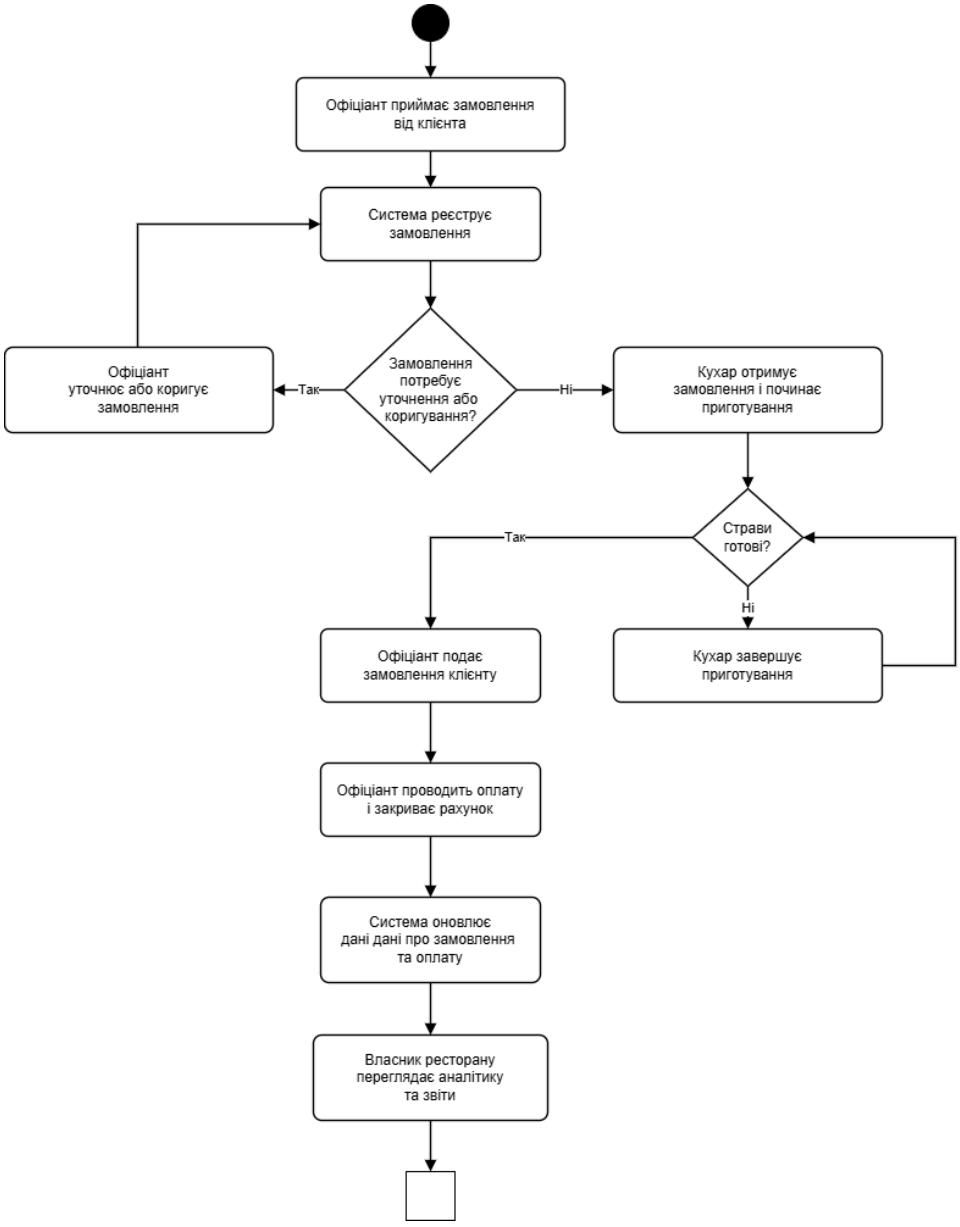


Рис. 3 Діаграма активності

Діаграма активності відображає основний процес роботи інформаційної системи ресторану під час обробки замовлення клієнта. Процес починається з прийняття офіціантом замовлення та його реєстрації у системі. Після цього виконується перевірка необхідності уточнення або коригування замовлення. У разі потреби офіціант вносить зміни до замовлення, після чого система повторно обробляє дані [14].

Якщо замовлення не потребує коригування, воно передається кухарю для приготування. Далі система контролює стан готовності страв. Після завершення приготування офіціант подає замовлення клієнту, проводить оплату та закриває рахунок. На завершальному етапі система оновлює інформацію про замовлення та оплату, а власник ресторану отримує можливість переглядати аналітику та сформовані звіти щодо діяльності закладу [14].

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

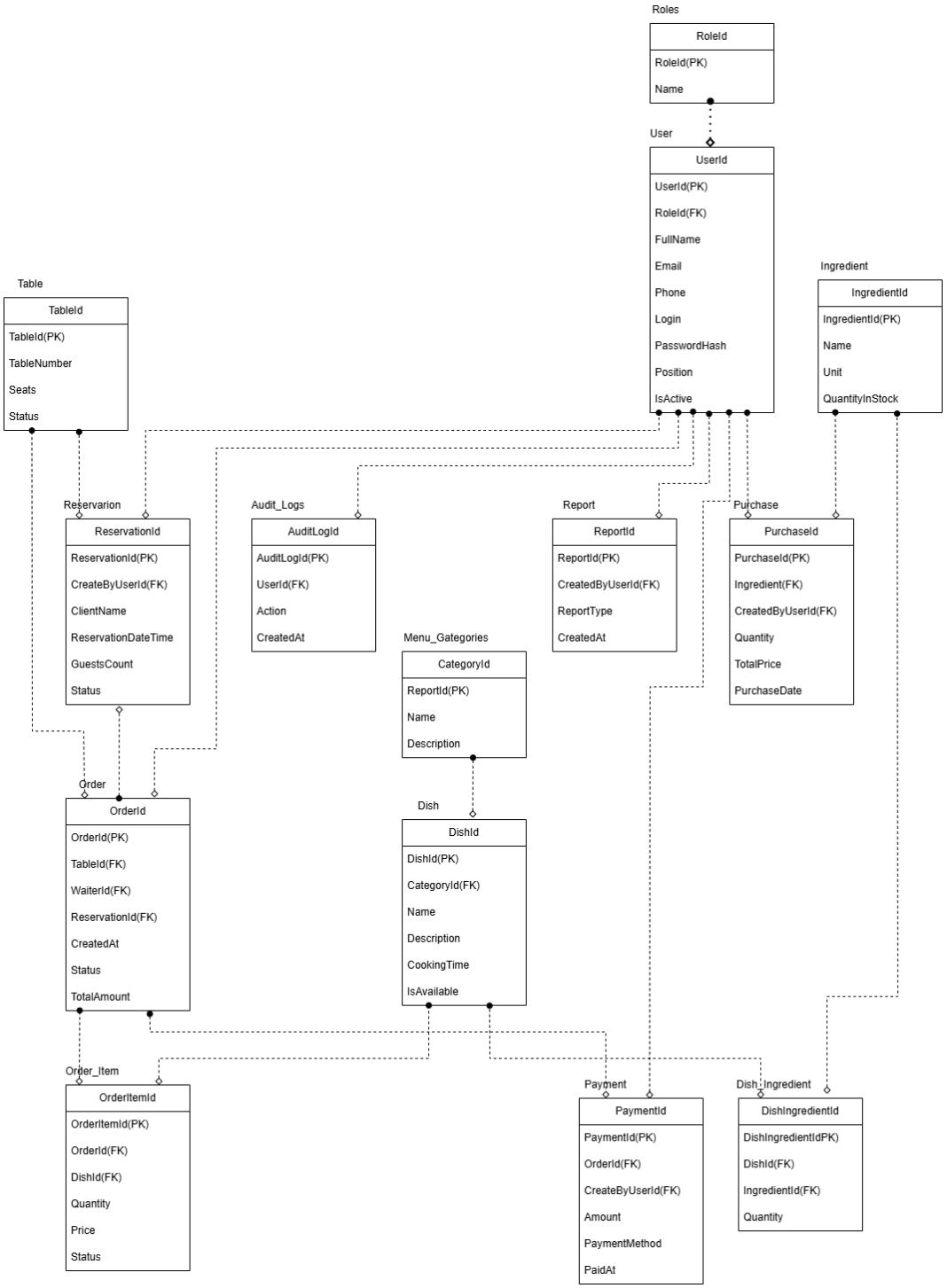


Рис. 4 ER-діаграма

На етапі проєктування інформаційного забезпечення була створена логічна модель даних. Логічна модель представлена у вигляді ER-діаграми [1; 9], яка описує основні сутності системи, їх атрибути та зв’язки між ними. Побудована модель дозволяє формалізувати структуру бази даних та визначити взаємодію між окремими компонентами системи [1; 3].

Основною сутністю моделі є таблиця User, у якій зберігається інформація про користувачів системи: адміністраторів, офіціантів та власника ресторану. Для розмежування прав доступу використовується таблиця Roles, яка пов’язана з таблицею користувачів зв’язком один-до-багатьох. Кожен користувач має певну роль у системі [1; 3; 16].

Для організації роботи із замовленнями у системі використовуються таблиці Order, Order_Item, Table, Reservation та Payment. Таблиця Order містить основну інформацію про замовлення, його статус, дату створення та загальну суму. Таблиця Order_Item використовується для зберігання окремих позицій замовлення та забезпечує зв’язок між замовленнями і стравами. Таблиця Payment містить інформацію про оплату замовлень.

Облік меню реалізований за допомогою таблиць Menu_Categories, Dish, Ingredient та Dish_Ingredient. Таблиця Dish містить інформацію про страви ресторану, а таблиця Menu_Categories використовується для поділу страв за категоріями меню. Для реалізації зв’язку між стравами та інгредієнтами використовується проміжна таблиця Dish_Ingredient, що дозволяє реалізувати зв’язок типу багато-до-багатьох.

Для ведення обліку закупівель використовується таблиця Purchase, у якій зберігаються дані про закуплені інгредієнти, їх кількість та вартість. Таблиці Audit_Logs та Report призначені для ведення журналу дій користувачів та формування звітної інформації [18].

Побудована модель даних є реляційною, оскільки інформація представлена у вигляді взаємопов’язаних таблиць, між якими встановлено зв’язки за допомогою первинних та зовнішніх ключів. У кожній таблиці

визначено первинний ключ (PK), який забезпечує унікальність записів, а зовнішні ключі (FK) використовуються для встановлення зв’язків між сутностями.

Модель відповідає вимогам третьої нормальної форми, оскільки:

— кожна таблиця описує окрему сутність;
— усунуто дублювання даних;
— неключові атрибути залежать лише від первинного ключа;
— зв’язки багато-до-багатьох винесені в окремі проміжні таблиці.

2.2 Вибір системи управління інформаційною базою

Після побудови логічної моделі даних та приведення її до третьої нормальної форми було виконано вибір системи управління базою даних для реалізації інформаційної системи обліку процесів ресторану. Оскільки структура даних є реляційною та складається із взаємопов’язаних таблиць, для реалізації інформаційної бази було обрано реляційну систему управління базами даних Microsoft SQL Server.

Microsoft SQL Server є однією з найбільш поширених реляційних СУБД та широко використовується для створення корпоративних інформаційних систем. Дана система забезпечує надійне зберігання даних, підтримку багатокористувацького режиму роботи та ефективну обробку SQL-запитів [5].

Вибір Microsoft SQL Server обумовлений особливостями розроблюваної системи, оскільки інформаційна система ресторану повинна забезпечувати:

— зберігання інформації про користувачів, ролі, замовлення, бронювання, оплату, меню та складські запаси;
— підтримку одночасної роботи декількох користувачів;
— контроль цілісності даних;
— швидке виконання операцій додавання, пошуку та оновлення інформації;

— формування звітів та аналітичної інформації;

— захист даних та розмежування прав доступу.

Основними перевагами Microsoft SQL Server, які були використані під час розробки системи, є:

— підтримка реляційної моделі даних та третьої нормальної форми;

— можливість використання первинних та зовнішніх ключів для реалізації зв’язків між таблицями;

— підтримка багатокористувацького режиму роботи;

— підтримка механізму транзакцій для забезпечення цілісності інформації;

— висока швидкість виконання SQL-запитів;

— підтримка індексування для оптимізації пошуку даних;

— можливість резервного копіювання та відновлення бази даних;

— підтримка засобів адміністрування та моніторингу;

— інтеграція з ASP.NET Core та Entity Framework Core;

— підтримка ролей та механізмів розмежування доступу користувачів [5; 25].

Використання первинних та зовнішніх ключів дозволило реалізувати зв’язки між таблицями та забезпечити цілісність даних у системі. Наприклад, таблиці замовлень, оплат, бронювань та користувачів пов’язані між собою за допомогою зовнішніх ключів, що дозволяє уникнути дублювання інформації та забезпечити правильність збереження даних.

Підтримка багатокористувацького режиму роботи є важливою для інформаційної системи ресторану, оскільки в системі одночасно можуть працювати адміністратор, офіціанти, кухарі та власник ресторану. Microsoft SQL Server дозволяє коректно обробляти одночасні запити користувачів без втрати або пошкодження інформації.

Механізм транзакцій дозволяє забезпечити надійність виконання операцій у системі [3; 5]. Наприклад, під час створення замовлення одночасно

виконуються операції додавання замовлення, створення позицій замовлення та оновлення статусу столика. У випадку помилки всі зміни можуть бути скасовані, що забезпечує цілісність даних.

Підтримка індексування дозволила підвищити швидкість пошуку інформації про замовлення, бронювання, оплату та складські запаси. Це особливо важливо для забезпечення швидкої роботи системи під час активної роботи ресторану.

Для взаємодії програмної частини системи з базою даних використовується технологія Entity Framework Core, яка забезпечує роботу з базою даних за допомогою ORM-моделі. Це дозволило спростити виконання CRUD-операцій та прискорити процес розробки програмного забезпечення.

Для адміністрування інформаційної бази використовується SQL Server Management Studio (SSMS), що забезпечує зручне створення таблиць, налаштування зв’язків, виконання SQL-запитів та контроль структури бази даних [12].

Оскільки розроблена модель даних відповідає вимогам реляційності та третій нормальній формі, використання документоорієнтованих, графових або векторних баз даних для даного проєкту є недоцільним. Структура системи має чітко визначені сутності та взаємозв’язки між ними, тому реляційна СУБД є найбільш ефективним рішенням для реалізації інформаційної системи обліку процесів ресторану.

2.3 Створення інформаційної бази

Інформаційна база інформаційної системи обліку процесів ресторану реалізована у вигляді централізованої бази даних [1; 3]. Усі дані системи зберігаються в єдиній базі даних Microsoft SQL Server, до якої підключаються користувачі системи через програмний додаток [5]. Такий підхід дозволяє

забезпечити цілісність інформації, спростити адміністрування системи та централізовано контролювати доступ до даних.

Структурно система складається з клієнтської частини та центральної бази даних. Користувачі системи працюють через вебінтерфейс або програмний модуль, який взаємодіє з сервером бази даних. Усі операції додавання, зміни, видалення та пошуку інформації виконуються безпосередньо через централізовану базу даних.

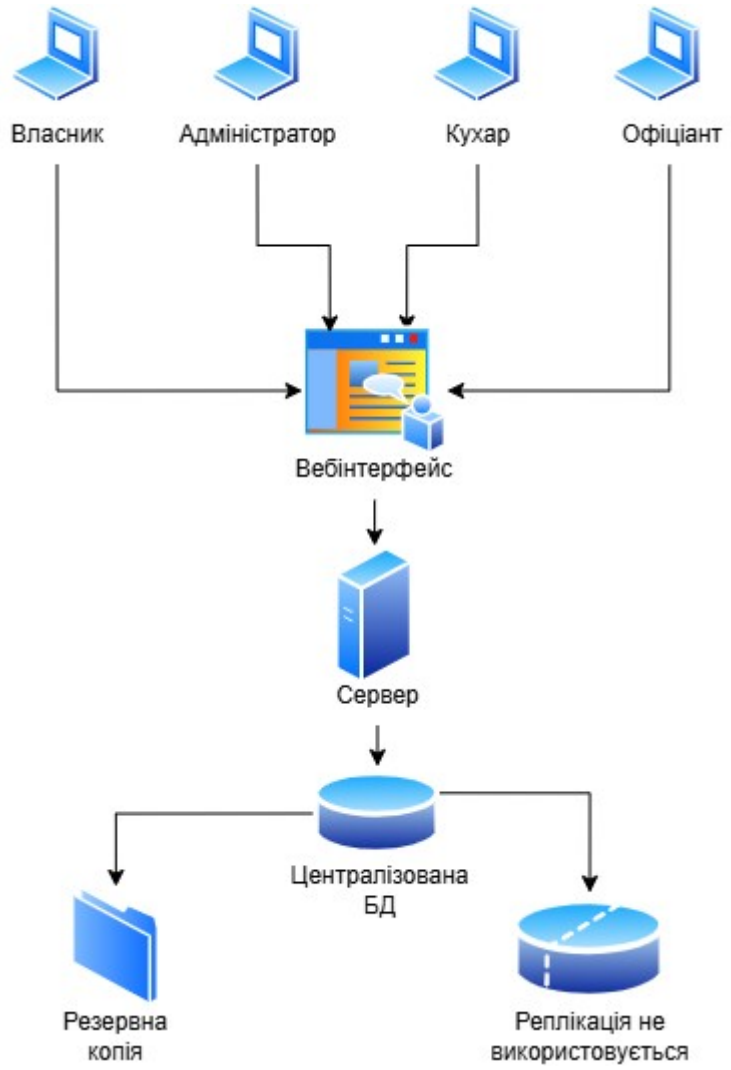


Рис. 5 Структурна схема системи

Інформаційна база побудована за реляційним принципом і включає таблиці користувачів, ролей, меню, категорій страв, інгредієнтів, замовлень, бронювань, оплат, закупівель, звітів та журналу дій користувачів. Для

забезпечення цілісності даних між таблицями реалізовані зв’язки за допомогою первинних та зовнішніх ключів.

До основних таблиць інформаційної бази належать:

— Users — зберігання інформації про користувачів системи;
— Roles — ролі користувачів;
— MenuCategories — категорії меню;
— MenuItems — страви ресторану;
— Ingredients — інгредієнти;
— Orders — замовлення;
— OrderItems — позиції замовлень;
— Reservations — бронювання столиків;
— Payments — оплати;
— AuditLogs — журнал дій користувачів;
— GeneratedReports — сформовані звіти.

Приклад створення таблиці з первинним ключем:

```
CREATE TABLE dbo.Roles
(
    RoleId INT IDENTITY(1,1) NOT NULL,
    Name NVARCHAR(50) NOT NULL,
    Description NVARCHAR(255) NULL,
    CONSTRAINT PK_Roles PRIMARY KEY (RoleId),
    CONSTRAINT UQ_Roles_Name UNIQUE (Name)
);
GO
```

Рис. 6 Створення таблиці Roles

У наведеному прикладі таблиця Roles використовується для збереження ролей користувачів системи, а поле RoleId є первинним ключем таблиці.

Приклад створення таблиці із зовнішнім ключем:

```sql
USE RestaurantManagementDB;
GO
CREATE TABLE dbo.Users
(
    UserId INT IDENTITY(1,1) NOT NULL,
    Username NVARCHAR(50) NOT NULL,
    PasswordHash NVARCHAR(255) NOT NULL,
    FullName NVARCHAR(150) NOT NULL,
    Email NVARCHAR(150) NOT NULL,
    Phone NVARCHAR(20) NULL,
    RoleId INT NOT NULL,
    IsActive BIT NOT NULL CONSTRAINT DF_Users_IsActive DEFAULT (1),
    CreatedAt DATETIME2(0) NOT NULL CONSTRAINT DF_Users_CreatedAt DEFAULT (SYSDATETIME()),

    CONSTRAINT PK_Users PRIMARY KEY (UserId),
    CONSTRAINT UQ_Users_Username UNIQUE (Username),
    CONSTRAINT UQ_Users_Email UNIQUE (Email),
    CONSTRAINT FK_Users_Roles FOREIGN KEY (RoleId) REFERENCES dbo.Roles(RoleId)
);
GO
```

Рис. 7 Створення таблиці Users

У даному прикладі зовнішній ключ RoleId забезпечує зв’язок між таблицями Users та Roles.

Для реалізації зв’язку багато-до-багатьох між стравами та інгредієнтами використовується проміжна таблиця DishIngredients.

```sql
USE RestaurantManagementDB;
GO
CREATE TABLE dbo.DishIngredients
(
    DishIngredientId INT IDENTITY(1,1) NOT NULL,
    MenuItemId INT NOT NULL,
    IngredientId INT NOT NULL,
    QuantityNeeded DECIMAL(14,3) NOT NULL,

    CONSTRAINT PK_DishIngredients PRIMARY KEY (DishIngredientId),
    CONSTRAINT FK_DishIngredients_MenuItems FOREIGN KEY (MenuItemId) REFERENCES dbo.MenuItems(MenuItemId),
    CONSTRAINT FK_DishIngredients_Ingredients FOREIGN KEY (IngredientId) REFERENCES dbo.Ingredients(IngredientId),
    CONSTRAINT UQ_DishIngredients_MenuItem_Ingredient UNIQUE (MenuItemId, IngredientId),
    CONSTRAINT CK_DishIngredients_QuantityNeeded_Positive CHECK (QuantityNeeded > 0)
);
GO
```

Рис. 8 Створення таблиці DishIngredients

Дана таблиця дозволяє зберігати рецептуру страв та визначати кількість необхідних інгредієнтів для кожної позиції меню.

```
USE RestaurantManagementDB;
GO
CREATE TABLE dbo.Orders
(
    OrderId INT IDENTITY(1,1) NOT NULL,
    TableId INT NOT NULL,
    WaiterUserId INT NOT NULL,
    CreatedAt DATETIME2(0) NOT NULL CONSTRAINT DF_Orders_CreatedAt DEFAULT (SYSDATETIME()),
    ClosedAt DATETIME2(0) NULL,
    Status NVARCHAR(20) NOT NULL CONSTRAINT DF_Orders_Status DEFAULT (N'New'),
    Notes NVARCHAR(500) NULL,
    TotalAmount DECIMAL(14,2) NOT NULL CONSTRAINT DF_Orders_TotalAmount DEFAULT (0),

    CONSTRAINT PK_Orders PRIMARY KEY (OrderId),
    CONSTRAINT FK_Orders_DiningTables FOREIGN KEY (TableId) REFERENCES dbo.DiningTables(TableId),
    CONSTRAINT FK_Orders_Users FOREIGN KEY (WaiterUserId) REFERENCES dbo.Users(UserId),
    CONSTRAINT CK_Orders_Status CHECK (Status IN (N'New', N'InPreparation', N'Ready', N'Served', N'Closed', N'Cancelled')),
    CONSTRAINT CK_Orders_TotalAmount_NonNegative CHECK (TotalAmount >= 0),
    CONSTRAINT CK_Orders_ClosedAt CHECK (ClosedAt IS NULL OR ClosedAt >= CreatedAt)
);
GO
```

Рис. 9 Створення таблиці Orders

Таблиця Orders призначена для збереження інформації про замовлення клієнтів у системі обліку процесів ресторану. Вона містить дані про столик, офіціанта, дату створення та закриття замовлення, його статус і загальну суму. Первинний ключ OrderId забезпечує унікальну ідентифікацію кожного замовлення, а зовнішні ключі TableId та WaiterUserId реалізують зв’язок із таблицями DiningTables і Users. Для автоматизації роботи системи використано значення за замовчуванням для дати створення замовлення, початкового статусу та загальної суми. У таблиці також реалізовані обмеження CHECK, які контролюють допустимі статуси замовлення, не дозволяють від’ємні значення суми та перевіряють коректність дати закриття замовлення.

```
USE RestaurantManagementDB;
GO

CREATE VIEW dbo.vw_DailySales
AS
SELECT
    CAST(p.PaidAt AS DATE) AS SalesDate,
    COUNT(DISTINCT p.OrderId) AS PaidOrdersCount,
    SUM(p.Amount) AS TotalSalesAmount
FROM dbo.Payments p
WHERE p.Status = N'Paid'
GROUP BY CAST(p.PaidAt AS DATE);
GO
```

Рис. 10 Створення уявлення vw_DailySales

Для формування аналітичної інформації у базі даних реалізовано SQL-представлення vw_DailySales. Воно використовується для отримання щоденної статистики продажів. У представленні виконується групування оплат за датою, підраховується кількість оплачених замовлень та загальна сума продажів за кожен день. Це дозволяє спростити формування фінансової аналітики та швидко отримувати зведену інформацію про результати роботи ресторану.

```
USE RestaurantManagementDB;
GO

CREATE PROCEDURE dbo.sp_CreateOrder
    @TableId INT,
    @WaiterUserId INT,
    @Notes NVARCHAR(500) = NULL,
    @NewOrderId INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM dbo.DiningTables WHERE TableId = @TableId)
    BEGIN
        RAISERROR(N'Table not found.', 16, 1);
        RETURN;
    END;

    IF NOT EXISTS (SELECT 1 FROM dbo.Users WHERE UserId = @WaiterUserId AND IsActive = 1)
    BEGIN
        RAISERROR(N'Waiter user not found or inactive.', 16, 1);
        RETURN;
    END;

    INSERT INTO dbo.Orders (TableId, WaiterUserId, CreatedAt, Status, Notes, TotalAmount)
    VALUES (@TableId, @WaiterUserId, SYSDATETIME(), N'New', @Notes, 0);

    SET @NewOrderId = SCOPE_IDENTITY();

    UPDATE dbo.DiningTables
    SET Status = N'Occupied'
    WHERE TableId = @TableId
      AND Status IN (N'Free', N'Reserved');

    INSERT INTO dbo.AuditLogs (UserId, Action, EntityName, EntityId, CreatedAt, Details)
    VALUES
    (
        @WaiterUserId,
        N'CREATE',
        N'Order',
        @NewOrderId,
        SYSDATETIME(),
        N'Order created via sp_CreateOrder'
    );
END;
GO
```

Рис. 11 Створення процедури sp_CreateOrder

Для автоматизації процесу створення замовлення у базі даних реалізовано збережену процедуру sp_CreateOrder. Процедура приймає ідентифікатор столика, ідентифікатор офіціанта, примітку до замовлення та повертає ідентифікатор створеного замовлення. Перед додаванням запису виконується перевірка наявності столика та активного користувача-офіціанта. Після створення замовлення система змінює статус столика на «Occupied» та додає запис до журналу дій користувачів. Це дозволяє централізувати логіку створення замовлення на рівні бази даних і забезпечити контроль коректності даних.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Організаційна структура прикладного програмного забезпечення інформаційної системи обліку процесів ресторану представлена у вигляді діаграми пакетів. Система поділяється на клієнтський та серверний рівні, що забезпечує логічне розділення інтерфейсу користувача, бізнес-логіки та роботи з базою даних [2; 4; 20].

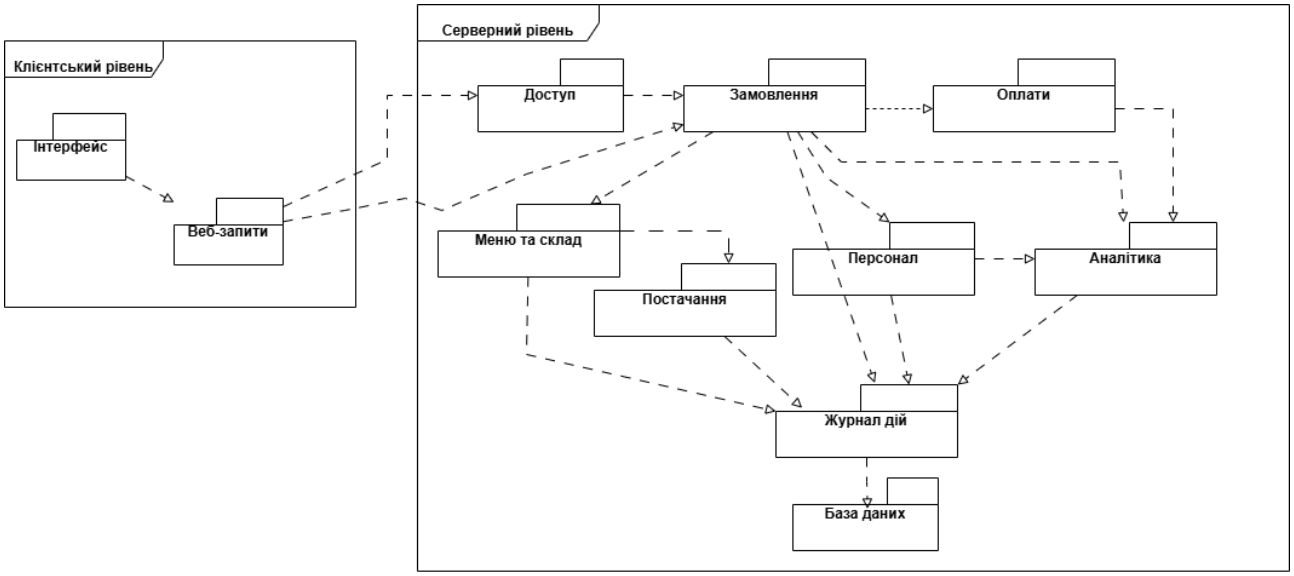


Рис. 12 Діаграма пакетів

Клієнтський рівень включає пакети «Інтерфейс» та «Веб-запити». Пакет «Інтерфейс» забезпечує взаємодію користувачів із системою через сторінки, форми та таблиці, а пакет «Веб-запити» передає дії користувачів до серверної частини системи [22].

Серверний рівень містить основні функціональні модулі системи: «Доступ», «Замовлення», «Оплати», «Меню та склад», «Постачання», «Персонал», «Аналітика», «Журнал дій» та «База даних». Пакет «Доступ» відповідає за авторизацію користувачів та перевірку ролей. Пакет «Замовлення» реалізує створення і обробку замовлень, а пакет «Оплати» забезпечує

збереження інформації про платежі. Пакет «Меню та склад» використовується для роботи зі стравами, категоріями меню та складськими залишками. Пакет «Постачання» забезпечує облік закупівель інгредієнтів, а пакет «Персонал» — управління працівниками ресторану.

Пакет «Аналітика» використовується для формування звітів та аналізу діяльності ресторану. Окремо виділено пакет «Журнал дій», який фіксує основні операції користувачів у системі. Усі модулі взаємодіють із пакетом «База даних», який забезпечує централізоване збереження інформації.

3.2 Вибір інструментарію для створення ППЗ

Для розробки інформаційної системи обліку процесів ресторану було використано сучасний набір інструментальних засобів та технологій, які забезпечують стабільну роботу програмного забезпечення, підтримку централізованої бази даних, реалізацію бізнес-логіки та створення зручного користувацького інтерфейсу. Вибір мови програмування, середовища розробки та допоміжних технологій здійснювався з урахуванням функціональних потреб системи, можливості подальшого розвитку програмного забезпечення, зручності підтримки проєкту та сумісності між окремими компонентами системи [20].

Основною мовою програмування серверної частини системи було обрано C# [10]. Дана мова широко використовується для створення сучасних вебзастосунків та корпоративних інформаційних систем [10]. Використання C# дозволило реалізувати бізнес-логіку системи, механізми авторизації та обробки даних, а також забезпечити взаємодію із реляційною базою даних Microsoft SQL Server.

Основними перевагами використання C# є:

– підтримка об’єктно-орієнтованого програмування;

– висока продуктивність роботи;

– підтримка асинхронного програмування;

– велика кількість готових бібліотек;

– інтеграція з платформою .NET;

– зручна реалізація серверної логіки та вебтехнологій.

Для створення вебзастосунку було використано платформу ASP.NET Core MVC [2; 4]. Дана технологія дозволяє організувати структуру проєкту за принципом Model–View–Controller, що забезпечує розділення логіки роботи системи, інтерфейсу користувача та обробки даних. ASP.NET Core MVC забезпечує зручну маршрутизацію запитів, підтримку HTTP-запитів, систему ролей користувачів та інтеграцію з Entity Framework Core і Microsoft SQL Server [4; 5; 7].

Основними перевагами ASP.NET Core MVC є:

– поділ системи на модель, представлення та контролери;

– зручна організація структури програмного забезпечення;

– підтримка авторизації та ролей користувачів;

– підтримка маршрутизації та HTTP-запитів;

– висока продуктивність та масштабованість;

– можливість подальшого розвитку та підтримки системи [4; 15; 16].

Для реалізації інформаційної бази було використано Microsoft SQL Server. Дана система управління базами даних була обрана через підтримку реляційної моделі даних, механізмів забезпечення цілісності інформації та підтримку складних SQL-запитів. Використання Microsoft SQL Server дозволило реалізувати централізоване збереження інформації про користувачів, ролі, меню, замовлення, бронювання, оплату, складські залишки, закупівлі та аналітичні дані.

Основними перевагами Microsoft SQL Server є:

– підтримка реляційної моделі даних;

– підтримка первинних та зовнішніх ключів;

– забезпечення цілісності даних;

– підтримка транзакцій;

– підтримка SQL-представлень та збережених процедур;

– підтримка механізмів резервного копіювання та відновлення;

– висока надійність і стабільність роботи.

Для взаємодії між програмним кодом і базою даних було використано Entity Framework Core [7]. Дана технологія реалізує ORM-підхід, що дозволяє працювати з таблицями бази даних через класи C#. Використання Entity Framework Core значно спрощує виконання CRUD-операцій, формування LINQ-запитів та підтримку структури бази даних через систему міграцій [7].

Основними перевагами Entity Framework Core є:

– підтримка LINQ-запитів;

– автоматизація CRUD-операцій;

– підтримка міграцій бази даних;

– зменшення кількості SQL-коду;

– спрощення роботи з реляційною базою даних;

– інтеграція з ASP.NET Core MVC [7; 17; 24].

Для створення інтерфейсу користувача було використано HTML, CSS, Bootstrap та JavaScript. HTML використовується для створення структури вебсторінок та елементів інтерфейсу. CSS забезпечує оформлення зовнішнього вигляду системи, налаштування кольорів, шрифтів та розташування елементів. JavaScript використовується для реалізації інтерактивних елементів та динамічної взаємодії користувача із системою. Для створення адаптивного інтерфейсу було використано Bootstrap [11], який дозволяє швидко реалізувати сучасний дизайн вебзастосунку [21; 22; 23].

Основними перевагами Bootstrap є:

– підтримка адаптивного дизайну;

– велика кількість готових UI-компонентів;

– спрощення оформлення сторінок;

– підтримка різних розмірів екранів;

– сучасний зовнішній вигляд системи.

Середовищем розробки програмного забезпечення було обрано Microsoft Visual Studio 2022 [13]. Дане середовище підтримує роботу з ASP.NET Core MVC, Entity Framework Core та Microsoft SQL Server. Використання Visual Studio 2022 дозволило спростити створення проєкту, налагодження програмного коду та роботу з базою даних.

Основними перевагами Visual Studio 2022 є:

– вбудовані засоби налагодження;

– автоматичне підсвічування помилок;

– підтримка роботи з Git;

– інструменти для роботи з базою даних;

– підтримка міграцій Entity Framework Core;

– зручний інтерфейс середовища розробки.

3.3 Алгоритмізація та програмування програмних модулів

У процесі розробки програмного продукту було реалізовано програмні модулі, які забезпечують автоматизацію основних процесів роботи ресторану. Для кожного модуля виконувалась побудова алгоритмів обробки даних, взаємодії між компонентами системи та роботи з базою даних [19; 20].

Під час розробки особлива увага приділялась створенню логічно структурованих алгоритмів, які забезпечують коректне виконання операцій, контроль доступності даних, перевірку бізнес-правил системи та підтримку цілісності інформації. Для опису логіки роботи програмних модулів використовувались UML-діаграми послідовності, що дозволяють наочно відобразити взаємодію користувачів, програмних модулів і бази даних під час виконання окремих операцій [14; 19].

Алгоритмізація програмних модулів виконувалась з урахуванням необхідності обробки альтернативних сценаріїв роботи системи, перевірки коректності введених даних та обробки можливих помилок. Усі алгоритми

будувались таким чином, щоб забезпечити послідовність виконання операцій, правильне оновлення інформації у базі даних та стабільність роботи системи [19; 20].

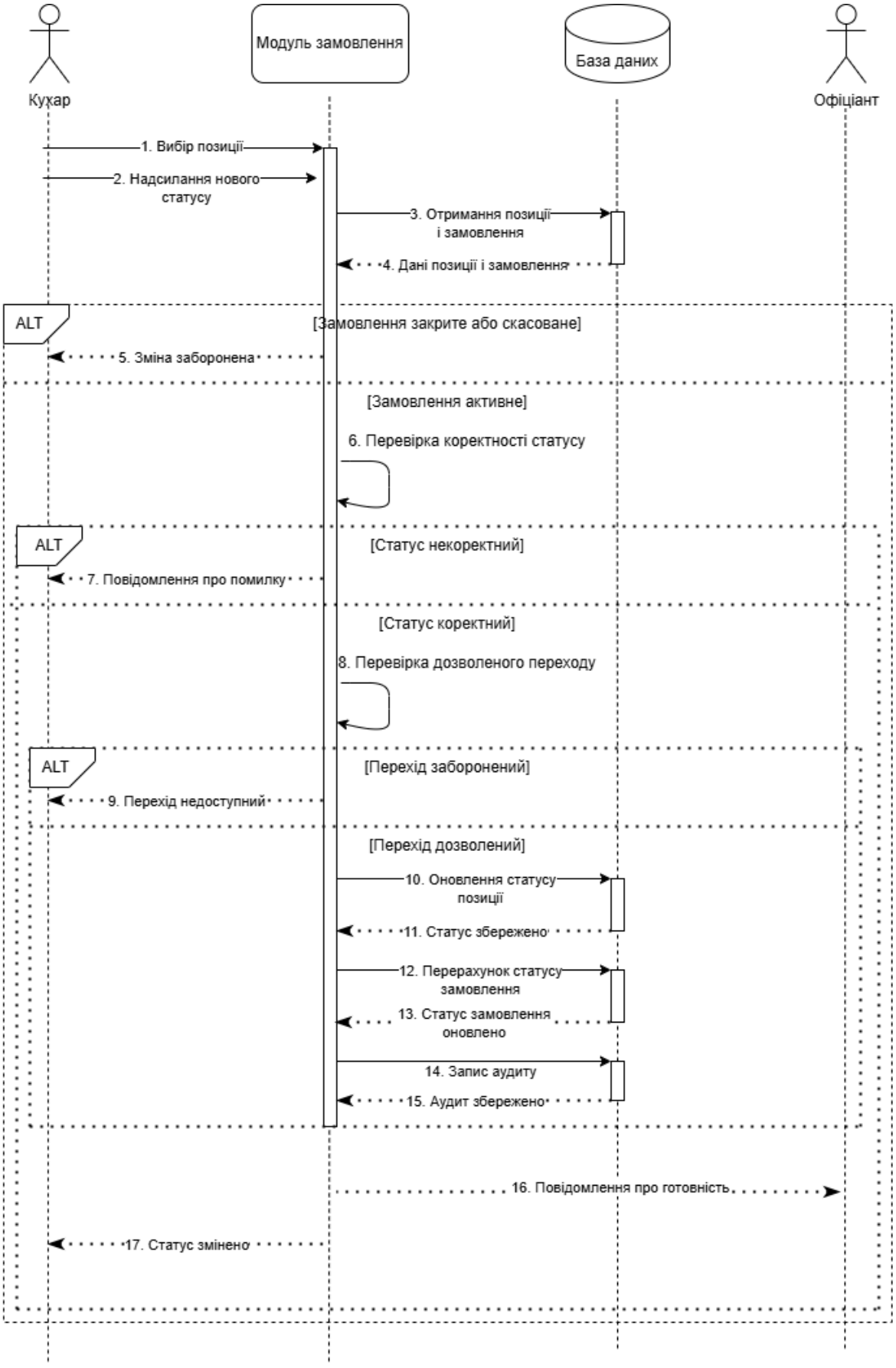


Рис. 13 Діаграма послідовності алгоритму зміни статусу приготування

Дана UML-діаграма послідовності описує алгоритм зміни статусу приготування позиції замовлення у системі автоматизації ресторану. У процесі роботи кухар обирає необхідну позицію замовлення та надсилає новий статус до модуля замовлень.

Після отримання нового статусу модуль замовлень звертається до бази даних для отримання інформації про позицію та пов’язане замовлення. На основі отриманих даних система перевіряє, чи не є замовлення закритим або скасованим. Для опису альтернативних сценаріїв використовується ALT-блок.

Якщо замовлення закрите або скасоване, система забороняє зміну статусу та повертає кухарю відповідне повідомлення. Якщо замовлення активне, модуль замовлень виконує перевірку коректності нового статусу та перевірку дозволеного переходу між статусами. Дані перевірки реалізовані у вигляді self-call, оскільки виконуються всередині самого модуля без звернення до інших компонентів системи.

У разі виявлення помилки або забороненого переходу система повертає кухарю повідомлення про неможливість зміни статусу. Якщо всі перевірки пройдено успішно, модуль замовлень оновлює статус позиції у базі даних, виконує перерахунок загального статусу замовлення та записує подію до журналу аудиту.

Після успішного оновлення статусу система додатково надсилає офіціанту повідомлення про готовність страви, якщо новий статус позиції дорівнює «Готово». Після завершення всіх операцій кухар отримує підтвердження успішної зміни статусу [14].

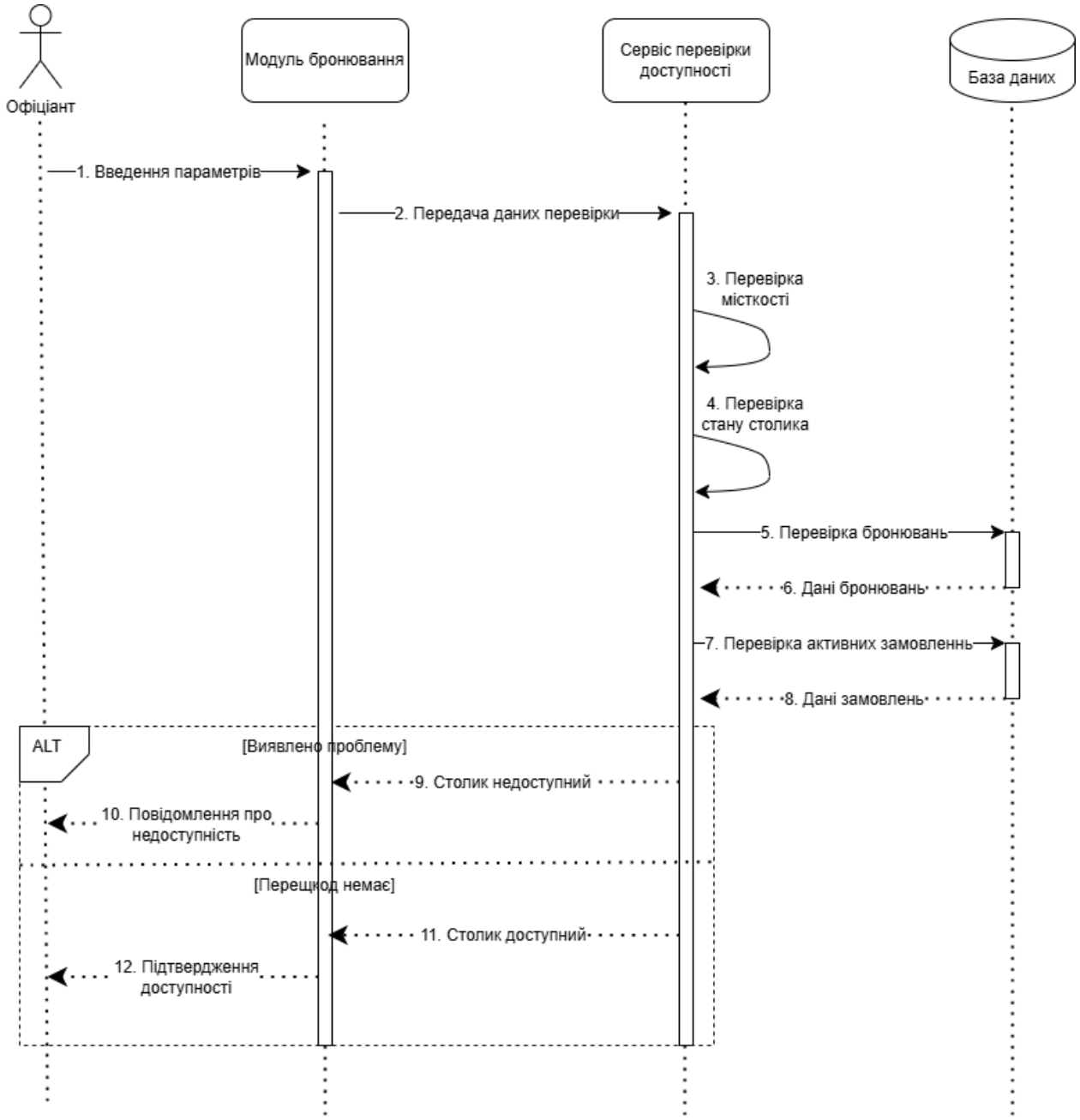


Рис. 14 Діаграма послідовності алгоритму перевірки доступності столика

Дана UML-діаграма послідовності описує алгоритм перевірки доступності столика у системі автоматизації ресторану. Діаграма відображає взаємодію між офіціантом, модулем бронювання, сервісом перевірки доступності та базою даних під час виконання перевірки можливості бронювання або використання конкретного столика.

У процесі роботи алгоритму офіціант вводить параметри перевірки столика, зокрема кількість гостей або потрібний столик, після чого ці дані

передаються до модуля бронювання. Модуль бронювання надсилає отримані параметри до сервісу перевірки доступності для виконання подальших перевірок.

На наступному етапі сервіс перевірки доступності виконує внутрішні перевірки місткості столика та його поточного стану. Дані перевірки реалізовані у вигляді self-call, оскільки виконуються всередині самого сервісу без звернення до інших компонентів системи. Після цього сервіс звертається до бази даних для отримання інформації про активні бронювання та поточні замовлення, які можуть впливати на доступність столика.

База даних повертає інформацію про наявні бронювання та активні замовлення, після чого сервіс аналізує отримані дані та визначає, чи доступний столик для використання. Для опису альтернативних сценаріїв роботи використовується ALT-блок.

У випадку, якщо система виявляє проблему, наприклад столик уже заброньований або використовується в активному замовленні, сервіс повертає модулю бронювання повідомлення про недоступність столика. Після цього модуль бронювання надсилає офіціанту повідомлення про неможливість бронювання або використання вибраного столика.

Якщо ж перешкод не виявлено, сервіс повідомляє модуль бронювання про доступність столика, а модуль бронювання повертає офіціанту підтвердження успішної перевірки доступності [14].

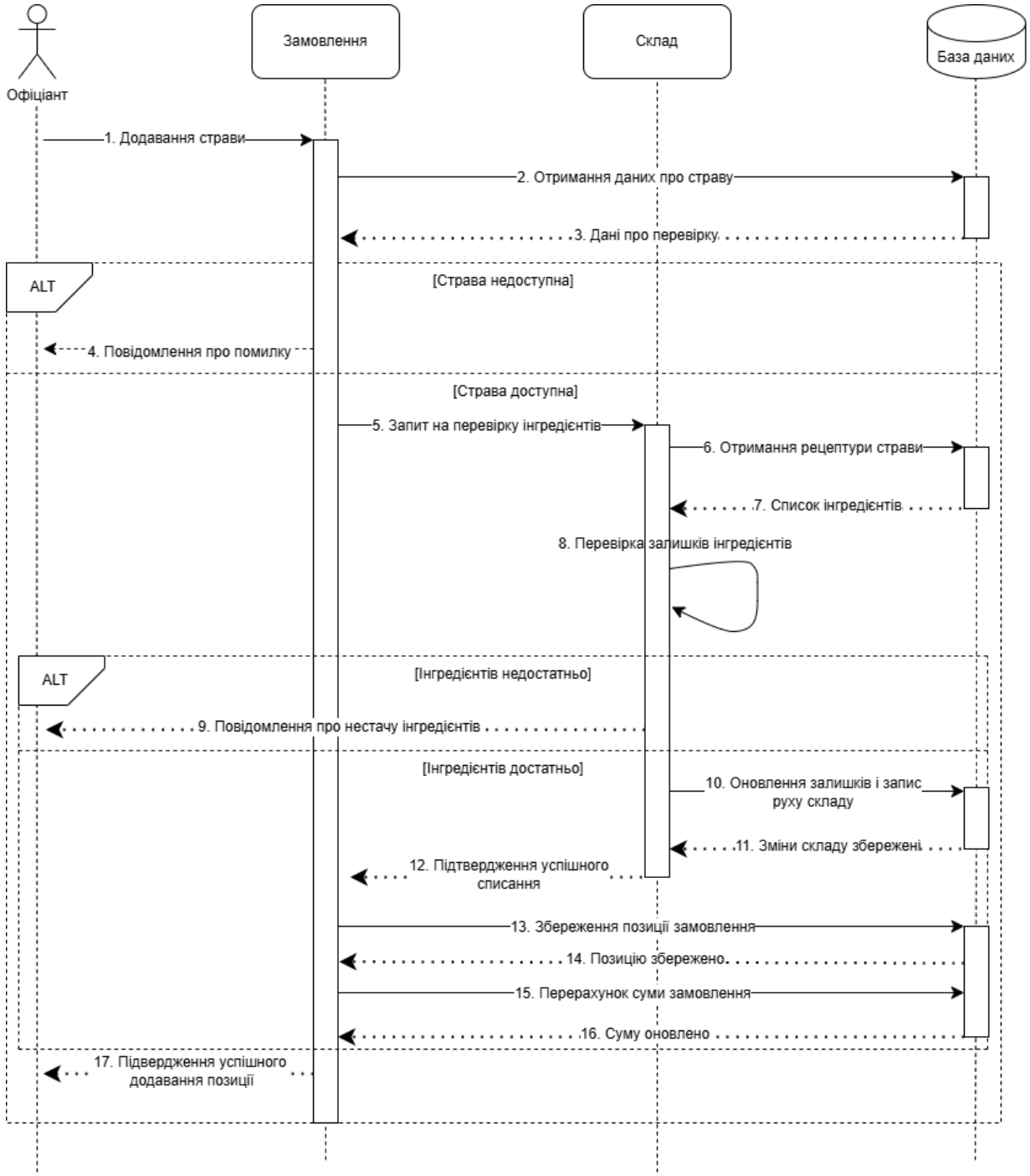


Рис. 15 Діаграма послідовності алгоритму перевірки інгредієнтів і списання залишків

Дана UML-діаграма послідовності описує алгоритм перевірки інгредієнтів і списання залишків у системі автоматизації ресторану. У процесі роботи офіціант додає страву до замовлення, після чого модуль замовлень звертається до бази даних для отримання інформації про вибрану страву.

Після отримання даних система перевіряє доступність страви. Для відображення альтернативних сценаріїв використовується ALT-блок. Якщо страва недоступна, модуль замовлень надсилає офіціанту повідомлення про помилку. Якщо страва доступна, модуль замовлень передає запит до модуля складського обліку для перевірки наявності необхідних інгредієнтів.

Модуль складського обліку звертається до бази даних для отримання рецептури страви та списку інгредієнтів. Після цього система виконує внутрішню перевірку залишків інгредієнтів, яка реалізована у вигляді self-call, оскільки виконується всередині самого модуля складського обліку.

У разі недостатньої кількості інгредієнтів система повертає повідомлення про нестачу, яке передається офіціанту. Якщо інгредієнтів достатньо, модуль складського обліку оновлює залишки та записує рух складу у базі даних. Після успішного списання модуль замовлень зберігає позицію замовлення та виконує перерахунок загальної суми замовлення.

Після завершення всіх операцій офіціант отримує підтвердження успішного додавання позиції до замовлення [14].

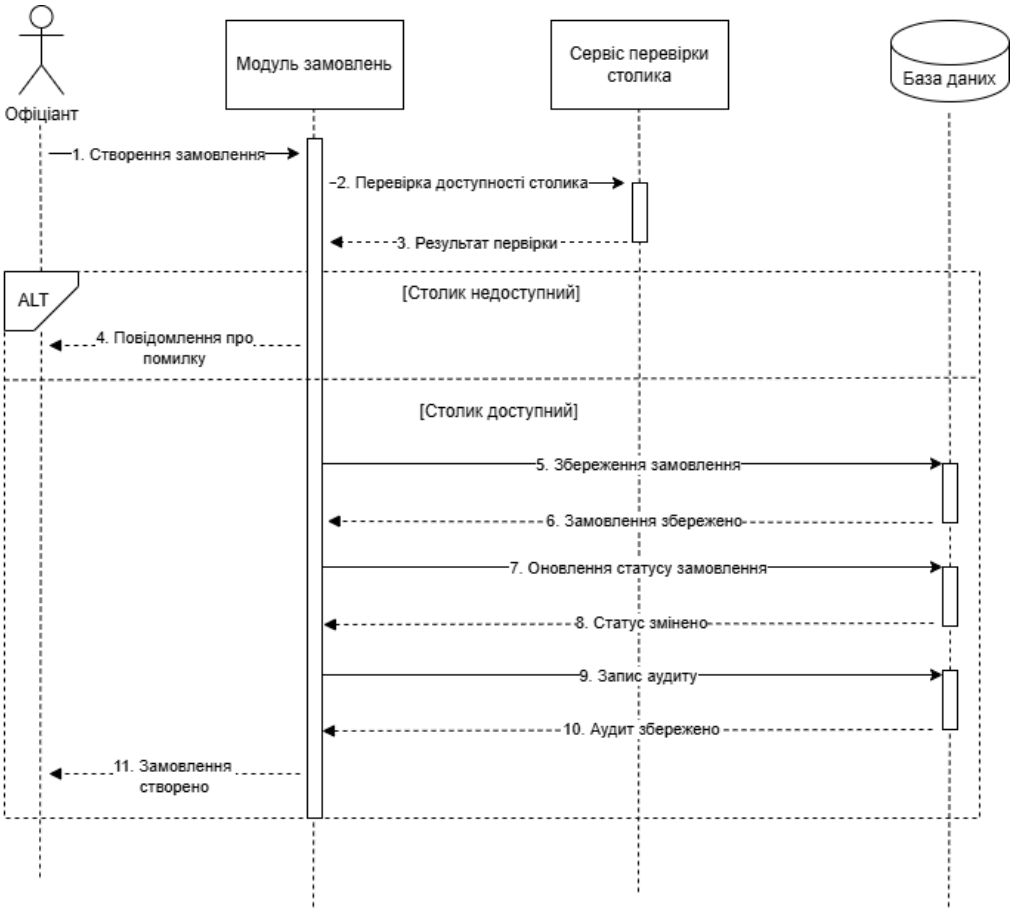


Рис. 16 Діаграма послідовності алгоритму створення замовлення

Дана UML-діаграма послідовності описує алгоритм створення замовлення у системі автоматизації ресторану. У процесі роботи офіціант створює нове замовлення, після чого модуль замовлень ініціює перевірку доступності столика через сервіс перевірки столика.

Сервіс перевірки столика аналізує можливість використання вибраного столика та повертає результат перевірки до модуля замовлень. Для відображення альтернативних сценаріїв використовується ALT-блок.

Якщо столик недоступний, система надсилає офіціанту повідомлення про помилку та забороняє створення замовлення. Якщо столик доступний, модуль замовлень зберігає нове замовлення у базі даних та отримує підтвердження успішного збереження.

Після цього система оновлює статус замовлення та записує інформацію про виконану операцію до журналу аудиту. Після успішного завершення всіх операцій офіціант отримує підтвердження про створення замовлення [14; 18].

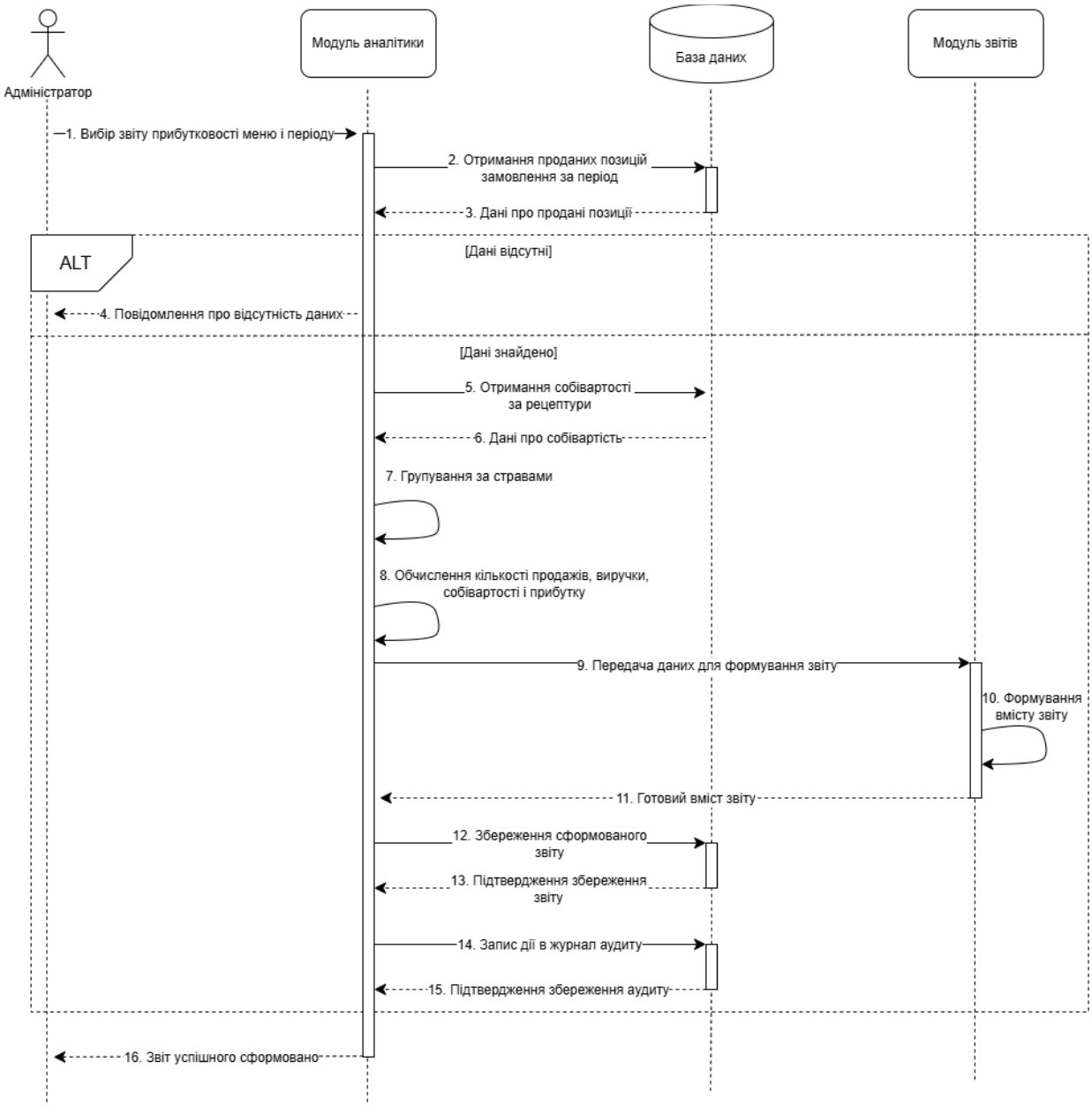


Рис. 17 Діаграма послідовності алгоритму формуванню звіту прибутковості меню

Дана UML-діаграма послідовності описує алгоритм формування звіту прибутковості меню у системі автоматизації ресторану. У процесі роботи адміністратор обирає звіт прибутковості меню та задає необхідний період формування звіту, після чого запит передається до модуля аналітики.

Модуль аналітики звертається до бази даних для отримання інформації про продані позиції замовлень за вказаний період. Після отримання даних система перевіряє їх на наявність. Для відображення альтернативних сценаріїв використовується ALT-блок.

Якщо дані відсутні, система надсилає адміністратору повідомлення про відсутність інформації для формування звіту. Якщо необхідні дані знайдено, модуль аналітики отримує інформацію про собівартість страв за рецептурами та виконує внутрішню обробку даних, зокрема групування за стравами та обчислення кількості продажів, виручки, собівартості та прибутку.

Після виконання розрахунків модуль аналітики передає підготовлені дані до модуля звітів для формування готового вмісту звіту. Сформований звіт зберігається у базі даних, а інформація про виконану операцію записується до журналу аудиту.

Після успішного завершення всіх операцій адміністратор отримує повідомлення про успішне формування звіту.

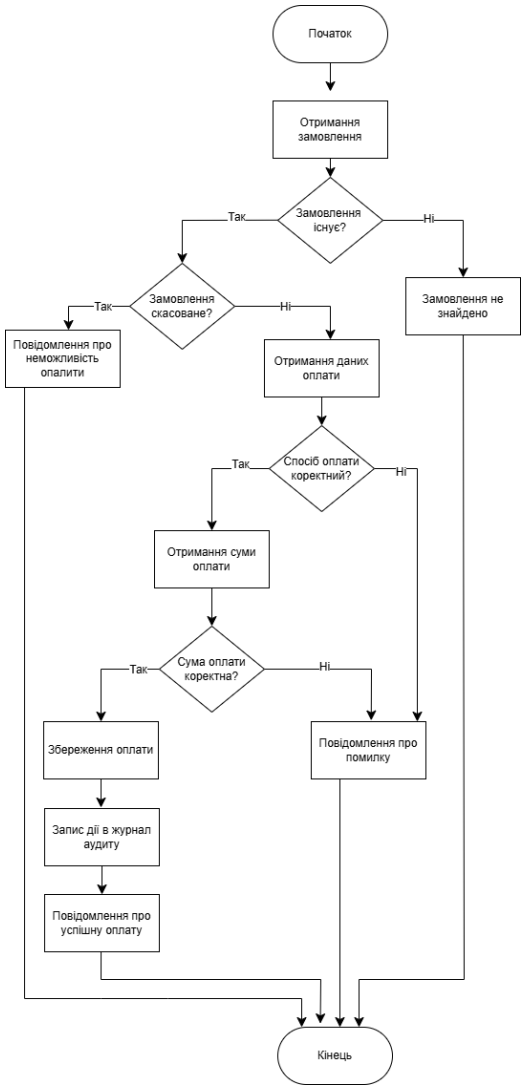


Рис. 18 Блок-схема алгоритму перерахунку загального статусу замовлення

Блок-схема процесу проведення оплати замовлення відображає послідовність дій, що реалізована в модулі замовлень інформаційної системи ресторану. Її призначення полягає у забезпеченні коректної реєстрації факту оплати, перевірці допустимості цієї операції та фіксації результату в системі. Такий підхід дозволяє уникнути помилкового введення платіжних даних, а також забезпечує цілісність обліку фінансових операцій.

На початку алгоритму система отримує замовлення та перевіряє факт його існування. Якщо замовлення не знайдено, подальше виконання процесу припиняється. Якщо замовлення існує, система перевіряє, чи не має воно статусу скасованого. Для скасованого замовлення проведення оплати не допускається, тому користувачеві виводиться відповідне повідомлення. Така перевірка

потрібна для того, щоб унеможливити реєстрацію платежу для операції, яка вже втратила актуальність.

Після цього система переходить до обробки платіжних даних. На даному етапі виконується перевірка способу оплати, а також отримання і перевірка суми платежу. Якщо спосіб оплати вказано некоректно або сума не відповідає встановленим вимогам, система формує повідомлення про помилку та завершує виконання алгоритму без збереження даних. Це забезпечує правильність введення інформації та запобігає появі помилкових записів у базі даних.

У разі успішного проходження всіх перевірок створюється запис про оплату, який зберігається в базі даних. Після цього система фіксує виконану дію в журналі аудиту, що дозволяє відстежувати, хто і коли виконав операцію. Завершальним етапом алгоритму є виведення повідомлення про успішне проведення оплати. Отже, блок-схема повністю відображає логіку реалізованого в системі процесу та демонструє послідовність перевірки, реєстрації й контролю платіжної операції [19].

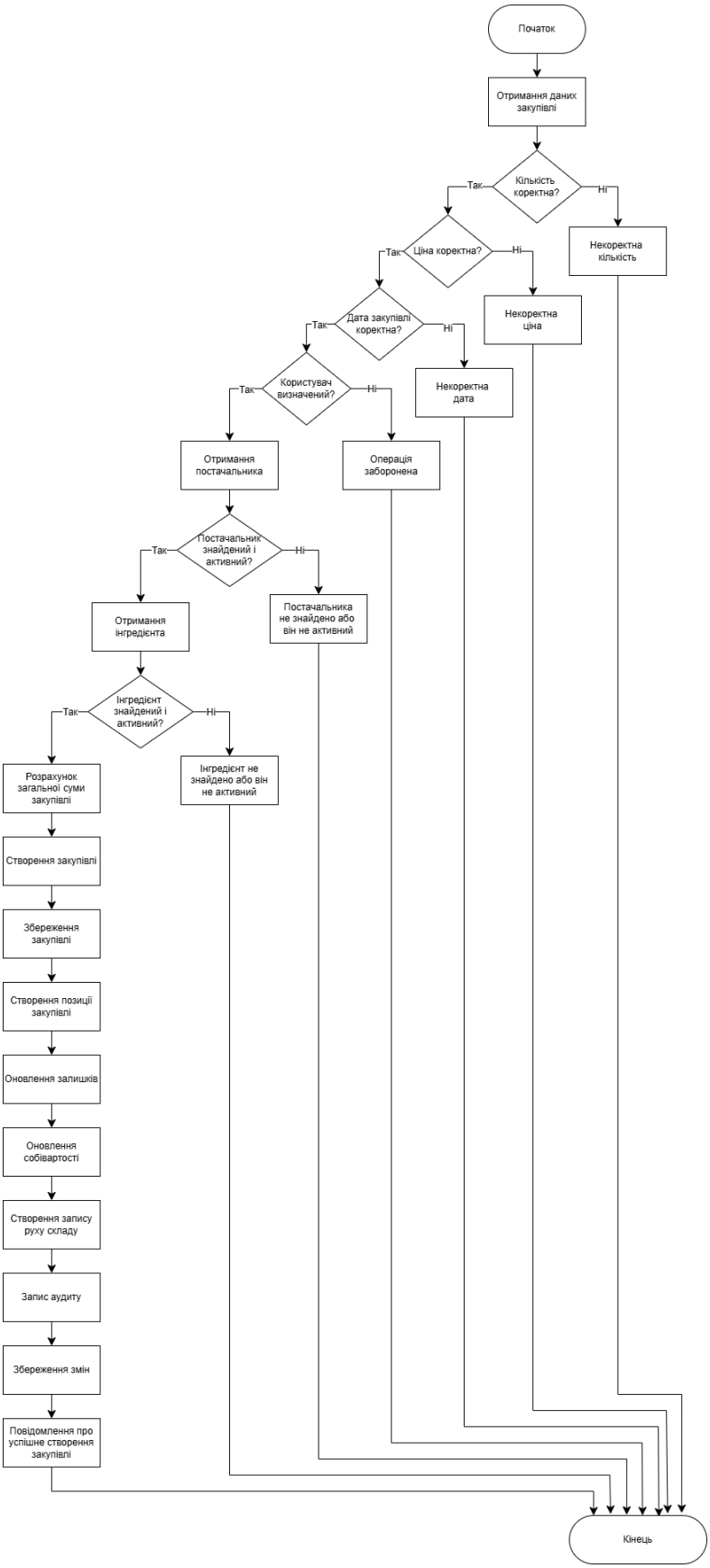


Рис. 19 Блок-схема алгоритму створення закупівлі та поповнення складських залишків

Блок-схема процесу створення закупівлі та поповнення складських залишків відображає послідовність дій, що реалізована в модулі складського обліку інформаційної системи ресторану. Її основне призначення полягає у забезпеченні коректного введення даних про закупівлю, перевірці пов’язаних довідникових сутностей та автоматичному оновленні складської інформації після успішного проведення операції. Такий алгоритм дозволяє поєднати реєстрацію закупівлі з актуалізацією залишків інгредієнтів і формуванням історії складських змін.

На початку алгоритму система отримує дані закупівлі та послідовно перевіряє їх коректність. Спочатку контролюється кількість товару, потім закупівельна ціна та дата закупівлі. Якщо хоча б один із цих параметрів не відповідає встановленим вимогам, система завершує процес із відповідним повідомленням про помилку. Такий підхід забезпечує первинну валідацію введеної інформації ще до переходу до змін у базі даних.

Після успішної перевірки вхідних даних система визначає користувача, який виконує операцію. Якщо користувач не ідентифікований, виконання закупівлі забороняється. Далі послідовно перевіряється наявність та активність постачальника, а також наявність та активність інгредієнта. У разі відсутності будь-якої з цих сутностей або їх неактивного стану система припиняє виконання алгоритму та інформує користувача про неможливість продовження операції.

Якщо всі перевірки завершуються успішно, система переходить до виконання основного функціонального сценарію. На цьому етапі розраховується загальна сума закупівлі, створюється запис закупівлі, після чого він зберігається в базі даних. Далі формується позиція закупівлі, оновлюються складські залишки відповідного інгредієнта, коригується його собівартість і створюється запис руху складу. Окремо виконується запис дії до журналу аудиту, що дає змогу забезпечити контроль за всіма господарськими операціями в системі.

Завершальним етапом алгоритму є збереження всіх внесених змін та формування повідомлення про успішне створення закупівлі. Отже, блок-схема

повністю відображає реалізовану в системі логіку створення закупівлі та демонструє взаємозв’язок між перевіркою введених даних, контролем довідникових об’єктів, оновленням складського стану й аудитом виконаної операції [19].

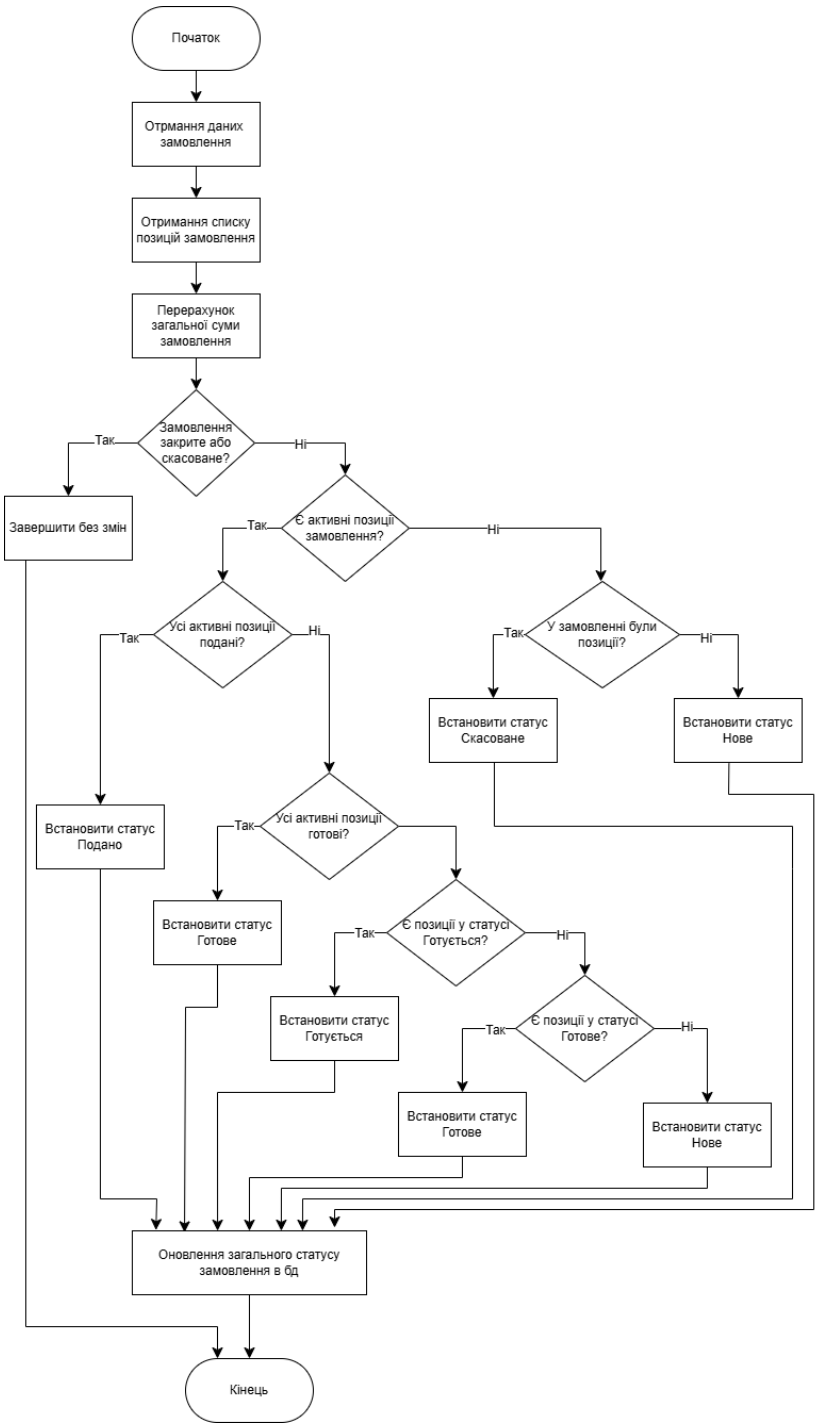


Рис. 20 Блок-схема алгоритму перерахунку загального статусу замовлення

Блок-схема перерахунку загального статусу замовлення відображає внутрішню логіку роботи модуля замовлень інформаційної системи ресторану. Її призначення полягає у визначенні актуального стану всього замовлення на основі поточного стану його окремих позицій. Такий алгоритм є необхідним для забезпечення узгодженості даних між позиціями замовлення та його загальним статусом, що особливо важливо під час роботи офіціанта, кухаря та при формуванні звітності.

На початку алгоритму система отримує дані замовлення та список його позицій, після чого виконує перерахунок загальної суми замовлення. Далі перевіряється, чи не має замовлення статусу закритого або скасованого. Якщо така умова виконується, алгоритм завершується без внесення змін, оскільки для завершених або анульованих замовлень подальший перегляд статусу не потрібний.

У випадку, коли замовлення залишається активним, система перевіряє наявність активних позицій, тобто таких, що не перебувають у статусі скасованих. Якщо активних позицій немає, виконується додаткова перевірка: чи були в замовленні позиції взагалі. Якщо позиції були, замовленню присвоюється статус Скасоване; якщо ж позицій не було, встановлюється статус Нове. Така логіка дозволяє розрізняти порожнє замовлення та замовлення, усі позиції якого були скасовані.

Якщо активні позиції існують, алгоритм послідовно аналізує їхні стани. Насамперед перевіряється, чи всі активні позиції мають статус Подано. У такому випадку загальному замовленню також присвоюється статус Подано. Якщо ця умова не виконується, система перевіряє, чи всі активні позиції мають статус Готове. Якщо так, замовленню встановлюється статус Готове. Далі система визначає, чи є хоча б одна позиція у статусі Готується. За наявності хоча б однієї такої позиції замовлення отримує статус Готується.

На наступному етапі алгоритм перевіряє, чи є серед активних позицій хоча б одна у статусі Готове. Якщо така позиція є, для всього замовлення

встановлюється статус Готове. У протилежному випадку замовленню присвоюється статус Нове. Після визначення відповідного стану виконується оновлення загального статусу замовлення в базі даних, після чого система завершує роботу алгоритму [19].

```
[Authorize(Roles = "Administrator")]
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Generate(ReportGenerateViewModel model, CancellationToken cancellationToken)
{
    model.ReportTypes = ReportDisplayHelper.BuildReportTypeOptions(model.ReportType);

    if (model.PeriodTo.Date < model.PeriodFrom.Date)
    {
        ModelState.AddModelError(nameof(model.PeriodTo), "Дата завершення не може бути меншою за дату початку.");
    }

    if (!ModelState.IsValid)
    {
        return View("Create", model);
    }

    var reportContent = await BuildReportContentAsync(model.ReportType, model.PeriodFrom.Date, model.PeriodTo.Date, cancellationToken);
    var currentUserId = GetCurrentUserId();

    if (!currentUserId.HasValue)
    {
        return Forbid();
    }

    var generatedReport = new GeneratedReport
    {
        ReportType = model.ReportType,
        Title = $"{ReportDisplayHelper.ToDisplayName(model.ReportType)} за {model.PeriodFrom:dd.MM.yyyy} – {model.PeriodTo:dd.MM.yyyy}",
        PeriodFrom = model.PeriodFrom.Date,
        PeriodTo = model.PeriodTo.Date,
        CreatedByUserId = currentUserId.Value,
        CreatedAt = DateTime.UtcNow,
        ReportDataJson = JsonSerializer.Serialize(reportContent, _jsonOptions),
        Status = "Generated"
    };

    _dbContext.GeneratedReports.Add(generatedReport);
    await _dbContext.SaveChangesAsync(cancellationToken);

    AddAuditLog(currentUserId, "GenerateReport", "GeneratedReport", generatedReport.ReportId,
        $"Адміністратор сформував звіт: {generatedReport.Title}.");
    await _dbContext.SaveChangesAsync(cancellationToken);

    TempData["AnalyticsMessage"] = "Звіт успішно сформовано.";
    return RedirectToAction(nameof(Details), new { id = generatedReport.ReportId });
}
```

Рис. 21 Функція генерації та збереження сформованого звіту

Наведений фрагмент коду реалізує процес формування та збереження звіту в модулі аналітики інформаційної системи ресторану. Спочатку виконується перевірка прав доступу та коректності введеного періоду, зокрема виконується перевірка, щоб дата завершення не була меншою за дату початку. Якщо вхідні дані не проходять валідацію, користувач повертається до форми створення звіту.

Після успішної перевірки система формує вміст звіту за допомогою окремого методу, визначає поточного користувача та створює об’єкт

GeneratedReport, у якому зберігаються тип звіту, період, дата створення та серіалізований вміст звіту. Далі звіт записується в базу даних, а інформація про виконану дію фіксується в журналі аудиту.

Завершальним етапом є виведення повідомлення про успішне формування звіту та перенаправлення користувача на сторінку перегляду його деталей.

```
public async Task<IActionResult> DeletePurchase(int purchaseId, CancellationToken cancellationToken)
{
    var purchase = await _dbContext.Purchases
        .Include(x => x.PurchaseItems)
            .ThenInclude(x => x.Ingredient)
        .FirstOrDefaultAsync(x => x.PurchaseId == purchaseId, cancellationToken);

    if (purchase is null)
    {
        return NotFound();
    }

    foreach (var item in purchase.PurchaseItems)
    {
        if (item.Ingredient is not null)
        {
            item.Ingredient.QuantityInStock = Math.Max(0, item.Ingredient.QuantityInStock - item.Quantity);
        }
    }

    var movements = await _dbContext.StockMovements
        .Where(x => x.SourceDocument == $"PUR-{purchaseId}")
        .ToListAsync(cancellationToken);

    _dbContext.StockMovements.RemoveRange(movements);
    _dbContext.PurchaseItems.RemoveRange(purchase.PurchaseItems);
    _dbContext.Purchases.Remove(purchase);

    AddAuditLog(GetCurrentUserId(), "Видалення закупівлі", "Purchases", purchaseId,
        $"Видалено закупівлю #{purchaseId} та скориговано складські залишки.");

    await _dbContext.SaveChangesAsync(cancellationToken);
```

Рис. 22 Функція Видалення закупівлі з автоматичним коригуванням складських залишків

Наведений фрагмент коду реалізує процес видалення закупівлі в модулі складського обліку інформаційної системи ресторану. Його призначення полягає не лише у вилученні запису про закупівлю, а й у коригуванні всіх пов’язаних даних, які були змінені внаслідок її створення. Такий підхід забезпечує цілісність складського обліку та узгодженість даних у системі.

На початку методу система отримує з бази даних закупівлю разом із її позиціями та пов’язаними інгредієнтами. Якщо запис закупівлі не знайдено, виконується повернення стандартної відповіді NotFound(), а подальші дії не

здійснюються. Якщо закупівля існує, система проходить по всіх її позиціях і для кожного інгредієнта зменшує поточний складський залишок на кількість, яка була раніше додана цією закупівлею. Для запобігання появі від'ємних значень використовується функція Math.Max(0, ...).

Після коригування залишків система отримує всі записи руху складу, що були створені для цієї закупівлі, та видаляє їх. Далі видаляються позиції закупівлі та сам запис закупівлі. Окремо виконується запис інформації про цю дію до журналу аудиту, що дозволяє зафіксувати факт видалення закупівлі та подальшого коригування складських даних.

Завершальним етапом є збереження змін у базі даних.

```
if (newStatus == OrderItemStatus.Cancelled
    && orderItem.Status != OrderItemStatus.Cancelled
    && orderItem.Status != OrderItemStatus.Served)
{
    await RestoreIngredientsForOrderItemAsync(orderItem, "Скасування позиції замовлення", cancellationToken)
}

orderItem.Status = newStatus;
_dbContext.Entry(orderItem).Property(x => x.Status).IsModified = true;

AddAuditLog(GetCurrentUserId(), "Зміна статусу позиції", "Orders", orderId,
    $"Статус позиції #{orderItemId} змінено на {newStatus}.");

await _dbContext.SaveChangesAsync(cancellationToken);
await RecalculateOrderAsync(orderId, cancellationToken);
await _dbContext.SaveChangesAsync(cancellationToken);
```

Рис. 23 Функція повернення інгредієнтів на склад при скасуванні позиції замовлення (фрагмент 1)

```
private async Task RestoreIngredientsForOrderItemAsync(OrderItem orderItem, string reason, CancellationToken cancellationToken)
{
    var menuItemName = await _dbContext.MenuItems
        .AsNoTracking()
        .Where(x => x.MenuItemId == orderItem.MenuItemId)
        .Select(x => x.Name)
        .FirstOrDefaultAsync(cancellationToken) ?? "Страва";

    var recipeIngredients = await _dbContext.DishIngredients
        .Include(x => x.Ingredient)
        .Where(x => x.MenuItemId == orderItem.MenuItemId)
        .ToListAsync(cancellationToken);

    foreach (var recipeIngredient in recipeIngredients.Where(x => x.Ingredient is not null))
    {
        var totalQuantity = recipeIngredient.QuantityNeeded * orderItem.Quantity;
        recipeIngredient.Ingredient!.QuantityInStock += totalQuantity;

        _dbContext.StockMovements.Add(new StockMovement
        {
            IngredientId = recipeIngredient.IngredientId,
            MovementType = StockMovementType.In,
            Quantity = totalQuantity,
            MovementDate = DateTime.UtcNow,
            SourceDocument = $"ORD-{orderItem.OrderId}-ITEM-{orderItem.OrderItemId}-RETURN",
            Comment = $"{reason}: {menuItemName}"
        });
    }

    await _dbContext.SaveChangesAsync(cancellationToken);
}
```

Рис. 24 Функція повернення інгредієнтів на склад при скасуванні позиції замовлення (фрагмент 2)

Наведений фрагмент коду реалізує механізм повернення інгредієнтів на склад у разі скасування позиції замовлення. Його призначення полягає у відновленні складських залишків тих інгредієнтів, які були попередньо списані для приготування страви. Такий підхід забезпечує узгодженість між модулем замовлень і модулем складського обліку та дозволяє підтримувати коректний стан запасів у системі.

На початку методу система визначає назву страви, до якої належить скасована позиція замовлення. Далі з бази даних отримується рецептура цієї страви, тобто перелік інгредієнтів і кількість кожного з них, необхідна для приготування однієї порції. Після цього для кожного інгредієнта обчислюється загальний обсяг, який потрібно повернути на склад, з урахуванням кількості скасованих порцій у замовленні.

Наступним етапом алгоритму є фактичне відновлення складських залишків: система збільшує значення QuantityInStock для кожного відповідного інгредієнта. Одночасно створюється запис у таблиці складських рухів StockMovements, у якому фіксується надходження інгредієнтів назад на склад. У записі вказуються ідентифікатор інгредієнта, тип руху, кількість, дата операції, джерело документа та текстове пояснення причини повернення.

Завершальним етапом є збереження змін у базі даних. Даний фрагмент коду демонструє власне програмне рішення, у якому операція скасування позиції замовлення автоматично супроводжується коригуванням складських запасів і документуванням цього процесу в журналі руху матеріальних ресурсів. Це забезпечує точність обліку та підвищує надійність роботи інформаційної системи ресторану.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування інформаційної системи є одним із найважливіших етапів розробки програмного забезпечення, оскільки саме на цьому етапі перевіряється правильність роботи системи, стабільність її функціонування, відповідність поставленим вимогам та можливість використання в реальних умовах. Основною метою тестування є виявлення помилок, недоліків та можливих збоїв у роботі системи ще до моменту її повноцінного впровадження та експлуатації [20].

Тестування програмного забезпечення — це процес перевірки роботи інформаційної системи шляхом виконання певних дій та аналізу отриманих результатів. Під час тестування здійснюється перевірка того, чи правильно система виконує необхідні функції, чи відповідають результати очікуваним значенням, а також чи відсутні критичні помилки, які можуть впливати на роботу користувачів або цілісність даних.

У процесі розробки інформаційної системи обліку процесів для власника ресторану тестування проводилось на різних етапах створення програмного продукту. Це дозволило своєчасно виявляти помилки, перевіряти правильність реалізації функціоналу та забезпечити стабільну роботу системи.

Для перевірки працездатності розробленої системи було використано функціональне тестування. Даний тип тестування дозволяє перевірити правильність виконання основних функцій системи відповідно до поставлених вимог. Під час функціонального тестування увага приділялась роботі модулів

авторизації, управління замовленнями, управління меню, складського обліку, а також роботі ролей користувачів.

Процес тестування відбувався шляхом виконання певних сценаріїв роботи користувачів у системі. Для кожного функціонального модуля перевірялась коректність введення даних, правильність їх збереження у базі даних, оновлення інформації та відображення результатів у користувацькому інтерфейсі.

Під час тестування модуля авторизації перевірялась можливість входу користувачів до системи відповідно до їх ролей. Було протестовано правильність введення логіна та пароля, перевірку некоректних даних. У результаті тестування було підтверджено, що система правильно виконує авторизацію користувачів та забезпечує розмежування прав доступу.

При тестуванні модуля управління замовленнями перевірялась можливість створення нового замовлення, додавання позицій меню, зміна статусів замовлення та автоматичне оновлення загального статусу замовлення залежно від стану окремих позицій. Також було перевірено правильність передачі інформації між офіціантом, кухарем та базою даних системи. У ході тестування було встановлено, що система коректно зберігає інформацію про замовлення та забезпечує правильну взаємодію між користувачами.

Окремо проводилось тестування модуля складського обліку. Було перевірено правильність списання інгредієнтів після оформлення замовлення, оновлення залишків продуктів на складі та відображення актуальної інформації у системі. Також перевірялась робота системи при недостатній кількості інгредієнтів. Результати тестування підтвердили правильність роботи механізмів складського обліку та контролю залишків.

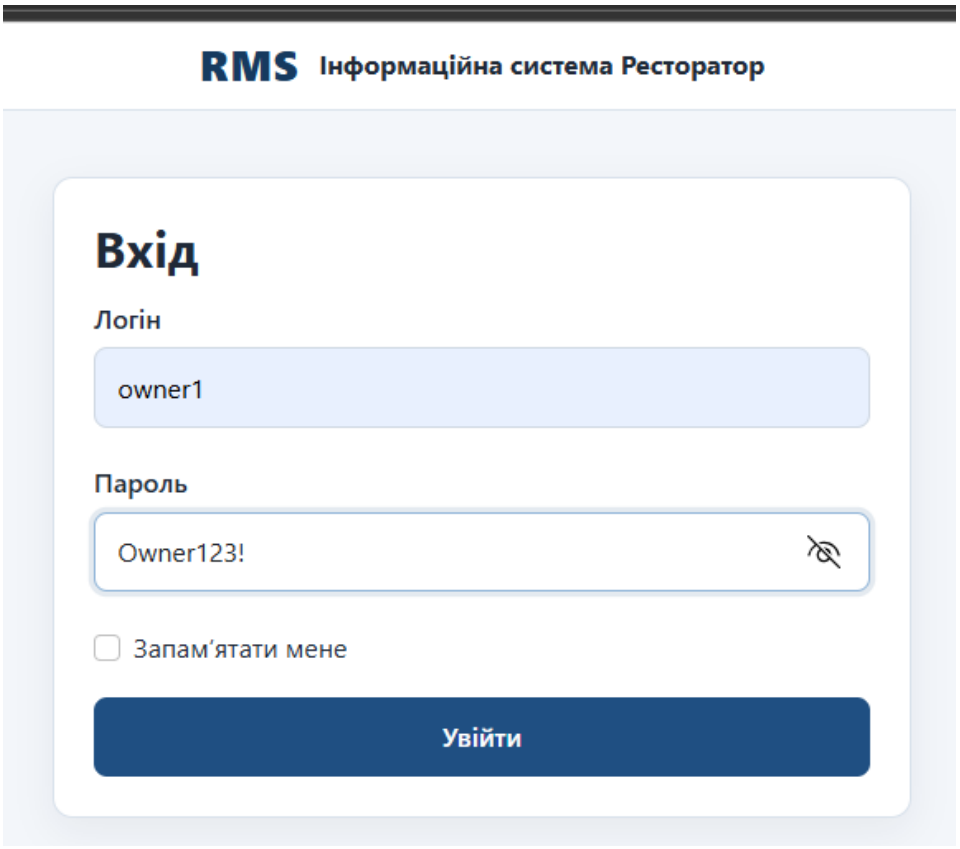


Рис. 25 Демонстрація входу у систему (фрагмент 1)

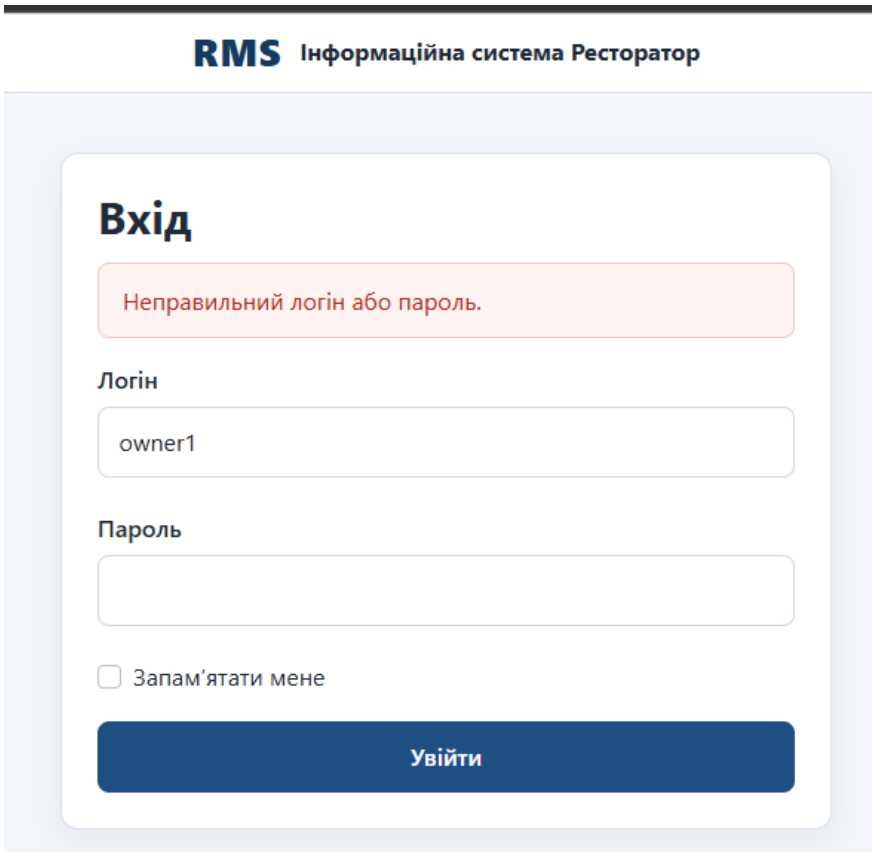


Рис. 26 Демонстрація входу у систему (фрагмент 2)

На рисунках 25–26 наведено демонстрацію процесу входу користувача до інформаційної системи ресторану. На першому фрагменті показано форму автентифікації, у якій користувач вводить логін і пароль для входу до системи. Інтерфейс містить основні елементи авторизації: поля для введення облікових даних, прапорець запам’ятовування користувача та кнопку підтвердження входу.

Другий фрагмент ілюструє реакцію системи на введення некоректних облікових даних. У разі помилки автентифікації користувачеві виводиться відповідне повідомлення про неправильний логін або пароль, а сама форма залишається доступною для повторного введення даних. Така реалізація забезпечує зрозумілий механізм взаємодії з користувачем, підвищує зручність роботи із системою та демонструє наявність базового контролю доступу до функціональних модулів.

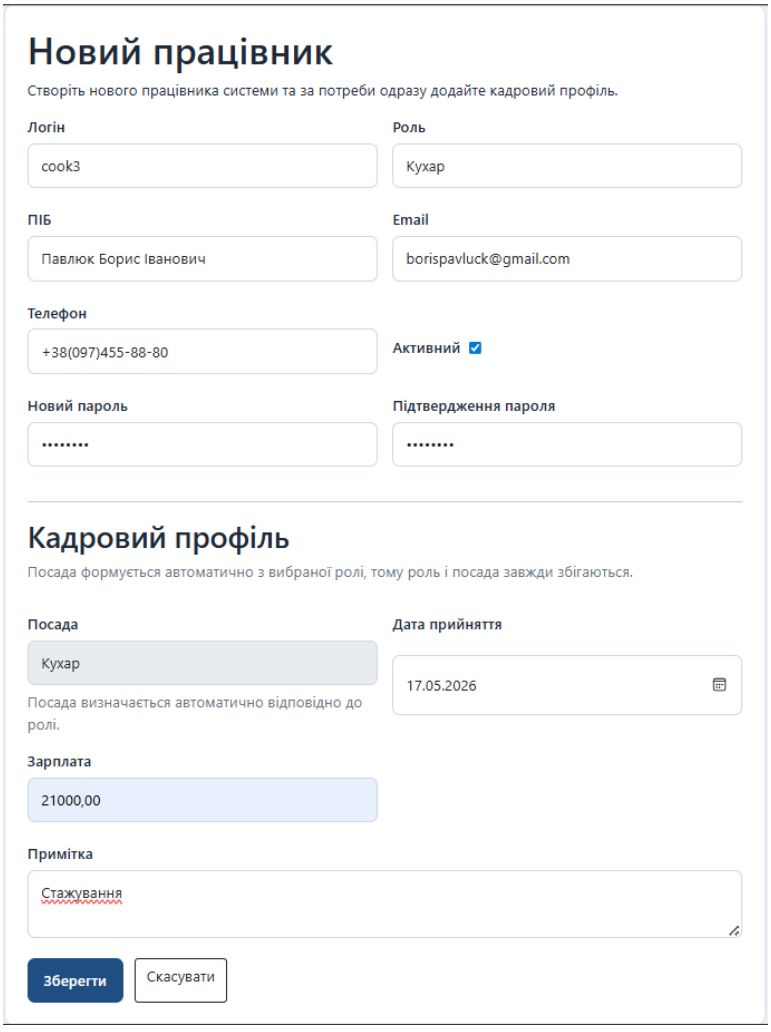


Рис. 27 Демонстрація реєстрації нового працівника

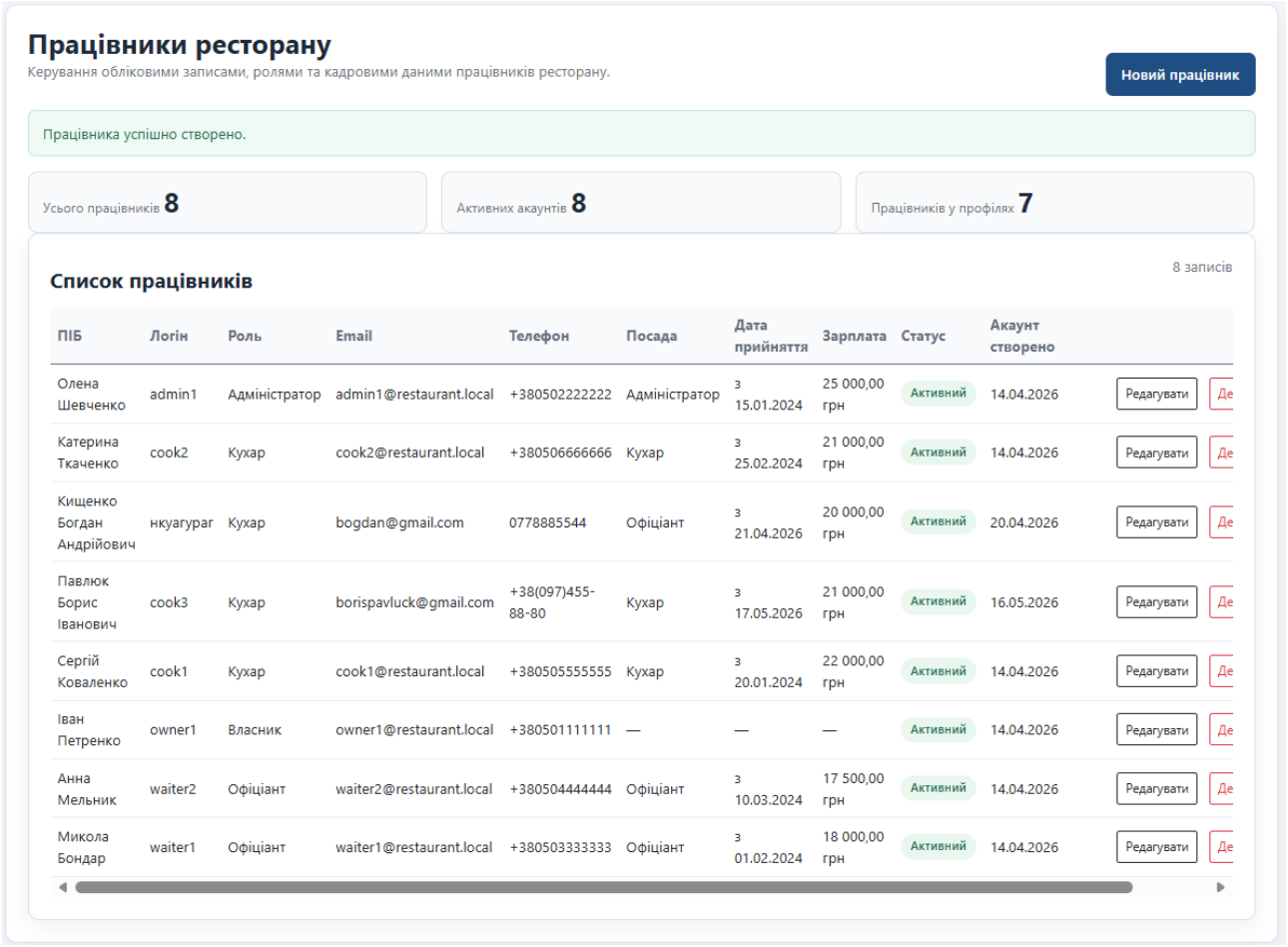


Рис. 28 Демонстрація переліку працівників ресторану

На рисунку 27 наведено демонстрацію форми реєстрації нового працівника в інформаційній системі ресторану. У даній формі передбачено введення основних облікових і кадрових даних працівника, зокрема логіна, ролі, прізвища, імені та по батькові, електронної адреси, номера телефону, пароля та його підтвердження. Окремо відображається блок кадрового профілю, у якому вказуються посада, дата прийняття на роботу, заробітна плата та примітка. Така структура форми дозволяє поєднати створення облікового запису працівника з одночасним заповненням кадрової інформації.

На рисунку 28 показано перелік працівників ресторану, сформований у вигляді таблиці. У списку відображаються основні відомості про кожного працівника: ПІБ, логін, роль, електронна адреса, номер телефону, посада, дата

прийняття на роботу, розмір заробітної плати, поточний статус облікового запису та дата створення акаунта. Також інтерфейс містить узагальнені показники щодо кількості працівників і активних акаунтів. Поданий перелік забезпечує зручний перегляд кадрової інформації та спрощує адміністрування працівників у межах системи.

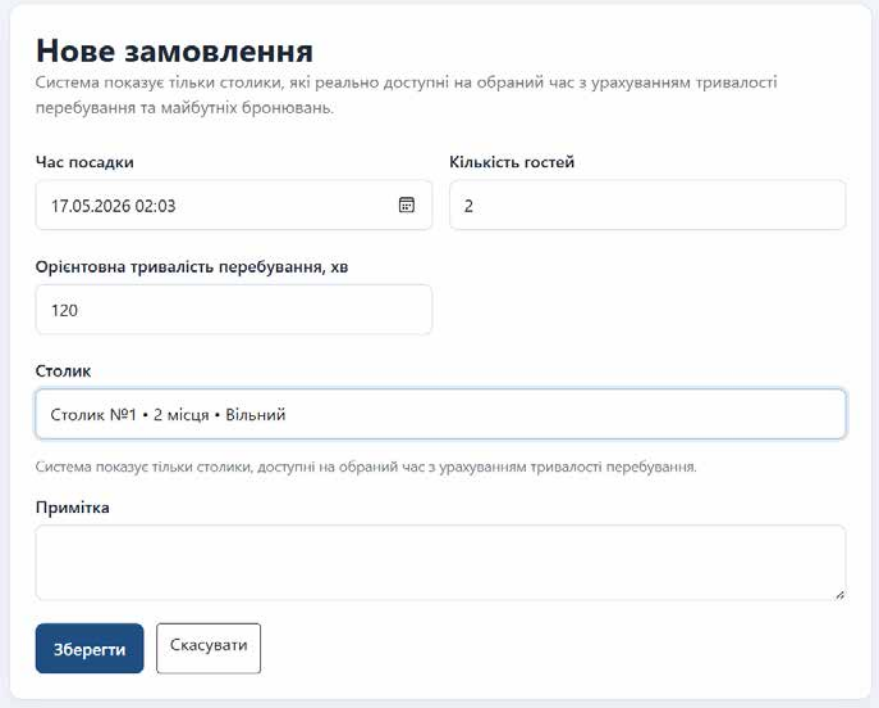


Рис. 29 Демонстрація створення нового замовлення

На рисунку наведено демонстрацію форми створення нового замовлення в інформаційній системі ресторану. У даному інтерфейсі користувач задає основні параметри обслуговування клієнтів, зокрема час посадки, кількість гостей, орієнтовну тривалість перебування та примітку до замовлення. Важливою особливістю форми є динамічний вибір столика, оскільки система відображає лише ті столики, які реально доступні на вказаний час з урахуванням майбутніх бронювань і прогнозованої тривалості перебування гостей.

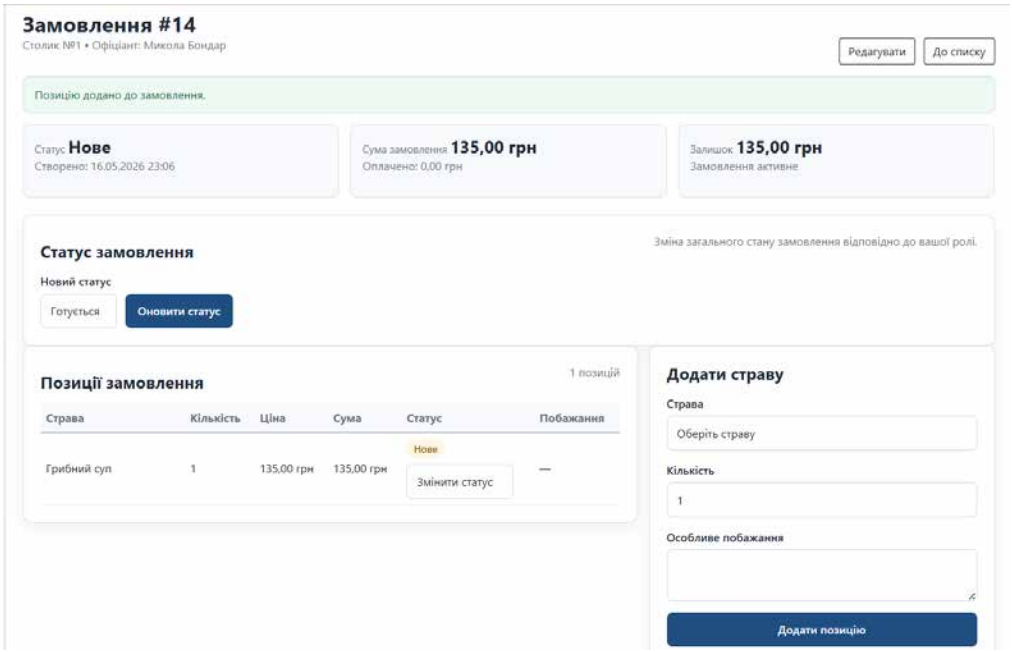


Рис. 30 Демонстрація інтерфейсу перегляду та редагування замовлення

На рисунку 30 наведено інтерфейс перегляду та редагування замовлення в інформаційній системі ресторану. У верхній частині сторінки відображаються основні відомості про замовлення, зокрема його номер, прив’язка до столика, офіціант, поточний статус, загальна сума та залишок до оплати. Це дає змогу швидко оцінити поточний стан обслуговування клієнта та фінансові параметри замовлення.

Нижче подано блок керування статусом замовлення, який дозволяє змінювати його загальний стан відповідно до ролі користувача. Окремо відображається таблиця позицій замовлення, де для кожної страви вказуються назва, кількість, ціна, сума, поточний статус та особливі побажання. Поряд розміщено форму додавання нової позиції до замовлення, що дає можливість оперативно доповнювати замовлення новими стравами без переходу до інших сторінок системи.

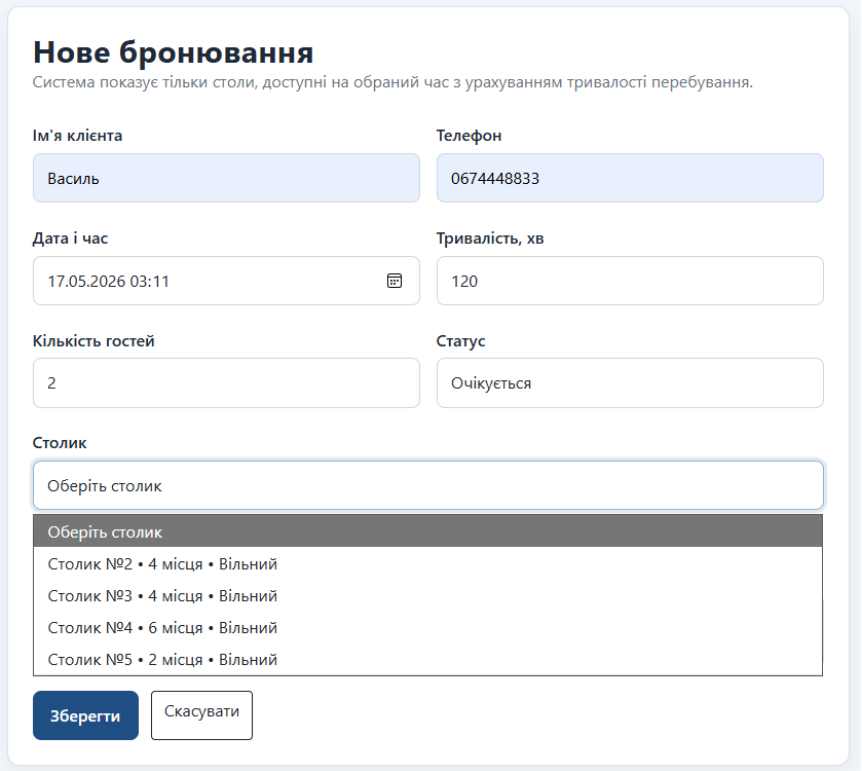


Рис. 31 Демонстрація створення та перегляду бронювання столика

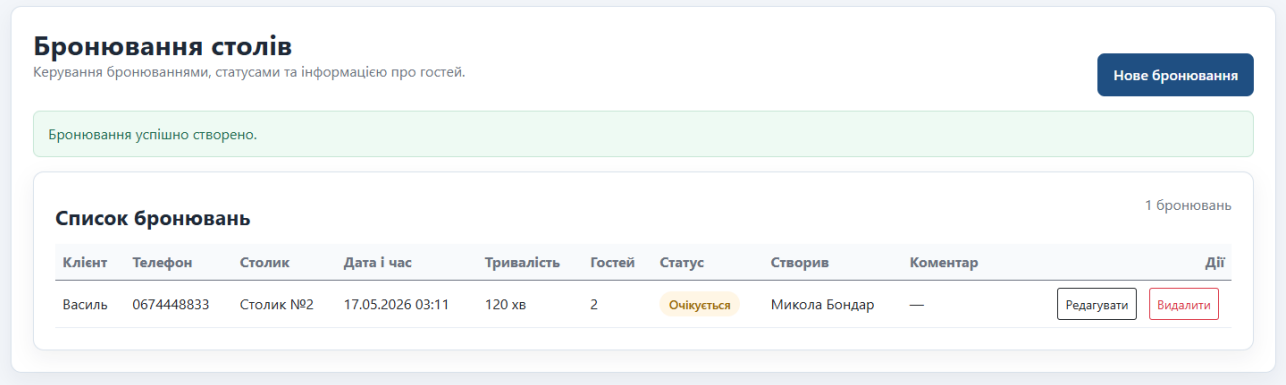


Рис. 32 Демонстрація переліку списку бронювань

На рисунку 31 наведено приклад роботи з модулем бронювання столиків в інформаційній системі ресторану. У верхній частині зображення показано форму створення нового бронювання, у якій користувач вводить основні дані про клієнта: ім’я, номер телефону, дату і час бронювання, тривалість перебування,

кількість гостей та статус бронювання. Особливістю даного інтерфейсу є те, що система відображає лише ті столики, які доступні на обраний час з урахуванням тривалості перебування, що дозволяє уникати конфліктів із вже наявними бронюваннями.

На рисунку 32 подано сторінку переліку бронювань, де відображаються збережені записи із зазначенням клієнта, контактного телефону, номера столика, дати і часу, тривалості, кількості гостей, статусу бронювання, користувача, який створив запис, а також доступних дій для редагування чи видалення. Подане зображення демонструє практичну реалізацію механізму резервування столиків, який поєднує введення даних, контроль доступності ресурсів залу та централізований облік створених бронювань.

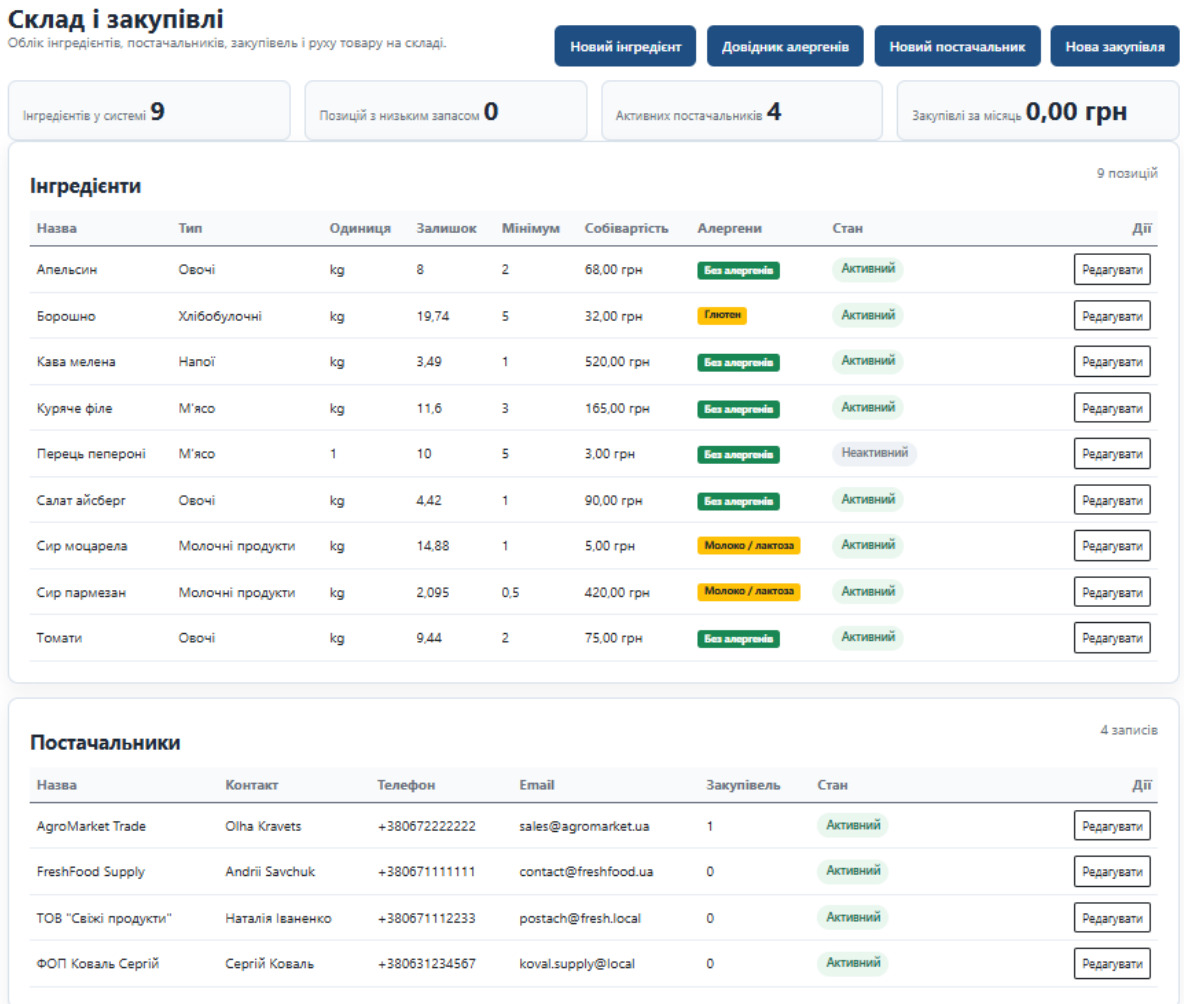
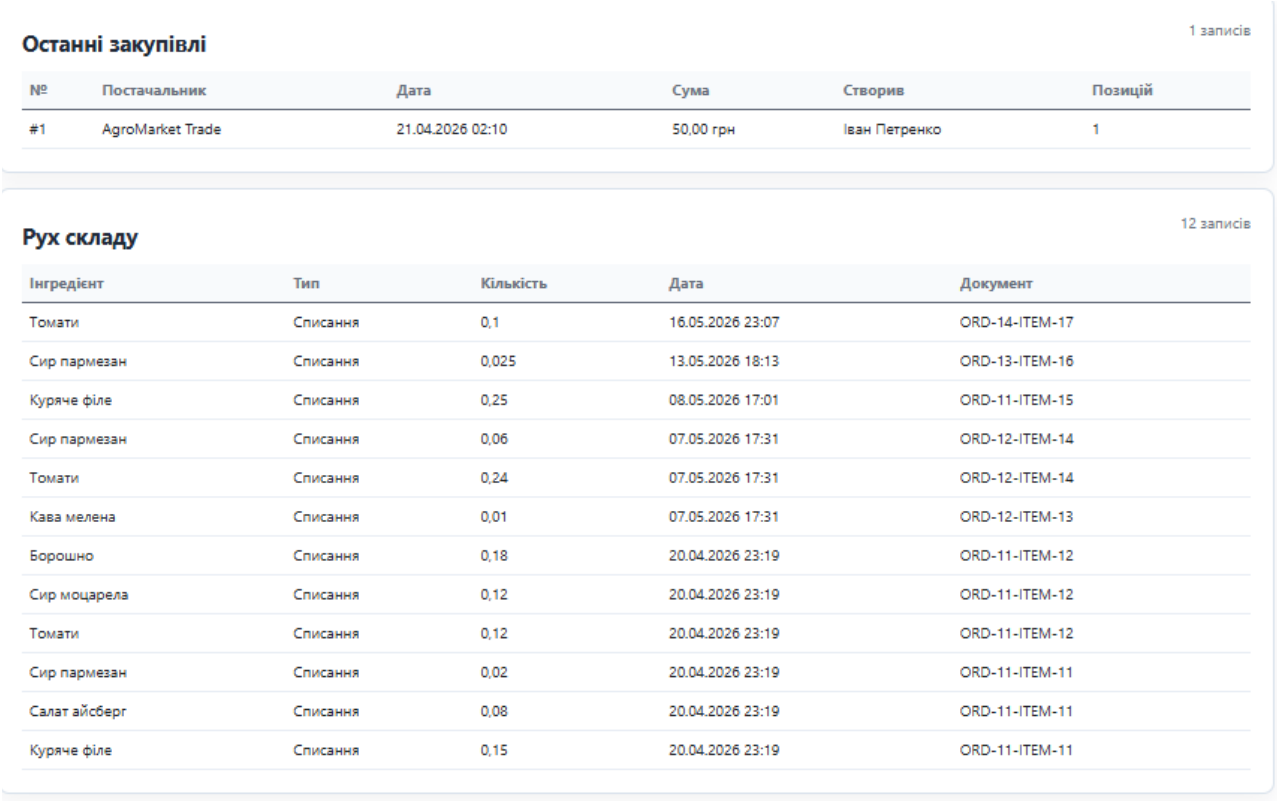


Рис. 33 Демонстрація модуля складського обліку та закупівель (фрагмент 1)



Останні закупівлі

1 записів

№	Постачальник	Дата	Сума	Створив	Позицій
#1	AgroMarket Trade	21.04.2026 02:10	50,00 грн	Іван Петренко	1

Рух складу

12 записів

Інгредієнт	Тип	Кількість	Дата	Документ
Томати	Списання	0,1	16.05.2026 23:07	ORD-14-ITEM-17
Сир пармезан	Списання	0,025	13.05.2026 18:13	ORD-13-ITEM-16
Куряче філе	Списання	0,25	08.05.2026 17:01	ORD-11-ITEM-15
Сир пармезан	Списання	0,06	07.05.2026 17:31	ORD-12-ITEM-14
Томати	Списання	0,24	07.05.2026 17:31	ORD-12-ITEM-14
Кава мелена	Списання	0,01	07.05.2026 17:31	ORD-12-ITEM-13
Борошно	Списання	0,18	20.04.2026 23:19	ORD-11-ITEM-12
Сир моцарела	Списання	0,12	20.04.2026 23:19	ORD-11-ITEM-12
Томати	Списання	0,12	20.04.2026 23:19	ORD-11-ITEM-12
Сир пармезан	Списання	0,02	20.04.2026 23:19	ORD-11-ITEM-11
Салат айсберг	Списання	0,08	20.04.2026 23:19	ORD-11-ITEM-11
Куряче філе	Списання	0,15	20.04.2026 23:19	ORD-11-ITEM-11

Рис. 34 Демонстрація модуля складського обліку та закупівель (фрагмент 1)

На рисунках наведено інтерфейс модуля складського обліку та закупівель інформаційної системи ресторану. У верхній частині сторінки відображено узагальнені показники, що характеризують поточний стан складського господарства, зокрема кількість інгредієнтів у системі, число позицій із низьким запасом, кількість активних постачальників та обсяг закупівель за місяць. Таке подання дає можливість швидко оцінити загальний стан запасів і постачання.

У центральній частині інтерфейсу подано перелік інгредієнтів із зазначенням їх типу, одиниці виміру, поточного залишку, мінімального запасу, собівартості, алергенів та статусу активності. Нижче відображається перелік постачальників із контактними даними та інформацією про кількість закупівель. Окремо подано блок останніх закупівель, де фіксуються дата, сума, відповідальний користувач та кількість позицій у закупівлі, а також блок руху

складу, який містить дані про списання інгредієнтів, дати операцій і пов’язані документи.

Поданий інтерфейс демонструє, що система забезпечує не лише зберігання довідникової інформації про інгредієнти й постачальників, а й повноцінний облік закупівель та контроль руху запасів. Це дозволяє підтримувати актуальний стан складських ресурсів, відстежувати господарські операції та забезпечувати достовірність даних у межах ресторанної інформаційної системи.

4.2 Вимоги до апаратного та програмного забезпечення

Для визначення вимог до функціонування інформаційної системи ресторану доцільно виходити з її топології розміщення. На діаграмі розміщення системи відображено основні вузли, між якими розподіляються функціональні компоненти: робоче місце власника, робоче місце адміністратора, робоче місце офіціанта, робоче місце кухаря, сервер застосунку, сервер бази даних та сервер резервного копіювання. Користувачі взаємодіють із системою через клієнтський вебінтерфейс, а основна бізнес-логіка зосереджена на сервері застосунку. Зберігання даних виконується на сервері бази даних RestaurantManagementDB, а для забезпечення надійності роботи передбачено окреме сховище резервних копій.

Така архітектура відповідає централізованій моделі побудови системи, у якій обробка запитів, виконання бізнес-логіки та доступ до даних здійснюються на серверному боці. На сервері застосунку розміщуються компоненти контролю доступу, керування замовленнями, бронюваннями, меню, складом, закупівлями та формуванням звітів. Робочі місця користувачів не потребують встановлення окремих спеціалізованих програмних модулів, оскільки доступ до функціональних можливостей системи здійснюється через веббраузер. Це спрощує адміністрування, зменшує вимоги до клієнтських пристроїв та забезпечує єдину точку оновлення програмного забезпечення.

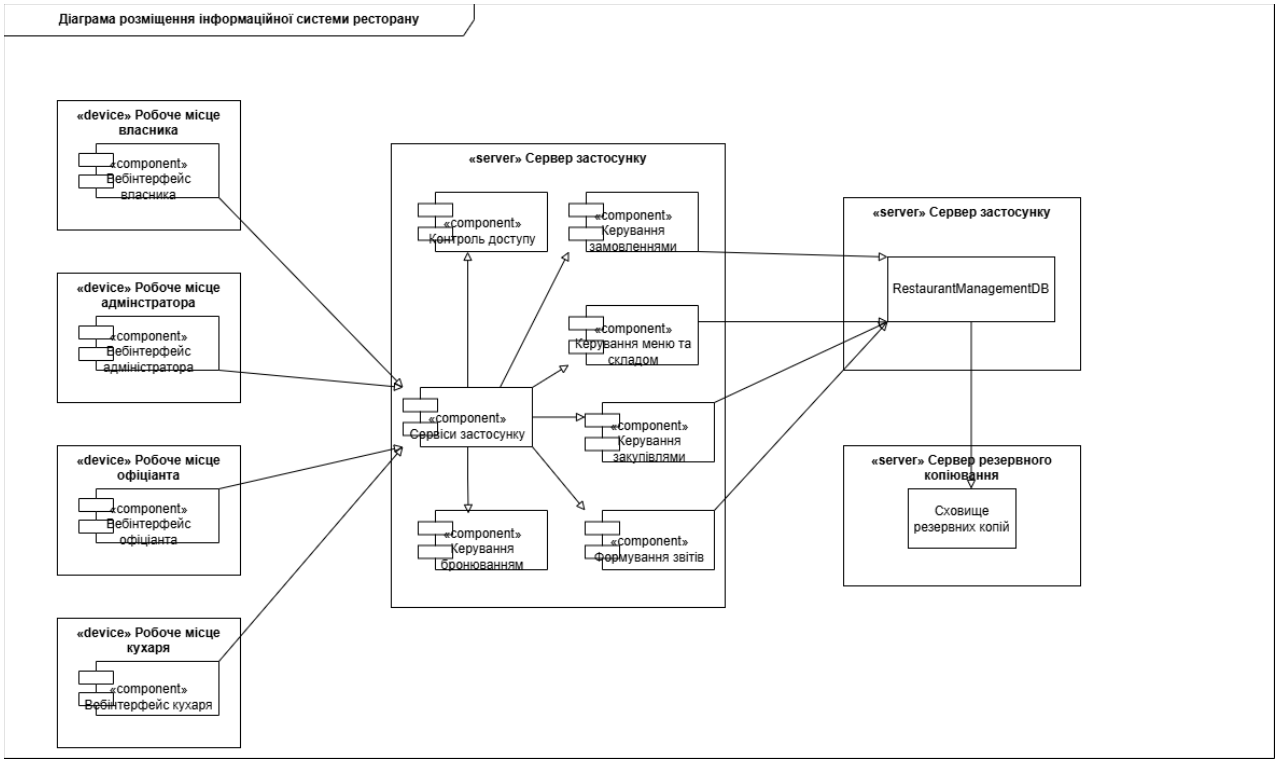


Рис. 35 Діаграма розгортання

Відповідно до наведеної топології до апаратного забезпечення клієнтських робочих місць висуваються помірні вимоги. Для роботи власника, адміністратора, офіціанта та кухаря достатньо персонального комп’ютера, ноутбука або планшета із сучасним процесором, оперативною пам’яттю не менше 4 ГБ, мережевим підключенням та екраном, що забезпечує зручну роботу з вебінтерфейсом. Для офіціанта доцільним є використання планшета або легкого мобільного пристрою, оскільки це підвищує оперативність роботи із замовленнями безпосередньо в залі ресторану. Для кухаря більш зручним є стаціонарний комп’ютер або моноблок, що забезпечує стабільний доступ до інформації про позиції замовлень і зміну їх статусу.

Сервер застосунку повинен забезпечувати стабільне виконання вебзастосунку та одночасне обслуговування кількох користувачів. Для цього доцільно використовувати сервер або робочу станцію з процесором рівня Intel Core i5 / AMD Ryzen 5 або вище, оперативною пам’яттю не менше 8 ГБ,

твердотілим накопичувачем SSD та стабільним мережевим з’єднанням. Якщо система експлуатується в межах одного закладу й кількість одночасних користувачів є невеликою, зазначених характеристик достатньо для її коректної роботи. Для масштабування системи в майбутньому серверні ресурси можуть бути збільшені залежно від навантаження.

До сервера бази даних висуваються підвищені вимоги щодо надійності зберігання інформації. База даних містить відомості про користувачів, працівників, замовлення, бронювання, меню, складські залишки, закупівлі, платежі та сформовані звіти, тому сервер повинен забезпечувати достатню швидкодію та стійкість до втрати даних. Рекомендовано використовувати окремий фізичний або віртуальний сервер із оперативною пам’яттю не менше 8 ГБ, SSD-накопичувачем та можливістю регулярного резервного копіювання. Для збереження резервних копій доцільно використовувати окремий носій або сервер резервного копіювання, що мінімізує ризик втрати даних у разі технічних збоїв.

Щодо програмного забезпечення, клієнтські робочі місця повинні мати операційну систему, яка підтримує сучасні веббраузери, наприклад Windows 10/11, Linux або Android для планшетних пристроїв. Для доступу до системи достатньо встановленого браузера Google Chrome, Microsoft Edge або Mozilla Firefox актуальної версії. Оскільки система реалізована як вебзастосунок, додаткове клієнтське програмне забезпечення не є обов’язковим.

Серверна частина системи потребує встановлення операційної системи, сумісної з платформою .NET та сервером баз даних. Для реалізації застосунку використовується середовище ASP.NET Core MVC, тому на сервері застосунку має бути розгорнуте відповідне середовище виконання .NET. Як система керування базами даних використовується Microsoft SQL Server, що забезпечує зберігання та обробку структурованих даних системи. Для адміністрування бази даних може використовуватися SQL Server Management Studio, а для супроводу та оновлення програмної частини — Visual Studio або інші сумісні засоби розроблення [4; 5; 12; 13].

4.3 Інсталяційний пакет

Для введення інформаційної системи ресторану в експлуатацію необхідно виконати її інсталяцію відповідно до діаграми розміщення. Оскільки система побудована за централізованою архітектурою, інсталяційний пакет має враховувати розподіл компонентів між клієнтськими робочими місцями, сервером застосунку, сервером бази даних та засобами резервного копіювання. Основна частина програмних компонентів розміщується на серверному боці, тоді як на клієнтських пристроях достатньо наявності веббраузера для доступу до функціональних можливостей системи [21; 22].

Інсталяційний пакет повинен містити всі файли та ресурси, необхідні для розгортання, налаштування й запуску системи. Його склад доцільно поділити на кілька логічних груп.

До складу інсталяційного пакету серверної частини входять:

- скомпільовані файли вебзастосунку RestaurantManagementSystem.Web;
- конфігураційні файли застосунку, зокрема appsettings.json та appsettings.Production.json;
- службові бібліотеки та залежності платформи .NET, необхідні для виконання застосунку;
- статичні ресурси вебінтерфейсу: стилі, сценарії, зображення, шрифти;
- файли представлень, контролерів і бізнес-логіки, якщо система розгортається у форматі framework-dependent deployment;
- файл або набір скриптів запуску вебзастосунку.
- До складу інсталяційного пакету бази даних входять:
- SQL-скрипт створення бази даних RestaurantManagementDB;
- SQL-скрипти створення таблиць, зв’язків, індексів та обмежень;
- SQL-скрипти для початкового наповнення довідників;
- за потреби, резервна копія бази даних у форматі .bak для швидкого розгортання;

- скрипти оновлення структури бази даних у разі повторної інсталяції або модернізації системи.

Відповідно до діаграми розміщення інсталяція виконується поетапно. На сервері бази даних розгортається база RestaurantManagementDB, створюється її структура та завантажуються початкові дані. На сервері застосунку розміщуються файли вебсистеми, виконується налаштування конфігураційних параметрів, насамперед рядка підключення до бази даних, після чого застосунок приводиться у робочий стан. На клієнтських робочих місцях окрема інсталяція спеціалізованого програмного забезпечення не потрібна, оскільки доступ до системи здійснюється через браузер. Для забезпечення надійності роботи окремо налаштовується механізм резервного копіювання бази даних і збереження резервних копій у відповідному сховищі [25].

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено інформаційну систему ресторану, призначену для автоматизації основних операційних, облікових та аналітичних процесів закладу. У межах роботи досягнуто поставленої мети, яка полягала у створенні програмного продукту для централізованого керування замовленнями, бронюваннями, меню, складськими залишками, закупівлями, працівниками та звітністю. Отримані результати підтверджують, що розроблена система відповідає завданню на бакалаврську кваліфікаційну роботу та може бути використана як основа для практичного впровадження в діяльність ресторану.

У процесі дослідження було проаналізовано предметну область, визначено основні функціональні вимоги та побудовано архітектуру системи. Практичним результатом роботи стало створення централізованого веборієнтованого застосунку на базі ASP.NET Core MVC і Microsoft SQL Server. До складу реалізованої системи увійшли модулі контролю доступу, керування замовленнями, бронюваннями, меню, працівниками, складом, закупівлями, аналітикою та журналом аудиту. Це дозволило сформувати цілісне програмне рішення, у якому всі основні процеси ресторану об’єднані в межах єдиного інформаційного середовища.

Одним із важливих результатів роботи є реалізація прикладної бізнес-логіки, яка виходить за межі простого збереження даних. У системі реалізовано динамічну перевірку доступності столиків з урахуванням часових конфліктів між бронюваннями та активними замовленнями, автоматичне списання інгредієнтів зі складу відповідно до рецептур під час додавання позицій до замовлення, а також повернення інгредієнтів у разі скасування позиції. Окремо реалізовано механізм формування звіту прибутковості меню, що дозволяє визначати виручку, собівартість і прибуток за вибраний період.

З погляду техніко-економічної ефективності розроблена система дає змогу зменшити кількість ручних операцій у роботі персоналу, скоротити час оформлення замовлень і бронювань, підвищити точність складського обліку та покращити контроль за господарськими операціями. Централізоване зберігання даних і використання журналу аудиту забезпечують додаткову надійність та прозорість роботи системи. Використання єдиного вебзастосунку також спрощує адміністрування, оновлення та експлуатацію програмного продукту.

Порівняння отриманих результатів із поставленим завданням показує, що всі основні функціональні вимоги були реалізовані. Система забезпечує автентифікацію та розмежування прав доступу, ведення меню, реєстрацію працівників, створення й супровід замовлень, бронювання столиків, ведення складського обліку, оформлення закупівель, реєстрацію оплат та формування звітів. У порівнянні з сучасними аналогічними системами розроблений програмний продукт поступається комерційним рішенням за рівнем інтеграції із зовнішніми сервісами, фіскальним обладнанням і мобільними платформами, однак має суттєву перевагу як цілісна навчально-практична система, що об’єднує в одному середовищі операційні, складські та аналітичні процеси ресторану.

Новим технічним рішенням, покладеним в основу роботи, є побудова централізованої веборієнтованої інформаційної системи ресторану з інтеграцією модулів замовлень, бронювань, складу, закупівель та аналітики в межах єдиної бази даних. Перевага такого підходу полягає в тому, що зміни в одному модулі безпосередньо відображаються в інших модулях, що забезпечує узгодженість даних і підвищує точність обліку. Особистий внесок полягає у розробленні структури бази даних, архітектури програмного рішення, бізнес-логіки системи та користувацького інтерфейсу, а також у реалізації механізмів перевірки доступності столиків, автоматичного списання і повернення інгредієнтів, перерахунку статусів замовлень та формування звітів прибутковості меню.

Отримані результати можуть бути використані в діяльності малих і середніх закладів ресторанного господарства, а також як основа для подальшого

вдосконалення подібних програмних рішень. До перспективних напрямів розвитку системи можна віднести інтеграцію з платіжними сервісами та фіскальним обладнанням, розширення аналітичного модуля, впровадження механізмів автоматичних сповіщень для клієнтів і створення мобільного інтерфейсу для персоналу. Результати виконаної роботи підтверджують практичну цінність розробленої системи, її відповідність поставленому завданню та можливість подальшого розвитку відповідно до потреб реального закладу ресторанного господарства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Трофименко О. Г., Прокопенко О. В. Бази даних та системи управління базами даних : навчальний посібник. – Київ : Видавництво Ліра-К, 2021. – 304 с. (дата звернення: 01.05.2026).
2. Фаронов В. В. ASP.NET Core MVC. Розробка сучасних веб-додатків. – Харків : Ранок, 2022. – 512 с. (дата звернення: 01.05.2026).
3. Date C. J. An Introduction to Database Systems. – 8th ed. – Boston : Pearson Education, 2019. – 1024 p. (дата звернення: 01.05.2026).
4. Microsoft ASP.NET Core Documentation – офіційна документація платформи ASP.NET Core для розробки веб-застосунків. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-10.0 (дата звернення: 01.05.2026).
5. Microsoft SQL Server Documentation – офіційна документація системи управління базами даних Microsoft SQL Server. URL: https://learn.microsoft.com/en-us/visualstudio/?view=visualstudio (дата звернення: 01.05.2026).
6. Гужва В. М. Інформаційні системи і технології на підприємствах : навч. посіб. — Київ : КНЕУ, 2019. — 400 с. (дата звернення: 23.04.2026).
7. Microsoft Entity Framework Core Documentation – офіційна документація Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 01.05.2026)
8. Ситник В. Ф. Основи інформаційних систем : навч. посіб. – Київ : КНЕУ, 2020. – 420 с. (дата звернення: 23.04.2026).
9. Elmasri R., Navathe S. Fundamentals of Database Systems. – 7th ed. – Boston : Pearson, 2017. – 1272 p. (дата звернення: 02.05.2026).
10. Microsoft C# Documentation – офіційна документація мови програмування C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 01.05.2026).
11. Bootstrap Documentation – офіційна документація Bootstrap. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 27.04.2026).
12. SQL Server Management Studio Documentation. URL: https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms (дата звернення: 10.02.2026).

13. Visual Studio Documentation. URL: https://learn.microsoft.com/en-us/visualstudio/?view=visualstudio (дата звернення: 18.04.2026).
14. Object Management Group. UML Specification Version 2.5. URL: https://www.omg.org/spec/UML/2.5/ (дата звернення: 11.05.2026).
15. Microsoft Learn. ASP.NET Core MVC Tutorial – офіційний навчальний курс з розробки веб-застосунків на ASP.NET Core MVC. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-10.0&tabs=visual-studio (дата звернення: 01.03.2026).
16. Microsoft Learn. Authentication and Authorization in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-10.0 (дата звернення: 26.04.2026).
17. Microsoft Learn. Working with Data in EF Core. URL: https://learn.microsoft.com/en-us/ef/core/querying/ (дата звернення: 28.04.2026).
18. Microsoft Learn. Logging in .NET and ASP.NET Core. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/logging/overview?tabs=command-line (дата звернення: 10.04.2026).
19. Бородкіна І.Л., Бородкін Г.О. Теорія алгоритмів: Навчальний посібник. – Київ : Центр учбової літератури, 2018. – 184 с. (дата звернення: 08.02.2026).
20. Бородкіна І.Л., Бородкін Г.О. Інженерія програмного забезпечення: Навчальний посібник. – Київ : Центр учбової літератури, 2018. - 204 с. (дата звернення: 08.02.2026).
21. MDN Web Docs. Responsive design. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design (дата звернення: 26.04.2026).
22. Microsoft Learn. Views in ASP.NET Core MVC. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/overview (дата звернення: 17.04.2026).
23. Mozilla Developer Network. CSS Grid Layout. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Flexbox (дата звернення: 26.04.2026).
24. Microsoft. Installing Entity Framework Core. URL: Installing Entity Framework Core - EF Core | Microsoft Learn (дата звернення: 23.02.2026).
25. Microsoft. Create a Full Database Backup – SQL Server. URL: Create a Full Database Backup - SQL Server | Microsoft Learn (дата звернення: 23.02.2026).

ДОДАТОК А

Код сервісу перевірки доступності столиків

```
using Microsoft.EntityFrameworkCore;
using RestaurantManagementSystem.Domain.Entities;
using RestaurantManagementSystem.Domain.Enums;
using RestaurantManagementSystem.Infrastructure.Persistence;

namespace RestaurantManagementSystem.Web.Services;

public sealed class TableAvailabilityService
{
    public const int BufferMinutes = 15;
    public const int DefaultReservationDurationMinutes = 120;
    public const int DefaultOrderDurationMinutes = 120;

    private readonly ApplicationDbContext _dbContext;

    public TableAvailabilityService(ApplicationDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task NormalizeLegacyReservedStatusesAsync(CancellationToken cancellationToken)
    {
        var reservedTables = await _dbContext.DiningTables
            .Where(x => x.Status == TableStatus.Reserved)
            .ToListAsync(cancellationToken);

        if (reservedTables.Count == 0)
        {
            return;
        }

        foreach (var table in reservedTables)
        {
            table.Status = TableStatus.Free;
        }

        await _dbContext.SaveChangesAsync(cancellationToken);
    }

    public async Task<TableAvailabilityResult> IsTableAvailableAsync(
        int tableId,
        DateTime startDateTime,
        int durationMinutes,
        int guestsCount,
        int? excludeReservationId,
        CancellationToken cancellationToken)
    {
        var table = await _dbContext.DiningTables
            .AsNoTracking()
            .FirstOrDefaultAsync(x => x.TableId == tableId, cancellationToken);

        if (table is null)
        {
            return TableAvailabilityResult.Unavailable("Обраний столик не існує.");
        }
```

```
    return await EvaluateTableAsync(
        table,
        startDateTime,
        durationMinutes,
        guestsCount,
        excludeReservationId,
        null,
        cancellationToken);
}

public async Task<IReadOnlyCollection<TableAvailabilityOption>> GetAvailableTablesAsync(
    DateTime startDateTime,
    int durationMinutes,
    int guestsCount,
    int? excludeReservationId,
    CancellationToken cancellationToken)
{
    var tables = await _dbContext.DiningTables
        .AsNoTracking()
        .OrderBy(x => x.TableNumber)
        .ToListAsync(cancellationToken);

    var result = new List<TableAvailabilityOption>();

    foreach (var table in tables)
    {
        var availability = await EvaluateTableAsync(
            table,
            startDateTime,
            durationMinutes,
            guestsCount,
            excludeReservationId,
            null,
            cancellationToken);

        result.Add(BuildOption(table, availability));
    }

    return result;
}

public async Task<IReadOnlyCollection<TableAvailabilityOption>> GetAvailableTablesForWalkInAsync(
    DateTime startTime,
    int durationMinutes,
    int guestsCount,
    int? ignoreOrderId,
    CancellationToken cancellationToken)
{
    var tables = await _dbContext.DiningTables
        .AsNoTracking()
        .OrderBy(x => x.TableNumber)
        .ToListAsync(cancellationToken);

    var result = new List<TableAvailabilityOption>();

    foreach (var table in tables)
    {
        var availability = await EvaluateTableAsync(
            table,
            startTime,
            durationMinutes,
            guestsCount,
            null,
```

```
            ignoreOrderId,
            cancellationToken);

        result.Add(BuildOption(table, availability));
    }

    return result;
}

public async Task<TableAvailabilityResult> IsTableAvailableForOrderAsync(
    int tableId,
    DateTime startDateTime,
    int expectedDurationMinutes,
    int guestsCount,
    int? ignoreOrderId,
    CancellationToken cancellationToken)
{
    var table = await _dbContext.DiningTables
        .AsNoTracking()
        .FirstOrDefaultAsync(x => x.TableId == tableId, cancellationToken);

    if (table is null)
    {
        return TableAvailabilityResult.Unavailable("Обраний столик не існує.");
    }

    return await EvaluateTableAsync(
        table,
        startDateTime,
        expectedDurationMinutes,
        guestsCount,
        null,
        ignoreOrderId,
        cancellationToken);
}

private async Task<TableAvailabilityResult> EvaluateTableAsync(
    DiningTable table,
    DateTime startDateTime,
    int durationMinutes,
    int guestsCount,
    int? excludeReservationId,
    int? ignoreOrderId,
    CancellationToken cancellationToken)
{
    if (table.SeatsCount < guestsCount)
    {
        return TableAvailabilityResult.Unavailable("Недостатньо місць за столиком.");
    }

    if (table.Status is TableStatus.Cleaning or TableStatus.OutOfService)
    {
        return TableAvailabilityResult.Unavailable("Цей столик зараз фізично недоступний.");
    }

    var requestedStart = startDateTime;
    var requestedEnd = startDateTime.AddMinutes(durationMinutes + BufferMinutes);

    var conflictingReservation = await _dbContext.Reservations
        .AsNoTracking()
        .Where(x =>
            x.TableId == table.TableId
            && x.ReservationId != excludeReservationId
            && (x.Status == ReservationStatus.Pending || x.Status == ReservationStatus.Confirmed))
```

```
            .OrderBy(x => x.ReservationDateTime)
            .FirstOrDefaultAsync(
                x => x.ReservationDateTime < requestedEnd
                    && x.ReservationDateTime.AddMinutes(x.DurationMinutes + BufferMinutes) > requestedStart,
                cancellationToken);

        if (conflictingReservation is not null)
        {
            return TableAvailabilityResult.Unavailable(
                "Обраний столик недоступний на вказаний час.",
                conflictingReservation.ReservationDateTime.AddMinutes(conflictingReservation.DurationMinutes +
BufferMinutes));
        }

        var activeOrder = await _dbContext.Orders
            .AsNoTracking()
            .Where(x =>
                x.TableId == table.TableId
                && x.OrderId != ignoreOrderId
                && x.Status != OrderStatus.Closed
                && x.Status != OrderStatus.Cancelled)
            .OrderBy(x => x.CreatedAt)
            .FirstOrDefaultAsync(cancellationToken);

        if (activeOrder is not null)
        {
            var activeOrderStart = activeOrder.CreatedAt.Kind == DateTimeKind.Utc
                ? activeOrder.CreatedAt.ToLocalTime()
                : activeOrder.CreatedAt;

            if (activeOrderStart < DateTime.Now)
            {
                activeOrderStart = DateTime.Now;
            }

            var activeOrderEnd = activeOrderStart.AddMinutes(DefaultOrderDurationMinutes + BufferMinutes);

            if (activeOrderStart < requestedEnd && activeOrderEnd > requestedStart)
            {
                return TableAvailabilityResult.Unavailable(
                    "За цим столиком уже є активне замовлення.",
                    activeOrderEnd);
            }
        }

        var nextReservationStart = await _dbContext.Reservations
            .AsNoTracking()
            .Where(x =>
                x.TableId == table.TableId
                && x.ReservationId != excludeReservationId
                && (x.Status == ReservationStatus.Pending || x.Status == ReservationStatus.Confirmed)
                && x.ReservationDateTime >= requestedStart)
            .OrderBy(x => x.ReservationDateTime)
            .Select(x => (DateTime?)x.ReservationDateTime)
            .FirstOrDefaultAsync(cancellationToken);

        return TableAvailabilityResult.Available(nextReservationStart);
    }

    private static TableAvailabilityOption BuildOption(DiningTable table, TableAvailabilityResult availability)
    {
        var availabilityText = availability.IsAvailable
            ? availability.AvailableUntil.HasValue
                ? $"Доступний до {availability.AvailableUntil.Value:HH:mm}"
```

```
                : "Вільний"
            : availability.AvailableUntil.HasValue
                ? $"Недоступний до {availability.AvailableUntil.Value:HH:mm}"
                : "Недоступний";

        return new TableAvailabilityOption
        {
            TableId = table.TableId,
            TableNumber = table.TableNumber,
            SeatsCount = table.SeatsCount,
            IsAvailable = availability.IsAvailable,
            AvailableUntil = availability.AvailableUntil,
            Reason = availability.Reason,
            Label = $"Столик №{table.TableNumber} • {table.SeatsCount} місця • {availabilityText}"
        };
    }
}

public sealed class TableAvailabilityOption
{
    public int TableId { get; init; }
    public int TableNumber { get; init; }
    public int SeatsCount { get; init; }
    public bool IsAvailable { get; init; }
    public DateTime? AvailableUntil { get; init; }
    public string? Reason { get; init; }
    public string Label { get; init; } = string.Empty;
}

public sealed class TableAvailabilityResult
{
    public bool IsAvailable { get; init; }
    public string? Reason { get; init; }
    public DateTime? AvailableUntil { get; init; }

    public static TableAvailabilityResult Available(DateTime? availableUntil = null) => new()
    {
        IsAvailable = true,
        AvailableUntil = availableUntil
    };

    public static TableAvailabilityResult Unavailable(string reason, DateTime? availableUntil = null) => new()
    {
        IsAvailable = false,
        Reason = reason,
        AvailableUntil = availableUntil
    };
}
```

ДОДАТОК Б

Фрагмент коду формування звіту прибутковості меню

```
private async Task<GeneratedReportContentViewModel> BuildMenuProfitReportAsync(
    DateTime periodStart,
    DateTime periodEndExclusive,
    CultureInfo moneyCulture,
    CancellationToken cancellationToken)
{
    var soldItems = await _dbContext.OrderItems
        .AsNoTracking()
        .Where(x => x.Order != null
            && x.Order.CreatedAt >= periodStart
            && x.Order.CreatedAt < periodEndExclusive
            && x.Status != OrderItemStatus.Cancelled)
        .Select(x => new
        {
            x.MenuItemId,
            Name = x.MenuItem != null ? x.MenuItem.Name : "Страва",
            CategoryName = x.MenuItem != null && x.MenuItem.Category != null ? x.MenuItem.Category.Name : "Без
категорії",
            Price = x.UnitPrice,
            x.Quantity,
            x.LineTotal
        })
        .ToListAsync(cancellationToken);

    var menuItemIds = soldItems
        .Select(x => x.MenuItemId)
        .Distinct()
        .ToList();

    var costPriceByMenuItemId = await _dbContext.DishIngredients
        .AsNoTracking()
        .Where(x => menuItemIds.Contains(x.MenuItemId))
        .GroupBy(x => x.MenuItemId)
        .Select(x => new
        {
            MenuItemId = x.Key,
            CostPrice = x.Sum(y => y.QuantityNeeded * (y.Ingredient != null ? y.Ingredient.CostPerUnit : 0m))
        })
        .ToDictionaryAsync(x => x.MenuItemId, x => x.CostPrice, cancellationToken);
    var rows = soldItems
        .GroupBy(x => new
        {
            x.MenuItemId,
            x.Name,
            x.CategoryName,
            x.Price
        })
        .Select(x =>
        {
            var soldQuantity = x.Sum(y => y.Quantity);
            var revenue = x.Sum(y => y.LineTotal);
            var unitCostPrice = costPriceByMenuItemId.GetValueOrDefault(x.Key.MenuItemId);
            var totalCostPrice = soldQuantity * unitCostPrice;
            return new
            {
```

```
                x.Key.Name,
                x.Key.CategoryName,
                x.Key.Price,
                Quantity = soldQuantity,
                Revenue = revenue,
                CostPrice = totalCostPrice
            };
        })
        .OrderByDescending(x => x.Revenue)
        .Take(20)
        .ToList();

    var totalRevenue = rows.Sum(x => x.Revenue);
    var totalCost = rows.Sum(x => x.CostPrice);

    return new GeneratedReportContentViewModel
    {
        Metrics =
        [
            new ReportMetricItemViewModel { Label = "Страв у звіті", Value = rows.Count.ToString() },
            new ReportMetricItemViewModel { Label = "Загальна виручка", Value = $"{totalRevenue.ToString("N2",
moneyCulture)} грн", OwnerOnly = true },
            new ReportMetricItemViewModel { Label = "Загальна собівартість", Value = $"{totalCost.ToString("N2",
moneyCulture)} грн", OwnerOnly = true },
            new ReportMetricItemViewModel { Label = "Загальний прибуток", Value = $"{(totalRevenue -
totalCost).ToString("N2", moneyCulture)} грн", OwnerOnly = true }
        ],
        Sections =
        [
            new ReportTableSectionViewModel
            {
                Title = "Прибутковість меню",
                EmptyMessage = "За обраний період дані по стравах відсутні.",
                Columns =
                [
                    new ReportTableColumnViewModel { Key = "name", Label = "Страва" },
                    new ReportTableColumnViewModel { Key = "category", Label = "Категорія" },
                    new ReportTableColumnViewModel { Key = "quantity", Label = "Продано" },
                    new ReportTableColumnViewModel { Key = "price", Label = "Ціна", OwnerOnly = true },
                    new ReportTableColumnViewModel { Key = "cost", Label = "Собівартість", OwnerOnly = true },
                    new ReportTableColumnViewModel { Key = "profit", Label = "Прибуток", OwnerOnly = true }
                ],
                Rows = rows.Select(x => new Dictionary<string, string>
                {
                    ["name"] = x.Name,
                    ["category"] = x.CategoryName,
                    ["quantity"] = x.Quantity.ToString(),
                    ["price"] = $"{x.Price.ToString("N2", moneyCulture)} грн",
                    ["cost"] = $"{x.CostPrice.ToString("N2", moneyCulture)} грн",
                    ["profit"] = $"{(x.Revenue - x.CostPrice).ToString("N2", moneyCulture)} грн"
                }).ToList()
            }
        ]
    };
}
```

ДОДАТОК В

Фрагмент коду модуля створення та видалення закупівель

```
public async Task<IActionResult> CreatePurchase(PurchaseFormViewModel model, CancellationToken cancellationToken)
{
   model = await BuildPurchaseFormAsync(model, cancellationToken);

   if (!TryParseDecimal(model.Quantity, out var quantity) || quantity <= 0)
   {
      ModelState.AddModelError(nameof(model.Quantity), "Вкажіть коректну кількість.");
   }

   if (!TryParseDecimal(model.UnitPrice, out var unitPrice) || unitPrice < 0)
   {
      ModelState.AddModelError(nameof(model.UnitPrice), "Вкажіть коректну ціну.");
   }

   if (!DateTime.TryParseExact(model.PurchasedAt.Trim(), "yyyy-MM-ddTHH:mm", CultureInfo.InvariantCulture,
DateTimeStyles.None, out var purchasedAt))
   {
      ModelState.AddModelError(nameof(model.PurchasedAt), "Вкажіть коректну дату закупівлі.");
   }

   if (!ModelState.IsValid)
   {
      return View("PurchaseForm", model);
   }

   var currentUserId = GetCurrentUserId();
   if (currentUserId is null)
   {
      return Forbid();
   }

   var supplier = await _dbContext.Suppliers.FirstOrDefaultAsync(x => x.SupplierId == model.SupplierId && x.IsActive,
cancellationToken);
   var ingredient = await _dbContext.Ingredients.FirstOrDefaultAsync(x => x.IngredientId == model.IngredientId &&
x.IsActive, cancellationToken);

   if (supplier is null)
   {
      ModelState.AddModelError(nameof(model.SupplierId), "Оберіть активного постачальника.");
   }

   if (ingredient is null)
   {
      ModelState.AddModelError(nameof(model.IngredientId), "Оберіть активний інгредієнт.");
   }

   if (!ModelState.IsValid)
   {
      return View("PurchaseForm", model);
   }

   var lineTotal = quantity * unitPrice;

   var purchase = new Purchase
   {
```

```
            SupplierId = supplier!.SupplierId,
            PurchasedAt = purchasedAt,
            TotalAmount = lineTotal,
            CreatedByUserId = currentUserId.Value,
            Comment = string.IsNullOrWhiteSpace(model.Comment) ? null : model.Comment.Trim()
        };

        _dbContext.Purchases.Add(purchase);
        await _dbContext.SaveChangesAsync(cancellationToken);

        _dbContext.PurchaseItems.Add(new PurchaseItem
        {
            PurchaseId = purchase.PurchaseId,
            IngredientId = ingredient!.IngredientId,
            Quantity = quantity,
            UnitPrice = unitPrice,
            LineTotal = lineTotal
        });

        ingredient.QuantityInStock += quantity;
        ingredient.CostPerUnit = unitPrice;

        _dbContext.StockMovements.Add(new StockMovement
        {
            IngredientId = ingredient.IngredientId,
            MovementType = StockMovementType.In,
            Quantity = quantity,
            MovementDate = DateTime.UtcNow,
            SourceDocument = $"PUR-{purchase.PurchaseId}",
            Comment = $"Закупівля від постачальника {supplier.Name}"
        });

        AddAuditLog(currentUserId.Value, "Створення закупівлі", "Purchases", purchase.PurchaseId,
            $"Створено закупівлю #{purchase.PurchaseId} для інгредієнта {ingredient.Name}.");

        await _dbContext.SaveChangesAsync(cancellationToken);

        TempData["InventoryMessage"] = "Закупівлю успішно створено, залишки оновлено.";
        return RedirectToAction(nameof(Index));
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeletePurchase(int purchaseId, CancellationToken cancellationToken)
    {
        var purchase = await _dbContext.Purchases
            .Include(x => x.PurchaseItems)
                .ThenInclude(x => x.Ingredient)
            .FirstOrDefaultAsync(x => x.PurchaseId == purchaseId, cancellationToken);

        if (purchase is null)
        {
            return NotFound();
        }

        foreach (var item in purchase.PurchaseItems)
        {
            if (item.Ingredient is not null)
            {
                item.Ingredient.QuantityInStock = Math.Max(0, item.Ingredient.QuantityInStock - item.Quantity);
            }
        }

        var movements = await _dbContext.StockMovements
```

```
        .Where(x => x.SourceDocument == $"PUR-{purchaseId}")
        .ToListAsync(cancellationToken);

    _dbContext.StockMovements.RemoveRange(movements);
    _dbContext.PurchaseItems.RemoveRange(purchase.PurchaseItems);
    _dbContext.Purchases.Remove(purchase);

    AddAuditLog(GetCurrentUserId(), "Видалення закупівлі", "Purchases", purchaseId,
        $"Видалено закупівлю #{purchaseId} та скориговано складські залишки.");

    await _dbContext.SaveChangesAsync(cancellationToken);

    TempData["InventoryMessage"] = "Закупівлю успішно видалено.";
    return RedirectToAction(nameof(Index));
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteStockMovement(int stockMovementId, CancellationToken cancellationToken)
{
    var movement = await _dbContext.StockMovements
        .Include(x => x.Ingredient)
        .FirstOrDefaultAsync(x => x.StockMovementId == stockMovementId, cancellationToken);

    if (movement is null)
    {
        return NotFound();
                }
```

ДОДАТОК Г

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Кищенко Богдан Андрійович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Інформаційна система обліку процесів для власника ресторану» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [X] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [X] стилістичного редагування та перевірки граматики;
        - [X] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Богдан Кищенко**

(підпис)