Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту

Завідувач
кафедри комп'ютерних наук,
_____к.т.н., доцент_____
(науковий ступінь, вчене звання)

_____ Ірина Машкіна
(підпис)
« ___ » _____ 202__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 Комп'ютерні науки
Освітня програма 122.00.01 Інформатика

Тема:
РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ВНУТРІШНІХ ПРОЦЕСІВ УПРАВЛІННЯ У РЕСТОРАННОМУ БІЗНЕСІ

Виконав
студент групи ІНб-1-21-4.0д
___Казанцев Микита Ігорович_
(ПІБ)

_____
(підпис)

Науковий керівник:
_____к.т.н., доцент_____
(науковий ступінь, вчене звання)

_____ Євген Іваніченко
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»

Завідувач
кафедри комп'ютерних наук,
к.т.н., доцент
(науковий ступінь, вчене звання)

_____ Ірина Машкіна
(підпис)
« ___ » _____ 202__ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

«Розробка інформаційної системи для автоматизації внутрішніх процесів управління у ресторанному бізнесі»
Виконавець – студент групи ІНб-1-21-4.0д,
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
Казанцев Микита Ігорович

1. Вихідні дані: *Розроблена інформаційна система для автоматизації внутрішніх процесів управління у сфері ресторанного бізнесу.*
2. Основні завдання: *Виявити ключові проблеми в організації внутрішніх процесів управління у сфері ресторанного бізнесу. Ознайомитися з існуючими рішеннями, провести їх порівняльний аналіз, виокремити сильні та слабкі сторони. На основі отриманих висновків сформувати вимоги до власної інформаційної системи, визначити її функціональні та архітектурні особливості. Обґрунтувати вибір технологічного стеку для реалізації клієнтської та серверної частин застосунку. Розробити архітектуру системи, з урахуванням принципів масштабованості, підтримуваності та модульності.*
3. Пояснювальна записка: *Обсяг – до 70 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.*
4. Графічні матеріали: *не передбачено.*
5. Додатки: *лістинг програми.*
6. Строк подання роботи на кафедру: *«___» _____2025 р.*

Науковий керівник
к.т.н., доцент
_____ Іваніченко Є.В.
дата

Виконавець:
_____ Казанцев М.І.
дата

Календарний план

№	Назва етапу дипломної роботи	Строк виконання етапу роботи	Примітка
1	Затвердження теми кваліфікаційної робот	15.11.2024	Виконано
2	Ознайомлення з вимогами до структури та оформлення роботи	01.12.2024	Виконано
3	Формування попереднього змісту та плану-графіка виконання	09.12.2024	Виконано
4	Пошук джерел для дослідження	25.12.2024	Виконано
5	Створення вступної частини	03.01.2025	Виконано
6	Написання першого розділу	31.01.2024	Виконано
7	Написання другого розділу	01.03.2025	Виконано
8	Написання третього розділу	30.03.2025	Виконано
9	Дооформлення роботи	14.04.2025	Виконано
10	Оформлення додаткових матеріалів для захисту	10.05.2025	Виконано
11	Передзахист матеріалів дипломної роботи на засіданні кафедри	19.05.2025	Виконано

Здобувач_____

Керівник роботи _____

РЕФЕРАТ БАКАЛАВРСЬКОЇ РОБОТИ

Кваліфікаційна робота: 70 с., 5 рис., 5 табл., 35 літературних джерел.

Актуальність теми роботи: необхідності розробки системи для автоматизації внутрішніх процесів організації та управління з метою підвищення якості обслуговування у закладах громадського харчування.

Об’єкт дослідження: системи організації персоналу та управління в закладах громадського харчування.

Предмет дослідження: архітектурні та функціональні рішення щодо створення програмного забезпечення для автоматизації та управління внутрішніх процесів ресторану.

Мета роботи: розробити інформаційну систему для автоматизації внутрішніх процесів управління в ресторані, зосередившись на покращенні взаємодії персоналу та підвищенні ефективності обслуговування.

Висновки: було розглянуто особливості внутрішньої організації та управління у закладах громадського харчування, проаналізовано наявні інформаційні системи, що використовуються в цій сфері, виділено їх сильні й слабкі сторони. На основі отриманих результатів, сформульовано вимоги до майбутнього рішення, а також спроєктовано архітектуру системи. Реалізовано прототип веб-застосунку, який дозволяє автоматизувати ключові процеси взаємодії між працівниками, обробки замовлень та внутрішнього контролю. Для побудови клієнтської частини було використано Angular та Material UI, серверна частина створена з використанням NestJS, а збереження даних використовується MongoDB.

Практичне значення: запропонована система має практичне застосування в ресторанах для підвищення ефективності обслуговування через автоматизацію внутрішніх процесів.

Ключові слова: ресторанний бізнес, комунікація між працівниками, автоматизація взаємодії персоналу, інформаційна система, система управління замовленнями, клієнт-серверна архітектура, Angular, NestJs, MongoDB

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

CRUD (Create, Read, Update, Delete) - основні операції зі збереження та обробки даних у базі.

UI (User Interface) - інтерфейс користувач.

RxJS (Reactive Extensions for JavaScript) - бібліотека для реактивного програмування в Angular.

WebSocket - протокол для двосторонньої комунікації між клієнтом і сервером у режимі реального часу.

EventBus - архітектурний шаблон для передачі подій між модулями.

Стоп-лист - перелік страв або позицій, тимчасово недоступних для замовлення.

Позиція - одиниця меню (страва або напій), яку можна додати до замовлення.

Оркестратор - сервіс, який координує взаємодію між окремими модулями, сервісами й подіями.

Репозиторій - шар доступу до бази даних, який інкапсулює операції читання/запису.

Фронтенд - клієнтська частина застосунку, що відповідає за взаємодію з користувачем.

Бекенд - серверна частина застосунку, що відповідає за логіку, зберігання даних і взаємодію з клієнтом.

ЗМІСТ

ВСТУП....8
РОЗДІЛ 1. СИСТЕМНІ НЕДОЛІКИ ВНУТРІШНЬОЇ ВЗАЄМОДІЇ ПЕРСОНАЛУ ТА ОГЛЯД ЗАСОБІВ ЇХ АВТОМАТИЗОВАНОГО УСУНЕННЯ....10
1.1 Аналіз проблем традиційної моделі обслуговування в ресторані.... 10
1.2 Інструменти автоматизації обслуговування: огляд сучасних підходів.... 16
1.3 Висновок до розділу....20
РОЗДІЛ 2. ОПИС ВИМОГ ТА АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ....21
2.1 Опис ключових об’єктів інформаційної системи та їх функціональності.... 21
2.2 Вибір технологічного стеку та загальної архітектури....24
2.2 Вибір технологій для серверної частини застосунку....25
2.3 Вибір технологій для клієнтської частини застосунку.... 26
2.4 Висновок до розділу....29
РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ.... 31
3.1 Розробка моделей бази даних.... 31
3.2 Реалізація взаємодії між базою даних та сервером....32
3.3 Реалізація оркестраторів....33
3.4 Реалізація взаємодії в режимі реального часу.... 34
3.5 Використання EventBus у застосунку. Зв’язування ізольованих модулів....35
3.6 Rest API....36
3.7 Реалізація логіки по збереженню зображень для страв.... 37
3.8 Архітектурна організація клієнтської частини веб застосунку.... 39
3.9 Реєстрація навігації на основі ролей користувачів....40
3.10 Розробка клієнтських сервісів і управління даними....41
3.11 Логіка компонентів і принципи повторного використання коду....42
3.12 Реактивна модель взаємодії з даними: використання RxJS в Angular.... 43
3.13 Форматування даних у шаблонах за допомогою пайпів.... 45
3.14 Реалізація адаптивної верстки....45
3.15 Висновок до розділу....46
ВИСНОВОК.... 48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.... 49
Додаток А.... 52
Додаток Б.... 56
Додаток В.... 57

ВСТУП

У ресторанному бізнесі якість обслуговування напряму впливає на успіх закладу. Вчасно подані страви, злагоджена робота персоналу, точність у виконанні замовлень - усе це формує загальне враження клієнта. Проте на практиці багато ресторанів стикаються з типовими для сфери труднощами: помилки у комунікації між залом і кухнею, затримки, неузгодженість дій.

На ринку існує чимало інформаційних систем для автоматизації внутрішніх процесів. Переважно системи обмежуються базовими можливостями: частину важливих функцій узагалі не передбачено, а інші доступні лише за додаткову плату. Наприклад, не всі рішення дозволяють гнучко керувати статусами замовлень, працювати зі «стоп-листами», чи відправляти повідомлення про оновлення готовності страв у режимі реального часу. У результаті - офіціанти та кухарі змушені компенсувати недоліки систем вручну, що знижує ефективність та швидкість обслуговування.

Мета роботи: розробити інформаційну систему для автоматизації внутрішніх процесів управління в ресторані, зосередившись на покращенні взаємодії персоналу та підвищенні ефективності обслуговування.

Для досягнення мети були сформульовані такі **завдання**:

- виявити основні проблеми класичної моделі обслуговування;
- проаналізувати наявні рішення, їх переваги та недоліки;
- сформувати вимоги до системи на основі визначених проблем;
- адаптувати найкращі практики з існуючих рішень;
- реалізувати програмне забезпечення з урахуванням масштабованості та гнучкості налаштувань.

Об’єкт дослідження: системи організації персоналу та управління в закладах громадського харчування.

Предмет дослідження: архітектурні та функціональні рішення щодо створення програмного забезпечення для автоматизації та управління внутрішніх процесів ресторану.

Методи дослідження: у роботі застосовано аналіз документації, порівняння функціоналу існуючих рішень, вивчення практичного досвіду з відкритих джерел, загальноприйняті підходи до розробки інформаційних систем.

Практичне значення: запропонована система має практичне застосування в ресторанах для підвищення ефективності обслуговування через автоматизацію внутрішніх процесів.

Галузь застосування: заклади громадського харчування , що прагнуть модернізувати внутрішню взаємодію персоналу без додаткових витрат на комерційні модулі, з найнижчими витратами.

РОЗДІЛ 1. СИСТЕМНІ НЕДОЛІКИ ВНУТРІШНЬОЇ ВЗАЄМОДІЇ ПЕРСОНАЛУ ТА ОГЛЯД ЗАСОБІВ ЇХ АВТОМАТИЗОВАНОГО УСУНЕННЯ

1.1 Аналіз проблем традиційної моделі обслуговування в ресторані

Однією з найпоширеніших проблем у класичній моделі обслуговування у закладах громадського харчування є неефективна комунікація між основними підрозділами закладу - залом, кухнею та менеджментом. Гарно організована взаємодія між офіціантами, кухарями та адміністраторами значно підвищує якість сервісу. Від того, наскільки швидко й точно передається інформація між працівниками, пропорційно залежить рівень ефективності, обслуговування та загальне враження гостей. Якщо взаєморозуміння між командою порушене, або комунікація відбувається з певними труднощами, це стає причиною неналежного обслуговування: страви подаються із запізненням, замовлення можуть бути неповними чи переплутаними, а інформація про наявність інгредієнтів - неузгодженою. Такі помилки не лише псують загальне враження відвідувачів, а й негативно впливають на внутрішню атмосферу в колективі. Негативний вплив на робочий клімат підтверджується дослідженням компанії Lightspeed: 62 % працівників закладів громадського харчування у США та Канаді регулярно стикаються з труднощами у взаємодії між залом і кухнею. Понад 72 % опитаних назвали це основною причиною внутрішніх конфліктів і напруженої атмосфери в колективі [1].

Проблема комунікації не зводиться до одного недопрацьованого ланцюжка - вона має ряд нюансів, і саме в дрібницях найчастіше ховаються найбільші проблеми ефективності. Якщо глянути на класичну структуру процесу обслуговування, здається, що все логічно: офіціант приймає замовлення, передає його кухні, кухар готує, страва повертається в зал. Усе ніби працює, але в реальності цей ланцюжок має декілька неочевидних проблем. Після того, як замовлення передано, інформація про його подальшу долю часто зависає в

повітрі. Зазвичай, у ресторанах немає механізму, який би чітко повідомляв офіціанту, яка саме страва готова. Через відсутність чіткого сигналу, доводиться йти на кухню і перевіряти готовність власноруч. І робити це знову з певною періодичністю, бо в іншому випадку клієнт може чекати, а страва охолоне за певний час. Така модель породжує нескінченну кількість марних кроків, більшість з яких - просто втрачений час і витрачена енергія. У дослідженні Quantification of Ergonomic Exposures for Restaurant Servers (2016) [2] зафіксовано, що середній офіціант проходить понад 600 кроків щогодини, і проводить на ногах понад 75% усієї зміни. Сидять вони лише близько 25% часу, і у більшості випадків - це не завжди повноцінний відпочинок. І хоча така активність на роботі є природною, проблема в тому, що частина цього руху не має жодної практичної користі. Виходить, що офіціант витрачає сили не на обслуговування клієнта, а на перекриття інформаційної «дірки» між залом і кухнею. Врешті-решт, фізична втома накопичується, а ближче до завершення зміни падає уважність, а з цим зростає ризик помилки та потенційни прогалини в обслуговуванні. У довгій перспективі такої активності виникає ризик хронічних болей у спині, ногах, венозної недостатності, як це також зазначено в згаданому дослідженні. І все це - через те, що система не передбачає елементарного способу дати людині знати, що робити. Можна було б сказати, що проблема втоми - це «частина професії», але ні. Частина професії - працювати з людьми, а не бігати кухнею, намагаючись вгадати момент подачі страви. Тому відсутність зворотного оповіщення - фактор, який системно знижує ефективність персоналу і підриває якість сервісу.

Розглянемо інший, менш очевидний, але набагато ширший сегмент внутрішньої комунікації - обмін інформацією про поточну наявність страв у меню. Проблема тут виникає не через технічні нюанси, або складність організації, а через нестачу певних інгредієнтів, і кухар фізично не має з чого приготувати страву. Формально така позиція вноситься до так званого «стоп-листа», але те, як саме ця інформація циркулює всередині закладу - окрема історія. Працюючи офіціантом у середньому за розміром ресторані, де на зміні було два-три офіціанта, я неодноразово стикався з ситуацією, коли деякі страви виявлялися

недоступними. Найчастіше це траплялося наприкінці тижня, коли складські запаси вже вичерпувалися, а постачання ще не надійшло. В ідеалі про такі зміни мали б повідомляти до початку роботи - щоб кожен офіціант чітко знав, чого немає. На практиці ж усе виглядало трохи інакше.

У кращому разі, ми дізнавалися про оновлення «стоп-листа» на початку зміни, якщо адміністратор встиг усе зібрати і донести дану інформацію. У гіршому - вже в момент роботи, часто у доволі незручний момент: коли замовлення зі стравою, якої немає, вже було прийняте й передане на кухню. Іноді адміністратору доводилося втручатися, вручну редагувати замовлення, скасовувати позицію, але найнеприємніше починалося тоді, коли офіціанту доводилося йти до клієнта і пояснювати, що замовлену ним страву приготувати неможливо.

Бувало й інакше: інформацію про «стоп-лист» ніхто не забував передати, але під час вечірніх навантажень, самі офіціанти просто про неї забували. Людський фактор, звісно, але результат один і той самий: зайві пояснення, вибачення, зіпсоване перше враження, яке у сфері обслуговування грає чи не головну роль.

Такі епізоди спричиняли напругу не лише у спілкуванні з відвідувачами, а й всередині команди. Після кількох подібних ситуацій стосунки між офіціантами та адміністраторами ставали дедалі більш дратівливими. Хтось когось звинувачував у неуважності, хтось - у недомовленості й ця мікронапруга осідала в робочій атмосфері. На тлі всіх інших логістичних процесів у ресторані, ситуація зі «стоп-листом» може здаватися дрібницею, але саме такі дрібниці накопичуються й створюють відчуття хаосу: для персоналу - нервове перевантаження, для клієнта - враження неорганізованості.

Окресливши організаційні проблеми діяльності закладу, доцільно зупинитися на труднощах, з якими стикаються офіціанти, кухарі та адміністратори окремо. Серед особливостей, що відрізняють роботу офіціанта, варто відзначити необхідність добре знати меню закладу. Це - окремий аспект, який доповнює раніше згадані вимоги до цієї посади. Професія офіціанта традиційно

характеризується високою плинністю кадрів, що створює додаткові виклики щодо стабільності знань серед працівників. Зазвичай, на початку роботи адміністрація закладу висуває вимогу ретельно вивчити меню, включаючи страви, інгредієнти та особливості приготування. Проте з часом у офіціантів спостерігається природна тенденція до часткового забування раніше вивченого матеріалу, особливо якщо асортимент страв регулярно оновлюється, або доповнюється новими позиціями. У дослідженні Т. Мірабеллі «The Language and Literacy of Food Service Workers» зазначається, що рівень знання меню безпосередньо впливає на ефективність комунікації з клієнтами, а також на здатність офіціанта точно і впевнено рекомендувати страви, відповідати на питання гостей щодо складу чи особливостей приготування страв [3]. Знання про можливі алергени у складі страв, також є важливим аспектом роботи, оскільки своєчасне та достовірне інформування відвідувача з цього питання, впливає на його безпеку здоров'я й задоволення від перебування у закладі. Як свідчить дослідження у сфері ресторанного обслуговування, офіціанти з часом можуть втрачати точність у запам’ятовуванні складу страв, що у підсумку, може мати негативний вплив на рівень сервісу та враження відвідувачів.

Однак важливо враховувати, що якість обслуговування залежить не лише від ефективної роботи офіціантів, але й від організації процесів на кухні. Проблеми, що виникають у роботі кухарів, заслуговують на окрему увагу, адже саме від їхніх дій залежить, якою буде кінцева якість страв і чи відповідатимуть вони встановленим стандартам. Серед можливих критичних моментів - недостатнє знання рецептурного складу та порушення у дотриманні вагових норм на етапі приготування.

Як і офіціанти, кухарі повинні добре знати меню закладу. Водночас їхня відповідальність не обмежується лише загальним розумінням структури страви - вони мають досконало володіти інформацією про точну грамовку кожного інгредієнта, відповідно до технологічної карти.

У випадку, якщо страва, подана гостю, не відповідає зазначеним у меню, або технічному опису показникам ваги, заклад ризикує зіткнутися не лише з

негативними відгуками чи втратою клієнта, але й із юридичними наслідками. Відповідно до статті 17 Закону України «Про захист прав споживачів», споживач має право на достовірну, своєчасну та повну інформацію про продукцію, яку він споживає, зокрема її кількість й вагу. У законі прямо зазначено, що на вимогу споживача продавець (виконавець) зобов'язаний надати йому можливість перевірити вагу товару за допомогою контрольно-вимірювальних приладів [4].

Недотримання цих вимог може стати підставою для звернення клієнта до контролюючих органів, зокрема до Державної служби з питань безпечності харчових продуктів та захисту споживачів. За підсумками перевірки до закладу можуть бути застосовані штрафи, або тимчасове обмеження у здійсненні діяльності. Якщо ж порушення мають систематичний характер, це може стати підставою для адміністративної, а іноді й судової відповідальності.

Ще однією проблемою, яка на перший погляд може здатися незначною, але на практиці має суттєвий вплив на швидкість подачі є питання пріоритизації готування страв. Ця проблема особливо гостро проявляється у періоди пікового навантаження, коли на кухню одночасно надходить велика кількість замовлень із залу, доставки або самовивозу. У таких умовах кухарі змушені самостійно орієнтуватися у списку поточних замовлень, приймати рішення щодо черговості приготування та оцінювати, які страви варто готувати першими з урахуванням складності технологічного процесу, обмежень по часу очікування та індивідуальних вимог клієнтів. Цей процес пріоритизації, хоча й виглядає як частина щоденної рутини, фактично забирає певний час, який міг би бути витрачений безпосередньо на приготування. Крім того, коли розподіл замовлень між кухарями здійснюється вручну, або за домовленістю між членами команди, виникає ризик неузгоджених дій, що може призвести до повторного приготування однієї й тієї ж страви або, навпаки, випадкового пропуску окремих позицій. Така ситуація особливо ускладнюється у великих закладах з розгалуженим меню, де за різні етапи приготування відповідають окремі кухарі - наприклад, за холодні страви, гарячі, гарніри, вироби з грилю тощо. У подібних умовах чітке узгодження послідовності дій та розподілу обов’язків набуває критичного значення.

Додатковою, хоча й менш помітною проблемою у роботі кухаря є дотримання нормативного часу приготування страв. У більшості закладів громадського харчування встановлені чіткі часові рамки для приготування кожної позиції з меню. Проте в умовах класичної моделі обслуговування ця вимога часто залишається поза увагою, або фактично ігнорується.

Тож крім знань складу страви та точну грамовку кожного інгредієнта, кухарі мають ще й орієнтуватися в технологічному часі її приготування. Це створює додаткове когнітивне навантаження - обсяг інформації, яку потрібно тримати в пам’яті, збільшується, і не вся ця інформація сприймається як першочергова у щоденній роботі. Внаслідок цього можливі ситуації, коли страви готуються довше, ніж передбачено, як результат - призводить до затримок у подачі замовлень.

Як бачимо, робота кухарів, так само як і офіціантів, супроводжується низкою викликів, що мають безпосередній вплив на загальну якість обслуговування. Водночас важливо враховувати, що жоден елемент ресторану не функціонує ізольовано. Весь процес взаємодії персоналу потребує координації, і в цій системі саме адміністратор виконує об’єднувальну роль.

У його зону відповідальності входить як загальне управління, так і контроль за щоденною роботою персоналу. Він відповідає за комунікацію між залом і кухнею, контроль за дотриманням стандартів обслуговування, формування змін, моніторинг роботи персоналу, а також за вирішення конфліктних ситуацій, що можуть виникати у ході обслуговування клієнтів.

У низці закладів поширена практика, за якої адміністратор у періоди інтенсивного навантаження бере участь у процесі обслуговування гостей, підсилюючи команду офіціантів. Проблема полягає в тому, що в межах традиційної моделі адміністратор може оцінити ситуацію лише безпосередньо, перебуваючи в залі, або на кухні. Його спроможність своєчасно визначити потребу у втручанні залежить від візуального спостереження, аналізу кількості зайнятих столів, темпу обробки замовлень та загального стану персоналу. Усе це потребує

часу і досвіду, а відсутність миттєвого зворотного зв’язку ускладнює прийняття рішень в умовах підвищеного навантаження.

Окремою другорядною, проблемою в межах класичного підходу до організації роботи є використання паперового меню. У випадку внесення змін - таких як оновлення цін, додавання або вилучення страв - меню потребує повторного друку. Це спричиняє додаткові фінансові витрати на виготовлення нових примірників, а також витрати часу на їх заміну в залі. Частота оновлень прямо впливає на обсяг витрат, що в умовах обмеженого бюджету може стати додатковим фінансовим навантаженням для закладу.

1.2 Інструменти автоматизації обслуговування: огляд сучасних підходів

Автоматизація процесів стала невіддільною частиною сучасного ефективного управління. Інформаційні системи не лише прискорюють обслуговування, а й допомагають позбутися багатьох типових труднощів, характерних для традиційної моделі роботи.

У цьому підрозділі буде розглянуто приклади сучасних автоматизованих рішень, які впроваджуються для покращення роботи офіціантів, кухарів та адміністраторів. Також буде проаналізовано, якими саме технічними засобами та функціональними механізмами досягається покращення координації, зменшення витрат часу й ресурсів, а також підвищення загальної якості обслуговування гостей.

Аналіз варто розпочати з програмних рішень, що забезпечують координацію між залом, кухнею та адміністративним персоналом. Для прикладу були обрані системи **Poster POS**, **R-Keeper** та **Alto’s POS**, оскільки вони досить поширені у ресторанному бізнесі й дозволяють оцінити підходи до вирішення проблем класичного обслуговування.

У Poster POS і R-Keeper використовують модульну структуру: систему поділено на кілька окремих частин: касовий термінал, панель адміністратора,

мобільний застосунок для офіціанта та екран на кухні. Кожен із цих елементів виконує свою конкретну роль у загальному процесі. Щоб мати доступ до повного набору функцій, за даними офіційних джерел [5, 6], бізнесу потрібно підключити підписку або окремо придбати потрібні модулі. Сукупність цих модулів автоматизує процес: офіціант може одразу оформити замовлення біля клієнта, а система сама розподіляє страви між кухонними зонами. Щойно страва готова, кухар змінює її статус, і ця інформація автоматично оновлюється на пристрої офіціанта. Завдяки цьому немає потреби постійно ходити на кухню, як це було раніше.

Окремої уваги заслуговує також система Alto’s POS [7]. На відміну від багатьох інших, вона одразу включає основні модулі до стандартного функціоналу - без потреби щось додатково оплачувати. Комунікація між офіціантом і кухнею тут побудована через вбудований дисплей, що спрощує роботу з замовленнями. Водночас є й свої нюанси: статус готовності страв не оновлюється автоматично. Щоб побачити зміни, офіціанту потрібно вручну перезавантажити інтерфейс - це уповільнює обслуговування, особливо в години пік.

Як бачимо, завдяки використанню сучасних інформаційних технологій та можливості обміну даними в режимі реального часу, розглянуті системи дозволяють покращити внутрішню взаємодію персоналу. Передача замовлень і статусів приготування страв відбувається без участі проміжних фізичних дій - таких як усні повідомлення, паперові нотатки чи особисте з’ясування інформації між офіціантом і кухарем. Таким чином було усунуто зайву активність офіціантів, які раніше змушені були регулярно повертатися на кухню, щоб дізнатись готовність страви. Тепер, завдяки автоматичній зміні статусу та його відображенню в інтерфейсі, офіціант може йти на станцію лише тоді, коли страва дійсно готова до подачі, завдяки чому замовлення обробляються швидше, а персонал може повністю зосередитися на роботі з відвідувачами.

Ще однією проблемою є відсутність злагодженої інформативності щодо стоп-листа, що нерідко спричиняє операційні збої в роботі офіціанта. Розглянуті

системи вирішують це питання по-різному. У Poster POS адміністрація може оперативно приховувати певні страви з інтерфейсу офіціанта через адміністративну панель. Щойно позиція потрапляє до стоп-листа, вона автоматично зникає зі списку доступних до замовлення страв, що зменшує ризик помилок [8]. Схожий підхід реалізовано в R-Keeper: тимчасово недоступні позиції просто не відображаються у вікні формування замовлення, тому офіціант фізично не може їх додати до чеку.

У Alto’s POS, на відміну від вищезгаданих систем, повноцінного стоп-листа не передбачено. Страва, яка не може бути приготована через відсутність інгредієнтів, усе ще може бути додана до замовлення, оскільки інтерфейс ніяк не вирішує цю проблему.

Наступним кроком є аналіз того, як сучасні інформаційні системи допомагають персоналу справлятися з повсякденними труднощами на робочому місці. Зокрема, одна з проблем, пов’язаних з роботою офіціанта - знання складу страв була вирішена завдяки впровадженню відповідного інтерфейсу. У мобільному додатку **Mobile Waiter**, що входить до складу системи Poster POS, офіціант має доступ до розширеного опису кожної страви, включно зі списком основних інгредієнтів. Завдяки цьому офіціант може швидко орієнтуватися в меню та надати клієнту повну інформацію про склад страви, навіть якщо не пам’ятає окремі інгредієнти.

Розглядаючи функціональні рішення для кухонних станцій, варто звернути окрему увагу на інтерфейси, якими користуються кухарі. У більшості сучасних рішень кухар має змогу швидко переглянути склад та грамовку кожної страви без необхідності звертатися до повної технологічної карти. Реалізація цього функціоналу суттєво пришвидшує процес приготування страв, особливо в умовах інтенсивного навантаження.

Щодо пріоритезації приготування, то кожна система застосовує власні методи управління чергою страв та контролю за часом. У Poster POS, наприклад,

застосовано ручний запуск таймера: кухар сам ініціює початок приготування, обравши потрібну позицію в інтерфейсі. Система починає відлік часу, що дозволяє орієнтуватися на фактичну тривалість процесу без фіксованих обмежень.

Натомість у **Vouch POS** акцент зроблено на зворотному відліку: кухар бачить, скільки часу залишилося до завершення приготування згідно з установленим стандартом. Окрім цього, інтерфейс змінює кольори залежно від статусу страв: для тих, що ще готуються, використовується одна кольорова схема, прострочені мають інше оформлення, а завершені - позначаються окремим стилем [9].

Усі ці механізми спрямовані на покращення взаємодії між кухнею та залом, але для стабільної роботи всієї системи важливо також, щоб адміністратор міг швидко реагувати на зміни та координувати персонал.

Питання контролю з боку адміністратора, також вирішується за допомогою функціоналу самих систем. Як правило, адміністратор має змогу не лише стежити за роботою персоналу в реальному часі, а й швидко реагувати на зміни. Наприклад, він може редагувати меню, оновлювати стоп-листи, слідкувати за часом приготування страв і координувати взаємодію між кухнею та залом.

Крім робочих труднощів персоналу, вирішення отримала й проблема оновлення меню. У більшості сучасних систем адміністратор має змогу редагувати склад меню безпосередньо в панелі управління, після чого зміни одразу відображаються у клієнтському інтерфейсі, зокрема через QR-код. Це дає змогу відмовитись від паперових версій меню, зменшити супровідні витрати та забезпечити відвідувачам доступ до актуальної інформації незалежно від частоти її оновлення. Такий підхід стає стандартом у більшості цифрових рішень, орієнтованих на заклади швидкого та комбінованого обслуговування.

1.3 Висновок до розділу

Аналіз традиційної моделі обслуговування показав, що в роботі персоналу часто виникають типові труднощі. Найбільше це відчувається у взаємодії між залом, кухнею та адміністрацією. Сучасні автоматизовані системи прагнуть усунути ці недоліки шляхом впровадження окремих функціональних модулів.

Системи на зразок Poster POS або R-Keeper працюють досить ефективно, але їхній повний функціонал доступний тільки після купівлі додаткових модулів. Для багатьох закладів це може стати бар’єром через обмежений бюджет. З іншого боку, є рішення, як-от Alto’s POS, де багато потрібних функцій уже доступні «з коробки». Проте можливості таких систем обмежені, і вони не завжди покривають усе, що потрібно для великого ресторану.

Саме Тому виникла ідея створити власну систему без прив’язки до окремих платних блоків. Вона має поєднувати сильні сторони вже існуючих рішень, але залишатися гнучкою, щоб її можна було легко доопрацьовувати й масштабувати без зовнішніх залежностей.

РОЗДІЛ 2. ОПИС ВИМОГ ТА АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У цьому розділі буде сформовано вимоги до майбутньої інформаційної системи на основі проведеного в попередньому розділі аналізу визначених проблем. Також враховуючи досвід та функціонал вже існуючих автоматизованих рішень, буде визначено перелік функціональних і нефункціональних характеристик. Сформульовані вимоги ляжуть в основу створення архітектури інформаційної системи, адаптованої до специфіки роботи закладів громадського харчування.

2.1 Опис ключових об’єктів інформаційної системи та їх функціональності

Першим етапом розробки програмного забезпечення є виявлення ключових сутностей, навколо яких формуватиметься подальша структура проєкту. У контексті автоматизації процесів ресторанного обслуговування такими сутностями є працівники, замовлення та позиції меню. Саме їхня взаємодія формує ядро бізнес-логіки майбутнього застосунку.

У межах формування вимог, доцільно почати з найпростіших елементів - страв. Ця сутність є основою для формування замовлень, оскільки саме страви та напої складають змістовну частину обслуговування клієнтів. Система повинна забезпечувати можливість додавання, видалення та редагування позицій меню. Кожна позиція меню повинна містити набір характеристик, необхідних для відображення та обробки в системі. Серед них - загальні відомості про страву (назва, зображення, склад), параметри приготування (відповідальна зона, орієнтовна тривалість), економічні дані (ціна), а також ознака наявності у стоп-листі, що впливає на можливість додавання позиції до замовлення. Інтерфейс додатку має дозволяти користувачам вводити, переглядати та зручно знаходити позиції меню за допомогою фільтрів і пошуку.

Наступною, більш структурно складною сутністю виступають працівники, які в системі є єдиною категорією користувачів. Для кожного з них зберігається основна інформація: ім’я, прізвище, роль (офіціант, кухар або адміністратор), а

також персональний код доступу для авторизації. У випадку кухарів додатково потрібно вказувати зону приготування, до якої вони прикріплені. Система повинна підтримувати чітке розмежування доступу до функцій відповідно до ролі користувача.

Після запуску додатку користувач повинен потрапити на екран авторизації, де має ввести свій персональний код доступу. Система повинна автоматично визначити його роль та здійснити переадресацію на інтерфейс, що відповідає функціональним обов’язкам, закріпленим за цією роллю.

Адміністратор, зокрема, матиме найбільше повноважень: він працюватиме з ключовими об'єктами: персонал, меню та замовлення. Інтерфейс надаватиме йому зручні інструменти для перегляду, зміни, додавання або видалення даних. Усі дії адміністратор виконуватиме через відповідні пункти меню. Також в інтерфейсі повинне передбачатися табличне відображення даних, для зручного перегляду списків працівників й позицій, з відповідними формами фільтрації. Зокрема, працівників можна буде фільтрувати за ролями, а позиції - за назвою страви та зоною приготування. Крім цього, адміністратор матиме доступ до перегляду поточних замовлень разом із їхніми статусами, щоб мати можливість оцінити загальну ситуацію в закладі та розуміти рівень завантаженості персоналу в реальному часі.

Офіціант працюватиме з інтерфейсом, призначеним для обслуговування клієнтів. На головному екрані повинна відображатися схема залу зі столиками - кожен столик має бути інтерактивним. Натискаючи на нього, повинне відкриватись модальне вікно з трьома вкладками: «Позиції», «Підтвердження» та «Рахунок». У вкладці **«Позиції»** офіціант зможе обирати страви з меню. Для зручності кожна позиція відображатиметься у вигляді картки з фото, назвою та описом складу. Навігацію спростить пошук і фільтрація за назвою. Якщо страва перебуває у стоп-листі, система автоматично заблокує можливість додавання її до замовлення. Перейшовши до вкладки **«Підтвердження»**, працівник побачить перелік вибраних позицій. Тут він матиме змогу перевірити правильність замовлення та за потреби його відредагувати перед передачею на кухню. У розділі

«Рахунок» система згенерує фінальний чек. Після підтвердження й закриття замовлення, стіл буде звільнено для нових гостей, і офіціант зможе сформувати наступне замовлення.

Після підтвердження замовлення система автоматично розподіляє страви між відповідними зонами приготування. Коли страва буде готова, офіціант отримує сповіщення з її назвою та номером столика. Після подачі офіціант повинен підтвердити отримання, натиснувши відповідну кнопку на інтерфейсі.

Кухар після авторизації отримуватиме доступ до робочої панелі, налаштованої під його функціонал. Нові замовлення надходитимуть у вигляді карток, що автоматично розміщуватимуться в межах призначеної зони приготування, відповідно до налаштувань позиції меню. Картка міститиме назву страви, номер столика та таймер, який активуватиметься в момент надходження замовлення.

Механізм таймера базуватиметься на підході, використаному в системі VouchPOS Kitchen Display System [9], де відображається зворотний відлік приготування. У даному випадку функціональність буде доповнена вдосконаленою логікою кольорового кодування для візуального визначення пріоритетів. Кожна картка проходитиме через три рівні стану залежно від частки часу, що залишився до завершення приготування. Якщо залишатиметься понад 66% нормативного часу, картка підсвічуватиметься зеленим кольором. У проміжку між 66% і 33% колір змінюватиметься на жовтий. При зменшенні часу нижче 33%, або у випадку перевищення ліміту картка набуватиме червоного кольору. Такий підхід дозволятиме швидко зорієнтуватися у послідовності виконання замовлень без потреби у додаткових розрахунках.

Також система повинна забезпечувати можливість перегляду складу страви для кухарів безпосередньо в момент обробки замовлень. При натисканні на картку страви має відкриватися модальне вікно з зображенням та переліком інгредієнтів відповідно до технологічної карти. Після завершення приготування кухар повинен мати змогу відзначити страву як готову через натискання відповідного елементу інтерфейсу

Останню сутносутнестою системи є замовлення. Воно охоплює взаємодію між офіціантом, кухонним персоналом, а тому його обробка має бути максимально гнучкою та зручною. Система повинна надавати змогу створювати нові замовлення, вносити зміни в процесі обслуговування (наприклад, при зміні побажань клієнта) і завершувати замовлення після виконання всіх необхідних дій.

Замовлення має включати перелік страв, кожна з яких супроводжується власним статусом, що вказує на її поточний етап у процесі обслуговування. Залежно від дій персоналу, статус може набувати значення «готується», «готово» або «видано». Крім цього, саме замовлення також повинне мати загальний статус, що відображає його стан у межах системи: активне - якщо замовлення ще не завершено, та закрите - після повного виконання.

Система повинна забезпечувати збереження всіх позицій, що були додані до замовлення, а також передбачати можливість додавання нових страв до переліку до моменту закриття замовлення.

2.2 Вибір технологічного стеку та загальної архітектури

Для реалізації функціональності інформаційної системи було обрано клієнт-серверну архітектуру. Такий підхід є загальноприйнятим у проєктуванні сучасних застосунків, особливо тих, що передбачають динамічну обробку даних і потребу в централізованому управлінні процесами.

Клієнт-серверний підхід передбачає логічне розділення системи на дві частини. З одного боку - інтерфейс, з яким взаємодіє користувач (front-end), з іншого - серверна частина (back-end), що обробляє запити, зберігає дані та реалізує прикладну логіку. Така структура забезпечує незалежний розвиток окремих компонентів системи, спрощує її підтримку й масштабування. Окрім цього, вона відкриває широкі можливості для інтеграції з різними типами клієнтів: від мобільних застосунків і настільних програм до браузерних інтерфейсів, що робить систему гнучкою й універсальною в користуванні.

На стартовому етапі розробки систему планується реалізувати у вигляді веб-додатку, що працює безпосередньо через браузер. Це рішення дозволить

обійтися без встановлення додаткових програм і робить продукт доступним для користувачів незалежно від типу пристрою - будь то комп’ютер, планшет чи смартфон.

2.2 Вибір технологій для серверної частини застосунку

Після аналізу доступних серверних фреймворків було прийнято рішення реалізувати серверну частину системи на базі NestJS - фреймворку для Node.js, що використовує TypeScript як основну мову програмування. У документації зазначено, що NestJS створений для розробки масштабованих серверних застосунків і поєднує в собі модульність, інжекцію залежностей та підтримку різних транспортів, включаючи WebSocket [10].

NestJS виявився найбільш зручним і доречним у контексті мого проєкту: він невеликий за обсягом, а строки реалізації - обмежені, тому важливо було обрати технологію, яка забезпечує швидкий старт без потреби в складній конфігурації. Порівняно зі Spring Boot, що працює на Java, NestJS має кілька ключових переваг. Зокрема, розробка з NestJS проходить швидше завдяки нижчому порогу входження, а його логіка ближча до фронтенд-підходів, з якими я вже мав практичний досвід. Для наочності нижче наведено узагальнену таблицю з основними відмінностями між NestJS і Spring Boot (див. табл. 2.1).

Таблиця 2.1

Порівняльна характеристика фреймворків NestJS і Spring Boot

Критерій	**NestJS**	**Spring Boot**
Мова програмування	Type Script	Java
Швидкість розробки	Висока	Середня
Порог входження для розробника	Низький	Високий

Продовження таблиці 2.1

Масштабованість системи	Висока	Висока
Гнучкість структури застосунку	Висока	Висока
Інтегровані Інструменти	Так	Ні

Для зберігання даних у цьому проєкті було обрано MongoDB - документоорієнтовану NoSQL-базу, яка органічно поєднується з архітектурою NestJS. Вона працює з JSON-подібними документами, підтримує гнучку схему, а також забезпечує високу швидкість обробки запитів [11]. Такий підхід особливо зручний у випадках, коли окремі сутності мають унікальні поля, які не актуальні для інших. Наприклад, для кухарів доречно зазначати зону приготування, тоді як для офіціантів ця інформація не є релевантною. Крім того, використання офіційної бібліотеки Mongoose спрощує інтеграцію з NestJS, забезпечуючи швидку реалізацію типових CRUD-операцій і даючи змогу зосередитися на розробці прикладної логіки системи.

2.3 Вибір технологій для клієнтської частини застосунку

Клієнтська частина системи повинна працювати в браузері - як уже зазначалося, такий підхід усуває потребу в установці додаткового програмного забезпечення та гарантує кросплатформену доступність.

На етапі вибору фронтенд-технології було проаналізовано три найбільш популярні рішення - React, Vue та Angular, які активно застосовуються у сучасній веб розробці. Для обґрунтування рішення було виконано порівняльний аналіз за низкою технічних параметрів (див. табл. 2.2).

Таблиця 2.2

Порівняльна характеристика фреймворків React, Vue та Angular

Критерій	**React**	**Vue**	**Angular**
Підхід	Функціональний	Декларативний	Об'єктно-орієнтований
Тип	Бібліотека	Фреймворк	Фреймворк
Мова	JavaScript + JSX	JavaScript / TypeScript	TypeScript
Крива навчання	Помірна	Низька	Висока
Підтримка DI	Ні	Частково	Так (вбудований DI-контейнер)
Документація	Добра	Відмінна	Відмінна
Розширюваність	Гнучка	Висока	Висока
Підтримка великих проєктів	Частково	Менш бажана	Найбільш придатний для масштабування
Продуктивність	Висока	Висока	Висока

Згідно з офіційною документацією, Angular розроблено для створення масштабованих односторінкових вебзастосунків (SPA). Він пропонує повний набір вбудованих інструментів: маршрутизацію, двостороннє зв’язування даних, обробку форм, модульну структуру, підтримку HTTP-запитів та реактивне програмування через RxJS [12; 15].

Попри простоту Vue [13] та гнучкість React [14], я свідомо обрав Angular. Такий вибір зумовлений як власним досвідом роботи з ним у процесі навчання, так функціональними вимогами застосунку.

Фреймворк Angular розроблений із чітким акцентом на модульність і масштабованість, що робить його особливо зручним для створення середніх і великих проєктів. Завдяки вбудованій системі інжекції залежностей (DI) розробник отримує простий і гнучкий механізм керування об’єктами, що

полегшує підтримку та тестування застосунку. До того ж, використання TypeScript як основної мови програмування додає суворої типізації, що допомагає виявляти потенційні помилки ще на етапі написання коду [12].

Важливою перевагою є інтеграція Angular з бібліотекою RxJS, яка забезпечує реактивну обробку потоків даних. Завдяки цьому стає можливим зручно реалізовувати складні асинхронні сценарії й будувати динамічні, гнучкі інтерфейси [15].

Окрім технічних аспектів, архітектура Angular повністю відповідає принципам об’єктно-орієнтованого програмування (OOP) [16], а також DRY, KISS та SOLID - фундаментальним підходам до створення гнучкого, розширюваного та підтримуваного коду [17].

Остаточний вибір на користь Angular зумовлений поєднанням низки факторів: практичний досвід роботи з фреймворком, відповідність актуальним архітектурним підходам, а також технічні переваги, які особливо важливі в умовах обмежених ресурсів. У межах бакалаврської кваліфікаційної роботи, саме Angular виявився найбільш збалансованим варіантом - він забезпечує швидку реалізацію необхідної функціональності та не ускладнює подальший супровід системи.

Для пришвидшення процесу розробки клієнтського застосунку, було прийнято рішення використовувати готову бібліотеку UI-компонентів, сумісну з Angular. Такі бібліотеки значно прискорюють створення інтерфейсу, забезпечуючи вже готові елементи керування, які відповідають вимогам адаптивності, доступності та єдиного стилю оформлення.

Серед найбільш поширених рішень було розглянуто Angular Material та PrimeNG. Обидві бібліотеки активно використовуються в Angular-розробці, мають широку екосистему та хорошу документацію. Однак їх функціональні можливості, гнучкість і призначення дещо відрізняються.

PrimeNG вирізняється великою кількістю готових компонентів (понад 80), які охоплюють широкий спектр інтерфейсних елементів: розширені таблиці, вкладки, діалогові вікна, графіки, календарі тощо [19]. Бібліотека підтримує гнучку темізацію, дає можливість змінювати зовнішній вигляд елементів і має преміум-версії з розширеним функціоналом.

У межах дипломного проєкту було обрано Angular Material - офіційну бібліотеку компонентів, розроблену командою Angular [18]. Вона реалізує принципи Material Design від Google і поєднує в собі консистентність, продуману документацію, тісну інтеграцію з Angular CLI та підтримку доступності. Компоненти Angular Material органічно вписуються в архітектуру застосунку, логічно взаємодіють між собою, мають зрозумілий API й легко підключаються до двостороннього зв’язування даних.

Компонентна бібліотека Angular Material стане основою для формування візуального інтерфейсу системи. Вона охоплює всі ключові елементи - форми, таблиці, діалогові вікна, навігацію - і дозволяє зберігати єдиний стиль оформлення, спрощуючи розробку.

2.4 Висновок до розділу

У цьому розділі було визначено функціональні та нефункціональні вимоги до майбутньої інформаційної системи, з урахуванням специфіки ресторанної сфери та особливостей бізнес-процесів. На основі цих вимог описано перелік ключових об’єктів, які становлять ядро логіки застосунку: працівники, позиції меню та замовлення.

Окрім цього, обґрунтовано вибір клієнт-серверної архітектури, яка забезпечує розподіл відповідальності між клієнтською та серверною частинами, сприяє масштабованості, гнучкості й підтримуваності системи у майбутньому. У якості бекенду було обрано NestJS у поєднанні з MongoDB, що дає змогу швидко реалізовувати рішення з динамічною структурою даних. Фронтенд реалізація буде

виконана за допомогою Angular, який у поєднанні з Angular Material забезпечує швидку розробку інтерфейсу з вже реалізованими ui компонентами.

Таким чином, сформовано надійну технічну основу для реалізації всіх поставлених функцій. Обрані рішення не лише відповідають сучасним вимогам до веб застосунків, а й враховують практичні обмеження щодо ресурсів, часу й подальшої підтримки системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

У цьому розділі описано технічні підходи, що застосовувалися в ході розробки інформаційної системи. Розглянуто реалізацію як клієнтської, так і серверної частин застосунку, наведено структуру проєкту, перелік використаних бібліотек, інструментів та архітектурних рішень. Також описано основні патерни, підходи до організації коду й принципи взаємодії між компонентами системи.

3.1 Розробка моделей бази даних

Після визначення загальної архітектури взаємодії застосунку з базою даних було здійснено перехід до моделювання сутностей, які становлять логічну структуру внутрішнього представлення інформації. Як зазначалося у другому розділі, для реалізації взаємодії з базою даних обрано ORM-фреймворк Mongoose, що забезпечує ефективну роботу з MongoDB у середовищі Node.js.

У процесі розробки застосовано підхід *Code First*, при якому структура даних задається безпосередньо у коді через класи та інтерфейси. На їхній основі формуються відповідні документи у колекціях бази даних. Такий підхід дає змогу будувати моделі з підтримкою типізації, валідації та інших обмежень, без потреби попереднього налаштування схем у самій базі. Усі зміни реалізуються на рівні коду, що значно спрощує супровід та оновлення проєкту [20].

У відповідності до функціональних вимог системи, було визначено три ключові групи сутностей: працівники, позиції меню та замовлення.

Модель Dish відповідає за зберігання інформації про кожну страву. Вона містить базові характеристики - назву, орієнтовний час приготування, ціну, тип зони (кухня, бар або суші-збар), а також ім’я файлу із зображенням. Крім основних параметрів, у моделі присутній масив компонентів, що описують склад страви. Для цього використовується допоміжна структура, за допомогою якої можна зберігати дані про інгредієнти без необхідності створення окремих сутностей у базі.

Модель Order описує структуру замовлення та його поведінку в системі. Вона включає номер столика, посилання на працівника (офіціанта), список замовлених позицій, а також позначку, що вказує на активний чи завершений стан. Страви в межах одного замовлення представлені як масив вкладених об’єктів - такий підхід дає змогу фіксувати поточний статус кожної позиції, відстежувати час її подачі й визначати відповідні зони приготування.

Модель User зберігає інформацію про персонал закладу. Вона містить поля для імені, прізвища, ролі в системі (офіціант, кухар або адміністратор), унікального коду авторизації, а також - у випадку з кухарями вказівку на відповідну зону приготування.

Нижче наведено UML-діаграму класів, яка відображає повну структуру взаємозв’язків між сутностями бази даних.

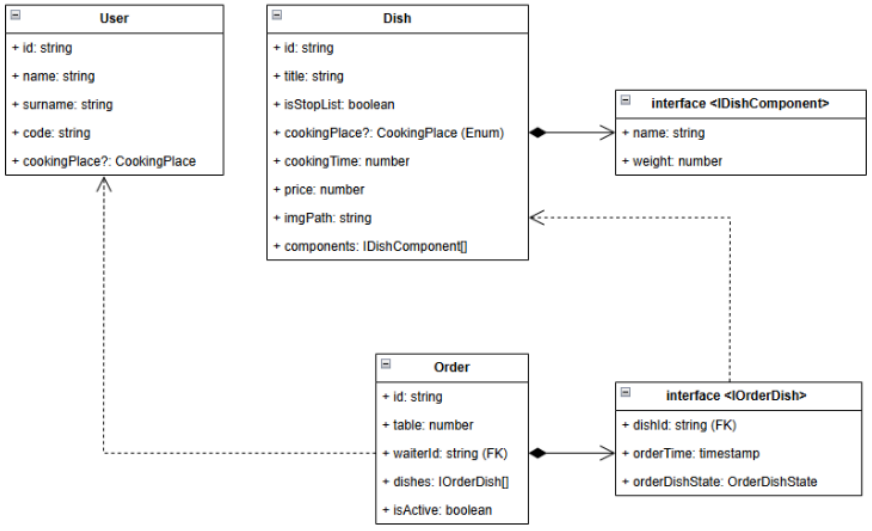

Рисунок 3.1. UML-діаграма взаємозв’язків між сутностями бази даних

3.2 Реалізація взаємодії між базою даних та сервером

Після завершення етапу проєктування моделей сутностей, були створені класи, відповідальні за обробку та зберігання даних. Для цього було застосовано

патерн репозиторію, який передбачає інкапсуляцію логіки доступу до даних та ізоляцію бізнес-логіки від специфіки конкретної бази даних. Репозиторій надає абстрактний інтерфейс для роботи з джерелом даних, що сприяє підтримуваності та тестованості коду. Детальніше про концепцію репозиторію можна ознайомитися у відкритій документації Microsoft [21].

У межах реалізації було створено узагальнений інтерфейс IBaseRepository, який визначає базовий набір CRUD-операцій. На його основі реалізовано три сервіси: DishRepository, OrderRepository та UserRepository. Кожен з них взаємодіє з відповідною моделлю, створеною з використанням бібліотеки Mongoose, та реалізує усі методи, передбачені інтерфейсом.

У системі Mongoose модель (model) - це програмна конструкція, яка поєднує визначену схему даних із набором методів для взаємодії з колекцією в базі MongoDB. Модель забезпечує можливість створення, читання, оновлення та видалення документів, виступаючи повноцінним інтерфейсом між застосунком і базою даних [22].

Завдяки використанню патерну «репозиторій» було виділено окремий рівень абстракції, який відповідає за обробку запитів до бази даних. Це дозволило зберегти логіку доступу до даних в одному місці, зробити код більш структурованим і зручним для подальшої підтримки та розвитку.

3.3 Реалізація оркестраторів

На етапі проєктування архітектури було прийнято рішення додати окремий логічний рівень - оркестратори. Їх використання дало змогу зосередити реалізацію бізнес-логіки в одному місці, чітко розмежувати відповідальність між складовими системи та спростити масштабування. Оркестратори виступають як посередники між сервісами, репозиторіями й іншими модулями, формуючи цілісний шар керування прикладною логікою

У структурі застосунку ключову роль відіграють три окремі оркестратори: DishOrchestrator, UserOrchestrator та OrderOrchestrator. Кожен із них працює у тісному зв’язку з відповідними репозиторіями та бере на себе обробку логіки, що виходить за межі базового зчитування чи збереження даних.

DishOrchestrator відповідає за роботу з інформацією про страви - від отримання повного переліку позицій до додавання, редагування або видалення конкретної страви за ідентифікатором. Крім того, цей сервіс готує допоміжні дані, які необхідні для правильного відображення меню на стороні клієнта.

UserOrchestrator реалізує операції з користувачами системи. При створенні нового облікового запису генерується унікальний код авторизації, що проходить перевірку на неповторюваність. Крім того, у межах оновлення зберігається незмінним ключовий ідентифікатор користувача, а видалення супроводжується перевіркою наявності запису в базі.

OrderOrchestrator виконує дії, пов’язані із замовленнями: створення, оновлення, зміна статусів окремих позицій, додавання нових страв у вже відкриті замовлення. У процесі формування нового замовлення, кожна страва отримує початковий стан, час початку приготування та інші службові атрибути. Також створено логіку пошуку активного замовлення за номером столика та оновлення стану окремих елементів замовлення у відповідь на дії персоналу.

3.4 Реалізація взаємодії в режимі реального часу

Згідно з вимогами, викладеними у другому розділі, однією з ключових функціональних особливостей системи є забезпечення взаємодії в режимі реального часу. Для досягнення цієї мети було обрано стандартний технологічний підхід - використання WebSocket-протоколу, що широко застосовується для реалізації двосторонньої комунікації між клієнтом і сервером у реальному часі [23]. Реалізацію цієї функціональності у серверній частині системи здійснено за

допомогою інструментів фреймворку NestJS, який надає вбудовану підтримку для роботи з WebSocket через модуль @WebSocketGateway.

Компонент WebSocketGateway відповідає за встановлення зв’язку в реальному часі між клієнтами та сервером. Він обробляє підключення, веде облік активних сесій і забезпечує обмін повідомленнями в обох напрямках. У класі передбачено методи для обробки ключових подій: наприклад, коли користувач підключається, система відкриває двосторонній канал комунікації. Надалі через цей канал передаються всі повідомлення, пов’язані із замовленнями - від створення нового запису до зміни статусу чи оновлення інформації про страви.

3.5 Використання EventBus у застосунку. Зв’язування ізольованих модулів

Під час реалізації застосунку було використано модульний підхід, за якого кожна функціональна частина системи - зокрема модулі користувачів, позицій меню, замовлень та WebSocket - працює автономно, виконуючи чітко визначений набір завдань. Водночас, із розвитком бізнес-логіки виникла потреба у налагодженні взаємодії між цими модулями. Наприклад, після створення нового замовлення або зміни статусу страви, інші компоненти системи повинні своєчасно отримувати відповідні повідомлення для забезпечення коректної синхронізації процесів.

Для цього було впроваджено подієвий механізм обміну даними між модулями - Event Bus. У межах фреймворку NestJS така логіка реалізується за допомогою бібліотеки EventEmitter і обробляється через декоратори @OnEvent, що дозволяють реагувати на події, які публікуються в системі [33]. При виникненні події (наприклад, створення замовлення) сервіс формує об’єкт події з відповідними даними та передає його до глобального шини подій за допомогою методу emit(). Підписані слухачі, зокрема у WebSocket-модулі, автоматично отримують цю подію й здійснюють необхідні дії, наприклад, надсилають повідомлення підключеним клієнтам у режимі реального часу.

```
private notifyAboutCreatingOrder(orderedDishes: OrderedDish[]) {
    const message: IWebsocketMessage = {
        type: WebsocketMessageType.OrderCreatingMessage,
        channel: 'Order',
        payload: orderedDishes
    }

    this.eventEmitter.emit(EmittedEvents.WebSocketMessage, message);
}
```

Рисунок 3.2. Передача події створення замовлення через EventEmitter

```
@WebSocketServer()
server: Server

@OnEvent(EmittedEvents.WebSocketMessage)
broadcastMessage(message: IWebsocketMessage) {
    this.server.emit('message', message);
}
```

Рисунок 3.3. Передача події створення замовлення через EventEmitter

Використання Event Bus дозволило досягти кількох важливих цілей: зберегти слабкий зв’язок між модулями без прямої залежності, централізувати обробку подій, забезпечити простоту масштабування системи, а також спростити додавання нової функціональності без необхідності змін у вже реалізованих модулях. Завдяки такому рішенню код залишився чистим, логіка - ізольованою, а система - стійкою до розширення.

3.6 Rest API

Один із ключових елементів серверної частини системи - створення REST API на основі фреймворку NestJS. Через цей інтерфейс здійснюється взаємодія між фронтендом і бекендом: клієнт надсилає запити, а сервер формує й повертає відповідь. Основна логіка обробки HTTP-звернень зосереджена у контролерах, які відповідають за маршрутизацію, початкову обробку даних та передавання їх до

відповідних сервісів. Такий розподіл обов'язків допомагає зберігати чітку архітектуру, полегшує тестування та робить систему гнучкою до змін.

Архітектура контролерів побудована згідно з принципами REST, що дозволяє забезпечити логічну, передбачувану та масштабовану взаємодію з клієнтською частиною. REST (Representational State Transfer) - це стиль архітектури, який передбачає використання стандартних HTTP-методів (GET, POST, PUT, DELETE) для маніпулювання ресурсами, уніфіковану адресацію URL до кожної сутності, відсутність збереження стану між запитами, передачу даних у форматі JSON, а також чітке розмежування відповідальності між шаром обробки запитів та бізнес-логікою [25].

У розробленому застосунку кожен контролер відповідає за окремий домен системи та відображає певну логічну сутність, наприклад: страви, замовлення або користувачі. Така організація сприяє впорядкованості коду й полегшує подальше масштабування програмного рішення.

У додатку Б наведено UML-схему класів, що містить повний перелік методів реалізованих HTTP-ендпойнтів і відображає загальну структуру API.

3.7 Реалізація логіки по збереженню зображень для страв

У другому розділі було визначено вимогу щодо можливості прикріплення зображень до об'єктів типу «страва». На етапі технічного аналізу було розглянуто два варіанти реалізації: зберігання зображень у вигляді коду Base64 у базі даних, або у вигляді файлів у локальній файловій системі [34] (див. табл. 3.1).

Таблиця 3.1

Порівняння способів зберігання зображень: Base64 і файлова система

Критерій	**Зберігання у форматі Base64**	**Зберігання у файловій системі**
Розмір передаваних даних	Збільшується на 30–40%	Мінімальний, передається лише файл
Навантаження на базу даних	Високе, через вбудовані бінарні дані	Низьке, зберігається лише шлях до файлу
Швидкість завантаження	Повільніше через великий обсяг даних	Вища, залежить лише від файлової системи
Зручність обробки	Складно, потрібне декодування Base64	Проста обробка через файлові потоки
Гнучкість масштабування	Обмежена	Легко масштабувати
Простота реалізації	Висока на початку, складніше в підтримці	Просте зберігання та віддача через HTTP

Після аналізу технічних характеристик було обрано збереження зображень у файловій системі. Такий підхід зменшує навантаження на базу даних, полегшує обробку на сервері й забезпечує кращу продуктивність при роботі з великою кількістю файлів.

Передача даних між клієнтською частиною та сервером відбувається через HTTP-запити у форматі multipart/form-data. Такий формат дає змогу надсилати текстову інформацію про страву разом із бінарним файлом зображення. Це стандартне рішення для подібних задач і активно застосовується в сучасних вебсистемах [26].

На стороні сервера обробка таких запитів побудована так, що кожному завантаженому файлу одразу призначається унікальна назва. Далі цей файл зберігається у спеціальній директорії файлової системи, а його назва фіксується у відповідному полі моделі. Завдяки цьому можна сформувати повне посилання на зображення, не зберігаючи сам файл безпосередньо в базі даних.

Щоб забезпечити доступ до збережених зображень, у конфігурації серверного додатку передбачено окремий маршрут для обслуговування статичних файлів. Таким чином, клієнтська частина може отримати доступ до зображення безпосередньо за URL-адресою, сформованою на основі збереженого шляху.

3.8 Архітектурна організація клієнтської частини веб застосунку

Проєктування клієнтської частини веб-додатку передбачало необхідність сформувати логічну, масштабовану та зручну для супроводу структуру. Враховуючи попередній досвід розробки з використанням Angular, було прийнято рішення дотримуватись модульного підходу до структурування проєкту, з урахуванням принципів повторного використання компонентів, типізації та ізоляції відповідальностей.

На початковому етапі було сформовано власну структуру директорій, яка враховує функціональні блоки застосунку та забезпечує зручну навігацію для розробника (див. табл. 3.2) .

Таблиця 3.2

Призначення основних директорій клієнтської частини застосунку

Директорія	**Опис**
constants, enums, interfaces	Містять типи, константи та перерахування, які використовуються по всьому застосунку для забезпечення типізації та повторного використання.
layouts	Шаблони сторінок, що визначають загальну структуру інтерфейсу.
modals	Модальні вікна для підтвердження дій або введення інформації.
pipes	Власні пайпи для форматування даних у шаблонах..
screens	Основні екрани застосунку (авторизація, адмін, офіціант) та їх дочірні компоненти.
services	Сервіси з реалізацією бізнес-логіки, що повторно використовуються в компонентах.

Продовження таблиці 3.2

shared	Багаторазові UI-компоненти , не прив’язані до конкретних сторінок.
utils	Функції загального призначення.

Запропонована структура проєкту чітко розмежовує зони відповідальності, сприяє дотриманню принципів повторного використання коду (DRY) і полегшує командну розробку. Такий підхід відповідає сучасним практикам Angular-розробки та рекомендований в офіційному Angular Style Guide [27].

3.9 Реєстрація навігації на основі ролей користувачів

Реалізація маршрутизації в клієнтській частині враховує ролі користувачів і забезпечує контрольований доступ до відповідних розділів інтерфейсу. У системі визначено три типи ролей: адміністратор, офіціант і кухар. Після автентифікації кожен користувач автоматично перенаправляється на свій розділ, відповідно до функціональних обов’язків.

Для реалізації цієї логіки було створено окремий файл конфігурації маршрутів, який зареєстровано в модулі маршрутизатора RouterModule [35]. Після реєстрації шляхи імпортуються до головного модуля програми.

Кожен маршрут прив’язано до конкретного компонента, який відповідає інтерфейсу певної ролі. У разі відсутності автентифікації користувача або переходу за недійсною адресою, система автоматично перенаправляє користувача на сторінку входу.

Таблиця 3.3

Маршрути, компоненти та їх призначення

Маршрут (URL)	**Компонент**	**Призначення**
/admin	AdminComponent	Відображення інтерфейсу адміністратора.
/waiter	WaiterComponent	Інтерфейс для роботи офіціанта.
/cook	CookScreenComponent	Робоче середовище для кухаря.
/sign-in	SignInComponent	Сторінка входу користувача до системи.

3.10 Розробка клієнтських сервісів і управління даними

У структурі клієнтської частини особливе місце займають сервіси - вони забезпечують обробку даних, спілкування з сервером та підтримку локального стану застосунку. У відповідності до принципів Angular-архітектури, сервіси використовуються для реалізації всієї бізнес-логіки, тоді як компоненти зосереджені виключно на відображенні інтерфейсу.

Сервіси в системі реалізовані як інжектабельні класи, що централізовано виконують обмін даними між UI та серверною частиною. Уся взаємодія з API зосереджена в окремому сервісі, який використовує HTTP-клієнт Angular. На його основі побудовані спеціалізовані сервіси для роботи з окремими доменами, такими як страви, замовлення чи користувачі. Це дозволило уніфікувати логіку запитів та мінімізувати дублювання коду.

Управління станом в застосунку базується на реактивному підході, який забезпечує бібліотека RxJS [15]. Дані надходять у вигляді потоків, а компоненти, підписавшись на них, автоматично реагують на будь-які зміни. Такий механізм дозволяє підтримувати інтерфейс у постійній актуальності, без необхідності вручну оновлювати відображення.

Важливим елементом стало впровадження логіки взаємодії з WebSocket-сервером. Для цього був створений окремий сервіс, що відповідає за з’єднання, приймання повідомлень і їх розповсюдження всередині системи. Весь обмін у реальному часі побудований на слабкозв’язаній інтеграції з внутрішнім механізмом подій - завдяки цьому структуру легко адаптувати під нові сценарії без втручання в основну логіку з’єднання. Як транспорт було обрано бібліотеку Socket.IO, яка забезпечує стабільне двостороннє спілкування між клієнтом і сервером [28].

У клієнтській частині централізовано організовано логіку авторизації, управління замовленнями, роботи з користувачами та іншими об’єктами системи. Відповідальні за це сервіси виступають посередниками між інтерфейсом і сервером, забезпечуючи цілісність даних та відповідність функціональним вимогам.

3.11 Логіка компонентів і принципи повторного використання коду

У межах цього підрозділу не розглядатиметься детальна реалізація кожного окремого компонента, оскільки основна увага зосереджена не на технічних деталях шаблонів чи стилізації, а на загальній архітектурній побудові клієнтського інтерфейсу та логіці взаємодії з користувачем.

Компонентна структура застосунку була організована з урахуванням ролей користувачів, відповідно до функціональних вимог, визначених у другому розділі. Усі інтерфейсні елементи згруповані відповідно до трьох ключових сценаріїв використання системи: адміністратор, офіціант і кухар. Кожен із цих сценаріїв реалізований у межах окремого модуля, що містить відповідні компоненти інтерфейсу.

Загальний підхід до реалізації логіки компонентів зосереджений на скороченні обробки даних безпосередньо в шаблонах. Вся бізнес-логіка винесена в сервіси, з якими працюють компоненти, отримуючи необхідну інформацію через реактивні потоки. Дані передаються за допомогою підписки на Observables, тож

інтерфейс завжди відображає актуальний стан без потреби в ручному оновленні. У випадках змін у реальному часі - наприклад, при надходженні нових замовлень або оновленні статусу страв - відповідні сигнали надходять до компонентів через сервіси, і стан автоматично синхронізується.

Важливим аспектом реалізації стала підтримуваність і повторне використання коду. Компоненти, що не залежать від певної ролі користувача, винесені в окрему директорію shared. Такий підхід зменшує дублювання, зберігає єдиний стиль і логіку взаємодії. Уся структура відповідає принципам, викладеним у офіційному Angular Style Guide [24].

3.12 Реактивна модель взаємодії з даними: використання RxJS в Angular

Одним із ключових архітектурних принципів клієнтської частини застосунку, реалізованої на Angular, є використання бібліотеки RxJS для роботи з асинхронними даними. Компоненти, що відповідають за візуалізацію динамічного контенту, побудовані за принципами реактивного програмування - це дає змогу підтримувати інтерфейс у синхронному стані з поточними даними сервісів без потреби в ручному оновленні.

Реактивний підхід базується на потоках даних і автоматичному реагуванні на зміни. Відповідно до офіційної документації, йдеться про орієнтацію на асинхронні потоки подій, де зміна стану системи сприймається як послідовність реакцій на зовнішні сигнали [25].

Одним із прикладів практичного застосування реактивного підходу в межах даного проєкту є компонент CookScreenComponent, призначений для користувачів із роллю кухаря. Основна функція цього екрану - динамічне відображення замовлень, що перебувають у процесі приготування, з можливістю змінювати їхній стан після завершення готування. Потік даних preparingDishes$ створюється на основі реактивного джерела і трансформується безпосередньо в межах

компонента з урахуванням ролі користувача, фільтрації контекстних замовлень (див. рис. 3.2).

```
export class CookScreenComponent {

  constructor(public orderService: OrderService,
    public applicationStateService: ApplicationStateService,
    public modalService: ModalService
    ) {

  }


  preparingDishes$: Observable<OrderedDish[]> = this.orderService.preparingDishes$.pipe(
    map(dishes => dishes
      ?.filter(dish => dish.orderDishState === OrderDishState.InProgress && this.applicationStateService.user?.cookingPlace === dish.cookingPlace)
    )
  );
```

Рисунок 3.4. Фрагмент коду формування потоку замовлень у компоненті кухаря

Використання пайпа async у шаблонах Angular спрощує роботу з Observables: значення автоматично підтягуються до інтерфейсу без додаткового керування підписками. Це зменшує ризик витоків пам’яті та допомагає зберегти чисту, зрозумілу архітектуру.

```
...
<app-preparing-dishes [hasOnCardClickAction]="true"
                      [orderedDishes]="preparingDishes$ | async"
                      (onCardClick)="onCardClick($event)">
</app-preparing-dishes>

```

Рис. 3.5. Приклад використання пайпа async у шаблоні Angular

RxJS також активно використовується на інших екранах застосунку, зокрема в інтерфейсах адміністратора й офіціанта. Компоненти, що працюють з динамічними даними, реалізовані як реактивні: кожна зміна стану джерела автоматично відображається у візуальному представленні, що підвищує узгодженість інтерфейсу. Крім зменшення технічної складності при роботі з асинхронністю, реактивний підхід забезпечує прозорість і передбачуваність у поведінці компонентів.

3.13 Форматування даних у шаблонах за допомогою пайпів

Одним із поширених рішень при побудові інтерфейсів в Angular є використання пайпів (pipes) - компактного інструменту для трансформації значень безпосередньо у шаблоні компонента. Завдяки цьому механізму можливо ефективно відокремити логіку форматування від компонента, чи сервісу й зробити розмітку більш зрозумілою. Пайп реалізується у вигляді класу з інтерфейсом PipeTransform, що визначає метод transform(), у якому й описується відповідна логіка перетворення.

Згідно з офіційною документацією Angular, пайп визначається як інструмент, що приймає дані на вході та трансформує їх у бажане представлення для відображення в шаблоні: «A pipe takes in data as input and transforms it to a desired output for display in a template» [29].

Завдяки пайпам логіку форматування можна винести за межі компонента, залишаючи шаблон чистим, декларативним і зрозумілим. Це особливо корисно, коли одна й та сама операція форматування використовується у різних частинах застосунку.

У межах розробки було створено користувацький пайп NegativeTimePipe, який забезпечує форматування числового значення, що представляє час у мілісекундах, у вигляді рядка формату мм:сс. Якщо значення є від’ємним, результат додатково містить знак мінус перед часом. Реалізація NegativeTimePipe передбачає попереднє приведення вхідного значення до абсолютного для обчислення хвилин і секунд, після чого результати трансформуються у символьне представлення з провідними нулями. Якщо початкове значення було негативним, до результату додається відповідний знак мінус.

3.14 Реалізація адаптивної верстки

Візуальна частина застосунку була спроєктована з урахуванням потреб користувачів, які можуть взаємодіяти з системою як на стаціонарних, так і на

мобільних пристроях. Особливу увагу було приділено адаптивній верстці, що забезпечує коректне відображення інтерфейсу незалежно від розміру екрана. Такий підхід особливо важливий у контексті ресторанного бізнесу, де офіціанти можуть користуватись застосунком з планшетів або смартфонів.

Адаптивність інтерфейсу передусім була реалізована для екрана офіціанта, який взаємодіє з відвідувачами та обслуговує столи. Основна сітка побудована за допомогою CSS Grid Layout - інструмента для створення двовимірної структури сторінки з гнучким керуванням розташуванням елементів [30]. Така побудова не лише підвищує зручність сприйняття, а й відтворює логіку розміщення столів у залі, що спрощує орієнтацію працівника в робочому просторі.

У тих випадках, коли потрібно було реалізувати динамічне вирівнювання або зміну напряму розміщення елементів, застосовувалась технологія Flexbox. Її використовували для створення навігаційних блоків, панелей керування, форм і кнопок. Flexbox забезпечує зручну одноосну верстку, яка легко підлаштовується під різні розміри екрана [31].

Крім того, у стилях активно використовувалися медіазапити (@media), які дозволяють підлаштовувати розмір шрифтів, відображення блоків, кількість колонок та внутрішні відступи залежно від ширини екрана. Це надало змогу створити інтерфейс, що залишається повністю функціональним і візуально збалансованим як на десктопних, так і на мобільних пристроях [32].

Поєднання CSS Grid, Flexbox і медіазапитів дало змогу створити гнучке, адаптивне компонування інтерфейсу без втрати логічної структури й семантики. Завдяки цьому компоненти, зокрема інтерфейс офіціанта, коректно відображаються на різних типах пристроїв, зберігаючи зручність у користуванні.

3.15 Висновок до розділу

У цьому розділі було висвітлено загальні принципи та технічні підходи, що лягли в основу проєктування клієнтської та серверної частин інформаційної системи.

При формуванні архітектури застосовувались сучасні інженерні практики, зокрема модульність, інкапсуляція, розділення відповідальностей, а також багаторівнева організація коду.

Використання таких технік, як шаблон репозиторію, централізовані оркестратори, реактивне управління станом та слабкозв’язана взаємодія між компонентами, дозволило сформувати гнучке програмне середовище. Усі ці підходи були спрямовані на досягнення ключових цілей — забезпечення масштабованості, зручності супроводу, розширюваності системи та підтримки її стабільної роботи у динамічному середовищі.

ВИСНОВОК

У межах дослідження було проаналізовано основні внутрішні проблеми, характерні для традиційної моделі обслуговування у сфері ресторанного бізнесу. Йшлося, зокрема, про неузгодженість дій персоналу, недостатню автоматизацію, ускладнене управління замовленнями та неефективне використання ресурсів. З метою глибшого розуміння ситуації було вивчено низку вже реалізованих систем автоматизації, що дозволило виокремити їхні переваги, недоліки та найбільш вдалі технічні та організаційні рішення.

На основі проведеного аналізу було сформовано власне бачення архітектури та функціональності системи, що поєднує перевірені на практиці підходи й найкращі галузеві практики. Запропоноване рішення спрямоване на покращення координації між працівниками, підвищення оперативності обслуговування, автоматизацію внутрішніх процесів та управління діяльністю закладу.

В основі реалізації застосунку лежить використання перевірених патернів, модульної структури та актуальних підходів до організації коду. Такий підхід дозволив створити стабільну, легко підтримувану систему, готову до масштабування та подальшої інтеграції зі сторонніми сервісами, такими як аналітичні платформи, системи лояльності чи CRM-рішення. Це робить проєкт придатним для використання у реальному середовищі з можливістю поступового розширення функціональності відповідно до змін потреб бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Communication Issues Among Top Challenges in Commercial Kitchens Across the US and Canada [Електронний ресурс]. – Режим доступу: https://www.lightspeedhq.com/news/communication-issues-among-top-challenges-in-commercial-kitchens-across-the-us-and-canada/

2. **Davis K.G., Wills A.C., Kotowski S.E.** Quantification of Ergonomic Exposures for Restaurant Servers // Journal of Ergonomics. – 2016. – Vol. 6, Issue 3. – DOI: 10.4172/2165-7556.1000166. – Режим доступу: https://www.researchgate.net/publication/273296845 (дата звернення: 17.01.2025).

3. Mirabelli T. The Language and Literacy of Food Service Workers [Електронний ресурс]. – Режим доступу: https://www.umsl.edu/~alexanderjm/mirabelli_learning-to-serve.pdf

4. Закон України «Про захист прав споживачів» від 12.05.1991 №1023-XII [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1023-12

5. Про компанію R-Keeper [Електронний ресурс]. – Режим доступу: https://rkeeper.com.ua/#about-1

6. Poster POS – тарифи [Електронний ресурс]. – Режим доступу: https://joinposter.com/ua/pricing

7. Altos POS [Електронний ресурс]. – Режим доступу: https://altospos.com/

8. Poster. Stop List Application [Електронний ресурс]. – Режим доступу: https://joinposter.com/applications/out-of-stock-list

9. Vouch POS. Restaurant Kitchen Display System [Електронний ресурс]. – Режим доступу: https://vouchpos.com/restaurant-kitchen-display-system/

10. NestJS Documentation. Creating scalable server-side applications [Електронний ресурс]. – NestJS, 2024. – Режим доступу: https://docs.nestjs.com

11. MongoDB Manual. Introduction to MongoDB [Електронний ресурс]. – MongoDB Inc., 2024. – Режим доступу: https://www.mongodb.com/docs/manual

12. Angular documentation. What is Angular? [Електронний ресурс]. – Режим доступу: https://angular.io/guide/what-is-angular

13. Vue.js Guide. Introduction [Електронний ресурс]. – Режим доступу: https://vuejs.org/guide/introduction.html

14. React documentation. Getting Started [Електронний ресурс]. – Режим доступу: https://reactjs.org/docs/getting-started.html

15. RxJS Documentation. Overview and Guide [Електронний ресурс]. – Режим доступу: https://rxjs.dev/guide/overview

16. Object-Oriented Programming (OOP) – GeeksforGeeks [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java

17. KISS, DRY, SOLID, YAGNI — навіщо дотримуватись принципів програмування? – Senior.ua [Електронний ресурс]. – Режим доступу: https://senior.ua/articles/kiss-dry-solid-yagni--navscho-dotrimuvatis-principv-programuvannya

18. Angular Material documentation. Overview [Електронний ресурс]. – Режим доступу: https://material.angular.io/

19. PrimeNG documentation. Components Library for Angular [Електронний ресурс]. – Режим доступу: https://primeng.org/

20. Mongoose – Guides. Define models using code-first approach [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs/models.html

21. Repository pattern in software development – Microsoft Learn [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/repository-pattern

22. Mongoose documentation. Models and schemas [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs/models.html

23. WebSockets – MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

24. NestJS documentation. Events [Електронний ресурс]. – Режим доступу: https://docs.nestjs.com/events

25. Fielding R. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс]. – Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

26. HTML Standard: Form Submission – multipart/form-data – W3C [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org/multipage/form-control-infrastructure.html#multipart-form-data

27. Angular Style Guide. Angular documentation [Електронний ресурс]. – Режим доступу: https://angular.io/guide/styleguide

28. Socket.IO Documentation [Електронний ресурс]. – Режим доступу: https://socket.io/docs/v4/

29. Angular Pipes [Електронний ресурс]. – Режим доступу: https://angular.io/guide/pipes

30. CSS Grid Layout – MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout

31. CSS Flexible Box Layout – MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout

32. Responsive Design with Media Queries – MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

33. NestJS. Events – Офіційна документація NestJS [Електронний ресурс]. – Режим доступу: https://docs.nestjs.com/techniques/events

34. Base64 — Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Base64

35. Angular. RouterModule — API documentation [Електронний ресурс]. – Режим доступу: https://angular.dev/api/router/RouterModule

Додаток А

Рисунок А.1. UML-діаграма варіантів використання для ролі кухаря

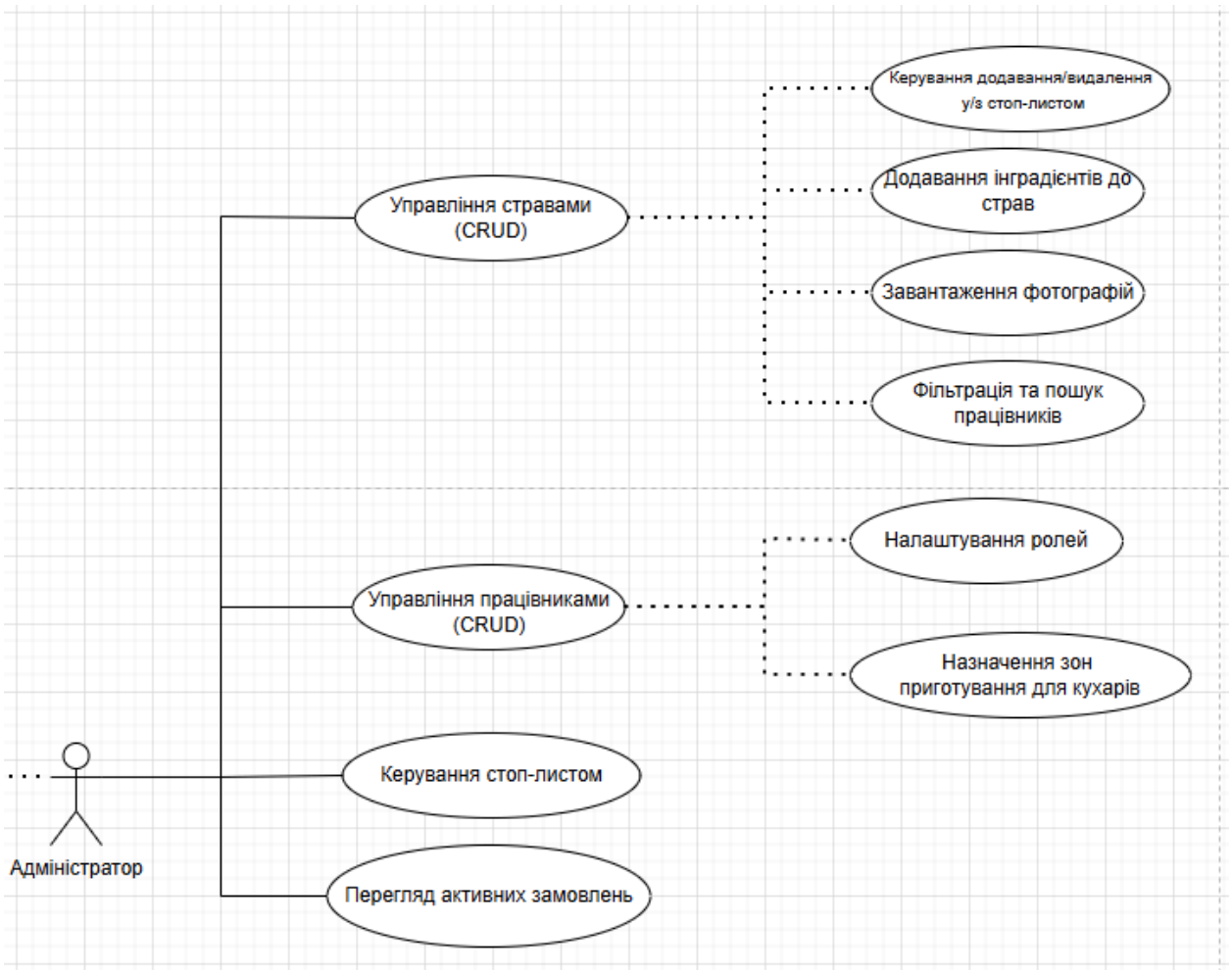

Рисунок А.2. UML-діаграма варіантів використання для ролі адміністратора

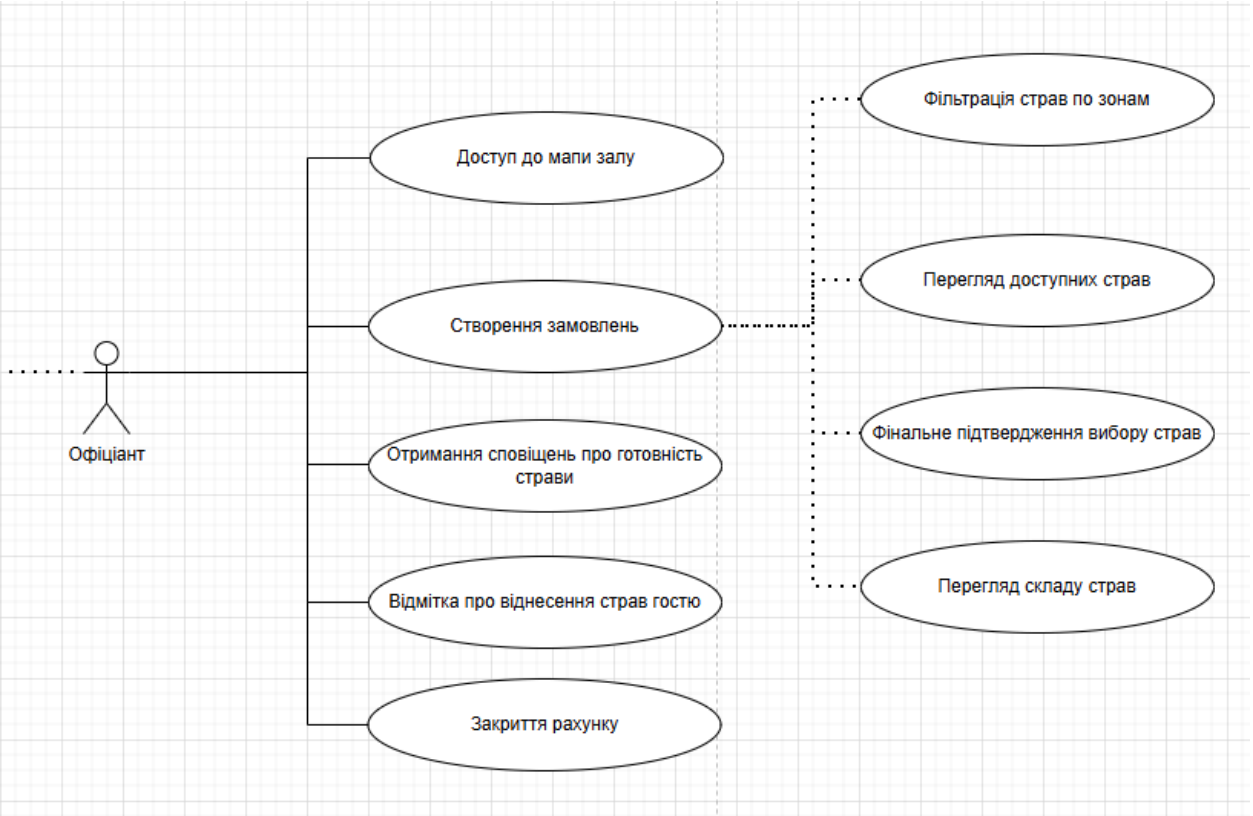

Рисунок А.3. UML-діаграма варіантів використання для ролі офіціанта

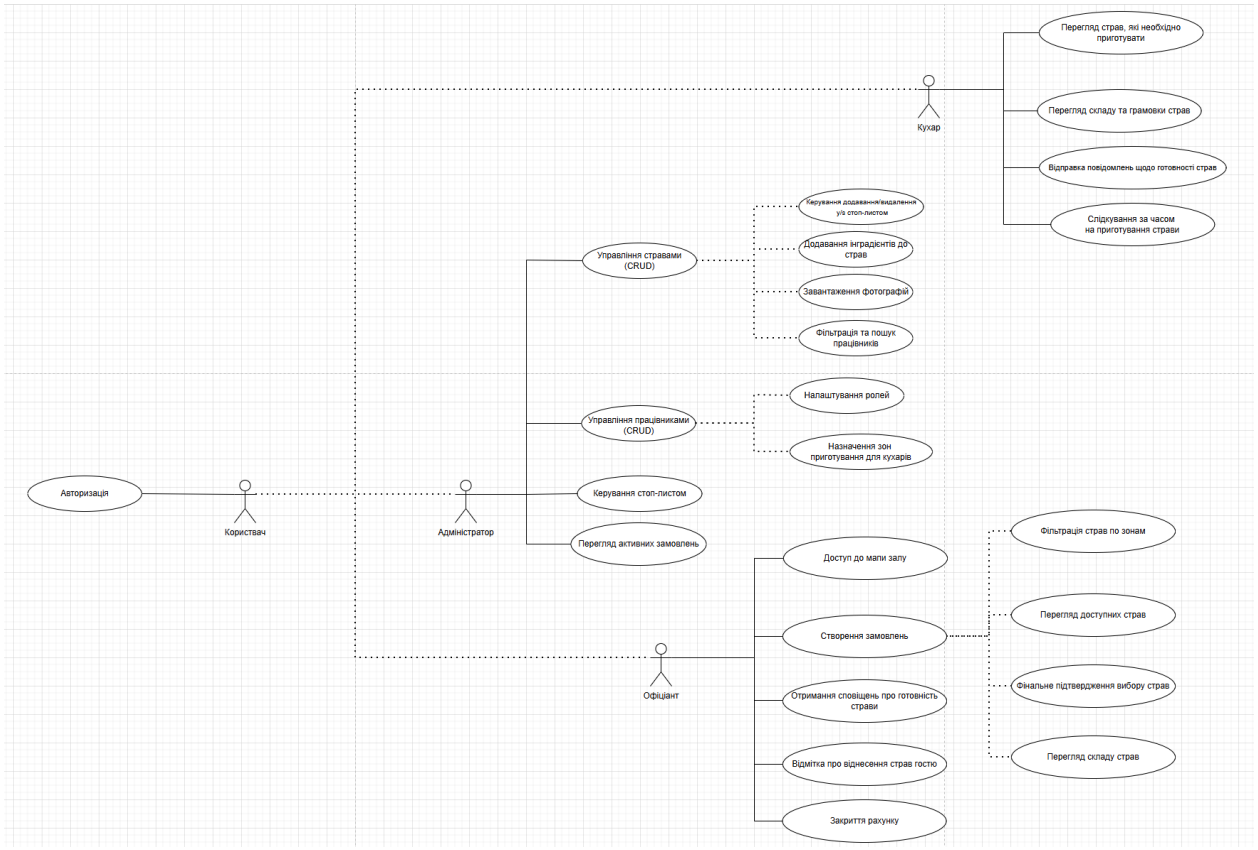

Рисунок А.4. Загальна UML-діаграма варіантів для ролей в системі

Додаток Б

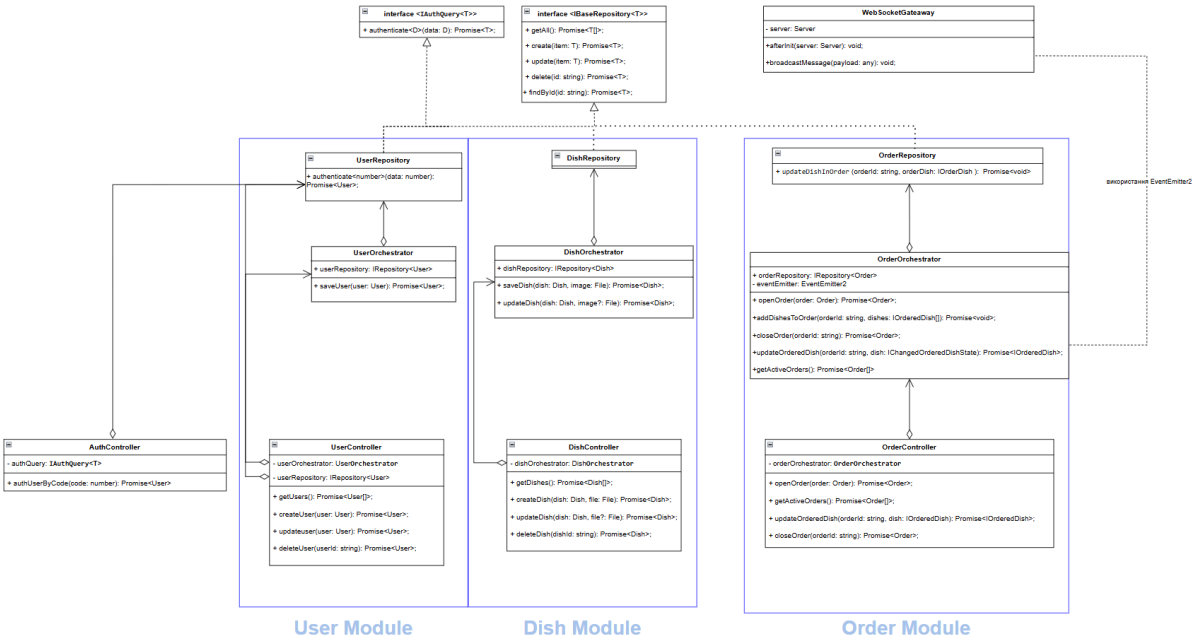

Рисунок Б.1. Діаграма класів серверної частини додатку

Додаток В

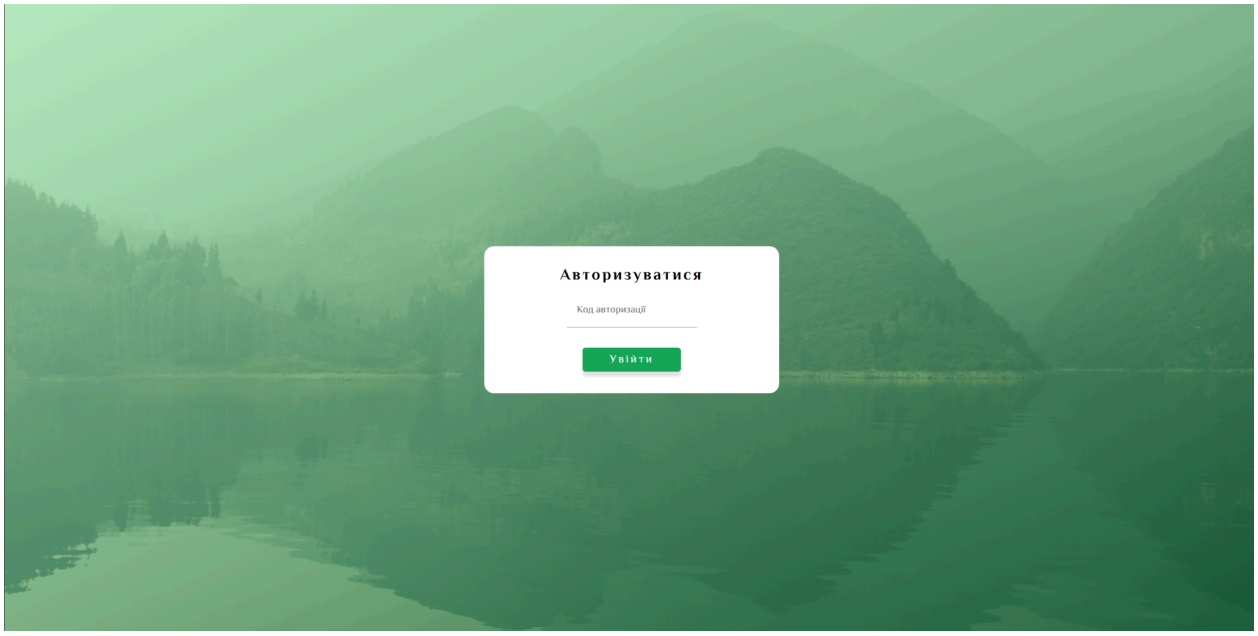

Рисунок В.1. Сторінка авторизації

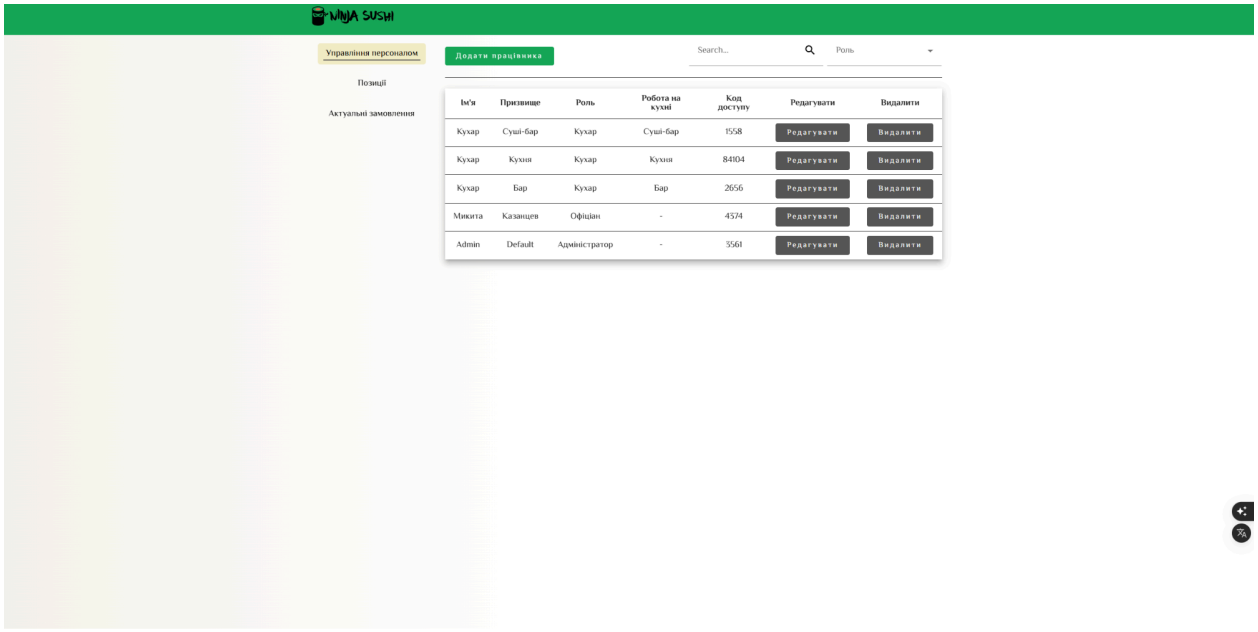

Рисунок В.2. Сторінка керування персоналом (інтерфейс адміністратора)

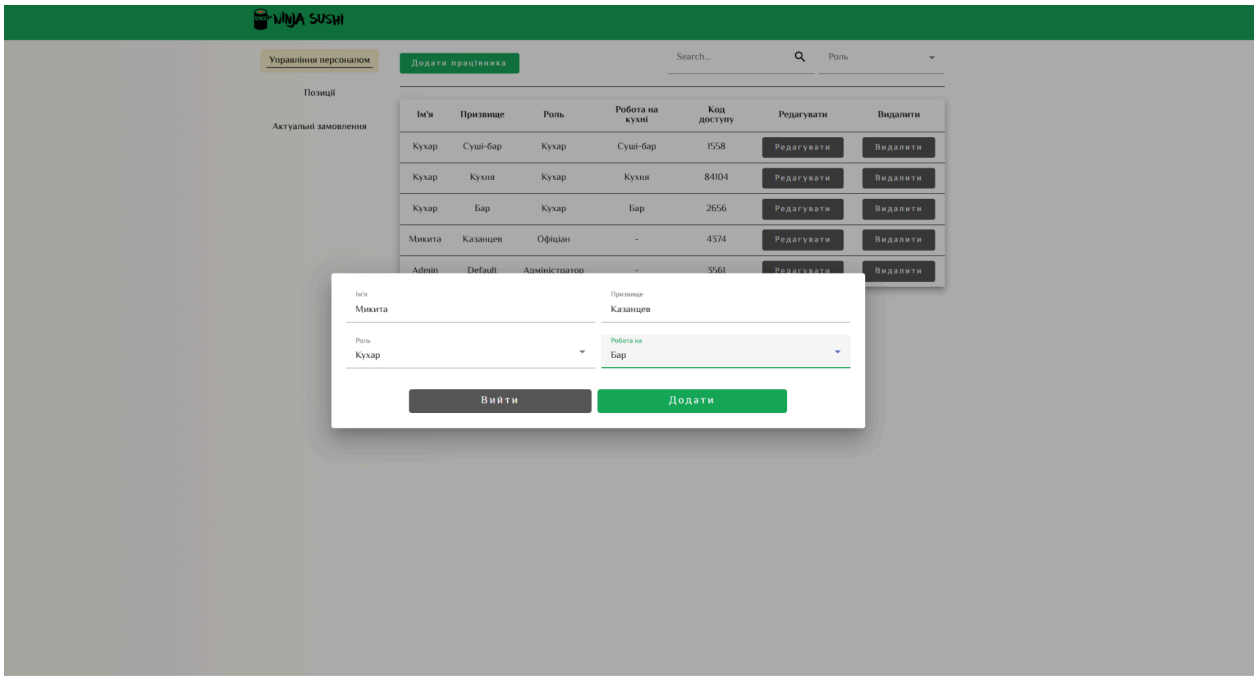

Рисунок В.3. Модальне вікно додавання нового працівника (інтерфейс адміністратора)

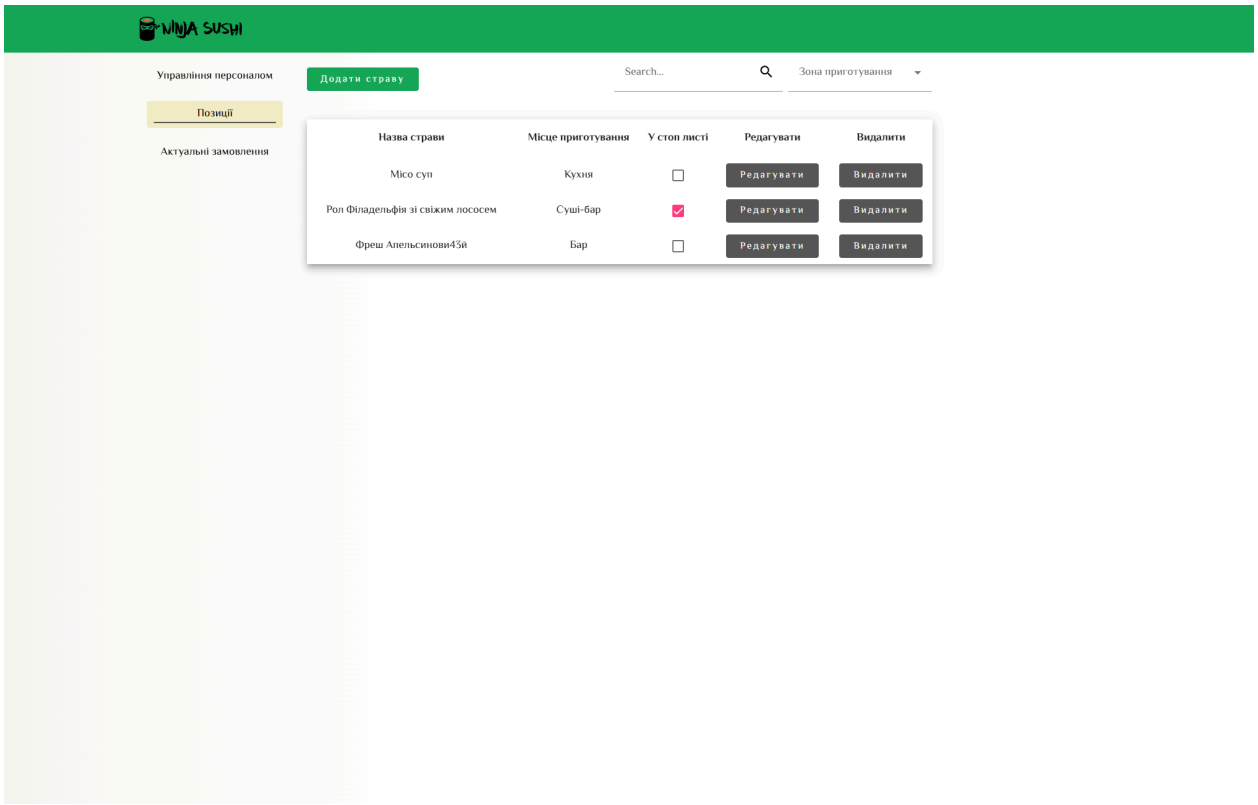

Рисунок В.4. Сторінка керування позиціями меню (інтерфейс адміністратора)

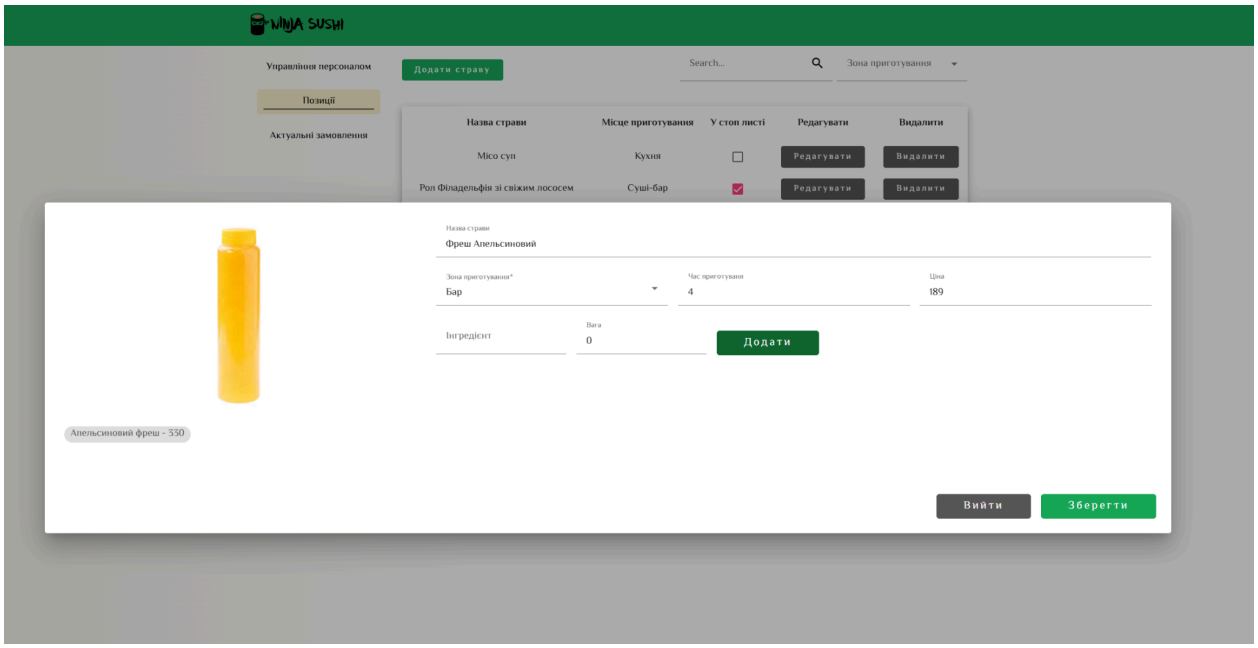

Рисунок В.5. Модальне вікно редагування страви (інтерфейс адміністратора)

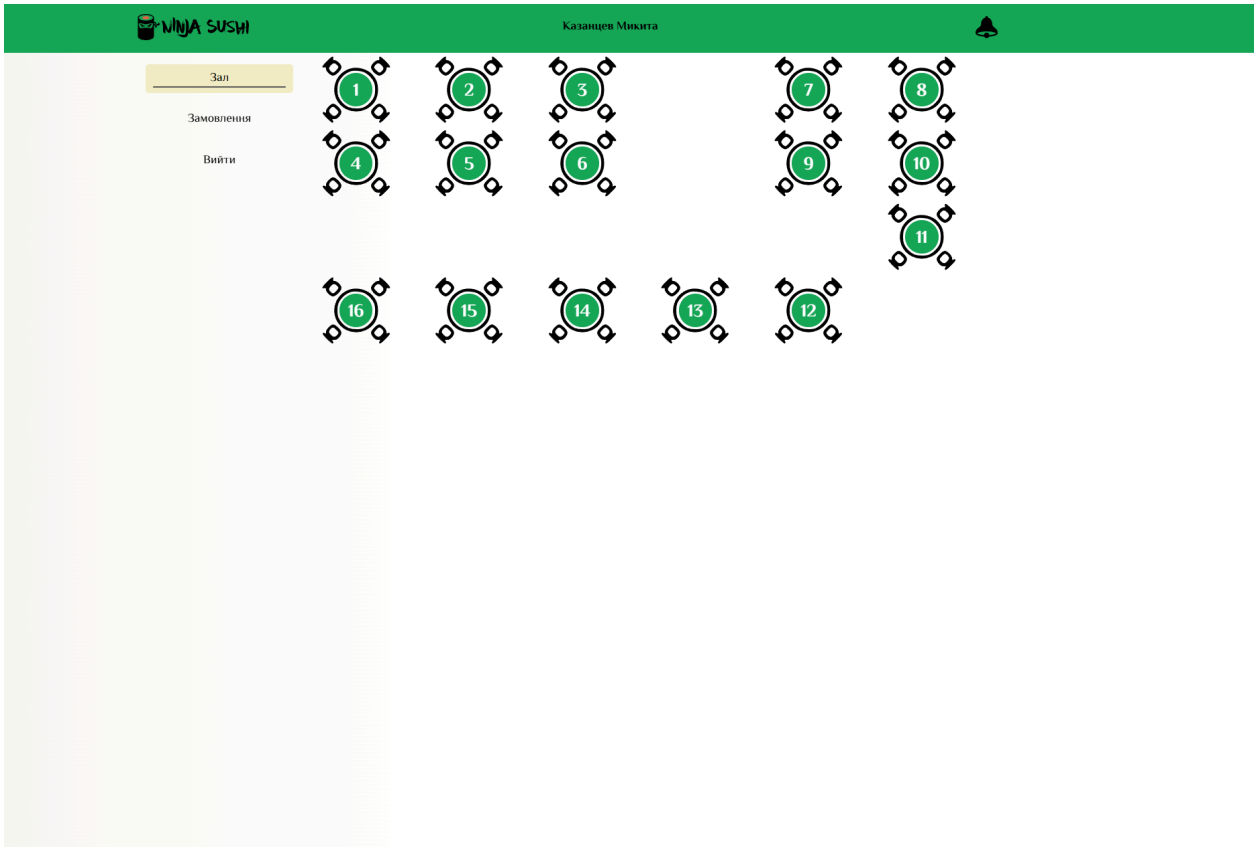

Рисунок В.6. Сторінка мапи залу (інтерфейс офіціанта)

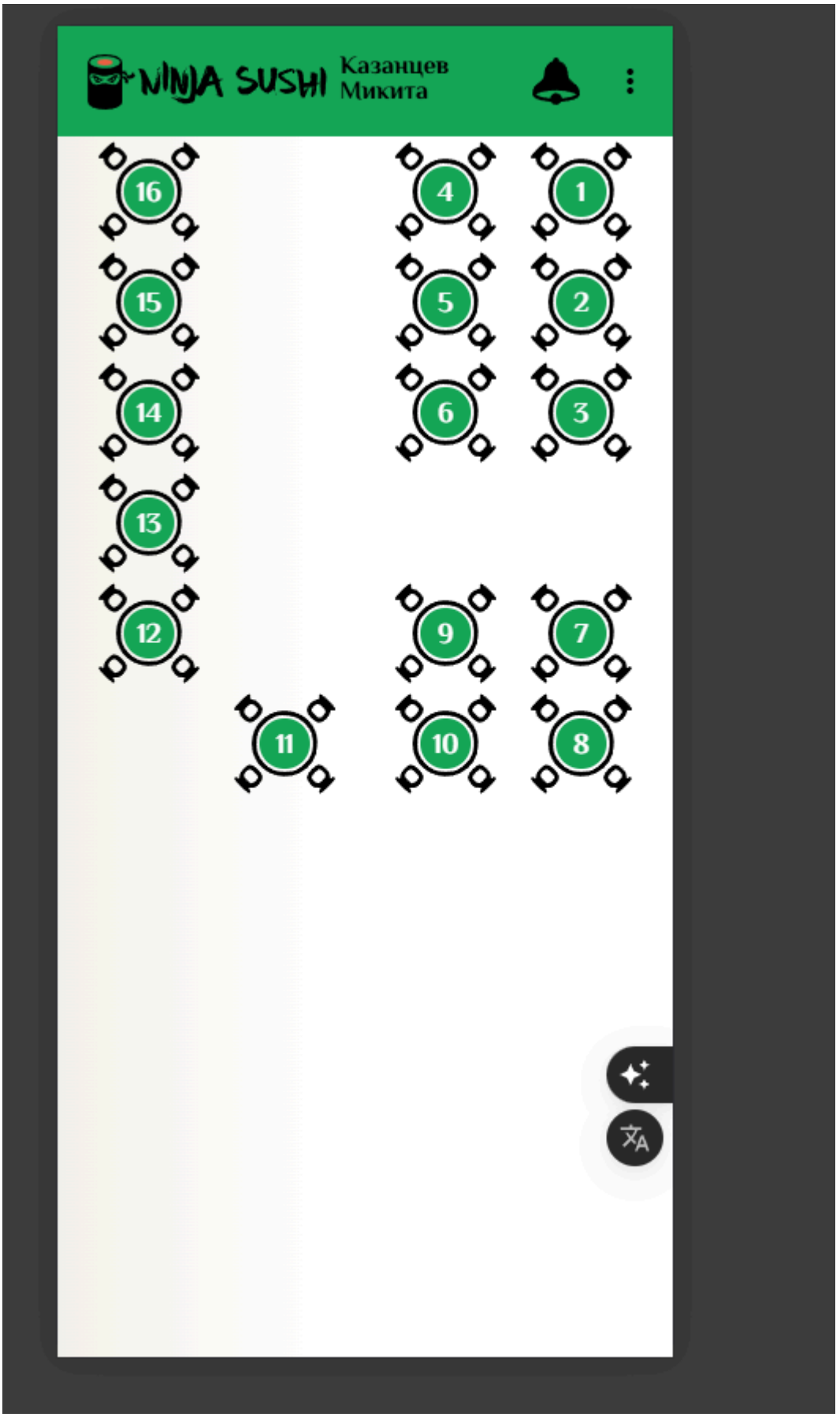

Рисунок В.7. Сторінка мапи залу для екрану смартфону (інтерфейс офіціанта)

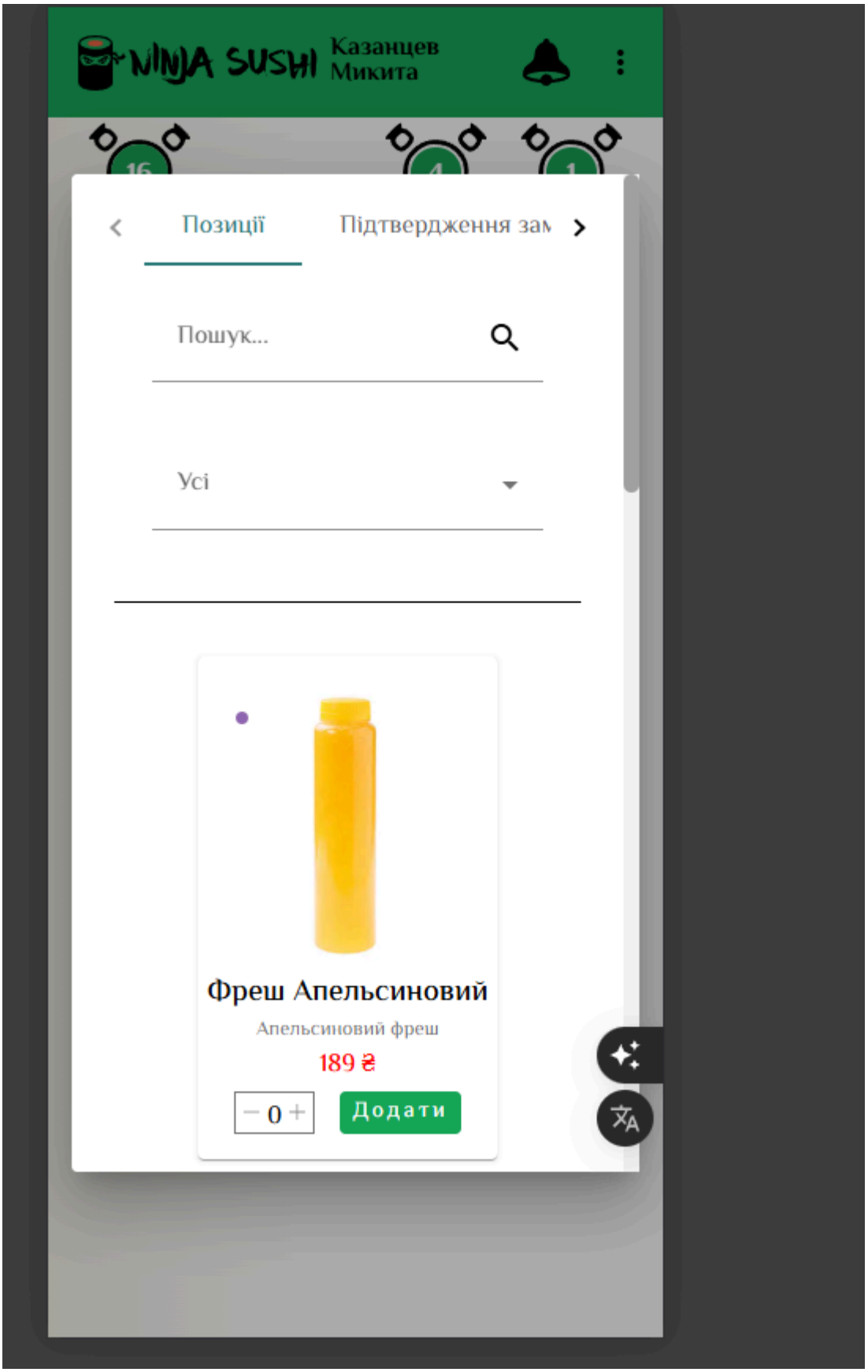

Рисунок В.8. Вибір позиції з меню в мобільній версії (інтерфейс офіціанта)

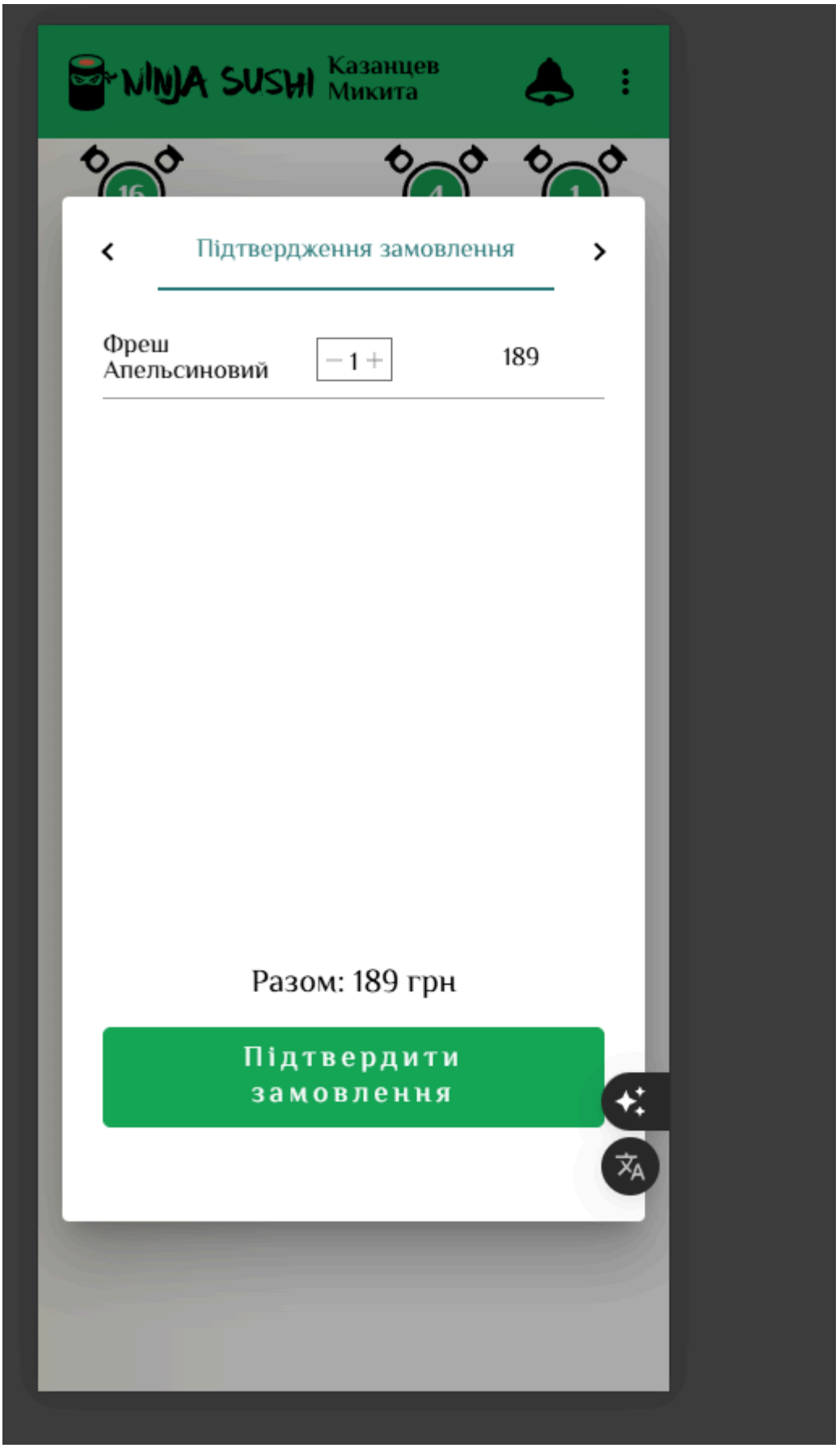

Рисунок В.9. Підтвердження замовлення в мобільній версії (інтерфейс офіціанта)

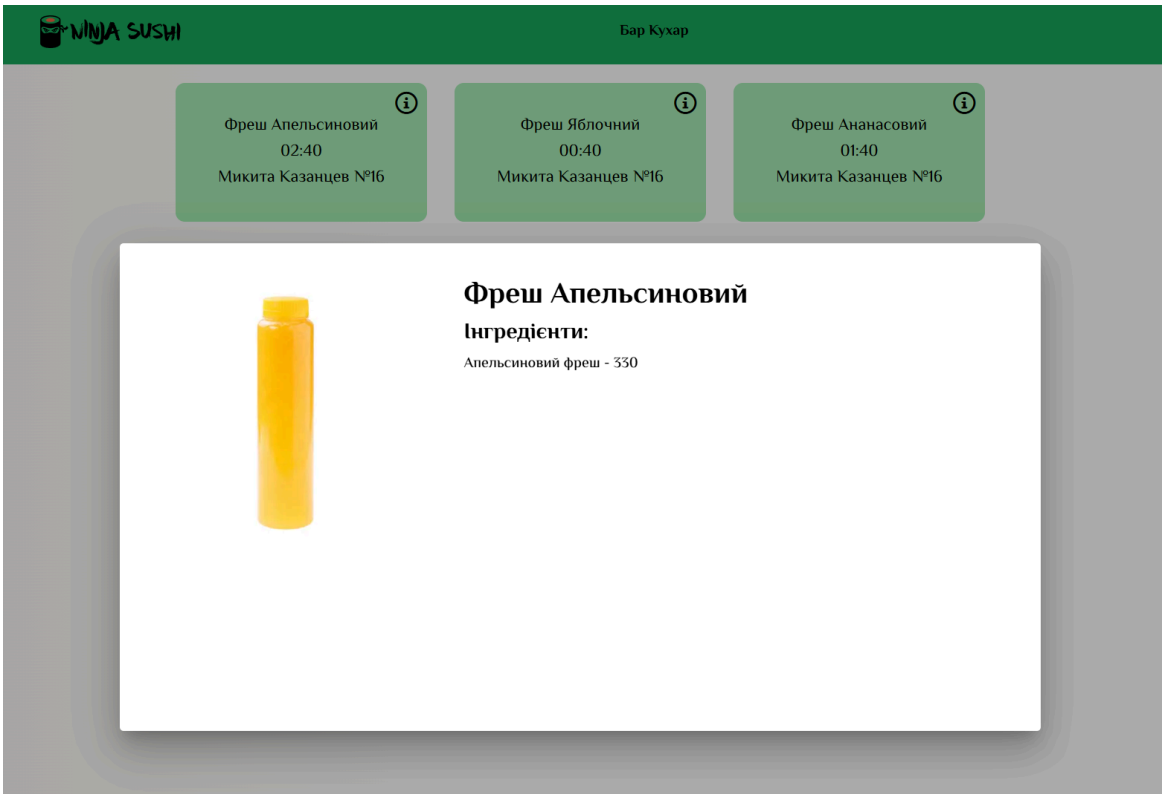

Рисунок В.10. Інтерфейс кухаря з деталями замовлення та складом страви

Рисунок В.11. Список активних замовлень з таймерами приготування (інтерфейс кухаря)

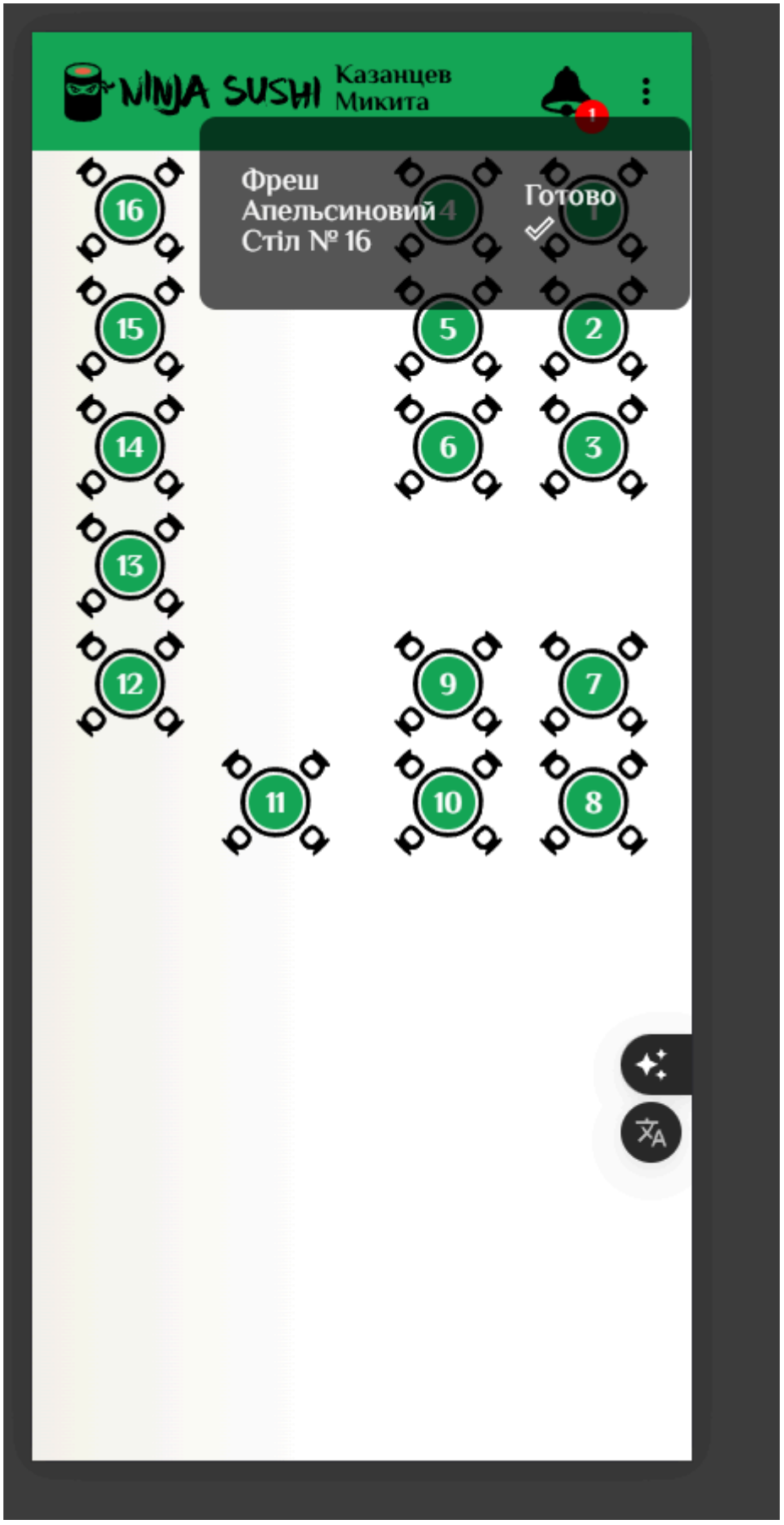

Рисунок В.12. Повідомлення офіціанту про готовність страви

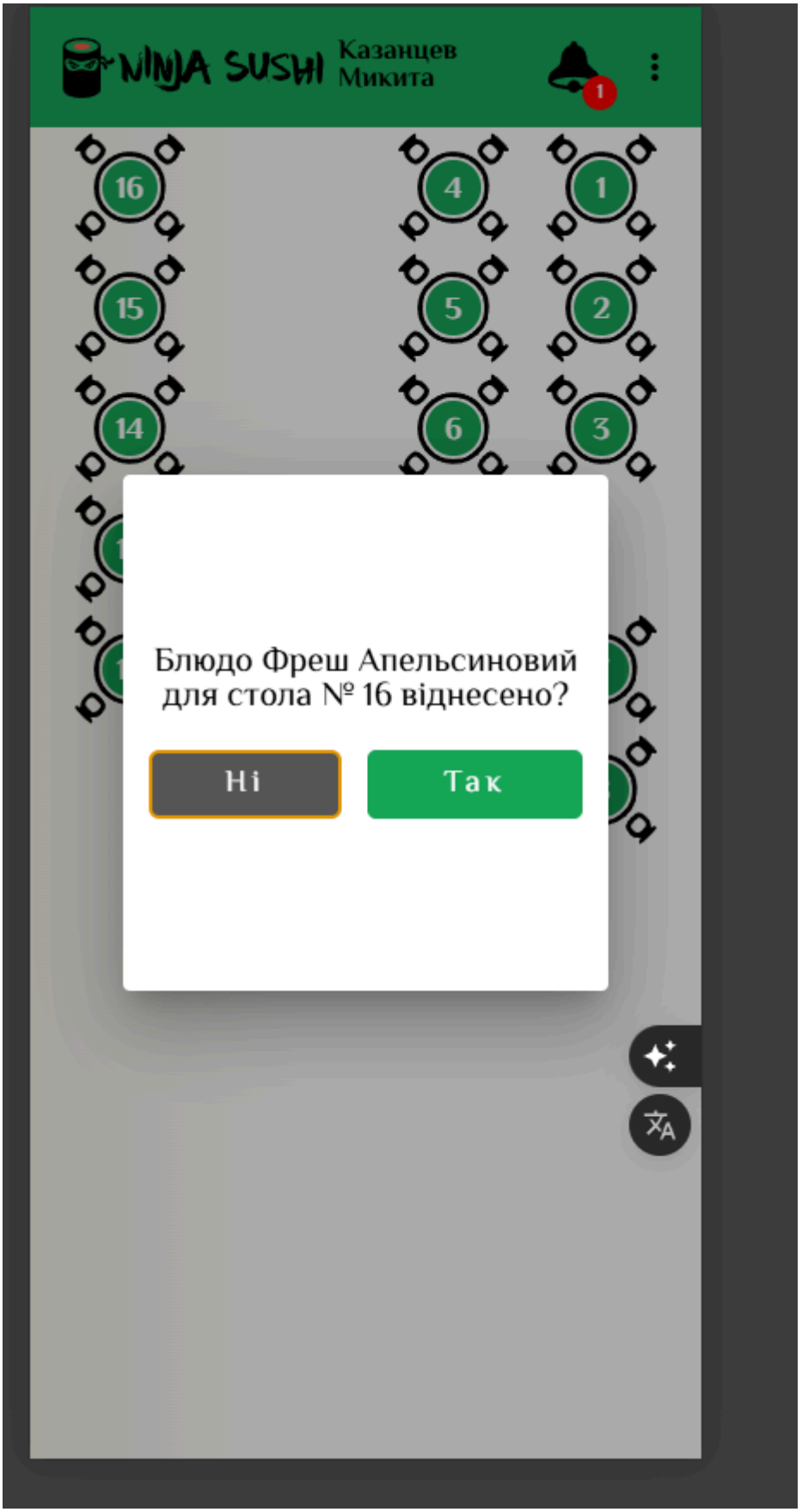

Рисунок В.13. Підтвердження доставки страви офіціантом

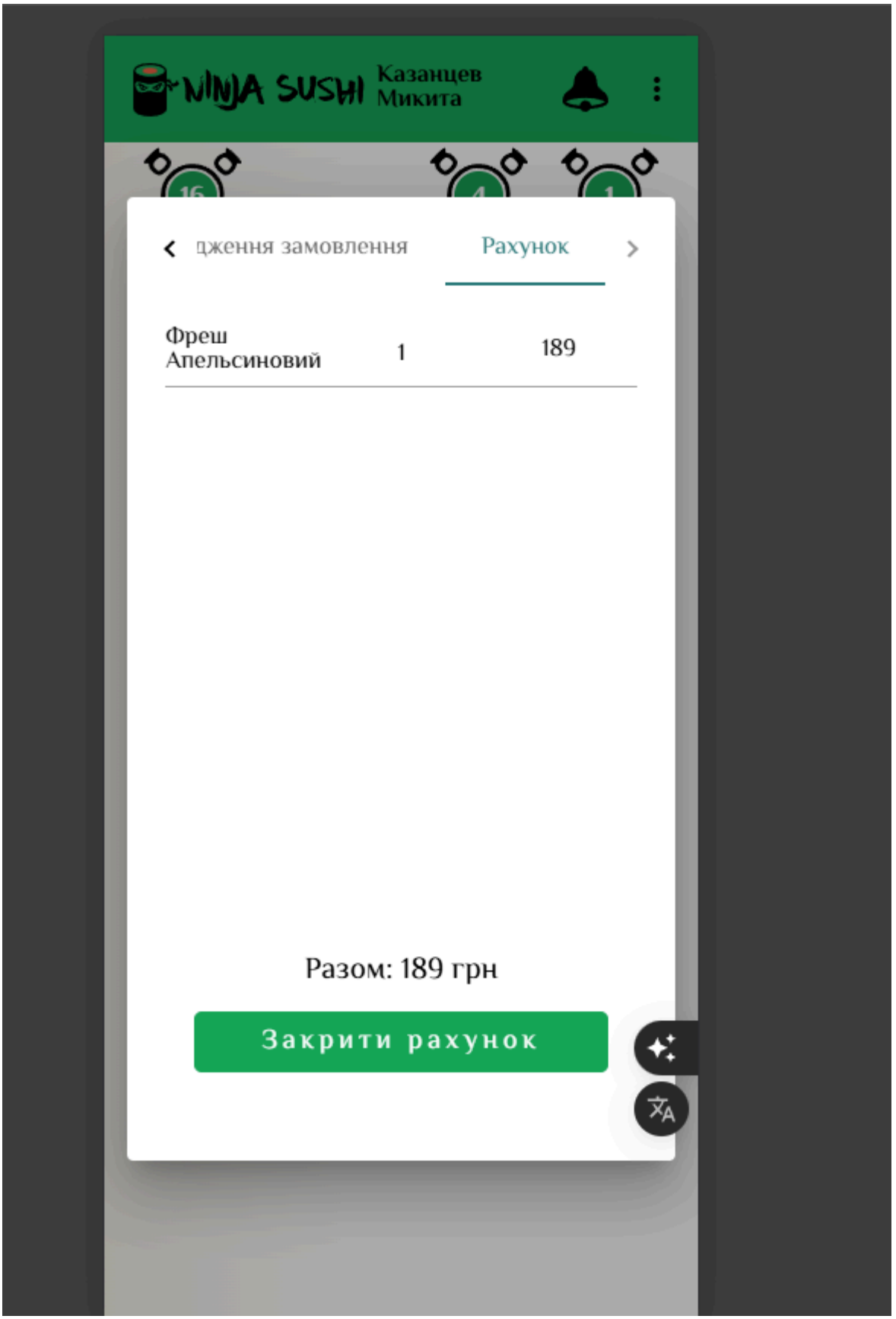

Рисунок В.14. Перегляд рахунку та завершення замовлення (інтерфейс офіціанта)

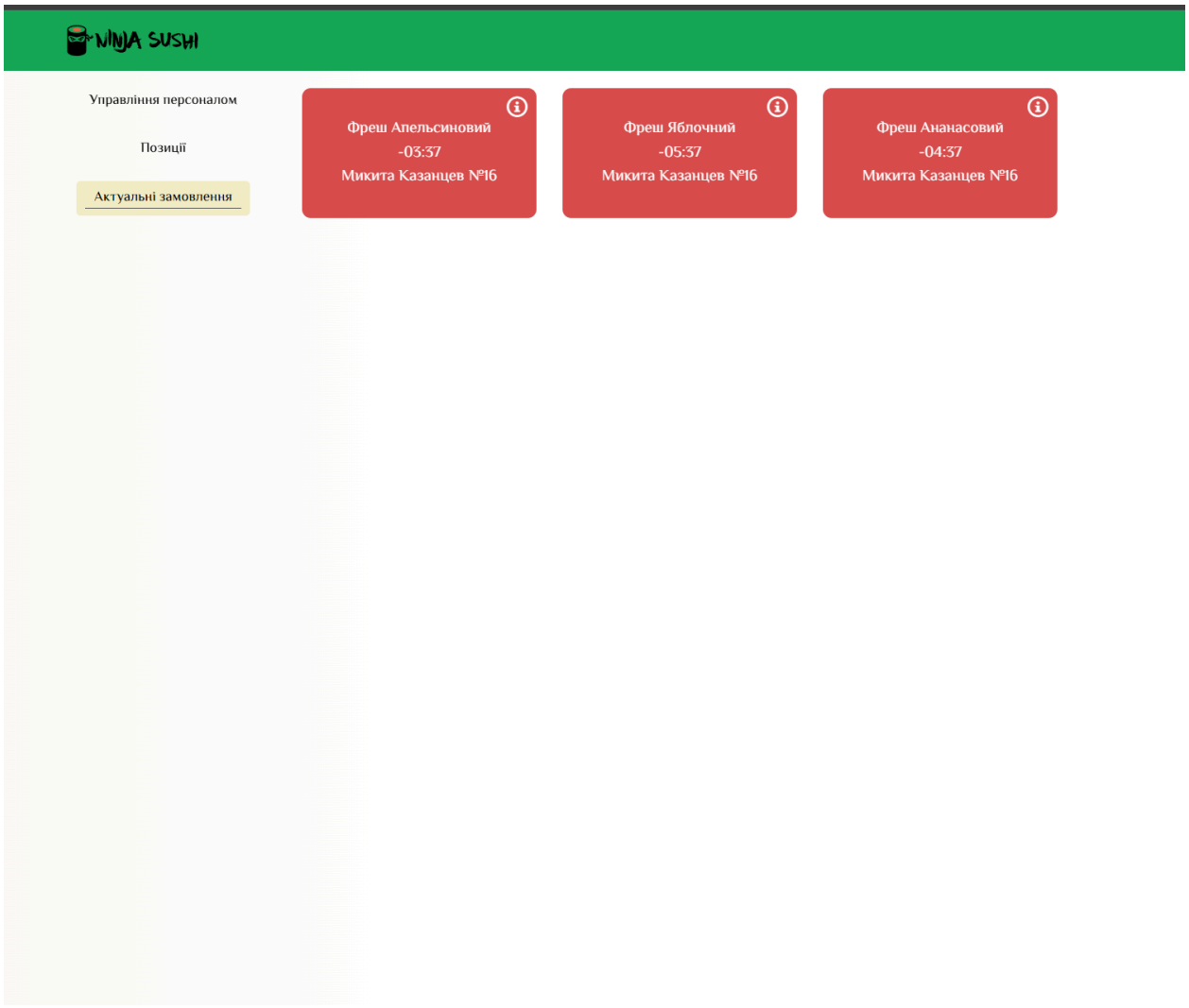

Рисунок В.15. Перегляд актуальних замовлень в інтерфейсі адміністратора