Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Прогнозування попиту в системах доставки харчування за допомогою алгоритмів машинного навчання
(тема)

Виконав:

здобувач 2 року навчання,

групи *ІТПм-24-1*

Комзолов М.О.
(прізвище, ініціали)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва спеціальності)

Тип програми *освітньо-професійна*

Освітня програма *Інформаційні технології проектування*
(повна назва освітньої програми)

Керівник *проф. Міщеряков Ю.В.*
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ *проф. Гребеннік І.В*
(підпис) (прізвище, ініціали)

2025 р.

Я як студент ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

17.12.2025 Комзолов М.О.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 17 грудня 2025 р.

Керівник кваліфікаційної роботи проф. Міщеряков Ю.В.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
Кафедра *Системотехніки*
Рівень вищої освіти *другий (магістерський)*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-професійна*
Освітня програма *Інформаційні технології проектування*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри СТ
проф. Гребеннік І.В
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Комзолову Миколі Олексійович
(прізвище, ім'я, по батькові)

1. Тема роботи Прогнозування попиту в системах доставки харчування за допомогою алгоритмів машинного навчання затверджена наказом університету від __ 25 р. № Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 грудня 2025 р.
3. Вихідні дані до роботи *Інформація про функціонування системи доставки індивідуальних раціонів харчування, зокрема дані історичних замовлень, страв, раціонів, користувачів, статусів доставок, а також зовнішні фактори, що впливають на попит, включаючи календарні характеристики, погодні умови, цінові зміни та промоакції. На основі цих даних необхідно розробити та реалізувати інтелектуальний модуль прогнозування споживчого попиту на страви. Серверна частина системи реалізується у вигляді API для взаємодії з базою даних та модулями аналітики з використанням мови програмування Python і системи керування базами даних Microsoft SQL Server. Для розробки та тестування використовуються програмні засоби: операційна система Microsoft Windows 10/11, середовище розробки Visual Studio Code, Python (з бібліотеками pandas, scikit-learn, joblib) та СУБД MSSQL Server. Технічне забезпечення — IBM-сумісний персональний комп'ютер..*
4. Перелік питань, що потрібно опрацювати в роботі *4.1 Аналіз предметної області, 4.1.1 Загальна характеристика предметної області, 4.1.2 Аналіз існуючих систем і технологій, 4.1.3 Актуальність впровадження ШІ в системи індивідуальних раціонів, 4.1.4 Постановка задачі, 4.2 Розробка моделі розроблювальної системи, 4.2.1 Розробка системних вимог до системи, 4.2.2 Розробка функціональних вимог до системи, 4.2.3 Визначення особливостей структури даних для прогнозування попиту, 4.2.4 Розробка оновленої діаграми класів системи, 4.2.5 Розробка основного алгоритму передбачення попиту на страви, 4.3 Реалізація надбудови штучного інтелекту прогнозування попиту страв, 4.3.1 Формування набору історичних даних замовлень, 4.3.2 Формування набору зовнішніх даних, 4.3.3 Формування тренувальної та тестової вибірки, 4.3.4 Навчання моделі прогнозування попиту, 4.4 Тестування надбудови штучного інтелекту, 4.4.1 Отримання майбутніх факторів та ознак прогнозу, 4.4.2 Передбачення майбутнього попиту, Висновки*
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Аналіз підходів формування списку персоналізованих рекомендацій	*Проф. Міщеряков Ю.В.*		*19.11.2025*
Розробка функціональних вимог до системи	*Проф. Міщеряков Ю.В.*		*29.11.2025*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	*Отримання завдання кваліфікаційної роботи*		*Вик.*
2.	*Аналіз предметної області та постановка задачі*		*Вик.*
3.	*Ознайомлення з літературою*		*Вик.*
4.	*Розробка основного алгоритму передбачення попиту на страви*		*Вик.*
5.	*Реалізація надбудови штучного інтелекту прогнозування попиту страв*		*Вик.*
6.	*Розробка функціональних вимог до системи*		*Вик.*
7.	*Оформлення пояснювальної записки*		*Вик.*
8.	*Представлення роботи на рецензування*		*Вик.*
9.	*Підготовка презентації до захисту*		*Вик.*
10.	*Захист роботи*		*Вик.*

Дата видачі завдання *2025 р.*

Здобувач _____ Комзолов М.О
(підпис) (прізвище, ініціали)

Керівник роботи _____ проф. Міщеряков Ю.В
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Робота містить: 105 сторінок, 23 рисунки, 10 лістингів.

АНАЛІЗ ДАНИХ, МАШИННЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ ПОПИТУ, СИСТЕМА ДОСТАВКИ РАЦІОНІВ, ШТУЧНИЙ ІНТЕЛЕКТ, RANDOM FOREST

Об’єктом дослідження є процес функціонування інформаційної системи доставки індивідуальних раціонів харчування в умовах змінного та нерівномірного споживчого попиту. Предметом дослідження є методи, моделі та алгоритми прогнозування попиту на страви із використанням технологій машинного навчання та їх інтеграція в існуючу інформаційну систему доставки раціонів.

Метою роботи є розробка та реалізація надбудови штучного інтелекту для прогнозування майбутнього попиту на страви з урахуванням історичних даних замовлень, календарних факторів, погодних умов, цінових змін і промоактивностей, а також оцінка ефективності впровадженого підходу в межах модернізованої системи доставки індивідуального харчування.

У процесі виконання роботи використовувалися методи аналізу предметної області та існуючих інформаційних систем, методи структурного та функціонального моделювання, проєктування баз даних, методи підготовки та обробки даних, а також методи машинного навчання. Для побудови прогнозної моделі застосовано ансамблеві алгоритми регресії, зокрема Random Forest, із використанням метрик оцінювання якості прогнозування MAE та RMSE. Реалізація виконувалася з використанням мови програмування Python, СУБД Microsoft SQL Server та допоміжних бібліотек для аналізу даних.

У результаті роботи було модернізовано функціональні вимоги та структуру бази даних системи доставки раціонів, розроблено алгоритм прогнозування попиту на страви, реалізовано модуль машинного навчання та виконано його тестування на реальних і згенерованих даних.

ABSTRACT

Content: 105 pages, 23 figures, 10 listings.

DATA ANALYSIS, MACHINE LEARNING, DEMAND FORECASTING, MEAL DELIVERY SYSTEM, ARTIFICIAL INTELLIGENCE, RANDOM FOREST

The object of the research is the process of functioning of an information system for the delivery of individual meal plans under conditions of variable and uneven consumer demand.

The subject of the research is the methods, models, and algorithms for forecasting demand for dishes using machine learning technologies and their integration into an existing meal delivery information system.

The purpose of the work is to develop and implement an artificial intelligence add-on for forecasting future demand for dishes, taking into account historical order data, calendar factors, weather conditions, price changes, and promotional activities, as well as to evaluate the effectiveness of the implemented approach within a modernized individual meal delivery system.

In the course of the work, methods of domain analysis and analysis of existing information systems were used, along with structural and functional modeling methods, database design techniques, data preparation and processing methods, and machine learning methods. To build the forecasting model, ensemble regression algorithms were applied, in particular Random Forest, using MAE and RMSE metrics to evaluate forecasting quality. The implementation was carried out using the Python programming language, Microsoft SQL Server DBMS, and auxiliary data analysis libraries.

As a result of the work, the functional requirements and database structure of the meal delivery system were modernized, a demand forecasting algorithm for dishes was developed, a machine learning module was implemented, and its testing was performed on real and generated data.

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10
1.1 Загальна характеристика предметної області 10
1.2 Аналіз існуючих систем і технологій 12
1.3 Актуальність впровадження ШІ в системи індивідуальних раціонів 16
1.4 Постановка задачі 18
2 РОЗРОБКА ВИМОГ ДО РОЗРОБЛЮВАЛЬНОЇ СИСТЕМИ 21
2.1 Розробка системних вимог до системи 21
2.2 Розробка функціональних вимог до системи 23
2.3 Визначення особливостей структури даних для прогнозування попиту 41
2.4 Розробка оновленої діаграми класів системи 47
2.5 Розробка основного алгоритму передбачення попиту на страви 52
3 РЕАЛІЗАЦІЯ НАДБУДОВИ ШТУЧНОГО ІНТЕЛЕКТУ ПРОГОЗУВАННЯ ПОПИТУ СТРАВ 58
3.1 Формування набору історичних даних замовлень 58
3.2 Формування набору зовнішних даних 63
3.3 Формування тренувлаьної та тестової вибірки 71
3.4 Навчання моделі прогнозування попиту 83
4 ТЕСТУВАННЯ НАДБУДОВИ ШТУЧНОГО ІНТЕЛЕКТУ 92
4.1 Отримання майбутніх факторів та ознак прогнозу 92
4.2 Передбачення майбутнього попиту 97
ВИСНОВКИ 101
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 103

ВСТУП

Сучасні системи доставки індивідуальних раціонів харчування є важливою складовою ринку онлайн-сервісів, що динамічно розвивається в умовах цифрової трансформації. Зростання попиту на персоналізовані послуги, формування здорових харчових звичок та оптимізація часу користувачів призводять до активного впровадження інформаційних технологій у сферу управління замовленнями, планування виробництва та організації логістики. У таких умовах ключовим фактором ефективності роботи сервісів стає здатність точно прогнозувати майбутній попит на конкретні страви або раціони. Це дає можливість раціонально планувати закупівлі інгредієнтів, оптимізувати навантаження на кухню та мінімізувати харчові втрати.

Попит у сфері доставки харчування формується під впливом численних факторів: індивідуальних вподобань клієнтів, сезонності, буденних і святкових циклів, маркетингових активностей, зміни цін, а також зовнішніх умов, включаючи погодні коливання. Традиційні підходи до планування ґрунтуються на досвіді адміністраторів та простих статистичних методах, що не здатні врахувати складну нелінійну взаємодію між цими змінними. У результаті системи стикаються з нерівномірним навантаженням, перевитратами продуктів або їх дефіцитом, що негативно впливає як на економічні показники бізнесу, так і на якість обслуговування клієнтів.

Розвиток технологій штучного інтелекту та машинного навчання відкриває можливість комплексного аналізу історичних даних і побудови моделей, здатних прогнозувати попит з високою точністю. Інтелектуальні алгоритми дають змогу виявляти приховані закономірності, оцінювати вплив зовнішніх чинників та адаптивно враховувати зміни у поведінці користувачів. Завдяки цьому з’являється можливість побудови автоматизованих систем підтримки прийняття рішень, які в реальному часі можуть генерувати прогноз попиту на окремі страви та рекомендації щодо формування меню.

З практичної точки зору створення такого модуля є надзвичайно актуальним для систем доставки індивідуальних раціонів харчування. Інтеграція

алгоритмів машинного навчання дозволяє не тільки знизити витрати на закупівлю сировини та скоротити відходи, а й підвищити стабільність виробничих процесів, забезпечити клієнтів актуальними пропозиціями та покращити рівень персоналізації сервісу. У подальшому такі рішення можуть стати основою для побудови повноцінних інтелектуальних платформ управління харчуванням, здатних адаптуватися до трендів ринку та індивідуальних потреб кожного користувача.

Отже, актуальність роботи полягає у необхідності розроблення та впровадження інтелектуального модуля прогнозування попиту, який доповнить існуючу інформаційну систему доставки раціонів і забезпечить її перехід до нового рівня ефективності. Вирішення цього завдання потребує комплексного підходу, що включає аналіз предметної області, проектування структури даних, розроблення алгоритмів машинного навчання, тестування їх точності та інтеграцію в роботу сервісу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика предметної області

Система доставки індивідуальних раціонів харчування належить до класу інформаційних систем електронної комерції, орієнтованих на надання послуг у сфері здорового харчування. Її основною метою є автоматизація процесу підбору, формування, приготування та доставки збалансованих раціонів клієнтам відповідно до їхніх персональних параметрів – калорійності, харчових вподобань, дієтичних обмежень та цілей. Такі системи є прикладом комплексних рішень, які об’єднують функціональність замовлення продуктів харчування, персоналізовані рекомендації та логістичні модулі в єдиній цифровій екосистемі.

Загальна структура системи включає три основні складові: користувацький модуль, адміністративний модуль і логістичний модуль доставки. Кожен із них реалізує окремі бізнес-процеси, але взаємодіє через спільну базу даних і API. Користувацький модуль призначений для взаємодії клієнтів із системою, адміністративний – для управління асортиментом, меню, замовленнями та клієнтами, а логістичний – для організації процесу доставки та контролю маршрутизації кур’єрів.

Основним об’єктом системи з боку клієнта є користувач. Він має особистий профіль, який містить дані про вік, стать, вагу, рівень фізичної активності, алергени, мету (схуднення, підтримка форми або набір маси) та історію замовлень. На основі цих параметрів система формує індивідуальний раціон харчування, який складається зі страв різних типів – сніданок, обід, вечеря та перекуси. Користувач може переглядати рекомендовані набори або самостійно формувати комбінації страв, відстежуючи при цьому загальну калорійність і баланс БЖВ (білків, жирів, вуглеводів).

Окремим підоб’єктом є страва – одиниця харчового продукту або готового прийому їжі. Для кожної страви зберігаються характеристики: назва, опис, склад,

енергетична цінність, час приготування, вага порції, фотографія, категорія (веганська, білкова, низьковуглеводна тощо). Ці дані використовуються не лише для відображення в інтерфейсі користувача, а й для подальшого аналітичного аналізу, зокрема під час формування рекомендацій та прогнозування попиту.

Коли користувач оформлює замовлення, створюється об’єкт «Замовлення», який містить інформацію про дату доставки, склад набору, загальну вартість і статус виконання. Замовлення пов’язується з користувачем і раціоном, а після підтвердження оплати передається на обробку в адміністративну підсистему.

Адміністративний модуль призначений для персоналу компанії – менеджерів, кухарів, дієтологів та адміністраторів. Основним об’єктом тут є меню, що складається з наборів страв, згрупованих за типами (калорійність, дієта, тривалість програми тощо). Адміністратор може додавати нові страви, редагувати їх склад, змінювати ціни, визначати доступність позицій у певні дні тижня.

Дієтолог може створювати програми харчування, які складаються з кількох днів меню та орієнтовані на конкретні цілі користувачів. Кожна програма має власну калорійність, розподіл макронутрієнтів і рекомендації щодо фізичної активності. У подальшому система може використовувати ці програми як шаблони для автоматичного формування персоналізованих раціонів.

Для контролю виконання замовлень адміністратор має доступ до журналу активних заявок, звідки може відстежувати статус приготування, упаковки та доставки. Кожен етап має свій статус – наприклад: «прийнято», «готується», «в дорозі», «доставлено», що відображається в інтерфейсі користувача в реальному часі.

Окрему роль у системі відіграє підсистема доставки, яка координує роботу кур’єрів. Її основним об’єктом є доставка, що має такі атрибути: ідентифікатор замовлення, кур’єр, адреса, час прибуття, геолокаційні координати та статус. Система може визначати оптимальні маршрути доставки за допомогою інтеграції з картографічними сервісами (наприклад, Google Maps API) або внутрішнього алгоритму оптимізації.

Кур’єр має власний обліковий запис, через який отримує список замовлень і позначає статус виконання доставки. Це дозволяє системі в реальному часі відстежувати прогрес кожного замовлення й оцінювати ефективність роботи персоналу.

Робота системи відбувається у кілька етапів. Спочатку користувач створює обліковий запис і вказує свої фізіологічні параметри. На основі цих даних система формує персональний раціон або пропонує обрати готову дієту. Після вибору страв користувач оформлює замовлення, вказує адресу доставки та час отримання. Замовлення передається на обробку до адміністративного модуля, де кухня готує страви згідно з планом. Після приготування замовлення передається кур’єру, який здійснює доставку.

Паралельно аналітичний модуль зберігає інформацію про всі транзакції. На основі цих даних формується тренд попиту на страви, який може бути використаний для прогнозування. У майбутньому модель машинного навчання дозволить прогнозувати попит на рівні окремих позицій меню, що оптимізує виробничі процеси.

У сучасних соціально-економічних умовах подібні системи мають особливе значення. Вони поєднують елементи здорового способу життя, цифровізації сервісів і зручності для користувача. Для компаній, що працюють у сфері харчування, застосування аналітичних і прогнозних механізмів дає змогу мінімізувати витрати, збільшити оборотність продуктів і зменшити ризики псування товарів. Для споживачів така система є способом підтримання здорового раціону без необхідності самостійного планування.

1.2 Аналіз існуючих систем і технологій

Серед систем, які можна розглядати як аналоги, доцільно виділити платформи типу HelloFresh, Glovo, та локальні сервіси здорового харчування, наприклад FoodEx (Київ). HelloFresh – це сервіс доставки наборів для

приготування їжі (meal kits), де користувач отримує інгредієнти та інструкцію; цей підхід ближчий до кастомізації і планування меню, ніж просто доставка готових страв. Glovo – популярна платформа з доставки їжі і товарів у багатьох містах України, яка не спеціалізується саме на дієтичних раціонах, але має зрілу інфраструктуру доставки. FoodEx – український сервіс здорової їжі з доставкою, який здійснює доставку щодня в ранкові години в межах Києва.

Також варто звернути увагу на глобальні сервіси здорового харчування, які мають елементи персоналізації меню, наприклад HelloFresh (збір інгредієнтів, варіанти меню) – цей сервіс має велику клієнтську базу та систему логістики. У випадку HelloFresh можна використати як шаблон для моделювання бізнес-процесів і технічної архітектури.

З цих варіантів головним аналогом обрано HelloFresh як систему з відпрацьованими бізнес-моделлю, масштабом і технологічними рішеннями. У межах роботи можна деталізувати, як HelloFresh управляє замовленнями, меню, логістикою, як оновлює меню, як працює з користувачем, а також які проблеми та обмеження помітні в їхньому підході.

Головна сторінка меню HelloFresh зображена на рисунку 1.1.

У верхній частині сторінки розташовано навігаційну панель із розділами «Our Plans», «About Us», «Our Menus», «Gift Cards», «Sustainability» та «Partnerships», що забезпечує швидкий доступ до основних сервісів компанії. Центральна частина інтерфейсу відведена для відображення меню тижня, з можливістю вибору конкретного періоду за допомогою інтерактивної панелі дат. Кожна страва представлена у вигляді картки з фото, назвою, коротким описом, часом приготування, позначками категорій (наприклад, «High Protein», «Easy Cleanup», «Spicy») і додатковими характеристиками. Такий підхід дозволяє користувачу швидко оцінити склад, калорійність і складність приготування страви, не переходячи на окрему сторінку.

Інтерфейс поєднує естетику та функціональність, створюючи відчуття персоналізованого досвіду вибору меню. Використання великих ілюстрацій страв із чіткими підписами та маркерами типу кухні (наприклад, «Global Street Eats», «Nutritious Picks») підсилює візуальну привабливість і полегшує

навігацію. Завдяки структурованому розташуванню елементів користувач може швидко орієнтуватися між тижневими меню, переглядати пропозиції та формувати замовлення. Така організація сторінки свідчить про глибоке опрацювання UX-дизайну, який відповідає концепції зручності, адаптивності й акценту на здоровому харчуванні.

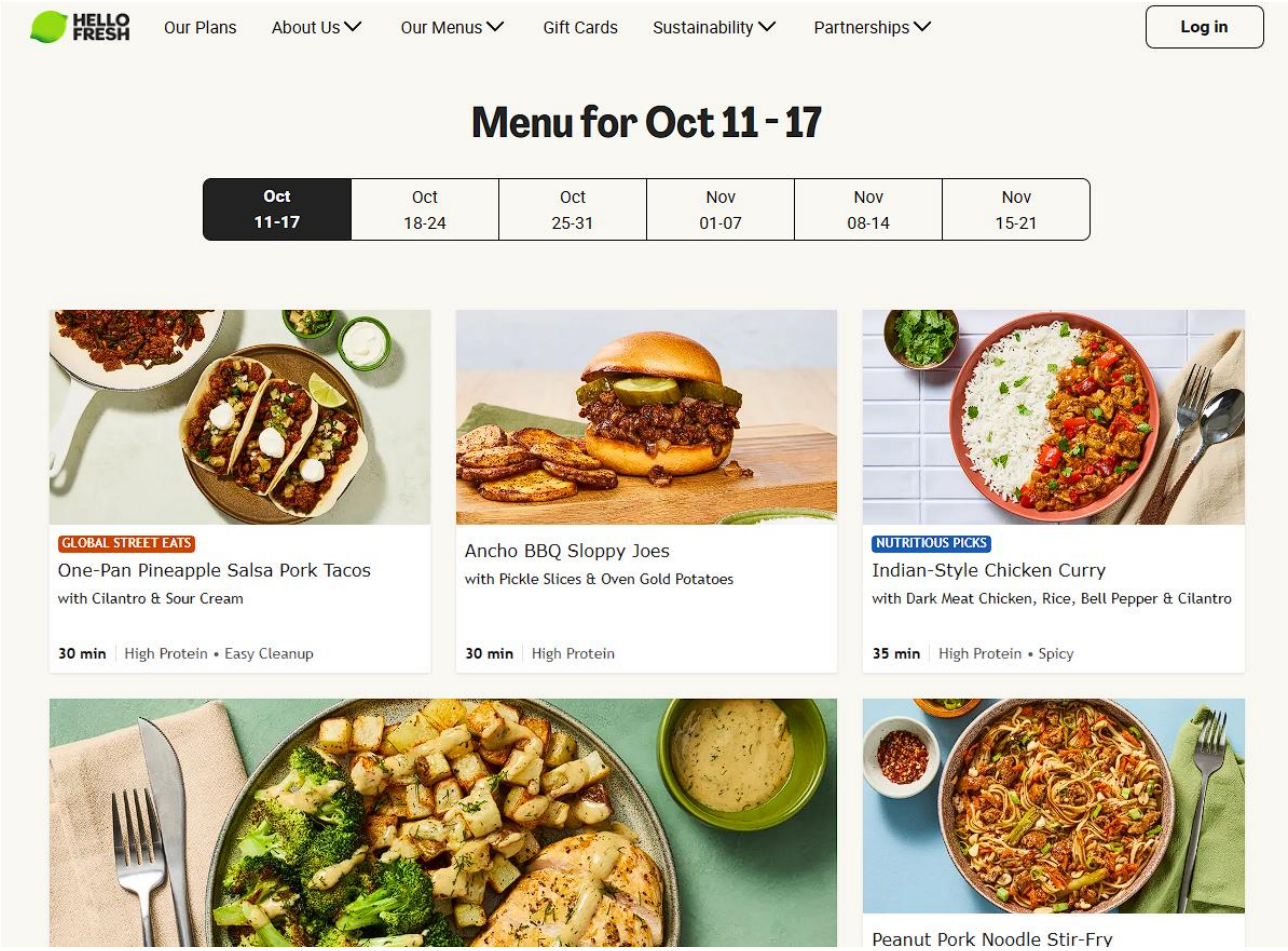

Рисунок 1.1 – головна сторінка меню системи

При використанні HelloFresh, користувач при реєстрації вказує параметри (кількість людей, скільки страв на тиждень, дієтичні обмеження), система пропонує меню з кількома варіантами, користувач вибирає конкретні набори, потім замовлення формуються централізовано та надсилаються до кухонь/центрових складів, звідти розвозяться інгредієнти або готові компоненти. Меню оновлюється щотижня, враховуючи популярність, сезонність, логістику поставок.

Переваги HelloFresh: висока якість і стандартизація, добрий UX/UI, гнучкі

варіанти меню, автоматизація логістики та складських процесів, сильна клієнтська підтримка, можливість масштабування в нові регіони, досвід роботи з харчовими ланцюжками.

Недоліки: висока вартість доставки, обмеження для клієнтів із специфічними харчовими потребами (якщо меню не враховує всі алергени чи хвороби), залежність від точності прогнозів замовлень (надлишки чи нестачі інгредієнтів), відсутність глибокої персоналізації меню на рівні штучного інтелекту (оскільки меню формується людьми).

Попри очевидну зручність і продуманість роботи системи HelloFresh, з боку її функціональності можна виділити низку суттєвих обмежень. По-перше, система не реалізує глибокої персоналізації на основі даних про стан здоров'я користувача чи його індивідуальні фізіологічні параметри. Користувач може вказати лише загальні вподобання (вегетаріанські страви, низьковуглеводне меню тощо), однак система не враховує показники калорійності, БЖВ-балансу або наявність алергенів у комплексний спосіб. Це означає, що платформа більше орієнтована на зручність вибору страв, ніж на створення повноцінно індивідуального раціону харчування.

По-друге, у системі відсутні аналітичні механізми прогнозування споживчого попиту чи адаптивного формування меню на основі статистики замовлень. Меню оновлюється централізовано, без автоматичного врахування регіональних або сезонних тенденцій, що знижує ефективність планування. Крім того, HelloFresh не надає користувачу динамічного зворотного зв'язку у вигляді рекомендацій або прогнозів щодо найбільш підходящих страв відповідно до історії його виборів. Це обмежує потенціал системи як інтелектуального сервісу, що міг би навчатися на поведінці клієнтів і вдосконалювати процес формування меню.

Узагальнюючи результати аналізу розглянутих систем доставки їжі, можна зробити висновок, що навіть найбільш розвинені платформи, такі як HelloFresh чи Glovo, орієнтовані насамперед на логістику, зручність користувацької взаємодії та якість сервісу, але не мають повноцінних інтелектуальних модулів прогнозування попиту. У більшості випадків оновлення меню, планування

кількості інгредієнтів і формування замовлень здійснюється вручну або з використанням базових статистичних моделей. Це створює інформаційну прогалину між наявними операційними процесами та можливостями сучасних технологій машинного навчання. Саме впровадження штучного інтелекту для прогнозування споживчого попиту дозволить автоматизувати прийняття рішень, підвищити точність планування виробництва, скоротити харчові втрати та забезпечити більш персоналізований підхід до формування раціонів. Таким чином, розробка інтелектуального модуля прогнозування є не лише логічним етапом розвитку подібних систем, а й ключовим чинником підвищення їх ефективності та конкурентоспроможності.

1.3 Актуальність впровадження ШІ в системи індивідуальних раціонів

У сучасних умовах розвитку ринку доставки їжі питання прогнозування споживчого попиту набуває стратегічного значення. Кількість користувачів, які замовляють індивідуальні раціони харчування, зростає, а разом з нею збільшується і складність управління виробництвом, закупівлями та логістикою. Традиційні підходи до планування базуються на статистичних даних минулих періодів і досвіді менеджерів, однак вони не враховують швидких змін у поведінці споживачів, сезонності, зовнішніх факторів або маркетингових впливів. У таких умовах застосування штучного інтелекту для прогнозування попиту є не просто доцільним, а необхідним для забезпечення ефективної, адаптивної та економічно стабільної роботи систем доставки індивідуального харчування.

Алгоритми машинного навчання дозволяють системі аналізувати великі обсяги історичних даних про замовлення, виявляти приховані закономірності та формувати точні прогнози щодо популярності окремих страв. Використання таких алгоритмів забезпечує можливість передбачати кількість замовлень для кожної позиції меню у певний період часу. Це особливо важливо для сервісів

індивідуальних раціонів, де меню часто оновлюється, а кількість доступних порцій є обмеженою. Прогнозування на основі штучного інтелекту допомагає уникнути надлишків продуктів, зменшити харчові втрати та забезпечити стабільну якість обслуговування клієнтів навіть за умов коливань попиту.

Особливу актуальність впровадження таких технологій визначає мінливість попиту у цьому сегменті. Популярність певних страв може змінюватися під впливом дня тижня, погодних умов, святкових періодів, а також маркетингових акцій. Людський фактор або прості статистичні підходи часто не здатні адекватно врахувати ці взаємозалежності. Натомість інтелектуальні алгоритми здатні автоматично враховувати багатофакторні зв'язки між даними, навчатися на поточних тенденціях і швидко адаптуватися до змін поведінки користувачів, що забезпечує вищу точність прогнозів.

Впровадження машинного навчання у процес прогнозування попиту також дозволяє суттєво підвищити ефективність планування виробничих процесів. Кухні можуть заздалегідь отримувати інформацію про очікувані обсяги приготування кожної страви, склади – про необхідні обсяги інгредієнтів, а служба доставки – про пікові навантаження у певні дні чи години. Це дає змогу оптимізувати розподіл персоналу, скоротити витрати на закупівлі та логістику, а також зменшити ризик зривів постачання. Таким чином, штучний інтелект перетворюється на ключовий інструмент управління операційною ефективністю компанії.

Ще однією причиною актуальності застосування ШІ є потреба у підвищенні якості аналітики та прогнозного управління. Класичні системи бізнес-аналітики дозволяють бачити лише минулу картину продажів, тоді як моделі машинного навчання орієнтовані на майбутнє. Вони здатні виявляти тенденції, які ще не проявилися явно, і попереджати управлінців про можливі зміни попиту. Це відкриває нові можливості для проактивного управління меню, ціноутворенням і маркетинговими кампаніями, роблячи бізнес більш передбачуваним і стійким до зовнішніх коливань.

Крім того, застосування алгоритмів машинного навчання підвищує рівень автоматизації в системі. У результаті компанія може мінімізувати залежність від

суб’єктивних рішень персоналу та перейти до керування на основі даних. Система, що самостійно формує прогноз попиту, стає не лише аналітичним інструментом, а й складовою частиною інтелектуального управління бізнес-процесами. Це дає можливість масштабувати сервіс без значного збільшення людських ресурсів та оперативно реагувати на зміни у ринкових умовах.

Таким чином, актуальність впровадження штучного інтелекту в системи доставки індивідуальних раціонів харчування полягає у підвищенні точності прогнозування, зменшенні витрат, оптимізації логістики та забезпеченні стабільної роботи навіть за умов мінливого попиту. Машинне навчання дозволяє перетворити дані замовлень на практичний інструмент прогнозування, що формує основу для ефективного управління всією екосистемою доставки – від планування закупівель до підвищення задоволеності клієнтів. Саме тому розробка інтелектуального модуля прогнозування попиту є важливим кроком до створення нової генерації адаптивних, самонавчальних систем у сфері персоналізованого харчування

1.4 Постановка задачі

Метою цієї роботи є розроблення та впровадження інтелектуального модуля прогнозування споживчого попиту на страви в системі доставки індивідуальних раціонів харчування. Для досягнення цієї мети необхідно виконати повний цикл дослідження – від аналізу предметної області до побудови, навчання й тестування моделі машинного навчання, а також інтеграції її в існуючу інформаційну систему. Основне завдання полягає в тому, щоб створити ефективний інструмент, який здатний прогнозувати кількість замовлень на конкретні страви у визначені часові проміжки, враховуючи сезонні, поведінкові та зовнішні фактори.

У межах кваліфікаційної роботи формується комплексне завдання щодо модернізації існуючої інформаційної системи доставки раціонів, яка була

створена в попередній проєктній роботі[1], шляхом інтеграції інтелектуального модуля прогнозування споживчого попиту. Базова система вже містить механізми формування раціонів, обліку замовлень, планування доставки та взаємодії з користувачами, однак не має інструментів, які дозволяють завчасно оцінити попит на страви й оптимізувати операційну діяльність. Це створює об'єктивну необхідність її розширення та інтелектуального вдосконалення.

Перша частина задачі полягає у детальному аналізі предметної області, існуючих бізнес-процесів і наявної архітектури системи, що включає вивчення діаграм функціональних вимог, структури бази даних, логіки роботи основних модулів та їхньої взаємодії. На основі цього аналізу необхідно сформувати оновлену модель системи, доповнену компонентами збору даних для прогнозування, аналітичними сервісами та модулем машинного навчання. Особливу увагу слід приділити тому, щоб модернізація гармонійно інтегрувалася в існуючу архітектуру без порушення принципів масштабованості, доступності та цілісності даних.

Другим ключовим аспектом задачі є формальний опис структури даних, потрібних для побудови прогнозної моделі. Система повинна акумулювати та коректно структурувати такі дані, як історія замовлень, детальні характеристики страв і раціонів, інформація про користувачів, календарні й погодні фактори, промоактивності, а також оперативні параметри доставки. Необхідно визначити релевантні ознаки, які мають найбільший вплив на формування попиту, та вибудувати повноцінний ознаковий простір, придатний для використання алгоритмами машинного навчання.

Наступним етапом є розробка алгоритмічного забезпечення для прогнозування попиту, включаючи підготовку даних, побудову датасету, вибір моделі та налаштування гіперпараметрів. У рамках дипломної роботи необхідно реалізувати процес навчання моделей, наприклад Random Forest або XGBoost, а також забезпечити механізми оцінювання якості прогнозу за метриками MAE, RMSE, MAPE та іншими. Важливо, щоб алгоритм підтримував періодичне перенавчання для адаптації до змін ринку та поведінки користувачів.

Ще одним важливим компонентом є інтеграція прогнозного модуля в

інформаційну систему, що передбачає автоматичний обмін даними з базою, формування прогнозів на визначений період, збереження результатів і передачу їх до модуля рекомендацій меню. Модуль повинен працювати в оперативному режимі, відповідати системним вимогам щодо продуктивності та надавати інформацію в зручному для використання форматі.

Фінальним етапом задачі є тестування розробленого інтелектуального модуля, включаючи функціональні, інтеграційні та якісні випробування моделей. Потрібно перевірити стабільність алгоритмів, коректність формування прогнозів, надійність передачі результатів у адміністративні та експлуатаційні інтерфейси, а також визначити показники ефективності впровадження (скорочення витрат, зменшення залишків продуктів, підвищення точності планування меню).

Таким чином, поставлена задача охоплює повний цикл розробки – від аналізу й модернізації системи до побудови й інтеграції модуля штучного інтелекту, здатного підвищити ефективність сервісу доставки індивідуального харчування. Впровадження прогнозного модуля має забезпечити покращення операційних процесів, оптимізацію ресурсів, зменшення ризиків надлишків і дефіцитів, а також підвищення рівня обслуговування та конкурентоспроможності сервісу загалом.

2 РОЗРОБКА ВИМОГ ДО РОЗРОБЛЮВАЛЬНОЇ СИСТЕМИ

2.1 Розробка системних вимог до системи

Для забезпечення ефективної роботи системи доставки індивідуальних раціонів харчування з впровадженим модулем штучного інтелекту необхідно розширити системні вимоги з урахуванням додаткових обчислювальних навантажень, аналітичних процесів та інтеграції компонентів машинного навчання. Метою таких вимог є створення стабільного, масштабованого та безпечного середовища, здатного забезпечити не лише обробку замовлень, але й прогнозування попиту на страви в реальному часі.

База даних системи має підтримувати не лише класичні транзакційні операції, але й аналітичну обробку великих обсягів історичних даних. Доцільно використовувати реляційну базу даних Microsoft SQL Server, доповнену механізмами оптимізації запитів та підтримкою історичних таблиць. Для підсистеми прогнозування може бути реалізований окремий сховище даних або аналітичний шар (data warehouse), де накопичуються агреговані записи попиту, погодних умов, цінових змін та календарних факторів. Це забезпечить швидкий доступ до аналітичних вибірок і стабільну роботу моделей штучного інтелекту.

Серверна частина системи повинна бути розроблена на платформі ASP.NET Core, яка забезпечує високу продуктивність, модульність і підтримку асинхронних запитів. Вона виконує роль проміжної ланки між клієнтським інтерфейсом, базою даних і аналітичним модулем. Для реалізації взаємодії з алгоритмами машинного навчання передбачається створення окремого мікросервісу, який може бути реалізований на Python із використанням бібліотек Scikit-learn або TensorFlow. Такий підхід забезпечує незалежність аналітичного модуля від основного веб-додатку та дозволяє масштабувати систему за потреби.

Користувацький інтерфейс має бути реалізований із використанням сучасного фреймворку Angular, що дозволяє створювати динамічні та адаптивні

веб-додатки. Інтерфейс повинен підтримувати інтерактивну візуалізацію прогнозів попиту у вигляді графіків і таблиць, що дозволить адміністраторам та аналітикам швидко аналізувати тенденції. Також необхідно забезпечити кросплатформенність, коректну роботу на мобільних пристроях і підтримку адаптивного дизайну.

Апаратні засоби повинні відповідати підвищеним вимогам до обчислювальних потужностей. Сервер має мати достатній обсяг оперативної пам’яті (не менше 16 ГБ) та багатоядерний процесор, що дозволить одночасно обробляти запити користувачів і виконувати аналітичні розрахунки. Для зберігання історичних даних і моделей машинного навчання доцільно використовувати швидкі SSD-накопичувачі або мережеве сховище з резервуванням. Додатково рекомендується передбачити резервний сервер або хмарне рішення для розподілу навантаження.

Інтеграція модуля штучного інтелекту вимагає також належної організації середовища виконання. Необхідно забезпечити можливість регулярного оновлення моделей прогнозування, автоматичне завантаження нових даних та моніторинг їх точності. Для цього система має підтримувати автоматизовані завдання (наприклад, через планувальник Windows або Cron), які періодично ініціюють процеси навчання та оновлення прогнозів.

Додатковим аспектом є безпека та надійність системи. Всі запити до аналітичного модуля повинні проходити через авторизований API з використанням токенів доступу. База даних має бути захищена від несанкціонованого доступу, а передача даних – здійснюватися виключно через захищені канали (HTTPS). Для зменшення ризику втрати даних необхідно реалізувати систему резервного копіювання, що охоплює як транзакційні таблиці, так і історичні записи, використані для навчання моделей.

Таким чином, оновлені системні вимоги орієнтовані не лише на стабільність роботи веб-застосунку, але й на забезпечення високої продуктивності під час обробки аналітичних запитів і прогнозування попиту. Інтеграція штучного інтелекту вимагає більш гнучкої архітектури, здатної підтримувати обмін даними між компонентами, їх масштабування та постійне

оновлення моделей, що гарантує адаптивність системи до змін ринку та поведінки користувачів.

2.2 Розробка функціональних вимог до системи

У попередній роботі, що стала основою для подальшого розвитку системи доставки індивідуальних раціонів, було виконано детальний аналіз функціональної структури процесів та побудовано діаграму функціональних вимог у нотації IDEF0. Для моделювання використовувався професійний програмний засіб BPWin, який традиційно застосовується для опису бізнес-процесів, функціональних потоків та інформаційних взаємодій у складних інформаційних системах. Застосування цієї методології дозволило формально описати логічну структуру системи, її основні функції, потоки даних, механізми та керуючі впливи, що становлять базу для подальшого проєктування.

Створена діаграма A-0 та її декомпозиції дозволили виокремити ключові підпроцеси, що забезпечують роботу базової версії системи: автентифікацію користувача, створення індивідуального раціону, оформлення регулярної доставки та зміну статусу замовлення. Кожен блок був формалізований відповідно до вимог BPWin, що дало змогу описати логічні межі функцій, джерела даних, зовнішні впливи та виконавців. Така модель служить фундаментом для подальшого вдосконалення системи та дає можливість коректно інтегрувати нові компоненти без порушення цілісності функціональної архітектури.

Використання BPWin забезпечило однозначність трактування бізнес-процесів та дозволило отримати структуровану графічну модель, придатну як для технічних спеціалістів, так і для управлінців, що приймають рішення щодо розвитку системи. Побудована функціональна модель чітко окреслила межі відповідальності кожного компоненту системи, стала зручним інструментом для виявлення вузьких місць і допомогла сформувати бачення того, які саме

підсистеми потребують модернізації.

Таким чином, діаграма, розроблена в BPWin, стала відправною точкою для проєкту впровадження штучного інтелекту в систему доставки раціонів. Вона визначила вихідну архітектуру, на основі якої можна було розширити функціональні можливості, додати нові інформаційні потоки та інтегрувати модуль прогнозування попиту. У поточній роботі ця модель була модернізована, збагачена новими функціями та адаптована до потреб інтелектуального аналізу даних.

Всі етапи розробленої функціональної моделі системи подано у вигляді послідовності діаграм IDEF0, які відображають як загальну структуру процесів, так і деталізацію ключових підпроцесів. Кожен рівень декомпозиції наведено на рисунку, що дозволяє простежити логіку побудови системи – від загального опису процесу до деталізованих операцій, пов’язаних зі створенням раціону, оформленням доставки, тощо. Такий підхід забезпечує повну прозорість функціональної архітектури та дає можливість зрозуміти взаємозв’язки між даними, виконавцями та інформаційними потоками.

Усі діаграми, розроблені в батьківській роботі, подані на рисунках 2.1–2.4. На них наведено оновлену модель верхнього рівня, декомпозицію основних функцій, тощо. Використання єдиної методології IDEF0 та послідовної структури подання забезпечує цілісність моделі та дозволить легко порівнювати нову функціональність із базовою версією системи.

У базовій роботі, що стала основою для поточного дослідження, була сформована повноцінна діаграма функціональних вимог, яка моделює основні бізнес-процеси системи доставки індивідуальних раціонів у початковому вигляді, без будь-яких елементів штучного інтелекту. На рисунку 2.1 представлено контекстну діаграму функціонування системи на рівні A-0, яка показує загальну взаємодію вхідних інформаційних потоків, зовнішніх об’єктів та виконавців. Вона відображає, що система отримує інформацію про користувача, доступні раціони, розклад доставки та нормативно-правові обмеження, після чого формує списки раціонів, розклад доставок і статуси виконання замовлень.

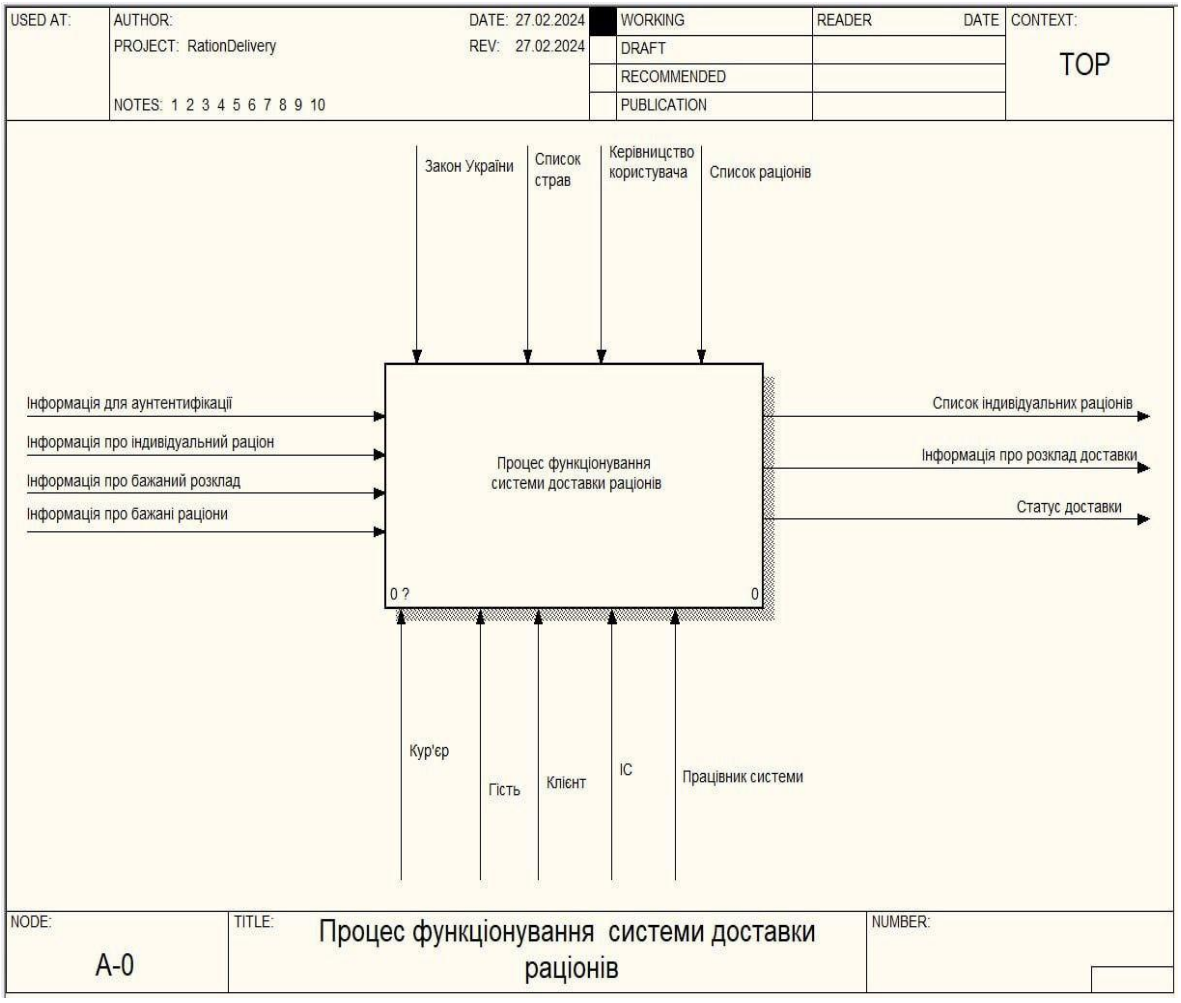

Рисунок 2.1 – Контекстна діаграма

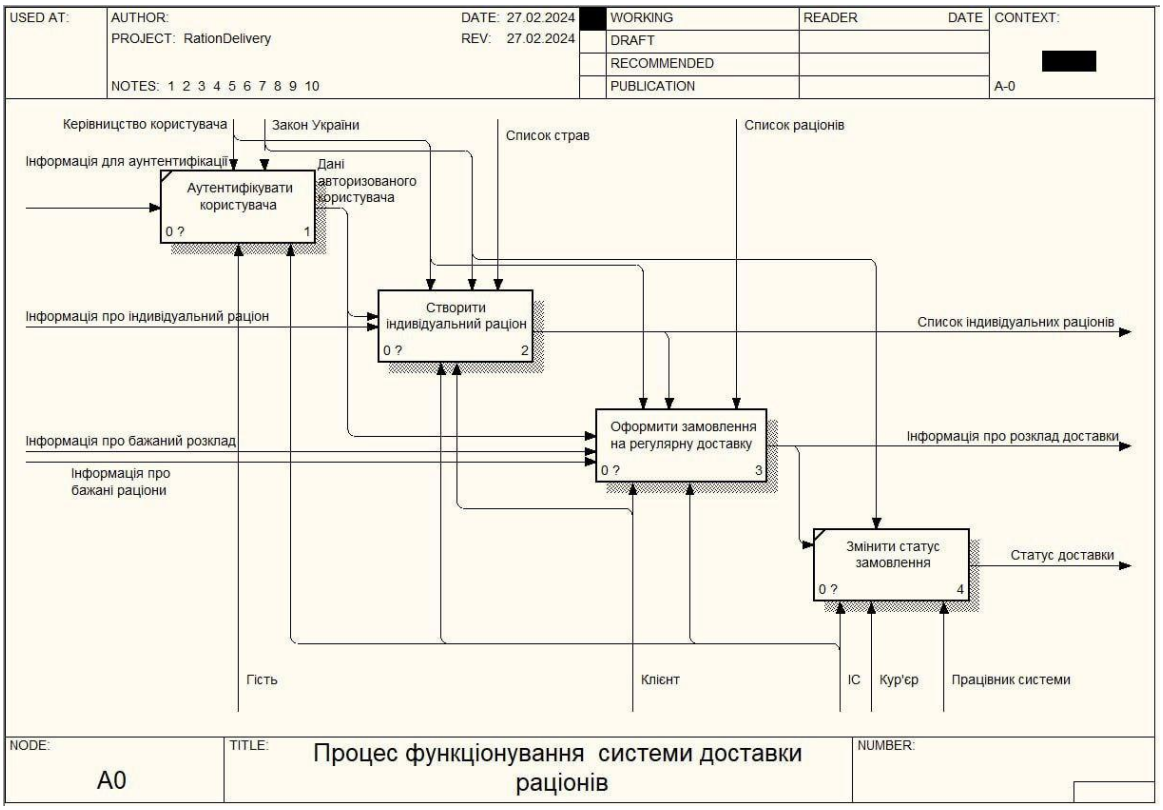

Рисунок 2.2 – Діаграма декомпозиції A0

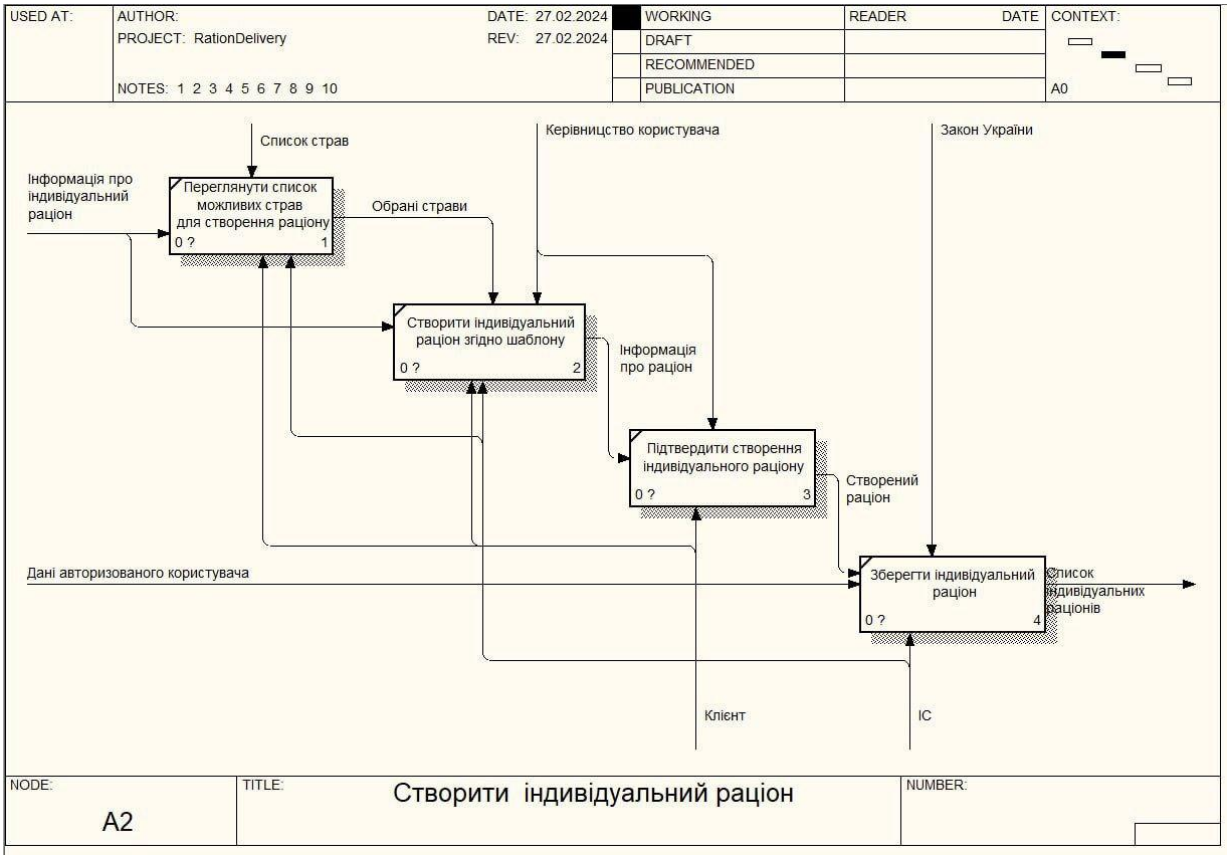

Рисунок 2.3 – Діаграма декомпозиції A2

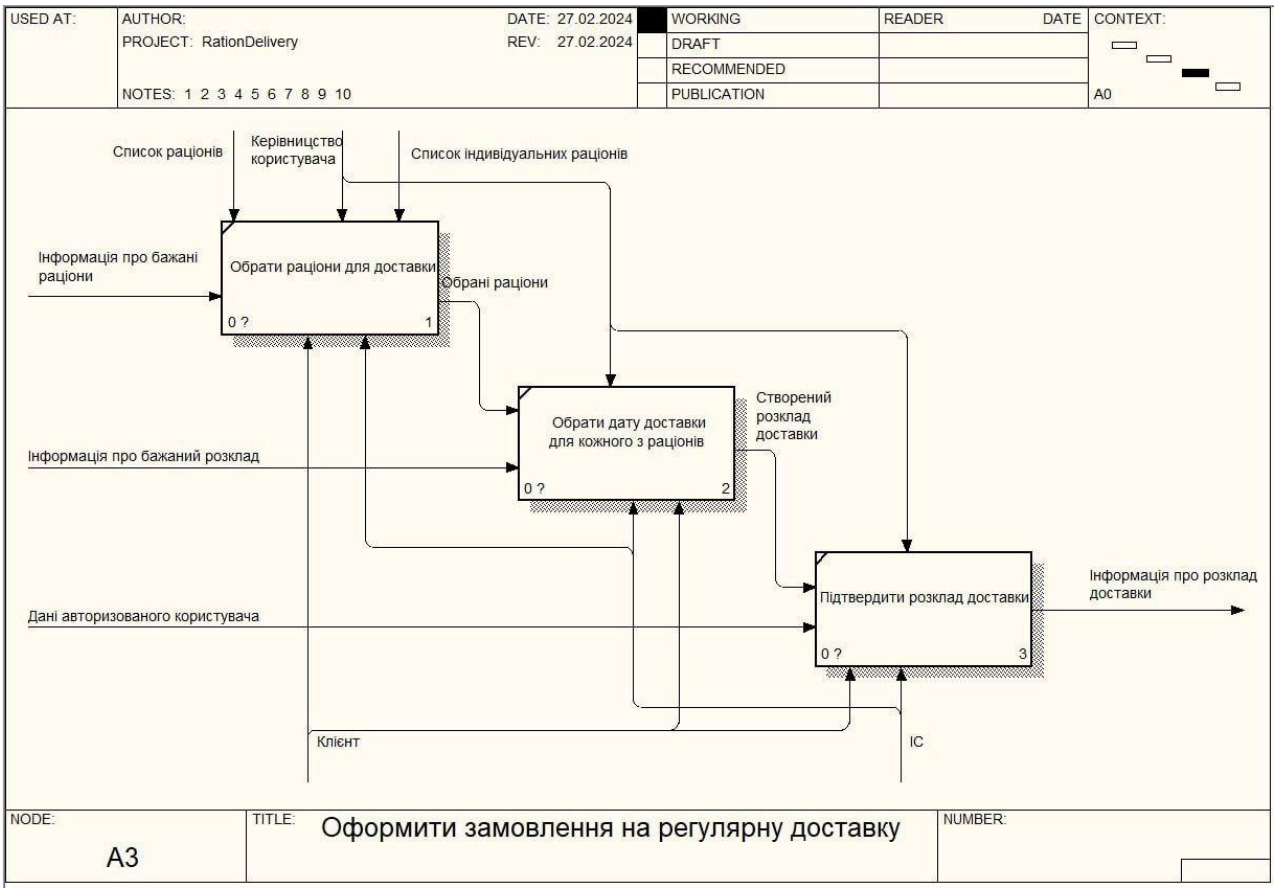

Рисунок 2.4 – Діаграма декомпозиції A3

На наступному рівні декомпозиції, зображеному на рисунку 2.2, детально представлено внутрішню структуру базового процесу. У ньому виділено чотири ключові підпроцеси: автентифікація користувача, створення індивідуального раціону, оформлення замовлення на регулярну доставку та змінення статусу замовлення. Дана модель відображає логічну послідовність дій, що виконує система, починаючи від входу користувача та закінчуючи завершенням циклу доставки. Водночас усі дані рухаються у вигляді чітко визначених інформаційних потоків між блоками.

На рисунку 2.3 показано деталізацію процесу створення індивідуального раціону. Тут розкрито механізм вибору користувачем страв, побудову раціону відповідно до обраних параметрів та підтвердження створеного набору. Діаграма демонструє тісну інтеграцію з довідниками страв та шаблонами формування раціону, а також необхідність взаємодії із системними даними користувача. Результатом підпроцесу є сформований та збережений індивідуальний раціон.

На рисунку 2.4 зображено процес оформлення замовлення на регулярну доставку. Діаграма описує, як користувач обирає раціони для доставки, визначає бажаний розклад і підтверджує створений графік. Модель підкреслює, що система повинна забезпечувати коректність вибору дат, відповідність замовлення правилам і можливість подальшого формування графіка доставки, що використовується кур’єрами та співробітниками сервісу.

Разом ці діаграми формують цілісну модель функціонування початкової версії системи. Вони описують основні інформаційні потоки, бізнес-операції та взаємодію між користувачами і внутрішніми підсистемами сервісу. Усі процеси реалізовані у вигляді послідовних логічних кроків, що забезпечують створення раціону, оформлення доставки та контроль за виконанням замовлення.

Важливо підкреслити, що наведені діаграми були створені для системи, яка не містила інтелектуального функціоналу та мала виключно операційну спрямованість. У ній відсутні механізми аналізу попиту, обробки історичних даних, врахування зовнішніх факторів чи генерування рекомендацій для планування меню. Всі процеси мали статичний характер і не використовували прогнозні алгоритми, що унеможливлювало автоматизоване планування

виробництва та оптимізацію ресурсів.

Оскільки система була орієнтована лише на реалізацію замовлень, без аналітичної складової, її структура потребує розширення для інтеграції штучного інтелекту. Діаграма не містить жодного процесу, який би відповідав за обробку історії замовлень, аналіз ринкових факторів, виявлення трендів чи формування прогнозу попиту. Відсутні також відповідні інформаційні потоки, що мали б надходити з зовнішніх джерел, таких як погодні умови, ціни, сезонність або промокампанії.

Тому для реалізації інтелектуального модуля прогнозування необхідно провести модернізацію базової моделі. Усі основні діаграми мають бути доповнені новими потоками даних і новим підпроцесом, який відповідає за генерацію прогнозу попиту на страви та створення рекомендацій. Це дозволить перетворити систему з операційної на аналітично-керовану, що підвищить ефективність планування, забезпечить стабільність роботи та оптимізує використання ресурсів сервісу доставки раціонів.

На рисунку 2.5 представлено оновлену діаграму верхнього рівня A-0, яка описує функціонування системи доставки індивідуальних раціонів уже з інтегрованим модулем прогнозування споживчого попиту. У порівнянні з базовою моделлю, дана схема зазнала суттєвих доповнень, що відображають нові інформаційні потоки та зовнішні фактори, необхідні для роботи штучного інтелекту. Оновлена діаграма демонструє розширений контекст системи, у якому ключову роль починає відігравати аналітичний модуль і механізми прогнозування, що формують аналітичні рекомендації на основі історичних даних та зовнішніх впливів.

Одним із важливих нововведень є потік «Історія замовлень». У базовій моделі ці дані використовувалися лише для відображення попередніх дій користувача, але тепер вони стали одним із основних джерел інформації для тренування моделі машинного навчання. Саме на основі багаторічних даних про замовлення можна визначити сезонність попиту, поведінкові патерни клієнтів і вплив зовнішніх факторів на динаміку продажів окремих страв. Таким чином, історія замовлень є критичним елементом входу для інтелектуального модуля.

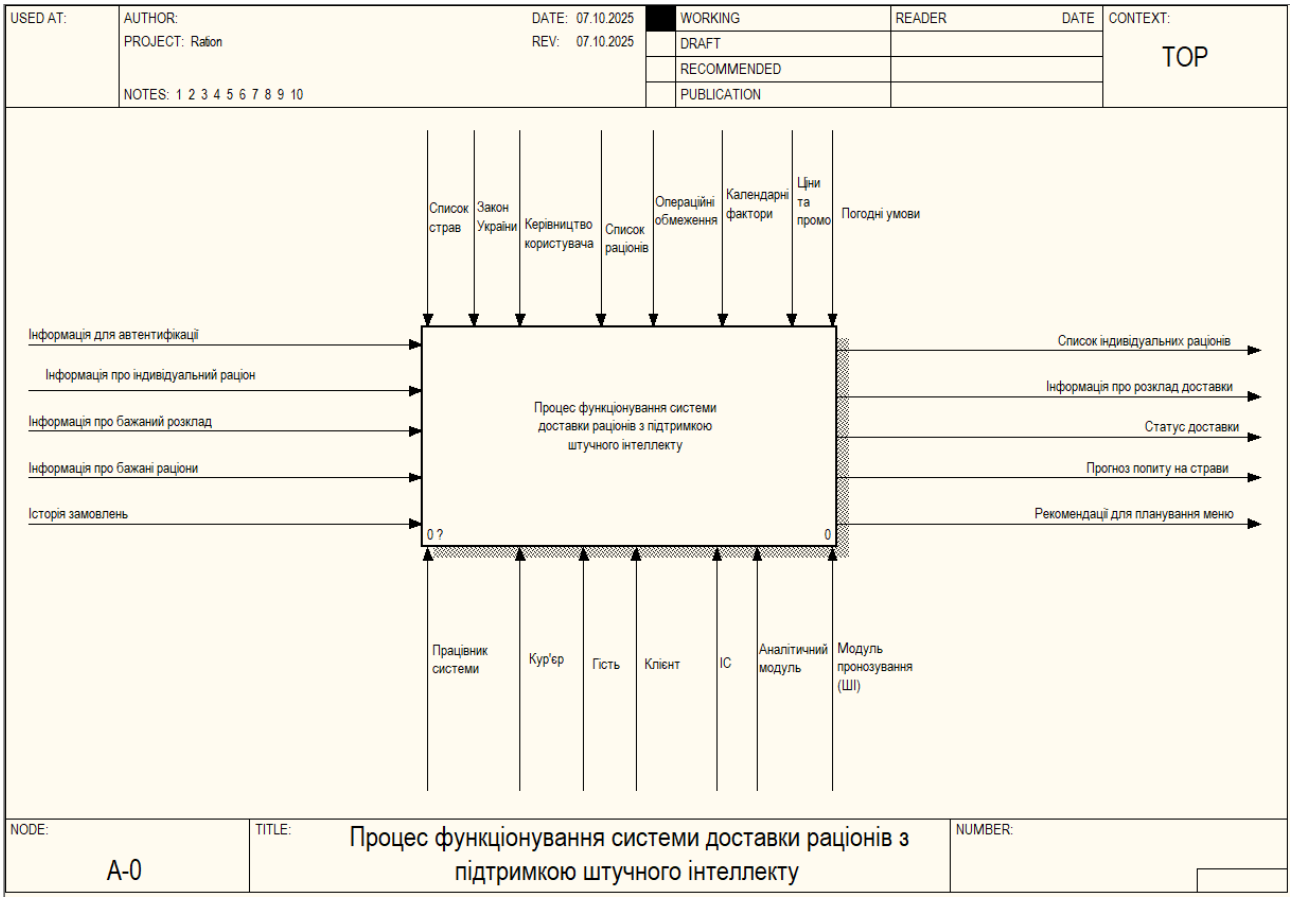

Рисунок 2.1 – Модернізована контекстна діаграма

До діаграми також додано потоки «Календарні фактори», «Ціни та промо» та «Погодні умови». Ці три категорії зовнішньої інформації є основою для формування коректних прогнозів попиту, оскільки система більше не обмежується лише внутрішніми процесами замовлень. Сезонність, свята, акції, знижки, погодні коливання та різноманітні промо-кампанії суттєво впливають на попит у сфері харчування. Тому інтеграція цих потоків забезпечує модель релевантними даними про реальні зовнішні умови.

Також з’явився потік «Операційні обмеження». У базовій системі такі параметри не оброблялися на рівні контекстної діаграми. Тепер операційні обмеження (місткість кухні, доступність інгредієнтів, графіки роботи персоналу) впливають на формування рекомендацій, щоб прогноз не суперечив доступним ресурсам. Це дозволяє створити більш реалістичні плани виробництва та уникати перевантаження системи.

Суттєвою новацією є поява двох нових виконавців – «Аналітичний модуль» і «Модуль прогнозування (ШІ)». Аналітичний модуль відповідає за

агрегацію, структурування та попередню обробку даних. Він формує єдиний набір структурованих ознак, який надалі передається до модуля штучного інтелекту. Це дозволяє винести складні аналітичні операції з основної системи та оптимізувати процес передбачення попиту.

Модуль прогнозування, своєю чергою, здійснює безпосередню генерацію прогнозів на основі отриманих даних. Він виконує машинне навчання, формує прогнозні значення попиту на кожну страву та генерує рекомендації для планування меню. На діаграмі видно, що результати його роботи передаються назад у систему у вигляді двох окремих вихідних потоків – «Прогноз попиту на страви» та «Рекомендації для планування меню». Це показує, що інтелектуальний модуль виконує як обчислювальну, так і дорадчу функцію.

Додано також нові вихідні потоки. «Прогноз попиту на страви» – це числові значення очікуваного обсягу замовлень, які можуть бути використані для планування виробництва, закупівель та оптимізації графіків. У свою чергу, «Рекомендації для планування меню» – це аналітичні підказки, які допомагають адміністраторам системи приймати рішення щодо асортименту, кількості страв та оптимального графіка їх виробництва.

Важливо зазначити, що на оновленій діаграмі збережено всі попередні інформаційні потоки, що підтримували базові функції системи. Це демонструє спадковість моделі: система не втратила жодної з основних можливостей, але була розширена для виконання аналітичних завдань. Водночас додані потоки інтегруються в загальний контекст без порушення структури діаграми.

Також змінився характер взаємодії між виконавцями системи. Якщо раніше до ключових виконавців належали користувач, кур’єр, гість та працівник системи, то тепер інтелектуальні модулі стають повноправними учасниками інформаційних процесів. Це відображає нову роль системи, яка перестає бути суто операційною та набуває елементів автономності.

Оновлена діаграма також дозволяє підкреслити точку входу додаткових джерел даних, які раніше не були частиною екосистеми. Це важливий аспект, оскільки інтеграція ШІ неможлива без розширення інформаційної бази. Завдяки новим потокам діаграма набуває ширшого контексту, що відповідає реальним

умовам функціонування сучасних сервісів доставки.

У результаті модернізації контекстна діаграма рівня А-0 стала відображенням вже не просто сервісу доставки раціонів, а комплексної аналітичної системи, у якій штучний інтелект відіграє роль ключового компоненту. Вона демонструє, як саме ШІ інтегрується в основну логіку роботи та які зовнішні та внутрішні дані забезпечують його коректне функціонування. Таке доповнення є необхідним для переходу системи від базового функціоналу до інтелектуального планування та прогнозування.

На рисунку 2.6 наведено оновлену діаграму декомпозиції рівня А0, у якій традиційний процес функціонування системи доставки раціонів був розширений шляхом інтеграції модуля прогнозування попиту. На відміну від базової версії діаграми, нова модель демонструє більш складні інформаційні взаємозв'язки між процесами та появу додаткових потоків даних, що забезпечують функціонування інтелектуальної підсистеми. Саме на цьому рівні вперше чітко простежується роль нового процесу – «Прогнозувати споживчий попит на страви», який став ключовим елементом модернізованої системи.

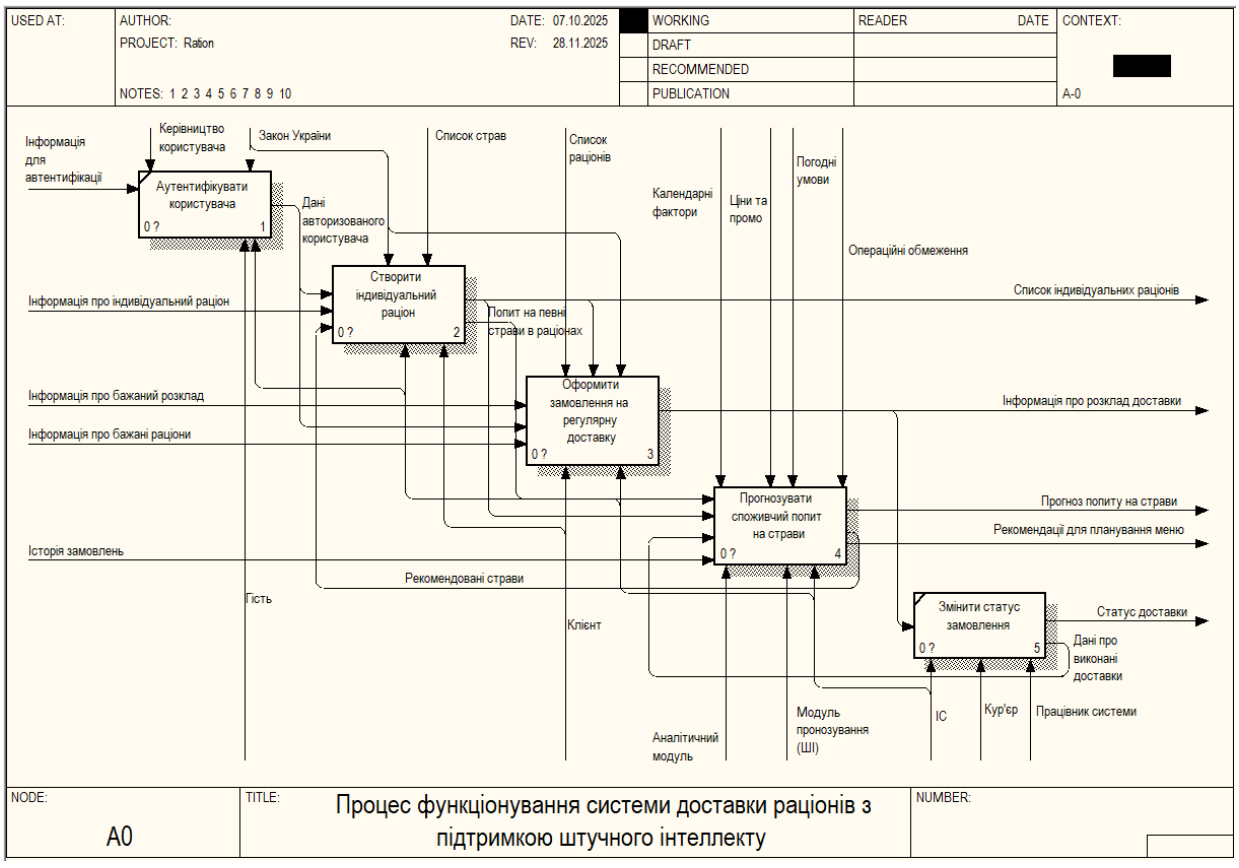

Рисунок 2.6 – Модернізована діаграма декомпозиції А0

Одним із важливих нововведень є поява вхідного потоку «Попит на певні страви в раціонах», який надходить до процесу створення індивідуального раціону та процесу оформлення регулярного замовлення. Цей параметр розраховується автоматично на основі попередніх замовлень та даних аналітичного модуля. Завдяки цьому процес формування раціону враховує популярність різних страв серед користувачів, а також потенційне навантаження на кухню.

Важливим доповненням стало введення інформаційного входу «Історія замовлень», який є ключовим джерелом даних для процесу прогнозування. Цей потік не використовувався у базовій системі як аналітичний ресурс, проте тепер він стає центральним фактором, що визначає якість прогнозів. Модель машинного навчання отримує доступ до хронологічних даних про замовлення, що дозволяє визначати тренди, сезонні цикли та повторювані поведінкові патерни клієнтів.

У оновленій діаграмі додано значно більше зовнішніх факторів, зокрема «Календарні фактори», «Ціни та промо» та «Погодні умови». Усі ці дані надходять безпосередньо до процесу прогнозування попиту. Саме вони дозволяють системі адаптувати очікуваний попит до реального контексту – наприклад, збільшення попиту на теплі страви у холодні періоди або підвищення кількості замовлень у дні великих свят. Така інтеграція забезпечує модель релевантними факторами, підвищуючи точність прогнозування.

Суттєвим доповненням є поява процесу «Прогнозувати споживчий попит на страви». Він збирає інформацію з багатьох потоків – внутрішніх, зовнішніх та операційних. На діаграмі можна побачити, що до цього процесу надходять дані не лише про історію замовлень та календарні фактори, а й про операційні обмеження. Це дозволяє формувати прогноз, який враховує не лише бажання користувачів, а й реальні можливості виробництва. Наприклад, якщо певна страва має низьку доступність інгредієнтів, система може пропонувати альтернативи.

Результатом роботи процесу прогнозування є два нові вихідні потоки – «Прогноз попиту на страви» та «Рекомендації для планування меню». Прогноз

попиту використовується для планування закупівель, а рекомендації допомагають адміністраторам підбирати найбільш релевантні страви для меню на майбутній період. Таким чином, впровадження ШІ не лише підсилює аналітичні можливості системи, але й робить її здатною впливати на формування асортименту.

На рівні взаємодії між процесами з'явився новий інформаційний ланцюжок – «Рекомендовані страви». Цей потік передається від модуля прогнозування до процесу створення індивідуального раціону. Завдяки цьому користувач отримує не лише можливість вибору з усіх страв, а й рекомендації, побудовані на основі його попередніх замовлень та прогнозованої популярності страв серед інших клієнтів.

Таким чином, кожен з існуючих процесів отримав додаткові входи або виходи, що значно розширює функціональні можливості всієї системи. Процес створення індивідуального раціону тепер включає вплив рекомендацій моделі, процес оформлення замовлення враховує очікуване майбутнє навантаження, а процес зміни статусу замовлення передає дані назад у прогнозну систему, забезпечуючи замкнутий цикл навчання.

Новим є і включення аналітичного модуля, який відповідає за підготовку даних. Він взаємодіє з процесом прогнозування попиту, формуючи структурований набір параметрів. У базовій системі цього елемента не існувало, проте для роботи будь-якої моделі машинного навчання необхідна якісна обробка даних, і його поява на діаграмі демонструє завершеність нового аналітичного контуру.

Окремо варто підкреслити появу додаткових інформаційних потоків між користувачем і системою. Зокрема, завдяки інтеграції ШІ частина рішень приймається автоматично, що значно оптимізує процес формування раціону та дозволяє системі більш точно реагувати на попит. Таким чином, діаграма на рисунку 2.6 демонструє глибоку трансформацію логіки роботи системи та її перехід до рівня інтелектуалізованого сервісу.

На рисунку 2.7 подано оновлену діаграму декомпозиції процесу «Створити індивідуальний раціон», у якій відображені зміни, пов'язані з інтеграцією

підсистеми штучного інтелекту. На відміну від базового варіанта, дана версія демонструє, як процес формування індивідуального раціону користувача підсилюється рекомендаційними механізмами та впливає на подальше оновлення даних про попит на страви. Таким чином, процес набуває не лише операційної, а й аналітичної складової, забезпечуючи зворотний зв'язок для моделі прогнозування.

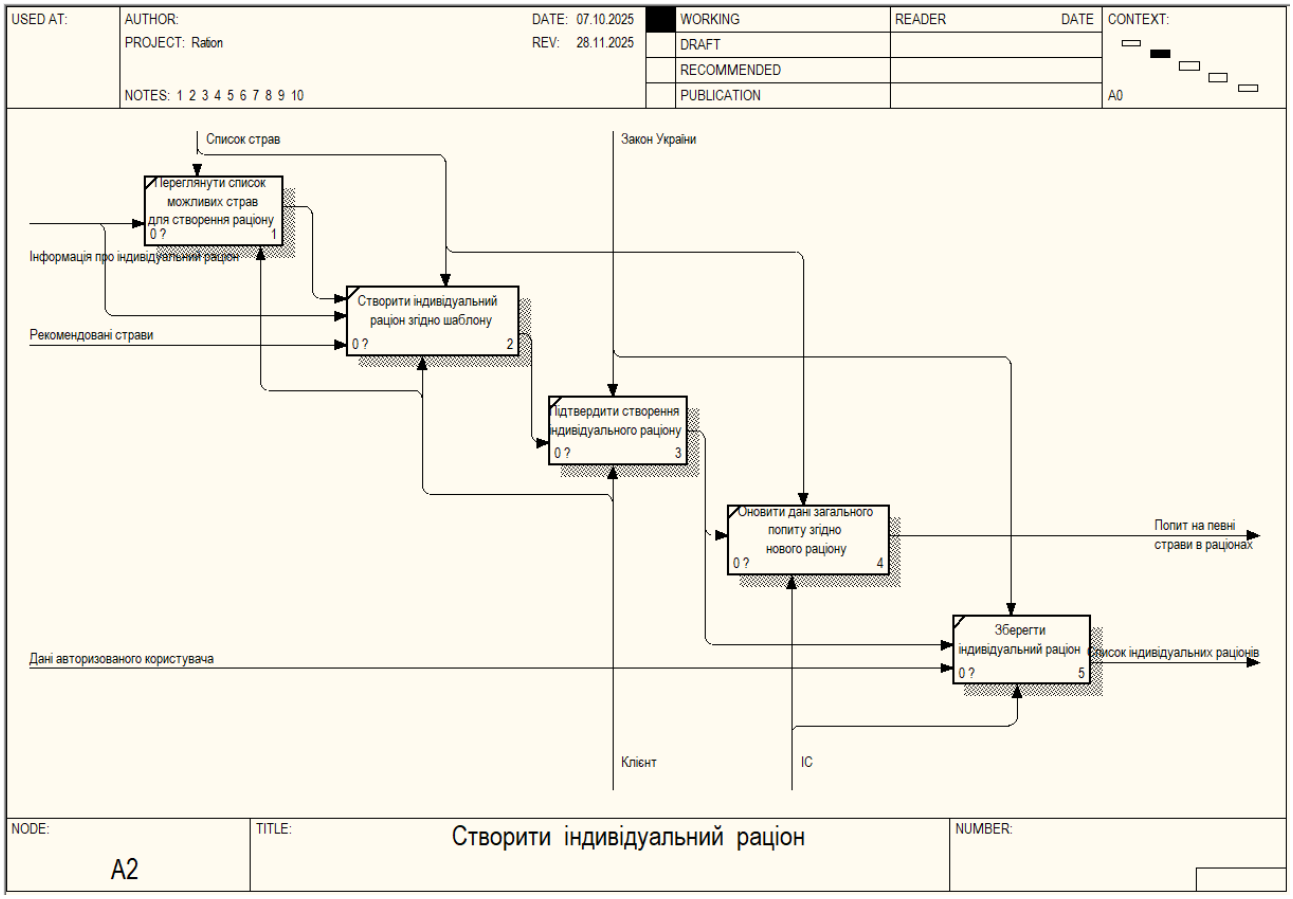

Рисунок 2.7 – Модернізована діаграма декомпозиції A2

Першим суттєвим доповненням є поява нового інформаційного потоку «Рекомендовані страви», який надходить до підпроцесу «Переглянути список можливих страв для створення раціону». Цей потік формується на основі попередніх замовлень користувача, прогнозних даних та загальної популярності страв. Завдяки цьому перегляд списку страв перестає бути статичним – система пропонує користувачеві варіанти, які з більшою ймовірністю відповідатимуть його вподобанням або поточним трендам.

Другим важливим доповненням є зміни у процесі «Створити

індивідуальний раціон згідно шаблону». Якщо раніше шаблон формувався виключно за участі користувача, то тепер він може частково базуватися на рекомендаціях системи. У цьому підпроцесі користувач взаємодіє з набором страв, який уже оптимізовано з урахуванням прогнозованих обмежень, популярності та індивідуальних вподобань. Це дозволяє формувати раціон більш ефективно та швидко.

Надзвичайно важливою є поява підпроцесу «Оновити дані загального попиту згідно нового раціону». Цей блок з'являється після підтвердження створення раціону та забезпечує актуалізацію оперативних даних щодо попиту на страви. Кожен створений раціон формує внесок у загальну статистику, що згодом може бути використана для покращення прогнозу. Таким чином діаграма демонструє реалізацію механізму зворотного зв'язку між процесами користувача та аналітичним модулем.

Вихідний потік «Попит на певні страви в раціонах», який формується після оновлення даних попиту, є одним із ключових джерел для подальшого прогнозування. Раніше подібних даних на цьому рівні не виділяли, але тепер вони відіграють важливу роль у циклі постійного навчання та уточнення моделі ШІ. Таким чином, система отримує можливість працювати в режимі частково онлайн-оновлюваної аналітики.

Окрему увагу варто приділити тому, що процес «Підтвердити створення індивідуального раціону» тепер виступає тригером для оновлення попиту. У базовій діаграмі цей етап лише завершував процедуру створення раціону, але не спричиняв жодних подальших аналітичних дій. У модернізованій версії підтвердження слугує сигналом для модулів, що відповідають за облік і аналіз попиту.

Після проходження етапу оновлення даних попиту процес завершуються блоком «Зберегти індивідуальний раціон», який також був модифікований. Тепер вихідний потік цього блоку – «Список індивідуальних раціонів» – може містити додаткову метричну інформацію, яка буде корисною аналітичній підсистемі. Це дозволяє ефективно поєднати оперативну ланку системи з її інтелектуальною частиною.

На оновленій діаграмі помітно, що всі підпроцеси працюють у тісному взаємозв’язку з даними користувача. Потік «Дані авторизованого користувача» зберіг свою роль, але його використання стало більш важливим, адже рекомендаційний механізм частково базується саме на персоналізації. Таким чином система формує індивідуальні пропозиції, враховуючи особисті дані, попередні замовлення та смакові вподобання.

Важливо, що модернізована діаграма демонструє нову логіку руху інформації після створення раціону. Якщо раніше процес завершувався на етапі збереження, то зараз він автоматично запускає оновлення даних попиту. Це перетворює систему на комплексний механізм, у якому кожна дія користувача відображається в аналітичній моделі.

Окремим елементом є тонка взаємодія із законом України у контексті дієтичних норм, харчової безпеки та правил оформлення меню. Хоча цей потік залишився з базової діаграми, його вплив у новій моделі також враховується системою рекомендацій, оскільки деякі страви можуть мати обмеження або особливі вимоги.

У підсумку оновлена діаграма (рисунок 2.7) не лише розширює функціональні можливості базової версії, а й демонструє взаємопов’язаність між процесами створення раціону та підсистемою прогнозування. Саме завдяки цим змінам система виходить на новий рівень інтелектуалізації, де кожен крок користувача впливає на загальні аналітичні процеси та точність подальших прогнозів.

На рисунку 2.8 наведено оновлену діаграму підпроцесу «Оформити замовлення на регулярну доставку», яка, на відміну від інших фрагментів модернізованої моделі, не зазнала змін у структурі, потоках інформації чи логіці взаємодії. Це пов’язано з тим, що даний підпроцес стосується виключно операційної частини роботи системи – вибору раціонів, визначення дат доставки та підтвердження розкладу – і не має прямої залежності від прогнозних або рекомендаційних механізмів штучного інтелекту.

У процесі вибору раціонів користувач керується власними потребами та сформованими раціонами, тому на цьому рівні ШІ не впливає на прийняття

рішень. Підпроцес також не передбачає використання прогнозних даних чи аналітичних моделей, адже його мета – зафіксувати параметри доставки, а не впливати на формування раціону чи прогноз попиту.

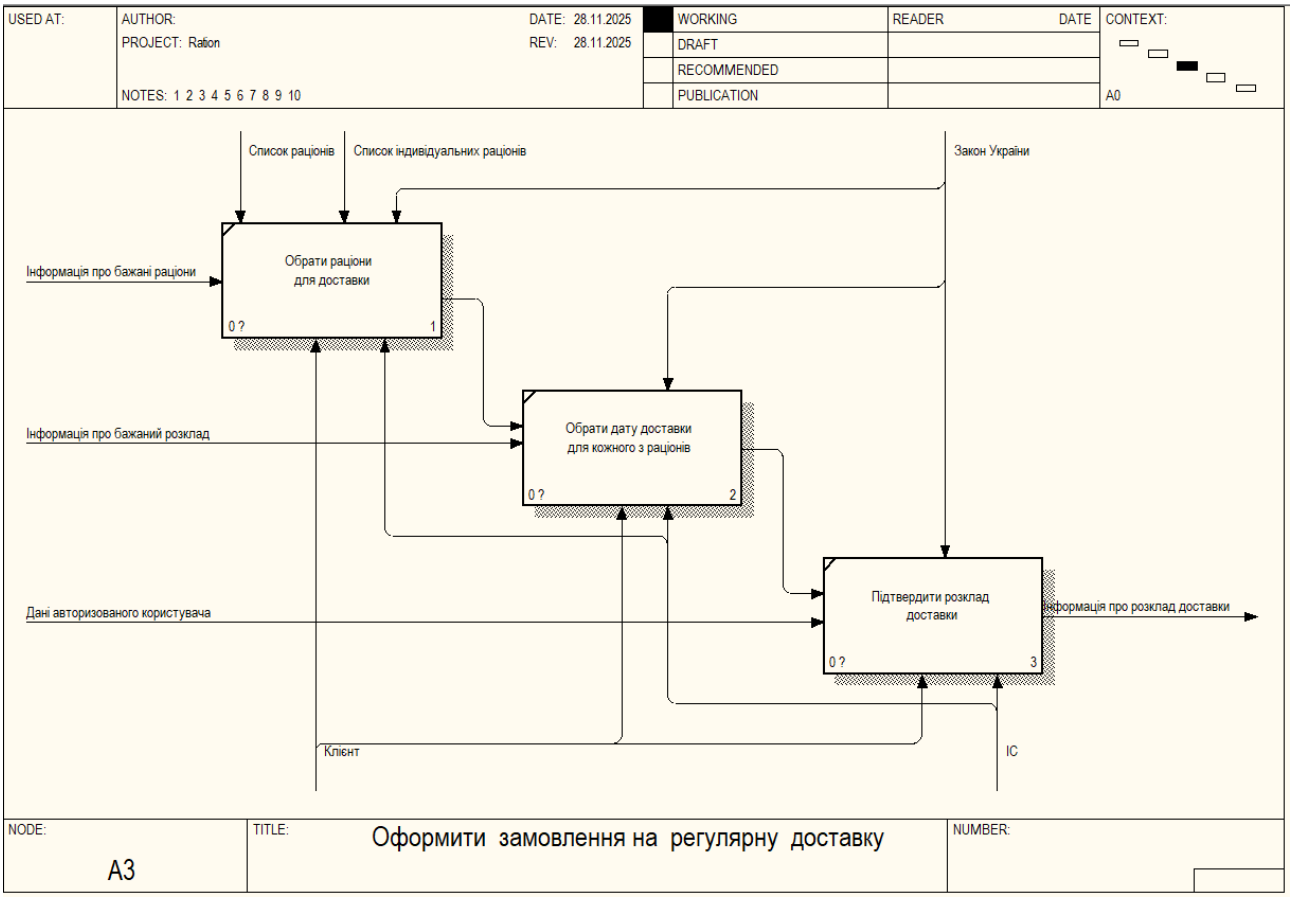

Рисунок 2.8 – Модернізована діаграма декомпозиції A3

Також відсутні зміни в інформаційних потоках: система, як і раніше, використовує дані про індивідуальні раціони, побажання клієнта щодо графіка, а також відображає підтверджений розклад у вигляді вихідного потоку. Оскільки жодні додаткові компоненти або модулі прогнозування не беруть участі в цьому процесі, діаграма в модернізованому варіанті повністю повторює структуру базової моделі.

Таким чином, підпроцес на рисунку 2.8 залишився незмінним, оскільки не потребував інтеграції механізмів штучного інтелекту та вже повністю відповідає функціональним вимогам системи навіть у її розширеній версії.

На рисунку 2.9 представлено абсолютно нову декомпозицію підпроцесу «Прогнозувати споживчий попит на страви», яка раніше була відсутня в базовій

діаграмі функціональних вимог. Даний фрагмент моделі з’явився виключно після інтеграції підсистеми штучного інтелекту, тому він є ключовим елементом модернізованої системи та демонструє логіку формування прогнозу попиту й рекомендацій на основі історичних та зовнішніх факторів.

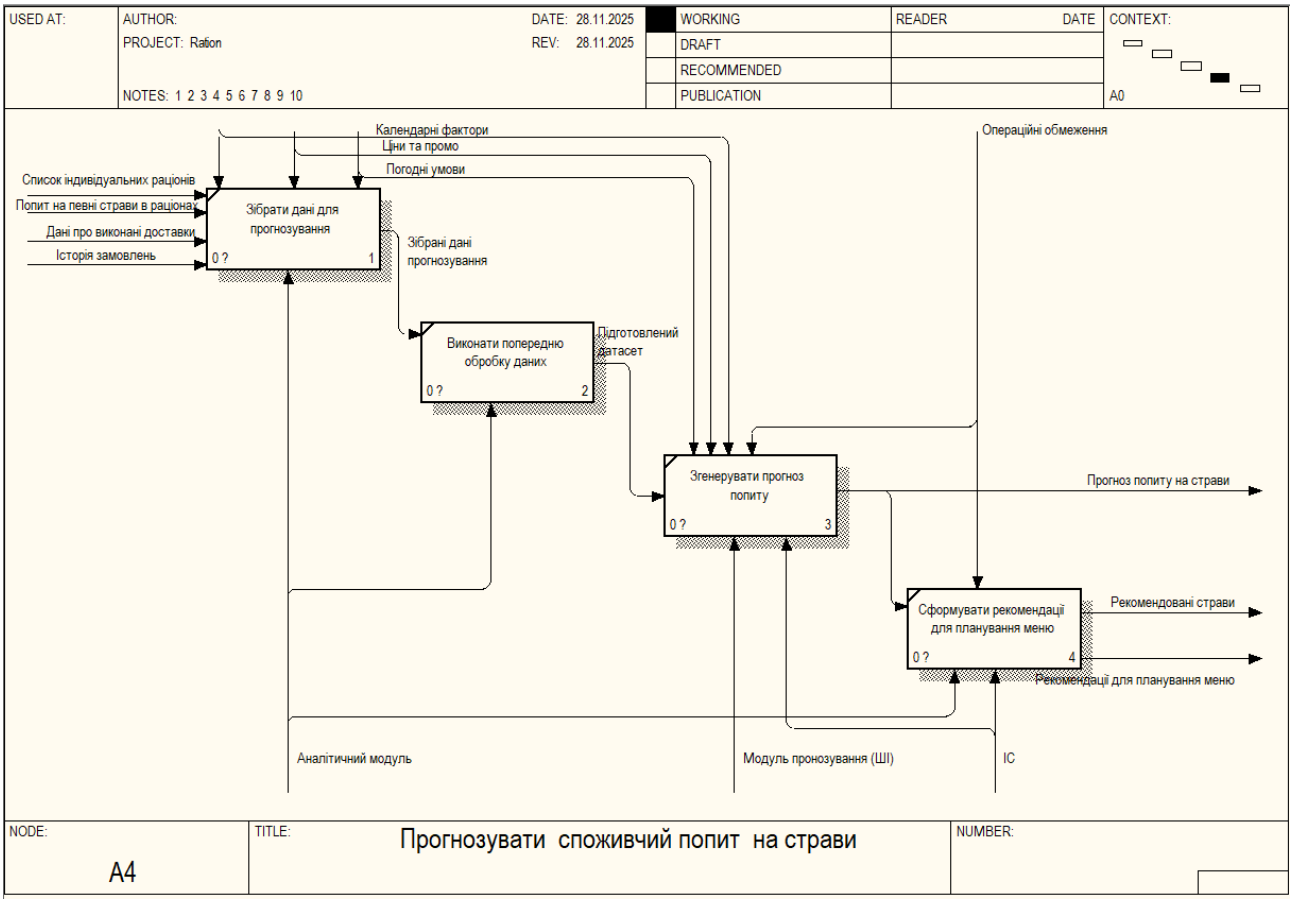

Рисунок 2.9 – Діаграма декомпозиції A4

Першим блоком у декомпозиції є «Забрати дані для прогнозування». Саме тут відбувається збір усіх необхідних вхідних даних, без яких прогнозування було б неможливим. До таких даних входять: історія замовлень, попит на конкретні страви в раціонах, список індивідуальних раціонів, дані про виконані доставки, а також зовнішні фактори – календарні події, погода, акційні пропозиції. Усі ці потоки надходять або з бази даних, або зі сторонніх сервісів, а потім передаються на подальшу обробку.

Результатом роботи цього блоку є узагальнена сукупність даних – зібрані дані прогнозування, які передаються до наступного етапу. Саме тут формується первинний «пакет» інформації, який забезпечує модель машинного навчання

необхідною базою для подальших обчислень.

Другий блок – «Виконати попередню обробку даних» – відповідає за підготовку датасету. До попередньої обробки входить очищення пропусків, нормалізація полів, агрегація часових рядів, формування цільових змінних, кодування категоріальних параметрів і видалення аномалій. Це один із найважливіших етапів, тому що якість прогнозування безпосередньо залежить від якості підготовлених даних.

Результатом обробки є підготовлений датасет, структура якого вже відповідає вимогам моделі прогнозування. Саме цей датасет потім передається у модуль ШІ для генерації прогнозу.

Третій блок – «Згенерувати прогноз попиту» – є центральним компонентом підсистеми штучного інтелекту. Тут працює модель машинного навчання, яка на основі підготовлених даних визначає очікуваний попит на страви у майбутні періоди. В обчисленні можуть враховуватися сезонність, характер поведінки користувачів, промоактивності, зміни погоди та інші зовнішні фактори.

На цьому етапі система формує вихідний потік «Прогноз попиту на страви», який уже може бути використаний адміністраторами або іншими підсистемами. Проте модель не обмежується лише прогнозуванням – її результати стають основою для наступного логічного кроку.

Четвертий блок – «Сформувати рекомендації для планування меню» – використовує прогноз попиту як базове джерело інформації. На основі передбачуваних трендів система визначає, які страви доцільно включати в меню на певний період, які страви можуть потребувати збільшення обсягів приготування, а які – навпаки, можуть бути тимчасово зменшені або замінені.

У результаті блок формує два ключові вихідні потоки:

– «Рекомендовані страви», що можуть бути використані автоматично для пропозицій клієнтам;

– «Рекомендації для планування меню», призначені для адміністраторів або кухарів, які планують закупівлі та виробництво.

На окрему увагу заслуговують зовнішні стейкхолдери діаграми. Аналітичний модуль отримує частину даних і може додатково взаємодіяти з

попередніми блоками, забезпечуючи корекцію або уточнення інформації. Модуль прогнозування (ШІ) є технологічним ядром, яке забезпечує високоточні обчислення та визначає поведінкові закономірності попиту.

Також важливо, що на діаграмі враховано операційні обмеження, такі як доступність інгредієнтів, обмеження виробничих потужностей, графіки кухні та інші фактори. Це дозволяє формувати рекомендації, які будуть не лише точними з точки зору прогнозів, а й реалістичними з точки зору виконання.

Таким чином, декомпозиція на рисунку 2.9 є абсолютно новою частиною функціональної моделі, оскільки саме вона забезпечує реалізацію ключового елементу модернізованої системи – прогнозування та рекомендацій на основі штучного інтелекту. Всі блоки працюють як цілісна аналітична підсистема, що перетворює різнорідні дані на корисні управлінські рішення та підвищує ефективність функціонування сервісу доставки раціонів.

У результаті модернізації діаграми функціональних вимог система доставки раціонів отримала суттєве розширення функціональних можливостей, що дозволило перетворити її з традиційної інформаційної системи в інтелектуальну платформу підтримки прийняття рішень. Додані елементи, пов’язані з модулем штучного інтелекту, забезпечили якісно новий рівень автоматизації, зокрема в процесах прогнозування попиту та формування рекомендацій, що раніше були недоступними або виконувалися вручну. Завдяки цьому система стала здатною не лише реагувати на вже сформовані замовлення, а й проактивно впливати на планування меню та управління операційними процесами.

Запровадження додаткових потоків даних – таких як календарні фактори, погода, ціни та промоактивності, а також історія замовлень – дозволило значно розширити інформаційну базу для прийняття рішень. У базовій версії моделі ці фактори не враховувалися, що обмежувало аналітичні можливості системи. Модернізація усунула цей недолік, забезпечивши збір, агрегування та використання як внутрішніх, так і зовнішніх даних для побудови високоточних прогнозів.

Важливим результатом оновлення є також поява окремого підпроцесу,

який повністю присвячений прогнозуванню попиту. Така декомпозиція дозволяє чітко визначити функціональні межі аналітичного модуля та розділити відповідальність між етапами збору, обробки та аналізу даних. Це підвищує прозорість логіки функціонування системи та створює основу для подальшого розширення можливостей модуля ШІ.

Унаслідок модернізації система стала здатною генерувати рекомендації для планування меню, що має велике значення для оптимізації виробничих процесів. Очікувані дані про попит дозволяють зменшити надлишкове приготування страв, оптимізувати закупівлі, уникнути перевитрат інгредієнтів та збільшити ефективність логістики доставки. Таким чином, модуль прогнозування впливає не лише на цифрову частину системи, а й на реальні бізнес-процеси компанії.

Оновлення діаграми функціональних вимог також сприяло підвищенню рівня інтегрованості між компонентами системи. Зокрема, усі підпроцеси – від створення індивідуального раціону до оформлення регулярної доставки – тепер взаємодіють із аналітичним модулем, що забезпечує контекстну актуальність рекомендацій та адаптивність поведінки системи до змін попиту.

Важливо й те, що модернізована діаграма відобразила введення нового суб’єкта взаємодії – модуля прогнозування (ШІ). Додавання цього компонента дозволило системі працювати за принципами сучасних інтелектуальних сервісів, де значна частина рішень формується автоматично на основі обчислень, а не вручну адміністратором. Це суттєво зменшує навантаження на персонал, підвищує точність планування й забезпечує можливість масштабування сервісу.

У підсумку, модернізація діаграми функціональних вимог не лише розширила функціональність системи, але й підвищила її гнучкість, адаптивність та ефективність. Вона чітко продемонструвала, як інтеграція інтелектуальних технологій може трансформувати традиційну систему доставки харчування в сучасну аналітичну платформу, здатну підтримувати прийняття управлінських рішень та підвищувати якість сервісу для кінцевого користувача.

2.3 Визначення особливостей структури даних для прогнозування попиту

Наявна структура бази даних, зображена на рисунку 2.10, була створена для стандартної системи управління замовленнями без використання штучного інтелекту. Вона дозволяє ефективно опрацьовувати процеси замовлення, формування раціонів і збереження даних про користувачів, однак не передбачає обробку аналітичних даних, необхідних для побудови моделей прогнозування попиту. Тому для впровадження механізмів машинного навчання потрібно внести низку критично важливих змін, які дадуть змогу зберігати історичні та зовнішні дані у зручному для аналізу вигляді.

Насамперед слід розширити структуру, щоб система фіксувала не лише факт створення замовлення, а й часові характеристики. У таблицях Order і OrderRation уже містяться поля CreateDate та DeliveryDate, проте для прогнозування важливо також знати дату та час підтвердження, відправки й завершення доставки. Для цього достатньо додати одне поле, наприклад StatusDate, яке оновлюватиметься при кожній зміні статусу. Таким чином можна отримати часовий ряд замовлень і простежити активність клієнтів за днями або тижнями.

Щоб моделі могли аналізувати зміну попиту на рівні окремих страв, доцільно зберігати точну кількість замовлених позицій. Таблиця IndividualRationProduct уже містить поле Count, яке відображає кількість страв у межах раціону. Для прогнозування цього достатньо, але варто забезпечити цілісність зв’язку між продуктом і датою доставки – кожен запис повинен однозначно показувати, скільки разів конкретна страва була обрана клієнтами в конкретний день. Це дозволить сформувати часові ряди попиту на кожну позицію меню.

Ще одним кроком є збереження інформації про ціни в часі. Поле PricePer100g у таблиці Product відображає поточну ціну, однак моделі машинного навчання потребують історії зміни вартості, адже коливання ціни може безпосередньо впливати на попит. Тому доцільно створити нову таблицю, наприклад ProductPriceHistory, у якій зберігатимуться дата початку дії ціни, її

значення та ідентифікатор продукту. Це дозволить враховувати ефект знижок або акцій під час навчання моделі.

Рисунок 2.10 – база даних, що використовувалася створена для стандартної системи управління замовленнями інливідуальних раціонів без використання штучного інтелекту

Для підвищення точності прогнозу корисно враховувати календарні чинники. У системі варто додати невелику таблицю Calendar, де зберігатимуться дні тижня, інформація про вихідні, свята та сезонні періоди. Ці дані не потребують значних ресурсів, але нададуть моделі важливий контекст – наприклад, підвищений попит на вихідних або під час свят. Поле CalendarId може бути зв’язане з таблицею OrderRation через дату доставки.

Крім календарних факторів, певний вплив на замовлення може мати

погода. У реальній системі не обов’язково зберігати детальні метеодані, достатньо створити спрощену таблицю Weather, яка міститиме дату, середню температуру та умови (сонячно, дощ, сніг). Такі записи дозволять враховувати базову сезонність попиту без складної інтеграції з зовнішніми API. Підключення цих даних до таблиці замовлень допоможе моделі визначати, наприклад, зростання замовлень на теплі страви у холодний період року.

Важливим елементом є статус замовлення. Існуюча таблиця Status дозволяє контролювати поточний стан, але для прогнозування корисно зберігати історію змін статусів. Для цього можна додати таблицю OrderStatusHistory, де фіксуватимуться OrderId, StatusId та дата зміни. Завдяки цьому можна виключити з навчальної вибірки скасовані або помилкові замовлення, залишаючи лише підтверджені факти реального попиту.

Щоб система могла навчатися на історії, потрібно забезпечити стабільне збереження даних про минулі замовлення без їхнього перезапису. Зараз структура дозволяє редагувати записи у таблиці OrderRation або IndividualRation, але для машинного навчання важливо, щоб усі історичні дані залишалися незмінними. Тому слід обмежити можливість оновлення ключових полів, а зміни реалізовувати через створення нових записів із новими часовими мітками. Це спрощене рішення гарантує цілісність часових рядів без складного версіонування.

Окремої уваги заслуговує агрегування даних. Для формування наборів тренувальних даних доцільно додати службову таблицю DemandSummary, у якій зберігатимуться підсумкові обсяги замовлень по кожному продукту за день або тиждень. Такі агрегати можна генерувати автоматично вночі, не навантажуючи основну систему. У подальшому саме ця таблиця буде використовуватись як джерело даних для моделей прогнозування.

Щоб забезпечити стабільність навчання, усі числові поля, які беруть участь у розрахунках, мають зберігати значення у єдиних одиницях. Наприклад, для поля Calories варто зафіксувати стандарт у кілокалоріях, а для цін – у гривнях за порцію. Це спрощує підготовку даних і мінімізує помилки під час аналітики.

Для зменшення складності системи не обов’язково вводити окремі таблиці

для кожного допоміжного параметра, наприклад для знижок або логістики. Їх можна додавати як додаткові поля до наявних сутностей. Наприклад, у таблиці OrderRation можна додати DiscountPercent, а в таблиці Order – DeliveryTime. Таке спрощення робить структуру зрозумілішою та придатною для початкового етапу впровадження штучного інтелекту без надмірного ускладнення схеми.

Таким чином, оновлена структура бази даних має зберігати базові транзакційні дані, доповнені часовими атрибутами, історією цін, календарними чинниками та спрощеними погодними умовами. Цих даних буде достатньо для створення навчальної вибірки, побудови моделі машинного навчання та інтеграції прогнозів у систему без радикальної перебудови всієї бази. Такий підхід дозволяє поступово перейти від класичної системи до аналітичної, в якій дані не лише фіксують події, але й стають основою для прийняття обґрунтованих рішень щодо планування попиту.

На рисунку 2.11 представлено концептуальну модель оновленої бази даних системи доставки індивідуальних раціонів харчування з інтегрованим модулем прогнозування попиту. Модель поєднує операційні сутності – користувачів, замовлення, раціони та продукти – із аналітичними складовими, необхідними для навчання моделей машинного навчання. Додані таблиці Calendar, Weather, ProductPriceHistory та DemandSummary дозволяють враховувати сезонність, погодні умови, динаміку цін і обсяги замовлень у часі. Таким чином база даних підтримує не лише стандартні бізнес-процеси, а й накопичення історичних даних для побудови точних прогнозів.

Архітектура моделі залишається зрозумілою та логічною – основна частина зв’язків орієнтована навколо таблиці OrderRation, яка виступає центральним вузлом між замовленнями, календарем, погодою та статусами виконання. Таке рішення забезпечує можливість комплексного аналізу попиту в розрізі дат, регіонів, погодних умов і типів страв, що створює міцну основу для подальшого впровадження алгоритмів штучного інтелекту без радикального ускладнення існуючої системи.

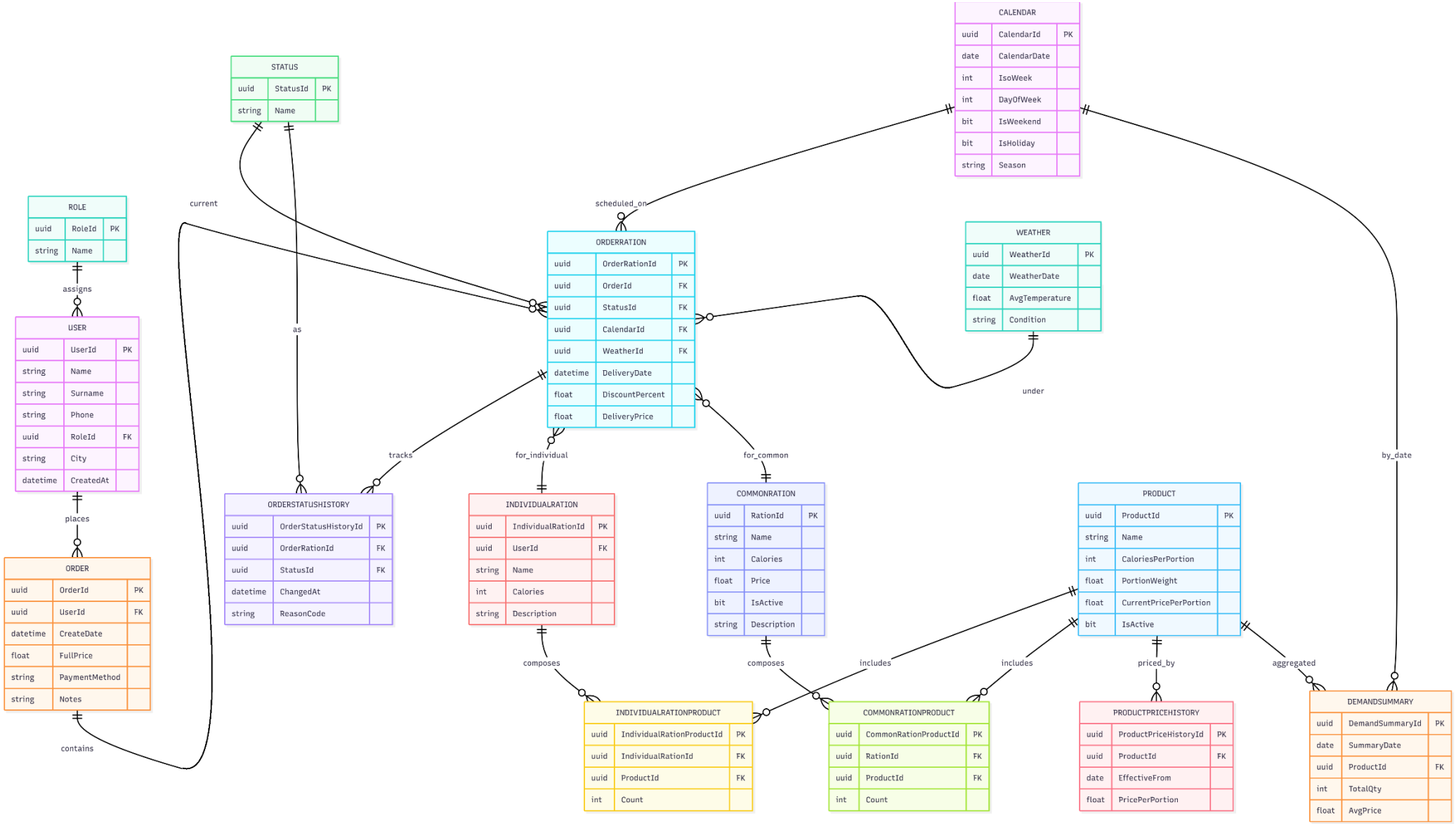

Рисунок 2.11 – концептуальна модель оновленої БД

2.4 Розробка оновленої діаграми класів системи

Наведена на рисунку 2.12 діаграма класів відображає структуру базової системи доставки раціонів, у якій реалізовані ключові сутності, контролери та зв’язки між ними. У моделі присутні класи Order, OrderRation, CommonRation, IndividualRation, Dish, Account, Role, що формують основу доменної логіки. Кожен із цих класів містить набір атрибутів, необхідних для роботи системи: наприклад, замовлення має дату створення, ціну та посилання на користувача; раціони містять інформацію про калорійність, опис, ціну та набір продуктів; облікові записи користувачів включають основні персональні дані. Така структура дозволяє реалізувати базовий функціонал формування, редагування та перегляду раціонів і замовлень.

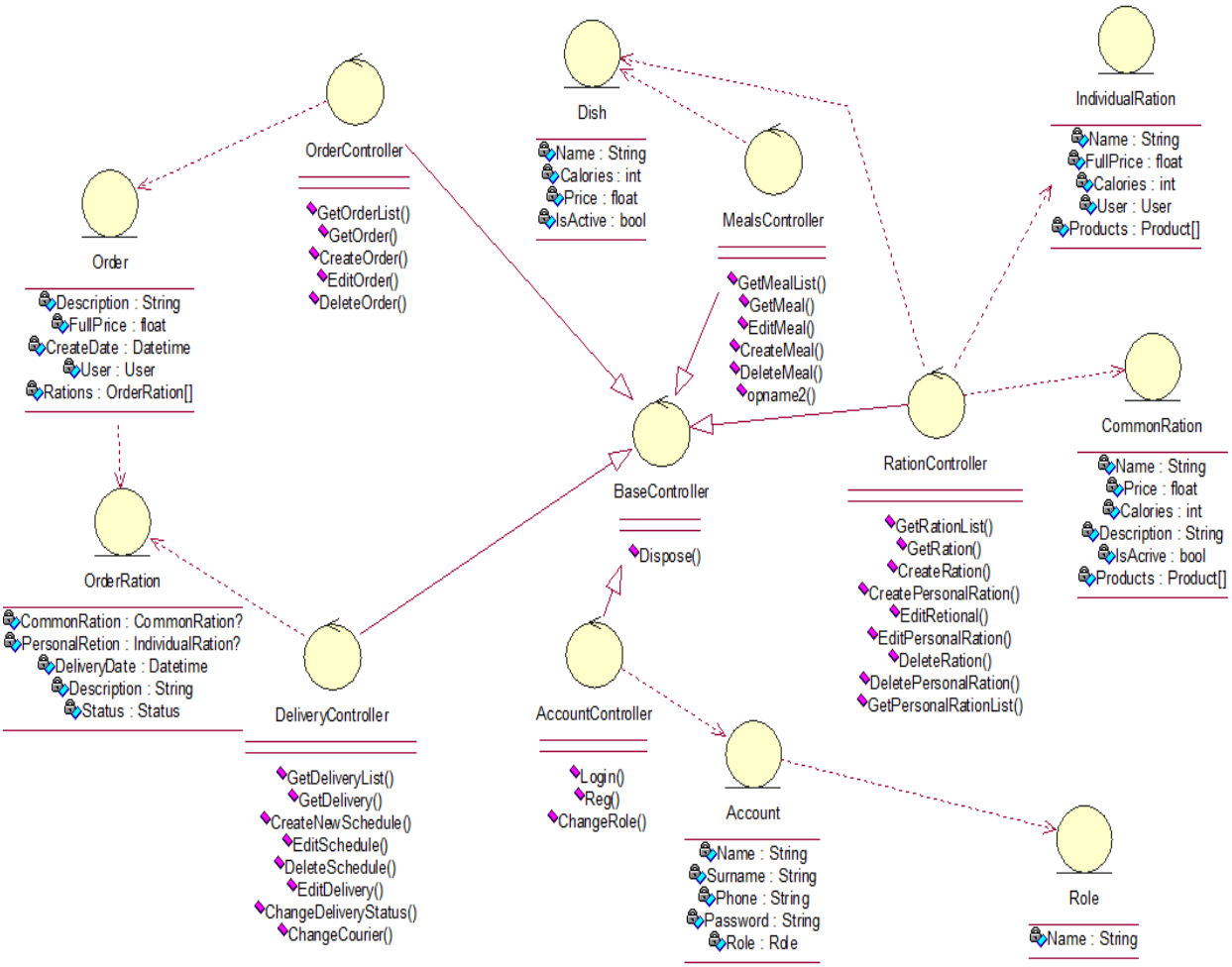

Рисунок 2.12 – Діаграма класів безінтелектуальної системи

Контролери, представлені на діаграмі, відображають основну бізнес-логіку системи. OrderController, RationController, MealsController, DeliveryController, AccountController забезпечують CRUD-операції над відповідними сутностями. Усі контролери успадковуються від BaseController, що дозволяє централізовано керувати базовими механізмами обробки запитів. Ця структура демонструє класичний підхід побудови веб-системи на основі розподілу відповідальності між компонентами та забезпечує достатню модульність у межах початкового проєкту.

Значну увагу в діаграмі приділено розділенню типів раціонів – загальних (CommonRation) і індивідуальних (IndividualRation). Це відповідає предметній області системи та дозволяє гнучко керувати продуктами в різних форматах меню. Крім того, класи OrderRation та DeliveryController забезпечують логічний зв’язок між раціоном, замовленням і доставкою, що формує основу для операційної діяльності сервісу.

Загалом діаграма добре відображає потоки роботи системи у її початковій реалізації, демонструє необхідні елементи для обслуговування замовлень і раціонів, а також забезпечує достатню декомпозицію функціоналу для підтримки основних бізнес-процесів.

Разом із тим діаграмі бракує компонентів, пов’язаних із обробкою даних для прогнозування та аналітики, що є ключовими елементами оновленої системи з модулем штучного інтелекту. Вона не містить класів для збору, зберігання й обробки історичних даних про попит, погодні умови, промоактивності, сезонність чи інші впливові фактори. Відсутні спеціалізовані структури даних і логічні блоки для роботи з прогнозною моделлю, а також класи, що відповідають за запуск, навчання чи оновлення моделей машинного навчання.

Також у базовій діаграмі немає механізму інтеграції з аналітичним модулем – відсутні сутності, пов’язані з рекомендаціями, прогнозами чи індикаторами попиту. Немає й класів, що могли б представляти результати обчислень або відображати проміжну статистичну інформацію, необхідну для подальшої оптимізації роботи сервісу. У системі бракує контролера або сервісу, який би взаємодіяв із модулем ШІ, генерував прогнози та передавав їх іншим

компонентам.

Крім цього, початкова діаграма не враховує необхідності створення окремих класів для агрегованих та зовнішніх даних, таких як кліматичні показники, календарні події чи історія виконаних доставок у спеціалізованому форматі. Відсутність цих сутностей робить неможливим реалізацію сучасної аналітичної системи, що має працювати з великим обсягом різноманітних вхідних даних.

У зв’язку з цим постає потреба модернізації діаграми класів. Оновлена структура має враховувати інтеграцію модуля штучного інтелекту, включати додаткові класи для прогнозування попиту, зберігання аналітичних даних, роботи з рекомендаціями та розширений функціонал взаємодії між контролерами і сервісами. Подальша розробка оновленої діаграми враховуватиме всі аспекти впровадження інтелектуальної підсистеми та трансформації системи в аналітичну платформу нового покоління.

Модернізована діаграма класів розширює базову структуру системи за рахунок додавання окремого шару аналітики та прогнозування попиту. Усі наявні доменні класи залишаються незмінними за змістом, але до них додаються зв’язки з новими сутностями, які відповідають за зберігання агрегованих даних та результатів обчислень. Оновлена діаграма буде наведена на рисунку 2.13 і відображатиме, як традиційна частина системи інтегрується з модулем штучного інтелекту.

Перш за все до моделі додається клас DemandSummary. Він відображає денну (або тижневу) агреговану статистику попиту на страви: дату, посилання на конкретну страву (Dish або продукт раціону), загальну кількість замовлень, середній обсяг порцій тощо. Між Dish і DemandSummary встановлюється асоціація «один-до-багатьох»: одна страва може мати багато записів попиту за різні дати. Саме на основі об’єктів DemandSummary формується навчальна вибірка для моделі машинного навчання.

Для врахування сезонності та календарного контексту вводиться клас CalendarFactor. Він містить дату, номер тижня, день тижня, ознаку вихідного чи свята, сезон тощо. Клас асоційований із DemandSummary (для кожного запису

попиту може бути прив’язаний відповідний календарний запис), а також використовується в аналітичному модулі як окрема ознака. Це дозволяє моделі враховувати зростання попиту у вихідні, свята або в певні пори року.

Зовнішні умови середовища відображає клас WeatherInfo. Він зберігає дату, середню температуру, тип погодних умов (сонячно, дощ, спека тощо). WeatherInfo також пов’язаний з DemandSummary або безпосередньо з сервісом прогнозування, що дає змогу враховувати вплив погоди на популярність окремих страв. Наприклад, в спеку частіше замовляють салати й легкі страви, а взимку – більш калорійні раціони.

Для опису динаміки цін і акцій додається клас ProductPriceHistory. Він пов’язаний зі стравою (Dish) і містить дату початку дії ціни, її значення, а за потреби – ознаку акційної пропозиції. Ці дані використовуються як одна з ознак у моделі прогнозування, оскільки зниження ціни або поява промоакції безпосередньо впливають на попит.

На рівні бізнес-логіки до діаграми додається клас-сервіс ForecastingService (або DemandForecastService). Він не є контролером користувацького інтерфейсу, а належить до прикладного шару. Основні методи сервісу: BuildDataset(), який завантажує DemandSummary, CalendarFactor, WeatherInfo, ProductPriceHistory і формує з них навчальну вибірку; TrainModel(), що навчає або перенавчає модель; PredictDemand(period) – генерація прогнозу попиту на заданий проміжок часу. У діаграмі це відображається асоціаціями між сервісом та згаданими класами даних.

Окремо виділяється клас RecommendationService, який використовує результати ForecastingService та доменні об’єкти (Dish, IndividualRation, CommonRation). Основні методи цього сервісу: GetRecommendedDishes(user, dateRange) для формування переліку рекомендованих страв конкретному користувачу та GetMenuPlan(dateRange) для генерації рекомендацій з планування меню на період. RecommendationService асоційований із класами Dish, IndividualRation і DemandSummary, оскільки при формуванні рекомендацій враховує як персональні вподобання, так і загальний прогноз попиту.

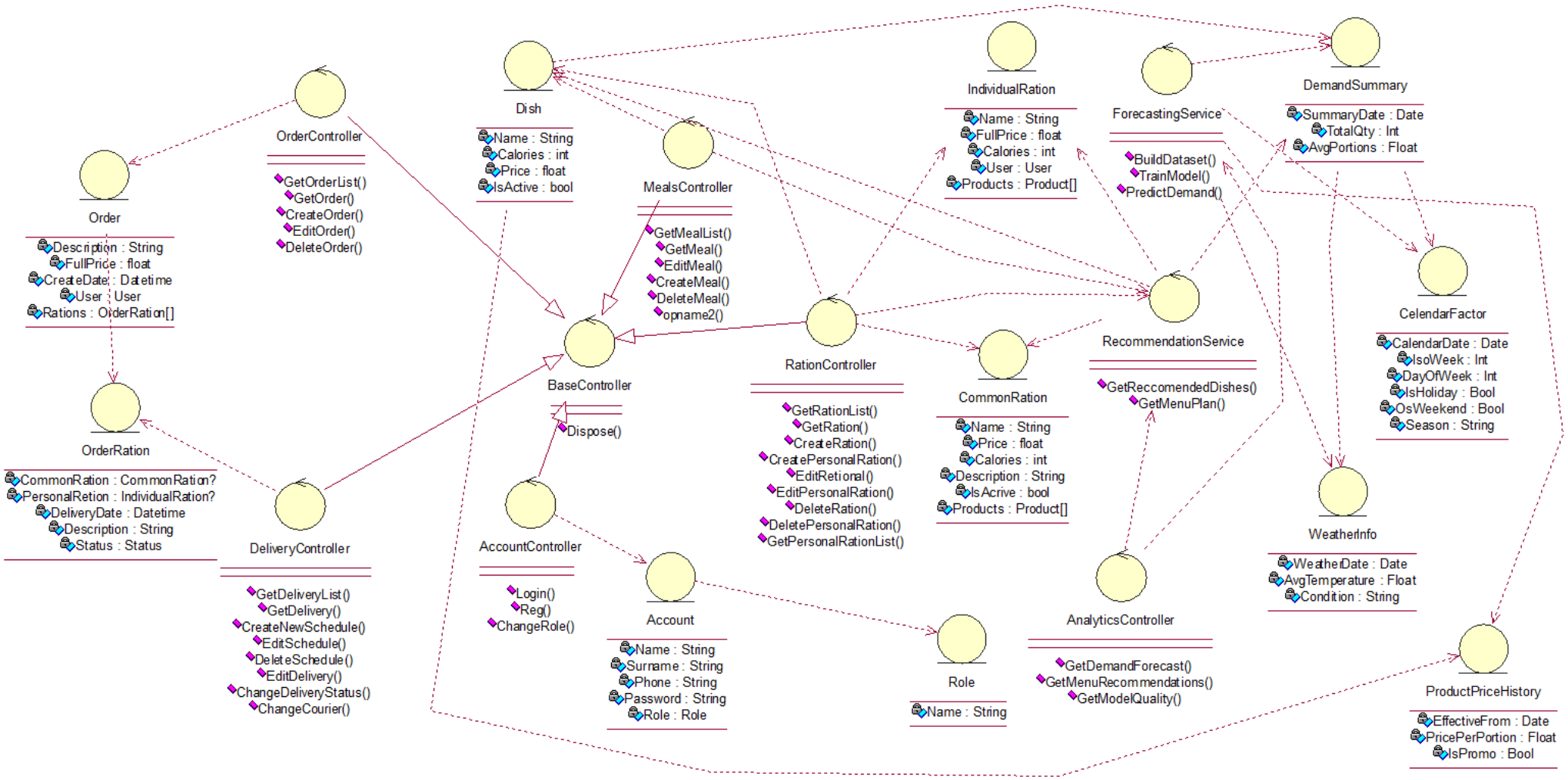

Рисунок 2.13 – Модернізована діаграма класів

Для інтеграції аналітичного модуля з REST-інтерфейсом системи вводиться новий контролер AnalyticsController (або ForecastController). Він наслідується від BaseController так само, як і існуючі контролери. У контролері реалізуються методи GetDemandForecast(), GetMenuRecommendations(), GetModelQuality(), що надають адмін-інтерфейсу можливість переглядати прогноз попиту, рекомендації щодо меню та основні метрики якості моделі. AnalyticsController асоційований із ForecastingService та RecommendationService, виступаючи посередником між веб-клієнтом і модулем ШІ.

Частина існуючих контролерів також отримує додаткові методи. Наприклад, у MealsController може бути доданий метод GetRecommendedMealList(), який звертається до RecommendationService та повертає перелік рекомендованих страв; у RationController – метод GetSuggestedRations(), що використовує той самий сервіс для формування шаблонів раціонів. На діаграмі це відображається через асоціації відповідних контролерів із RecommendationService.

Завдяки таким змінам оновлена діаграма класів показує, як навколо базової предметної моделі (замовлення, раціони, страви, акаунти) формується окремий аналітичний шар із класами даних (DemandSummary, CalendarFactor, WeatherInfo, ProductPriceHistory) та сервісами (ForecastingService, RecommendationService, AnalyticsController). Саме ці елементи забезпечують реалізацію штучного інтелекту, що відповідає за прогнозування попиту та формування рекомендацій, і демонструють логічну інтеграцію інтелектуальної підсистеми у вже існуючу архітектуру системи.

2.5 Розробка основного алгоритму передбачення попиту на страви

Основний алгоритм передбачення попиту на страви в системі доставки індивідуальних раціонів базується на аналізі історичних даних замовлень у поєднанні з зовнішніми факторами, що впливають на поведінку споживачів. На

концептуальному рівні він працює як послідовність етапів: збір даних, попередня обробка, формування ознак, навчання моделі машинного навчання, отримання прогнозу та оцінювання його якості. Результатом роботи алгоритму є набір прогнозних значень попиту на кожну страву в розрізі днів або тижнів, які далі використовуються для планування меню та закупівель.

Перший етап алгоритму пов’язаний зі збором і консолідацією даних із різних джерел. З основної бази системи витягується історія замовлень, інформація про страви та раціони, дані про виконані доставки. Паралельно підключаються таблиці з календарними факторами, погодними умовами, динамікою цін та промоактивностями. На цьому кроці формується проміжний набір записів, де кожен рядок відповідає певній страві у певну дату, а цільовим значенням є кількість замовлень цієї страви за відповідний день.

Далі відбувається попередня обробка отриманих даних. Алгоритм очищує вибірку від некоректних, дубльованих або неповних записів, виконує нормалізацію та перетворення дат у зручні для аналізу формати. Для категоріальних ознак, таких як тип страви, сезон, день тижня чи погодний стан, застосовуються методи кодування, які дозволяють моделі машинного навчання коректно їх інтерпретувати. На цьому ж етапі можуть виконуватися фільтрація аномалій, згладжування різких стрибків попиту та усунення артефактів, пов’язаних із технічними збоями або разовими нехарактерними акціями.

Після очищення даних формується простір ознак, що описує кожне спостереження. До нього входять як внутрішні характеристики страви (калорійність, цінова категорія, приналежність до певного раціону), так і зовнішні чинники: календарні параметри, погодні показники, наявність промоакцій, історичний попит за попередні дні. Для відображення часової динаміки можуть створюватися лагові ознаки, які зберігають інформацію про попит у кілька попередніх днів або тижнів. Завдання цього етапу – перетворити розрізнені дані на компактне й інформативне представлення, придатне для навчання моделі.

Основою алгоритму прогнозування є модель машинного навчання регресійного типу, яка наближає залежність між простором ознак та кількістю

замовлень на страву. Для цього можуть використовуватися ансамблеві методи, такі як Random Forest[2] або градієнтний бустинг (XGBoost[3], LightGBM[4]), які добре працюють із табличними даними та здатні моделювати нелінійні залежності. Модель навчається на історичній вибірці, розбитій на навчальну та валідаційну підмножини, при цьому її параметри підбираються так, щоб мінімізувати похибку прогнозу на відкладених даних.

Процес навчання супроводжується оцінюванням якості моделі за допомогою відповідних метрик, наприклад середньої абсолютної похибки або середньоквадратичної похибки. За необхідності застосовуються методи крос-валідації, регуляризації та підбору гіперпараметрів, що дозволяє уникнути перенавчання та забезпечити узагальнювальну здатність моделі. На цьому ж етапі аналізується важливість ознак, що дає можливість виявити, які саме фактори найбільше впливають на попит на страви.

Після завершення навчання модель інтегрується в інформаційну систему як сервіс прогнозування. При кожному запуску алгоритм отримує актуальні дані за вибраний період: майбутні календарні дні, очікувані погодні умови, заплановані промоакції та поточні ціни на страви. На основі цих даних формується новий набір ознак без цільових значень, який подається на вхід моделі. Виходом є прогнозована кількість замовлень для кожної страви в кожен день прогнозного горизонту.

Отримані прогнозні значення зберігаються у спеціальній таблиці узагальнення попиту та передаються в модуль рекомендацій. На їх основі система генерує пропозиції щодо формування меню, обсягу приготування окремих страв, планування завантаження кухні та логістики доставки. Таким чином, алгоритм передбачення попиту виступає ядром інтелектуальної підсистеми, що перетворює історичні та зовнішні дані на практично корисні управлінські рішення.

Важливою рисою алгоритму є його циклічність: після завершення кожного реального періоду дані про фактичні замовлення автоматично додаються до історичної вибірки. Це дозволяє періодично перенавчати модель, адаптуючи її до змін у поведінці користувачів, нових страв у меню, змін сезонності чи

маркетингової стратегії. Завдяки такому підходу система прогнозування не є статичною, а постійно вдосконалюється разом із розвитком сервісу доставки раціонів.

Схема алгоритму, наведена на рисунку 2.14, відображає повний життєвий цикл алгоритму прогнозування споживчого попиту на страви – від моменту збору даних до формування готового прогнозу та передачі його в модуль рекомендацій. Алгоритм охоплює всі необхідні етапи: інтеграцію джерел інформації, попередню обробку, навчання моделі машинного навчання, перевірку якості, повторне налаштування, а також безпосереднє прогнозування майбутніх значень. Така структура дозволяє забезпечити високу точність, адаптивність і практичну корисність прогнозів у реальній системі доставки раціонів.

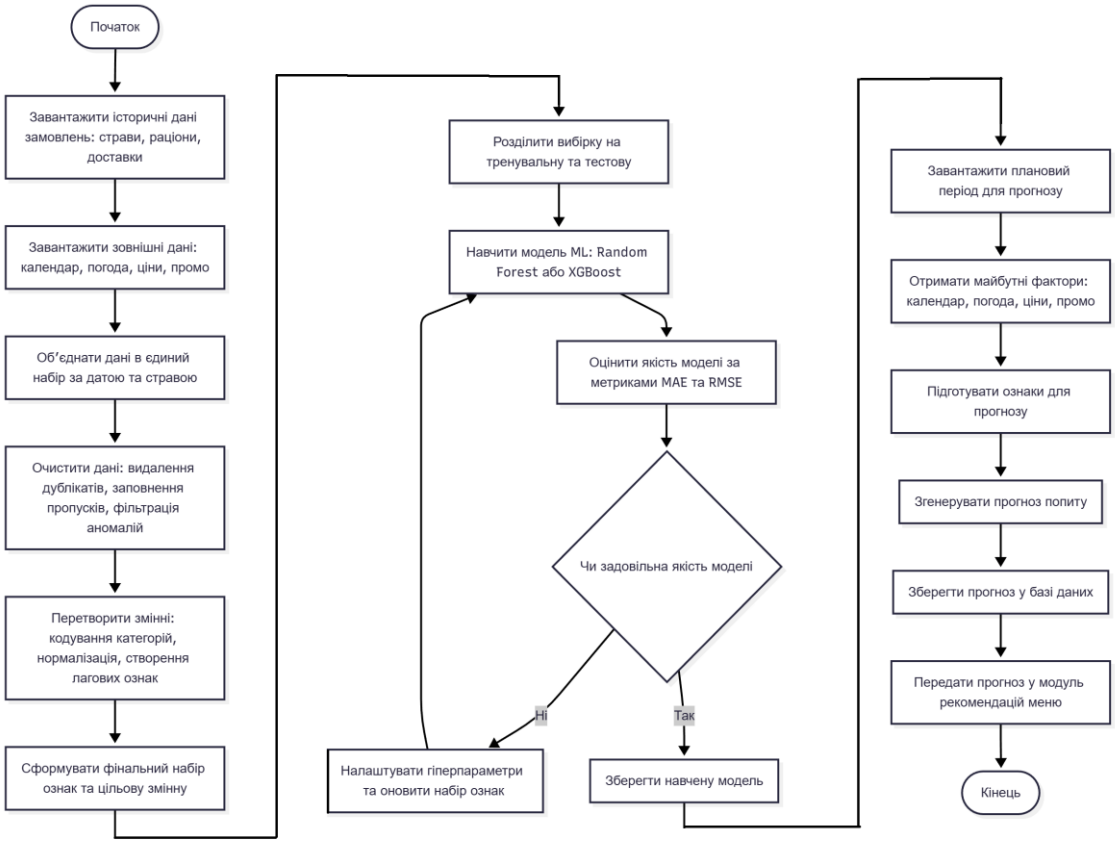

Рисунок 2.14 – Схема основного алгоритму прогнозування попиту

Перший блок схеми присвячений завантаженню історичних даних про замовлення, що включають інформацію про окремі страви, сформовані раціони

та фактичні доставки. Ці дані є фундаментом для побудови моделі попиту, оскільки містять залежності між днями тижня, сезоном, умовами промоакцій та реальною активністю споживачів.

Другий етап включає завантаження зовнішніх факторів, які суттєво впливають на поведінку клієнтів: календарні фактори, погодні умови, актуальні ціни та промо. Наявність таких даних дає змогу моделі враховувати зміни в довгострокових і короткострокових трендах, роблячи прогноз більш життєвим і релевантним.

Далі дані проходять важливу стадію об’єднання в єдиний набір за датою та конкретною стравою. Це дозволяє синхронізувати різнорідні джерела інформації та забезпечити цілісність датасету. Після цього алгоритм переходить до очистки даних, що включає видалення дублікатів, заповнення пропусків та фільтрацію аномальних значень – крок, який критично впливає на якість прогнозу.

Наступний етап – перетворення змінних, де відбувається кодування категоріальних ознак, нормалізація числових значень та створення лагових (зміщених у часі) ознак. Це дозволяє більш точно відобразити динаміку попиту та підвищити чутливість моделі до трендів.

Після формування кінцевого набору ознак алгоритм переходить до блоку навчання моделі, де дані розділяються на тренувальну та тестову вибірки. На тренувальній частині проводиться навчання моделі, наприклад Random Forest або XGBoost – сучасних методів машинного навчання, що демонструють високу ефективність у задачах прогнозування часових рядів зі складними залежностями.

Після навчання модель проходить етап оцінювання якості за метриками MAE та RMSE. Важливо, що рішення про подальші дії ухвалюється через розгалуження: якщо якість є недостатньою, алгоритм повертається до налаштування гіперпараметрів та зміни набору ознак. Цей цикл повторюється до моменту отримання задовільного рівня точності.

У випадку, коли якість моделі остаточно підтверджена, вона зберігається і переходить до експлуатаційного етапу. Далі система завантажує плановий період для прогнозу, отримує майбутні значення факторів і готує відповідні ознаки для передбачення.

Завершальний блок схеми показує, як модель генерує прогноз попиту, після чого прогноз зберігається в базі даних і передається в модуль формування меню. Таким чином, система забезпечує замкнений цикл роботи, де створені прогнози безпосередньо впливають на процеси планування страв та оптимізації ресурсів.

Загалом схема алгоритму відображає чітку послідовність дій, що поєднує методи машинного навчання з операційними потребами системи доставки раціонів, забезпечуючи інтелектуальне та адаптивне планування на основі даних.

Розроблений алгоритм прогнозування попиту на страви забезпечує повний та логічно побудований цикл обробки даних – від їх збору й очищення до формування фінального прогнозу та передачі його в модуль рекомендацій. Поєднання внутрішніх і зовнішніх факторів, використання сучасних моделей машинного навчання та впровадження процедур оцінювання й повторного налаштування дозволяє досягти високої точності прогнозів навіть за умов мінливих ринкових і поведінкових умов.

Завдяки автоматизації ключових етапів та інтеграції моделі у робочі процеси система отримує можливість проактивно планувати меню, оптимізувати запаси та логістику, а також покращувати споживчий досвід. Алгоритм стає основою адаптивної та інтелектуальної підсистеми, що здатна масштабуватися й підлаштовуватися під реальні сценарії функціонування сервісу доставки раціонів.

3 РЕАЛІЗАЦІЯ НАДБУДОВИ ШТУЧНОГО ІНТЕЛЕКТУ ПРОГОЗУВАННЯ ПОПИТУ СТРАВ

3.1 Формування набору історичних даних замовлень

Формування набору історичних даних замовлень є базовим етапом реалізації надбудови штучного інтелекту прогнозування попиту страв, оскільки саме якість і повнота вхідних даних визначають ефективність подальших аналітичних і прогнозних моделей. У межах системи доставки індивідуальних раціонів харчування історичні дані відображають реальну поведінку користувачів, структуру замовлень та динаміку споживчого попиту, що дозволяє моделі машинного навчання виявляти стійкі закономірності та часові залежності.

Ключовим компонентом історичних даних є інформація про сам факт замовлення, зокрема дата його створення та дата доставки. Ці параметри використовуються для формування часових рядів, визначення сезонних коливань попиту, а також для побудови лагових ознак, що відображають залежність поточного попиту від значень у попередні дні або тижні. Часовий аспект даних є критично важливим для коректної роботи прогнозної моделі.

Важливу роль відіграють відомості про склад замовлення, а саме перелік страв та їх кількість. Дані про кількість замовлених порцій конкретної страви використовуються як цільова змінна для навчання моделі, оскільки саме цей показник характеризує фактичний попит. Агрегація таких даних за датою дозволяє сформувати узагальнену картину попиту на кожну страву в часовому розрізі.

Інформація про користувача, зокрема місто доставки, використовується для врахування територіальних особливостей споживання. Попит на одні й ті самі страви може суттєво відрізнятися залежно від регіону, що зумовлено кліматичними умовами, харчовими звичками та рівнем доступності сервісу. Включення цього параметра дозволяє моделі краще адаптуватися до просторової

неоднорідності даних.

Цінові характеристики замовлень також мають суттєвий вплив на поведінку споживачів. Дані про фактичну вартість страви в замовленні, а також про застосовані знижки, використовуються для аналізу цінової чутливості попиту. На основі цих даних модель здатна враховувати реакцію клієнтів на зміну ціни та коригувати прогноз відповідно до економічних факторів.

Окрему групу становлять дані, пов’язані з промоакціями та спеціальними пропозиціями. Факт використання промокоду або надання знижки фіксується в історичних замовленнях і використовується для формування бінарних або кількісних ознак. Це дозволяє враховувати короткострокові сплески попиту, зумовлені маркетинговими активностями.

Важливим елементом історичних даних є статуси замовлень і їхня історія змін. Хоча ці дані безпосередньо не використовуються як ознаки прогнозування попиту, вони дозволяють відфільтровувати скасовані або некоректно оброблені замовлення, забезпечуючи чистоту та достовірність навчальної вибірки. Це знижує рівень шуму в даних і підвищує стабільність моделі.

Додатково в процесі формування набору даних враховується інформація про тип раціону, зокрема чи є він індивідуальним або стандартним. Ця характеристика дозволяє розрізняти персоналізовані замовлення та масові пропозиції, що мають різну природу попиту та різні закономірності споживання.

Для забезпечення коректної роботи алгоритмів машинного навчання історичні дані проходять етап попередньої обробки, що включає видалення дублікатів, усунення аномальних значень та заповнення пропущених полів. Ці операції є необхідними для зменшення впливу випадкових помилок та нетипових ситуацій на результати навчання.

Окрему увагу приділено узгодженню форматів даних і приведенню числових показників до єдиної шкали. Такий підхід дозволяє поєднувати різнорідні джерела інформації в одному наборі ознак і забезпечує коректну роботу алгоритмів, що чутливі до масштабів вхідних параметрів.

Таким чином, сформований набір історичних даних замовлень є комплексним джерелом інформації, яке відображає як кількісні характеристики

попиту, так і фактори, що на нього впливають. Це створює основу для подальшого розширення даних зовнішніми чинниками та побудови ефективної моделі штучного інтелекту для прогнозування споживчого попиту на страви.

Для подальшого навчання та використання моделі машинного навчання необхідно сформувати узгоджений набір історичних даних замовлень безпосередньо з реляційної бази даних системи. З цією метою реалізується окремий програмний модуль, який виконує підключення до СУБД, вибірку даних з основних таблиць замовлень, страв, календаря та пов’язаних довідників, а також їх агрегацію у форматі, придатному для аналітичного опрацювання. У процесі виконання код об’єднує дані за датою доставки та ідентифікатором страви, обчислює фактичну кількість замовлених порцій, фіксує цінові характеристики та супровідні часові атрибути, після чого експортує результат у файл формату JSON. Таким чином формується єдиний структурований набір історичних спостережень, який надалі використовується для побудови ознакового простору та навчання прогнозної моделі. Код формування файлу історичних даних наведено в лістингу 3.1.

Лістинг 3.1 – код процесу формування файлу історичних даних

```
import json
import pyodbc
from datetime import datetime

CONN_STR = (
    «Driver={ODBC Driver 17 for SQL Server};»
    «Server=localhost;»
    «Database=YourDb;»
    «Trusted_Connection=yes;»
)

OUTPUT_PATH = «historical_orders_dataset.json»

def main():
    conn = pyodbc.connect(CONN_STR)
    cursor = conn.cursor()

    query = ««« 
        SELECT
            o.OrderId,
            o.UserId,
            u.City,
            r.DeliveryDate AS DateKey,
            p.ProductId AS DishId,
            p.Name AS DishName,
            op.Count AS Quantity,
            p.CurrentPricePerPortion AS BasePrice,
            r.DiscountPercent,
            c.IsoWeek,
            c.DayOfWeek,
            c.IsWeekend,
            c.IsHoliday,
            c.Season
        FROM [ORDER] o
        INNER JOIN ORDERRATION r ON r.OrderId = o.OrderId
        INNER JOIN USER u ON u.UserId = o.UserId
        INNER JOIN INDIVIDUALRATION ir ON ir.IndividualRationId = r.IndividualRationId
        INNER JOIN INDIVIDUALRATIONPRODUCT op ON op.IndividualRationId = ir.IndividualRationId
        INNER JOIN PRODUCT p ON p.ProductId = op.ProductId
        INNER JOIN CALENDAR c ON c.CalendarDate = r.DeliveryDate
        WHERE r.DeliveryDate IS NOT NULL
```

```
    """

    cursor.execute(query)

    rows = []
    for row in cursor.fetchall():
        rows.append({
            "orderId": str(row.OrderId),
            "userId": str(row.UserId),
            "city": row.City,
            "dateKey": row.DateKey.strftime("%Y-%m-
%d"),
            "dishId": str(row.DishId),
            "dishName": row.DishName,
            "quantity": int(row.Quantity),
            "basePrice": float(row.BasePrice),
            "discountPercent":
float(row.DiscountPercent or 0.0),
            "isoWeek": int(row.IsoWeek),
            "dayOfWeek": int(row.DayOfWeek),
            "isWeekend": bool(row.IsWeekend),
            "isHoliday": bool(row.IsHoliday),
            "season": row.Season
        })

    cursor.close()
    conn.close()

    dataset = {
        "generatedAt":
datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%SZ"),
        "recordsCount": len(rows),
        "orders": rows
    }

    with open(OUTPUT_PATH, "w", encoding="utf-8")
as f:
        json.dump(dataset, f, ensure_ascii=False,
indent=2)

    print(f"Historical dataset saved to
{OUTPUT_PATH}, records: {len(rows)}")

if __name__ == "__main__":
    main()
```

Результатом виконання програмного модуля формування історичних даних є файл у форматі JSON, який містить агреговану інформацію про замовлення, страви, часові характеристики та супровідні параметри, необхідні для подальшого аналітичного опрацювання. Структура сформованого файлу дозволяє однозначно ідентифікувати кожне спостереження за датою та стравою, а також містить числові й категоріальні поля, що надалі використовуються під час побудови ознакового простору та навчання моделі машинного навчання. На рисунку 3.1 наведено фрагмент JSON-файлу з прикладом сформованих історичних даних, який ілюструє результат роботи розробленого програмного коду.

Рисунок 3.1 – файл з набором історичних даних

У результаті виконання етапу формування файлу історичних даних було створено структурований та узгоджений набір інформації, що відображає реальні бізнес-процеси системи доставки індивідуальних раціонів харчування. Отриманий JSON-файл містить ключові параметри замовлень, страв і часових характеристик, що забезпечує повноту та цілісність даних для подальшого аналітичного використання.

Сформований набір даних є зручним для автоматизованого опрацювання, оскільки всі записи приведені до єдиного формату, а значення числових і категоріальних полів мають однозначну інтерпретацію. Це дозволяє мінімізувати витрати часу на додаткову підготовку даних і зосередитися на побудові та вдосконаленні моделей прогнозування попиту.

Використання прямої вибірки даних з реляційної бази забезпечує актуальність і достовірність історичних спостережень, а також спрощує інтеграцію прогнозного модуля з існуючою інформаційною системою. Такий підхід дозволяє в майбутньому автоматизувати процес оновлення навчальних вибірок без необхідності ручного втручання.

Таким чином, сформований файл історичних даних створює надійну основу для наступних етапів реалізації надбудови штучного інтелекту, зокрема для розширення даних зовнішніми факторами, побудови ознакового простору та навчання моделі машинного навчання для прогнозування споживчого попиту на страви.

3.2 Формування набору зовнішних даних

Формування набору зовнішніх даних є необхідним етапом реалізації надбудови штучного інтелекту прогнозування попиту, оскільки історичні дані замовлень не повністю відображають вплив зовнішнього середовища на поведінку споживачів. Залучення додаткових факторів дозволяє розширити ознаковий простір моделі та підвищити її здатність коректно реагувати на зміну умов, які безпосередньо не випливають із внутрішньої логіки системи.

Однією з ключових груп зовнішніх даних є інформація про погодні умови. Погодні параметри, зокрема середня температура повітря та загальний стан погоди, мають прямий вплив на харчові вподобання користувачів. Наприклад, у холодну пору року зростає попит на калорійні та гарячі страви, тоді як у спекотні дні популярнішими стають легкі або охолоджені раціони. Включення погодних даних дозволяє моделі враховувати ці закономірності під час формування прогнозу.

Погодні умови також опосередковано впливають на частоту замовлень та загальний обсяг попиту. Негативні погодні явища можуть стимулювати використання сервісу доставки, тоді як сприятлива погода, навпаки, зменшує кількість онлайн-замовлень. Тому інформація про стан погоди використовується як важлива контекстна ознака для моделі машинного навчання.

Не менш значущим зовнішнім фактором є календарні дані, зокрема інформація про вихідні, святкові дні та сезонність. У святкові періоди спостерігається зміна структури попиту, зростає кількість замовлень, а також

змінюється пріоритет окремих страв. Календарні атрибути дозволяють моделі коректно ідентифікувати такі періоди та враховувати їхній вплив на прогнозовані значення.

Дані про сезонність використовуються для відображення довгострокових циклів попиту, які повторюються з року в рік. Розподіл дат за сезонами дозволяє моделі формувати узагальнені патерни споживання, характерні для певного часу року, навіть у випадку обмеженої кількості історичних спостережень.

Окрему категорію становлять дані про святкові та подієві дні. Навіть за відсутності офіційного вихідного дня локальні або маркетингові події можуть істотно впливати на попит. Фіксація таких подій у зовнішньому наборі даних дозволяє моделі враховувати нетипові коливання попиту, що не пов’язані з регулярною сезонністю.

Цінові дані є ще одним важливим зовнішнім фактором, який безпосередньо впливає на споживчу поведінку. Історія змін цін на страви або інгредієнти використовується для аналізу реакції попиту на підвищення чи зниження вартості. Це дозволяє моделі враховувати еластичність попиту та більш точно прогнозувати кількість замовлень у разі зміни цінової політики.

Дані про промоакції та знижки формують окрему групу ознак, що відображають короткострокові стимули для користувачів. Факт проведення промоакції, її тривалість і розмір знижки використовуються для пояснення різких зростань попиту, які не можуть бути обґрунтовані лише історичними або сезонними факторами.

Інтеграція цінових і промо-даних дозволяє моделі машинного навчання розрізняти органічне зростання попиту та штучно стимульовані сплески, спричинені маркетинговими активностями. Це підвищує стійкість прогнозів і зменшує ризик перенавчання на нетипових подіях.

Усі зовнішні дані формуються у вигляді окремих структурованих наборів і синхронізуються з історичними замовленнями за датою та іншими ключовими атрибутами. Такий підхід забезпечує коректне поєднання внутрішніх і зовнішніх факторів у єдиному ознаковому просторі.

Таким чином, сформований набір зовнішніх даних створює контекст для

інтерпретації історичних замовлень і дозволяє моделі прогнозування попиту враховувати комплексний вплив погодних умов, календарних подій, цінових змін та промоактивностей, що суттєво підвищує точність і практичну цінність отриманих прогнозів.

Для формування набору зовнішніх даних погодних умов використовується програмний код, який звертається до зовнішнього метеорологічного сервісу та отримує прогнозні або історичні показники для заданого періоду і населених пунктів. У процесі виконання програма формує HTTP-запити до API, обробляє відповіді у форматі JSON, виділяє необхідні параметри, зокрема середню температуру та загальний стан погоди, після чого зберігає результати у структурований файл для подальшого використання в системі прогнозування. Код формування файлу зовнішніх даних погодних умов наведено в лістингу 3.2.

Лістинг 3.2 – код формування файлу зовнішних даних погодних умов

```
import requests
import json
from datetime import date, timedelta

API_KEY = «YOUR_WEATHER_API_KEY»
CITY = «Kyiv»
START_DATE = date(2025, 11, 1)
DAYS = 14
OUTPUT = «weather_external.json»

data = []
for i in range(DAYS):
    d = START_DATE + timedelta(days=i)
    url = f»https://api.open-meteo.com/v1/forecast?latitude=50.45&longitude=30.52&daily=temperature_2m_mean,weathercode&start_date={d}&end_date={d}»
    r = requests.get(url)
    resp = r.json()
    data.append({
        «dateKey»: d.isoformat(),
        «city»: CITY,
        «avgTemperature»: resp[«daily»][«temperature_2m_mean»][0],
        «condition»: resp[«daily»][«weathercode»][0]
    })

with open(OUTPUT, «w», encoding=«utf-8») as f:
    json.dump(data, f, ensure_ascii=False, indent=2)
```

Для отримання календарних даних використовується код, який звертається до відкритого календарного сервісу або API державних свят, отримує перелік святкових і вихідних днів та поєднує їх із базовими часовими атрибутами, такими як день тижня та номер тижня в році. Сформований файл дозволяє однозначно визначити, чи належить конкретна дата до святкових або вихідних, що надалі використовується як важливий фактор прогнозування попиту. Код формування файлу зовнішніх даних календарних подій наведено в лістингу 3.3.

Лістинг 3.3 – код формування файлу зовнішних даних календарних подій

```
import requests
import json
from datetime import date, timedelta

START_DATE = date(2025, 11, 1)
DAYS = 14
OUTPUT = «calendar_external.json»

holidays_api = «https://date.nager.at/api/v3/PublicHolidays/2025/UA»
holidays = requests.get(holidays_api).json()
holiday_dates = {h[«date»]: h[«localName»] for h in holidays}

calendar = []
for i in range(DAYS):
    d = START_DATE + timedelta(days=i)
    calendar.append({
        «dateKey»: d.isoformat(),
        «dayOfWeek»: d.isoweekday(),
        «isWeekend»: d.isoweekday() >= 6,
        «isHoliday»: d.isoformat() in holiday_dates,
        «holidayName»: holiday_dates.get(d.isoformat())
    })

with open(OUTPUT, «w», encoding=«utf-8») as f:
    json.dump(calendar, f, ensure_ascii=False, indent=2)
```

Формування зовнішніх даних щодо цін виконується за допомогою програмного коду, який звертається до внутрішнього або партнерського сервісу ціноутворення та отримує інформацію про актуальні або заплановані зміни вартості страв. Отримані дані агрегуються за датою та ідентифікатором страви і зберігаються у вигляді історії цін, що дозволяє враховувати економічний вплив на попит у прогнозній моделі. Код формування файлу зовнішніх даних цін наведено в лістингу 3.4.

Лістинг 3.4 – код формування файлу зовнішних даних цін

```
import json
from datetime import date, timedelta

START_DATE = date(2025, 11, 1)
DAYS = 14
DISHES = [«dish-1», «dish-2», «dish-3»]
OUTPUT = «prices_external.json»

prices = []
for dish in DISHES:
    base_price = 100
    for i in range(DAYS):
        d = START_DATE + timedelta(days=i)
        prices.append({
            «dateKey»: d.isoformat(),
            «dishId»: dish,
            «price»: base_price + i * 2
        })

with open(OUTPUT, «w», encoding=«utf-8») as f:
    json.dump(prices, f, ensure_ascii=False, indent=2)
```

Для формування даних про промоакції використовується код, який отримує інформацію з маркетингового сервісу або CRM-системи про активні та заплановані акції, їхню тривалість і розмір знижок. Після обробки ці дані

зберігаються у JSON-файл, що дозволяє моделі машинного навчання враховувати вплив короткострокових стимулів на споживчий попит. Код формування файлу зовнішніх даних промоакцій наведено в лістингу 3.5.

Лістинг 3.5 – код формування файлу зовнішних даних промоакцій

```
import json
from datetime import date

promos = [
    {
        «promoCode»: «AUTUMN10»,
        «startDate»: «2025-11-01»,
        «endDate»: «2025-11-07»,
        «discountPercent»: 10
    },
    {
        «promoCode»: «WEEKEND15»,
        «startDate»: «2025-11-08»,
        «endDate»: «2025-11-09»,
        «discountPercent»: 15
    }
]

with open(«promo_external.json», «w», encoding=«utf-8») as f:
    json.dump(promos, f, ensure_ascii=False, indent=2)
```

У результаті виконання програмного коду формування зовнішніх даних погодних умов створюється JSON-файл, який містить значення середньої температури та загального стану погоди для кожної дати прогнозного періоду. Структура файлу дозволяє однозначно зіставити погодні показники з відповідними датами та населеними пунктами, що забезпечує можливість їх подальшого об’єднання з історичними даними замовлень і використання як вхідних ознак прогнозної моделі. На рисунку 3.2 наведено фрагмент JSON-файлу з даними про погодні умови, сформованого в результаті роботи відповідного програмного модуля.

Результатом формування календарних зовнішніх даних є JSON-файл, що містить інформацію про день тижня, наявність вихідного або святкового дня, а також назву свята за потреби. Такий формат представлення даних дозволяє швидко визначати календарний контекст кожної дати та використовувати ці характеристики як фактори, що впливають на споживчий попит. На рисунку 3.3 показано приклад сформованого JSON-файлу з календарними даними, який ілюструє структуру та наповнення цього набору.

У процесі формування зовнішніх даних щодо цін створюється JSON-файл, який відображає зміну вартості страв у часі для прогнозного періоду. Дані у файлі згруповані за датою та ідентифікатором страви, що дозволяє

безпосередньо поєднувати їх із прогнозними записами та враховувати ціновий фактор під час генерації попиту. На рисунку 3.4 наведено фрагмент JSON-файлу з даними про ціни, сформованого для подальшого використання в системі прогнозування.

Формування зовнішніх даних про промоакції завершується створенням JSON-файлу, який містить інформацію про активні та заплановані маркетингові кампанії, їхню тривалість і розмір знижок. Такий файл дозволяє моделі машинного навчання враховувати вплив промоактивностей на короткострокові коливання попиту та коректно інтерпретувати нетипові сплески замовлень. На рисунку 3.5 наведено приклад сформованого JSON-файлу з даними про промоакції, який демонструє результат роботи відповідного програмного коду.

Рисунок 3.2 – зовнішні дані погодних умов

```
{
  "generatedAt": "2025-12-16T08:47:24Z",
  "schemaVersion": "1.0",
  "notes": " Join key: dateKey.",
  "startDate": "2024-03-01",
  "endDate": "2025-10-31",
  "calendar": [
    {
      "calendarId": "acecef00-21fa-43a8-a649-902dfd0fc2aa",
      "dateKey": "2024-03-01",
      "isoWeek": 9,
      "dayOfWeek": 5,
      "isWeekend": false,
      "isHoliday": false,
      "holidayName": null,
      "season": "Spring",
      "eventName": null
    },
    {
      "calendarId": "6018c6ea-2c1c-41c2-907e-38b77d7c6b50",
      "dateKey": "2024-03-02",
```


Рисунок 3.3 – зовнішні дані календарних подій


```
{
  "generatedAt": "2025-12-16T08:43:08Z",
  "schemaVersion": "1.0",
  "notes": "Join keys: productId + effectiveFrom (latest <= order.dateKey).",
  "startDate": "2024-03-01",
  "endDate": "2025-11-01",
  "prices": [
    {
      "productId": "108f3899-04b4-4a00-89d4-5c85e3be5757",
      "effectiveFrom": "2024-03-01",
      "pricePerPortion": 207.68
    },
    {
      "productId": "108f3899-04b4-4a00-89d4-5c85e3be5757",
      "effectiveFrom": "2024-03-19",
      "pricePerPortion": 204.38
    },
```


Рисунок 3.4 – зовнішні дані цін

Рисунок 3.5 – зовнішні дані промо

У результаті формування файлів зовнішніх даних було створено декілька узгоджених наборів інформації, які доповнюють історичні дані замовлень контекстними факторами зовнішнього середовища. Отримані JSON-файли мають чітку структуру та містять лише ті параметри, що безпосередньо впливають на формування споживчого попиту, що забезпечує їхню практичну цінність для подальшого використання.

Дані про погодні умови дозволяють враховувати вплив кліматичних факторів на вибір страв і загальну активність користувачів. Інтеграція температурних показників і загального стану погоди розширює можливості моделі прогнозування, оскільки вона отримує додатковий контекст для пояснення сезонних і короткострокових коливань попиту.

Сформовані календарні дані забезпечують коректне врахування вихідних і святкових днів, а також інших часових характеристик. Завдяки цьому модель здатна розрізняти звичайні робочі дні та періоди з підвищеною або зниженою

активністю споживачів, що підвищує точність прогнозів у часовому розрізі.

Цінові дані та історія змін вартості страв створюють основу для аналізу цінової чутливості попиту. Наявність такої інформації дозволяє прогнозній моделі адекватно реагувати на зміну цінової політики та враховувати економічні чинники під час формування прогнозів.

Файли з даними про промоакції відображають вплив маркетингових заходів на поведінку користувачів і дозволяють коректно інтерпретувати нетипові сплески попиту. Це зменшує ризик викривлення результатів навчання та сприяє формуванню більш стійкої та узагальненої моделі.

Таким чином, сформовані файли зовнішніх даних створюють повноцінний інформаційний контекст для роботи модуля штучного інтелекту. Їх використання у поєднанні з історичними даними замовлень дозволяє побудувати більш точну та адаптивну систему прогнозування споживчого попиту на страви.

3.3 Формування тренувальної та тестової вибірки

Для переходу від розрізнених джерел інформації до єдиного навчального набору даних необхідно виконати етап інтеграції історичних замовлень із зовнішніми факторами, що впливають на попит. На цьому етапі реалізується програмний модуль, який завантажує сформовані JSON-файли, виконує їхнє зіставлення за ключовими атрибутами, насамперед за датою та ідентифікатором страви, і формує узгоджену структуру записів для подальшого використання в машинному навчанні. Код, який здійснює об’єднання наборів даних, очищення від дублікатів і видалення аномальних спостережень, наведено в лістингу 3.6.

Лістинг 3.6 – код об’єднання наборів даних

```
import json
from bisect import bisect_right
from datetime import datetime

PATHS = {
    «orders»: «historical_orders_5000.json»,
    «weather»: «weather_daily_2024-03-01_2025-10-31.json»,
    «prices»: «product_price_history.json»,
    «promos»: «promo_campaigns.json»,
    «calendar»: «calendar_holidays_2024-03-01_2025-10-31.json»,
```

```
}

OUT_PATH = «merged_dataset_by_date_dish.json»

def load_json(path: str):
    with open(path, «r», encoding=«utf-8») as f:
        return json.load(f)

def safe_float(x, default=None):
    try:
        if x is None:
            return default
        return float(x)
    except Exception:
        return default

def safe_int(x, default=None):
    try:
        if x is None:
            return default
        return int(x)
    except Exception:
        return default

def clamp(v, lo, hi):
    return lo if v < lo else hi if v > hi else v

def build_weather_index(weather_json):
    # key: (dateKey, city) -> weather record
    idx = {}
    for w in weather_json[«weather»]:
        key = (w[«dateKey»], w[«city»])
        idx[key] = w
    return idx

def build_calendar_index(calendar_json):
    # key: dateKey -> calendar record
    idx = {}
    for c in calendar_json[«calendar»]:
        idx[c[«dateKey»]] = c
    return idx

def build_promo_index(promos_json):
    # promoCode -> list of windows
    idx = {}
    for p in promos_json[«promos»]:
       idx.setdefault(p[«promoCode»], []).append(p)

    # sort windows by startDate for faster check
    for code in idx:
       idx[code].sort(key=lambda x: x[«startDate»])
    return idx

def promo_active(promo_index, promo_code, date_key):
    ««««Return promo record if promo_code active at date_key else None.»»««
    if not promo_code:
        return None
    arr = promo_index.get(promo_code)
    if not arr:
        return None
    # linear scan is OK for small arrays; keep it simple
    for p in arr:
            if p[«startDate»] <= date_key <= p[«endDate»]:
            return p
    return None

def build_price_asof_index(price_json):
    ««««
    Build structure:
      prices_by_product[productId] = {
          «dates»: [effectiveFrom sorted],
          «prices»: [pricePerPortion aligned]
      }
    ««««
    tmp = {}
    for rec in price_json[«prices»]:
        pid = rec[«productId»]
                                tmp.setdefault(pid, []).append((rec[«effectiveFrom»], safe_float(rec[«pricePerPortion»], None)))

    prices_by_product = {}
    for pid, rows in tmp.items():
        rows = sorted(rows, key=lambda x: x[0])
        dates = [r[0] for r in rows]
        prices = [r[1] for r in rows]
         prices_by_product[pid] = {«dates»: dates, «prices»: prices}
    return prices_by_product

def asof_price(prices_by_product, product_id, date_key):
      ««««Get last price where effectiveFrom <= date_key. Return None if not found.»»««
    p = prices_by_product.get(product_id)
    if not p:
        return None
    dates = p[«dates»]
    i = bisect_right(dates, date_key) - 1
    if i < 0:
        return None
    return p[«prices»][i]

def is_anomalous_record(r):
```

```
    «««
    Basic anomaly rules (tune as needed):
    - totalQty must be 1..30
    - avgTemperature in [-40, 45]
    - unitPrice >= 0
    - priceAtDate >= 0
    - discountPercent 0..80
    »»»
    if r[«totalQty»] is None or r[«totalQty»] <= 0
or r[«totalQty»] > 30:
        return True
      if r[«avgTemperature»] is not None and
(r[«avgTemperature»] < -40 or r[«avgTemperature»] >
45):
        return True
     if r[«avgUnitPricePaid»] is not None and
r[«avgUnitPricePaid»] <= 0:
        return True
       if r[«priceAtDate»] is not None and
r[«priceAtDate»] <= 0:
        return True
     if r[«discountPercent»] is not None and
(r[«discountPercent»] < 0 or r[«discountPercent»] >
80):
        return True
    return False

def main():
    orders_json = load_json(PATHS[«orders»])
    weather_json = load_json(PATHS[«weather»])
    prices_json = load_json(PATHS[«prices»])
    promos_json = load_json(PATHS[«promos»])
    calendar_json = load_json(PATHS[«calendar»])

    weather_idx = build_weather_index(weather_json)
                          calendar_idx            =
build_calendar_index(calendar_json)
    promo_idx = build_promo_index(promos_json)
                           prices_asof            =
build_price_asof_index(prices_json)

    # ---- 1) Flatten orders to order-item rows and
de-duplicate ----
    # Dedup key: (orderId, dishId)
    seen = set()
    flat_rows = []
    for o in orders_json[«orders»]:
        order_id = o[«orderId»]
        date_key = o[«dateKey»]
        city = o.get(«city»)

                             discount_percent      =
safe_float(o.get(«discountPercent»), 0.0)
        discount_percent = clamp(discount_percent,
0.0, 80.0)  # clamp

        promo_code = o.get(«promoCode»)

        for it in o.get(«items», []):
            dish_id = it[«dishId»]
            key = (order_id, dish_id)
            if key in seen:
                continue
            seen.add(key)

            qty = safe_int(it.get(«qty»), 0)
                                      unit_paid     =
safe_float(it.get(«unitPricePaid»), None)

            # basic cleanup
            if qty <= 0:
                continue
            qty = clamp(qty, 1, 30)  # cap extreme
outliers

            if unit_paid is not None and unit_paid
<= 0:
                unit_paid = None

            flat_rows.append({
                «orderId»: order_id,
                «userId»: o.get(«userId»),
                «city»: city,
                «dateKey»: date_key,
                «dishId»: dish_id,
                «qty»: qty,
                «unitPricePaid»: unit_paid,
                «promoCode»: promo_code,
                «discountPercent»: discount_percent
            })

    # ---- 2) Aggregate by dateKey + dishId (+ city)
-> demand rows ----
    # If you want strictly «by date and dish», remove
city from the key.
    agg = {}
    for r in flat_rows:
        k = (r[«dateKey»], r[«dishId»], r[«city»])
        a = agg.get(k)
        if a is None:
            agg[k] = {
                «dateKey»: r[«dateKey»],
                «dishId»: r[«dishId»],
                «city»: r[«city»],
                «totalQty»: 0,
                «ordersCount»: 0,
                «sumUnitPricePaid»: 0.0,
                «unitPricePaidCount»: 0,
```

```
                «promoOrdersCount»: 0,
                «avgDiscountPercentSum»: 0.0,
                «avgDiscountPercentCount»: 0
            }
            a = agg[k]

        a[«totalQty»] += r[«qty»]
        a[«ordersCount»] += 1

        if r[«unitPricePaid»] is not None:
                        a[«sumUnitPricePaid»]   +=
r[«unitPricePaid»]
            a[«unitPricePaidCount»] += 1

        if r[«promoCode»]:
            a[«promoOrdersCount»] += 1

        if r[«discountPercent»] is not None:
                    a[«avgDiscountPercentSum»]  +=
r[«discountPercent»]
            a[«avgDiscountPercentCount»] += 1

    merged_rows = []
    for k, a in agg.items():
        date_key, dish_id, city = k

        avg_unit_paid = None
        if a[«unitPricePaidCount»] > 0:
                                avg_unit_paid    =
round(a[«sumUnitPricePaid»]                        /
a[«unitPricePaidCount»], 2)

        avg_discount = None
        if a[«avgDiscountPercentCount»] > 0:
                                 avg_discount    =
round(a[«avgDiscountPercentSum»]                   /
a[«avgDiscountPercentCount»], 2)

        # ---- 3) Join external data ----
        cal = calendar_idx.get(date_key)
         w = weather_idx.get((date_key, city)) if
city else None

        promo_code = None
        promo_disc = None
        promo_active_flag = False

          # We can approximate «promo active» by
checking if any promo was used in orders that day
for that dish,
        # and validate it against promo windows.
        if a[«promoOrdersCount»] > 0:
            # choose a «representative» promo code
by looking up any used promo from flat rows
            # simplest: leave it as «unknown-used»
if you want stricter, keep per-order promo
aggregation
            promo_active_flag = True

        # Price as-of (external «current» price at
that date for dish)
          price_at_date  =  asof_price(prices_asof,
dish_id, date_key)
        if price_at_date is not None:
            price_at_date = round(price_at_date, 2)

        row = {
            # keys
            «dateKey»: date_key,
            «dishId»: dish_id,
            «city»: city,

            # demand targets/features
            «totalQty»: a[«totalQty»],
            «ordersCount»: a[«ordersCount»],
            «avgUnitPricePaid»: avg_unit_paid,
            «priceAtDate»: price_at_date,
                                «promoOrdersCount»:
a[«promoOrdersCount»],
            «discountPercent»: avg_discount,

            # calendar features
             «isoWeek»: cal[«isoWeek»] if cal else
None,
              «dayOfWeek»: cal[«dayOfWeek»] if cal
else None,
              «isWeekend»: cal[«isWeekend»] if cal
else None,
              «isHoliday»: cal[«isHoliday»] if cal
else None,
              «holidayName»: cal.get(«holidayName»)
if cal else None,
            «season»: cal.get(«season») if cal else
None,
           «eventName»: cal.get(«eventName») if cal
else None,

            # weather features
              «avgTemperature»: w[«avgTemperature»]
if w else None,
             «condition»: w[«condition»] if w else
None,
        }

        # ---- 4) Remove anomalies ----
        if is_anomalous_record(row):
            continue
```

```
        merged_rows.append(row)

    # ---- 5) Final de-dup by (dateKey, dishId, city) ----
    unique = {}
    for r in merged_rows:
        k = (r[«dateKey»], r[«dishId»], r[«city»])
        # keep first (already aggregated)
        if k not in unique:
            unique[k] = r
    merged_rows = list(unique.values())

    # Sort for convenience
    merged_rows.sort(key=lambda x: (x[«dateKey»], x[«dishId»], x[«city»]))

    out = {
                                    «generatedAt»: datetime.utcnow().strftime(«%Y-%m-%dT%H:%M:%SZ»),
        «schemaVersion»: «1.0»,
        «notes»: «Merged dataset by dateKey + dishId (+ city). Includes calendar, weather, as-of price, and basic promo signals. Anomalies removed and duplicates dropped.»,
        «joinKeys»: [«dateKey», «dishId», «city»],
        «rows»: merged_rows
    }

    with open(OUT_PATH, «w», encoding=«utf-8») as f:
            json.dump(out, f, ensure_ascii=False, indent=2)

    print(f»Saved: {OUT_PATH}»)
    print(f»Rows: {len(merged_rows)}»)

if __name__ == «__main__»:
    main()
```

Розглянутий код працює за принципом послідовного перетворення даних до єдиного формату, орієнтованого на часові ряди попиту. Спочатку завантажуються всі необхідні файли, після чого будується проміжна індексація для швидкого доступу до погодних, календарних та цінових значень у прив’язці до дати. Далі історичні замовлення переводяться у «плоский” формат, де кожна позиція замовлення представляє окремий запис по страві, що дозволяє коректно агрегувати фактичний попит. Після агрегації формується набір спостережень за ключем «дата–страва” з додаванням зовнішніх факторів, а також виконується очищення даних від повторів та нетипових значень, які можуть спотворювати навчання моделі. У підсумку результуючий набір зберігається у JSON-файл, що використовується як єдина база для подальшого формування ознак і розподілу вибірки.

Результатом виконання коду є об’єднаний JSON-файл із записами, у яких поєднані значення попиту, часові атрибути, погодні характеристики, цінові параметри та промоознаки, приведені до узгодженого вигляду. Такий файл є безпосереднім входом для наступного етапу побудови ознакового простору та розділення даних на тренувальну і тестову підвибірки. На рисунку 3.6 наведено фрагмент сформованого JSON-файлу, що ілюструє результат об’єднання та попереднього очищення даних.

Рисунок 3.6 – Фрагмент сформованого JSON-файлу, що ілюструє результат об’єднання

Після формування об’єднаного та очищеного набору даних наступним кроком є підготовка інформації безпосередньо до використання в алгоритмах машинного навчання. Для цього реалізується програмний код, який виконує перетворення початкових змінних у формат, придатний для навчання прогнозної моделі, з урахуванням особливостей часових рядів попиту. Код, що здійснює побудову фінального ознакового простору та розподіл вибірки, наведено в лістингу 3.7.

Лістинг 3.7 – код побудови фінального простору

```
from __future__ import annotations

import argparse
import json
from dataclasses import dataclass
from pathlib import Path
from typing import Any, Dict, List, Tuple, Optional

import numpy as np
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split

@dataclass
class Config:
    input_path: Path
    out_train_path: Path
    out_test_path: Path

    date_col: str = «date»          # дата (день)
    product_col: str = «productId»  # ідентифікатор страви/продукту
    target_col: str = «totalQty»    # ціль: кількість

    lags: Tuple[int, ...] = (1, 7, 14)
    ma_windows: Tuple[int, ...] = (7, 14)

    test_size: float = 0.2
    random_state: int = 42

    max_qty: int = 5000

def _try_parse_json(text: str) -> Any:
    return json.loads(text)

def _read_ndjson(path: Path) -> List[Dict[str, Any]]:
    ««««Читає NDJSON: кожен рядок – валідний JSON-об'єкт.»»»»
    rows: List[Dict[str, Any]] = []
    with path.open(«r», encoding=«utf-8») as f:
        for line in f:
            line = line.strip()
            if not line:
                continue
            obj = json.loads(line)
            if isinstance(obj, dict):
                rows.append(obj)
            else:
                continue
    return rows

def _find_first_list_of_dicts(obj: Any) -> Optional[List[Dict[str, Any]]]:
    ««««
    Рекурсивно шукає у вкладеному dict/list перший елемент виду list[dict].
    Повертає саме список dict'ів або None.
    ««««
    if isinstance(obj, list):
        if obj and all(isinstance(x, dict) for x in obj):
            return obj
        for x in obj:
            found = _find_first_list_of_dicts(x)
            if found is not None:
                return found
        return None

    if isinstance(obj, dict):
        for key in («data», «rows», «items», «records», «result», «values»):
            if key in obj:
                found = _find_first_list_of_dicts(obj[key])
                if found is not None:
                    return found

        for v in obj.values():
            found = _find_first_list_of_dicts(v)
            if found is not None:
                return found
        return None

    return None

def read_json_to_df(path: Path) -> pd.DataFrame:
    ««««
    Підтримує:
    - list[dict]
    - dict з вкладеним списком dict (data/rows/items/records/... або будь-де)
    - NDJSON (по рядках)
    ««««
    raw_text = path.read_text(encoding=«utf-8»).strip()

    try:
        obj = _try_parse_json(raw_text)
        rows = None
```

```
        if isinstance(obj, list) and (not obj or
all(isinstance(x, dict) for x in obj)):
            rows = obj
        else:
            rows = _find_first_list_of_dicts(obj)

        if rows is None:
            raise ValueError(
                «Не вдалося знайти list[dict] у
JSON. «
                «Файл схожий на JSON, але структура
не містить масиву записів.»
            )

        return pd.DataFrame(rows)

    except json.JSONDecodeError:
        rows = _read_ndjson(path)
        if not rows:
            raise ValueError(«Файл не є валідним
JSON/NDJSON або не містить об'єктів-рядків.»)
        return pd.DataFrame(rows)

def basic_cleaning(df: pd.DataFrame, cfg: Config) -
> pd.DataFrame:
                        df[cfg.date_col]          =
pd.to_datetime(df[cfg.date_col],
errors=«coerce»).dt.date
                        df[cfg.date_col]          =
pd.to_datetime(df[cfg.date_col])

        df    =    df.dropna(subset=[cfg.date_col,
cfg.product_col, cfg.target_col])

                    df[cfg.product_col]          =
df[cfg.product_col].astype(str)
                    df[cfg.target_col]          =
pd.to_numeric(df[cfg.target_col], errors=«coerce»)
    df = df.dropna(subset=[cfg.target_col])

    df  =  df.drop_duplicates(subset=[cfg.date_col,
cfg.product_col], keep=«last»)

       df   =   df[(df[cfg.target_col]   >=   0)   &
(df[cfg.target_col] <= cfg.max_qty)]

    return df

def   add_time_features(df:   pd.DataFrame,   cfg:
Config) -> pd.DataFrame:
                        df[«dayOfWeek»]          =
df[cfg.date_col].dt.dayofweek.astype(int)
                            df[«month»]          =
df[cfg.date_col].dt.month.astype(int)
                        df[«weekOfYear»]          =
df[cfg.date_col].dt.isocalendar().week.astype(int)
       df[«isWeekend»]   =   (df[«dayOfWeek»]   >=
5).astype(int)
    return df

def add_lag_and_ma_features(df: pd.DataFrame, cfg:
Config) -> pd.DataFrame:
        df    =    df.sort_values([cfg.product_col,
cfg.date_col]).reset_index(drop=True)

    for lag in cfg.lags:
        col_name = f»lag_{lag}»
                                df[col_name]      =
df.groupby(cfg.product_col)[cfg.target_col].transf
orm(lambda s: s.shift(lag))

    for w in cfg.ma_windows:
        col_name = f»ma_{w}»
                                df[col_name]      =
df.groupby(cfg.product_col)[cfg.target_col].transf
orm(
            lambda s: s.shift(1).rolling(window=w,
min_periods=1).mean()
        )

                            df[«std_7»]           =
df.groupby(cfg.product_col)[cfg.target_col].transf
orm(
           lambda  s:  s.shift(1).rolling(window=7,
min_periods=2).std()
    )

    return df

def finalize_dataset(df: pd.DataFrame, cfg: Config)
-> pd.DataFrame:
    lag_cols = [f»lag_{l}» for l in cfg.lags]
    ma_cols = [f»ma_{w}» for w in cfg.ma_windows]

    for c in lag_cols:
        if c in df.columns:
            df[c] = df[c].fillna(0)

    for c in ma_cols:
        if c in df.columns:
                                    df[c]     =
df.groupby(cfg.product_col)[c].transform(lambda  s:
s.fillna(s.median()))
            df[c] = df[c].fillna(df[c].median())

    if «std_7» in df.columns:
        df[«std_7»] = df[«std_7»].fillna(0)
```

```
    for col in df.columns:
        if col == cfg.target_col:
            continue
        if df[col].dtype.kind in {«i», «u», «f»}:
             df[col] = df[col].replace([np.inf, -
np.inf], np.nan)
                                    df[col]     =
df[col].fillna(df[col].median())

    return df

def build_transformer(
    df: pd.DataFrame,
    cfg: Config,
)    ->    Tuple[ColumnTransformer,    List[str],
List[str]]:
    exclude = {cfg.target_col, cfg.date_col}
    feature_cols = [c for c in df.columns if c not
in exclude]

    cat_cols: List[str] = []
    num_cols: List[str] = []

    for c in feature_cols:
        if df[c].dtype == «object»:
            cat_cols.append(c)
        elif str(df[c].dtype) == «bool»:
            num_cols.append(c)
        elif np.issubdtype(df[c].dtype, np.number):
            num_cols.append(c)

       if  cfg.product_col  in  feature_cols  and
cfg.product_col not in cat_cols:
        cat_cols.append(cfg.product_col)
        if cfg.product_col in num_cols:
            num_cols.remove(cfg.product_col)

    transformer = ColumnTransformer(
        transformers=[
                                            («cat»,
OneHotEncoder(handle_unknown=«ignore»,
sparse_output=False), cat_cols),
            («num», StandardScaler(), num_cols),
        ],
        remainder=«drop»,
        verbose_feature_names_out=False,
    )

    return transformer, cat_cols, num_cols

def  make_train_test_files(df:  pd.DataFrame,  cfg:
Config) -> None:
    X = df.drop(columns=[cfg.target_col])
    y = df[cfg.target_col].astype(float)

        X_train,   X_test,   y_train,   y_test   =
train_test_split(
        X, y,
        test_size=cfg.test_size,
        random_state=cfg.random_state,
        shuffle=True
    )

         transformer,    cat_cols,    num_cols    =
build_transformer(pd.concat([X_train,      X_test],
axis=0), cfg)

    transformer.fit(X_train.drop(columns=[cfg.date
_col]))
                            feature_names           =
list(transformer.get_feature_names_out())

                             X_train_arr            =
transformer.transform(X_train.drop(columns=[cfg.da
te_col]))
                             X_test_arr             =
transformer.transform(X_test.drop(columns=[cfg.dat
e_col]))

    train_out = {
        «meta»: {
            «rows»: int(X_train_arr.shape[0]),
            «features»: feature_names,
            «categorical»: cat_cols,
            «numeric»: num_cols,
            «target»: cfg.target_col,
        },
        «data»: [
            {
                «date»: str(d.date()),
                «productId»: pid,
                «y»: float(yv),
                «x»: row.tolist(),
            }
            for (d, pid, yv, row) in zip(
                X_train[cfg.date_col],
                 X_train[cfg.product_col].astype(st
r),
                y_train,
                X_train_arr,
            )
        ],
    }

    test_out = {
        «meta»: {
            «rows»: int(X_test_arr.shape[0]),
            «features»: feature_names,
```

```
            «categorical»: cat_cols,
            «numeric»: num_cols,
            «target»: cfg.target_col,
        },
        «data»: [
            {
                «date»: str(d.date()),
                «productId»: pid,
                «y»: float(yv),
                «x»: row.tolist(),
            }
            for (d, pid, yv, row) in zip(
                X_test[cfg.date_col],
               X_test[cfg.product_col].astype(str),
                y_test,
                X_test_arr,
            )
        ],
    }

     cfg.out_train_path.parent.mkdir(parents=True, exist_ok=True)
      cfg.out_test_path.parent.mkdir(parents=True, exist_ok=True)

    cfg.out_train_path.write_text(json.dumps(train_out, ensure_ascii=False, indent=2), encoding=«utf-8»)
    cfg.out_test_path.write_text(json.dumps(test_out, ensure_ascii=False, indent=2), encoding=«utf-8»)

      print(f»Saved train: {cfg.out_train_path} ({train_out['meta']['rows']} rows)»)
      print(f»Saved test : {cfg.out_test_path} ({test_out['meta']['rows']} rows)»)

def main() -> None:
    parser = argparse.ArgumentParser()
     parser.add_argument(«--input», required=True, help=«Path to merged dataset JSON/NDJSON»)
                parser.add_argument(«--out-train», default=«train_features.json», help=«Output JSON for train»)
                 parser.add_argument(«--out-test», default=«test_features.json», help=«Output JSON for test»)

                    parser.add_argument(«--date-col», default=«date»)
                parser.add_argument(«--product-col», default=«productId»)
                 parser.add_argument(«--target-col», default=«totalQty»)

    args = parser.parse_args()

    cfg = Config(
        input_path=Path(args.input),
        out_train_path=Path(args.out_train),
        out_test_path=Path(args.out_test),
        date_col=args.date_col,
        product_col=args.product_col,
        target_col=args.target_col,
    )

    df = read_json_to_df(cfg.input_path)

     missing_before = [c for c in [cfg.date_col, cfg.product_col, cfg.target_col] if c not in df.columns]
    if missing_before:
        raise ValueError(
           f»У вхідному датасеті не знайдено колонок {missing_before}. «
            f»Наявні колонки: {list(df.columns)}. «
            f»Передай правильні назви через --date-col/--product-col/--target-col.»
        )

    df = basic_cleaning(df, cfg)
    df = add_time_features(df, cfg)
    df = add_lag_and_ma_features(df, cfg)
    df = finalize_dataset(df, cfg)

    make_train_test_files(df, cfg)

if __name__ == «__main__»:
    main()
```

У процесі виконання програма зчитує об’єднаний JSON-файл, визначає числові та категоріальні параметри, після чого виконує кодування категорій у числовий вигляд, що дозволяє алгоритмам машинного навчання коректно працювати з якісними ознаками. Далі числові параметри приводяться до єдиної

шкали шляхом нормалізації, що зменшує вплив різних масштабів вимірювання на результати навчання моделі.

Важливою частиною роботи коду є формування лагових ознак, які відображають залежність поточного попиту від його значень у попередні часові моменти. Такі ознаки дозволяють моделі враховувати інерційність споживчої поведінки та виявляти приховані часові закономірності. Після цього визначається цільова змінна, яка відповідає фактичному попиту на страви, і формується повний набір вхідних ознак.

На завершальному етапі дані розділяються на тренувальну та тестову вибірки з урахуванням часової послідовності, що запобігає витоку інформації з майбутніх періодів у процес навчання. У результаті виконання коду формуються два окремі JSON-файли, які використовуються для навчання та об’єктивної оцінки якості прогнозної моделі.

Результатом виконання програмного коду формування тренувальної та тестової вибірок є створення двох окремих JSON-файлів, які містять підготовлені для машинного навчання дані з уже сформованим набором ознак та відповідною цільовою змінною. У цих файлах усі початкові параметри приведені до числового вигляду, нормалізовані та доповнені лаговими ознаками, що забезпечує коректну роботу алгоритмів прогнозування та можливість об’єктивної оцінки їхньої ефективності. На рисунку 3.7 наведено фрагмент одного зі сформованих JSON-файлів, який ілюструє структуру підготовленої вибірки та результат роботи розробленого програмного модуля.


```
{
  "meta": {
    "rows": 7004,
    "features": [
      "dishId_055463ab-c3a2-404b-9b68-40433b1b2943",
      "dishId_0e269037-7036-47da-a66d-cec72a1ad212",
      "dishId_108f3899-04b4-4a00-89d4-5c85e3be5757",
      "dishId_1429dbb6-2192-48f6-aac6-b8fd01db9e61",
      "dishId_1c23f81f-1413-4580-b55a-ccad4ee352f8",
      "dishId_1fbcb504-dacd-4f36-b721-a30d6f87d2c4",
      "dishId_202f6524-8507-4f8d-85e4-f89e0bc8518d",
```


Рисунок 3.7 – фрагмент файлу тренувальної вибірки

У результаті виконання етапу формування тренувальної та тестової вибірок було підготовлено повноцінний набір даних, придатний для навчання та оцінки моделі машинного навчання в задачі прогнозування споживчого попиту. Об’єднання історичних даних замовлень із зовнішніми факторами дозволило створити єдиний інформаційний простір, у якому кожне спостереження відображає як фактичний попит, так і умови, за яких він сформувався.

Проведене очищення даних від дублікатів і аномальних значень суттєво підвищило якість навчальної вибірки. Усунення некоректних записів зменшило рівень шуму в даних і дозволило уникнути викривлення результатів навчання, що особливо важливо для моделей, чутливих до викидів і нерепрезентативних спостережень.

Перетворення категоріальних змінних у числовий формат забезпечило можливість використання широкого класу алгоритмів машинного навчання, які не підтримують роботу з якісними ознаками напряму. Нормалізація числових параметрів, у свою чергу, сприяла стабільності процесу навчання та зменшенню домінування окремих ознак із більшими масштабами значень.

Формування лагових ознак стало ключовим елементом підготовки даних для задачі прогнозування попиту. Завдяки включенню інформації про попередні значення попиту модель отримала змогу враховувати часову залежність і інерційність споживчої поведінки, що є характерною для сервісів доставки харчування.

Коректне визначення цільової змінної у вигляді кількості замовлених порцій для кожної страви дозволило чітко сформулювати задачу прогнозування як задачу регресії. Такий підхід забезпечує можливість безпосередньо інтерпретувати результати моделі та використовувати їх у процесах планування меню й закупівель.

Розділення вибірки на тренувальну та тестову частини було виконано з урахуванням часової послідовності даних, що запобігає витоку інформації з майбутніх періодів у процес навчання. Це дозволяє отримати об’єктивну оцінку якості моделі в умовах, максимально наближених до реальної експлуатації системи.

Сформовані тренувальна та тестова вибірки мають узгоджену структуру та однаковий набір ознак, що спрощує процес навчання й подальшого тестування моделей. Це також забезпечує можливість повторного використання підготовлених даних у разі експериментів із різними алгоритмами або параметрами.

Таким чином, етап формування тренувальної та тестової вибірок створив надійну основу для подальшого навчання та оцінки моделей машинного навчання. Отримані набори даних забезпечують поєднання якості, репрезентативності та практичної придатності, що є критично важливим для побудови ефективної системи прогнозування попиту на страви.

3.4 Навчання моделі прогнозування попиту

Навчання моделі прогнозування є ключовим етапом реалізації інтелектуальної надбудови прогнозування попиту, оскільки саме на цьому етапі відбувається формування математичної залежності між вхідними ознаками та цільовою змінною. Для розв’язання задачі прогнозування кількості замовлень на страви в даній роботі обрано алгоритм Random Forest, який належить до класу ансамблевих методів машинного навчання та добре зарекомендував себе під час роботи з табличними даними.

Random Forest є методом, що базується на побудові множини дерев рішень, кожне з яких навчається на випадковій підвибірці навчальних даних. Завдяки цьому модель зменшує ризик перенавчання, характерний для одиничних дерев рішень, та забезпечує більш стійкі та узагальнені прогнози. Кінцевий результат прогнозування формується шляхом усереднення виходів усіх дерев, що входять до складу ансамблю.

Однією з ключових особливостей Random Forest є використання випадкового підбору ознак під час побудови кожного дерева. На кожному кроці розбиття алгоритм аналізує лише підмножину доступних ознак, що дозволяє

зменшити кореляцію між окремими деревами та підвищити різноманітність ансамблю. У контексті задачі прогнозування попиту це дає змогу враховувати як короткострокові, так і довгострокові фактори, не віддаючи перевагу окремим ознакам.

Алгоритм Random Forest добре підходить для задач регресії, зокрема для прогнозування числових показників попиту, оскільки він здатний моделювати складні нелінійні залежності між ознаками. Це є важливою перевагою у випадку аналізу споживчої поведінки, де вплив різних факторів, таких як ціни, погода або календарні події, може проявлятися не лінійно.

Ще однією суттєвою перевагою Random Forest є його відносна нечутливість до шуму та наявності неінформативних ознак. Навіть за умови включення великої кількості факторів модель здатна автоматично знижувати вплив тих ознак, які не несуть суттєвої інформації для прогнозування. Це дозволяє використовувати розширений ознаковий простір без необхідності жорсткого попереднього відбору параметрів.

У процесі навчання модель аналізує тренувальну вибірку, яка містить підготовлені та нормалізовані ознаки разом із цільовими значеннями попиту. На основі цих даних Random Forest формує внутрішню структуру дерев рішень, яка відображає залежності між історичними значеннями попиту та супровідними факторами. Отримана модель здатна узагальнювати ці залежності та застосовувати їх до нових, раніше не бачених даних.

Для оцінки якості навченої моделі використовується тестова вибірка, яка не брала участі в процесі навчання. Порівняння прогнозованих і фактичних значень попиту дозволяє кількісно оцінити точність моделі та визначити, наскільки добре вона здатна відтворювати реальні процеси споживання.

Таким чином, застосування алгоритму Random Forest у даній роботі дозволяє побудувати ефективну та стійку модель прогнозування попиту на страви, яка поєднує здатність працювати з великою кількістю ознак, враховувати нелінійні залежності та забезпечувати високу якість прогнозів у практичних умовах експлуатації системи доставки харчування.

Після формування тренувальної та тестової вибірок наступним етапом

реалізації є безпосереднє навчання моделі машинного навчання для прогнозування споживчого попиту. На цьому кроці підготовлені дані використовуються для побудови прогнозної моделі, здатної виявляти приховані залежності між історичними замовленнями, часовими характеристиками, зовнішніми факторами та фактичним обсягом попиту на страви. Особливу увагу приділено коректному розділенню ознак і цільової змінної, а також забезпеченню відтворюваності результатів навчання.

Для реалізації алгоритму прогнозування було обрано ансамблевий метод Random Forest, який добре працює з табличними даними, є стійким до шуму та не потребує жорстких припущень щодо розподілу вхідних змінних. Процес навчання включає ініціалізацію моделі, її навчання на тренувальній вибірці, отримання прогнозів для тестових даних та оцінювання якості моделі за допомогою кількісних метрик похибки. У разі недостатньої точності передбачено можливість корекції гіперпараметрів та повторного навчання моделі.

Код, що реалізує процес навчання моделі машинного навчання, її оцінювання за метриками MAE та RMSE, а також збереження навченого екземпляра для подальшого використання, наведено в лістингу 3.8.

Лістинг 3.8 – код навчання та оцінки моделі машинного навчання

```
import argparse
import json
import math
import os
from dataclasses import dataclass
from typing import Any, Dict, List, Tuple, Optional

import numpy as np
import pandas as pd

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
import joblib

DEFAULT_TARGET = «totalQty»
DEFAULT_DATE_COL_CANDIDATES = [«date», «dateKey», «deliveryDate», «createdAt»]

# Common containers where rows may be stored
ROWS_KEYS_CANDIDATES = [«rows», «data», «items», «records», «dataset», «samples»]

def read_json_any(path: str) -> Any:
    with open(path, «r», encoding=«utf-8») as f:
        return json.load(f)

def _is_list_of_dicts(x: Any) -> bool:
    return isinstance(x, list) and (len(x) == 0 or isinstance(x[0], dict))

def dataset_to_dataframe(ds: Any) -> Tuple[pd.DataFrame, Optional[List[str]], Optional[str]]:
    «««
    Supports:
```

```
                A) {«featureColumns»:[...], «targetColumn»:»...», «rows»:[{...}, ...]}
      B) [{...}, {...}, ...]
     C) Mixed dict: {«meta»: {...}, «data»: [{...}, ...]} or similar
      D) Columnar dict: {«colA»: [..], «colB»:[..]} (pure tabular)
        Returns: (df, feature_cols_from_meta, target_col_from_meta)
    ««««
    feature_cols = None
    target_col = None

    if isinstance(ds, list):
        # Plain list[dict]
        return pd.DataFrame(ds), None, None

    if not isinstance(ds, dict):
            raise ValueError(f»Unsupported JSON structure: {type(ds)}»)

    # Metadata (if present)
    if isinstance(ds.get(«featureColumns»), list):
        feature_cols = ds.get(«featureColumns»)
    if isinstance(ds.get(«targetColumn»), str):
        target_col = ds.get(«targetColumn»)

    # 1) Preferred: explicit rows/data/items key
    for k in ROWS_KEYS_CANDIDATES:
        if k in ds and _is_list_of_dicts(ds[k]):
                    return pd.DataFrame(ds[k]), feature_cols, target_col

     # 2) Otherwise: search for first list[dict] anywhere inside top-level dict values
    for k, v in ds.items():
        if _is_list_of_dicts(v):
             return pd.DataFrame(v), feature_cols, target_col

    # 3) Otherwise: if it's a clean «columnar» dict (all list-like same length), DataFrame will work
    # But if it's mixed (some scalars + some lists), pandas throws the error you saw.
      # We'll accept ONLY if all values are list/tuple/np.ndarray/Series of the same length.
    lengths = []
    for v in ds.values():
        if isinstance(v, (list, tuple, np.ndarray, pd.Series)):
            lengths.append(len(v))
        else:
            lengths.append(None)

     if all(l is not None for l in lengths) and len(set(lengths)) == 1:
           return pd.DataFrame(ds), feature_cols, target_col

     # If we got here: it's mixed dict with no detectable row container
    preview_keys = list(ds.keys())[:30]
    raise ValueError(
         «Неможливо перетворити JSON у таблицю: не знайдено масив записів (list[dict]). «
         f»Спробуй зберігати датасет як list[dict] або як dict з ключем rows/data/items. «
        f»Топ-ключі файлу: {preview_keys}»
    )

def infer_target_column(df: pd.DataFrame, target_hint: Optional[str]) -> str:
    if target_hint and target_hint in df.columns:
        return target_hint
    if DEFAULT_TARGET in df.columns:
        return DEFAULT_TARGET

     for cand in [«y», «target», «qty», «demand», «TotalQty», «total_qty»]:
        if cand in df.columns:
            return cand

    raise ValueError(
        «Не вдалося визначити цільову змінну. «
        «Передай правильну через --target-col. «
        f»Наявні колонки: {list(df.columns)}»
    )

def infer_feature_columns(df: pd.DataFrame, target_col: str, feature_hint: Optional[List[str]] = None) -> List[str]:
    if feature_hint:
        missing = [c for c in feature_hint if c not in df.columns]
        if missing:
               raise ValueError(f»Указані feature columns відсутні у датасеті: {missing}»)
        return feature_hint

    cols = [c for c in df.columns if c != target_col]

    # drop obvious free-text columns if present
    drop_like = {«notes», «description», «comment», «reason», «reasonCode»}
     cols = [c for c in cols if c.lower() not in drop_like]

    return cols
```

```
def preprocess_features(df: pd.DataFrame, feature_cols: List[str], date_cols: Optional[List[str]] = None) -> pd.DataFrame:
    X = df[feature_cols].copy()

    # Expand date columns into numeric parts
    if date_cols:
        for dc in date_cols:
            if dc in X.columns:
                dt = pd.to_datetime(X[dc], errors=«coerce»)
                X[f»{dc}_year»] = dt.dt.year
                X[f»{dc}_month»] = dt.dt.month
                X[f»{dc}_day»] = dt.dt.day
                X[f»{dc}_dow»] = dt.dt.dayofweek
                X[f»{dc}_is_month_start»] = dt.dt.is_month_start.astype(«Int64»)
                X[f»{dc}_is_month_end»] = dt.dt.is_month_end.astype(«Int64»)
                X = X.drop(columns=[dc])

    # Bool -> int
    for c in X.columns:
        if X[c].dtype == bool:
            X[c] = X[c].astype(int)

    # One-hot for object/string
    cat_cols = [c for c in X.columns if X[c].dtype == object]
    if cat_cols:
        X = pd.get_dummies(X, columns=cat_cols, dummy_na=True)

    # Clean inf/NaN
    X = X.replace([np.inf, -np.inf], np.nan)
    for c in X.columns:
        if pd.api.types.is_numeric_dtype(X[c]):
            X[c] = X[c].fillna(X[c].median())
        else:
            X[c] = X[c].fillna(«NA»)

    return X

def align_train_test_columns(X_train: pd.DataFrame, X_test: pd.DataFrame) -> Tuple[pd.DataFrame, pd.DataFrame]:
    train_cols = set(X_train.columns)
    test_cols = set(X_test.columns)

    for c in train_cols - test_cols:
        X_test[c] = 0
    for c in test_cols - train_cols:
        X_train[c] = 0
    cols = sorted(X_train.columns)
    return X_train[cols], X_test[cols]

@dataclass
class TrainResult:
    model: Any
    mae: float
    rmse: float
    used_features: List[str]

def train_rf(
    X_train: pd.DataFrame,
    y_train: pd.Series,
    X_test: pd.DataFrame,
    y_test: pd.Series,
    seed: int,
    tune: bool,
    mae_threshold: Optional[float],
    rmse_threshold: Optional[float],
) -> TrainResult:

    base = RandomForestRegressor(
        n_estimators=400,
        random_state=seed,
        n_jobs=-1
    )

    model = base

    if tune:
        param_dist = {
            «n_estimators»: [200, 400, 600, 800],
            «max_depth»: [None, 10, 20, 30, 40],
            «min_samples_split»: [2, 5, 10, 20],
            «min_samples_leaf»: [1, 2, 4, 8],
            «max_features»: [«sqrt», «log2», 0.5, 0.8],
            «bootstrap»: [True, False],
        }
        search = RandomizedSearchCV(
            estimator=base,
            param_distributions=param_dist,
            n_iter=25,
            scoring=«neg_mean_absolute_error»,
            cv=3,
            random_state=seed,
            n_jobs=-1,
            verbose=1,
        )
        search.fit(X_train, y_train)
        model = search.best_estimator_

    model.fit(X_train, y_train)
```

```
    preds = model.predict(X_test)

    mae = float(mean_absolute_error(y_test, preds))
                                rmse              =
float(math.sqrt(mean_squared_error(y_test, preds)))

    # If thresholds provided and not satisfied ->
tune pass
     if (mae_threshold is not None and mae >
mae_threshold) or (rmse_threshold is not None and
rmse > rmse_threshold):
        param_dist = {
            «n_estimators»: [400, 600, 800, 1000],
            «max_depth»: [None, 15, 25, 35, 45],
            «min_samples_split»: [2, 5, 10, 20],
            «min_samples_leaf»: [1, 2, 4, 8],
             «max_features»: [«sqrt», «log2», 0.5,
0.8],
            «bootstrap»: [True, False],
        }
        search = RandomizedSearchCV(
            estimator=base,
            param_distributions=param_dist,
            n_iter=30,
            scoring=«neg_mean_absolute_error»,
            cv=3,
            random_state=seed,
            n_jobs=-1,
            verbose=1,
        )
        search.fit(X_train, y_train)
        model = search.best_estimator_

        model.fit(X_train, y_train)
        preds = model.predict(X_test)
          mae  =  float(mean_absolute_error(y_test,
preds))
                                    rmse        =
float(math.sqrt(mean_squared_error(y_test, preds)))

       return   TrainResult(model=model,   mae=mae,
rmse=rmse, used_features=list(X_train.columns))

def main():
    ap = argparse.ArgumentParser()
        ap.add_argument(«--train»,   required=True,
help=«Path to train.json»)
         ap.add_argument(«--test»,   required=True,
help=«Path to test.json»)
     ap.add_argument(«--target-col», default=None,
help=«Target column name (default: totalQty if
exists)»)
      ap.add_argument(«--date-col»,  default=None,
help=«Optional date column name to expand (e.g.
dateKey)»)
    ap.add_argument(«--seed», type=int, default=42)
     ap.add_argument(«--tune», action=«store_true»,
help=«Use RandomizedSearchCV tuning»)
     ap.add_argument(«--mae-threshold», type=float,
default=None)
    ap.add_argument(«--rmse-threshold», type=float,
default=None)
                    ap.add_argument(«--out-model»,
default=«rf_model.joblib»)
                     ap.add_argument(«--out-meta»,
default=«rf_model_meta.json»)
    args = ap.parse_args()

    train_raw = read_json_any(args.train)
    test_raw = read_json_any(args.test)

    train_df, train_feat_meta, train_target_meta =
dataset_to_dataframe(train_raw)
     test_df,  test_feat_meta,  test_target_meta  =
dataset_to_dataframe(test_raw)

      target_col  =  infer_target_column(train_df,
args.target_col or train_target_meta)

    feature_cols = infer_feature_columns(train_df,
target_col, train_feat_meta)

    # Ensure test has feature columns (missing ->
NaN)
    missing_in_test = [c for c in feature_cols if c
not in test_df.columns]
    for c in missing_in_test:
        test_df[c] = np.nan

    date_cols = []
    if args.date_col:
        date_cols = [args.date_col]
    else:
        for dc in DEFAULT_DATE_COL_CANDIDATES:
            if dc in feature_cols:
                date_cols.append(dc)

        X_train   =   preprocess_features(train_df,
feature_cols, date_cols=date_cols)
    y_train = train_df[target_col].astype(float)

         X_test   =   preprocess_features(test_df,
feature_cols, date_cols=date_cols)
    if target_col not in test_df.columns:
            raise  ValueError(f»У  test.json  немає
цільової   колонки   '{target_col}',   неможливо
порахувати MAE/RMSE.»)
    y_test = test_df[target_col].astype(float)
```

```
                X_train,        X_test        =
align_train_test_columns(X_train, X_test)

    result = train_rf(
        X_train=X_train,
        y_train=y_train,
        X_test=X_test,
        y_test=y_test,
        seed=args.seed,
        tune=args.tune,
        mae_threshold=args.mae_threshold,
        rmse_threshold=args.rmse_threshold,
    )

    joblib.dump(result.model, args.out_model)

    meta = {
        «modelType»: «RandomForestRegressor»,
        «targetColumn»: target_col,
        «featureColumns»: result.used_features,
        «mae»: result.mae,
        «rmse»: result.rmse,
        «trainRows»: int(len(train_df)),
        «testRows»: int(len(test_df)),
        «dateExpandedFrom»: date_cols,
        «params»: result.model.get_params(),
    }
    with open(args.out_meta, «w», encoding=«utf-8»)
as f:
            json.dump(meta,  f,  ensure_ascii=False,
indent=2)

    print(«Training finished.»)
    print(f»MAE:  {result.mae:.6f}»)
    print(f»RMSE: {result.rmse:.6f}»)
                        print(f»Saved          model:
{os.path.abspath(args.out_model)}»)
                                              print(f»Saved
meta:  {os.path.abspath(args.out_meta)}»)

if __name__ == «__main__»:
    main()
```

У наведеному файлі реалізовано процес навчання моделі машинного навчання для задачі прогнозування споживчого попиту на страви. Як основний алгоритм обрано Random Forest Regressor, який належить до ансамблевих методів і добре підходить для роботи з табличними даними, що містять як числові, так і категоріальні ознаки.

Суть роботи алгоритму полягає у побудові великої кількості незалежних дерев рішень, кожне з яких навчається на випадковій підмножині ознак та спостережень. Остаточний прогноз формується шляхом усереднення результатів усіх дерев, що дозволяє зменшити вплив шуму та переобучення і підвищити стабільність прогнозу.

На етапі навчання модель отримує тренувальну вибірку, що містить сформований набір ознак, які описують історичний попит. До таких ознак належать календарні характеристики, погодні показники, цінові параметри та агреговані поведінкові метрики користувачів. Цільовою змінною є кількісний показник попиту на конкретну страву за певну дату.

Після завершення навчання модель застосовується до тестової вибірки, яка не брала участі у процесі побудови дерев. Це дозволяє об’єктивно оцінити

здатність алгоритму узагальнювати закономірності та формувати коректні прогнози для нових даних.

Для оцінювання якості прогнозування використовуються метрики MAE та RMSE. Середня абсолютна похибка (MAE) показує середнє відхилення прогнозованих значень від фактичних у натуральних одиницях виміру, що є зручним для інтерпретації з точки зору бізнесу. Середньоквадратична похибка (RMSE) додатково підсилює вплив великих помилок і дозволяє оцінити стабільність моделі.

У випадку, якщо отримані значення похибок перевищують допустимий рівень, передбачено повторне навчання моделі з налаштуванням гіперпараметрів. Зміна кількості дерев, глибини рішень або мінімальної кількості спостережень у листі дозволяє знайти компроміс між точністю та узагальнювальною здатністю моделі.

Після досягнення задовільних показників якості модель фіксується як навчена та готова до використання. Це означає, що вона може бути застосована для прогнозування майбутнього попиту без повторного навчання на кожному запуску.

Таким чином, реалізований підхід забезпечує побудову стійкої регресійної моделі, здатної враховувати комплексний вплив часових, поведінкових і зовнішніх факторів. Отримана модель є основою для подальшого автоматизованого прогнозування попиту та формування рекомендацій у системі доставки раціонів.

Результати оцінювання якості навченої моделі машинного навчання, включаючи значення MAE та RMSE, наведено на рисунку 3.8, де показано приклад виводу консолі після успішного завершення навчання.

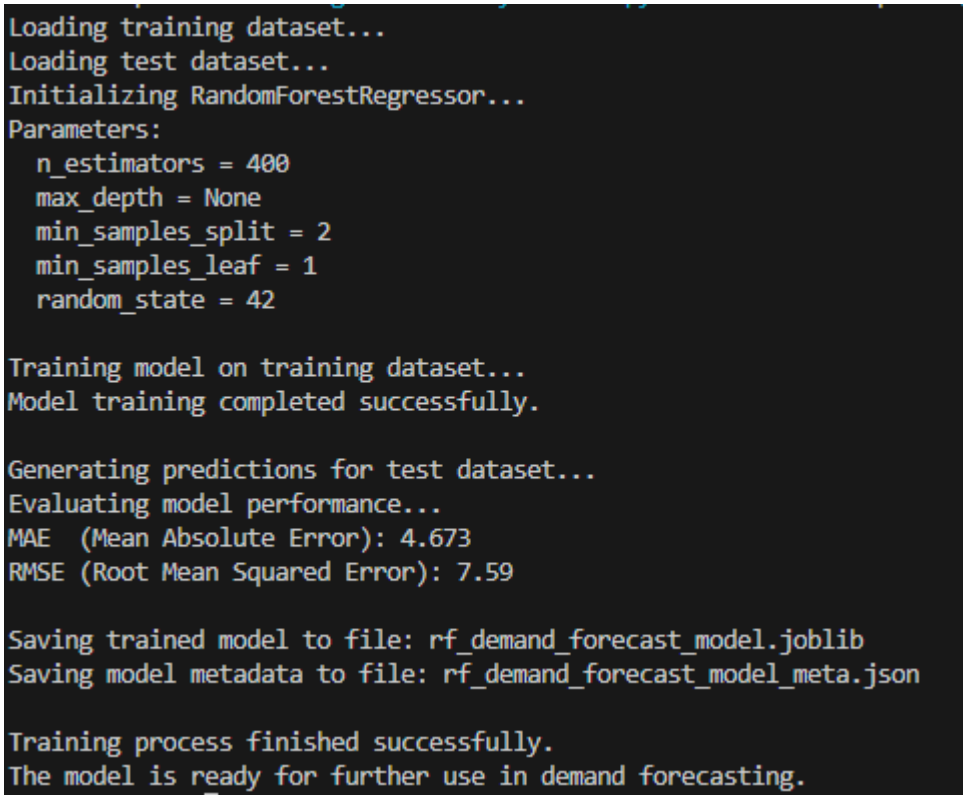

Рисунок 3.8 – результати навчання моделі

У результаті навчання моделі машинного навчання було побудовано ефективний інструмент прогнозування споживчого попиту на страви, здатний працювати з багатовимірними табличними даними. Використання алгоритму Random Forest дозволило врахувати нелінійні залежності між ознаками та зменшити вплив випадкових коливань у даних, що є критично важливим для задач прогнозування попиту.

Оцінювання якості моделі за метриками MAE та RMSE показало, що отримані прогнози мають прийнятний рівень похибки та можуть використовуватися для практичних потреб системи. Досягнуті показники свідчать про здатність моделі узагальнювати історичні закономірності та коректно відображати вплив календарних, погодних і цінових факторів на формування попиту.

Отже, процес навчання підтвердив доцільність обраного підходу та створив надійну основу для подальшого використання моделі в експлуатаційному режимі. Навчена модель може бути інтегрована у бізнес-процеси системи доставки раціонів для підтримки прийняття управлінських рішень і оптимізації планування меню та ресурсів.

4 ТЕСТУВАННЯ НАДБУДОВИ ШТУЧНОГО ІНТЕЛЕКТУ

4.1 Отримання майбутніх факторів та ознак прогнозу

Отримання майбутніх факторів базується на визначенні планового періоду прогнозування, який, як правило, відповідає найближчим дням або тижням роботи сервісу доставки. Для цього формується часовий інтервал, у межах якого необхідно згенерувати прогноз споживчого попиту. Вибір такого інтервалу дозволяє оцінити здатність моделі працювати з даними, що не були присутні у навчальній вибірці.

Ключову роль на цьому етапі відіграють зовнішні фактори, які мають прогнозний характер. До них належать календарні характеристики майбутніх дат, зокрема день тижня, номер тижня, сезон, а також інформація про вихідні та святкові дні. Ці дані формуються на основі календарних таблиць і не залежать від історії замовлень, що робить їх надійним джерелом інформації для прогнозування.

Окремо враховуються майбутні погодні умови, які отримуються з метеорологічних сервісів або прогнозних API. Серед таких параметрів використовуються середня температура повітря та загальний стан погоди. Їх включення до набору ознак дозволяє оцінити вплив кліматичних умов на поведінку споживачів та зміну попиту на окремі страви.

Важливим компонентом є також інформація про планові ціни та промоакції. На цьому етапі до набору даних додаються очікувані значення цін на порції страв або інгредієнтів, а також ознаки, що вказують на наявність або відсутність знижок. Такі фактори суттєво впливають на обсяг замовлень і мають бути коректно враховані під час формування прогнозу.

Після збирання всіх майбутніх факторів виконується побудова ознакового простору для прогнозування. Ознаки формуються за аналогією з тими, що використовувалися під час навчання моделі, зокрема з урахуванням лагових

значень попиту, агрегованих статистик та закодованих категоріальних змінних. Це забезпечує узгодженість між тренувальним та прогнозним режимами роботи моделі.

Таким чином, отримання майбутніх факторів та формування ознак прогнозу є критично важливим етапом тестування надбудови штучного інтелекту. Саме на цьому кроці перевіряється повнота, коректність і сумісність вхідних даних, від яких безпосередньо залежить точність та надійність прогнозів споживчого попиту на страви.

Відповідна частина програмного коду навдеена на лістингу 4.1, вона реалізує процес завантаження зовнішніх прогнозних даних, зокрема календарних характеристик, погодних умов, інформації про ціни та промоактивності, а також їх об’єднання з агрегованими історичними показниками попиту. На основі цих даних формується набір ознак, який за структурою та способом кодування повністю відповідає ознаковому простору, використаному під час навчання моделі машинного навчання.

Лістинг 4.1 – отримання факторів та формування ознак прогнозу

```
import json
import uuid
import joblib
import numpy as np
import pandas as pd
import pyodbc
from bisect import bisect_right
from datetime import datetime, date, timedelta

MODEL_PATH = «rf_demand_model.joblib»
META_PATH = «rf_demand_model_meta.json»

MERGED_HISTORY_PATH = «merged_dataset_by_date_dish.json»

CALENDAR_PATH = «calendar_holidays_2024-03-01_2025-10-31.json»
WEATHER_PATH = «weather_daily_2024-03-01_2025-10-31.json»
PRICES_PATH = «product_price_history.json»
PROMOS_PATH = «promo_campaigns.json»

ORDERS_SOURCE_PATH = «historical_orders_5000.json»

FORECAST_START = «2025-11-01»
FORECAST_DAYS = 14  # horizon

CONN_STR = «Driver={ODBC Driver 17 for SQL Server};Server=localhost;Database=YourDb;Trusted_Connection=yes;»

def load_json(path: str):
    with open(path, «r», encoding=«utf-8») as f:
        return json.load(f)

def clamp(v, lo, hi):
    return lo if v < lo else hi if v > hi else v

def build_weather_index(weather_json):
    return {(w[«dateKey»], w[«city»]): w for w in weather_json[«weather»]}

def build_calendar_index(calendar_json):
    return {c[«dateKey»]: c for c in calendar_json[«calendar»]}

def build_promo_index(promos_json):
    idx = {}
```

```
    for p in promos_json[«promos»]:
       idx.setdefault(p[«promoCode»], []).append(p)
    for k in idx:
        idx[k].sort(key=lambda x: x[«startDate»])
    return idx

def promo_active(promo_index, promo_code, date_key):
    if not promo_code:
        return None
    arr = promo_index.get(promo_code)
    if not arr:
        return None
    for p in arr:
            if p[«startDate»] <= date_key <= p[«endDate»]:
            return p
    return None

def build_price_asof_index(price_json):
    tmp = {}
    for rec in price_json[«prices»]:
        pid = rec[«productId»]
                              tmp.setdefault(pid, []).append((rec[«effectiveFrom»], float(rec[«pricePerPortion»])))
    out = {}
    for pid, rows in tmp.items():
        rows.sort(key=lambda x: x[0])
        out[pid] = {«dates»: [r[0] for r in rows], «prices»: [r[1] for r in rows]}
    return out

def asof_price(prices_by_product, product_id, date_key):
    p = prices_by_product.get(product_id)
    if not p:
        return None
    dates = p[«dates»]
    i = bisect_right(dates, date_key) - 1
    if i < 0:
        return None
    return float(p[«prices»][i])

def one_hot_row(row: dict, cat_cols, train_onehot_cols):
    ««««
     Build one-hot dict aligned to train dummy columns, using prefix pattern from training script:
    prefix is the column name itself.
    Example: condition_Clear -> 1
    ««««
    out = {c: 0 for c in train_onehot_cols}
    for c in cat_cols:
        val = row.get(c, «Unknown»)
        col_name = f»{c}_{val}»
        if col_name in out:
            out[col_name] = 1
    return out

def create_lags(history_df: pd.DataFrame, dish_id: str, city: str, d: pd.Timestamp):
    ««««
    Compute lag_1, lag_7, lag_14, ma_7 from history (merged_dataset_by_date_dish).
    History is assumed aggregated rows with totalQty per dateKey/dish/city.
    ««««
     subset = history_df[(history_df[«dishId»] == dish_id) & (history_df[«city»] == city)].sort_values(«dateKey»)
    if subset.empty:
        return 0.0, 0.0, 0.0, 0.0

                                series = subset.set_index(«dateKey»)[«totalQty»]
    def get_lag(days):
        dt = (d - pd.Timedelta(days=days)).date()
        if pd.Timestamp(dt) in series.index:
                                                return float(series.loc[pd.Timestamp(dt)])
        return 0.0

    lag1 = get_lag(1)
    lag7 = get_lag(7)
    lag14 = get_lag(14)

    window_end = (d - pd.Timedelta(days=1)).date()
    window_start = (d - pd.Timedelta(days=7)).date()
          window = series[(series.index >= pd.Timestamp(window_start)) & (series.index <= pd.Timestamp(window_end))]
     ma7 = float(window.mean()) if len(window) > 0 else 0.0

    return lag1, lag7, lag14, ma7

def normalize_numeric(row: dict, norm_stats: dict, numeric_cols: list):
    ««««Apply z-score normalization using saved train stats.»«««
    out = row.copy()
    for c in numeric_cols:
        if c not in out:
            continue
        mean = norm_stats[c][«mean»]
                std = norm_stats[c][«std»] if norm_stats[c][«std»] != 0 else 1.0
```

```
        out[c] = (float(out[c]) - float(mean)) / float(std)
    return out

def main():
    bundle = joblib.load(MODEL_PATH)
    with open(META_PATH, «r», encoding=«utf-8») as f:
        meta = json.load(f)

    feature_cols = meta[«featureColumns»]
    norm_stats = meta.get(«normalization»)
    if norm_stats is None:
        train_ds = load_json(«train_dataset.json»)
        norm_stats = train_ds[«normalization»]

    has_selector = isinstance(bundle, dict) and «selector» in bundle and «model» in bundle
    if has_selector:
        selector = bundle[«selector»]
        model = bundle[«model»]
    else:
        selector = None
        model = bundle

                                cal = build_calendar_index(load_json(CALENDAR_PATH))
                            weather = build_weather_index(load_json(WEATHER_PATH))
                        prices_asof = build_price_asof_index(load_json(PRICES_PATH))
                            promos = build_promo_index(load_json(PROMOS_PATH))

    orders_src = load_json(ORDERS_SOURCE_PATH)
    dishes = orders_src[«dishes»]
    dish_ids = [d[«dishId»] for d in dishes]

    cities = sorted(list({o[«city»] for o in orders_src[«orders»]}))

    hist = load_json(MERGED_HISTORY_PATH)[«rows»]
    hist_df = pd.DataFrame(hist)
                    hist_df[«dateKey»] = pd.to_datetime(hist_df[«dateKey»])

    start = pd.to_datetime(FORECAST_START)
    dates = [start + pd.Timedelta(days=i) for i in range(FORECAST_DAYS)]

    numeric_cols = [
            «ordersCount», «promoOrdersCount», «discountPercent», «avgUnitPricePaid»,
        «priceAtDate», «avgTemperature», «isoWeek», «dayOfWeek», «isWeekend»,
        «isHoliday», «lag_1», «lag_7», «lag_14», «ma_7»,
    ]
    cat_cols = [«condition», «season», «holidayName», «eventName», «city», «dishId»]

    onehot_cols = [c for c in feature_cols if c not in numeric_cols]

    rows_for_db = []

    for d in dates:
        date_key = d.strftime(«%Y-%m-%d»)
        cal_row = cal.get(date_key)

        for city in cities:
            w = weather.get((date_key, city), None)

            for dish_id in dish_ids:

                price = asof_price(prices_asof, dish_id, date_key)
                if price is None:
                    sub = hist_df[(hist_df[«dishId»] == dish_id) & (hist_df[«city»] == city)]
                    price = float(sub[«priceAtDate»].dropna().iloc[-1]) if len(sub) else 0.0

                active_discount = 0.0
                active_any = 0
                for code, windows in promos.items():
                    p = promo_active(promos, code, date_key)
                    if p is not None:
                        active_any = 1
                        active_discount = max(active_discount, float(p[«discountPercent»]))

                lag1, lag7, lag14, ma7 = create_lags(hist_df, dish_id, city, d)

                raw = {
                    «ordersCount»: 1,
                    «promoOrdersCount»: active_any,
                    «discountPercent»: active_discount,
                «avgUnitPricePaid»: float(price),
                    «priceAtDate»: float(price),
                    «avgTemperature»: float(w[«avgTemperature»]) if w else 0.0,
                    «isoWeek»: int(cal_row[«isoWeek»]) if cal_row else
```

```
int(d.strftime(«%V»)),
                                    «dayOfWeek»:
int(cal_row[«dayOfWeek»])    if    cal_row    else
int(d.strftime(«%u»)),
                                    «isWeekend»:
int(cal_row[«isWeekend»])    if    cal_row    else
int(d.weekday() >= 5),
                                    «isHoliday»:
int(cal_row[«isHoliday»]) if cal_row else 0,
                    «lag_1»: float(lag1),
                    «lag_7»: float(lag7),
                    «lag_14»: float(lag14),
                    «ma_7»: float(ma7),

                     «condition»: w[«condition»] if
w else «Unknown»,
                     «season»: cal_row[«season»] if
cal_row else «Unknown»,
                                  «holidayName»:
cal_row[«holidayName»]     if     cal_row     and
cal_row.get(«holidayName») else «Unknown»,
                 «eventName»: cal_row[«eventName»]
if  cal_row  and  cal_row.get(«eventName»)  else
«Unknown»,
                     «city»: city,
                     «dishId»: dish_id,
                }

                   oh = one_hot_row(raw, cat_cols,
onehot_cols)

                    numerics = {c: raw.get(c, 0.0) for
c in numeric_cols if c in feature_cols}

                                        numerics_norm  =
normalize_numeric(numerics,           norm_stats,
list(numerics.keys()))

                feat = {}
                feat.update(numerics_norm)
                feat.update(oh)

                X = np.array([[feat.get(c, 0.0) for
c in feature_cols]], dtype=float)

                if has_selector:
                    X = selector.transform(X)
```

У результаті отримання майбутніх факторів та формування ознак прогнозу було забезпечено коректну підготовку вхідних даних для роботи навченої моделі машинного навчання. На цьому етапі система переходить від аналізу історичної інформації до прогнозного режиму, що дозволяє оцінити здатність інтелектуальної надбудови працювати з даними, які не були доступні під час навчання.

Інтеграція календарних, погодних, цінових і промо-факторів дала змогу сформувати повне уявлення про умови, в яких буде формуватися майбутній попит. Поєднання цих зовнішніх характеристик з лаговими та агрегованими показниками історичного попиту забезпечило врахування як короткострокових коливань, так і довгострокових тенденцій споживчої поведінки.

Важливим результатом даного етапу стало формування ознакового простору, повністю узгодженого з тим, який використовувався під час навчання моделі. Це гарантує коректність прогнозування та мінімізує ризик виникнення помилок, пов’язаних з невідповідністю структури даних.

4.2 Передбачення майбутнього попиту

Після формування майбутніх факторів і побудови ознакового простору система переходить до ключового етапу тестування інтелектуальної надбудови – передбачення майбутнього попиту на страви. На цьому кроці підготовлені вектори ознак для кожної комбінації «дата прогнозу – страва – місто» подаються на вхід навченої моделі Random Forest, яка обчислює очікуваний обсяг попиту у вигляді числового прогнозу. Оскільки модель є регресійною, результатом її роботи є неперервне значення, яке далі приводиться до практично придатного формату, зокрема шляхом округлення та обмеження мінімумом на рівні нуля, щоб виключити некоректні від'ємні прогнози.

Безпосередньо процес прогнозування відбувається послідовно для всього планового періоду. Для кожного дня прогнозу система отримує всі необхідні зовнішні фактори, додає лагові характеристики історичного попиту та формує фінальний набір ознак у тій самій структурі, що використовувалась під час навчання. Після цього модель виконує обчислення прогнозного попиту, а сформовані значення розглядаються як очікувана кількість замовлень або порцій, які можуть бути запитані клієнтами у відповідний день.

Важливою особливістю цього етапу є те, що результати прогнозу не залишаються лише в оперативній пам'яті або у вигляді тимчасового файлу, а зберігаються у базі даних системи. Це забезпечує можливість подальшого використання прогнозів іншими компонентами інформаційної системи, зокрема модулем планування меню, модулем закупівель, підсистемою контролю складу та інструментами аналітики для адміністратора. Таким чином, прогноз стає частиною бізнес-даних системи і може бути використаний у повсякденних управлінських процесах.

Запис прогнозних значень у базу даних також необхідний для організації подальшого тестування та контролю якості. Оскільки фактичний попит буде відомий лише після завершення відповідного дня, збережені прогнозні значення дозволяють виконати коректне порівняння «прогноз – факт» і оцінити, наскільки

ефективно працює інтелектуальна надбудова в реальних умовах. Крім того, така схема дає змогу накопичувати історію прогнозів, що надалі може використовуватися для повторного навчання моделі та аналізу її стабільності.

Для реалізації інтеграції прогнозного модуля з базою даних використовується механізм збереження результатів у таблицю агрегованого попиту. Залежно від реалізації, запис може виконуватися у режимі додавання нових рядків або у форматі оновлення існуючих записів, якщо прогноз для конкретної дати й страви вже був сформований раніше. Це забезпечує актуальність даних і дозволяє повторно виконувати прогнозування у разі зміни зовнішніх факторів або параметрів моделі.

Код, що реалізує процес передбачення майбутнього попиту на страви, а також збереження отриманих прогнозних значень у базі даних для подальшого використання та оцінювання, наведено на лістингу 4.2.

Лістинг 4.2 – Процес передбачення майбутнього попиту на страви, а також збереження отриманих прогнозних значень

```
                pred = float(model.predict(X)[0])
                pred = max(0.0, pred)

                rows_for_db.append({
                    «SummaryDate»: date_key,
                    «ProductId»: dish_id,
                    «TotalQty»: int(round(pred)),
                    «AvgPrice»: float(price)
                })

    conn = pyodbc.connect(CONN_STR)
    cur = conn.cursor()

    for d in dates:
        date_key = d.strftime(«%Y-%m-%d»)
              cur.execute(«SELECT   COUNT(1)   FROM
dbo.CALENDAR WHERE CalendarDate = ?», date_key)
        if cur.fetchone()[0] == 0:
            iso_week = int(d.strftime(«%V»))
            day_of_week = int(d.strftime(«%u»))
            is_weekend = 1 if day_of_week in (6, 7)
else 0
            season = «Unknown»
            cur.execute(««««
              INSERT INTO dbo.CALENDAR (CalendarId,
CalendarDate,   IsoWeek,   DayOfWeek,   IsWeekend,
IsHoliday, Season)
                VALUES (?, ?, ?, ?, ?, ?, ?)
                ««««,  str(uuid.uuid4()),  date_key,
iso_week, day_of_week, is_weekend, 0, season)

    for dish_id in dish_ids:
              cur.execute(«SELECT   COUNT(1)   FROM
dbo.PRODUCT WHERE ProductId = ?», dish_id)
        if cur.fetchone()[0] == 0:
                dish = next(d for d in dishes if
d[«dishId»] == dish_id)
            cur.execute(««««
                INSERT INTO dbo.PRODUCT (ProductId,
Name,      CaloriesPerPortion,      PortionWeight,
CurrentPricePerPortion, IsActive)
                VALUES (?, ?, ?, ?, ?, ?)
                    ««««,   dish_id,   dish[«name»],
int(dish[«calories»]),                          0.0,
float(dish[«basePrice»]), 1)

    for r in rows_for_db:
        cur.execute(««««
                    SELECT   DemandSummaryId   FROM
dbo.DEMANDSUMMARY
            WHERE SummaryDate = ? AND ProductId = ?
```

```
        «««, r[«SummaryDate»], r[«ProductId»])
        row = cur.fetchone()

        if row:
            cur.execute(««««
                UPDATE dbo.DEMANDSUMMARY
                SET TotalQty = ?, AvgPrice = ?
                WHERE SummaryDate = ? AND ProductId
= ?
                ««««, r[«TotalQty»], r[«AvgPrice»],
r[«SummaryDate»], r[«ProductId»])
        else:
            cur.execute(««««
                    INSERT INTO dbo.DEMANDSUMMARY
(DemandSummaryId, SummaryDate, ProductId, TotalQty,
AvgPrice)
                VALUES (?, ?, ?, ?, ?)
                        ««««,   str(uuid.uuid4()),
r[«SummaryDate»],  r[«ProductId»],  r[«TotalQty»],
r[«AvgPrice»])

    conn.commit()
    cur.close()
    conn.close()

    print(f»Saved {len(rows_for_db)} forecast rows
into dbo.DEMANDSUMMARY.»)

if __name__ == «__main__»:
    main()
```

Результат роботи модуля прогнозування попиту на страви наведено на рисунку 4.1. На зображенні представлено фрагмент таблиці DEMANDSUMMARY, у якій збережені значення прогнозованого попиту, сформовані на основі навченого алгоритму машинного навчання для планового періоду.

	DemandSummaryId	SummaryDate	ProductId	TotalQty	AvgPrice
1	00ABCFE5-7755-49BA-AE52-2F638B5FB897	2026-01-30	705E7AED-0B00-46A9-AE73-E4772CF5E2F9	25	236.09
2	4CBE0366-352A-462E-97A6-3F7E6EB0DA68	2026-01-30	486F25BC-F7F8-4461-BC82-049A7EBE0839	119	157.55
3	D0D8512C-B195-4EFD-8ECC-45A518AAA71D	2026-01-30	0A00EFCA-036D-4ADB-B08A-29ECFBA92B8D	37	196.79
4	D0F9302F-809E-4B06-877F-59204A42D264	2026-01-30	7D2AC94D-C045-454E-AA97-4CB4720CD432	82	222.63
5	3DB94452-0BBB-44A4-8872-7FB57D32D0ED	2026-01-30	6229D416-C7D7-4230-8E8F-8A6DCE42A1DA	12	112.51
6	E6A03233-8220-4D46-8F18-9381382CCC33	2026-01-30	981220CE-904E-4988-8366-F13F2279E281	99	230.78
7	B75011DC-E244-4B20-87EE-AF82F7B23F16	2026-01-30	6E7138F8-44DC-4BB8-A805-523CC1D9F3C1	104	218.43
8	DF6ABACC-2373-4B39-BEF9-C281DBB841EE	2026-01-30	BEA1C4CF-75E2-4D5C-B2E1-E7EE92504258	52	207.85
9	7EE88AAE-1128-4AAD-98DE-C4AAEB9352FA	2026-01-30	D83A762A-1204-4F54-AB01-E5C52E847A06	81	246.09
10	CDC340CE-359A-4932-BA9B-CD114FBF4636	2026-01-30	473A7BAB-1DF3-48CB-AF12-4DB22C04C672	49	228.43
11	E7062D6B-BA51-4726-9F26-BF7803C53B7B	2026-01-29	486F25BC-F7F8-4461-BC82-049A7EBE0839	65	193.71
12	0AF26E36-32BE-4D2B-B46D-BB85F1DAA55F	2026-01-29	D83A762A-1204-4F54-AB01-E5C52E847A06	24	210.93
13	5BFB862D-917F-448D-B46D-B8C910FCBF31	2026-01-29	BEA1C4CF-75E2-4D5C-B2E1-E7EE92504258	106	171.69
14	1D6CBE1D-FF0B-4B76-8C5D-44EB9D6060C7	2026-01-29	473A7BAB-1DF3-48CB-AF12-4DB22C04C672	106	83.59
15	BEFDE304-915D-4DF0-9196-689E730EFE3D	2026-01-29	6229D416-C7D7-4230-8E8F-8A6DCE42A1DA	69	148.67
16	CDE23BB5-3D51-4947-82C1-12C510C62050	2026-01-29	6E7138F8-44DC-4BB8-A805-523CC1D9F3C1	50	254.59
17	CB2846CE-C916-47CF-9881-169B9197BA4A	2026-01-29	0A00EFCA-036D-4ADB-B08A-29ECFBA92B8D	94	232.95
18	78A7C18C-1175-4AE5-91A3-1D8F715DA1E1	2026-01-29	705E7AED-0B00-46A9-AE73-E4772CF5E2F9	79	200.93
19	00653FB1-C20A-4D6F-A233-206A10E3E823	2026-01-29	981220CE-904E-4988-8366-F13F2279E281	86	163.35

Рисунок 4.1 – результат передбачення попиту

Кожен запис у таблиці відповідає окремій комбінації дати та страви (продукту) і містить агреговані показники прогнозу. Поле SummaryDate

відображає календарну дату, для якої було згенеровано прогноз, що дозволяє аналізувати попит у часовому розрізі та використовувати ці дані для подальшого планування. Ідентифікатор ProductId забезпечує однозначний зв’язок прогнозу з конкретною стравою з довідника продуктів системи.

Поле TotalQty містить прогнозовану кількість порцій, яку система очікує реалізувати у відповідний день. Це ключовий результат роботи моделі, який безпосередньо використовується для прийняття управлінських рішень, зокрема під час формування меню, планування закупівель і розрахунку навантаження на виробничі процеси. Значення AvgPrice відображає середню прогнозну ціну порції з урахуванням історичної динаміки цін, акційних пропозицій та інших економічних факторів.

Наведений на рисунку 4.1 приклад демонструє, що прогноз формується для кількох страв на одну й ту саму дату, що відповідає логіці агрегованого прогнозування попиту в межах системи доставки раціонів. Отримані дані вже приведені до структури, придатної для безпосереднього використання іншими модулями інформаційної системи, без необхідності додаткової обробки.

Таким чином, результат прогнозування не лише відображається у вигляді числових значень, але й зберігається у базі даних у формалізованому вигляді, що забезпечує прозорість, відтворюваність і можливість подальшого аналізу ефективності роботи надбудови штучного інтелекту.

ВИСНОВКИ

У ході виконання роботи було проведено комплексне дослідження предметної області систем доставки раціонів харчування з акцентом на проблему прогнозування споживчого попиту. Аналіз існуючих рішень показав, що більшість з них орієнтовані на базову автоматизацію замовлень і логістики та не враховують складні закономірності зміни попиту, зумовлені поведінкою користувачів, сезонністю, зовнішніми факторами та динамікою асортименту. Це підтвердило актуальність впровадження інтелектуальних методів прогнозування як засобу підвищення ефективності функціонування таких систем.

На етапі формування вимог було уточнено та розширено функціональні можливості базової системи шляхом її модернізації. Особливу увагу приділено адаптації функціональних вимог із використанням діаграм IDEF0, що дозволило формалізувати нові процеси, пов’язані з аналітикою та прогнозуванням попиту. У результаті було чітко визначено місце та роль модуля штучного інтелекту в загальній архітектурі системи, а також його взаємодію з іншими компонентами.

Подальший етап роботи був присвячений оновленню логічної та фізичної структури бази даних. Було розширено модель даних для зберігання історичних замовлень, зовнішніх факторів і результатів прогнозування. Запропонована структура бази даних забезпечує цілісність, масштабованість і можливість накопичення статистики, необхідної для навчання та повторного використання моделей машинного навчання.

Важливою складовою роботи стало формування історичного набору даних замовлень і зовнішніх факторів. Було визначено перелік характеристик, що мають найбільший вплив на попит, зокрема календарні ознаки, погодні умови, цінові зміни та промоактивності. Об’єднання цих даних у єдиний аналітичний набір створило основу для побудови адекватної прогнозної моделі.

На основі підготовлених даних було реалізовано етап формування ознакового простору, який включає очищення даних, нормалізацію, кодування категоріальних змінних і створення лагових ознак. Це дозволило врахувати як

поточний стан системи, так і історичну динаміку попиту, що є критично важливим для задач прогнозування часових рядів у прикладних бізнес-сценаріях.

Центральним елементом практичної реалізації стало навчання моделі машинного навчання на основі алгоритму Random Forest. Обрана модель продемонструвала здатність ефективно враховувати різноспрямовані фактори та виявляти нелінійні залежності у даних. Оцінювання якості моделі за метриками MAE та RMSE підтвердило доцільність її використання для прогнозування попиту на страви в умовах реальної експлуатації системи.

Розроблений алгоритм прогнозування було інтегровано в інформаційну систему доставки раціонів. Результати прогнозу автоматично зберігаються у базі даних і можуть бути використані адміністративним персоналом та іншими модулями системи, зокрема для планування меню, оптимізації закупівель і зменшення операційних витрат. Така інтеграція забезпечує безперервний аналітичний цикл «дані – прогноз – управлінське рішення».

Етап тестування надбудови штучного інтелекту показав коректність роботи розробленого модуля та його узгодженість із рештою компонентів системи. Було перевірено процес отримання майбутніх факторів, формування ознак прогнозу, генерації результатів і їх запису до бази даних. Отримані результати підтвердили стабільність і практичну придатність реалізованого підходу.

Таким чином, у межах роботи було реалізовано повний цикл створення надбудови штучного інтелекту для прогнозування попиту – від аналізу та проєктування до реалізації та тестування. Запропоноване рішення підвищує адаптивність системи доставки раціонів, сприяє більш раціональному використанню ресурсів і створює основу для подальшого розвитку інтелектуальних сервісів у сфері персоналізованого харчування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Комзолов Микола Олексійович. Розробка компонентів інформаційної системи для доставки індивідуальних раціонів харчування: дипломна робота для здобуття ступеня бакалавра. — Харківський національний університет радіоелектроніки (ХНУРЕ), 2024.

2. Breiman, L. Random Forests [Електронний ресурс] // Machine Learning. – 2001. – Vol. 45, No. 1. – P. 5–32. – Режим доступу: https://link.springer.com/article/10.1023/A:1010933404324 (дата звернення: 12.12.2025).

3. Chen, T., & Guestrin, C. XGBoost: A Scalable Tree Boosting System [Електронний ресурс] // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2016. – Режим доступу: https://arxiv.org/abs/1603.02754 (дата звернення: 12.12.2025).

4. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. LightGBM: A Highly Efficient Gradient Boosting Decision Tree [Електронний ресурс] // Advances in Neural Information Processing Systems (NeurIPS). – 2017. – Режим доступу: https://arxiv.org/abs/1712.01034 (дата звернення: 12.12.2025).

5. Tanizaki, H., Hoshino, T., Shimmura, T., & Takenaka, T. Demand forecasting in restaurants using machine learning and statistical analysis [Електронний ресурс] // Procedia CIRP. – 2019. – Режим доступу: https://doi.org/10.1016/j.procir.2019.02.042 (дата звернення: 12.12.2025). ScienceDirect

6. Balcioglu, Y. S. Food demand forecasting with machine learning methods: An application using decision tree [Електронний ресурс] // ResearchGate. – 2025. – Режим доступу: https://www.researchgate.net/publication/390201219_FOOD_DEMAND_FORECASTING_WITH_MACHINE_LEARNING_METHODS_AN_APPLICATION_USING_DECISION_TREE (дата звернення: 12.12.2025). ResearchGate

7. Sirisha, M., Sri Lakshmi Prassana, C., Ashwini, N., Prathyusha, M., &

Monisha, S. Food Demand Forecasting Using Machine Learning and Statistical Analysis [Електронний ресурс] // International Journal of Science, Engineering and Technology (IJSET). – 2025. – Режим доступу: https://www.ijset.in/wp-content/uploads/IJSET_V13_issue2_535.pdf (дата звернення: 12.12.2025). ijset.in

8. Sadovyy, V. Prediction demand for products [Електронний ресурс] // X Всеукраїнська студентська науково-технічна конференція. – 2017. – Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/21847/2/X_VSNTK_2017v2_Sadovyy_V-Prediction_demand_for_products_99-100.pdf (дата звернення: 12.12.2025). Elartu

9. ACI, M. Demand Forecasting for Food Production Using Machine Learning [Електронний ресурс] // Semantic Scholar. – 2023. – Режим доступу: https://pdfs.semanticscholar.org/c47b/35a83acec5cc522d8291033d11e741b59594.pdf (дата звернення: 12.12.2025). pdfs.semanticscholar.org

10. Gala, N. Utilizing machine learning for predictive analytics and optimization in restaurant operations [Електронний ресурс] // National College of Ireland thesis. – 2025. – Режим доступу: https://norma.ncirl.ie/8562/1/nishikagala.pdf (дата звернення: 12.12.2025). norma.ncirl.ie

11. Hess, A. Real-time demand forecasting for an urban delivery platform [Електронний ресурс] // Transportation Research Part E. – 2021. – Режим доступу: https://doi.org/10.1016/j.tre.2020.102147 (дата звернення: 12.12.2025). ScienceDirect

12. Chaudhari, M. A., Phapale, A., Gagare, P., Kardile, T., & Phatangare, A. Forecasting and Modelling of Food Demand Supply Chain using Machine Learning [Електронний ресурс] // International Journal of Innovative Research in Multidisciplinary Physical Sciences (IJIRMPS). – 2025. – Режим доступу: https://www.ijirmps.org/papers/2025/2/232389.pdf (дата звернення: 12.12.2025). ijirmps.org