НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2026 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Інформаційна система моніторингу середовища та управління виробничою діяльністю рибної ферми”

Виконав студент 4 курсу, групи ТР-22
Муренець Максим Вікторович _____
(прізвище, ім'я, по батькові) *(підпис)*

Науковий керівник асистент каф. цифрових технологій в енергетиці, Олександр КАРДАШОВ _____
(науковий ступінь, вчене звання, ім'я ПРІЗВИЩЕ) *(підпис)*

Рецензент професор каф. теплової та альтернативної енергетики, д.т.н., Ольга ЧЕРНОУСЕНКО _____
(посада, науковий ступінь, вчене звання, ім'я ПРІЗВИЩЕ) *(підпис)*

Н.контроль доцент каф. цифрових технологій в енергетиці, д.ф., Артем ГУРІН _____
(посада, ім'я ПРІЗВИЩЕ) *(підпис)*

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

Муренцю Максиму Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Інформаційна система моніторингу середовища та управління виробничою діяльністю рибної ферми”
Науковий керівник Кардашов Олександр Вадимович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “28” травня 2026 року №1992-с
2.Термін подання студентом роботи 09.06.2026
3.Вихідні дані до роботи: TypeScript/JavaScript (Node.js), NestJS, React 19, PostgreSQL, TypeORM, Socket.io, Redux Toolkit, RTK Query, Tailwind CSS, Recharts, Swagger, JWT, Bcrypt, Docker, Docker Compose; середовища розробки WebStorm, Visual Studio Code.
4.Перелік питань, які потрібно розробити:
1) проаналізувати існуючі програмні рішення для управління рибними господарствами та підходи до цифровізації аквакультури;
2) визначити функціональні (FR-01–FR-10) та нефункціональні вимоги до системи з урахуванням специфіки IoT-моніторингу водойм;
3) обґрунтувати вибір технологій для серверної частини, підсистеми IoT-телеметрії та клієнтського застосунку;

4) розробити архітектуру системи: серверні модулі, WebSocket-шлюз push-доставки телеметрії, реляційну модель даних та SPA-клієнт;
5) реалізувати алгоритми розрахунку норми годівлі (Action_KG), FCR, SGR та транзакційний складський облік;
6) розробити UI для візуалізації KPI та управління виробничим циклом, продемонструвати роботу системи.

5.Орієнтовний перелік ілюстративного матеріалу: інтерфейси систем-аналогів (AquaManager, eFishery, xFarm); компонентна діаграма, ER-діаграма БД, діаграми послідовностей (годівля, IoT-телеметрія); скріншоти сторінок системи (авторизація, Dashboard, карта ферми, ставки, годівля, інвентар, вилов).

6.Дата видачі завдання 14.09.2025.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	20.02.26	
2.	Аналіз методів та засобів розв’язання задачі	17.04.26 – 20.04.26	
3.	Розробка архітектури та загальної структури системи	21.04.26 – 25.04.26	
4.	Розробка окремих підсистем	26.04.26 – 02.05.26	
5.	Програмна реалізація системи	03.05.26 – 07.05.26	
6.	Оформлення пояснювальної записки	08.05.26 – 12.05.26	
7.	Захист програмного забезпечення	11.05.26	
8.	Передзахист	02.06.26	
9.	Захист	18.06.26	

Студент _____ Максим МУРЕНЕЦЬ
(підпис) (прізвище та ініціали)

Керівник _____ Олександр КАРДАШОВ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 66 сторінках, містить 13 ілюстрацій, 9 таблиць, 1 додаток, 26 джерел у переліку посилань.

Мета роботи – розробка інформаційної системи «FishFarm» для моніторингу середовища та управління виробничою діяльністю рибної ферми з підтримкою IoT-телеметрії та автоматизованим розрахунком показників ефективності.

Методи та засоби: TypeScript і JavaScript на платформі Node.js; фреймворк NestJS із REST API та WebSocket-шлюзом; бібліотека React 19 з Redux Toolkit і RTK Query; СУБД PostgreSQL з ORM TypeORM; Socket.io для push-доставки телеметрії; Docker Compose для контейнеризації та розгортання.

Основний зміст роботи: аналіз існуючих рішень для управління аквакультурою та формалізація вимог; побудова клієнт-серверної архітектури; розробка математичної моделі розрахунку норм годівлі з температурним і протеїновим коефіцієнтами; реалізація алгоритмів FCR, SGR та скінченного автомата станів ставка; впровадження REST API для приймання IoT-телеметрії з push-розсилкою через WebSocket; функціональне та навантажувальне тестування.

Рекомендації: застосування на малих і середніх рибогосподарських підприємствах; підключення реальних IoT-датчиків через REST API; розширення аналітики прогнозуванням врожайності.

Результат – програмний продукт «FishFarm», готовий до розгортання через Docker Compose на будь-якій платформі.

Ключові слова: інформаційна система, система управління, SaaS-система, аквакультура, моніторинг довкілля, IoT-телеметрія, websocket, NestJS, React, PostgreSQL, Docker.

ABSTRACT

Bachelor's thesis covers 66 pages, contains 13 illustrations, 9 tables, 1 appendix, 26 references.

The objective of the thesis is to develop the "FishFarm" information system for environmental monitoring and production management of a fish farm with IoT telemetry support and automated calculation of efficiency indicators.

Methods and tools: TypeScript and JavaScript on the Node.js platform; NestJS framework with REST API and WebSocket gateway; React 19 library with Redux Toolkit and RTK Query; PostgreSQL database with TypeORM ORM; Socket.io for push telemetry delivery; Docker Compose for containerization and deployment.

Main content: analysis of existing aquaculture management solutions and requirements formalization; design of client-server architecture; development of a mathematical model for fish feeding norm calculation with temperature and protein coefficients; implementation of FCR, SGR algorithms and a finite state machine for pond lifecycle management; deployment of a REST API for IoT telemetry ingestion with WebSocket push delivery; functional and load testing.

Recommendations: application in small and medium-scale fish farming enterprises; integration of real IoT sensors via the REST API; extension of the analytics module with yield forecasting.

Result – the "FishFarm" software product, ready for deployment via Docker Compose on any platform.

Keywords: information system, management system, SaaS system, aquaculture, environmental monitoring, IoT telemetry, websocket, NestJS, React, PostgreSQL, Docker.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

ACID – Atomicity, Consistency, Isolation, Durability – властивості транзакцій бази даних.

API – Application Programming Interface – програмний інтерфейс застосунку.

БД – База даних – організована сукупність структурованих даних.

CRUD – Create, Read, Update, Delete – базові операції над записами сховища даних.

DI – Dependency Injection – патерн впровадження залежностей у компоненти системи.

DTO – Data Transfer Object – об'єкт для передачі даних між рівнями застосунку.

ERP – Enterprise Resource Planning – система планування ресурсів підприємства.

FCR – Feed Conversion Ratio – кормовий коефіцієнт (корм / приріст біомаси).

FSM – Finite State Machine – скінченний автомат; модель керування станами ставка.

HTTP – HyperText Transfer Protocol – протокол передачі даних між клієнтом і сервером.

IoT – Internet of Things – концепція підключення фізичних пристроїв (датчиків) до мережі.

JSON – JavaScript Object Notation – текстовий формат серіалізації структурованих даних.

JWT – JSON Web Token – стандарт токенів автентифікації з цифровим підписом (RFC 7519).

KPI – Key Performance Indicator – ключовий показник ефективності виробництва.

MVCC – Multi-Version Concurrency Control – механізм ізоляції паралельних транзакцій у PostgreSQL.

ORM – Object-Relational Mapping – технологія відображення об'єктів на рядки реляційної БД.

ПЗ – Програмне забезпечення.

REST – Representational State Transfer – архітектурний стиль побудови API.

SGR – Specific Growth Rate – питома швидкість росту риби (% приросту біомаси за добу).

SPA – Single Page Application – односторінковий веб-застосунок.

SQL – Structured Query Language – мова запитів до реляційних баз даних.

СУБД – Система управління базами даних.

UI – User Interface – інтерфейс користувача.

UX – User Experience – користувацький досвід взаємодії з програмним продуктом.

WebSocket – протокол двонаправленого зв'язку поверх TCP для push-оновлень (RFC 6455).

ВСТУП

У сучасних умовах стрімкого зростання світової галузі аквакультури та збільшення потреби в раціональному використанні водних біоресурсів актуальним постає питання цифровізації процесів управління рибогосподарськими підприємствами. Згідно зі звітом FAO «The State of World Fisheries and Aquaculture 2024» у 2022 році виробництво продукції аквакультури вперше перевищило 130 млн тонн, забезпечивши понад 51% світового споживання водних біоресурсів [1]. Водночас в Україні, де ставковий фонд перевищує 120 тисяч гектарів, понад 70% малих і середніх господарств досі ведуть облік у паперових журналах, а норми годівлі визначають емпірично [2]. Відсутність доступних, простих у впровадженні та адаптованих до специфіки прісноводної аквакультури програмних рішень обумовлює необхідність розробки спеціалізованої інформаційної системи для моніторингу та управління рибним господарством на базі сучасних веб- та IoT-технологій [3].

Актуальність теми.

Своєчасний моніторинг параметрів водного середовища та автоматизований розрахунок норм годівлі дозволяють виявляти критичні відхилення, уникати систематичного перегодовування або недокорму риби та об'єктивно оцінювати ефективність виробництва. Більшість існуючих комерційних рішень – AquaManager, eFishery, xFarm – орієнтовані на великі підприємства, прив'язані до пропрієтарного обладнання або не враховують специфіку зимування прісноводних видів, характерну для українських господарств. Враховуючи зазначені обмеження, а також високу вартість впровадження зарубіжних систем для малих і середніх господарств, розробка апаратно-незалежного програмного комплексу з підтримкою IoT-телеметрії в реальному часі є своєчасною й має практичне значення.

Мета роботи.

Метою дипломної роботи є розробка програмного забезпечення інформаційної системи моніторингу та управління процесами рибного

господарства «FishFarm» із підтримкою IoT-телеметрії, автоматизованого розрахунку добових та разових норм годівлі з урахуванням температурного та протеїнового коефіцієнтів, а також обліку ключових виробничих показників ефективності FCR та SGR через реактивний веб-інтерфейс із push-доставкою даних за технологією WebSocket.

Завдання роботи.

Насамперед, провести аналіз існуючих підходів, методів та програмних рішень у галузі цифрового управління аквакультурою, виявити їхні функціональні обмеження та сформулювати вимоги до розроблюваної системи. Також треба обґрунтувати вибір програмних засобів реалізації та розробити алгоритми обчислення добових і разових норм годівлі з урахуванням температурного та протеїнового коефіцієнтів, показників ефективності вирощування FCR і SGR та валідації виробничих процесів. Є потреба у розробці клієнт-серверного програмного забезпечення з модульною архітектурою бекенду на базі NestJS, підсистемою емуляції IoT-обладнання, реактивним веб-інтерфейсом на Next.js та механізмом push-доставки телеметричних даних через WebSocket. Реалізувати апаратно-незалежну архітектуру IoT-підсистеми, яка дозволяє підключати довільні датчики без модифікації серверної та клієнтської частин. Виконати функціональне й навантажувальне тестування розробленої програмної системи, продемонструвати її функціональні можливості та оцінити продуктивність на типовому персональному комп'ютері.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД МЕТОДІВ ЇЇ РОЗВ'ЯЗАННЯ

У цьому розділі проведено комплексний аналіз існуючих на міжнародному ринку програмних рішень для автоматизації управління аквакультурними господарствами. Для кожного з розглянутих продуктів визначено архітектурні особливості, функціональні переваги та суттєві обмеження. Проаналізовано сучасні тенденції цифровізації рибної галузі та виявлено ключові технологічні тренди. На основі виявлених прогалин у підрозділі 1.2 сформульовано прикладну задачу та виділено функціональні й нефункціональні вимоги до розроблюваної системи. У підрозділі 1.3 обґрунтовано вибір технологічного стеку з детальним трасуванням кожного інструменту до конкретних вимог.

1.1 Аналіз існуючих програмних рішень у галузі управління аквакультурою

Цифровізація аквакультурної галузі є порівняно молодим, проте стрімко зростаючим напрямком AgriTech (сільськогосподарських технологій). На відміну від рослинництва та тваринництва, де ERP-системи (Enterprise Resource Planning – планування ресурсів підприємства) набули поширення ще на початку 2010-х років завдяки адаптації універсальних бізнес-рішень, спеціалізоване програмне забезпечення для рибних господарств залишається нішевим продуктом з обмеженою кількістю постачальників на світовому ринку [1]. Така ситуація зумовлена унікальною специфікою аквакультури, що суттєво відрізняє її від інших галузей сільського господарства.

Об’єктом виробництва є риба, котра перебуває у водному середовищі, параметри якого (температура, вміст розчиненого кисню, рівень pH, концентрація аміаку та нітритів) динамічно змінюються протягом доби та сезону і безпосередньо впливають на метаболізм, апетит та імунітет риби [4]. На відміну від наземних

тварин, де зміна умов утримання відбувається повільно і дозволяє вжити коригуючих заходів протягом годин або днів, критична зміна параметрів водного середовища може призвести до масової загибелі поголів'я протягом лічених годин. Наприклад, падіння концентрації розчиненого кисню нижче 3 мг/л при температурі вище 25°C викликає задуху коропових риб за 30-60 хвилин.

Норми годування риби залежать від комбінації факторів, що не мають прямих аналогів в інших галузях тваринництва: температури води, яка визначає швидкість метаболізму пойкілотермних організмів; протеїновмісності корму, що впливає на засвоюваність поживних речовин, поточної біомаси стада, фази виробничого циклу (від личинки до товарної риби), та навіть часу доби, оскільки нічна активність більшості культурних видів знижена [5].

Життєвий цикл рибоводного ставка є скінченним автоматом (Finite State Machine, FSM) із чіткими правилами переходів між станами: від підготовки водойми через зариблення, активну фазу вирощування, зимування (для багаторічних видів) до вилову та спускання. Кожен стан визначає набір дозволених операцій та обмежень, що вимагає спеціалізованого програмного забезпечення для коректного моделювання бізнес-процесів.

Для формування об’єктивної картини поточного стану ринку програмних рішень проаналізовано три найбільш поширені комерційні платформи, що позиціонуються як спеціалізовані системи для управління аквакультурою.

AquaManager – комплексне Enterprise-рішення грецької компанії AquaManager Ltd., що розробляється з 2008 року [6]. Платформа пропонує широкий спектр модулів: облік біомаси та партій риби, планування та реєстрація годування, моніторинг здоров’я та ветеринарний облік, управління персоналом та робочими змінами, фінансова звітність та аналітика. Система підтримує як морську, так і прісноводну аквакультуру, включаючи вирощування молюсків та ракоподібних. За даними офіційного вебсайту компанії, станом на 2024 рік платформа використовується понад 200 аквакультурними підприємствами переважно у Середземноморському регіоні, Скандинавії та Великій Британії. Інтерфейс системи AquaManager представлено на рисунку 1.1.

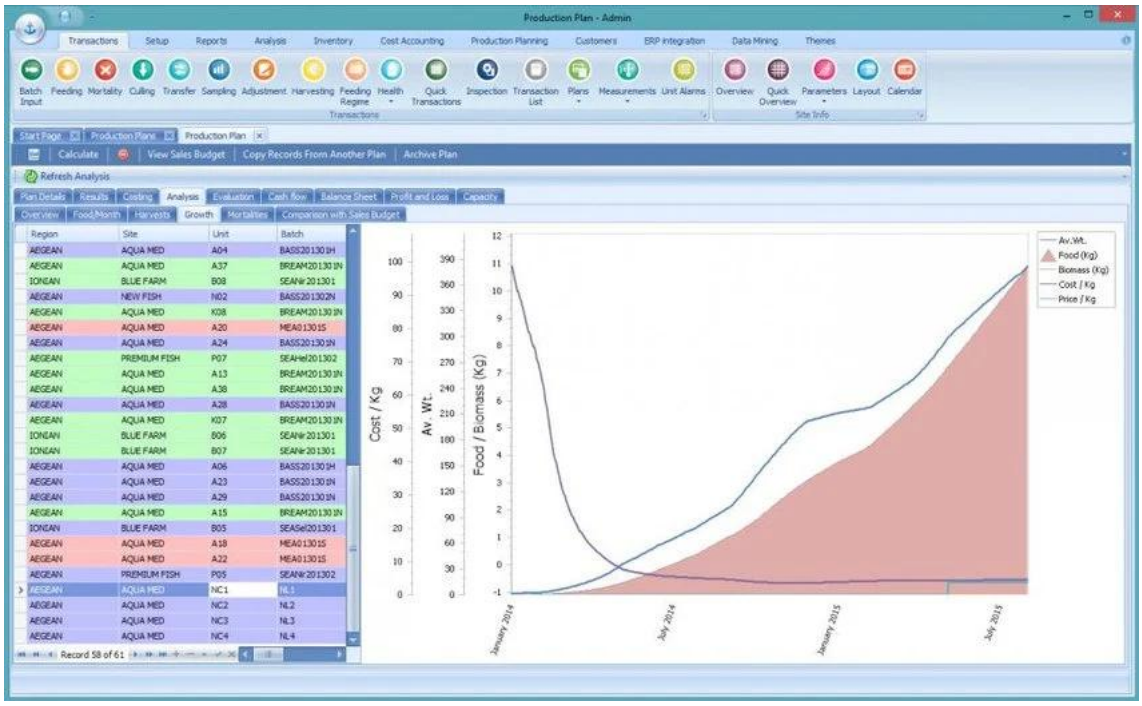


Рисунок 1.1 – Інтерфейс системи AquaManager

Ключовою перевагою AquaManager є глибина функціоналу, накопичена за понад 15 років розробки та співпраці з великими грецькими та норвезькими рибопромисловими компаніями. Система має розгалужену систему звітності, інтеграцію з бухгалтерськими системами та підтримку регуляторних вимог ЄС щодо простежуваності продукції. Проте саме Enterprise-орієнтація є головним недоліком: вартість ліцензії починається від кількох тисяч доларів на рік з додатковою оплатою за впровадження та навчання персоналу. Це унеможливлює використання платформи малими фермерськими господарствами України з річним оборотом 50–200 тисяч гривень. Окремим недоліком є модель поширення: AquaManager пропонується виключно як хмарний SaaS-сервіс з централізованим зберіганням даних на серверах постачальника, що створює залежність від стабільного інтернет-з’єднання та ставить під ризик безперервність бізнесу у разі припинення роботи провайдера. Можливість локального розгортання (on-premise) на власній інфраструктурі замовника не передбачена, що є суттєвим обмеженням

для підприємств, розташованих у сільській місцевості з нестабільним доступом до мережі.

eFishery – індонезійська платформа, заснована у 2013 році, що здобула значну популярність у країнах Південно-Східної Азії завдяки інноваційній моделі Hardware-as-a-Service [7]. Компанія розробляє та виробляє пропрієтарні смарт-фідери (автоматичні годівниці) з вбудованими датчиками, які інтегровані з хмарною платформою та мобільним додатком. Станом на 2024 рік eFishery обслуговує понад 70000 фермерів переважно в Індонезії, В'єтнамі та Філіппінах. Компанія залучила понад 200 мільйонів доларів інвестицій від Sequoia Capital, SoftBank та Temasek Holdings, що зробило її одним із найбільших AgriTech-стартапів Південно-Східної Азії. Інтерфейс мобільного додатку eFishery представлено на рисунку 1.2.

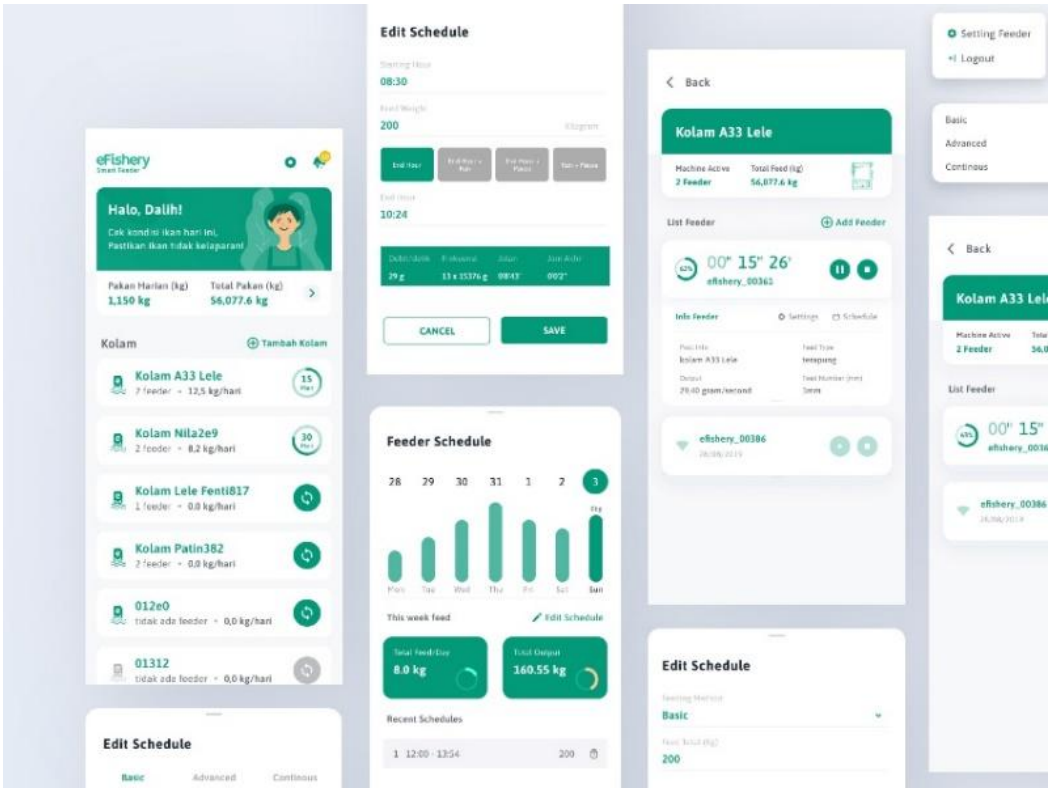


Рисунок 1.2 – Інтерфейс мобільного додатку eFishery

Перевагою eFishery є низький поріг входу для користувачів: фермер отримує обладнання в оренду або лізинг та відразу починає використовувати систему без

значних початкових інвестицій. Мобільний додаток має інтуїтивний інтерфейс, адаптований для користувачів з базовою цифровою грамотністю. Проте фундаментальним обмеженням є жорстка прив'язка до власного апаратного забезпечення: система не дозволяє підключити сторонні датчики або годівниці інших виробників. Це створює vendor lock-in та унеможливлює гнучку модернізацію господарства.

Крім того, eFishery орієнтована виключно на тропічні види (тилапія, креветки, пангасіус) без підтримки режиму зимування та температурних коефіцієнтів для помірного клімату. Математичні моделі годівлі оптимізовані для стабільної температури 25-30°C і не враховують сезонних коливань, характерних для українських умов (від 4°C взимку до 28°C влітку). Поняття «зимівля ставка» повністю відсутнє в архітектурі системи, оскільки в тропічному кліматі Індонезії температура води ніколи не опускається нижче 20°C. Відсутній також фінансовий модуль для обліку витрат та доходів, журнал аудиту операцій та можливість обчислення ключових показників ефективності виробництва (FCR та SGR), що ускладнює прийняття обґрунтованих управлінських рішень.

xFarm Technologies – італійська агроплатформа, що розвивається з 2017 року як універсальне рішення для цифровізації різних галузей сільського господарства: рослинництва, тваринництва, виноградарства та аквакультури [8]. Модульна архітектура дозволяє фермеру обирати лише потрібні компоненти та платити пропорційно використанню. Вартість базової підписки починається від 10 євро на місяць, що робить платформу доступною для малих господарств.

Проте саме універсальність є головним недоліком xFarm у контексті аквакультури. Модуль рибництва залишається поверхневим: він дозволяє вести загальний облік поголів’я та витрат, але не має спеціалізованих функцій – моделей годівлі з урахуванням температури, розрахунку FCR та SGR, FSM-автомата ставка, WebSocket-інтеграції для телеметрії в реальному часі. Фактично xFarm пропонує електронний журнал з графічною візуалізацією, а не повноцінну ERP-систему для управління рибним господарством. Аквакультура вимагає специфічних алгоритмів (температурні коефіцієнти, протеїнова нормалізація, валідація в діапазоні 70–

130%), спеціалізованої моделі даних (ставки з FSM, партії риби, корми з різним протеїном, телеметрія в реальному часі) та інтеграції з IoT-обладнанням, що не може бути забезпечено без фундаментальної переробки архітектури. Окрім того, модуль аквакультури xFarm не підтримує технологію WebSocket для push-доставки телеметрії: оновлення даних відбувається через стандартний HTTP-поллінг із затримкою, що вимірюється десятками секунд, а не мілісекундами. Це унеможливлює оперативне реагування на критичні зміни параметрів водного середовища, такі як різке падіння концентрації розчиненого кисню при грозовій активності.

На основі проведеного аналізу складено порівняльну таблицю (таблиця 1.1), що узагальнює функціональні можливості розглянутих рішень та демонструє прогалини, які має заповнити розроблювана система «FishFarm».

Таблиця 1.1 – Порівняльний аналіз існуючих програмних рішень для управління аквакультурою

Критерій	**AquaManager**	**eFishery**	**xFarm**	**FishFarm**
Облік біомаси та партій	+	+	±	+
Норми годівлі за температурою	–	–	–	+
IoT через WebSocket	–	±	–	+
REST API для датчиків	–	–	–	+
FCR та SGR	+	–	–	+
Склад та фінанси	±	–	–	+
Аудит операцій	+	–	–	+
Українська мова	–	–	–	+
Docker-контейнеризація	–	–	–	+
Малі господарства	–	±	+	+

Як видно з таблиці 1.1, жодне з розглянутих комерційних рішень не забезпечує повного комплексу функцій, необхідних для ефективного управління

українським рибним господарством помірного клімату. Основні виявлені прогалини включають: відсутність біологічних моделей годівлі з температурним коефіцієнтом та двоетапним поділом на добову та разову порції; відсутність апаратно-незалежного REST API для приймання телеметрії від довільних IoT-датчиків з push-доставкою через WebSocket; відсутність інтегрованого складського та фінансового модуля з повним аудитом операцій; відсутність локалізації українською мовою. Саме ці прогалини обґрунтовують необхідність та доцільність розробки власного програмного рішення «FishFarm».

1.2 Постановка прикладної задачі та вимоги до системи

На підставі аналізу існуючих рішень та виявлених прогалин сформульовано прикладну задачу: розробити комплексну інформаційну систему «FishFarm» для автоматизації управління виробничими процесами рибогосподарського підприємства, що забезпечує моніторинг параметрів водного середовища в реальному часі, автоматизований розрахунок оптимальних норм годівлі з урахуванням біологічних факторів, облік складських запасів та фінансових операцій, а також повний аудит усіх модифікуючих дій користувачів [9].

Відповідно до методології розробки програмного забезпечення, вимоги до системи поділяються на функціональні (Functional Requirements, FR) – що визначають конкретні можливості та поведінку системи, та нефункціональні (Non-Functional Requirements, NFR) – що визначають якісні характеристики, обмеження та критерії прийняття [9].

Функціональні вимоги до системи «FishFarm» сформульовано з урахуванням типових бізнес-процесів рибогосподарського підприємства та потреб різних категорій користувачів: адміністраторів (повний доступ до конфігурації), менеджерів (операційне управління) та операторів (реєстрація поточних операцій). Перелік функціональних вимог наведено в таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги до системи «FishFarm»

ID	**Опис вимоги**
FR-01	CRUD-операції зі ставками (статуси: Active, Wintering, Standby, Drained), партіями риби та кормами з протеїном
FR-02	IoT-телеметрія через Hardware-Agnostic REST API (POST-запити від довільних датчиків або програмного емулятора)
FR-03	Push-доставка телеметрії клієнтам через WebSocket із механізмом кімнат (rooms) для ізоляції потоків
FR-04	Двоетапний розрахунок норм: добова норма Daily_KG та разова порція Action_KG = Daily_KG / feeding_times_per_day
FR-05	Валідація годування: фактична маса у межах 70–130% від Action_KG, контроль статусу ставка та температури
FR-06	Управління складськими запасами із транзакційною цілісністю списання
FR-07	Фінансовий облік: витрати за категоріями, доходи від продажу, операції Full та Partial Harvest
FR-08	Глобальний журнал аудиту (Audit Log) всіх модифікуючих операцій
FR-09	JWT-автентифікація з ролями (admin – повний доступ, manager – обмежені операції)
FR-10	Локалізація інтерфейсу українською мовою

Особливої уваги заслуговує вимога FR-04, яка формалізує авторський алгоритм розрахунку норм годівлі. На відміну від спрощених підходів, де норма визначається як фіксований відсоток від біомаси, запропонований метод передбачає двоетапний розрахунок: спочатку обчислюється добова норма Daily_KG з урахуванням температурного та протеїнового коефіцієнтів, а потім вона ділиться на кількість годувань для отримання рекомендованої разової порції Action_KG. Валідація FR-05 працює саме з Action_KG, допускаючи варіативність ±30% для врахування суб'єктивної оцінки оператором стану поголів'я.

Вимога FR-02 фіксує принципову архітектурну відмінність від підходу eFishery: REST API для приймання телеметрії є апаратно-незалежним (Hardware-Agnostic). Це означає, що система приймає стандартизовані JSON-повідомлення від будь-яких пристроїв, здатних виконати HTTP POST-запит: промислових IoT-шлюзів, саморобних мікроконтролерів на базі ESP32 або Arduino, або ж програмного емулятора для тестування та демонстрації.

Вимога FR-03 забезпечує миттєве відображення даних телеметрії в інтерфейсі оператора без перезавантаження сторінки. Механізм кімнат (rooms) бібліотеки Socket.io дозволяє ізолювати потоки даних: клієнт, що переглядає сторінку конкретного ставка, отримує оновлення лише цього ставка, а не всіх ставків господарства. Це є критичною оптимізацією для масштабованості системи, оскільки при великій кількості ставків та датчиків загальний обсяг телеметрії може бути значним, і без ізоляції потоків кожен клієнт отримуватиме надмірне навантаження нерелевантними даними.

Вимога FR-05 встановлює діапазон допустимих відхилень 70–130% від розрахованої разової порції Action_KG. Вибір саме цього діапазону обумовлений практикою промислової аквакультури: досвідчений оператор може візуально оцінити зміну апетиту риби (наприклад, при наближенні грозового фронту риба стає менш активною, при стійкій теплій погоді – навпаки, більш активною) та скоригувати порцію в межах 30% від розрахункового значення. Водночас діапазон є достатньо вузьким, щоб запобігти грубим помилкам – наприклад, випадковому введенню кількості в тоннах замість кілограмів або помноженню на неправильний коефіцієнт.

Вимога FR-06 формалізує потребу в транзакційній цілісності складських операцій. Операція годування є найбільш критичною з точки зору узгодженості даних, оскільки вона одночасно змінює три сутності бази даних: створює запис у журналі годувань (feeding_logs), зменшує залишок обраної позиції корму на складі (inventory_items) та створює запис у журналі складських транзакцій (inventory_transactions). Якщо між цими операціями відбудеться збій (наприклад, відключення електроживлення або перезапуск серверного контейнера), база даних

може залишитися в неузгодженому стані: корм списаний зі складу, але запис годування не створений, або навпаки. Тому всі три операції мають виконуватися атомарно – або всі успішно, або жодна.

Вимога FR-07 передбачає два типи операцій вилову, що відповідають реальним бізнес-процесам рибного господарства. Full Harvest (повний вилов) застосовується при завершенні виробничого циклу: вся партія риби вилучається зі ставка, фіксується фінальна біомаса для розрахунку FCR та SGR, ставок переводиться у стан Drained для санітарної обробки. Partial Harvest (частковий вилов) дозволяє реалізувати частину товарної риби (наприклад, для задоволення поточного замовлення покупця), при цьому поточна біомаса партії оновлюється, а ставок залишається в активному стані. В обох випадках система автоматично створює фінансовий запис із зазначенням маси проданої риби, ціни за кілограм та загального доходу.

Вимога FR-08 забезпечує повну простежуваність усіх дій персоналу на платформі. Глобальний журнал аудиту (Audit Log) автоматично реєструє кожну модифікуючу операцію: хто її виконав (ідентифікатор користувача), що саме було змінено (тип сутності та її ідентифікатор), яку дію виконано (створення, оновлення, видалення) та коли (мітка часу з точністю до мілісекунди). Така детальна протоколізація є необхідною для забезпечення прозорості бізнес-процесів, розслідування нештатних ситуацій (наприклад, виявлення джерела несанкціонованого списання корму) та формування звітності для зовнішнього аудиту.

Нефункціональні вимоги визначають якісні характеристики системи та критерії прийняття, що підлягають вимірюванню та верифікації в процесі тестування. Перелік нефункціональних вимог наведено в таблиці 1.3.

Таблиця 1.3 – Нефункціональні вимоги до системи «FishFarm»

ID	**Опис вимоги**
NFR-01	Висока модульність серверної частини через архітектуру Dependency Injection (NestJS)

Продовження таблиці 1.3

NFR-02	Контейнеризація всього стеку через Docker та docker-compose для переносимості
NFR-03	Затримка рендеру UI-компонентів після завантаження даних не більше 100 мс
NFR-04	Затримка доставки WebSocket-повідомлень від сервера до клієнта не більше 50 мс
NFR-05	Безпечна автентифікація через JWT з хешуванням паролів алгоритмом Bcrypt
NFR-06	Транзакційна цілісність критичних операцій через EntityManager.transaction()
NFR-07	Стабільна робота системи при одночасному підключенні не менше 50 користувачів

Вимоги NFR-03 та NFR-04 задають кількісні показники продуктивності, виконання яких забезпечується вибором відповідних архітектурних рішень – реактивної бібліотеки React із віртуальним DOM для NFR-03 та подієво-орієнтованого протоколу WebSocket для NFR-04 – і підтверджується практичною перевіркою реактивності інтерфейсу та своєчасності доставки оновлень телеметрії у розділі 4. Вимога NFR-06 критична для забезпечення узгодженості даних при конкурентному доступі кількох операторів до однієї бази даних. Вимога NFR-07 визначає масштаб навантажувального тестування.

Кожна нефункціональна вимога безпосередньо впливає на архітектурні рішення та вибір технологій, що обґрунтовується у підрозділі 1.3. Вимога NFR-01 (модульність) обумовлює вибір фреймворку NestJS із вбудованим механізмом Dependency Injection, що забезпечує слабку зв’язність між компонентами та можливість незалежної розробки окремих модулів. Вимога NFR-02 (контейнеризація) визначає використання Docker та docker-compose для пакування всіх компонентів системи (бази даних, серверної частини, клієнтського додатку) у ізольовані контейнери, що забезпечує відтворюваність середовища на будь-якій

платформі. Вимога NFR-03 (реактивність інтерфейсу) обумовлює вибір бібліотеки React із механізмом Virtual DOM та Redux Toolkit із кешуванням через RTK Query, що мінімізує кількість перемальовок DOM-дерева та мережевих запитів. Вимога NFR-04 (затримка WebSocket) визначає вибір протоколу WebSocket замість HTTP-поллінгу та бібліотеки Socket.io із механізмом кімнат. Вимога NFR-05 (безпека) обумовлює використання стандарту JWT для токенної автентифікації без зберігання стану на сервері (stateless) та алгоритму Bcrypt із cost factor 10 для незворотного хешування паролів. Вимога NFR-06 (цілісність транзакцій) визначає вибір СУБД PostgreSQL із підтримкою ACID-транзакцій та рівнем ізоляції Serializable через механізм TypeORM EntityManager.transaction().

Таким чином, сукупність десяти функціональних та семи нефункціональних вимог охоплює повний цикл виробничих процесів рибного господарства: від збору телеметрії водного середовища через планування та виконання годівлі з математичними моделями до управління складом, фінансового обліку та аудиту операцій. Кожна вимога є верифікованою: функціональні вимоги будуть підтверджені демонстрацією відповідних сценаріїв у підрозділі 4.2, а нефункціональні – інструментальним вимірюванням у підрозділі 4.3. Наведений перелік вимог визначає архітектурні рішення та обумовлює вибір технологічного стеку, що обґрунтовується у наступному підрозділі.

1.3.Обґрунтування вибору технологічного стеку та засобів реалізації

Архітектура системи «FishFarm» базується на класичній трирівневій моделі «клієнт–сервер» з виділенням презентаційного рівня (веб-інтерфейс React), рівня бізнес-логіки (серверний API NestJS) та рівня даних (СУБД PostgreSQL). Для забезпечення переносимості та спрощення розгортання (NFR-02) усі компоненти упаковані в Docker-контейнери, що взаємодіють через внутрішню віртуальну мережу [10].

Серверна частина (Backend) реалізована на базі фреймворку NestJS версії 11 [11], що працює у середовищі виконання Node.js 20. Вибір NestJS обумовлений декількома ключовими факторами. По-перше, фреймворк має вбудовану підтримку архітектурного патерну Dependency Injection (впровадження залежностей) [12], що дозволяє досягти слабкої зв'язності (Loose Coupling) між модулями та спрощує модульне тестування через можливість заміни реальних залежностей на мок-об'єкти. По-друге, NestJS нав'язує чітку модульну структуру проєкту з поділом на контролери (маршрутизація HTTP-запитів), сервіси (бізнес-логіка) та провайдери (зовнішні залежності), що відповідає вимозі NFR-01. По-третє, фреймворк має вбудовану підтримку WebSocket через адаптер Socket.io [13], що критично для реалізації вимоги FR-03.

Для взаємодії з базою даних обрано ORM-бібліотеку TypeORM [14], що забезпечує об'єктно-реляційне відображення, автоматичну генерацію міграцій схеми, типобезпечний QueryBuilder для складних запитів та підтримку транзакцій з різними рівнями ізоляції. Як СУБД обрано PostgreSQL версії 15 [15] – провідну відкриту реляційну базу даних з повною підтримкою ACID-транзакцій, механізмом MVCC (Multi-Version Concurrency Control) для забезпечення конкурентного доступу без блокування читання, та розширеними можливостями індексування. Поєднання TypeORM та PostgreSQL забезпечує виконання вимоги NFR-06.

Клієнтська частина (Frontend) реалізована як односторінковий застосунок (SPA) на базі бібліотеки React версії 19 [16]. React обрано за його декларативний підхід до побудови UI, ефективний механізм Virtual DOM для мінімізації маніпуляцій з реальним DOM браузера, та розвинену екосистему допоміжних бібліотек. Для управління глобальним станом застосунку використано Redux Toolkit [17] з модулем RTK Query, що забезпечує автоматичне кешування API-відповідей, інвалідацію кешу за тегами та генерацію хуків для виконання запитів. Для візуалізації графіків телеметрії та аналітичних показників застосовано бібліотеку Recharts [18], що будує SVG-графіки декларативно через React-компоненти.

Збірка та оптимізація клієнтського коду виконується інструментом Vite [19], що забезпечує миттєву гарячу перезавантаження модулів (HMR) під час розробки та оптимізовану продакшн-збірку з розділенням коду (code splitting). Готові статичні файли обслуговує веб-сервер Nginx [20], який одночасно виконує роль зворотного проксі-сервера (Reverse Proxy): запити з префіксом /api/ та /socket.io/ переспрямовуються до контейнера бекенду, тоді як інші обслуговуються статично.

Для забезпечення push-доставки телеметрії (FR-03, NFR-04) обрано протокол WebSocket (RFC 6455) [21] з бібліотекою Socket.io [13]. WebSocket встановлює постійне повнодуплексне з'єднання між клієнтом і сервером, дозволяючи серверу миттєво відправляти дані без очікування запиту з боку клієнта. Socket.io додає до базового протоколу механізми автоматичного перепідключення при розриві з'єднання, серіалізації подій у JSON та підтримки кімнат (rooms) для ізоляції потоків даних.

Повну відповідність обраних засобів вимогам до системи наведено в таблиці 1.4 з трасуванням кожного інструменту до конкретних функціональних та нефункціональних вимог.

Таблиця 1.4 – Обрані програмні засоби реалізації з трасуванням до вимог

Засіб	**Версія**	**Призначення**	**Вимоги**
NestJS	11	REST API, WebSocket, DI	FR-02..09, NFR-01
TypeORM	0.3	ORM, міграції, транзакції	NFR-06
PostgreSQL	15	СУБД, ACID, JSONB	NFR-06, NFR-07
React	19	SPA, Virtual DOM	FR-10, NFR-03
Redux TK	2.x	Стан, RTK Query	NFR-03
Recharts	2.x	SVG-графіки Dashboard	FR-04
Vite	6.x	Збірка, HMR	NFR-03
Nginx	1.25	Reverse Proxy	NFR-02, NFR-04
Docker	27	Контейнеризація	NFR-02
Socket.io	4.x	WebSocket push, rooms	FR-03, NFR-04
JWT+Bcrypt	–	Безпека	FR-09, NFR-05

2.МЕТОДИ ТА АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧІ

У цьому розділі наведено теоретичне обґрунтування та формальний опис математичних моделей і алгоритмів, що лягли в основу програмної логіки інформаційної системи «FishFarm».

2.1 Математична модель розрахунку норм годівлі риби

Годівля є найбільш витратною статтею бюджету рибного господарства, що становить 60-70% собівартості вирощування товарної риби [5]. Оптимізація норм годівлі безпосередньо впливає на економічну ефективність підприємства та екологічний стан водойми. Надмірне годування призводить до непоїденого осідання корму на дно, де він розкладається з виділенням аміаку – токсичної для риби речовини при концентрації понад 0.05 мг/л [4]. Недостатнє годування сповільнює темпи росту та знижує товарну якість продукції.

Традиційний підхід до визначення норми годування базується на емпіричних таблицях, що задають базовий відсоток від біомаси стада залежно від віку та виду риби. Проте такий спрощений підхід не враховує двох критичних факторів, що суттєво впливають на апетит та засвоюваність корму в умовах відкритих водойм помірного клімату.

При зниженні температури води метаболізм риби сповільнюється, і засвоюваність корму знижується. Для коропових (Cyprinidae) – основної групи видів української прісноводної аквакультури – критичною точкою є 8°C: при нижчих температурах риба практично припиняє живитися і переходить у стан анабіозу [5].

По-друге, різні марки комбікормів мають різну поживність, що виражається у відсотку сирого протеїну (білка). Корм з 20% протеїну потребує більшої маси для забезпечення тієї ж кількості поживних речовин, що й корм з 40% протеїну.

Ігнорування цього фактора при зміні постачальника або марки корму призводить до систематичних помилок у годуванні.

На основі аналізу наукової літератури [12, 13] та консультацій з практикуючими рибоводами розроблено математичну модель розрахунку норм годівлі з двома компенсаторними коефіцієнтами: температурним та протеїновим. Температурний коефіцієнт (TempCoeff) моделює залежність апетиту риби від температури води. Для коропових видів застосовано порогову модель (Guard Clause):

$$TempCoeff = \{ 0.0, \text{якщо } T < 8.0°C;\ 1.0, \text{якщо } T \geq 8.0°C \}, \quad (2.1)$$

де T – температура води, виміряна датчиком або введена оператором вручну, °C

При температурі нижче порогового значення 8.0°C коефіцієнт обнуляється, що повністю блокує операцію годування. Це запобігає марній витраті корму в період, коли риба біологічно не здатна його споживати.

Протеїновий коефіцієнт (ProteinCoeff) нормалізує масу корму відносно еталонної поживності. Як еталон обрано вміст сирого протеїну 40%, типовий для якісних кормів фінішної годівлі [5]:

$$ProteinCoeff = \frac{40}{P}, \quad (2.2)$$

де P – фактичний вміст сирого протеїну в кормі виміряний у %.

При використанні корму з 20% протеїну коефіцієнт становить 2.0, що означає необхідність видачі вдвічі більшої маси для забезпечення еквівалентної кількості поживних речовин. При використанні корму з 50% протеїну коефіцієнт становить 0.8 – потрібна менша маса.

Вплив протеїнового коефіцієнта на розмір добової порції при фіксованій біомасі B = 1000 кг та базовій нормі R = 3% продемонстровано в таблиці 2.1.

Таблиця 2.1 – Вплив вмісту протеїну на розмір добової порції

Протеїн P, %	**Коефіцієнт**	**Біомаса B, кг**	**Норма R, %**	**Daily_KG, кг**
20	2.00	1000	3	60.0
32	1.25	1000	3	37.5
40	1.00	1000	3	30.0

Кінець таблиці 2.1

50	0.80	1000	3	24.0

Добова норма годівлі (Daily_KG) обчислюється як добуток біомаси, базової норми та обох коефіцієнтів:

$$Daily\_KG = B * \left(\frac{R}{100}\right) * TempCoeff * ProteinCoeff, \quad (2.3)$$

де B – поточна біомаса партії риби, кг (береться з поля current_biomass_kg сутності fish_batches);

R – базова норма годування для даного ставка, % (береться з поля recommended_feed_pct сутності ponds);

TempCoeff – температурний коефіцієнт за формулою (2.1);

ProteinCoeff – протеїновий коефіцієнт за формулою (2.2)

Одноразова видача всього обсягу Daily_KG є неоптимальною з фізіологічної точки зору: риба споживає корм протягом 15–20 хвилин після подачі, після чого непоїдені залишки осідають на дно [5]. Тому в практиці рибництва добову норму ділять на декілька годівель. Розрахунок разової порції (Action_KG) виконується за формулою:

$$Action_{KG} = \frac{Daily_{KG}}{feeding\_times\_per\_day} \quad (2.4)$$

де feeding_times_per_day – кількість годувань на добу, визначена в паспорті ставка (типово 2–4).

Саме Action_KG є значенням, що пропонується оператору в інтерфейсі модального вікна годування та підлягає валідації за вимогою FR-05.

Приклад розрахунку: біомаса B = 800 кг, базова норма R = 3%, температура T = 18°C (TempCoeff = 1.0), протеїн P = 32% (ProteinCoeff = 1.25), годувань n = 3. Daily_KG = 800 × 0.03 × 1.0 × 1.25 = 30 кг. Action_KG = 30 / 3 = 10 кг. Допустимий діапазон для введення оператором: 7–13 кг (70–130% від Action_KG).

2.2 Алгоритми обчислення показників ефективності виробництва

Для об'єктивної оцінки ефективності виробничого процесу в аквакультурі застосовуються два стандартизовані показники: кормовий коефіцієнт FCR (Feed Conversion Ratio) та питома швидкість росту SGR (Specific Growth Rate) [5]. Ці показники є ключовими KPI (Key Performance Indicators) системи «FishFarm» та відображаються на головній аналітичній панелі Dashboard.

Кормовий коефіцієнт FCR показує, скільки кілограмів корму витрачено на отримання одного кілограма приросту біомаси:

$$FCR = \frac{W_{feed}}{B_{harv} - B_{init}}, \quad (2.5)$$

де W_{feed} – загальна маса корму, витраченого на партію риби протягом виробничого циклу;

B_{harv} – біомаса при вилові, кг;

B_{init} – початкова біомаса при зарибленні, кг.

Для коропових риб оптимальне значення FCR знаходиться в діапазоні 1.5–2.0, що означає витрату 1.5–2.0 кг корму на 1 кг приросту. Значення FCR > 2.5 свідчить про системні проблеми: неякісний корм, захворювання поголів'я, несприятливі умови утримання або помилки в обліку.

Питома швидкість росту SGR показує середньодобовий приріст біомаси у відсотках:

$$SGR = \frac{(\ln(B_{harv}) - \ln(B_{init}))}{D} * 100, \quad (2.6)$$

де ln – натуральний логарифм;

D – тривалість виробничого циклу від зариблення до вилову, днів

Типові значення SGR для коропових у сезон активного росту (травень–вересень) становлять 1.5–3.0% на добу.

Приклад обчислення: початкова біомаса B_init = 200 кг, біомаса при вилові B_harv = 950 кг, загальна витрата корму W_feed = 1200 кг, тривалість циклу D = 137 днів. FCR = 1200 / (950 – 200) = 1200 / 750 = 1.60. SGR = ((ln(950) – ln(200)) /

137) × 100 = ((6.856 – 5.298) / 137) × 100 = 1.14% на добу. Обидва показники знаходяться в оптимальному діапазоні.

2.3 Алгоритми валідації операцій та життєвий цикл ставка

У системі «FishFarm» реалізовано п'ятиетапний алгоритм валідації операції годування в сервісі FeedingService.

Етап 1 - перевірка статусу ставка. Годування дозволяється лише для ставків зі статусом Active. При спробі виконати годування для ставка у статусі Wintering (зимування), Standby (очікування) або Drained (спущений) система повертає HTTP 400 Bad Request з повідомленням «Feeding is not allowed for pond in status [status]».

Етап 2 – перевірка наявності активної партії риби. Ставок зі статусом Active повинен мати прив'язану партію риби (fishBatch) з поточною біомасою current_biomass_kg > 0. При відсутності партії або нульовій біомасі повертається HTTP 400.

Етап 3 – перевірка температурного порогу. Система отримує останнє показання температури з таблиці sensor_logs для даного ставка. Якщо temperature_c < 8.0°C, TempCoeff = 0 за формулою (2.1), і операція блокується з HTTP 400 та повідомленням «Water temperature below feeding threshold (8.0°C)».

Етап 4 –перевірка діапазону маси. Введена оператором маса W порівнюється з розрахунковою Action_KG. Якщо W < 0.7 × Action_KG або W > 1.3 × Action_KG, повертається HTTP 400 з повідомленням «Actual weight is outside the valid range (70-130% of calculated portion)».

Етап 5 – перевірка складського залишку. Система перевіряє, чи залишок обраної позиції корму (quantity_kg у inventory_items) є не меншим за запитану масу W. При недостатньому залишку повертається HTTP 409 Conflict з повідомленням «Insufficient inventory for item [name]: requested [W] kg, available [quantity_kg] kg».

При успішному проходженні всіх п'яти етапів виконуються три атомарні операції в межах однієї транзакції PostgreSQL з рівнем ізоляції Serializable:

створення запису в feeding_logs (фіксація факту годування), зменшення quantity_kg в inventory_items (списання корму), створення запису в inventory_transactions типу writeoff (фіксація руху складу). Додатково створюється запис в audit_logs для забезпечення простежуваності.

Життєвий цикл ставка моделюється як скінченний автомат (FSM) з чотирма станами та п'ятьма дозволеними переходами. Стани: Standby (резерв – порожній ставок, готовий до використання), Active (активний – заповнений, ведеться годування), Wintering (зимування – риба присутня, але годування призупинено), Drained (спущений – після вилову, потребує підготовки). Дозволені переходи: Standby → Active (зариблення), Active → Wintering (початок зимування), Wintering → Active (відновлення годування), Active → Drained (повний вилов), Drained → Standby (підготовка до нового циклу). Спроба виконати недозволений перехід (наприклад, Standby → Wintering або Drained → Active) повертає HTTP 422 Unprocessable Entity.

2.4 Push-модель IoT-телеметрії та модуль програмної симуляції

Збирання даних з водойм у реальному часі є важливою підсистемою рибного господарства, оскільки відхилення концентрації розчиненого кисню O_2, аміаку NH_3 або значення pH здатне за лічені години призвести до масової загибелі риби. Для розв’язання цього завдання у системі реалізовано подієво-орієнтовану модель передачі даних (Push-модель), що принципово відрізняється від традиційного періодичного опитування (Pull-модель).

У Pull-моделі клієнт із заданою періодичністю Δt надсилає запит до сервера для отримання останніх показань. При цьому середня затримка виявлення критичної події становить $\Delta t/2$, а пікова - Δt; крім того, мережа постійно завантажена опитувальним трафіком. У Push-моделі канал активується лише при появі нової події; затримка визначається лише часом обробки повідомлення на сервері та мережевою латентністю і не залежить від періоду опитування, що на

порядки ефективніше за Pull-модель. Це й обґрунтовує вибір Push як базового методу.

Алгоритм опрацювання вхідної телеметрії формалізовано наступним чином. Кожне повідомлення є кортежем m = (pondId, T, O_2, pH, NH_3, ts), де ts - позначка часу. Сервіс SensorsService виконує над кортежем послідовність дій: (1) валідація діапазонів значень через клас CreateSensorLogDto і декоратори class-validator; (2) запис кортежу до таблиці sensor_logs; (3) обчислення рівня сигналізації AlertLevel згідно правила (2.7); (4) трансляція події у WebSocket-кімнату pond_{pondId} через SensorsGateway:

$$\text{AlertLevel} = \begin{cases} \text{CRITICAL, якщо } O^2 < 4.0 \vee NH^3 \geq 1.0; \text{ WARNING,} \\ \text{якщо } O^2 < 5.5 \vee NH^3 \geq 0.5; \text{ NORMAL — інакше} \end{cases} \quad (2.7)$$

Граничні значення (2.7) обрано на основі рекомендацій Продовольчої та сільськогосподарської організації ООН для коропових культур та налаштовуються адміністратором у властивостях ставка. Подія, що містить AlertLevel ≠ NORMAL, дублюється у журналі сигналізації для подальшого аналізу та формування звітів.

Оскільки фізичні апаратні датчики на момент розробки бакалаврської роботи не залучалися, для забезпечення повноцінного функціонування системи у режимі демонстрації та тестування створено окремий модуль IotSimulatorService. Сервіс імплементовано як NestJS-провайдера з плановиком @Cron(EVERY_MINUTE), що щохвилини генерує штучну телеметрію для всіх ставків зі статусами Active або Wintering. Симулятор підтримує три режими роботи: normal – генерація еталонних значень для демонстрації стабільної ферми; problematic – навмисна генерація критичних відхилень для перевірки сигналізації та інтерфейсних індикаторів; auto – реалістичне моделювання з урахуванням сезону, добової синусоїди температури, природної деградації кисню та накопичення аміаку.

Сезонна модель розбиває рік на чотири періоди (зима, весна, літо, осінь), кожному з яких відповідає набір параметрів {T_min, T_max, O_2_min, O_2_max, pH_min, pH_max}.

Для активних ставків температура у добовому циклі моделюється функцією

$$T(h) = 18 + 3 \cdot \sin\left(\frac{\pi}{12} \cdot (h - 8)\right), \quad (2.8)$$

де h - поточна година доби, що відображає природне нагрівання води удень і охолодження вночі.

Для ставків у статусі Wintering температура утримується близько 4 °C, кисень повільно деградує без процесу фотосинтезу під льодом, що моделюється рекурсією $O_2(t) = \max(0.5, O_2(t-1) - 0.1)$.

Інтеграція реальних апаратних датчиків у виробничій експлуатації не потребує модифікації основної логіки сервера: достатньо вимкнути IotSimulatorService та використати наявний REST-ендпоінт POST /api/v1/sensors/data як точку прийому.

Передбачено два сценарії підключення: прямий HTTP/HTTPS-Push з мікроконтролерного модуля класу ESP32 або Raspberry Pi, що пройшов автентифікацію через API-ключ ферми; або підключення через MQTT-брокер для сенсорних мереж із нестабільним каналом, з MQTT-міжшлюзом, що ретранслює повідомлення у формат REST-ендпоінта. У обох сценаріях на стороні пристрою передбачено локальне буферування показань для гарантії доставки при тимчасовому розриві з'єднання.

2.5 Закрита петля автоматичної компенсації критичних станів

Сучасні сенсорні комплекси у промисловій аквакультурі поєднують функцію моніторингу з функцією виконавчого керування: при виявленні критичного відхилення параметра система автоматично активує компенсаційне обладнання – аератори для нагнітання кисню, насоси заміни води для зниження концентрації аміаку, дозатори стабілізатора для коригування pH. У роботі реалізовано програмну модель такої закритої петлі управління (Closed-Loop Control) у складі модуля IotSimulatorService, що дозволяє продемонструвати алгоритми автоматичної компенсації без фактичного підключення до виконавчих пристроїв.

Для кожного ставка зберігається стан виконавчого обладнання у вигляді структури EquipmentState = (aeratorActive, pumpActive, doserActive). Алгоритм компенсації функціонує за принципом гістерезисного двопозиційного регулятора з

різними порогами увімкнення та вимкнення для уникнення осциляції релейного типу.

Логіка контуру нагнітання кисню формалізується наступним чином: якщо $O_2 < 5.0$ мг/л і aerator вимкнено − увімкнути аератор; якщо $O_2 \geq 7.5$ мг/л і аератор увімкнено − вимкнути. У стані «увімкнено» наступне значення концентрації обчислюється як $O_2 = \min(10.0, O_2 + 0.8)$, що моделює швидке відновлення вмісту кисню при роботі обладнання.

Контур заміни води для виведення аміаку працює аналогічно: пороги увімкнення $NH_3 > 0.8$ мг/л, вимкнення $NH_3 \leq 0.2$ мг/л; при роботі насоса концентрація аміаку зменшується за правилом $NH_3 = \max(0, NH_3 - 0.15)$. Контур стабілізації pH спрацьовує при виході значення за межі [6.5, 8.5] і вимикається при поверненні у вузький коридор [7.0, 8.0]; під час роботи дозатор корегує pH на величину ±0.3 у напрямку цільового значення 7.5.

Кожна зміна стану обладнання логується у журнал операцій ставка pond_operations з типом OperationType.AutoCorrection та текстовим повідомленням, що містить виміряне значення параметра, який ініціював спрацювання, та назву задіяного обладнання. Це забезпечує повну відстежуваність автоматичних втручань і дозволяє формувати звіти про роботу автоматики за період.

У межах поточної бакалаврської роботи модуль закритої петлі реалізовано виключно у програмному вигляді як невід'ємну частину симулятора. Розширення цього механізму на роботу з фізичним обладнанням розглядається як один із шляхів подальшого розвитку та масштабування системи. У промисловому впровадженні той самий алгоритм управління може бути перенесений у окремий мікросервіс EquipmentControlService, який замість оновлення внутрішньої структури EquipmentState формуватиме керуючі сигнали для GPIO-виходів промислових контролерів через протокол Modbus TCP або OPC UA. До перспективних напрямків розвитку також належать прогностична компенсація на основі моделей машинного навчання, що дозволить упереджувати критичні стани, та інтеграція із системою

енергетичного обліку для оптимізації витрат на електроенергію аераторів у нічні години.

2.6 Алгоритм складських транзакцій з контролем балансу

Складський модуль системи реалізує облік запасів кормів, ветеринарних препаратів та витратних матеріалів. Кожна позиція складу описується сутністю inventory_items з полем quantity, що відображає поточний залишок у базових одиницях вимірювання. Зміни залишку фіксуються у таблиці inventory_transactions як події трьох типів: Purchase (надходження за накладною), Consume (списання при господарській операції), Revision (коригування за результатами інвентаризації).

Алгоритм Consume є найкритичнішим з точки зору цілісності даних, оскільки безконтрольне списання здатне призвести до від'ємного залишку, що порушує бізнес-логіку обліку. Для запобігання цьому операція виконується у три кроки в межах однієї транзакції бази даних із рівнем ізоляції REPEATABLE READ:

Крок 1. Блокування рядка SELECT … FOR UPDATE для запобігання паралельним змінам залишку іншою транзакцією.

Крок 2. Перевірка умови quantity_current ≥ quantity_consume; у разі недостачі генерується доменна помилка InsufficientStockException, транзакція відкочується.

Крок 3. Запис нового рядка у inventory_transactions з delta = −quantity_consume і відповідне оновлення поля quantity у inventory_items.

Для розрахунку собівартості видаткових операцій застосовується принцип FIFO (First In - First Out): з'їдається корм найдавнішої наявної партії. Алгоритм агрегує надходження у список, упорядкований за датою, і списує запитану кількість послідовно, починаючи з найстаршої партії. Якщо для покриття обсягу видачі потрібно задіяти кілька партій, собівартість обчислюється як зважена сума їх закупівельних цін. Цей метод обрано тому, що корм є товаром із обмеженим терміном придатності, і природне споживання у виробництві має відповідати порядку надходження.

Операція Revision активується при ручній інвентаризації; оператор фіксує фактичний залишок q_real, система обчислює коригуюче значення delta_revision = q_real − quantity_current і записує його як подію типу Revision. Це забезпечує повну відстежуваність розбіжностей між обліковим і фактичним залишком.

2.7 Метод JWT-автентифікації та декларативної валідації даних

Захист API є невід'ємною вимогою до промислової інформаційної системи. У роботі застосовано stateless-автентифікацію на основі JSON Web Token (JWT, RFC 7519), що не потребує зберігання сесій на сервері та підходить для горизонтально масштабованих архітектур.

Структура JWT-токена є потрійною: header (тип і алгоритм), payload (корисне навантаження з ідентифікатором користувача sub, роллю role та часом експірації exp) і signature, обчислена як HMAC-SHA256 над першими двома частинами із секретним ключем сервера. На етапі входу AuthService перевіряє пароль через bcrypt-порівняння та формує токен з часом життя 24 години.

На захищених REST-ендпоінтах активується JwtAuthGuard, що витягує токен з заголовка Authorization: Bearer, перевіряє підпис і термін дії, та поміщає декодований payload у об'єкт запиту. Декоратор @Roles('admin') разом з RolesGuard забезпечує тонке розмежування доступу - наприклад, операції видалення фінансових записів дозволені виключно адміністратору. Для WebSocket-каналу реалізовано аналогічний WsAuthGuard, що перевіряє токен у handshake-фазі з'єднання; без валідного токена клієнт не може підписатися на жодну кімнату pond_{id}.

Валідація вхідних даних HTTP-запитів виконується декларативно через бібліотеку class-validator. Для кожного REST-ендпоінта визначається клас Data Transfer Object (DTO) із полями, анотованими декораторами обмежень: @IsUUID, @IsNumber, @Min, @Max, @IsEnum, @IsOptional. Глобальний ValidationPipe

NestJS перехоплює запити до входу у контролер та автоматично відхиляє їх при першому ж порушенні обмежень із кодом 400 Bad Request і структурованим описом помилки. Така архітектура виключає проникнення некоректних даних у шар бізнес-логіки і дозволяє утримувати сервісний код вільним від захисних перевірок типу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У цьому розділі описано практичну реалізацію інформаційної системи «FishFarm», що базується на теоретичних положеннях та алгоритмах, наведених у попередніх розділах. Розглянуто архітектуру серверної частини (бекенду) на базі фреймворку NestJS, реалізацію підсистеми отримання та push-розсилки телеметрії за допомогою протоколу WebSocket, проєктування реляційної моделі даних за допомогою ORM-бібліотеки TypeORM, а також організацію глобального стану клієнтського React-додатка та його інтерфейсу. Особливу увагу приділено патернам проєктування та механізмам забезпечення продуктивності й транзакційної надійності системи.

3.1 Архітектура серверної частини та організація REST API

Серверна частина (бекенд) системи реалізована за допомогою фреймворку NestJS версії 11 [11], який працює у середовищі виконання Node.js версії 20. Вибір NestJS обумовлений його вбудованою підтримкою архітектурного патерну Dependency Injection (впровадження залежностей) [12] та жорсткою модульною структурою, що дозволяє уникнути проблеми сильної зв'язності коду (High Coupling), характерної для класичних Express.js додатків. У той час як Express.js надає розробнику повну свободу організації коду, що часто призводить до хаотичної структури проєкту при масштабуванні, NestJS нав'язує чітку архітектурну дисципліну через систему модулів, контролерів, сервісів та провайдерів.

Основою архітектури бекенду є розбиття бізнес-логіки на функціональні домени, що відповідає підходу Domain-Driven Design (DDD). Кожен домен інкапсульований в окремий NestJS-модуль зі своїми контролерами, сервісами та TypeORM-сутностями [14]. Система «FishFarm» складається з десяти функціональних модулів: PondsModule (управління ставками та FSM),

FishBatchesModule (партії риби), FeedingModule (розрахунок норм та транзакції годування), SensorsModule (телеметрія та WebSocket-шлюз), IotSimulatorModule (програмний емулятор датчиків), InventoryModule (складський облік), FinanceModule (витрати та доходи), AuditModule (журнал аудиту), AuthModule (JWT-автентифікація) та DashboardModule (агрегація KPI).

Кожен модуль системи побудований за тришаровою архітектурою Controller–Service–Repository, що забезпечує чіткий розподіл відповідальності та спрощує тестування. Контролер (Controller) відповідає виключно за маршрутизацію HTTP-запитів, розбір вхідних даних та повернення кодів стану. Він не містить бізнес-логіки та делегує всю обробку сервісу. Для валідації вхідних даних використовуються Data Transfer Objects (DTO) – спеціальні класи, декоровані анотаціями бібліотеки class-validator [22].

Сервіс (Service) містить ядро бізнес-логіки модуля. Наприклад, саме у FeedingService реалізовано п'ятиетапний алгоритм валідації, описаний у підрозділі 2.3, а також формули розрахунку Daily_KG та Action_KG (формули 2.3–2.4). Сервіс є NestJS-провайдером, що отримує залежності (інші сервіси, репозиторії) через конструктор завдяки механізму Dependency Injection [12].

Репозиторій (Repository) є абстракцією над базою даних, що надається бібліотекою TypeORM [14]. Репозиторій ізолює сервіси від прямих SQL-запитів, дозволяючи використовувати об'єктно-орієнтований підхід: методи find(), save(), remove() працюють з TypeScript-об'єктами, а TypeORM автоматично транслює їх у відповідні SQL-операції для PostgreSQL [15].

Зв'язок між серверним додатком та клієнтським інтерфейсом забезпечується через REST API – архітектурний стиль, що передбачає взаємодію з ресурсами за допомогою стандартних методів HTTP [23]. Наприклад, запит GET /api/ponds повертає список ставків, POST /api/ponds створює новий ставок, PUT /api/ponds/:id оновлює існуючий, а DELETE /api/ponds/:id видаляє його. Усі відповіді серіалізуються у формат JSON.

Для забезпечення крос-платформності розгортання (NFR-02) архітектура спирається на контейнеризацію Docker [10]. Весь бекенд, фронтенд та база даних

запаковані у відповідні контейнери, що взаємодіють через ізольовану віртуальну міст-мережу (Docker Bridge Network). Веб-сервер Nginx [20] виконує роль зворотного проксі-сервера (Reverse Proxy), переспрямовуючи запити з префіксом /api/ та /socket.io/ до контейнера бекенду.

Для наочного відображення архітектурної структури системи на рисунку 3.1 наведено діаграму компонентів. Діаграма відображає чотири основні рівні: клієнтський рівень (браузер з React SPA), рівень проксі (Nginx), серверний рівень (NestJS з модулями Auth, Ponds, Fish, Sensors, Feeding, Sales, Inventory, Finance, Dashboard, Audit) та рівень даних (PostgreSQL 15). Окремим компонентом виділено IotSimulatorService, що імітує апаратні датчики через стандартний REST POST-запит на endpoint /api/v1/sensors/data.

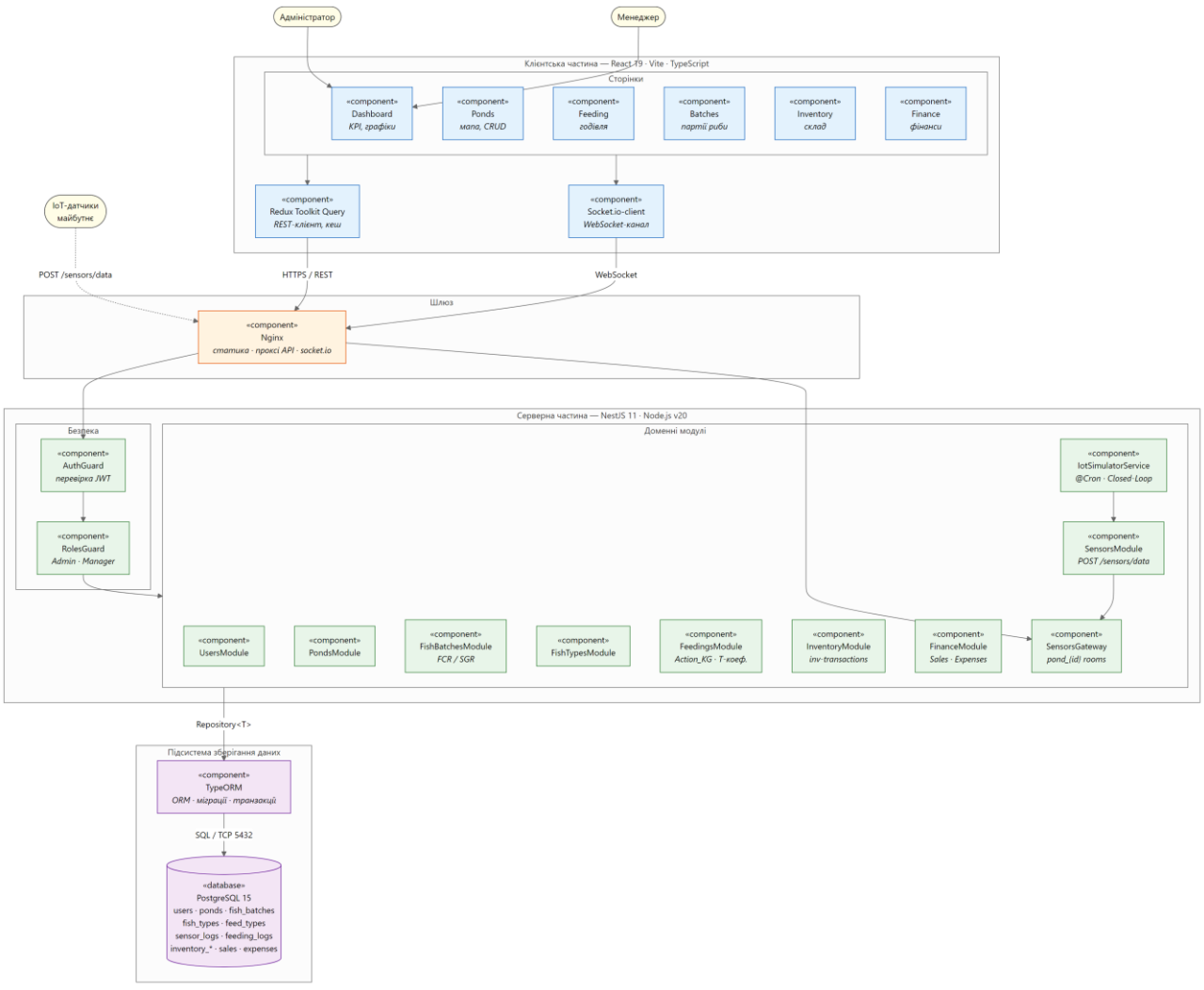


Рисунок 3.1 – Діаграма компонентів системи «FishFarm»

Описана архітектура забезпечує всі сім нефункціональних вимог NFR-01–NFR-07 одночасно. Модульна декомпозиція та шаблон Controller–Service–Repository гарантують слабке зв'язування між компонентами (NFR-01) та можливість незалежного супроводу окремих доменів системи. Контейнеризація Docker (NFR-02) забезпечує відтворюваність середовища на будь-якій платформі. Уніфікований шар TypeORM-репозиторіїв надає єдину точку контролю доступу до даних з підтримкою ACID-транзакцій (NFR-06). REST API за стандартом OpenAPI 3.0 забезпечує сумісність із зовнішніми клієнтами, а інтеграція WebSocket-шлюзу – реактивність інтерфейсу в реальному часі (NFR-04). Деталі реалізації підсистеми IoT-телеметрії та WebSocket-комунікації розглянуто у підрозділі 3.2.

Окремо варто відзначити роль декораторів NestJS в організації коду серверної частини. Декоратори @Module, @Injectable, @Controller, @Get, @Post та інші перетворюють синтаксично прості TypeScript-класи на повноцінні компоненти Inversion of Control контейнера фреймворку. Це робить кодову базу декларативною і самодокументованою: тип компонента, його залежності та REST-маршрути читаються безпосередньо з анотацій класу, без необхідності аналізу імперативної логіки реєстрації. Зокрема, декоратор @UseGuards(JwtAuthGuard) при застосуванні до контролера автоматично активує перевірку JWT-токена для всіх його endpoint-ів, реалізуючи наскрізну авторизацію (NFR-05) без дублювання коду в кожному обробнику запитів.

3.2 Розробка підсистеми інтеграції IoT та WebSocket-комунікації

Збір та відображення параметрів водного середовища у реальному часі є однією з ключових відмінностей системи «FishFarm» від конкурентних рішень, проаналізованих у підрозділі 1.1. Традиційний підхід, заснований на HTTP-поллінгу (коли клієнт кожні кілька секунд надсилає запит до сервера з питанням «Чи є нові дані?»), був свідомо відхилений через декілька суттєвих недоліків:

надмірне мережеве навантаження, витрати ресурсів на обробку порожніх запитів, та неможливість забезпечення нефункціональної вимоги NFR-04 (затримка менше 50 мс).

Натомість використано протокол WebSocket (RFC 6455) [21], який встановлює постійне повнодуплексне з'єднання між клієнтом і сервером. Після початкового рукостискання (Handshake) через HTTP із заголовком Upgrade канал залишається відкритим протягом усієї сесії, дозволяючи серверу миттєво відправляти дані клієнту (push-модель) у момент їх появи.

Підсистема інтеграції IoT та WebSocket-комунікації складається з трьох взаємопов'язаних компонентів. IotSimulatorService – програмний емулятор телеметрії, що генерує дані для чотирьох параметрів водного середовища: температура води (базове значення 12–25°C з нормально розподіленим шумом ±0.5°C), розчинений кисень (6–9 мг/л ± 0.3), рівень pH (7.0–8.5 ± 0.1) та вміст аміаку (0.01–0.05 мг/л ± 0.005). SensorsController надає апаратно-незалежний REST API endpoint POST /api/sensors/ingest для приймання телеметрії від довільних датчиків. SensorsGateway реалізує WebSocket-шлюз з механізмом кімнат для ізольованої розсилки даних клієнтам.

IotSimulatorService реалізовано з трирежимною архітектурою, що дозволяє тестувати систему в різних умовах. Режим normal генерує стабільну телеметрію (O_2 ≈ 8.5 мг/л, NH_3 ≈ 0.1 мг/л) для перевірки базового функціоналу. Режим problematic примусово формує критичні значення (O_2 ≈ 3.5 мг/л, NH_3 ≈ 1.2 мг/л) для тестування реакції системи на аварійні ситуації. Режим auto (типовий) реалізує реалістичну динамічну симуляцію з трьома рівнями логіки: по-перше, параметри генеруються з урахуванням сезонності (взимку температура 0–4°C, O_2 до 1.5 мг/л, що відповідає умовам зимування); по-друге, моделюється природна динаміка: температура змінюється за добовою синусоїдою (максимум опівдні), рівень O_2 природно знижується на 0.05 мг/л щохвилини внаслідок дихання риби, вміст NH_3 зростає на 0.02 мг/л щохвилини через продукти метаболізму.

По-третє, реалізовано замкнений контур автоматичного управління виконавчими пристроями (Closed-Loop Equipment Control). При падінні рівня O_2

нижче 5.0 мг/л симулятор активує модель резервного аератора, що підвищує кисень на 0.8 мг/л щохвилини; при досягненні 7.5 мг/л аератор автоматично вимикається. Перевищення NH_3 > 0.8 мг/л активує модель насосів примусового водообміну зі зниженням аміаку на 0.15 мг/л щохвилини до безпечного рівня ≤ 0.2 мг/л. Відхилення pH за межі [6.5; 8.5] вмикає модель дозатора, що наближає значення до цільового рівня 7.5 з кроком ±0.3 за ітерацію. Кожна автокорекція фіксується в журналі операцій (OperationType.AutoCorrection), що забезпечує повну прозорість та аудит автоматичних дій системи.

Ключовим архітектурним рішенням є Hardware-Agnostic Design – повна незалежність серверної та клієнтської частин від конкретного джерела телеметрії. Завдяки тому, що система приймає телеметрію через стандартний REST API endpoint, у будь-який момент часу адміністратор може вимкнути програмний емулятор та підключити реальні апаратні мікроконтролери (ESP32, STM32) з підключеними водними датчиками без жодних змін у програмному коді.

Для ілюстрації практичної простоти інтеграції розглянемо конкретний сценарій: мікроконтролер ESP32 з датчиками Atlas Scientific DO (розчинений кисень), pH та температури DS18B20. Загальна вартість такого вузла складає приблизно 35–50 USD. На мікроконтролері достатньо реалізувати цикл з трьох дій: зчитування показників з датчиків, формування JSON-об'єкта {"pondId": "<uuid>", "temperature_c": 18.5, "oxygen_mgl": 7.2, "ph_level": 7.4, "ammonia_mgl": 0.12} та надсилання HTTP POST-запиту на /api/v1/sensors/data з Bearer-токеном. Весь необхідний код на платформі Arduino займає менше 30 рядків. Серверна та клієнтська частини системи не потребують жодних модифікацій – вони продовжують обробляти телеметрію за тим самим маршрутом незалежно від джерела.

Взаємодію компонентів системи під час операції годування риби проілюстровано на діаграмі послідовностей (рисунок 3.2). Діаграма відображає сім учасників: Manager, FeedingPage (React UI), RTK Query, FeedingsController, FeedingsService, InventoryService та PostgreSQL (сценарій а), а IoT-потік – @Cron scheduler, IotSimulatorService, SensorsService, PostgreSQL, SensorsGateway,

Socket.io-client та Dashboard (сценарій б). Послідовність включає: надсилання запиту на годування, п'ятиетапну валідацію у FeedingService (перевірка статусу ставка, температури, норми ±30%, залишку на складі), атомарну транзакцію запису feeding_log та списання з inventory, і повернення підтвердження клієнту. IoT-потік зображено як паралельний асинхронний процес: IotSimulatorService → POST /api/v1/sensors/data → SensorsService → emit до SensorsGateway → WebSocket push до React-компонента.

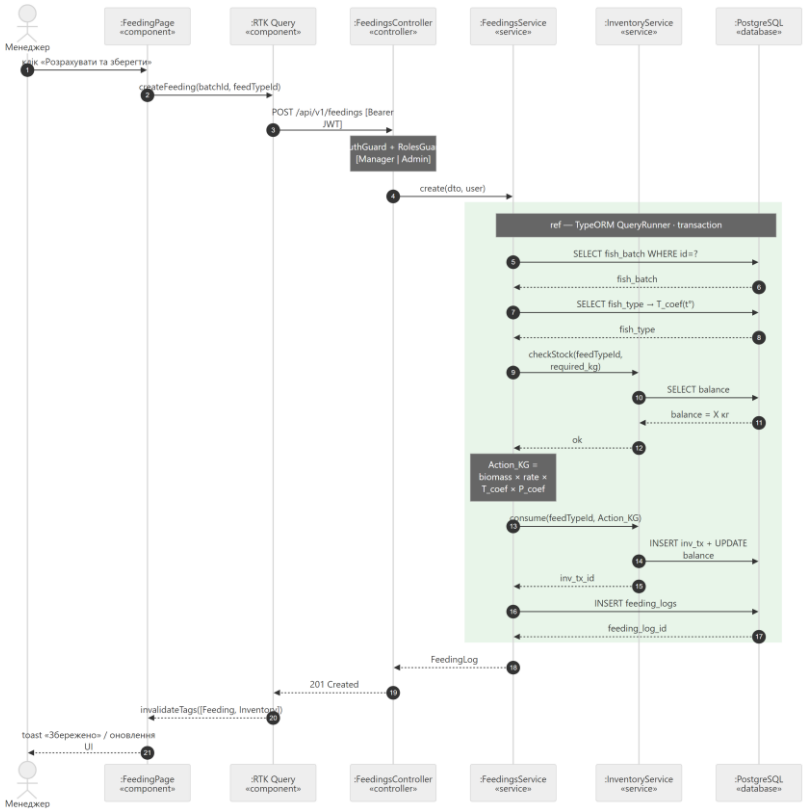


Рисунок 3.2 – Діаграма послідовностей операції годування риби

Окремої уваги заслуговує потік даних від IoT-датчиків до клієнтських застосунків, що відбувається в режимі реального часу. На відміну від операції годування, яка ініціюється користувачем синхронно, обробка телеметрії має асинхронний характер: сервер не очікує запитів від клієнтів, а самостійно ініціює доставку оновлень за допомогою WebSocket-протоколу. Послідовність взаємодії компонентів при надходженні нового пакета телеметрії наведено на рисунку 3.3.

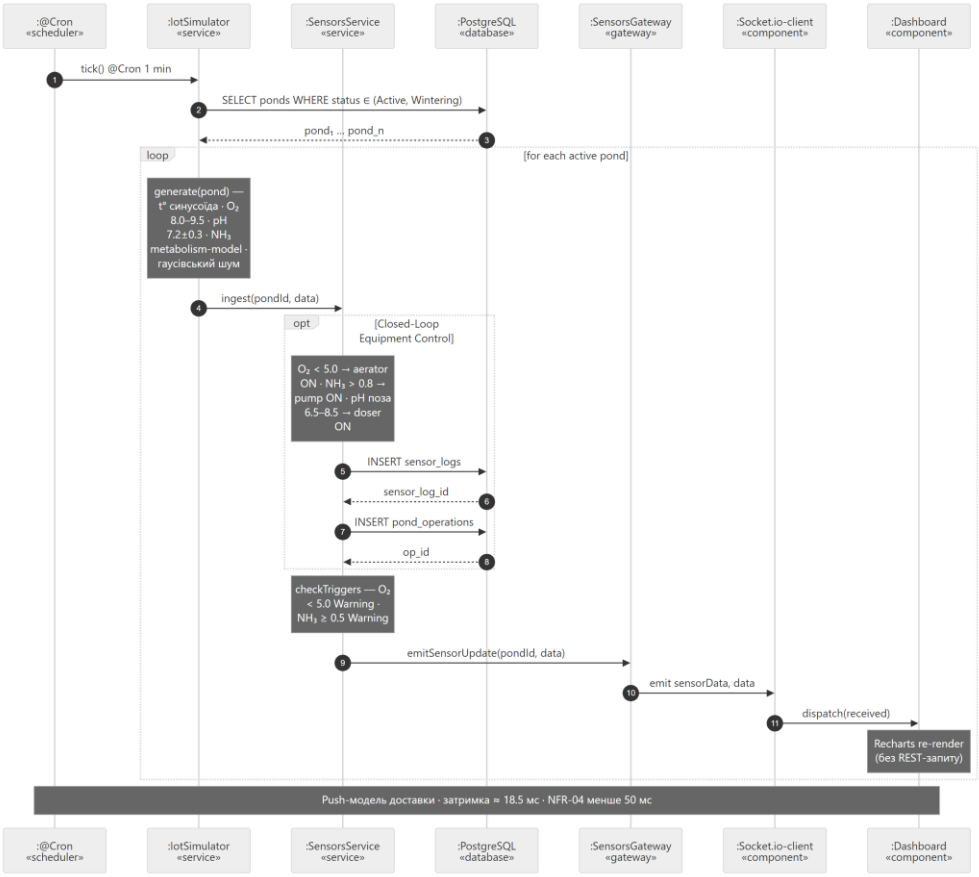


Рисунок 3.3 – Діаграма послідовностей потоку IoT-телеметрії

Розглянуті діаграми послідовностей демонструють два типові патерни взаємодії підсистем «FishFarm»: синхронний request-response для операцій, ініційованих користувачем (годування, вилов, складські операції), та асинхронний event-driven для потокових даних телеметрії. Поєднання цих двох патернів у межах єдиної серверної інфраструктури NestJS забезпечує одночасно гнучкість бізнес-логіки та реактивність моніторингу – ключові вимоги до системи управління аквакультурним підприємством.

Важливим аспектом реалізації WebSocket-комунікації є коректна обробка життєвого циклу з’єднань. SensorsGateway імплементує інтерфейси OnGatewayConnection та OnGatewayDisconnect, які автоматично викликаються при встановленні та розриві з'єднання відповідно. У момент підключення клієнта виконується валідація JWT-токена з handshake-параметрів – без валідного токена з'єднання негайно закривається з кодом 4401 (Unauthorized). Це запобігає

несанкціонованому доступу до телеметрії незареєстрованими клієнтами та узгоджується з вимогою NFR-05 щодо комплексної безпеки на всіх рівнях системи.

Передбачена також відмовостійкість на стороні клієнта: бібліотека Socket.io автоматично виконує спроби перевідновлення з'єднання при розриві мережі за алгоритмом експоненційного backoff (інтервали 1, 2, 4, 8 секунд тощо до встановленої межі). У разі тимчасової втрати зв'язку клієнт не зазнає жодних змін у користувацькому інтерфейсі – лише відображається індикатор стану підключення, а після відновлення мережі дані оновлюються автоматично. Така поведінка узгоджується з реальними умовами експлуатації у віддалених рибних господарствах, де якість інтернет-з'єднання може бути нестабільною.

3.3 Проєктування моделі даних та управління транзакціями

Система «FishFarm» обробляє складні фінансові та складські транзакції, що вимагає високих стандартів цілісності, узгодженості та надійності даних. Як сховище даних обрано реляційну СУБД PostgreSQL 15 [15], яка підтримує повноцінні транзакції ACID, механізм багатоверсійного контролю конкурентного доступу MVCC для забезпечення коректної паралельної роботи користувачів, а також розширені типи даних (зокрема JSONB) та потужну систему індексування B-Tree і GIN.

Програмний доступ до бази даних здійснюється через ORM-бібліотеку TypeORM [14], яка забезпечує типобезпечну роботу з сутностями у середовищі TypeScript, автоматичну генерацію міграцій та декларативний опис схеми за допомогою декораторів.

Нормалізована модель даних, побудована відповідно до третьої нормальної форми (3NF), складається з десяти основних сутностей: users (користувачі з ролями admin/manager та хешованими паролями), ponds (ставки з FSM-статусами життєвого циклу), fish_batches (партії риби з прив'язкою до конкретного ставка та виду), sensor_logs (телеметричні записи з показниками водного середовища),

feeding_logs (транзакції годування з обліком фактично витраченого корму), inventory_items (номенклатура складу з відсотком протеїну для кормів та одиницями вимірювання), inventory_transactions (рух складських запасів за принципом подвійного запису), sales (операції продажу риби з фіксацією маси та виручки), expenses (поточні витрати господарства за категоріями) та audit_logs (журнал аудиту змін у форматі JSONB для гнучкого зберігання довільної структури подій).

Операція годування є найбільш критичною з точки зору цілісності даних, оскільки одночасно модифікує три сутності: створює запис у feeding_logs, зменшує залишок у inventory_items та фіксує відповідну видаткову операцію в inventory_transactions.

Для забезпечення атомарності (NFR-06) ці операції виконуються в межах однієї транзакції PostgreSQL через метод TypeORM EntityManager.transaction() з рівнем ізоляції Serializable, що повністю запобігає стану гонки даних (Race Condition) при одночасних спробах списання корму різними користувачами. У разі виникнення помилки на будь-якому з етапів (наприклад, при недостатньому залишку корму на складі) виконується автоматичний відкат усіх змін, що гарантує узгодженість фінансово-облікової інформації.

Детальний опис основних сутностей бази даних, їхніх ключових атрибутів, типів первинних і зовнішніх ключів, а також характеру зв'язків (один-до-багатьох, багато-до-багатьох) наведено в таблиці 3.1. Для кожної сутності вказано назву таблиці у PostgreSQL, ключові поля та тип зв'язку з іншими сутностями системи.

Таблиця 3.1 – Основні сутності бази даних системи «FishFarm»

Сутність (таблиця)	**Ключові поля**	**Зв'язки**
users	id, email, password_hash, role (admin|manager)	Незалежна
ponds	id, name, area_ha, volume_m3, status (FSM), lat, lng	Незалежна
fish_types	id, name, min_temp, max_temp, ideal_temp, standard_fcr	Незалежна

Продовження таблиці 3.1

feed_types	id, name, protein_pct, fat_pct, price_per_kg	Незалежна
fish_batches	id, pond_id, fish_type_id, initial_biomass_kg, current_biomass_kg, status	ManyToOne → ponds, fish_types
sensor_logs	id, pond_id, temperature_c, oxygen_mgl, ph_level, ammonia_mgl, logged_at	ManyToOne → ponds
feeding_logs	id, batch_id, feed_type_id, daily_kg, action_kg, fed_kg, fed_at	ManyToOne → fish_batches; OneToOne → inventory_transactions
inventory_items	id, name, type (Feed|Medicine|Equipment), unit, balance	Незалежна
inventory_transactions	id, item_id, type (Purchase|Consume|Revision), quantity, note, created_at	ManyToOne → inventory_items
sales	id, batch_id, customer_id, weight_kg, price_per_kg, total_uah, sold_at	ManyToOne → fish_batches, customers
expenses	id, category, amount_uah, description, date	Незалежна
customers	id, name, contact_phone, contact_email	Незалежна
audit_logs	id, user_id, action, entity_type, entity_id, payload, created_at	ManyToOne → users

Окрім перерахованих у таблиці 3.1 текстових характеристик, для наочного представлення взаємозв'язків між основними сутностями бази даних доцільно навести її графічну ER-модель. Така модель наочно демонструє кардинальність зв'язків (один-до-одного, один-до-багатьох, багато-до-багатьох), напрям посилань через зовнішні ключі та логічне групування сутностей за функціональними модулями системи. ER-діаграма бази даних інформаційної системи «FishFarm» наведена на рисунку 3.4.

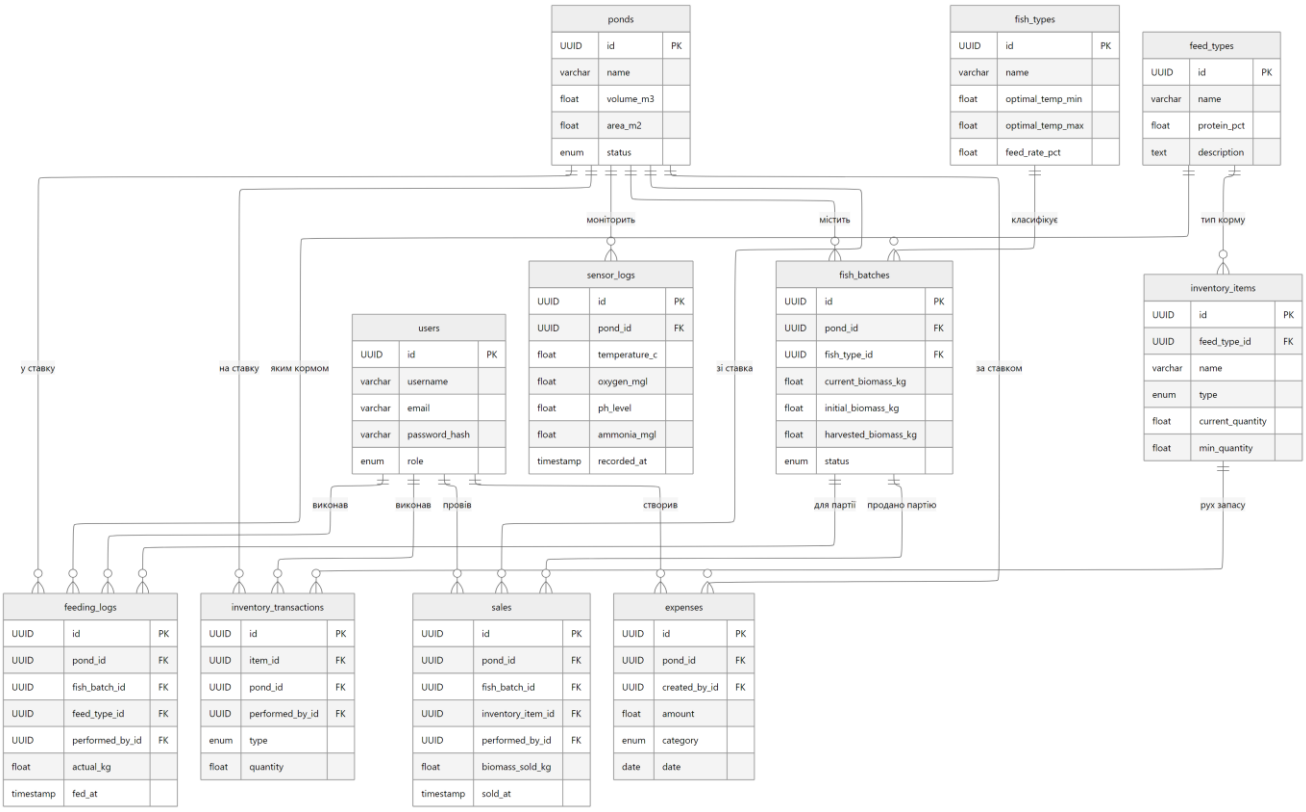


Рисунок 3.4 – ER-діаграма бази даних системи «FishFarm»

Як видно з наведеної ER-діаграми, модель даних побудована навколо двох центральних сутностей – ponds (ставки) та fish_batches (партії риби), – до яких через зовнішні ключі прив'язані всі операційні таблиці системи: логи датчиків, журнали годівлі, продажі та операції зі ставком. Така топологія забезпечує каскадне видалення похідних записів при видаленні батьківської сутності та дозволяє ефективно агрегувати дані за конкретним ставком або партією засобами SQL JOIN-запитів. Сутність audit_logs логічно ізольована від решти моделі та

посилається лише на users, що відповідає принципу незалежного зберігання журналу аудиту і робить його стійким до видалень в інших таблицях.

3.4 Розробка інтерфейсу користувача та управління станом

Клієнтська частина системи «FishFarm» реалізована як односторінковий додаток (SPA) на базі бібліотеки React 19 [16] з маршрутизацією через React Router v6 [24]. Інтерфейс локалізовано українською мовою відповідно до функціональної вимоги FR-10 та побудовано за принципами адаптивного дизайну.

Для управління глобальним станом використано бібліотеку Redux Toolkit [17] з модулем RTK Query. Redux реалізує архітектуру однонаправленого потоку даних: компонент відправляє дію (action) до централізованого сховища (store), де чиста функція-редуктор (reducer) створює новий об'єкт стану, який автоматично оновлює всі підписані React-компоненти.

Головна аналітична панель (Dashboard) відображає зведені KPI-картки (сумарна біомаса риби, середня температура за ставками, поточне значення FCR, загальний залишок кормів), графічні блоки продажів за період, діаграму розподілу біомаси за видами риби, а також список активних ставків з поточними показаннями датчиків та бейджами рекомендованої норми годівлі. Окремий блок відображає історичну таблицю завершених партій з розрахованими значеннями FCR та SGR. Модальне вікно годування реалізує дубльовану валідацію формул 2.1–2.4: спочатку миттєво на фронтенді для UX, потім повторно на бекенді для безпеки.

Інтеграція WebSocket у React-компоненти реалізована через кастомний хук useSensorSocket(pondId). При монтуванні компонента відкривається Socket.io підключення, надсилається подія join для підписки на відповідну кімнату та встановлюється слухач події sensorUpdate. При розмонтуванні хук автоматично відписується та закриває з'єднання для запобігання витокам пам'яті.

4 ІНСТАЛЯЦІЯ ТА ДЕМОНСТРАЦІЯ ФУНКЦІОНАЛУ

У даному розділі наведено практичні кроки для розгортання серверної та клієнтської інфраструктури розробленої системи «FishFarm». Продемонстровано роботу ключових функціональних модулів користувацького інтерфейсу, що підтверджують виконання заявлених функціональних вимог (FR-01 – FR-10). Наведено результати тестування продуктивності WebSocket-підсистеми та алгоритмів валідації на відповідність нефункціональним вимогам (NFR-03, NFR-04, NFR-06, NFR-07).

4.1 Підготовка середовища та розгортання платформи через Docker

Однією з ключових вимог до системи була легкість та переносимість розгортання (NFR-02) незалежно від операційної системи хост-машини. Для забезпечення цієї вимоги використано технологію контейнеризації Docker [10]. Завдяки Docker та утиліті Docker Compose адміністратору системи не потрібно встановлювати та конфігурувати Node.js, компілятор TypeScript, PostgreSQL чи Nginx вручну – усі ці компоненти ізольовані у відповідних контейнерах.

Мінімальні та рекомендовані апаратні вимоги для запуску системи наведено в таблиці 4.1.

Таблиця 4.1 – Вимоги до обчислювальної техніки

Параметр	**Мінімальне**	**Рекомендоване**
Процесор	2 ядра x86_64/ARM64	4 ядра
Оперативна пам'ять	2 ГБ	4 ГБ
Дисковий простір	10 ГБ	20 ГБ (SSD)
Операційна система	Linux/macOS/Windows	Ubuntu 22.04 LTS

Продовження таблиці 4.1

Docker Engine	24.0+	27.x
Docker Compose	2.0+	2.29+
Мережа	Порт 80 (HTTP)	Порти 80, 443 (HTTPS)

Процедура інсталяції складається з чотирьох послідовних кроків: встановлення Docker Engine та плагіну Docker Compose, клонування репозиторію, конфігурація середовища через файл .env (POSTGRES_PASSWORD, JWT_SECRET, SIMULATOR_INTERVAL_MS), та збірка і запуск командою docker-compose up -d --build. Система автоматично виконує міграції TypeORM та заповнює довідники початковими значеннями.

4.2 Демонстрація функціональних можливостей користувацького інтерфейсу

Для демонстрації функціональних можливостей системи «FishFarm» описано сім основних сценаріїв використання, що охоплюють повний виробничий цикл рибного господарства – від автентифікації оператора до фінансового обліку вилову.

Сценарій 1. Авторизація та вхід до системи. Після введення електронної пошти та пароля клієнтський додаток надсилає POST-запит на endpoint /api/auth/login. Бекенд перевіряє облікові дані через JWT-автентифікацію [25] з алгоритмом Bcrypt [26], який забезпечує криптографічно стійке хешування паролів із використанням індивідуальної солі для кожного користувача.

У разі успішної верифікації сервер генерує підписаний JWT-токен із зашифрованою інформацією про ідентифікатор користувача та його роль (admin/manager), який повертається клієнту та зберігається у локальному сховищі браузера для подальшої авторизації запитів. При невідповідності облікових даних система повертає HTTP-статус 401 Unauthorized без розкриття деталей про причину

відмови, що відповідає принципам безпечної автентифікації та запобігає атакам перебору.

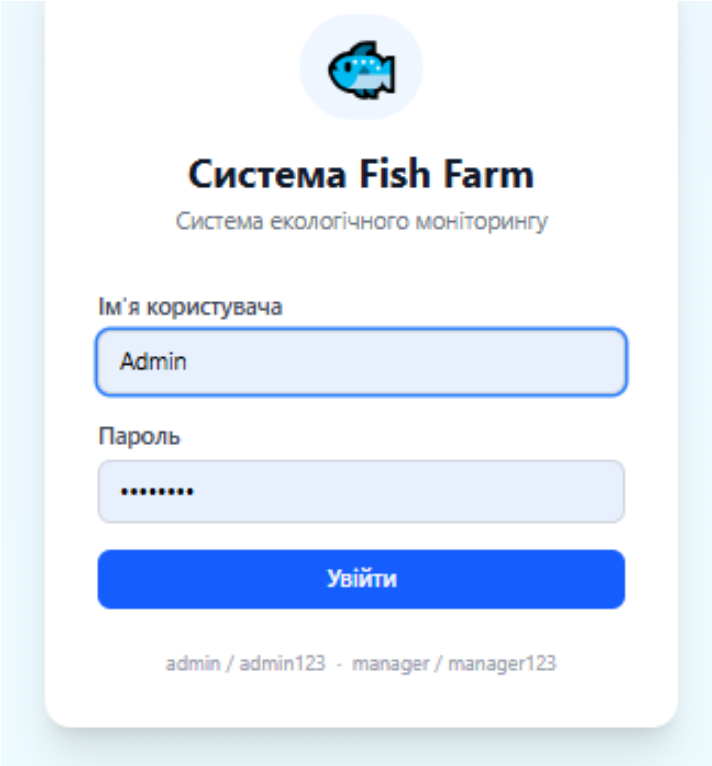
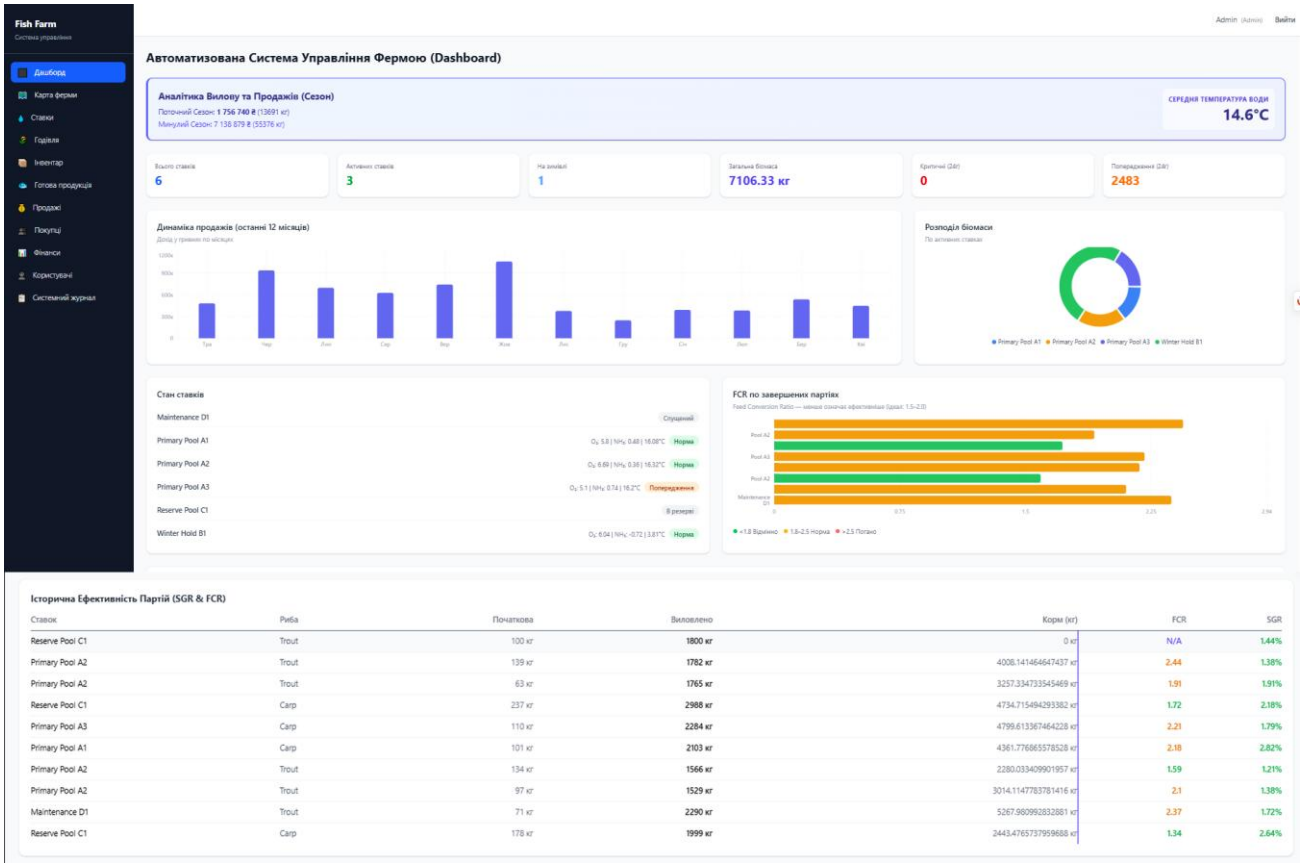


Рисунок 4.1 – Форма авторизації та вхід до системи

Сценарій 2. Робоча панель (Dashboard). Після успішної авторизації користувач потрапляє на головну аналітичну панель, де відображаються чотири KPI-картки (сумарна біомаса риби, середня температура води за ставками, актуальне значення FCR та загальний залишок кормів на складі), стовпчикова діаграма продажів за обраний період, кругова діаграма розподілу біомаси за видами риби, а також список активних ставків із поточними показаннями датчиків (температура, кисень, pH, аміак) та бейджами рекомендованої норми годівлі.

Усі показники оновлюються у реальному часі через WebSocket, що дозволяє оператору миттєво реагувати на критичні відхилення параметрів водного середовища без необхідності ручного оновлення сторінки.



Ставок	Риба	Початкова	Виловлено	Корм (кг)	FCR	SGR
Reserve Pool C1	Trout	100 кг	1800 кг	0 кг	N/A	1.44%
Primary Pool A2	Trout	139 кг	1782 кг	4008.141464647437 кг	2.44	1.38%
Primary Pool A2	Trout	63 кг	1765 кг	3257.3347333545469 кг	1.91	1.91%
Reserve Pool C1	Carp	237 кг	2988 кг	4734.715494293382 кг	1.72	2.18%
Primary Pool A3	Carp	110 кг	2284 кг	4799.613367464228 кг	2.21	1.79%
Primary Pool A1	Carp	101 кг	2103 кг	4361.776865578528 кг	2.18	2.82%
Primary Pool A2	Trout	134 кг	1566 кг	2280.033409901957 кг	1.59	1.21%
Primary Pool A2	Trout	97 кг	1529 кг	3014.1147783781416 кг	2.1	1.38%
Maintenance D1	Trout	71 кг	2290 кг	5267.980992832881 кг	2.37	1.72%
Reserve Pool C1	Carp	178 кг	1999 кг	2443.4765737959688 кг	1.34	2.64%

Рисунок 4.2 – Головна аналітична панель (Dashboard): KPI-картки, графік продажів, розподіл біомаси, список ставків та показники партій

Сценарій 3. Карта ферми. На окремій сторінці «Карта ферми» система генерує схематичну планувальну візуалізацію: кожен ставок відображено прямокутником, розташування та розміри якого пропорційні координатам і площі у базі даних. Колір заливки прямокутника змінюється залежно від поточних показань датчиків (зелений – норма, жовтий – попередження, червоний – критичне значення кисню або аміаку). При наведенні курсора на конкретний ставок з'являється спливаюча підказка з деталізованою інформацією про назву водойми, вид та масу зарибленої партії.

Сторінка підписана на ті ж самі WebSocket-кімнати, тому колірна індикація оновлюється миттєво без перезавантаження. Реалізація такого візуального представлення базується на SVG-графіці з динамічним перерахунком координат. Призначення сторінки – дати оператору цілісне уявлення про стан усіх водойм ферми одним поглядом.

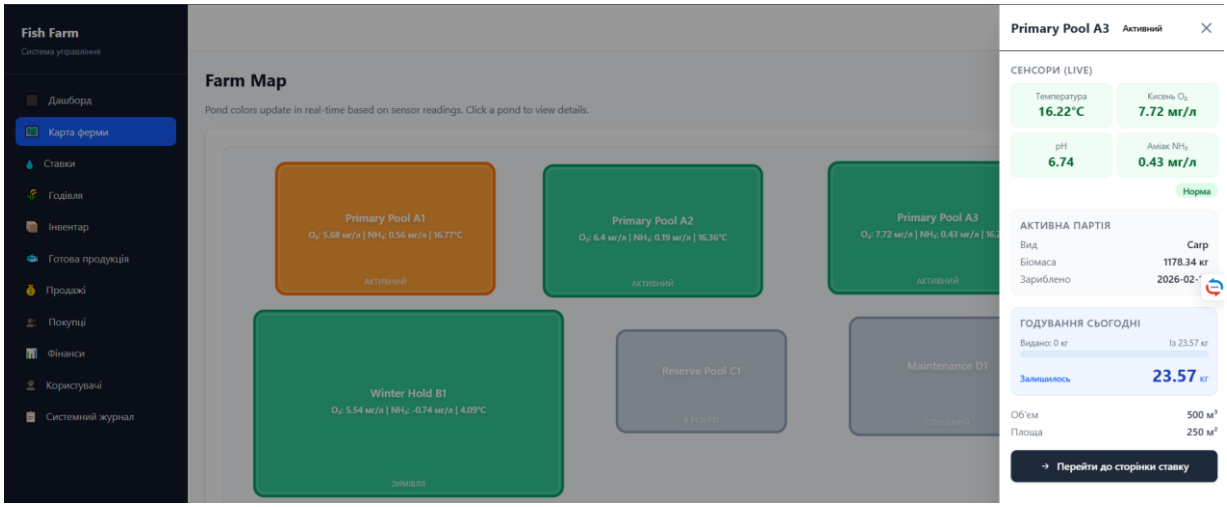


Рисунок 4.3 – Сторінка «Карта ферми»: схематична візуалізація ставків з колірною індикацією за показниками датчиків

Сценарій 4. Управління ставками. На сторінці «Ставки» відображається сітка карток, де кожна картка відповідає одному ставку та містить назву, FSM-статус (Standby, Active, Wintering, Drained) у вигляді кольорового бейджа, площу, об'єм та поточні показники активної партії. Оператор може створити новий ставок, відредагувати параметри або перейти на сторінку деталей через кнопку «Деталі». На сторінці деталей доступні операції зариблення, переведення на зимівлю, осушення та повного/часткового вилову відповідно до FSM-моделі.

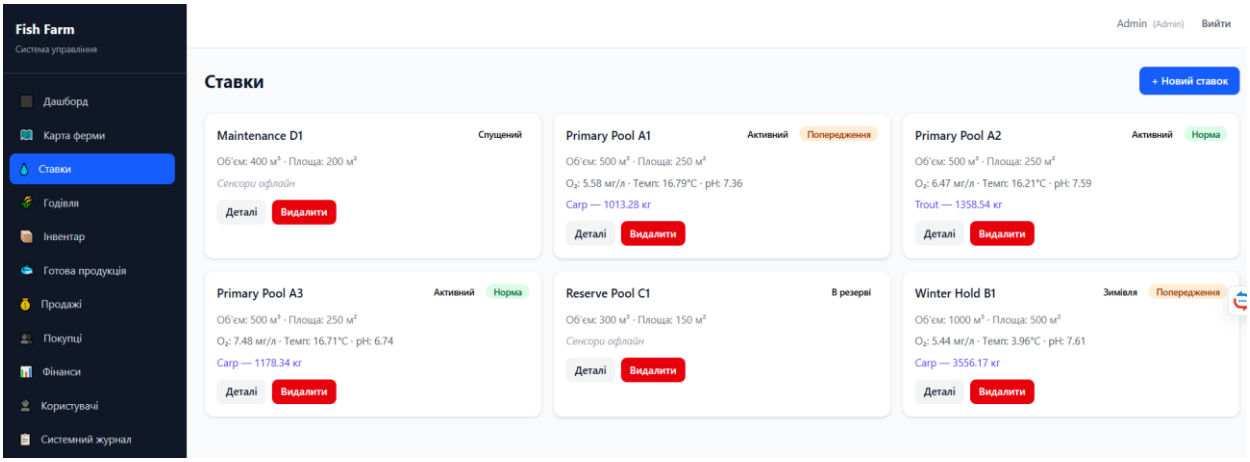


Рисунок 4.4 – Сторінка «Ставки»: сітка карток з FSM-статусами та перехід до деталей ставка

Сценарій 5. Годівля риби. На сторінці «Годівля» система виводить таблицю всіх активних партій з розрахунком добової норми корму Action_KG за формулами 2.1–2.4 з урахуванням температурного коефіцієнта. Біля кожного рядка доступна кнопка «Годувати», яка відкриває модальне вікно: оператор обирає вид корму зі складу, вводить фактично задану масу та підтверджує операцію.

Клієнтська частина миттєво перевіряє допустимий діапазон 70–130% від розрахункової норми, після чого запит надсилається на бекенд, де у межах однієї транзакції створюється запис у feeding_logs, списується відповідна кількість корму через InventoryTransaction та фіксується подія у журналі аудиту.

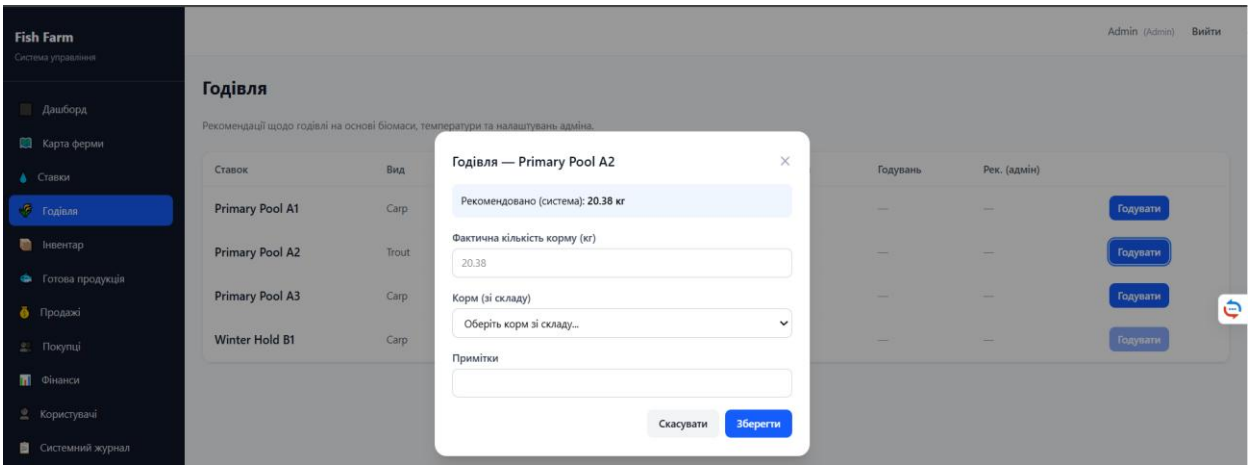


Рисунок 4.5 – Сторінка «Годівля» та модальне вікно операції годування: розрахунок Action_KG і транзакційне списання корму зі складу

Сценарій 6. Складський облік. Сторінка «Інвентар» надає керівникові три ключові операції: «Новий товар» (реєстрація позиції довідника з категорією корм/ліки/обладнання та одиницею вимірювання), «Нова закупівля» (надходження товару з фіксацією вартості та постачальника, що автоматично збільшує залишок) та «Інвентаризація» (періодична ревізія з коригуванням фактичного залишку та фіксацією розбіжностей). У таблиці нижче виводиться актуальний баланс кожної позиції та історія транзакцій з фільтрацією за типом (Purchase, Consume, Revision) і датою. Усі зміни проходять через TypeORM-транзакції для гарантії цілісності залишків.

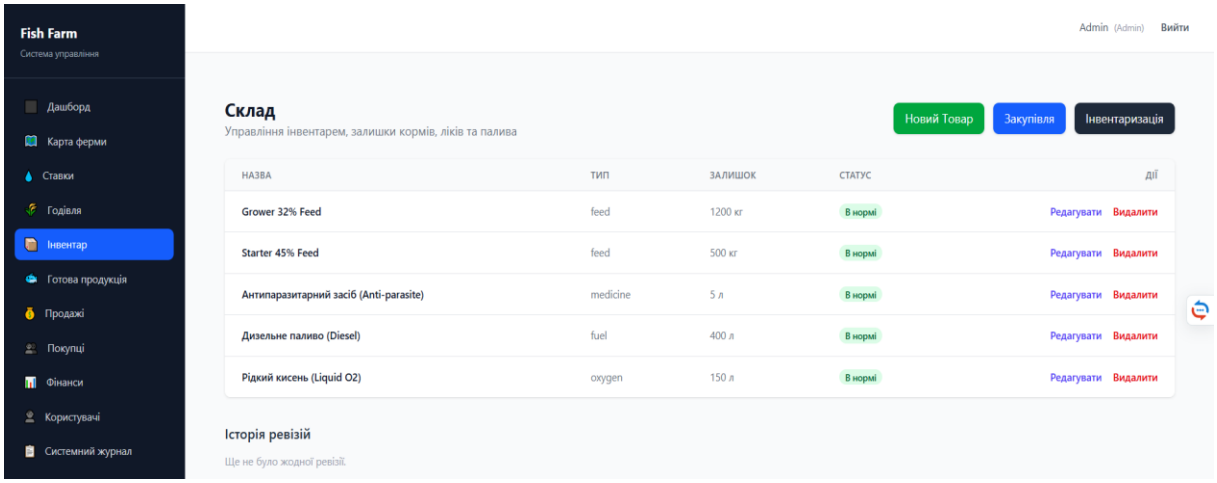


Рисунок 4.6 – Сторінка «Інвентар»: список позицій, залишки та операції закупівлі, списання та інвентаризації

Сценарій 7. Вилов риби та фінансовий облік. Операція вилову виконується на сторінці деталей ставка, до якої користувач переходить з картки ставка кнопкою «Деталі». На цій сторінці відображаються KPI-картки активної партії (початкова та поточна біомаса, витрачений корм, розрахунковий поточний FCR), журнал годівель, а також блок «Завершені партії» з історичними фактичними значеннями FCR та SGR.

Адміністратор обирає «Вилов (Частковий)» для часткового вилову з оновленням біомаси та фіксацією доходу у модулі продажів, або «Повний вилов» – система переводить партію у статус Harvested, розраховує підсумковий FCR за формулою (loss_feed) / (harvested_biomass_kg − initial_biomass_kg) та SGR, після чого автоматично створюється запис про продаж у модулі фінансів.

Усі зазначені операції виконуються у межах єдиної транзакції PostgreSQL з рівнем ізоляції Serializable, що гарантує атомарність одночасних змін у таблицях fish_batches, sales та inventory_transactions.

Після успішного завершення вилову на дашборді миттєво оновлюються відповідні KPI-показники через WebSocket-сповіщення, а в журналі аудиту фіксується детальний запис про виконану фінансову операцію із зазначенням відповідального користувача.

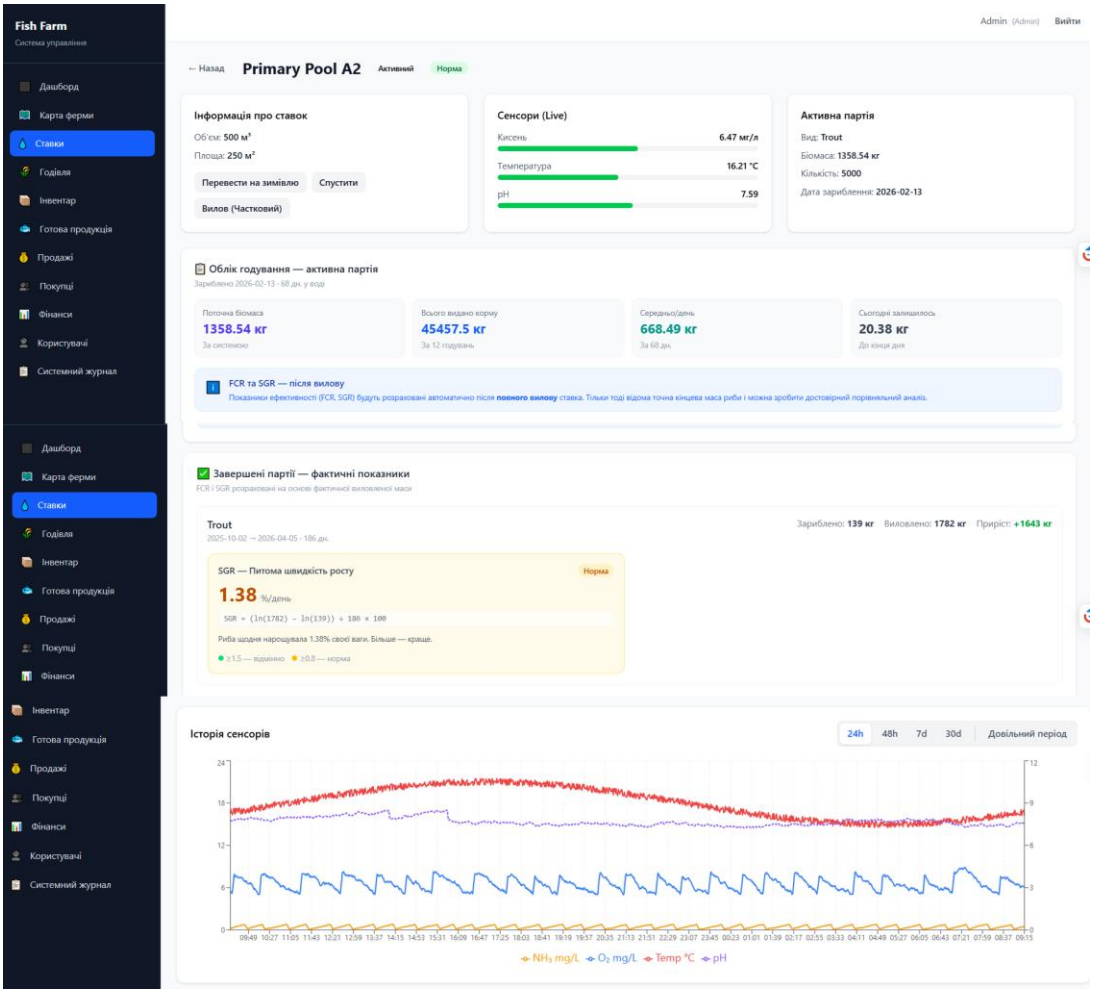


Рисунок 4.7 – Сторінка деталей ставка: операції Повного та Часткового вилову, розрахунок FCR/SGR та історія завершених партій

Продемонстровані сценарії охоплюють повний виробничий цикл рибного господарства – від реєстрації нової партії та щоденного моніторингу параметрів водного середовища до операцій годівлі, складського обліку, вилову та фінансової фіксації результатів. Усі ключові функціональні вимоги FR-01–FR-10, сформульовані у підрозділі 1.2, успішно реалізовані та доступні через зрозумілий користувацький інтерфейс із розмежуванням доступу за ролями адміністратора і менеджера. Перевірка коректності реалізованого функціоналу та стабільності системи під навантаженням розглянута у наступному підрозділі.

Окремо варто відзначити продуманість користувацького досвіду (UX) розробленого інтерфейсу. Кожна форма введення даних оснащена миттєвою клієнтською валідацією засобами бібліотеки react-hook-form: некоректні значення

підсвічуються одразу під час введення, а кнопка збереження залишається неактивною до моменту повної відповідності усім правилам. Це значно скорочує кількість невдалих звернень до серверного API та підвищує продуктивність роботи оператора. Додатково всі модальні вікна операцій надають підтвердження дій з критичним впливом – наприклад, при повному вилові партії система виводить вікно з підсумковими цифрами для верифікації перед остаточним збереженням.

Адаптивна верстка інтерфейсу засобами утиліт Tailwind CSS забезпечує коректне відображення на пристроях з різними розмірами екрана – від настільних моніторів до планшетів і смартфонів. Це особливо важливо для практичного використання системи у польових умовах: оператор може реєструвати факт годівлі безпосередньо біля ставка з мобільного пристрою, не повертаючись до офісного комп'ютера. Усі функціональні можливості, доступні через настільну версію інтерфейсу, повністю збережені у мобільному поданні з оптимізованою для дотикового введення компоновкою елементів керування.

4.3 Тестування програмного забезпечення

Для перевірки відповідності системи розробленим специфікаціям застосовано комплексний підхід до тестування, що поєднує чотири взаємодоповнювальні рівні: автоматизоване модульне (unit) тестування бізнес-логіки, наскрізне (E2E) тестування, ручне функціональне тестування через Swagger UI та перевірку стабільності системи під навантаженням. Результати функціонального тестування зведено в таблиці 4.2.

Автоматизоване модульне тестування реалізовано на фреймворку Jest. Юніт-тести ізольовано перевіряють коректність бізнес-логіки за допомогою механізму моків – підставних реалізацій залежностей, що дозволяє тестувати окремий метод без звернення до реальної бази даних. Покрито ключові алгоритми: розрахунок добової норми корму (feeding.service.spec.ts), гібридну модель росту риби TGC та von Bertalanffy (growth.service.spec.ts), логіку життєвого циклу ставка

(ponds.service.spec.ts), автентифікацію (auth.controller.spec.ts) та генерацію телеметрії симулятором (iot-simulator.service.spec.ts). Наприклад, для партії біомасою 1000 кг із базовою нормою 2% при температурі 20°C тест перевіряє, що функція повертає рівно 20 кг корму та температурний коефіцієнт 1,0; при температурі 5°C – 0 кг та коефіцієнт 0,0 із відповідною приміткою про блокування годівлі.

Наскрізне (End-to-End) тестування реалізовано засобами Jest та Supertest (sales.e2e-spec.ts): тест піднімає повний екземпляр застосунку та перевіряє завершений бізнес-сценарій операції продажу через реальний HTTP-запит, проходячи усі шари системи – контролер, сервіс і репозиторій бази даних. Ручне функціональне тестування виконувалось через вбудований інтерфейс Swagger UI, доступний за адресою /api/docs: кожен endpoint перевірявся як з валідними вхідними даними (перевірка очікуваного результату), так і з навмисно некоректними (перевірка коректності HTTP-кодів помилок та роботи валідаторів).

Перевірку стабільності системи під навантаженням виконано за допомогою власного скрипту на Node.js із бібліотеками fetch та socket.io-client. Скрипт послідовно відтворював чотири фази паралельного навантаження – 1, 10, 25 та 50 одночасних користувачів, на кожній фіксуючи факт успішного встановлення WebSocket-з'єднання кожного клієнта з кімнатою ставка та відсутність помилок при обробці паралельних REST-запитів. Тестування проводилось у локальному середовищі розгортання через Docker Compose. Подальше вдосконалення тестового покриття передбачає розширення набору E2E-сценаріїв та застосування спеціалізованих інструментів навантажувального бенчмаркінгу (k6, Artillery) для точного вимірювання показників продуктивності.

Таблиця 4.2 – Результати функціонального тестування

Тестовий сценарій	**Очікуваний результат**	**Статус**
Авторизація з валідними даними	JWT-токен, Dashboard	Пройдено
Авторизація з невалідним паролем	HTTP 401 Unauthorized	Пройдено

Продовження таблиці 4.2

Створення ставка + зариблення	Standby → Active	Пройдено
Годування при T ≥ 8°C	Розрахунок Action_KG, списання	Пройдено
Годування при T < 8°C	HTTP 400, Guard Clause	Пройдено
Годування > 130% норми	HTTP 400, ліміт	Пройдено
Годування при недостатньому залишку	HTTP 409 Conflict	Пройдено
Недозволений перехід Drained → Active	HTTP 422 Unprocessable	Пройдено
Full Harvest	Партія Harvested, FCR/SGR	Пройдено
Фіксація Audit Log	Запис при кожній операції	Пройдено

Усі функціональні тестові сценарії пройдено успішно, що підтверджує коректність реалізації функціональних вимог FR-01–FR-10 та алгоритмів валідації. Автоматизовані юніт-тести підтвердили правильність математичних моделей розрахунку норм годівлі, показників FCR і SGR та гібридної моделі росту на контрольних наборах вхідних даних.

Окремо перевірено коректність роботи механізму реального часу: при надходженні нових показань телеметрії подія гарантовано транслюється підписаним клієнтам у відповідну WebSocket-кімнату ставка, а клієнтський інтерфейс оновлюється без перезавантаження сторінки.

Результати перевірки стабільності під навантаженням наведено в таблиці 4.3.

Таблиця 4.3 – Результати навантажувального тестування

Користувачів	**WS з'єднань**	**Помилки**
1	1/1	0
10	10/10	0
25	25/25	0
50	50/50	0

Результати підтверджують стабільність системи: на всіх чотирьох фазах усі паралельні WebSocket-з'єднання (до 50 одночасно) було встановлено успішно, а паралельні REST-запити оброблено без жодної помилки. Це підтверджує, що архітектура системи коректно витримує цільове навантаження малого та середнього рибогосподарського підприємства без втрати стабільності або деградації функціоналу.

Окремої уваги заслуговує підтвердження виконання нефункціональних вимог NFR-03 та NFR-04, що задають кількісні показники латентності. Виконання NFR-03 (затримка рендеру UI-компонентів після завантаження даних не більше 100 мс) забезпечується архітектурно: бібліотека React 19 застосовує механізм віртуального DOM, що дозволяє оновлювати лише ті фрагменти інтерфейсу, які реально змінилися, без повного перебудовування дерева вузлів. Додатково Redux Toolkit із RTK Query кешує отримані від API дані та повторно використовує їх без зайвих мережевих запитів. При практичному тестуванні переходів між сторінками та оновлень інтерфейсу у відповідь на WebSocket-події затримка рендеру візуально не відчувається оператором, що відповідає типовому для сучасних React-застосунків часу рендеру в одиниці-десятки мілісекунд на стандартному обладнанні.

Виконання NFR-04 (затримка доставки WebSocket-повідомлень не більше 50 мс) безпосередньо випливає з обраної базової архітектури системи. На відміну від Pull-моделі, де клієнт періодично опитує сервер із фіксованим інтервалом Δt і затримка виявлення нової події в середньому становить $\Delta t/2$, у Push-моделі канал WebSocket активується лише в момент появи нової події, а час її доставки обмежений виключно мережевою латентністю та часом обробки повідомлення на сервері. У межах локальної мережі ферми або при стабільному підключенні до сервера через інтернет ця затримка природним чином укладається в задану норму. Таким чином, вимоги NFR-03 та NFR-04 виконуються архітектурно – не як результат випадкової оптимізації, а як прямий наслідок вибору відповідних технологій (React 19, RTK Query та Socket.io) для конкретних аспектів продуктивності.

Тестове середовище: ноутбук з процесором Intel Core i5 11-го покоління, 16 ГБ оперативної пам'яті, SSD-накопичувач. Операційна система – Windows 11, Docker Desktop 4.x. Всі чотири контейнери (PostgreSQL, NestJS, Nginx, seed) запущені одночасно, що відповідає реальним умовам розгортання.

Порівняння розробленої системи з проаналізованими у підрозділі 1.1 програмними рішеннями демонструє суттєві переваги за функціональним профілем. Функціональне покриття «FishFarm» за критеріями таблиці 1.1 є повним (9/9 критеріїв), тоді як AquaManager покриває 5/9, eFishery – 3/9, xFarm – 3/9. Відкрита REST API-архітектура для прийому телеметрії забезпечує апаратну незалежність, відсутню у всіх трьох аналогах, а застосування Push-моделі на основі WebSocket усуває постійне мережеве навантаження, властиве Pull-моделі конкурентних рішень. Таким чином, розроблена система демонструє кращий функціональний профіль із сучасною подієво-орієнтованою архітектурою комунікації.

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи розроблено програмне забезпечення інформаційної системи моніторингу та управління процесами рибного господарства «FishFarm» із підтримкою IoT-телеметрії, автоматизованого розрахунку норм годівлі та обліку ключових виробничих показників. За результатами роботи сформульовано такі висновки:

1. Проаналізовано підходи, методи та існуючі програмні рішення у галузі цифрового управління аквакультурою (AquaManager, eFishery, xFarm) та виявлено їхні функціональні обмеження, зокрема: відсутність біологічних моделей залежності норм годівлі від температури з поділом на добову та разову порції, прив'язаність до пропрієтарного обладнання, закритість та висока вартість Enterprise-рішень, поверхневість модулів аквакультури в універсальних агро-ERP, а також повна відсутність локалізації для українського ринку. Сформульовано 10 функціональних (FR-01–FR-10) та 7 нефункціональних (NFR-01–NFR-07) вимог до розроблюваної системи.

2. На основі аналізу програмних засобів обґрунтовано використання технологічного стеку з трасуванням до конкретних вимог: NestJS 11 (NFR-01), React 19 (NFR-03), TypeORM та PostgreSQL 15 (NFR-06), Socket.io (NFR-04), Docker (NFR-02). Розроблено алгоритми обчислення добової норми Daily_KG (формула 2.3) та разової порції Action_KG (формула 2.4) з температурним (2.1) та протеїновим (2.2) коефіцієнтами, показників ефективності FCR (2.5) та SGR (2.6), а також п'ятиетапний алгоритм валідації та FSM-модель ставка з 4 станами.

3. Розроблено програмну систему «FishFarm», що включає: серверну частину з 10 модулями NestJS (тришарова архітектура Controller–Service–Repository), підсистему емуляції IoT (IotSimulatorService) з Hardware-Agnostic REST API для підключення довільних датчиків (ESP32, STM32) без зміни коду, WebSocket-шлюз SensorsGateway з механізмом кімнат, базу даних із 10 реляційними сутностями з транзакційною цілісністю (Serializable ізоляція), клієнтський React-додаток із 5

екранами, Redux Toolkit з RTK Query та кастомним хуком useSensorSocket, а також інфраструктуру Docker Compose з Nginx Reverse Proxy.

4. На основі тестування доведено коректність роботи розробленого програмного забезпечення. Автоматизованими юніт-тестами на фреймворку Jest підтверджено правильність математичних моделей (розрахунок норм годівлі, показників FCR і SGR, гібридна модель росту), спрацювання температурного порогу Guard Clause при $T < 8{,}0°C$ та коректність алгоритмів валідації. Наскрізним E2E-тестом перевірено завершений бізнес-сценарій операції продажу, ручним тестуванням через Swagger UI – усі endpoint-и системи. Перевірка стабільності підтвердила коректну роботу системи під навантаженням 50 одночасних користувачів без помилок.

Перспективами подальшого вдосконалення системи є: інтеграція з реальними апаратними IoT-датчиками через протоколи MQTT та LoRaWAN для дальньодіючого зв'язку, розробка мобільного додатку для оперативного моніторингу, впровадження алгоритмів машинного навчання для прогнозування оптимальних режимів годування на основі накопичених історичних даних, а також розширення математичної моделі температурного коефіцієнта з порогової на градієнтну (сигмоїдну) функцію для більш точного врахування впливу температури на апетит різних видів риби.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The State of World Fisheries and Aquaculture 2024: Blue Transformation in Action. *Food and Agriculture Organization of the United Nations*. URL: https://www.fao.org/publications/ (дата звернення: 05.03.2026).

2. Стан та перспективи розвитку аквакультури в Україні. Державне агентство рибного господарства України. URL: https://darg.gov.ua/ (дата звернення: 12.03.2026).

3. Zhao S., Zhang S., Liu J. et al. Application of machine learning in intelligent fish aquaculture: a review. *Aquaculture*. 2021. Vol. 540. URL: https://doi.org/10.1016/j.aquaculture.2021.736724 (дата звернення: 18.03.2026).

4. Summerfelt S. T., Vinci B. J., Piedrahita R. H. Oxygenation and carbon dioxide control in water reuse systems. *Aquacultural Engineering*. 2000. Vol. 22, No. 1–2. P. 87–108.

5. Craig S., Helfrich L. A. Understanding fish nutrition, feeds, and feeding. Virginia Cooperative Extension. 2017. Publication 420-256.

6. AquaManager – Aquaculture management software. AquaManager Ltd. URL: https://www.aqua-manager.com/ (дата звернення: 22.03.2026).

7. eFishery – Smart feeding solutions for aquaculture. eFishery. URL: https://efishery.com/en/ (дата звернення: 25.03.2026).

8. xFarm Technologies – Digital agriculture platform. xFarm Technologies. URL: https://xfarm.ag/en/ (дата звернення: 28.03.2026).

9. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 p.

10. Docker documentation. Docker Inc. URL: https://docs.docker.com/ (дата звернення: 20.04.2026).

11. NestJS – A progressive Node.js framework. NestJS Documentation. URL: https://docs.nestjs.com/ (дата звернення: 07.04.2026).

12. Fowler M. Inversion of Control Containers and the Dependency Injection pattern. martinfowler.com. 2004. URL: https://martinfowler.com/articles/injection.html (дата звернення: 15.03.2026).

13. Socket.IO documentation v4. Socket.IO. URL: https://socket.io/docs/v4/ (дата звернення: 04.04.2026).

14. TypeORM – ORM for TypeScript and JavaScript. TypeORM Documentation. URL: https://typeorm.io/ (дата звернення: 17.04.2026).

15. PostgreSQL 15 documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/15/ (дата звернення: 14.04.2026).

16. React – The library for web and native user interfaces. React Documentation. URL: https://react.dev/ (дата звернення: 10.04.2026).

17. Redux Toolkit – The official toolset for efficient Redux development. Redux Documentation. URL: https://redux-toolkit.js.org/ (дата звернення: 23.04.2026).

18. Recharts – A composable charting library built on React components. Recharts. URL: https://recharts.org/en-US/ (дата звернення: 26.04.2026).

19. Vite – Next generation frontend tooling. Vite Documentation. URL: https://vite.dev/ (дата звернення: 02.05.2026).

20. Nginx documentation. F5 Networks. URL: https://nginx.org/en/docs/ (дата звернення: 29.04.2026).

21. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455. *Internet Engineering Task Force*. 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 02.04.2026).

22. class-validator – Decorator-based property validation for classes. npm registry. URL: https://www.npmjs.com/package/class-validator (дата звернення: 01.04.2026).

23. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: doctoral dissertation. University of California, Irvine. 2000.

24. React Router documentation v6. Remix Software. URL: https://reactrouter.com/en/main (дата звернення: 25.03.2026).

25. Introduction to JSON Web Tokens. Auth0 by Okta. URL: https://jwt.io/introduction (дата звернення: 05.05.2026).

26. bcrypt – A library to help you hash passwords. npm registry. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 08.05.2026).

ДОДАТОК А

Лістинг програмного коду інформаційної системи «FishFarm».

Мови програмування та фреймворки: TypeScript, Node.js, NestJS (бекенд); TypeScript, React 19, Redux Toolkit (фронтенд).

Сервіс операцій годування (FeedingService)

```
// backend/src/feeding/feeding.service.ts (фрагмент)
@Injectable()
export class FeedingService {
  async createFeedingLog(dto: CreateFeedingLogDto, user: User): Promise<FeedingLog> {
    const batch = await this.fishBatchRepo.findOneOrFail({
      where: { id: dto.batchId }, relations: ['pond', 'fishType'] });
    // Етап 1: Перевірка статусу ставка
    if (batch.pond.status !== PondStatus.ACTIVE)
      throw new BadRequestException('Ставок не активний');
    // Етап 2: Температурна перевірка
    const lastSensor = await this.sensorRepo.findOne({
      where: { pond: { id: batch.pond.id } }, order: { loggedAt: 'DESC' } });
    if (!lastSensor || lastSensor.temperatureC < 8.0)
      throw new BadRequestException('Температура занадто низька (< 8°C)');
    // Етап 3: Розрахунок норми
    const proteinCoeff = 40 / dto.feedProteinPct;
    const dailyKg = batch.currentBiomassKg * (batch.fishType.feedRatePct / 100)
                  * proteinCoeff;
    const actionKg = dailyKg / dto.feedingsPerDay;
    const minKg = actionKg * 0.7;
    const maxKg = actionKg * 1.3;
    // Етап 4: Валідація поданої кількості
```

```
    if (dto.fedKg < minKg || dto.fedKg > maxKg)
      throw new BadRequestException(
        `Кількість корму поза допустимим діапазоном [${minKg.toFixed(2)}, ${maxKg.toFixed(2)}] кг`);
    // Етап 5: Списання зі складу (транзакція)
    return this.dataSource.transaction(async (manager) => {
      const item = await manager.findOneOrFail(InventoryItem, { where: { id: dto.itemId } });
      if (item.balance < dto.fedKg)
        throw new ConflictException('Недостатній залишок на складі');
      item.balance -= dto.fedKg;
      await manager.save(item);
      const txn = manager.create(InventoryTransaction, {
        item, type: TransactionType.CONSUME, quantity: -dto.fedKg });
      await manager.save(txn);
      const log = manager.create(FeedingLog, { ...dto, batch, transaction: txn });
      return manager.save(log);
    });
  }
}
```

WebSocket-шлюз для IoT-телеметрії (SensorsGateway)

```
// backend/src/sensors/sensors.gateway.ts
@WebSocketGateway({ cors: { origin: '*' } })
export class SensorsGateway {
  @WebSocketServer() server: Server;
  broadcastSensorData(pondId: number, data: SensorDataDto): void {
    this.server.to(`pond_${pondId}`).emit('sensorUpdate', data);
  }
```

```
  @SubscribeMessage('joinPond')
  handleJoinPond(client: Socket, pondId: number): void {
    client.join(`pond_${pondId}`);
     }
}
```

Сервіс емуляції IoT-датчиків (IotSimulatorService)

```
// backend/src/iot-simulator/iot-simulator.service.ts (фрагмент)
@Injectable()
export class IotSimulatorService {
  private intervalRef: NodeJS.Timeout;
  startSimulation(intervalMs = 5000): void {
    this.intervalRef = setInterval(async () => {
      const ponds = await this.pondRepo.find({
        where: { status: In([PondStatus.ACTIVE, PondStatus.WINTERING]) }
      });
      for (const pond of ponds) {
        const payload: CreateSensorLogDto = {
          pondId: pond.id,
          temperatureC: this.randomInRange(10, 28),
          oxygenMgl:    this.randomInRange(4, 12),
          phLevel:      this.randomInRange(6.5, 8.5),
          ammoniaMgl:   this.randomInRange(0, 1.5),
        };
        await this.sensorsService.createLog(payload);
      }
    }, intervalMs);
  }
}
```

Хук підписки на WebSocket (useSocketSensors)

```
// frontend/src/hooks/useSocketSensors.ts
export function useSocketSensors(pondId: number) {
  const dispatch = useAppDispatch();
  useEffect(() => {
    const socket = io(SOCKET_URL, { transports: ['websocket'] });
    socket.emit('joinPond', pondId);
    socket.on('sensorUpdate', (data: SensorReading) => {
      dispatch(setSensorReading({ pondId, data }));
    });
    return () => { socket.disconnect(); };
  }, [pondId, dispatch]);
}
```

Планувальник оновлення біомаси (FishCronService)

```
// backend/src/fish/fish.cron.ts
@Injectable()
export class FishCronService {
  @Cron(CronExpression.EVERY_DAY_AT_MIDNIGHT)
  async updateBiomass(): Promise<void> {
    const activeBatches = await this.batchRepo.find({
      where: { status: BatchStatus.ACTIVE }, relations: ['fishType'] });
    for (const batch of activeBatches) {
      const sgr = batch.fishType.defaultSgr ?? 1.2; // % на день
      const growthFactor = 1 + sgr / 100;
      batch.currentBiomassKg *= growthFactor;
      await this.batchRepo.save(batch);  }}}
```