КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

УДК 004.415.2
DOI https://doi.org/10.32851/tnv-tech.2021.6.1

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ МІНІМІЗАЦІЇ ХАРЧОВИХ ВІДХОДІВ У МЕЖАХ КОНЦЕПЦІЇ «ZERO WASTE»

Герасименко О.Ю. *– кандидат технічних наук,*
доцент кафедри мережевих та інтернет-технологій
Київського національного університету ім. Тараса Шевченка
ORCID ID: 0000-0001-6804-2125
Фекете Д.М. *– бакалавр із телекомунікацій та радіотехніки,*
розробник програмного забезпечення в Zoolatech
ORCID ID: 0000-0003-3394-1053

Поняття «zero waste» означає концепцію, яка передбачає повну мінімізацію відходів для збереження навколишнього середовища та ресурсів планети. В Україні проблема зі сміттям стоїть дуже гостро. Зменшивши кількість відходів у сфері торгівлі харчовими продуктами та громадського харчування, можна зменшити загальну кількість сміття. У статті представлено розробку архітектури для інформаційної системи, яка покликана мінімізувати харчові відходи в межах концепції «zero waste». Під час дослідження було проаналізовано три архітектури, а саме: монолітну архітектуру, мікросервісну архітектуру та подійно-керовану архітектуру; визначено їх переваги і недоліки. Базовою було обрано подійно-керовану сервіс-орієнтовану архітектуру з використанням CQRS-шаблону, який передбачає відділення операцій читання від операцій запису. Ця система спроєктована як розподілена система з мікросервісною архітектурою та керується подіями. Публікація подій відбувається за допомогою сучасної системи обміну повідомленнями з відкритим кодом NATS. Внутрішні сервіси спілкуються за допомогою NATS, а REST-запити використовуються для комунікації між клієнтською та серверною частинами. Для збереження даних використовуються джерела подій. Впровадження CQRS допомагає максимізувати ефективність, безпечність та масштабованість додатка, дозволяє побудувати різні моделі зчитування й запису, які можуть бути оптимізовані під вимоги системи. Застосунок проєктувався таким чином, щоб у майбутньому його можна було легко масштабувати та розвивати. На початковому етапі спроєктовано високорівневу архітектуру системи, яка включає три основні мікросервіси; представлено чотири компоненти системи та один агрегат; визначено базовий функціонал системи та описано порядок взаємодії елементів системи для реалізації основного функціоналу. Подальший розвиток роботи можливий у напрямі вдосконалення функціоналу наявних сервісів

та розробки нового функціоналу, доведення інформаційної системи до рівня мінімально життєздатного продукту (MVP, Minimum Viable Product).

Ключові слова: *архітектура інформаційної системи, подійно-керована архітектура, шаблон «Публікація – Підписка», «food waste», шаблон CQRS.*

Herasymenko O.Yu., Fekete D.M. Architecture of information system for food waste minimization according the "zero waste" concept

"Zero waste" is a term that implies waste reduction in order to decrease environmental polution and save planet resources. The problem with waste is very acute in Ukraine. By reducing the amount of waste from food trade and catering, we can reduce the total amount of rubbish. This study presents an information system that aims to help decrease food waste, and is devoted to the design of the architecture of this system. Three architectures were analyzed in this study, which are monolithic architecture, microservice architecture and event-driven architecture, and their advantages and disadvantages were identified. Design decision of the project is based on microservice architecture with event-driven communication which uses CQRS-pattern, which provides for the separation of read operations from write operations. The system is build as distributed and driven by events. Events are published with the help of a modern open source messaging system called NATS. Backend services communicate using publisher-subscriber pattern which NATS service provides and REST-methods are used for client-service message exchange. Implementing the CQRS pattern helps maximize application efficiency, security, and scalability; and also allows developers to separate read and write models that can be optimized for system requirements. Application has the ability to scale to cope with increased load and designed to be easy to evolve. A high-level architecture of the system is proposed in this research. At the initial stage it includes three main microservices. Four system components and one aggregate are also presented in this paper. The basic functionality of the system is outlined in this article and its implementation is represented in the form of a sequence of system elements interaction. Further development of this work is possible in the direction of improving the functionality of existing services and developing new functionality. It is planned to bring the information system to the level of the minimum viable product (MVP).

Key words: *system architecture, event-driven architecture, pattern "Publisher – Subscriber", "food waste", CQRS pattern.*

Вступ. У середньому близько 1,3 млрд тонн їжі втрачається чи викидається щороку у світі – це одна третина від загальної кількості їжі, що призначена для людського споживання, згідно з даними Food and Agriculture Organization of the United Nations (далі – FAO) [1]. Загальна вартість втраченої їжі сягає близько 2,6 трлн доларів у рік [2], її обсяг є достатнім для того, щоб нагодувати всі 815 млн голодуючих людей у світі. Придатна до споживання їжа втрачається майже в кожній точці харчового ланцюга: починаючи від ферм та виробництв і закінчуючи магазинами, ресторанами та людськими кухнями [3].

Однією з найбільших перешкод на шляху до досягнення мети «zero waste» є колосальна втрата їжі на планеті («food waste» і «food lost»). Ресторани та харчові сервіси відіграють одну з ключових ролей у втраті їжі й продукуванні відходів (food lost and waste – FWL), починаючи від розробки меню та закінчуючи управлінням залишками з тарілок відвідувачів [4]. Враховуючи той факт, що близько 800 млн людей у світі, зокрема, у країнах, що розвиваються, страждають від хронічної нестачі їжі [5], дуже прикро усвідомлювати, що одна третина їжі викидається.

У сучасних реаліях цифрові технології мають невичерпний потенціал, який можна використати для вдосконалення чи повної трансформації вартості та структури харчових ланцюгів (FVCs), а також значним чином посприяти розробці більш продуктивних і стійких систем, що забезпечують обробку або зберігання їжі [6]. Є сотні прикладів успішної диджиталізації невеликих фермерських угідь, які змогли зменшити харчові відходи мало не до 2% від загального обсягу, покращивши умови зберігання продуктів та забезпечивши пряму доставку на ринок.

Використання технологій на фермах чи виробництвах – це лише невелика частина можливостей, які сучасний світ може запропонувати для повної трансформації культури відходів. Одну з ключових ролей у збільшенні обсягів втрати продуктів відіграють стандарти ринку, за якими смачні та свіжі продукти відправляються в смітник через зовнішні стандарти. Науковці переконані, що уникнення стандартів краси під час вибору продуктів і зважання на те, що смак усередині жодним чином не залежить від зовнішніх факторів, можуть побороти світовий голод [7]. На щастя, вже з'явилися застосунки, які допомагають недосконалим продуктам потрапити на стіл споживача або на годування тварин замість загнивання на звалищі.

Постановка проблеми. Поточна ситуація на ринку екозастосунків України залишає бажати кращого, тоді як можна часто бачити багато викинутих продуктів у смітниках поблизу великих магазинів. І це на фоні того, що кожне велике місто потерпає від проблем, пов'язаних із вивезенням та захороненням сміття. Відповідно, розробка такого застосунку є надзвичайно актуальною. Ідея застосунку полягає в тому, щоб надати можливість для збуту за зниженою ціною харчових продуктів, які не продалися через недосконалий зовнішній вигляд або в яких закінчується термін споживання, тощо.

Створення будь-якої інформаційної системи є нетривіальним завданням і починається з етапу проєктування. Питання вибору та проєктування архітектури визначається на початковому етапі процесу розробки, що допомагає програмістам отримати детальний огляд системи. Правильно підібраний дизайн є рушійною силою у полегшенні комунікації між зацікавленими сторонами, суттєвому зниженні вартості обслуговування, спрощенні обслуговування системи; робить її більш гнучкою та легшою у масштабуванні. Визначальними критеріями під час проєктування такої інформаційної системи є:

– кількість потенційних користувачів;
– легкість масштабування системи;
– легкість додавання у систему нового функціоналу;
– бюджет, доступний для програмування та підтримки системи;
– можливості монетизації застосунку.

Таким чином, основним завданням дослідження є вибір та розробка архітектури інформаційної системи для мінімізації харчових відходів у межах концепції «zero waste».

Метою дослідження є аналіз наявних архітектур програмного забезпечення; вибір та розробка архітектури високонавантаженої інформаційної системи для мінімізації харчових відходів у межах концепції «zero waste», яка за потреби може бути легко масштабована.

Аналіз останніх досліджень і публікацій. Загалом у світі тема програмного забезпечення, що допомагає зберегти природу, надзвичайно поширена. Останніми роками інвестиції в застосунки подібного роду збільшилися в десятки, а то й сотні разів. Усвідомлення катастрофічної екологічної ситуації, що склалася на землі, є рушійною силою, яка спонукає корпорації, великих та малих інвесторів вкладати кошти у розвиток технологій, що потенційно могли б допомогти уникнути глобальних катаклізмів. У статті для прикладу варто розглянути два найбільш відомі проєкти, які наразі перебувають на етапі стартапів, що дуже близькі до теми, котра розглядається: Olio [8] та Full Harvest [9].

Olio – мобільний додаток, що забезпечує можливість обміну їжею з метою зменшення харчових відходів. Користувачі, які мають залишки їжі, можуть легко

та безкоштовно поділитися нею з тими, хто цього потребує. Їжа має бути придатною для споживання, вона може бути як сирою, так і приготованою, закритою чи відкритою [10]. Перший реліз проєкту був представлений у 2015 р. засновницями Тесою Кларк та Сашею Селестіал-Ван [11]. До початку жовтня 2017 р. компанія зібрала близько 2,2 млн доларів інвестицій [12]. До вересня 2020 р. додатком уже користувалося близько 2,3 млн користувачів.

Історія Olio ще раз підтверджує думку, що ця тема надзвичайно актуальна на сьогоднішній день.

Full Harvest – ще один молодий американський стартап, який бореться з проблемою «food waste». Це застосунок, який допомагає продавати чи купувати продукти харчування, що не потрапили на ринок через виключно зовнішні фактори. Надзвичайно популярний стартап, який вже привернув увагу таких гігантів, як Forbes, Shape, Wall Street Journal. Ідеологія його дуже проста: у користувачів є можливість як придбати товари, так і позбутися їх у сфері B2B [13].

За результатами аналізу розглянутих програмних продуктів можна зробити висновок, що з часом вони стають дедалі популярнішими, тож під час розробки архітектури інформаційної системи цей нюанс повинен бути врахований. Коротко розглянемо поняття архітектури програмного забезпечення та найбільш поширені сучасні архітектури.

Архітектура програмного забезпечення призначена для відображення того, як має бути організована інформаційна система (далі – ІС), з яких компонентів вона буде складатися та як вони будуть взаємодіяти між собою, реалізуючи функціонал ІС. Дизайн архітектури описує високорівневу структуру системи та її компонентів. Мета архітектури полягає в зображенні відношень між компонентами та їх базової взаємодії [14].

У цьому контексті варто розглянути різні види архітектури, що вважаються базовими, їхні переваги та недоліки.

Монолітна архітектура. Монолітний застосунок працює як єдина програма та запускається як один виконуваний файл. З ростом кодової бази складність розробки та управління проєктом зростає. Проте такий підхід дає деякі переваги у роботі. Можна спілкуватися з різними частинами проєкту за допомогою прямих викликів методів, спільного використання пам'яті чи передачі повідомлень. Це дозволяє уникнути затримок у передачі повідомлень.

Основні недоліки монолітної архітектури [15]:

– великі моноліти складно підтримувати та розвивати;

– будь-яка зміна в проєкті вимагає повного перезапуску, що може стати причиною тривалого простою. Цей недолік також може бути причиною значного сповільнення розробки та тестування;

– моноліти мають обмеження в масштабуванні. Єдиний вихід, що дасть змогу легко обробляти нові запити, – це створення додаткового екземпляра моноліту, щоб розділити навантаження. Але трафік може тільки погіршити ситуацію з обробкою запитів у деяких частинах проєкту, тому в цьому випадку створення нового екземпляра не буде ефективним;

– така архітектура повністю обмежує вибір технологій. Модулі мають бути написані однією мовою програмування, потрібно використовувати одні й ті ж фреймворки.

Оскільки завдання дизайну системи, яка розглядається у дослідженні, вимагає можливості швидкої розробки з мінімальними затратами, а також можливості бути високоефективною навіть за величезних навантажень, використання різних

технологій та можливості розроблятися паралельно декількома командами у майбутньому, монолітна архітектура для розробки не підходить.

Мікросервісна архітектура. Мікросервісна архітектура є повною протилежністю монолітної. Моноліт зберігає весь функціонал в одному виконуваному об'єкті, а мікросервіси розподіляють функціонал на маленькі частини. Ця архітектура допомагає вирішити деякі проблеми, що притаманні монолітній:

– завдання кожного мікросервісу полягає в утриманні маленької частини функціоналу, що робить кодову базу простішою і набагато легшою у підтримці та тестуванні;

– мікросервіси легко переводити на нову версію. В разі оновлення функціоналу стара версія може працювати паралельно з новою доти, доки всі інші сервіси не почнуть використовувати нову версію;

– масштабування мікросервісної архітектури не вимагає дублювання всіх сервісів. Розробники можуть збільшувати чи зменшувати кількість екземплярів кожного сервісу залежно від потреби;

– ця архітектура обмежує вибір технологій тільки в межах сервісу. Розробники різних сервісів можуть обирати технології за власним бажанням та потребами сервісу.

Мікросервіс – це додаток, що може масштабуватися, запускатися та тестуватися повністю незалежно від усієї системи. Це вирішує питання єдиної відповідальності, коли сервіс відповідає лише за одну проблему, яку легко зрозуміти [16]. Інша важлива перевага такого підходу – це надійність.

Подійно-керована архітектура (Event driven architecture, EDA). Під час розробки мікросервісів та розподілених систем виникає потреба в комунікації та координації між сервісами і додатками. Це саме те, що допомагає вирішити така архітектура.

Подійно-керована архітектура – це архітектура програмного забезпечення, де застосунок публікує, визначає та відповідає на події. Подія репрезентує зміну в стані системи. Архітектура включає видавців (сутностей, які публікують події), процесори, відповідачів та посилання комунікацій [17].

Цей підхід застосовується для публікації подій в разі зміни стану системи. Коли стан визначеного компонента застосунку оновлюється, відбувається публікація нової події, що свідчить про цю зміну. Така подія може бути опублікована в «темі» (іменованому логічному каналі), підписники якої будуть оповіщені про цю зміну. Отримувач може застосувати деякі функції після одержання події і згодом опублікувати нове повідомлення для своїх підписників. Схема шаблону «Публікація – Підписка» зображена на рис. 1.

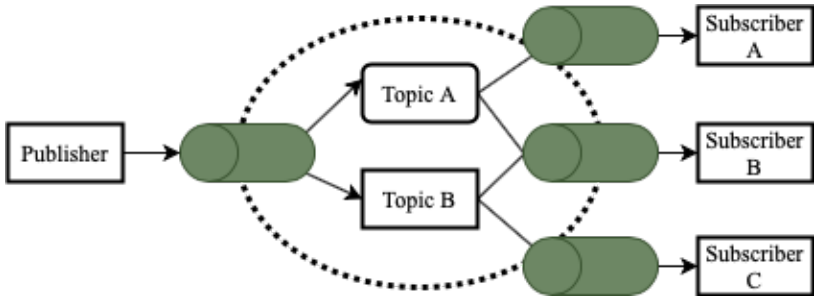

Рис. 1. Схема роботи шаблону «Публікація – Підписка»

Така система може бути спроєктована для вирішення питань узгодженості у розподілених і мікросервісних застосунках, де використання розподілених транзакцій є дуже складним чи не вистачає підтримки в базах даних [18]. За такої архітектури додатки можуть використовувати події в якості сигналів для інших систем про те, що вони виконали визначені дії. В разі, якщо виконання було перервано чи не здійснено, додатки можуть створити подію, яка буде свідчити про помилку в системі.

Однією з найбільших переваг керованих подіями систем є те, що система буде слабозв'язана. Це означає, що сервісам необхідно дуже мало інформації один про одного. Така система надзвичайно легко та швидко масштабується, що відповідає одній з головних вимог інформаційної системи, яка розглядається у дослідженні.

Виклад основного матеріалу. Базовою для інформаційної системи було обрано подійно-керовану сервіс-орієнтовану архітектуру [19] з використанням CQRS-шаблону. CQRS (Command-Query Responsibility Segregation) – шаблон розділення відповідальності на команди та запити; розділяє операції зчитування та оновлення сховища даних. Внутрішні сервіси спілкуються за допомогою NATS, а REST-запити використовуються для комунікації між клієнтською та серверною частинами. Для збереження даних використовуються джерела подій. На початковому етапі було спроєктовано три мікросервіси. На рис. 2 представлена високорівнева архітектура мікросервісної частини.

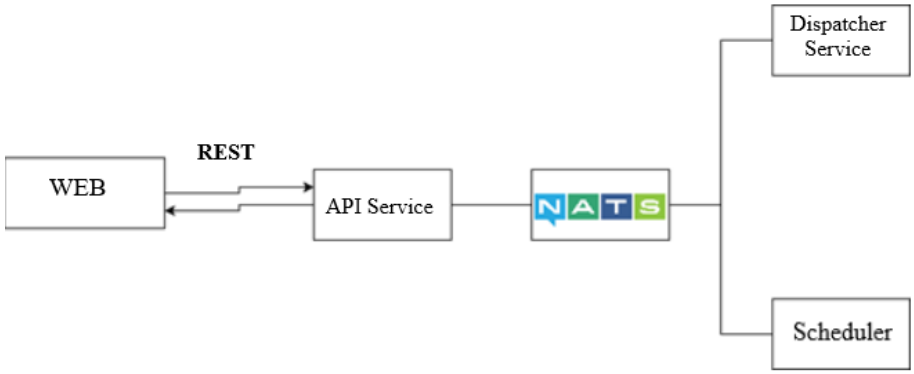

Рис. 2. Високорівнева архітектура інформаційної системи

Наслідуючи CQRS-шаблон, відділено моделі зчитування та запису. Таким чином, лише API-сервіс буде отримувати REST-запити з клієнтської частини та створювати команди для Dispatcher-сервісу, який своєю чергою записуватиме події та агрегати у сховище подій.

Спілкування між API, Dispatcher та Scheduler-сервісами відбувається на основі обміну повідомленнями, використовуючи відомий шаблон «Публікація – Підписка». Сервіси працюють незалежно один від одного, зникає необхідність в очікуванні відповіді тощо. Основна ідея полягає в тому, що API-сервіс публікує подію (повідомлення), яка свідчить про те, що відбулася зміна стану системи, і будь-який сервіс, якому цікава ця подія, може її прочитати.

API-сервіс відповідає за отримання та обробку запитів від клієнтської частини застосунку, використовуючи REST-методи. Після обробки запитів сервіс створює повідомлення, публікація яких відбувається за допомогою сучасної системи обміну повідомленнями з відкритим кодом NATS.

Dispatcher-сервіс виконує роль сховища подій та часткового проєктора. Для подальшого розвитку можна буде розділити цей сервіс на два компоненти: читач та видавець (reader and writer), як у класичному CQRS-шаблоні. Цей сервіс отримує команди та в разі успішної перевірки генерує подію і записує її у сховище. Також цей сервіс використовується для генерації агрегатів та збереження їх у сховище.

Scheduler – це сервіс, який виконує роль планувальника завдання за часом (cronjob). Він підписується на події та в разі необхідності звертається у сховище агрегатів, щоб оновити їх стан.

Система складається з 4 основних компонентів та 1 агрегата. Компонент «Продукт» – це елемент, що доступний для збуту, який використовується під час створення нової позиції. Йому належать поля:

– Id – унікальний номер продукту;
– UserId – унікальний номер користувача, який вніс продукт у сховище даних;
– Unit – це об’єкт, що зберігає одиницю виміру продукту (наприклад, продукт «Яблука» вимірюється в кілограмах);
– Name – назва продукту.

Компонент «Позиція» – це доступний елемент для резервування та подальшого підтвердження замовлення. Він може бути створений ресторанами, магазинами та іншими користувачами, які шукають можливості для збуту продуктів. Містить поля:

– Id – унікальний номер позиції;
– UserId – унікальний номер користувача, який створив позицію;
– ProductId – унікальний номер продукту, що публікується;
– Type – тип позиції (перелік);
– Quantity – кількість продукту;
– CreatedAt – час створення позиції (використовується для того, щоб відстежувати закінчення терміну придатності та закриття позиції за необхідності);
– ExpiresAt – час завершення терміну дії пропозиції.

Типи позиції використовуються як ідентифікатори для створення подій (рис. 3), оскільки мета полягає в детальному відстеженні історії за всіма позиціями, відстеженні утримань тощо.

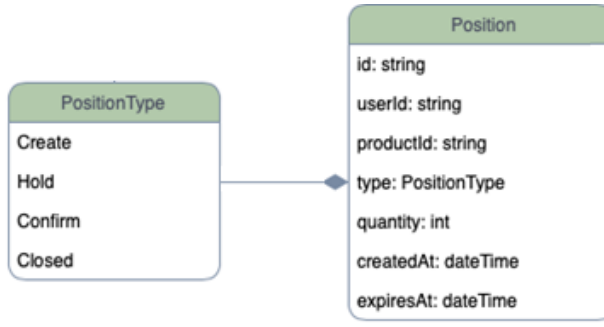

Рис. 3. Структура компонента «Позиція»

Агрегат «Позиція» – це внутрішній тип, який використовується для відстеження поточного стану компонента «Позиція» й оновлюється з кожною новою подією над позицією, що належить цьому агрегату. Містить поля:

– Id – унікальний номер позиції;

– UserId – унікальний номер користувача, який створив позицію;
– ProductId – унікальний номер продукту, доступний у позиції;
– Available – кількість доступного продукту;
– OnHold – кількість продукту, що перебуває на утриманні.

Компоненту «Користувач» належать поля:
– Id – унікальний номер користувача;
– FirstName – ім'я користувача;
– LastName – прізвище користувача.

Компонент «Утримання» використовується для відстеження позицій, які перебувають на утриманні, та користувачів, які їх створили. Наприклад, Користувач1 створює утримання на Позиція1 на 10 одиниць продукту – це свідчить про те, що користувач має намір забрати якусь частину продуктів із позиції. Позиція може містити невизначену кількість утримань від невизначеної кількості користувачів (допоки не закінчиться доступна кількість продуктів). Поля цього компонента:
– Id – унікальний номер утримання;
– UserId – номер користувача, що створив утримання;
– PositionId – номер позиції, що утримується;
– Quantity – кількість елементів позиції, що утримується;
– CreatedAt – дата створення утримання.

Опишемо основні принципи функціонування системи. Користувач взаємодіє з додатком за допомогою клієнтської частини, яка надсилає HTTP-запити до серверної частини. Кожна взаємодія користувача і додатка – це окремий виклик доступних REST-методів, які отримує API-сервіс, призначений для обробки запитів та створення команд для всієї системи. Такий підхід є високоефективним та легким у підтримці.

Базовий функціонал, доступний для взаємодії користувача та додатка, включає:
– Додати продукт;
– Створити позицію;
– Отримати всі актуальні позиції для окремого користувача;
– Отримати позиції, доступні до утримання;
– Утримати визначену кількість продукту з позиції;
– Отримати всі утримання для окремого користувача;
– Скасувати створення позиції;
– Підтвердити утримання;
– Скасувати утримання.

Базовий функціонал, що відбувається без взаємодії з користувачем, включає:
– Постійну перевірку поточного стану позицій і скасування позицій у разі закінчення терміну придатності;
– Оновлення поточного стану системи;
– Запис усіх змін станів позицій у сховище подій.

Розглянемо детальніше внутрішню архітектуру найважливіших та найскладніших методів.

Створення позиції відбувається за допомогою методу POST із такими параметрами:
– ProductId – продукт, що доступний у позиції;
– Quantity – кількість доступного продукту.

API-сервіс отримує параметричний HTTP-метод POST, валідує його та записує повідомлення для створення нової позиції. Dispatcher-сервіс «слухає» всі оновлення щодо позицій та під час отримання нового повідомлення формує подію з типом «Create» і зберігає її в сховище подій.

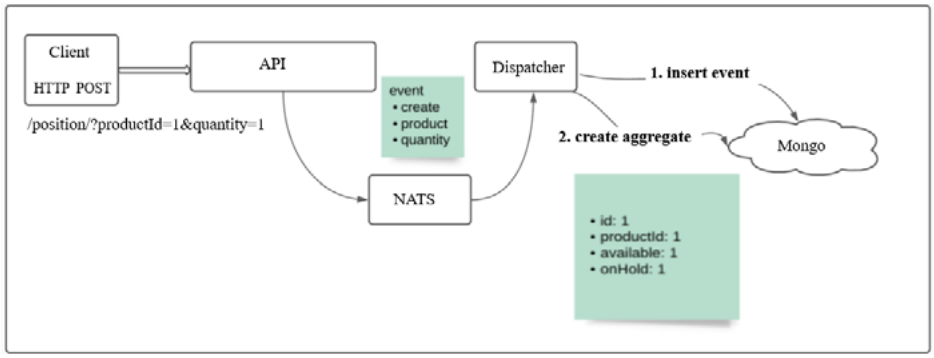

Рис. 4. Схематичне представлення взаємодії компонентів під час створення позиції

Також на етапі створення події з типом «Create» ініціалізується порожній агрегат на цю позицію та також зберігається у сховище. За шаблоном CQRS агрегати та події зберігаються в окремих сховищах. На рис. 4 зображена візуалізація вищезазначеного процесу.

Отримання всіх позицій відбувається за допомогою HTTP-методу GET, залежно від вказаного шляху, можна обрати позиції, створені поточним користувачем, або ж всі доступні позиції до утримання. API-сервіс виконує функцію читача зі сховища агрегатів та повертає необхідні дані у форматі JSON. Цей підхід забезпечує миттєву обробку запитів, що є надзвичайно важливим під час розробки високонавантажених систем. На рис. 5 зображена візуалізація вищезазначеного процесу з усіма переліченими елементами.

Одним із найскладніших елементів системи, що керується подіями, є забезпечення узгодженості даних. Розподілені менеджери блокувань (lock-managers) використовуються для організації та серіалізації доступів до ресурсів.

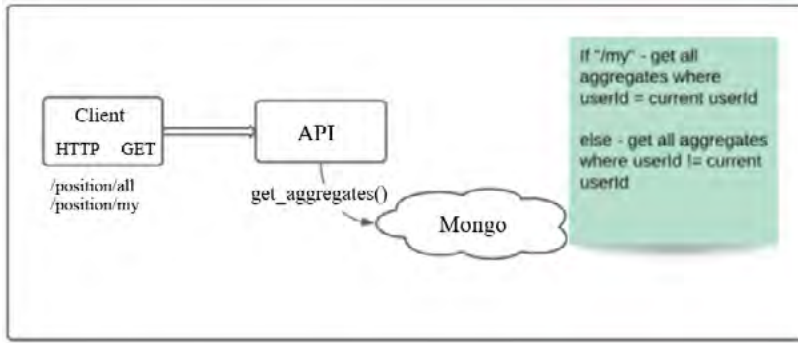

Рис. 5. Схематичне представлення взаємодії компонентів під час отримання позицій

У випадку системи, яка розглядається, ситуація неузгодженості даних може виникнути, коли два користувачі одночасно створюють утримання на одну й ту ж позицію. В такому разі в момент отримання кожного запиту необхідно «блокувати» оновлення позиції, поки попередні зміни не будуть збережені. На рис. 6 можна побачити схему взаємодії всіх елементів, задіяних у процесі створення утримання. Процес повернення всіх створених утримань відбувається аналогічно

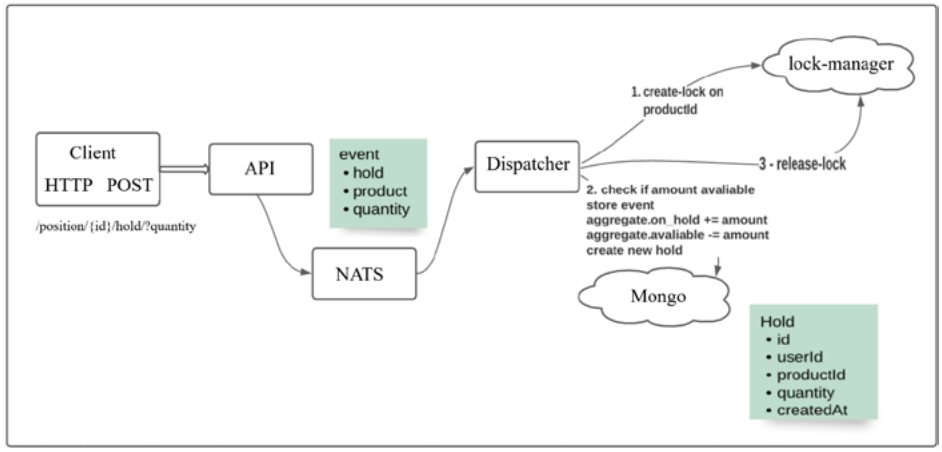

Рис. 6. Схема взаємодії компонентів під час створення утримання

процесу отримання всіх позицій – за допомогою GET-методу. Є два різні методи: отримання всіх утримань користувача та отримання всіх утримань з обраної позиції (доступно для користувачів, що мають відкриті позиції). На рис. 7 зображена схема взаємодії компонентів під час обробки отримання утримань.

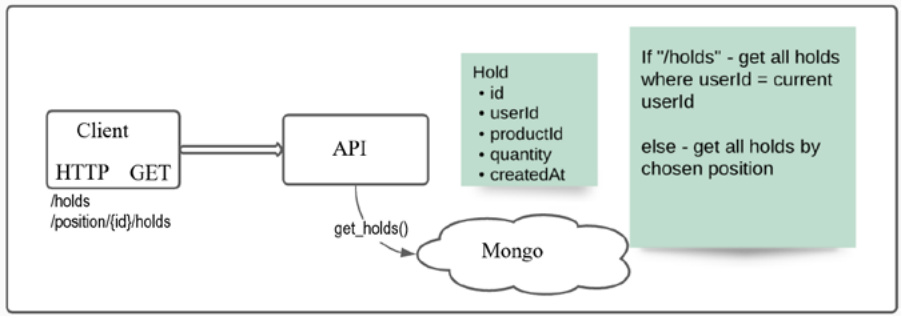

Рис. 7. Схема взаємодії компонентів під час отримання утримань

Таким чином, було детально розглянуто внутрішню архітектуру обробки запитів, створення команд та записів подій, якими оперує ця система.

Висновки та пропозиції. У дослідженні було проведено аналіз ринку застосунків для мінімізації харчових відходів суспільства та аргументовано необхідність створення цифрового продукту, який може забезпечити зменшення втрат їжі, придатної до споживання.

У роботі виконано детальний аналіз наявних архітектур програмних систем і технологій, які можуть використовуватися для їх реалізації. За результатами дослідження запропоновано подійно-керовану мікросервісну архітектуру для відповідної інформаційної системи. Також розроблено схеми для кожного компонента системи, розглянуто атрибути компонентів, їх типи та властивості, визначено схеми взаємодії компонентів системи. У роботі запропоновано використати шаблон CQRS для організації внутрішньої взаємодії та шаблон «Публікація – Підписка» як технологію обміну повідомленнями у застосунку. Інформаційна система спроєктована так, що може легко масштабуватися.

Соціальна цінність цієї роботи полягає у створенні архітектури цифрового продукту, який може забезпечити зменшення втрат їжі, ще придатної до споживання.

Подальший розвиток роботи можливий у напрямі вдосконалення функціоналу наявних сервісів та розробки нового функціоналу, розробки сервісів для блокування, реалізації DLQ для ефективного репроцесингу помилок і перетворення продукту на MVP.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Technical Platform on the Measurement and Reduction of Food Loss and Waste. *Food and Agriculture Organisation of United Nations* : вебсайт. URL: http://www.fao.org/platform-food-loss-waste/en/ (дата звернення: 02.06.2021 р.).

2. Food Wastage Footprint. *Food and Agriculture Organisation of United Nations* : вебсайт. URL: http://www.fao.org/nr/sustainability/food-loss-and-waste/en/ (дата звернення: 02.06.2021 р.).

3. Buzby J.C., Wells H.F., Hyman J. The Estimated Amount, Value, and Calories of Postharvest Food Losses at the Retail and Consumer Levels in the United States, EIB-121, U.S. Department of Agriculture, Economic Research Service. February 2014. 39 p.

4. Food Loss and Waste in Fish Value Chains. *Food and Agriculture Organisation of United Nations* : вебсайт. URL: http://www.fao.org/flw-in-fish-value-chains/value-chain/retail/restaurants-and-catering/en/ (дата звернення: 02.06.2021 р.).

5. World hunger: facts & how to help. World Vision : вебсайт. URL: https://www.worldvision.ca/stories/food/world-hunger-facts-how-to-help (дата звернення: 05.06.2021 р.).

6. Đuric' I. Digital technology and agricultural markets – Background paper for The State of Agricultural Commodity Markets (SOCO). 2020. Rome, FAO. DOI: /10.4060/cb0701en (дата звернення: 04.06.2021 р.).

7. Beauty (and taste!) are on the inside. FAO. *Food and Agriculture Organisation of United Nations* : вебсайт. URL: http://www.fao.org/fao-stories/article/en/c/1100391/ (дата звернення: 04.06.2021 р.).

8. Застосунок Olio. *Вікіпедія: вільна енциклопедія.* URL: https://en.wikipedia.org/wiki/Olio_(app) (дата звернення: 02.06.2021 р.).

9. Застосунок FullHarvest : вебсайт. URL: https://www.fullharvest.com (дата звернення: 02.06.2021 р.).

10. Harris S.A. Food Waste App OLIO Has Become A Lifeline For Those Who Can't Afford To Feed Themselves : вебсайт. URL: https://www.huffingtonpost.co.uk/entry/food-waste-app-olio-hidden-hunger_uk_595f4212e4b0d5b458e97c36 (дата звернення: 02.06.2021 р.).

11. Dymoke A. Food for London: Olio, the app matching surplus food to hungry Londoners : вебсайт. URL: https://www.standard.co.uk/news/foodforlondon/food-for-london-the-app-matching-surplus-food-to-hungry-londoners-a3387641.html (дата звернення: 02.06.2021 р.).

12. Smith M.J. Don't Toss That Lettuce – Share It. *Stanford graduate school of business* : вебсайт. URL: https://www.gsb.stanford.edu/insights/dont-toss-lettuce-share-it (дата звернення: 02.06.2021 р.).

13. Manning L. How Full Harvest is Using Technology to Connect the Dots in the B2B Food Waste Space. *AFN* : вебсайт. URL: https://agfundernews.com/how-full-harvest-is-using-technology-to-connect-the-dots-in-the-b2b-food-waste-space.html (дата звернення: 02.06.2021 р.).

14. Sommerville I. Software engineering. Tenth edition, global edition. Boston, Mass. Amsterdam Cape Town : Pearson Education Limited, 2016. 810 p.

15. Dragoni N. et al. Microservices: yesterday, today, and tomorrow. URL: http://arxiv.org/abs/1606.04036 (дата звернення: 06.06.2021 р.).

16. Thönes J. Microservices. *IEEE Software*. 2015. Vol. 32. No. 1. pp. 116. DOI: 10.1109/MS.2015.11.

17. Encyclopedia of database systems / ed. L. Liu, Özsu M. Tamer. New York : Springer, 2009. DOI: 10.1007/978-0-387-39940-9.

18. Challenges and solutions for distributed data management. *Microsoft Docs*:вебсайт.URL:https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/distributed-data-management (дата звернення: 06.06.2021 р.).

19. Петренко О.О. Порівняння типів архітектури систем сервісів. *System Research & Information Technologies.* 2015. № 4. С. 48–62.

REFERENCES:

1. Food and Agriculture Organisation of United Nations (2021) *Technical Platform on the Measurement and Reduction of Food Loss and Waste*. URL: http://www.fao.org/platform-food-loss-waste/en/.

2. Food and Agriculture Organisation of United Nations (2014) *Food Wastage Footprint*. URL: http://www.fao.org/nr/sustainability/food-loss-and-waste/en/.

3. Buzby, J.C., Wells, H.F., Hyman, J. (2014) The Estimated Amount, Value, and Calories of Postharvest Food Losses at the Retail and Consumer Levels in the United States, EIB-121, U.S. Department of Agriculture, Economic Research Service. February 2014. 39 p.

4. Food and Agriculture Organisation of United Nations (2021) *Food Loss and Waste in Fish Value Chains.* URL: http://www.fao.org/flw-in-fish-value-chains/value-chain/retail/restaurants-and-catering/en/.

5. World Vision (2020) *World hunger: facts & how to help.* URL: https://www.worldvision.ca/stories/food/world-hunger-facts-how-to-help.

6. Duric', I. (2020) Digital technology and agricultural markets – Background paper for The State of Agricultural Commodity Markets (SOCO) 2020. Rome, FAO. DOI: 10.4060/cb0701en.

7. Food and Agriculture Organisation of United Nations (2018) *Beauty (and taste!) are on the inside. FAO.* URL: http://www.fao.org/fao-stories/article/en/c/1100391/.

8. Application Olio (2021) In *Wikipedia*. URL: https://en.wikipedia.org/wiki/Olio_(app).

9. Application FullHarvest. URL: https://www.fullharvest.com.

10. Harris, S.A. (2017) Food Waste App OLIO Has Become A Lifeline For Those Who Can't Afford To Feed Themselves. URL: https://www.huffingtonpost.co.uk/entry/food-waste-app-olio-hidden-hunger_uk_595f4212e4b0d5b458e97c36.

11. Dymoke, A. (2016) Food for London: Olio, the app matching surplus food to hungry Londoners. URL: https://www.standard.co.uk/news/foodforlondon/food-for-london-the-app-matching-surplus-food-to-hungry-londoners-a3387641.html.

12. Smith, M.J. (2017) Don't Toss That Lettuce – Share It. Stanford graduate school of business. URL: https://www.gsb.stanford.edu/insights/dont-toss-lettuce-share-it.

13. Manning, L. (2019). How Full Harvest is Using Technology to Connect the Dots in the B2B Food Waste Space. URL: https://agfundernews.com/how-full-harvest-is-using-technology-to-connect-the-dots-in-the-b2b-food-waste-space.html.

14. Sommerville, I. (2016) Software engineering. Tenth edition, global edition. Boston, Mass. Amsterdam Cape Town : Pearson Education Limited.

15. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R. (2017) Microservices: yesterday, today, and tomorrow. arXiv.org. http://arxiv.org/abs/1606.04036.

16. Thönes, J. (2015) Microservices. *IEEE Software*. Vol. 32. No. 1, pp. 116. DOI: 10.1109/MS.2015.11.

17. Liu, L., Tamer Özsu. M. (eds.). (2009) Encyclopedia of database systems. Springer. DOI: 10.1007/978-0-387-39940-9.

18. Microsoft (2021) Challenges and solutions for distributed data management. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/distributed-data-management.

19. Petrenko, O.O. (2015) Comparison of service system architecture types. *System Research & Information Technologies*. № 4, pp. 48–62.