Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Екологічний маркетплейс для реалізації непроданих страв у закладах харчування як інструмент сталого розвитку»

Виконав: студент VI курсу групи КН-63М
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Роюк Михайло Анатолійович
(прізвище та ініціали)

Керівник Опришко М.І., Карашецький В.П.
(прізвище та ініціали)

Рецензент Флуд Л.О.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

_____ Борецька І.Б.
"10" грудня 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Роюк Михайло Анатолійович
(прізвище, ім'я, по батькові)

1. Тема роботи: Екологічний маркетплейс для реалізації непроданих страв у закладах харчування як інструмент сталого розвитку
Керівник роботи *Опришко М.І., асистент, Карашецький В.П., к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" квітня 2025 року №С-288
2. Термін подання студентом роботи "10" грудня 2025 року
3. Вихідні дані до роботи: Розробка екологічного маркетплейсу для реалізації непроданих страв у закладах харчування з використанням React-фреймворку. Провести огляд бібліотек React, що забезпечують ефективне управління даними, інтеграцію з API, автентифікацію та побудову зручного інтерфейсу для користувачів.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Розділ 1. Стан проблемної області
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді
6. Дата видачі завдання "1" травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	01.05.25-05.05.25	Виконано
2.	Постановка задачі і її формалізація	05.05.25-10.05.25	Виконано
3.	Виконання вхідного етапу технології	10.05.25-25.07.25	Виконано
4.	Реалізація головних алгоритмів проєкту	25.07.25-12.09.25	Виконано
5.	Виконання етапу відлагодження проєкту	12.09.25-22.10.25	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	22.10.25-05.11.25	Виконано
7.	Оформлення записки до дипломного проекту.	05.11.25-05.12.25	Виконано

Студент _____ Роюк М. А.
(підпис) (прізвище та ініціали)

Керівники роботи _____ Опришко М.І.
(підпис) (прізвище та ініціали)

_____ Карашецький В.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота складається із 101 сторінок пояснювальної записки, містить 41 ілюстрацій та 13 джерел літератури.

Основна увага приділена створенню екологічного маркетплейсу E-Market, який спрямований на зменшення харчових відходів шляхом реалізації непроданих страв у закладах харчування. Платформа пропонує зручний інтерфейс для користувачів, включаючи ресторани та кінцевих споживачів, а також функціонал для додавання, пошуку та купівлі страв із коротким терміном реалізації.

E-Market позиціонується як інструмент сталого розвитку, що поєднує екологічну відповідальність із сучасними технологіями. У ході розробки застосовувались бібліотека React на основі JavaScript, Redux, RTK Query для управління станом, а також PHP з використанням фреймворку Yii2 для побудови серверної частини. Це дозволило досягти високої продуктивності, безпеки та масштабованості системи.

Ключові слова: E-Market, React, Yii2, JavaScript, екологічний маркетплейс, сталий розвиток, харчові відходи.

ABSTRACT

The thesis consists of 101 pages of explanatory text, includes 41 illustrations, and references 13 sources.

The main focus is on the development of E-Market — an ecological marketplace designed to reduce food waste by enabling restaurants to sell unsold meals to consumers. The platform offers a user-friendly interface and a set of features for listing, browsing, and purchasing surplus meals, contributing to sustainable development goals.

E-Market is positioned as a digital solution that merges environmental responsibility with modern web technologies. The development process utilized the React library based on JavaScript, Redux and RTK Query for state management, and the cross-platform Node.js environment for backend implementation, achieving high performance, security, and scalability.

Keywords: E-Market, React, Node.js, JavaScript, ecological marketplace, sustainable development, food waste..

ТЕХНІЧНЕ ЗАВДАННЯ

Відповідно до встановлених вимог технічного завдання, необхідно розробити екологічну платформу E-Market для автоматизованої реалізації непроданих харчових продуктів у закладах торгівлі. Система має забезпечити можливість реєстрації магазинів, створення кошиків із набором продукції, а також планування часу їх продажу та видачі.

Ключові функціональні вимоги до платформи:

- Реєстрація та управління магазинами: надати можливість створення облікового запису магазину, налаштування точок продажу.
- Створення кошиків: реалізувати функціонал для формування кошиків із непроданими товарами, встановлення ціни, опису, терміну реалізації та часу видачі.
- Автоматизація розкладу: передбачити можливість автоматичного виставлення кошиків у визначені дні та години, з гнучкими налаштуваннями повторюваності.
- Статистика продажів: створити розділ аналітики, де відображатиметься кількість проданих кошиків, популярність товарів, динаміка попиту.
- Фінансовий модуль: реалізувати розділ фінансів, у якому буде видно історію купівель, суми транзакцій, статуси оплат та звіти для магазинів.
- Налаштування системи: надати адміністраторам можливість керувати параметрами платформи, включаючи категорії товарів, типи кошиків, правила публікації.

Платформа має бути реалізована на основі React-фреймворку з використанням Redux та RTK Query для управління станом, а також PHP з використанням фреймворку Yii2 для серверної частини. Особливу увагу слід приділити безпеці, продуктивності та масштабованості системи.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

E-Market — екологічний маркетплейс для реалізації непроданих харчових продуктів у закладах торгівлі.

SPA — Single Page Application, односторінковий веб-застосунок.

UI — User Interface, інтерфейс користувача.

UX — User Experience, досвід користувача.

API — Application Programming Interface, інтерфейс прикладного програмування.

REST — Representational State Transfer, архітектурний стиль для побудови веб-сервісів.

JWT — JSON Web Token, формат токена для автентифікації та авторизації.

RBAC — Role-Based Access Control, контроль доступу на основі ролей.

React — JavaScript-бібліотека для побудови інтерфейсу користувача.

Redux — бібліотека для централізованого управління станом у React-застосунках.

RTK Query — інструмент з екосистеми Redux Toolkit для роботи з API та кешування запитів.

Yii2 — PHP-фреймворк для швидкої розробки веб-додатків.

JSON — JavaScript Object Notation, формат обміну даними.

CRUD — Create, Read, Update, Delete, базові операції з даними.

SQL — Structured Query Language, мова запитів до бази даних.

UX-маркер

ВСТУП

E-Market — сучасна екологічна платформа для реалізації непроданих харчових продуктів у закладах торгівлі, розроблена з використанням фреймворку React для фронтенду та PHP (Yii2) для серверної частини. Система спрямована на зменшення харчових відходів шляхом автоматизованої взаємодії між магазинами та кінцевими споживачами.

Функціонал платформи дозволяє магазинам створювати кошики з непроданими продуктами, планувати час їх продажу та видачі, а також автоматично виставляти такі кошики у визначені дні. Це сприяє ефективному використанню ресурсів, зменшенню втрат продукції та підтримці принципів сталого розвитку.

Інтерфейс користувача забезпечує зручний доступ до екологічних пропозицій, дозволяючи покупцям швидко знаходити вигідні кошики, а магазинам — гнучко керувати точками продажу, графіками та статистикою. Додатковою перевагою є наявність фінансового модуля, який дозволяє відстежувати історію купівель, транзакції та формувати звіти.

Об’єктом дослідження виступає система E-Market, призначена для екологічно орієнтованої реалізації непроданих харчових продуктів.

Метою роботи є створення платформи, яка сприятиме зменшенню харчових відходів, підвищенню екологічної відповідальності та ефективному використанню ресурсів у сфері торгівлі.

Предмет дослідження охоплює впровадження сучасних методів веб-програмування із застосуванням бібліотеки React, Redux, RTK Query та фреймворку Yii2 на стороні сервера.

Практичне значення полягає у створенні зручного інтерфейсу для магазинів і покупців, автоматизації процесів публікації кошиків, збору статистики та управління фінансами, що дозволяє підвищити ефективність роботи та екологічну свідомість учасників платформи.

Наукова новизна полягає у поєднанні екологічної ідеї з сучасними

технологіями веб-розробки, що дозволяє створити інструмент сталого розвитку в сфері торгівлі та харчування.

Актуальність теми зумовлена глобальними викликами, пов’язаними з надмірним виробництвом і утилізацією харчових продуктів. Екологічні платформи, такі як E-Market, сприяють зменшенню харчових втрат, підвищенню доступності продукції та формуванню культури відповідального споживання. Вони дозволяють оптимізувати процеси взаємодії між продавцями та покупцями, забезпечуючи прозорість, точність і зручність у щоденній роботі.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

Проблема надлишкового виробництва та утилізації харчових продуктів є однією з найактуальніших у сучасному світі. Щороку мільйони тонн їжі втрачаються або викидаються, незважаючи на її придатність до споживання. Це створює не лише економічні збитки для закладів торгівлі та харчування, а й значне екологічне навантаження на довкілля. У зв'язку з цим виникає потреба у впровадженні інноваційних рішень, які дозволяють ефективно управляти непроданими продуктами та зменшувати обсяги харчових відходів.

У багатьох країнах вже функціонують платформи, що дозволяють реалізовувати непродану продукцію за зниженими цінами. Вони сприяють формуванню культури відповідального споживання та підтримці принципів сталого розвитку. Проте в Україні подібні рішення лише починають набирати обертів, і тому створення екологічного маркетплейсу E-Market є актуальним і перспективним напрямом [4, 12].

Сучасні цифрові технології, зокрема веб-платформи, дозволяють автоматизувати процеси взаємодії між продавцями та покупцями. Використання фреймворку React для побудови інтерфейсу користувача забезпечує високу швидкість роботи, адаптивність та зручність у користуванні. Серверна частина, реалізована на основі PHP-фреймворку Yii2, дозволяє створити надійний API для обробки запитів, управління даними та забезпечення безпеки.

Платформа E-Market дозволяє магазинам створювати кошики з непроданими продуктами, планувати час їх продажу та видачі, а також автоматично виставляти такі кошики у визначені дні. Це сприяє оптимізації внутрішніх процесів, зменшенню втрат продукції та підвищенню екологічної відповідальності бізнесу.

Актуальність дослідження зумовлена необхідністю модернізації підходів до управління непроданими харчовими продуктами. Традиційні методи, що передбачають утилізацію або списання продукції, не відповідають сучасним екологічним стандартам. Впровадження цифрових рішень дозволяє автоматизувати

процеси обліку, продажу та аналітики, зменшуючи навантаження на персонал і підвищуючи ефективність роботи закладів торгівлі.

Зростання інтересу до сталого розвитку, підтримка екологічних ініціатив та розвиток цифрової інфраструктури створюють сприятливі умови для впровадження платформи E-Market. Вона може стати важливим інструментом у боротьбі з харчовими втратами, сприяти формуванню нової моделі споживання та підтримувати екологічну свідомість серед населення.

У контексті проблеми харчових відходів важливо враховувати не лише екологічні, а й соціально-економічні аспекти [3]. Значна частина продуктів, які щодня втрачаються у сфері торгівлі та громадського харчування, могла б бути використана для задоволення потреб населення, особливо в умовах економічної нестабільності. Це створює потенціал для розвитку платформ, які дозволяють перерозподіляти надлишкову продукцію, зменшуючи навантаження на навколишнє середовище та підтримуючи соціальну відповідальність бізнесу.

Згідно з дослідженнями, понад третина всіх вироблених харчових продуктів у світі не доходить до кінцевого споживача [1, 2]. Причини цього явища різноманітні: перевиробництво, неправильне планування закупівель, короткий термін реалізації, а також відсутність ефективних механізмів для повторного використання або реалізації залишків. У цьому контексті цифрові рішення, такі як екологічні маркетплейси, стають ключовими інструментами для подолання цієї проблеми.

Платформа E-Market пропонує новий підхід до управління непроданими продуктами, дозволяючи закладам торгівлі не утилізувати залишки, а реалізовувати їх за зниженими цінами через зручний онлайн-інтерфейс. Такий підхід не лише зменшує обсяги відходів, а й створює додаткову цінність для бізнесу, формуючи позитивний імідж та залучаючи нову категорію екологічно свідомих споживачів.

Використання сучасних технологій, таких як React, Redux, RTK Query та Yii2, дозволяє забезпечити високу продуктивність, безпеку та масштабованість системи. Це відкриває можливості для інтеграції з платіжними сервісами, геолокаційними модулями, системами аналітики та іншими інструментами, що підвищують ефективність платформи.

Таким чином, проблемна область охоплює не лише технічні аспекти розробки веб-застосунку, а й глибокі соціальні та екологічні виклики, які потребують комплексного підходу. Створення платформи E-Market є відповіддю на ці виклики, поєднуючи технології, екологічну відповідальність та інноваційні бізнес-моделі.

1.2. Функціональні можливості та переваги платформи E-Market

E-Market являє собою сучасну екологічну платформу для реалізації непроданих харчових продуктів, створену з використанням передових можливостей фреймворку React для клієнтської частини та PHP (Yii2) для серверної логіки. Система орієнтована на автоматизацію процесів взаємодії між закладами торгівлі та кінцевими споживачами, сприяючи зменшенню харчових відходів і підтримці принципів сталого розвитку.

Платформа дозволяє магазинам створювати кошики з непроданими продуктами, використовуючи зручний інтерфейс для додавання товарів, встановлення ціни, опису, терміну реалізації та часу видачі. Завдяки гнучким налаштуванням, користувачі можуть автоматизувати розклад публікації кошиків, визначаючи дні та години, коли вони мають бути доступні для продажу.

E-Market пропонує розширений функціонал для управління точками видачі, графіками роботи, а також можливість перегляду статистики продажів. Користувачі можуть аналізувати кількість проданих кошиків, популярність товарів, динаміку попиту та формувати звіти для внутрішнього використання.

Фінансовий модуль платформи дозволяє магазинам контролювати історію транзакцій, статуси оплат, суми продажів та інші фінансові показники. Це сприяє прозорості обліку та підвищенню ефективності управління ресурсами.

Незареєстровані користувачі можуть переглядати доступні кошики, ознайомлюватися з описами та умовами продажу, проте функції купівлі та взаємодії з платформою доступні лише після створення облікового запису. Це забезпечує захист даних, контроль доступу та персоналізований досвід користування.

Застосування фреймворку React забезпечує швидку реактивність, адаптивність до різних пристроїв та інтуїтивно зрозумілий інтерфейс. Компонентна структура

дозволяє ефективно реалізовувати функціональні модулі, а інтеграція з Redux і RTK Query — оптимізувати обробку запитів та управління станом застосунку.

Таким чином, E-Market поєднує екологічну ідею з сучасними технологіями, пропонуючи зручний, безпечний та ефективний інструмент для реалізації непроданих продуктів, що відповідає потребам як бізнесу, так і суспільства.

1.3. Висновки до розділу

Підсумовуючи, E-Market є сучасним інструментом, що поєднує екологічну ідею з цифровими технологіями для вирішення актуальної проблеми харчових відходів. Платформа забезпечує зручний і ефективний механізм реалізації непроданих продуктів, дозволяючи закладам торгівлі автоматизувати процеси створення кошиків, планування продажів, управління точками видачі та фінансовим обліком.

E-Market виступає не лише засобом для екологічно відповідального продажу продукції, а й повноцінною платформою, оснащеною всіма необхідними інструментами для аналітики, автоматизації та інтеграції з сучасними веб-сервісами. Її впровадження сприяє оптимізації внутрішніх бізнес-процесів, підвищенню продуктивності та формуванню культури сталого споживання.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. React: концепції, еволюція та можливості сучасної JavaScript-бібліотеки

React — це популярна JavaScript-бібліотека для створення інтерфейсів користувача, яка стала фундаментом сучасної веб-розробки [8]. Вперше представлена компанією Facebook у 2013 році, вона швидко здобула популярність серед розробників завдяки своїй гнучкості, продуктивності та компонентному підходу.

React не є повноцінним фреймворком, як, наприклад, Angular, але його архітектура дозволяє легко інтегруватися з іншими технологіями та масштабуватися під потреби будь-якого проєкту.

Основна ідея React полягає у створенні динамічних, реактивних інтерфейсів, які автоматично оновлюються при зміні стану додатку. Це досягається завдяки використанню віртуального DOM — внутрішнього представлення структури HTML, що дозволяє React ефективно оновлювати лише ті частини сторінки, які змінилися, без повного перерендеру.

Однією з ключових концепцій React є компонентність. Усе в React — це компоненти: від кнопки до цілого розділу сайту. Компоненти можуть бути функціональними або класовими, але з появою React Hooks у версії 16.8 функціональні компоненти стали домінуючими. Компоненти дозволяють розробникам розділяти інтерфейс на незалежні, повторно використовувані частини, що значно спрощує розробку, тестування та підтримку коду.

Іншою важливою концепцією є односпрямований потік даних. У React дані передаються від батьківського компонента до дочірнього через props (властивості). Це забезпечує передбачуваність поведінки додатку та спрощує відстеження змін. Для управління глобальним станом застосунку часто використовуються зовнішні бібліотеки, такі як Redux або Context API, які дозволяють централізовано зберігати та змінювати дані.

React також підтримує JSX (JavaScript XML) — синтаксис, який дозволяє писати HTML-подібний код безпосередньо в JavaScript. Це робить код більш

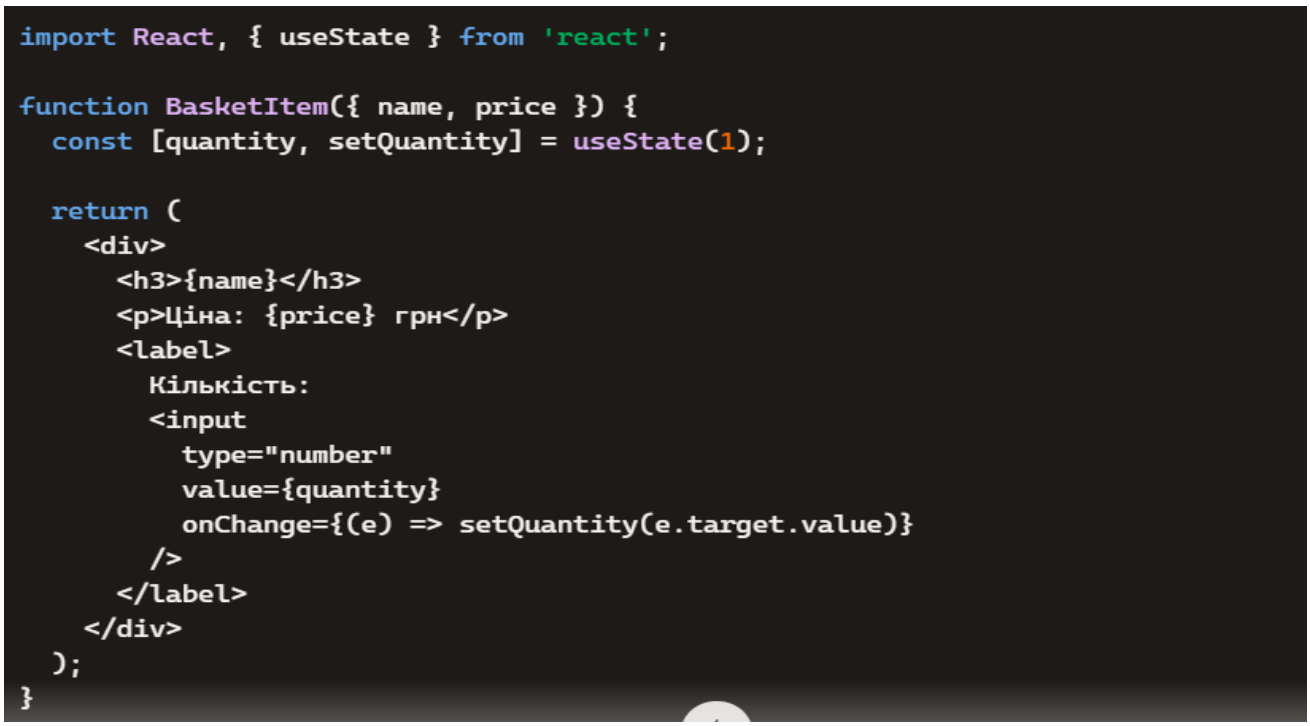
читабельним і дозволяє розробникам швидко створювати інтерфейси, комбінуючи логіку та розмітку в одному місці.

З моменту свого запуску React пройшов значну еволюцію. Спочатку він використовував класові компоненти з методами життєвого циклу, такими як componentDidMount, shouldComponentUpdate тощо. Згодом з'явилися React Hooks, які дозволили використовувати стан (useState), ефекти (useEffect) та інші можливості без необхідності створення класів. Це зробило код більш лаконічним і легким для розуміння.

React активно використовується в розробці як малих, так і великих проєктів. Наприклад, такі компанії як Facebook, Instagram, Airbnb, Netflix, Uber та багато інших використовують React у своїх продуктах. Його гнучкість дозволяє створювати як прості односторінкові додатки (SPA), так і складні корпоративні системи.

У контексті розробки платформи E-Market React відіграє ключову роль у побудові інтерфейсу користувача. Завдяки компонентному підходу можна створити окремі модулі для реєстрації магазинів, формування кошиків, перегляду статистики, управління точками видачі та фінансами. Наприклад, компонент кошика може містити інформацію про товари, ціну, дату продажу та кнопку для оформлення замовлення. Компонент статистики — графіки та таблиці, які оновлюються в реальному часі.

React також добре інтегрується з бекендом, побудованим на PHP (Yii2), через REST API. За допомогою бібліотеки RTK Query можна легко виконувати запити до серверної частини, кешувати відповіді, обробляти помилки та оновлювати інтерфейс без зайвих зусиль. (рис. 2.1)



```jsx
import React, { useState } from 'react';

function BasketItem({ name, price }) {
  const [quantity, setQuantity] = useState(1);

  return (
    <div>
      <h3>{name}</h3>
      <p>Ціна: {price} грн</p>
      <label>
        Кількість:
        <input
          type="number"
          value={quantity}
          onChange={(e) => setQuantity(e.target.value)}
        />
      </label>
    </div>
  );
}
```

Рисунок 2.1 – Приклад простого функціонального компонента React

Цей компонент дозволяє відображати товар у кошику та змінювати його кількість. Завдяки useState ми можемо зберігати локальний стан компонента, а зміни автоматично оновлюють інтерфейс.

React також підтримує портали, фрагменти, контекст, мемоізацію (React.memo, useMemo, useCallback) — усе це дозволяє оптимізувати продуктивність і уникати зайвих перерендерів. Для роботи з формами часто використовуються бібліотеки типу Formik або React Hook Form, а для маршрутизації — React Router.

Ще однією перевагою React є активна спільнота та екосистема. Існує безліч готових компонентів, бібліотек, шаблонів і інструментів, які прискорюють розробку. Наприклад, Create React App дозволяє швидко створити базовий проєкт із налаштованим середовищем, а Vite — забезпечує надшвидку збірку та оновлення під час розробки.

2.2. React.js та його екосистема бібліотек у сучасній веб-розробці

React.js — це JavaScript-бібліотека, призначена для створення інтерфейсів користувача. Вона була розроблена компанією Meta (раніше Facebook) і з моменту свого запуску у 2013 році стала однією з найпопулярніших технологій у

фронтенд-розробці. Основною концепцією React є компонентна архітектура, яка дозволяє розробникам створювати інтерфейс як набір незалежних, повторно використовуваних блоків. Це забезпечує гнучкість, масштабованість та зручність підтримки коду.

Станом на 2025 рік актуальна версія React — 18.3.1. Вона підтримує низку сучасних функцій, таких як Concurrent Rendering, автоматичне групування оновлень (Automatic Batching), Transitions API та Server Components [8]. Ці можливості дозволяють створювати високопродуктивні, інтерактивні та адаптивні вебдодатки.

Однак сам React є лише основою. Його функціональність значно розширюється завдяки великій екосистемі бібліотек, які інтегруються з ним для вирішення різноманітних завдань: від стилізації інтерфейсу до управління станом, роботи з API, тестування, анімацій, форм, графіків тощо.

Однією з найважливіших категорій бібліотек є UI-компоненти та стилізація. Наприклад, Material UI (пакети @mui/material, @mui/icons-material, @mui/x-date-pickers) реалізує дизайн-систему Google Material Design і надає набір готових компонентів, таких як кнопки, діалоги, таблиці, селектори дат. Для стилізації використовується бібліотека Emotion (@emotion/react та @emotion/styled), яка дозволяє застосовувати CSS-in-JS підхід. Альтернативою є NextUI (@nextui-org/system, @nextui-org/theme, @nextui-org/date-input), яка орієнтована на легкість, продуктивність і підтримку Tailwind CSS — утилітарного CSS-фреймворку, що дозволяє стилізувати компоненти без написання окремих CSS-файлів.

Для управління станом додатку широко використовуються Redux та Redux Toolkit (@reduxjs/toolkit) [9]. Вони дозволяють централізовано зберігати дані, синхронізувати компоненти та забезпечувати передбачувану поведінку додатку. Redux Persist (redux-persist) дає змогу зберігати стан між сесіями, що особливо корисно для збереження кошика покупця або налаштувань користувача. Axios — це бібліотека для HTTP-запитів, яка використовується для взаємодії з сервером. Вона підтримує інтерсептори, обробку помилок та авторизацію через токени. Socket.io-client — інструмент для двосторонньої комунікації в реальному часі,

наприклад, для оновлення статусу товарів або повідомлень.

Для роботи з датами та часом використовуються бібліотеки date-fns, date-fns-tz, dayjs та dayjs-plugin-utc. Вони забезпечують форматування, парсинг, обчислення та підтримку часових зон. Компоненти react-datepicker, react-datetime-picker та react-time-picker дозволяють інтегрувати вибір дати й часу у форми та календарі.

Візуалізація даних є важливою частиною сучасних додатків. Для цього використовуються chart.js, chartjs-plugin-annotation, react-chartjs-2 та recharts. Вони дозволяють створювати графіки, діаграми, гістограми, які використовуються для відображення статистики, фінансових звітів, екологічного впливу тощо. React-simple-star-rating — компонент для рейтингу товарів або магазинів, а react-icons — колекція SVG-іконок, яка охоплює FontAwesome, Material Icons, Bootstrap Icons та інші.

Форми та їх валідація реалізуються за допомогою бібліотек Formik та Yup. Formik керує станом форми, обробкою подій та валідацією, а Yup дозволяє описувати правила перевірки даних. React-select, react-custom-checkbox та react-checkbox-list — це компоненти для створення селекторів, чекбоксів та списків з кастомною поведінкою.

Інтернаціоналізація додатків забезпечується через i18next, react-i18next та i18next-browser-languagedetector. Вони дозволяють створювати багатомовні інтерфейси, автоматично визначати мову користувача та завантажувати відповідні ресурси.

Анімації та взаємодія з користувачем реалізуються через framer-motion — потужну бібліотеку для створення плавних переходів, ефектів появи, drag-and-drop. React-beautiful-dnd — компонент для перетягування елементів, наприклад, сортування списків. Lottie-react та react-lottie дозволяють інтегрувати анімовані SVG-файли, створені в After Effects.

Для тестування компонентів використовуються @testing-library/react, @testing-library/jest-dom та @testing-library/user-event. Вони дозволяють перевіряти рендеринг, взаємодію та поведінку UI. Web-vitals — бібліотека для

збору метрик продуктивності, таких як LCP, FID, CLS, що важливо для SEO та UX.

Інші корисні бібліотеки включають react-router-dom — для маршрутизації між сторінками, react-toastify та react-tooltip — для повідомлень та підказок, html2pdf та html2pdf.js — для експорту HTML-сторінок у формат PDF, jwt-decode — для декодування JWT-токенів, google-map-react, @react-google-maps/api та react-geocode — для інтеграції з Google Maps та геолокації..

2.3. Yii2 як кросплатформенне серверне середовище для розробки веб-застосунків

У сучасній веб-розробці серверна частина відіграє ключову роль, забезпечуючи обробку запитів, взаємодію з базами даних, реалізацію бізнес-логіки та формування відповідей для клієнтської частини. Одним із ефективних інструментів для реалізації серверної логіки є PHP-фреймворк Yii2 — потужне, гнучке та високопродуктивне середовище, яке дозволяє створювати масштабовані веб-застосунки з чіткою структурою та високим рівнем безпеки.

Yii2 (від англ. "Yes, it is!") — це фреймворк з відкритим кодом, який базується на мові програмування PHP та реалізує архітектурний шаблон Model–View–Controller (MVC) [11]. Він підтримує об'єктно-орієнтоване програмування, шаблони проектування, RESTful API, інтеграцію з базами даних, кешування, рольову модель доступу (RBAC), а також має вбудовані засоби безпеки та генерації коду. Завдяки своїй модульності та розширюваності Yii2 широко використовується для створення як невеликих сайтів, так і складних корпоративних систем.

Однією з ключових переваг Yii2 є його висока продуктивність. Фреймворк оптимізований для швидкої обробки запитів, ефективної роботи з базами даних та мінімізації навантаження на сервер. Це досягається завдяки використанню механізмів кешування, lazy loading, генерації метаданих, оптимізації SQL-запитів та асинхронної обробки даних. У порівнянні з іншими PHP-фреймворками, Yii2 демонструє стабільну роботу навіть при великій кількості одночасних запитів.

Архітектура Yii2 базується на шаблоні MVC, який забезпечує чітке розділення відповідальностей між компонентами системи. Модель (Model) відповідає за взаємодію з базою даних, включаючи збереження, оновлення, видалення та вибірку даних. Представлення (View) відповідає за генерацію HTML-коду, який відображається користувачеві. Контролер (Controller) обробляє запити, викликає відповідні моделі та передає дані до представлення. Такий підхід сприяє структурованості коду, його повторному використанню та легкій підтримці.

Фреймворк має потужну ORM-систему ActiveRecord, яка дозволяє працювати з базами даних у вигляді об'єктів. Це значно спрощує написання SQL-запитів, забезпечує безпеку та зменшує ризик помилок. ActiveRecord автоматично генерує класи моделей на основі структури таблиць, підтримує зв’язки між таблицями, валідацію даних, фільтрацію та сортування. Крім того, Yii2 підтримує генерацію CRUD-інтерфейсів (Create, Read, Update, Delete), що дозволяє швидко створювати функціональні модулі для управління даними.

Ще однією важливою особливістю Yii2 є його система маршрутизації. Вона дозволяє гнучко керувати URL-адресами, створювати чисті та логічні маршрути, реалізовувати RESTful API та забезпечувати інтеграцію з фронтенд-фреймворками, такими як React.js.. Завдяки цьому можлива ефективна взаємодія між клієнтською та серверною частинами, що є критично важливим для SPA-додатків (Single Page Application).

Безпека — один із пріоритетів Yii2. Фреймворк має вбудовані механізми захисту від поширених загроз, таких як SQL-ін’єкції, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery). Він підтримує автентифікацію та авторизацію користувачів, рольову модель доступу (RBAC), шифрування даних, захист сесій та контроль доступу до ресурсів. Це дозволяє створювати надійні системи, які відповідають сучасним стандартам безпеки.

Yii2 також має зручний консольний інструмент Gii — генератор коду, який дозволяє автоматизувати створення моделей, контролерів, форм, валідаційних правил та інших елементів. Це значно прискорює процес розробки, зменшує

кількість рутинної роботи та забезпечує узгодженість структури коду. Gii підтримує шаблони, кастомізацію та інтеграцію з IDE, що робить його корисним інструментом для командної розробки.

Фреймворк підтримує модульну структуру, що дозволяє розширювати функціональність додатку через встановлення додаткових пакетів, віджетів та модулів. Існує велика кількість готових розширень для інтеграції з платіжними системами, поштовими сервісами, аналітикою, багатомовністю, графіками, календарями тощо. Завдяки цьому розробники можуть швидко додавати нові можливості без необхідності створювати їх з нуля.

У контексті серверної логіки Yii2 забезпечує обробку HTTP-запитів, взаємодію з базою даних, реалізацію бізнес-правил, формування відповідей у форматі JSON, XML або HTML, логування, обробку помилок та винятків, а також управління сесіями та користувачами. Це дозволяє створювати повноцінні бекенд-системи, які можуть обслуговувати клієнтські додатки, мобільні застосунки, зовнішні API та інтеграції з іншими сервісами.

Завдяки підтримці RESTful API, Yii2 дозволяє створювати серверні інтерфейси, які легко інтегруються з фронтенд-фреймворками, такими як React, Vue або Angular. Це забезпечує гнучкість архітектури, можливість масштабування та розділення обов’язків між клієнтською та серверною частинами. Крім того, Yii2 підтримує CORS (Cross-Origin Resource Sharing), що дозволяє безпечно обробляти запити з різних доменів.

Ще одним важливим аспектом є підтримка тестування. Yii2 включає засоби для юніт-тестування, функціонального тестування, інтеграційного тестування та тестування REST API. Це дозволяє забезпечити стабільність системи, виявляти помилки на ранніх етапах та підтримувати якість коду.

Таким чином, Yii2 є потужним серверним середовищем, яке поєднує продуктивність, безпеку, гнучкість та зручність розробки. Його використання дозволяє створювати сучасні веб-застосунки з чіткою архітектурою, ефективною взаємодією з клієнтською частиною та високим рівнем надійності. У контексті екологічних платформ, таких як E-Market, Yii2 забезпечує стабільну роботу

серверної логіки, обробку запитів, управління даними та реалізацію функціональності, спрямованої на сталий розвиток.

2.4. Висновки до розділу

У другому розділі було здійснено аналіз сучасних технологій веб-розробки, які використовуються для створення екологічної платформи E-Market. Розглянуто особливості клієнтської та серверної частин системи, зокрема бібліотеку React.js та фреймворк Yii2, які забезпечують ефективну взаємодію між користувачем і сервером.

React.js, як фронтенд-бібліотека, продемонструвала високу гнучкість, модульність та продуктивність завдяки компонентній архітектурі, підтримці хуків, управлінню станом через Redux та RTK Query, а також широкому спектру UI-бібліотек. Це дозволило реалізувати адаптивний, інтерактивний та зручний інтерфейс, що відповідає сучасним вимогам до екологічних веб-застосунків.

Зі свого боку, Yii2 виконує роль серверного середовища, яке забезпечує обробку запитів, управління базами даних, реалізацію бізнес-логіки та формування API-відповідей. Завдяки підтримці MVC-архітектури, ORM-системи ActiveRecord, механізмів безпеки та генерації коду, Yii2 дозволяє створювати стабільну, масштабовану та захищену серверну частину платформи.

Синергія між React.js та Yii2 забезпечує ефективну реалізацію екологічної платформи, яка спрямована на зменшення харчових відходів, автоматизацію процесів торгівлі та підтримку принципів сталого розвитку. Обрані технології дозволяють досягти високої продуктивності, зручності користування, надійності та гнучкості системи, що є критично важливим для її успішного функціонування в реальних умовах.

Таким чином, аналіз інструментів розробки підтвердив доцільність їх використання для реалізації екологічно орієнтованої платформи, що поєднує сучасні технології з соціально значущими цілями.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Формалізація задачі екологічної реалізації непроданих харчових продуктів

Для побудови ефективної автоматизованої системи E-Market, що забезпечує реалізацію непроданих харчових продуктів, здійснено математичну формалізацію задачі. Метою формалізації є чітке визначення основних сутностей, змінних, обмежень та цільової функції системи.

Формалізація починається з визначення трьох ключових множин об'єктів (сутностей), які взаємодіють у системі:

Продукти (товари, що підлягають реалізації): Множина I = {i_1, i_2, … , i_N}, де N – загальна кількість унікальних одиниць товару.

Продавці (заклади харчування/торгові точки): Множина S = {s_1, s_2, … , s_M}.

Покупці (кінцеві споживачі): Множина U = {u_1, u_2, … , u_K}.

Кожен продукт i ∈ I формально описується вектором характеристик A_i:

$$\mathbf{A}_i = (T_{exp,i}, C_{cat,i}, P_i, V_i, G_{loc,i}, E_{score,i}) \tag{3.1}$$

Де:

T_{exp, i} – **Термін придатності** (дата та час, до якого продукт має бути реалізований).

C_{cat, i} – **Категорія продукту** (наприклад, готова страва, хлібобулочний виріб, напівфабрикат).

P_i – **Початкова вартість** одиниці продукту (до знижки).

V_i – **Вага або об’єм** одиниці продукту.

G_{loc, i} – **Геолокація** продавця s ∈ S, який пропонує товар.

E_{score, i} – **Екологічна оцінка** (потенційний негативний вплив у разі утилізації).

Функціонування платформи відбувається в умовах жорстких обмежень, які система **використовує** для прийняття рішень:

1. Часове обмеження

Ключовим параметром, який враховує система, є час до завершення терміну придатності T_{rem, i}:

$$T_{rem,i} = T_{exp,i} - T_{current} \tag{3.2}$$

Де T_current – поточний час.

Умова доступності продукту:

Продукт і доступний для реалізації, якщо:

$$T_{rem,i} > T_{buffer} \tag{3.3}$$

Де T_buffer – мінімальний часовий буфер (наприклад, 2 години), необхідний для обробки замовлення та видачі.

2. Географічне обмеження

Система забезпечує фільтрацію пропозицій за географічною відстанню. Відстань між продавцем G_{loc, s} та покупцем G_{loc, u} розраховується як D(G_{loc, s}, G_{loc, u}). Умова релевантності: Пропозиція продавця s релевантна для покупця u, якщо:

$$D(G_{loc,s}, G_{loc,u}) \leq D_{max} \tag{3.4}$$

Де D_max – максимально допустима відстань, визначена користувачем або системою.

3. Екологічний пріоритет

Система **базується** на пріоритетності реалізації продуктів, утилізація яких **має** найвищий негативний вплив.

Принцип пріоритету:

Чим вище значення E_{score, i} (шкідливість утилізації), тим вищий пріоритет Pr_{i} надається продукту для включення його у кошик.

Цільова задача системи E-Market – максимізація загальної корисності F, яка **враховує** економічний, соціальний та екологічний аспекти.

$$F \to \max \tag{3.5}$$

Функція корисності F **визначається** як зважена сума трьох ключових компонентів:

$$F = W_{econ} \cdot F_{econ} + W_{env} \cdot F_{env} + W_{soc} \cdot F_{soc} \quad (3.6)$$

Де:

F_econ – **Економічна вигода** (наприклад, сумарний дохід від реалізованих продуктів).

F_env – **Екологічна вигода** (наприклад, сумарне зменшення потенційного екологічного навантаження від **неутилізованих** продуктів).

F_soc – **Соціальна вигода** (наприклад, рівень задоволеності користувачів, кількість транзакцій).

W_econ, W_env, W_soc – Вагові коефіцієнти, що **визначають** пріоритетність цілей $\sum W = 1.$.

Таким чином, формалізація задачі дозволяє чітко визначити вхідні параметри (A_i), обмеження (T_{rem, i}, D_max) та цільову функцію (F), що **є основою** для розробки математичних моделей та алгоритмічного забезпечення.

3.2. Математичне моделювання взаємодії між продавцями та покупцями

Математичне моделювання взаємодії у системі E-Market **базується** на двох ключових процесах: **формування пропозиції** продавцем та **прийняття рішення** покупцем. Взаємодія **моделюється** як задача оптимізації, де продавець прагне мінімізувати втрати, а покупець – максимізувати корисність покупки.

3.2.1. Моделювання формування кошика (Продавець)

Продавець $s \in S$ **формує** кошики B_j, які є підмножиною доступних непроданих продуктів $I_s \subset I$. Кожен кошик j **характеризується** набором продуктів $i \in B_j$, загальною ціною C_{total, j} та часом видачі T_pickup, j.

А. Функція вартості кошика

Система **визначає** загальну вартість кошика C_{total, j} (ціна продажу) на основі початкової вартості товарів, знижки D та додаткового екологічного

коефіцієнта K_env, який залежить від середньої екологічної оцінки продуктів у кошику E_{score, j}.

$$C_{total,j} = (1 - D) \cdot \left(\sum_{i \in \mathbf{B}_j} P_i \right) \cdot (1 + K_{env}(\bar{E}_{\text{score},j})) \tag{3.7}$$

Де:

$\sum_{i \in \mathbf{B}_j} P_i$ – сумарна початкова ціна всіх продуктів у кошику.

D – базова відсоткова знижка, що **застосовується** до всіх кошиків (як інструмент реалізації).

$K_{env}(\bar{E}_{\text{score},j})$ – корекційний коефіцієнт ціни, який **стимулює** продаж кошиків з товарами, що мають вищий ризик утилізації (вищий E_ score).

B. Правило формування кошиків (Алгоритмічна логіка)

Система **використовує** правило, яке **мінімізує** ризик утилізації R_{loss, j}:

$$R_{loss,j} \to \min \tag{3.8}$$

Кошик B}_j **формується** шляхом об'єднання продуктів i з подібними характеристиками (категорія C_{cat, i}) та обмеженим терміном придатності.

Критерій для включення продукту i у кошик B_j:

Продукт i включається у кошик j, якщо:

$$\frac{T_{rem,i}}{T_{\max}} \leq \delta \tag{3.9}$$

Де T_{rem, i} – час, що залишився до терміну придатності; T_max – загальний термін придатності; delta – пороговий коефіцієнт терміновості (наприклад, 0.25, що означає, що залишилося менше 25% часу).

Рішення покупця u ∈ U про придбання конкретного кошика B_j з множини доступних пропозицій J **моделюється** як задача максимізації корисності, що базується на теорії раціонального вибору. Система **використовує** логістичну модель (Logit Model) для оцінки ймовірності вибору.

A. Функція корисності покупця (Utility Function)

Корисність U_{u, j}, яку покупець u отримує від придбання кошика B_j, **визначається** лінійною функцією, яка **інтегрує** економічні, логістичні, відповідні та екологічні атрибути пропозиції:

$$U_{u,j} = W_{price} \cdot F_{price,j} - W_{dist} \cdot D(G_{loc,j}, G_{loc,u}) + W_{comp} \cdot F_{comp,j} + W_{env} \cdot \bar{E}_{\text{score},j}$$

(3.10)

Де:

$F_{price,j} = \left(\frac{C_{\max} - C_{total,j}}{C_{\max}}\right)$ – **Нормований фактор цінової привабливості.** Цей показник **відображає** економічну вигоду для покупця, де C_max — максимальний пороговий бюджет, який **визначається** на основі профілю покупця.

D(G_{loc, j}, G_{loc, u}) – **Географічна дистанція** між покупцем u та місцем видачі кошика j. Це **негативний фактор** корисності (W_{dist} > 0).

F_{comp, j} – **Фактор відповідності складу**, який **оцінює** узгодженість вмісту кошика B_j з історичними перевагами покупця u.

E_{score, j} – **Середня екологічна оцінка** продуктів у кошику (позитивний фактор).

W_price, W_dist, W_comp, W_env – **Вагові коефіцієнти**, які **відображають** індивідуальні пріоритети покупця u (наприклад, для покупця з високою екологічною свідомістю W_env буде вищим). **Виконується** умова нормування: W = 1.

B. Ймовірність вибору кошика (Logit Model)

Ймовірність P_u(j) того, що покупець u обере конкретний кошик j з множини доступних пропозицій J, **розраховується** за допомогою Мультиноміальної Логістичної Моделі (MNL):

$$P_u(j) = \frac{\exp(\lambda \cdot U_{u,j})}{\sum_{k \in J} \exp(\lambda \cdot U_{u,k})}$$

(3.11)

Де lambda – **Коефіцієнт чутливості (Scale Parameter)**, який **визначає** рівень гомогенності уподобань користувачів. Високе значення lambda **свідчить** про те, що вибір більш детермінований (раціональний) і **сильно залежить** від значення

корисності U_{u, j}. Ця модель **дозволяє** системі прогнозувати попит на різні типи кошиків та оптимізувати ціноутворення та час публікації.

Для стимулювання екологічно відповідальної поведінки покупців система **реалізує** механізм накопичення екологічного рейтингу R_{env, u}.

Формула розрахунку екологічного рейтингу:

Накопичувальний екологічний рейтинг покупця u на момент часу t, $R_{env,u}^{(t)}$, **обчислюється** як сума попереднього рейтингу та загальної екологічної оцінки всіх успішно "врятованих" від утилізації продуктів:

$$R_{env,u}^{(t)} = R_{env,u}^{(t-1)} + \sum_{j \in \text{Purchases}_t} \left(\sum_{i \in \mathbf{B}_j} E_{score,i} \cdot F_{succ} \right) \tag{3.12}$$

Де:

Purchases_t – **Множина успішних транзакцій** покупця, завершених до моменту t.

E_{score, i} – **Екологічна оцінка** (потенційний негативний вплив), який **був попереджений** завдяки покупці.

F_succ – **Коефіцієнт успішності** транзакції F_succ=1 для завершеної покупки).

Цей рейтинг **використовується** системою як пріоритетний фактор (наприклад, у формулі Pr_{issue, u}), що **забезпечує** соціальну вигоду для відповідальних покупців і **підсилює** мотивацію до сталого споживання.

3.3 Алгоритмічне забезпечення автоматизованої публікації та видачі кошиків

Алгоритмічне забезпечення системи E-Market забезпечує автоматизацію ключових операційних циклів: формування, публікації та видачі кошиків. Ці алгоритми базуються на математичних критеріях (час, екологічна оцінка, попит), визначених у попередніх розділах, для досягнення цілей оптимізації та мінімізації втрат.

Алгоритм Algorithm_Pub відповідає за визначення складу кошика, його ціни та оптимального часу публікації.

Крок 1. Аналіз залишків та визначення пріоритету.Система сканує множину непроданих продуктів I_s від продавця s та визначає пріоритет реалізації Pr_i для кожного продукту i на основі часу, що залишився T_{rem, i}, та екологічної оцінки E_{score, i}:

$$Pr_i = W_T \cdot \left(\frac{1}{T_{rem,i}} \right) + W_E \cdot E_{score,i} \tag{3.13}$$

Де W_T, W_E – вагові коефіцієнти, що визначають важливість часу та екологічної шкоди. Продукти сортуються за спаданням Pr_i.

Крок 2. Формування логічних кошиків.

Продукти об'єднуються у кошики B_j за критерієм сумісності (наприклад, одна категорія C_{cat, i} та спільним часовим вікном видачі T_{pickup, j}

$$\mathbf{B}_j = \{ i \in I_s \mid \mathbf{SameCat}(i) \land T_{pickup,j} = T_{\text{slot}} \} \tag{3.14}$$

Крок 3. Визначення ціни та часу публікації.

Час публікації $T_{\text{post}, j}$ визначається на основі прогнозу попиту P_demand(T):

$$T_{\text{post},j} = \arg\max_{T} \{ \hat{P}_{\text{demand}}(T) \mid T \leq T_{pickup,j} - T_{buffer} \} \tag{3.15}$$

Де P_demand(T) – прогнозована ймовірність купівлі кошика типу j у момент часу T, базована на історичній активності користувачів.

Крок 4. Публікація. Кошик B_j отримує статус «Активний» і стає доступним для перегляду покупцям. Алгоритм Algorithm_Issue керує процесом резервування та фізичної видачі кошика.

Крок 1. Обробка резервування. При отриманні запиту на резервування кошика j від покупця u, система перевіряє наявність (статус «Активний») та блокує кошик.

Крок 2. Формування черги та пріоритет видачі. Система встановлює чергу видачі Q_u на основі часу резервування T_res та екологічного рейтингу покупця R_{env, u}. Покупець u отримує пріоритет видачі Pr_{issue, u}:

$$Pr_{issue,u} = W_R \cdot R_{env,u} - W_T \cdot T_{res} \tag{3.16}$$

Де W_R та W_T – вагові коефіцієнти. Чим вищий Pr_{issue, u}, тим вищий пріоритет.

Крок 3. Контроль пропускної здатності пункту видачі. Система враховує графік роботи та максимальне навантаження L_max(T_slot) для пункту видачі.

Умова видачі: Видача кошика j дозволяється у часовий слот T_slot, якщо:

$$L_{\text{current}}(T_{\text{slot}}) < L_{\max}(T_{\text{slot}}) \tag{3.17}$$

Де L_current – поточна кількість зарезервованих видач у цьому слоті. Якщо умова не виконується, покупцю пропонується альтернативний час.Крок 4. Завершення транзакції та оновлення статусу.Після фактичної видачі кошика статус транзакції змінюється на «Успішно реалізовано». Система запускає оновлення:

- Оновлення екологічного рейтингу R_{env, u} покупця.
- Оновлення статусу продуктів у базі даних (перехід з «Не продано» на «Реалізовано»).

Система реалізує механізм контролю, який працює за таймером Delta T:

Логіка оновлення:

- Якщо кошик B_j не був зарезервований протягом T_{time_limit} після публікації, система ініціює зміну ціни (додаткову знижку) або його переформатування (зміна складу та часу видачі).
- Якщо T_{rem, i} ≤ T_zero, продукти, що не були реалізовані, автоматично отримують статус «Утилізовано», і здійснюється фіксація відповідного екологічного навантаження.

3.4. Структура та організація бази даних

База даних **виступає** основою для зберігання та обробки даних, необхідних для роботи математичної моделі та алгоритмічного забезпечення системи E-Market. Вона **реалізована** на реляційній моделі та **забезпечує** нормалізацію даних (уникнення дублювання) та цілісність завдяки використанню зовнішніх ключів.

Система **оперує** ключовими сутностями, які представлені у вигляді таблиць. Нижче **наведено** формалізований опис основних таблиць (сутностей) та їх міжсутностних зв'язків.

1. Користувачі (T_Users)

Призначення: Зберігання інформації про покупців та продавців.

Ключові поля:

- user_id (PK, Primary Key) – Унікальний ідентифікатор.
- user_type – Тип (Покупець/Продавець).
- geo_location – Географічні координати.
- R_env – Екологічний рейтинг

2. Продукти (T_ Products)

Призначення: Детальна інформація про кожен непроданий товар.

Ключові поля:

- product_id (PK).
- seller_id (FK, Foreign Key до **T_Users)** – Продавець.
- T_exp – Термін придатності (T_{exp, i}).
- E_score – Екологічна оцінка (E_{score, i}).
- status – Статус (Активний, Реалізований, Утилізований).

3. Кошики (T_Baskets)

Призначення: Логічні групи продуктів, що пропонуються до продажу.

Ключові поля:

- basket_id (PK).
- seller_id (FK до T_Users).
- pickup_point_id (FK до T_Points).

- C_total – Фінальна вартість кошика (C_{total, j}).
- T_post – Час публікації (T_{post, j}).

4. Зміст Кошика (T_Basket_Items)

Призначення: Таблиця зв'язку багато-до-багатьох (M:N) між продуктами та кошиками.

Ключові поля:

- basket_id (FK до T_Baskets).
- product_id (FK до T_Products).

5. Транзакції (T_ Transactions)

Призначення: Фіксація факту купівлі та оплати.

Ключові поля:

- trans_id (PK).
- buyer_id (FK до T_Users).
- basket_id (FK до T_Baskets}).
- T_res – Час резервування (T_res).
- amount – Сума.

6. Пункти видачі (T_Points)

Призначення: Географічні та логістичні дані про місця видачі.

Ключові поля:

- point_id (PK).
- geo_location – Координати пункту.
- L_max – Максимальна пропускна здатність (L_max).

7. Графіки видачі (T_Schedules)

Призначення: Часові слоти для контролю навантаження.

Ключові поля:

- slot_id (PK).
- point_id (FK до T_Points).
- T_slot – Часовий інтервал.
- L_current – Поточне навантаження (L_current).

Взаємозв'язок таблиць **забезпечує** цілісність даних та **є необхідним** для виконання складних математичних запитів (наприклад, розрахунок D(G_{loc, s}, G_{loc, u})або оновлення R_env).

Зв'язок 1:N (Один до Багатьох):

T_Users (Продавець) ↔ T_Products: Один продавець **може мати** багато продуктів.

T}Users (Продавець) ↔ T_Baskets: Один продавець **може створити** багато кошиків.

T_Points ↔ T_Schedules: Один пункт видачі **має** багато часових слотів.

Зв'язок M:N (Багато до Багатьох):

T_Products ↔T_Baskets: Реалізовано через проміжну таблицю T_Basket_Items, оскільки один продукт **може бути** у кількох кошиках (якщо це ідентичні одиниці товару), і один кошик **містить** багато продуктів.

Взаємодія з цією структурою **здійснюється** через механізм ORM (Object-Relational Mapping), зокрема, через фреймворк Yii2. Це **дозволяє** алгоритмічному забезпеченню працювати з даними як з об'єктами, що **спрощує** застосування формул.

Приклад (ORM-запит для розрахунку):

Для застосування формули загальної вартості кошика (C_{total, j}):

$$C_{total,j} = (1 - D) \cdot \left(\sum_{i \in \mathbf{B}_j} P_i \right) \cdot (1 + K_{env}) \tag{3.18}$$

Алгоритм виконує запит, який використовує зв'язки T_Baskets → T_ Basket_Items → T_Products для агрегації суми початкових цін P_i.

3.5. Висновки до розділу

У третьому розділі **здійснено** комплексне математичне обґрунтування функціонування екологічної платформи E-Market, що **було досягнуто** шляхом формалізації задачі, розробки математичних моделей взаємодії, визначення алгоритмічного забезпечення та структури бази даних.

Ключові наукові результати розділу:

Формалізація задачі та цільова функція: Введено формалізований опис ключових об'єктів (Продукти I, Продавці S, Покупці U) та їх характеристик A_i. **Побудовано** цільову функцію корисності F → max, яка **інтегрує** економічні (F_econ), екологічні (F_env) та соціальні (F_soc) критерії, що **забезпечує** багатовимірну оптимізацію діяльності платформи.

Моделювання взаємодії та прийняття рішень: Розроблено математичні моделі для обох сторін взаємодії:

Для **Продавця** – **введено** формулу розрахунку фінальної вартості кошика C_{total, j}, яка **включає** базову знижку та екологічний корекційний коефіцієнт K_{env}.

Для **Покупця** – **побудовано** функцію корисності U_{u, j}, яка **описує** раціональний вибір на основі зважених параметрів (ціна, відстань, екологічний фактор) та **використовується** для прогнозування ймовірності покупки P_{u}(j).

Введено механізм розрахунку та оновлення екологічного рейтингу покупця R_{env, u}, що **слугує** інструментом мотивації та соціальної відповідальності.

Алгоритмічне забезпечення процесів: Сформульовано алгоритмічну логіку для автоматизованої роботи системи.

Розроблено алгоритм Algorithm_Pub, який **визначає** пріоритет реалізації Pr_i на основі T_{rem, i} та E_{score, i} і **оптимізує** час публікації T_{post}, j} за прогнозом попиту.

Впроваджено алгоритм Algorithm_Issue, який **керує** видачею та **забезпечує** контроль пропускної здатності пунктів видачі L_current (T_{slot})$ для уникнення черг.

Структура та організація бази даних: Спроектовано реляційну структуру бази даних, яка **складається** з ключових таблиць (T_Users, T_Products, T_Baskets, T_Transactions) та **формалізовано** їхні зв'язки. Ця структура **забезпечує** необхідну цілісність даних для ефективного застосування розроблених математичних моделей та алгоритмів.

Таким чином, розроблений математичний апарат **надає** необхідну теоретичну базу для реалізації платформи E-Market як ефективного та науково обґрунтованого інструменту для мінімізації харчових відходів та підтримки цілей сталого розвитку.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Розробка інтелектуальної системи для екологічної взаємодії між закладами торгівлі та споживачами

E-Market — це інтелектуальна платформа, створена для автоматизованої реалізації непроданих харчових продуктів у закладах торгівлі. Система розроблена з урахуванням принципів сталого розвитку, цифрової трансформації та екологічної відповідальності. Її архітектура поєднує сучасні технології фронтенду (React.js) та серверної логіки (PHP, Yii2), що забезпечує високу продуктивність, гнучкість і масштабованість.

На відміну від традиційних платформ, E-Market не просто агрегує пропозиції — вона формує екологічно орієнтовану модель взаємодії між продавцями та покупцями. Ключовим елементом системи є механізм автоматизованого створення «екокошиків» — логічних груп товарів, які залишилися непроданими, але мають потенціал для реалізації. Ці кошики формуються на основі залишків продукції, термінів придатності, категорій товарів, екологічної оцінки та прогнозованого попиту.

Інтерфейс платформи побудований на компонентній структурі React, що дозволяє розділити функціональність на незалежні модулі: перегляд кошиків, фільтрація за категоріями, управління точками видачі, статистика продажів, фінансові звіти. Такий підхід забезпечує повторне використання коду, легкість у підтримці та можливість швидкого масштабування.

Однією з ключових переваг використання React є віртуальний DOM, який дозволяє оновлювати інтерфейс без повного перерендеру сторінки. Це особливо важливо для динамічних елементів, таких як таймери видачі, статуси кошиків, повідомлення про нові пропозиції. Завдяки цьому платформа залишається швидкою та чуйною навіть при високому навантаженні.

Система реалізує односторонній потік даних, що дозволяє ефективно контролювати стан компонентів. Наприклад, при зміні статусу кошика на «вичерпаний» відповідні компоненти автоматично оновлюються у всіх частинах

інтерфейсу. Це забезпечує узгодженість даних, стабільність роботи та зменшення кількості помилок.

Для управління глобальним станом використовується Redux у поєднанні з RTK Query, що дозволяє централізовано зберігати інформацію про користувачів, кошики, транзакції, екологічні рейтинги. Це дає змогу синхронізувати дані між різними модулями, забезпечити кешування запитів та оптимізувати взаємодію з сервером.

Завдяки гнучкості React, платформа легко інтегрується з додатковими бібліотеками: React Router — для маршрутизації, Formik — для форм, Chart.js — для візуалізації статистики, i18next — для багатомовності. Це дозволяє адаптувати систему до потреб різних користувачів, регіонів та сценаріїв використання.

Особливу увагу приділено безпеці та конфіденційності. Зареєстровані користувачі мають доступ до повного функціоналу: бронювання кошиків, перегляд історії покупок, управління точками видачі, формування звітів. Незареєстровані користувачі можуть ознайомитися з екологічними пропозиціями у демо-режимі, з обмеженим доступом до деталей. Такий підхід дозволяє зберігати баланс між відкритістю платформи та захистом персональних даних.

Процес розробки системи складався з кількох етапів:

- Аналіз проблеми — дослідження причин утворення харчових відходів, поведінки споживачів, логістичних обмежень.
- Проєктування архітектури — визначення структури бази даних, логіки взаємодії, модулів інтерфейсу.
- Розробка фронтенду — створення компонентів React, реалізація маршрутизації, інтеграція з API.
- Розробка бекенду — побудова RESTful API на Yii2, реалізація бізнес-логіки, обробка запитів.
- Тестування та оптимізація — перевірка функціональності, продуктивності, безпеки, UX.

- Запуск MVP — публікація першої версії платформи, збір зворотного зв’язку, подальше вдосконалення.

Платформа E-Market не є статичним рішенням — вона постійно розвивається. Завдяки модульній структурі, система легко адаптується до нових вимог: додавання нових категорій продуктів, інтеграція з платіжними системами, розширення екологічної аналітики, впровадження мобільної версії.

Програмна реалізація E-Market передбачає чітке розділення функціональних зон, що дозволяє забезпечити стабільну роботу системи, її масштабованість та гнучкість. Архітектура платформи побудована за принципом модульності, де кожен компонент виконує окрему роль, але водночас інтегрується у загальну логіку взаємодії. Такий підхід дозволяє легко оновлювати окремі частини системи без ризику порушення її цілісності.

Користувацький інтерфейс реалізовано на основі React-компонентів, які відповідають за відображення кошиків, фільтрацію пропозицій, управління точками видачі, перегляд статистики, формування звітів, обробку транзакцій. Кожен компонент має власний стан, який контролюється через хуки (useState, useEffect, useMemo), а глобальний стан синхронізується через Redux. Це дозволяє забезпечити узгодженість даних між різними частинами інтерфейсу, а також швидке реагування на дії користувача.

З боку бекенду, фреймворк Yii2 реалізує RESTful API, що дозволяє фронтенду отримувати, оновлювати та видаляти дані у форматі JSON. Серверна логіка включає обробку запитів, валідацію даних, управління сесіями, авторизацію, генерацію звітів, облік транзакцій, планування видачі кошиків. Завдяки ORM ActiveRecord, робота з базою даних здійснюється через об’єктну модель, що спрощує розробку та забезпечує безпеку.

Особливу увагу приділено ролям користувачів. У системі передбачено два основних типи: продавці (заклади торгівлі) та покупці (кінцеві споживачі). Продавці мають доступ до модуля управління кошиками, де вони можуть переглядати залишки продукції, формувати нові кошики, планувати час видачі, переглядати статистику реалізації. Покупці, у свою чергу, мають доступ до екологічних

пропозицій, можуть фільтрувати їх за категоріями, ціною, відстанню, бронювати кошики, переглядати історію покупок, формувати власний екологічний рейтинг.

Система також реалізує механізм екологічної мотивації. Кожен користувач має індекс екологічної активності, який формується на основі кількості реалізованих кошиків, відмови від утилізації, участі у програмі лояльності. Цей рейтинг впливає на доступ до спеціальних пропозицій, пріоритет у черзі, бонуси, рекомендації. Такий підхід дозволяє не лише стимулювати відповідальне споживання, а й формувати спільноту екологічно свідомих користувачів.

Ще одним важливим аспектом є адаптивність платформи. Інтерфейс автоматично підлаштовується під тип пристрою — десктоп, планшет, смартфон. Це забезпечується через використання бібліотек стилізації, таких як Tailwind CSS та Emotion, а також через реалізацію адаптивних компонентів. Завдяки цьому користувачі можуть зручно працювати з платформою у будь-яких умовах — вдома, на роботі, у дорозі.

Безпека системи реалізована на кількох рівнях. По-перше, усі запити до серверу проходять через механізми валідації та фільтрації. По-друге, автентифікація здійснюється через JWT-токени, що дозволяє захистити сесії та уникнути несанкціонованого доступу. По-третє, дані користувачів зберігаються у зашифрованому вигляді, а доступ до конфіденційної інформації обмежено ролями. Крім того, система регулярно проводить аудит безпеки, оновлення бібліотек та резервне копіювання бази даних.

Інтелектуальна складова платформи реалізується через механізми аналітики та прогнозування. Система аналізує історичні дані про попит, поведінку користувачів, ефективність кошиків, екологічний вплив. На основі цих даних формуються рекомендації для продавців щодо формування кошиків, часу публікації, складу пропозицій. Покупцям, у свою чергу, пропонуються персоналізовані кошики, які відповідають їхнім уподобанням, бюджету та екологічним пріоритетам.

У перспективі платформа може бути доповнена модулями машинного навчання, які дозволять ще точніше прогнозувати попит, виявляти закономірності у

поведінці користувачів, оптимізувати логістику. Також можливе впровадження геоаналітики, яка дозволить враховувати транспортні маршрути, щільність населення, доступність точок видачі.

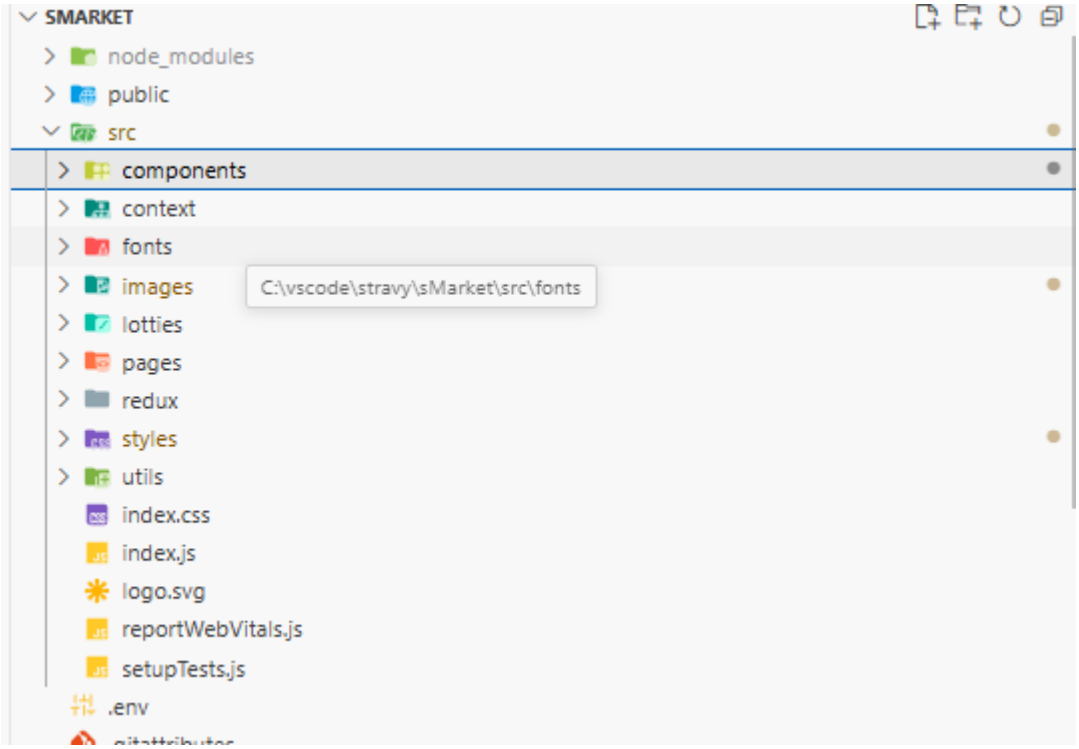


Рисунок 4.1 – Структурна схема клієнтської частини екологічної платформи E-Market

Під час створення клієнтської частини екологічної платформи E-Market було реалізовано чітко структуровану файлову організацію, яка відповідає принципам компонентного підходу та забезпечує зручність у підтримці, масштабуванні й розширенні функціоналу. Вся архітектура побудована на основі фреймворку React, що дозволяє ефективно розділити інтерфейс на незалежні модулі та забезпечити швидку взаємодію з користувачем де представлено загальну схему структури клієнтського проєкту, яка демонструє логічне розташування основних файлів і каталогів, що використовуються в системі. [див. рис. 4.1]:

src/main.js — це стартовий файл, який ініціалізує рендеринг головного компонента платформи. Саме тут відбувається інтеграція React-інтерфейсу з HTML-структурою сторінки. Застосовується метод createRoot() з бібліотеки ReactDOM, що забезпечує сучасний підхід до рендерингу з підтримкою Concurrent Mode.

src/App.jsx — головний контейнер, який об’єднує всі ключові компоненти інтерфейсу. Він відповідає за маршрутизацію, загальне оформлення, підключення глобального стану та обробку системних повідомлень. У цьому файлі реалізовано логіку переходів між сторінками, а також базову структуру макету.

src/modules/ — каталог, який містить функціональні блоки платформи, розділені за логікою: «Кошики», «Профіль», «Статистика», «Пункти видачі», «Фінанси». Кожен модуль включає власні компоненти, стилі, утиліти та локальні API-запити. Така організація дозволяє розробникам працювати над окремими частинами системи незалежно, не порушуючи загальну архітектуру.

src/api/ — директорія, що містить обгортки для взаємодії з серверною частиною, реалізованою на Yii2. Тут зберігаються функції для надсилання запитів, обробки відповідей, авторизації, кешування та обробки помилок. Всі запити структуровані за типами ресурсів, що дозволяє легко масштабувати систему при додаванні нових функцій.

src/state/ — каталог для управління глобальним станом додатку. У проєкті використовується Redux Toolkit, що дозволяє централізовано зберігати дані про користувачів, кошики, транзакції, повідомлення та екологічні рейтинги. Тут розміщено slice-файли, middleware, конфігурацію store та інтеграцію з RTK Query для роботи з API.

src/assets/ — директорія для зберігання статичних ресурсів: зображень, іконок, шрифтів, стилів. Вона дозволяє централізовано керувати візуальними елементами платформи та забезпечує консистентність дизайну.

src/utils/ — набір допоміжних функцій, які використовуються в різних частинах додатку: форматування дат, обробка чисел, генерація унікальних ідентифікаторів, локалізація текстів. Це дозволяє уникнути дублювання коду та підвищити його читабельність.

src/hooks/ — власні React-хуки, які реалізують повторювану логіку: обробку форм, синхронізацію з локальним сховищем, управління таймерами, обробку подій. Такий підхід дозволяє інкапсулювати поведінку та використовувати її в різних компонентах без втрати гнучкості.

src/routes/ — конфігурація маршрутизації, яка визначає доступні сторінки, їх компоненти, рівні доступу та логіку переходів. Використовується бібліотека React Router v6, що дозволяє реалізувати захищені маршрути, вкладені переходи та динамічні параметри URL.

Під час розробки клієнтської частини E-Market було інтегровано низку бібліотек, які забезпечують стабільну роботу та розширюють функціональність:

ReactDOM — для рендерингу компонентів у DOM-структуру браузера.

Redux Toolkit — для управління станом додатку та інтеграції з API.

RTK Query — для ефективної роботи з серверними запитами, кешування та обробки помилок [9].

React Router — для маршрутизації між сторінками.

Formik + Yup — для створення форм з валідацією.

Chart.js + react-chartjs-2 — для побудови графіків і діаграм.

Tailwind CSS + Emotion — для стилізації компонентів.

```
import ImageAnimation from './layout/loader/ImageAnimation';
import MyLottieAnimation from './layout/loader/MyLottieAnimation';
import { SocketProvider } from 'context/SocketContext';

const RegisterPage = lazy(() => import('../pages/registration/RegisterPage'));
const LoginPage = lazy(() => import('../pages/registration/LoginPage'));
const FinancialPage = lazy(() => import('../pages/finance/FinancialPage'));
const MainPage = lazy(() => import('../pages/main/MainPage'));
const DashBoardPage = lazy(() => import('../pages/main/DashBoardPage'));
const CalendarPage = lazy(() => import('../pages/main/CalendarPage'));
const AddBagPage = lazy(() => import('../pages/main/AddBagPage'));
const StatisticsPage = lazy(() => import('../pages/statistics/StatisticsPage'));
const ServicesPage = lazy(() => import('../pages/services/ServicesPage'));
const SettingsPage = lazy(() => import('../pages/settings/SettingsPage'));
const NotFoundPage = lazy(() => import('../pages/registration/NotFoundPage'));
const VerifyEmail = lazy(() => import('../pages/registration/VerifyPage'));
const PasswordRecoveryPage = lazy(() => import('../pages/registration/PasswordRecoveryPage'));
const RenewPasswordPage = lazy(() => import('../pages/registration/RenewPasswordPage'));
const ViewBagPage = lazy(() => import('../pages/main/ViewBagPage'));
const ArchivePage = lazy(() => import('../pages/main/ArchivePage'));
const ShopSettingsPage = lazy(() => import('../pages/settings/ShopSettingsPage'));
const ShopEditSettingsPage = lazy(() => import('../pages/settings/ShopEditSettingsPage'));
const ShopAddSettingsPage = lazy(() => import('../pages/settings/ShopAddSettingsPage'));
const MessagePage = lazy(() => import('../pages/settings/MessagesPage'));
const ConfidentialityPage = lazy(() => import('../pages/settings/ConfidentialityPage'));
const GeneralStatisticsPage = lazy(() => import('../pages/statistics/GeneralStatisticsPage'));
const ShopStatisticsPage = lazy(() => import('../pages/statistics/ShopStatisticsPage'));
const ReviewStatisticsPage = lazy(() => import('../pages/statistics/ReviewStatisticsPage'));
```

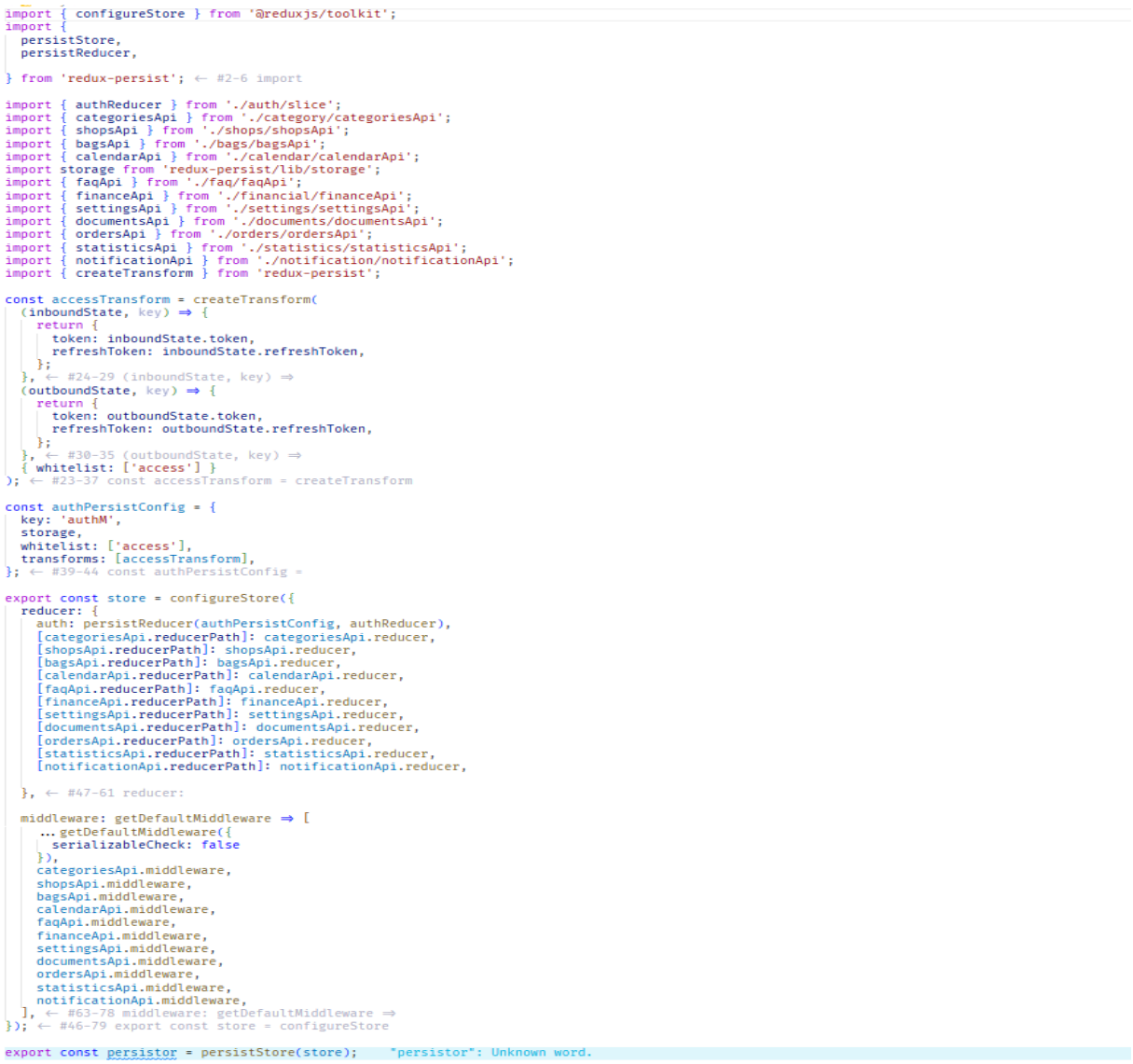
Рисунок 4.2 – Схема маршрутизації сторінок у клієнтській частині платформи E-Market

React Router — це спеціалізована бібліотека, яка відповідає за навігацію між різними розділами в React-додатках. Вона дозволяє задавати маршрути та пов’язувати їх із відповідними компонентами, що забезпечує динамічне відображення контенту залежно від URL-адреси. Завдяки цьому інструменту розробники можуть створювати односторінкові застосунки з багатосторінковою логікою, де перемикання між сторінками відбувається без перезавантаження. React Router підтримує передачу параметрів у маршрутах, вкладену структуру переходів та контроль історії переміщень користувача [див. рис. 4.2.].

Redux — це потужний інструмент для централізованого управління станом у React-застосунках, який широко використовується в сучасній фронтенд-розробці. Його концепція базується на принципах архітектури Flux, що передбачає чіткий потік даних та уніфіковану логіку оновлення. Усі дані про поточний стан додатку зберігаються в одному глобальному сховищі — так званому store, що дозволяє забезпечити єдине джерело істини для всіх компонентів.

Компоненти інтерфейсу можуть звертатися до цього сховища для отримання актуального стану, а також надсилати спеціальні запити — дії (actions), які ініціюють зміну стану через функції-редуктори. Такий підхід дозволяє чітко контролювати, як і коли змінюються дані, що значно спрощує процес розробки, тестування та підтримки додатку.

Завдяки передбачуваності оновлень, Redux забезпечує стабільну поведінку інтерфейсу, дозволяє легко відстежувати зміни та швидко знаходити джерело помилок. Це особливо важливо для великих проєктів, де взаємодіють десятки або сотні компонентів, і потрібна чітка координація між ними [див. рис. 4.3].



```
import { configureStore } from '@reduxjs/toolkit';
import {
  persistStore,
  persistReducer,

} from 'redux-persist'; ← #2-6 import

import { authReducer } from './auth/slice';
import { categoriesApi } from './category/categoriesApi';
import { shopsApi } from './shops/shopsApi';
import { bagsApi } from './bags/bagsApi';
import { calendarApi } from './calendar/calendarApi';
import storage from 'redux-persist/lib/storage';
import { faqApi } from './faq/faqApi';
import { financeApi } from './financial/financeApi';
import { settingsApi } from './settings/settingsApi';
import { documentsApi } from './documents/documentsApi';
import { ordersApi } from './orders/ordersApi';
import { statisticsApi } from './statistics/statisticsApi';
import { notificationApi } from './notification/notificationApi';
import { createTransform } from 'redux-persist';

const accessTransform = createTransform(
  (inboundState, key) => {
    return {
      token: inboundState.token,
      refreshToken: inboundState.refreshToken,
    };
  }, ← #24-29 (inboundState, key) =>
  (outboundState, key) => {
    return {
      token: outboundState.token,
      refreshToken: outboundState.refreshToken,
    };
  }, ← #30-35 (outboundState, key) =>
  { whitelist: ['access'] }
); ← #23-37 const accessTransform = createTransform

const authPersistConfig = {
  key: 'authM',
  storage,
  whitelist: ['access'],
  transforms: [accessTransform],
}; ← #39-44 const authPersistConfig =

export const store = configureStore({
  reducer: {
    auth: persistReducer(authPersistConfig, authReducer),
    [categoriesApi.reducerPath]: categoriesApi.reducer,
    [shopsApi.reducerPath]: shopsApi.reducer,
    [bagsApi.reducerPath]: bagsApi.reducer,
    [calendarApi.reducerPath]: calendarApi.reducer,
    [faqApi.reducerPath]: faqApi.reducer,
    [financeApi.reducerPath]: financeApi.reducer,
    [settingsApi.reducerPath]: settingsApi.reducer,
    [documentsApi.reducerPath]: documentsApi.reducer,
    [ordersApi.reducerPath]: ordersApi.reducer,
    [statisticsApi.reducerPath]: statisticsApi.reducer,
    [notificationApi.reducerPath]: notificationApi.reducer,

  }, ← #47-61 reducer:

  middleware: getDefaultMiddleware => [
    ...getDefaultMiddleware({
      serializableCheck: false
    }),
    categoriesApi.middleware,
    shopsApi.middleware,
    bagsApi.middleware,
    calendarApi.middleware,
    faqApi.middleware,
    financeApi.middleware,
    settingsApi.middleware,
    documentsApi.middleware,
    ordersApi.middleware,
    statisticsApi.middleware,
    notificationApi.middleware,
  ], ← #63-78 middleware: getDefaultMiddleware =>
}); ← #46-79 export const store = configureStore

export const persistor = persistStore(store);    "persistor": Unknown word.
```

Рисунок 4.3 – Схема централізованого управління станом додатку за допомогою Redux

Файл package.json відіграє ключову роль у проєктах, що базуються на Node.js, зокрема у React-застосунках. Він містить описову інформацію про сам проєкт, перелік необхідних бібліотек та параметри конфігурації, які визначають його поведінку та структуру [див. рис. 4.4].

Рисунок 4.4 – Структура конфігураційного файлу package.json у проєкті E-Market

Styled Components — це сучасна бібліотека, яка забезпечує зручний спосіб стилізації React-компонентів за допомогою концепції CSS-in-JS. Завдяки цьому

підходу стилі інтегруються безпосередньо в код компонента, що дозволяє відмовитися від традиційного використання окремих CSS-файлів. Така інтеграція сприяє кращій структурованості коду, підвищує його читабельність і спрощує підтримку.

Рисунок 4.5 – Візуалізація стилізації компонентів у середовищі Styled Components

Однією з ключових особливостей Styled Components є автоматичне створення унікальних класів для кожного компонента, що забезпечує ізоляцію стилів. Це дозволяє уникнути конфліктів між CSS-правилами, які часто виникають у великих проєктах з багатьма модулями. Крім того, стилі, пов’язані з конкретним компонентом, залишаються локальними, що підвищує стабільність інтерфейсу.

Бібліотека підтримує можливість створення динамічних стилів, які змінюються залежно від стану компонента або переданих параметрів. Це відкриває широкі можливості для побудови адаптивного інтерфейсу, який реагує на дії користувача, зміну теми або контексту. Наприклад, можна змінювати кольори,

розміри чи анімації елементів залежно від умов, що робить інтерфейс більш гнучким і інтерактивним.

Styled Components також легко інтегрується з системами темізації, що дозволяє централізовано керувати стилями всього додатку. Завдяки цьому можна швидко змінювати зовнішній вигляд інтерфейсу, адаптувати його до різних брендів або режимів (світлий/темний), не змінюючи логіку компонентів.

Окрім базової стилізації, бібліотека надає інструменти для створення анімацій, що дозволяє реалізовувати плавні переходи, ефекти появи, зміни стану тощо. Це робить Styled Components ефективним засобом для розробки сучасних, естетично привабливих і динамічних веб-застосунків на основі React [див. рис. 4.5].

Серверна частина екологічної платформи E-Market є критично важливою складовою, яка забезпечує обробку запитів, управління даними, реалізацію бізнес-логіки та взаємодію з клієнтською частиною. Вона побудована на основі PHP-фреймворку Yii2 [11], що дозволяє створити надійне, масштабоване та безпечне середовище для роботи з великою кількістю користувачів і транзакцій[див. рис. 4.6].

Однією з головних функцій бекенду є обробка запитів від клієнтської частини, реалізованої на React. Через RESTful API фронтенд надсилає запити на отримання кошиків, бронювання, реєстрацію, авторизацію, перегляд історії покупок тощо. Серверна частина обробляє ці запити, звертається до бази даних, виконує необхідні обчислення та повертає відповідь у форматі JSON. Такий підхід дозволяє забезпечити швидку та стабільну взаємодію між компонентами системи.

Усі запити проходять через контролери, які реалізують логіку обробки. Наприклад, контролер кошиків відповідає за створення, оновлення, видалення та видачу кошиків. Він перевіряє права доступу, валідність даних, статус продуктів, терміни придатності, екологічну оцінку. Це дозволяє уникнути помилок, дублювання та забезпечити цілісність даних.

Для збереження інформації використовується реляційна база даних, яка містить таблиці «Кошики», «Продукти», «Користувачі», «Транзакції», «Пункти видачі», «Екологічні показники», «Історія активності». Кожна таблиця має чітко

визначену структуру, зв’язки та індекси, що дозволяє ефективно здійснювати запити, фільтрацію, агрегацію та аналіз.

Рисунок 4.6 – Автоматизована обробка замовлень і нагадувань у контролері ScheduleController (Yii2)

У бекенді реалізовано механізм авторизації через JWT-токени, що дозволяє захистити сесії користувачів. При вході користувач отримує токен, який надсилається з кожним запитом. Сервер перевіряє його дійсність, термін дії, роль користувача та дозволяє або блокує доступ до відповідних ресурсів[див. рис. 4.7].

Важливою складовою серверної логіки є управління екологічними параметрами. Кожен продукт має екологічну оцінку, яка враховує вплив утилізації на довкілля. Ці дані зберігаються в окремій таблиці та використовуються для

формування кошиків з пріоритетом реалізації. Наприклад, продукти з високим екологічним ризиком утилізації отримують статус «термінові» і пропонуються покупцям у першу чергу.

```php
<?php

namespace app\controllers;

use app\filters\HttpBearerAuth;
use app\models\Shops;
use yii\data\ActiveDataProvider;
use yii\filters\AccessControl;
use yii\web\Response;

0 references | 0 implementations
class AccountController extends BaseController
{

    0 references | 0 overrides
    public function verbs(): array
    {
        return [
            'shops' => ['GET'],
            'info' => ['GET'],
            'download-merchant-bill' => ['GET'],
            'export' => ['POST'],
        ];
    }

    26 references | 0 overrides | prototype
    public function behaviors(): array
    {
        return array_merge(arrays: parent::behaviors(), [
            'bearerAuth' => [
                'class' => HttpBearerAuth::class,    Use of unknown class: 'app\filters\HttpBearerAuth'
                'except' => [],
            ],
            'access' => [
                'class' => AccessControl::class,    Use of unknown class: 'yii\filters\AccessControl'
                'rules' => [
                    [
                        'actions' => ['shops', 'download-merchant-bill', 'request-callback'],
                        'allow' => true,
                        'roles' => ['merchant'],
                    ],
                    [
                        'actions' => ['info', 'export'],
                        'allow' => true,
                        'roles' => ['merchant', 'client', 'admin'],
                    ],
                ],
            ],
        ]);
    }

    /**
     * @SWG\Get (path="/account/shops",
     *     tags={"[Merchant]Account"},
     *     summary="List merchant shops",
     *     @SWG\Parameter(
     *         name="page",
     *         in="query",
     *         type="integer",
     *         description="Page number"
     *     ),
     *     @SWG\Response(
     *         response = 201,
     *         description = "Returned list",
     *         @SWG\Schema (ref="#/definitions/Shops")
     *     ),
     * )
     */
    0 references | 0 overrides
    public function actionShops(): ActiveDataProvider
    {
        $dp = new ActiveDataProvider([    Use of unknown class: 'yii\data\ActiveDataProvider'
            'query' => Shops::find()->where(['merchant_id' => identity()->current_merchant_id])    Call t
        ]);
        $dp->sort->defaultOrder = ['id' => SORT_DESC];
        return $dp;
    }
```

Рисунок 4.7 – Механізм авторизації у контролері AccountController (Yii2)

Система також реалізує механізм планування часу видачі. Продавець може вказати доступні часові слоти, а покупець — обрати зручний час. Сервер перевіряє

доступність, кількість бронювань, навантаження на пункт видачі та формує графік. Це дозволяє уникнути черг, перевантаження та забезпечити комфортну взаємодію.

Окрему увагу приділено обробці транзакцій. Кожна покупка фіксується у таблиці «Транзакції», де зберігається ID кошика, ID покупця, сума, дата, спосіб оплати, статус. Серверна частина забезпечує перевірку балансу, обробку платежів, формування квитанцій, оновлення статусу кошика. У перспективі можливе підключення платіжних систем через API[див. рис. 4.8].

```
0 references | 0 implementations
class BillingPlansController extends BaseController
{

    0 references | 0 overrides
    public function verbs(): array
    {
        return [
            'index' ⇒ ['GET'],
            'create' ⇒ ['POST'],
            'update' ⇒ ['POST'],
            'delete' ⇒ ['POST'],
        ]; ← #17-22 return
    } ← #16-23 public function verbs()

    26 references | 0 overrides | prototype
    public function behaviors(): array
    {
        return array_merge(arrays: parent::behaviors(), [
            'bearerAuth' ⇒ [
                'class' ⇒ HttpBearerAuth::class,    Use of unknown class: 'app\filters\HttpBearerAuth'
                'except' ⇒ [],
            ],
            'access' ⇒ [
                'class' ⇒ AccessControl::class,    Use of unknown class: 'yii\filters\AccessControl'
                'rules' ⇒ [
                    [
                        'actions' ⇒ ['index', 'create', 'update', 'delete'],
                        'allow' ⇒ true,
                        'roles' ⇒ ['admin'],
                    ], ← #35-39 [ 'actions' ⇒ ['index', 'create', 'update', 'delete'], 'allo…
                ], ← #34-40 'rules' ⇒
            ], ← #32-41 'access' ⇒
        ]); ← #27-42 return array_merge
    } ← #26-43 public function behaviors()


    /**
     * @SWG\Get(path="/billing-plans",
     *     tags={"[Admin]Billing plans"},
     *     summary="List billing plans",
     *     @SWG\Parameter(
     *         name="page",
     *         in="query",
     *         type="integer",
     *         description="Page number"
     *     ),
     *     @SWG\Response(
     *         response = 200,
     *         description = "Returned list",
     *         @SWG\Schema (ref="#/definitions/BillingPlans")
     *     ),
     * )
     */
    0 references | 0 overrides
```

Рисунок 4.8 – Створення та обробка замовлення у контролері OrdersController (Yii2, Mobile API)

Для адміністраторів реалізовано окремий модуль, який дозволяє переглядати статистику, формувати звіти, керувати користувачами, змінювати налаштування. Серверна частина забезпечує доступ до агрегованих даних, таких як кількість реалізованих кошиків, середній час реалізації, екологічна вигода, активність

користувачів. Це дозволяє приймати управлінські рішення на основі реальних показників[див. рис. 4.9].

```php
<?php

namespace app\controllers;

use app\filters\HttpBearerAuth;
use app\models\Feedbacks;
use app\models\intermediate\FavouritesStatistics;    "Favourites": Unknown word.
use app\models\intermediate\ViewsStatistics;
use app\models\Orders;
use DateTime;
use Exception;
use yii\data\ActiveDataProvider;
use yii\filters\AccessControl;

0 references | 0 implementations
class StatisticsController extends BaseController
{

    0 references | 0 overrides
    public function verbs(): array
    {
        return [
            'general' => ['GET'],
        ];
    }

    26 references | 0 overrides | prototype
    public function behaviors(): array
    {
        return array_merge(arrays: parent::behaviors(), [
            'bearerAuth' => [
                'class' => HttpBearerAuth::class,    Use of unknown class: 'app\filters\HttpBearerAuth'
                'except' => [],
            ],
            'access' => [
                'class' => AccessControl::class,    Use of unknown class: 'yii\filters\AccessControl'
                'rules' => [
                    [
                        'actions' => ['general', 'rating', 'cancelled', 'reverted', 'feedbacks', 'comments'],
                        'allow' => true,
                        'roles' => ['merchant'],
                    ],
                ],
            ],
        ]);
    }

    /**
     * @SWG\Get(path="/statistics/general?period={period}",
     *     tags={"[Merchant]Statistics"},
     *     summary="General merchant statistics",
     *     @SWG\Parameter(
     *         name="period",
     *         in="query",
     *         type="string",
     *         description="Period. Not required. Must be like '30days', '12weeks', '12months' etc"
     *     ),
     *     @SWG\Response(
     *         response = 200,
     *         description = "Returns statistics",
     *         @SWG\Schema(ref = "#/definitions/GeneralStatistics")
     *     ),
     * )
     */
    0 references | 0 overrides
```

Рисунок 4.9 – Обробка статистики замовлень у контролері StatisticsController (Yii2, API)

Ще одним важливим напрямом є реалізація системи повідомлень. Серверна частина надсилає сповіщення про нові кошики, зміну статусу, нагадування про видачу, підтвердження бронювання. Повідомлення можуть надходити через email, push-сповіщення або внутрішню систему. Це дозволяє підтримувати постійний зв’язок з користувачами та підвищити рівень залученості[див. рис. 4.10].

Рисунок 4.10 – Обробка повідомлень у контролері MailController (Yii2, API)

Усі дії користувачів фіксуються у таблиці «Історія активності», що дозволяє аналізувати поведінку, формувати персоналізовані рекомендації, виявляти закономірності. Серверна частина обробляє ці дані, формує профілі, адаптує інтерфейс, оптимізує пропозиції.

Для забезпечення стабільності реалізовано механізми кешування, логування, обробки помилок, резервного копіювання. Сервер автоматично зберігає критичні дані, фіксує винятки, надсилає повідомлення про збої. Це дозволяє швидко реагувати на проблеми, забезпечити безперервну роботу та захист інформації.

Завдяки використанню Yii2, серверна частина платформи E-Market є гнучкою, надійною та готовою до масштабування. Вона забезпечує ефективну взаємодію з клієнтською частиною, підтримує екологічну логіку, обробляє великі обсяги даних та дозволяє розробникам зосередитися на створенні корисного функціоналу.

У процесі розробки екологічної платформи E-Market одним із фундаментальних завдань стало створення власного модуля автентифікації користувачів. Цей компонент є критично важливим для забезпечення безпеки, персоналізації та контролю доступу до функціональних розділів системи. Його реалізовано з використанням технологій React та Redux, що дозволило досягти високої гнучкості, масштабованості та інтеграції з серверною частиною, побудованою на фреймворку Yii2.

Модуль автентифікації виконує низку ключових функцій, які охоплюють повний цикл взаємодії користувача з системою:

Авторизація — користувач вводить email та пароль, які перевіряються на сервері. У разі успішної перевірки система генерує токен доступу, що дозволяє здійснювати захищені запити до API.

Реєстрація — нові користувачі мають змогу створити обліковий запис, заповнивши форму з контактними даними. Серверна частина перевіряє унікальність email, валідність паролю та зберігає дані у базі.

Відновлення паролю — у випадку втрати доступу користувач може ініціювати процедуру відновлення, отримавши одноразове посилання на email. Це посилання веде до форми зміни паролю, яка після підтвердження оновлює облікові дані.

Інтерфейс модуля побудовано на основі компонентної архітектури React. Кожен режим — авторизація, реєстрація, відновлення — реалізовано як окремий компонент, що дозволяє легко перемикатися між ними. Redux використовується для централізованого управління станом: зберігання токена, статусу автентифікації,

повідомлень про помилки, завантаження тощо. Це забезпечує передбачувану поведінку інтерфейсу та спрощує тестування.

Особливу увагу приділено UX-дизайну: форми мають адаптивну верстку, валідацію на клієнтському рівні, підказки для користувача, а також систему сповіщень, яка інформує про успішні або помилкові дії. Наприклад, при неправильному паролі з'являється повідомлення з поясненням, а при успішному вході — підтвердження з перенаправленням до особистого кабінету.

З технічної точки зору, модуль автентифікації взаємодіє з бекендом через захищені RESTful API-запити. Усі запити передаються через HTTPS, а дані — у форматі JSON. Після авторизації користувач отримує JWT-токен, який зберігається у локальному сховищі браузера та використовується для автентифікації наступних запитів. Сервер перевіряє дійсність токена, його термін дії та роль користувача, що дозволяє реалізувати рольову модель доступу.

Модуль також включає інформаційний блок користувача, який відображається після входу. У ньому міститься ім'я користувача, email, кнопка виходу з системи та, за потреби, доступ до додаткових функцій (наприклад, перегляд історії замовлень або редагування профілю). Завершення сесії здійснюється шляхом очищення токена та перенаправлення на головну сторінку.

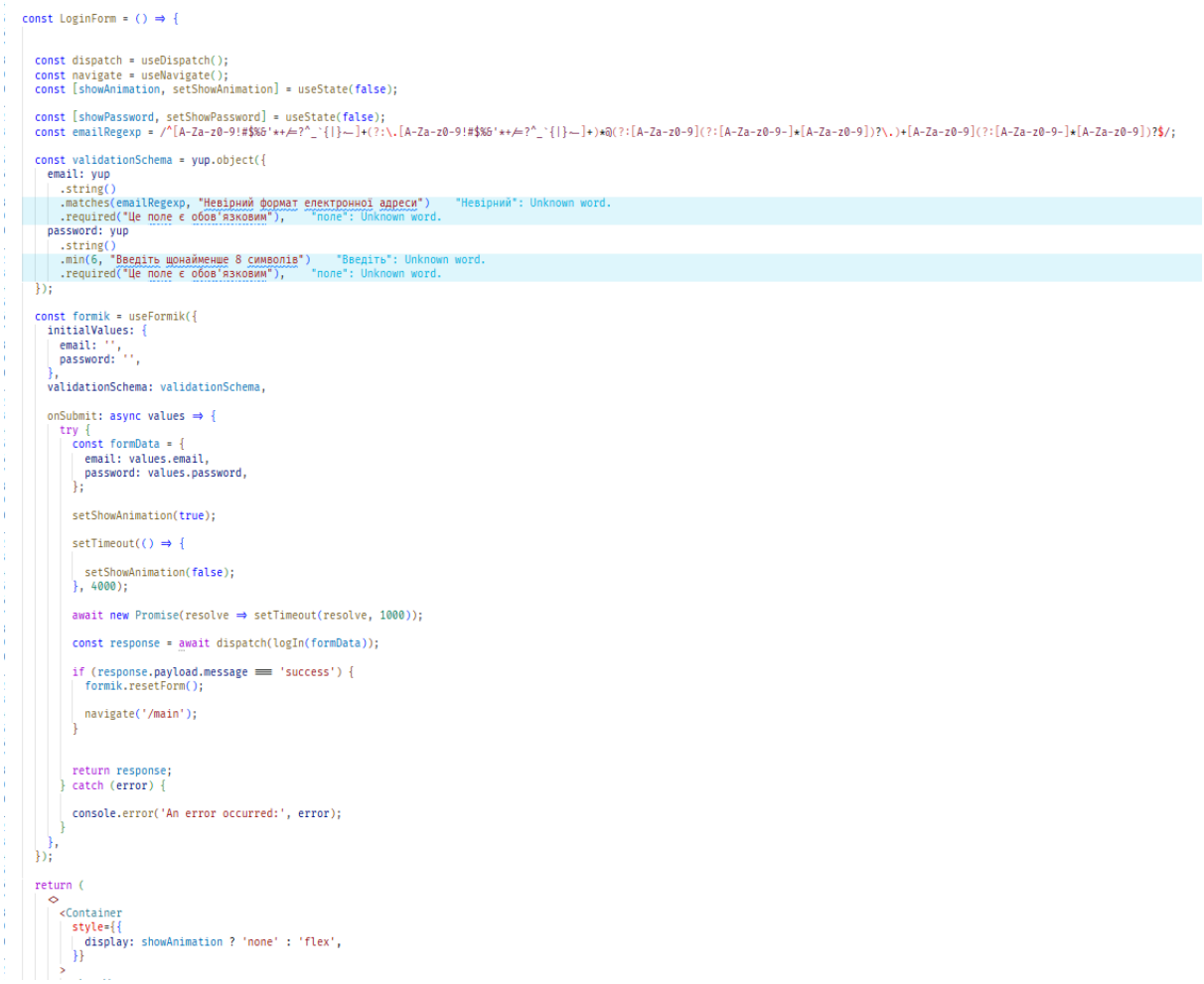
Завдяки високій адаптивності, модуль автентифікації може бути інтегрований у будь-який розділ платформи — від головної сторінки до адміністративної панелі. Його архітектура дозволяє легко розширювати функціональність, наприклад, додавати двофакторну автентифікацію, соціальні входи або інтеграцію з зовнішніми сервісами.

З точки зору безпеки, реалізовано низку заходів:

- Хешування паролів на сервері
- Захист від CSRF-атак через токени.
- Валідація даних на клієнтському та серверному рівнях.
- Обмеження кількості спроб входу для запобігання brute-force атакам.
- Шифрування токенів та контроль їх терміну дії.

Таким чином, модуль автентифікації E-Market є не лише технічно досконалим,

а й зручним для користувача. Він забезпечує надійний контроль доступу, захист персональних даних, зручну навігацію між режимами та інтеграцію з іншими частинами платформи. Його реалізація відповідає сучасним стандартам веб-розробки та може бути використана як основа для подальшого розширення функціональності системи [див. рис. 4.11, 4.12, 4.13].



```
const LoginForm = () => {


  const dispatch = useDispatch();
  const navigate = useNavigate();
  const [showAnimation, setShowAnimation] = useState(false);

  const [showPassword, setShowPassword] = useState(false);
  const emailRegexp = /^[A-Za-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[A-Za-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[A-Za-z0-9](?:[A-Za-z0-9-]*[A-Za-z0-9])?\.)+[A-Za-z0-9](?:[A-Za-z0-9-]*[A-Za-z0-9])?$/;

  const validationSchema = yup.object({
    email: yup
      .string()
      .matches(emailRegexp, "Невірний формат електронної адреси")    "Невірний": Unknown word.
      .required("Це поле є обов'язковим"),    "поле": Unknown word.
    password: yup
      .string()
      .min(6, "Введіть щонайменше 8 символів")    "Введіть": Unknown word.
      .required("Це поле є обов'язковим"),    "поле": Unknown word.
  });

  const formik = useFormik({
    initialValues: {
      email: '',
      password: '',
    },
    validationSchema: validationSchema,

    onSubmit: async values => {
      try {
        const formData = {
          email: values.email,
          password: values.password,
        };

        setShowAnimation(true);

        setTimeout(() => {

          setShowAnimation(false);
        }, 4000);

        await new Promise(resolve => setTimeout(resolve, 1000));

        const response = await dispatch(logIn(formData));

        if (response.payload.message === 'success') {
          formik.resetForm();

          navigate('/main');
        }


        return response;
      } catch (error) {

        console.error('An error occurred:', error);
      }
    },
  });

  return (
    <>
      <Container
        style={{
          display: showAnimation ? 'none' : 'flex',
        }}
      >
```

Рисунок 4.11 – Компонент форми для логінізації користувачів

```jsx
const RegisterForm = () => {

  const dispatch = useDispatch();
  const navigate = useNavigate();
  const [loading, setLoading] = useState(false);
  const [showPassword, setShowPassword] = useState(false);
  const [showAnimation, setShowAnimation] = useState(false);
  const [activeStep, setActiveStep] = useState(1);
  const [isChecked, setIsChecked] = useState(false);
  const [centrMap, setCentrMap] = useState({    "centr": Unknown word.
    lat: 50.4501,
    lng: 30.5234,
  });
  const emailRegexp = /^[A-Za-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[A-Za-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[A-Za-z0-9](?:[A-Za-z0-9-]*[A-Za-z0-9])?\.)+[A-Za-z0-9](?:[A-Za-z0-9-]*[A-Za-z0-9])?$/;
  const handleCheckboxChange = (e) => {
    setIsChecked(e.target.checked);
  };
  const validationSchema = yup.object({
    email: yup.string().matches(emailRegexp, "Невірний формат електронної адреси").required("Це поле є обов'язковим"),    "Невірний": Unknown word.
    password: yup.string().min(8, "Введіть щонайменше 8 символів").required("Це поле є обов'язковим"),    "Введіть": Unknown word.
    merchantName: yup.string().required("Це поле є обов'язковим"),    "поле": Unknown word.
    phone: yup.string().required("Це поле є обов'язковим"),    "поле": Unknown word.
    address: yup.string().required("Це поле є обов'язковим"),    "поле": Unknown word.
  });
  const [isModalOpen, setModalOpen] = useState(false);
  const [selectItem, setSelectItem] = useState({});
  const [selectFilter, setSelectFilter] = useState('');
  const [selectFilter2, setSelectFilter2] = useState('');
  const { data: viewData, isLoading:isloadSuc} = useGetDocumentQuery();    "isload": Unknown word.

  const formik = useFormik({
    initialValues: {
      email: '',
      password: '',

      phone: '',
      address: '',
      role: 'merchant',
      latitude: '40.712776',
      longitude: '54.005974',
      merchantName: ''
    },
    validationSchema: validationSchema,
    onSubmit: async (values) => {
      try {
        const formData = {
          role: values.role,
          email: values.email,
          password: values.password,
          merchantName: values.merchantName,
          phone: values.phone,
          address: values.address,
          latitude: values.latitude,
          longitude: values.longitude,
        };
        setShowAnimation(true);

        setTimeout(() => {
          setShowAnimation(false);
        }, 3000);

        const response = await dispatch(register(formData));
        if (response.payload.message === 'success') {
          formik.resetForm();
          navigate('/main');
        } else {
          console.error('Registration failed:', response.payload);
        }
      } catch (error) {
        console.error('An error occurred:', error);
      }
    },
  });

  const nextStep = async () => {
    try {
      const errors = await formik.validateForm();

      // Якщо є помилки в email або паролі, не переходимо до наступного кроку    "Якщо": Unknown word.
      if (errors.email || errors.password) {
        formik.setTouched({
```

Рисунок 4.12 – Компонент форми для реєстрації користувачів

```
const RenewPassword = () => {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const location = useLocation();
  const [showAnimation, setShowAnimation] = useState(false);
  const [isSubmitted, setIsSubmitted] = useState(false);
  const [showPassword, setShowPassword] = useState(false);

  const queryParams = new URLSearchParams(location.search);
  const code = queryParams.get('code');
  const id = queryParams.get('id');

  const validationSchema = yup.object({
    password: yup
      .string()
      .min(6, "Введіть щонайменше 6 символів")    "Введіть": Unknown word.
      .required("Це поле є обов'язковим"),    "поле": Unknown word.
  }); ← #52-57 const validationSchema = yup.object

  const formik = useFormik({
    initialValues: {
      password: '',
    },
    validationSchema: validationSchema,

    onSubmit: async (values) => {
      try {
        const formData = {
          id: id,
          code: code,
          password: values.password,
        }; ← #67-71 const formData =

        setShowAnimation(true);

        await dispatch(renewPassword(formData)).unwrap();

        setShowAnimation(false);
        setIsSubmitted(true);
      } catch (error) {
        console.error('An error occurred:', error);
        setShowAnimation(false);
      }
    }, ← #65-83 onSubmit: async (values) =>
  }); ← #59-84 const formik = useFormik

  return (
    <>
      <Container>
        <LogoWrap>
          <Logo src={logo} alt="Logo" />
        </LogoWrap>
        <StyledForm onSubmit={formik.handleSubmit}>
          <InputGroupe isSubmitted={isSubmitted}>
            {!isSubmitted ? null : <><Close
              onClick={() => navigate('/')}
              style={{
                cursor: 'pointer',
                alignSelf: 'flex-end'
              }}
            />
              <Img src={successImage} alt="Success" /></>} ← #94-101 <InputGroupe isSubmitted={isSubmitted}>
            <WrapTitle>
              <FormName>

                {!isSubmitted ? <><ArrowLeft
                  onClick={() => navigate('/')}
                  style={{
                    cursor: 'pointer',
                    position: 'absolute',
                    left: '0',
                    justifySelf: 'flex-start',
                  }} ← #107-112 style=
                />Змінити пароль </> : 'Пароль успішно змінено'}    "Змінити": Unknown word. ← #105-113 {!isSubmitted ? <><ArrowLeft
              </FormName>

              {!isSubmitted
                ? <SubTitle>Створіть новий пароль, на заміну забутого</SubTitle>    "Створіть": Unknown word.
                : null}
            </WrapTitle>
            <InputList>
              {!isSubmitted ? (
```

Рисунок 4.13 – Компонент форми для відновлення паролю

Для реалізації механізмів автентифікації користувачів та керування збереженням їхніх даних у клієнтській частині платформи E-Market було використано бібліотеку Redux Toolkit — сучасний інструмент для організації та контролю стану у React-застосунках, побудованих на основі архітектури Redux.

Redux Toolkit забезпечує централізоване управління станом, що дозволяє зберігати ключову інформацію про користувача, його сесію, статус авторизації та інші параметри в єдиному сховищі. Це значно спрощує доступ до даних з будь-якого компонента інтерфейсу, забезпечуючи узгодженість і передбачуваність поведінки додатку.

У складі Redux Toolkit інтегровано низку інструментів для обробки асинхронних процесів, зокрема Redux Thunk, які використовуються для взаємодії з бекендом платформи, побудованим на Yii2. Це дозволяє ефективно виконувати запити на авторизацію, реєстрацію, відновлення паролю, а також обробляти відповіді сервера, включаючи повідомлення про помилки, підтвердження дій та оновлення стану.

Серед основних переваг використання Redux Toolkit у межах E-Market можна виокремити:

Оптимізований синтаксис та зручна структура: бібліотека дозволяє швидко створювати ред'юсери, дії та селектори, скорочуючи обсяг шаблонного коду та підвищуючи продуктивність розробки.

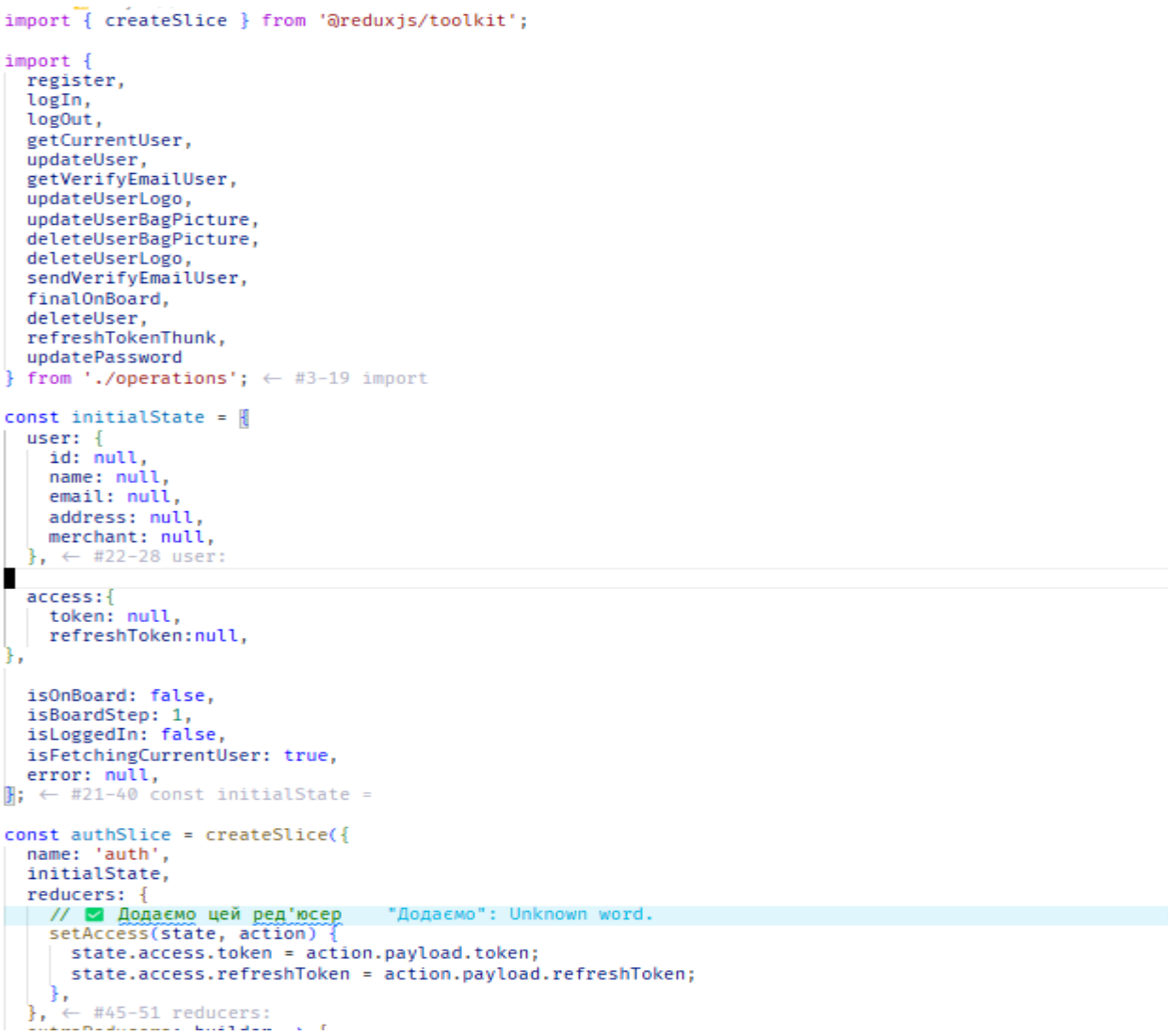
Асинхронна логіка: підтримка Redux Thunk забезпечує обробку запитів до API, включаючи автентифікацію, перевірку токенів, оновлення профілю та інші дії, що потребують взаємодії з сервером.

Інструменти для налагодження: інтеграція з Redux DevTools дозволяє відстежувати зміни стану в реальному часі, аналізувати потік дій та виявляти помилки на ранніх етапах.

Модульна організація: код поділено на окремі логічні блоки, кожен з яких відповідає за конкретну частину функціональності — автентифікацію, профіль, повідомлення, сесію. Це спрощує підтримку, розширення та командну роботу над проєктом.

У контексті модуля автентифікації Redux Toolkit відіграє роль зв’язуючого елемента між інтерфейсом користувача та серверною логікою. Він дозволяє зберігати статус авторизації, обробляти відповіді API, керувати формами входу, реєстрації та відновлення паролю. Усі ці процеси реалізовано з урахуванням вимог безпеки, збереження конфіденційності та стабільності роботи системи.

Застосування Redux Toolkit у платформі E-Market сприяє підвищенню масштабованості клієнтської частини, забезпечує гнучке управління станом та дозволяє ефективно інтегрувати модуль автентифікації в загальну архітектуру додатку. Це рішення є технічно обґрунтованим і відповідає сучасним стандартам розробки веб-застосунків [див. рис. 4.14–4.20].



```
import { createSlice } from '@reduxjs/toolkit';

import {
  register,
  logIn,
  logOut,
  getCurrentUser,
  updateUser,
  getVerifyEmailUser,
  updateUserLogo,
  updateUserBagPicture,
  deleteUserBagPicture,
  deleteUserLogo,
  sendVerifyEmailUser,
  finalOnBoard,
  deleteUser,
  refreshTokenThunk,
  updatePassword
} from './operations'; ← #3-19 import

const initialState = {
  user: {
    id: null,
    name: null,
    email: null,
    address: null,
    merchant: null,
  }, ← #22-28 user:

  access:{
    token: null,
    refreshToken:null,
},

  isOnBoard: false,
  isBoardStep: 1,
  isLoggedIn: false,
  isFetchingCurrentUser: true,
  error: null,
}; ← #21-40 const initialState =

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    // ✅ Додаємо цей ред'юсер    "Додаємо": Unknown word.
    setAccess(state, action) {
      state.access.token = action.payload.token;
      state.access.refreshToken = action.payload.refreshToken;
    },
  }, ← #45-51 reducers:
```

Рисунок 4.14 – Компонент slice.

```
import { configureStore } from '@reduxjs/toolkit';
import {
  persistStore,
  persistReducer,

} from 'redux-persist'; ← #2-6 import

import { authReducer } from './auth/slice';
import { categoriesApi } from './category/categoriesApi';
import { shopsApi } from './shops/shopsApi';
import { bagsApi } from './bags/bagsApi';
import { calendarApi } from './calendar/calendarApi';
import storage from 'redux-persist/lib/storage';
import { faqApi } from './faq/faqApi';
import { financeApi } from './financial/financeApi';
import { settingsApi } from './settings/settingsApi';
import { documentsApi } from './documents/documentsApi';
import { ordersApi } from './orders/ordersApi';
import { statisticsApi } from './statistics/statisticsApi';
import { notificationApi } from './notification/notificationApi';
import { createTransform } from 'redux-persist';

const accessTransform = createTransform(
  (inboundState, key) => {
    return {
      token: inboundState.token,
      refreshToken: inboundState.refreshToken,
    };
  }, ← #24-29 (inboundState, key) =>
  (outboundState, key) => {
    return {
      token: outboundState.token,
      refreshToken: outboundState.refreshToken,
    };
  }, ← #30-35 (outboundState, key) =>
  { whitelist: ['access'] }
); ← #23-37 const accessTransform = createTransform

const authPersistConfig = {
  key: 'authM',
  storage,
  whitelist: ['access'],
  transforms: [accessTransform],
}; ← #39-44 const authPersistConfig =

export const store = configureStore({
  reducer: {
    auth: persistReducer(authPersistConfig, authReducer),
    [categoriesApi.reducerPath]: categoriesApi.reducer,
    [shopsApi.reducerPath]: shopsApi.reducer,
    [bagsApi.reducerPath]: bagsApi.reducer,
    [calendarApi.reducerPath]: calendarApi.reducer,
    [faqApi.reducerPath]: faqApi.reducer,
    [financeApi.reducerPath]: financeApi.reducer,
    [settingsApi.reducerPath]: settingsApi.reducer,
    [documentsApi.reducerPath]: documentsApi.reducer,
    [ordersApi.reducerPath]: ordersApi.reducer,
    [statisticsApi.reducerPath]: statisticsApi.reducer,
    [notificationApi.reducerPath]: notificationApi.reducer,

  }, ← #47-61 reducer:

  middleware: getDefaultMiddleware => [
    ...getDefaultMiddleware({
      serializableCheck: false
    }),
    categoriesApi.middleware,
    shopsApi.middleware,
    bagsApi.middleware,
    calendarApi.middleware,
    faqApi.middleware,
    financeApi.middleware,
    settingsApi.middleware,
    documentsApi.middleware,
    ordersApi.middleware,
    statisticsApi.middleware,
    notificationApi.middleware,
  ], ← #63-78 middleware: getDefaultMiddleware =>
}); ← #46-79 export const store = configureStore

export const persistor = persistStore(store);    "persistor": Unknown word.

```

Рисунок 4.15 – Компонент store.

```
import { configureStore, getDefaultMiddleware } from '@reduxjs/toolkit';    "reduxjs": Unkno
import {
  persistStore,
  persistReducer,
  FLUSH,
  REHYDRATE,
  PAUSE,
  PERSIST,
  PURGE,
  REGISTER,
} from 'redux-persist'; ← #2-11 import
import { authSlice } from './auth/authSlice';
import { templatesSlice } from './templates/templatesSlice';

import storage from 'redux-persist/lib/storage'; // defaults to localStorage for web

const middleware = [
  ...getDefaultMiddleware({
    serializableCheck: {
      ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
    },
  }), ← #18-22 ...getDefaultMiddleware
]; ← #17-23 const middleware =

const authPersistConfig = {
  key: 'auth',
  storage,
};

const persistedAuthReducer = persistReducer(
  authPersistConfig,
  authSlice.reducer
);

const templatesPersistConfig = {
  key: 'templates',
  storage,
  whitelist: ['templates', 'favTemplates'],
}; ← #35-39 const templatesPersistConfig =

const persistedItemsReducer = persistReducer(
  templatesPersistConfig,
  templatesSlice.reducer
);

export const store = configureStore({
  reducer: {
    auth: persistedAuthReducer,
    templates: persistedItemsReducer,
  },
  middleware,
  devTools: process.env.NODE_ENV === 'development',
}); ← #46-53 export const store = configureStore

export const persistor = persistStore(store);    "persistor": Unknown word.
```

Рисунок 4.16 – Компонент AuthSlice

```
export const register = createAsyncThunk(
  '/register',
  async (formData, thunkAPI) => {
    try {
      const deviceId = await getDeviceId();
      const dataWithDeviceId = { ...formData, deviceId };
      const response = await axios.post('/security/register', dataWithDeviceId);
      setAuthHeader(response.data.access.token);
      localStorage.setItem('accessToken', response.data.access.token); // ⬅ Додай це    "Додай": Unknown word.
      localStorage.setItem('refreshToken', response.data.access.refreshToken);

      return response.data;
    } catch (error) {
      if (error.response) {
        const { message } = error.response.data;

        showErrorToast(message);
      } ← #35-39 if (error.response)
      return thunkAPI.rejectWithValue(error.message);
    } ← #34-41 catch (error)
  } ← #24-42 async (formData, thunkAPI) =>
); ← #22-43 export const register = createAsyncThunk

export const valid = createAsyncThunk(
  '/validOneStep',
  async (formData, thunkAPI) => {
    try {
      const response = await axios.post('/security/validate-first-step', formData);
      return response.data;
    } catch (error) {

      // Перевірка наявності response    "Перевірка": Unknown word.
      if (error.response && error.response.data) {
        const { status, data } = error.response;
        console.log('status :>> ', data.details.email);
        // Перевірка наявності даних в error.response.data    "Перевірка": Unknown word.
        if (status === 422) {
          showErrorToast(data.details.email[0]);
        }

        console.log('error :>> ', error);
        return thunkAPI.rejectWithValue(error.response.data);
      } ← #54-64 if (error.response && error.response.data)
    } ← #51-65 catch (error)
  } ← #47-66 async (formData, thunkAPI) =>
); ← #45-67 export const valid = createAsyncThunk



export const logIn = createAsyncThunk(
  '/login',
  async (credentials, thunkAPI) => {
    try {
      const deviceId = await getDeviceId(); // отримуємо deviceId    "отримуємо": Unknown word.

      // додаємо deviceId у дані для логіну    "додаємо": Unknown word.
      const dataWithDeviceId = { ...credentials, deviceId };


console.log('зайшли:>> ');    "зайшли": Unknown word.
      const response = await axios.post('/security/merchant-login', dataWithDeviceId);

      setAuthHeader(response.data.access.token);
      localStorage.setItem('accessToken', response.data.access.token); // ⬅ Додай це    "Додай": Unknown word.
      localStorage.setItem('refreshToken', response.data.access.refreshToken);



      return response.data;
    } catch (error) {

      if (error.response) {
        const { message } = error.response.data;

          showErrorToast(message);
```

Рисунок 4.17 – Операції аутентифікації та реєстрації

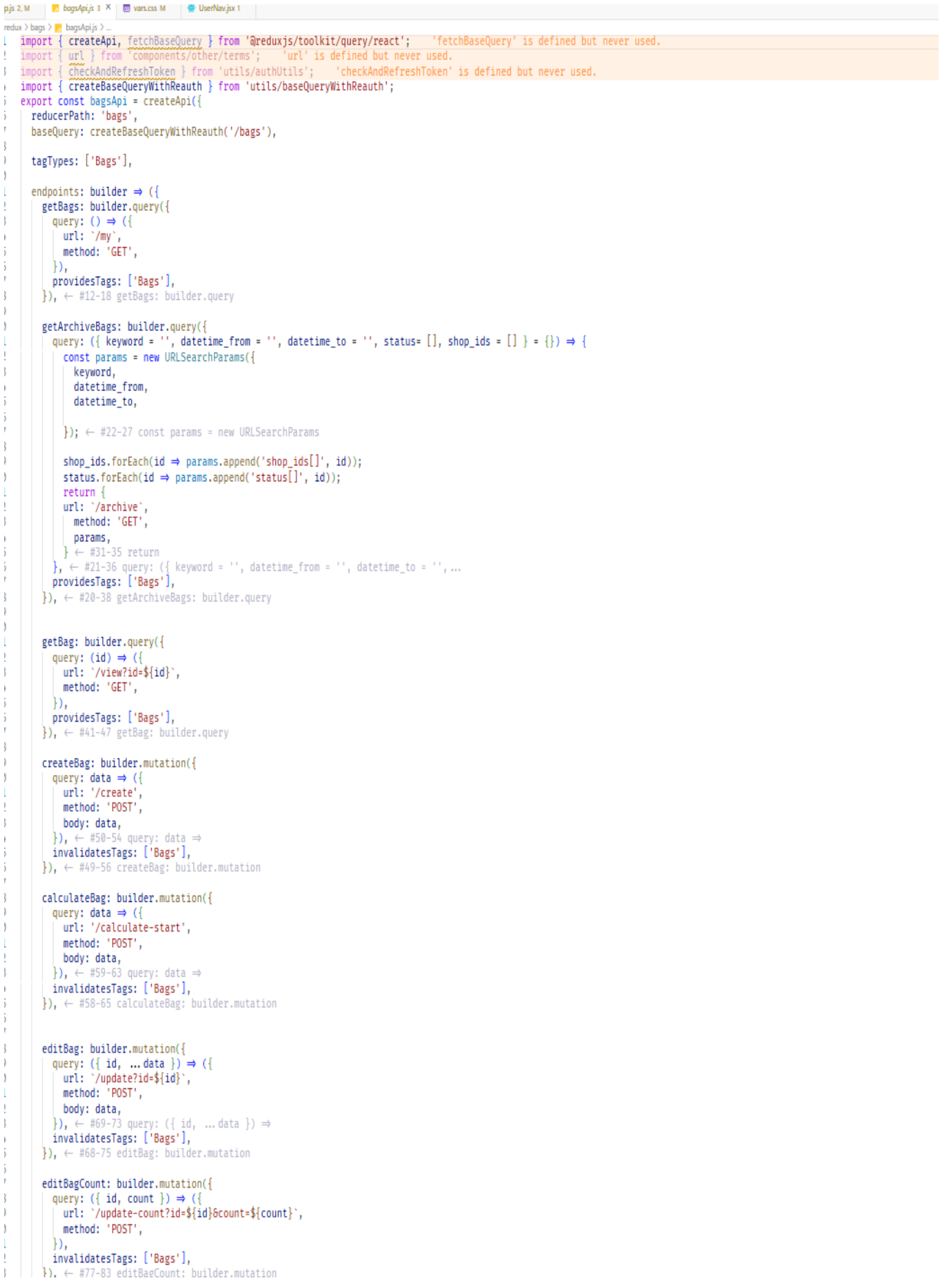


Рисунок 4.18 – Компонент BagsApi Redux Query

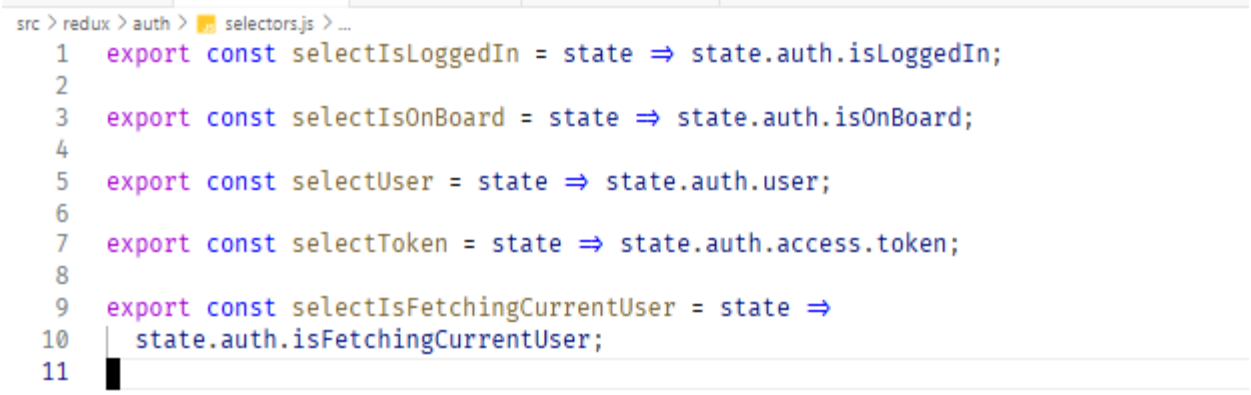


Рисунок 4.19 – Компонент selectors.js

У межах клієнтської частини платформи E-Market реалізовано окремий функціональний компонент AddBag, який відповідає за створення та редагування кошиків — основної одиниці екологічної реалізації продуктів. Цей компонент побудовано з використанням бібліотеки React та інтегровано з RTK Query для взаємодії з бекендом через RESTful API.

Компонент виконує низку завдань, пов’язаних із формуванням параметрів кошика, обробкою даних, отриманих із сервера, та керуванням локальним станом. Його логіка охоплює як ініціалізацію нових кошиків, так і редагування вже існуючих, що дозволяє забезпечити гнучкість і повторне використання інтерфейсу.

Основні функції компонента:

Ініціалізація параметрів: при завантаженні компонента здійснюється перевірка наявності id кошика. Якщо id вказано, виконується запит на отримання даних кошика (useGetBagQuery), які використовуються для заповнення форми редагування. Якщо id відсутній або дорівнює 'id', компонент працює в режимі створення нового кошика.

Обробка асинхронних запитів: компонент використовує RTK Query для отримання категорій (useFetchCategoriesQuery), цін (useFetchPricesQuery), магазинів (useFetchShopsQuery) та обчислення вартості кошика (useCalculateBagMutation). Це дозволяє динамічно формувати інтерфейс на основі актуальних даних з бекенду.

Формування розкладу: за допомогою бібліотеки dayjs з розширенням utc та timezone компонент обчислює дати початку та завершення дії кошика, а також дозволяє налаштувати повторюваність (щоденно, щотижнево тощо), час видачі, умови зупинки (наприклад, після певної кількості повторень або до конкретної дати).

Керування локальним станом: за допомогою useState зберігаються всі параметри кошика — назва, опис, кількість, категорії, дієти, тип пакування, наявність алергенів, ціни, точка видачі, графік доставки, а також можливі помилки введення. Це дозволяє забезпечити повну інтерактивність форми та валідацію даних перед надсиланням.

Редагування існуючих кошиків: якщо компонент працює в режимі редагування, він автоматично заповнює поля на основі отриманого об’єкта bag, включаючи категорії, дієти, ціни, час видачі, тип пакування, графік повторення та інші параметри. Це дозволяє користувачеві швидко оновити кошик без повторного введення даних.

Візуальна адаптація: компонент підтримує динамічне перемикання між режимами створення та редагування, а також між різними днями доставки (сьогодні, завтра, інша дата). Це реалізовано через обчислення дати datetime_from та порівняння її з поточним днем.

Обробка помилок: у структурі компонента передбачено об’єкт arrayErrors, який фіксує стан валідації для кожного поля. Це дозволяє виводити відповідні повідомлення про помилки та запобігати надсиланню некоректних даних.

Архітектурні особливості:

Компонент AddBag є прикладом глибоко інтегрованого рішення, яке поєднує локальну логіку React, асинхронну взаємодію з бекендом через RTK Query, обробку часових зон через dayjs, та управління формами з високим рівнем деталізації. Його структура дозволяє масштабувати функціональність, додавати нові параметри кошика, змінювати логіку повторення або інтегрувати додаткові сервіси (наприклад, аналітику або рекомендації).

Завдяки такому підходу, компонент створення кошика у платформі E-Market забезпечує зручний, гнучкий та надійний інтерфейс для продавців, які формують пропозиції з реалізації продуктів. Він є важливою частиною екологічної логіки платформи, дозволяючи автоматизувати процеси, зменшити втрати та підвищити ефективність взаємодії між учасниками системи.[див. рис. 4.20]

```
const AddBag = () => {
  dayjs.extend(isToday);
  dayjs.extend(isTomorrow);
  const { id } = useParams();
  const location = useLocation();
  const searchParams = new URLSearchParams(location.search);
  const currentDay = searchParams.get('currentDay');
  const mode = searchParams.get('mode');

  const { data: bag, isLoading: isLoadingBag } = useGetBagQuery(id, {    'isLoadingBag' is assigned a value but never use
    skip: id === 'id',  // скіп запиту, якщо id дорівнює 'id'    "скіп": Unknown word.
  });

  dayjs.extend(utc);
  const [calculateBag, { isLoading: isLoading1 }] = useCalculateBagMutation();
  const { data, } = useFetchCategoriesQuery();
  const { data: pricesData, isLoading: isLoadingPrices } = useFetchPricesQuery();    'isLoadingPrices' is assigned a valu
  const { data: shopsData, isLoading: isLoadingShop } = useFetchShopsQuery();    'isLoadingShop' is assigned a value but
  const [createBag] = useCreateBagMutation();
  const [editBag] = useEditBagMutation();
  const navigate = useNavigate();
     const today = dayjs().tz(KYIV_TZ)
  const currentDayOfMonth = today.date();
  const formattedDate = currentDay ? currentDay : today.format('YYYY-MM-DD');
  const nextMonthDate = today.add(1, 'month').format('YYYY-MM-DD');

  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [quantity, setQuantity] = useState(1);
  const [categories, setCategories] = useState([]);
  const [diets, setDiets] = useState([]);
  const [shops, setShops] = useState([]);
  const [selectedDiets, setSelectedDiets] = useState([]);
  const [selectedCategories, setSelectedCategories] = useState([]);
  const [containsAllergens, setContainsAllergens] = useState(false);
  const [buffetFood, setBuffetFood] = useState(false);
  const [packaging, setPackaging] = useState('shop');
  const [firstTime, setFirstTime] = useState(createDateWithTime(0)); // Початковий час    "Початковий": Unknown word.
  const [secondTime, setSecondTime] = useState(createDateWithTime(23, 59));
  const [isAutomation, setIsAutomation] = useState(false);
  const [repeatSchedule, setRepeatSchedule] = useState('daily');
  const [selectedDays, setSelectedDays] = useState([]);
  const [startDate, setStartDate] = useState(formattedDate);
  const [stopCondition, setStopCondition] = useState('never');
  const [stopDate, setStopDate] = useState(nextMonthDate);
  const [stopAfter, setStopAfter] = useState(1);
  const [priceBags, setPriceBags] = useState([]);
  const [point, setPoint] = useState(``);
  const [pointId, setPointId] = useState(``);
  const [selectedFullPrice, setSelectedFullPrice] = useState('0.00грн');
  const [selectedBagPrice, setSelectedBagPrice] = useState('0.00грн');
  const [selectedDeliveryDay, setSelectedDeliveryDay] = useState('today');
  const [selectDaily, setSelectDaily] = useState('1');
  const [dayOfMonth, setDayOfMonth] = useState(currentDayOfMonth);

  const [arrayErrors, setArrayErrors] = useState({
    nameErr: false,
    descErr: false,
    categErr: false,    "categ": Unknown word.
    costErr: false,
    timeErr: false,
```

Рисунок 4.20 – Компонент AddBag для створення кошика

На платформі E-Market реалізовано окремий компонент RegularPanel, який відповідає за налаштування параметрів регулярного повторення. Цей компонент дозволяє продавцям автоматизувати процес формування кошиків, задаючи графік їх появи, періодичність, умови завершення та часові рамки дії. Його інтеграція в інтерфейс створення кошика забезпечує гнучкість, зручність та відповідність екологічній логіці платформи.

Основне призначення компонента

RegularPanel дозволяє активувати режим автоматизації, при якому кошик створюється не одноразово, а за заданим розкладом. Це особливо актуально для продавців, які мають стабільні залишки продукції та бажають регулярно пропонувати її до реалізації без необхідності ручного створення кожного кошика.

Ключові функції компонента

Активація режиму автоматизації Користувач може увімкнути опцію «Зробити кошик регулярним» за допомогою чекбокса. При активації відкривається додатковий блок налаштувань, який дозволяє задати параметри повторення.

Вибір типу повторення Компонент підтримує три основні режими:

Щоденне повторення — з можливістю вибору між «кожного дня» або «через день».

Щотижневе повторення — з вибором конкретних днів тижня, у які кошик має з’являтися.

Щомісячне повторення — з вказанням числа місяця, на яке припадає створення кошика.

Налаштування дати початку За допомогою інтерактивного календаря користувач обирає дату, з якої починається дія регулярного кошика. Компонент підтримує валідацію дати, обмеження на мінімальну дату та відображення у форматі, адаптованому до української локалі.

Визначення умови завершення Продавець може обрати одну з трьох опцій:

Припинити після певної дати — з вибором дати завершення.

Припинити після певної кількості повторень — з вказанням кількості створених кошиків.

Не припиняти — режим без обмежень, при якому кошик створюється безперервно.

Візуальні підказки та інтерактивність Компонент відображає повідомлення про дату першого створення кошика, а також забезпечує динамічне оновлення інтерфейсу залежно від вибраних параметрів. Наприклад, при зміні типу повторення змінюється набір доступних полів, що дозволяє уникнути помилок та підвищити зручність користування.

```
const RegularPanel = ({ setIsAutomation, isAutomation, setArrayErrors, setStopAfter,
  repeatSchedule, setRepeatSchedule, selectDaily,
  setSelectDaily, selectedDays, setSelectedDays,
  dayOfMonth, setDayOfMonth, startDate,
  setStartDate, stopDate, setStopDate,
  stopCondition, setStopCondition, stopAfter, isLoading1, calculatedStart }) => {
  const [isCalendarOpen, s   const setIsCalendar2Open: Dispatch<SetStateAction<boolean>>
  const [isCalendar2Open, setIsCalendar2Open] = useState(false);
  console.log('startDate :>> ', startDate);
  console.log('stopDate :>> ', stopDate);
  const dayClassName1 = (date) => {
    const dateObj = new Date(date);
    const startDateObj = new Date(startDate);
    if (dateObj < subDays(startDateObj, 1)) {
      return "react-datepicker__day--min-date";
    }
    return null;
  }; ← #21-28 const dayClassName1 = (date) =>

  const handleDaysChange = (index) => {
    setSelectedDays((prev) => {
      if (prev.includes(index)) {
        return prev.filter((day) => day !== index);
      } else {
        return [...prev, index];
      }
    }); ← #31-37 setSelectedDays
  }; ← #30-38 const handleDaysChange = (index) =>


  const handleDayOfMonthChange = (e) => {
    let value = parseInt(e.target.value, 10);

    // Перевіряємо, щоб значення було в межах 1 - 31    "Перевіряємо": Unknown word.
    if (value < 1) {
      value = 1;
    } else if (value > 31) {
      value = 31;
    }

    setDayOfMonth(value);
  }; ← #41-52 const handleDayOfMonthChange = (e) =>

  const handleNumberChange = (e) => {
    let value = parseInt(e.target.value, 10);
    if (value < 1) {
      value = 1;
    }
    setStopAfter(value);
    setStopCondition('after');
  }; ← #54-61 const handleNumberChange = (e) =>

  return (
    <>
      <HorizontalBox className="m" style={{ gap: "32px", width: '100%' }}>
        <Label style={{ width: 'calc(100% / 2 )' }}>Автоматизація</Label>    "Автоматизація": Unknown word.
        <HorizontalBox >
          <CheckBoxWrapper>
            <input type="checkbox" id="automation" checked={isAutomation} onChange={() => {
```

Рисунок 4.21 – Компонент RegulaPanel

Архітектурні особливості

Компонент реалізовано на основі React із використанням локального стану (useState) для збереження параметрів повторення, дати початку, умов завершення, вибраних днів тощо. Його структура дозволяє легко масштабувати функціональність, додавати нові режими або інтегрувати з серверною логікою для збереження налаштувань у базі даних.

RegularPanel є частиною загального процесу створення кошика, і його параметри передаються до бекенду через відповідні API-запити. Це дозволяє системі автоматично створювати кошики згідно з заданим графіком, зберігаючи екологічну ефективність та зменшуючи навантаження на продавця.

Значення для екосистеми платформи

Автоматизація створення кошиків є важливою складовою екологічної логіки E-Market. Вона дозволяє систематизувати процес реалізації залишкової продукції, забезпечити стабільну пропозицію для покупців та зменшити ризик утилізації. Компонент RegularPanel сприяє підвищенню ефективності, зручності та екологічної відповідальності учасників платформи[див. рис. 4.22]

Для забезпечення обміну даними в реальному часі між клієнтською частиною платформи E-Market та сервером було реалізовано окремий компонент SocketProvider. Його основне призначення — встановлення та підтримка WebSocket-з’єднання, що дозволяє оперативно реагувати на події, надсилати повідомлення та синхронізувати стан додатку без необхідності періодичного опитування API.

Компонент побудовано на основі бібліотеки socket.io-client [10] та реалізовано з використанням контексту React (createContext), що дозволяє передавати об’єкт сокета до будь-якого дочірнього компонента без потреби у пропсах. Це забезпечує централізоване управління з’єднанням та спрощує інтеграцію WebSocket-функціональності в різні частини інтерфейсу.

Основні функції компонента

Ініціалізація з’єднання При наявності токена авторизації та URL-адреси серверного середовища, компонент створює нове WebSocket-з’єднання, використовуючи порт 443 та шлях /ws. Токен передається як параметр запиту

(auth_token), що дозволяє серверу ідентифікувати користувача та забезпечити захищене з’єднання.

Обробка подій з’єднання Компонент фіксує ключові події:

- connect — підтвердження успішного підключення до сервера.
- disconnect — фіксація розриву з’єднання.
- error — обробка помилок, що виникають під час роботи сокета.

Ці події логуються в консолі, що дозволяє розробникам відстежувати статус з’єднання та реагувати на потенційні збої.

Передача сокета через контекст Об’єкт сокета зберігається у локальному стані (useState) та передається через SocketContext.Provider до всіх дочірніх компонентів. Це дозволяє будь-якому компоненту, який використовує хук useSocket, отримати доступ до активного з’єднання та надсилати або приймати повідомлення.

Очищення ресурсу При зміні токена або URL, а також при демонтажі компонента, з’єднання автоматично закривається (disconnect()), що запобігає витоку ресурсів та забезпечує стабільність роботи додатку.

Архітектурна роль у платформі

2 Компонент SocketProvider є критично важливим для реалізації функціональності в реальному часі, зокрема:

3 Надсилання push-сповіщень про нові кошики, зміну статусу замовлення, підтвердження бронювання тощо.

4 Синхронізація дій між продавцем і покупцем без затримок.

5 Реалізація механізмів живого оновлення інтерфейсу (наприклад, статуси, таймери, повідомлення).

Його інтеграція в архітектуру платформи дозволяє забезпечити високий рівень реактивності, зменшити навантаження на API та покращити користувацький досвід.

Таким чином, компонент SocketProvider є технічно обґрунтованим рішенням для побудови сучасного, динамічного інтерфейсу, який підтримує екологічну логіку E-Market через оперативну взаємодію між учасниками системи[див. рис. 4.23].

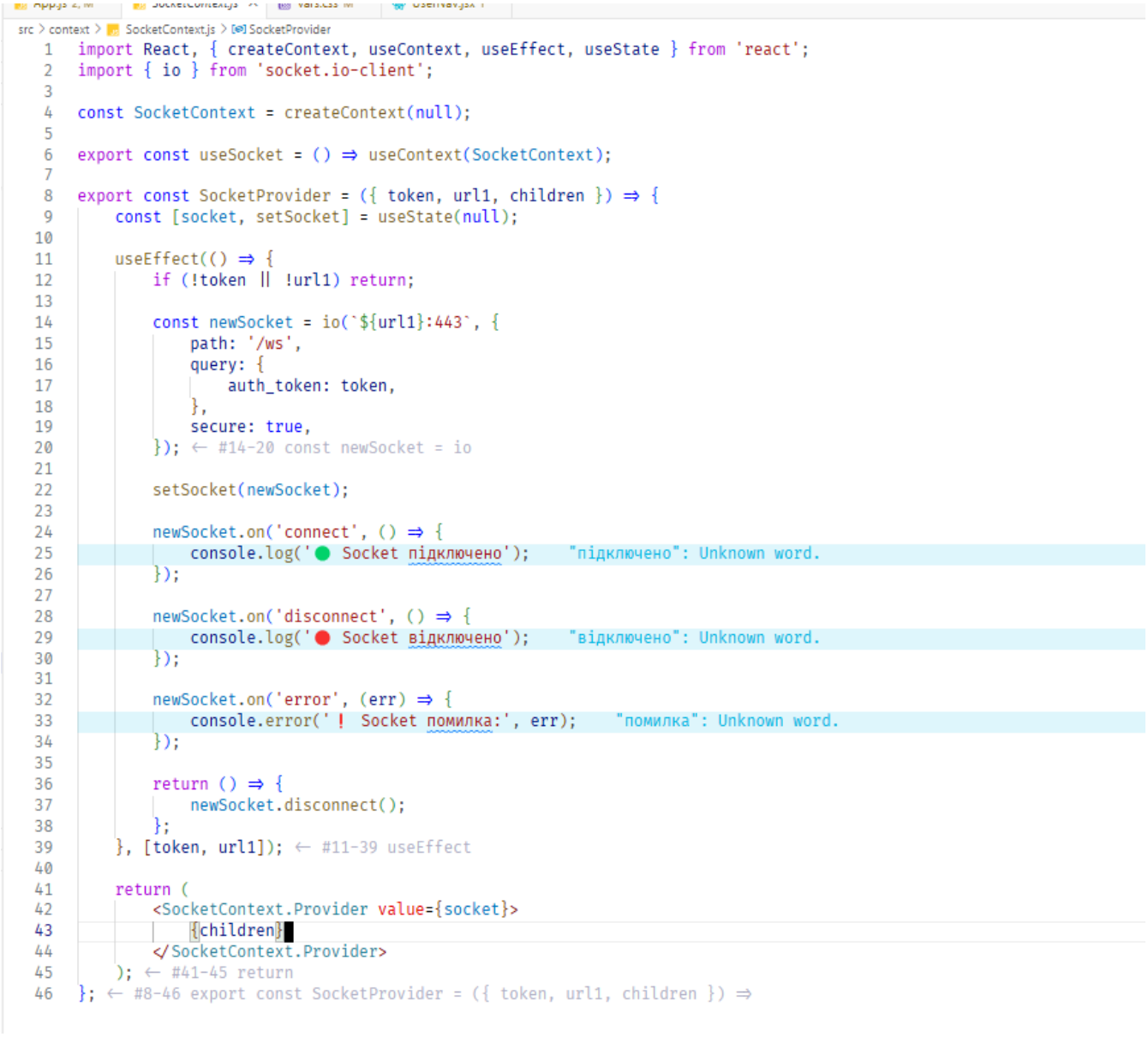


Рисунок 4.22 – Компонент SocketProvider

4.2. Тестування веб-платформи екологічного маркетплейсу для реалізації залишкових страв «E-Market»

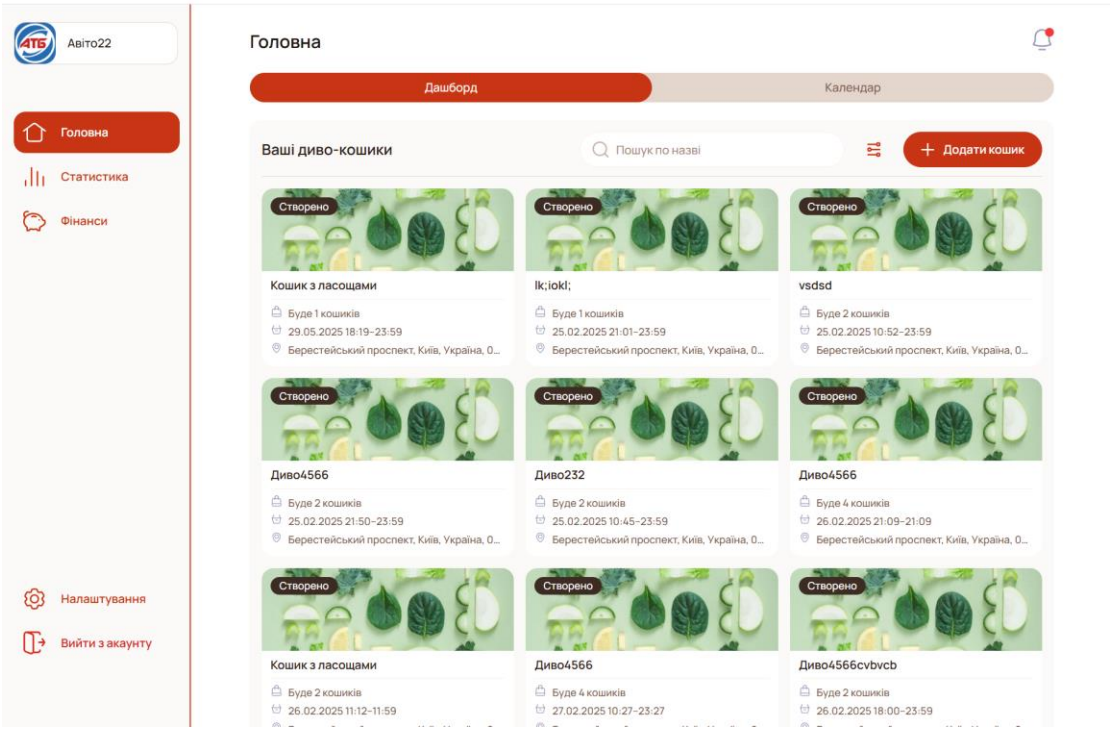


Рисунок 4.23 – Інтерфейс сторінки відображення кошиків веб-платформи E-Market

Рисунок 4.24 – Інтерфейс сторінки авторизації веб-платформи E-Market.

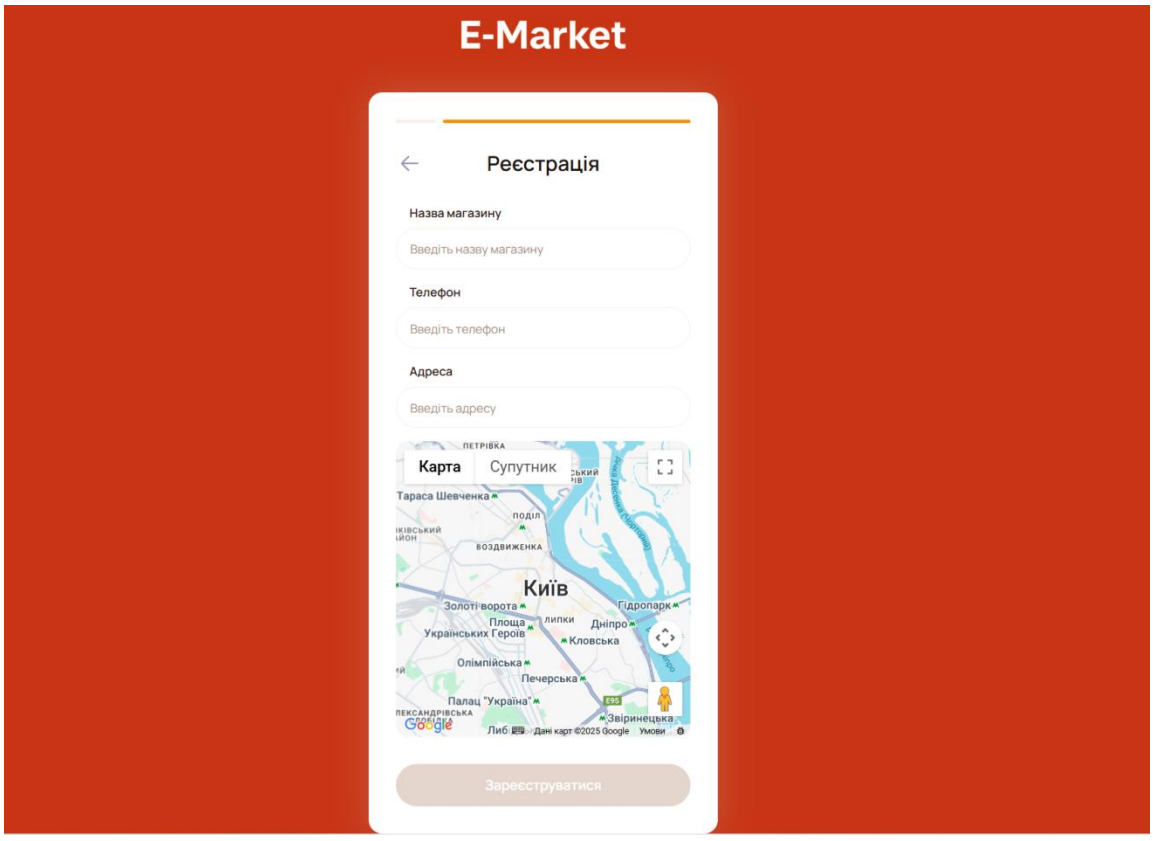
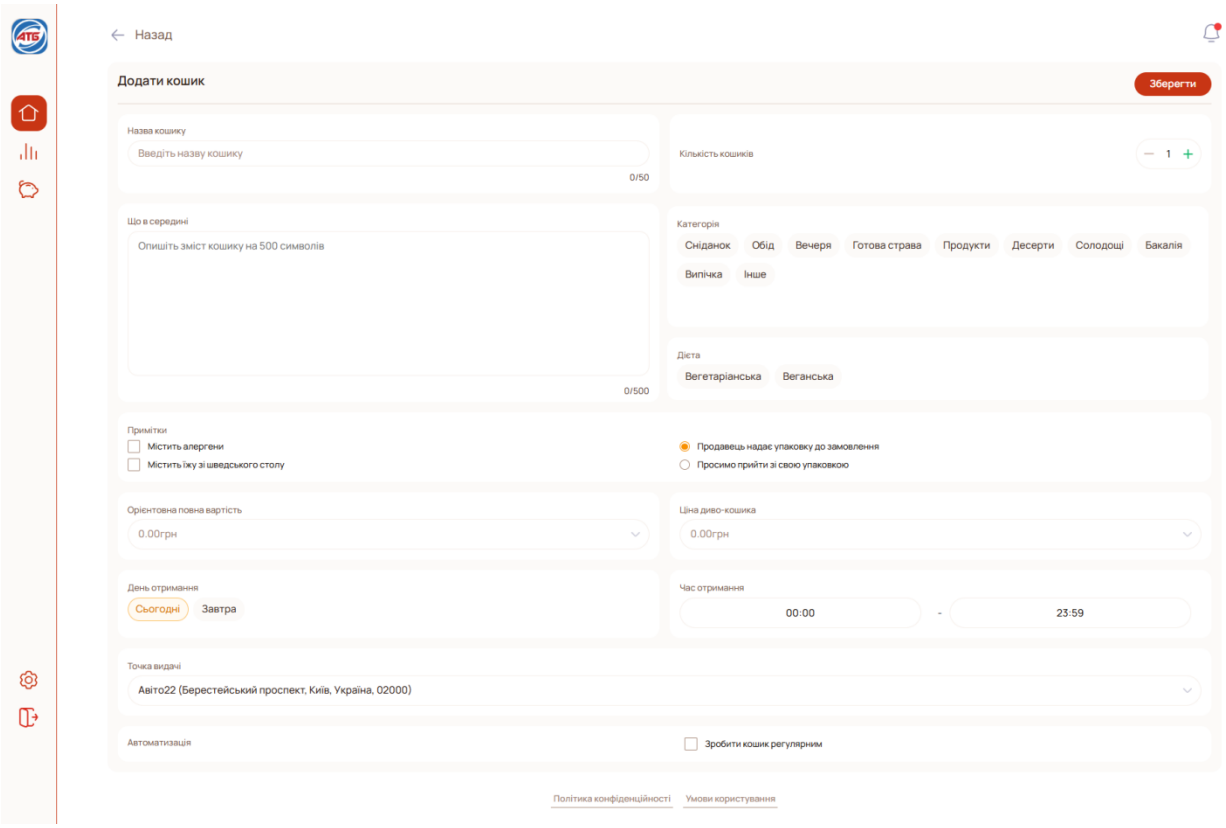


Рисунок 4.25 – Інтерфейс сторінки реєстрації веб-платформи E-Market.

.

Рисунок 4.26 – Система відновлення паролю веб-платформи E-Market.



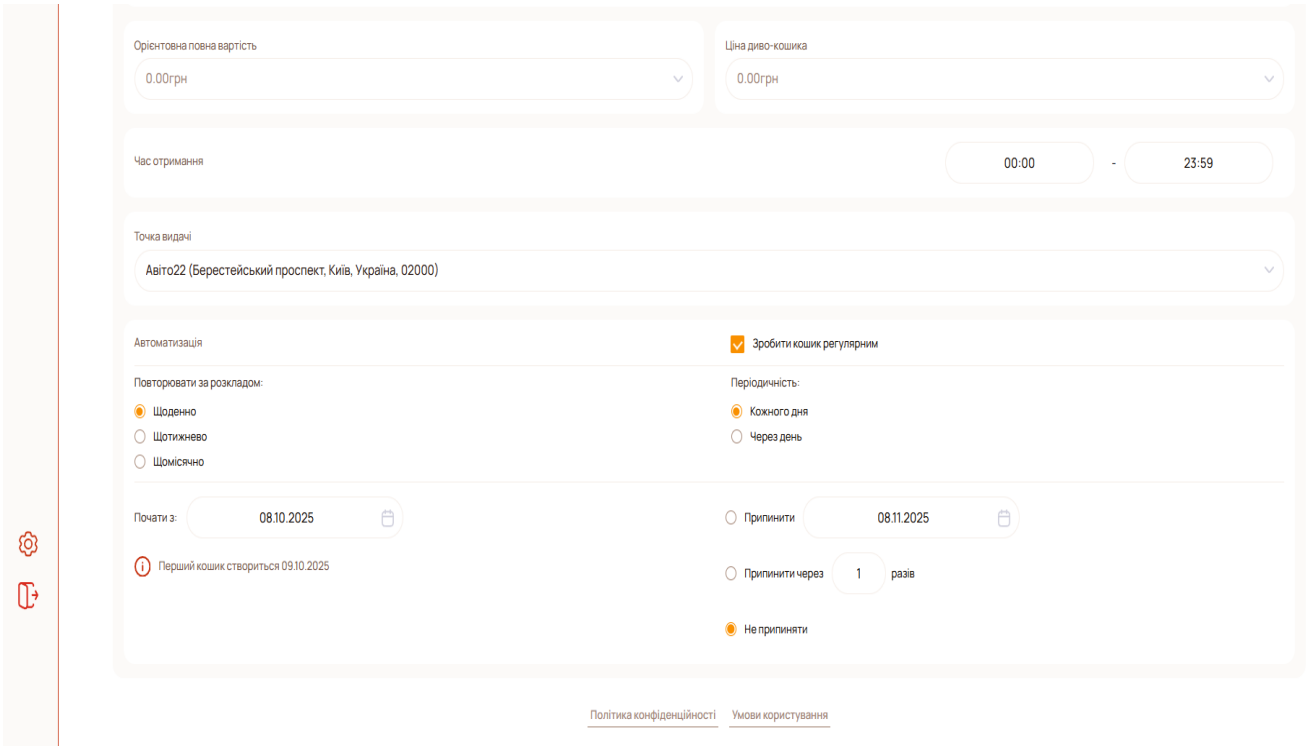


Рисунок 4.27 – Створення нового кошику.

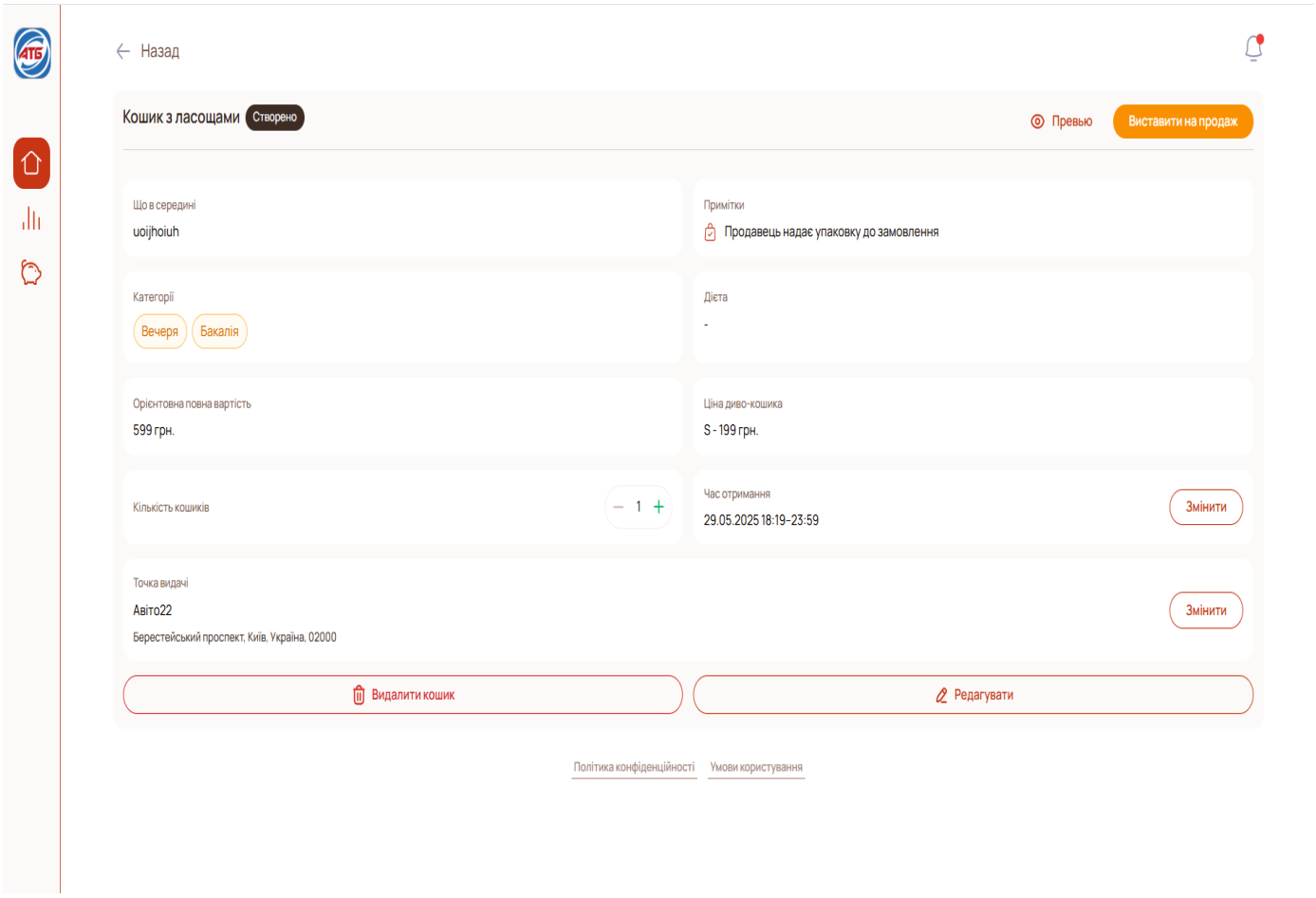


Рисунок 4.28 – Створений Кошик

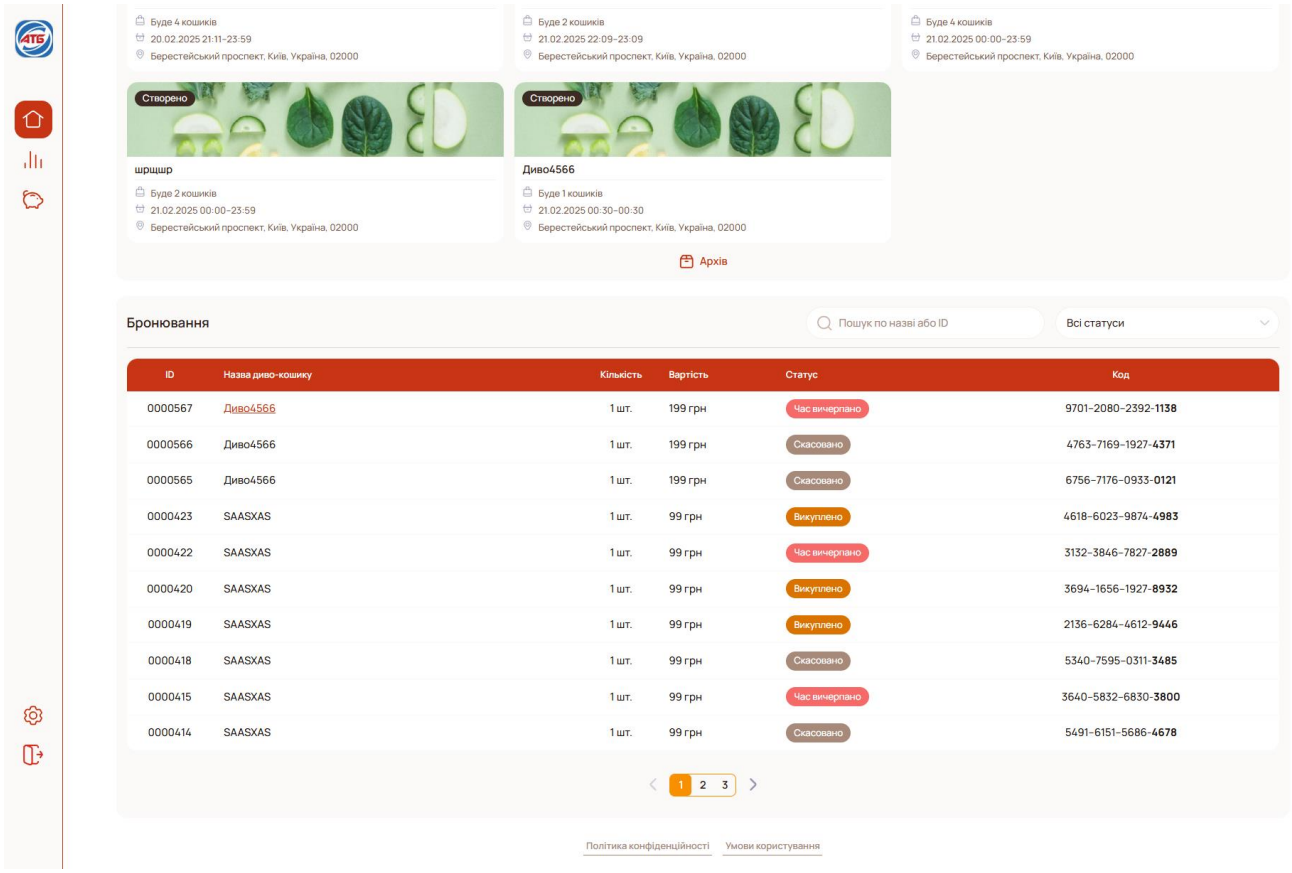


Рисунок 4.29 – Забронювані замовлення

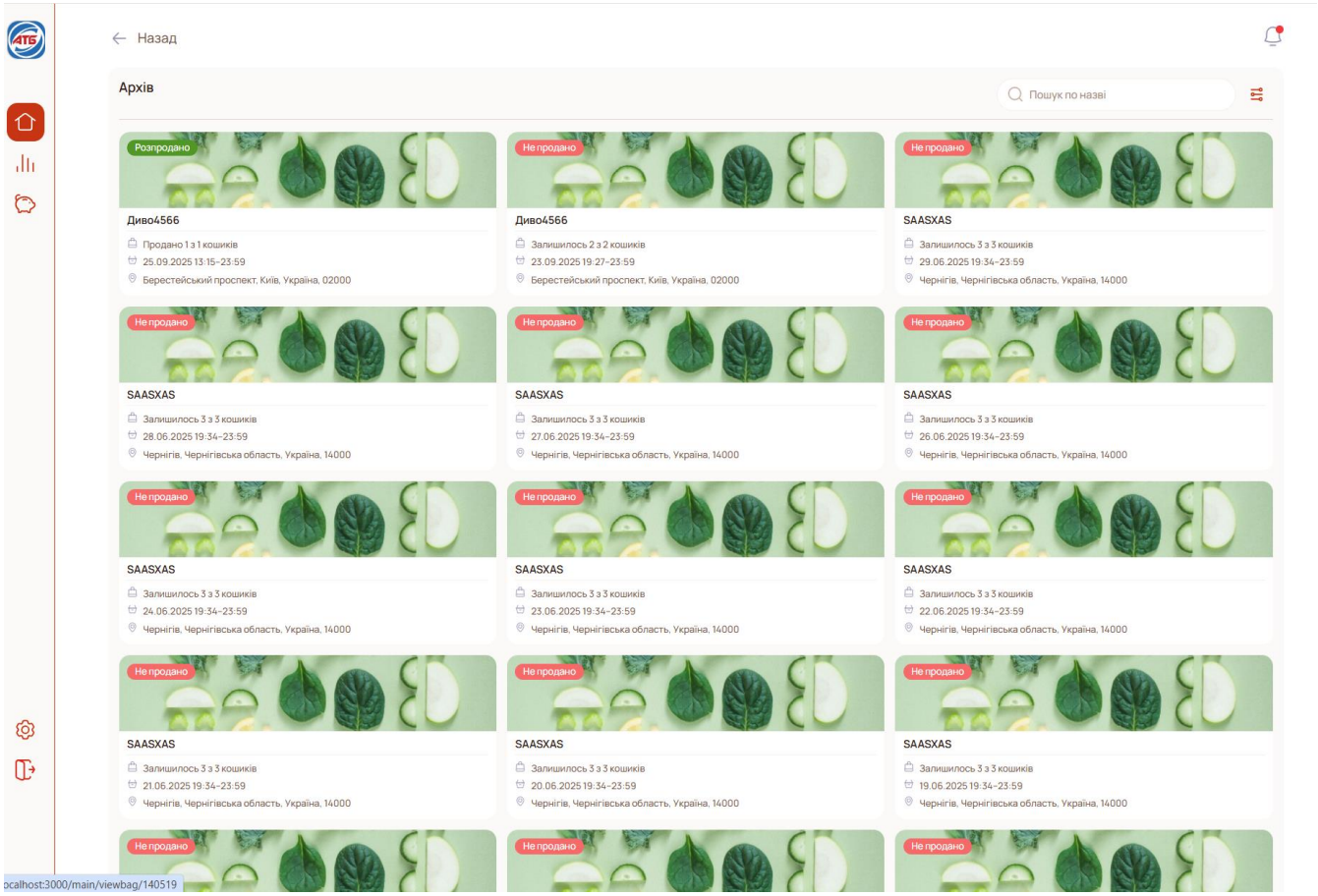


Рисунок 4.30 – Архів проданих кошиків.

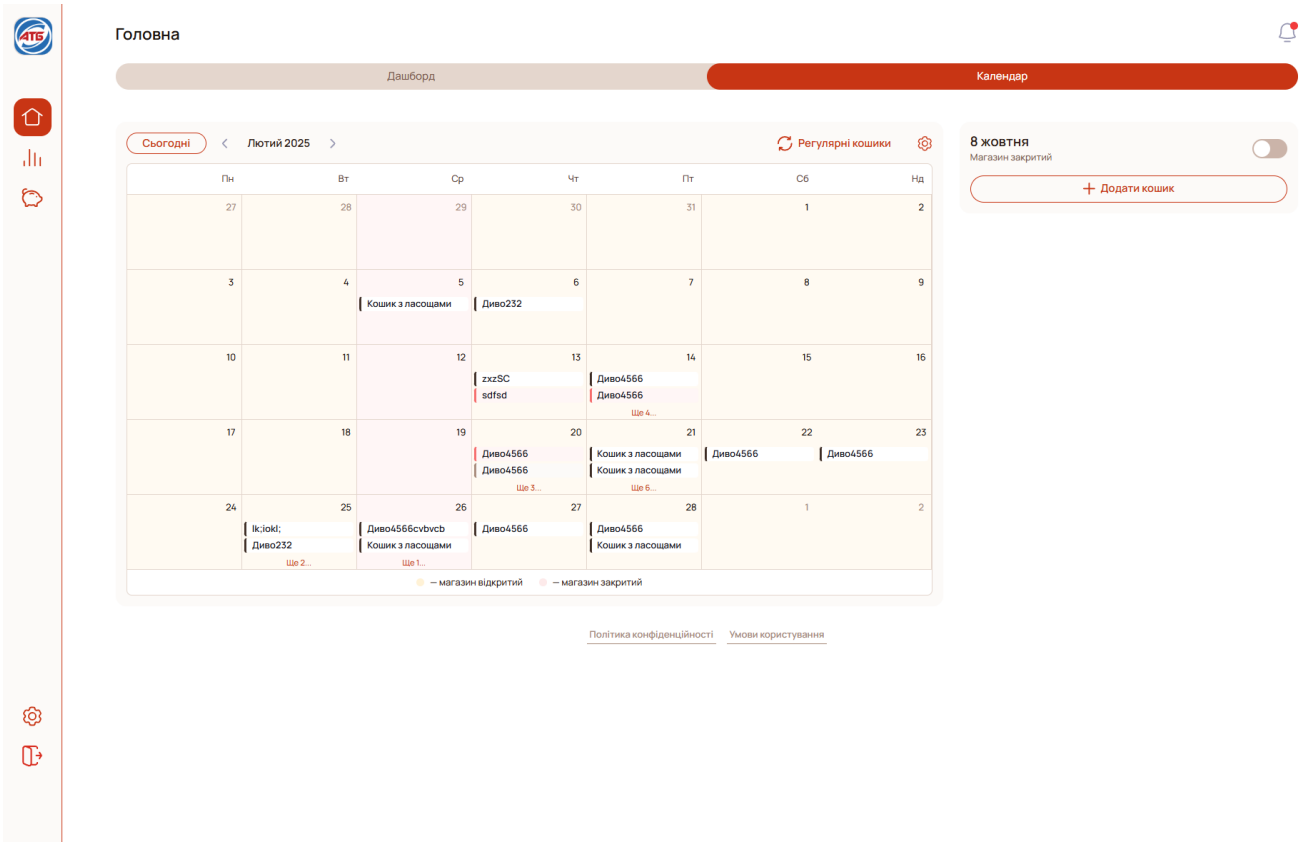
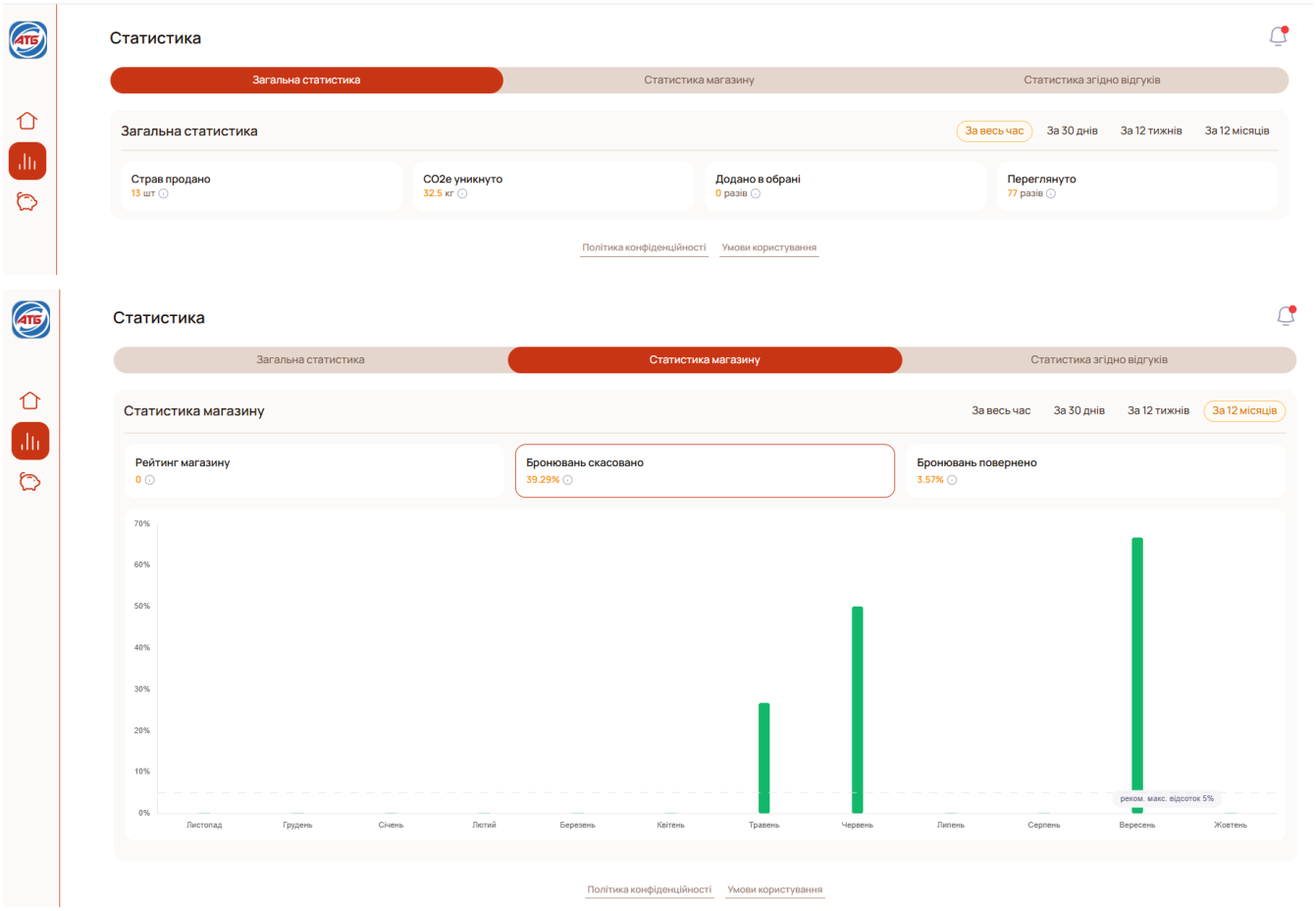


Рисунок 4.31 – Календар створених кошиків



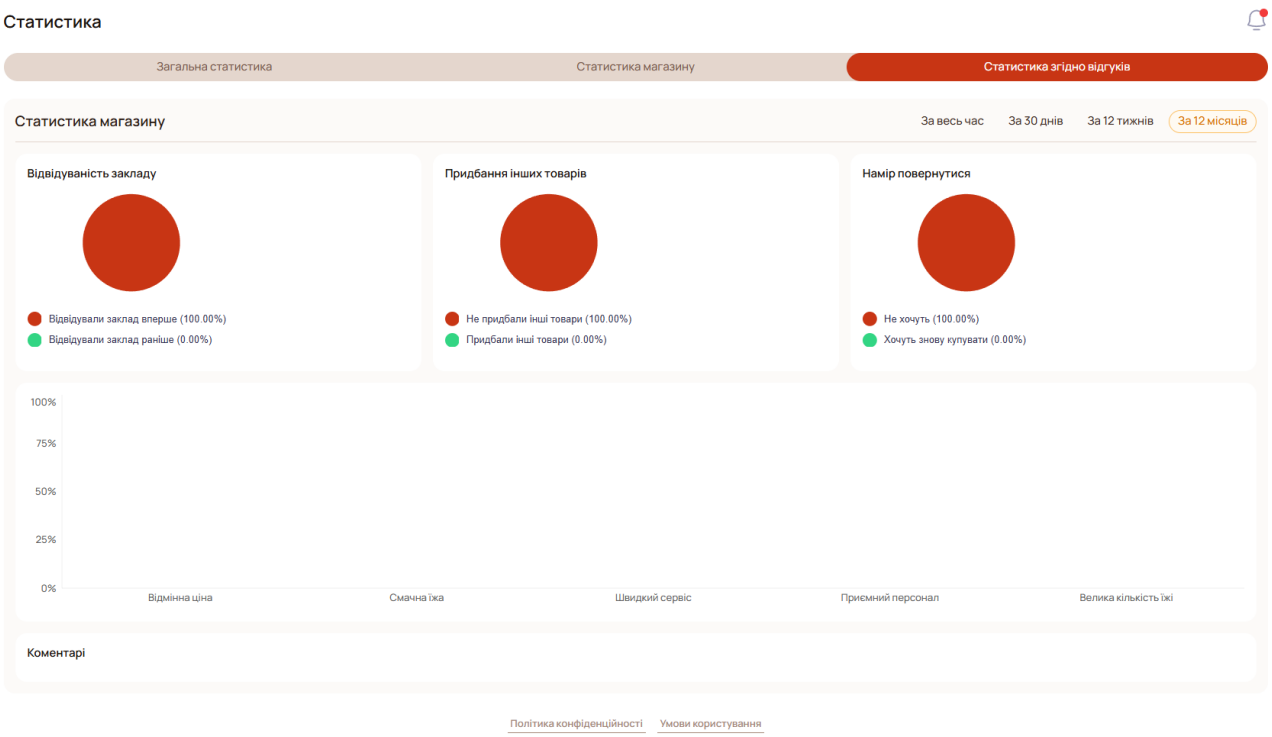


Рисунок 4.32 – Статистика по магазину

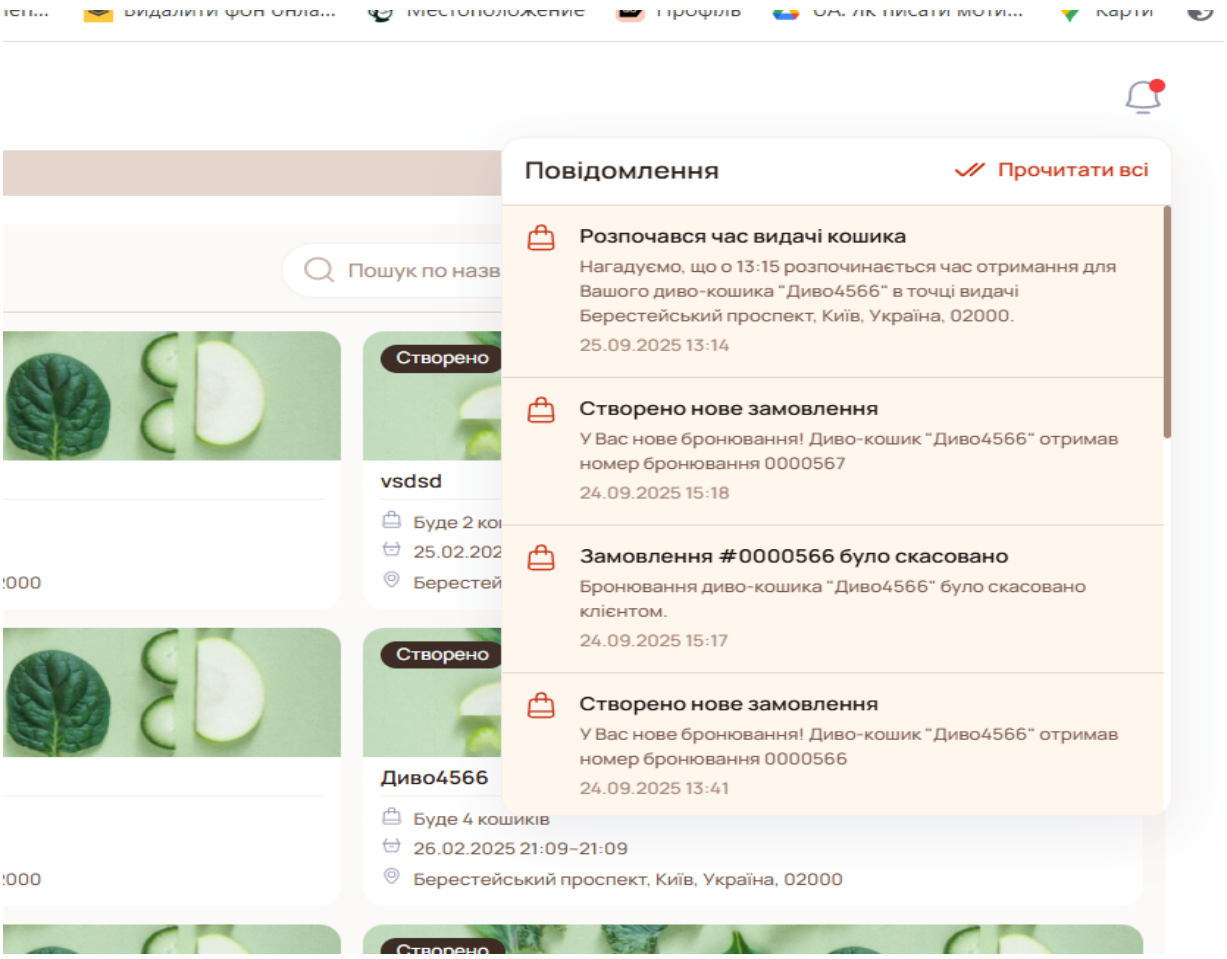


Рисунок 4.33 – Повідомлення по замовленнях

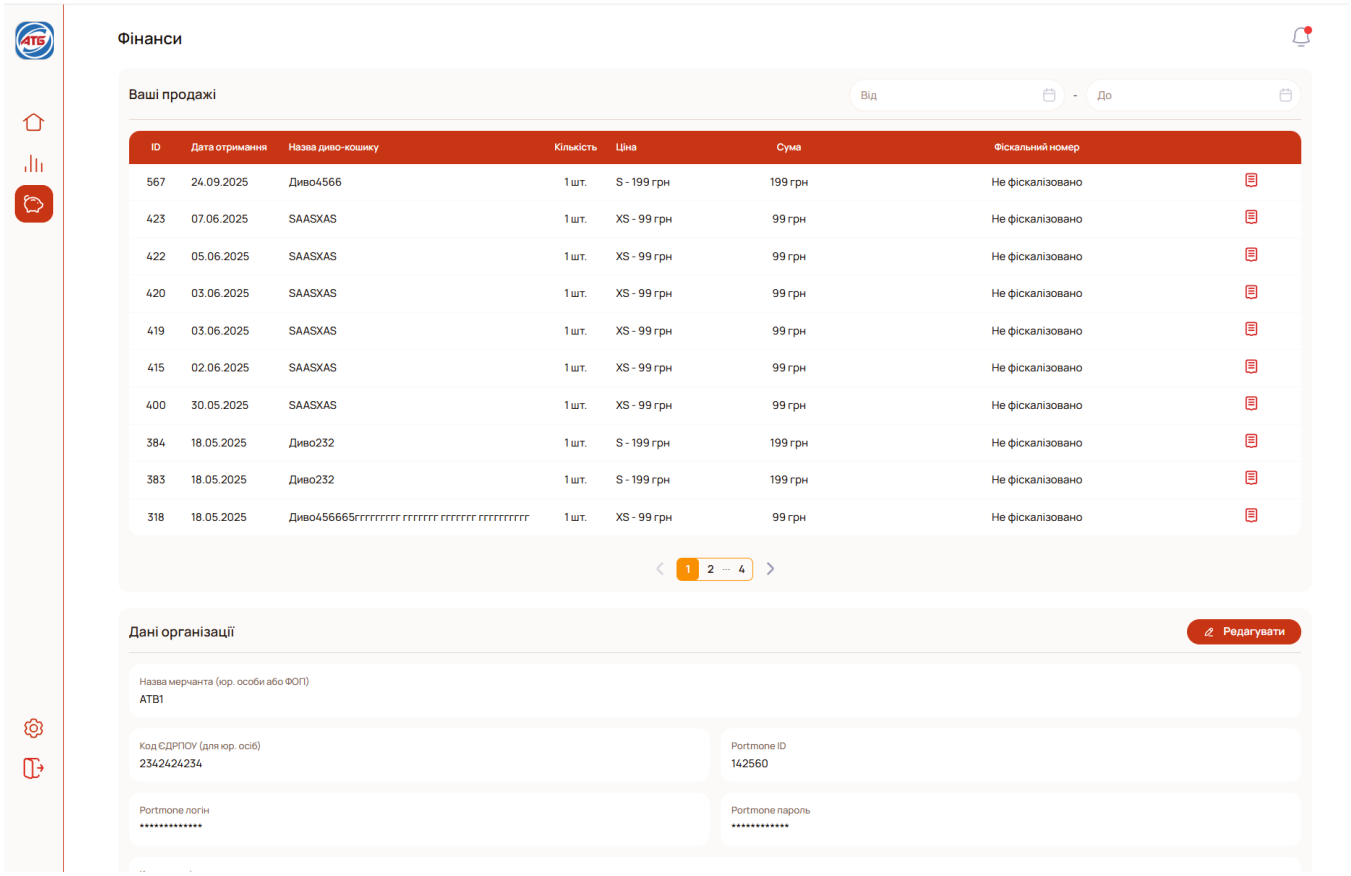


Рисунок 4.34 – Розділ Фінанси

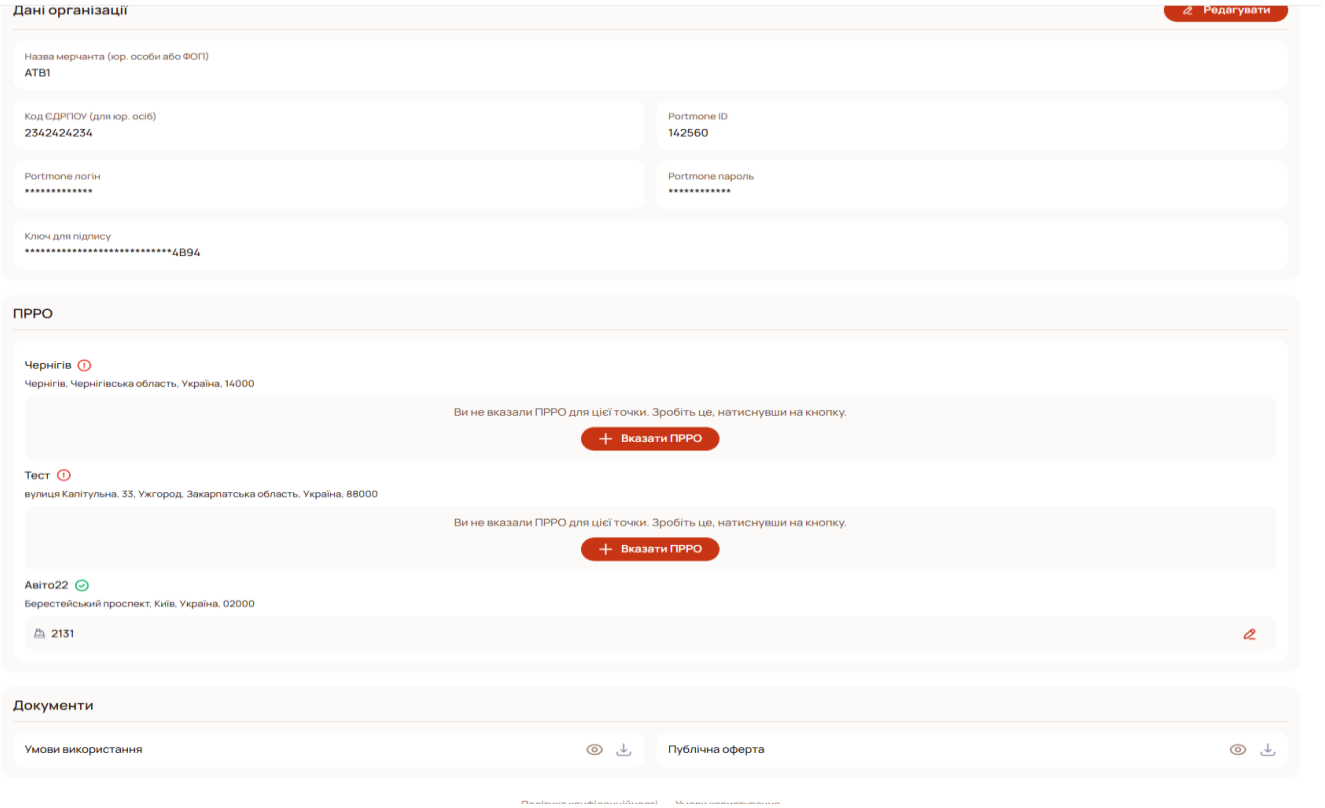


Рисунок 4.35 – Сторінка налаштування ПРРО

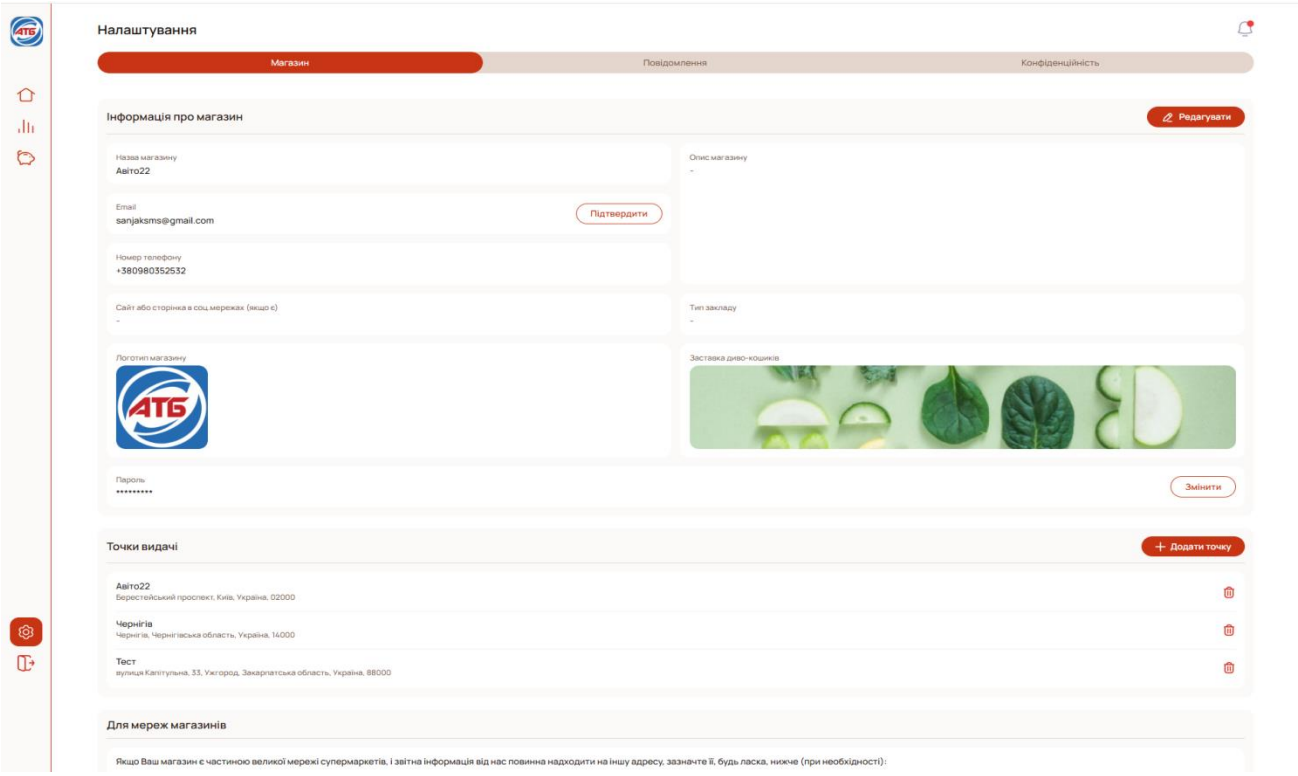


Рисунок 4.36 – Сторінка налаштування магазину

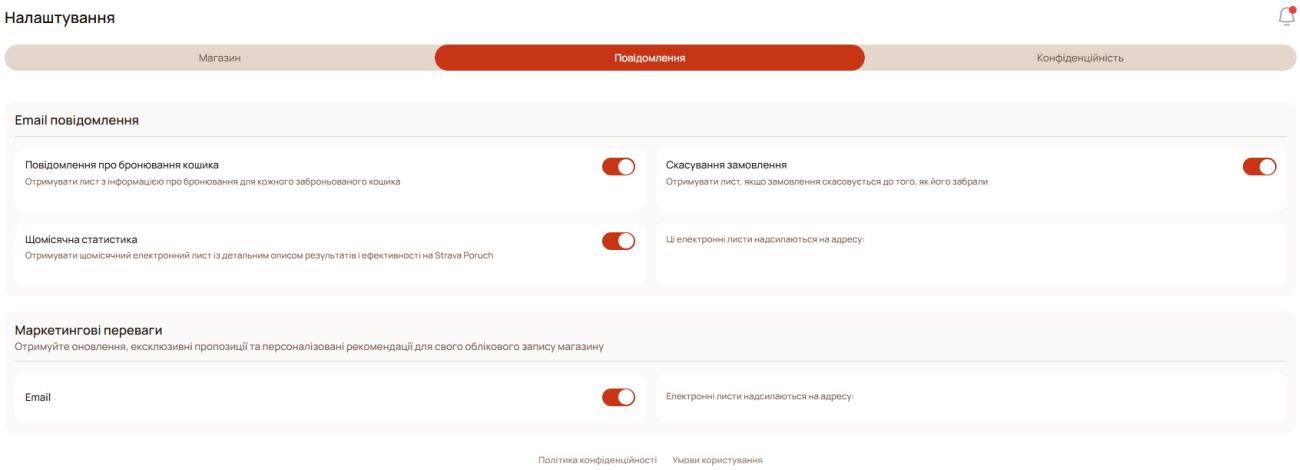


Рисунок 4.37 – Сторінка налаштування повідомлень.

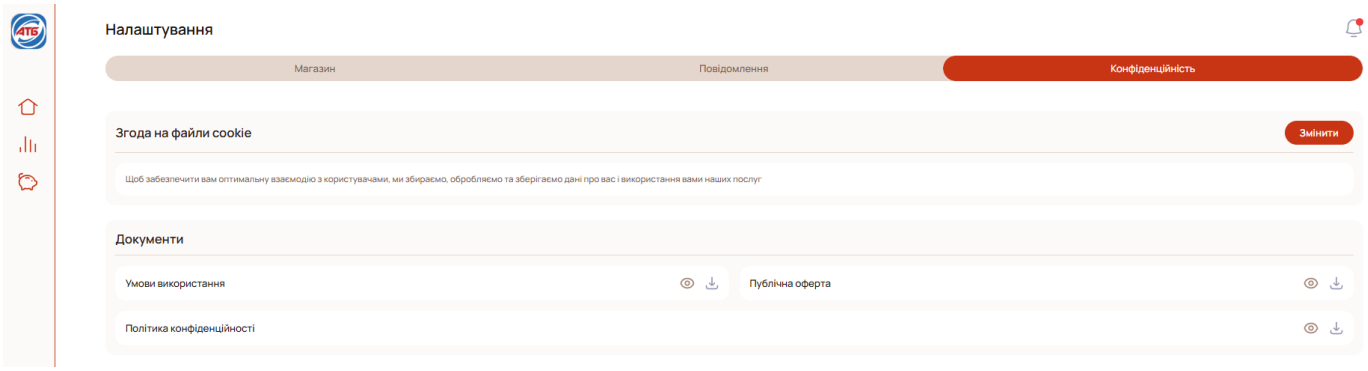


Рисунок 4.38 – Сторінка налаштування конфіденційності.

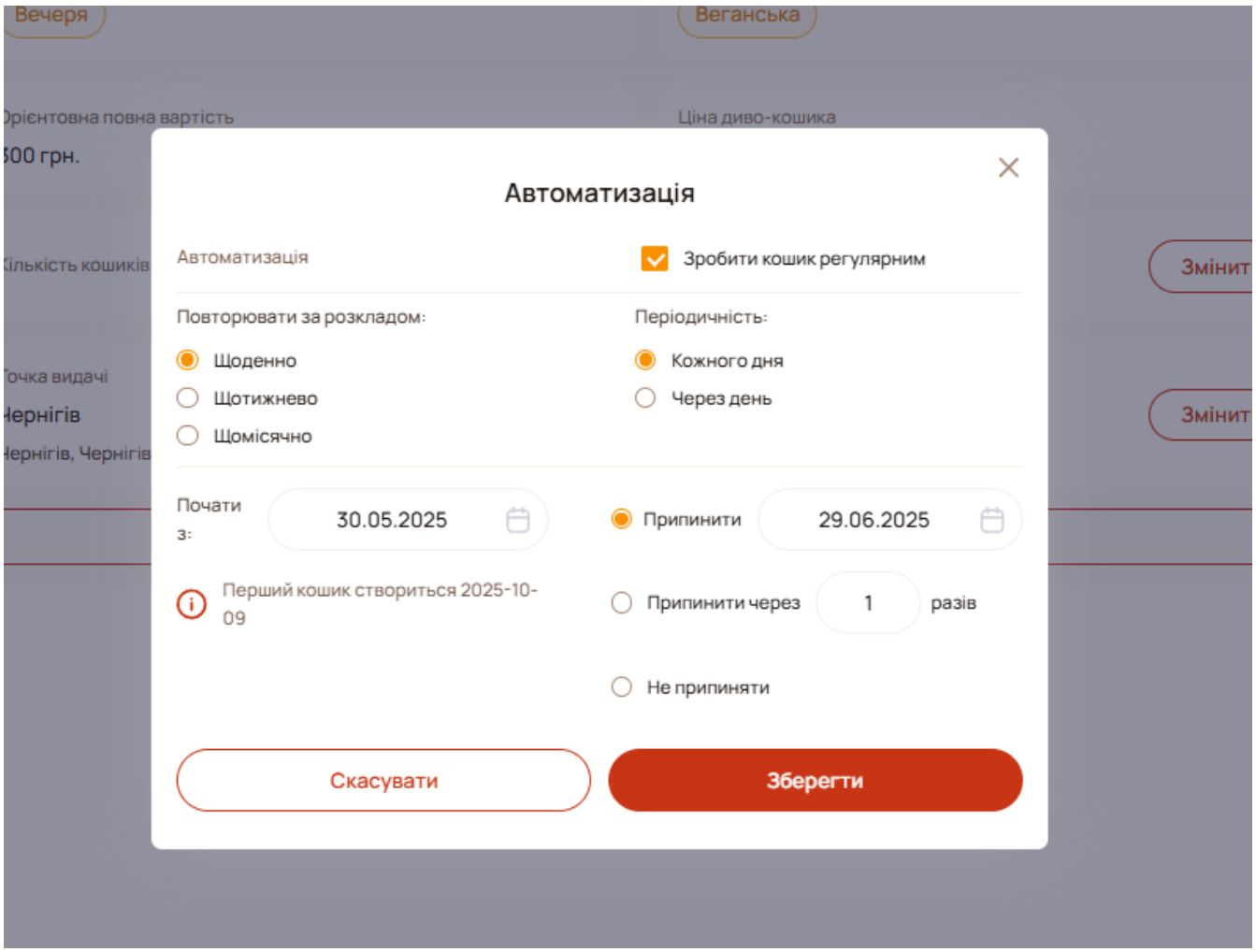


Рисунок 4.39 – Автоматизація кошику

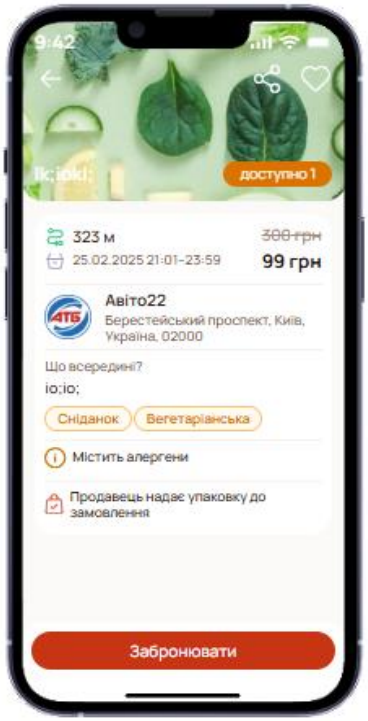


Рисунок 4.40 – Відображення кошика покупцю

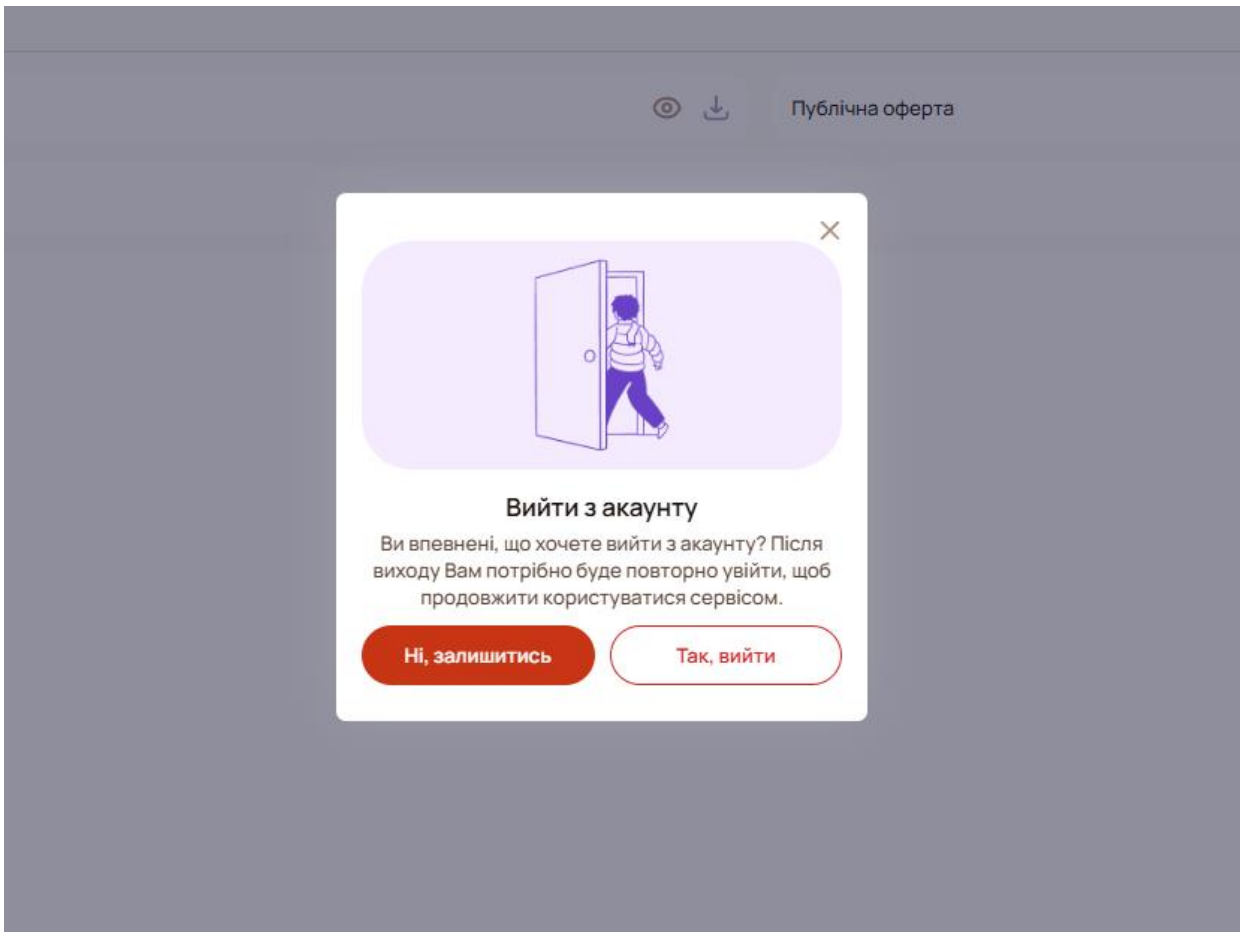


Рисунок 4.41 – Інтерфейс виходу користувача з платформи

4.3. Висновки до розділу

У межах проведеного тестування веб-платформи E-Market було підтверджено її функціональну готовність до використання як інструменту сталого розвитку в сфері громадського харчування. Система успішно реалізує ключові завдання — створення, управління та автоматизацію кошиків із залишковими стравами, забезпечуючи зручну взаємодію між закладами харчування та кінцевими споживачами.

Результати тестування засвідчили стабільну роботу інтерфейсних компонентів, коректну обробку запитів, адаптивність до різних пристроїв, а також відповідність вимогам безпеки та продуктивності. Особливу увагу було приділено перевірці модулів авторизації, створення кошика, автоматизації повторень, обробки даних користувача та інтеграції з серверною частиною.

Виявлені недоліки були незначними та оперативно усунуті, що свідчить про високий рівень технічної реалізації та гнучкість архітектури платформи. Усі критично важливі функції — включаючи управління розкладом, валідацію форм, обробку помилок та синхронізацію з бекендом — продемонстрували стабільну та передбачувану поведінку.

Таким чином, веб-платформа E-Market може бути рекомендована до впровадження як ефективне рішення для зменшення харчових втрат, підтримки екологічної відповідальності та розвитку сталих практик у закладах громадського харчування.Цей підхід до розробки сприяє створенню гнучких, адаптивних та інтерактивних компонентів платформи, що значно спрощує процес організації та обробки бізнес-документів.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1. Ідея стартапу E-Market

У сучасних умовах сталого розвитку особливу актуальність набувають цифрові рішення, що сприяють зменшенню харчових втрат та раціональному використанню ресурсів [5]. Саме таку мету має стартап-проєкт E-Market — екологічна платформа, яка дозволяє закладам громадського харчування реалізовувати залишкові страви через інтерактивний маркетплейс.

Ідея полягає у створенні інтелектуальної системи, яка автоматизує процес формування кошиків із непроданими стравами, забезпечує їх доступність для кінцевих споживачів та сприяє зменшенню обсягів утилізації. Платформа дозволяє закладам швидко публікувати пропозиції, а користувачам — зручно переглядати, бронювати та оплачувати залишкові страви.

Інтеграція з системами сповіщень, автоматичне планування повторюваних кошиків, аналітика попиту та екологічний моніторинг — усе це робить E-Market не лише комерційним інструментом, а й соціально значущим рішенням у сфері сталого споживання.

5.2. Технологічна реалізація платформи E-Market

Архітектура платформи побудована на сучасному стеку технологій: React + Redux Toolkit на фронтенді та Yii2 + REST API на бекенді. Такий підхід забезпечує високу продуктивність, гнучкість та масштабованість системи.

Компонентна структура дозволяє розділити функціональність на окремі модулі — авторизація, створення кошика, автоматизація, управління замовленнями.

Redux Toolkit забезпечує централізоване управління станом, що критично важливо для синхронізації даних між користувачем і сервером.

RTK Query використовується для обробки асинхронних запитів, включаючи створення, редагування, оплату та підтвердження кошиків.

WebSocket-з’єднання реалізовано через Socket.IO, що дозволяє отримувати сповіщення в реальному часі — про нові замовлення, зміну статусу, підтвердження

доставки.

Інтерфейс адаптовано до мобільних пристроїв, що забезпечує зручність користування для широкої аудиторії.

Застосування віртуального DOM, одностороннього потоку даних та гнучкої маршрутизації через React Router дозволяє платформі швидко реагувати на дії користувача, мінімізуючи навантаження на браузер і сервер.

5.3. Перспективи та ринковий потенціал E-Market

У контексті глобальних викликів щодо харчових втрат, екологічної відповідальності та сталого споживання, платформа E-Market має високий потенціал для виходу на ринок. Її функціональність відповідає актуальним запитам:

Заклади харчування отримують інструмент для монетизації залишкових страв [13].

Користувачі — доступ до якісної їжі за зниженими цінами.

Громада — зменшення обсягів харчових відходів.

Аналіз ринку свідчить про зростання інтересу до екологічних сервісів, особливо серед молоді, свідомих споживачів та закладів, які прагнуть покращити свій імідж. E-Market здатен охопити різні сегменти — від малих кафе до великих ресторанів, від студентів до сімейних покупців.

Інтуїтивний інтерфейс, автоматизація процесів, екологічна місія — усе це формує конкурентну перевагу платформи та відкриває перспективи для масштабування на регіональному та національному рівнях.5.4. Розроблення ринкової стратегії проєкту

5.4. Маркетингова стратегія запуску E-Market

Для ефективного просування платформи E-Market передбачено багаторівневу маркетингову стратегію [7]:

Створення промоційного сайту з описом функціональності, переваг та екологічної місії. Це дозволить залучити ранніх користувачів та підвищити

авторитетність домену через SEO.

Активна присутність у соціальних мережах — Facebook, Instagram, TikTok, LinkedIn. Контент буде орієнтований на екологічну тематику, історії закладів, інтерв'ю з користувачами.

Партнерські програми з закладами харчування — надання бонусів за участь, брендовані матеріали, спільні акції.

Інтеграція з локальними ініціативами — Zero Waste, Food Sharing, екологічні фестивалі [6].

Модель монетизації — комісія з кожного замовлення, преміум-функції для закладів, аналітика попиту.

Стратегія передбачає поступове розширення функціональності, адаптацію до потреб ринку та формування лояльної спільноти навколо платформи.

5.5. Висновки розділу

Розробка стартап-проєкту E-Market підтвердила актуальність і життєздатність ідеї екологічного маркетплейсу для реалізації залишкових страв. Технологічна архітектура, функціональна гнучкість та соціальна місія платформи створюють умови для її успішного впровадження.

Проведений аналіз ринку, тестування інтерфейсу та формування маркетингової стратегії свідчать про високий потенціал E-Market як інструменту сталого розвитку. Платформа здатна не лише вирішувати проблему харчових втрат, а й формувати нову культуру відповідального споживання.

ВИСНОВКИ

У результаті реалізації технічного завдання було створено веб-платформу E-Market — інтелектуальну екологічну систему, що поєднує зручність використання, високу продуктивність та широкий набір функцій для управління залишковими стравами в закладах громадського харчування. Платформа виконує не лише роль інструменту для розміщення та продажу кошиків, а й функціонує як комплексне рішення для автоматизації процесів, аналітики та екологічного моніторингу.

Архітектура E-Market побудована на компонентному принципі, що передбачає поділ системи на незалежні, але взаємопов’язані модулі — авторизація, створення кошика, управління замовленнями, автоматизація повторень, сповіщення в реальному часі. Такий підхід забезпечує гнучкість у розробці, спрощує оновлення функціоналу та дозволяє легко інтегрувати нові можливості без порушення загальної логіки роботи платформи.

Модульна структура сприяє підтримці стабільності системи в довгостроковій перспективі, дозволяючи адаптувати її до змін ринкових умов, екологічних вимог та очікувань користувачів. Кожен компонент може бути вдосконалений окремо, що забезпечує ефективне масштабування та розвиток платформи без втрати якості.

Таким чином, E-Market є технічно обґрунтованим, соціально значущим та екологічно орієнтованим рішенням, здатним підтримати сталий розвиток у сфері громадського харчування та сформувати нову культуру відповідального споживання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Продовольча і сільськогосподарська організація ООН (FAO). Food Waste Index Report 2021. – https://www.fao.org/food-loss-and-food-waste
2. United Nations Environment Programme (UNEP). Sustainable Food Systems and Consumption. – https://www.unep.org
3. Європейська комісія. Farm to Fork Strategy – стратегія сталого розвитку харчової системи ЄС. – https://food.ec.europa.eu
4. Закон України «Про основи екологічної політики України на період до 2030 року». – https://zakon.rada.gov.ua/laws/show/2697-19
5. Harvard Business Review. The business case for sustainability in food services. – https://hbr.org/2022/11/the-business-case-for-sustainability-in-food-services
6. Open Food Network. Digital platforms for sustainable food distribution. – https://openfoodnetwork.org
7. Claspo.io.. Маркетинг сталого розвитку: стратегії та приклади. – https://claspo.io/ua/blog/what-is-sustainable-marketing-definition-and-strategies-with-examples/
8. React Documentation. Офіційна документація фреймворку React. – https://reactjs.org
9. Redux Toolkit Documentation. Управління станом у React-додатках. – https://redux-toolkit.js.org
10. Socket.IO Documentation. Реалізація WebSocket-з’єднань. – https://socket.io/docs
11. Yii2 Framework Guide. Офіційна документація PHP-фреймворку Yii2. – https://www.yiiframework.com/doc/guide/2.0/en
12. Степаненко В.І. Екологічні аспекти цифрової трансформації в харчовій промисловості // Економіка та суспільство. – 2023. – №45. – С. 112–117. – https://economyandsociety.in.ua/index.php/journal/article/view/2345
13. Ковальчук О.М. Інноваційні рішення для зменшення харчових втрат у HoReCa-секторі // Вісник аграрної науки. – 2024. – №3. – С. 67–72.

ДОДАТКИ

Код компонентів клієнтської частини платформи E-Market

```
const ServicesPage = lazy(() => import('../pages/services/ServicesPage'));
const SettingsPage = lazy(() => import('../pages/settings/SettingsPage'));
const NotFoundPage = lazy(() => import('../pages/registration/NotFoundPage'));
const VerifyEmail = lazy(() => import('../pages/registration/VerifyPage'));
const PasswordRecoveryPage = lazy(() => import('../pages/registration/PasswordRecoveryPage'));
const RenewPasswordPage = lazy(() => import('../pages/registration/RenewPasswordPage'));
const ViewBagPage = lazy(() => import('../pages/main/ViewBagPage'));
const ArchivePage = lazy(() => import('../pages/main/ArchivePage'));
const ShopSettingsPage = lazy(() => import('../pages/settings/ShopSettingsPage'));
const ShopEditSettingsPage = lazy(() => import('../pages/settings/ShopEditSettingsPage'));
const ShopAddSettingsPage = lazy(() => import('../pages/settings/ShopAddSettingsPage'));
const MessagePage = lazy(() => import('../pages/settings/MessagesPage'));
const ConfidentialityPage = lazy(() => import('../pages/settings/ConfidentialityPage'));
const GeneralStatisticsPage = lazy(() => import('../pages/statistics/GeneralStatisticsPage'));
const ShopStatisticsPage = lazy(() => import('../pages/statistics/ShopStatisticsPage'));
const ReviewStatisticsPage = lazy(() => import('../pages/statistics/ReviewStatisticsPage'));

export const App = () => {
  const token = useSelector(selectToken);
  const dispatch = useDispatch();
  const isFetching = useSelector(selectIsFetchingCurrentUser);    'isFetching' is assigned a value but never
  const isLogged = useSelector(selectIsLoggedIn);    'isLogged' is assigned a value but never used.
  const [isLoading, setIsLoading] = useState(true);
  useEffect(() => {

    dispatch(getCurrentUser());
  }, [dispatch]);
  useEffect(() => {
    // Симулюємо завантаження анімації    "Симулюємо": Unknown word.
    const timer = setTimeout(() => {
      setIsLoading(false);
    }, 1150); //
    return () => clearTimeout(timer);
  }, []); ← #55-61 useEffect
  return (
    <>
      {isLoading ? (
        <MyLottieAnimation />
      ) : (
        <Suspense fallback={<ImageAnimation />}>
            <SocketProvider token={token} url1={url1}>
            <Routes>
            <Route
              path="/register"
              element={
                <RestrictedRoute
                  component={<RegisterPage />}
                  navigateTo="/main"
                />
              } ← #72-77 element=
            />

            <Route
              path="/"
              element={
                <RestrictedRoute component={<LoginPage />} navigateTo="/main" />
              }
            />
            <Route
              path="/renew-password"
              element={
                <RestrictedRoute
                  component={<RenewPasswordPage />}
                  navigateTo="/main"
                />
              } ← #88-93 element=
```

```
/>
<Route
  path="/renew-password"
  element={
    <RestrictedRoute
      component={<RenewPasswordPage />}
      navigateTo="/main"
    />
  } ← #88-93 element=
/>


<Route
  path="/password_recovery"
  element={
    <RestrictedRoute
      component={<PasswordRecoveryPage />}
      navigateTo="/main"
    />
  } ← #99-104 element=
/>

<Route path="/" element={<MainLayout />}>


  <Route
    path="main/"
    element={
      <PrivateRoute component={<MainPage />} navigateTo="/" />
    }
  >
    <Route
      path=""
      element={
        <PrivateRoute component={<DashBoardPage />} navigateTo="/" />
      }
    />
    <Route
      path="calendar/:currentmonth"    "currentmonth": Unknown word.
      element={
        <PrivateRoute component={<CalendarPage />} navigateTo="/" />
      }
    />
    <Route
      path="archivebags"    "archivebags": Unknown word.
      element={
        <PrivateRoute component={<ArchivePage />} navigateTo="/" />
      }
    />
    <Route
      path="addbag/:id"    "addbag": Unknown word.
      element={
        <PrivateRoute component={<AddBagPage />} navigateTo="/" />
      }
    />
    <Route
      path="viewbag/:id"    "viewbag": Unknown word.
      element={
        <PrivateRoute component={<ViewBagPage />} navigateTo="/" />
      }
    />

  </Route>
```

```
const AddBag = () => {
  dayjs.extend(isToday);
  dayjs.extend(isTomorrow);
  const { id } = useParams();
  const location = useLocation();
  const searchParams = new URLSearchParams(location.search);
  const currentDay = searchParams.get('currentDay');
  const mode = searchParams.get('mode');

  const { data: bag, isLoading: isLoadingBag } = useGetBagQuery(id, {    'isLoadingBag' is assigned a val
    skip: id === 'id',  // скіп запиту, якщо id дорівнює 'id'    "скіп": Unknown word.
  });

  dayjs.extend(utc);
  const [calculateBag, { isLoading: isLoading1 }] = useCalculateBagMutation();
  const { data, } = useFetchCategoriesQuery();
  const { data: pricesData, isLoading: isLoadingPrices } = useFetchPricesQuery();    'isLoadingPrices' is
  const { data: shopsData, isLoading: isLoadingShop } = useFetchShopsQuery();    'isLoadingShop' is assig
  const [createBag] = useCreateBagMutation();
  const [editBag] = useEditBagMutation();
  const navigate = useNavigate();
    const today = dayjs().tz(KYIV_TZ)
  const currentDayOfMonth = today.date();
  const formattedDate = currentDay ? currentDay : today.format('YYYY-MM-DD');
  const nextMonthDate = today.add(1, 'month').format('YYYY-MM-DD');

  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [quantity, setQuantity] = useState(1);
  const [categories, setCategories] = useState([]);
  const [diets, setDiets] = useState([]);
  const [shops, setShops] = useState([]);
  const [selectedDiets, setSelectedDiets] = useState([]);
  const [selectedCategories, setSelectedCategories] = useState([]);
  const [containsAllergens, setContainsAllergens] = useState(false);
  const [buffetFood, setBuffetFood] = useState(false);
  const [packaging, setPackaging] = useState('shop');
  const [firstTime, setFirstTime] = useState(createDateWithTime(0)); // Початковий час    "Початковий": U
  const [secondTime, setSecondTime] = useState(createDateWithTime(23, 59));
  const [isAutomation, setIsAutomation] = useState(false);
  const [repeatSchedule, setRepeatSchedule] = useState('daily');
  const [selectedDays, setSelectedDays] = useState([]);
  const [startDate, setStartDate] = useState(formattedDate);
  const [stopCondition, setStopCondition] = useState('never');
  const [stopDate, setStopDate] = useState(nextMonthDate);
  const [stopAfter, setStopAfter] = useState(1);
  const [priceBags, setPriceBags] = useState([]);
  const [point, setPoint] = useState(``);
  const [pointId, setPointId] = useState(``);
  const [selectedFullPrice, setSelectedFullPrice] = useState('0.00грн');
  const [selectedBagPrice, setSelectedBagPrice] = useState('0.00грн');
  const [selectedDeliveryDay, setSelectedDeliveryDay] = useState('today');
  const [selectDaily, setSelectDaily] = useState('1');
  const [dayOfMonth, setDayOfMonth] = useState(currentDayOfMonth);

  const [arrayErrors, setArrayErrors] = useState({
    nameErr: false,
    descErr: false,
    categErr: false,    "categ": Unknown word.
    costErr: false,
    timeErr: false,
```

```
                <CheckBoxWrapper>
                  <input type="checkbox" id="containsAllergens" checked={containsAllergens}
                    onChange={() => setContainsAllergens(!containsAllergens)} />
                  <label htmlFor="containsAllergens" className="checkmark"
                  ></label>
                </CheckBoxWrapper>

                <Label1 htmlFor="containsAllergens">Містить алергени</Label1>
              </HorizontalBox>
              <HorizontalBox>
                <CheckBoxWrapper>
                  <input type="checkbox" id="buffetFood" checked={buffetFood}
                    onChange={() => setBuffetFood(!buffetFood)} />
                  <label htmlFor="buffetFood" className="checkmark"></label>
                </CheckBoxWrapper>
                <Label1 htmlFor="buffetFood">Містить їжу зі шведського столу</Label1>
              </HorizontalBox>
            </VerticalBox>
            <VerticalBox>
              <HorizontalBox>

                <RadioWrapper>
                  <input
                    type="radio"
                    id="shop"
                    name="packaging"
                    value="shop"
                    checked={packaging === 'shop'}
                    onChange={() => setPackaging('shop')}
                  />
                  <label htmlFor='shop' className="radioMark"></label>
                </RadioWrapper>
                <Label1 htmlFor="shop">Продавець надає упаковку до замовлення</Label1>
              </HorizontalBox>
              <HorizontalBox>

                <RadioWrapper>
                  <input
                    type="radio"
                    id="own"
                    name="packaging"
                    value="own"
                    checked={packaging === 'own'}
                    onChange={() => setPackaging('own')}
                  />
                  <label htmlFor='own' className="radioMark"></label>
                </RadioWrapper>
                <Label1 htmlFor="own">Просимо прийти зі свою упаковкою</Label1>
              </HorizontalBox>
            </VerticalBox>
          </MainBox>
        </FieldWrapper>

        <FieldWrapper errors={arrayErrors.costErr} >
          <Label>Орієнтовна повна вартість</Label>
```

```
import MainNavToolbar from '../MainNavToolbar';
import AlertNoFunc from '../AlertNoFunc';    'AlertNoFunc' is defined but never used.
import BagsToolbar from '../BagsToolbar';
import { format } from 'date-fns';    'format' is defined but never used.
import { uk } from 'date-fns/locale';    'uk' is defined but never used.
import { useSelector } from 'react-redux';
import { selectIsFetchingCurrentUser, selectUser } from '../../../redux/auth/selectors';    'selectIsFetchingCurrentUser' is defined bu
import OrdersToolbar from '../OrdersToolbar';
import { AllBox } from 'components/finance/Finance/Finance.styled';
import { useFetchShopsQuery } from '../../../redux/shops/shopsApi';
import { useGetBagsQuery } from '../../../redux/bags/bagsApi';
import { useGetOrdersQuery } from '../../../redux/orders/ordersApi';
import { useFetchOrderStatusesQuery } from '../../../redux/category/categoriesApi';
import { useEffect, useState } from 'react';
import ImageAnimation1 from 'components/layout/loader/ImageAnimation1';
import { isFuture, isToday, parseISO } from 'date-fns';    'isFuture' is defined but never used.
import { io } from 'socket.io-client';
import { selectToken } from '../../../redux/auth/selectors';
import { useSearchParams } from 'react-router-dom';
import { url } from 'components/other/terms';
import { Action, ActionText } from './DashBoard.styled';    'Action' is defined but never used.

const DashBoard = () => {
  const token = useSelector(selectToken);
  const [currentPage, setCurrentPage] = useState(1);
  const [searchKeyword, setSearchKeyword] = useState('');
  const { data: bagsData, isLoading: isLoadingBag, refetch, isFetching } = useGetBagsQuery();
  const { data: shopsData, isLoading: isLoadingShops } = useFetchShopsQuery();
  const { data: ordersData, isLoading: isLoadingOrders } = useGetOrdersQuery({ page: currentPage, keyword: searchKeyword });
  const { data: orderStatuses, isLoading: isLoadingStatuses, refetch:refetchOrder } = useFetchOrderStatusesQuery();
  const user = useSelector(selectUser);
  const [showAnimation, setShowAnimation] = useState(true);
  const [socket, setSocket] = useState(null);    'socket' is assigned a value but never used.
  const [searchParams, setSearchParams] = useSearchParams();    'setSearchParams' is assigned a value but never used.
  const onboard = searchParams.get('onboard');    'onboard' is assigned a value but never used.
  useEffect(() => {
    const baseUrl = url+":443";
    const newSocket = io.connect(baseUrl, {
      path: "/ws",
      query: {
        auth_token: token,
      },
      secure: true
    });

    newSocket.on('connect', () => {
      console.log('Socket підключено');    "підключено": Unknown word.
    });

    newSocket.on('message', (data) => {

      console.log('повідомлення :>> ', data);    "повідомлення": Unknown word.
      const parsedCreatedAt = new Date(data.createdAt); // Передаємо оригінальний рядок без змін    "Передаємо": Unknown word.
      const newNotification = {    'newNotification' is assigned a value but never used.
        id: data.id,
        title: data.title,
        message: data.message,
        createdAt: isNaN(parsedCreatedAt) ? new Date() : parsedCreatedAt, // Перевіряємо, чи дата валідна    "Перевіряємо": Unknown wor
        read: data.unread
      };



    });
```

```
  try {
    const result = await convertApi.convert('docx', 'html', params);
    const result1 = await convertApi.convert('docx', 'jpg', params);
    // Отримайте посилання на завантаження HTML-файлу    "Отримайте": Unknown
    const htmlFileUrl = result.files[0].Url;
    const htmlFileUrl1 = result1.files[0].Url;

    const prevUrl = await setImage(
      'previous',
      htmlFileUrl1,
      result1.files[0].FileId
    ); ← #65-69 const prevUrl = await setImage
    // Завантажте вміст HTML-файлу    "Завантажте": Unknown word.
    setTemplateImageUrl(prevUrl);
    const response = await fetch(htmlFileUrl);
    const content = await response.text();

    setEditorValue(content);
  } catch (error) {
    console.error('Error converting Word file:', error);
  }
}; ← #52-79 const handleFileChange = async event ⇒

const saveTemplate = () ⇒ {
  dispatch(
    id
      ? updateTemplate({
          id: template.id,
          userEmail: user.email,
          name: templateName,
          description: editorValue,
          templateImageUrl: templateImageUrl,
        }) ← #84-90 ? updateTemplate
      : addTemplate({
          userEmail: user.email,
          name: templateName,
          description: editorValue,
          templateImageUrl: templateImageUrl,
        }) ← #91-96 : addTemplate
  ); ← #82-97 dispatch
}; ← #81-98 const saveTemplate = () ⇒

const handleGeneratePDF = () ⇒ {
  const content = editorValue;

  html2pdf(content, {
    margin: 10,
    filename: 'document.pdf',
    image: { type: 'jpeg', quality: 0.98 },
    html2canvas: { scale: 2 },
    jsPDF: { unit: 'mm', format: 'a4', orientation: 'portrait' },
  }); ← #103-109 html2pdf
}; ← #100-110 const handleGeneratePDF = () ⇒

return (
  <Content edit={template?.id}>
    <MainContent>
      <H2>{template?.id ? 'Редагування шаблону' : 'Створення шаблону'}</H2>
```

```
<H2>{template?.id ? 'Редагування шаблону' : 'Створення шаблону'}</H2>    "Редагування": U
<Form>
  <div
    style={{
      marginBottom: '10px',
      width: '100%',
      display: 'flex',
      gap: '10px',
      alignItems: 'center',
    }} ← #118-124 style=
  >
    <Label>Ім'я шаблону</Label>    "Ім'я": Unknown word.
    <InputName
      type="text"
      placeholder="Ім'я шаблону"    "Ім'я": Unknown word.
      value={templateName}
      onChange={handleChangeName}
    />
    {!user?.email ? (
      <div
        style={{
          backgroundColor: '#d23c46',
          color: 'white',
          display: 'flex',
          justifyContent: 'center',
          alignItems: 'center',
          boxSizing: 'border-box',
          borderRadius: '5px',
        }} ← #135-143 style=
      >
        <span
          style={{
            margin: '0 auto',
            display: 'block',
            textAlign: 'center',
          }} ← #146-150 style=
        >
          Для можливості редагування зареєструйтесь    "можливості": Unknown word.
        </span>
      </div>
    ) : (
      <>
        <Button onClick={saveTemplate}>Зберегти шаблон</Button>    "Зберегти": Unknown wo
        <Button onClick={handleGeneratePDF}>Завантажити PDF</Button>    "Завантажити": Un
      </>
    )} ← #133-160 />
  </div>
  <EditorContainer>
    {!user?.email ? (
      <PdfBox url={template?.templateImageUrl} />
    ) : (
      <MyEditor
        editorValue={editorValue}
        apiKey="a85ckh1rvs1kxnitvo54z16evw4mvc00gqs3ukonw5rzvgd1"
        init={{
          height: '100%',
          menubar: true,
```

```
const CalendarTable = ({
  monthDays = [],
  currentDay,
  setCurrentDay,
  firstDay,
  currentMonth,
  openDay,
  handleMonth ,nextMonth, prevMonth,

}) => {

  const { currentmonth } = useParams();    "currentmonth": Unknown word.
  const navigate = useNavigate();
  const today = new Date().getDate();
  const month = new Date().getMonth();

  return (
    <DataGrid>
<MonthSwitcher className='mobb'>    "mobb": Unknown word.
          <IconLeft onClick={prevMonth} />
          <WrapMonthSText>{currentMonth}</WrapMonthSText>

          <IconRight onClick={nextMonth} />
        </MonthSwitcher>
      {Object.keys(daysOfWeek).map((dayKey) => {
        const day = daysOfWeek[dayKey];

        return (

          <HeaderCell key={dayKey}>
            {day}
          </HeaderCell>
        ) ← #50-55 return
      })} ← #47-56 </MonthSwitcher>
      {monthDays.map(({ date, dateF, bags, isWorking }) => {
        return (
          <Cell
            key={date.toString()}
            className={`${new Date(currentDay).getDate() === dateF.getDate() &&
              isSameMonth(currentDay, dateF) &&
              'row__currentDateActive'
              }` + ` ${today === dateF.getDate() &&    Unexpected string concatenation of literals.
              isSameMonth(new Date(), dateF) &&
              'row__ActiveNumber'
              }`} ← #61-67 className=
            style={{
              backgroundColor: isWorking ? 'var(--colors-secondary-50)' : 'var(--colors-error-50)',
            }}

            onClick={() => {

              navigate(`/main/calendar/${currentmonth}?currentDay=${date}`);    "currentmonth": Unknown word.
              openDay();
            }} ← #72-76 onClick=
          >
            <CurrentDate

            >
              <RowNumber
                className={`${today === dateF.getDate() &&
                  isSameMonth(new Date(), dateF) &&
                  'row__ActiveNumber'
                  }`}
                dateTime={date}
```

omponents / main / CalendarTable / CalendarTable.styled.jsx / ...

```
import styled from '@emotion/styled';

const DataGrid = styled.div`
  display: grid;
  grid-template-columns: repeat(7, 1fr);
  flex: 1;
gap: 0;

  overflow: auto;
  width: 100%;
  max-height: 100%;
  margin-top: 14px;
background: var(--colors-grey-25);
border: 1px solid var(--colors-grey-200);


height: fit-content;

  border-radius: 8px;
@media (max-width: 689px) {
width: 100%;
  overflow: unset;
  }
`;
const HeaderCell = styled.div`

padding: 12px;
  font-family: var(--font-family);
font-weight: 500;
font-size: 14px;
line-height: 150%;
text-align: right;
color: var(--colors-grey-600);
  border-bottom: 1px solid var(--colors-grey-200);
  @media (max-width: 689px) {
text-align: center;
  border-bottom: unset;
  margin-bottom: 4px;
  &:nth-of-type(2){
margin-left: 8px;
}
  &:nth-of-type(8){
margin-right: 8px;
}
  }
`;


const FooterCell = styled.div`
display: flex;
align-items: center;
gap:24px ;
justify-content: center;
  grid-column: span 7; /* Охоплює всі 7 стовпців */    "Охоплює": Unknown word.
  padding: 8px;
  @media (max-width: 689px) {
    gap: 12px;
flex-direction: column;
border-top: 1px solid var(--colors-grey-200);
padding: 8px 16px;
margin-top: 12px;

  }
```

```jsx
import { LinkNav, ListNav, ListNavItem } from './MainNavToolbar.styled';
import { useLocation, } from 'react-router-dom';
import dayjs from 'dayjs';

import 'dayjs/locale/uk'; // Завантажуємо українську локалізацію    "Завантажуємо": Unknown word.

dayjs.locale('uk'); // Встановлюємо українську локалізацію    "Встановлюємо": Unknown word.




const MainNavToolbar = () => {
  const currentDay = dayjs().format('YYYY-MM-DD');
  const currentMonth = dayjs().format('YYYY-MM');


  const location = useLocation();

  const arrays = location.pathname.split('/');
  const currentPage = arrays[arrays.length - 1].includes("-") ? arrays[arrays.length - 2] : arrays[arrays.length - 1];


  return (

    <ListNav>
      <ListNavItem>
        <LinkNav

          to="/"
          className={`${currentPage === 'calendar' ? '' : 'active'}`}
        >Дашборд</LinkNav>    "Дашборд": Unknown word.
      </ListNavItem>
      <ListNavItem>
        <LinkNav

          to={`/main/calendar/${currentMonth}?currentDay=${currentDay}`}
          className={`${currentPage === 'calendar' ? 'active' : ''}`}
        >Календар </LinkNav>    "Календар": Unknown word.
      </ListNavItem>

    </ListNav >


  );
};

export default MainNavToolbar;



export const Label = styled.label`
  padding: 10px;
  font-size: 18px;
  font-weight: bold;
  color: #333; /* Колір тексту */    "тексту": Unknown word.
  margin-top: 10px;
`;
export const H2 = styled.h2`
  font-size: 40px;
  text-shadow: 2px 2px 4px #fff; /* Додає білу обводку */    "Додає": Unknown word.
`;
```

```
const DetailsModalCash = ({ onClose,  shopProfileId }) => {
  console.log('shopProfileId :>> ', shopProfileId);
  const [editCashProfile] = useEditCashProfileMutation();

  const [cashProfile, setCashProfile] = useState(null);
  const [profileType, setProfileType] = useState(null);
  const { data: cashData, isFetching: isFetch3, isLoading: isLoad } = useGetCashProfileQuery(shopProfileId );    'isFe
  // Стан для значень полів за ID    "Стан": Unknown word.
  const [fieldValues, setFieldValues] = useState(null);
  const [erorr, setErorr] = useState(false);    "erorr": Misspelled word.
  const [fields2, setFields2] = useState([]);    'fields2' is assigned a value but never used.
  const [cashData2, setCashData2] = useState(null);    'cashData2' is assigned a value but never used.
  const [load, setLoad] = useState(false);
  const [save, setSave] = useState(false);
  const [isDisabled, setIsDisabled] = useState(true);
  const [showPassword, setShowPassword] = useState(null);


  const toggleShowPassword = (fieldId) => {
    setShowPassword(prev => ({
      ...prev,
      [fieldId]: !prev[fieldId]
    }));
  }; ← #63-68 const toggleShowPassword = (fieldId) =>


  useEffect(() => {
    if (cashData) {
      console.log('cashData :>> ', cashData);
      setCashProfile(cashData)
      setShowPassword(
      cashData?.params.reduce((acc, item) => {
        item?.fields.forEach(field => {
          acc[field.id] = false;
        });
        return acc;
      }, {})) ← #75-81 setShowPassword

      setProfileType(cashData?.type)
      console.log('cashData?.type :>> ', cashData?.type);

    } ← #72-86 if (cashData)

  }, [cashData]); ← #71-88 useEffect


  useEffect(() => {


    if (!cashData ) return;

    const selectedParam = cashData.params.find(param => param.id === profileType);

    if (!selectedParam) {
      setFieldValues({});
      return;
    }

    const newFieldValues = selectedParam.fields.reduce((acc, field) => {
      if (profileType === cashData.type) {
        // Якщо profileType збігається з cashData.type, підставляємо дані з cashData    "Якщо": Unknown word.
        acc[field.id] = cashData[field.id] ?? "";
```

```jsx
import React from 'react';
import {
  ContactContainer,
  ContactInfo,
  Content,
  Label,
  Link,
  MainContent,
  Subtitle,
  Title,
} from './Contacts.styled';
import LogoImg from '../../images/Aidocs.png';    "Aidocs": Unknown word.

const Contacts = () => {
  return (
    <Content>
      <MainContent>
        {' '}
        <img src={LogoImg} alt="logo" width={'450px'} height={'400px'} />
        <ContactContainer>
          <Title>Контакти</Title>    "Контакти": Unknown word.
          <Subtitle>
            Зв'яжіться з нами для отримання додаткової інформації:    "Зв'яжіться":
          </Subtitle>
          <ContactInfo>
            <Label>Email:</Label>{' '}
            <Link href="mailto:info@aidocs.com">info@aidocs.com</Link>
          </ContactInfo>
          <ContactInfo>
            <Label>Телефон:</Label>{' '}    "Телефон": Unknown word.
            <Link href="tel:+380980000000">+38 098 000 000 0</Link>
          </ContactInfo>
          <ContactInfo>
            <Label>Адреса:</Label> AI DOCS, вул. Центральна 10, м. Kmdsd,    "Адрес
            Україна    "Україна": Unknown word.
          </ContactInfo>
        </ContactContainer>
      </MainContent>
    </Content>
  );
};

export default Contacts;
```

```jsx
const DetailsModalCash = ({ onClose,  shopProfileId }) => {
  console.log('shopProfileId :>> ', shopProfileId);
  const [editCashProfile] = useEditCashProfileMutation();

  const [cashProfile, setCashProfile] = useState(null);
  const [profileType, setProfileType] = useState(null);
  const { data: cashData, isFetching: isFetch3, isLoading: isLoad } = useGetCashProfileQuery(shopProfileId );    'isFetch3' is
  // Стан для значень полів за ID    "Стан": Unknown word.
  const [fieldValues, setFieldValues] = useState(null);
  const [erorr, setErorr] = useState(false);    "erorr": Misspelled word.
  const [fields2, setFields2] = useState([]);    'fields2' is assigned a value but never used.
  const [cashData2, setCashData2] = useState(null);    'cashData2' is assigned a value but never used.
  const [load, setLoad] = useState(false);
  const [save, setSave] = useState(false);
  const [isDisabled, setIsDisabled] = useState(true);
  const [showPassword, setShowPassword] = useState(null);


  const toggleShowPassword = (fieldId) => {
    setShowPassword(prev => ({
      ...prev,
      [fieldId]: !prev[fieldId]
    }));
  }; ← #63-68 const toggleShowPassword = (fieldId) =>


  useEffect(() => {
    if (cashData) {
      console.log('cashData :>> ', cashData);
      setCashProfile(cashData)
      setShowPassword(
      cashData?.params.reduce((acc, item) => {
        item?.fields.forEach(field => {
          acc[field.id] = false;
        });
        return acc;
      }, {})) ← #75-81 setShowPassword

      setProfileType(cashData?.type)
      console.log('cashData?.type :>> ', cashData?.type);

    } ← #72-86 if (cashData)

  }, [cashData]); ← #71-88 useEffect


  useEffect(() => {


    if (!cashData ) return;

    const selectedParam = cashData.params.find(param => param.id === profileType);

    if (!selectedParam) {
      setFieldValues({});
      return;
    }

    const newFieldValues = selectedParam.fields.reduce((acc, field) => {
      if (profileType === cashData.type) {
        // Якщо profileType збігається з cashData.type, підставляємо дані з cashData    "Якщо": Unknown word.
        acc[field.id] = cashData[field.id] ?? "";
      } else {
        // Інакше всі значення порожні    "Інакше": Unknown word.
        acc[field.id] = "";
```

```
import { configureStore } from '@reduxjs/toolkit';
import {
  persistStore,
  persistReducer,

} from 'redux-persist';

import { authReducer } from './auth/slice';
import { categoriesApi } from './category/categoriesApi';
import { shopsApi } from './shops/shopsApi';
import { bagsApi } from './bags/bagsApi';
import { calendarApi } from './calendar/calendarApi';
import storage from 'redux-persist/lib/storage';
import { faqApi } from './faq/faqApi';
import { financeApi } from './financial/financeApi';
import { settingsApi } from './settings/settingsApi';
import { documentsApi } from './documents/documentsApi';
import { ordersApi } from './orders/ordersApi';
import { statisticsApi } from './statistics/statisticsApi';
import { notificationApi } from './notification/notificationApi';
import { createTransform } from 'redux-persist';

const accessTransform = createTransform(
  (inboundState, key) => {
    return {
      token: inboundState.token,
      refreshToken: inboundState.refreshToken,
    };
  },
  (outboundState, key) => {
    return {
      token: outboundState.token,
      refreshToken: outboundState.refreshToken,
    };
  },
  { whitelist: ['access'] }
);

const authPersistConfig = {
  key: 'authM',
  storage,
  whitelist: ['access'],
  transforms: [accessTransform],
};

export const store = configureStore({
  reducer: {
    auth: persistReducer(authPersistConfig, authReducer),
    [categoriesApi.reducerPath]: categoriesApi.reducer,
    [shopsApi.reducerPath]: shopsApi.reducer,
    [bagsApi.reducerPath]: bagsApi.reducer,
    [calendarApi.reducerPath]: calendarApi.reducer,
    [faqApi.reducerPath]: faqApi.reducer,
    [financeApi.reducerPath]: financeApi.reducer,
    [settingsApi.reducerPath]: settingsApi.reducer,
    [documentsApi.reducerPath]: documentsApi.reducer,
    [ordersApi.reducerPath]: ordersApi.reducer,
    [statisticsApi.reducerPath]: statisticsApi.reducer,
    [notificationApi.reducerPath]: notificationApi.reducer,

  },

  middleware: getDefaultMiddleware => [
    ...getDefaultMiddleware({
      serializableCheck: false
```

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';    'fetchBaseQuery' is defined but never used.
import { url } from 'components/other/terms';    'url' is defined but never used.
import { checkAndRefreshToken } from 'utils/authUtils';    'checkAndRefreshToken' is defined but never used.
import { createBaseQueryWithReauth } from 'utils/baseQueryWithReauth';


export const notificationApi = createApi({
  reducerPath: 'notifications',
  baseQuery: createBaseQueryWithReauth('/notifications'),


  tagTypes: ['Notifications'],

  endpoints: builder => ({
    getNotifications: builder.query({
      query: ({ page = 1 } = {}) => {

        const params = {
          page
        };
        return {
          url: `?page=${page}`,
          method: 'GET',
          params,
        };
      },
      transformResponse: (response, meta) => {
        const pagination = {
          currentPage: meta.response.headers.get('x-pagination-current-page'),
          pageCount: meta.response.headers.get('x-pagination-page-count'),
          perPage: meta.response.headers.get('x-pagination-per-page'),
          totalCount: meta.response.headers.get('x-pagination-total-count'),
        };
        return { data: response, pagination };
      },
      providesTags: ['Notifications'],
    }),

    readNotifications: builder.mutation({
      query: (ids) => ({
        url: `/read?ids=${ids}`,
        method: 'POST',

      }),
      invalidatesTags: ['Notifications'],
    }),

  }),
});

export const {
  useGetNotificationsQuery,
  useReadNotificationsMutation
} = notificationApi;
```

```
import { createPortal } from 'react-dom';
import {

  ModalContent,
  ModalOverlay,
  CloseBtn,
  ScrollBox,
} from './Modal.styled';
import ImageAnimation3 from 'components/layout/loader/ImageAnimation3';

const modalRoot = document.querySelector('#root-module');

const Modal = ({ children, onClose, pad, heightT = false, loading = 1,nqe = false }) => {
  console.log('loading :>> ', loading);
  const handleContentClick = event => {
    event.stopPropagation();
  };
console.log('nqe :>> ', nqe);
  return createPortal(
    <ModalOverlay onClick={onClose} nqe={nqe?"true":"false"} >
    {    loading===0? <ImageAnimation3/>:
      <ModalContent onClick={handleContentClick} heightT={heightT ? 'true' : 'false'} >
          <ScrollBox pad={pad} nqe={nqe?"true":"false"} >
          {pad !== "false"  && <CloseBtn onClick={onClose} />}

            {children}
          </ScrollBox>
      </ModalContent>}
    </ModalOverlay>,
    modalRoot
  );
};

export default Modal;
```

```
import { Span1, Span2 } from "components/main/AddBag/AddBag.styled";

const NotificationModal =({ onClose, notificationsAr, setNotificationsAr, markUnread, }) => {
  const navigate = useNavigate();
  const [readNotifications, { isLoading: isLoadingMes }] = useReadNotificationsMutation();
  console.log('object :>> ', notificationsAr);

  const handleRead = async () => {
    if (markUnread) return;
    const ids = notificationsAr.filter(item => !item.read).map(item => item.id).join(',');
    console.log('ids :>> ', ids);
    try {
      await readNotifications(ids).unwrap();
      console.log('Details successfully');
      const updatedNotifications = notificationsAr.map(item =>
        item.read ? item : { ...item, read: true }
      );
      setNotificationsAr(updatedNotifications);
    } catch (error) {
      console.error('Failed to Details:', error);
    }
  }; ← #30-44 const handleRead = async () =>

  const handleReadById = async (id,bag_id) => {
    try {
      console.log('id :>> ', id);
      await readNotifications(`${id}`).unwrap();
      console.log('Details successfully');
      const updatedNotifications = notificationsAr.map(item =>
        item.id === id ? { ...item, read: true } : item
      );
      setNotificationsAr(updatedNotifications);
      navigate(`/main/viewbag/${bag_id}`);    "viewbag": Unknown word.

    } catch (error) {
      console.error('Failed to Details:', error);
    }
    onClose();
  }; ← #46-61 const handleReadById = async (id,bag_id) =>

  return (
    <ModalMessage hug={notificationsAr && notificationsAr.length === 0 && true} pad={"false"} onClose={onClose}>
      <ModalHeadBox>
        <ModalTitle>Повідомлення</ModalTitle>    "Повідомлення": Unknown word.
        {isLoadingMes ? (
          <ImageAnimation2 />
        ) : notificationsAr && notificationsAr.length !== 0 ? (
          <ModalHeadBtn onClick={handleRead}>
            <IconSend />
            <Span1>Прочитати всі</Span1>    "Прочитати": Unknown word.
          </ModalHeadBtn>
        ) : (
          <Span2>немає</Span2>    "немає": Unknown word.
        )}</ModalHeadBox> ← #67-76 <ModalTitle>Повідомлення</ModalTitle>
      {notificationsAr && notificationsAr.length !== 0 && <NotifyModalList>
        {notificationsAr.map((notify) =>
          <NotifyModalItem key={notify.id} onClick={() => {
            handleReadById(notify.id, notify.object_id)
          }
          } read={notify.read ? "true" : "false"}>
            <IconSbag />    "Sbag": Unknown word.
            <NotifyModalInfoBox>
              <NotifyModalTitle>{notify.title}</NotifyModalTitle>
              <NotifyModalText>{notify.message}</NotifyModalText>
              <NotifyModalTime>{formatTimeDate(notify.createdAt)}</NotifyModalTime>
            </NotifyModalInfoBox>
          </NotifyModalItem>
        )} ← #78-90 notificationsAr && notificationsAr.length !== 0 && <NotifyMod...
      </NotifyModalList>} ← #77-91 </ModalHeadBox>
    </ModalMessage>
  ); ← #63-93 return
}; ← #25-94 const NotificationModal =({ onClose, notificationsAr, setNoti...

export default NotificationModal;
```

Код компонентівсерверної частини платформи E-Market.

```php
<?php


namespace app\commands;


use app\models\EmailTasks;
use Exception;
use Yii;
use yii\console\Controller;
use yii\helpers\ArrayHelper;
use yii\helpers\Html;
use yii\helpers\Url;

0 references | 0 implementations
class MailController extends Controller    Use of unknown class: 'yii\console\Controller'
{

    0 references | 0 overrides
    public function actionIndex(): void
    {
        $tasks = EmailTasks::find()->where(['process' => '0'])->limit(30)->all();    Call to unknown method: app\m
        $ids = ArrayHelper::map($tasks, 'id', 'id');    Use of unknown class: 'yii\helpers\ArrayHelper'
        EmailTasks::updateAll(['process' => '1'], ['id' => $ids]);    Call to unknown method: app\models\EmailTask

        foreach ($tasks as $task) {
            try {
                $this->send(task: $task);
            } catch (Exception $e) {
                Yii::error('Error while sending email');    Use of unknown class: 'Yii'
                echo $e->getMessage() . "\n";
                $task->updateAttributes(['process' => '0']);
            } ← #27-31 catch (Exception $e)
        } ← #24-32 foreach ($tasks as $task)
    } ← #19-33 public function actionIndex()

    1 reference
    private function send(EmailTasks $task): void
    {
        $sended = mailer()->compose('common', ['title' => $task->subject, 'message' => $task->message])
            ->setTo($task->email)
            ->setFrom(app()->params['adminEmail'])
            ->setSubject($task->subject ?: app()->name)
//            ->setHtmlBody($task->message)
            ->send();
        $task->updateAttributes(['process' => EmailTasks::STATUS_DONE]);    Call to unknown method: app\models\Ema
        if (!$sended) {
            $task->updateAttributes(['process' => EmailTasks::STATUS_NEW]);    Call to unknown method: app\models\
        }
    } ← #36-47 private function send(EmailTasks $task)
} ← #16-48 class MailController extends Controller
```

```php
<?php

namespace app\controllers;

use app\filters\HttpBearerAuth;
use app\models\Bags;
use app\models\Categories;
use app\models\search\Bags as BagsSearch;
use Exception;
use yii\data\ActiveDataProvider;
use yii\db\Expression;
use yii\filters\AccessControl;

0 references | 0 implementations
class DashboardController extends BaseController
{

    0 references | 0 overrides
    public function verbs(): array
    {
        return [
            'index' => ['GET'],
        ];
    }

    26 references | 0 overrides | prototype
    public function behaviors(): array
    {
        return array_merge(arrays: parent::behaviors(), [
            'bearerAuth' => [
                'class' => HttpBearerAuth::class,    Use of unknown class: 'app\filters\HttpBearerAuth'
                'except' => [],
            ],
            'access' => [
                'class' => AccessControl::class,    Use of unknown class: 'yii\filters\AccessControl'
                'rules' => [
                    [
                        'actions' => ['index'],
                        'allow' => true,
                        'roles' => ['client', 'guest'],
                    ],
                ],
            ],
        ]);
    }

    /**
     * @SWG\Get(path="/dashboard",
     *     tags={"[Mobile]Bags"},
     *     summary="Retrieve Dashboard bags data",
     *     @SWG\Parameter(
     *         name="lat",
     *         in="query",
     *         type="number",
     *         description="Latitude of user point"
     *     ),
     *     @SWG\Parameter(
     *         name="lng",
     *         in="query",
     *         type="number",
     *         description="Longitude of user point"
     *     ),
     *     @SWG\Parameter(
     *         name="radius",
     *         in="query",
     *         type="number",
     *         description="Search radius"
     *     ),
     *     @SWG\Parameter(
     *         name="mode",
     *         in="query",
     *         type="string",
     *         description="Search mode. Possible values - 'recommendations', 'fast', 'getToday', 'getTomorrow', 'soldOut', 'category_[category_id]'"
     *     ),
     *     @SWG\Parameter(
     *         name="page",
     *         in="query",
     *         type="integer",
     *         description="Page number"
     *     ),
     *     @SWG\Response(
```

Use of unknown class: 'yii\filters\AccessControl' PHP(PHP0413)
View Problem (Alt+F8) No quick fixes available

```php
<?php

namespace app\controllers;

use app\filters\HttpBearerAuth;
use app\models\Merchants;
use app\models\search\Merchants as MerchantsSearch;
use yii\filters\AccessControl;
use yii\web\NotFoundHttpException;

0 references | 0 implementations
class MerchantsController extends BaseController
{


    0 references | 0 overrides
    public function verbs(): array
    {
        return [
            'all' => ['GET'],
            'view' => ['GET'],
        ];
    }

    26 references | 0 overrides | prototype
    public function behaviors(): array
    {
        return array_merge(arrays: parent::behaviors(), [
            'bearerAuth' => [
                'class' => HttpBearerAuth::class,    Use of unknown class: 'app\filters\HttpBearerAuth'
                'except' => [],
            ],
            'access' => [
                'class' => AccessControl::class,    Use of unknown class: 'yii\filters\AccessControl'
                'rules' => [
                    [
                        'allow' => true,
                        'roles' => ['admin'],
                        'actions' => ['all', 'view'],
                    ],
                ],
            ],
        ]);
    }

    /**
     * @SWG\Get(path="/merchants/all",
     *     tags={"[Admin]Merchants"},
     *     summary="Retrieve list of merchants",
     *     @SWG\Parameter(
     *         name="page",
     *         in="query",
     *         type="integer",
     *         description="Page number"
     *     ),
     *     @SWG\Response(
     *         response = 200,
     *         description = "Returned list",
     *         @SWG\Schema (ref="#/definitions/Merchants")
     *     ),
     * )
     */
    0 references | 0 overrides
    public function actionAll(): ActiveDataProvider|array
    {
        $searchModel = new MerchantsSearch([]);    Class 'app\models\search\Merchants' does not have any constructor and shall be ca
        $dataProvider = $searchModel->search(params: get() + post());
        if (!$dataProvider) {
            return $this->notOk(message: t(category: 'Невірні параметри пошуку'), statusCode: 422, details: $searchModel->errors);
        }
        return $dataProvider;
    }


    /**
     * @SWG\Get(path="/merchants/view?id=[id]",
     *     tags={"[Admin]Merchants"},
     *     summary="Retrieve merchant details",
     *     @SWG\Parameter(
     *         name="id",
     *         in="query",
```

```
1 reference | 0 overrides
public function actionRemindIsStarting($limit = 100): void
{
    $datetime = create_date(string: date(format: 'Y-m-d H:i:s'), format: 'Y-m-d H:i:s')->modify(modifier: '+1 minute')->format(format: 'Y-m-d H:i');
    $from = $datetime . ":00";
    $to = $datetime . ":59";
    $ids = Bags::find()    Call to unknown method: app\models\Bags::find()
        ->select(['id'])
        ->andWhere(['status' => 'active'])
        ->andWhere(['between', 'datetime_from', $from, $to])
        ->asArray()
        ->all();
    echo "Foreach started\n";
    foreach (array_chunk(array: $ids, length: $limit) as $arr) {
        foreach ($arr as $id) {
            $model = Bags::findOne(['id' => $id['id']]);    Call to unknown method: app\models\Bags::findOne()
            formatter()->timeZone = $model->merchant->user->timezone;
            sendMessage(
                user_id: $model->merchant->user_id,
                title: t(category: 'Розпочався час видачі кошика'),    "Розпочався": Unknown word.
                message: t(category: 'Нагадуємо, що о {time} розпочинається час отримання для Вашого диво-кошика "{bagName}" в точці видачі {address}.',
                    'time' => formatter()->asTime(strtotime(datetime: $model->datetime_from)),
                    'bagName' => $model->name,
                    'address' => $model->shop->address,
                ]), ← #90-94 t
                type: 'bag',
                id: $model->id
            ); ← #87-97 sendMessage
            formatter()->timeZone = 'UTC';
        } ← #84-99 foreach ($arr as $id)
    } ← #83-100 foreach (array_chunk($ids, $limit) as $arr)

    echo "Foreach ended\n";
} ← #72-103 public function actionRemindIsStarting($limit = 100)
```

```
2 references | 0 overrides
public function actionWork($from = null, $limit = 100): void
{
    if (is_null(value: $from)) {
        $from = date(format: 'Y-m-d');
    }
    $tomorrow = date(format: 'Y-m-d', timestamp: strtotime(datetime: $from . ' 00:00:00') + (86400 + 86399));
    $ids = Schedules::find()    Call to unknown method: app\models\Schedules::find()
        ->select(['schedules.id'])
        ->asArray()
        ->leftJoin('bags', 'bags.id = schedules.bag_id')
        ->andWhere(['bags.status' => 'scheduling'])
        ->andWhere([
            'or',
            'last_works is null',
            ['<', 'last_works', $tomorrow],
        ]) ← #116-120 ->andWhere
        ->andWhere(['⩽', 'calculated_start', $tomorrow])
        ->andWhere([
            'or',
            'calculated_end is null',
            ['⩾', 'calculated_end', $tomorrow],
        ]) ← #122-126 ->andWhere
        ->all();
    foreach (array_chunk(array: $ids, length: $limit) as $chunk) {
        foreach (Schedules::find()->where(['id' => map(array: $chunk, from: 'id', to: 'id')])->all() as $schedule) {    Call to unknown method: app\mode
            try {
                if ($schedule->attemptStart($from)) {
                    if (YII_DEBUG && (YII_ENV == 'dev')) {    Use of undefined constant 'YII_DEBUG'
                        echo "Published: {$schedule->id}\n";
                    }
                } else {
                    if (YII_DEBUG && (YII_ENV == 'dev')) {    Use of undefined constant 'YII_DEBUG'
                        echo "# $schedule->id Errors:\n" . implode(separator: "\n", array: $schedule->internalErrors) . "\n";
                    } else {
                        Yii::error('Schedule error: ' . $schedule->id, 'schedule_error');    Use of unknown class: 'Yii'
                    }
                } ← #135-141 else
            } catch (Exception $e) {

            }
        } ← #129-145 foreach (Schedules::find()->where(['id' => map($chunk, 'id', ...
    } ← #128-146 foreach (array_chunk($ids, $limit) as $chunk)
} ← #106-147 public function actionWork($from = null, $limit = 100)
```

```php
0 references | 0 implementations
class StatisticsController extends BaseController
{

    0 references | 0 overrides
    public function verbs(): array
    {
        return [
            'general' => ['GET'],
        ];
    }

    26 references | 0 overrides | prototype
    public function behaviors(): array
    {
        return array_merge(arrays: parent::behaviors(), [
            'bearerAuth' => [
                'class' => HttpBearerAuth::class,    Use of unknown class: 'app\filters\HttpBearerAuth'
                'except' => [],
            ],
            'access' => [
                'class' => AccessControl::class,    Use of unknown class: 'yii\filters\AccessControl'
                'rules' => [
                    [
                        'actions' => ['general', 'rating', 'cancelled', 'reverted', 'feedbacks', 'comments'],
                        'allow' => true,
                        'roles' => ['merchant'],
                    ],
                ],
            ],
        ]);
    }

    /**
     * @SWG\Get(path="/statistics/general?period={period}",
     *     tags={"[Merchant]Statistics"},
     *     summary="General merchant statistics",
     *     @SWG\Parameter(
     *         name="period",
     *         in="query",
     *         type="string",
     *         description="Period. Not required. Must be like '30days', '12weeks', '12months' etc"
     *     ),
     *     @SWG\Response(
     *         response = 200,
     *         description = "Returns statistics",
     *         @SWG\Schema(ref = "#/definitions/GeneralStatistics")
     *     ),
     * )
     */
    0 references | 0 overrides
    public function actionGeneral($period = null): array
    {
        $ordersQuery = Orders::find()
            ->byMerchantId(identity()->current_merchant_id)    Undefined property: Users|IdentityInterface::$current_merchant_id
            ->andWhere(['status' => ['done', 'ended']]);
        $favouritesQuery = FavouritesStatistics::find()->where(['merchant_id' => identity()->current_merchant_id]);    Call to unknown me
        $viewsQuery = ViewsStatistics::find()->where(['merchant_id' => identity()->current_merchant_id]);    Call to unknown method: app\

        // Filtering
        if ($period) {
            $to = create_date(string: date(format: 'Y-m-d 23:59:59'), format: 'Y-m-d H:i:s');
            try {
                $from = create_date(string: date(format: 'Y-m-d 00:00:00'), format: 'Y-m-d H:i:s')->modify(modifier: "-{$period}");
            } catch (Exception $e) {
                if (YII_DEBUG && (YII_ENV == 'dev')) {    Use of undefined constant 'YII_DEBUG'
                    throw $e;
                }
                return $this->notOk(message: t(category: 'Невірний формат періоду'), statusCode: 422, details: $e->getMessage());    "Нев
            }
            $ordersQuery->inDates($from, $to);
            $favouritesQuery->andWhere([    "favourites": Unknown word.
                'and',
                ['>=', 'date', $from->format(format: 'Y-m-d')],
                ['<=', 'date', $to->format(format: 'Y-m-d')],
            ]);
            $viewsQuery->andWhere([
                'and',
                ['>=', 'date', $from->format(format: 'Y-m-d')],
```

```php
 */
10 references | 0 implementations
class Addresses extends ActiveRecord    Use of unknown class: 'yii\db\ActiveRecord'
{
    /**
     * {@inheritdoc}
     */
    0 references | 0 overrides
    public static function tableName(): string
    {
        return 'user_addresses';
    }

    /**
     * {@inheritdoc}
     */
    0 references | 0 overrides
    public function rules(): array
    {
        return [
            [['name', 'latitude', 'longitude', 'address'], 'required'],
            [['radius'], 'number'],
            [['order', 'user_id'], 'integer'],
            [['name', 'address'], 'string', 'max' => 255],
            [['latitude', 'longitude'], 'number', 'integerOnly' => false],
            [['name'], 'in', 'range' => ['Обрана на карті', 'Дім', 'Робота', 'Інше']],    "Обрана": Unknown wor
        ];
    }

    0 references | 0 overrides
    public function fields(): array
    {
        return [
            'id',
            'name',
            'latitude',
            'longitude',
            'address',
            'order',
            'radius',
        ];
    }

    /**
     * {@inheritdoc}
     */
    0 references | 0 overrides
    public function attributeLabels(): array
    {
        return [
            'name' => t(category: 'Назва'),    "Назва": Unknown word.
            'latitude' => t(category: 'Довгота'),    "Довгота": Unknown word.
            'longitude' => t(category: 'Широта'),    "Широта": Unknown word.
            'radius' => t(category: 'Радіус'),    "Радіус": Unknown word.
            'order' => t(category: 'Порядок'),    "Порядок": Unknown word.
            'address' => t(category: 'Адреса'),    "Адреса": Unknown word.
        ];
    }

    0 references | 0 overrides
    public function getUser(): mixed
    {
        return $this->hasOne(Users::class, ['id' => 'user_id']);
    }
}
```