ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика покращення продуктивності web-застосунку підтримки клієнтів біржі»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення

освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Микола РУДКОВСЬКИЙ

Виконав:
здобувач вищої освіти Микола РУДКОВСЬКИЙ
група ПДМ-63

Керівник: Світлана ШЕВЧЕНКО
к.п.н.,доцент

Рецензент:
_____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедрою
Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Рудковському Миколі Олеговичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Методика покращення продуктивності web-застосунку підтримки клієнтів біржі»

керівник кваліфікаційної роботи Світлана ШЕВЧЕНКО к.п.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з питань, пов'язаних покращення продуктивності web-застосунку, програмні засоби розробки web-застосунку підтримки клієнтів біржі.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз продуктивності web-застосунку.

Аналіз клієнтоорієнтованості користувачів біржі.
Розробка методики покращення продуктивності web-застосунку підтримки клієнтів біржі.
Тестування розробленої методики у вигляді web-застосунку підтримки клієнтів біржі.

5. Перелік графічного матеріалу: *презентація*
   1. Мета, об’єкт та предмет дослідження.
   2. Аналіз та порівнянн методики роботи чат-ботів бірж.
   3. Математична модель методики підтримки клієнтів біржи.
   4. Діаграми класів збереження даних.
   5. Діаграма діяльності.
   6. Програмні засоби реалізації.
   7. Відеодемонстрація роботи чату підтримки.
   8. Практичні результати покращення продуктивності.
   9. Висновки.
   10. Публікації та апробація роботи.

6. Дата видачі завдання«16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури.	19.10-05.11.24	
2	Аналіз продуктивності web-застосунку	05.11-12.11.24	
3	Аналіз клієнтоорієнтованості користувачів біржі	13.11-19.11.24	
4	Дослідження та покращення продуктивності web-застосунку підтримки клієнтів біржі.	20.11-25.11.24	
5	Основні розділи.	27.11-03.12.24	
6	Розробка обов’язкових матеріалів.	04.12-10.12.24	
7	Попередній захист роботи.	11.12-20.12.24	
8	Пред'явлення роботи в деканат.	16.12-21.12.24	

Здобувач вищої освіти _____ Микола РУДКОВСЬКИЙ

Керівник
кваліфікаційної роботи _____ Світлана ШЕВЧЕНКО

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 86 стор., 5 табл., 22 рис., 32 джерел.

Мета роботи – підтримка клієнтів біржі за рахунок створення web-застосунку.

Об'єкт дослідження – процеси функціонування web-застосунків підтримки клієнтів у середовищі біржових платформ.

Предмет дослідження – методика покращення продуктивності web-застосунку.

Короткий зміст роботи: робота присвячена розробці та впровадженню методики покращення продуктивності web-застосунку підтримки клієнтів біржі. Для забезпечення швидкості, надійності та масштабованості системи в умовах високого навантаження у дослідженні розглянуто сучасні технології оптимізації, такі як кешування, балансування навантаження, оптимізація баз даних, використання CDN та хмарних сервісів. Робота також включає аналіз вузьких місць, тестування продуктивності та практичну реалізацію розроблених методів із подальшою оцінкою ефективності.

КЛЮЧОВІ СЛОВА: ПРОДУКТИВНІСТЬ WEB-ЗАСТОСУНКІВ, ВЕБ-ТЕХНОЛОГІЇ, СЕРВЕРНА АРХІТЕКТУРА, АСИМЕТРИЧНА МАРШРУТИЗАЦІЯ, МАСШТАБОВАНІСТЬ СИСТЕМИ, БІРЖОВІ ПЛАТФОРМИ, БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

ABSTRACT

Text of the qualification work for obtaining a master's degree: 86 pages, 5 table, 22 figures, 32 references.

Purpose of the work – to support exchange clients by ensuring the speed, reliability, and scalability of the system under high load conditions through the development of a web application.

Object of research – the processes of functioning of web applications for client support within exchange platforms.

Subject of research – the methodology for improving the performance of a web application, including the optimization of server-side and client-side components, as well as the use of modern architectural solutions and technologies.

Summary of the work: the study focuses on the development and implementation of a methodology for enhancing the performance of a web application supporting exchange clients. The main goal is to ensure system speed, reliability, and scalability under high load conditions. The research explores modern optimization technologies such as caching, load balancing, database optimization, the use of CDNs, and cloud services. The work also includes bottleneck analysis, performance testing, and practical implementation of the developed methods with subsequent evaluation of their effectiveness.

KEYWORDS: WEB APPLICATION PERFORMANCE, WEB TECHNOLOGIES, SERVER ARCHITECTURE, ASYMMETRIC ROUTING, SYSTEM SCALABILITY, EXCHANGE PLATFORMS, LOAD BALANCING

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8
ВСТУП 9
1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРОДУКТИВНОСТІ WEB-ЗАСТОСУНКІВ 12
1.1 Сучасні тенденції у розробці web-застосунків для біржових платформ 12
1.2 Характеристика продуктивності: фактори впливу та критерії оцінки 24
1.3 Огляд технологій і підходів для підвищення продуктивності web-застосунків 27
2 ДОСЛІДЖЕННЯ ПРОБЛЕМ ТА РОЗРОБКА МЕТОДИКИ ОПТИМІЗАЦІЇ 32
2.1 Виявлення вузьких місць у роботі web-застосунків підтримки клієнтів бірж . 32
2.2 Обґрунтування вибору методів і технологій для покращення продуктивності 35
2.3 Розробка методики оптимізації web-застосунку 44
3 РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОЇ МЕТОДИКИ 49
3.1 Опис реалізації програми, включаючи вибір платформи та технологій, розробка архітектури додатку. 49
3.2 Опис функціонування web-додатку 75
ВИСНОВКИ 85
ПЕРЕЛІК ПОСИЛАНЬ 87
Додаток А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

UI/UX – User Interface/User Experience

HTTP – Hypertext Transfer Protocol

CDN – Content Delivery Network

SQL – Structured Query Language

SPA – Single Page Application

REST – Representational State Transfer

JSON – JavaScript Object Notation

QoS – Quality of Service

ВСТУП

У сучасному світі стрімкого розвитку інформаційних технологій web-застосунки стали невід'ємною складовою бізнес-процесів у різних галузях, зокрема й у сфері біржових платформ. Ефективна робота таких застосунків визначає швидкість, надійність і якість обслуговування клієнтів, що особливо важливо в умовах високої конкуренції та великих обсягів даних. Водночас зростаючі вимоги до продуктивності та масштабованості систем висувають перед розробниками нові виклики, які потребують комплексних рішень для оптимізації як технічних, так і архітектурних аспектів web-застосунків.

В умовах цифровізації бізнес-процесів та зростаючого обсягу інформації, що обробляється в режимі реального часу, продуктивність web-застосунків відіграє ключову роль у забезпеченні високої якості обслуговування клієнтів. Для біржових платформ, де швидкість обробки запитів та надійність системи мають критичне значення, неефективна робота web-застосунків може призводити до фінансових втрат, зниження конкурентоспроможності та негативного впливу на репутацію компанії.

Високий попит на швидкі та надійні рішення, здатні витримувати пікові навантаження, вимагає впровадження інноваційних підходів до оптимізації як серверної, так і клієнтської частини застосунку. У той же час розвиток технологій, таких як хмарні обчислення, балансування навантаження, кешування даних і застосування асинхронних алгоритмів, відкриває нові можливості для підвищення ефективності роботи систем підтримки клієнтів.

Проблема продуктивності web-застосунків у контексті біржових платформ має наукову й практичну цінність, оскільки забезпечення стабільної роботи таких систем є необхідним для оперативного реагування на запити клієнтів, швидкої обробки даних і підтримки високих стандартів сервісу.

Таким чином, розробка методики покращення продуктивності web-застосунку підтримки клієнтів біржі є актуальним напрямком досліджень, який сприяє

підвищенню ефективності функціонування біржових платформ, адаптації до зростаючих вимог ринку та покращенню взаємодії з клієнтами.

Мета роботи – підтримка клієнтів біржі за рахунок створення web-застосунку.

Об'єкт дослідження – процеси функціонування web-застосунків підтримки клієнтів у середовищі біржових платформ.

Предмет дослідження – методика покращення продуктивності web-застосунку.

Для реалізації вказаної мети необхідно реалізувати ряд *завдань:*

1. Провести аналіз сучасних підходів і технологій для забезпечення продуктивності web-застосунків.

2. Дослідити архітектурні особливості систем підтримки клієнтів біржових платформ.

3. Визначити основні проблеми, які впливають на швидкість та надійність роботи web-застосунків.

4. Розробити методику оптимізації web-застосунку з урахуванням специфіки біржових платформ.

5. Реалізувати запропоновану методику на практичному прикладі web-застосунку підтримки клієнтів біржі.

6. Провести тестування продуктивності застосунку до та після впровадження методики для оцінки її ефективності.

Для написання роботи використано такі методи дослідження**:** аналіз даних, порівняльний, системно-структурний, методи математичного моделювання, методи програмної інженерії.

Наукова новизна одержаних результатів. Розроблено математичну модель методики підтримки клієнтів біржі, яка включає алгоритми оптимізації обробки запитів, розподілу ресурсів та прогнозування навантаження.

*Практичне значення одержаних результатів***.** Розроблено web-застосунок для підтримки клієнтів біржі, який забезпечує стабільну обробку запитів у режимі реального часу, показує високу швидкість виконання операцій на 10%.

Апробація результатів магістерської роботи.

1. 1. Рудковський М.О., Шевченко С. М., Покращення продуктивності web-застосунку підтримки клієнтів біржі // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 24.11.2024, ДУІКТ, м. Київ. – подано до друку

2. Рудковський М.О., Шевченко С. М. Елементи методики підтримки клієнтів біржі // ІІ міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії», 25-27 грудня 2024, ДУІКТ, м. Київ. – подано до друку

1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРОДУКТИВНОСТІ WEB-ЗАСТОСУНКІВ

Web-застосунки є ключовим інструментом взаємодії між біржовими платформами та їхніми клієнтами, забезпечуючи доступ до фінансових даних, оперативність укладення угод і високий рівень сервісу. Проте зростання обсягів даних і кількості користувачів, які працюють у режимі реального часу, ставить перед розробниками серйозні виклики у сфері продуктивності таких систем.

1.1 Сучасні тенденції у розробці web-застосунків для біржових платформ

Web-застосунок – це програмне забезпечення, яке працює у веб-браузері та взаємодіє з користувачем через інтернет. На відміну від традиційних програм, які встановлюються локально на пристрій, веб-застосунки функціонують на сервері, а користувач отримує доступ до них через інтерфейс браузера [1]. Такі застосунки можуть мати різний функціонал, включаючи обробку даних, взаємодію з базами даних, виконання бізнес-логіки. Завдання web-застосунків біржових платформ наведені в таблиці 1.1.

Web-застосунки відіграють критичну роль у функціонуванні біржових платформ, оскільки саме вони забезпечують взаємодію між користувачами (інвесторами, трейдерами) і біржовою інфраструктурою. Основні аспекти цієї ролі включають:

1. Забезпечення доступу до фінансової інформації:

– Web-застосунки надають клієнтам доступ до біржових котирувань, графіків, звітів, аналітики та інших важливих даних.

– Забезпечують оновлення інформації в реальному часі, що є критичним для прийняття швидких рішень у динамічному середовищі.

– Використання технологій, таких як WebSocket або REST API, дозволяє мінімізувати затримки в передачі даних.

2. Обробка запитів клієнтів:

– Web-застосунки дозволяють користувачам подавати торгові запити, наприклад, купувати чи продавати активи.

– Вони відповідають за обробку транзакцій, включаючи перевірку даних, підтвердження запитів і передачу інформації до серверів біржі.

– Забезпечують автоматизацію процесів, знижуючи потребу в ручній обробці запитів [1].

3. Захист даних і забезпечення безпеки:

– Біржові web-застосунки працюють із конфіденційною інформацією, включаючи особисті дані клієнтів, їхні фінансові активи та торгові стратегії.

– Застосунки впроваджують засоби захисту, такі як:

– Шифрування (SSL/TLS) для передачі даних.

– Двофакторна автентифікація для доступу до системи.

– Захист від атак, наприклад, DDoS або SQL-ін’єкцій.

– Безпека є основою довіри клієнтів до біржі.

4. Зручність користувацького досвіду (UX):

– Web-застосунки надають інтуїтивно зрозумілий інтерфейс для взаємодії з платформою, дозволяючи користувачам швидко орієнтуватися в системі та виконувати торгові операції.

– Підтримують багатоплатформеність, дозволяючи доступ через комп’ютери, смартфони та планшети.

5. Масштабованість і гнучкість:

– Біржові web-застосунки здатні обслуговувати велику кількість користувачів одночасно завдяки масштабованим архітектурним рішенням.

– Вони легко адаптуються до змін у бізнес-вимогах та впроваджують нові функції (наприклад, підтримка нових видів активів).

Таблиця 1.1

Опис завдань web-застосунків біржових платформ

Завдання	**Опис**	**Приклади реалізації**
Обробка клієнтських запитів	Прийом, перевірка та виконання торгових операцій, надання звітів про виконання угод.	Виконання операцій купівлі-продажу акцій, обробка ордерів.
Забезпечення доступу до інформації	Надання клієнтам актуальних біржових даних, котирувань і новин у реальному часі.	Відображення динамічних графіків і біржових котирувань.
Розрахунок і аналітика	Проведення автоматичних розрахунків, аналіз трендів і побудова прогнозів.	Вбудовані інструменти технічного аналізу, калькулятори комісій.
Захист даних	Захист конфіденційної інформації клієнтів, попередження атак і збереження цілісності даних.	Використання SSL/TLS, двофакторна автентифікація.
Забезпечення швидкодії	Мінімізація часу обробки запитів і завантаження інтерфейсу для користувачів.	Використання CDN, кешування даних, балансування навантаження.
Зручність користувацького досвіду	Забезпечення інтуїтивного інтерфейсу для роботи з платформою.	Адаптивний дизайн, прості інструменти для навігації.
Підтримка масштабованості	Стабільна робота системи навіть під час пікових навантажень.	Використання хмарних обчислень і мікросервісної архітектури.
Автоматизація бізнес-процесів	Спрощення складних операцій і мінімізація втручання людини у процеси.	Роботизація обробки великих масивів даних і звітності.

Тенденції розвитку технологій:

1. Хмарні обчислення (Cloud Computing)

Хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud, надають потужні ресурси для зберігання даних, обчислювальних потужностей та масштабованості. Веб-застосунки, розроблені для хмари, можуть адаптуватися до змінних навантажень, що дає змогу зменшити витрати на інфраструктуру та швидко реагувати на зростання потреб користувачів [2].

Підходи:

– Використання Serverless архітектур, де ресурси автоматично виділяються лише при потребі.

– Edge Computing для покращення швидкості обробки даних, розподілених по географії.

– Інтеграція з хмарними базами даних, наприклад, Cloud SQL або NoSQL базами даних.

2. Мікросервісна архітектура (Microservices Architecture)

Мікросервіси дозволяють розділяти веб-застосунки на незалежні компоненти, кожен з яких має свою відповідальність і може розвиватися та масштабуватися окремо.

Переваги:

– Знижує складність розробки, дозволяючи різним командам працювати над різними частинами програми одночасно.

– Легко масштабувати певні частини системи без необхідності масштабувати увесь застосунок.

– Можливість використання різних технологій для різних мікросервісів.

Технології:

– Docker та Kubernetes для контейнеризації та оркестрації мікросервісів.

– Використання API Gateway для маршрутизації запитів до різних мікросервісів.

3. DevOps і CI/CD (Continuous Integration/Continuous Deployment)

DevOps передбачає інтеграцію процесів розробки та операційного управління для швидшого та надійнішого запуску продуктів. CI/CD дозволяють автоматизувати процеси тестування, збору та розгортання програмного забезпечення [3].

Особливості:

– Continuous Integration дозволяє розробникам часто інтегрувати зміни в основну гілку, що дає змогу швидше знаходити помилки та виправляти їх.

– Continuous Deployment дозволяє автоматично деплоїти зміни на продакшн-середовище, зменшуючи час від розробки до релізу.

Інструменти:

– Jenkins, GitLab CI, CircleCI для автоматизації процесів CI/CD.

– Terraform для управління інфраструктурою як кодом.

4. Адаптивний дизайн і реактивні інтерфейси

Сучасні веб-застосунки часто використовують реактивні фреймворки (як-от React, Vue.js, Angular) для створення динамічних інтерфейсів. Адаптивний дизайн дозволяє застосункам підлаштовуватися під різні пристрої (мобільні телефони, планшети, десктопи).

Особливості:

– Інтерфейси, що змінюються на основі введення користувача або зовнішніх даних.

– Використання Single Page Applications (SPA) для покращення досвіду користувачів.

5. Безпека як частина процесу розробки (Security by Design)

Враховуючи зростаючі загрози безпеці, багато організацій впроваджують безпеку на етапі проектування застосунків, використовуючи принципи Zero Trust, encryption на всіх рівнях і двухфакторну аутентифікацію (2FA).

– Інструменти для сканування вразливостей: OWASP ZAP, Burp Suite.

– Використання WAF (Web Application Firewall) для захисту від атак на веб-застосунки.

6. Штучний інтелект та машинне навчання

Веб-застосунки все більше інтегрують штучний інтелект (AI) та машинне навчання (ML) для персоналізації контенту, аналізу даних та автоматизації рутинних завдань.

AI в веб-застосунках:

– Рекомендаційні системи.

– Чат-боти та віртуальні помічники.

– Розпізнавання зображень і тексту.

Зростання кількості клієнтів, транзакцій та обсягу даних безпосередньо впливає на вимоги до швидкодії, масштабованості та надійності веб-застосунків. Ось як це може позначатися на їхніх характеристиках:

1. Швидкодія (Performance)

Вплив зростання кількості клієнтів:

– Завантаження серверів, кількість одночасних користувачів може призвести до підвищеного навантаження на сервери, що знижує швидкість відповіді та загальну продуктивність веб-застосунку.

– Проблеми з затримками, коли велика кількість користувачів звертається до системи одночасно, зростає час відгуку. Це може впливати на досвід користувачів, особливо у високонавантажених застосунках.

– Оптимізація ресурсів, для підтримки високої швидкодії потрібно оптимізувати код і використовувати кешування для зменшення навантаження на сервери (наприклад, кешування на рівні браузера, серверне кешування через CDN або Redis).

Вплив зростання транзакцій:

– Час обробки, кількість транзакцій може збільшити час їх обробки, що веде до затримок у відповідях, особливо якщо система не має достатньої обчислювальної потужності.

– База даних, інтенсивні запити до бази даних можуть створювати затримки, що вимагає оптимізації індексів, використання масштабованих рішень для баз даних або навіть переходу до NoSQL баз даних, якщо цього вимагає характер даних.

– Вплив зростання обсягу даних:

– Збільшення часу на обробку запитів, більше даних для обробки може збільшити час відгуку при пошуку та фільтрації інформації. Важливо використовувати відповідні індекси та структури даних для швидкої обробки великих масивів даних.

– Оптимізація зберігання, для роботи з великими обсягами даних можуть знадобитися рішення для архівування, шардінгу або кластеризації баз даних, щоб забезпечити високий рівень швидкодії.

2. Масштабованість (Scalability)

Вплив зростання кількості клієнтів:

– Горизонтальне масштабування, для обробки зростаючої кількості користувачів часто застосовують горизонтальне масштабування (додавання нових серверів або інстансів додатків) замість вертикального масштабування (додавання потужностей на одному сервері).

– Мікросервісна архітектура, коли система досягає великої складності, мікросервіси дозволяють масштабувати окремі частини системи, які найбільше навантажені, без необхідності масштабувати всю систему.

– Вплив зростання транзакцій:

– Масштабованість транзакцій, для обробки великої кількості транзакцій важливо, щоб система могла масштабуватися під високі навантаження, використовуючи кластеризацію серверів або баз даних, а також забезпечити розподілену обробку запитів (наприклад, через Kafka або RabbitMQ для асинхронної обробки).

– Транзакційна консистентність, при високому навантаженні важливо враховувати проблему консистентності даних. Для цього часто використовуються стратегія Event Sourcing або CQRS.

Вплив зростання обсягу даних:

– Масштабоване зберігання даних, для обробки великих обсягів даних важливо використовувати рішення, які дозволяють автоматично збільшувати обсяг

пам'яті або обчислювальні ресурси, такі як хмарні сервіси, що підтримують автоматичне масштабування [4].

– Розподілене зберігання, використання distributed file systems (як-от HDFS, Ceph) або object storage (наприклад, AWS S3) дозволяє зберігати величезні обсяги даних і забезпечувати їх швидкий доступ.

3. Надійність (Reliability)

Вплив зростання кількості клієнтів:

– З ростом кількості користувачів збільшується ймовірність виникнення відмов у системі. Тому необхідно реалізовувати механізми резервного копіювання, відновлення та автоматичного переключення на резервні інстанси при виході з ладу основних.

– Розподілені системи, веб-застосунки мають використовувати розподілені системи, які здатні безперебійно працювати навіть у разі виходу з ладу окремих компонентів.

Вплив зростання транзакцій:

– Цілісність даних, збільшення кількості транзакцій вимагає забезпечення їх консистентності та точності. Це може бути досягнуто через механізми, такі як ACID (атомарність, консистентність, ізоляція, довговічність) для реляційних баз даних або Eventual Consistency для розподілених систем.

– Резервування та реплікація, для забезпечення високої доступності і зниження ризику втрати даних використовуються механізми реплікації та резервного копіювання, що дозволяють підтримувати цілісність даних навіть при високих навантаженнях.

Вплив зростання обсягу даних:

– Збереження даних та їх цілісність, потрібно ретельно продумати стратегії збереження та резервування великих обсягів даних, а також забезпечити їх швидкий доступ при високому навантаженні.

– Моніторинг і аналіз, для гарантування надійності важливо впровадити ефективний моніторинг, що дозволяє виявляти аномалії, збої або проблеми в роботі системи до того, як вони призведуть до серйозних відмов.

Приклади сучасних веб-застосунків у криптовалютних біржах, таких як OKX, BitGet, BingX та Binance, демонструють інноваційні підходи до обробки великих обсягів даних, високих навантажень та забезпечення надійності. Ці біржі використовують передові технології для забезпечення швидкої та безпечної торгівлі, а також пропонують користувачам різноманітні функції, такі як кредитне плече, соціальна торгівля, стейкінг та інші. Незважаючи на схожість у кількості криптовалют та торгових пар, кожна з цих бірж має свої унікальні переваги, що дозволяє їм задовольняти різні потреби користувачів, від новачків до професійних трейдерів [5].

OKX – одна з провідних криптовалютних бірж (рис.1.1), заснована у 2017 році. Вона пропонує спотову та ф'ючерсну торгівлю, а також P2P-платформу. OKX підтримує понад 348 криптовалют та 597 торгових пар. Комісії на спотовій торгівлі становлять до 0,1%, а на ф'ючерсах — до 0,05%. Біржа забезпечує високу безпеку та надійність, що підтверджується її рейтингом 10 за версією CoinGecko.

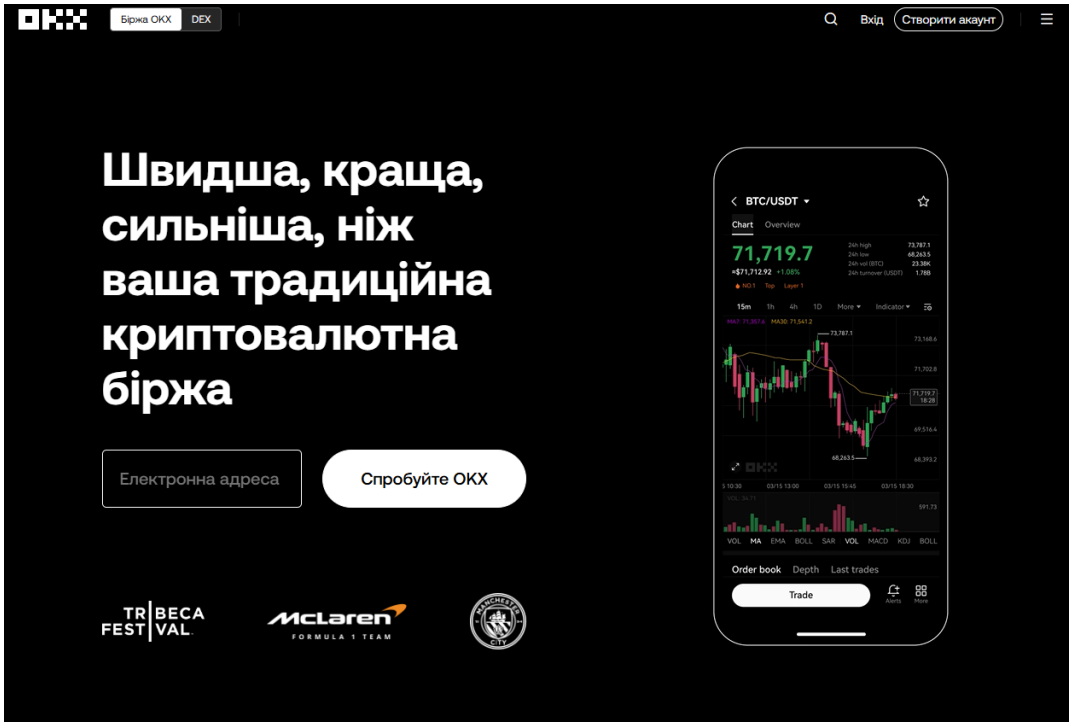


Рис. 1.1 Офіційний сайт та логотип OKX [2]

BitGet – криптовалютна біржа, заснована у 2018 році в Сінгапурі (рис. 1.2). Вона спеціалізується на ф'ючерсній торгівлі та пропонує спотову торгівлю,

контракти та додаткові функції. BitGet підтримує спотову торгівлю з комісіями до 0,1% та ф'ючерсну торгівлю з комісіями до 0,04%. Біржа відома своєю соціальною торгівлею та функціями копіювання угод [6].

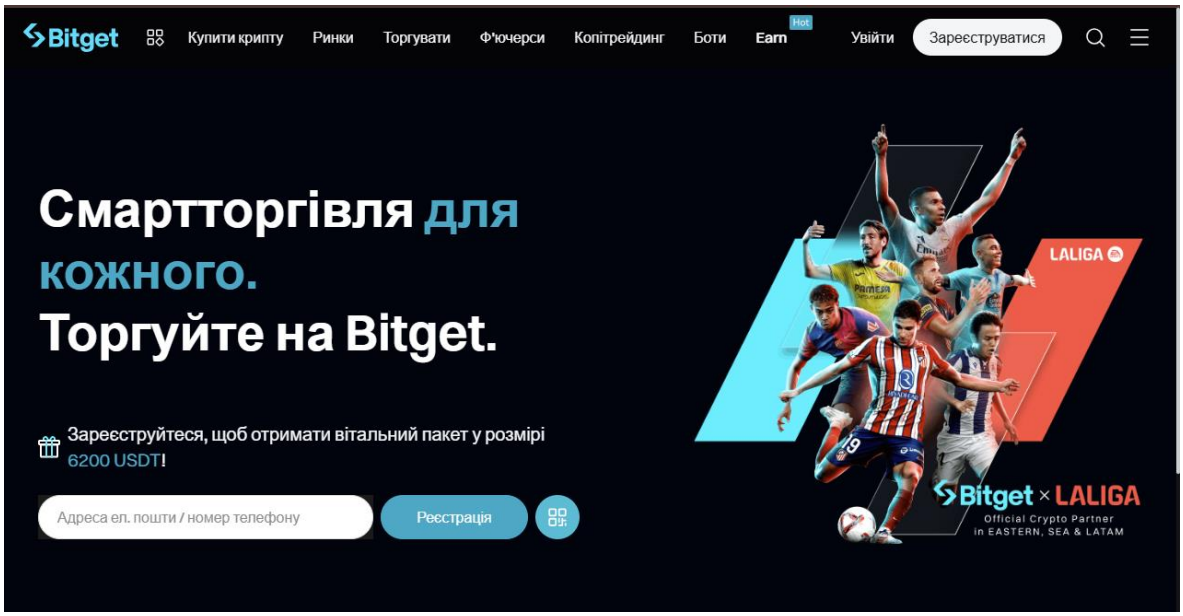


Рис. 1.2 Офіційний сайт та логотип BitGet [2]

BingX (рис 1.3) – криптовалютна біржа, яка також пропонує спотову та ф'ючерсну торгівлю, а також унікальні функції, такі як копіювання торгівлі. BingX підтримує понад 700 криптовалют та пропонує кредитне плече до 150x для основних криптовалют, таких як BTC та ETH. Біржа забезпечує високу безпеку та надійність, що підтверджується її рейтингом 10 за версією CoinGecko.

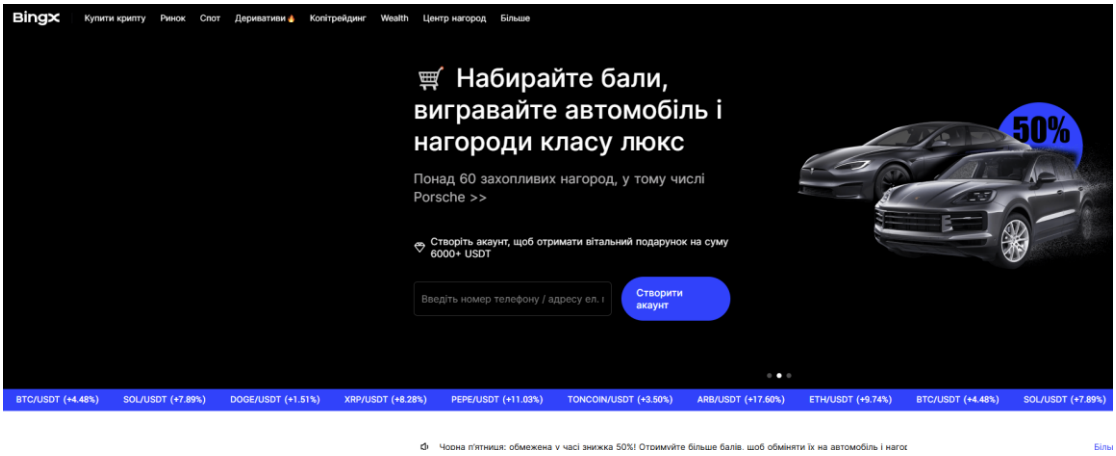


Рис. 1.3 Офіційний сайт та логотип BingX [2]

Binance – одна з найбільших криптовалютних бірж у світі, заснована у 2017 році (рис 1.4). Вона пропонує широкий спектр послуг, включаючи спотову та ф'ючерсну торгівлю, маржинальну торгівлю, NFT-маркетплейс, стейкінг та інші фінансові інструменти. Binance підтримує понад 350 криптовалют та 600 торгових пар. Комісії на спотовій торгівлі становлять до 0,1%, а на ф'ючерсах — до 0,04%. Біржа забезпечує високу безпеку та надійність, що підтверджується її рейтингом 10 за версією CoinGecko [7].

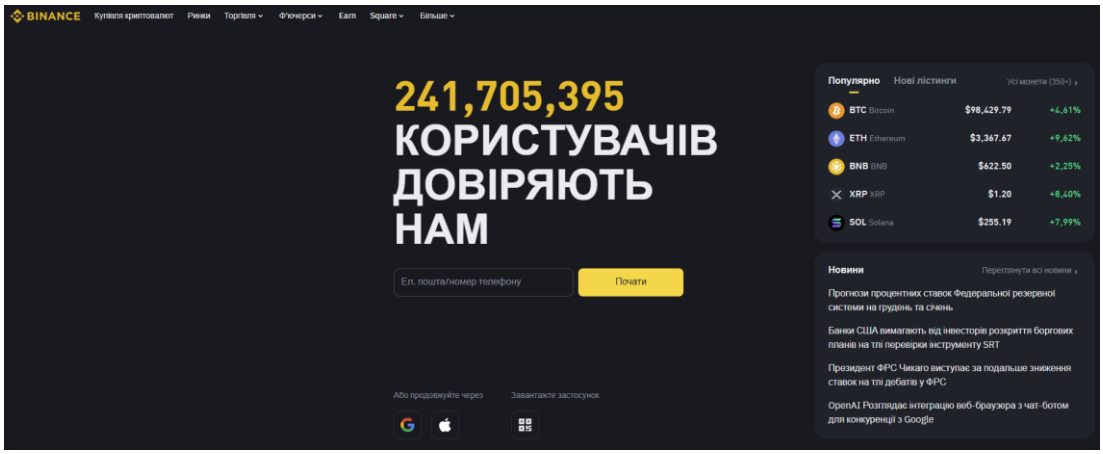


Рис. 1.4 Офіційний сайт та логотип Binance [4]

Усі ці біржі впроваджують передові технології для забезпечення високої продуктивності, масштабованості та надійності своїх веб-застосунків, що дозволяє їм ефективно обробляти великі обсяги даних та забезпечувати безперебійну роботу навіть при високих навантаженнях. В таблиці 1.2 наведена порівняння технічних характеристик крипто-бірж.

Таблиця 1.2

Порівняльння технічних характеристик криптовалютних бірж

Характеристика	**OKX**	**BitGet**	**BingX**	**Binance**
Рік заснування	2017	2018	2018	2017
Країна реєстрації	Сейшельські острови	Сінгапур	Сінгапур	Кайманові острови
Кількість підтримуваних криптовалют	348	300+	700+	400+
Кількість торгових пар	597	700+	700+	1 700+
Комісія за спотову торгівлю	До 0,1%	До 0,1%	До 0,1%	До 0,1%
Комісія за ф'ючерсну торгівлю	До 0,05%	До 0,04%	До 0,05%	До 0,04%
Кредитне плече	До 125x для ф'ючерсів	До 125x для ф'ючерсів	До 150x для основних криптовалют (BTC, ETH)	До 125x для ф'ючерсів
Додаткові функції	Стейкінг, кредитування, P2P-платформа, NFT-маркетплейс	Соціальна торгівля, копіювання угод, стейкінг	Копіювання торгівлі, стейкінг, NFT-маркетплейс	Стейкінг, кредитування, NFT-маркетплейс, дебетова карта з кешбеком
Рейтинг безпеки (за CoinGecko)	10/10	10/10	10/10	10/10

1.2 Характеристика продуктивності: фактори впливу та критерії оцінки

Продуктивність web-застосунків – це сукупність характеристик, що визначають швидкість, ефективність та стабільність роботи системи під час виконання її функцій у визначених умовах навантаження. У контексті web-застосунків продуктивність вимірюється часом відгуку системи на запити користувачів, кількістю одночасно оброблюваних запитів, стабільністю роботи під високим навантаженням, а також оптимальним використанням ресурсів сервера і клієнта [8].

Цей показник має критичне значення для користувацького досвіду (UX), оскільки затримки в роботі системи можуть призводити до незадоволення користувачів, втрати клієнтів та зниження репутації платформи. Особливо важлива продуктивність у сферах, де час реагування є вирішальним фактором, як, наприклад, у біржових платформах. Тут кожна секунда може впливати на фінансові результати клієнтів.

Ключові характеристики продуктивності web-застосунків:

– Час відгуку (Response Time) – це період, необхідний для обробки запиту користувача та повернення відповіді. Оптимальний час відгуку має становити від 1 до 3 секунд для зручної роботи, оскільки більш тривалі затримки можуть знижувати ефективність взаємодії.

– Пропускна здатність (Throughput), визначає кількість запитів, які система може обробити за одиницю часу. Цей параметр є важливим для оцінки, як система працює під високим навантаженням. Наприклад, біржові платформи повинні обробляти тисячі транзакцій одночасно.

– Стабільність роботи, відображає здатність web-застосунку підтримувати належну продуктивність при збільшенні навантаження. Ця характеристика є ключовою під час пікових періодів, таких як активна торгівля або запуск нових функцій.

– Ефективність використання ресурсів, оптимальне використання

апаратних і програмних ресурсів сервера (процесора, пам’яті, дискових операцій) та мінімальне споживання ресурсів клієнта (час завантаження сторінок, обробка скриптів) [9].

– Масштабованість, здатність системи адаптуватися до зростання кількості користувачів або обсягу даних без втрати продуктивності. Для web-застосунків це досягається за допомогою розподіленої архітектури, хмарних обчислень і балансування навантаження.

У web-застосунках продуктивність визначає якість обслуговування клієнтів та ефективність бізнес-процесів. Повільна робота системи може призводити до таких негативних наслідків:

– Зниження лояльності користувачів, клієнти відмовляються від роботи з платформою через тривалий час очікування.

– Фінансові втрати, затримки у виконанні транзакцій можуть призводити до втрати прибутку, особливо у фінансовій сфері.

– Погіршення репутації, відгуки про повільну роботу системи можуть негативно впливати на бренд.

Продуктивність web-застосунків оцінюється за допомогою певних ключових показників, які дозволяють визначити ефективність роботи системи та покращити користувацький досвід. Одним із найважливіших параметрів є час завантаження сторінки, який вимірюється від моменту надсилання запиту до повного завантаження сторінки у браузері. Цей показник включає час завантаження HTML, CSS, JavaScript, зображень та інших ресурсів. Ідеальний час завантаження не перевищує трьох секунд [10].

Ще одним параметром є кількість запитів, які сервер здатний обробити за секунду, що дозволяє оцінити масштабованість системи та її здатність працювати під навантаженням. Поряд із цим, середній час відповіді сервера є важливим показником швидкості системи; він демонструє, скільки часу необхідно серверу, щоб обробити запит і відправити відповідь.

Пропускна здатність системи також відіграє важливу роль, адже вона показує, скільки даних сервер може обробити за одиницю часу. Ще одним

важливим параметром є час до першого байта (TTFB), який визначає швидкість відгуку сервера від моменту надсилання запиту до отримання першого байта інформації.

Крім цього, час виконання JavaScript на стороні клієнта може впливати на продуктивність, особливо у випадку односторінкових застосунків (SPA). Нарешті, показники використання серверних ресурсів, таких як CPU, оперативна пам'ять та дисковий простір, дозволяють визначити ефективність використання інфраструктури. Ці метрики дають комплексну картину продуктивності web-застосунку та є основою для його оптимізації.

Для ефективної оцінки та покращення продуктивності web-застосунків існує широкий спектр інструментів, кожен із яких має свої переваги та спеціалізацію. У цій таблиці представлені найбільш поширені засоби для аналізу продуктивності, такі як Google Lighthouse, Apache JMeter, New Relic та інші. Вони охоплюють різні аспекти роботи системи, включаючи тестування навантаження, моніторинг у реальному часі, аналіз клієнтської частини та серверної продуктивності. Таблиця 1.3 містить ключові особливості та цільову аудиторію кожного інструмента, що допоможе визначити найкращий варіант залежно від потреб проєкту [11].

Таблиця 1.3

Порівняння інструментів оцінки продуктивності

Інструмент	Основне призначення	Тип аналізу	Особливості	Цільова аудиторія
Google Lighthouse	Аналіз продуктивності, доступності, SEO	Аналіз клієнтської частини	Вбудований у Chrome, детальні рекомендації	Розробники, SEO-фахівці
Apache JMeter	Тестування навантаження та продуктивності	Навантажувальне тестування	Висока масштабованість, багатопотоковість	Інженери продуктивності
New Relic	Моніторинг продуктивності у реальному часі	Реальний моніторинг у продакшені	Потужна аналітика, інтеграція з іншими сервісами	Операційні команди, адміністратори

Продовження таблиці 1.3

Інструмент	Основне призначення	Тип аналізу	Особливості	Цільова аудиторія
GTmetrix	Аналіз швидкості завантаження сторінки	Аналіз швидкості завантаження	Рекомендації щодо оптимізації ресурсів	Розробники, аналітики швидкості
Pingdom Tools	Моніторинг доступності та швидкості	Моніторинг доступності з різних локацій	Підтримка географічних тестів	Веб-адміністратори, тестувальники
Dynatrace	Моніторинг усього стеку технологій	Моніторинг сервера, бази даних та клієнта	Використання штучного інтелекту	Ентерпрайз-користувачі
Browser DevTools	Аналіз продуктивності клієнтської частини	Локальний аналіз у браузері	Зручність для розробників, безкоштовний	Розробники, тестувальники

1.3 Огляд технологій і підходів для підвищення продуктивності web-застосунків

Для забезпечення високої продуктивності та стабільності web-застосунків важливо застосовувати сучасні технології та підходи до оптимізації серверної частини. Основними методами є використання кешування, балансування навантаження між серверами та оптимізація роботи баз даних.

Використання кешування – це однин із ключових методів підвищення продуктивності web-застосунків. Воно дозволяє зменшити навантаження на сервери, скорочуючи кількість звернень до бази даних або інших джерел інформації. Серед популярних інструментів кешування виділяють:

Memcached – високопродуктивна система для зберігання тимчасових даних у пам'яті, що дозволяє значно зменшити час доступу до часто використовуваних даних [12].

Redis – потужний інструмент кешування, що підтримує зберігання складніших структур даних, таких як списки, множини та хеші. Redis також має

функції реплікації та забезпечення високої доступності, що робить його універсальним рішенням для веб-застосунків.

Завдяки кешуванню запити обробляються швидше, що позитивно впливає на час відповіді системи і загальну продуктивність.

Балансування навантаження між серверами.

Для підтримки стабільності роботи web-застосунку під високими навантаженнями необхідно використовувати балансування навантаження, яке забезпечує рівномірний розподіл запитів між серверами. Основними технологіями для цього є:

NGINX – веб-сервер та балансувальник навантаження, який дозволяє ефективно розподіляти запити між серверами, а також підтримує функції кешування та оптимізації з'єднань.

AWS Elastic Load Balancer – хмарний сервіс балансування навантаження від Amazon, що автоматично масштабується в залежності від навантаження. Він інтегрується з іншими сервісами AWS і забезпечує високу надійність та безпеку.

Балансування навантаження не тільки покращує продуктивність системи, але й забезпечує її стійкість до відмов, перенаправляючи запити у разі виходу одного із серверів із ладу.

Оптимізація баз даних.

Бази даних є важливим компонентом web-застосунків, тому їх оптимізація значно впливає на продуктивність системи. Серед основних підходів можна виділити:

Індексація – створення індексів для таблиць бази даних значно скорочує час виконання запитів, особливо при роботі з великими обсягами даних.

Реплікація – копіювання даних між декількома серверами баз даних. Це дозволяє зменшити навантаження на основний сервер і підвищити доступність даних.

Оптимізація запитів – переписування та вдосконалення SQL-запитів для зменшення часу їх виконання. Це включає використання ефективних JOIN-ів, обмеження кількості обраних полів та уникнення підзапитів там, де це можливо.

Застосування цих підходів у сукупності дозволяє значно підвищити продуктивність серверної частини web-застосунків, забезпечуючи швидкість обробки запитів та стабільність системи навіть при високих навантаженнях [13].

Покращення продуктивності клієнтської частини web-застосунків є ключовим фактором для забезпечення швидкого та плавного користувацького досвіду. Досягнення цієї мети можливе через впровадження сучасних підходів до оптимізації розміру ресурсів, обробки зображень та структури застосунків.

Зменшення розміру сторінок, які передаються між сервером і клієнтом, дозволяє значно скоротити час завантаження сторінок. Для цього використовуються наступні методи:

– Мінімізація CSS/JS – видалення зайвих пробілів, коментарів та неефективного коду з файлів стилів і скриптів. Інструменти, такі як UglifyJS, Terser або CSSNano, допомагають автоматизувати цей процес.

– Об'єднання файлів – зменшення кількості HTTP-запитів шляхом об'єднання кількох файлів CSS або JavaScript в один.

Ці заходи дозволяють не тільки знизити розмір сторінок, але й зменшити затримки через мережу, що особливо важливо для користувачів з низькою швидкістю інтернету.

Використання сучасних форматів зображень, зображення складають значну частину трафіку web-застосунків, тому їх оптимізація є критичною для швидкості завантаження. Сучасні формати, такі як WebP, забезпечують високу якість зображень при меншому розмірі файлів у порівнянні з традиційними форматами JPEG або PNG. WebP підтримує як стиснення із втратами, так і без втрат, що робить його універсальним для різних типів контенту. Використання цього формату разом із lazy loading (відкладене завантаження зображень) дозволяє значно покращити швидкість відображення сторінок.

Односторінкові застосунки (Single Page Applications) є сучасною архітектурою, яка зменшує навантаження на мережу, оскільки основні ресурси завантажуються один раз, а подальші зміни в інтерфейсі відбуваються шляхом

взаємодії з API. У SPA перехід між сторінками здійснюється без перезавантаження, що значно покращує швидкість і зручність роботи користувача.

SPA зазвичай реалізуються з використанням таких фреймворків, як React, Angular або Vue.js. Вони забезпечують ефективну взаємодію між клієнтом і сервером, мінімізуючи обсяг даних, які передаються, та підвищуючи загальну продуктивність.

Хмарні обчислення стали невід'ємною частиною сучасного розвитку web-застосунків, пропонуючи високу масштабованість, продуктивність і доступність. Провідні платформи, такі як AWS, Google Cloud і Microsoft Azure, забезпечують набір інструментів і сервісів, які допомагають ефективно вирішувати завдання обробки великих обсягів даних, зберігання та масштабування [15].

– AWS (Amazon Web Services) пропонує рішення для автоматичного масштабування (Auto Scaling), балансування навантаження (Elastic Load Balancer), хмарного зберігання (S3) та інструменти для обчислень (EC2). Завдяки цим сервісам можна динамічно збільшувати або зменшувати ресурси залежно від навантаження.

– Google Cloud забезпечує інструменти для обробки великих даних (BigQuery), хмарного машинного навчання, а також має потужні рішення для хмарного зберігання й обчислень.

– Microsoft Azure інтегрує хмарні сервіси з іншими продуктами Microsoft, такими як Office 365 і Dynamics, а також забезпечує гнучкі рішення для DevOps, аналітики та віртуалізації.

Хмарні сервіси допомагають підвищити продуктивність за рахунок використання віртуалізованих обчислювальних ресурсів і автоматизації інфраструктури. Вони також забезпечують високу надійність і доступність завдяки географічно розподіленим дата-центрам.

Content Delivery Network є мережею серверів, розташованих у різних частинах світу, які кешують статичний контент, такий як зображення, CSS, JavaScript, і доставляють його кінцевим користувачам з найближчого серверу. Це дозволяє значно скоротити час завантаження сторінок, зменшити затримки через

мережу та знизити навантаження на основний сервер. Серед популярних провайдерів CDN – Cloudflare, Akamai, AWS CloudFront.

Впровадження асинхронного обміну даними через WebSocket або REST API

Асинхронний обмін даними дозволяє web-застосункам оновлювати інформацію на сторінці без її повного перезавантаження. Сучасні методи включають:

WebSocket – технологія, яка забезпечує двосторонній обмін даними між клієнтом і сервером у реальному часі. WebSocket ефективний для чату, оновлення даних на панелях адміністратора чи біржових платформ.

REST API – широко використовуваний підхід до побудови веб-сервісів, що дозволяє клієнтам отримувати дані у форматах JSON чи XML. Асинхронні запити через REST API забезпечують плавний користувацький досвід і зменшують навантаження на сервер [16].

Використання технологій SSR (Server-Side Rendering) та JAMstack

Сучасні підходи до рендерингу сторінок значно впливають на швидкість їх завантаження та оптимізацію SEO.

SSR (Server-Side Rendering) – технологія, яка забезпечує генерацію HTML на стороні сервера перед відправкою сторінки клієнту. Це зменшує час завантаження першого екрану та покращує індексацію сторінок пошуковими системами. SSR широко застосовується у фреймворках, таких як Next.js і Nuxt.js.

JAMstack (JavaScript, API, Markup) – сучасний підхід до розробки статичних веб-сайтів із використанням попередньо згенерованих сторінок і API для динамічного контенту. Це забезпечує високу швидкість завантаження та легке масштабування.

Розглянуті підходи до підвищення продуктивності web-застосунків демонструють, наскільки важливими є комплексні заходи оптимізації як серверної, так і клієнтської частини. Використання сучасних технологій, таких як хмарні обчислення, CDN, асинхронний обмін даними та SSR/JAMstack, дозволяє значно підвищити швидкість роботи, масштабованість і надійність систем.

2 ДОСЛІДЖЕННЯ ПРОБЛЕМ ТА РОЗРОБКА МЕТОДИКИ ОПТИМІЗАЦІЇ

2.1 Виявлення вузьких місць у роботі web-застосунків підтримки клієнтів бірж

Аналіз вузьких місць у роботі веб-застосунків є важливим етапом оптимізації їх продуктивності. Біржові платформи вимагають високої швидкодії, надійності та масштабованості, адже навіть короткочасні збої або затримки можуть призводити до фінансових втрат і втрати довіри клієнтів. У цьому розділі представлено процес виявлення проблемних аспектів у роботі таких систем.

Методи виявлення вузьких місць.

Для ідентифікації вузьких місць у роботі веб-застосунків необхідно застосовувати комплексний підхід, який охоплює моніторинг серверів, аналіз логів, тестування продуктивності та збір відгуків від кінцевих користувачів.

Моніторинг продуктивності системи:

Серверна частина:

– New Relic, Dynatrace, Prometheus: Використовуються для моніторингу серверів, баз даних і API, що дозволяє виявляти сповільнення, перевантаження або помилки.

– Показники: час відповіді серверів, завантаженість CPU і RAM, кількість активних з’єднань.

Клієнтська частина:

– Google Lighthouse: Аналізує швидкість завантаження сторінок, ефективність рендерингу, використання ресурсів і продуктивність JavaScript.

– GTmetrix: Вимірює час завантаження сторінки, час до першого байта (TTFB) і дає рекомендації щодо оптимізації.

Аналіз логів:

– Збір даних з логів веб-серверів (NGINX, Apache) і систем баз даних:

– Пошук довготривалих запитів, які викликають затримки.

– Аналіз повторюваних помилок, таких як 5xx або 4xx відповіді.

– Інструменти: ELK Stack (Elasticsearch, Logstash, Kibana).

Стрес-тестування:

Для імітації пікових навантажень використовуються інструменти, такі як:

– Apache JMeter: Створення сценаріїв навантаження для оцінки реакції системи на підвищену кількість запитів.

– Loader.io або k6: Тестування продуктивності при зростанні навантаження.

Зворотній зв’язок від користувачів:

– Збір скарг від користувачів на повільну роботу, затримки в оновленні даних або помилки у веб-застосунку.

– Використання систем обробки запитів підтримки (HelpDesk).

Моніторинг використання ресурсів:

Оцінка серверних ресурсів:

– Показники завантаження CPU, RAM, дискового простору та мережі.

– Інструменти: Grafana, Nagios, Zabbix.

– Аналіз мережевого трафіку для виявлення перевантажень або зловмисних дій (наприклад, DDoS-атак).

Типові вузькі місця у веб-застосунках біржових платформ зосереджені на трьох основних аспектах: серверній частині, клієнтській частині та мережевих взаємодіях. Серверна частина часто стикається з надмірним навантаженням на базу даних. Це може бути пов'язано з виконанням неоптимальних SQL-запитів, які вимагають повного сканування таблиць замість використання індексів. Подібні проблеми можуть виникати через відсутність реплікації бази даних, що обмежує її доступність і збільшує час відповіді. Ще однією поширеною проблемою є перевантаження серверів, яке виникає через недостатнє використання кешування або відсутність механізмів горизонтального масштабування. Це призводить до того, що сервери не можуть ефективно обробляти велику кількість одночасних запитів, особливо у пікові години [17].

Клієнтська частина часто страждає від повільного завантаження сторінок. Основними причинами цього є надмірний обсяг даних, які передаються між сервером і клієнтом, наприклад, великі зображення у форматах без стиснення або неефективне використання сучасних форматів, таких як WebP. Крім того, на швидкість впливає велика кількість HTTP-запитів до різних ресурсів, що викликає затримки у завантаженні. Значний вплив має і неоптимальний JavaScript, який перевантажує клієнтську частину і викликає затримки у рендерингу сторінок. У деяких випадках відсутність ефективного кешування на стороні клієнта, наприклад, за допомогою браузерів або CDN, ще більше посилює ці проблеми, оскільки статичні ресурси завантажуються заново при кожному запиті.

Мережеві аспекти також є значним джерелом проблем для біржових платформ. Затримки можуть бути викликані географічною віддаленістю серверів, через що запити потребують більше часу для досягнення клієнтів. Це особливо актуально для платформ, які не використовують Content Delivery Network (CDN), що дозволяє зменшити відстань між сервером і користувачем шляхом розташування кешуючих серверів ближче до кінцевих точок. Перевантаження мережі через велику кількість переданих даних або недостатню пропускну здатність також може уповільнити роботу системи. Крім того, зловмисні дії, такі як DDoS-атаки, створюють штучне навантаження на сервери, знижуючи доступність системи та час відповіді.

Безпека також є значущим аспектом, який впливає на продуктивність веб-застосунків. SQL-ін’єкції, DDoS-атаки та інші види зловмисних дій можуть призводити до уповільнення виконання запитів, перевантаження серверів і навіть повного виходу системи з ладу. Відсутність належних заходів безпеки, таких як шифрування, двофакторна автентифікація та моніторинг аномальної активності, робить системи більш уразливими до атак, що ускладнює їх роботу під час високих навантажень або зростання кількості користувачів.

Таким чином, ключові проблеми продуктивності веб-застосунків біржових платформ виникають через комбінацію технічних обмежень, неоптимальної архітектури і недостатньої уваги до деталей безпеки та масштабованості.

Виявлення цих проблем є важливим етапом для забезпечення стабільності, швидкодії та безперебійної роботи систем у реальних умовах експлуатації.

2.2 Обґрунтування вибору методів і технологій для покращення продуктивності

Критерії вибору технологій:

Інтеграція з WordPress, CMS WordPress обрана через її гнучкість, розширюваність та широке ком'юніті підтримки. Завдяки плагінам та темам можна швидко реалізувати як стандартні, так і унікальні функції без потреби розробки з нуля. Вбудована підтримка REST API дозволяє інтегрувати сторонні системи.

Швидкість розробки, використання PHP як серверної мови програмування, що є основою WordPress, забезпечує швидку розробку та налаштування функціоналу. Інтеграція з Laravel як додатковим фреймворком дозволяє створювати окремі мікросервіси або складні бізнес-логіки без втрати продуктивності [18].

Фронтенд-оптимізація, CSS і JavaScript використовуються для покращення візуальної привабливості та інтерактивності. Використання сучасних бібліотек (наприклад, jQuery або власних скриптів) дозволяє зменшити час завантаження сторінок.

Середовище розробки: IDE PHPStorm обране через його інтегровані можливості для налагодження коду, підсвічування синтаксису, а також інтеграції з системами контролю версій та локальними серверами, такими як Open Server. Це сприяє швидкій діагностиці та виправленню помилок.

Масштабованість, застосування Laravel як backend-фреймворку дає можливість створення масштабованих і продуктивних рішень. Його модульна структура підходить для інтеграції з біометричними системами контролю доступу (СКД).

Безпека, використання технологій із підтримкою сучасних стандартів безпеки, таких як шифрування даних, дозволяє захистити конфіденційні біометричні дані.

Локальне тестування, Open Server надає можливість швидкого налаштування локального середовища для тестування та розробки, що економить час і ресурси під час розробки.

Методи покращення продуктивності:

– Оптимізація запитів до бази даних.

– Використання кешування на рівні сервера (оптимізація сторінок, зменшення часу завантаження).

– Мінімізація CSS, JS-файлів та зображень.

– Використання CDN для статичного контенту.

– Інтеграція інструментів моніторингу продуктивності для діагностики вузьких місць.

Laravel є одним із найпопулярніших PHP-фреймворків завдяки його багатому функціоналу та зручності у використанні. Він дозволяє створювати потужну серверну частину з урахуванням сучасних вимог до масштабованості, безпеки та оптимізації коду [19].

Основні переваги Laravel:

1. Масштабованість: Laravel має модульну архітектуру, що дає змогу додавати нові функції без шкоди для існуючого коду. Це забезпечує легку інтеграцію нових компонентів, таких як обробка біометричних даних у системі контролю доступу.

2. Чистота та оптимізація коду, використання вбудованого шаблону проектування MVC (Model-View-Controller) сприяє структурованій організації коду, зменшує дублювання та спрощує підтримку. Завдяки цьому код стає зрозумілим і легко розширюваним.

3. Підтримка баз даних, Laravel пропонує ORM Eloquent, що дозволяє працювати з базами даних на високому рівні абстракції. Він підтримує складні

реляційні запити, що особливо важливо для роботи з великими обсягами даних у біометричних системах.

4. Інтеграція API, Laravel має вбудовану підтримку REST API, що спрощує інтеграцію з іншими системами, такими як WordPress, біометричні пристрої або інші зовнішні сервіси.

5. Безпека, Laravel забезпечує вбудовані механізми захисту від SQL-ін'єкцій, XSS-атак, CSRF-атак, що є критично важливим при роботі з чутливими даними, такими як біометричні дані.

6. Підтримка кешування, Фреймворк надає можливість легко налаштувати кешування на стороні сервера, що значно зменшує час обробки запитів та підвищує швидкість роботи системи.

7. Асинхронна обробка завдань, використання черг завдань (queues) дозволяє виконувати ресурсоємні процеси (наприклад, аналіз біометричних даних) у фоновому режимі, не впливаючи на швидкість відгуку для користувача.

8. Масштабування для навантаження, Laravel підтримує горизонтальне масштабування завдяки використанню таких сервісів, як Redis або AWS, що дозволяє обробляти збільшення кількості користувачів та запитів без зниження продуктивності.

PHP, CSS+JS, і HTML є основними технологіями, які забезпечують повний цикл розробки веб-застосунків, об'єднуючи серверну логіку, структуру сторінки, стилізацію та інтерактивність.

PHP (Hypertext Preprocessor) виконує роль серверної мови програмування, відповідальної за обробку запитів, взаємодію з базою даних і генерацію динамічного контенту. Завдяки простому синтаксису та великій кількості вбудованих функцій, PHP дозволяє реалізувати складні бізнес-логіки швидко та ефективно. Він є основою для CMS WordPress, що робить його ідеальним вибором для інтеграції у веб-застосунки. Крім того, PHP забезпечує високу гнучкість у розробці, легко взаємодіє з фронтенд-технологіями та базами даних, такими як MySQL або PostgreSQL [20].

HTML (HyperText Markup Language) є основою структури кожної веб-сторінки. Він забезпечує ієрархію елементів, що дозволяє чітко організовувати контент. Сучасні семантичні теги, такі як <section>, <article> або <header>, підвищують доступність і SEO-оптимізацію веб-ресурсу. HTML забезпечує базу для інтеграції інших технологій, таких як CSS і JavaScript, роблячи сторінки функціональними та привабливими.

CSS (Cascading Style Sheets) відповідає за стилізацію веб-сторінок. Завдяки сучасним інструментам, таким як Flexbox, Grid Layout та медіа-запити, CSS забезпечує адаптивний дизайн, що працює на будь-яких пристроях — від смартфонів до десктопів. Крім того, використання CSS препроцесорів, таких як SASS або LESS, спрощує написання стилів, дозволяє зменшити розмір файлів і покращує продуктивність.

JavaScript додає веб-застосункам динамічність і інтерактивність. З його допомогою можна створювати функції, які реагують на дії користувачів у реальному часі, наприклад, інтерактивні форми, анімації чи оновлення контенту без перезавантаження сторінки через AJAX. Сучасні бібліотеки та фреймворки, такі як jQuery, React або Vue.js, роблять розробку більш ефективною, а використання стандартів ES6+ сприяє оптимізації й підтримці коду.

У поєднанні ці три технології формують потужний інструментарій для створення динамічних, продуктивних і естетично привабливих веб-застосунків. PHP забезпечує серверну логіку, HTML формує структуру сторінки, CSS додає стиль, а JavaScript забезпечує інтерактивність. Це дозволяє створювати сучасні, адаптивні та функціональні рішення, що відповідають найвищим вимогам бізнесу та користувачів [21].

Open Server — це програмний комплекс, який використовується як локальне серверне середовище для тестування та розробки веб-додатків. Цей інструмент надає всі необхідні компоненти для повноцінної роботи веб-серверів, баз даних і інших служб у межах локального середовища розробки.

Основна перевага Open Server — це можливість створення локального серверного середовища, яке імітує роботу реального веб-сервера. Це дозволяє

розробникам тестувати та налаштовувати свої проєкти без потреби в онлайн-хостингу, що економить час і ресурси [22].

Open Server підтримує широкий набір технологій, зокрема:

– Веб-сервери, Apache та Nginx, які є базою для роботи більшості веб-додатків.

– Бази даних, MySQL, MariaDB, PostgreSQL, що дозволяють зберігати та обробляти дані.

– Мови програмування, повна підтримка PHP різних версій для роботи з динамічним контентом і серверною логікою.

– Додаткові інструменти, Redis, Memcached, FTP-сервер, а також системи для роботи з email (наприклад, Sendmail).

Однією з ключових функцій Open Server є можливість одночасного тестування кількох проєктів, завдяки багатодоменній підтримці. Вбудований інтерфейс для налаштування дозволяє з легкістю керувати конфігурацією серверів, обирати версії PHP чи баз даних для конкретного проєкту.

Open Server також підтримує використання SSL-сертифікатів, що дозволяє тестувати веб-додатки у середовищі HTTPS. Це особливо важливо для проєктів, які вимагають високого рівня безпеки, наприклад, систем контролю доступу або роботи з конфіденційними даними.

Оптимізація продуктивності є важливим аспектом розробки, особливо для серверної частини веб-додатків. У цьому контексті використання фреймворку Laravel дозволяє впроваджувати сучасні методи оптимізації, які значно підвищують ефективність системи.

Оптимізація серверної частини з використанням Laravel, використання Eloquent ORM для ефективної роботи з базою даних:

Eloquent ORM, вбудований в Laravel, забезпечує зручну та потужну роботу з базами даних через об'єктно-орієнтований підхід. Це дозволяє:

– Зменшити кількість ручного написання SQL-запитів завдяки використанню простого синтаксису.

– Оптимізувати запити за допомогою відкладеного завантаження (lazy loading) та попереднього завантаження (eager loading), що знижує кількість звернень до бази даних.

– Ефективно працювати з реляційними зв’язками між таблицями (hasMany, belongsTo, тощо).

Кешування запитів із застосуванням Laravel Cache:

Laravel Cache надає зручний механізм для зберігання результатів оброблених даних у кеші, що дозволяє зменшити кількість звернень до бази даних або інших ресурсів [23].

– За допомогою Redis, Memcached або файлового кешу можна зберігати часто використовувані дані, наприклад, налаштування, результати складних запитів або статистику.

– Кешування дозволяє значно скоротити час виконання повторюваних запитів, підвищуючи швидкість роботи серверної частини.

– Інструменти кешування легко інтегруються у Laravel завдяки універсальному API, що підтримує різні драйвери.

Впровадження мідлварів для зменшення навантаження на сервер

Middleware (мідлвари) в Laravel дозволяють виконувати попередню обробку HTTP-запитів перед тим, як вони досягнуть основної логіки програми. Це дає змогу:

– Реалізувати обмеження доступу (rate limiting) для контролю кількості запитів від одного клієнта, що захищає сервер від перевантаження.

– Виконувати аутентифікацію та авторизацію користувачів, пропускаючи лише валідні запити.

– Здійснювати перевірку даних, щоб уникнути виконання непотрібної логіки або обробки помилкових запитів.

– Кешувати дані у мідлварах, що дозволяє зменшити обробку одних і тих самих запитів.

Для підвищення продуктивності веб-застосунку важливо не лише оптимізувати серверну частину, але й ефективно працювати з фронтендом.

Оптимізація CSS, JS та HTML дозволяє значно скоротити час завантаження сторінок, зменшити використання ресурсів і покращити взаємодію користувача із системою [24].

1. Мінімізація файлів за допомогою інструментів Laravel Mix

Laravel Mix є зручним інструментом для автоматизації роботи з ресурсами фронтенду. Його використання забезпечує:

Мінімізацію файлів CSS та JS, зменшення розміру файлів за рахунок видалення зайвих пробілів, коментарів і непотрібного коду. Це дозволяє скоротити час завантаження сторінки.

Об'єднання файлів, Laravel Mix дозволяє комбінувати кілька CSS та JS файлів в один, що зменшує кількість HTTP-запитів до сервера.

Комплексну обробку, Mix інтегрується з препроцесорами CSS, такими як SASS або LESS, що полегшує створення й оптимізацію стилів.

Використання версійності файлів: Генерація унікальних хешів для кожного оновленого файлу дозволяє уникати кешування застарілих версій браузером.

2. Використання асинхронного завантаження JS для зменшення часу завантаження сторінки

Асинхронне завантаження JavaScript допомагає уникнути блокування основного потоку під час завантаження сторінки. Це досягається через:

Використання атрибутів async та defer:

async дозволяє завантажувати скрипти одночасно із завантаженням HTML, що зменшує загальний час завантаження.

defer гарантує, що скрипт виконається лише після завершення парсингу HTML, забезпечуючи стабільне завантаження сторінки.

Динамічне завантаження модулів: Використання сучасного підходу до модульності JS (ES6 import/export) дозволяє завантажувати тільки ті компоненти, які необхідні для конкретної сторінки.

3. Використання сучасних CSS-методів для покращення швидкодії

Сучасні підходи до стилізації, такі як flexbox і grid, забезпечують не лише адаптивність, але й високу швидкодію.

Flexbox:

Використовується для вирівнювання елементів у межах контейнера. Він зменшує складність коду, позбавляючи необхідності писати зайвий CSS для позиціонування.

Grid Layout:

Забезпечує потужний інструментарій для створення складних макетів з мінімальним використанням додаткового коду. Це покращує продуктивність браузера, знижуючи обсяг обчислень.

Медіа-запити, використання адаптивних стилів для різних пристроїв дозволяє економити ресурси, завантажуючи лише потрібні стилі для конкретного розширення екрана.

Стиснення та попередня обробка: CSS-препроцесори (SASS, LESS) спрощують підтримку стилів, а їхнє стиснення через Laravel Mix допомагає зменшити розмір файлів.

WordPress є популярною CMS, яка надає широкий спектр інструментів для оптимізації продуктивності веб-сайту. Завдяки розширенням і плагінам можна легко впровадити ефективні рішення для покращення швидкодії та стабільності системи [26].

1. Оптимізація бази даних через плагіни (наприклад, WP Optimize) База даних у WordPress може накопичувати зайві записи, чернетки, спам-коментарі або тимчасові таблиці, що впливає на швидкодію. WP Optimize — це плагін, який забезпечує:

– Очищення бази даних: Видалення непотрібних записів, таких як автоматичні чернетки, ревізії та тимчасові таблиці.

– Стиснення таблиць: Оптимізація структури бази даних для покращення швидкості доступу до інформації.

– Планування автоматичної оптимізації: Плагін дозволяє налаштувати регулярне очищення та оптимізацію без потреби втручання. Завдяки цим функціям база даних працює швидше, а запити до неї виконуються ефективніше.

– Впровадження плагіна для кешування, такого як W3 Total Cache Кешування є ключовим методом оптимізації веб-сайтів. Плагін W3 Total Cache дозволяє значно знизити навантаження на сервер і прискорити завантаження сторінок:

– Кешування сторінок та об'єктів: Збереження HTML-версій сторінок для швидшого доступу користувачів.

– Мінімізація та об’єднання файлів: Зменшення розміру CSS, JS і HTML для зменшення часу завантаження.

– Підтримка CDN: Інтеграція з мережею доставки контенту (CDN) для прискорення завантаження статичних ресурсів.

– Браузерне кешування: Збереження ресурсів на стороні клієнта для зменшення повторних запитів до сервера. Впровадження такого плагіна значно покращує загальну продуктивність сайту, знижує час завантаження сторінок і забезпечує кращий користувацький досвід.

Open Server використовується як локальне серверне середовище, що забезпечує зручний та ефективний процес розробки і тестування.

Ефективна локальна розробка: Open Server дозволяє створити серверне середовище, яке повністю відповідає реальним умовам роботи сайту. Завдяки цьому можна тестувати оптимізації, такі як кешування або оптимізація бази даних, у безпечному середовищі.

Швидке тестування змін: Локальне середовище дозволяє розробникам вносити зміни в код, тестувати плагіни або перевіряти роботу кешування без необхідності розгортання на хостингу.

Підтримка різних конфігурацій: Open Server дозволяє легко перемикатися між версіями PHP, баз даних або веб-серверів (Apache/Nginx), що важливо для тестування сумісності та продуктивності.

У таблиці 2.1 представлено основні можливості та переваги трьох ключових інструментів, які використовуються для оптимізації веб-розробки: Laravel, WordPress та Open Server.

Таблиця 2. 1

Переваги використаних інструментів

Технологія	Основні можливості	Переваги
Laravel	- Готові інструменти для кешування (Laravel Cache, Redis).	- Швидка інтеграція рішень для продуктивності.
	- Ефективна робота з базою даних через Eloquent ORM.	- Підтримка сучасних стандартів та масштабованість.
	- Зручні засоби для організації роботи з міграціями та схемами баз даних.	
WordPress	- Легка інтеграція плагінів для оптимізації (WP Optimize, W3 Total Cache).	- Мінімізація витрат часу на налаштування функціоналу.
	- Підтримка сучасних API для розширення можливостей сайту (REST API, GraphQL).	- Гнучкість завдяки розширюваності плагінами і темами.
	- Розвинена екосистема для SEO та продуктивності (наприклад, Yoast, Rank Math).	
Open Server	- Локальне серверне середовище для розробки та тестування (Apache, Nginx, MySQL, Redis).	- Мінімізація часу на налаштування середовища розробки.
	- Підтримка кількох конфігурацій серверів і версій PHP.	- Зручний інтерфейс для керування налаштуваннями.
	- Можливість роботи з SSL-сертифікатами для тестування HTTPS.	- Ефективне тестування змін у локальному середовищі без ризиків для продакшн-сервера.

2.3 Розробка методики оптимізації web-застосунку

Процес оптимізації веб-застосунку включає кілька ключових етапів, спрямованих на покращення швидкодії та зниження навантаження на сервер і клієнтські ресурси. Першим кроком у розробці методики оптимізації є аналіз продуктивності, що дозволяє зібрати необхідні дані для подальших кроків.

Аналіз продуктивності є першим важливим етапом оптимізації, який дозволяє виявити слабкі місця в роботі веб-застосунку. На цьому етапі зібрані дані

допомагають приймати обґрунтовані рішення щодо подальших кроків оптимізації. Для цього використовуються такі методи:

– Збір даних через плагіни WordPress і моніторинг Laravel
Для ефективного збору даних використовуються спеціальні плагіни WordPress і інструменти моніторингу для Laravel. Наприклад, плагін WP Debugging на WordPress дозволяє отримати інформацію про помилки, попередження та інші аспекти роботи сайту, що можуть впливати на продуктивність. В Laravel використовується інструмент Laravel DebugBar, який надає детальну інформацію про виконання запитів до бази даних, час завантаження сторінок та інші метрики, що дозволяє виявити проблеми в серверній частині застосунку.

– Оцінка швидкодії сторінок за допомогою Google Lighthouse Google Lighthouse – це потужний інструмент для оцінки продуктивності веб-сайтів. Він дозволяє оцінити різні аспекти продуктивності сторінки, такі як час завантаження, індекс швидкодії (Performance Score), доступність, SEO, а також можливість використання прогресивних веб-додатків (PWA). Lighthouse допомагає зібрати конкретні рекомендації щодо покращення швидкодії, зокрема через зменшення розміру ресурсів, оптимізацію зображень та поліпшення часу першого відображення сторінки [27].

Ці інструменти забезпечують чітке розуміння поточної продуктивності web-застосунку і надають необхідні дані для прийняття рішень щодо подальших кроків оптимізації, таких як кешування, зменшення розмірів файлів, поліпшення роботи з базами даних або оптимізація кодів.

Оптимізація серверної частини.

1. Реалізація кешування через Laravel Cache.

Кешування є одним із найефективніших способів оптимізації серверної частини веб-застосунку. Використовуючи Laravel Cache, можна значно зменшити навантаження на сервер і скоротити час відповіді на запити. Laravel надає різноманітні драйвери для кешування, включаючи кешування в пам’яті, Redis та Memcached, що дозволяє вибирати оптимальний спосіб в залежності від потреб проєкту. Кешування результатів запитів, сторінок або навіть цілих блоків контенту

допомагає значно пришвидшити доступ до даних та зменшити кількість запитів до бази даних, що призводить до поліпшення загальної продуктивності застосунку.

2. Застосування ефективних SQL-запитів з використанням Eloquent ORM Для забезпечення швидкодії веб-застосунку важливо оптимізувати взаємодію з базою даних. Eloquent ORM в Laravel дозволяє працювати з базою даних через об'єкти, що значно спрощує роботу з моделями та зв’язками між таблицями. Проте важливо застосовувати ефективні SQL-запити, мінімізуючи кількість виконуваних запитів і уникати зайвих "n+1" запитів (коли для кожного елементу виконується окремий запит). Для цього можна використовувати методи eager loading та lazy loading, які дозволяють завантажувати зв’язані дані тільки за необхідності, що значно покращує ефективність роботи з великими об’ємами даних.

Оптимізація клієнтської частини.

1. Мінімізація CSS та JS файлів через Laravel Mix Один із основних аспектів оптимізації клієнтської частини – це зменшення розміру файлів CSS і JavaScript. З використанням Laravel Mix можна автоматизувати процес мінімізації цих файлів, що дозволяє значно знизити їхній розмір та прискорити завантаження сторінок. Laravel Mix підтримує такі інструменти, як Webpack, що дозволяє об’єднувати кілька файлів у один, а також автоматично стискати їх. Мінімізація файлів за допомогою Mix також допомагає зменшити час обробки на сервері та покращити продуктивність веб-застосунку, адже браузер отримує менш об'ємні ресурси для завантаження [28].

2. Використання lazy loading для зображень Lazy loading є ефективною технікою для оптимізації завантаження зображень на сторінці. За допомогою цього методу зображення завантажуються тільки тоді, коли вони потрапляють у видиму частину екрану користувача. Це дозволяє значно зменшити початковий час завантаження сторінки, знижуючи навантаження на сервер і забезпечуючи кращу взаємодію з користувачем. В Laravel для реалізації lazy loading можна використовувати JavaScript-бібліотеки, які автоматично активують завантаження зображень, коли користувач прокручує сторінку, що також допомагає покращити швидкодію та зменшити використання пропускної здатності.

Інтеграція з WordPress.

1. Впровадження плагінів для кешування та стиснення даних

Інтеграція плагінів для кешування та стиснення даних є важливим кроком для покращення продуктивності веб-сайту на WordPress. Використання плагінів для кешування, таких як W3 Total Cache або WP Super Cache, дозволяє зберігати статичні версії сторінок на сервері, що значно зменшує час завантаження та навантаження на сервер при повторних запитах. Плагіни для стиснення даних, наприклад, Autoptimize або WP Rocket, допомагають зменшити розмір CSS, JavaScript та HTML файлів, що дозволяє скоротити час завантаження сторінки. Вони автоматично мінімізують файли, об’єднують їх у менші пакети та стискають зображення без втрати якості, що сприяє покращенню користувацького досвіду та зниженню часу завантаження.

2. Використання оптимізованих тем із мінімальним впливом на швидкодію.

Використання добре оптимізованих тем для WordPress є важливим аспектом забезпечення високої швидкодії сайту. Оптимізовані теми мінімізують використання надмірних скриптів, важких зображень та ресурсомістких елементів, що дозволяє зменшити час завантаження сторінки. Багато сучасних тем для WordPress вже підтримують такі функції, як адаптивний дизайн, мобільну оптимізацію та вбудоване кешування. Важливо вибирати теми, які використовують мінімум зовнішніх бібліотек і плагінів, а також підтримують швидке завантаження зображень та інші техніки, що покращують ефективність сайту. В результаті, правильний вибір теми може значно знизити вплив на швидкодію та підвищити продуктивність веб-сайту, зокрема на мобільних пристроях.

У процесі оптимізації веб-застосунку важливо враховувати як серверні, так і клієнтські аспекти, щоб забезпечити максимальну продуктивність та швидкість завантаження. У таблиці 2.2, яка описує ключові методи оптимізації для технологій Laravel, WordPress та фронтенду, з акцентом на їхній внесок у загальну продуктивність веб-застосунку.

Таблиця 2.2

Технічні деталі реалізації оптимізації web-застосунку

Технологія	Дія	Опис
Laravel	Впровадження config:cache та route:cache для кешування налаштувань і маршрутів	Використання команд Artisan для кешування конфігураційних файлів і маршрутів, що дозволяє зменшити час завантаження застосунку, особливо у великих проєктах.
	Використання Redis як кеш-сервісу	Redis дозволяє кешувати дані в пам'яті, значно зменшуючи час доступу до часто запитуваних даних і зменшуючи навантаження на базу даних.
WordPress	Очищення бази даних за допомогою WP Optimize	Плагін WP Optimize автоматично очищує базу даних від зайвих записів, таких як старі ревізії постів та спам-коментарі, що покращує швидкість роботи сайту.
	Установлення CDN через плагіни (наприклад, Cloudflare)	CDN забезпечує швидку доставку контенту з різних точок, що зменшує навантаження на основний сервер і покращує час завантаження сторінок, особливо для глобальних користувачів.
Фронтенд	Впровадження відкладеного завантаження скриптів через атрибут defer	Використання атрибуту defer для відкладеного завантаження JavaScript, що дозволяє браузеру спочатку завантажити вміст сторінки, а потім виконувати скрипти, покращуючи час завантаження.
	Мінімізація CSS та JS файлів через Laravel Mix	Використання Laravel Mix для автоматичної мінімізації CSS та JS файлів, що зменшує їхній розмір і покращує швидкість завантаження сторінки.

3 РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОЇ МЕТОДИКИ

3.1 Опис реалізації програми, включаючи вибір платформи та технологій, розробка архітектури додатку.

У даній роботі розробляється веб-застосунок для підтримки клієнтів криптобіржі. Основною метою проекту є реалізація методики покращення продуктивності такого застосунку, що забезпечить зручність використання, високу швидкість роботи та надійність.

У якості основної мови програмування обрано PHP. Це одна з найпоширеніших мов для розробки серверної частини веб-додатків. PHP дозволяє ефективно обробляти запити користувачів, працювати з базами даних та створювати інтерактивні динамічні сторінки. PHP має широку підтримку серед розробників, великий набір бібліотек та фреймворків, які спрощують процес створення сучасних веб-застосунків.

Для розробки клієнтської частини використовується комбінація мов HTML, CSS та JavaScript. HTML забезпечує структуру веб-сторінок, CSS відповідає за їх оформлення, а JavaScript надає інтерактивність і можливості динамічної взаємодії з користувачем. Ці мови є стандартом для веб-розробки і забезпечують кросплатформенну сумісність [29].

У якості бази даних обрано MySQL – одну з найпоширеніших систем керування реляційними базами даних. MySQL забезпечує високу швидкість обробки даних, гнучкість у побудові складних запитів та масштабованість для роботи з великими обсягами інформації. Її використання дозволяє організувати надійне зберігання та обробку даних клієнтів криптобіржі.

Для організації серверної частини застосунку було обрано фреймворк Laravel, який є одним з найпопулярніших інструментів для розробки на PHP. Laravel забезпечує високу продуктивність, простоту розробки та підтримку

сучасних підходів, таких як MVC (Model-View-Controller), що дозволяє відокремлювати бізнес-логіку від представлення. Laravel також має вбудовані засоби для роботи з базами даних, маршрутизації, аутентифікації та інших функціональних можливостей.

У якості веб-сервера використовується OpenServer – зручне середовище для локальної розробки, яке включає сервери Apache і Nginx, підтримку PHP і MySQL. Це забезпечує розробнику можливість швидкого налаштування середовища для роботи над проектом.

Також проект інтегрується з системою управління контентом (CMS) WordPress, що дозволяє спростити управління контентом і забезпечити легку адаптацію під різні потреби клієнтів. WordPress відомий своєю гнучкістю, великою кількістю плагінів та зручністю використання.

Для написання та налагодження коду використовується інтегроване середовище розробки (IDE) PHPStorm. Це потужний інструмент, який забезпечує підсвічування синтаксису, автодоповнення, інтеграцію з системами контролю версій та іншими сервісами, що підвищує продуктивність розробника.

Мультикросплатформеність проекту забезпечується використанням сучасних веб-технологій. Застосунок розробляється у форматі веб-застосунку, що дозволяє отримати доступ до нього з будь-якого пристрою, який має браузер: настільних комп’ютерів (Windows, macOS, Linux), планшетів чи смартфонів (Android, iOS). Завдяки цьому користувачі можуть взаємодіяти із системою незалежно від типу операційної системи чи пристрою, що робить застосунок гнучким і зручним у використанні [30].

Завдяки використанню сучасних технологій, таких як PHP, MySQL, Laravel і WordPress, наш веб-застосунок є мультикросплатформеним, високопродуктивним та надійним, що повністю відповідає вимогам сучасних криптобірж.

Архітектура програмного забезпечення.

Файлова структура web-застосунку представлена на рисунку 3.1 – 3.4:

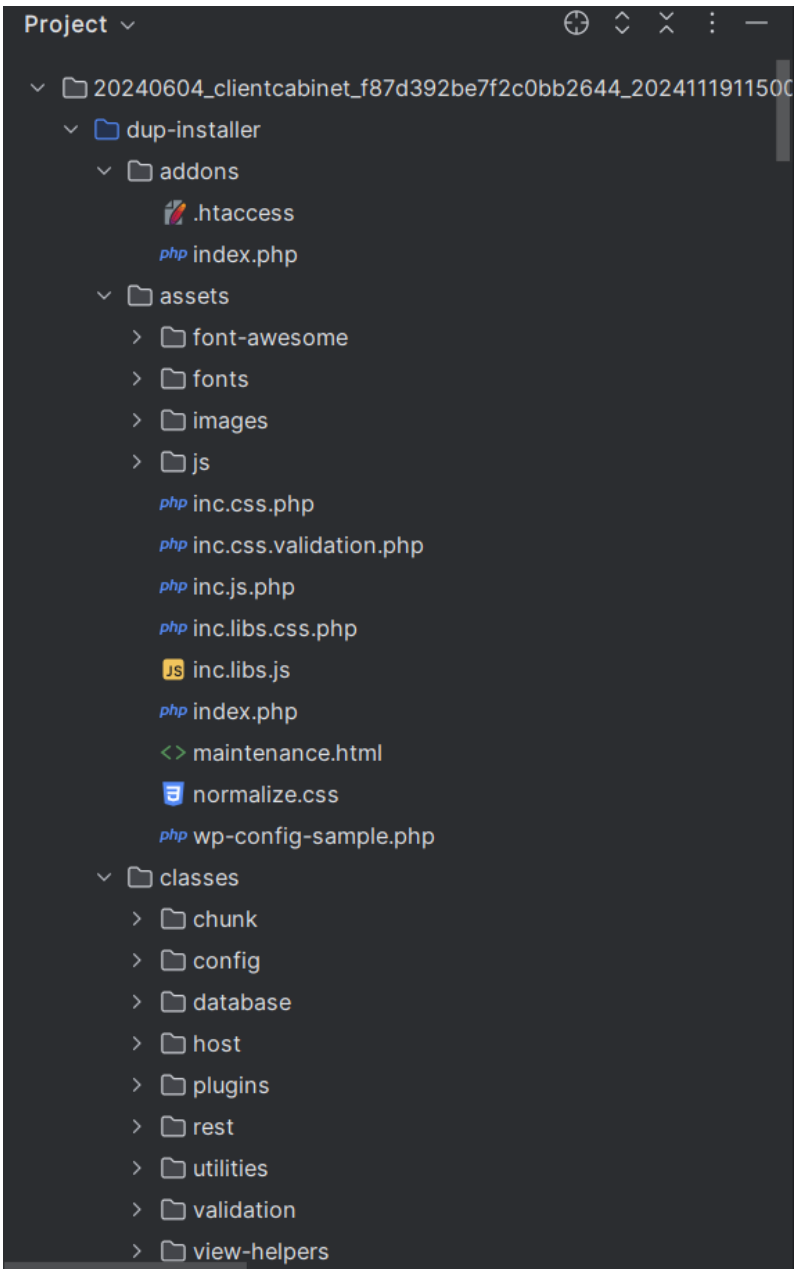


Рис. 3.1 Файлова структура розробленого застосунку

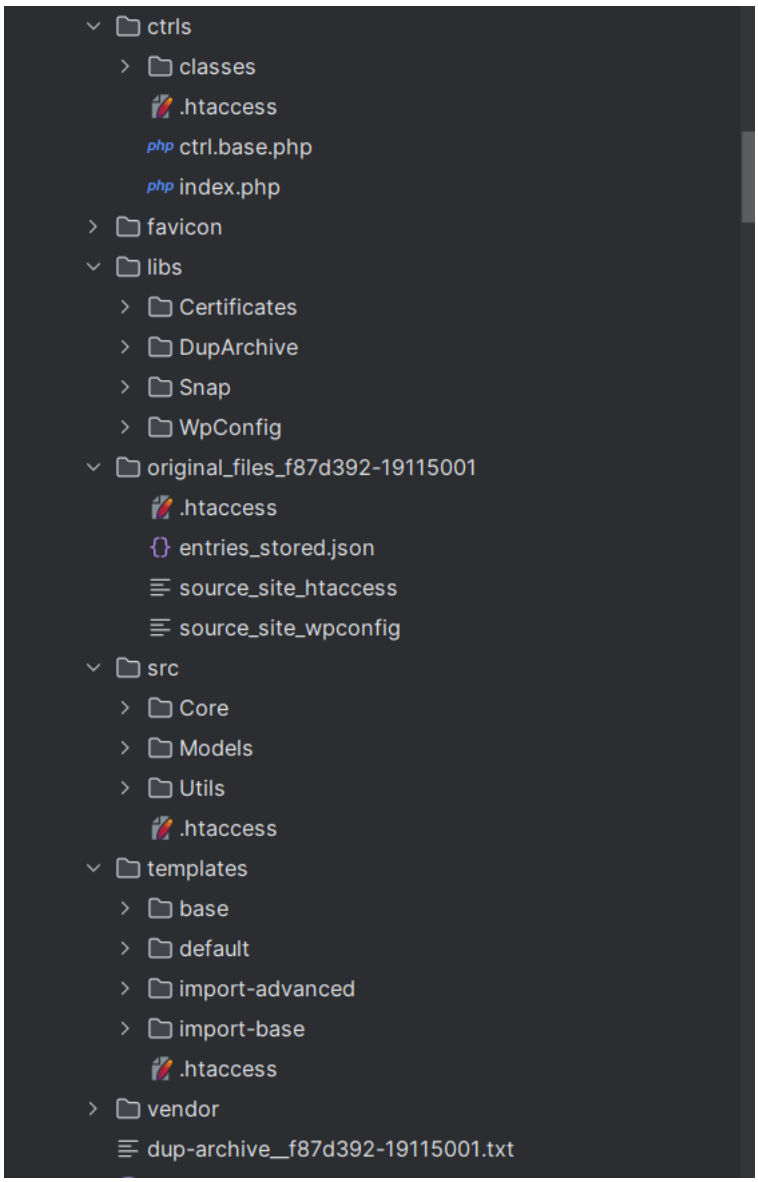


Рис. 3.2 Файлова структура розробленого застосунку (продовження)

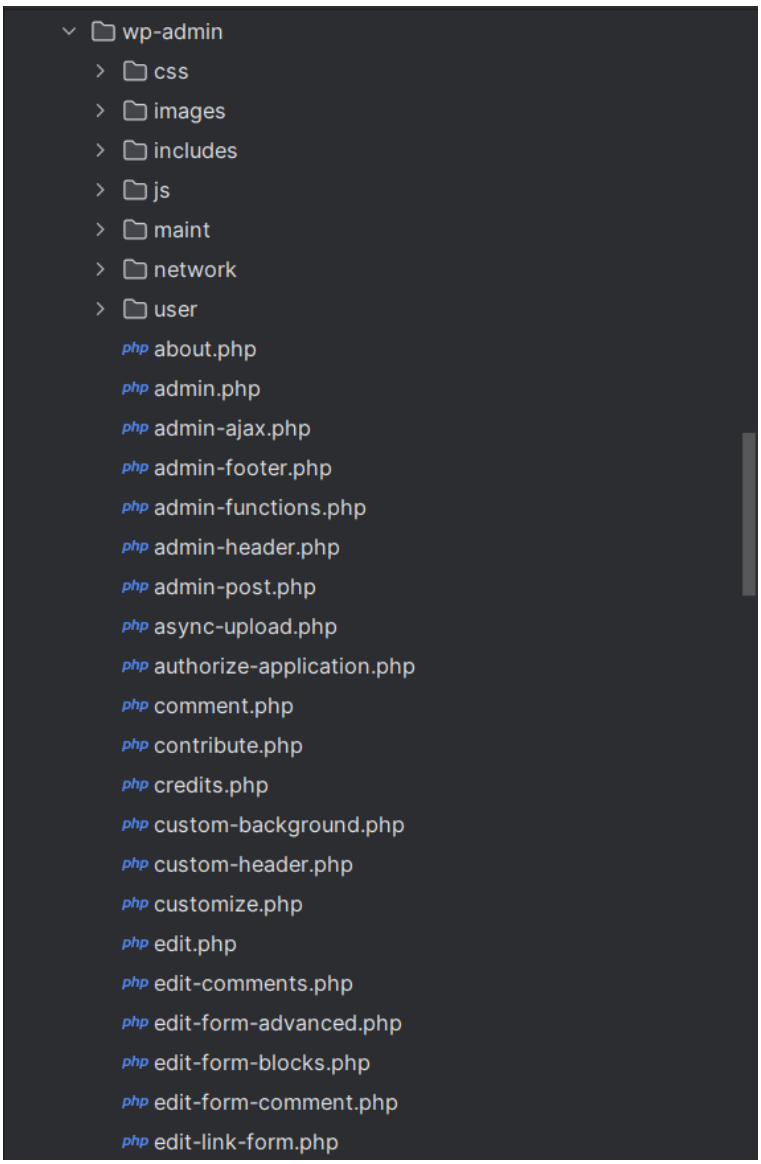


Рис. 3.3 Файлова структура розробленого застосунку (продовження)

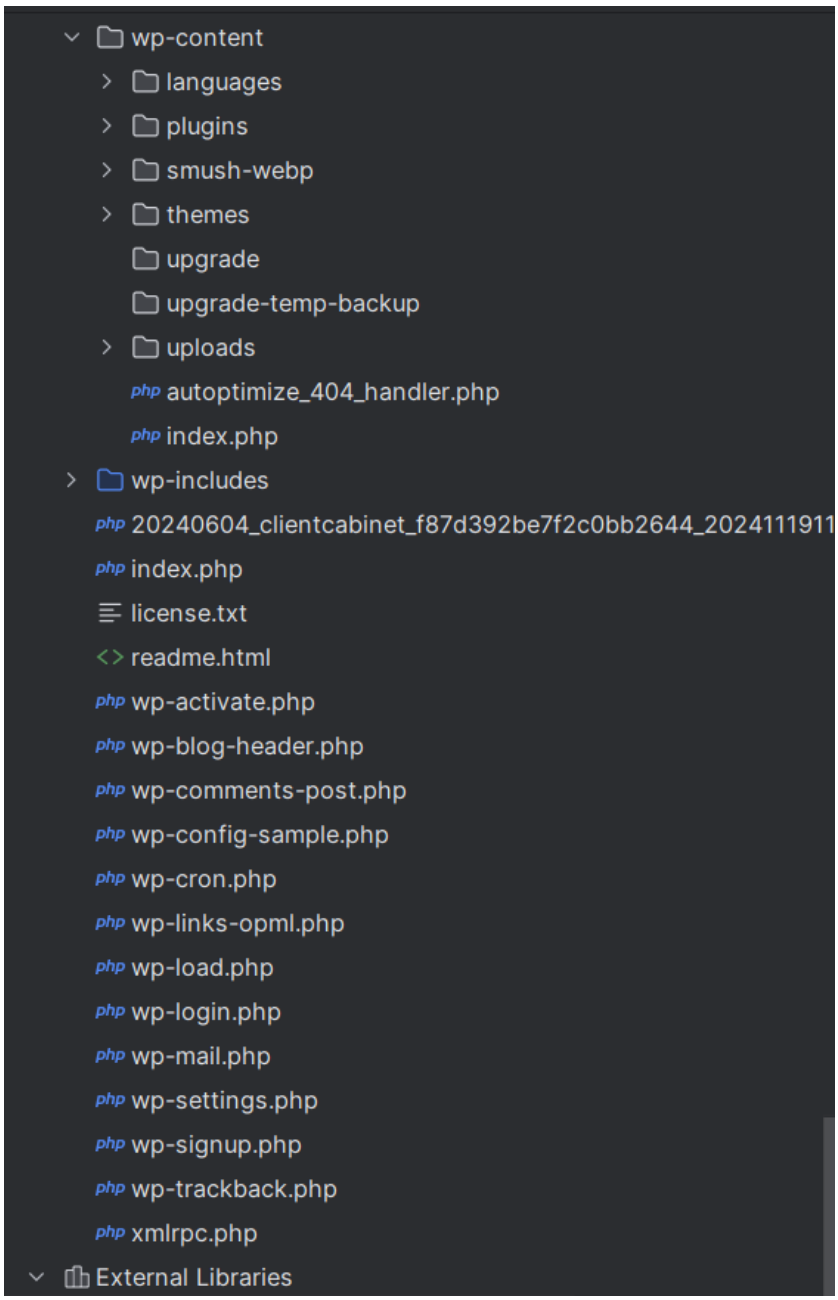


Рис. 3.4 Файлова структура розробленого застосунку (продовження)

Файлова структура dup-installer.

Директорія addons.

addons/.htaccess

Цей файл обмежує доступ до PHP-файлів у директорії для захисту від несанкціонованого доступу.

Код:

<Files *.php>

Order Deny,Allow

Deny from all

</Files>

<Files index.php>

Order Allow,Deny

Allow from all

</Files>

- Захищає всі файли PHP, крім index.php.
- Гарантує, що тільки дозволені скрипти можуть виконуватися.
- *addons/index.php*

  Файл, що, скоріш за все, відповідає за маршрутизацію або виконує простий перехоплювач для захисту директорії від прямого перегляду її вмісту.

  Код:

  <?php

  // Silence is golden.

Директорія *assets.*

- Містить ресурси для стилізації та шрифтів.

  assets/fon-awesome

  css/all.min.css

- Об'єднаний файл стилів Font Awesome, що використовується для додавання іконок у проект.
- Використовується для забезпечення візуальної привабливості.
- css/index.php

Захисний файл:

<?php

// silent

- Захищає директорію від прямого доступу до її файлів.

assets/webfonts

- Містить шрифти Font Awesome, що відповідають за відображення іконок у різних форматах:
- *fa-brands-400.woff, fa-brands-400.woff2* — шрифти для брендових іконок.
- *fa-regular-400.woff, fa-regular-400.woff2* — звичайні іконки.
- *fa-solid-900.woff, fa-solid-900.woff2* — іконки з жирним стилем.
- index.php - Захисний файл для цієї директорії.
- Директорія dots
- *dots-font.css* — CSS-стилі для шрифту "Dots".
- *dotsfont.eot, dotsfont.svg, dotsfont.ttf, dotsfont.woff, dotsfont.woff2*
- Різні формати шрифту "Dots", щоб забезпечити сумісність з усіма браузерами.

Директорія *images.*

- Містить зображення, які були використані у проекті.
- Ролі: логотипи, графічні елементи для UI, декоративні ілюстрації.

Директорія *js.*

Містить основні JavaScript-файли для роботи веб-застосунку:

inc.css.php, inc.css.validation.php - Генерація стилів CSS динамічно або їх валідація

inc.js.php, inc.libs.js.php - Динамічне створення або підключення JavaScript-бібліотек.

index.php - Захисний файл для запобігання несанкціонованого доступу.

maintenance.html - Сторінка, яка відображається під час обслуговування веб-сервера чи застосунку.

*normalize.cs*s - CSS-файл для уніфікації стилів у різних браузерах, що підвищує кросбраузерну сумісність.

wp-config-sample.php - Шаблон конфігураційного файлу для інтеграції з WordPress.

Взаємодія та роль у проекті.

1. Стилізація та UI.
- Директорія assets забезпечує візуальну частину проекту.
- fon-awesome і fonts відповідають за іконки та спеціальні шрифти.
- normalize.css уніфікує стилі між браузерами.
2. Функціональність.
- JavaScript-файли (js) реалізують динамічну поведінку інтерфейсу.
- Файли PHP у addons можуть виконувати функції бекенду, такі як обробка запитів чи налаштування.
3. Безпека.
- Файли .htaccess та index.php у кожній директорії обмежують доступ, запобігаючи прямому перегляду файлів і каталогу.
4. Кросплатформенність.
- Завдяки веб-архітектурі, застосунок працює у будь-якому браузері.
5. Сумісність із CMS WordPress.
- Наявність wp-config-sample.php свідчить про інтеграцію або сумісність із WordPress, що спрощує управління контентом.

Реалізація механізму розбиття великих завдань на ітерації, збереження та відновлення стану.

Клас DUPX_ChunkingManager є абстрактним класом, який надає базову логіку для обробки великих завдань шляхом розбиття їх на менші частини або "шматки" (chunks), що можуть бути виконані окремими запитами. Це дозволяє ефективно обробляти складні задачі, не перевантажуючи систему. Основними

функціями класу є управління ітераціями, збереження та відновлення стану між ітераціями, а також відстеження прогресу виконання завдання. Клас надає абстрактні методи, які підкласам дозволяють визначити специфічні дії для кожної ітерації, взаємодіяти з ітераторами і зберігати дані. Це дає змогу реалізувати деталі для кожного конкретного типу задачі в залежності від її вимог [31].

Клас DUPX_ChunkingManager_file розширює DUPX_ChunkingManager, додаючи можливість зберігання стану задачі в JSON-файлі. Це важлива особливість для реалізацій, які потребують збереження даних на файловій системі, що дає змогу відновити виконання завдання у випадку його переривання. Методи saveStoredData та getStoredData відповідають за збереження та відновлення стану задачі в JSON-форматі, причому метод deleteStoredData реалізує механізм видалення цих даних. Це дозволяє забезпечити безперервність виконання задачі навіть у разі аварійних зупинок або переривань.

Клас DUPX_chunkS3Manager є підкласом DUPX_ChunkingManager_file, який спеціалізується на виконанні завдань, пов'язаних з обробкою даних для S3-сховищ. Він реалізує специфічну логіку для кожного етапу виконання завдання, враховуючи специфіку роботи з S3. Залежно від поточного етапу завдання, метод action виконує різноманітні операції, такі як очищення бази даних, пошук і заміну значень у таблицях, або оновлення конфігурацій через методи класу DUPX_S3_Funcs. Метод getProgressPerc дозволяє оцінити прогрес виконання завдання в процентах, що робить відображення статусу виконання завдання зручним і наочним для користувача (рис. 3.5).

Ітератор DUPX_s3_iterator забезпечує перебір етапів виконання завдання, працюючи у тандемі з класом DUPX_chunkS3Manager, щоб розбивати складний процес на менші підзадачі. Він надає методи для роботи з поточним станом і переміщенням між "шматками" даних, наприклад, методи getPosition і gSeek. Це дозволяє точно контролювати, на якому етапі виконання знаходиться завдання і яке конкретне підзавдання потрібно виконати наступним.

Вони взаємодіють між собою, щоб забезпечити ефективне, масштабоване виконання великих задач. Спочатку ініціалізується екземпляр класу

DUPX_chunkS3Manager, який визначає параметри ітерацій, такі як максимальна кількість ітерацій, таймаут і затримка між ітераціями. Після кожного кроку виконується збереження поточного стану через методи класу DUPX_ChunkingManager_file, що дозволяє відновити виконання у разі переривання. Клас DUPX_chunkS3Manager виконує конкретні дії залежно від поточного етапу через метод action, а прогрес оцінюється за допомогою методу getProgressPerc, що дає змогу відображати поточний статус виконання завдання в реальному часі. Якщо завдання було перервано, метод getStoredData відновлює його з того місця, де виконання було зупинено, що забезпечує безперервність роботи системи [32].

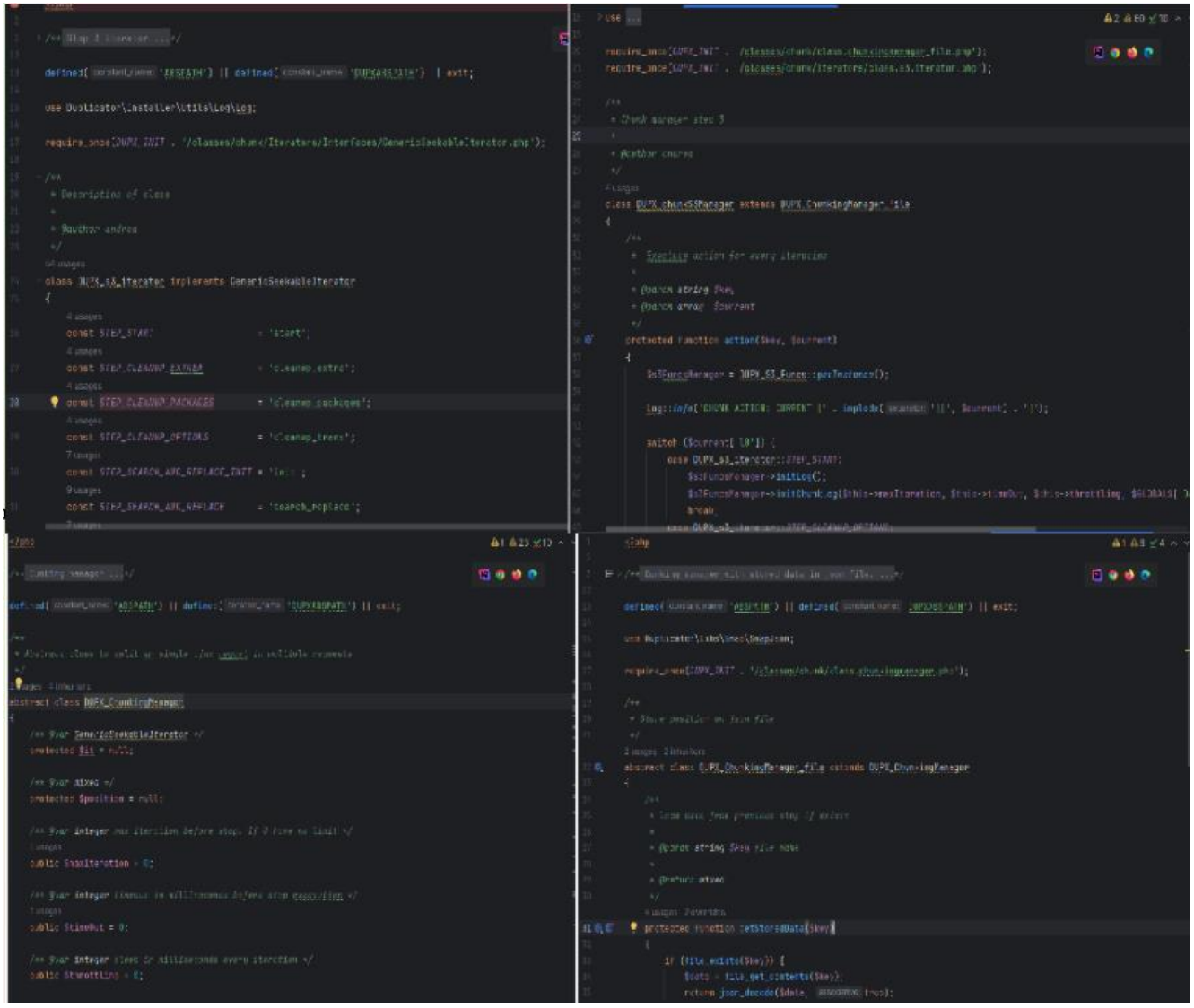


Рис. 3.5 Структура класів елементу Iterators

Інтерфейс EventIterator:

Визначає методи для роботи з подіями в контексті ітерації. Він розширює стандартний інтерфейс Iterator (рис. 3.6), що дозволяє об'єктам цього типу бути ітерабельними (підтримувати операції, як-от перебір елементів за допомогою циклів). Крім стандартних методів ітератора, інтерфейс включає два специфічних методи для обробки подій. Перший метод, setEventCallback, дозволяє встановити зворотний виклик (callback) для обробки подій. Цей callback приймає три параметри: саму подію, її індекс і поточне значення. Другий метод, doEvent, фактично виконує подію, використовуючи зворотний виклик, передаючи подію, ключ і значення як параметри для обробки.

```php
<?php

/** Standard: PSR-2 ...*/

defined( constant_name: 'ABSPATH') || defined( constant_name: 'DUPXABSPATH') || exit;

/**
 * Event iterator
 */
interface EventIterator extends Iterator
{
    /**
     * The callback accept 3 params
     * $event , $index and $current values
     *
     * @param null | callable $callback
     */
    no usages
    public function setEventCallback($callback = null);

    /**
     * Execute event
     *
     * @param mixed $event
     * @param mixed $key
     * @param mixed $current
     */
    no usages
    public function doEvent($event, $key, $current);
}
```

Рис. 3.6 Структура інтерфейсу EventIterator

Інтерфейс GenericSeekableIterator:

Інтерфейс розширює стандартний інтерфейс Iterator і додає додаткові методи для керування ітерацією. Метод gSeek дозволяє переміщувати ітератор до конкретної позиції в колекції, метод getPosition повертає поточну позицію, а

stopIteration забезпечує звільнення ресурсів після завершення ітерації. Метод itCount дозволяє отримати загальну кількість ітерацій. Це дає більшу гнучкість при роботі з колекціями, де необхідно не лише перебирати елементи, а й керувати позицією в середині ітерації.

Сегмент Headers наведений в рисунку 3.7.

1. DupArchiveHeaderU

Цей клас містить утиліти для читання стандартних полів заголовків з архіву. В основному використовується іншими класами для зчитування конкретних полів з архівних заголовків (наприклад, мітки часу, розміри файлів та інші метадані). Клас має метод readStandardHeaderField, який дозволяє прочитати конкретне поле заголовка з архіву, перевіряючи правильність його початкового та кінцевого маркерів.

2. DupArchiveReaderHeader

Це базовий клас для читання заголовка архіву. Він відповідає за зчитування метаданих архіву, таких як версія та інформація про стиснення. Клас використовує метод readFromArchive, щоб зчитати ці дані з архіву, перевіряючи коректність маркерів на початку і в кінці заголовка архіву. Цей клас є основою для інших класів, які працюють з конкретними частинами архіву.

3. DupArchiveHeader

Цей клас відповідає за заголовок архіву, який зберігає версію та інформацію про те, чи є архів стиснутим. Він створюється через статичний метод create, що встановлює ці параметри. Після створення, заголовок можна записати в архів через метод writeToArchive. Клас зазвичай використовується при початковому створенні архіву, коли потрібно записати метадані архіву, такі як версія та параметр стиснення.

4. DupArchiveReaderDirectoryHeader

Цей клас відповідає за читання заголовка директорії в архіві. Він включає метадані про директорію, такі як час модифікації, дозволи на доступ, довжину і шлях відносного файлу. Клас використовує метод readFromArchive, щоб зчитати ці дані з архіву, перевіряючи відповідні маркери. Він також дозволяє пропускати

деякі елементи заголовка, якщо потрібно. Цей клас зчитує і обробляє заголовки директорій, які є частинами архіву.

5. DupArchiveFileHeader

Цей клас відповідає за заголовок файлів в архіві. Він містить метадані про файл, такі як розмір, час модифікації, дозволи та хеш для перевірки цілісності. Клас має два методи створення заголовка: createFromFile, який створює заголовок з фізичного файлу, та createFromSrc, який дозволяє створити заголовок на основі даних з рядка. Методи запису заголовка в архів і зчитування його з архіву також доступні через методи цього класу.

6. DupArchiveReaderFileHeader

Клас, що відповідає за зчитування заголовків файлів з архіву. Він зчитує всі метадані, пов'язані з файлом, включаючи розмір, час модифікації, дозволи і хеш. Використовує метод readFromArchive, щоб зчитати ці дані з архіву, а також дає можливість пропускати вміст файлів при необхідності. Це дає можливість контролювати, які частини архіву будуть зчитані чи пропущені.

7. DupArchiveGlobHeader

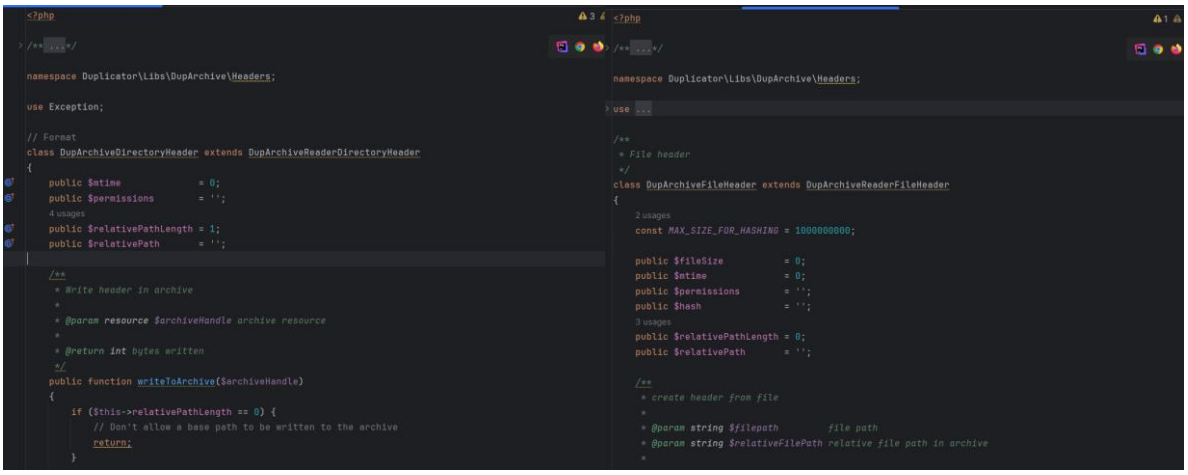
Цей клас відповідає за заголовки глобальних блоків (glob), які можуть містити частини великих файлів або даних в архіві. Він зберігає метадані про розміри оригінального та збереженого блоку та хеш для перевірки цілісності. Клас має методи для зчитування цих заголовків з архіву та для пропуску вмісту глобальних блоків, якщо це потрібно. Використовується в поєднанні з іншими класами для обробки великих файлів чи розділених блоків в архіві.

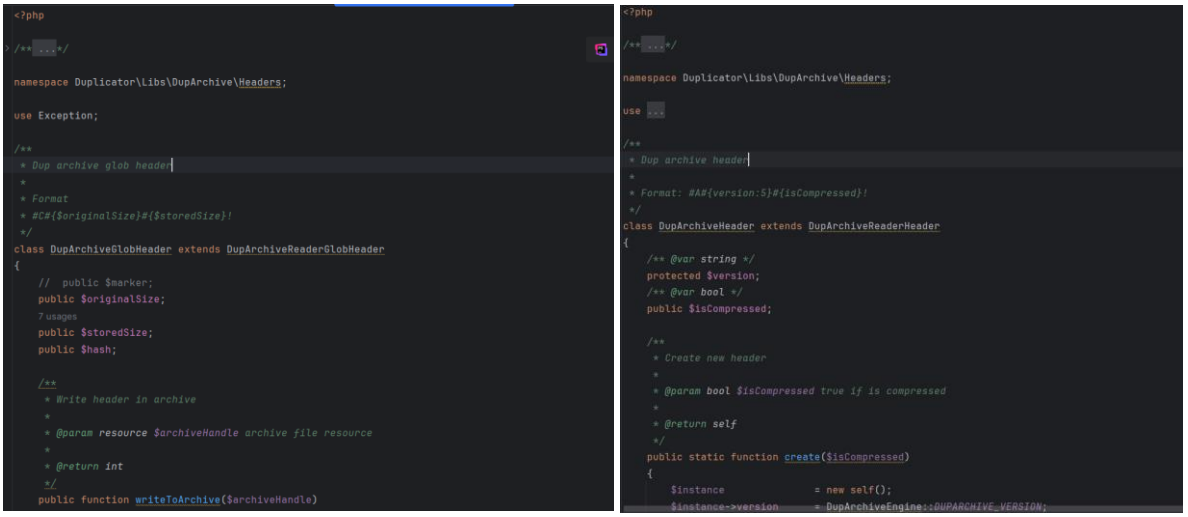
8. DupArchiveFileHeader

Цей клас є доповненням до DupArchiveReaderFileHeader, і зберігає заголовок файлу в архіві з даними, як розмір, дата модифікації та хеш. Клас також підтримує механізми для роботи з великими файлами, включаючи перевірку цілісності через хешування, та запис цих заголовків в архів.

Взаємодія класів:

- DupArchiveHeader і DupArchiveReaderHeader взаємодіють для того, щоб записати або прочитати основні метадані архіву, як-от версію та параметр стиснення.
- DupArchiveFileHeader, DupArchiveReaderFileHeader, DupArchiveGlobHeader та DupArchiveReaderDirectoryHeader є класами для роботи з конкретними частинами архіву: файлами, директоріями та блоками даних. Вони використовують DupArchiveHeaderU для зчитування стандартних полів заголовків.
- DupArchiveReaderFileHeader і DupArchiveReaderGlobHeader мають схожі методи для зчитування даних з архіву, але обробляють різні типи заголовків (файли і глобальні блоки).
- Кожен клас для заголовків (файл, директорія, глобальний блок) містить методи для зчитування або запису в архів, що дозволяє використовувати ці класи як частину загальної архівної системи для маніпуляції файлами та даними архіву.





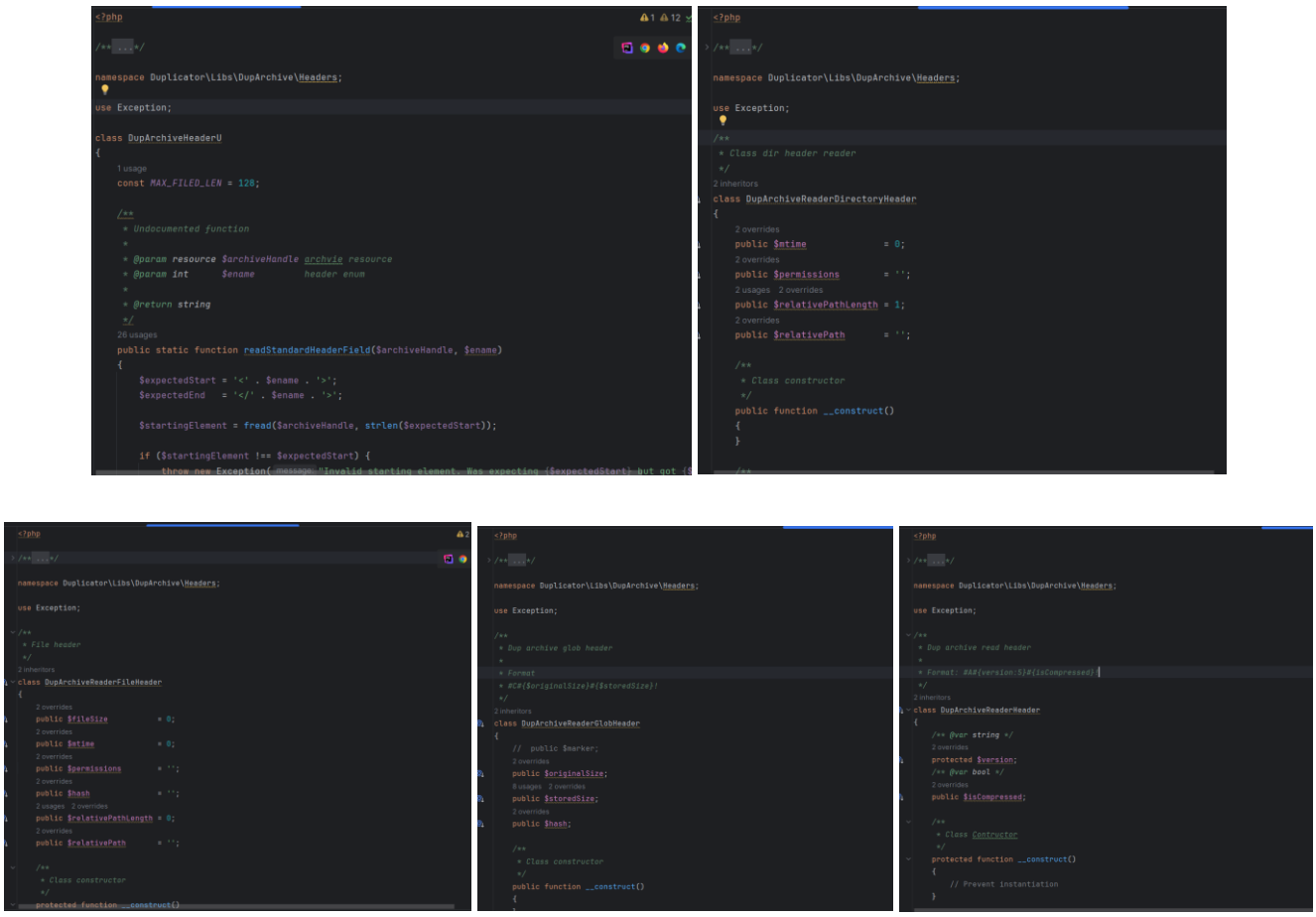


Рис. 3.7 Структура класів елементу Headers

Сегмент Info

1. DupArchiveExpanderInfo

- Цей клас служить для збереження інформації під час розпакування архіву з використанням класу Duplicator. Він зберігає handle архіву ($archiveHandle), поточний заголовок файлу, шлях призначення ($destDirectory), а також кількість записів до директорії ($directoryWriteCount) і файлів ($fileWriteCount).
- Також клас містить прапор, який вказує, чи архів стиснутий ($isCompressed), і прапор, який дозволяє записувати до архіву ($enableWrite).
- Метод getCurrentDestFilePath() повертає шлях до поточного призначення файлу, що розпаковується, в архіві. Якщо значення властивості $destDirectory дорівнює null, метод поверне null, інакше він поверне абсолютний шлях до розпакованого файлу.

2. *DupArchiveReaderFileHeader*

- Цей клас представляє заголовок файлу в архіві. Кожен заголовок файлу містить інформацію про розмір файлу ($fileSize), дату та час модифікації ($mtime), права доступу ($permissions), хеш-функцію файлу ($hash) та шлях до файлу відносно архіву ($relativePath).
- Метод readFromArchive() дозволяє прочитати заголовок файлу з архіву, розпізнавши початковий елемент (<F>). Після розпізнавання початкового елемента, метод витягує значення кожної властивості заголовку (розмір файлу, мітка часу, права доступу, хеш, відносний шлях) з архіву.
- Якщо встановлений прапор $skipContents, метод може пропускати вміст файлу, залишаючи лише заголовок.

Загалом, взаємодія між цими двома класами відбувається під час процесу розпакування архіву (рис. 3.9). Об'єкт DupArchiveExpanderInfo використовується для збереження поточних даних про архів, в той час як об'єкт DupArchiveReaderFileHeader забезпечує доступ до інформації про окремі файли всередині архіву. Під час роботи з архівом, об'єкт DupArchiveExpanderInfo обирає заголовки файлів із архіву за допомогою методів DupArchiveReaderFileHeader::readFromArchive() та визначає шлях до розпакованого файлу за допомогою методу getCurrentDestFilePath() з об'єкта DupArchiveExpanderInfo. Це дозволяє коректно зберігати файли під час розпаковування архіву в належне місце.

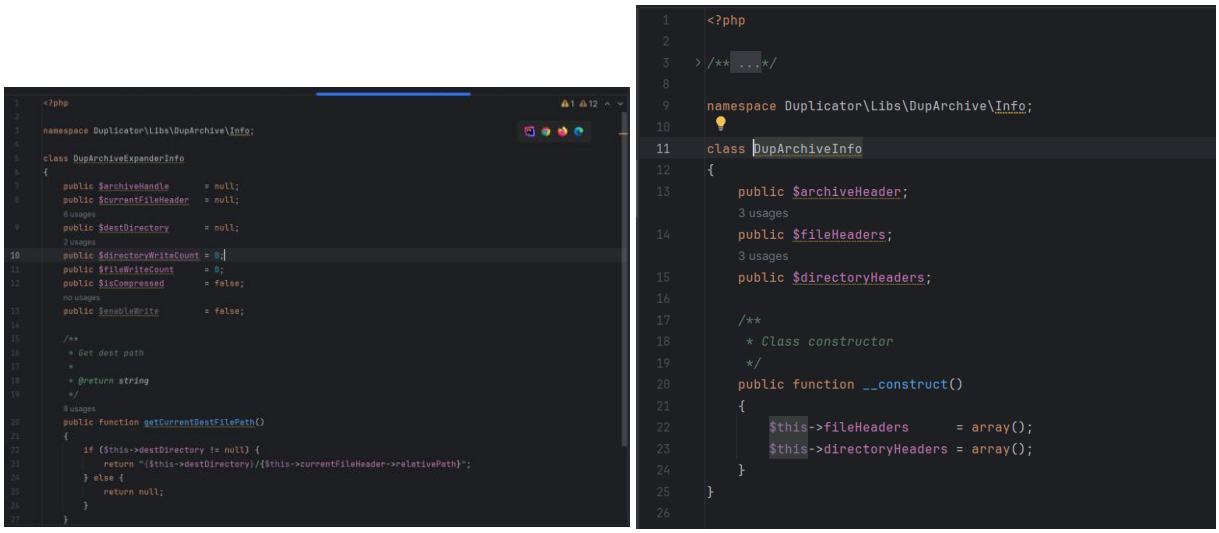


Рис. 3.9 Структура класів елементу Info

Сегмент Processors

Ось опис взаємодії між трьома класами: DupArchiveExpanderInfo, DupArchiveDirectoryProcessor та DupArchiveProcessingFailure.

1. DupArchiveExpanderInfo

Цей клас відповідає за зберігання інформації про поточний архів, зокрема:

- $archiveHandle — дескриптор архіву, який використовується для доступу до архіву.
- $currentFileHeader — заголовок поточного файлу в архіві.
- $destDirectory — шлях до директорії, куди потрібно розпакувати файли.
- Метод getCurrentDestFilePath повертає шлях до поточного файлу в архіві, що дозволяє отримати інформацію про файл, який зараз обробляється.

2. DupArchiveDirectoryProcessor

Цей клас відповідає за обробку та запис директорій в архіви. Основна функція класу — це метод writeDirectoryToArchive, який записує директорії в архів, зберігаючи метаінформацію про директорію (права доступу, час модифікації, відносний шлях). Це дозволяє створювати архіви з необхідною інформацією про директорії та файли в них.

3. DupArchiveProcessingFailure

Цей клас використовується для опису помилок при обробці архіву. Клас містить:

- type — тип помилки (файл або директорія).
- description — опис помилки.
- subject — предмет помилки (файл або директорія).
- isCritical — вказує, чи є помилка критичною.

Взаємодія класів:

- DupArchiveDirectoryProcessor використовує DupArchiveExpanderInfo, щоб отримати інформацію про поточний стан архіву під час запису директорії. Метод writeDirectoryToArchive може звертатися до DupArchiveExpanderInfo для

отримання шляхів до файлів і директорій, а також для визначення метаінформації про архів, яка повинна бути записана в процесі створення архіву.

- Якщо під час обробки архіву виникає помилка (наприклад, файл або директорія не можуть бути оброблені), DupArchiveProcessingFailure створюється для опису помилки. Цей клас дає змогу зберігати деталі про тип помилки, її опис і статус критичності. DupArchiveDirectoryProcessor може використовувати цей клас для реєстрації та обробки помилок, що виникають під час запису архіву (рис. 3.10).

1. DupArchiveDirectoryProcessor ініціалізує процес створення архіву і починає запис директорії. Він використовує DupArchiveExpanderInfo, щоб дістати метаінформацію про поточний файл або директорію.

2. Під час запису архіву можуть виникнути помилки. Якщо це трапляється, DupArchiveProcessingFailure фіксує тип помилки, її опис і інші деталі.

3. Якщо помилка є критичною (наприклад, неможливо записати важливий файл), DupArchiveProcessingFailure може встановити isCritical = true, що дасть змогу прийняти відповідні дії (наприклад, припинити процес або повідомити користувача).

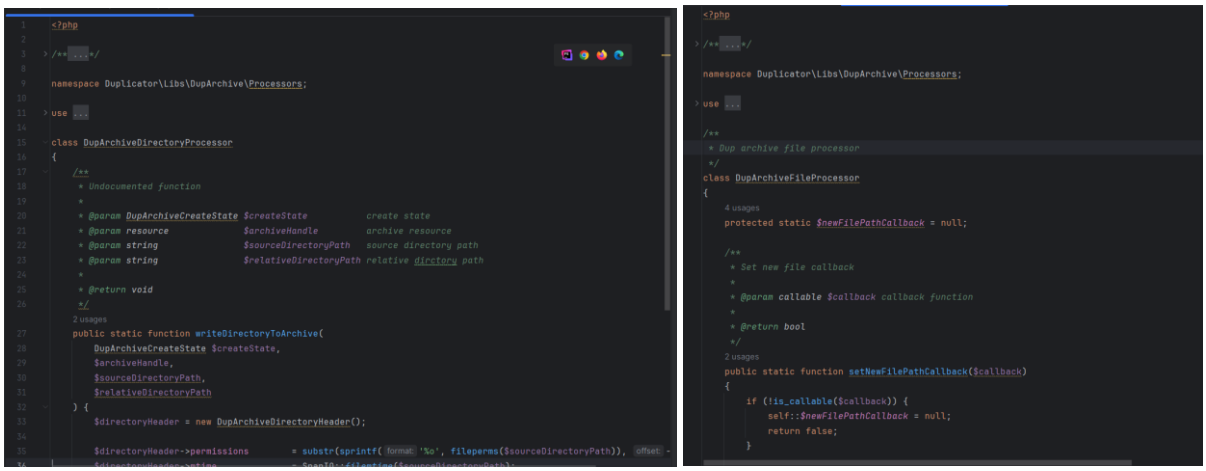
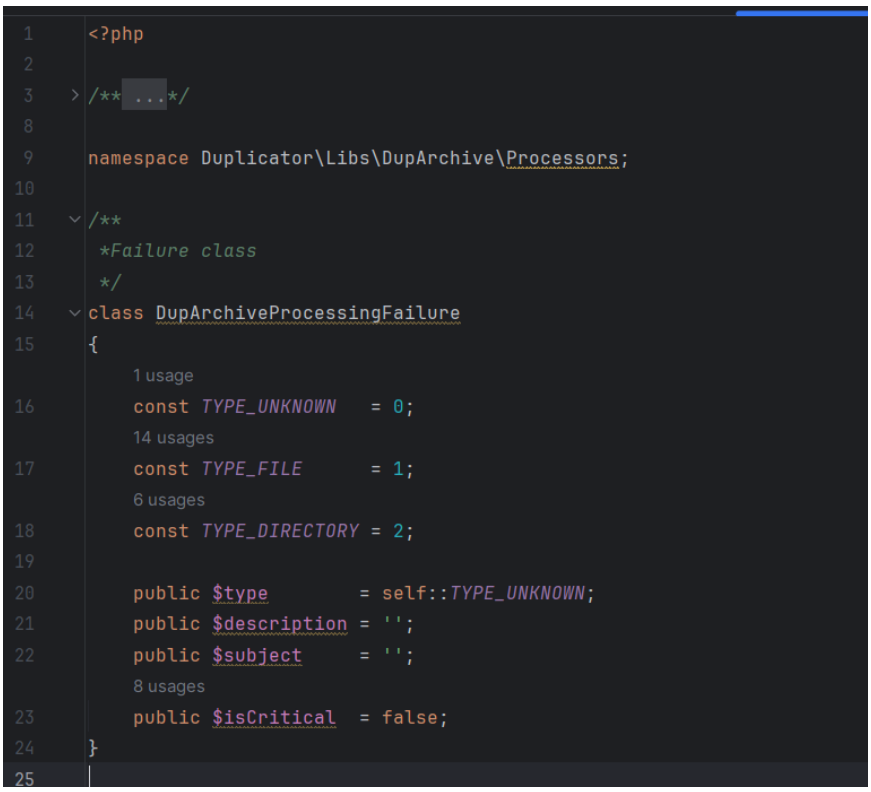


Рис. 3.10 Структура класів елементу Processors



```php
<?php

/** ...*/

namespace Duplicator\Libs\DupArchive\Processors;

/**
 *Failure class
 */
class DupArchiveProcessingFailure
{
    1 usage
    const TYPE_UNKNOWN   = 0;
    14 usages
    const TYPE_FILE      = 1;
    6 usages
    const TYPE_DIRECTORY = 2;

    public $type        = self::TYPE_UNKNOWN;
    public $description = '';
    public $subject     = '';
    8 usages
    public $isCritical  = false;
}
```

Рис. 3.11 Структура класу DupArchiveProcessingFailure елементу Processors

Сегмент States

1. DupArchiveProcessingFailure.

Цей клас відповідає за представлення невдач при обробці архівів. Він зберігає такі параметри:

- type: Тип помилки (файл, директорія тощо).
- description: Опис помилки.
- subject: Тема або об'єкт, до якого відноситься помилка.
- isCritical: Прапор, що вказує, чи є помилка критичною.

Цей клас використовується для зберігання та обробки помилок, які виникають під час операцій з архівом, таких як обробка файлів або директорій.

2. DupArchiveStateBase.

Цей клас є базовим класом для всіх станів архіву. Він містить загальні властивості та методи для роботи з архівами:

- Властивості для зберігання інформації про архів (шлях до архіву, чи він стиснутий, відмітки часу тощо).

- Механізм обробки помилок за допомогою масиву failures та лічильника помилок failureCount.
- Методи для додавання помилок (addFailure), перевірки на критичні помилки (isCriticalFailurePresent), а також зберігання статистики та тайм-аутів для обробки архіву (startTimer, timedOut).

Цей клас є основою для більш специфічних класів, які керують створенням чи розпакуванням архівів.

3. DupArchiveCreateState.

Цей клас є абстрактним класом для стану створення архіву. Він розширює DupArchiveStateBase і додає специфічні властивості та методи для роботи зі створенням архіву:

- Властивості для управління процесом створення архіву, такі як basepathLength, currentDirectoryIndex, currentFileIndex та інші, що використовуються для індексації файлів і директорій.
- Абстрактний метод save(), який повинен бути реалізований у підкласах для збереження поточного стану.

Цей клас вказує на те, що архівування є складним процесом, що вимагає зберігання та обробки великої кількості даних на різних етапах (рис. 3.12).

4. DupArchiveSimpleCreateState.

Цей клас є конкретною реалізацією DupArchiveCreateState і відповідає за стан під час простого створення архіву. В конструкторі класу ініціалізуються властивості, що визначають поточні індекси для директорій та файлів:

- currentDirectoryIndex, currentFileIndex і currentFileOffset.
- Метод save() наразі не реалізовано, але він буде зберігати стан створення архіву.

Цей клас є найпростішим варіантом для обробки створення архіву, де всі індекси стартують з нуля, і немає додаткової складної логіки.

5. DupArchiveExpandState.

Цей клас є абстрактним класом для стану розпакування архіву, також розширює DupArchiveStateBase. Власне, цей клас надає загальні властивості та методи для роботи з процесом розпакування архіву.

- В ньому визначено basepathLength, currentDirectoryIndex, currentFileIndex для індексації при розпакуванні.
- Абстрактний метод save(), що має бути реалізований у підкласах для збереження стану розпакування.

1. Обробка помилок:

- Якщо під час будь-якої операції виникає помилка, клас DupArchiveProcessingFailure створює об'єкт помилки, який додається до масиву failures в класі DupArchiveStateBase.
- Клас DupArchiveStateBase дозволяє перевіряти, чи є критичні помилки через метод isCriticalFailurePresent() та надавати зведену інформацію про помилки через getFailureSummary().

2. Створення архіву:

- Клас DupArchiveCreateState визначає загальні параметри для створення архіву, такі як розмір "блоків" файлів, індексація файлів та директорій. Підкласи цієї бази, наприклад, DupArchiveSimpleCreateState, ініціалізують ці параметри і керують процесом створення архіву.
- Метод save() в підкласах відповідає за збереження поточного стану.

3. Розпакування архіву:

- Клас DupArchiveExpandState і його підкласи (наприклад, DupArchiveSimpleExpandState) працюють з процесом розпакування архіву, використовуючи схожі принципи індексації та обробки помилок, але в зворотному напрямку.

4. Збереження стану:

- Кожен із класів (DupArchiveCreateState і DupArchiveExpandState) має абстрактний метод save(), що дозволяє зберігати поточний стан процесу, будь то створення чи розпакування архіву.

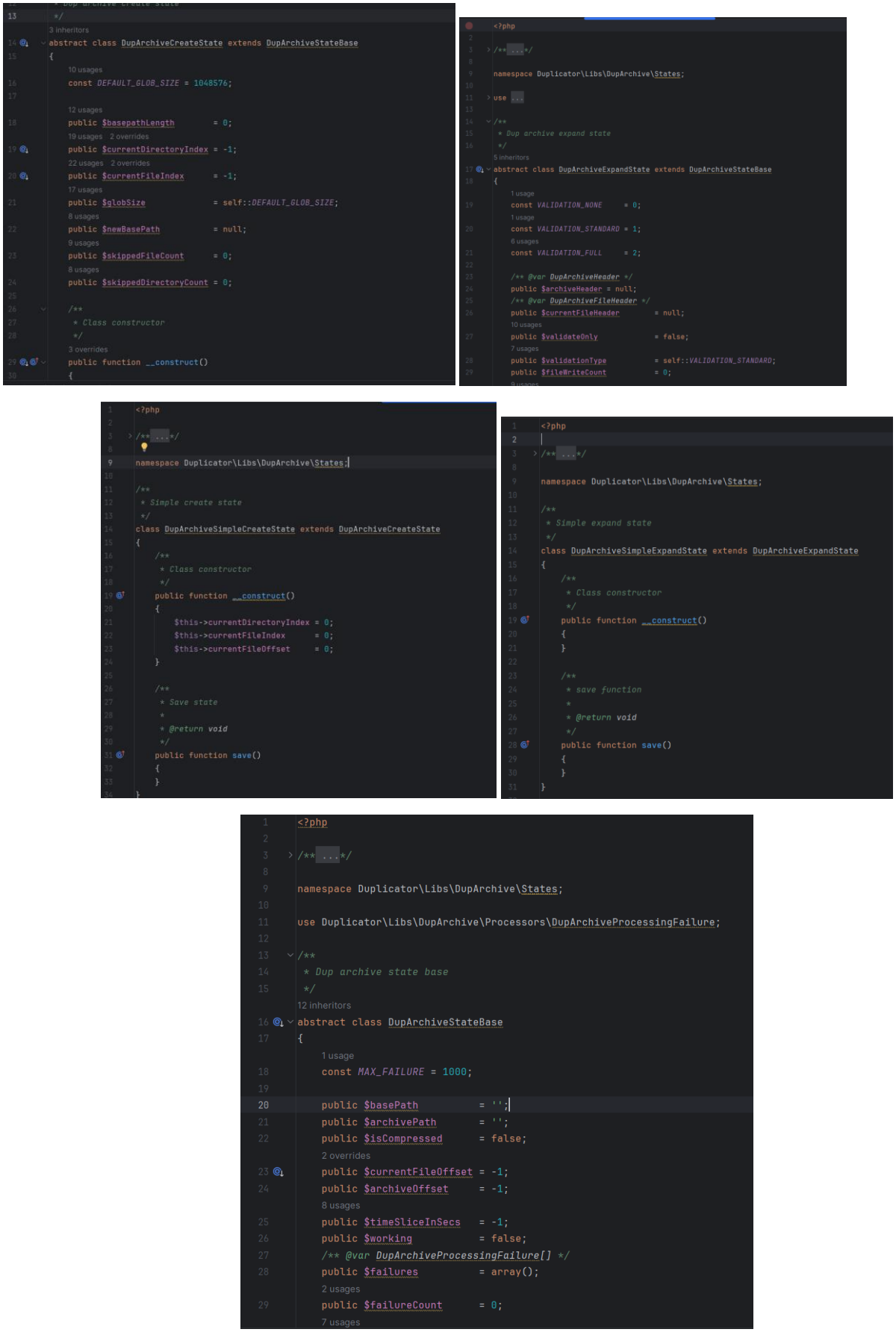


Рис. 3.12 Структура класів елементу States

Сегмент Utils

1. DupArchiveLoggerBase:

✓ Це абстрактний клас, який задає базову структуру для ведення журналу.

✓ Клас має один абстрактний метод log, який використовується для логування повідомлень. Цей метод може бути реалізований у підкласах для конкретних типів журналів, таких як файл або база даних.

✓ Параметр $callingFunctionOverride дозволяє змінювати функцію для зворотного виклику для ведення журналу, надаючи гнучкість у налаштуванні механізмів логування.

2. DupArchiveExpandBasicEngine:

✓ Цей клас відповідає за розпакування архівів і обробку директорій і файлів. Він взаємодіє з DupArchiveLoggerBase через статичний метод log, який викликає логування під час різних етапів процесу розпакування.

✓ Метод log записує повідомлення про процес розпакування, в тому числі успішні операції або помилки. Наприклад, якщо файл успішно розпаковано, буде викликано self::log("Expanding file $filePath");. Якщо виникне помилка під час запису файлу, буде викликано логування помилки.

✓ DupArchiveExpandBasicEngine використовує функцію зворотного виклику log для того, щоб зробити можливим налаштування різних способів ведення журналу (наприклад, логування в файл, базу даних або інші засоби).

3. DupArchiveExpanderInfo:

✓ Цей клас зберігає інформацію, необхідну для розпакування архіву, включаючи поточний файл або директорію, до якої потрібно записати дані, а також властивості, пов'язані з тим, чи є архів стиснутим.

✓ DupArchiveExpanderInfo може бути використаний у класі DupArchiveExpandBasicEngine для визначення поточного шляху файлу, куди потрібно записати вміст, і визначення того, чи потрібен файл для розпакування.

✓ При розпакуванні, якщо виникають помилки, то через метод log з DupArchiveExpandBasicEngine записується інформація про помилки.

4. DupArchiveReaderHeader та інші заголовки:

✓ В класі DupArchiveExpandBasicEngine методи DupArchiveReaderHeader::readFromArchive та інші використовуються для читання заголовків архіву і розпізнавання типів файлів або директорій.

✓ Наприклад, коли зчитується заголовок файлу за допомогою DupArchiveReaderFileHeader::readFromArchive, в разі помилок (якщо не вдається знайти файл або виникає інша проблема), буде викликано логування через log для фіксації помилки.

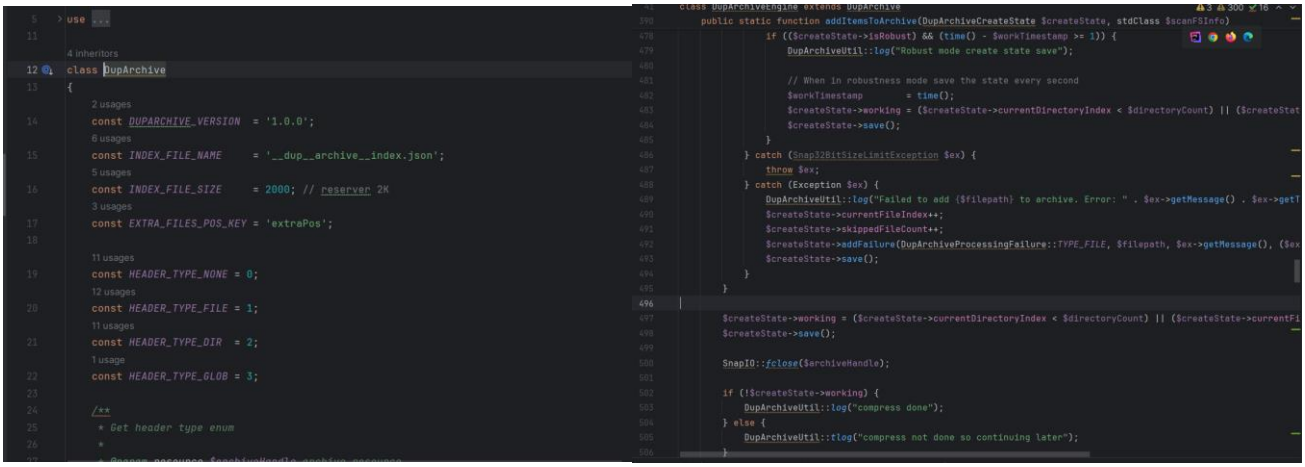
✓ Клас DupArchiveExpandBasicEngine визначає типи заголовків архіву, з якими він працює (наприклад, файли, директорії), і кожна така операція може бути зафіксована в журналі для подальшого аналізу.

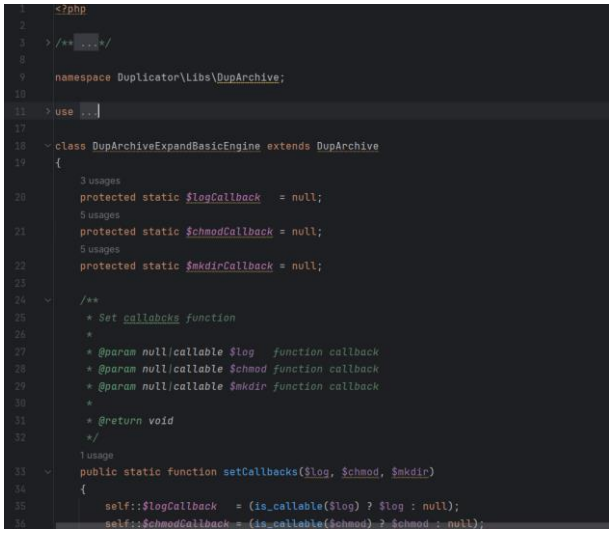
Взаємодія.

1. *Логування.* Основна функція логування в проекті передбачає, що клас DupArchiveExpandBasicEngine буде використовувати метод log для запису важливої інформації про процеси архівації та розпакування. Це може включати як успішні операції, так і помилки.

2. *Обробка архівів.* Клас DupArchiveExpandBasicEngine працює безпосередньо з файлами і директоріями з архіву, використовуючи DupArchiveExpanderInfo для збереження інформації про кожен файл, директорію і поточний стан. Логування допомагає стежити за тим, що відбувається під час цього процесу.

3. *Визначення типів файлів.* Клас DupArchiveReaderHeader і його похідні визначають, які елементи містить архів, дозволяючи DupArchiveExpandBasicEngine виконувати відповідні дії. Логування допомагає контролювати ці етапи і реєструвати будь-які аномалії або важливі моменти (рис. 3.13).





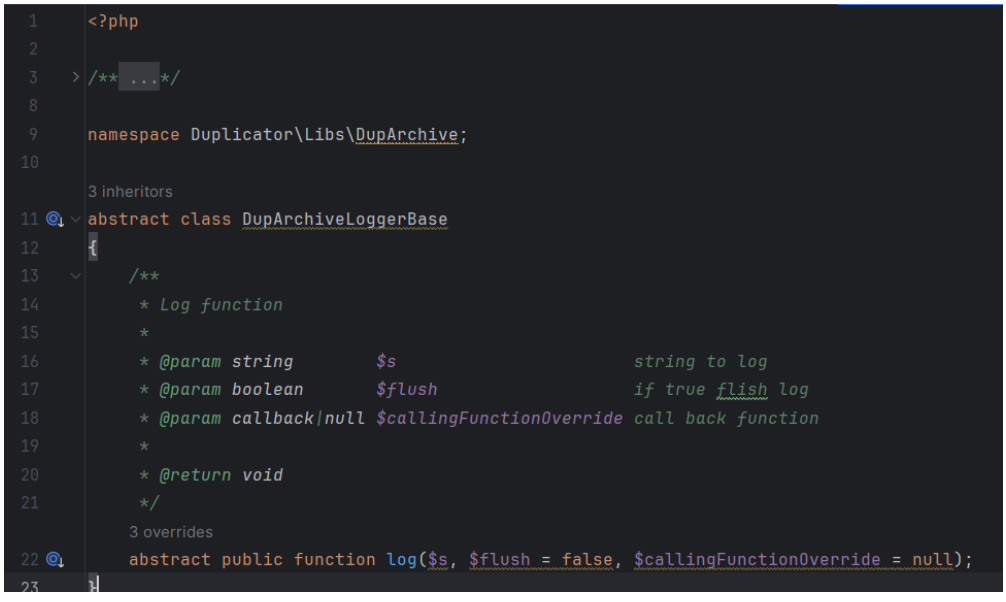


Рис. 3.13 Структура класів елементу Utils

На рисунку 3.14 представлена діаграма класів представлених вище сегментів та порядок їх взаємодії між собою.

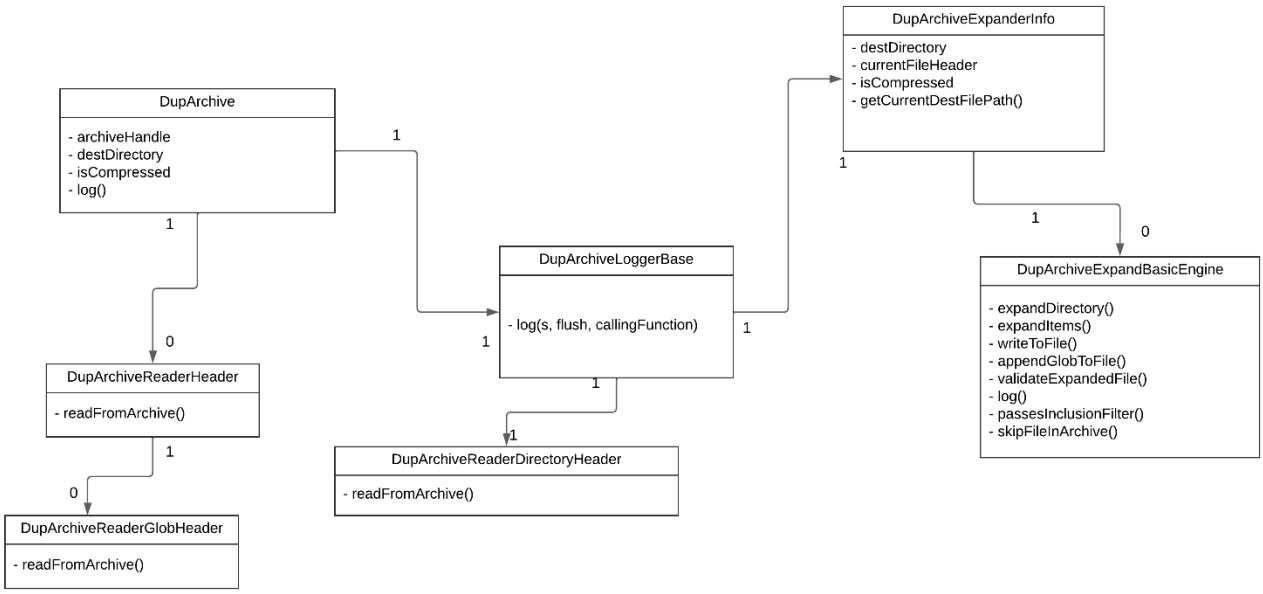


Рис. 3.14 Діаграма класів

3.2 Опис функціонування web-додатку

При переході на web-сторінку нашого застосунку за хостнеймом https://usdt1000.space/ нас зустрічає панель авторизації в додаток, авторизуємось за допомогою адмінської обліковки (рис. 3.15 – 3.16).

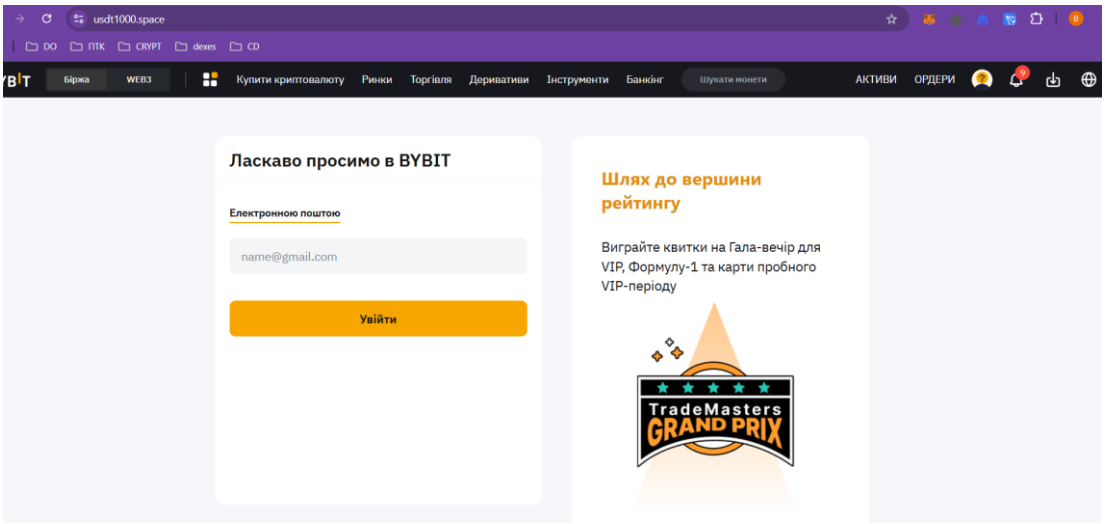


Рис. 3.15 Консоль авторизації в додаток

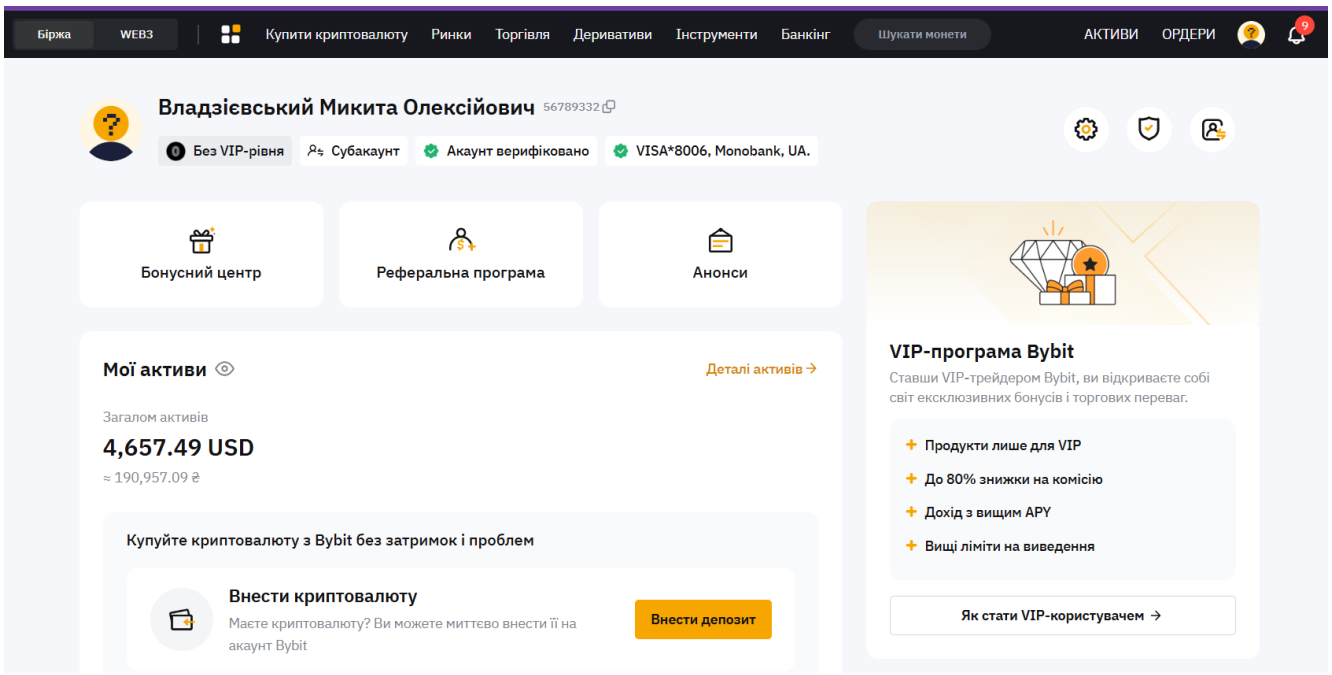


Рис. 3.16 Головний інтерфейс аккаунту

Web-додаток оснований на структурі однієї з головних криптобірж Bybit. Але якщо ми переглянемо наше звдання до кваліфікаційної, основне завдання було зробити для потенційного користувача зручну та оперативну системи онлайн підтримки у вигляді онлайн боту (рис. 3.17).

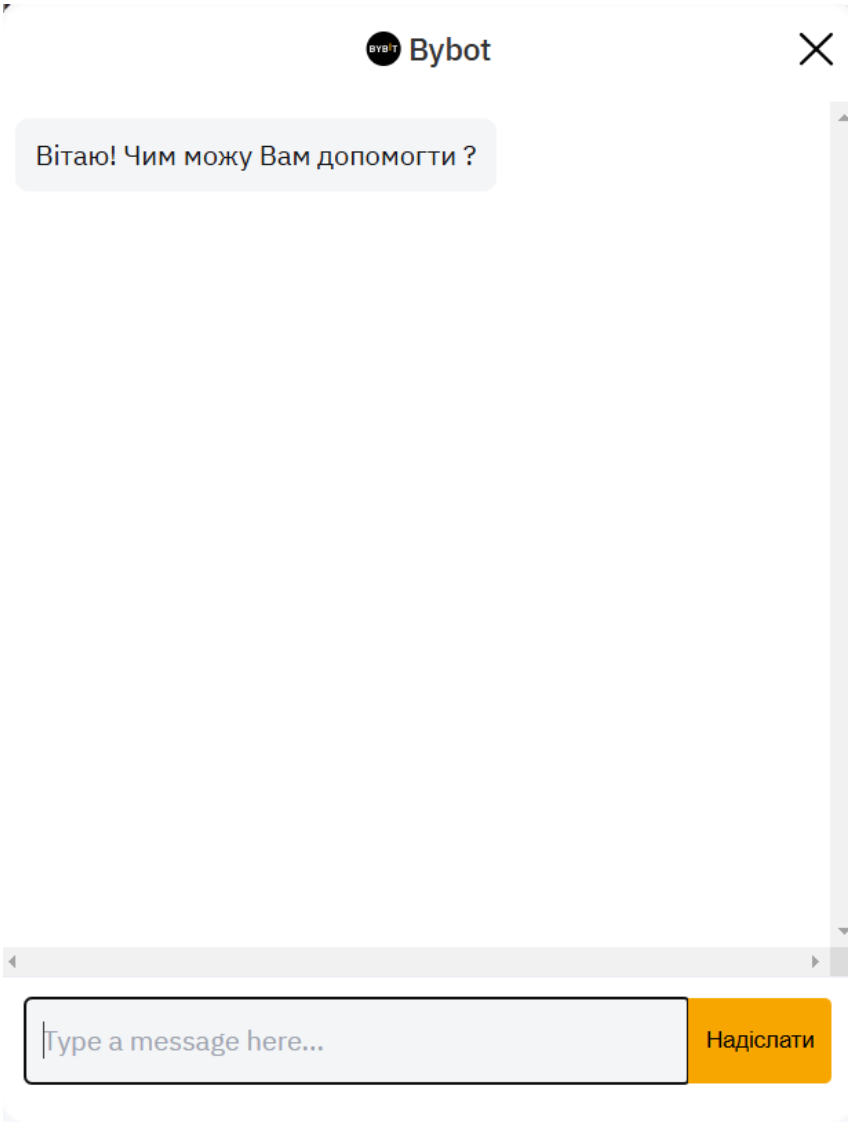


Рис. 3.17 Інтерфейс чату онлайн підтримки додатку

Ми я к клієнт хочемо звернутись за підтримкою до чату, щоб він відповів на наші питання. Особливість чату в тому, що він виступає у вигляді штучного інтелекту, тобто не пропонує теми для вирішення різноманітних питань, так як це забирає у користувача час на вивчення питань, особливо незрозумілою мовою. Тому було розроблено інтелектуальне вивчення питання, та пошук найшвидшого варіанту розв’язування проблеми (рис. 3.18).

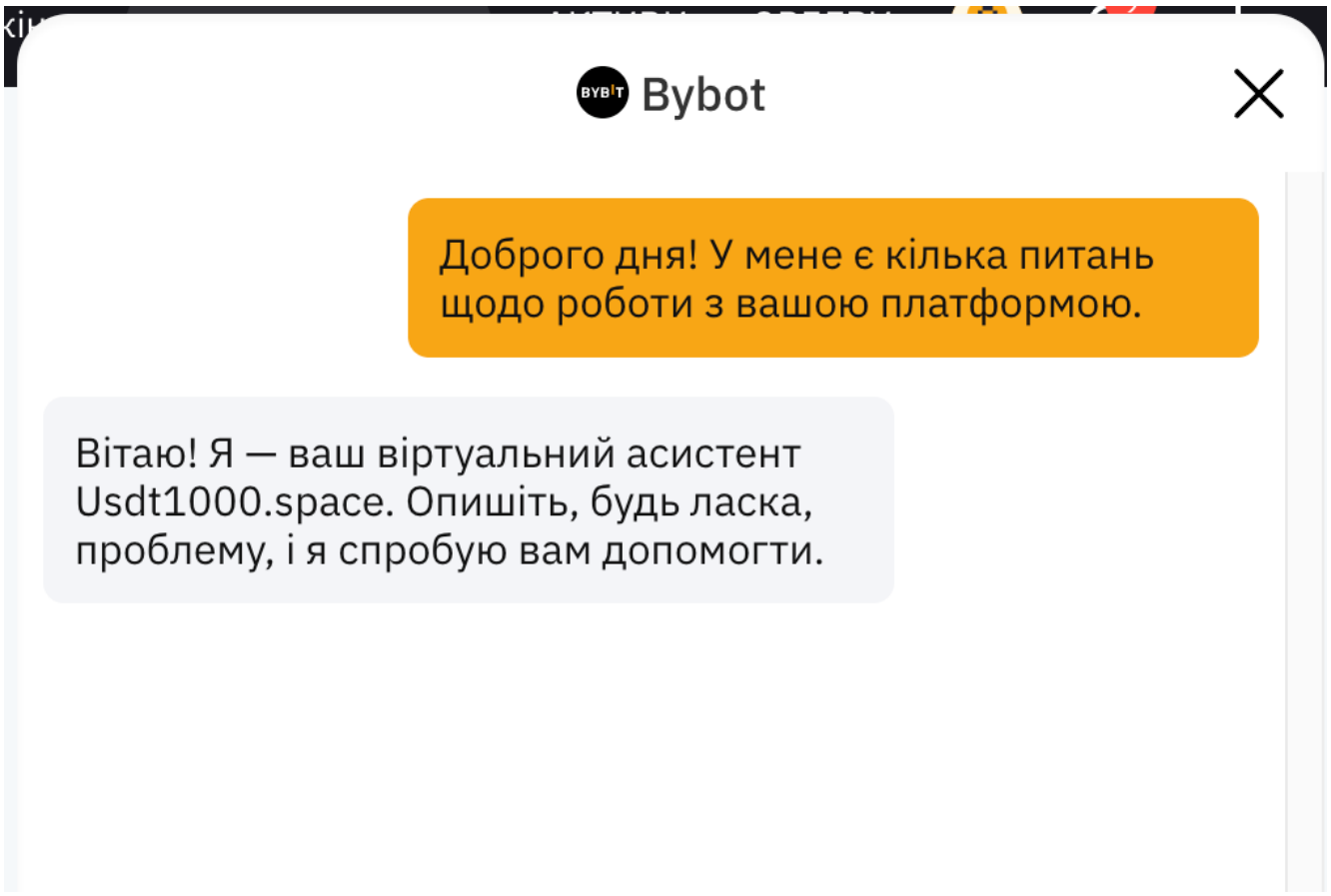


Рис. 3.18 Клієнт задає боту питання

Бот отримав запит, і починає аналізувати питання, очікуємо на відповідь (рис. 3.19).

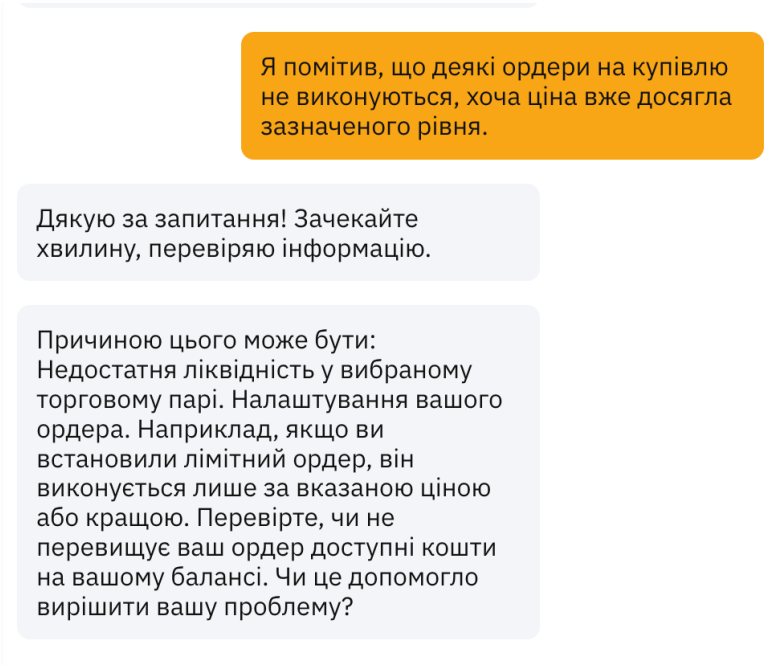


Рис. 3.19 Аналіз запиту та відповідь на питання

Було зроблено акцент на таку річ як, продовження допомоги клієнту відносно його корінного запитання, зазвичай можна стикнутись з такими моментом коли бот, може надати користувачеві посилання на додаткове джерело, а не допомогти продовжити розвивати вирішення та надавати альтернативні варіанти допомоги (рис. 3.20 3.21).

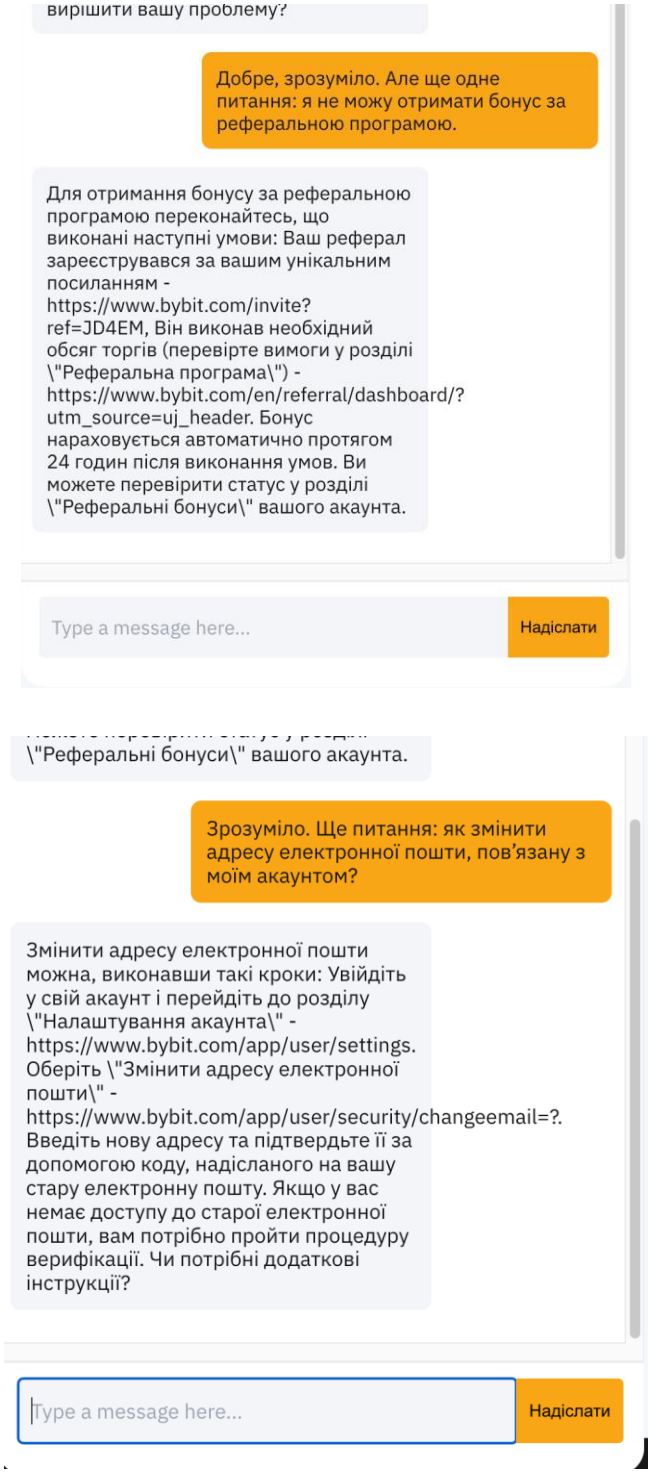


Рис. 3.20 Аналіз запиту та відповідь на питання

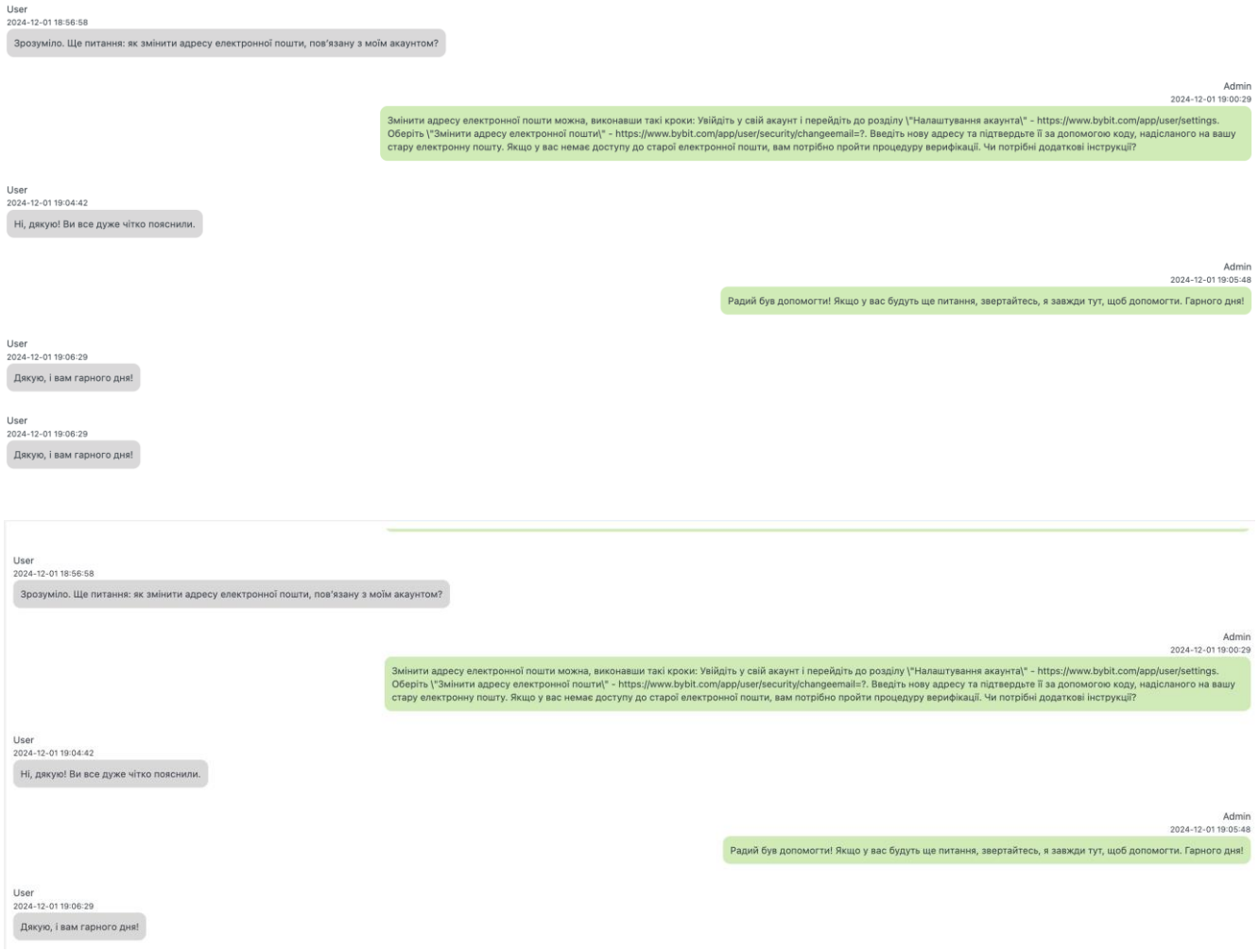


Рис. 3.21 Діалог з клієнтом вигляд серверної сторони

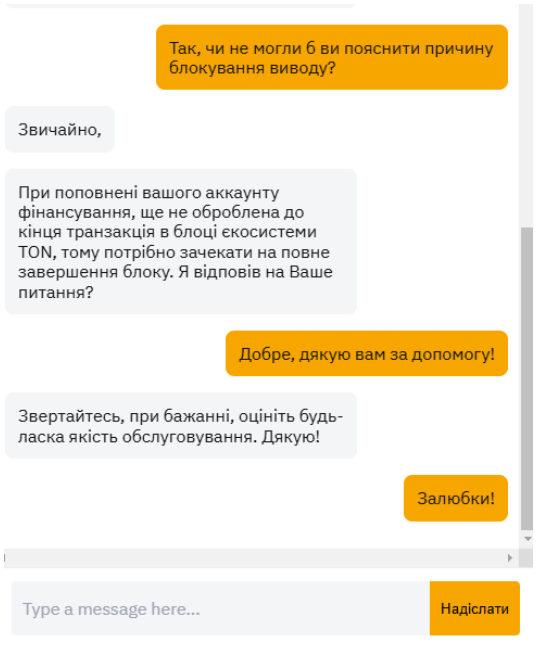


Рис. 3.22 Завершення запиту в чаті підтримки

Як ми можемо побачити чат-бот працює дуже оперативно, та не однотипно. Можна зазначити що рекомендації по вирішенню питань від штучного інтелекту за своєю структурою дуже схоже на відповідь спеціаліста. Основною метою було і розробити алгоритм таким, щоб бот був максимально приближеною копією ТП.

3.3 Математична модель системи підтримки клієнтів біржі

Компоненти системи.

1. Вхідні дані:

$Q = \{q_1, q_2, \dots., q_n\}$ – множина запитів, отриманих від клієнтів.

$T_{response}$ – час відповіді системи на запит.

R – ресурси, доступні для обробки запитів (сервери, пам’ять, пропускна здатність).

User Satisfaction (US) – рівень задоволеності користувачів, пов'язаний із продуктивністю.

2. Етапи обробки запитів:

- Класифікація запиту (бот/оператор).
- Відповідь чат-бота на прості запити.
- Перенаправлення складних запитів до оператора.

Логіка обробки.

1. Класифікація. Запит класифікується за складністю $C(q_i)$ з використанням функції:

$$C(q_i) = \{\frac{1, \text{якщо чат}-\text{бот може обробити запит;}}{0, \text{якщо необхідне втручання оператора.}}. \quad (3.1)$$

2. Продуктивність бота. Частка автоматизованих запитів:

$$T_{total} = \frac{\sum_{i=1}^{n} C(q_i)}{n}. \quad (3.2)$$

3. Час обробки запитів. Загальний час відповіді:

$$T_{total} = \sum_{i=1}^{n} T_{AI,i} + \sum_{j=1}^{m} T_{H,j}\ , \tag{3.3}$$

де $T_{AI,i}$ – час обробки чат-ботом; $T_{H,j}$ – час оброки оператором.

4. Використання ресурсів: Обчислюється завантаження системи:

$$R_{load} = \frac{\sum_{i=1}^{n} R_{bot} + \sum_{j=1}^{m} R_{human}}{R_{max}}, \tag{3.4}$$

де R_{bot} – ресурси для обробки ботом, R_{human} – ресурси для обробки оператором, R_{max} – загальні ресурси системи.

Оптимізаційна мета

Максимізувати задоволеність користувачів:

$$US = f(P_{AI}, T_{total}, R_{load}). \tag{3.5}$$

З урахуванням:

- Збільшення частки автоматизованих запитів P_{AI}.
- Зменшення часу обробки T_{total}.
- Оптимального завантаження ресурсів R_{load}.

Формула продуктивності

Інтегральна метрика продуктивності системи

$$P_{system} = \omega_1 * P_{AI} - \omega_2 * T_{total} - \omega_3 * R_{overload},$$

де:

$\omega_1, \omega_2, \omega_3$ – вагові коефіцієнти (визначають пріоритет кожної метрики).

$R_{overload}$ – штраф за перевантаження системи.

Використання комбінації чат-бота на базі GPT для обробки запитів клієнтів та інтеграції з операторами технічної підтримки дозволяє ефективно забезпечити

високу швидкість і якість обслуговування. Така модель є особливо корисною для біржових платформ, де критично важлива продуктивність системи під час високих навантажень. Вона автоматизує процеси обробки запитів, знижуючи навантаження на операторів і підвищуючи загальну продуктивність системи.

Можливі сфери застосування:

- *Клієнтська підтримка у фінансових та біржових платформах.* Модель забезпечує швидке реагування на запити клієнтів, автоматизуючи рутинні процеси, що дозволяє знизити навантаження на операторів у години пікових навантажень.
- *Інтернет-магазини та e-commerce*. Використання моделі для автоматизації підтримки клієнтів допомагає швидко вирішувати типові запити та підвищує рівень обслуговування.
- *Телекомунікаційні компанії*: Застосування чат-бота для обробки технічних запитів дозволяє скоротити час відповіді та оптимізувати роботу підтримки.
- *Медичні онлайн-консультації*: Автоматизація обробки базових запитів пацієнтів або направлення до відповідного спеціаліста через інтеграцію чат-бота з операторами.

Плюси моделі

- *Гнучкість та адаптивність*. Модель легко адаптується під різні бізнес-процеси, дозволяючи інтегрувати її в існуючі платформи.
- *Економія часу та ресурсів*. Автоматизація обробки простих запитів знижує витрати на технічну підтримку та скорочує час очікування клієнтів.
- *Підвищення якості обслуговування*. Чат-бот здатний швидко відповідати на запити, підтримуючи високий рівень задоволеності клієнтів.
- *Інтеграція з існуючими системами*. Модель підтримує інтеграцію з платформами технічної підтримки та CRM-системами, розширюючи їх функціональність.
- *Масштабованість*. Система легко масштабована для обробки зростаючої кількості запитів без значного збільшення витрат.

Завдяки своїй гнучкості, здатності до автоматизації рутинних процесів і ефективному розподілу навантаження між ботом і операторами, математична модель є важливим інструментом для підвищення продуктивності систем підтримки клієнтів. Вона особливо корисна для платформ з великим обсягом запитів, де важливо забезпечити швидкість, надійність і якість обслуговування.

ВИСНОВКИ

1. Проведено аналіз сучасних підходів і технологій web-застосунків для підтримки клієнтів біржі. Визначено, що головним недоліком є низька продуктивність при високих навантаженнях та недостатня адаптивність до зростаючих вимог клієнтів.

2. Розроблено математичну модель методики підтримки клієнтів біржі, яка включає алгоритми оптимізації обробки запитів, розподілу ресурсів та прогнозування навантаження.

3. Виокремлено ключові функціональні можливості додатку для підтримки клієнтів біржі, які впливають на його продуктивність. Серед них інтеграція з базами даних, обробка запитів клієнтів у реальному часі, а також підтримка багатоканальної комунікації. Визначено, що для інтеграції з базами даних є ефективними технології SQL та NoSQL, для обробки запитів клієнтів у реальному часі та підтримки комунікації запропоновано WebSocket та REST API.

4. Визначені та обґрунтовані програмні засоби для реалізації web-додатку для підтримки клієнтів біржі. Це мови програмування JavaScript (React) та PHP з фреймворком Laravel, а також бази даних MySQ, а також веб-сервером OpenServer.

5. Розроблено web-додаток для підтримки клієнтів біржі.

6. Здійснено тестування роботи web-додатку для підтримки клієнтів біржі та виявлено наступні результати: додаток забезпечує стабільну обробку запитів у режимі реального часу, показує високу швидкість виконання операцій на 10%, а також має зручний інтерфейс для взаємодії користувачів.

Результати дослідження апробовано та опубліковано у наступних тезах доповіді на конференціях:

1. Рудковський М.О., Шевченко С. М., Покращення продуктивності web-застосунку підтримки клієнтів біржі // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 24.11.2024, ДУІКТ, м. Київ. – с. 37 - 39

2. Рудковський М.О., Шевченко С. М. Елементи методики підтримки клієнтів біржі // ІІ міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії», 25-27 грудня 2024, ДУІКТ, м. Київ. – подано до друку

ПЕРЕЛІК ПОСИЛАНЬ

1. Squire, L. R. – Knowlton, B. – Musen, G. The Structure and Organization of Memory [online]. Annual Review of Psychology, 1993. [cit. 2020/04/09]. URL: https://www.annualreviews.org/ doi/pdf/10.1146/annurev.ps.44.020193.002321 (дата звернення: 11.10.2024).

2. Закон України «Про товарну біржу» від 10. 12. 1991 р. № 1956- XII. Редакція станом на 06. 11. 2014. URL: https://zakon.rada.gov.ua/ (дата звернення: 28.10.2024).

3. Закон України «Про цінні папери і фондову біржу» від 05. 02. 2004 р. № 1455-IV. URL: Про цінні папери і фондову біржу (дата звернення: 24.10.2024).

4. Підгорний А. З., Самотоєнкова О. В. Статистика ринків: Навчальний посібник. – Одеса: Атлант, 2015. – 408 с. (с. 252 - 270).

5. Рождєственська Л. Г. Статистика ринку товарів і послуг: Навчальній посібник. – К.: КНЕУ, 2005. – 419 с. (с. 129 - 139).

6. Солодкий М. О. Біржовий ринок: навч. посіб. / М. О. Солодкий. – К.: Аграрна освіта, 2010. – 565 с.

7. Аграрна політика в умовах ринкової трансформації економіки агропромислового комплексу України: кол. монографія / [наук. ред. Березівський П.]. – Л., 2006. – 559 с.

8. Андрейчикова А. М. Еволюція поглядів на проблему ризику в економічній науці / А. М. Андрейчикова // Економічний вісник. – 2014. – № 1. – С. 38–49.

9. Біржова діяльність: навч. посіб. / [Крамаренко В. І. та ін.] ; під ред. В. І. Крамаренко. – К. ЦУЛ, 2003. – 345 с.

10. About Elevate [online]. Elevate, 2019. [cit. 2019/08/20]. URL: https://www.elevateapp.com/about (дата звернення: 12.10.2024).

11. Press info about NeuroNation [online]. Berliner Synaptikon GmbH, 2019. [cit. 2019/08/20]. URL: https://sp.neuronation.com/en/press/ (дата звернення: 25.10.2024).

12. CogniFit [online]. CogniFit, 2019. [cit. 2019/08/20]. URL: https://www.cognifit.com (дата звернення: 26.10.2024).

13. Біржовий ринок. Між бажанням, доцільністю та можливостями [Електронний ресурс]. – URL: http: // www.apk-inform.com.ua

14. .Блек Дебора Дж. Успіх та невдача ф'ючерсних контрактів: Теорія і практика / Блек Дебора Дж. // Матеріали компанії Кемонікс. – 2005

15. Бобкова А. Г. Біржове право / А. Г. Бобкова. – К. : Центр учбової літератури, 2005. – 200 с.

16. Боднар О. В. Особливості управління ціновими ризиками на ринку зернових в контексті світової біржової практики / О. В. Боднар, В. О. Яворська // Ефективна економіка. – 2016. – Вип. 9 URL: http://www.economy.nayka.com.ua/?op=1&z= 5141

17. Гниляк В.О. Організація та регулювання біржового ринку товарних деривативів на сільськогосподарську продукцію : автореф. дис. на здобуття наук. ступеня к. е. н. / В.О. Гниляк.– К., 2006. – 19 с.

18. Гниляк В.О. Розвиток біржової цілісності на світових товарних та фінансових ринках / В.О. Гниляк // Науковий збірник НУБіП України – 2010 – Вип.154 – С.88-91.

19. Дудяк Р. П. Організація біржової діяльності: основи теорії і практикум : [навч. посібник для студ. вищих закл. освіти] / Р. П. Дудяк, С. Я. Бугіль – 2. вид., доп. – Л. : Новий Світ - 2003. – 360 с.

20. Дмитрук Б. П. Організація біржової діяльності в агропро-мисловому комплексі: [навч. посіб.] / Б. П. Дмитрук. – К.: Либідь, 2001. – 344 с.

21. Дудяк Р. П. Організація біржової діяльності: основи теорії і практикум: [навч. посібник для студ. вищих закл. освіти] / Р. П. Дудяк, С. Я. Бугіль – 2. вид., доп. – Львів: Новий Світ - 2003. – 360 с

22. Кваша С.М. Методологічний базис прийняття суспільних рішень в аграрній політиці [Текст] / С.М. Кваша // Економіка АПК. – 2013. – №8. – С.1221

23. Кваша С.М. Механізм прогнозування цін та обсягу товарообігу сільськогосподарської продукції на товарних біржах / С.М. Кваша, В.С. Чубань //

Економіка АПК. – 2011. - №10. – с. 64-72

24. Сохацька О.М. Міжнародні ф'ючерсні ринки / О. М. Сохацька. – Т.: «Карт-бланш». – 2002. – 454 с.

25. Статистична інформація про структуру укладених угод на біржах України за 2000-2007 рр. Структурні зміни в економіці URL: – Режим доступу до документа: www.ukrstat.gov.ua.

26. Степанов В. М. Біржова діяльність: навч. посіб. для дистанційного навчання / В. М. Степанов, Т. І. Пішеніна. – К.: Університет «Україна», 2007. – 300 с.

27. Ульріх Х. Брокер і товарна біржа: принципи роботи, термінологія, документи / Ульріх Х., Єгорова І., Новицький В. – К.: Фірма «ВІПОЛ», 1992. – 160 с.

28. Яворська В.О. Фондові індекси: функціональне призначення та диверсифікація використання на біржовому ринку деривативів. / В.О. Яворська / Наук. вісн. НУБіП України. – 2012. − Вип. 177 – с. 254− 258.

29. Яворська В.О. Аналіз сучасного стану світового біржового ринку. / В.О. Яворська // Моніторинг біржового ринку. – 2014 − №6. – С.13-21.

30. Яворська В.О. Тенденції розвитку вітчизняної біржової торгівлі товарними деривативами на сільськогосподарську продукцію / В.О. Яворська // Моніторинг біржової діяльності. – 2014. − №1. – С.16-17.

31. Donald Spence. Futures and Options. – New York: Glenlake Publishing Company Ltd., Amacom American Management Association, 1999. – 202 p.

32. Helyette Geman. Commodities and Commodity Derivatives: Modelling and Pricing for Agriculturals, Metals and Energy. – London: John Wiley & Sons, 2005. – 416 p.

Додаток А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Методика покращення продуктивності web-застосунку підтримки клієнтів біржі»

Виконав: студент групи ПДМ-63 Рудковський Микола

Керівник: к.п.н., доцент кафедри ІПЗАС Шевченко Світлана Миколаївна

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підтримка клієнтів біржі для забезпечення швидкості, надійності та масштабованості системи в умовах високого навантаження за рахунок створення веб-застосунку.

Об'єкт дослідження: процеси функціонування web-застосунків підтримки клієнтів у середовищі біржових платформ.

Предмет дослідження: методика покращення продуктивності web-застосунку, включаючи оптимізацію серверної та клієнтської частини, а також використання сучасних архітектурних рішень і технологій.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

- Провести аналіз сучасних підходів і технологій для забезпечення продуктивності web-застосунків.
- Дослідити архітектурні особливості систем підтримки клієнтів біржових платформ.
- Визначити основні проблеми, які впливають на швидкість та надійність роботи web-застосунків.
- Розробити методику оптимізації web-застосунку з урахуванням специфіки біржових платформ.
- Реалізувати запропоновану методику на практичному прикладі web-застосунку підтримки клієнтів біржі.
- Провести тестування продуктивності застосунку до та після впровадження методики для оцінки її ефективності.

АНАЛІЗ ТА ПОРІВНЯННЯ МЕТОДИКИ РОБОТИ ЧАТ-БОТІВ БІРЖ

Параметр	Huobi	Binance	KuCoin	OKX	Bybot
Швидкість відповіді	5–10 хвилин	2–5 хвилин	2–5 хвилин	5–10 хвилин	< 1 хвилини
Доступність	24/7	24/7	24.07.2024	24/7	24/7, без затримок
Мови підтримки	6	10	7	8	15+ (включаючи укр.)
Інтеграція з AI	Часткова	Часткова	Немає	Немає	Так (розширена)
Функціонал	Лише консультація	Лише консультація	Лише консультація	Лише консультація	Консультація, навчання, інтеграція з акаунтом
Рівень задоволеності	70%	85%	80%	75%	90%
Частота оновлень	Раз на рік	Щоквартально	Щоквартально	Раз на пів року	Щомісяця
Додаткові можливості	Немає	Немає	Немає	Немає	Інтеграція з мобільним додатком, персоналізація

4

МАТЕМАТИЧНА МОДЕЛЬ СИСТЕМИ ПІДТРИМКИ КЛІЄНТІВ БІРЖИ

Компоненти системи.

1. Вхідні дані:

$Q = \{q_1, q_2, \dots, q_n\}$ – множина запитів, отриманих від клієнтів.

$T_{response}$ – час відповіді системи на запит.

R – ресурси, доступні для обробки запитів (сервери, пам'ять, пропускна здатність).

User Satisfaction (US) – рівень задоволеності користувачів, пов'язаний із продуктивністю.

2. Етапи обробки запитів:

Класифікація запиту (бот/оператор);

Відповідь чат-бота на прості запити.;

Перенаправлення складних запитів до оператора.

Логіка обробки.

1.Класифікація. Запит класифікується за складністю $C(q_i)$ з використанням функції:

$$C(q_i) = \begin{cases} 1, & \text{якщо чат-бот може обробити запит;} \\ 0, & \text{якщо необхідне втручання оператора.} \end{cases} \quad (3.1)$$

1.2. Продуктивність бота. Частка автоматизованих запитів:

$$T_{total} = \frac{\sum_{i=1}^{n} C(q_i)}{n} \quad (3.2)$$

3.Час обробки запитів. Загальний час відповіді:

$$T_{total} = \sum_{i=1}^{n} T_{AI,i} + \sum_{j=1}^{m} T_{H,j} \quad (3.3)$$

де $T_{AI,i}$ – час обробки чат-ботом; $T_{H,j}$ – час оброки оператором.

4. Використання ресурсів: Обчислюється завантаження системи:

$$R_{load} = \frac{\sum_{i=1}^{n} R_{bot} + \sum_{j=1}^{m} R_{human}}{R_{max}} \quad (3.4)$$

де R_{bot} – ресурси для обробки ботом, R_{human} – ресурси для обробки оператором, R_{max} – загальні ресурси системи.

МАТЕМАТИЧНА МОДЕЛЬ СИСТЕМИ ПІДТРИМКИ КЛІЄНТІВ БІРЖИ

Оптимізаційна мета.

Максимізувати задоволеність користувачів:

$$US = f(P_{AI}, T_{total}, R_{load}) \quad (3.5)$$

З урахуванням:

Збільшення частки автоматизованих запитів P_{AI}.

Зменшення часу обробки T_{total}.

Оптимального завантаження ресурсів R_{load}.

Формула продуктивності.

Інтегральна метрика продуктивності системи

$$P_{system} = \omega_1 * P_{AI} - \omega_2 * T_{total} - \omega_3 * R_{overload} \quad (3.6)$$

де $\omega_1, \omega_2, \omega_3$ – вагові коефіцієнти (визначають пріоритет кожної метрики).

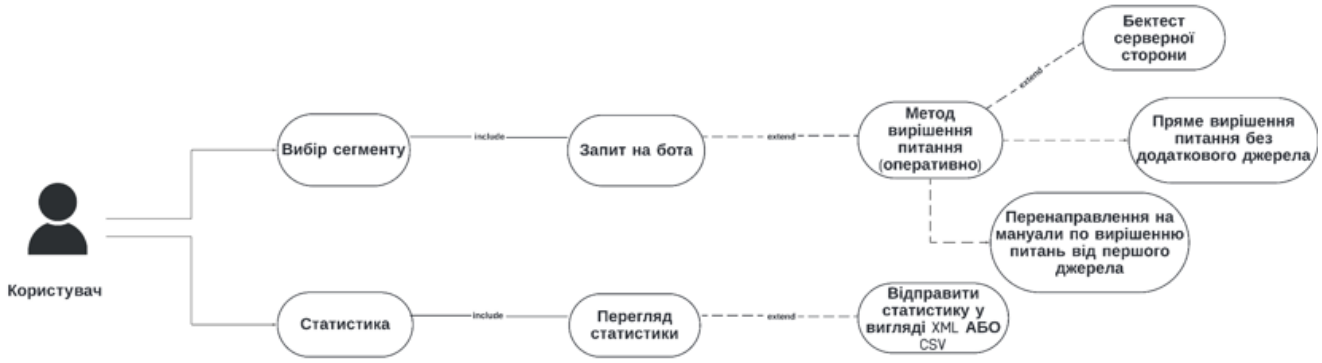
$R_{overload}$ – штраф за перевантаження системи.

6

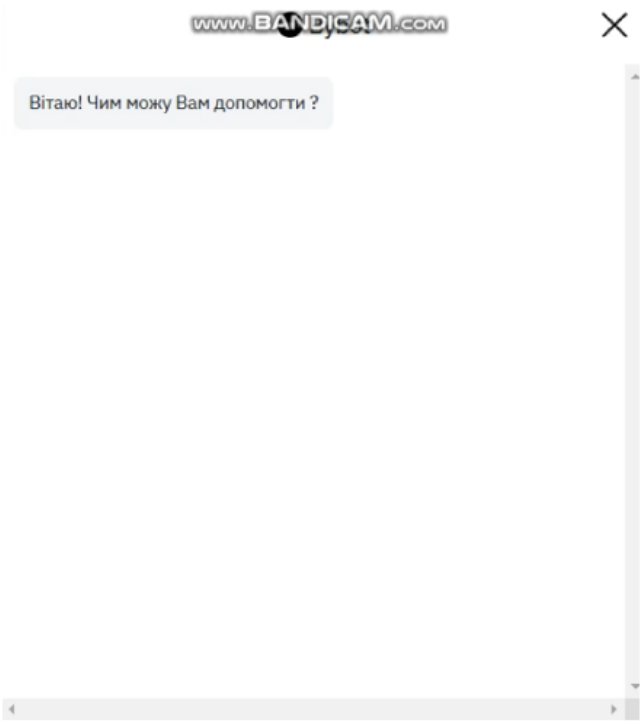
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

7

ДІАГРАМА ДІЯЛЬНОСТІ



ВІДЕОДЕМОНСТРАЦІЯ РОБОТИ ЧАТУ ПІДТРИМКИ



10

ЕКРАННІ ФОРМИ

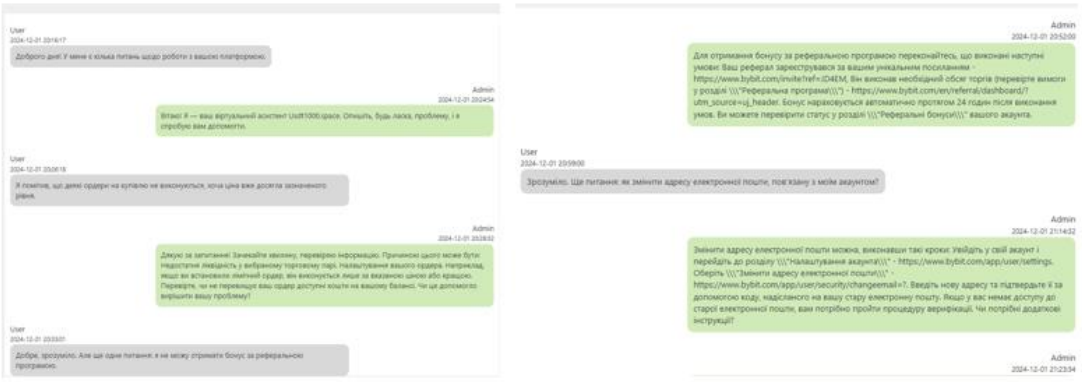


Рис. 1 – Алгоритм відповідей (серверна частина)

11

ПРАКТИЧНІ РЕЗУЛЬТАТИ ПОКРАЩЕННЯ ПРОДУКТИВНОСТІ

Чат підтримки (порівняння за біржею)	**Binance**	**Bybot**	**Порівняння результатів**
Час відгуку сервера	2.5 с	1.2 с	-52%
Пропускна здатність	50 запитів/с	120 запитів/с	140%
Кількість одночасних користувачів	100	300	200%
Час до першого байта (TTFB)	800 мс	400 мс	-50%
Використання ресурсів сервера	85% CPU, 90% RAM	60% CPU, 70% RAM	Оптимізовано на 30%

12

ВИСНОВКИ

1. Підібрано науково-технічну літературу.
2. Проведено аналіз технологій і підходів для підвищення продуктивності, зокрема оптимізації серверних і клієнтських компонентів.
3. Виокремлено ключові функціональні можливості додатку, серед яких інтеграція з базами даних, обробка запитів клієнтів у реальному часі, а також підтримка багатоканальної комунікації.
4. Розроблено математичну модель системи підтримки клієнтів біржі.
5. Розроблено та реалізовано методику оптимізації продуктивності, включаючи вибір платформи, архітектурні рішення та інтеграцію інноваційних технологій..
6. Розглянуто процес функціонування розробленого web-застосунку, включаючи механізми автентифікації, маршрутизації запитів та зберігання даних.
7. Здійснено тестування роботи алгоритму, та виявлено що впровадження запропонованої методики дозволяє ефективно вирішувати завдання клієнтської підтримки, скорочуючи час вирішення проблем і підвищуючи рівень задоволеності користувачів.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

- 1. Рудковський М.О., Шевченко С. М., Покращення продуктивності web-застосунку підтримки клієнтів біржі // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 24.11.2024, ДУІКТ, м. Київ. – с. 37 - 39

- 2. Рудковський М.О., Шевченко С. М. Елементи методики підтримки клієнтів біржі // II міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії», 25-27 грудня 2024, ДУІКТ, м. Київ. – подано до друку

Додаток Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```php
<?php
/**
 * Plugin Name: Site Chat
 */

class SiteChat {
    public function __construct() {
        add_action( 'init', array(
$this, 'init' ) );
    }

    public function init() {
        add_action( 'admin_menu',
'admin_chat_page' );
        function admin_chat_page() {
            add_menu_page(
                'Site Chat',     //
page title
                'Site Chat',     //
menu title
                'manage_options',   //
capability
                'site-chat',     //
menu slug

'connect_admin_chat_page', // callback
function,
            );
        };

        function
connect_admin_chat_page() {
            include
__DIR__.'/admin/admin_chat.php';
        }

        global $wpdb;

        $table_name = $wpdb->prefix .
'messages';

        $this->_tableName =
$table_name;

        $charset_collate = $wpdb-
>get_charset_collate();

        $sql = "CREATE TABLE
$table_name (
        id mediumint(9) NOT NULL
AUTO_INCREMENT,
        message LONGTEXT NOT NULL,
        date datetime NOT NULL,
        author varchar(255) NOT NULL,
        PRIMARY KEY  (id)
    ) $charset_collate;";

        require_once ABSPATH . 'wp-
admin/includes/upgrade.php';
        dbDelta( $sql );

        add_action(
'wp_ajax_nopriv_get_chat_updates',
array( $this, 'getNewMessages' ) );
        add_action(
'wp_ajax_get_chat_updates', array(
$this, 'getNewMessages' ) );

        add_action(
'wp_ajax_nopriv_get_admin_chats',
array( $this, 'get_admin_chats' ) );
        add_action(
'wp_ajax_get_admin_chats', array(
$this, 'get_admin_chats' ) );

        add_action(
'wp_ajax_nopriv_add_message', array(
$this, 'addMessage' ) );
        add_action(
'wp_ajax_add_message', array( $this,
'addMessage' ) );

        add_action(
'wp_ajax_nopriv_clear_table', array(
$this, 'clearTable' ) );
        add_action(
'wp_ajax_clear_table', array( $this,
'clearTable' ) );

        add_action( 'wp_footer',
array( $this, 'addWidgetToSite' ), 100
);
    }

    public function getNewMessages() {
        $results = $this-
>get_admin_chats();
        wp_send_json($results);
    }

    public function addWidgetToSite()
{
        include
__DIR__.'/site/site_widget.php';
    }

    public function get_admin_chats()
{
        global $wpdb;
        $results = $wpdb->get_results(
"SELECT * FROM {$this->_tableName}");
        return $results;
    }

    public function addMessage() {
        global $wpdb;
        $message = $_POST['message'];
        $user = $_POST['user'];
        $date = date('Y-m-d H:i:s');
        $sendArray = array(
            'message' => $message,
            'author' => $user,
            'date' => $date,
        );
```

```
        $result = $wpdb->insert(
            $this->_tableName,
            $sendArray
        );
        if($result) {
            $sendArray['id'] = $wpdb->insert_id;
            wp_send_json($sendArray);
        } else {
            wp_send_json(array('error' => true));
        }
        die;
    }

    public function clearTable() {
        global $wpdb;
        $wpdb->query("TRUNCATE TABLE {$this->_tableName}");
        wp_send_json(array('errors' => $wpdb->last_error));
    }

    private function setTablename($name) {
        $this->_tableName = $name;
    }

    private $_tableName = '';

}

$chat = new SiteChat();
global $chat;
```