Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня
Бакалавр

(назва освітнього ступеня)

на тему: Розробка системи оплат для інтернет-магазину дропшипінгу з використанням ключів ідемпотентності

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Поліщук Р.В. (прізвище та ініціали)

Керівник (підпис) Заярний М.А. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)
Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Поліщуку Руслану Володимировичу
(прізвище, ім'я, по батькові)

н Розробка системи оплат для інтернет-магазину дропшипінгу з використанням ключів ідемпотентності

Керівник роботи Заярний М.А.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи _____
3. Вихідні дані до роботи Реалізована система оплат для інтернет магазину дропшипінгу з використанням ключів ідемпотентності

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз предметної області та теоретичних основ сучасних систем оплат
Визначення принципів роботи системи оплат
Реалізація системи оплат
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,			
основи охорони праці	Лазарюк В.В. доц. каф. МТ		
Нормоконтроль	Стоянов Ю.М. к.т.н., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз завдання		
3	Виконання розділу 1		
4	Виконання розділу 2		
5	Виконання розділу 3		
6	Оформлення пояснювальної записки		
7	Оформлення графічного та презентаційного матеріалу		
8	Попередній захист		
9	Захист		

Студент _____ Поліщук Р.В
(підпис) (прізвище та ініціали)

Керівник роботи _____ Заярний М.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра, виконана Поліщук Руслан Володимирович, студентом групи СП-42 Тернопільського національно технічного університету імені І. Пулюя, присвячена впровадженню ідемпотентних ключів для забезпечення цілісності транзакцій у системах дропшипінгу. Робота має обсяг 75 сторінок, включає 20 рисунків, 3 додатків та бібліографію з 32 джерел.

Метою роботи є розробка та впровадження надійної стратегії управління платіжними транзакціями в системах дропшипінгу за допомогою ідемпотентних ключів. Використання ідемпотентних ключів дозволяє забезпечити унікальність ідентифікації транзакцій, запобігти їх дублюванню та забезпечити стабільність роботи системи навіть у випадку технічних збоїв або повторних запитів. У роботі детально розглянуті теоретичні основи ідемпотентності, методи генерації та обробки ключів, а також практичні аспекти їх впровадження.

Розроблений прототип системи включає інтеграцію з Redis для ефективного зберігання ідемпотентних ключів та забезпечення швидкого доступу до них. Використання middleware у серверному оточенні дозволяє централізовано обробляти ідемпотентні ключі без необхідності дублювати код у кожному маршруті.

Результати тестування прототипу демонструють його високу ефективність та здатність забезпечити надійність і безпеку транзакцій у системах дропшипінгу. Робота підкреслює значення ідемпотентності у сучасних платіжних системах та її вплив на розвиток електронної комерції.

Ключові слова роботи: ідемпотентність, платіжні системи, дропшипінг, ідемпотентні ключі, Redis, middleware, безпека транзакцій.

ABSTRACT

Bachelor's qualification work, carried out by Ruslan Polishchuk, a student of group SP-42 at Ivan Puluj Ternopil National Technical University, focuses on the implementation of idempotency keys to ensure transaction integrity in dropshipping systems. The work consists of 75 pages, includes 20 figures, 3 appendices, and a bibliography with 32 sources.

The primary objective of this work is to develop and implement a reliable strategy for managing payment transactions in dropshipping systems using idempotency keys. The use of idempotency keys ensures the uniqueness of transaction identification, prevents duplication, and maintains system stability even in the event of technical failures or repeated requests. The thesis thoroughly examines the theoretical foundations of idempotency, methods for generating and processing keys, as well as practical aspects of their implementation.

The developed system prototype includes integration with Redis for efficient storage and quick access to idempotency keys. Utilizing middleware in the server environment allows centralized processing of idempotency keys without the need to duplicate code in each route.

The testing results of the prototype demonstrate its high efficiency and ability to ensure the reliability and security of transactions in dropshipping systems. This work highlights the importance of idempotency in modern payment systems and its impact on the development of e-commerce.

Key Words: idempotency, payment systems, dropshipping, idempotency keys, Redis, middleware, transaction security.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

RAG – Retrieval-Augmented Generation

SEO – Search Engine Optimization

NLP – Natural Language Processing

ML – Machine Learning

AI – Artificial Intelligence

UI – User Interface

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

URL – Uniform Resource Locator

DOM – Document Object Model

SPA – Single Page Application

MPA – Multi Page Application

PWA – Progressive Web Application

JSX – JavaScript XML

XML – Extensible Markup Language

E2E – End-to-End

TS – TypeScript

ROI – Return on Investment

UX – User Experience

GA – Google Analytics

HTML – HyperText Markup Language

ВСТУП

У сучасному світі електронної комерції та діджиталізації бізнес-процесів, надійність платіжних систем є критично важливою. Зокрема, у контексті дропшипінгу, де продавці не володіють товаром на власність і відправляють його напряму від постачальника до кінцевого споживача, забезпечення цілісності транзакцій набуває особливого значення. Одним із сучасних рішень цієї проблеми є застосування ідемпотентних ключів — унікальних ідентифікаторів, що гарантують, що кожна транзакція буде оброблена тільки один раз, незалежно від кількості спроб її виконання.

Метою даної роботи є аналіз та вдосконалення механізмів ідемпотентності в платіжних системах, особливо зосереджуючись на підходах, які використовують компанії PayPal і Stripe. Основна увага буде приділена розробці стратегій, які можуть забезпечити високу надійність обробки платежів, знижуючи ризик дублювання транзакцій, що є критично важливим для ефективності дропшипінгу.

Актуальність дослідження полягає у зростанні обсягів електронної комерції, збільшенні частоти транзакцій та вимог до їх безпеки та надійності. Невирішеність проблеми дублювання платежів може призвести до фінансових втрат для усіх сторін: споживачів, продавців та платіжних посередників.

У нашому дослідженні ми ставимо перед собою мету розробити і проаналізувати стратегії застосування ідемпотентних ключів для забезпечення цілісності транзакцій у платіжних системах, особливо зосереджуючись на платформах PayPal і Stripe. Це актуально, адже надійність платежів, мінімізація ризику дублікацій, і підвищення загальної безпеки є критично важливими у сфері електронної комерції та дропшипінгу.

Для досягнення поставленої мети ми виконуємо низку завдань: проводимо теоретичний аналіз ідемпотентності, систематизуємо існуючі підходи до використання ідемпотентних ключів у системах PayPal та Stripe, проводимо експериментальне дослідження, аналізуємо вплив ідемпотентності на

користувацький досвід та розробляємо рекомендації для підвищення ефективності та безпеки платіжних систем у контексті дропшипінгу.

Об'єктом нашого дослідження є ідемпотентність як механізм, що забезпечує цілісність транзакцій в цифрових платіжних системах. Зокрема, ми зосереджуємось на технологіях і методах, застосованих компаніями PayPal і Stripe. Детальне вивчення цих методів допомагає оцінити широкий спектр застосувань ідемпотентних ключів та їхній вплив на різні бізнес-моделі, особливо у сфері дропшипінгу.

Наше дослідження фокусується на вирішенні трьох ключових питань, що стосуються застосування ідемпотентності у платіжних системах. Перше питання стосується визначення основних методів та технологій, які використовуються для забезпечення ідемпотентності в таких системах. Друге питання аналізує інтеграцію ідемпотентних ключів у платіжні архітектури провідних компаній, таких як PayPal та Stripe. Третє питання зосереджено на вивченні переваг і потенційних проблем, що можуть виникнути при використанні ідемпотентних ключів у контексті дропшипінгу. Особлива увага приділяється аналізу, як впровадження цих ключів може вплинути на операції дропшипінгу, зокрема на зниження ризиків пов'язаних з повторними транзакціями та покращення надійності та безпеки платежів.

У заключенні, впровадження ідемпотентних ключів у платіжні системи, особливо в таких компаніях як PayPal та Stripe, є важливим кроком до забезпечення надійності та безпеки транзакцій в умовах швидко зростаючого ринку електронної комерції. Наше дослідження підкреслює значення цих механізмів у запобіганні дублюванню платежів та підвищенні загальної ефективності платіжних процесів. Особливо це актуально для дропшипінгу, де ефективність і точність транзакцій безпосередньо впливають на успіх бізнесу. Використання ідемпотентних ключів не лише підвищує довіру споживачів, але й сприяє стабілізації та оптимізації бізнес-моделей, роблячи їх більш стійкими до помилок у транзакціях. Наше дослідження відкриває нові перспективи для подальшого розвитку та вдосконалення платіжних систем, що в свою чергу може привести до значних поліпшень у сфері цифрових платежів.

1. ОГЛЯД ТЕОРЕТИЧНИХ ОСНОВ ІДЕМПОТЕНТНОСТІ В СИСТЕМАХ ОБРОБКИ ПЛАТЕЖІВ

Огляд теоретичних основ ідемпотентності в системах обробки платежів забезпечує глибоке розуміння ключових концепцій і механізмів, які є важливими для використання ідемпотентних ключів. Вивчення теоретичних праць, стандартів і найкращих практик показує, як ідемпотентність інтегрована в такі великі платіжні системи, як PayPal та Stripe [1].

Ідемпотентність має ключове значення в сучасних платіжних технологіях. Історія розвитку ідемпотентних операцій підкреслює, як ці механізми стали необхідними для забезпечення надійності і безпеки в автоматизованих платіжних системах.

Аналіз різних підходів до використання ідемпотентних ключів виявляє, що платіжні сервіси, такі як PayPal і Stripe, інтегрують ці ключі у свої API, щоб запобігти дублюванню транзакцій. Ідемпотентні ключі використовуються для відстеження та управління станом транзакцій, забезпечуючи, що кожна операція обробляється лише один раз.

Належне використання ідемпотентних ключів може суттєво підвищити ефективність обробки платежів і зменшити ризик фінансових втрат через помилки в транзакціях. Технічні аспекти та бізнес-вимоги, що ставляться до платіжних систем у сфері електронної комерції та дропшипінгу, відіграють ключову роль у забезпеченні стійкості і надійності цих систем [2].

1.1 Основи ідемпотентності та їх значення

Ідемпотентність — це фундаментальний принцип в області інформаційних технологій і програмування, особливо у контексті платіжних систем, де він відіграє

вирішальну роль у забезпеченні цілісності та надійності фінансових транзакцій. Цей принцип забезпечує, що незалежно від кількості разів виконання певної операції, результат залишається незмінним після першого успішного виконання. В контексті платіжних систем, ідемпотентність запобігає можливим фінансовим втратам чи помилкам, що можуть виникнути через повторні транзакції.

Основна ідея ідемпотентності полягає в тому, що певна операція, коли вона виконується багато разів з тим самим набором вхідних даних, виробляє один і той же результат. Наприклад, якщо користувач намагається провести платіж, але з-за технічної помилки або затримки не бачить підтвердження та спробує здійснити платіж повторно, система, що використовує ідемпотентні методи, гарантуватиме, що кошти будуть списані лише один раз. Використання ідемпотентних ключів є одним із способів імплементації цього принципу у платіжних системах. Ключі дозволяють системі розпізнати повторні запити та управляти їх обробкою належним чином [3,4].

Значення ідемпотентності у платіжних системах є величезним, оскільки це допомагає забезпечити надійність і безпеку фінансових транзакцій. Воно знижує ризик фінансових помилок, підвищує довіру користувачів до системи та покращує загальний користувацький досвід, забезпечуючи, що платежі обробляються швидко та без помилок навіть у випадках технічних збоїв або затримок у мережі. Така надійність є особливо важливою в умовах, де висока частота транзакцій і значні обсяги оброблюваних даних ставлять підвищені вимоги до стабільності та відповідності системи.

1.1.1 Визначення ідемпотентності

Ідемпотентність — це властивість деяких операцій у математиці та інформатиці, за якою операції можуть бути виконані кілька разів без зміни

результату після першого виконання. У контексті платіжних систем і більш широко в програмуванні, це визначення застосовується для забезпечення, що повторні запити на транзакцію не спричинять додаткових змін чи втрат, якщо транзакція вже була успішно здійснена [5].

Використання ідемпотентності важливе у веб-сервісах і API, де мережеві запити можуть бути відправлені кілька разів через помилки з’єднання, користувацькі помилки або інші проблеми, які можуть спричинити повторне надсилання одного і того ж запиту. Застосування ідемпотентних операцій у таких ситуаціях гарантує, що не залежно від кількості спроб, результати операцій будуть однакові, що критично для забезпечення консистентності та надійності даних.

Ідемпотентність досягається шляхом реалізації різних механізмів, наприклад, за допомогою ідемпотентних ключів. При виконанні операцій, таких як оплата чи зміна стану, система приймає унікальний ідентифікатор запиту (ідемпотентний ключ). Якщо система отримує інший запит із тим же ключем, вона не виконує нову операцію, а лише повертає результат оригінального запиту.

Таким чином, ідемпотентність допомагає уникнути багатьох типових проблем в інтернет-транзакціях, включаючи непотрібне дублювання платежів або виконання інших операцій, що можуть негативно вплинути на користувачів та системи. Це особливо важливо в середовищах, де високі стандарти надійності та точності є критичними.

1.1.2 Історичний контекст та розвиток концепції

Концепція ідемпотентності в інформаційних технологіях, хоч і має свої корені в математиці, знайшла своє значуще застосування в обробці транзакцій і веб-розробці протягом останніх декількох десятиліть. Ця ідея стала особливо актуальною з розвитком інтернету та електронної комерції, де потреба в надійному

управлінні транзакціями стала очевидною через збільшення обсягів та швидкості обміну даними.

У ранніх днях розробки веб-додатків, коли інтернет ще не мав настільки розширену інфраструктуру, повторні запити та відмови серверів часто призводили до помилкових транзакцій або подвійних платежів, спричиняючи збитки для користувачів та сервісів. Це спонукало розробників шукати рішення, яке б забезпечило обробку кожної унікальної операції один раз, навіть якщо запит було відправлено кілька разів [6].

З того часу ідемпотентні механізми були інтегровані у багато аспектів комп'ютерних наук, особливо в бази даних, дистрибуовані системи і розробку API. Великі технологічні компанії, такі як Google, Amazon і Facebook, розробили свої системи з урахуванням ідемпотентності для забезпечення надійності і скальованості своїх додатків. Платіжні системи, зокрема PayPal і Stripe, використовують ідемпотентність як критичний елемент у своїх API, забезпечуючи, що транзакції можуть бути безпечно повторювані без ризику подвійного списання коштів.

Розвиток концепції ідемпотентності також стимулював законодавчі та стандартизаційні ініціативи, що забезпечують уніфіковані підходи до транзакційних систем в різних юрисдикціях. Наприклад, стандарти PCI DSS (Payment Card Industry Data Security Standard) включають рекомендації щодо використання ідемпотентності для забезпечення безпеки платіжних даних.

Таким чином, історія і розвиток ідемпотентності в IT відіграли ключову роль у формуванні надійних та безпечних платіжних систем, які є вітальними в сучасному світі цифрової економіки. Цей принцип продовжує залишатися важливим у контексті постійного розвитку технологій і вимог до безпеки та стійкості електронної комерції.

1.1.3 Значення ідемпотентності у сучасних платіжних системах

Значення ідемпотентності у сучасних платіжних системах важко переоцінити, особливо в контексті глобалізації та зростання цифрових транзакцій. Цей принцип забезпечує стабільність і надійність обробки платежів, що є критично важливим у сфері електронної комерції, де транзакції мають високі вимоги до швидкості, точності та безпеки, розглянемо детальніше:

а) Запобігання подвійним списанням коштів. Однією з найважливіших переваг ідемпотентності є запобігання подвійному списанню коштів. У сценаріях, коли користувач намагається здійснити платіж і з-за технічної помилки чи перебоїв у з'єднанні запит може бути відправлений кілька разів, ідемпотентні системи забезпечують, що фінансова операція буде виконана тільки один раз, незалежно від кількості запитів. Запобігання подвійному списанню коштів є однією з найважливіших функцій ідемпотентності у платіжних системах. Використання ідемпотентних ключів дозволяє системам розпізнавати повторні запити на одну і ту ж транзакцію. Коли користувач намагається виконати платіж більше одного разу через помилки в інтерфейсі або мережеві збої, система використовує зазначений ключ для перевірки, чи була ця транзакція вже оброблена. Якщо так, система просто повертає результат оригінального платежу, запобігаючи подальшому списанню коштів. Це забезпечує точність фінансових операцій і запобігає ненавмисному збагаченню або збиткам як для клієнтів, так і для бізнесів.

б) Збільшення довіри та задоволення клієнтів. Клієнти очікують, що їхні транзакції будуть оброблені швидко і без помилок. Ідемпотентність у платіжних системах гарантує користувачам, що їхні платежі захищені від можливих технічних неполадок, тим самим підвищуючи довіру та загальне задоволення користувачів. Це, у свою чергу, сприяє підвищенню репутації компанії та лояльності клієнтів. Ідемпотентні платіжні системи забезпечують високий рівень довіри і задоволення клієнтів, оскільки вони мінімізують помилки під час транзакцій. Коли клієнти знають, що їхні платежі обробляються надійно і що система має механізми захисту

від помилок і дублікацій, вони більш схильні користуватися цим сервісом у майбутньому. Така безперебійність і безпомилковість веде до підвищення лояльності клієнтів та може спонукати їх рекомендувати сервіс іншим, що у свою чергу підвищує репутацію і ринковий успіх компанії.

в) Покращення бізнес-операцій. Впровадження ідемпотентності сприяє оптимізації бізнес-процесів, оскільки компанії можуть бути впевнені в точності та консистентності обробки транзакцій. Це знижує ризик фінансових помилок і потребу в затратному виправленні таких помилок, тим самим підвищуючи загальну ефективність операцій. Ідемпотентність також покращує загальну ефективність бізнес-операцій, оскільки зменшує потребу в ручній перевірці та виправленні помилок транзакцій. Бізнеси, які імплементують ідемпотентність, можуть заощадити значні ресурси, що раніше витрачались на обслуговування клієнтів, пов'язане з подвійними списаннями та іншими помилками. Це, у свою чергу, дозволяє перерозподілити ресурси на інші аспекти діяльності, такі як розвиток продукту, маркетинг та розширення.

г) Сумісність і масштабованість. У масштабованих системах, які обслуговують мільйони транзакцій, ідемпотентність дозволяє забезпечити сумісність і стабільність системи, навіть у складних, розподілених середовищах. Це критично важливо для глобальних платіжних платформ, які мають забезпечувати надійність та стабільність своїх послуг на різних ринках із різними вимогами до обробки даних. Ідемпотентність є критичною для сумісності і масштабованості сучасних платіжних систем, особливо у глобальному контексті. Вона дозволяє платіжним системам ефективно обробляти величезну кількість транзакцій з різних джерел без ризику порушення цілісності даних. Ідемпотентні механізми можуть бути легко інтегровані в різні архітектури і технологічні стеки, що робить їх ідеальними для використання у різних типах бізнесу від стартапів до великих корпорацій. Така масштабованість є необхідною для забезпечення сталого зростання та адаптації до швидкозмінних умов ринку.

Отже, ідемпотентність у сучасних платіжних системах відіграє ключову роль у забезпеченні надійності, безпеки, і ефективності фінансових транзакцій, що є

основою для стабільного розвитку електронної комерції та інших секторів, які залежать від цифрових платежів.

1.2 Методики використання ідемпотентних ключів в PayPal та Stripe

Методики використання ідемпотентних ключів у таких компаніях, як PayPal і Stripe, демонструють передові практики в застосуванні цього принципу для забезпечення надійності та безпеки платіжних процесів. Ці підходи мають забезпечити, що транзакції будуть оброблені лише один раз, навіть якщо відбувається кілька спроб здійснити один і той же платіж [7].

PayPal використовує ідемпотентні ключі у своїх API для забезпечення безпечних та надійних транзакцій. При здійсненні платежу через API, розробники можуть вказувати унікальний ідемпотентний ключ у запиті. Це дозволяє PayPal ідентифікувати повторні запити і запобігати обробці однакових транзакцій більше одного разу.

PayPal також має механізми, що автоматично генерують і зберігають ці ключі на час обробки запиту, щоб згодом використовувати їх для виявлення та блокування дублювання. Це особливо корисно в середовищах з високою частотою транзакцій, де ризик дублювання платежів є значним.

Stripe також активно використовує ідемпотентність у своїх платіжних операціях. Коли розробники використовують Stripe API для здійснення платежів, вони можуть передавати ідемпотентний ключ з кожним запитом. Stripe зберігає результати операцій, асоційовані з цими ключами, протягом 24 годин. Це означає, що якщо здійснити повторний запит із тим самим ключем протягом цього часу, система поверне результат оригінальної транзакції замість того, щоб знову обробляти платіж.

Stripe використовує цю техніку для забезпечення надійності своєї платіжної інфраструктури, зниження ризику помилкових платежів та підвищення загальної

задоволеності користувачів. Цей підхід також зменшує навантаження на обробку платежів, що є важливим для забезпечення швидкості та масштабованості послуг.

Обидві компанії, PayPal і Stripe, демонструють, як ідемпотентність може бути використана для забезпечення більш стабільної та безпечної обробки платежів, що є критично важливим в умовах сучасної глобальної електронної комерції [8, 9].

1.2.1 Процес генерації ідемпотентних ключів

Процес генерації ідемпотентних ключів є фундаментальним для правильного впровадження ідемпотентності у платіжні системи та інші типи веб-додатків. Ідемпотентний ключ — це унікальний ідентифікатор, що використовується для розпізнавання та управління повторними запитами на одну і ту ж транзакцію, забезпечуючи, що кожна операція виконується лише один раз. Ось основні аспекти, які стосуються генерації і використання ідемпотентних ключів [1, 2, 10]:

Генерація ключів

— Автоматична генерація: Багато систем автоматично генерують ідемпотентні ключі, коли користувач ініціює транзакцію. Ці ключі часто базуються на комбінації хеш-функцій, часових міток та інших унікальних даних, що гарантує їх унікальність.

— Користувацька генерація: У деяких випадках користувачі або розробники можуть самостійно генерувати ідемпотентні ключі, використовуючи алгоритми створення унікальних ідентифікаторів, такі як UUID (Universally Unique Identifier). UUID забезпечує велику ентропію і низьку ймовірність збігів.

Використання ключів

— 1. Відправка ключів з запитом: При здійсненні транзакції через API, ідемпотентний ключ включається у заголовок запиту. Це дозволяє платіжній системі визначити, чи була дана транзакція вже оброблена.

— Перевірка і збереження стану: Коли система отримує запит з ідемпотентним ключем, вона перевіряє, чи існує вже запис про транзакцію з цим ключем. Якщо такий запис існує, система повертає результат існуючої транзакції, не ініціюючи нову обробку. Якщо запису не існує, система обробляє запит як новий.

— Часові обмеження на ключі: Деякі системи, такі як Stripe, обмежують час зберігання інформації про транзакції, асоційованих з ідемпотентними ключами. Наприклад, Stripe зберігає інформацію про транзакції протягом 24 годин, що зменшує ризик випадкового повторного здійснення транзакцій, але забезпечує достатній час для обробки повних запитів.

Забезпечення унікальності

— Використання хеш-функцій: Щоб забезпечити унікальність ідемпотентних ключів, часто використовуються криптографічні хеш-функції. Наприклад, комбінація даних про користувача, часових міток та випадкових чисел може бути перетворена на унікальний хеш, який потім використовується як ідемпотентний ключ.

— Використання UUID: UUID (Universally Unique Identifier) є стандартним методом для генерації унікальних ключів. UUID складається з 128 бітів і має вкрай низьку ймовірність збігу, що робить його ідеальним для застосування в ідемпотентних операціях.

Практичні приклади

— Генерація у Stripe: У Stripe розробники можуть генерувати ідемпотентні ключі на стороні клієнта перед відправкою запиту. Ключ включається у заголовок "Idempotency-Key" і відправляється разом з запитом до сервера. Якщо запит повторно відправляється з тим самим ключем, Stripe повертає той самий результат, що і для першого запиту.

— Генерація у PayPal: У PayPal розробники також можуть включати ідемпотентний ключ у заголовок запиту. PayPal використовує цей ключ для перевірки, чи була транзакція вже оброблена. Якщо запит із тим самим ключем надходить знову, PayPal повертає результат першої обробленої транзакції.

Використання у бізнесі

— Запобігання помилок: Генерація ідемпотентних ключів і їх використання дозволяє уникнути помилкових транзакцій, знижуючи ризики фінансових втрат як для користувачів, так і для бізнесу. Це особливо важливо для великих платформ, де навіть невеликий відсоток помилок може призвести до значних збитків.

— Підвищення надійності: Використання ідемпотентних ключів підвищує надійність платіжних систем, оскільки забезпечує стабільність і передбачуваність результатів транзакцій. Це критично важливо для збереження довіри користувачів і підтримки високого рівня обслуговування [11].

Генерація ідемпотентних ключів є складовою частиною забезпечення надійності і безпеки сучасних платіжних систем. Правильне впровадження цього механізму дозволяє уникнути дублювання транзакцій, підвищити довіру клієнтів і забезпечити стабільну роботу платформи в умовах високих навантажень.

1.2.2 Інтеграція ключів у API платіжних систем

Інтеграція ідемпотентних ключів у API платіжних систем є важливим аспектом забезпечення надійності та безпеки транзакцій. Цей процес передбачає вбудовування механізмів, що дозволяють платіжним системам розпізнавати і обробляти повторні запити, запобігаючи дублюванню транзакцій. Нижче наведено основні етапи та принципи інтеграції ідемпотентних ключів у API платіжних систем на прикладі PayPal і Stripe.

Основні етапи інтеграції

а) Генерація ключа на стороні клієнта. Першим кроком є генерація ідемпотентного ключа на стороні клієнта перед відправленням запиту на сервер.

Цей ключ може бути створений за допомогою алгоритмів генерації UUID або інших методів, що забезпечують унікальність ключа.

б) Включення ключа у запит. Ідемпотентний ключ додається до заголовку HTTP-запиту. Це забезпечує передачу ключа разом із даними транзакції до серверу платіжної системи. Наприклад, у Stripe це виглядає так як показано на рисунку 1.1

```http
POST /v1/charges
Idempotency-Key: a1b2c3d4-e5f6-7g8h-9i0j-k1l2m3n4o5p6
```

Рисунок 1.1- Приклад ключа в Stripe

Обробка запиту на сервері

а) Перевірка наявності ключа. На сервері платіжної системи відбувається перевірка наявності ідемпотентного ключа у запиті. Якщо ключ відсутній, запит обробляється як звичайний.

б) Збереження стану транзакції. Якщо ключ присутній, система перевіряє, чи існує вже запис з цим ключем у базі даних або кеші (наприклад, у Redis). Якщо запису не існує, система обробляє транзакцію як нову, зберігаючи інформацію про неї разом з ідемпотентним ключем.

в) Повторна обробка запиту. Якщо сервер отримує повторний запит з тим самим ідемпотентним ключем, система не виконує нову транзакцію, а повертає результат першої обробки, використовуючи збережені дані. Це дозволяє уникнути дублювання операцій.

Приклади інтеграції у конкретних системах:

У Stripe розробники можуть додавати ідемпотентний ключ до заголовків HTTP-запитів для різних типів транзакцій, таких як створення платежів або повернення коштів. Stripe зберігає результати транзакцій, асоційованих з ключами, протягом 24 годин. Якщо повторний запит з тим самим ключем надходить протягом цього часу, Stripe повертає результат оригінального запиту, запобігаючи повторній обробці [12].

У PayPal ідемпотентні ключі використовуються аналогічним чином. Розробники можуть додавати ключ до заголовку запиту через API, що дозволяє системі визначити, чи був запит вже оброблений. Якщо ключ був використаний, PayPal повертає результат попередньої транзакції, що запобігає дублюванню платежів.

Переваги інтеграції

а) Підвищена надійність. Інтеграція ідемпотентних ключів підвищує надійність платіжних систем, забезпечуючи, що кожна транзакція буде оброблена лише один раз.

б) Зменшення навантаження. Цей механізм допомагає знизити навантаження на сервери, оскільки повторні запити не призводять до додаткової обробки транзакцій.

в) Покращення користувацького досвіду. Користувачі можуть бути впевнені, що їхні платежі будуть оброблені правильно, навіть якщо вони випадково надішлють запит кілька разів.

Інтеграція ідемпотентних ключів у API платіжних систем є критично важливою для забезпечення стабільної, безпечної та ефективної роботи платіжних платформ, що є основою для підтримки довіри клієнтів та безперебійного функціонування бізнесу.

1.2.3 Аналіз рішень по обробці ключів

Аналіз рішень по обробці ідемпотентних ключів є важливим аспектом для розуміння, як платіжні системи забезпечують надійність та безпеку транзакцій. У цьому контексті розглянемо підходи, які використовують такі системи, як PayPal та Stripe, для ефективного управління ідемпотентними ключами та обробки запитів.

Основні аспекти обробки ідемпотентних ключів є:

Зберігання ключів:

а) Бази даних. Одним із основних методів зберігання ідемпотентних ключів є використання баз даних. Кожен ідемпотентний ключ асоціюється з транзакцією і зберігається разом з інформацією про цю транзакцію. Наприклад, при успішному завершенні транзакції, ключ і відповідні дані зберігаються у базі даних для подальшої перевірки повторних запитів.

б) Кешування. Інший підхід полягає у використанні систем кешування, таких як Redis. Кешування дозволяє швидко зберігати і отримувати дані про транзакції, зменшуючи навантаження на базу даних і забезпечуючи швидкий доступ до інформації про оброблені запити. Цей метод особливо ефективний для систем з високою частотою запитів.

Перевірка наявності ключа:

а) Порівняння збережених ключів. Коли система отримує новий запит з ідемпотентним ключем, вона перевіряє, чи існує вже запис з цим ключем у базі даних або кеші. Якщо запис існує, це означає, що транзакція вже була оброблена, і система повертає результат попередньої транзакції.

б) Обробка нового запиту. Якщо ключ не знайдено у збережених записах, система обробляє запит як новий. Після успішного завершення транзакції, новий ідемпотентний ключ та відповідні дані зберігаються для подальшої перевірки.

Управління терміном дії ключів:

а) Термін зберігання. Ідемпотентні ключі мають обмежений термін дії, щоб уникнути накопичення великої кількості даних і забезпечити актуальність інформації. Наприклад, у Stripe термін зберігання ідемпотентних ключів зазвичай становить 24 години. Це означає, що повторний запит з тим самим ключем буде оброблений як новий після закінчення цього терміну.

б) Автоматичне видалення. Системи можуть автоматично видаляти старі ключі після завершення їх терміну дії. Це допомагає зберігати базу даних або кеш у чистоті та ефективності.

PayPal використовує комбінацію баз даних і кешування для обробки ідемпотентних ключів. Коли запит надходить з ідемпотентним ключем, система перевіряє кеш, щоб швидко визначити, чи був запит вже оброблений. Якщо ключ не знайдено в кеші, відбувається перевірка у базі даних. Цей підхід забезпечує високу швидкість обробки повторних запитів та надійність зберігання інформації.

Stripe акцентує увагу на швидкості обробки запитів і використовує Redis для кешування ідемпотентних ключів. Це дозволяє забезпечити швидкий доступ до інформації про транзакції, зменшуючи навантаження на основну базу даних. Термін зберігання ключів у Redis зазвичай становить 24 години, після чого ключі автоматично видаляються.

Ефективне управління ідемпотентними ключами є критично важливим для забезпечення надійності та безпеки платіжних систем. Використання баз даних і кешування дозволяє швидко обробляти повторні запити, запобігаючи дублюванню транзакцій. Обмеження терміну дії ключів і автоматичне видалення старих записів допомагає підтримувати ефективність системи. Підходи, які використовуються PayPal та Stripe, демонструють, як можна успішно впровадити ці механізми для забезпечення стабільної і надійної роботи платіжних платформ.

1.3 Ідемпотентні ключі в API

1.3.1 Визначення і значення ідемпотентності в API

У контексті API ідемпотентність означає, що виконання запиту кілька разів призводить до одного й того самого стану сервера. Наприклад, багато HTTP методів, таких як GET, PUT, DELETE, визначені як ідемпотентні, оскільки вони можуть виконуватися кілька разів без небажаних побічних ефектів. Проте, POST і PATCH запити зазвичай не є ідемпотентними, оскільки вони можуть змінювати стан сервера з кожним новим запитом [13, 14].

Ідемпотентні ключі дозволяють клієнту чітко вказувати, чи повторюється запит через невдачу, чи це нова операція. Це особливо важливо для транзакцій, що мають вирішальне значення, таких як платежі. Без ідемпотентних ключів клієнт і сервер можуть не мати узгодженості щодо стану запиту, що може призвести до небажаних повторних операцій.

У платіжних системах ідемпотентність має критичне значення для забезпечення надійності та безпеки транзакцій. Вона запобігає дублюванню платежів, що може статися через проблеми з мережею або інші технічні збої. Розглянемо приклад, коли клієнт надсилає запит на платіж і встановлює таймаут на 2 секунди – рисунок 1.2.

```
await axios.post(
    '/payments',
    { to: 'user@example.com', value: 2000 },
    { timeout: 2000 }
);
```

Рисунок 1.2- Приклад встановлення таймеру на 2 секунди

Якщо запит не завершується протягом 2 секунд, HTTP клієнт може вважати його невдалим і запропонувати повторити спробу. Проте, сервер може вже успішно завершити платіж, але не встигнути надіслати відповідь. Це створює невизначеність щодо стану транзакції [15].

З використанням ідемпотентних ключів, клієнт може уникнути цієї проблеми (див. рис. 1.3).

Після завершення запиту будь-які майбутні запити з тим самим ідемпотентним ключем не створять нову операцію — вони повернуть збережену відповідь з попереднього запиту. Це означає, що клієнт може безпечно повторити запит, якщо не впевнений, чи сервер отримав його.

```
curl https://api.stripe.com/v1/charges \
  --user sk_test_4eC39HqLyjWDarjtT1zdp7dc: \
  --header "Idempotency-Key: uWeBuDsZPxxvdhND" \
  --data amount=2000 \
  --data currency=usd \
  --data source=tok_mastercard \
  --data-urlencode description="Creating a charge"
```

Рисунок 1.3- Приклад використання ідемпотентного ключа в Stripe API

Розглянемо приклад API без ідемпотентності – рисунок 1.4

```
const express = require("express");
const app = express();
const port = process.env.PORT || 3001;
app.post("/things", (req, res) =>
  res.json({
    message: `Created a new thing: ${Math.random() * 100}`,
  })
);
app.listen(port, () => console.log(`Listening on port ${port}`));
```

Рисунок 1.4- Приклад API без ідемпотентності

Кожен запит до /things створює новий запис, що унеможливлює безпечне повторення запитів. Додавання ідемпотентності до цього API можна побачити на рисунку 1.5:

```
// Load up the middleware
const idempotency = require("express-idempotency");
// Register idempotency middleware
app.use(idempotency.idempotency());
// Updated route
app.post("/things", (req, res) => {
  const idempotencyService = idempotency.getSharedIdempotencyService();
  if (idempotencyService.isHit(req)) {
    return;
  }
  res.json({
    message: `New random number: ${Math.random() * 100}`,
  });
});
```

Рисунок 1.5- Створення нових записів

З цим кодом, API розпізнає повторні запити і повертає ту саму відповідь, якщо ідемпотентний ключ вже використаний, що забезпечує стабільність та надійність транзакцій.

1.3.2 Впровадження ідемпотентних ключів

Генерація ідемпотентних ключів є важливим кроком у забезпеченні надійності API, особливо в платіжних системах. Ідемпотентні ключі дозволяють клієнту та серверу чітко ідентифікувати повторні запити та уникати дублювання операцій [16].

Ідемпотентні ключі можуть бути згенеровані клієнтом або автоматично створені системою. На рисунку 1.6 є приклад генерації ідемпотентного ключа у JavaScript, використовуючи бібліотеку uuid.

```
const { v5: uuidv5 } = require('uuid');
// Генерація ідемпотентного ключа при завантаженні форми
const idempotencyKey = uuidv5();
// Використання ключа у запиті
fetch("https://example.org/api/things", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Idempotency-Key": idempotencyKey,
  },
  body: JSON.stringify(req.body),
})
.then((response) => response.json());
```

Рисунок 1.6- Генерація ідемпотентного ключа

У цьому прикладі ідемпотентний ключ генерується при завантаженні форми, що дозволяє уникнути плутанини між повторними запитами та новими операціями.

На серверній стороні ідемпотентні ключі використовуються для розпізнавання повторних запитів. Ми використовували реалізацію ідемпотентності з допомогою Node.js з Express (див. рис. 1.7)

```
const express = require("express");
const idempotency = require("express-idempotency");
const app = express();
const port = process.env.PORT || 3001;
// Підключення middleware для ідемпотентності
app.use(idempotency.idempotency());
// Оновлений маршрут
app.post("/things", (req, res) => {
  const idempotencyService = idempotency.getSharedIdempotencyService();
  if (idempotencyService.isHit(req)) {
    return;
  }
  res.json({
    message: `New random number: ${Math.random() * 100}`,
  });
});
app.listen(port, () => console.log(`Listening on port ${port}`));
```

Рисунок 1.7- Реалізація ідемпотентності на серверній стороні

Цей код використовує middleware для обробки ідемпотентних ключів. Якщо ключ вже був використаний, сервер повертає збережений результат, уникаючи повторної обробки запиту.

Зберігання ідемпотентних ключів є важливим аспектом для забезпечення їх ефективного використання. Ключі повинні зберігатися протягом обмеженого часу, щоб запобігти накопиченню великої кількості даних і зберігати актуальність інформації [17].

У виробничих середовищах використання Redis для зберігання ідемпотентних ключів є хорошою практикою, оскільки це забезпечує спільний доступ до даних з усіх серверів і незалежність від розгортання серверів (рис 1.8).

```
const express = require("express");
const idempotency = require("express-idempotency");
const redis = require("redis");
const app = express();
const port = process.env.PORT || 3001;
// Підключення Redis-клієнта
const redisClient = redis.createClient();
// Підключення middleware для ідемпотентності з використанням Redis
app.use(idempotency.idempotency({
  store: new idempotency.RedisStore(redisClient)
}));
// Оновлений маршрут
app.post("/things", (req, res) => {
  const idempotencyService = idempotency.getSharedIdempotencyService();
  if (idempotencyService.isHit(req)) {
    return;
  }
  res.json({
    message: `New random number: ${Math.random() * 100}`,
  });
});
app.listen(port, () => console.log(`Listening on port ${port}`));
```

Рисунок 1.8- Redis для зберігання ідемпотентних ключів

У цьому прикладі ідемпотентні ключі та їх відповіді кешуються у Redis. Це дозволяє зберігати ключі навіть після перезапуску серверів, запобігаючи дублюванню операцій.

Щоб уникнути проблем з накопиченням даних, ідемпотентні ключі повинні мати обмежений термін дії. Наприклад, у Stripe термін зберігання ідемпотентних ключів зазвичай становить 24 години. Це дозволяє ефективно управляти інформацією про транзакції та запобігає використанню ключів у майбутньому, що зменшує ризики безпеки.

Ідемпотентні ключі значно покращують надійність та безпеку платіжних систем, дозволяючи уникнути дублювання операцій та забезпечуючи стабільність роботи сервера навіть у випадку технічних збоїв або повторних запитів.

1.3.3 Приклади використання та уникнення ризиків

Stripe використовує ідемпотентні ключі для запобігання дублюванню транзакцій. Клієнти можуть включати ідемпотентний ключ у заголовок HTTP-запиту, що дозволяє серверу розпізнавати повторні запити та уникати дублювання операцій (див. рис. 1.9).

```
curl https://api.stripe.com/v1/customers \
  --user sk_test_4eC39HqLyjWDarjtT1zdp7dc: \
  --header "Idempotency-Key: a1b2c3d4e5f6g7h8i9j0" \
  --data email="customer@example.com" \
  --data description="New customer for example.com"
```

Рисунок 1.9- Приклад використання ідемпотентного ключа в Stripe API

У цьому прикладі клієнт надсилає запит на створення нового клієнта з ідемпотентним ключем у заголовку. Якщо цей запит буде повторно надісланий з тим самим ключем, сервер поверне збережену відповідь, запобігаючи створенню нового клієнта.

PayPal також використовує ідемпотентні ключі для забезпечення надійності своїх транзакцій. Розробники можуть додавати ідемпотентний ключ у заголовок запиту через API, що дозволяє системі визначити, чи був запит вже оброблений(див рис. 1.10).

У цьому прикладі ідемпотентний ключ PayPal-Request-Id використовується для уникнення дублювання виплат. Якщо цей запит буде надісланий повторно з тим самим ключем, сервер PayPal поверне результат попередньої транзакції .

Middleware дозволяє легко інтегрувати ідемпотентні ключі у ваші API, забезпечуючи обробку ключів без необхідності дублювати код у кожному маршруті. Використовуючи Express та бібліотеку express-idempotency, ви можете додати підтримку ідемпотентних ключів до вашого сервера(див рис. 1.11).

```
curl -X POST https://api.paypal.com/v1/payments/payouts \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer <Access-Token>" \
  --header "PayPal-Request-Id: unique-key-12345" \
  --data '{
    "sender_batch_header": {
      "sender_batch_id": "batch-12345",
      "email_subject": "You have a payment"
    },
    "items": [{
      "recipient_type": "EMAIL",
      "amount": {
        "value": "10.00",
        "currency": "USD"
      },
      "receiver": "receiver@example.com",
      "note": "Payment for your service",
      "sender_item_id": "item-12345"
    }]
  }'
```

Рисунок 1.10- Приклад ідемпотентного ключа PayPal-Request-Id

```
const express = require("express");
const idempotency = require("express-idempotency");
const app = express();
const port = process.env.PORT || 3001;
// Підключення middleware для ідемпотентності
app.use(idempotency.idempotency());
// Оновлений маршрут
app.post("/orders", (req, res) => {
  const idempotencyService = idempotency.getSharedIdempotencyService();
  if (idempotencyService.isHit(req)) {
    return;
  }
  res.json({
    message: `Order created: ${Math.random() * 100}`,
  });
});
app.listen(port, () => console.log(`Listening on port ${port}`));
```

Рисунок 1.11- Middleware для обробки ідемпотентних ключів

У цьому прикладі middleware обробляє ідемпотентні ключі для всіх POST-запитів до “/orders”. Якщо ключ вже був використаний, сервер поверне збережений результат, уникаючи повторної обробки запиту.

Для зберігання ідемпотентних ключів у виробничих середовищах використання Redis є гарною практикою. Redis забезпечує спільний доступ до даних з усіх серверів і незалежність від розгортання серверів(див рис. 1.12).

```
const express = require("express");
const idempotency = require("express-idempotency");
const redis = require("redis");
const app = express();
const port = process.env.PORT || 3001;
// Підключення Redis-клієнта
const redisClient = redis.createClient({
  host: 'localhost',
  port: 6379
});
// Підключення middleware для ідемпотентності з використанням Redis
app.use(idempotency.idempotency({
  store: new idempotency.RedisStore(redisClient)
}));
// Оновлений маршрут
app.post("/payments", (req, res) => {
  const idempotencyService = idempotency.getSharedIdempotencyService();
  if (idempotencyService.isHit(req)) {
    return;
  }
  res.json({
    message: `Payment processed: ${Math.random() * 100}`,
  });
});
app.listen(port, () => console.log(`Listening on port ${port}`));
```

Рисунок 1.12- Приклад інтеграції Redis з Express

У цьому прикладі ідемпотентні ключі та їх відповіді кешуються у Redis. Це дозволяє зберігати ключі навіть після перезапуску серверів, запобігаючи дублюванню операцій. Використання Redis також забезпечує високу швидкість доступу до даних, що покращує продуктивність системи.

Ідемпотентні ключі є важливим інструментом для забезпечення надійності та безпеки транзакцій в API. Вони дозволяють уникнути дублювання операцій при повторних запитах, забезпечуючи стабільність роботи сервера навіть у випадку технічних збоїв або повторних запитів.

2. АРХІТЕКТУРА ІДЕМПОТЕНТНОЇ СИСТЕМ. РОЗРОБКА ТА ТЕАЛІЗАЦІЯ СТРАТЕГІЇ УПРАВЛІННЯ ПЛАТИЖАМИ

Ідемпотентність у контексті платіжних систем означає здатність системи забезпечити, що повторний запит на виконання однієї й тієї ж операції не спричинить дублювання результату. Це досягається через використання унікальних ідентифікаторів транзакцій (ідемпотентних ключів), які дозволяють системі визначити, чи була дана операція вже оброблена.

Використання ідемпотентних систем для управління платежами є ефективним інструментом для забезпечення цілісності та надійності транзакцій у дропшипінгу. Розробка та впровадження стратегії, що базується на ідемпотентності, дозволяє мінімізувати ризики дублювання транзакцій та забезпечити стабільну роботу платіжних систем.

2.1 Структура ідемпотентних ключів для проєкту дропшипінгу

2.1.1 Технічні характеристики ключів

У нашому проєкті для дропшипінгу ми використовуємо ідемпотентні ключі для забезпечення надійності та цілісності платіжних транзакцій. Це допомагає уникнути дублювання транзакцій та знижує ризики помилок. Нижче наведено типи ідемпотентних ключів, що використовуються у нашій системі, та їхню структуру.

Ми використовуємо такі типи ідемпотентних ключів:

— Ключі замовлень (Order Keys): генеруються для кожного нового замовлення та використовуються для відстеження платіжної транзакції, пов'язаної з цим замовленням [18].

— Ключі сесій (Session Keys): створюються на початку користувацької сесії та використовуються для всіх транзакцій, що здійснюються протягом цієї сесії.

— Ключі повернень (Refund Keys): застосовуються для управління процесом повернення коштів, забезпечуючи унікальність кожної операції повернення.

Структура ідемпотентного ключа у нашому проєкті включає кілька компонентів, які забезпечують його унікальність та безпеку. Нижче наведено детальний опис структури кожного типу ключа.

а) Ключі замовлень (Order Keys):

```
ORD-{OrderID}-{Timestamp}-{MAC}
```

— ORD – префікс, що вказує на ключ замовлення.

— OrderID – унікальний ідентифікатор замовлення.

— Timestamp – мітка часу, що вказує на момент створення замовлення.

— MAC – код автентифікації повідомлення для забезпечення цілісності ключа.

б) Ключі сесій (Session Keys):

```
SES-{SessionID}-{Timestamp}-{MAC}
```

— SES – префікс, що вказує на ключ сесії.

— SessionID – унікальний ідентифікатор сесії користувача.

— Timestamp – мітка часу, що вказує на момент початку сесії.

— MAC – код автентифікації повідомлення для забезпечення безпеки ключа.

в) Ключі повернень (Refund Keys):

```
REF-{RefundID}-{OrderID}-{Timestamp}-{MAC}
```

— REF – префікс, що вказує на ключ повернення.

— RefundID – унікальний ідентифікатор операції повернення.

— OrderID – ідентифікатор замовлення, до якого прив'язаний повернення.

— Timestamp – мітка часу, що вказує на момент створення операції повернення.

— MAC – код автентифікації повідомлення для забезпечення цілісності ключа.

Нижче наведено приклади ключів для кожного типу:

Використання ідемпотентних ключів у нашому проєкті для дропшипінгу дозволяє забезпечити унікальність та надійність платіжних транзакцій, що є критичним для стабільної роботи системи. Правильна структура ключів, що включає унікальні ідентифікатори, мітки часу та коди автентифікації, допомагає мінімізувати ризики дублювання та підвищує загальну безпеку платіжної інфраструктури.

2.1.2 Стандарти та протоколи використання

Для забезпечення надійності та безпеки платіжних транзакцій у нашому проєкті для дропшипінгу ми використовуємо різні стандарти та протоколи. Це допомагає нам підтримувати високий рівень захисту даних і відповідати вимогам сучасних платіжних систем. Нижче наведено основні стандарти та протоколи, які ми задіяли.

Перш за все, ми дотримуємося стандарту PCI DSS (Payment Card Industry Data Security Standard), який забезпечує захист даних карткових платежів. Він встановлює вимоги до безпечного зберігання, обробки та передачі даних власників карток. Всі платіжні транзакції в нашій системі відповідають вимогам PCI DSS, що забезпечує захист даних клієнтів від несанкціонованого доступу.

Для забезпечення безпечної передачі даних між клієнтами та серверами ми використовуємо протокол TLS (Transport Layer Security). Цей протокол забезпечує шифрування інформації під час передачі, що дозволяє захистити платіжні дані від можливих атак під час їхнього транспортування [19].

Протокол OAuth 2.0, який ми також задіяли, дозволяє додаткам отримувати обмежений доступ до користувацьких даних без передачі паролів. Це забезпечує безпечну авторизацію користувачів та інтеграцію з сторонніми сервісами, що підвищує загальний рівень безпеки нашої системи.

Ми використовуємо JSON Web Tokens (JWT) для створення токенів аутентифікації та авторизації. JWT є відкритим стандартом для передачі інформації між сторонами у вигляді JSON-об'єктів, що підписуються для забезпечення цілісності та автентичності даних. Це дозволяє нам безпечно обмінюватися даними між компонентами нашої системи.

Також ми дотримуємося стандарту ISO/IEC 27001, який визначає вимоги до системи управління інформаційною безпекою (ISMS). Цей стандарт забезпечує захист інформаційних активів, що дозволяє нам ефективно керувати інформаційною безпекою та забезпечувати конфіденційність, цілісність і доступність даних.

Для перевірки цілісності та автентичності повідомлень ми використовуємо HMAC (Hash-based Message Authentication Code). HMAC є механізмом, що використовує криптографічні хеш-функції та секретний ключ. Це дозволяє нам забезпечити цілісність та автентичність ідемпотентних ключів та інших критичних даних у нашій системі.

Застосування цих стандартів та протоколів забезпечує високий рівень безпеки та надійності платіжних транзакцій у нашому проєкті для дропшипінгу. Це дозволяє нам захищати дані наших клієнтів і забезпечувати безперебійну роботу системи, відповідаючи сучасним вимогам до безпеки та захисту інформації.

2.2 Механізм роботи ключів

Механізм роботи ідемпотентних ключів дозволяє уникнути дублювання транзакцій, забезпечуючи надійність та цілісність платіжних операцій. Загальний процес роботи можна побачити на діаграма послідовності(див. рис. 2.1).

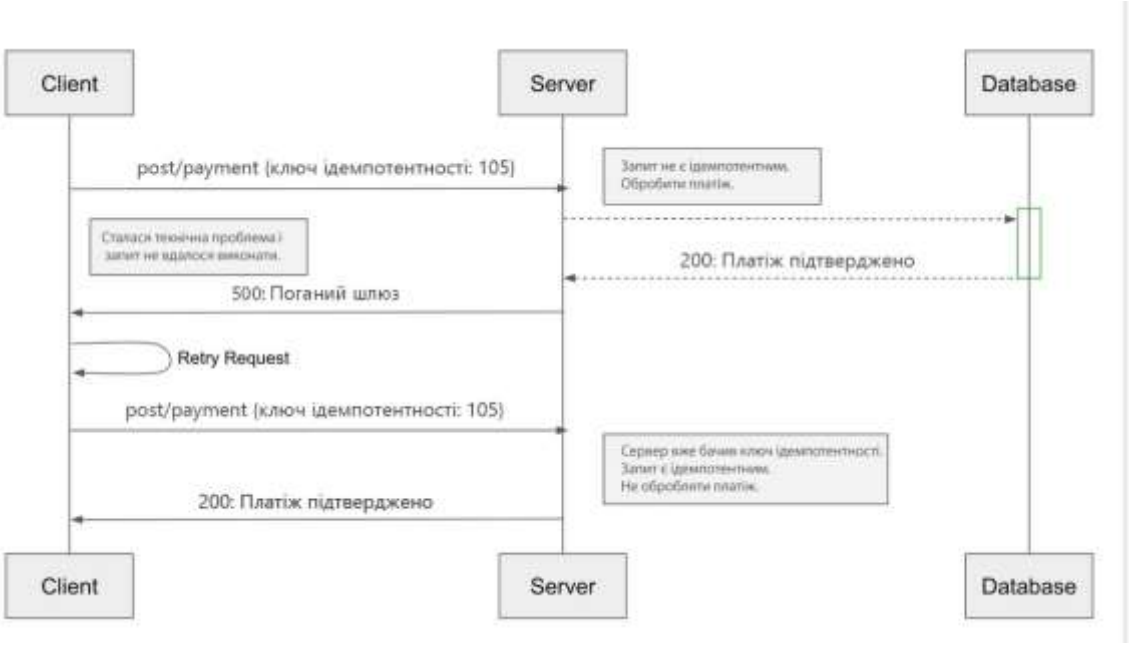

Рисунок 2.1 – Діаграма послідовності для ключа ідемпотентності

Діаграма послідовності показує взаємодію між клієнтом, сервером і базою даних під час процесу платіжної транзакції з використанням ідемпотентних ключів. Вона демонструє, як ідемпотентні ключі допомагають уникнути дублювання платежів у випадку технічних проблем або повторних запитів.

Процес роботи ідемпотентних ключів

а) Відправка запиту клієнтом

— Клієнт надсилає запит на сервер для проведення платежу з ідемпотентним ключем. Наприклад, post/payment {idempotency-key: 105}.

б) Обробка запиту сервером

— Сервер отримує запит і перевіряє, чи цей ідемпотентний ключ вже використовувався раніше. Якщо запит є неідемпотентним (новим), сервер продовжує обробку платежу.

— Якщо запит є новим, сервер обробляє платіж і надсилає підтвердження платежу з кодом 200 (200: Payment Confirmed). Інформація про платіж зберігається в базі даних.

в) Виникнення технічної проблеми

— Якщо під час обробки запиту виникає технічна проблема, сервер повертає клієнту повідомлення про помилку (500: Bad Gateway).

— У цьому випадку клієнт отримує повідомлення про те, що сталася технічна проблема і запит не було успішно завершено.

г) Повторний запит клієнта

— Клієнт, отримавши повідомлення про помилку, повторно надсилає той самий запит з ідемпотентним ключем (post/payment {idempotency-key: 105}).

д) Обробка повторного запиту сервером

— Сервер отримує повторний запит і знову перевіряє ідемпотентний ключ. Оскільки сервер вже бачив цей ідемпотентний ключ раніше, він визначає, що запит є ідемпотентним і не обробляє платіж повторно.

— Сервер повертає клієнту той самий результат, що й раніше – підтвердження платежу з кодом 200 (200: Payment Confirmation).

Цей механізм гарантує, що навіть у випадку технічних збоїв або повторних запитів платіжна операція не буде проведена двічі. Ідемпотентні ключі дозволяють уникнути дублювання транзакцій та забезпечити надійність платіжних систем.

2.3 Redis та базах даних

Імплементація ідемпотентних ключів у платіжних системах потребує надійних і швидких механізмів зберігання даних. Redis та традиційні реляційні бази

даних є популярними виборами для цієї задачі. У цьому розділі розглянемо, як реалізувати ідемпотентність за допомогою Redis та баз даних (див рис. 2.2).

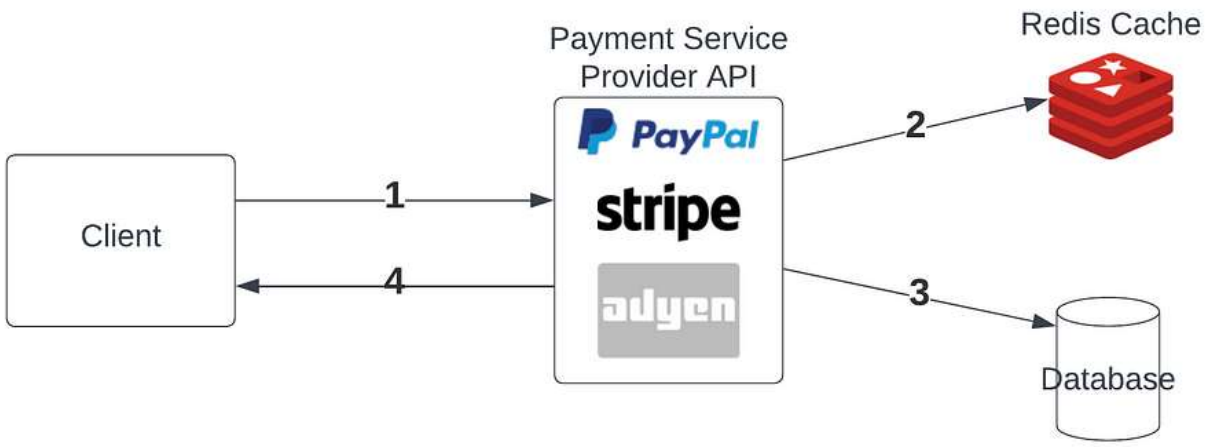

Рисунок 2.2 – Зв’язки між основними компонентами

Redis є високопродуктивною системою зберігання даних у пам'яті, що забезпечує швидкий доступ до даних. Вона підходить для зберігання ідемпотентних ключів завдяки своїм властивостям низької затримки та підтримці наборів ключів із обмеженим строком життя.

2.1.1 Методи зберігання ідемпотентних ключів

Зберігання ідемпотентних ключів є критично важливим аспектом для забезпечення надійності та цілісності платіжних транзакцій. Ідемпотентні ключі дозволяють визначити, чи запит вже був оброблений, тим самим запобігаючи дублюванню транзакцій. У цьому розділі розглянемо основні методи зберігання ідемпотентних ключів, а також їхні переваги та недоліки.

Реляційні бази даних:

— Опис: Ідемпотентні ключі зберігаються у таблицях реляційних баз даних, таких як MySQL або PostgreSQL. Кожен запис включає унікальний ідентифікатор ключа, мітку часу створення та статус обробки запиту.

— Переваги: Надійність, підтримка складних запитів, транзакційна цілісність.

— Недоліки: Можливі проблеми з масштабованістю при великій кількості транзакцій, складність у налаштуванні для високого навантаження.

Вибір методу зберігання ідемпотентних ключів залежить від конкретних вимог проєкту, таких як обсяги транзакцій, потреба у швидкості доступу, наявність складних запитів та необхідність у транзакційній цілісності. У багатьох випадках доцільно використовувати комбінований підхід, зберігаючи ідемпотентні ключі як у реляційній або NoSQL базі даних для постійного зберігання, так і в in-memory сховищі для швидкого доступу (див. рис. 2.3).

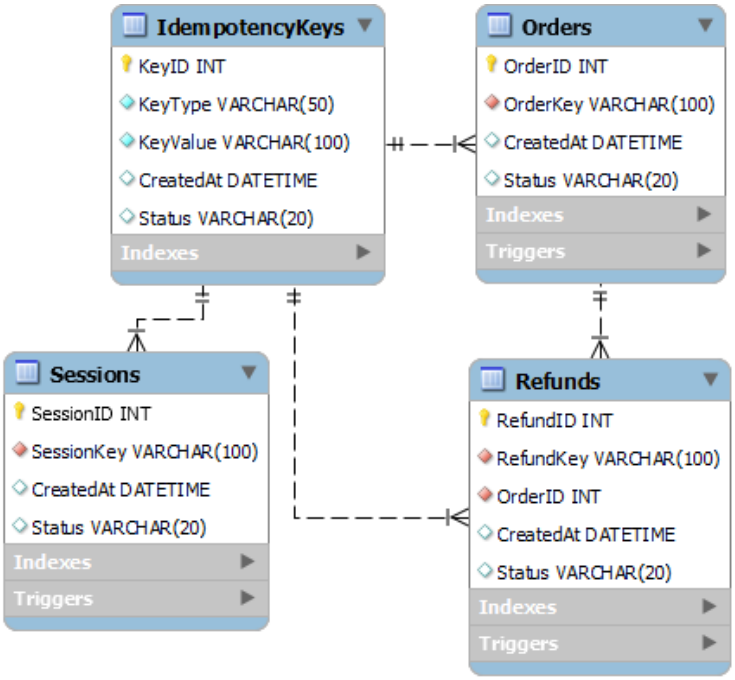

Рисунок 2.3 – Зв’язки між основними компонентами

Зв'язки між таблицями

— IdempotencyKeys → Orders: Таблиця Orders посилається на IdempotencyKeys через поле OrderKey, яке є зовнішнім ключем, що забезпечує унікальність та запобігає дублюванню замовлень.

— IdempotencyKeys → Sessions: Таблиця Sessions посилається на IdempotencyKeys через поле SessionKey, яке є зовнішнім ключем, що забезпечує унікальність та цілісність сесій.

— IdempotencyKeys → Refunds: Таблиця Refunds посилається на IdempotencyKeys через поле RefundKey, яке є зовнішнім ключем, що забезпечує унікальність та цілісність повернень.

— Orders → Refunds: Таблиця Refunds посилається на Orders через поле OrderID, яке є зовнішнім ключем, що зв'язує повернення з відповідними замовленнями.

Ця модель забезпечує надійне зберігання ідемпотентних ключів та забезпечує цілісність даних, запобігаючи дублюванню транзакцій.

2.1.2 Загальний вигляд бази даних

Даний підхід в зберіганні даних ключів ідемпотентності забезпечує гнучкість в зберіганні та відповідає загальному контексту предметної області (див. рис. 2.4).

База даних має наступний набір основних таблиць з даними:

— IdempotencyKeys. Таблиця для зберігання всіх ідемпотентних ключів. Включає тип ключа, значення ключа, дату створення та статус.

Поля: KeyID, KeyType, KeyValue, CreatedAt, Status.

— Orders. Таблиця для зберігання інформації про замовлення. Включає ідемпотентний ключ замовлення, ідентифікатор клієнта, дату створення та статус.

Поля: OrderID, OrderKey, CustomerID, CreatedAt, Status.

— Sessions. Таблиця для зберігання інформації про сесії користувачів. Включає ідемпотентний ключ сесії, ідентифікатор користувача, дату створення та статус.

Поля: SessionID, SessionKey, UserID, CreatedAt, Status.

— Refunds. Таблиця для зберігання інформації про повернення. Включає ідемпотентний ключ повернення, ідентифікатор замовлення, дату створення та статус.

Поля: RefundID, RefundKey, OrderID, CreatedAt, Status.

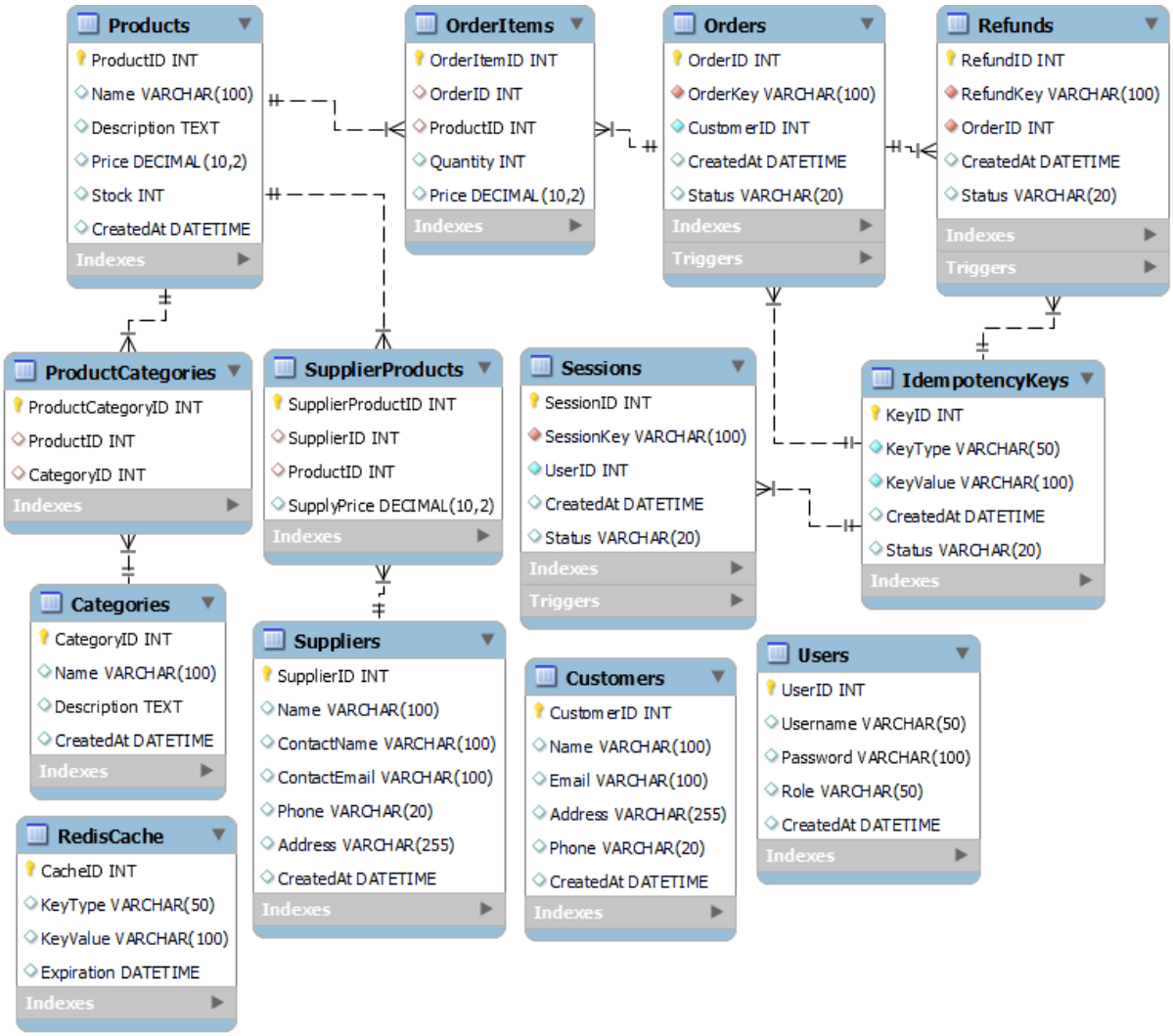

Рисунок 2.4 – Основні сутності бази даних

— Customers. Таблиця для зберігання інформації про клієнтів. Включає ім'я, електронну пошту, адресу, телефон та дату створення.

Поля: CustomerID, Name, Email, Address, Phone, CreatedAt.

— Users. Таблиця для зберігання інформації про користувачів. Включає ім'я користувача, пароль, роль та дату створення.

Поля: UserID, Username, Password, Role, CreatedAt.

— RedisCache. Таблиця для зберігання кешованих ідемпотентних ключів у Redis (псевдокод для ілюстрації). Включає тип ключа, значення ключа та дату закінчення терміну дії.

Поля: CacheID, KeyType, KeyValue, Expiration.

— Products. Таблиця для зберігання інформації про продукти. Включає назву, опис, ціну, кількість на складі та дату створення.

Поля: ProductID, Name, Description, Price, Stock, CreatedAt.

— OrderItems. Таблиця для зберігання інформації про замовлені продукти. Включає ідентифікатор замовлення, ідентифікатор продукту, кількість та ціну.

Поля: OrderItemID, OrderID, ProductID, Quantity, Price.

— Suppliers. Таблиця для зберігання інформації про постачальників. Включає назву, контактне ім'я, контактну електронну пошту, телефон, адресу та дату створення.

Поля: SupplierID, Name, ContactName, ContactEmail, Phone, Address, CreatedAt.

— SupplierProducts. Таблиця для зберігання зв'язків постачальників з продуктами. Включає ідентифікатор постачальника, ідентифікатор продукту та ціну постачання.

Поля: SupplierProductID, SupplierID, ProductID, SupplyPrice.

— Categories. Таблиця для зберігання інформації про категорії продуктів. Включає назву, опис та дату створення.

Поля: CategoryID, Name, Description, CreatedAt.

— ProductCategories. Таблиця для зберігання зв'язків продуктів з категоріями. Включає ідентифікатор продукту та ідентифікатор категорії.

Поля: ProductCategoryID, ProductID, CategoryID.

Зв'язки між таблицями

— Orders посилається на IdempotencyKeys через OrderKey.

— Sessions посилається на IdempotencyKeys через SessionKey.

— Refunds посилається на IdempotencyKeys через RefundKey і на Orders через OrderID.

— Orders посилається на Customers через CustomerID.

— Sessions посилається на Users через UserID.

— OrderItems посилається на Orders через OrderID і на Products через ProductID.

— SupplierProducts посилається на Suppliers через SupplierID і на Products через ProductID.

— ProductCategories посилається на Products через ProductID і на Categories через CategoryID.

Ця модель бази даних забезпечує зберігання всіх необхідних даних та зв'язків між ними для роботи дропшипінг магазину, включаючи ідемпотентні ключі для забезпечення надійності транзакцій.

2.1.3 Переваги використання Redis для кешування ключів

Використання Redis для кешування ідемпотентних ключів у нашому проєкті має ряд переваг, які роблять його ідеальним вибором для забезпечення надійності та швидкості роботи системи.

Алгоритм використання Redis для кешування ідемпотентних ключів:

1) Створення ідемпотентного ключа та збереження його в Redis:

— При отриманні нового запиту генерується ідемпотентний ключ.

— Ключ зберігається у Redis з встановленням часу життя (TTL).

2) Перевірка наявності ключа в Redis:

— При отриманні повторного запиту перевіряється наявність ключа у Redis.

— Якщо ключ існує, система повертає результат з кешу, не обробляючи запит повторно.

— Якщо ключ не існує, запит обробляється і ключ зберігається у Redis.

Нижче наведено приклад коду на JavaScript з використанням бібліотеки ioredis для кешування ідемпотентних ключів у Redis(див. рис. 2.5)

```
const Redis = require('ioredis');
const { v4: uuidv4 } = require('uuid');

// Підключення до Redis
const redis = new Redis({
  host: 'localhost',
  port: 6379,
  db: 0,
});
// Функція для генерації ідемпотентного ключа
function generateIdempotencyKey() {
  return uuidv4();
}
// Функція для збереження ідемпотентного ключа в Redis з TTL (Time-To-Live)
async function saveIdempotencyKey(key, ttl = 3600) {
  await redis.set(key, 'exists', 'EX', ttl);
}
// Функція для перевірки наявності ідемпотентного ключа в Redis
async function checkIdempotencyKey(key) {
  const exists = await redis.exists(key);
  return exists === 1;
}
// Приклад використання
(async () => {
  const idempotencyKey = generateIdempotencyKey();
  // Збереження ключа в Redis
  await saveIdempotencyKey(idempotencyKey);
  // Перевірка наявності ключа в Redis
  if (await checkIdempotencyKey(idempotencyKey)) {
    console.log('Key exists in Redis. Returning cached response.');
  } else {
    console.log('Key does not exist in Redis. Processing request.');
  }
})();
```

Рисунок 2.5 – ‘ioredis’ для кешування ідемпотентних ключів у Redis

Використання Redis для кешування ідемпотентних ключів у нашій системі дозволяє значно підвищити швидкість доступу до даних, зменшити навантаження

на основну базу даних і забезпечити надійність та стабільність роботи системи. Це робить Redis ефективним інструментом для реалізації кешування у сучасних високонавантажених додатках.

2.4 Структура проєкту, основні компоненти

Робота має велику кількість структурних одиниць. Так як, за основу було обрано мову програмування JS, що реалізовує функціональний підхід до розробки, було вирішено зобразити загальну структуру у вигляді діаграми компоненті. Графічне відображення діаграми компонентів допоможе краще зрозуміти структуру вашого проєкту та взаємодію між його частинами (див. Рис).

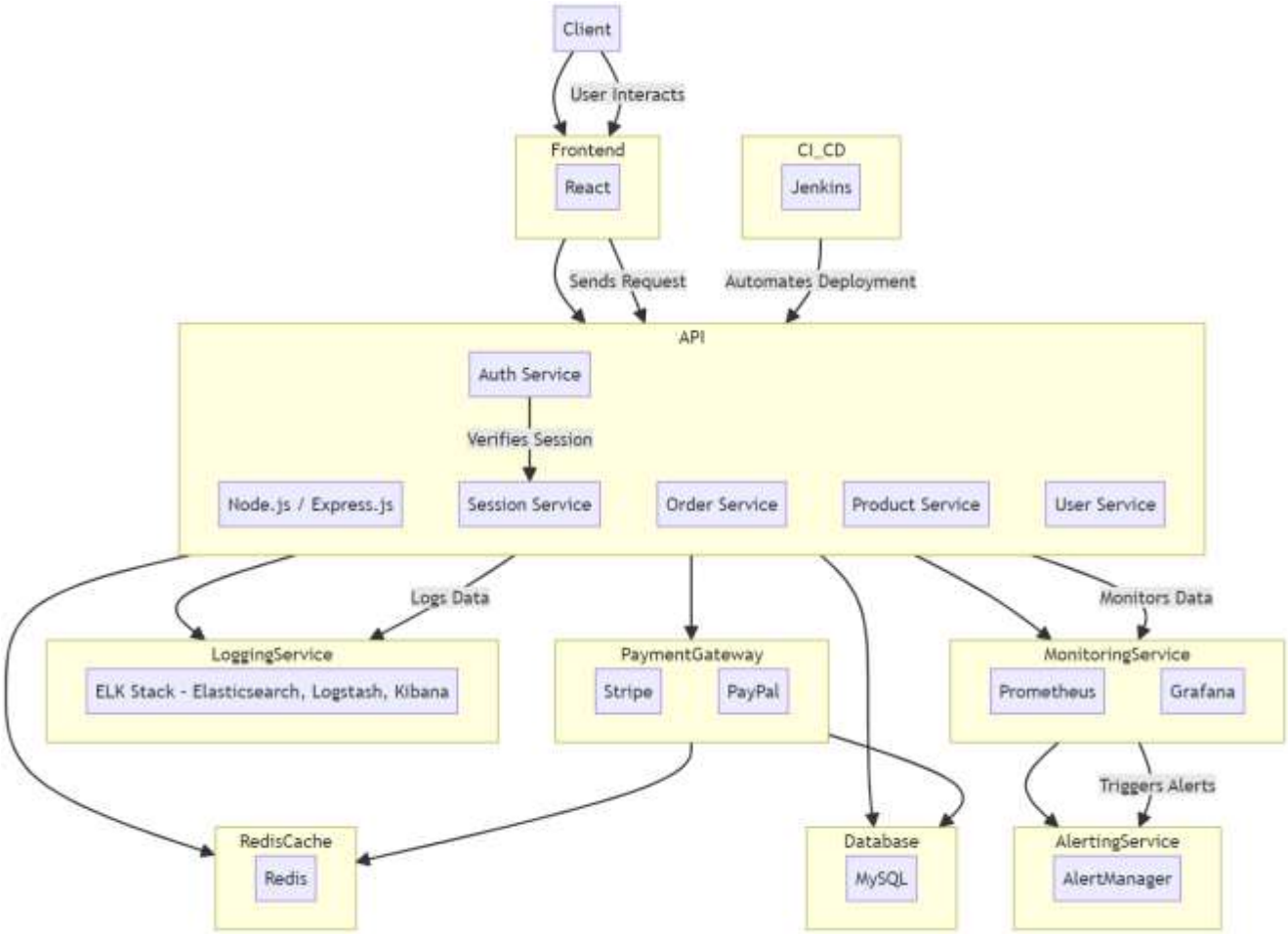

Рисунок 2.5 -Діаграма основних компонентів

Опис компонентів:

— Client: Клієнтська частина, яка взаємодіє з користувачем через браузер.

— Frontend: Фронтенд додаток, побудований з використанням фреймворків React, Vue або Angular.

— API: Серверна частина, побудована на Node.js з використанням Express.js для обробки запитів.

- AuthService: Сервіс для аутентифікації та авторизації користувачів.
- OrderService: Сервіс для обробки замовлень.
- ProductService: Сервіс для управління продуктами.
- UserService: Сервіс для управління користувачами.
- SessionService: Сервіс для управління сесіями користувачів.

— RedisCache: In-memory кеш для зберігання ідемпотентних ключів та прискорення доступу до даних.

— Database: Основна база даних для зберігання даних про замовлення, користувачів, продукти тощо.

— PaymentGateway: Платіжні шлюзи для обробки онлайн-платежів, такі як Stripe, PayPal і Adyen.

— LoggingService: Система для логування даних, використовуючи ELK Stack (Elasticsearch, Logstash, Kibana).

— MonitoringService: Система для моніторингу даних, використовуючи Prometheus і Grafana.

— AlertingService: Система для сповіщення, використовуючи AlertManager.

— CI/CD: Система для безперервної інтеграції та розгортання, використовуючи Jenkins.

Розширена діаграма компонентів надає повніший огляд архітектури проєкту, включаючи всі основні сервіси та їх взаємодію, що допоможе краще зрозуміти структуру та функціональність системи.

2.5 Реалізований проєкт

При першому успішному відпрацюванні запиту ви отримаєте відповідь із заголовками, що вказують на успішну обробку запиту. При повторному відправленні запиту з тим самим ідемпотентним ключем, сервер розпізнає цей запит як дубльований і повертає той самий результат, що й при першій обробці (див. рис. 2.6).

1. Повторний запит із тим самим ідемпотентним ключем	2. Успішне відпрацювання запиту
HTTP/1.1 200 OK	HTTP/1.1 200 OK
Content-Type: application/json	Content-Type: application/json
Content-Length: 256	Content-Length: 256
Cache-Control: no-cache	Cache-Control: no-cache
Date: Wed, 29 May 2024 12:01:00 GMT	Date: Wed, 29 May 2024 12:00:00 GMT
Server: nginx/1.19.0	Server: nginx/1.19.0
Connection: keep-alive	Connection: keep-alive
ETag: "5d8c72a7-4e8a"	ETag: "5d8c72a7-4e8a"
Set-Cookie: sessionId=abc123; Path=/; HttpOnly	Set-Cookie: sessionId=abc123; Path=/; HttpOnly
Access-Control-Allow-Origin: *	Access-Control-Allow-Origin: *
Idempotency-Key: e7bfa2d5-ff0b-4cd5-8b33-54b4ef0a2a29	Idempotency-Key: e7bfa2d5-ff0b-4cd5-8b33-54b4ef0a2a29
Idempotency-Replayed: true	

Рисунок 2.6 – Результати запитів

Ключові елементи заголовкових файлів:

— Idempotency-Key: Це ключ, що дозволяє визначити, чи був запит раніше оброблений, і забезпечити ідемпотентність.

— Idempotency-Replayed: Додатковий заголовок, який сигналізує, що запит вже був оброблений раніше і повертається кешований результат.

Ці заголовки допомагають клієнтам і серверам розпізнавати та обробляти повторні запити правильно, уникати дублювання операцій і забезпечувати надійність та узгодженість у веб-додатках.

У нашому проєкті ми реалізували механізм ідемпотентності для забезпечення надійності та уникнення дублювання транзакцій під час обробки платіжних

запитів. Використання ідемпотентних ключів дозволяє розпізнавати повторні запити та відповідати на них тим самим результатом, що був отриманий при першій обробці запиту. Це особливо важливо в системах, де можливі технічні збої або користувачі можуть випадково відправляти один і той самий запит кілька разів.

Для кожного нового запиту на оплату клієнт генерує унікальний ідемпотентний ключ, який передається разом із запитом. Сервер перевіряє наявність цього ключа в Redis. Якщо ключ відсутній, сервер обробляє запит, зберігає результат у Redis з часом життя (TTL) і повертає результат клієнту. Якщо ключ вже існує, сервер повертає раніше збережений результат, не обробляючи запит повторно.

Забезпечення надійності полягає в уникненні дублювання транзакцій. Це дозволяє зменшити навантаження на сервер, оскільки повторні запити не обробляються повторно. Це також покращує користувацький досвід, запобігаючи випадковим подвійним оплатам і зменшуючи час очікування відповіді(рис. 2.7).

Рисунок 2.7 – Відповідь сервера

Реалізація ідемпотентності у нашій системі дозволяє забезпечити стабільність і надійність обробки платіжних запитів, мінімізуючи ризики дублювання транзакцій. Це важливий аспект для будь-якої системи, що обробляє критичні фінансові операції, підвищуючи її ефективність і довіру користувачів.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Ліквідація наслідків надзвичайних ситуацій

Проблема запобігання виникнення надзвичайних ситуацій техногенного походження та ліквідації їх наслідків в Україні є однією з найактуальніших.

У статті 20 Кодексу цивільного захисту України наголошено, що керівництво підприємств, установ та організацій незалежно від форм власності і підпорядкування забезпечує своїх працівників засобами індивідуального та колективного захисту, організовує здійснення евакуаційних заходів, створює сили для ліквідації наслідків надзвичайних ситуацій та забезпечує їх готовність до практичних дій, виконує інші заходи з ЦЗ і несе пов’язані з цим матеріальні та фінансові витрати [20-24].

Сутність рятувальних та інших невідкладних робіт – це усунення безпосередньої загрози життю та здоров’ю людей, відновлення життєзабезпечення населення, запобігання або значні зменшення матеріальних збитків. Рятувальні та інші невідкладні роботи включають також усунення пошкоджень, які заважають проведенню рятувальних робіт, створення умов для наступного проведення відновлювальних робіт. РіНР поділяють на рятувальні роботи та невідкладні роботи [25-27].

До рятувальних робіт відносять:

— розвідка маршруту руху сил, визначення обсягу та ступеню руйнувань, розмірів зон зараження, швидкості і напрямку розповсюдження зараженої хмари чи пожежі;

— локалізація та гасіння пожеж на маршруті руху сил та ділянках робіт;

— визначення об’єктів і населених пунктів, яким безпосередньо загрожує небезпека;

— визначення потрібного угрупування сил і засобів запобігання і локалізації небезпеки;

— пошук уражених та звільнення їх з-під завалів, пошкоджених та палаючих будинків, із загазованих та задимлених приміщень;

— розкриття завалених захисних споруд та рятування з них людей;

— вивіз або вивід населення із небезпечних місць у безпечні райони;

— організація комендантської служби, охорона матеріальних цінностей і громадського порядку;

— відновлення життєздатності населених пунктів та об’єктів;

— санітарна обробка уражених;

— знезараження одягу, взуття, засобів індивідуального захисту, територій, споруд, а також техніки;

— соціально-психологічна реабілітація населення.

До невідкладних робіт відносять:

— прокладання колонних шляхів та улаштування проїздів (проходів) у завалах та на зараженій території;

— локалізація аварій на водопровідних, енергетичних, газових і технологічних мережах;

— ремонт та тимчасове відновлення роботи комунально-енергетичних систем та мереж зв’язку для забезпечення рятувальних робіт;

— зміцнення або руйнування конструкцій, які загрожують обвалом і безпечному веденню робіт;

РіНР здійснюють у три етапи. На першому етапі вирішуються завдання:

— щодо екстреного захисту населення;

— з запобігання розвитку чи зменшення впливу наслідків;

— з підготовки до виконання РіНР.

Основними заходами щодо захисту населення є:

— оповіщення про небезпеку;

— використання засобів захисту;

— додержання режимів поведінки;

— евакуація з небезпечних у безпечні райони;

— здійснення санітарно-гігієнічної, протиепідемічної профілактики і надання медичної допомоги;

— локалізація аварій;

— зупинка чи заміна технологічного процесу виробництва;

— попередження (запобігання) і гасіння пожеж.

На другому етапі проводять:

— пошук потерпілих;

— витягання потерпілих з-під завалів, з палаючих будинків, пошкоджених транспортних засобів;

— евакуація людей із зони лиха, аварії, осередку ураження;

— надання медичної допомоги;

— санітарна обробка людей;

— знезараження одягу, майна, техніки, території;

— проведення інших невідкладних робіт, що сприяють і забезпечують здійснення рятувальних робіт;

— надання потерпілим першої допомоги та евакуація їх (при необхідності) у лікувальні заходи.

На третьому етапі вирішуються завдання щодо забезпечення життєдіяльності населення у районах, які потерпіли від наслідків НС6

— відновлення чи будівництво житла;

— відновлення енерго-, тепло-, водо- та газопостачання, ліній зв’язку;

— організація медичного обслуговування;

— забезпечення продовольством і предметами першої необхідності;

— знезараження харчів, води, фуражу, техніки, майна, територій;

— соціально-психологічна реабілітація;

— відшкодування збитків;

Відновлювальні роботи ЦЗ не виконує, їх здійснює спеціально створені підрозділи (бригади). Залежно від рівня надзвичайної ситуації (загальнодержавного, регіонального, місцевого, чи об’єктового) для проведення

РіНР залучають сили та засоби ЦЗ центрального, регіонального або об’єктового підпорядкування.

3.2 Маркетингова діяльність на підприємстві

Маркетингова діяльність на підприємстві охоплює не лише стратегії та методи просування продукції, але й дотримання правил охорони праці та законодавчих норм. Розглядаючи маркетинг з точки зору охорони праці, можна виділити наступні аспекти [28, 29]:

1) Робоче місце та умови праці: Закон України «Про рекламу» та інші нормативні документи не встановлюють прямих вимог до фізичного середовища праці маркетологів, однак загальні норми охорони праці вимагають, щоб робочі місця були безпечними і не шкодили здоров'ю працівників. Важливо забезпечити належне ергономічне обладнання, достатнє освітлення та оптимальні кліматичні умови [30].

2) Психологічна безпека.

Маркетингова діяльність часто супроводжується високим рівнем стресу через терміни виконання завдань та велику кількість комунікацій. Підприємству необхідно вживати заходів для зменшення стресу, надаючи підтримку і тренінги з управління стресом, щоб сприяти психологічному благополуччю своїх працівників.

3) Безпека при використанні технологій: Закони про телебачення, радіомовлення та використання електрозв'язку містять положення про безпеку інформаційних технологій. Маркетологи, які використовують різноманітне програмне забезпечення для створення та розповсюдження реклами, повинні знати основи кібербезпеки і забезпечувати захист даних [31].

4) Соціальна відповідальність: Під час розробки рекламних кампаній важливо дотримуватися вимог Закону України «Про захист суспільної моралі» і

стандартів «Недискримінаційна реклама за ознакою статі», що сприяє створенню недискримінаційного та рівного середовища для всіх працівників і клієнтів.

5) Навчання та інструктажі з охорони праці: Дотримання рекламних стандартів та правил охорони праці вимагає регулярного навчання працівників. Підприємства повинні організовувати тренінги і семінари, які допомагають працівникам ознайомитися з актуальними законодавчими змінами та методами роботи в безпечних умовах.

Забезпечення відповідності цим аспектам допомагає створити безпечне робоче середовище, зменшує ризики професійних захворювань і забезпечує високу продуктивність праці на підприємстві.

Правове регулювання рекламної діяльності є невід'ємною частиною економічної системи суспільства. Діяльність рекламодавців регулюється широким спектром законодавчих і нормативних актів, що регламентують методи рекламування й продажів товарів.

Основою правового регулювання національної реклами є Закон України «Про рекламу», а світової – «Міжнародний кодекс реклами»

Закон України «Про рекламу» визначає:

— національні норми і сферу їх застосування;

— авторське право;

— мову;

— принципи;

— вимоги;

— ідентифікацію;

— заборону недобросовісної реклами;

— порівнянність і соціальність реклами;

— порядок висвітлення реклами на телебаченні і радіо, у друкованих засобах масової інформації, з використанням електрозв'язку; розміщення;

— реклама і діти.

Спеціальні розділи закону передбачають:

— особливості рекламування деяких видів товарів;

— контроль за дотриманням законодавства про рекламу і відповідальність за його порушення.

Міжнародний Кодекс встановлює такі норми:

— благопристойність;

— чесність;

— правдивість;

— порівняння;

— доказ і свідчення;

— захист прав особистості;

— використання доброго ім'я (репутації);

— імітація; ототожнення рекламного послання;

— безпека; діти і молодь;

— відповідальність.

Крім того законодавство про маркетингову діяльність в деякій мірі відображено в інших законах України:

— «Про захист прав споживачів»;

— «Про заходи щодо попередження та зменшення вживання тютюнових виробів і їх шкідливого впливу на здоров'я населення»;

— «Про захист суспільної моралі»;

— «Про охорону прав на знаки для товарів і послуг»;

— «Про захист від недобросовісної конкуренції»;

— «Про забезпечення функціонування української мови як державної»;

— «Про телебачення та радіомовлення»;

— «Про друковані засоби масової інформації (пресу) в Україні».

Урядом України затверджені:

— «Типові правила розміщення зовнішньої реклами»,

— «Порядок накладення штрафів за порушення законодавства про рекламу» та інші.

У маркетинговій діяльності підприємства слід враховувати Стандарт організацій України «Недискримінаційна реклама за ознакою статі» СОУ 21708654-002-2011 та інші стандарти, а також накази, інструкції, тарифи, правила, інструктивні листи органів державної влади України і органів місцевого самоврядування [32].

ВИСНОВКИ

Це дослідження мало на меті проаналізувати та впровадити ідемпотентні ключі у платіжні системи, зокрема в контексті дропшипінгу. Ідемпотентність гарантує, що багаторазове виконання операції призводить до одного й того самого результату, що є критично важливим для операцій, які вимагають високої точності, таких як платежі.

В першому розділі було розглянуто теоретичні основи ідемпотентності в системах обробки платежів. Було визначено концепцію ідемпотентності та її значення для платіжних систем. Розглянуто історичний контекст розвитку цієї концепції та її впровадження у сучасних платіжних системах. Також було проаналізовано методики використання ідемпотентних ключів в PayPal та Stripe, включаючи процес генерації, інтеграцію ключів у API платіжних систем та рішення по обробці ключів.

Другий розділ присвячений архітектурі ідемпотентної системи та розробці стратегії управління платежами для проєкту дропшипінгу. Було розглянуто структуру ідемпотентних ключів, їх технічні характеристики, стандарти та протоколи використання. Також було описано механізм роботи ключів, методи зберігання ідемпотентних ключів у Redis та базах даних, а також переваги використання Redis для кешування ключів. Завершальний підрозділ включає опис структури проєкту, основних компонентів та реалізованого проєкту.

Значення ідемпотентності в платіжних системах полягає у забезпеченні надійності та безпеки транзакцій. Ідемпотентність запобігає дублюванню операцій при повторних запитах, що є критично важливим для систем дропшипінгу, де можуть виникати технічні збої або проблеми з мережею. Використання ідемпотентних ключів дозволяє уникнути дублювання операцій, підвищує довіру клієнтів та забезпечує стабільність роботи системи.

Процес генерації і обробки ідемпотентних ключів включає створення унікальних ключів, їх включення у заголовки HTTP-запитів та обробку на сервері.

Використання стандартів та протоколів забезпечує узгодженість та надійність цих операцій. Ідемпотентні ключі генеруються на стороні клієнта або автоматично системою, а їх унікальність забезпечується використанням UUID. Ключі зберігаються у базі даних або кеші протягом обмеженого часу для запобігання накопиченню великої кількості даних.

Впровадження ідемпотентних ключів у системи дропшипінгу забезпечує надійність та безпеку транзакцій, запобігаючи дублюванню операцій та забезпечуючи стабільність роботи сервера навіть у випадку технічних збоїв або повторних запитів.

Впровадження ідемпотентних ключів у платіжні системи вимагає врахування кількох важливих практичних аспектів для забезпечення надійної та ефективної роботи. Одним з ефективних способів реалізації ідемпотентності є використання middleware. Middleware дозволяє обробляти ідемпотентні ключі на рівні запитів до сервера, забезпечуючи централізовану обробку без необхідності дублювати код у кожному маршруті. Це дозволяє значно спростити процес інтеграції ідемпотентних ключів у систему та забезпечити єдині підходи до обробки запитів.

Для зберігання ідемпотентних ключів та відповідей на запити у виробничих середовищах використання Redis є рекомендованою практикою. Redis забезпечує спільний доступ до даних з усіх серверів у кластері, дозволяє зберігати ключі навіть після перезапуску серверів і забезпечує високу швидкість доступу до даних. Це значно покращує продуктивність системи та забезпечує надійне зберігання ідемпотентних ключів.

Впровадження ідемпотентних ключів може супроводжуватись певними ризиками та викликами, які потрібно враховувати. Однією з потенційних проблем є використання одного ідемпотентного ключа різними клієнтами, що може призвести до витоку конфіденційних даних. Крім того, зберігання великої кількості ідемпотентних ключів може призвести до проблем з продуктивністю та використанням пам'яті. Щоб уникнути цих проблем, слід забезпечити унікальність ідемпотентних ключів та встановити обмеження на термін їх дії. Використання

UUID для генерації ключів дозволяє мінімізувати ризик збігу, а обмеження терміну дії запобігає накопиченню великої кількості даних.

Майбутні дослідження можуть зосереджуватися на оптимізації алгоритмів генерації та зберігання ідемпотентних ключів, а також на розробці нових методів забезпечення безпеки. Одним із можливих напрямків є розробка більш ефективних методів виявлення та обробки дубльованих запитів, а також дослідження питання масштабування систем з ідемпотентними ключами для забезпечення їх ефективної роботи у великих розподілених середовищах.

Покращення існуючих методів включає оптимізацію процесу зберігання ключів для зменшення використання ресурсів, а також впровадження більш надійних заходів безпеки для захисту від можливих атак. Інтеграція ідемпотентності з іншими протоколами та стандартами також може сприяти підвищенню надійності та ефективності платіжних систем. Ці рекомендації допоможуть у подальшому розвитку та вдосконаленні систем обробки платежів з використанням ідемпотентних ключів, що забезпечить їх стабільну та безпечну роботу в умовах швидкозмінного технологічного середовища.

Дослідження, присвячене ідемпотентності в платіжних системах, підкреслює її важливість для забезпечення надійності та безпеки транзакцій. Ідемпотентність відіграє ключову роль у запобіганні дублюванню операцій, що є критичним для систем дропшипінгу та інших платформ електронної комерції. Завдяки ідемпотентним ключам, платіжні системи стають більш стійкими до технічних збоїв і проблем з мережею, що підвищує довіру та задоволеність користувачів.

Впровадження ідемпотентності сприяє розвитку платіжних систем, дозволяючи їм ефективно обробляти велику кількість транзакцій без ризику дублювання. Це особливо важливо в сучасному світі електронної комерції, де надійність та безпека фінансових операцій є вирішальними факторами успішного функціонування. Загалом, ідемпотентність забезпечує стабільність роботи платіжних систем, сприяючи їх подальшому розвитку та адаптації до потреб сучасного ринку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1) Stripe. Idempotent Requests. [Електронний ресурс] – Режим доступу до ресурсу: https://stripe.com/docs/api/idempotent_requests.

2) PayPal. Idempotency in Payment APIs. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.paypal.com/docs/api-basics/notifications/idempotency/.

3) Redis. Redis Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/documentation.

4) Mozilla Developer Network (MDN). HTTP Headers. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

5) Fowler, M. (2010). Patterns of Enterprise Application Architecture. – Addison-Wesley Professional, 2010 p. c. 560.

6) Tilkov, S., & Vinoski, S. (2010). REST: Advanced Research Topics and Practical Applications. – Addison-Wesley Professional, 2010 p. c. 352.

7) Campbell, C., & Loy, J. (2019). Designing and Building Scalable Payment Systems. – O'Reilly Media, 2019 p. c. 360.

8) Bashir, I. (2017). Mastering Blockchain: Unlocking the Power of Cryptocurrencies and Distributed Ledgers. – Packt Publishing, 2017 p. c. 540.

9) Microsoft Azure. Design Patterns for Payment Systems. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/architecture/patterns/payment-system-design.

10) Banerjee, A. (2020). Building Secure and Reliable Systems. – O'Reilly Media, 2020 p. c. 608.

11) Richardson, C., & Smith, F. (2019). Microservices Patterns: With Examples in Java. – Manning Publications, 2019 p. c. 520.

12) Shoup, R. (2018). Practical Cloud Security: A Guide for Secure Design and Deployment. – O'Reilly Media, 2018 p. c. 416.

13) NGINX. Building Idempotent APIs with NGINX. [Електронний ресурс] – Режим доступу до ресурсу: https://www.nginx.com/resources/webinars/building-idempotent-apis-nginx/.

14) Іванов В.М. Програмування та адміністрування баз даних – м. Київ, 2020 р. с. 324.

15) Ковальчук Т.А., Мельник О.П. Сучасні стратегії маркетингу – м. Київ, 2020 р. с. 284.

16) Жук І.В., Петренко В.Г. Цифровий маркетинг: новітні технології та інструменти – м. Львів, 2021 р. с. 312.

17) Бондаренко О.Ю., Герасимчук В.В. Маркетинг у соціальних медіа – м. Одеса, 2022 р. с. 298.

18) Васильєва Т.С., Костенко Н.П. Маркетингові комунікації в сучасному бізнесі – м. Харків, 2023 р. с. 256.

19) Руденко М.Л., Литвиненко І.І. Маркетинговий аналіз: методи та практика – м. Дніпро, 2024 р. с. 275.

20) Кодекс цивільного захисту України [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/5403-17#Text

21) Про затвердження Положення про підсистему реагування на надзвичайні ситуації, проведення аварійно-рятувальних та інших невідкладних робіт єдиної державної системи цивільного захисту : наказ Міністерства внутрішніх справ України від 04.05.2016 №356.

22) Про затвердження Положення про штаб з ліквідації наслідків надзвичайної ситуації та видів оперативно-технічної і звітної документації штабу з ліквідації наслідків надзвичайної ситуації : наказ Міністерства внутрішніх справ України від 26.12.2014 №1406.

23) Про організацію роботи штабу з ліквідації наслідків надзвичайної ситуації та забезпечення його готовності : наказ Державної служби України з надзвичайних ситуацій від 16.03.2015 №149.

24) Рекомендації для роботодавців щодо організації виконання робіт підвищеної небезпеки під час воєнних (бойових) дій [Електронний ресурс] – Режим

доступу: https://dsp.gov.ua/faq/rekomendatsii-dlia-robotodavtsiv-shchodo-orhanizatsii-vykonannia-robit-pidvyshchenoi-nebezpeky-pid-chas-voiennykh-boiovykh-dii/

25) Аветисян В.Г., Тригуб В.В., Грицина І.М., Остапов К.М. Організації аварійнорятувальних робіт : курс лекцій – Х: НУЦЗУ, 2017. – 141 с.

26) Організація аварійно-рятувальних робіт : навчальний посібник /Р.Т. Ратушний, В.Б. Лоїк, О.Д. Синельніков, В.М. Ковальчук – Львів: Видавництво ЛДУ БЖД, 2020. – 394 с.

27) Методичні рекомендації щодо розроблення планів локалізації і ліквідації аварій та їх наслідків : наказ Державної служби України з надзвичайних ситуацій від 17 травня 2022 року за №253

28) Ілляшенко С. М. Маркетингова діяльність на підприємстві // Економіка підприємства: підручник / за заг. ред. д. е. н., проф. Л.Г. Мельника. Суми: Університетська книга, 2012. С. 528-559.

29) Петруня Ю. Є., Петруня В. Ю. Маркетинг: навч. Посібник.Дніпропетровськ: Університет митної справи та фінансів, 2016. 362 с.

30) Закон України «Про рекламу» №1121-IV від 11.07.2003 р. (Нова ред. із змінами, внесеними згідно із Законом №270/96-ВР від 27.04.2024 р.: [Електронний ресурс]. – Режим доступу : https://zakon.rada.gov.ua/laws/show/1121-15#Text

31) Закон України «Про електронні комунікації» № 1089-IX від 01.01.2024 р. [Електронний ресурс]. – Режим доступу : https://zakon.rada.gov.ua/laws/show/1089-20#Text

32) Стандарт організацій України «Недискримінаційна реклама за ознакою статі» (СОУ 21708654-002-2011)-К., «ДП УкрДНЦ» 2011. – 31 с.

ДОДАТКИ

VII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 621.326
Поліщук Р. – –ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

СУЧАСНІ СИСТЕМИ ОПЛАТ З ВИКОРИСТАННЯМ КЛЮЧІВ ІНДПОТЕНТНОСТІ

Науковий керівник: маг. Заярний М.А.

Polishchuk R.

Ternopil Ivan Puluj National TechnicalUniversity

MODERN PAYMENT SYSTEMS USING IMPOWERMENT KEYS

Supervisor: m, Zayarnyi M.

Ключові слова: Система оплат, ключі ідемпотентності веб-застосунок.
Key words: Payment system, idempotence keys of web applications.

Людство динамічно розвивається у сфері цифрових платежів, де забезпечення надійності та безпечності трансакцій є ключовими факторами. Одним з сучасних підходів до підвищення ефективності та надійності систем оплат є використання ключів ідемпотентності. Ідемпотентність — це властивість операцій, яка гарантує, що повторне виконання операції не змінює результат після першого виконання. В контексті веб-застосунків, таке рішення дозволяє уникнути подвійних списань або зарахувань коштів на рахунок користувача у випадку повторних запитів.

Застосування ключів ідемпотентності в платіжних системах дозволяє не тільки знизити ризики фінансових збитків для користувачів та підприємств, але й підвищити довіру до електронних платіжних сервісів. Ця технологія ефективно використовується у великих платіжних системах, таких як PayPal, Stripe, і банківських додатках, де потреба у забезпеченні високої доступності та надійності є критично важливою.

Інтеграція ідемпотентності у фінансові операції забезпечується через використання унікальних ідентифікаторів для кожного запиту, що відправляється до сервера. Це дозволяє системі визначити, чи була вже оброблена попередня трансакція з таким ідентифікатором, і уникнути її повторного виконання. В результаті, підвищується загальна прозорість та контроль за проведенням фінансових операцій, що є особливо важливим у сучасному цифровому світі.

Розглядаючи ключі ідемпотентності в контексті сучасних платіжних систем, можна виділити кілька важливих аспектів, які підкреслюють їх значення та потенційні напрямки розвитку: збільшення надійності мікросервісних архітектур, підвищення резистентності до помилок, оптимізація взаємодії з API, покращення користувацького досвіду,

регуляторні та правові аспекти, дослідження та інновації. Використання ключів ідемпотентності у платіжних системах є одним із напрямків, який має значний потенціал для розширення та вдосконалення в контексті швидкого розвитку цифрових технологій.

В сучасних обчислювальних середовищах, де використовуються мікросервіси, ідемпотентність допомагає забезпечувати стабільність і надійність розподілених систем. Це особливо актуально для фінансових інституцій, які реалізують складні транзакційні системи з високими вимогами до безпеки і точності обробки даних.

Використання ідемпотентних ключів може значно знизити вплив збоїв або помилок у мережі. Якщо трансакція не завершується успішно через технічні збої, система може безпечно повторити запит, знаючи, що це не призведе до подвійного списання коштів або інших непередбачених ефектів. Веб-сервіси часто використовують ідемпотентні ключі при дизайні своїх API, щоб запобігти непередбаченим наслідкам повторних запитів. Це особливо важливо для API, які управляють платежами, записами користувачів та іншими критичними даними.

Проблематика ідемпотентності також актуальна у контексті розвитку хмарних рішень та сервісів, де забезпечення стабільності та надійності обробки великих обсягів даних є ключовим аспектом. Розробка та оптимізація механізмів ідемпотентності можуть значно вплинути на ефективність бізнес-процесів, зменшуючи час та ресурси, необхідні для вирішення конфліктних ситуацій, пов'язаних з обробкою платежів.

Таким чином, використання ключів ідемпотентності у сучасних системах оплат стає все більш поширеним та необхідним рішенням для забезпечення високого рівня безпеки та ефективності обробки трансакцій у цифрову епоху.

Література:

1. Kumar, A., & Sharma, V. (2019). "Ensuring data integrity using idempotence in distributed financial transactions." International Journal of Financial Studies.
2. Lee, P. (2020). "Advanced API design: Ensuring idempotent behavior in RESTful services." Procedia Computer Science.
3. O'Neill, M. (2021). "Practical approaches to idempotency in banking applications." Journal of Financial Innovation.
4. Peterson, K. & Brown, L. (2017). "Idempotence patterns in software design." Software: Practice and Experience.
5. Zhang, X., & Li, Y. (2018). "Design and implementation of idempotent mechanisms in cloud-based microservice architectures." IEEE Transactions on Cloud Computing.

Лістинг коду Б.1 - Серверна частина: Node.js з Express та Redis

```
const express = require('express');
const bodyParser = require('body-parser');
const Redis = require('ioredis');
const { v4: uuidv4 } = require('uuid');
// Ініціалізація Redis
const redis = new Redis({
  host: 'localhost',
  port: 6379,
  db: 0
});
const app = express();
app.use(bodyParser.json());
// Псевдо-функція для обробки платежу
async function processPayment(paymentData) {
  // Імітація обробки платежу
  return {
    success: true,
    paymentId: uuidv4(),
    amount: paymentData.amount,
    currency: paymentData.currency,
    description: paymentData.description
  };
}
// Обробка запиту на оплату
app.post('/api/payments', async (req, res) => {
  const idempotencyKey = req.headers['idempotency-key'];
  const paymentData = req.body;
  if (!idempotencyKey) {
    return res.status(400).json({ error: 'Idempotency key is
required' });
  }
  // Перевірка наявності ідемпотентного ключа в Redis
  const exists = await redis.exists(idempotencyKey);
  if (exists) {
    // Ключ існує, повертаємо попередній результат
    const cachedResult = await redis.get(idempotencyKey);
    return res.json(JSON.parse(cachedResult));
  } else {
    // Ключ не існує, обробляємо платіж
    try {
      const paymentResult = await processPayment(paymentData);
      // Збереження результату в Redis з TTL (наприклад, 1 година)
      await redis.set(idempotencyKey, JSON.stringify(paymentResult),
'EX', 3600);
      return res.json(paymentResult);
    } catch (error) {
      return res.status(500).json({ error: 'Payment processing
failed' });
    }
```

```
  }
});
// Запуск сервера
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Лістинг коду Б.2 - Клієнтська частина: Запит з використанням Axios

```
const axios = require('axios');
const { v4: uuidv4 } = require('uuid');
// Генерація ідемпотентного ключа
const idempotencyKey = uuidv4();
// Дані платежу
const paymentData = {
  amount: 100,
  currency: 'USD',
  source: 'card_1JY7k2Hhbgq3Hvs1VnW1Yps1',
  description: 'Payment for order #12345'
};
// Відправка запиту на сервер
axios.post('http://localhost:3000/api/payments', paymentData, {
  headers: {
    'Idempotency-Key': idempotencyKey
  }
})
.then(response => {
  console.log('Payment processed:', response.data);
})
.catch(error => {
  console.error('Error processing payment:', error);
});
```

Лістинг коду Б.3 - Налаштування запиту в Postman - Body (JSON)

```
{
  "amount": 100,
  "currency": "USD",
  "source": "card_1JY7k2Hhbgq3Hvs1VnW1Yps1",
  "description": "Payment for order #12345"
}
```

Лістинг коду Б.4 - SQL скрипт для створення таблиць

```
CREATE DATABASE DropshippingStoreDB;

USE DropshippingStoreDB;

-- Таблиця для зберігання ідемпотентних ключів

CREATE TABLE IdempotencyKeys (

    KeyID INT AUTO_INCREMENT PRIMARY KEY,

    KeyType VARCHAR(50) NOT NULL,
```

```
    KeyValue VARCHAR(100) NOT NULL UNIQUE,
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    Status VARCHAR(20) DEFAULT 'PENDING'
);
-- Таблиця для зберігання інформації про замовлення
CREATE TABLE Orders (
    OrderID INT AUTO_INCREMENT PRIMARY KEY,
    OrderKey VARCHAR(100) NOT NULL,
    CustomerID INT NOT NULL,
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    Status VARCHAR(20) DEFAULT 'PENDING',
    FOREIGN KEY (OrderKey) REFERENCES IdempotencyKeys(KeyValue)
);
-- Таблиця для зберігання інформації про сесії
CREATE TABLE Sessions (
    SessionID INT AUTO_INCREMENT PRIMARY KEY,
    SessionKey VARCHAR(100) NOT NULL,
    UserID INT NOT NULL,
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    Status VARCHAR(20) DEFAULT 'ACTIVE',
    FOREIGN KEY (SessionKey) REFERENCES IdempotencyKeys(KeyValue)
);
-- Таблиця для зберігання інформації про повернення
CREATE TABLE Refunds (
    RefundID INT AUTO_INCREMENT PRIMARY KEY,
    RefundKey VARCHAR(100) NOT NULL,
    OrderID INT NOT NULL,
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    Status VARCHAR(20) DEFAULT 'PENDING',
    FOREIGN KEY (RefundKey) REFERENCES IdempotencyKeys(KeyValue),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
-- Таблиця для зберігання інформації про клієнтів
CREATE TABLE Customers (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Address VARCHAR(255),
    Phone VARCHAR(20),
```

```
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);
-- Таблиця для зберігання інформації про користувачів
CREATE TABLE Users (
    UserID INT AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(50),
    Password VARCHAR(100),
    Role VARCHAR(50),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);
-- Таблиця для зберігання кешованих ідемпотентних ключів у Redis
(псевдокод для ілюстрації)
CREATE TABLE RedisCache (
    CacheID INT AUTO_INCREMENT PRIMARY KEY,
    KeyType VARCHAR(50),
    KeyValue VARCHAR(100) UNIQUE,
    Expiration DATETIME
);
-- Індекси для прискорення пошуку
CREATE INDEX idx_orderkey ON Orders (OrderKey);
CREATE INDEX idx_sessionkey ON Sessions (SessionKey);
CREATE INDEX idx_refundkey ON Refunds (RefundKey);
-- Тригери для автоматичного оновлення статусу ключів
DELIMITER //
CREATE TRIGGER trg_update_orderkey_status
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
    UPDATE IdempotencyKeys
    SET Status = 'CONFIRMED'
    WHERE KeyValue = NEW.OrderKey;
END //
CREATE TRIGGER trg_update_sessionkey_status
AFTER INSERT ON Sessions
FOR EACH ROW
BEGIN
    UPDATE IdempotencyKeys
    SET Status = 'ACTIVE'
    WHERE KeyValue = NEW.SessionKey;
```

```
END //
CREATE TRIGGER trg_update_refundkey_status
AFTER INSERT ON Refunds
FOR EACH ROW
BEGIN
    UPDATE IdempotencyKeys
    SET Status = 'PENDING'
    WHERE KeyValue = NEW.RefundKey;
END //
DELIMITER ;
```

Лістинг коду Б.5 - Лістинг для оновлення статусу замовлень

```
const express = require('express');
const bodyParser = require('body-parser');
const Redis = require('ioredis');
const { v4: uuidv4 } = require('uuid');
// Ініціалізація Redis
const redis = new Redis({
  host: 'localhost',
  port: 6379,
  db: 0
});
const app = express();
app.use(bodyParser.json());
// Псевдо-функція для оновлення статусу замовлення
async function updateOrderStatus(orderId, status) {
  // Імітація оновлення статусу замовлення
  return {
    success: true,
    orderId: orderId,
    status: status,
    updatedAt: new Date()
  };
}
// Обробка запиту на оновлення статусу замовлення
app.post('/api/orders/:orderId/status', async (req, res) => {
  const { orderId } = req.params;
  const { status } = req.body;
```

```
    try {
      const updateResult = await updateOrderStatus(orderId, status);
      // Оновлення статусу замовлення в Redis
      const orderKey = `order:${orderId}`;
      await redis.hset(orderKey, 'status', status, 'updatedAt', new
Date().toISOString());
      return res.json(updateResult);
    } catch (error) {
      return res.status(500).json({ error: 'Failed to update order
status' });
    }
  });
  // Запуск сервера
  const PORT = process.env.PORT || 3001;
  app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
  });
```

Лістинг коду Б.6 - Лістинг для роботи з користувачами

```
  const express = require('express');
  const bodyParser = require('body-parser');
  const bcrypt = require('bcrypt');
  const Redis = require('ioredis');
  const { v4: uuidv4 } = require('uuid');
  // Ініціалізація Redis
  const redis = new Redis({
    host: 'localhost',
    port: 6379,
    db: 0
  });
  const app = express();
  app.use(bodyParser.json());
  // Псевдо-функція для створення користувача
  async function createUser(userData) {
    const hashedPassword = await bcrypt.hash(userData.password, 10);
    const userId = uuidv4();
    return {
      userId: userId,
```

```
        username: userData.username,
        password: hashedPassword,
        role: userData.role,
        createdAt: new Date()
      };
    }
    // Обробка запиту на створення користувача
    app.post('/api/users', async (req, res) => {
      const userData = req.body;
      try {
        const user = await createUser(userData);
        // Збереження користувача в Redis
        const userKey = `user:${user.userId}`;
        await redis.hmset(userKey, 'username', user.username, 'password',
user.password, 'role', user.role, 'createdAt',
user.createdAt.toISOString());
        return res.json(user);
      } catch (error) {
        return res.status(500).json({ error: 'Failed to create user' });
      }
    });
    // Запуск сервера
    const PORT = process.env.PORT || 3002;
    app.listen(PORT, () => {
      console.log(`Server is running on port ${PORT}`);
    });
```

Лістинг коду Б.7 - Лістинг для генерації звітів про замовлення

```
    const express = require('express');
    const bodyParser = require('body-parser');
    const Redis = require('ioredis');
    const { v4: uuidv4 } = require('uuid');
    // Ініціалізація Redis
    const redis = new Redis({
      host: 'localhost',
      port: 6379,
      db: 0
    });
```

```
const app = express();
app.use(bodyParser.json());
// Псевдо-функція для генерації звіту про замовлення
async function generateOrderReport() {
  const orders = await redis.keys('order:*');
  const orderDetails = [];
  for (const orderKey of orders) {
    const order = await redis.hgetall(orderKey);
    orderDetails.push(order);
  }
  return orderDetails;
}
// Обробка запиту на генерацію звіту про замовлення
app.get('/api/reports/orders', async (req, res) => {
  try {
    const report = await generateOrderReport();
    return res.json(report);
  } catch (error) {
    return res.status(500).json({ error: 'Failed to generate order
report' });
  }
});
// Запуск сервера
const PORT = process.env.PORT || 3003;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Лістинг коду Б.7 - Лістинг для сесій користувачів

```
const express = require('express');
const bodyParser = require('body-parser');
const Redis = require('ioredis');
const { v4: uuidv4 } = require('uuid');
// Ініціалізація Redis
const redis = new Redis({
  host: 'localhost',
  port: 6379,
  db: 0
```

```
});
const app = express();
app.use(bodyParser.json());
// Псевдо-функція для створення сесії
async function createSession(userId) {
  const sessionId = uuidv4();
  return {
    sessionId: sessionId,
    userId: userId,
    createdAt: new Date(),
    status: 'ACTIVE'
  };
}
// Обробка запиту на створення сесії
app.post('/api/sessions', async (req, res) => {
  const { userId } = req.body;
  try {
    const session = await createSession(userId);
    // Збереження сесії в Redis
    const sessionKey = `session:${session.sessionId}`;
    await redis.hmset(sessionKey, 'userId', session.userId,
'createdAt', session.createdAt.toISOString(), 'status', session.status);
    return res.json(session);
  } catch (error) {
    return res.status(500).json({ error: 'Failed to create session'
});
  }
});
// Запуск сервера
const PORT = process.env.PORT || 3004;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Додаток В – Диск із кваліфікаційною роботою бакалавра