

ПВНЗ «ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ»

КАФЕДРА Кафедра комп'ютерних наук та програмної інженерії

СПЕЦІАЛЬНІСТЬ 121 Інженерія програмного забезпечення

(шифр та назва)

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА Програмна інженерія

(назва)

**ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА
РОБОТА**

на тему:

Мультиплатформенний інструмент для інтернет-магазину

(назва затвердженої теми)

(на прикладі _____)

Студент (ка) _____
(підпис студента)

Фокін Владислав Сергійович
(прізвище, ім'я по-батькові студента)

Науковий
керівник _____
(підпис наукового керівника)

Ващенко М.В.
(прізвище, ініціали, наукове звання і ступінь)

Допущено до захисту “ _____ ” 2025 р.

Завідувач кафедри _____
(підпис)

(прізвище, ініціали, наукове звання і ступінь)

Декан факультету _____
(підпис)

Яровий Р.О.
(прізвище, ініціали)

Завідувач
навчальної частини _____
(підпис)

(прізвище, ініціали)

Київ – 2025

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- JWT – JSON Web Tokens
- ПЗ – програмне забезпечення
- БД – база даних
- PCI DSS – Payment Card Industry Data Security Standard

ТЕМА

«Мультиплатформенний інструмент для інтернет-магазину: методи та технології розробки платіжного Node.js-пакету для інтеграції з популярними e-commerce платформами»

ВСТУП

Актуальність теми

Сучасний розвиток електронної комерції кардинально змінює підходи до організації бізнес-процесів у сфері продажів. Інтернет-магазини стали невід'ємною частиною глобального ринку, що ставить перед ними нові вимоги щодо гнучкості, швидкості та безпеки обробки платежів. У цьому контексті особливо важливою є розробка мультиплатформенних рішень, здатних забезпечити інтеграцію з різними e-commerce платформами. Такий підхід дозволяє оптимізувати робочі процеси, знизити витрати на розробку індивідуальних рішень для кожної платформи та забезпечити єдиний стандарт обробки транзакцій.

Інтеграція українських платіжних систем у глобальні e-commerce платформи має особливе значення з кількох причин. По-перше, активне впровадження українських платіжних рішень сприяє розвитку національного фінтех-сектору, дозволяючи українським компаніям конкурувати на світовому ринку завдяки адаптації технологій до специфічних потреб місцевого ринку. По-друге, інтеграція українських сервісів дає змогу забезпечити більшу автономію у здійсненні фінансових операцій, що є важливим фактором з огляду на потребу у зменшенні залежності від зарубіжних сервісів. Крім того, локальні платіжні системи краще відповідають вимогам українського законодавства та стандартам безпеки, що є суттєвим для довіри як бізнесу, так і кінцевих споживачів.

Безумовно, частина цих глобальних e-commerce платформ вже мають власні рішення для впровадження оплати через українські сервіси. Втім, всі

вони мають різні реалізації однієї і тієї ж самої проблеми, що викликає потребу залишати в різних місцях конфіденційні дані. Деякі глобальні платформи, наприклад Shopify, не мають взагалі жодних інтегрованих рішень на платформі (як і жодних додатків) які б дозволили інтегрувати такі платіжні системи як Monobank або Wayforpay без стороннього розробника.

Саме тому потрібне особливе рішення для генерації сторінок оплат користувачів, яке б:

1. зберігало конфіденційні дані про платіжний сервіс в одній базі даних,
2. мало легке формулювання інформації про оплату,
3. було легке в імплементації на будь якій платформі.

Мета і завдання дослідження

Розробка Node.js-пакетів для генерації сторінок оплати та мультиплатформного платіжного інструменту для інтернет-магазинів, який буде інтегрувати ці пакети. Реалізація прототипу такого рішення дозволить оптимізувати процес обробки транзакцій, підвищити рівень безпеки фінансових операцій та спростити інтеграцію з популярними e-commerce платформами. Припускається, що прототип буде включати в себе тільки інтеграцію оплати для Shopify магазинів, але буде створений достатньо масштабованим, щоб була можливість швидко додати інтеграцію з іншими платформами.

Аналіз ринку та існуючих рішень

Провести дослідження сучасних технологій та платформ, що використовуються для інтеграції платіжних систем (Monobank Pay, LiqPay, Wayforpay).

Виявити основні переваги та недоліки існуючих рішень, зокрема з огляду на вимоги безпеки, масштабованості та зручності інтеграції.

Визначення вимог до системи

Сформуувати функціональні та нефункціональні вимоги до розроблюваних Node.js-пакетів. Розглянути специфіку інтеграції українських платіжних систем, враховуючи нормативні та технічні аспекти, а також вимоги до відповідності стандартам безпеки (наприклад, PCI DSS).

Проектування архітектури платіжного пакету

Розробити концептуальну модель та архітектурне рішення, що забезпечують модульність, розширюваність та гнучкість інтеграції з різними платформами.

Обрати сучасні технології (Node.js, TypeScript, інструменти для тестування, CI/CD), що сприятимуть реалізації високопродуктивного та надійного рішення.

Розробка функціонального прототипу

Реалізувати основні компоненти платіжного пакету: ініціалізацію платіжного сервісу, роутинг запитів (створення, підтвердження платежів, повернення коштів), обробку webhooks та механізми відновлення після збоїв.

Забезпечити механізми безпеки (OAuth, JWT, HMAC, TLS/SSL) для захисту фінансових операцій та даних користувачів.

Тестування та валідація прототипу

Провести модульне та інтеграційне тестування з використанням Mock API для симуляції платіжних запитів. Оцінити коректність роботи системи в різних сценаріях (як позитивних, так і негативних), а також забезпечити відповідність прототипу вимогам до безперебійності та масштабованості.

Розробка методичних рекомендацій щодо інтеграції

Створити рекомендації та приклади інтеграції розробленого платіжного пакету для Shopify. Надати інструкції щодо налаштування, розгортання та оптимізації роботи системи.

Об'єкт дослідження

Процес інтеграції платіжних рішень у мультиплатформенний інструмент для інтернет-магазинів. Це включає в себе аналіз підходів до об'єднання різних платіжних систем в єдиний інтерфейс, оптимізацію робочих процесів з обробки транзакцій, а також забезпечення безпеки та масштабованості реалізації перенаправлення клієнтів на сторінки де вони зможуть здійснити безпечні платіжні операції. Особливу увагу приділено інтеграції українських платіжних систем із глобальними e-commerce платформами, що сприяє розвитку національного фінтех-сектору та дозволяє бізнесу використовувати локальні технологічні переваги в умовах зростаючої конкуренції на міжнародному ринку.

Предмет дослідження

Предметом дослідження є методи та технології розробки Node.js-пакетів, які виступають у ролі гнучкого платіжного шлюзу для інтеграції з різними e-commerce платформами. Це стосується конкретних підходів до проектування архітектури рішення, використання інструментів Node.js, TypeScript, CI/CD, тестування з використанням Mock API, та впровадження ефективних механізмів обробки транзакцій, автентифікації користувачів та управління помилками. Розглядаються як загальні принципи розробки програмного забезпечення (ПЗ), так і специфічні аспекти інтеграції українських платіжних систем, зокрема забезпечення відповідності стандартам безпеки (наприклад, PCI DSS) та адаптації до вимог локального ринку.

Дослідження зосереджено на розробці зручного для використання ПЗ, яке дозволить спростити інтеграцію платіжних систем у мультиплатформенні інструменти для інтернет-магазинів, забезпечуючи високий рівень продуктивності, безпеки та зручності експлуатації.

РОЗДІЛ I

ПРОБЛЕМАТИКА ТА ОГЛЯД МЕТОДІВ РОЗРОБКИ ПЛАТІЖНИХ РІШЕНЬ ДЛЯ E-COMMERCE

1.1. Аналіз поточних платформ

У сучасному світі електронної комерції платіжні системи виступають ключовим елементом успішної взаємодії між інтернет-магазинами та їх клієнтами. Широкий вибір готових рішень та платформ дозволяє бізнесу обирати оптимальні за функціональністю, вартістю та умовами інтеграції рішення для обробки транзакцій.

Огляд популярних українських платіжних сервісів

LiqPay – українське рішення для електронних платежів, яке було розроблене фахівцями ПриватБанку з урахуванням унікальних особливостей української фінансової системи. Завдяки глибокій інтеграції з банківською інфраструктурою країни, сервіс дозволяє підприємцям обробляти оплату в гривні максимально швидко та без зайвих затримок. Інтерфейс LiqPay підтримує роботу з різноманітними каналами – від вебсайтів і мобільних додатків до чат-ботів у месенджерах – втім обмежен для використання наприклад в Shopify.

Система обробки платежів у реальному часі доволі швидка (callback-ки приходять майже миттєво), через що продавці отримують миттєві підтвердження транзакцій і можуть оперативно виконувати замовлення клієнтів.

Перевага LiqPay – тарифні плани (всього 1.5%), розраховані на потреби малого та середнього бізнесу, які дозволяють оптимізувати витрати на платіжні сервіси та знижують бар'єр входу для нових продавців. Крім того, сервіс підтримує гнучкі механізми налаштування платежів: можна як створювати одноразові лінки для переказу коштів, так і налаштовувати рекурентні списання за підписками.

Втім платформа стикається з тривалим процесом верифікації нових мерчантів, що затягує підключення до платформи й відстроє початки прийому платежів, що є мінусом платформи. До того ж архітектурні рішення, які лежать в основі створення платежів, дещо застаріли та ускладнюють інтеграцію з сучасними фреймворками, через що при створенні інтеграції доведеться витратити додатковий час на створення власних модулів.

Переваги:

1. Глибока інтеграція з банківською інфраструктурою ПриватБанку
2. Швидка обробка платежів у реальному часі (callback-ки майже миттєво)
3. Низький тариф — 1,5 % для малого та середнього бізнесу
4. Гнучкі механізми оплати: одноразові платіжні лінки та рекурентні списання
5. Підтримка веб-сайтів, мобільних додатків і чат-ботів

Недоліки:

1. Тривалий процес верифікації нових мерчантів
2. Обмежена готова підтримка інтеграції з платформами на кшталт Shopify
3. Застаріла архітектура API, через що інтеграція з сучасними фреймворками ускладнена
4. Необхідність власної розробки модулів для повноцінного підключення та налаштування

З огляду на зазначене, LiqPay — привабливий варіант для компаній на українському ринку й шукають надійну, але водночас вигідну платіжну систему, однак тим, хто прагне максимально швидкого старту без довгих перевірок і з мінімальними технічними труднощами, варто розглянути Wayforpay.

WayForPay теж створена з урахуванням потреб українського ринку, тому її інфраструктура і функціонал теж гарно орієнтовані на обробку платежів українського бізнесу.

Процедура верифікації у Wayforpay максимально спрощена: після подачі необхідних документів підключення платіжних каналів займає лічені години, що забезпечує швидкий старт для будь-якого підприємця. Одразу ж після підтвердження облікового запису можна налаштувати зовнішній вигляд та функціональні можливості платіжної сторінки: від вибору методів оплати до додавання власного логотипу та кольорової гами, що дозволяє зберегти єдність бренду й підвищити довіру клієнтів. Документація Wayforpay написана простою й зрозумілою мовою, містить численні приклади використання API і покрокові інструкції, тому Wayforpay була вибрана як перша платформа для інтеграції платежів.

Переваги:

1. Проста процедура верифікації та запуску платіжних процесів.
2. Можливість налаштування інтерфейсу та функціональних параметрів під вимоги користувача.
3. Наявність великої кількості вбудованих способів оплати та простота їх імплементації на сторінку оплати.
4. Зрозуміла, інтуїтивно зрозуміла документація.

Недоліки:

1. Застаріла та незручна архітектура для створення платежів.
2. Великі тарифи на фоні конкурентів.

Але деякі технічні рішення платформи застаріли: архітектура створення транзакцій вимагає використання застарілих методів створення платежів (як і в випадку Liqpay використання HTML форм для генерації сторінки замовлення),

що сповільнює процес підтримки й оновлення власних модулів. Окрім того, рівень комісії в WayForPay перевищує тарифи конкурентів (2%).

Plata by Mono – має певні інноваційні підходи до опрацювання платежів із перевіреною репутацією МоноБанку. Завдяки невеликому терміну існування, платіжна платформа створювалася з нуля на основі сучасних архітектурних патернів: її API побудовані навколо принципів REST та WebHook-ів, а сам механізм обробки транзакцій розгорнуто у вигляді мікросервісів, що дозволяє гнучко масштабувати ПЗ під пікові навантаження. Відсутність необхідності проходити довгу процедуру верифікації мерчантів дозволяє підключатися до прийому оплат миттєво — достатньо мінімального набору даних, які автоматично перевіряються в рамках екосистеми МоноБанку.

Завдяки цьому бізнес отримує можливість швидко запускати нові проекти без затримок, притаманних класичним платіжним агрегаторам. Інтеграція здійснюється через простий виклик кількох REST-ендпоінтів, а приклади коду, адаптовані під популярні мови програмування, мають забезпечити миттєвий старт роботі з платіжною формою. Платформа також підтримує налаштування webhooks для підписки на події неуспішних списань, чого немає та не вистачає на Liqpay з Wayforpay.

Але при всієї чудовості Монобанку та його сервісу, є певні виклики: документація, з якою доведеться працювати, поки що написана не у найзручніший спосіб. Відсутність чіткої структури розділів та пов'язаних між собою прикладів коду викликає плутанину, змушують витратити більше часу. Дружелюбною технічною підтримкою Монобанк також не славиться, тому на самому початку з ним працювати складніше.

Переваги:

1. Відсутність потреби в підтвердженні мерчанту.
2. Сучасна архітектура.

Недолік:

1. Складна, погано структурована документація.

Порівняльний аналіз

LiqPay – розроблений ПриватБанком, LiqPay орієнтований на обробку платежів у режимі реального часу, що є значним плюсом для бізнесів, яким важлива швидкість транзакцій. Проте, застаріла архітектура для створення платежів ускладнює процес інтеграції, а довге підтвердження мерчанту може негативно впливати на загальну ефективність роботи системи.

Wayforpay – позиціонується як платформа з гнучкою інтеграцією, що дозволяє налаштовувати рішення під конкретні потреби бізнесу. Серед її переваг – велика кількість вбудованих способів оплати та інтуїтивно зрозуміла документація, яка полегшує процес впровадження. Однак, подібно до LiqPay, вона страждає від застарілої архітектури для створення платежів, а ще високі тарифи можуть стати перешкодою для малого та середнього бізнесу.

Plata by mono – сервіс, розроблений МоноБанком, вирізняється сучасною архітектурою, що відповідає стандартам сьогодення. Відсутність потреби в підтвердженні мерчанту значно спрощує роботу з платіжною системою, забезпечуючи швидкий старт інтеграції. Основним недоліком є складна та погано структурована документація, що може вимагати додаткових зусиль при розробці для належного впровадження.

1.2. Проблематика інтеграції українських платіжних сервісів у глобальні платформи

Інтеграція українських платіжних сервісів у глобальні e-commerce платформи стикається з рядом специфічних проблем, що виникають на перетині технологічних, юридичних та безпекових аспектів.

Технічні обмеження

Несумісність API та застаріла архітектура: Багато українських платіжних сервісів, таких як LiqPay та Wayforpay, використовують застарілі рішення для створення та обробки платежів. Це проявляється у вигляді невідповідності сучасним стандартам REST чи GraphQL API, які є основою інтеграції у глобальні платформи. В результаті, розробникам доводиться створювати додаткові адаптери, що ускладнює процес інтеграції та підвищує витрати часу та ресурсів.

Обмежена та неструктурована документація: Недостатньо розроблена та застаріла документація українських сервісів створює додаткові труднощі для розробників. Відсутність детальних інструкцій, прикладів коду та рекомендацій щодо інтеграції призводить до помилок і збільшення часу на розв'язання технічних проблем при підключенні.

Юридичні та фінансові нюанси

Відмінності у регуляторних вимогах: E-commerce платформи працюють у міжнародному правовому полі, де діють суворі стандарти щодо протидії відмиванню грошей, фінансової звітності та інших неприємних аспектів. Українські платіжні системи, адаптовані до національного законодавства, іноді не відповідають цим вимогам, що створює правові та операційні бар'єри для інтеграції, які варто обходити такими чином, щоб не порушувати користувацьку угоду (або просто залишатися непоміченим з боку адміністрації платформи).

Фінансові бар'єри та комісійна політика: Різниця у тарифах та комісійних зборах між українськими та глобальними сервісами впливає на економічну доцільність інтеграції. Наприклад, високі тарифи та затримки в підтвердженні мерчанта гарантовано стануть стримуючим фактором для міжнародних компаній, що бажають впровадити локальні рішення.

Вимоги до безпеки та відповідність міжнародним стандартам

Дотримання стандартів безпеки: Платформи повинні відповідати суворим міжнародним стандартам безпеки, таким як PCI DSS. Часто міжнародні корпорації припускають, що українські платіжні сервіси мають проблеми з адаптацією до цих стандартів через специфіку своєї архітектури та процесів.

Захист даних та шифрування: Інтеграція вимагає використання сучасних технологій шифрування та захисту даних. Якщо українські системи не володіють сучасними механізмами безпеки або мають недоліки в обробці конфіденційної інформації, це може призвести до ризиків витоку даних або несанкціонованого доступу під час обробки міжнародних транзакцій.

Підсумок

Для успішного впровадження необхідно вирішити наступні завдання:

1. Модернізація API та архітектури: Перехід до сучасних технологій інтеграції для забезпечення сумісності з глобальними платформами.
2. Поліпшення документації та підтримки розробників: Забезпечення доступу до актуальної та структурованої інформації для спрощення процесу інтеграції.
3. Рішення повинні візуально підходити під стандарти міжнародних E-commerce платформ та не приводити до порушень користувацької угода з наступним, неминучем баном магазину.

1.3. Огляд підходів та методологій для створення мультиплатформених інструментів

Успішна розробка мультиплатформених інтернет-інструментів вимагає поєднання гнучкої архітектури з методами, що забезпечують легке розширення й підтримку на різних платформах. Перш за все, важливим кроком є вибір підходу API-first, коли дизайн інтерфейсу взаємодії відокремлюється від реалізації бізнес-логіки. Такий підхід дозволяє одночасно розробляти клієнтські

додатки під веб, покладаючись на єдиний контракт, який гарантує сталість обміну даними. Упровадження контракт-драйвен розробки забезпечує чіткість специфікацій і дозволяє командам паралельно працювати над різними компонентами системи, зменшуючи ризик непорозумінь і помилок на етапах інтеграції.

Щоб забезпечити масштабованість і стійкість до навантажень, архітектура системи повинна передбачати рознесення бізнес-логіки по незалежних сервісах, які взаємодіють через легковагові HTTP-запити. Через контейнеризацію й оркестрацію за допомогою сучасних рішень буде автоматично масштабувати окремі сервіси відповідно до зростання кількості користувачів або обсягів оброблюваних даних. Підтримка версіонування API полегшує одночасне обслуговування декількох клієнтських додатків без ризику порушення сумісності, а застосування підходів з адаптивним розгортанням дозволяє плавно переходити від старих версій інтерфейсів до нових.

Важливим є процес автоматизації процесів розробки та тестування. Вбудовані в CI/CD-конвеєри юніт-тести разом із інтеграційними перевірками гарантують, що зміни в одному модулі не призведуть до несподіваних збоїв у інших. На додаток, моніторинг та централізоване логування дають змогу оперативно виявляти аномалії в роботі сервісів і своєчасно реагувати на інциденти.

Використання Node.js для інтеграції з платіжними рішеннями

Node.js – технологій у сфері серверної розробки завдяки своїй неблокуючій моделі вводу-виводу, що забезпечує високу продуктивність навіть при великій кількості одночасних запитів. Це особливо важливо для платіжних систем, де час обробки транзакцій має критичне значення. Крім того, активна екосистема npm надає можливість створити власні модулі для оптимізованої інтеграції з платіжними API (що буде зроблено на прикладі Wayforpay) та створення прозорості обробки даних для створення сторінок оплати.

Модульність та пакетна структура

Одним із основних підходів у створенні мультиплатформених інструментів є модульність. Розробка частини ПЗ у вигляді npm-бібліотеки дозволяє розбити систему на незалежні компоненти.

Така архітектура забезпечує:

1. Розмежування відповідальностей: Кожен модуль відповідає за конкретну функціональність, що дозволяє ізолювати помилки та спрощує підтримку коду.
2. Легку масштабованість: Додавання нових функцій або заміна окремих компонентів можливі без впливу на загальну роботу системи.
3. Повторне використання: Компоненти, розроблені для одного проєкту, можуть бути адаптовані та використані в інших рішеннях, що сприяє економії ресурсів та часу на розробку.

Сучасні методології розробки

Забезпечення високої якості і ефективності при розробці платіжного інструменту спирається на застосування інтегрованих сучасних підходів, що охоплюють усі фази життєвого циклу продукту. Поєднання гнучкості й швидкості реакції на нові вимоги ринку досягається за допомогою ітеративного підходу, коли кожен етап створення функціоналу супроводжується постійним зворотним зв'язком із зацікавленими сторонами та кінцевими користувачами. Такий формат дозволяє не лише своєчасно коригувати пріоритети розробки, а й мінімізувати ризики витрат часу на непотрібні рішення, оскільки команда систематично перевіряє і підтверджує актуальність ключових компонентів архітектури платіжного сервісу.

Важливою складовою цієї екосистеми – інтеграція практик DevOps, котра усуває традиційний поділ між розробниками й операційними підрозділами.

Завдяки автоматизації процесів збірки й доставлення коду, налаштування інфраструктури й розгортання нових версій програмного забезпечення зводяться до кількох безпечних і відтестованих кроків. Впровадження конвеєрів CI/CD гарантує, що кожен коміт проходить через набір юніт- і інтеграційних тестів, а будь-яка виявлена помилка виявляється на ранньому етапі, ще до потрапляння змін у production. Крім того, завдяки автоматичному моніторингу конвеєра можна своєчасно відстежувати погіршення показників продуктивності чи стабільності, що суттєво підвищує рівень довіри до платіжного рішення з боку бізнесу та користувачів.

Ще одним критично важливим елементом, здатним гарантувати надійну роботу системи в реальних умовах, є побудова тестових середовищ із можливістю симуляції платіжних запитів за допомогою Mock API. Створюючи фейкові транзакції та відтворюючи різноманітні сценарії обробки платежів, команда розробників отримує змогу виявити як логічні, так і функціональні помилки ще до інтеграції з банківськими шлюзами. Уніфіковані стенди для тестування дають змогу відпрацювати налаштування обробки успішних, відхилених та досліджуваних випадків оплати, а також перевірити поведінку системи під навантаженням. В результаті забезпечується не лише висока стійкість платформи до несподіваних ситуацій, а й гнучкість у подальшому масштабуванні та оновленні інструментів обробки фінансових операцій.

Переваги мультиплатформених підходів

Розробка інструментів, здатних працювати одразу з різними торговельними платформами, відкриває можливість значно оптимізувати як технічну складову проєкту, так і бізнес-логіку, що стоїть за ним. Поєднання єдиної кодової бази з чітко продуманою структурою дозволяє реалізувати обробку платежів у Shopify, WooCommerce, Wix та будь-яких інших системах без необхідності писати окремий модуль під кожну з них. Завдяки цьому суттєво знижується обсяг роботи при підтримці та оновленні — удосконалення

однієї частини коду автоматично поширюється на всі інтеграції й усуває дублювання зусиль у командах розробників.

Ключовою умовою гнучкості такого рішення стає впровадження модульної архітектури: кожна функціональна одиниця — від обробки запитів до системи авторизації й до реалізації обробки помилок — існує у вигляді окремого компонента з чітко визначеними інтерфейсами. У результаті змінити бізнес-логіку під конкретні потреби замовника чи підлаштувати спосіб автентифікації, не загрожуючи стабільності всієї системи, можна без значних правок у загальній кодовій базі. Такий підхід також створює сприятливі умови для паралельної роботи різних команд: поки одні фахівці реалізують новий каналний шлюз, інші можуть удосконалювати логіку розрахунку комісій чи додавати внутрішній моніторинг транзакцій.

У міру зростання бізнес-вимог і збільшення кількості клієнтів необхідність у впровадженні нових функцій стає питанням не місяців, а тижнів чи навіть днів. Завдяки продуманій системі підключення додаткових сервісів, таких як аналітика, служби виявлення шахрайства чи нові способи обробки валютних операцій, розробники можуть вносити зміни мінімальними зусиллями. Це сприяє швидкій адаптації до ринкових трендів і дозволяє бізнесу оперативно запропонувати клієнтам інноваційні рішення.

Використання Node.js як основи серверної частини створює додаткові переваги: за рахунок неблокуючої моделі вводу-виводу система здатна обслуговувати тисячі одночасних з'єднань, а величезна кількість пакетів у екосистемі npm забезпечує готові інструменти для будь-яких потреб. Поєднання подібної технології з методологіями Agile, глибоким тестуванням і безперервною інтеграцією гарантує, що платіжне рішення буде і продуктивним, і стабільним, і економічно виправданим як для локальних проєктів, так і для глобальних торговельних майданчиків.

1.4 Інтеграція платіжних систем у Shopify: аналіз ринку та рішень

Проблема прийому локальних онлайн-платежів для українських інтернет-магазинів на Shopify набуває все більшого значення, адже відсутність офіційної підтримки таких провайдерів, як Wayforpay, Monobank (Plata by mono) та Liqpay, безпосередньо позначається на швидкості запуску та зручності користування сайтом. Підприємці змушені шукати обхідні шляхи, інвестуючи час і ресурси в розробку власних рішень або звертаючись до сторонніх сервісів-посередників. У результаті реалізація навіть простого кошику з можливістю оплати стає серйозним технічним завданням/

Кожен крок на шляху інтеграції супроводжується додатковими витратами: розробники мають опрацьовувати механізми авторизації транзакцій, підтримувати актуальність SSL-сертифікатів, а також проектувати архітектуру таким чином, щоб у випадку відмови одного з сервісів ПЗ могла автоматично переключатися на резервні шляхи обробки платежів. При цьому сам магазин залишається на платформі Shopify, а вся логіка здійснення платежів “виноситься” за її межі — у вигляді окремого мікросервісу або приватного додатку, який працює за принципом посередника між сайтом і банківським шлюзом. Такий підхід підвищує гнучкість і дозволяє оперативно оновлювати компоненти оплати без втручання в ядро Shopify, але водночас створює нові точки відмови й вимагає постійного моніторингу та підтримки.

На практиці багато онлайн-продавців обирають варіант із залученням готових рішень від локальних ІТ-агенцій, що вже встигли напрацювати стандартизовані конектори до Wayforpay, Monobank і Liqpay. Завдяки цьому процес інтеграції скорочується до налаштування веб-хуків та верифікації домену в особистому кабінеті платіжного оператора, але відсутність глибокої кастомізації означає, що деякі підприємці змушені обирати менш оптимальні для себе умови комісій чи обмежені ліміти транзакцій.

Wayforpay

Через відсутність офіційної підтримки, інтеграція WayForPay у Shopify здійснюється сторонніми розробниками шляхом вставки платіжної форми на сторінку підтвердження замовлення (Thank You Page). Фактично, замовлення спочатку створюється у Shopify, а оплата відбувається після цього через зовнішню сторінку WayForPay [1, 2]. Такий підхід реалізовано, наприклад, у сервісів 3lines.tech та TradeMinister. Вони пропонують підключення WayForPay за одноразову плату та щорічну абонплату за хостинг інтеграції [3]. Деякі інтегратори оцінюють підключення дорожче – наприклад, один із сервісів пропонував інтеграцію WayForPay за ~10 800 грн разом із першим роком обслуговування [3]. Важливо, що додаткових комісій за транзакції інтегратори не стягують – мерчант сплачує лише комісію WayForPay.

Продавці на Shopify зазначають, що немає простої офіційної інтеграції, доводиться вдаватися до кастомних рішень. Дехто писав, що “неможливо підключити” стандартно і потрібно робити через API і сторонню допомогу. Це все ускладнює процес для середньостатистичного власника магазину.

Типові проблеми при використанні:

Окрім згаданих технічних складнощів (відсутність прямого шлюзу в адмінці), бізнеси стикаються з такими незручностями:

Витрати часу і грошей на інтеграцію: треба шукати розробника або платити за готове рішення (від \$100 і більше), чекати 1-2 дні на встановлення і витрачати щорічно кошти на підтримку. Для малого бізнесу це додаткове навантаження.

Неінтуїтивний досвід для покупця: клієнт оформлює замовлення, його переадресовує на зовнішню сторінку оплати, що може викликати питання довіри. Якщо покупець закрив сторінку чи передумав платити, замовлення вже створене – його доведеться скасувати вручну.

Інвентар та облік: оскільки Shopify віднімає товар зі складу при створенні замовлення, неоплачені замовлення через WayForPay можуть тимчасово “зарезервувати” товар. Якщо багато кинутих оплат, це спотворює залишки товарів.

Відсутність підтримки підписок і часткових оплат: поточні кастомні шлюзи не інтегруються з системою регулярних платежів Shopify. Також опція “оплата частинами” (розстрочка), яку пропонує WayForPay чи Monobank, може бути недоступна без окремої реалізації – TradeMinister, наприклад, пропонує окремий варіант інтеграції з функцією передоплати/розстрочки за вищою ціною.

Monobank

Прямої інтеграції “з коробки” Shopify для Monobank немає, тож використовуються ті самі підходи через сторонні сервіси. Компанії на кшталт 3lines.tech, Walnut (WNT Agency) і TradeMinister впроваджують MonoPay у Shopify аналогічно до WayForPay – через скрипт на сторінці успішного замовлення.

Вартість інтеграції в цілому порівнянна з Wayforpay: близько \$200 одноразово за встановлення + перший рік підтримки, і ~\$80 на рік надалі [2]. За ці гроші інтегратор забезпечує налаштування шлюзу, хостинг проміжного сервера для обробки платежів та технічну підтримку. Наприклад, компанія WNT (Walnut) зазначає ціну \$200 і час підключення 1–2 дні. TradeMinister пропонує MonoPay за ~10 800 грн з подальшою щорічною оплатою за сервер, що фактично відповідає тій самій моделі (\$100 + \$60/рік).

Додаткові комісії від інтеграторів відсутні – сплачуються лише тарифи Monobank. Як передумову, бізнес повинен мати рахунок ФОП або ТОВ у Monobank для користування їхнім еквайрингом.

MonoPay в Shopify працює за тим же принципом “редиректу після замовлення”. Замовлення створюється в Shopify, після чого покупець

перенаправляється на платіжну сторінку Monobank для введення даних карти або підтвердження через monobank-app. У разі успішної оплати інтеграція помітить замовлення як оплачене (Paid) автоматично.

Повернення коштів також обмежене: їх доведеться проводити вручну через бізнес-кабінет monobank, оскільки Shopify не має прямого зв'язку з MonoPay для виконання refund (у описі інтеграції прямо сказано: “Повернення тільки через кабінет Монобанку”).

Що стосується Apple Pay / Google Pay, MonoPay їх підтримує нативно, і інтегратори підтверджують, що ці методи будуть доступні при оплаті. Це дозволяє швидкий платежі “в один дотик” для покупців з Apple/Android-пристроями. Водночас, інтеграція не є глобальною – MonoPay оперує в гривні; приймати інші валюти без конвертації не вийде. Якщо магазин має міжнародні продажі, MonoPay може бути менш зручним, адже платежі з іноземних карт проводяться з підвищеною комісією 2% і, ймовірно, з конвертацією валют в гривню. Ще одне потенційне обмеження – необхідність бути клієнтом Monobank. Хоча відкриття рахунку просте, деякі підприємці віддають перевагу працювати з іншими банками. Для них ця вимога створює додатковий крок.

LiqPay

Близько 2019–2020 років у Shopify Community адміністрація пропонувала LiqPay як альтернативу для України [4], але вказувала тільки загальні інструкції з API. Наразі доступні кілька підходів: перший – використання приватного Shopify-App “CartDNA LiqPay”.

Компанії на кшталт Oganro пропонують додаток, який нібито “забезпечує безшовну інтеграцію LiqPay з рідним чекАутом Shopify”. Судячи з опису, цей app встановлює у магазині метод оплати LiqPay, і керування відбувається через зовнішню панель CartDNA. Однак цей шлях не є масовим – кількість встановлень такого додатку обчислюється десятками, і невідомо, чи він

доступний широкому загалу через App Store (можливо, його потрібно запитувати у розробника).

Другий шлях – послуги інтеграторів, аналогічно до WayForPay/Monobank. 3lines.tech, TradeMinister, Wallnut Digital активно підключають LiqPay на Shopify-проекти. Цінова політика схожа: одноразова плата \$100–200 та щорічна підтримка за \$60–80. Зокрема, Wallnut заявляє вартість \$200 за інтеграцію LiqPay (включно з першим роком підтримки) і далі \$80/рік за обслуговування. Час підключення – до 2 робочих днів. TradeMinister також вказує \$100 + \$60/рік, що для клієнта фактично проявляється як ~10 800 грн стартово. Обидва інтегратори підкреслюють, що ніяких додаткових комісій Shopify чи їхніх власних на платежі немає – тільки стандартні 1,5% LiqPay.

РОЗДІЛ II

ОГЛЯД ТЕХНОЛОГІЙ ТА МЕТОДІВ, ВИКОРИСТАНИХ ДЛЯ ПОБУДОВИ ФУНКЦІОНАЛЬНОСТІ ПЛАТІЖНОГО ШЛЮЗУ

2.1. Архітектура платіжного пакету на Node.js

Архітектура платіжного пакету на Node.js проєктується як набір взаємозалежних, але незалежно розв’язуваних модулів, кожен з яких відповідає за певний аспект обробки запитів та взаємодії із зовнішніми системами. На рівні прийому HTTP GET-запитів реалізується універсальний шлюз-контролер, який здійснює первинну валідацію параметрів і виконує маршрутизацію до відповідного адаптера платіжного провайдера. Кожен адаптер, у свою чергу, інкапсулює специфіку API відповідної платіжної системи — від формування підписаних запитів до обробки колбеків і підтвердження транзакцій. Завдяки чітко визначеним інтерфейсам цих модулів нові платіжні шлюзи можна підключати без змін у базових компонентах ядра сервісу.

Для забезпечення високої продуктивності застосовується модель асинхронної обробки подій із використанням неблокуючих операцій введення-виведення, що дозволяє обслуговувати велику кількість одночасних з’єднань. Важливу роль відіграють механізми кешування на рівні проміжної пам’яті (Redis або in-memory), а також горизонтальне масштабування контейнерів із балансуванням навантаження. Винагородою за складнішу інфраструктуру стає швидке реагування на пік навантажень і відсутність «вузьких місць» у вигляді блокуючих запитів до бази даних.

Питання безпеки вирішується на декількох рівнях: шифрування чутливих даних у стані спокою та під час передачі, підпис запитів із використанням HMAC-ключів, обмеження частоти звернень (rate-limiting) і централізоване логування всіх транзакцій у захищену систему моніторингу. Завдяки такому поєднанню структурної модульності, асинхронності та багаторівневого захисту платіжний пакет демонструє не лише високу стійкість до навантажень та атак, а

й здатність оперативно розширюватися під нові ринкові вимоги й інтегруватися з будь-якими e-commerce платформами.

Конектор до API платіжної платформи

Цей модуль відповідає за встановлення та підтримання зв'язку з зовнішніми платіжними сервісами. Він реалізовує методи для створення платежів, отримання статусів транзакцій, повернення коштів та інші операції. Конектор абстрагує специфіку різних API, забезпечуючи єдиний інтерфейс для подальшої інтеграції.

Менеджер вебхуків

Для оперативного оновлення статусу транзакцій та синхронізації з платіжною системою використовується менеджер вебхуків. Він приймає callback-запити від сервісів, перевіряє їхню автентичність за допомогою JWT (JSON Web Tokens) та ініціює відповідні дії у системі – зміну статусу замовлення, обробку повернень, логування подій.

Система налаштувань та конфігурації

Пакет включає модуль, який дозволяє гнучко налаштовувати параметри інтеграції. Це API, URL-адреси endpoint-ів платіжних систем, налаштування безпеки, параметри логування та інші опції необхідні для кастомізації платежів.

Модуль обробки помилок та логування:

Висока надійність платежів вимагає ретельного контролю за помилками. Спеціальний компонент відслідковує помилки під час транзакцій, забезпечує їх коректну обробку та автоматичне повторне виконання (retry), і реєструє інформацію для подальшого аналізу. Це потрібно для оперативно виявлення та усунення багів, підвищуючи стабільність роботи ПЗ.

Система тестування

Для забезпечення якості коду та коректності роботи інтеграції використовуються як модульні, так і інтеграційні тести. Використання Mock

API дозволяє симулювати різні сценарії оплати, відмов та повторних запитів. Воно гарантує надійність рішення перед його впровадженням у продакшн.

Використання TypeScript

Застосування TypeScript у проекті платіжного пакету значно підвищує надійність усієї системи ще на етапі розробки. Наявність суворої типізації дає змогу виявляти невідповідності й помилки ще до запуску коду, оскільки компілятор перевіряє відповідність переданих даних очікуваним структурам. Це особливо важливо в роботі з фінансовими даними, де будь-яка неточність у форматі запиту чи відповіді може стати причиною відмови транзакції або, що гірше, витоку чутливої інформації.

Другою перевагою є зрозумілість і передбачуваність у співпраці розробників над різними модулями. Чіткі інтерфейси й декларації типів самі по собі документують увесь набір полів і методів, необхідних для коректної взаємодії компонентів. В результаті при зміні протоколу обміну з платіжним провайдером або додаванні нового адаптера команда одразу бачить, які частини коду впливаються, та безпомилково оновлює відповідні виклики й структури даних.

Нарешті, коли платіжний пакет розростається й до нього додаються нові функціональні блоки, TypeScript виступає гарантом збереження сумісності між старими та новими частинами системи. Завдяки жорстким контрактам, які визначаються в типах і інтерфейсах, будь-яка зміна в одному модулі моментально відображається в помилках компіляції там, де інші компоненти сподіваються на початкову поведінку. Це дозволяє впевнено розгортати оновлення та масштабувати платіжний сервіс без побоювань, що новий код поламає критично важливі процеси обробки транзакцій.

Сумарна архітектурна схема

Архітектура побудована за принципом розподіленої системи, де кожен компонент має чітко окреслену відповідальність [Рис. 2.1].

Це дозволяє:

1. Легко масштабувати рішення: Додавати нові функції або замінювати окремі модулі без необхідності переписувати весь код.
2. Адаптуватися до нових вимог ринку: Швидко інтегрувати підтримку додаткових платіжних систем або нових функціональних можливостей.
3. Підвищувати рівень безпеки: Ізоляція критичних функцій дозволяє ефективно контролювати доступ до конфіденційних даних та зменшувати ризики.

Спланована та розроблена архітектура ПЗ забезпечує надійну, масштабовану та високопродуктивну основу для інтеграції з різними платформами. Ця архітектура – той мінімум, який потрібен для випуску ПЗ на український ринок.

2.2. Схема обробки платежу та авторизації

Схема обробки платежу побудована на розподілі етапів взаємодії між інтернет-магазином, шлюзом REST API, Node.js-пакетом та платіжною платформою, що забезпечує безперебійну, безпечну та ефективну обробку транзакцій.

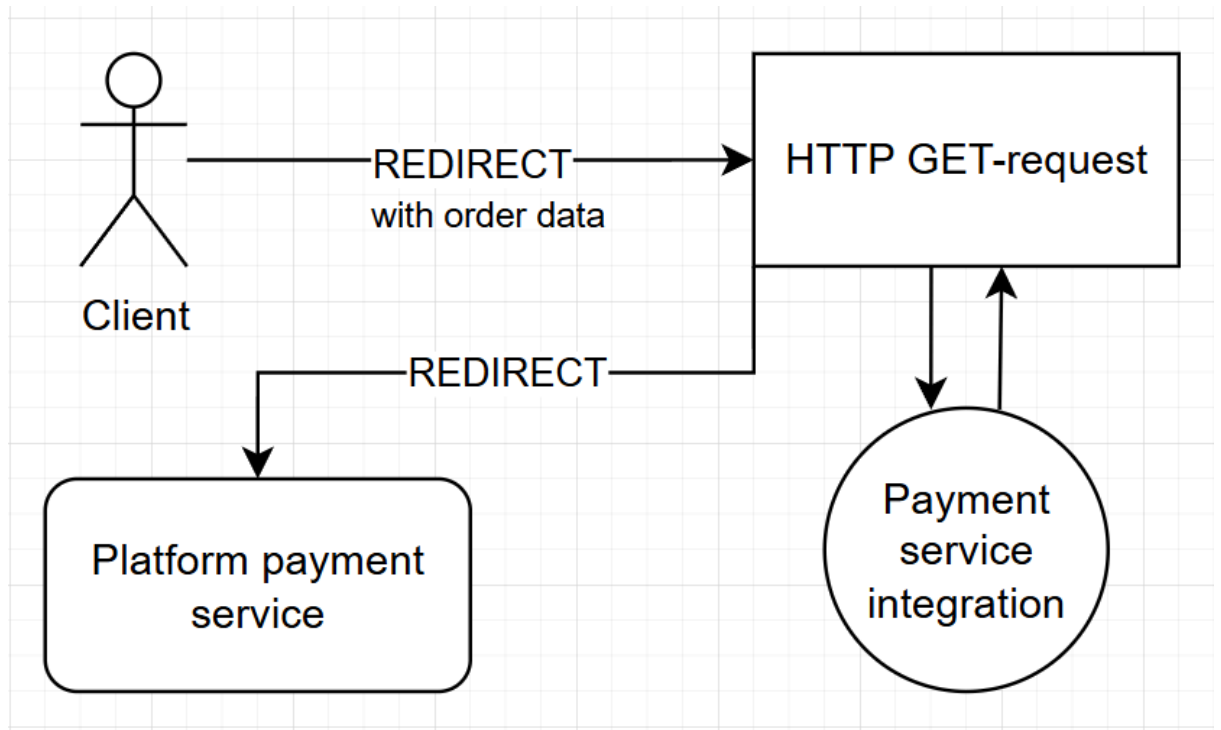


Рис. 2.1 Спрощена схема взаємодії клієнта із шлюзом

Ініціалізація платежу

Користувач обирає товар і переходить до оформлення замовлення в інтернет-магазині.

Після натискання кнопки «Оплатити» дані замовлення надсилаються на REST API, де розташований Node.js-пакет.

Пакет перевіряє правильність введених даних та формує початковий платіжний запит, повертаючи його покупцю.

Запит для ініціалізації транзакції формується з урахуванням вимог платіжної системи і підписується за допомогою HMAC (Hash-based Message Authentication Code) для захисту від підробок.

Покупець приймає згенерований запит та виконує його, звертаючись до платіжної платформи.

Платіжна система перевіряє автентичність запиту за допомогою HMAC-підпису.

Далі відбувається редірект покупця на сторінку оплати, де він сплачує замовлення по згенерованим даним.

Після оплати, платформа отправляє callback на REST API, яке має відповідає за логування та керування даними про замовлення.

Авторизація та формування запиту

Для забезпечення безпеки даних, середовище де відбувається налаштування, кастомізація оплати використовує OAuth для автентифікації продавців.

Для додаткової захищеності застосовуються JWT (JSON Web Tokens), які потрібні для перевірки цілісності і походження даних.

Механізми безпеки

1. OAuth: Використовується для автентифікації доступу до налаштування та кастомізації платежів з боку продавця.
2. JWT: Забезпечує захищену передачу даних між ПЗ, дозволяючи перевірити їхню цілісність і походження.
3. HMAC: Забезпечує цілісність даних, підписуючи запити і відповіді, що унеможлиблює їх підробку. Потрібно виключно для платіжних платформ.
4. TLS/SSL: Забезпечує шифрування даних під час їх передачі, що мінімізує ризик їх перехоплення або несанкціонованого доступу.

Переваги схеми

Висока безпека. Поєднання сучасних протоколів шифрування і автентифікації забезпечує захист даних користувачів на всіх етапах транзакції.

Масштабованість. Модульна архітектура дозволяє легко додавати підтримку нових платіжних платформ або розширювати функціонал без переривання роботи вже створених сервісів.

Інтеграція з e-commerce платформами. Цей потік даних гарно структурований і дозволяє інтегрувати шлюз у різні інтернет-магазини, зберігаючи узгодженість статусів замовлень та обробки транзакцій.

2.3. Застосування Mock API та тестових середовищ

Щоб забезпечити якість та стабільність шлюзу та пакету дуже важливо імплементувати тестування. В проекті використано Mock API для тестування самої роботи шлюзу для продакшена та Jest для тестування пакету wayforpay-ts-integration.

Роль Mock API у тестуванні

Mock API дозволяє емуляцію роботи зовнішніх платіжних сервісів без необхідності проведення реальних транзакцій. Завдяки цьому можна:

Імітувати різні сценарії. Створити як позитивні сценарії, кейси успішної оплати, так і негативні сценарії (відхилення платежу, помилки на стороні платіжної платформи, таймаут), що дозволяє протестувати реакцію ПЗ на різноманітні варіанти відповіді.

Прискорити процес розробки. Оскільки Mock API не вимагає звернення до реальних сервісів, це дозволяє розробникам швидко виявляти та виправляти помилки, не чекаючи результатів від зовнішніх API.

Знизити витрати. Тестування з використанням Mock API не вимагає реальних фінансових операцій, що зменшує ризики витрат під час розробки та валідації функціоналу.

Тестові середовища та їх використання

Mock API окрім своєї реалізації потребує також правильного енвайронмента для тестування. Для його створення є декілька підходів:

Локальне тестування. Розгортання локального серверу, де працює платіжний пакет, та налаштування симульованих платежів. Це дозволяє

перевірити роботу всіх компонентів – від обробки запитів до керування вебхуками.

CI/CD інтеграція. Автоматизація запуску тестів при кожному оновленні коду дозволяє впевнитися, що нові зміни не порушують існуючий функціонал. Інтеграційні тести, виконувані в рамках CI/CD, допомагають швидко виявити помилки на ранніх етапах розробки.

Тестування відмов та повторних запитів. В тестовому середовищі моделюються різні типи збоїв: затримки відповіді, випадкові відмови API або помилки під час обробки платежів. Дяки цьому можна перевірити коректність роботи механізмів повторних спроб (retry) та обробки помилок.

Приклади застосування

Сценарій успішної транзакції. Mock API повертає статус успішного платежу, після чого менеджер вебхуків оновлює статус замовлення на "оплачено" в БД інтернет-магазину.

Сценарій відмови платежу. При моделюванні відмови, Mock API генерує помилку (наприклад, через перевищення ліміту, неправильні дані карти або проблеми з авторизацією). ПЗ фіксує помилку, ініціює механізм повторної спроби та, у разі невдачі, інформує адміністратора.

Сценарій таймауту. Імітація затримки відповіді дозволяє перевірити, чи правильно ПЗ обробляє таймаути і чи запускається механізм повторного запиту, забезпечуючи стабільність роботи в умовах нестабільного з'єднання.

Через обмеженні ресурси, буде використано саме локальне тестування API програмного забезпечення. Воно дозволить швидко тестувати і не потребує багато зусиль в реалізації, вистачить налаштувати відповідним чином curl запити.

Jest же дозволить тестувати певні кейси роботи пакету [Рис. 2.2].

2.4. Інтеграція з іншими платформами

Інтеграція платіжного шлюзу з популярними e-commerce платформами є основною ідеєю роботи та найважливішим кроком в розширенні ринкового охоплення та забезпечення універсальності ПЗ. Завдяки модульній архітектурі Node.js-пакету, його можна легко адаптувати для роботи з різними платформами, враховуючи їх специфічні вимоги та механізми інтеграції.

Shopify

Інтеграція з Shopify реалізується через встановлення додатку з Shopify App Store. Рішення використовує REST API шопіфаю для обміну даними між платформою та платіжним шлюзом. Завдяки цьому забезпечується:

Безшовна інтеграція в процес оформлення замовлення. Покупця безпечно перенаправляє на перевірену сторінку оплати з логотипами платформ і банків, які вселяють простій людині довіру, та політиками платформи, де покупець і оформляє замовлення.

Автоматичне оновлення статусів транзакцій. Завдяки callback-запитам вебхуків, статус замовлення синхронізується з даними платіжної системи.

```

describe('Wayforpay.purchase', () => {
  let wayforpay: Wayforpay;

  beforeEach(() => {
    wayforpay = new Wayforpay(testMerchant);
  });

  it('should throw an error if merchantSecret is not specified', async () => {
    const instanceWithoutSecret = new Wayforpay({
      merchantLogin: 'testMerchant',
      merchantSecret: undefined,
    });

    await expect(instanceWithoutSecret.purchase([])).rejects.toThrowError(secretSpecifiedError);
  });

  it('should generate a valid HTML form for the purchase', async () => {
    const cart: TCartItem[] = [
      {
        product: { name: 'Product 1', price: 100 },
        quantity: 2,
      },
      {
        product: { name: 'Product 2"', price: 200 },
        quantity: 1,
      },
    ];

    const form = await wayforpay.purchase(cart, { domain: 'example.com', currency: 'UAH' });

    console.log('Purchase form:\n', form);

    expect(typeof form).toBe('string');
  });

  it('should include additional fields passed in the data parameter', async () => {
    const cart: TCartItem[] = [
      {
        product: { name: 'Product 1', price: 50 },
        quantity: 3,
      },
    ];
  });

```

Рис. 2.2 Код тестування пакету для створення оплати через Wayforpay

Можливість використання кастомних налаштувань. Розроблений модуль дозволяє адаптувати інтерфейс під бренд магазину та забезпечити підтримку додаткових функцій (наприклад, рекурентні платежі).

Wix надає свій App Market, де розробники можуть публікувати інтеграційні рішення. Для інтеграції платіжного пакету на платформі Wix можна використовувати:

Вбудовані API Wix. Інтерфейси Wix дозволяють інтегрувати зовнішні рішення, що забезпечують обробку платежів.

Налаштування плагінів. Подібно до Shopify, плагіни можуть бути розроблені для додавання платіжного модуля, який працює в рамках існуючої архітектури Wix.

WooCommerce, як плагін для WordPress, підтримує розширення через додаткові модулі. Інтеграція платіжного пакету здійснюється за допомогою:

Плагінної системи WordPress. Розробка спеціального плагіну, який забезпечує взаємодію з Node.js-пакетом через REST API, дозволяє приймати платежі безпосередньо в процесі оформлення замовлення.

Використання hooks та фільтрів. WooCommerce дозволяє налаштовувати поведінку системи за допомогою hooks, що спрощує інтеграцію та забезпечує гнучкість налаштувань.

Приклади (Proof of Concept) підключення до Shopify

Proof of Concept інтеграції для Shopify включає наступні етапи:

Розробка додатку для Shopify. Створюється невеликий додаток, який реєструється в Shopify App Store. Додаток забезпечує зв'язок між Shopify і сервером Node.js-пакету за допомогою REST API.

Налаштування платіжного процесу. При оформленні замовлення додаток ініціює платіж через Node.js-пакет. Покупець вводить дані платіжної картки безпосередньо у вбудованій формі, яка відображається на сторінці чека у Shopify.

Обробка транзакції та синхронізація статусів. Після обробки транзакції платіжна система надсилає callback (вебхук) з результатом оплати. Додаток отримує цей callback, оновлює статус замовлення у Shopify і відображає повідомлення для користувача.

2.5. Налаштування середовища розробки та початкова структура проекту

Важливим кроком буде налагодження стабільного середовища розробки для успішної реалізації платіжного пакету. Правильне налаштування інструментів забезпечує ефективну співпрацю команди (якщо б вона була), високий рівень підтримуваності коду та масштабованість ПЗ.

Вибір технологій та інструментів

Node.js та npm. Використання останньої LTS-версії Node.js гарантує стабільність і сумісність із сучасними пакетами. Менеджер пакетів npm використовується для управління залежностями, автоматизації скриптів запуску, тестування та збірки проекту.

TypeScript. Завдяки строгій типізації, TypeScript допомагає виявляти помилки на етапі розробки (через вбудовані інструменти IDE) та компіляції, що сприяє підвищенню якості коду. Створення конфігураційного файлу `tsconfig.json` дозволяє налаштувати параметри компіляції, визначити використовувані модулі та шляхи до вихідних файлів.

ESLint та Prettier. Для забезпечення єдиного стилю коду та попередження потенційних помилок впроваджується ESLint. У поєднанні з Prettier, що відповідає за форматування, ці інструменти сприяють підтриманню чистоти та консистентності коду в команді. Налаштування відбувається через конфігураційні файли `.eslintrc.json` та `.prettierrc`.

Jest. Для модульного та інтеграційного тестування обирається Jest – сучасний фреймворк, що дозволяє створювати автоматизовані тести. Налаштовується файл `jest.config.js`, який визначає середовище тестування, трансформери для TypeScript та інші необхідні параметри.

Початкова структура проекту

Структурування коду за принципами модульності дозволяє забезпечити масштабованість та легкість подальшої підтримки системи. Рекомендується створити окремі каталоги для основних компонентів:

/src – основний каталог з вихідними файлами. /src/api – модулі для роботи з API платіжних систем (конектори, адаптери). /src/webhooks – обробка callback-запитів та інтеграція вебхуків. /src/config.json – конфігураційні файли, налаштування. /src/utls – утиліти, логіка обробки помилок, функції для логування та повторних спроб. /src/types – визначення інтерфейсів і типів, що використовуються в ПЗ. /tests – каталог з тестовими файлами, що містить модульні та інтеграційні тести, реалізовані за допомогою Jest. package.json – основний файл проекту, де вказані всі залежності, скрипти для запуску, тестування та збірки додатку. .env – файл для зберігання конфіденційних даних (API-ключів, URL-адрес сервісів), який додається до .gitignore для забезпечення безпеки.

Додаткові аспекти налаштування

Окрім основних етапів організації середовища розробки, доцільно врахувати низку додаткових аспектів, які значно впливають на якість, стабільність та ефективність подальшої роботи над ПЗ. Ці підходи сприяють оптимізації потенційної командної роботи (яка могла би вестися), полегшують підтримку коду та забезпечують безперервну інтеграцію та деплоймент, що є мастхев у сучасних ПЗ.

Інтеграція з системою контролю версій

Системи контролю версій Git стане невід'ємною складовою процесу розробки. Ретельне налаштування репозиторію, включаючи створення відповідного файлу .gitignore для виключення файлів, які не потребують відстеження (директорія node_modules, файли конфігурації середовища та файли збірки), буде підтримувати чистоту історії версій. Це, у свою чергу,

сприяє злагодженій колективній роботі над проектом (в випадку одиничної роботи, просто бекапу коду), забезпечує прозорість змін у кодовій базі та полегшує аналіз внесених модифікацій у довгостроковій перспективі.

Документування коду

Документування буде ключовим елементом забезпечення ефективності майбутньої підтримки ПЗ. Використання коментарів, стандартів на зразок JSDoc та інших засобів документування допомагає детально описати функціональність кожного модуля, класу чи функції. Такий підхід сприяє швидкому орієнтуванню в архітектурі проекту і мінімізує ризик виникнення помилок при інтеграції нових компонентів, що в свою чергу підвищує загальну якість ПЗ.

Налаштування локального тестового серверу

Розгортання локального серверу, зокрема з використанням Express на Node.js, дозволить створити тестове середовище для емуляції роботи реальних API реквестів. Це дасть змогу провести глибоку валідацію функціональності ПЗ в умовах, максимально наближених до продакшена. Таке тестове середовище забезпечить можливість моделювання різноманітних сценаріїв взаємодії між компонентами, дозволяючи виявляти та усувати можливі недоліки на ранніх етапах розробки. Завдяки цьому, розробники отримують змогу оперативно коригувати логіку роботи ПЗ, що суттєво підвищує її надійність.

Автоматизація через CI/CD процеси

Впровадженням CI/CD (безперервної інтеграції та безперервного деплою) за допомогою сервісу GitHub Actions, можна автоматизувати процеси тестування, збірки та розгортання ПЗ. Автоматизація процесів буде гарантувати, що кожна зміна в коді проходить ретельну перевірку перед її потраплянням у продакшен середовище, що значно знизить ризик введення нових помилок у роботу ПЗ. Крім того, це забезпечить стабільність проекту,

дозволяє швидко отримувати зворотній зв'язок та адаптуватися до змін, що є особливо важливим у динамічно розвинутих системах.

Комплексне врахування додаткових аспектів налаштування середовища розробки забезпечить не лише технічну ефективність, а й сприяє підвищенню організаційної дисципліни, теоритично покращує взаємодію між розробниками та зменшенню операційних ризиків. Ці фактори надзвичайно важливі для створення стабільного, масштабованого та надійного платіжного пакету, здатного задовольнити вимоги сучасного ринку.

2.6. Опис розробки основних функцій платіжного пакету

Розробка основних функцій платіжного пакету є ключовим етапом реалізації проекту, адже саме ці функції забезпечують обробку транзакцій, інтеграцію з платіжними платформами та взаємодію з e-commerce платформами. В рамках даного розділу розглядаються основні компоненти функціоналу, їх логіка роботи та реалізація механізмів забезпечення стабільності, безпеки та масштабованості рішення.

Ініціалізація та налаштування платіжного сервісу

На першому етапі реалізації платіжного пакету здійснюється ініціалізація ПЗ, яка включає налаштування ключових параметрів для роботи з платіжними платформами. Це передбачає:

Конфігурацію параметрів. Дані про API-ключі, секрети, URL-адреси сервісів та інші необхідні налаштування зберігаються у конфігураційних файлах або змінних середовища. Такий підхід забезпечує безпечне зберігання конфіденційної інформації та дозволяє легко змінювати параметри для різних середовищ (development, staging, production).

Ініціалізацію конекторів. Створюються модулі, що відповідають за взаємодію з API платіжних систем. Ці конектори інкапсулюють специфічну логіку роботи з кожною платіжною платформою, забезпечуючи єдиний інтерфейс для подальшої інтеграції. Завдяки цьому, при необхідності підтримки

нової платіжної системи, досить додати новий модуль-конектор, не змінюючи загальну архітектуру.

Реалізація роутингу та обробка транзакцій

Основні функції платіжного пакету включають розробку роутів для створення, підтвердження та повернення платежів, а також обробку вебхуків:

Створення платежів. Після отримання запиту від клієнта, система генерує платіжну сесію. Для цього формується запит до платіжного шлюзу із зазначенням суми, валюти, ідентифікатора замовлення та інших параметрів. Запит підписується за допомогою НМАС, що забезпечує цілісність даних. Результатом є отримання унікального ідентифікатора транзакції, який використовується для подальшої перенаправлення користувача на оплату.

Підтвердження транзакцій. Після ініціалізації платежу платіжна платформа проводить авторизацію операції. Через вебхук ПЗ отримує callback, де міститься інформація про статус платежу. На цьому етапі здійснюється валідація отриманих даних НМАС, після чого оновлюється статус замовлення в БД інтернет-магазину. Таким чином, покупець та адміністратор отримують достовірну інформацію про завершення транзакції.

Повернення коштів. Для випадків, коли виникає необхідність повернення коштів, розроблений модуль надсилає запит до відповідного API платіжної системи з деталями повернення. Подібно до процесу створення платежу, запит включає в себе ідентифікатор транзакції, суму повернення та інші параметри. Після успішної обробки повернення ПЗ фіксує цю операцію, що дозволяє вести коректну бухгалтерську документацію та підтримувати прозорість операцій.

Обробка вебхуків. Вебхуки зручна складовою, адже вони дозволяють отримувати асинхронні повідомлення від платіжних платформ про зміни статусу транзакцій.

Модуль вебхуків відповідає за:

1. Прийом callback-запитів.
2. Валідацію автентичності даних.
3. Оновлення статусів замовлень у ПЗ.
4. Реєстрацію інформації про транзакції для подальшого аналізу та логування.

Обробка помилок, винятків та механізми повторних спроб

Надійність платіжного пакету залежить від здатності ПЗ коректно реагувати на непередбачувані ситуації та збої. Для цього впроваджено комплекс заходів:

Логування та моніторинг. Ретельне логування кожного реквесту та відповіді дозволяє відслідковувати роботу ПЗ в режимі реального часу. У разі виникнення помилок, журнал логів допомагає швидко ідентифікувати проблемні місця та впровадити необхідні виправлення.

Механізм повторних спроб (Retry). У випадках тимчасових збоїв (наприклад, мережових проблем або затримок у відповіді платіжної системи) реалізовано механізм автоматичного повторного виконання запитів. Це дозволяє зменшити ризик помилок у транзакціях та забезпечити безперебійну обробку платежів.

Обробка винятків. Уся логіка роботи модулів охоплена блоками try-catch, які дозволяють відловлювати неочікувані помилки та забезпечувати їхню належну обробку. У разі критичних збоїв ПЗ ініціює сповіщення для адміністратора, що дозволяє оперативно реагувати на інциденти.

2.7. Модульні та інтеграційні тести

Якісне ПЗ не вийде без якісного тестування, а для платіжного пакету, де помилка може призвести до серйозних наслідків, якість критична.

Модульне тестування

Модульні тести спрямовані на перевірку окремих компонентів ПЗ і пакету. Завдяки ізоляції кожного модуля від ПЗ, тестувальники можуть зосередитися на конкретних функціях та логіці, що мінімізує ймовірність появи критичних помилок. Основні аспекти модульного тестування включають:

Перевірка функціональності конекторів. Кожен конектор, який реалізує зв'язок із API платіжних систем, повинен проходити тестування на коректність формування запитів, обробку відповідей та валідацію даних. Наприклад, тестування має перевіряти правильність підписування запитів за допомогою НМАС, а також коректність роботи механізмів автентифікації.

Тестування логіки обробки помилок. Всі блоки, що відповідають за обробку винятків та механізми повторних спроб, повинні бути ретельно протестовані. Це дозволяє виявити ситуації, коли ПЗ може не зуміти відновитися після тимчасового збоїв у мережевому з'єднанні або при невдалій транзакції.

Валідація утиліт і функцій. Допоміжні функції, що відповідають за логування, форматування даних або виконання розрахунків, повинні бути охоплені тестами, що перевіряють їхню роботу з різноманітними вхідними даними, включаючи граничні випадки.

Інтеграційне тестування охоплює взаємодію між різними модулями ПЗ, перевіряючи, як компоненти працюють разом у реальному середовищі. Для ПЗ це означає перевірку взаємодії між ініціалізацією платежу, обробкою вебхуків та механізмами повернення коштів.

Основні напрямки інтеграційного тестування включають

Сценарії успішних транзакцій. Симуляція повного циклу оплати, де замовлення створюється, ініціюється платіж, а потім через вебхук отримується підтвердження успішної операції. Тести мають перевіряти коректне оновлення статусу замовлення та збереження інформації про транзакцію в базі даних.

Негативні кейси. Окрім позитивних сценаріїв, тестування охоплює ситуації, коли транзакція відхиляється або виникають помилки під час обробки. За допомогою Mock API можна змодельовати випадки некоректних даних, затримок у відповіді платіжної платформи або відмови в авторизації, що дозволяє перевірити коректність роботи механізмів повторних спроб і обробки помилок.

Синхронізація статусів і обробка вебхуків. Інтеграційні тести повинні підтверджувати, що отримані через вебхуки дані правильно обробляються, а статуси замовлень у системі синхронізуються із зовнішніми платіжними сервісами. Це буде важливо для забезпечення достовірності даних у режимі реального часу.

Використання Mock API та тестових середовищ

Для ефективного тестування пакету використовуються Mock API, які імітують роботу зовнішніх платіжних систем. Це дозволяє:

Зменшити залежність від зовнішніх сервісів. Використання Mock API забезпечує можливість тестувати функціональність без необхідності звернення до реальних платіжних платформ, що складне через обмеження доступу, комісійні витрати та затримки в обробці запитів.

Створити гнучке тестове середовище. Розробка локальних тестових серверів дозволяє змодельовати різноманітні сценарії, включаючи несподівані помилки або тимчасові збої, що дає змогу перевірити стійкість ПЗ до різних умов експлуатації.

Автоматизація тестування через CI/CD

Впровадження систем автоматичного тестування, інтегрованих у процес CI/CD, дозволяє забезпечити безперервну перевірку якості коду. Завдяки цьому кожна зміна в коді автоматично проходить набір тестів, що гарантує, що нові функції не порушують існуючий функціонал.

Скорочення циклу розробки. Автоматизоване тестування дозволяє швидко виявляти помилки, що суттєво прискорює процес розробки та впровадження нових функціональних можливостей.

Підвищення стабільності продукту. Безперервна інтеграція тестів дозволяє виявити навіть найдрібніші недоліки на ранніх етапах розробки, що зменшує ризик виникнення критичних збоїв у виробничому середовищі.

Забезпечення якості коду. Інтеграція тестових сценаріїв в CI/CD-процес гарантує, що кожен комміт в репозиторій проходить необхідні перевірки, забезпечуючи стабільність і надійність кінцевого продукту.

Підсумок

Комплексне модульне та інтеграційне тестування буде ключовим аспектом розробки платіжного пакету, воно забезпечить високий рівень якості, стабільності та безпеки ПЗ. Використання Mock API дозволить емулювати роботу зовнішніх платіжних платформ, що значно полегшує перевірку різноманітних сценаріїв роботи. Автоматизація тестування через CI/CD сприяє безперервному контролю якості коду, дозволяючи швидко виявляти та виправляти недоліки. Цей підхід є фундаментальним для створення надійного платіжного рішення, здатного відповідати сучасним вимогам ринку та забезпечувати ефективну обробку фінансових транзакцій.

2.8. Оптимізація та безпека

У сучасних платіжних платформах оптимізація продуктивності та забезпечення безпеки даних це критичний аспект, який визначає якість, стабільність та довіру до рішення. Платіжний пакет повинен не тільки оперативно обробляти велику кількість транзакцій у режимі реального часу, але й гарантувати захист конфіденційної інформації, що відповідає міжнародним стандартам безпеки.

Оптимізація продуктивності

Для забезпечення високої швидкодії роботи ПЗ важливо впровадити декілька заходів оптимізації:

Ефективне використання ресурсів Node.js. Завдяки неблокуючій моделі вводу-виводу Node.js дозволяє обробляти численні запити одночасно. Однак для досягнення максимальної продуктивності необхідно оптимізувати код, використовуючи асинхронні операції та зменшуючи кількість синхронних викликів, що можуть призводити до затримок.

Оптимізація запитів до бази даних. Впровадження індексів, використання оптимізованих запитів та застосування технологій кешування через Redis дозволяють значно скоротити час доступу до даних, що важливо для обробки транзакцій.

Балансування навантаження. Розподіл запитів між кількома серверами або використання мікросервісної архітектури допомагає уникнути перевантаження однієї інстанції, забезпечуючи стабільну роботу ПЗ навіть при пікових навантаженнях.

Забезпечення безпеки конфіденційних даних

Платіжний пакет має працювати з великою кількістю чутливої інформації, зокрема даними карток, токенами та секретними ключами.

Захищене зберігання секретів. Токени, API-ключі та інші конфіденційні дані мають зберігатися у захищених сховищах (наприклад, за допомогою AWS Secrets Manager або HashiCorp Vault), або використовувати шифровані змінні середовища. Це дозволяє мінімізувати ризик несанкціонованого доступу до критичних даних.

Обмеження доступу. Принцип найменших привілеїв має бути реалізований на всіх рівнях системи – доступ до конфіденційних даних буде мати лише обмежена кількість сервісів.

Використання сучасних протоколів безпеки

Для забезпечення безпеки обміну даними між компонентами системи використовуються сучасні стандарти та протоколи:

TLS/SSL – всі комунікації між сервером Node.js та зовнішніми платіжними платформами будуть проходити через захищені канали. Це гарантує шифрування переданої інформації та запобігає її перехопленню.

OAuth, JWT та HMAC – використання цих механізмів автентифікації і авторизації дозволить підтверджувати цілісність запитів і відповіді, захищаючи ПЗ від підробок та несанкціонованих дій. Ці протоколи забезпечують безпечну ідентифікацію користувачів і підтвердження транзакцій.

Відповідність міжнародним стандартам безпеки

ПЗ для роботи з платіжними платформами повинно відповідати вимогам стандарту PCI DSS (Payment Card Industry Data Security Standard), що забезпечує комплекс заходів для захисту даних платіжних карток. Дотримання цих стандартів включає:

Регулярний аудит. Проведення періодичних перевірок безпеки, що включають оцінку вразливостей, аналіз логів та тестування на проникнення.

Використання сучасних алгоритмів шифрування. Застосування криптографічних алгоритмів для захисту даних під час зберігання та передачі інформації буде гарантувати, що навіть у разі витоку даних, вони залишатимуться недоступними для злоумисників.

Контроль доступу та моніторинг. Забезпечення постійного моніторингу ПЗ за допомогою журналювання дій та інтеграції з системами управління інцидентами дозволяє оперативно виявляти та реагувати на потенційні загрози.

Оптимізація кешування

Для зменшення навантаження на сервер та підвищення швидкодії обробки запитів до БД будуть впровадженні стратегії кешування.

Використання Redis або схожих технологій. Зберігання часто використовуваних даних у пам'яті дозволить значно скоротити час відповіді ПЗ. Це особливо важливо для операцій, що повторюються, таких як перевірка статусу транзакції та отримання налаштувань ПЗ.

Контроль актуальності кешованих даних. Встановлення правил для інвалідизації кешу дозволяє гарантувати, що користувачі отримують актуальну інформацію, що особливо важливо для фінансових транзакцій.

Надійна система безпеки повинна постійно контролювати свою роботу:

Системи моніторингу. Використання інструментів моніторингу дозволить відслідковувати роботу серверів, виявляти аномалії в роботі ПЗ та оперативно реагувати на потенційні загрози.

Логування та аудит. Детальне логування операцій дозволяє не лише відслідковувати хід виконання транзакцій, але й проводити аналіз інцидентів безпеки. Регулярний аудит коду та інфраструктури сприяє виявленню недоліків, які можуть бути експлуатовані зловмисниками.

Підсумок

Оптимізація продуктивності та забезпечення безпеки стане основними складовими успішного функціонування платіжного пакету. Завдяки впровадженню сучасних технологій, алгоритмів шифрування, належній організації кешування та автоматизованих процесів моніторингу, система здатна ефективно обробляти великі обсяги транзакцій, забезпечуючи при цьому високий рівень захисту конфіденційної інформації. Дотримання міжнародних стандартів безпеки PCI DSS та регулярний аудит дозволяють мінімізувати ризики витоку даних і підвищують довіру як користувачів, так і потенційних партнерів.

РОЗДІЛ III

ПІДКЛЮЧЕННЯ ТА ІНТЕГРАЦІЯ ПАКЕТУ ДО РІЗНИХ E-COMMERCE ПЛАТФОРМ

3.1 Створення Node.js-пакету для обробки платежів

Під час розробки ПЗ, було визначено потребу у створенні модульного SDK для інтеграції з платформою Wayforpay (як приклад, на основі якого можна швидко розробити інтеграції з іншими платформами). Пакет wayforpay-ts-integration реалізовано на мові TypeScript та, по завершенню розробки, викладено в вигляді відкритого коду на Github [ДОДАТОК Б] та в якості npm пакету [ДОДАТОК В]. Даний SDK дозволяє спростити процес обміну даними з API Wayforpay для обробки платежів, створення платіжних форм, отримання списку транзакцій та роботи з регулярними платежами.

Функціональні вимоги

Обробка платежів: реалізація створення платежів із генерацією HTML-форми, яку користувач може автоматично виконати для оплати. Перевірка статусу платежів: надання можливості перевірки статусу платежу по orderReference. Отримання списку транзакцій: забезпечення зручного доступу до переліку транзакцій магазину за встановлений період (до 31 дня). Регулярні платежі: підтримка запитів для роботи із регулярними платежами (статус, пауза, відновлення, видалення). Додаткові функції: генерація посилань на платіжну сторінку, реалізація платіжних віджетів, верифікація карт та інші можливості, що відображені в README-проекті.

Нефункціональні вимоги

Безпека: критичне використання секретних ключів та мерчант-даних лише на серверній стороні. Масштабованість та підтримка розширень: модульна архітектура, що дозволяє додавати нові функції без значних змін до базової структури. Документованість: створення докладного README файлу та коментарів у коді для полегшення інтеграції з SDK сторонніми розробниками.

Планування розробки

1. На основі визначених вимог було розроблено наступний план:
2. Визначення архітектури рішення та інтерфейсів.
3. Розробка модулів для основних функцій: створення платежів, отримання транзакцій, регулярні платежі.
4. Інтеграція механізмів генерації HTML форм із унікальним підписом для безпечної обробки платежів.
5. Проведення тестування (unit- та інтеграційного) для забезпечення стабільності роботи.
6. Оформлення документації та прикладів використання.

Вибір технологій

Для розробки SDK обрано TypeScript, оскільки дана мова дозволяє забезпечити чітке визначення інтерфейсів (TCartElement, TProduct, TUserCartElement), що сприяє кращій підтримці та зрозумілості коду. Автоматичну перевірку типів під час компіляції, що знижує ризик помилок на етапі виконання. Легку інтеграцію з серверними рішеннями Node.js.

Пакет організовано як модульну бібліотеку, що містить головний клас Wayforpay, який інкапсулює основну логіку для обробки платежів, роботи з транзакціями та регулярними платежами.

1. purchase – генерує HTML-форму платежу.
2. getTransactions – повертає список транзакцій магазину.
3. checkStatus – перевіряє статус конкретного платежу.
4. regularPayment – працює з регулярними платежами (з можливістю виконання запитів типів STATUS, SUSPEND, RESUME, REMOVE).

Логіка роботи розділена на кілька модулів:

Клієнтський модуль (інтеграція): сервер викликає методи SDK, які повертають HTML-форму або дані про транзакції.

Модуль генерації форми: відповідає за формування HTML-коду із вбудованим скриптом для автоматичного виконання. Генерація форми відбувається з використанням даних платежу та унікального підпису.

Модуль обробки транзакцій: виконує операції з обробки відповіді від Wayforpay, аналіз статусу платежу та інших параметрів.

Генерація HTML-форми для платежу

Ключовим елементом пакету є метод `purchase`, який приймає кошик покупок, що представлено масивом об'єктів типу `TCartElement`, та об'єкт з налаштуваннями платежу (наприклад, домен, валюта, спосіб доставки). Усередині методу обчислюється унікальний підпис на основі даних замовлення та секретних ключів мерчанта, формується HTML-код, у якому вбудовано інструкції для автоматичного відправлення форми на сторінку Wayforpay.

Обов'язковим є те, що дані мерчанта (логін та секретний ключ) передаються лише із серверної сторони, що сприяє захисту конфіденційної інформації. Отримується список транзакцій та перевірка статусу платежів.

Для спрощення обробки даних взаємодія з API Wayforpay була реалізована через спеціалізовані методи:

1. `getTransactions`: формує запит до API для отримання списку транзакцій за заданий період (максимум 31 день).
2. `checkStatus`: перевіряє поточний статус платежу, використовуючи `orderReference`, що дозволяє здійснювати моніторинг та аналіз транзакцій у режимі реального часу.

Реалізація підтримки регулярних платежів включає:

Індивідуальний підхід для кожного магазину: інтеграція цього функціоналу потребує узгодження з Wayforpay (отримання MerchantPassword). Метод `regularPayment`, який підтримує виконання різних операцій з регулярними платежами (наприклад, отримання статусу, пауза, відновлення та видалення регулярного платежу). Використання окремих типів запитів для забезпечення коректної обробки операцій.

Документація до пакету містить декілька прикладів використання:

POST-запит: створення маршруту на сервері, який генерує HTML-форму для оплати на основі даних кошика покупця.

GET-запит: використання `Next.js` для генерації сторінки, що містить платіжну форму, що дозволяє уникнути додаткових запитів та перенаправити користувача безпосередньо на сторінку оплати.

Клієнтський компонент: React-компонент, який отримує форму з сервера, вставляє її у DOM та автоматично виконує для перенаправлення користувача на сторінку Wayforpay.

Кожен із прикладів демонструє практичну реалізацію логіки SDK та підкреслює важливість безпечного використання (операції з мерчант-даними виконуються виключно на сервері).

Підхід до тестування

Для забезпечення стабільності та коректності роботи SDK розробники використовували як `unit-тести`, так і інтеграційне тестування:

Unit-тестування: перевірка окремих методів класу Wayforpay (генерація підпису, валідація вхідних даних, формування HTML-форми).

Інтеграційне тестування: симуляція запитів до реального API Wayforpay (з використанням тестових ключів та мерчант-даних) для перевірки процесу платежів та роботи регулярних платежів.

Пакет інтегровано із загальним платіжним шлюзом, що використовує сучасні підходи до розробки API:

Конфігурація через змінні середовища: секретні дані (MERCHANT_LOGIN, MERCHANT_SECRET_KEY) зберігаються у файлах конфігурації та передаються через змінні середовища.

Безпечна комунікація: використання HTTPS та реалізація захищених HTTP POST запитів для взаємодії з API Wayforpay.

Модульність: можливість підключення пакету як окремого модуля в більшу систему забезпечує гнучкість та масштабованість рішення.

Оформлення документації

Окрім вихідного коду, було створено детальну документацію, що включає:

README.md: містить опис пакету, список можливостей, приклади використання та посилання на офіційну документацію Wayforpay. Приклади інтеграції: готові приклади для Node.js, React та Next.js для швидкого старту роботи із SDK. Коментарі у коді: використання докладних коментарів для пояснення логіки роботи окремих модулів.

Підтримка користувачів

Для спрощення інтеграції і подальшої підтримки користувачів було реалізовано:

1. Вказівки щодо безпеки: чітке зауваження про те, що методи пакету повинні використовуватись лише на серверній стороні.
2. Інструкції зі створення магазину у Wayforpay: пояснення процедури авторизації, створення магазину та отримання мерчант-даних.
3. Контактна інформація: рекомендації звертатись до підтримки Wayforpay для вирішення індивідуальних випадків, зокрема для роботи з регулярними платежами.

Підсумок та перспективи розвитку

Пакет `wayforpay-ts-integration` значно спрощує процес інтеграції з платіжною системою Wayforpay, забезпечуючи чистий та модульний інтерфейс для розробників. Розробка SDK відповідала сучасним вимогам безпеки та масштабованості, що є критичними для систем електронних платежів.

Перспективи подальшого розвитку включають розширення функціоналу для підтримки нових сервісів Wayforpay (наприклад, онлайн кредитування, оплата таксі тощо), автоматизацію тестування з використанням сучасних технологій CI/CD, оптимізацію продуктивності та розширення можливостей кастомізації платіжної форми.

Отже, створення пакету `wayforpay-ts-integration` є вагомим внеском у розробку інтегрованих систем електронних платежів та демонструє високий рівень технічної реалізації та документаційного супроводу.

3.2. Створення демо-рішення для Shopify

Створення демо-рішення для Shopify стане важливим етапом демонстрації практичної реалізації розробленого ПЗ та його здатності інтегруватися з однією з найпопулярніших e-commerce платформ. Це рішення дозволяє не лише перевірити технічну сумісність між системами, а й продемонструвати переваги використання мультиплатформенного підходу для обробки платежів у режимі реального часу.

Опис API Shopify та вимог інтеграції

Shopify пропонує розвинений набір API, серед яких основними є REST API та GraphQL API. Завдяки цим інтерфейсам, розробники отримують можливість:

Керувати замовленнями та транзакціями. API дозволяють створювати, оновлювати та відстежувати статуси замовлень, що є критично важливим для платіжних процесів.

Авторизація та безпека. Використання OAuth 2.0 забезпечує безпечну авторизацію застосунку, що дає можливість отримувати доступ до даних магазину.

Webhook інтеграція. Shopify підтримує вебхуки, які дозволяють в режимі реального часу отримувати інформацію про події (наприклад, зміну статусу платежу), що сприяє синхронізації даних між платіжним пакетом та системою замовлень.

Для інтеграції ПЗ необхідно дотримуватися вимог Shopify щодо безпеки, збереження даних і обробки транзакцій. Це включає як реалізацію механізмів автентифікації, так і забезпечення належної взаємодії між застосунком і платформою.

Розробка та публікація застосунку для Shopify

Демонстраційне рішення буде розгорнуто як незалежне ПЗ, яке, незважаючи на відсутність прямої інтеграції з платформою Shopify, дозволяє взаємодіяти з нею через API токен, наданий користувачем.

Основні компоненти

База даних API токенів. Власник Shopify магазину вводить API токен, який має права на доступ до інформації про продукти та замовлення, безпосередньо в базу даних ПЗ. Збереження токенів в окремій системі дозволяє централізовано управляти доступом і забезпечувати правильну авторизацію при зверненні до Shopify. [Рис. 3.1]

Інтеграційний скрипт. У магазині Shopify додається спеціально розроблений скрипт, який активується автоматично після оформлення замовлення клієнтом. Цей скрипт ініціює GET запит до REST API застосунку, забезпечуючи своєчасну обробку інформації для подальших платіжних операцій. [Рис. 3.3, 3.4, Додаток А]

Обробка запиту та взаємодія з Shopify. Після отримання GET запиту, застосунок звертається до Shopify через REST API із використанням ідентифікатора замовлення. Це дозволяє отримати детальну інформацію про замовлення, необхідну для підтвердження транзакції та її наступної обробки.

Формування платіжної сторінки. Отримані дані використовуються для створення індивідуальної платіжної сторінки. Для цього застосовується розроблений платіжний пакет wayforpay-ts-integration [Розділ III, 3.1], який забезпечує зручну, швидку і безпечну обробку платіжних операцій.

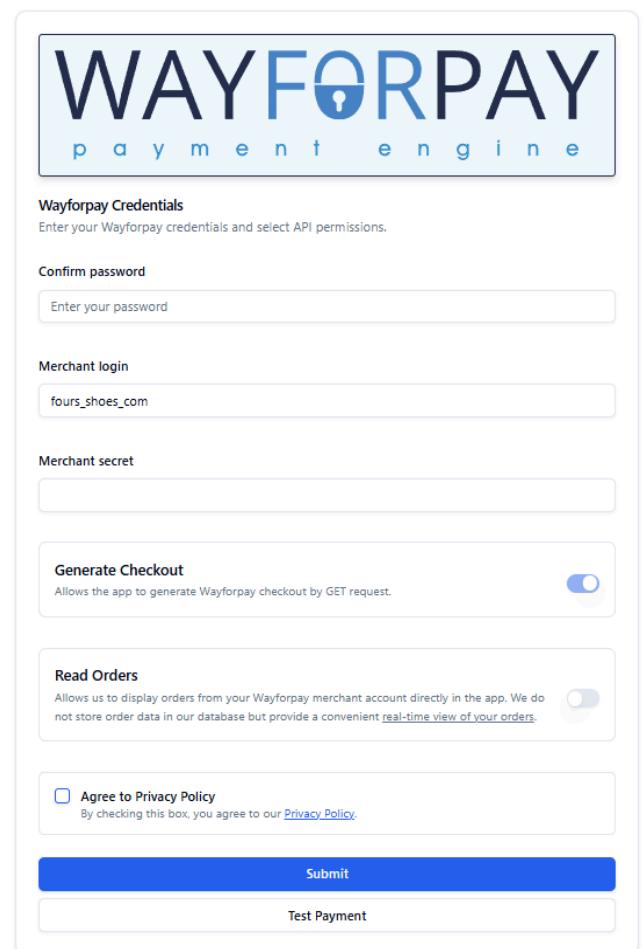
The image shows a web form titled 'WAYFORPAY payment engine'. Below the title, it says 'Wayforpay Credentials' and 'Enter your Wayforpay credentials and select API permissions.' The form contains several input fields: 'Confirm password' with a placeholder 'Enter your password', 'Merchant login' with the value 'fours_shoes_com', and 'Merchant secret'. There are two toggle switches: 'Generate Checkout' (which is turned on) and 'Read Orders' (which is turned off). Below the toggles is a checkbox for 'Agree to Privacy Policy' with a link to the 'Privacy Policy'. At the bottom, there is a blue 'Submit' button and a 'Test Payment' button.

Рис. 3.1 Форма для введення даних для доступу до Shopify магазину

Переваги реалізації

Модульність та незалежність. Окремий застосунок дозволяє легко масштабувати ПЗ, адаптувати його під конкретні потреби та оперативно впроваджувати нові функції без порушення роботи магазину на Shopify.

Безпека. Застосування API токенів для автентифікації знижує ризик несанкціонованого доступу до даних, адже власник магазину контролює, які дані можуть бути доступні зовнішній системі.

Автоматизація процесів. Інтеграційний скрипт забезпечує безперервну взаємодію між Shopify та застосунком, що мінімізує людський фактор і знижує можливості виникнення помилок під час обробки замовлень.

The image displays a configuration interface for a payment gateway, organized into several sections:

- Currency Settings:** A section titled "Currency Settings" with the subtitle "Configure your preferred currency options." It features a "Base Currency" dropdown menu currently set to "UAH - Ukrainian Hryvna" and a "Save" button.
- Language Settings:** A section titled "Language Settings" with the subtitle "Payment page language." It includes a "Checkout Language" dropdown menu set to "Ukrainian" and a "Save" button.
- Callback:** A section titled "Callback" containing three fields:
 - Return Url:** A dropdown menu set to "POST" and a text input field containing "https://gateways.agile-alpaca.com/api/public/v1/post-redirect". Below it, a note states: "The POST request with the payment result."
 - Return Url:** A dropdown menu set to "GET" and a text input field containing "https://fours-shoes.com/". Below it, a note states: "The page to which the user will be redirected after payment."
 - Service Url:** A text input field containing "https://api.domain/order". Below it, a note states: "URL, to which the system has to send a response with the payment result directly to the merchant."Each field has a "Save" button.
- Payment Systems (Top):** A section titled "Payment Systems" with the subtitle "The list of payment systems available for client in case of selection of payment method at the payment page." It lists various payment methods with checkboxes:
 - ☒ Credit/Debit Card
 - ☒ Google Pay
 - ☒ Apple Pay
 - ☒ Privat24
 - ☐ LP Terminal
 - ☐ Delayed Payment
 - ☐ Bank Cash
 - ☐ QR Code
 - ☐ MasterPass
 - ☐ Visa Checkout
 - ☐ Bot Payment
 - ☐ Pay Parts
 - ☐ Pay Parts Mono
 - ☐ Pay Parts Privat
 - ☐ Pay Parts Abank
 - ☐ Instant Abank
 - ☐ Globus Plus
 - ☐ Onus Installment
 - ☐ Pay Parts OTP
 - ☐ Pay Parts SportA "Save" button is located at the bottom right.
- Payment Systems (Bottom):** A second section titled "Payment Systems" with the same subtitle. It lists additional payment methods with checkboxes:
 - ☐ Nova Poshta (Нова Пошта)
 - ☐ Meest Express (Міст експрес)
 - ☐ Ukrposhta (Укрпошта)
 - ☐ InternationalA "Save" button is located at the bottom right.

Рис. 3.2 Можливості для налаштування оплати

Налаштування інтеграції та тестове середовище

Для тестування застосунку розгортається локальний сервер, який емуляцією роботи реальних API Shopify та платіжних систем. Використання

Mock API дозволяє симулювати як позитивні, так і негативні сценарії оплати, що сприяє виявленню помилок на ранніх етапах розробки.

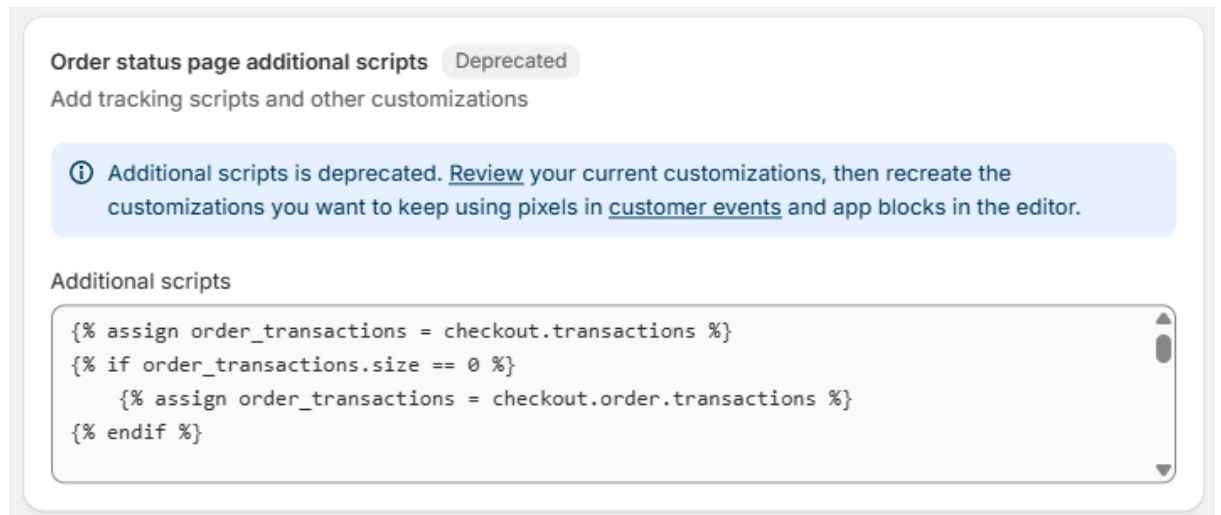


Рис. 3.3 Додатковий скрипт, який виконується після оформлення замовлення



Рис. 3.4 Повний код скрипту, який виконується після оформлення замовлення на боці Shopify

Підсумок

Розробка демо-рішення для Shopify є важливим етапом, що дозволяє наочно продемонструвати можливості розробленого платіжного пакету в реальному середовищі. Завдяки інтеграції з API Shopify, використанню сучасних протоколів безпеки та належній конфігурації Node.js-пакету, застосунок забезпечує безперебійну обробку платежів та синхронізацію даних між системами. Публікація застосунку, незалежно від того, чи буде він приватним або публічним, відкриває можливості для масштабування рішення, його адаптації до різних потреб бізнесу та підвищення рівня довіри користувачів до платіжних операцій, що є критично важливим для сучасних інтернет-магазинів.

3.3. Приклад інтеграції з WooCommerce (WordPress)

Інтеграція ПЗ з WooCommerce – популярним плагіном для WordPress, що забезпечує потужну функціональність інтернет-магазину – є ще одним напрямком розробки мультиплатформенних рішень. Такий підхід дозволяє адаптувати ПЗ під специфічні вимоги бізнесу, використовуючи гнучку екосистему WordPress та можливості налаштування через hooks і фільтри. Нижче розглядається приклад інтеграції, що демонструє основні етапи впровадження та ключові аспекти реалізації платіжного шлюзу в WooCommerce.

Розробка кастомного плагіну для WooCommerce

Архітектура інтеграції. Інтеграція здійснюється шляхом розробки кастомного плагіну, який виступає мостом між WooCommerce та серверною частиною платіжного пакету, реалізованою на Node.js. Основні функції плагіну включають:

Реєстрацію нового платіжного методу. Плагін додає власний шлюз до списку доступних способів оплати в налаштуваннях WooCommerce.

Інтерфейс користувача: Забезпечення користувача зручною формою для введення платіжних даних безпосередньо на сторінці оформлення замовлення.

Комунікацію із серверною частиною. Використання REST API для передачі даних про замовлення на сервер Node.js-пакету та отримання статусу транзакції через callback-запити.

Обробку вебхуків. Реєстрація відповідних хуків для прийому та обробки асинхронних повідомлень від платіжної системи, що дозволяє оновлювати статуси замовлень у WooCommerce.

Створення плагіну

Розробка базової структури плагіну згідно з рекомендаціями WordPress. Основний файл плагіну містить заголовок, що описує його призначення, версію та авторство.

Ініціалізація плагіну за допомогою функцій активації та деактивації, що дозволяє налаштовувати базову конфігурацію та створювати необхідні таблиці у базі даних, якщо це потрібно.

Реєстрація платіжного шлюзу

Використовуючи WooCommerce hooks, плагін реєструє новий платіжний метод. Для цього створюється клас, що реалізує всі необхідні методи, наприклад, ініціалізацію форми оплати, перевірку валідності даних та оновлення статусів замовлення.

Завдяки використанню фільтрів WooCommerce, плагін може адаптувати зовнішній вигляд форми оплати під загальний дизайн магазину, забезпечуючи інтегровану і послідовну взаємодію з користувачем.

Взаємодія з платіжним пакетом

Після того, як користувач заповнив платіжну форму, дані замовлення відправляються з плагіну на серверну частину платіжного пакету через REST API. Тут відбувається ініціалізація транзакції: формується запит із необхідними

параметрами, такими як сума платежу, валютні дані, ідентифікатор замовлення тощо.

Сервер Node.js-пакету обробляє запит, взаємодіє із зовнішніми платіжними системами, проводить валідацію та повертає результат операції. Результати можуть містити статус транзакції, ідентифікатор платежу та додаткову інформацію, яка необхідна для подальшої обробки замовлення.

Обробка вебхуків та оновлення статусів

Після завершення операції платіжна система надсилає callback (вебхук) із статусом транзакції. Плагін у WooCommerce за допомогою зареєстрованих хуків приймає цей сигнал і відповідно оновлює статус замовлення. Наприклад, у разі успішної оплати замовлення автоматично позначається як «оплачено».

Завдяки цій інтеграції адміністратор магазину отримує актуальну інформацію про платежі, а клієнт – підтвердження успішної транзакції без необхідності додаткових дій.

Переваги інтеграції з WooCommerce

Інтеграція з WooCommerce дозволяє використати повний потенціал платформи WordPress.

Гнучкість налаштувань. Використання hooks та фільтрів WordPress дає можливість адаптувати функціонал платіжного шлюзу до конкретних потреб магазину. Розробники можуть змінювати поведінку плагіну без зміни базового коду, що полегшує підтримку та оновлення системи.

Зручність для користувачів. Інтегрований спосіб оплати, що відбувається безпосередньо на сайті, покращує користувацький досвід, знижує кількість перенаправлень і сприяє підвищенню конверсії.

Можливість масштабування. Використання REST API для комунікації між плагіном та серверною частиною платіжного пакету забезпечує легкість розширення функціоналу. Наприклад, у майбутньому можна додати підтримку

нових методів оплати або інтеграцію з іншими системами без значних змін в архітектурі плагіну.

3.4. Тестування мультиплатформенності та узгодження інтерфейсів

Тестування мультиплатформенності та узгодження інтерфейсів важливий етап розробки, бо воно дозволить перевірити, як ПЗ взаємодіє з різними e-commerce платформами та забезпечує єдиний досвід користувача незалежно від обраної системи. Цей етап включає комплексне тестування як функціональності, так і взаємодії між інтерфейсами, гарантуючи коректну роботу рішення в умовах різних технічних середовищ.

Мета тестування мультиплатформенності

Основною метою є перевірка сумісності платіжного пакету з основними платформами, Shopify, WooCommerce, Wix та іншими. Це дозволяє:

Переконатися в універсальності рішення. Тестування має підтвердити, що пакет коректно обробляє транзакції незалежно від специфіки API чи механізмів інтеграції кожної платформи.

Виявити можливі розбіжності в інтерфейсах. Узгодження елементів взаємодії (наприклад, форми оплати, повідомлення про помилки, оновлення статусів замовлень) сприяє створенню єдиного досвіду для кінцевого користувача.

Провести аналіз продуктивності. Можна оцінити, як система справляється з підвищеним навантаженням на кожній платформі, і чи не виникають проблеми із затримками або невідповідностями у передачі даних.

Підходи до тестування інтерфейсів

Для забезпечення узгодженості інтерфейсів застосовується декілька методів тестування:

Функціональні тести. Перевірка роботи основних функцій платіжного пакету в умовах різних платформ. Це включає тестування створення,

підтвердження та повернення платежів, а також обробку callback-запитів. Функціональні тести дозволяють ідентифікувати, чи відбувається правильне оновлення статусів замовлень у системах електронної комерції.

UI/UX тестування. Аналіз користувацького досвіду при оформленні платежів, перевірка відображення елементів форми оплати, узгодженість візуального оформлення та поведінки інтерфейсу при різних сценаріях (успішна оплата, помилка, затримка відповіді).

Тестування сумісності. Перевірка роботи рішення на різних браузерях, мобільних пристроях та операційних системах, щоб забезпечити безперебійну роботу незалежно від того, як і де користувач здійснює оплату.

Стрес-тестування та тестування навантаження. Вимірювання продуктивності системи при великій кількості одночасних запитів, що дозволяє оцінити, наскільки платіжний пакет здатен обробляти пікові навантаження, які можуть бути типові для популярних інтернет-магазинів.

Узгодження інтерфейсів та робота з API

Оскільки інтеграція з кожною платформою може мати свої особливості, важливо забезпечити.

Щоб API платіжного пакету повністю відповідало специфікаціям кожної платформи:

Уніфікація даних. Всі дані, що надходять із застосунків, повинні бути стандартизовані. Це дозволяє уникнути плутанини при передачі інформації між різними системами.

Адаптивність. Платіжний пакет має містити механізми для адаптації форматів даних відповідно до вимог кожної платформи, наприклад, обробка специфічних форматів замовлень або валют.

Обробка помилок. Інтерфейс повинен надавати зрозумілі повідомлення про помилки як для кінцевих користувачів, так і для адміністраторів систем, що полегшує діагностику проблем при інтеграції.

Підсумок

Комплексне тестування мультиплатформенності та узгодження інтерфейсів дозволяє забезпечити єдиний, послідовний досвід для користувачів незалежно від обраної e-commerce платформи. Завдяки функціональним, UI/UX та навантажувальним тестам розробники можуть виявити та усунути потенційні недоліки інтеграції, що сприяє підвищенню надійності, стабільності та ефективності платіжного пакету. Таке тестування є ключовим для забезпечення конкурентоспроможності рішення на сучасному ринку електронної комерції, де швидкість, безпека та зручність взаємодії є визначальними факторами успіху.

ВИСНОВКИ

Підсумки виконаної роботи

Розроблена дипломна робота дозволила комплексно підійти до створення мультиплатформенного інструменту для інтеграції платіжних систем у сучасні інтернет-магазини. Проведений аналіз існуючих рішень, зокрема WayForPay, LiqPay та Monobank, виявив низку технічних, функціональних та організаційних недоліків, що обмежують ефективність їх використання у глобальних e-commerce платформах, зокрема Shopify, WooCommerce та інших. Отримані дані стали ґрунтом для формування вимог до розробки власного платіжного пакету на базі Node.js та в підсумку реалізації прототипу ПЗ.

Розроблене ПЗ характеризується високою модульністю, масштабованістю та безпекою, що досягається завдяки використанню сучасних технологій TypeScript, Express, ESLint та автоматизованого тестування через Jest. Інноваційність рішення полягає у створенні єдиного інтерфейсу, який забезпечує інтеграцію з різними e-commerce платформами, в прототипі реалізован конкретно Shopify, а також у впровадженні механізмів обробки помилок і повторних спроб, що гарантують надійну роботу ПЗ в умовах високого навантаження.

Ключовими результатами виконаної роботи є:

Комплексний аналіз ринку. Визначення основних переваг і недоліків існуючих платіжних систем для України, що дозволило сформулювати чітке бачення потреб сучасних інтернет-магазинів.

Розробка платіжного пакету. Створено Node.js-пакет, який забезпечує інтеграцію з популярними e-commerce платформами, підтримуючи функції ініціалізації платежу, обробки вебхуків, повернення коштів та забезпечення високого рівня безпеки.

Оптимізація та масштабованість. Впровадження механізмів кешування, балансування навантаження, а також автоматизованого тестування через CI/CD сприяють стабільній роботі ПЗ навіть при високих обсягах транзакцій.

Демонстрація практичної реалізації. Розроблено демо-рішення для Shopify, що наочно демонструє можливості ПЗ та її переваги перед існуючими аналогами.

Отже, виконана робота дозволяє стверджувати, що розроблений платіжний пакет відповідає поставленим цілям та завданням, забезпечуючи високий рівень продуктивності, безпеки та зручності інтеграції. Реалізація даного рішення відкриває можливості для подальшого вдосконалення функціоналу, розширення підтримки додаткових платіжних платформ та адаптації до потреб як локальних, так і міжнародних ринків електронної комерції.

Перспективи подальших досліджень та вдосконалення

Розроблена система є значним кроком у напрямку створення універсального платіжного інструменту для інтеграції з популярними e-commerce платформами. Проте, як і будь-яке рішення, вона відкриває широкі можливості для подальших досліджень і вдосконалення. Нижче наведено основні напрямки, які можуть бути розглянуті в подальшій роботі, в випадку створення повноцінної команди для розробки та отримання хоча б мінімального фінансування для продовження розробки.

Розширення функціональних можливостей

Подальший розвиток ПЗ може включати інтеграцію додаткових платіжних платформ, як локальних, так і міжнародних. Серед можливих напрямків:

Інтеграція з глобальними платіжними системами. Впровадження підтримки Stripe, PayPal та інших міжнародних провайдерів дозволить забезпечити єдиний інтерфейс для прийому платежів незалежно від географії клієнтів. Це вихід на міжнародний ринок.

Підтримка нових платіжних методів. Реалізація можливості оплати через мобільні гаманці, криптовалюти чи технології blockchain відкриє додаткові

канали для транзакцій, що може бути особливо актуальним в умовах швидкого розвитку фінтех-сектору.

Покращення продуктивності та масштабованості

Надалі слід звернути увагу на оптимізацію продуктивності ПЗ та її масштабованість:

Мікросервісна архітектура. Перехід до мікросервісного підходу дозволить розподілити навантаження між окремими сервісами, спрощуючи масштабування та забезпечуючи високу доступність системи. В даний момент ПЗ реалізовано як простий Express + Next.js додаток.

Контейнеризація та оркестрація. Використання Docker та Kubernetes сприятиме більш ефективному розгортанню та управлінню сервісами, забезпечуючи стабільну роботу при збільшенні кількості транзакцій.

Інтелектуальне кешування та оптимізація бази даних. Розробка алгоритмів для динамічного кешування та використання сучасних рішень для управління базами даних допоможе скоротити час обробки запитів і підвищити загальну продуктивність.

Забезпечення високого рівня безпеки

З огляду на критичну важливість безпеки фінансових операцій, перспективним напрямком є подальше вдосконалення механізмів захисту:

Розширене застосування криптографії. Впровадження сучасних алгоритмів шифрування та методів захисту даних може забезпечити ще більший рівень безпеки, що відповідає сучасним стандартам.

Функції виявлення шахрайства. Інтеграція систем штучного інтелекту для аналізу транзакцій і виявлення аномалій допоможе швидко ідентифікувати потенційні загрози, зменшуючи ризики шахрайства.

Регулярні аудити безпеки. Організація систематичних перевірок відповідності стандартам, таким як PCI DSS, сприятиме постійному вдосконаленню системи та підтриманню високих стандартів безпеки.

Покращення користувацького досвіду

Враховуючи важливість зручності користування системою, доцільно звернути увагу на удосконалення інтерфейсів:

Інтеграція з мобільними застосунками. Розробка адаптивних рішень для мобільних платформ дозволить забезпечити безшовний досвід оплати на смартфонах і планшетах.

Персоналізація інтерфейсу. Використання сучасних підходів до UX/UI-дизайну сприятиме створенню інтуїтивно зрозумілого та зручного інтерфейсу для кінцевих користувачів, що позитивно вплине на конверсію та задоволеність клієнтів.

Розширення аналітичних можливостей. Інтеграція систем аналітики для збору та аналізу даних про поведінку користувачів допоможе оптимізувати процес оформлення замовлень та оперативно реагувати на зміни в поведінці аудиторії.

Подальші дослідження економічного та юридичного аспектів

Окрім технічного удосконалення, важливо продовжувати дослідження економічних і юридичних аспектів впровадження платіжних платформ:

Економічна ефективність. Аналіз економічних показників впровадження мультиплатформенного рішення дозволить оцінити економію витрат для бізнесу, враховуючи зниження транзакційних зборів та оптимізацію операційних процесів.

Юридичні питання та відповідність нормам. Вивчення вимог регуляторних органів як на національному, так і на міжнародному рівні

сприятиме розробці рішень, що відповідають сучасним стандартам, зокрема в сфері захисту персональних даних і протидії відмиванню грошей.

Подальші дослідження та вдосконалення платіжного пакету відкривають широкі можливості для підвищення його конкурентоспроможності та адаптивності на ринку електронної комерції. Розширення функціональних можливостей, оптимізація продуктивності, посилення заходів безпеки та покращення користувацького досвіду є ключовими напрямками, які дозволять створити універсальне рішення, здатне задовольнити потреби як локальних, так і міжнародних інтернет-магазинів. Реалізація запропонованих перспектив сприятиме не лише технічному вдосконаленню, а й підвищенню економічної ефективності та юридичної відповідності, що є критично важливими факторами для стабільного розвитку сучасних фінансових технологій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. <https://freelancehunt.com/en/project/pidklyuchennya-oplati-wayforpay-na-sayt-shopify/1081924.html>
2. <https://wallnut.digital/shops/shopify/payments/liqpay>

3. <https://trademinister.com.ua/collections/all>
4. <https://community.shopify.com/c/payments-shipping-and/how-can-i-set-up-my-payment-method-in-ukraine>

ДОДАТОК А

Приклад скрипту для перенаправлення користувача на сторінку оплати після оформлення замовлення.

```
{% assign order_transactions = checkout.transactions %}  
{% if order_transactions.size == 0 %}
```

```

    {% assign order_transactions = checkout.order.transactions %}

{% endif %}

{% assign size = order_transactions.size | minus: 1 %}

{% if order_transactions[size].gateway == 'Wayforpay' %}

<script type='text/javascript'>

    const urlParams = new URLSearchParams(window.location.search);
    const isPaid = urlParams.get('paid') === 'true';

    if (!isPaid) {
        const h2Elements = document.querySelectorAll('h2');
        h2Elements.forEach((h2) => {
            h2.textContent = "Redirecting to Secure Payment...";
        });

        const returnUrl = window.location.href;

        function handlePay() {
            window.location.href =
`https://gateways.agile-alpaca.com/api/public/v1/wayforpay/8i9izd-2m.myshopify.com/byOrder?orderId={{ checkout.order_id }}&returnUrl=${returnUrl}`;
        }
    }

```

```
        {% if checkout.order.total_price %}  
        handlePay();  
        {% else %}  
        window.location.reload(true);  
        {% endif %}  
    } else {  
        console.log('Оплата пройшла успішно.');    }  
  
</script>  
{% endif %}
```

ДОДАТОК Б

<https://github.com/Wlad1slav/wayforpay-ts-integration/tree/main>

ДОДАТОК В

<https://www.npmjs.com/package/wayforpay-ts-integration>