УДК 004.65 DOI 10.18372/2073-4751.87.21696

Хрустовський А.,
orcid.org/0009-0001-0497-4836,
e-mail: 7496877@stud.kai.edu.ua,
Савченко А., д.т.н.,
orcid.org/0000-0001-8205-8852,
e-mail: alina.savchenko@npp.kai.edu.ua,
Дехтярук М.,
orcid.org/0000-0002-8709-5967,
e-mail: mykola.dekhtiaruk@npp.kai.edu.ua,
Райчев І., к.т.н.,
orcid.org/0000-0003-4058-1236,
e-mail: ihor.raichev@npp.kai.edu.ua

ВИКОРИСТАННЯ ХМАРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ АНАЛІЗУ ТА ОБРОБКИ ПЛАТІЖНИХ ДАНИХ У СИСТЕМАХ ОНЛАЙН-ЕКВАЙРИНГУ

Національний університет «Київський авіаційний інститут»

Вступ

Впровадження систем онлайн-платежів суб'єктами малого підприємництва або благодійними організаціями часто супроводжується низкою технічних та операційних ризиків. Незважаючи на наявність базової інфраструктури (веб-ресурсу та банківського рахунку), на етапі експлуатації нерідко виявляються критичні недоліки: відсутність механізмів логування транзакцій та аналітичного обліку. Зокрема, в умовах пікових навантажень спостерігається втрата стійкості системи, що призводить до відмови в обслуговуванні запитів та виникнення невизначеного статусу фінансових операцій. Наприклад, малий бізнес або благодійна організація вирішує запустити онлайн-прийом платежів. На перший погляд, завдання нескладне: є сайт, є банківський рахунок, є бажання отримувати кошти через інтернет. Але вже через тиждень після запуску виявляється, що платіжні дані нікуди не записуються, аналітики немає, а в день акції зі збору коштів, коли сервер отримує кілька сотень запитів на годину, система перестає відповідати. Кошти від донорів або покупців зависають в статусі «невідомо».

Це типова ситуація для організацій без власного технічного відділу. Сайт здебільшого зробив сторонній підрядник і тримається на звичайному комерційному хостингу — WordPress, Apache тощо. Переробляти його з нуля ніхто не буде: дорого, і є реальний ризик зламати те, що вже нормально працює. Потрібне рішення, яке додає платіжний канал поверх — без змін у решті.

Онлайн-еквайринг - це комплекс послуг банку або платіжного провайдера, що забезпечує приймання безготівкових платежів через інтернет. Технічно інтеграція реалізується через REST API платіжного сервісу та webhook. Кожна транзакція надходить як подія, що містить суму, статус, унікальний ідентифікатор рахунку і часову позначку.

Постановка проблеми

Проблема виникає при нерівномірному навантаженні. Звичайні дні — кілька платежів на день. Запуск нової кампанії — кілька сотень платежів за годину. Класичний сервер з постійно зарезервованою потужністю обходиться дорого у формі оренди (якщо купується потужність «про запас»), або впадає в простій (якщо придбана мінімальна потужність). Або організація витрачає час

і гроші на налаштування Kubernetes (система управління контейнерами) для автоматичного масштабування, що для малого бізнесу часто недоступно.

Додатково проблема набуває гостроти для організацій, які не мають власного технічного відділу і не можуть дозволити собі витрати на VPS-інфраструктуру з постійною резервованою потужністю. Потреба в масштабованому платіжному бекенді без значних початкових інвестицій є актуальним практичним завданням, яке потребує архітектурного вирішення засобами сучасних хмарних технологій.

Аналіз останніх досліджень та публікацій

Шойнер та Лайтнер у 2020 році провели мультивокальний огляд 112 робіт із продуктивності FaaS-платформ, охоплюючи 51 академічну та 61 промислову публікацію. Вони показали, що більшість досліджень фокусуються на мікробенчмарках AWS Lambda і вимірюванні cold start overhead, тоді як інтеграція FaaS із зовнішніми сервісами — API-шлюзами та реляційними сховищами — залишається малодослідженою [1]. Саме цей аспект реалізує архітектура, описана в даній статті.

Байя у 2025 році дослідив застосування безсерверних платформ (AWS Lambda, Azure Functions, Google Cloud Functions) у фінтех-рішеннях, зокрема для обробки платежів у реальному часі та виявлення шахрайства. Робота підтверджує, що подієво-орієнтована модель і оплата за фактичне використання роблять FaaS економічно обґрунтованим вибором для фінансових застосунків з нерівномірним трафіком. При цьому автор окреслює типові виклики: cold start latency і відповідність вимогам PCI DSS [2].

Дослідження, опубліковане у European Academic Journal у 2025 році, описує подієво-орієнтовану архітектуру для управління розподіленими транзакціями в системах із нерівномірним трафіком. Автори демонструють, що компенсаційні транзакції та ідемпотентні операції забезпечують консистентність даних у stateless FaaS-середовищі. Ці висновки підтверджують обрану в даній статті модель обробки webhook-подій [3].

Існуючі наукові дослідження у галузі платіжних систем зосереджуються переважно на аналізі безпеки криптографічних протоколів, поведінці користувачів у мобільних платежах та загальних моделях обробки великих масивів даних (Big Data).

Роботи у галузі безсерверних обчислень, зокрема технічний звіт Каліфорнійського університету (Берклі) [4], систематизують переваги FaaS-підходу (Functions as a Service) і підтверджують його ефективність для навантажень з нерівномірним трафіком. Офіційні документації платформ Cloudflare Pages [5-6], Supabase [7-9] та plata by mono [10-11] описують архітектурні можливості цих інструментів і слугують першоджерелами для практичного впровадження. Разом із тим, конкретні аспекти інтеграції вітчизняного платіжного API з безсерверною хмарною архітектурою для потреб НКО та малого бізнесу у науковій літературі практично не висвітлені.

Питання побудови малобюджетної безсерверної архітектури (serverless architecture — архітектури, де розробник не керує серверами) для малого бізнесу або НКО (некомерційних організацій) з реляційним сховищем даних і інтеграцією з вітчизняним платіжним API залишається малодослідженим та практично не висвітлено в науковій літературі.

Відсутній опис конкретної, відтворюваної архітектури, яку організація без DevOps-відділу може розгорнути протягом кількох днів, отримати повну аналітику платіжних операцій і при цьому не витрачати кошти на постійно зарезервовані серверні ресурси. Це і є тією невирішеною

практичною задачею, якій присвячена стаття.

Мета статті - розробити та обґрунтувати архітектуру системи обробки та зберігання платіжних даних на стеку Cloudflare Pages/Workers, Supabase та plata by mono, продемонструвати практичну реалізацію на конкретному прикладі та оцінити вартість та переваги запропонованого підходу порівняно з традиційною серверною інфраструктурою.

Основна частина

Архітектурні підходи до побудови серверної частини платіжної системи

Три основних архітектурних підходи до побудови серверної частини платіжної системи розглядаються у вітчизняній та міжнародній практиці.

Традиційний підхід: організація орендує VPS (Virtual Private Server — віртуальний приватний сервер), встановлює Node.js або інший runtime (середовище виконання коду), налаштовує PostgreSQL або MySQL, і розгортає власне написаний додаток. Цей підхід є інтуїтивним і забезпечує повний контроль.

Недоліки: сервер працює і коштує гроші навіть тоді, коли платежів немає взагалі. При піковому навантаженні його потужності може не вистачити. Масштабування вимагає ручного втручання — приходиться переходити на більш потужний тариф. Для регулярного резервного копіювання, оновлення ОС (операційної системи), моніторингу потрібен фахівець з адміністрування (DevOps).

Другий підхід: систему розбивають на незалежні мікросервіси, кожен з яких може масштабуватися окремо, налаштовується контейнеризація (упакування коду з залежностями) через Docker, розгортання через Kubernetes або подібну систему оркестрації (управління взаємодією між сервісами).

Цей підхід пропонує гнучкість у розробці та розгортанні. Але для команди з однієї-двох людей без глибокого досвіду в хмарних технологіях це стає надмірною складністю. Вартість розробки та подальшої підтримки зростає значно. Для малого бізнесу часто виявляється неекономічним.

Третій підхід — функції як сервіс (FaaS — Functions as a Service): серверний код розбивається на невеликі функції, кожна з яких виконується лише при надходженні конкретного запиту. Обчислювальні ресурси виділяються динамічно за меморе і часом виконання, а не резервуються постійно.

Ключова перевага: якщо трафіку немає — витрати мінімальні або нульові. Якщо раптово зросло навантаження — система автоматично створює нові екземпляри функцій. Розробник не керує серверами, лише пише код функцій і встановлює конфігурацію [4, 12].

В табл. 1 наведено порівняння трьох підходів за шістьма критеріями.

Таблиця 1. Порівняльний аналіз архітектурних підходів до побудови системи онлайн-еквайрингу.

Параметр	**VPS + Node.js + PostgreSQL**	**Мікросервісна (Docker + Kubernetes)**	**FaaS (Cloudflare Pages/Workers)**
Вартість у простої (немає трафіку)	$30–50/міс. постійно	$50–100/міс. постійно	~$0 або до $5/міс.
Масштабування при піку (500+ запитів/год)	вручну або передоплата	автоматичне	автоматичне
Складність розгортання	середня (3–5 днів)	висока (1–2 тижні)	низька (1–2 дні)
Потреба у DevOps	так, необхідний фахівець	так, необхідний досвід	мінімальний
Початкові інвестиції	орієнтовно $200–500	орієнтовно $1000–3000	орієнтовно $0
Час переходу до production	1–2 тижні	2–4 тижні	2–5 днів

Аналізуючи дані з табл. 1, можна побачити, що для організацій з обмеженим бюджетом та небагатьма розробниками FaaS-підхід надає найбільше переваг. Саме на цій моделі побудована архітектура, що описана у статті.

Вибір інструментів та їх обґрунтування

Cloudflare Pages — це JAMstack-платформа (JavaScript, API, Markup — архітектура для побудови вебсайтів без серверної логіки) для розгортання статичного та динамічного контенту [5]. Інтеграція з git-репозиторієм дозволяє розгортати нові версії кодом одного команди git push. Cloudflare Pages Functions — це середовище для виконання серверного коду безпосередньо у проєкті Pages без окремого налаштування Workers [5].

Функції зберігаються у папці /functions репозиторію з розширенням.js або.ts (TypeScript — розширена версія JavaScript). Код виконується на edge-мережі (географічно розподіленій мережі серверів) Cloudflare, розташованій у понад 330 точках присутності по всьому світу [5]. Це означає, що код запускається близько до користувача, затримка (latency) мінімальна, і не потрібна передача даних на центральний сервер.

Для субдомену api.site.org.ua достатньо додати CNAME-запис у DNS (Domain Name System — система перекладу доменних імен на IP-адреси) [6]. Cloudflare автоматично видає TLS (Transport Layer Security — протокол захисту транспортного рівня) сертифікат для цього субдомену, HTTPS активується без жодних додаткових дій розробника. Всі запити на api.site.org.ua проксіюються через мережу Cloudflare і обробляються Pages Functions [6].

Вартість: перші 100,000 запитів на день безкоштовно. Далі $0.50 за мільйон запитів. Для НКО з 10,000 транзакцій на місяць витрати залишаються у межах безплатного тарифу [5].

Supabase — це відкрита платформа на базі PostgreSQL (об'єктно-реляційна СУБД — система управління реляційними базами даних, побудована на основі об'єктної моделі) з автоматично згенерованим REST API [7-8]. Основні можливості:

- REST API: без писання SQL-коду браузер або Worker може виконувати операції CRUD (Create, Read, Update, Delete — створення, читання, оновлення, видалення) через HTTP-запити [7–8].

- Realtime (реальний час): Supabase слідкує за змінами у таблицях PostgreSQL через websocket і надсилає push-оновлення підписаним клієнтам [8].

- Row Level Security (RLS): контроль доступу на рівні рядків таблиці. Worker звертається через service key (ключ з повними правами), звичайні користувачі отримують обмежений доступ [8, 13].

Supabase офіційно підтримує інтеграцію з Cloudflare [9]. Після підключення в Cloudflare Dashboard змінні SUPABASE_URL та SUPABASE_SERVICE_KEY стають доступними у коді Worker через об'єкт env (environment — середовище). Взаємодія відбувається через HTTP та PostgREST (REST API для PostgreSQL), що уникає проблеми пулу підключень (connection pool exhaustion) — типової для serverless-середовищ [9].

Вартість: перші 500 МБ та 2 млн запитів на місяць безкоштовно. Pro тариф з більшим місцем зберігання коштує $25/місяць [8].

plata by mono — сервіс онлайн-еквайрингу Monobank, що надає REST API для прийому платежів [10-11]. Основні методи:

- POST /api/merchant/invoice/create — створення рахунку (інвойсу) для оплати. На вхід подається сума у копійках, валюта (980 для гривні), посилання webhook, дані кампанії. На вихід — invoiceId (унікальний ідентифікатор рахунку) та pageUrl (посилання на сторінку оплати monobank) [10, 14].

- GET /api/merchant/pubkey — отримання відкритого ключа для верифікації webhook-подій [11].

- Webhook на зміну статусу — после оплати або відмови plata by mono надсилає HTTP POST на вказаний URL із об'єктом JSON (JavaScript Object Notation — формат обміну даними) та заголовком X-Sign з цифровим підписом [11].

Верифікація підпису: X-Sign містить ECDSA-підпис (Elliptic Curve Digital Signature Algorithm — алгоритм цифрового підпису на основі еліптичних кривих) тіла webhook. Публічний ключ дістають через GET /api/merchant/pubkey, і перевіряють підпис через Web Crypto API (вбудований API для криптографічних операцій у браузерах і Workers) [11, 15].

Перевага plata by mono: вітчизняний сервіс, розрахунки в гривні, дана публічна API-документація із прикладами для JavaScript, Python та Go [10, 11, 16].

Архітектура системи

Організація не буде перебудовувати сайт з нуля. Сайт вже працює на якомусь хостингу (WordPress, Apache, будь-яка CMS). Завдання: додати до цього сайту форму прийому платежів так, щоб вона працювала безпечно, але не навантажувала поточний хостинг серверною логікою обробки платежів.

Архітектура ділиться на дві незалежні частини [17]:

- Існуючий сайт (наприклад, donate.org.ua) — залишається на своєму хостингу без змін, крім додавання HTML-форми та невеликого фрагменту JavaScript (порядок 50–100 рядків).

- Бекенд на субдомені (api.donate.org.ua) — новий компонент на Cloudflare Pages/Workers, де виконується вся серверна логіка: створення рахунків, обробка webhook від plata by mono, верифікація цифрових підписів, запис у базу даних.

Ця розподіленість дозволяє масштабувати бекенд незалежно, оновлювати його без впливу на основний сайт, і застосовувати окремі правила безпеки до API [17].

Щоб субдомен api.donate.org.ua вказував на Cloudflare Pages, потрібен один DNS-запис типу CNAME: ім'я — api, значення — donate-project.pages.dev, TTL — 3600 секунд.

Якщо домен donate.org.ua вже на Cloudflare — запис додається в Dashboard у розділі DNS, і все починає працювати одразу. Якщо домен у стороннього реєстратора — запис додається там, і зміни поширюються до 48 годин.

Після додавання CNAME Cloudflare автоматично видає TLS-сертифікат для api.donate.org.ua (це називається автоматичне видання Managed Certificate). HTTPS активується без ручного втручання [6].

На donate.org.ua з'являється одна HTML-форма із полями [10]:

- Сума пожертви (у гривнях)
- Назва кампанії (обов'язково)
- Email донора (опціонально)

JavaScript-код при натисканні кнопки "Оплатити" виконує fetch()-запит на api.donate.org.ua/create-payment, передавши дані форми у JSON-форматі. Жодної серверної логіки на основному хостингу немає [10].

На Cloudflare Pages розгортається проєкт з двома основними функціями [5, 18]:

1. POST /create-payment — приймає дані форми, звертається до plata by mono API для створення рахунку, отримує invoiceId та pageUrl, записує рядок у таблицю invoices у Supabase зі статусом "created", повертає pageUrl браузеру.

2. POST /webhook — приймає POST-запити від plata by mono після зміни статусу платежу. Обов'язково перевіряє ECDSA-підпис у заголовку X-Sign. Якщо підпис невірний — повертає HTTP 400 і нічого не записує. При успішній верифікації — оновлює рядок у таблиці invoices (операція upsert — вставка або оновлення за ключем) та додає запис у payment_events (для аудиту).

Коли браузер виконує fetch()-запит з donate.org.ua на api.donate.org.ua — це cross-origin запит (запит між різними джерелами), навіть якщо домен один і той самий. Браузер сам ініціює preflight OPTIONS-запит (попередній запит браузера для перевірки дозволів перед надсиланням основного запиту) [19-20].

Worker на api.donate.org.ua обробляє запити OPTIONS і повертає заголовки:

Access-Control-Allow-Origin: https://donate.org.ua
Access-Control-Allow-Methods: POST, OPTIONS
Access-Control-Allow-Headers: Content-Type
Access-Control-Max-Age: 86400

Це забезпечує, що до API можна звернутися тільки з дозволеного домену donate.org.ua [19-20].

Повна послідовність дій (рис. 1):

Крок 1: Користувач відкриває сторінку donate.org.ua (основний хостинг, без серверної логіки).

Крок 2: Вводить суму, вибирає кампанію, натискає кнопку "Оплатити". JavaScript виконує POST-запит на api.donate.org.ua/create-payment із JSON-тілом.

Крок 3: Cloudflare Pages перехоплює запит і виконує функцію create-payment. Функція читає тіло запиту та готує дані для plata by mono: сума у копійках, посилання webhook, дані кампанії.

Крок 4: Worker надсилає POST на plata by mono /api/merchant/invoice/create з токеном авторизації у заголовку X-Token. plata by mono повертає відповідь із invoiceId та pageUrl.

Крок 5: Worker зберігає нову транзакцію у Supabase: додає рядок у таблицю invoices з полями invoiceId, amount, status: "created", campaign_id, donor_email, created_at. Це дозволяє отримати повну історію платежів.

Крок 6: Worker повертає браузеру об'єкт JSON з полем pageUrl.

Крок 7: JavaScript браузера перенаправляє користувача на pageUrl. Користувач потрапляє на сторінку оплати Monobank.

Крок 8: Користувач вводить дані картки на сторінці monobank (ця сторінка живе на серверах Monobank, не на наших серверах) і натискає "Оплатити".

Крок 9: Monobank обробляє платіж, отримує результат від платіжної системи. Якщо платіж успішний — plata by mono генерує webhook-подію.

Крок 10: plata by mono надсилає HTTP POST на api.donate.org.ua/webhook з JSON-тілом, яке містить invoiceId, status ("success" або інший), та заголовком X-Sign з ECDSA-підписом.

Крок 11: Cloudflare Pages виконує функцію webhook. Функція читає X-Sign із заголовків, завантажує публічний ключ через GET /api/merchant/pubkey, перевіряє ECDSA-підпис через Web Crypto API.

Крок 12: Якщо підпис вірний (match) — Worker оновлює рядок у таблиці invoices (upsert за ключем invoiceId, змінює status на "success"). Одночасно Worker додає запис у таблицю payment_events для аудиту — з усім payload webhook.

Крок 13: Якщо підпис невірний — Worker повертає HTTP 400 Bad Request і не записує нічого до бази. Це запобігає записуванню підробних платежів.

Крок 14: Worker повертає HTTP 200 OK, підтверджуючи отримання webhook.

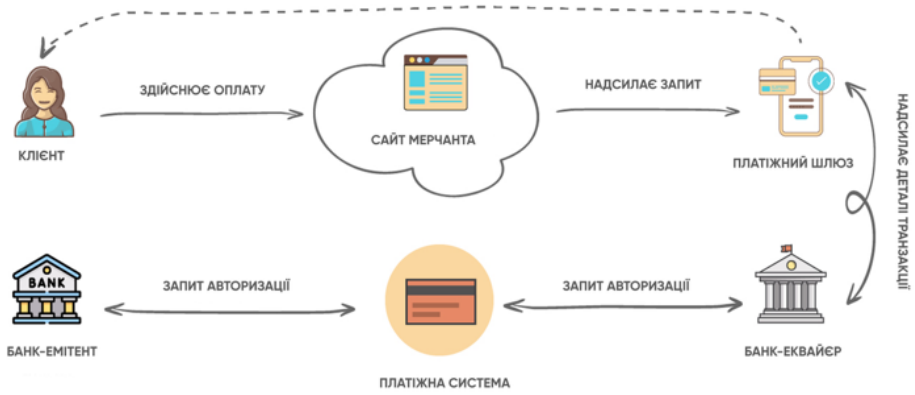

Рис. 1. Загальна архітектура системи онлайн-еквайрингу

Модель даних у Supabase

Для представлекння моделі даних було створено три таблиці з такими полями (рис. 2) [21]:

Актуальний стан кожного рахунку для оплати. Поля:

1. `id` (UUID, PRIMARY KEY) — унікальний ідентифікатор запису у Supabase
2. `invoice_id` (TEXT, UNIQUE) — ідентифікатор рахунку від plata by mono (наприклад, "p2_9ZgpZVsl3")
3. `amount` (INTEGER) — сума у копійках (для гривні), наприклад 42000 копійок = 420 гривень
4. `ccy` (INTEGER) — код валюти за ISO 4217 (980 для гривні)
5. `status` (TEXT) — статус платежу: "created", "success", "failed", "expired"
6. `campaign_id` (UUID, FOREIGN KEY) — посилання на таблицю campaigns
7. `donor_email` (TEXT, NULLABLE) — email донора (опціональне)
8. `created_at` (TIMESTAMP) — момент створення рахунку
9. `updated_at` (TIMESTAMP) — момент останнього оновлення
10. `page_url` (TEXT) — посилання на сторінку оплати monobank, збережене для історії.

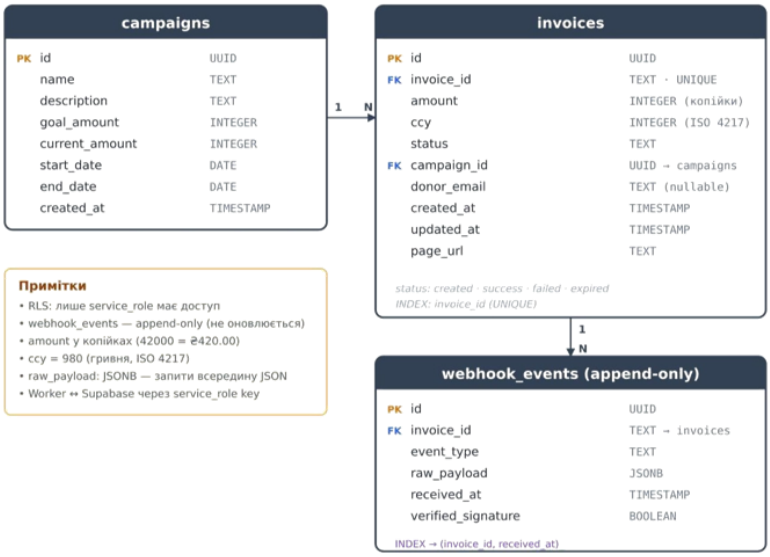

Рис. 2. ER-діаграма моделі бази даних платіжної системи

Append-only журнал усіх webhook-подій від plata by mono. Ці дані ніколи не оновлюються — лише додаються нові записи. Це критично важливо для аудиту [4, 17]. Поля:

1. id (UUID, PRIMARY KEY) — унікальний ідентифікатор запису
2. invoice_id (TEXT, FOREIGN KEY) — посилання на таблицю invoices

3. event_type (TEXT) — тип події: "status_changed", "payment_received", тощо

4. raw_payload (JSONB) — весь JSON от plata by mono без змін (JSONB — бінарний JSON у PostgreSQL, що дозволяє робити запити всередину JSON)

5. received_at (TIMESTAMP) — момент отримання webhook від plata by mono

6. verified_signature (BOOLEAN) — чи пройшла верифікація ECDSA-підпису.

Індекс (INDEX) на invoice_id та received_at для швидкого отримання історії платежу.

Довідник кампаній зі збору коштів. Поля:

1. id (UUID, PRIMARY KEY)

2. name (TEXT) — назва кампанії (наприклад, "Допомога ЗСУ — березень 2026")

3. description (TEXT) — опис

4. goal_amount (INTEGER) — цільова сума у копійках - current_amount (INTEGER) — поточна зібрана сума (оновлюється при кожному успішному платежі)

5. start_date (DATE) — дата початку кампанії

6. end_date (DATE) — дата завершення кампанії

7. created_at (TIMESTAMP) — момент створення запису.

NoSQL бази даних (MongoDB, DynamoDB тощо) добре підходять для гнучких, слабко структурованих даних. Але платіжні дані мають жорстку структуру і виконують критичні завдання :

1. ACID-транзакції (Atomicity, Consistency, Isolation, Durability — атомарність, узгодженість, ізоляція, довговічність). Якщо при оновленні рядка трапився crash — PostgreSQL гарантує, що або змінилося все, або нічого. У документних БД це набагато складніше.

2. Складні агрегації — при аналітиці часто потрібно групувати дані, робити JOIN (об'єднання таблиць), підраховувати суми. SQL це робить легко та швидко. У NoSQL потрібно писати складний код на додатку.

3. Структурна цілісність — заперечення (constraints) та зовнішні ключі (foreign keys) гарантують, що неможливо додати посилання на неіснуючу кампанію.

Тому PostgreSQL в Supabase є оптимальним вибором для платіжних даних.

Worker звертається до Supabase через service role key (ключ з повними правами на всі таблиці). Звичайні користувачі або прямі запити з браузера без аутентифікації взагалі не можуть звернутися до API [13].

На рівні RLS в SQL встановлюються правила:

```
ALTER TABLE invoices ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Allow service role full access"
ON invoices
FOR ALL
TO authenticated
USING (auth.role() = 'service_role');

CREATE POLICY "Deny public access"
ON invoices
FOR ALL
TO anon
USING (FALSE);
```

Це означає: лише service role (тобто Worker) може читати та писати. Anon (анонімні користувачі з браузера без ключа) отримають помилку 403 Forbidden [13].

Практична реалізація

Обробник створення платежу (create-payment.js) реалізує такий алгоритм:

1. Перевірка методу запиту. Якщо запит типу OPTIONS — повернути CORS-заголовки і завершити обробку. Якщо метод відмінний від POST — повернути статус 405.

2. Зчитування та валідація вхідних даних: поля amount, campaignId, donorEmail. Якщо сума менша за 100 копійок (1 грн) — повернути помилку 400:

```
const { amount, campaignId, donorEmail } = await
request.json();
if (!amount || amount < 100) {
return new Response(JSON.stringify({ error:
'Minimum amount is 1 UAH' }), { status: 400 });}
```

3. Звернення до API plata by mono (POST /api/merchant/invoice/create). Тіло запиту містить суму в копійках, код

валюти (980 — UAH), URL для редиректу після оплати і URL вебхука:

```
body: JSON.stringify({
amount: amount * 100,
ccy: 980,
redirectUrl: 'https://donate.org.ua/success',
webHookUrl: 'https://api.donate.org.ua/api/webhook',
})
```

4. Якщо платіжний сервіс повернув помилку — передати її клієнту зі збереженням оригінального HTTP-статусу.

5. Запис нового запису до таблиці invoices у Supabase через REST API. Фіксуються: ідентифікатор рахунку, сума, статус "created", кампанія, email донора.

6. Повернення клієнту pageUrl для переходу на сторінку оплати та invoiceId для подальшого відстеження.

Токен платіжного сервісу та ключі Supabase передаються через змінні середовища Cloudflare Secrets:

```
env.MONO_TOKEN
env.SUPABASE_URL
env.SUPABASE_SERVICE_KEY
```

Обробник вебхука (webhook.js) реалізує такий алгоритм:

1. Перевірка наявності заголовку X-Sign у вхідному запиті. Якщо відсутній — повернути 400.

2. Зчитування тіла запиту як рядка — це обов'язкова умова для коректної верифікації підпису, оскільки будь-яке перетворення тіла змінює його байтовий склад:

```
const xSign = request.headers.get('X-Sign');
const bodyText = await request.text();
const payload = JSON.parse(bodyText);
```

3. Отримання публічного ключа від plata by mono через GET /api/merchant/pubkey.

4. Верифікація ECDSA-підпису через Web Crypto API по алгоритму P-256/SHA-256:

```
const isValid = await crypto.subtle.verify(
  { name: 'ECDSA', hash: { name: 'SHA-256' } },
  pubKey,
  signatureBytes,
  messageBytes
);
if (!isValid) {
return new Response(JSON.stringify({ error: 'Invalid signature' }), { status: 400 });}
```

5. Оновлення статусу рахунку в таблиці invoices через PATCH-запит до Supabase REST API (поля status та updated_at).

6. Запис події до таблиці payment_events для аудиту: тип події, повне тіло запиту у форматі jsonb, прапорець verified_signature.

7. Повернення статусу 200 — plata by mono вважає доставку вебхука успішною лише при такій відповіді.

Ідемпотентність операцій забезпечена на рівні PATCH-запиту: якщо той самий вебхук надійде двічі, запис у invoices просто оновиться повторно з тим самим статусом, а до payment_events додасться ще один рядок аудиту.

Аналітика транзакцій

Запит у Supabase для отримання підсумків пожертв по кампаніях за день [22]:

```
SELECT
  c.id,
  c.name,
  DATE(i.created_at) AS donation_date,
  COUNT(*) AS donation_count,
  SUM(i.amount) / 100 AS total_amount_uah,
  AVG(i.amount) / 100 AS avg_donation_uah
FROM invoices i
LEFT JOIN campaigns c ON i.campaign_id = c.id
WHERE i.status = 'success'
  AND i.created_at >= NOW() - INTERVAL '30 days'
GROUP BY c.id, c.name, DATE(i.created_at)
ORDER BY DATE(i.created_at) DESC, c.name;
```

В табл. 2 представлено результат виконання запиту аналітики транзакцій.

Таблиця 2. Результат запиту аналітики транзакцій

id	name	donation_date	donation_count	total_amount_uah	avg_donation_uah
abc-123	“Допомога ЗСУ”	2026-03-15	45	18500	411.11
abc-123	“Допомога ЗСУ”	2026-03-14	32	14200	443.75
def-456	“Медикаменти”	2026-03-15	12	5600	466.67

Supabase Realtime дозволяє фронтенду слідкувати за змінами в таблиці на webhooks [8]:

```
// На сайті donate.org.ua
```

```
import { createClient } from '@supabase/supabase-js';

const supabase = createClient(SUPABASE_URL, SUPA-
BASE_ANON_KEY);

// Підписуємось на нові успішні платежі
const subscription = supabase
  .channel('invoices-changes')
  .on(
    'postgres_changes',
    { event: '*', schema: 'public', table: 'invoices'
},
    (payload) => {
      if (payload.new.status === 'success') {
            console.log('Нова успішна пожертва:',
payload.new.amount / 100, 'грн');
        updateDonationWidget(payload.new);
      }
    }
  )
  .subscribe();
```

Це дозволяє фронтенду отримувати push-оновлення при кожній новій успішній транзакції без polling (циклічного опитування) кожні N секунд [8, 23].

Безпека платіжних даних

Кожен webhook від plata by mono обов'язково містить ECDSA-підпис у заголовку X-Sign. Це гарантує, що webhook дійсно створений plata by mono, а не підробний запит від третьої особи [11].

Алгоритм верифікації [11]:

1. Завантажуємо публічний ключ через GET /api/merchant/pubkey
2. Дістаємо підпис з X-Sign
3. Обчислюємо SHA-256 хеш тіла webhook
4. Перевіряємо, чи підпис відповідає хешу та публічному ключу
5. Якщо відповідає — обробляємо webhook
6. Якщо не відповідає — відхиляємо запит з помилкою: 400.

Worker звертається через service key з повними правами. Всі інші з'єднання без ключа — заблоковані на рівні RLS [13]. Таким чином, таблиці з платіжними даними недоступні з браузера без авторизації.

Чутливі дані (токени, ключі) не зберігаються в коді репозиторію. Вони встановлюються через Cloudflare Dashboard у розділі Environment Variables (Encrypted) [24]. При розгортанні Worker вони автоматично доступні через об'єкт env.

Cloudflare автоматично видає TLS сертифікати для всіх доменів і субдоменів. Всі запити переводяться на HTTPS. Плюс до цього, запити між Worker та Supabase йдуть через HTTPS (зашифровані).

Критично важливо: номер картки, CVV, дата закінчення — ці дані не потрапляють на наші сервери. Користувач вводить їх на сторінці monobank (тобто на серверах Monobank, а не на нашій інфраструктурі). Ми обробляємо лише invoiceId, статус платежу та суму [10].

Це мінімізує область відповідності стандарту PCI DSS (Payment Card Industry Data Security Standard — стандарт безпеки для обробки платіжних карток). Замість того, щоб відповідати всім вимогам PCI DSS, нам потрібно лише забезпечити безпеку самих платіжних ідентифікаторів [25].

Висновки

Проведене дослідження підтвердило, що архітектура на основі Cloudflare Pages/Workers, Supabase та plata by mono забезпечує надійне та масштабоване рішення для прийому онлайн-платежів організаціями з обмеженими технічними і фінансовими ресурсами. Запропонований підхід дозволяє НКО та малому бізнесу розгорнути повноцінний платіжний бекенд без залучення DevOps-фахівця.

Принципово важливою перевагою запропонованої архітектури є розподіл між існуючим фронтендом і новим бекендом на субдомені. Організація не переробляє наявний сайт, а лише додає до нього форму та невеликий JavaScript-фрагмент. Уся серверна логіка — створення рахунків, обробка webhook, верифікація підписів і запис до бази даних — виконується на Cloudflare Pages/Workers незалежно від основного хостингу. При піковому навантаженні система масштабується автоматично, без ручного втручання і без попередньої оплати резервованих потужностей.

Реляційна модель у Supabase на базі PostgreSQL дозволяє виконувати складні аналітичні запити безпосередньо на рівні

бази даних. Агрегації по кампаніях, підрахунок конверсії та статистика унікальних донорів обчислюються за мілісекунди навіть для великих масивів транзакційних даних — без перенесення даних у додаток і без окремої аналітичної системи.

Безпека платіжних даних забезпечується на кількох рівнях: ECDSA-верифікація кожного вхідного webhook виключає запис підроблених транзакцій, Row Level Security у Supabase блокує будь-який прямий доступ до таблиць без авторизації, а зберігання токенів виключно у зашифрованих змінних Cloudflare унеможливлює їхнє потрапляння у відкритий код. Оскільки карткові дані не проходять через інфраструктуру системи, обсяг вимог стандарту PCI DSS суттєво скорочується.

Перспективами подальших досліджень є розширення системи на підтримку регулярних (recurring) платежів, реалізація мультивалютності та інтеграція з міжнародними платіжними провайдерами, а також розробка мобільних клієнтів з офлайн-синхронізацією через Supabase Realtime.

Література

1. Scheuner J., Leitner P. Function-as-a-Service Performance Evaluation: A Multivocal Literature Review. Journal of Systems and Software. 2020. Vol. 170. Article 110708. DOI: 10.1016/j.jss.2020.110708. URL: https://doi.org/10.1016/j.jss.2020.110708 (дата звернення: 28.03.2026).

2. Bayya A. K. The Role of Serverless Architectures in Revolutionizing FinTech Solutions. Asian Journal of Mathematics and Computer Research. 2025. Vol. 32(1). P. 1–26. DOI: 10.56557/ajomcor/2025/v32i19035. URL: https://ikprress.org/index.php/AJOMCOR/article/view/9035 (дата звернення: 28.03.2026).

3. Serverless Transaction Management: A Case Study of Real-Time Order Processing in Distributed Systems. European Academic Journals. 2025. https://doi.org/10.37745/ejcsit.2013/vol13n27105115.

4. Jonas E., Schleier-Smith J., Sreekanti V. et al. Cloud programming simplified: A Berkeley view on serverless computing. Technical Report UCB/EECS-2019-3. University of California, Berkeley, 2019. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf (дата звернення: 28.03.2026).

5. Cloudflare Pages — офіційна документація та продуктова сторінка. URL: https://pages.cloudflare.com (дата звернення: 28.03.2026).

6. Cloudflare Pages: Custom domains — офіційна документація. URL: https://developers.cloudflare.com/pages/configuration/custom-domains/ (дата звернення: 28.03.2026).

7. Supabase Database — офіційна документація. URL: https://supabase.com/docs/guides/database/overview (дата звернення: 28.03.2026).

8. Supabase Realtime — офіційна документація. URL: https://supabase.com/docs/guides/realtime (дата звернення: 28.03.2026).

9. Supabase + Cloudflare Workers — офіційна інтеграційна документація. URL: https://supabase.com/partners/integrations/cloudflare-workers (дата звернення: 28.03.2026).

10. plata by mono — офіційна API-документація еквайрингу. URL: https://monobank.ua/api-docs/acquiring (дата звернення: 28.03.2026).

11. plata by mono: Верифікація webhook-підпису (X-Sign, ECDSA) — офіційна документація. URL: https://monobank.ua/api-docs/acquiring/dev/webhooks/verify (дата звернення: 28.03.2026).

12. Supabase Edge Functions — офіційна документація. URL: https://supabase.com/docs/guides/functions (дата звернення: 28.03.2026).

13. Supabase Row Level Security — офіційна документація. URL:

https://supabase.com/docs/guides/auth/row-level-security (дата звернення: 28.03.2026).

14. plata by mono: Метод створення рахунку (POST /api/merchant/invoice/create) — офіційна API-документація. URL: https://monobank.ua/api-docs/acquiring/methods/ia/post–api–merchant–invoice–create (дата звернення: 28.03.2026).

15. Web Crypto API — специфікація W3C. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API (дата звернення: 28.03.2026).

16. Monobank API Community Docs — відкрита документація спільноти. GitHub: andrew-demb/monobank-api-community-docs.URL:https://github.com/andrew-demb/monobank-api-community-docs (дата звернення: 28.03.2026).

17. Cloudflare Workers: Best Practices — офіційна документація. URL: https://developers.cloudflare.com/workers/best-practices/workers-best-practices/ (дата звернення: 28.03.2026).

18. Cloudflare Pages Functions — офіційна документація. URL: https://developers.cloudflare.com/pages/functions/ (дата звернення: 28.03.2026).

19. CORS (Cross-Origin Resource Sharing) — офіційна документація Cloudflare. URL: https://developers.cloudflare.com/cache/cache-security/cors/ (дата звернення: 28.03.2026).

20. Cross-Origin Resource Sharing (CORS). Mozilla Developer Network. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 28.03.2026).

21. PostgreSQL: DDL Basics — офіційна документація. URL: https://www.postgresql.org/docs/current/ddl-basics.html (дата звернення: 28.03.2026).

22. PostgreSQL Aggregate Functions — офіційна документація. URL: https://www.postgresql.org/docs/current/functions-aggregate.html (дата звернення: 28.03.2026).

23. Supabase Realtime: Broadcast from Database — офіційний блог Supabase. URL:https://supabase.com/blog/realtime-broadcast-from-database (дата звернення: 28.03.2026).

24. Cloudflare Workers: Environment Variables and Secrets — офіційна документація. URL: https://developers.cloudflare.com/workers/configuration/environment-variables/ (дата звернення: 28.03.2026).

25. PCI DSS (Payment Card Industry Data Security Standard) — офіційні вимоги. URL: https://www.pcisecuritystandards.org/ (дата звернення: 28.03.2026).

Хрустовський А., Савченко А., Дехтярук М., Райчев І.

ВИКОРИСТАННЯ ХМАРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ АНАЛІЗУ ТА ОБРОБКИ ПЛАТІЖНИХ ДАНИХ У СИСТЕМАХ ОНЛАЙН-ЕКВАЙРИНГУ

Впровадження онлайн-еквайрингу в малому бізнесі та некомерційних організаціях часто стикається з типовою технічною проблемою: відсутність логування транзакцій, нестабільність системи під піковим навантаженням і невизначений статус фінансових операцій після збою. Ці проблеми виникають через використання традиційної хостингової інфраструктури, яка або не масштабується автоматично, або вимагає постійно зарезервованих ресурсів, які коштують гроші навіть у простої.

У статті описано архітектуру системи обробки та зберігання платіжних даних, побудованої на трьох компонентах: Cloudflare Pages/Workers як безсерверний обчислювальний рівень, Supabase на PostgreSQL як реляційне сховище з підтримкою реального часу, plata by mono як платіжний шлюз із REST API та верифікацією webhook-підписів. Запропонований підхід орієнтований на організації без власного технічного відділу, які не можуть дозволити собі витрати на VPS-інфраструктуру чи налаштування Kubernetes.

Розглянуто модель даних із трьома таблицями — invoices, campaigns і payment_events — та зв'язками між ними, проілюстрованими ER-діаграмою. Описано обробку webhook-подій із верифікацією ECDSA-підпису через Web Crypto API, що відповідає вимогам безпеки plata by mono. Наведено практичну реалізацію двох Cloudflare Workers-функцій: створення платіжного рахунку та обробки статусних подій. Продемонстровано аналітичні SQL-запити для підсумку транзакцій по кампаніях і підписку на оновлення в реальному часі через Supabase Realtime.

Порівняльний аналіз трьох архітектурних підходів — традиційний VPS, мікросервісна архітектура на Docker/Kubernetes і безсерверна FaaS — показує, що запропоноване рішення знижує витрати на інфраструктуру з $110–310 до $0–30 на місяць, не потребує DevOps-інженера і дає змогу розгорнути повноцінний платіжний бекенд за 2–5 днів. Заходи безпеки відповідають вимогам PCI DSS.

Ключові слова: *онлайн-еквайринг, безсерверна архітектура, Cloudflare, Supabase, PostgreSQL, plata by mono, платіжні дані, webhook, ECDSA, неприбуткові організації.*

Khrustovskyi A., Savchenko A., Dekhtyaruk M., Raichev I.

THE USE OF CLOUD COMPUTING TECHNOLOGIES FOR THE ANALYSIS AND PROCESSING OF PAYMENT DATA IN ONLINE ACQUIRING SYSTEMS

Small businesses and non-profit organizations that adopt online payment processing often face a common set of technical problems: transactions go unlogged, the system becomes unstable under peak load, and payment statuses remain unresolved after failures. These issues stem from relying on conventional hosting infrastructure that either does not scale automatically or requires constantly reserved resources that cost money even during idle periods.

This paper presents an architecture for processing and storing payment data, built on three components: Cloudflare Pages/Workers as the serverless compute layer, Supabase on PostgreSQL as the relational data store with real-time capabilities, and plata by mono as the payment gateway with a REST API and webhook signature verification. The proposed approach targets organizations without a dedicated technical team that cannot afford VPS infrastructure or Kubernetes configuration overhead.

The paper describes a three-table data model — invoices, campaigns, and payment_events — with entity relationships illustrated by an ER diagram. Webhook event handling is covered in detail, including ECDSA signature verification via the Web Crypto API, which satisfies the security requirements of plata by mono. Two Cloudflare Workers functions are described algorithmically: one for creating a payment invoice and one for processing status update events. Analytical SQL queries for aggregating transactions by campaign and real-time subscription through Supabase Realtime are also demonstrated.

A comparative analysis of three architectural approaches — traditional VPS, microservices on Docker/Kubernetes, and serverless FaaS — shows that the proposed solution reduces infrastructure costs from $110–310 to $0–30 per month, requires no DevOps engineer, and allows a fully functional payment backend to be deployed within 2–5 days. The security controls implemented are consistent with PCI DSS requirements.

Keywords: *online acquiring, serverless architecture, Cloudflare, Supabase, PostgreSQL, plata by mono, payment data, webhook, ECDSA, non-profit organizations.*

Стаття подана до редакції: 08*/09/2026*
Стаття прийнята до опублікування:14/09/2026
Стаття опублікована: 10/11/2026

Стаття поширюється на умовах ліцензії CC BY 4.0