**Мамалига Н. Є.**
**Паламарчук Є. А.**

# РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ СЕРВІСУ АВТОМАТИЗОВАНОЇ СИСТЕМИ ОБЛІКУ ФІНАНСІВ

Вінницький національний технічний університет

***Анотація***

*У роботі представлено побудову автоматизованої системи контролю та обліку фінансів по регіонах, побудованої на клієнт-серверній архітектурі. В основі серверної частини застосовано фреймворк Laravel із залученням пакетів для керування ролями, багатомовністю та безпекою, що забезпечує гнучку й захищену роботу з даними. На клієнтської частини використаний фреймворк Vue 3, завдяки чому реалізовано SPA-рішення зі швидкою навігацією та динамічним оновленням даних. Такий підхід уможливливив суттєве спрощення адміністративних процесів, скорочення витрат часу й ресурсів, а також підвищення прозорості та якості у формуванні фінансовій звітності. У роботі проаналізовано переваги обраного стеку технологій, окреслено механізми інтеграції серверної логіки з клієнтським інтерфейсом.*

***Ключові слова**: автоматизована система, фінансовий облік, клієнт-серверна архітектура, Laravel, Vue 3, SPA-технологія, безпека даних, багатомовність, інтеграція серверної та клієнтської частин*

***Abstract***

*The paper presents the construction of an automated system for controlling and accounting finances by region, built on a client-server architecture. The server part is based on the Laravel framework with the involvement of packages for role management, multilingualism and security, which provides flexible and secure work with data. The client part uses the Vue 3 framework, thanks to which an SPA solution with fast navigation and dynamic data updating was implemented. This approach made it possible to significantly simplify administrative processes, reduce time and resource costs, as well as increase transparency and quality in the formation of financial reporting. The paper analyzes the advantages of the selected technology stack, outlines the mechanisms for integrating server logic with the client interface.*

***Keywords**: automated system, financial accounting, client-server architecture, Laravel, Vue 3, SPA technology, data security, multilingualism, integration of server and client parts*

## Вступ

У типових умовах функціонування організацій із великим обсягом фінансових операцій контроль над надходженнями та витратами зазвичай покладається на окремого працівника (скарбника) або навіть на декількох осіб. Їх обов'язки включають щоденну перевірку платежів, ведення обліку та моніторинг витрат. Такий підхід потребує значних часових і трудових ресурсів, а також супроводжується підвищеним ризиком виникнення помилок через людський фактор. Сучасні системи фінансового обліку дедалі частіше впроваджують клієнт-серверні технології, орієнтовані на автоматизацію ключових процесів, зменшення навантаження на персонал і підвищення точності фінансових операцій. Це дозволяє не лише оптимізувати управлінські процеси, а й підвищити загальний рівень прозорості та контролю у фінансовій діяльності організацій.

Якщо ж автоматизувати всі ці процеси, то можна суттєво знизити витрати на адміністрування та одночасно позбавитися від багатьох «ручних» неточностей. Ідея створити систему, де кожен зможе бачити потрібну йому інформацію та навіть платити безпосередньо онлайн, відкриває широкий простір для спрощення роботи. По-перше, це дозволяє швидко формувати статистику й аналізувати фінансові потоки по регіонах. По-друге, зникає потреба у постійному ручному внесенні даних — усе робиться автоматично та з меншим ризиком

помилок. І, зрештою, це прозоро й зручно, бо керівництво організації може будь-коли перевірити реальний стан фінансів.

Тож розробка та впровадження такої автоматизованої системи є актуальною. Передбачається, що вона не лише заощаджує час і гроші, а й дає змогу оперативніше відслідковувати й аналізувати всі фінансові операції. Це допомагає уникати неприємних ситуацій із «загубленими» коштами й дозволяє швидше реагувати на зміни або проблеми з оплатами.

## Аналіз стану питання

Оцінюючи різні архітектурні підходи до побудови вебсистеми для автоматизованого контролю та обліку фінансів, було проведено аналіз їхніх переваг і недоліків у контексті вимог проєкту.

Монолітна архітектура

- Складність масштабування. Весь застосунок зібраний в один блок, і з ростом функціональності або навантаження виникають проблеми з продуктивністю, що потребують масштабування всього проєкту, а не його окремих частин.
- Тісний зв'язок компонентів. Ускладнює зміну або оновлення окремих модулів без ризику зламати решту системи.
- Менша гнучкість для майбутнього переходу на інші архітектурні моделі.

Мікросервісна архітектура

- Занадто складна для старту. Для невеликого чи середнього проєкту мікросервіси створюють зайву інфраструктурну складність (оркестрація, DevOps, CI/CD, моніторинг тощо).
- Надлишкова технічна вага. Потрібно впроваджувати окремі сервіси, шини повідомлень, API шлюзи — що не виправдано на етапі MVP або ранньої розробки.

SPA (Single Page Application) + API First архітектура

- Гнучка основа. Побудова клієнт-серверної архітектури з розділенням відповідальностей: фронтенд (Vue 3) взаємодіє через API з бекендом (Laravel), що дозволяє легко масштабуватись у майбутньому.
- Можливість еволюції. У майбутньому API-частину можна винести в окремі сервіси без критичних змін на фронті — на відміну від tightly coupled моноліту.
- Безпека. Аутентифікація реалізована через JWT, що дозволяє ефективно захищати маршрути, інтегруватись з мобільними додатками або сторонніми сервісами.
- Швидкість. Завдяки SPA, користувач отримує миттєві реакції інтерфейсу без повного перезавантаження сторінки, що критично для UX.

Таким чином, оптимальним вибором є збереження балансу між простотою, безпекою і можливістю масштабування. У цій роботі вибрана SPA архітектура.

Альтернатива — Modular Monolith

Проміжний підхід — модульний моноліт, коли застосунок розбито на незалежні модулі в межах одного репозиторію. Це дозволяє:

- логічно структурувати код,
- поступово виокремлювати частини системи в мікросервіси,
- при цьому уникати складнощів DevOps на ранніх етапах.

Хороший компроміс, але SPA + API-first наразі все ж виглядає як більш еволюційний і готовий до масштабування підхід.

**Побудова архітектури**

Для реалізації автоматизованої системи контролю й обліку фінансів було обрано чітко структурований стек технологій, до якого входять інструменти як для серверної частини, так і для клієнтської.

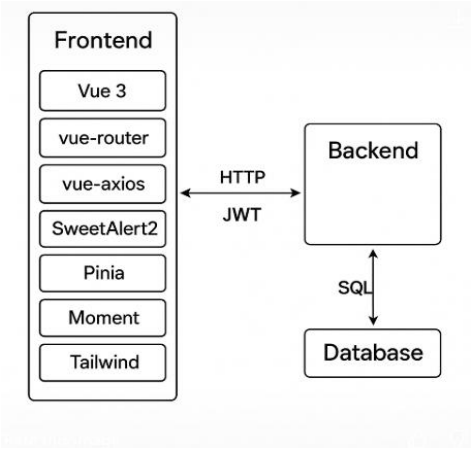



Рис. 1. Структура

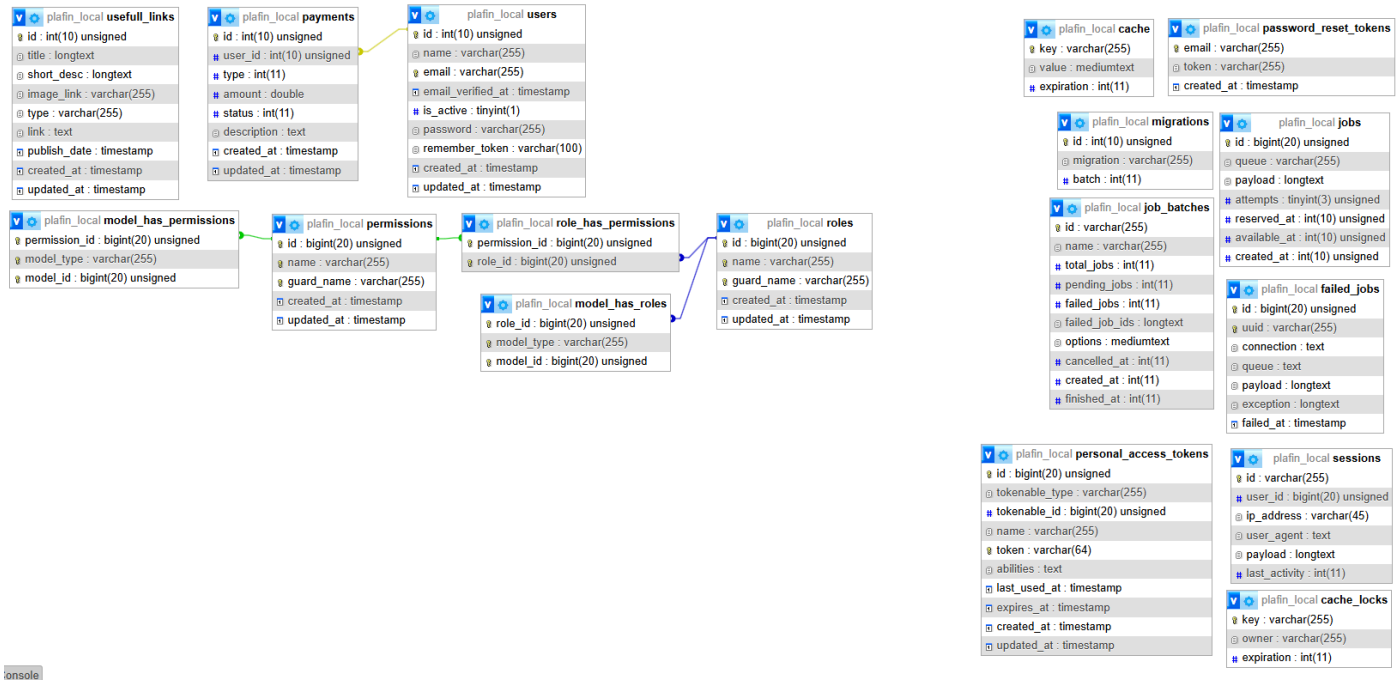
На рис. 1 наведено схему клієнт-серверної веб-системи:

- Frontend реалізовано на базі Vue 3 із використанням vue-router, axios (для HTTP-запитів), SweetAlert2, Pinia, Moment і Tailwind для UI.
- Axios надсилає запити до Backend через HTTP, використовуючи JWT для авторизації.
- Серверна частина обробляє запити, взаємодіє з базою даних (SQL) та повертає відповіді назад на фронт.

Система побудована з урахуванням модульності, безпеки та масштабованості.

Отже, вибрані технології взаємодоповнюють одна одну й відповідають сучасним вимогам до створення швидкого, надійного та масштабованого веб-додатку. В результаті ми отримуємо зручний для розробників серверної частини, потужний та гнучкий клієнтський інтерфейс, а також надійну базу для майбутнього розвитку функціоналу системи.

Була вибрана наступна архітектура для реалізації бази даних:




Ця схема відображає структуру бази даних для веб застосунку з підтримкою ролей, прав доступу, платежів та посилань. Вона умовно ділиться на функціональні таблиці (в центрі та ліворуч) та системні таблиці (праворуч).

Основні таблиці:

- users – основна таблиця користувачів із полями name, email, password, is_active тощо.
- payments – зберігає дані про платежі, включно з user_id, amount, status та description.
- usefull_links – таблиця з корисними посиланнями: title, short_desc, image_link, publish_date тощо.

Таблиці для авторизації (RBAC, spatie/laravel-permission):

- roles – ролі користувачів (id, name, guard_name).
- permissions – дозволи або права (id, name, guard_name).
- model_has_roles – зв'язок між моделями (наприклад, користувачами) і ролями.
- model_has_permissions – пряме призначення прав моделям.
- role_has_permissions – дозволи, закріплені за ролями.

Системні таблиці (правий блок):

Це технічні таблиці Laravel, які використовуються для обслуговування додатку:

- migrations, jobs, failed_jobs, job_batches — обробка черг задач.
- personal_access_tokens — токени для API (Laravel Sanctum).
- sessions — зберігання сесій користувачів.
- password_reset_tokens, cache, cache_locks — обслуговування кешу та відновлення паролів.

Ця структура повністю відповідає потребам масштабованого застосунку з аутентифікацією, гнучким управлінням ролями, підтримкою платежів та системними сервісами Laravel.

## Висновки

Розроблена система автоматизованого контролю та обліку фінансів по регіонах на основі клієнт-серверної архітектури демонструє ефективне поєднання сучасних серверних і клієнтських рішень. Використання Laravel з відповідними пакетами забезпечує гнучку та безпечну обробку даних, у той час як Vue 3 дає змогу реалізувати швидкий і зручний інтерфейс у форматі SPA. Така інтеграція дозволяє суттєво знизити адміністративні навантаження, підвищити прозорість роботи зі звітами та сприяє точнішому й оперативнішому прийняттю фінансових рішень.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Otwell T. Laravel: Documentation [Електронний ресурс]. – Режим доступу: https://laravel.com/docs.
2. Vue.js: The Progressive JavaScript Framework [Електронний ресурс]. – Режим доступу: https://vuejs.org.
3. Spatie: laravel-permission [Електронний ресурс]. – Режим доступу: https://spatie.be/docs/laravel-permission.
4. Tymon JWT Auth [Електронний ресурс]. – Режим доступу: https://github.com/tymondesigns/jwt-auth.
5. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML [Електронний ресурс]. – Режим доступу: https://tailwindcss.com.