5. Сидоренко В. А. Оптимізація IT-інфраструктури підприємства через впровадження методів Infrastructure as Code. Цифрова економіка та системи управління. 2024. Вип. 12. С. 15–22.

УДК 004.8:004.415

АРХІТЕКТУРНІ ПІДХОДИ ТА ІНЖЕНЕРНІ ПРАКТИКИ РОЗРОБКИ ДЕЦЕНТРАЛІЗОВАНИХ ВЕБ-ЗАСТОСУНКІВ НА БАЗІ БЛОКЧЕЙНУ

Скубій Є.В.
yevgeniyaskubiy@gmail.com
Черкаський державний фаховий бізнес-коледж
Дмитрюк В.В.
м.Черкаси, Україна

Розвиток Web 3.0 потребує створення децентралізованих застосунків (dApps), які забезпечують безпечну взаємодію з блокчейном без передачі приватних ключів. Перехід від моделі «клієнт-сервер» до децентралізованих систем вимагає нових підходів до архітектури, безпеки та оптимізації витрат (газ-комісій). Через складність управління станом транзакцій та специфіку UX, систематизація інженерних практик розробки dApps є критично важливою для забезпечення надійності та масштабованості сучасних вебсервісів.

Метою дослідження є аналіз сучасних архітектурних підходів та інженерних практик розробки децентралізованих вебзастосунків на базі блокчейну, систематизація вимог до їхнього проєктування та вивчення технологічних рішень для забезпечення безпеки, масштабованості й зручності використання [1].

Децентралізований застосунок базується на моделі «клієнт-блокчейн», що замінює класичну схему «клієнт-сервер». Архітектура включає три рівні: на рівні блокчейну смартконтракти (Solidity, Rust) реалізують логіку в детермінованому та невідворотному вигляді; проміжний рівень (JSON-RPC шлюзи та API) забезпечує зв'язок фронтенду з мережею; на рівні фронтенду інтерфейси взаємодіють із гаманцями (як MetaMask) для безпечного підписання транзакцій.

Така архітектура потребує переосмислення методів кешування та управління станом через високі затримки та вартість операцій у розподілених мережах [1, 4].

Безпека у блокчейн-застосунках досягається через комбінацію криптографічних механізмів, формальної верифікації та регулярного аудиту коду. Оскільки смартконтракти після розгортання зазвичай є незмінними (immutability), будь-які вразливості стають постійними, тому сучасна практика включає статичний аналіз коду (Slither, Mythril), динамічне тестування та зовнішні професійні аудити [4, 5]. На рівні фронтенду критичного значення набуває безпечна взаємодія з гаманцями: фішингові атаки та підробки сайтів потребують імплементації стандартів EIP (Ethereum Improvement Proposals), перевірки адрес отримувачів та попередження користувача перед підписанням. Використання перевірених бібліотек (Web3.js, ethers.js, viem) спрощує розробку, проте вимагає суворого дотримання код-рев'ю та моніторингу активності контрактів [1].

Сучасний технологічний стек розробки dApps зазвичай включає: смартконтракти (мова Solidity, фреймворки Hardhat або Foundry), фронтенд (React/Next.js, Vue), бібліотеки взаємодії з блокчейном (Web3.js, ethers.js, viem), індексування та запити до даних (The Graph) та засоби перевірки коду (Slither, Mythril).

Вибір інструментів впливає на швидкість розробки, безпеку та масштабованість. Hardhat забезпечує цілісне середовище розробки з вбудованою мережею для тестування та інтеграцією з популярними інструментами верифікації [2]. Foundry, написаний на Rust, пропонує швидший процес тестування та написання тестів на Solidity замість JavaScript [3]. Бібліотеки взаємодії з блокчейном спрощують підписання транзакцій та роботу з контрактами, тоді як OpenZeppelin надає перевірені реалізації стандартів безпеки для смартконтрактів [4].

Отже, провадження архітектурних підходів і інженерних практик у розробку децентралізованих вебзастосунків дозволяє забезпечити безпечну та надійну взаємодію користувачів із блокчейном. Використання сучасного

технологічного стеку, автоматизованих інструментів для тестування та візуалізації, а також систематизація процесів розробки сприяє підвищенню масштабованості, швидкості розробки та зменшенню впливу людських помилок. Це створює стабільну основу для розвитку Web 3.0 та впровадження нових децентралізованих сервісів.

Список використаних джерел:

1. Ethereum Development Documentation. Official Ethereum Protocol Documentation. URL: https://ethereum.org/en/developers/docs/ (дата звернення: 16.03.2025).
2. Hardhat Documentation. Ethereum development environment. URL: https://hardhat.org/ (дата звернення: 16.03.2025).
3. Foundry Book. Smart Contract Development Suite. URL: https://book.getfoundry.sh/ (дата звернення: 16.03.2025).
4. OpenZeppelin Contracts. Standard Smart Contract Library. URL: https://docs.openzeppelin.com/contracts/ (дата звернення: 16.03.2025).
5. Security Considerations for Smart Contracts / S. DeFi Security. Ethereum Foundation. URL: https://ethereum.org/en/developers/docs/smart-contracts/security/ (дата звернення: 16.03.2025).

УДК 004.42

УПРАВЛІННЯ З'ЄДНАННЯМИ З БАЗОЮ ДАНИХ У FLASK-ЗАСТОСУНКУ ДЛЯ ЗАПОБІГАННЯ БЛОКУВАННЮ SQLite

Валовий А.Є
angegter@gmail.com
Черкаський державний фаховий бізнес-коледж
Фальченко Н.Г.
м.Черкаси, Україна

Під час розробки веб-застосунку для автоматизованої перевірки алгоритмічних задач на Python виникла необхідність зберігати результати перевірок у базі даних. Як сховище було обрано SQLite через його простоту та