УДК 004.41:004.65

Д. В. Мунтян[1]
В. І. Перепелиця[1]
В. О. Зозуля[1]

АРХІТЕКТУРНІ РІШЕННЯ ТА СТРУКТУРА БАЗИ ДАНИХ ДЛЯ WEB-ЗАСТОСУНКУ ОБЛІКУ РОБОЧОГО ЧАСУ ПРАЦІВНИКІВ

[1]Вінницький національний технічний університет

Анотація

У даній роботі досліджується питання вибору та оптимізації архітектури для систем автоматизованого обліку робочого часу працівників. Проведено порівняльний аналіз монолітної та трирівневої клієнт-серверної архітектур з точки зору швидкодії, безпеки та зручності масштабування ресурсу. Обґрунтовано вибір трирівневої моделі з використанням стеку React, NestJS та реляційної СУБД PostgreSQL для забезпечення надійної фіксації часу та формування аналітичної звітності, що дозволяє підвищити прозорість управління людськими ресурсами та ефективність підприємства.

Ключові слова: WEB-застосунок, облік часу, трирівнева архітектура, клієнт-серверна модель, автоматизація, база даних, тайм-менеджмент.

Abstract

This paper investigates the selection and optimization of architecture for automated employee time tracking systems. A comparative analysis of monolithic and three-tier client-server architectures is conducted in terms of performance, security, and resource scalability. The choice of a three-tier model using the React, NestJS, and PostgreSQL relational DBMS stack is justified to ensure reliable time tracking and analytical reporting, which enhances the transparency of human resource management and enterprise efficiency.

Keywords: WEB-application, time tracking, three-tier architecture, client-server model, automation, database, time management.

Вступ

Ефективне управління сучасним підприємством безпосередньо залежить від можливості точного планування та обліку робочого часу співробітників. У період глобального переходу до гібридних та віддалених форматів роботи, потреба у впровадженні високопродуктивних автоматизованих систем стрімко зростає. Дослідження в галузі корпоративного менеджменту підтверджують, що використання сучасних WEB-інструментів дозволяє значно підвищити якість моніторингу завантаженості команд та уникнути фінансових втрат [1].

Проте процес розробки таких систем стикається з низкою технічних викликів: необхідністю підтримки цілісності транзакцій (тайм-логів), забезпеченням безпеки комерційної таємниці, а також масштабованістю при зростанні штату компанії [2]. Для вирішення цих завдань необхідно обрати оптимальне архітектурне рішення. Основними підходами до проєктування WEB-застосунків є монолітна архітектура та багаторівнева (зокрема, трирівнева клієнт-серверна) архітектура.

Сьогодні більшість організацій стикаються з проблемою вибору між цими двома моделями. Наприклад, монолітна архітектура забезпечує швидкість початкового розгортання, проте трирівнева архітектура дозволяє ефективніше розподіляти навантаження та гарантує вищий рівень безпеки завдяки ізоляції бази даних [3]. Затримки в обробці запитів або збої в системі обліку можуть призвести до помилок у нарахуванні заробітної плати та зниженні довіри в колективі.

Метою дослідження є порівняльний аналіз архітектурних рішень (монолітного та трирівневого) для оптимізації процесів автоматизованого обліку робочого часу працівників у сучасних умовах.

Результати дослідження

Ефективність систем автоматизації тайм-менеджменту безпосередньо залежить від обраної архітектури застосунку, яка має забезпечувати мінімальний час відгуку інтерфейсу та стабільну роботу з базою даних. У ході дослідження встановлено, що для фіксації робочого часу найбільш доцільним є використання реляційних баз даних, де інформація чітко структурована за сутностями (Користувачі, Проєкти, Завдання, Тайм-логи). Такий підхід дозволяє перетворювати сирі записи про години роботи в структуровану інформацію, придатну для швидкої генерації звітів та аналітики.

Першою дослідженою моделлю стала традиційна монолітна архітектура, де клієнтський інтерфейс, серверна логіка та взаємодія з базою даних об'єднані в єдиний програмний блок (рис. 1). Головною перевагою моноліту є відносна простота розробки та тестування на початкових етапах життєвого циклу проєкту [4]. Однак при масштабуванні підприємства така модель виявляє критичні недоліки: зміна логіки таймера вимагає перерозгортання всієї системи, а навантаження на сервер зростає непропорційно.

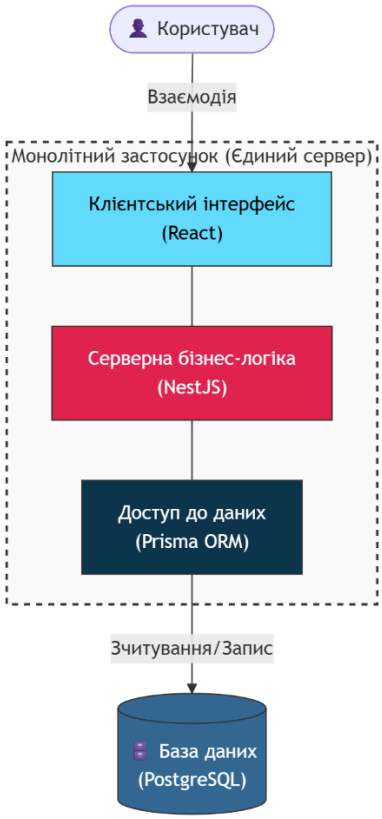

Рисунок 1 – Архітектура монолітного WEB-застосунку

Другим архітектурним рішенням є трирівнева клієнт-серверна архітектура (рис. 2), яка розділяє систему на логічно незалежні шари. Рівень презентації (Frontend) реалізується за допомогою бібліотеки React, що забезпечує миттєвий відгук інтерфейсу. Рівень бізнес-логіки (Backend) розробляється на фреймворку NestJS, обробляючи REST API запити та валідуючи дані. Третім шаром виступає рівень даних на базі реляційної СУБД PostgreSQL [5].

Використання трирівневої архітектури є доцільним для корпоративних систем, де критично важливою є гнучкість та безпека. Завдяки ізоляції бази даних від клієнтської частини, доступ до комерційної інформації здійснюється виключно через захищений сервер (із застосуванням JWT-авторизації) [6]. Крім

того, навантаження ефективно розділяється: браузер користувача відповідає за рендеринг графіків, а сервер виконує агрегацію даних для звітів.

Для забезпечення надійної та швидкої взаємодії між сервером (NestJS) та базою даних (PostgreSQL) доцільно використовувати сучасні інструменти об'єктно-реляційного відображення, такі як Prisma ORM. Це дозволяє уникнути аномалій при записі даних кількома працівниками одночасно та гарантує транзакційність (ACID).

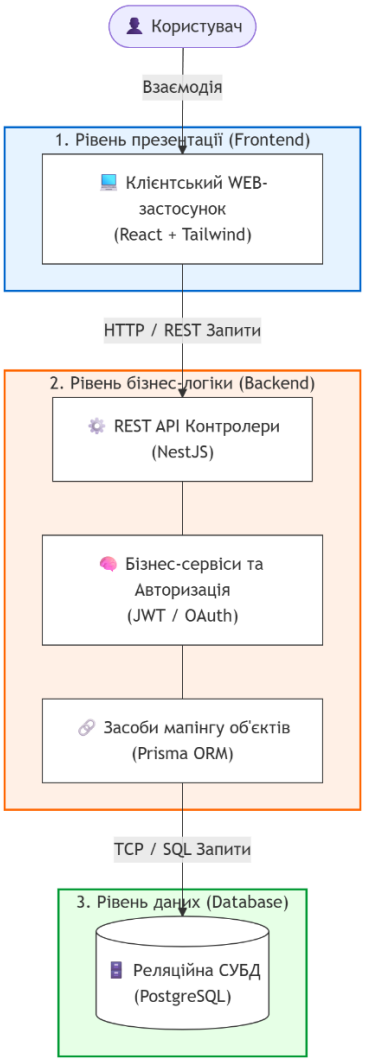

Рисунок 2 – Трирівнева клієнт-серверна архітектура WEB-застосунку

Проведений порівняльний аналіз дозволив визначити ключові критерії вибору. Основним параметром стала масштабованість та паралельна розробка: трирівнева архітектура демонструє абсолютну перевагу, дозволяючи фронтенд- та бекенд-розробникам працювати незалежно [7]. Натомість моноліт виявляється ефективним лише для невеликих локальних інструментів без перспективи розширення функціоналу.

З точки зору підтримки та оновлення системи, клієнт-серверна модель є значно гнучкішою. Вона дозволяє легко підключати нові клієнтські платформи (наприклад, мобільні застосунки) до єдиного API без зміни бізнес-логіки сервера [8].

Підводячи підсумок порівняння, можна стверджувати, що для систем оперативного обліку часу пріоритетною є трирівнева клієнт-серверна архітектура. Вибір правильної структури та використання реляційної бази даних дозволяє гарантувати цілісність логів, швидку генерацію дашбордів та високий рівень корпоративної безпеки.

Висновки

Результати проведеного дослідження підтверджують, що вибір оптимальної архітектури є критичним фактором для успішної розробки WEB-застосунку «Облік часу». Порівняльний аналіз показав, що для систем, які потребують надійної взаємодії багатьох користувачів та швидкого формування звітності, найбільш доцільним є використання трирівневої клієнт-серверної архітектури. Вона забезпечує необхідний рівень модульності та безпеки, перевершуючи традиційні монолітні рішення. Використання сучасного стеку технологій (React, NestJS) у поєднанні з реляційною СУБД PostgreSQL та Prisma ORM дозволяє створити високопродуктивну платформу, здатну автоматизувати рутинні процеси тайм-менеджменту. Впровадження таких архітектурних рішень суттєво підвищує оперативність управління персоналом та мінімізує адміністративні витрати. Перспективи подальших досліджень полягають в інтеграції розробленого WEB-ресурсу з зовнішніми системами (корпоративними месенджерами та таск-трекерами) для створення єдиної екосистеми управління ресурсами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. - Київ : Фабула, 2019. – 352 с.
2. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2012. – 560 p.
3. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. – O'Reilly Media, 2020. – 432 p.
4. Тулашвілі, Ю**.** Web-програмування. MERN fullstack розробка веб-додатків: навчальний посібник. Луцьк: Луцький національний технічний університет, 2023 URL: lntu.edu.ua. – 243 с.
5. NestJS Framework Documentation. Fundamentals and Techniques. URL: https://docs.nestjs.com/.
6. McDonald M. Web Security for Developers: Real Threats, Practical Defense. – San Francisco : No Starch Press, 2020. – 216 p.
7. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. – 2nd ed. –Sebastopol: O'Reilly Media, 2020. – 310 p.
8. Prisma ORM Documentation. Concepts and API Reference. URL: https://www.prisma.io/docs/.

Мунтян Дмитро Віталійович – студент кафедри комп’ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, e-mail: dmytromoon@gmail.com.

Перепелиця В’ячеслав Ігорович – доктор філософії, асистент кафедри «Комп’ютерних наук», e-mail: pvi_92@ukr.net; Вінницький національний технічний університет, Вінниця.

Зозуля Владислав Олександрович – студент кафедри комп’ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, e-mail: adunadan12@gmail.com.

Muntian Dmytro Vitaliyovych – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: dmytromoon @gmail.com.

Perepelytsia Viacheslav Ihorovych – PhD, assistant of the Department of Computer Science, e-mail: pvi_92@ukr.net; Vinnytsia National Technical University, Vinnytsia.

Zozulia Vladyslav Oleksandrovych – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: adunadan12@gmail.com.