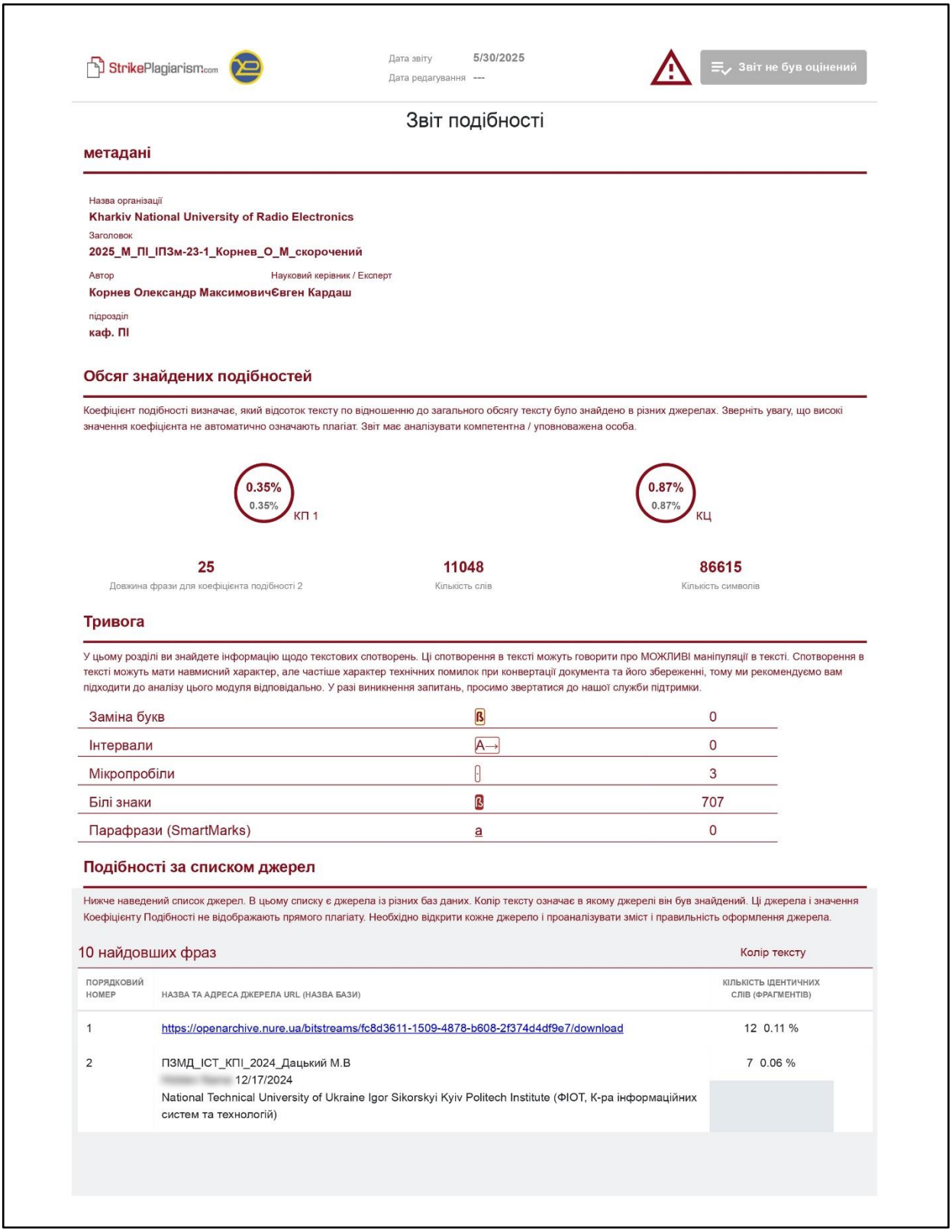
ДОДАТОК А

ЗВІТ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ НА УНІКАЛЬНІСТЬ ТЕКСТУ В БАЗІ ХНУРЕ



StrikePlagiarism.com

Дата звіту 5/30/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗм-23-1_Корнев_О_М_скорочений
Автор
Корнев Олександр Максимовичєвген Кардаш
Науковий керівник / Експерт
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.35% 0.35% КП 1

0.87% 0.87% КЦ

25 Довжина фрази для коефіцієнта подібності 2

11048 Кількість слів

86615 Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		3
Білі знаки		707
Парафрази (SmartMarks)		0

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/fc8d3611-1509-4878-b608-2f374d4df9e7/download	12 0.11 %
2	ПЗМД_ІСТ_КПІ_2024_Дацький М.В [redacted] 12/17/2024 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (ФІОТ, К-ра інформаційних систем та технологій)	7 0.06 %

Рисунок А.1 – Звіт з перевірки на унікальність (рисунок створено самостійно)

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

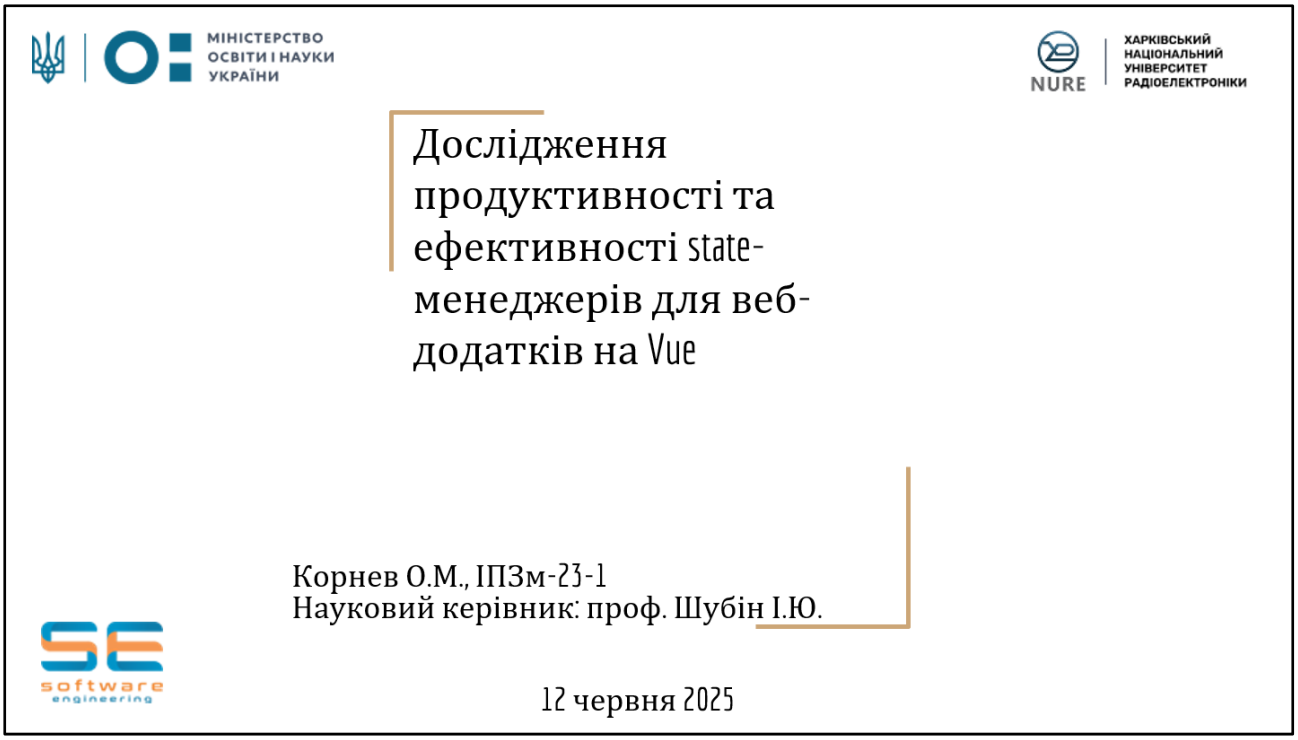


Рисунок Б.1 – Слайд 1 (рисунок створено самостійно)

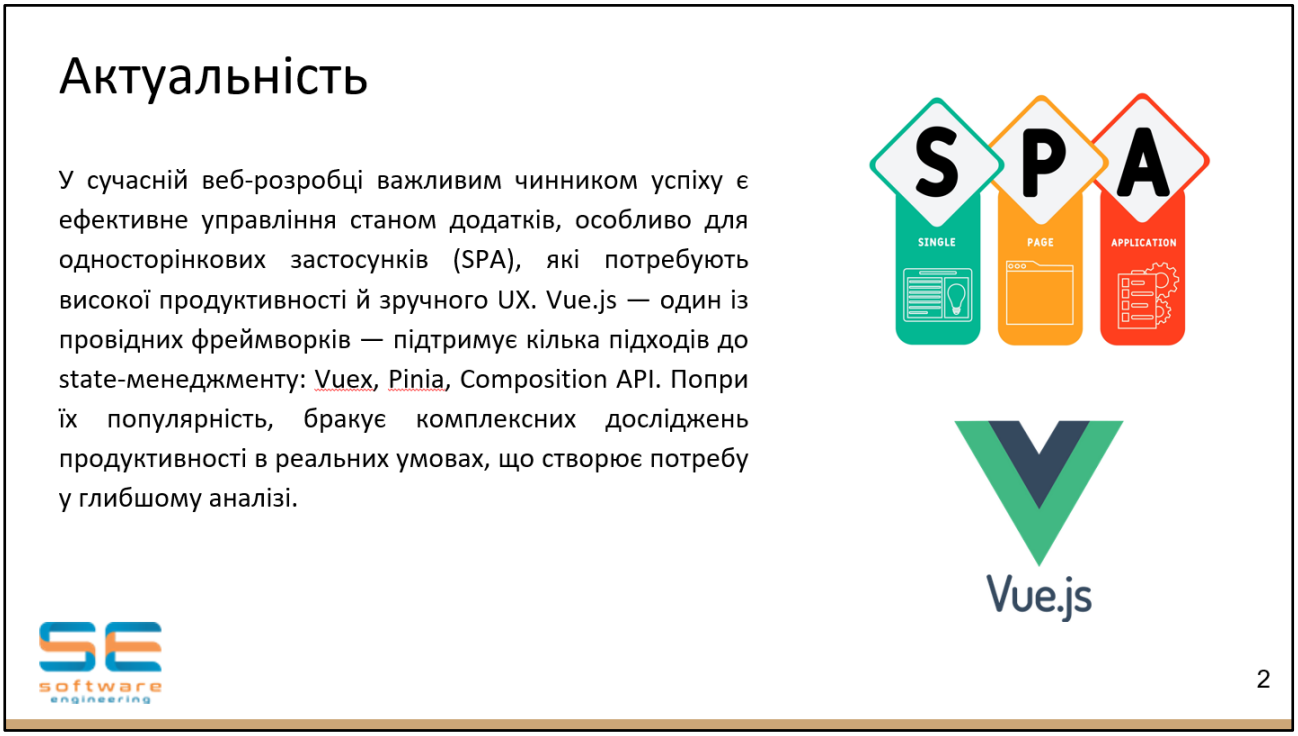


Рисунок Б.2 – Слайд 2 (рисунок створено самостійно)

Дослідження

Напрям дослідження
Порівняльне дослідження продуктивності та ефективності state-менеджерів Vuex, Pinia та Composition API у веб-додатках на Vue.js. Акцент зроблено на практичні аспекти вибору оптимального рішення для проєктів різного масштабу та напряму.

Об'єкт дослідження
Процес управління станом у веб-додатках, створених на базі Vue.js.

Предмет дослідження
Стейт менеджери Vuex, Pinia та Composition API, їх вплив на продуктівність та ефективність веб-додатків.

VUEX

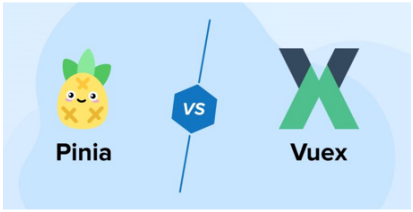
Composition API

software engineering

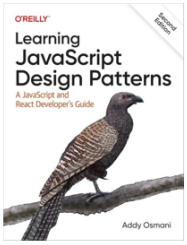
3

Рисунок Б.3 – Слайд 3 (рисунок створено самостійно)

Огляд літератури (аналогів)



Основні джерела та теорії
Проаналізовано офіційну документацію Vuex, Pinia та Composition API, наукові статті (*Pinia vs Vuex*, *FreeCodeCamp Guide on Reactive Programming*), а також прикладні книги (*Fullstack Vue*, *JavaScript Patterns*). Основна теоретична база — реактивне програмування, що лежить в основі сучасного state-менеджменту.



Виявлені прогалини
Наявні дослідження здебільшого обмежуються загальними висновками, не охоплюючи ключові технічні аспекти: затримку рендерингу, споживання пам'яті, зростання обсягу коду. Продуктивність часто не оцінюється експериментально. Також бракує уніфікованих сценаріїв для коректного порівняння state-менеджерів у реальних умовах.

software engineering

4

Рисунок Б.4 – Слайд 4 (рисунок створено самостійно)

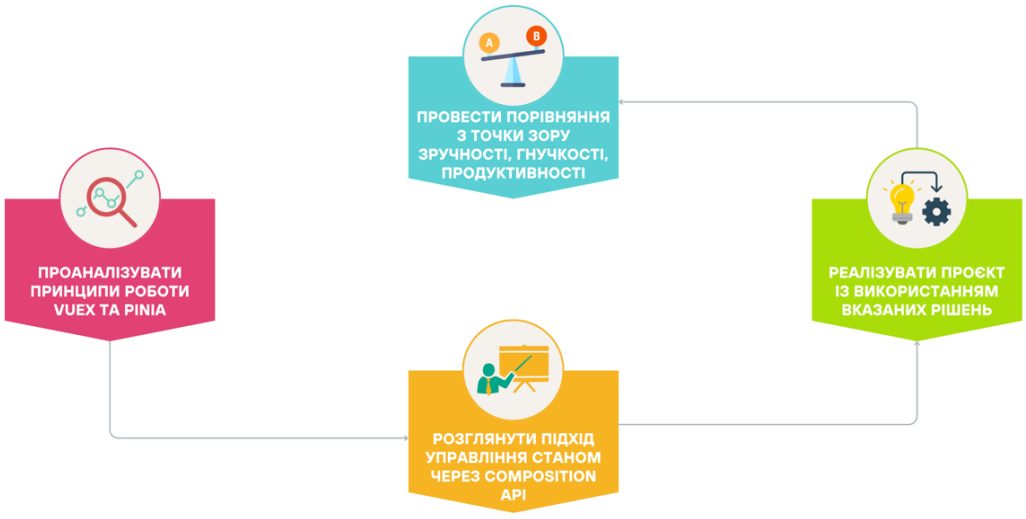
Постановка задачі

- Дослідити існуючі засоби управління станом у Vue.js
- Виявити переваги та недоліки кожного підходу
- Оцінити ефективність реалізації на практиці
- Сформувати рекомендації щодо вибору інструментів

5

Рисунок Б.5 – Слайд 5 (рисунок створено самостійно)

План завдання



6

Рисунок Б.6 – Слайд 6 (рисунок створено самостійно)

Методологія

Використані методи дослідження

Застосовано теоретичний аналіз (вивчення документації та наукових джерел), порівняльний аналіз (оцінка складності інтеграції та архітектурних особливостей), експериментальне тестування в реальних умовах.

Експериментальна частина

Розроблено три окремі монорепозиторні тестові додатки з однаковою функціональністю, але різними state-менеджерами. Порівняння виконувалось за такими метриками: час оновлення стану, затримка рендерингу, використання пам'яті, обсяг коду, рівень повторного використання логіки, темпи зростання коду при масштабуванні.

SE software engineering

7

Рисунок Б.7 – Слайд 7 (рисунок створено самостійно)

Інструменти та технології

- Vue.js (v3+), TypeScript, Composition API
- Для тестування: Chrome DevTools, Lighthouse, Jest, Cypress
- Для бекенду: Go (Clean Architecture) + PostgreSQL
- IDE: WebStorm, GoLand, Postman
- Розробка велась у монорепозиторії Nx з бібліотекою спільних компонентів

GO TS WS Jest GO Vue.js POSTMAN

SE software engineering

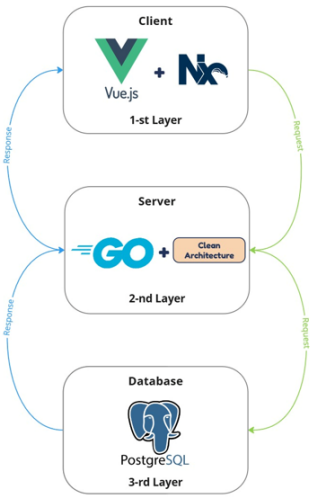
8

Рисунок Б.8 – Слайд 8 (рисунок створено самостійно)

Архітектура система для проведення експериментального дослідження

Розроблено тришарову клієнт-серверну архітектуру з розділенням відповідальності між клієнтом, сервером і базою даних.

Система реалізована у вигляді монорепозиторію з використанням NX, що забезпечує модульність і зручність масштабування.



9

Рисунок Б.9 – Слайд 9 (рисунок створено самостійно)

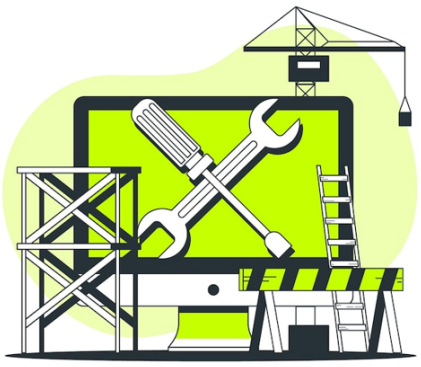
Зміст проведеного експерименту

Методи

Проведено експериментальне тестування трьох state-менеджерів у реальних умовах: Vuex, Pinia, Composition API. Для кожного з них було створено окремий функціонально ідентичний додаток.

Вхідні дані

Згенеровані тестові набори — масиви з понад 10 000 елементів, сценарії з частими змінами стану, багатократними оновленнями інтерфейсу, а також симуляція типових дій користувача (додавання, редагування, видалення, пошук).



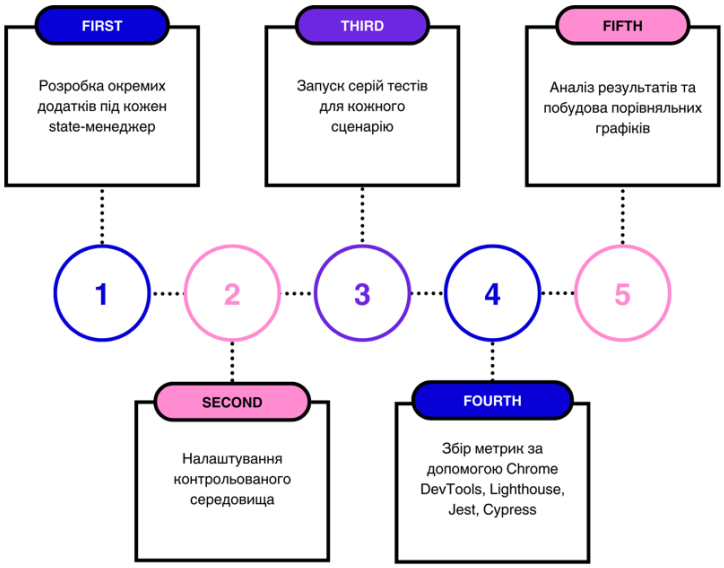
10

Рисунок Б.10 – Слайд 10 (рисунок створено самостійно)

Зміст проведеного експерименту

Критерії оцінювання:

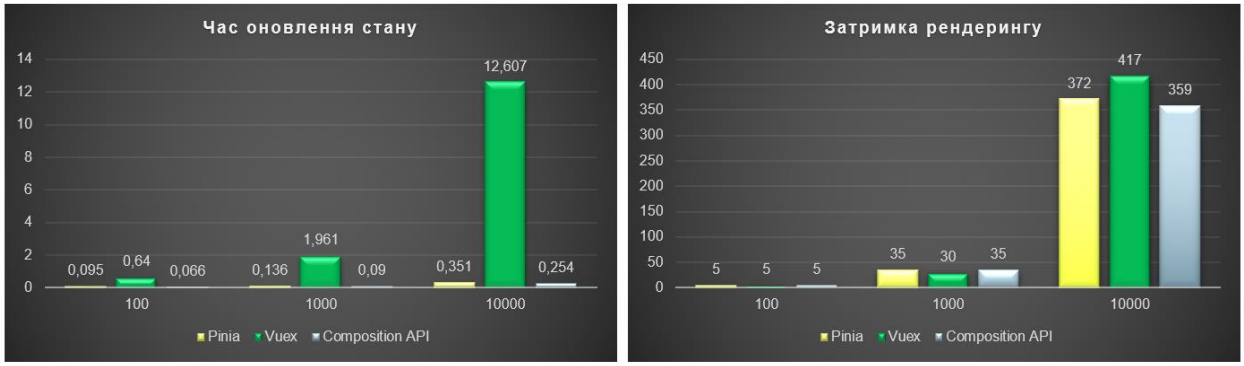
- Час оновлення стану
- Затримка рендерингу
- Використання пам’яті
- Обсяг коду
- Рівень повторного використання логіки
- Темпи зростання коду при масштабуванні



11

Рисунок Б.11 – Слайд 11 (рисунок створено самостійно)

Результати експерименту



12

Рисунок Б.12 – Слайд 12 (рисунок створено самостійно)

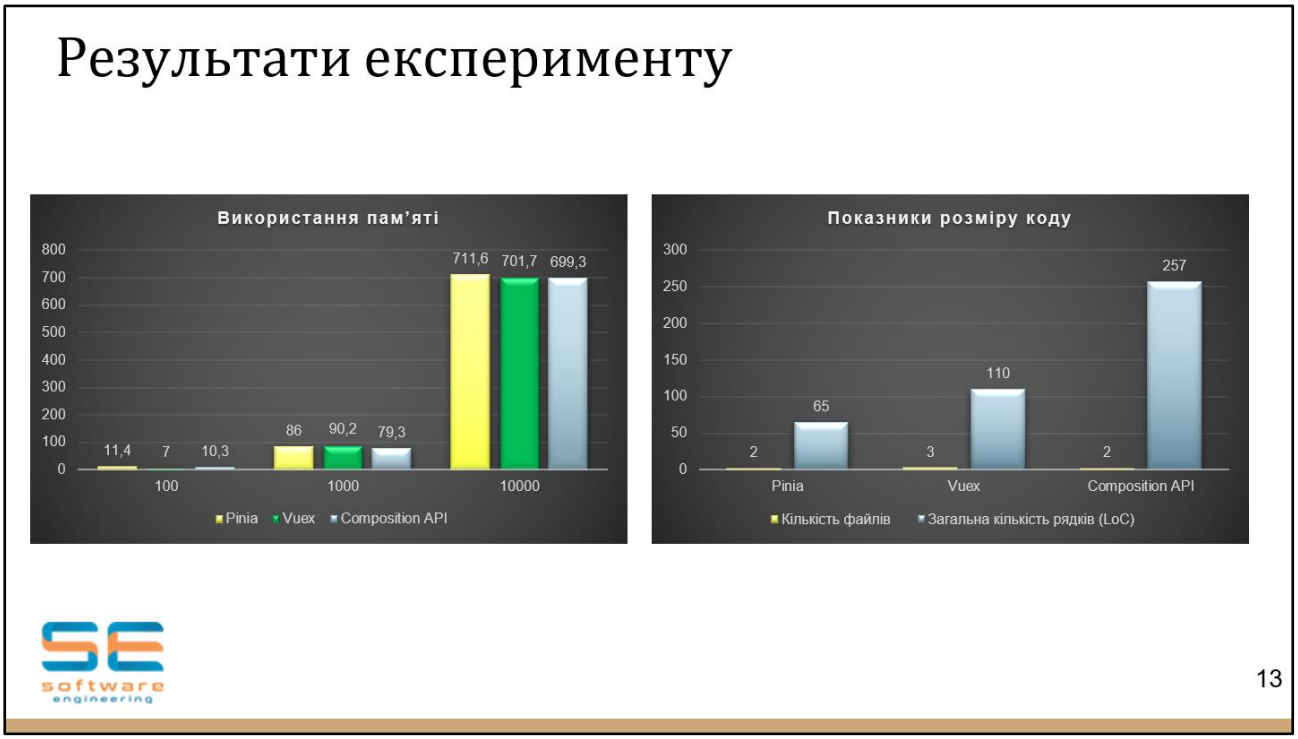


Рисунок Б.13 – Слайд 13 (рисунок створено самостійно)

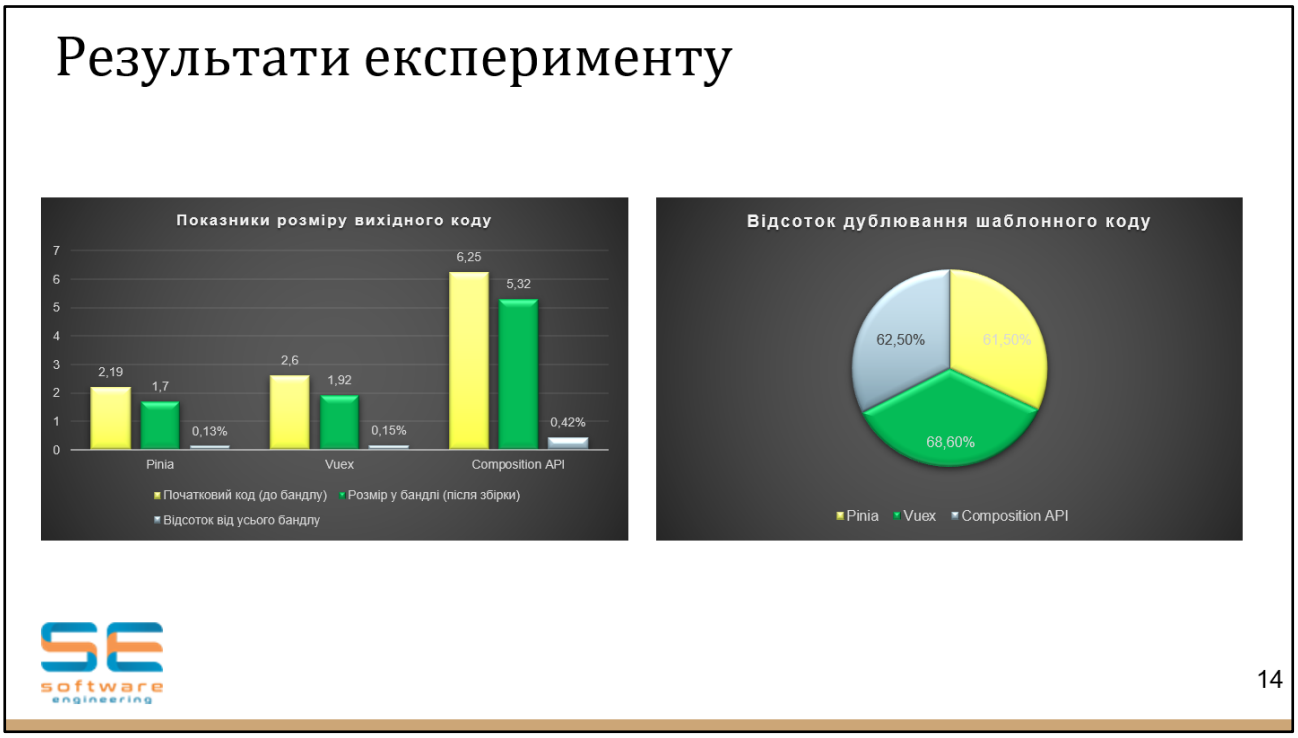


Рисунок Б.14 – Слайд 14 (рисунок створено самостійно)

Аналіз отриманих результатів

Критерій	Composition API	Pinia	Vuex
Час оновлення стану	< 0.3 мс	0.3–1 мс	> 10 мс
Затримка рендерингу	< 360 мс	360–400 мс	> 400 мс
Використання пам'яті	< 700 МБ	700–750 МБ	> 750 МБ
Розмір коду (рядків)	> 250	60–100	> 100
Розмір у бандлі (збірка)	> 5 KB	< 2 KB	< 2 KB
Дублювання коду	< 63%	60–65%	> 65%

15

Рисунок Б.15 – Слайд 15 (рисунок створено самостійно)

Аналіз отриманих результатів

Інтерпретація

- Composition API — для високопродуктивних SPA
- Pinia — для середніх та великих проєктів
- Vuex — для легасі-систем на Vue 2.

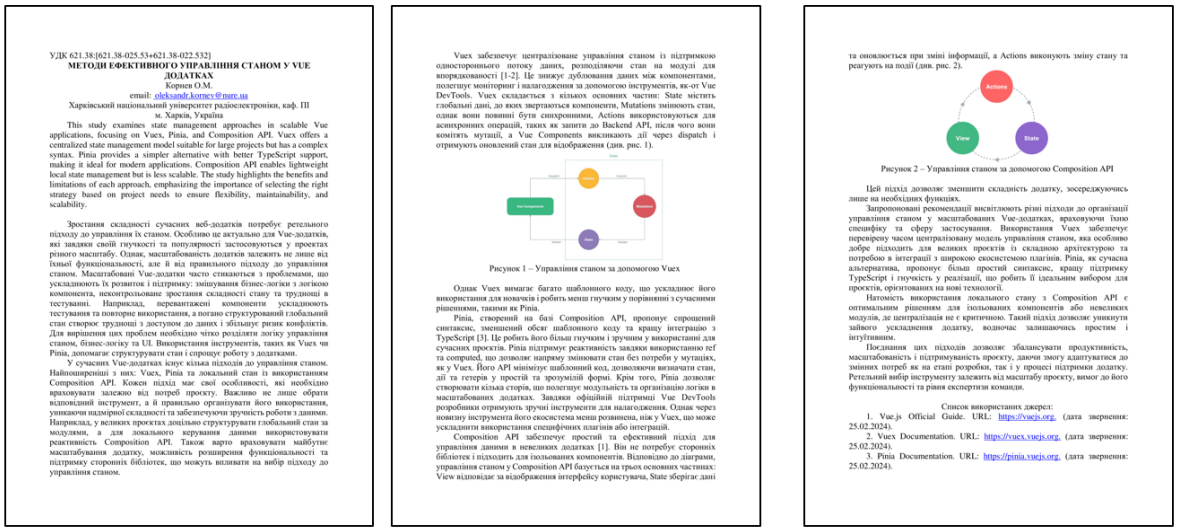
Вплив на практику

Результати підтверджують перехід екосистеми Vue до сучасних state-менеджерів і дають чіткі рекомендації для вибору в нових проєктах.

16

Рисунок Б.16 – Слайд 16 (рисунок створено самостійно)

Публікація результатів



УДК 621.38:[621.38-025.53+621.38-022.532]

МЕТОДИ ЕФЕКТИВНОГО УПРАВЛІННЯ СТАНОМ У VUE ДОДАТКАХ

Корнев О.М.
email: oleksandr.kornev@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study examines state management approaches in scalable Vue applications, focusing on Vuex, Pinia, and Composition API. Vuex offers a centralized state management model suitable for large projects but has a complex syntax. Pinia provides a simpler alternative with better TypeScript support, making it ideal for modern applications. Composition API enables lightweight local state management but is less scalable. The study highlights the benefits and limitations of each approach, emphasizing the importance of selecting the right strategy based on project needs to ensure flexibility, maintainability, and scalability.

Зростання складності сучасних веб-додатків потребує ретельного підходу до управління їх станом. Особливо це актуально для Vue-додатків, які завдяки своїй гнучкості та популярності застосовуються у проектах різного масштабу. Однак, масштабованість додатків залежить не лише від їхньої функціональності, але й від правильного підходу до управління станом. Масштабовані Vue-додатки часто стикаються з проблемами, що ускладнюють їх розвиток і підтримку: змішування бізнес-логіки з логікою компонента, неконтрольоване зростання складності стану та труднощі в тестуванні. Наприклад, перевантажені компоненти ускладнюють тестування та повторне використання, а погано структурований глобальний стан створює труднощі з доступом до даних і збільшує ризик конфліктів. Для вирішення цих проблем необхідно чітко розділяти логіку управління станом, бізнес-логіку та UI. Використання інструментів, таких як Vuex чи Pinia, допомагає структурувати стан і спрощує роботу з додатками.

У сучасних Vue-додатках існує кілька підходів до управління станом. Найпоширеніші з них: Vuex, Pinia та локальний стан із використанням Composition API. Кожен підхід має свої особливості, які необхідно враховувати залежно від потреб проєкту. Важливо не лише обрати відповідний інструмент, а й правильно організувати його використання, уникаючи надмірної складності та забезпечуючи зручність роботи з даними. Наприклад, у великих проєктах доцільно структурувати глобальний стан за модулями, а для локального керування даними використовувати реактивність Composition API. Також варто враховувати майбутнє масштабування додатку, можливість розширення функціональності та підтримку сторонніх бібліотек, що можуть вплинути на вибір підходу до управління станом.

Vuex забезпечує централізоване управління станом із підтримкою односторонього потоку даних, розподіляючи стан на модулі для впорядкованості [1-2]. Це знижує дублювання даних між компонентами, полегшує моніторинг і налагодження за допомогою інструментів, як-от Vue DevTools. Vuex складається з кількох основних частин: State містить глобальні дані, до яких звертаються компоненти, Mutations змінюють стан, однак вони повинні бути синхронними, Actions використовуються для асинхронних операцій, таких як запити до Backend API, після чого вони комітять мутації, а Vue Components викликають дії через dispatch і отримують оновлений стан для відображення (див. рис. 1).

Рисунок 1 – Управління станом за допомогою Vuex

Однак Vuex вимагає багато шаблонного коду, що ускладнює його використання для новачків і робить менш гнучким у порівнянні з сучасними рішеннями, такими як Pinia.

Pinia, створений на базі Composition API, пропонує спрощений синтаксис, зменшений обсяг шаблонного коду та кращу інтеграцію з TypeScript [3]. Це робить його більш гнучким і зручним у використанні для сучасних проєктів. Pinia підтримує реактивність завдяки використанню ref та computed, що дозволяє напряму змінювати стан без потреби у мутаціях, як у Vuex. Його API мінімізує шаблонний код, дозволяючи визначати стан, дії та гетерів у простій та зрозумілій формі. Крім того, Pinia дозволяє створювати кілька сторів, що полегшує модульність та організацію логіки в масштабованих додатках. Завдяки офіційній підтримці Vue DevTools розробники отримують зручні інструменти для налагодження. Однак через новизну інструмента його екосистема менш розвинена, ніж у Vuex, що може ускладнити використання специфічних плагінів або інтеграцій.

Composition API забезпечує простий та ефективний підхід для управління даними в невеликих додатках [1]. Він не потребує сторонніх бібліотек і підходить для ізольованих компонентів. Відповідно до діаграми, управління станом у Composition API базується на трьох основних частинах: View відповідає за відображення інтерфейсу користувача, State зберігає дані та оновлюється при зміні інформації, а Actions виконують зміну стану та реагують на події (див. рис. 2).

Рисунок 2 – Управління станом за допомогою Composition API

Цей підхід дозволяє зменшити складність додатку, зосереджуючись лише на необхідних функціях.

Запропоновані рекомендації висвітлюють різні підходи до організації управління станом у масштабованих Vue-додатках, враховуючи їхню специфіку та сферу застосування. Використання Vuex забезпечує перевірену часом централізовану модель управління станом, яка особливо добре підходить для великих проєктів із складною архітектурою та потребою в інтеграції з широкою екосистемою плагінів. Pinia, як сучасна альтернатива, пропонує більш простий синтаксис, кращу підтримку TypeScript і гнучкість у реалізації, що робить її ідеальним вибором для проєктів, орієнтованих на нові технології.

Натомість використання локального стану з Composition API є оптимальним рішенням для ізольованих компонентів або невеликих модулів, де централізація не є критичною. Такий підхід дозволяє уникнути зайвого ускладнення додатку, водночас залишаючись простим і інтуїтивним.

Поєднання цих підходів дозволяє збалансувати продуктивність, масштабованість і підтримуваність проєкту, даючи змогу адаптуватися до змінних потреб як на етапі розробки, так і у процесі підтримки додатку. Ретельний вибір інструменту залежить від масштабу проєкту, вимог до його функціональності та рівня експертизи команди.

Список використаних джерел:

1. Vue.js Official Guide. URL: https://vuejs.org. (дата звернення: 25.02.2024).
2. Vuex Documentation. URL: https://vuex.vuejs.org. (дата звернення: 25.02.2024).
3. Pinia Documentation. URL: https://pinia.vuejs.org. (дата звернення: 25.02.2024).

software engineering

17

Рисунок Б.17 – Слайд 17 (рисунок створено самостійно)

Публікація результатів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ДИПЛОМ

НАГОРОДЖУЄТЬСЯ

за **II місце**

здобувач вищої освіти **КОРНЕВ ОЛЕКСАНДР**

ХНУРЕ, каф. ПІ

науковий керівник: ШУБІН Ігор Юрійович

за участь у 29-му Міжнародному молодіжному форумі

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

за доповідь: «МЕТОДИ ЕФЕКТИВНОГО УПРАВЛІННЯ СТАНОМ У VUE ДОДАТКАХ»

В.о. ректора Юрій РОМАНЕНКОВ

17.04.2025 р.

software engineering

18

Рисунок Б.18 – Слайд 18 (рисунок створено самостійно)

Підсумки

Реалістичність та корисність

Отримані результати базуються на реальних сценаріях і типових задачах для сучасних Vue-додатків. Тестування охопило ключові аспекти продуктивності, тому висновки є практично застосовними для розробників при виборі state-менеджера.

Можливий розвиток досліджень:

- Дослідження масштабування state-менеджерів у високонавантажених системах
- Порівняння з альтернативами за межами Vue (наприклад, Redux, Recoil)
- Вивчення продуктивності в умовах SSR, офлайн-режимів або мобільних платформ
- Розробка комбінованих підходів, що об’єднують гнучкість Composition API та модульність Pinia

SE software engineering

19

Рисунок Б.19 – Слайд 19 (рисунок створено самостійно)

ДОДАТОК В

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

УДК 621.38:[621.38-025.53+621.38-022.532]

МЕТОДИ ЕФЕКТИВНОГО УПРАВЛІННЯ СТАНОМ У VUE ДОДАТКАХ

Корнев О.М.
email: oleksandr.kornev@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study examines state management approaches in scalable Vue applications, focusing on Vuex, Pinia, and Composition API. Vuex offers a centralized state management model suitable for large projects but has a complex syntax. Pinia provides a simpler alternative with better TypeScript support, making it ideal for modern applications. Composition API enables lightweight local state management but is less scalable. The study highlights the benefits and limitations of each approach, emphasizing the importance of selecting the right strategy based on project needs to ensure flexibility, maintainability, and scalability.

Зростання складності сучасних веб-додатків потребує ретельного підходу до управління їх станом. Особливо це актуально для Vue-додатків, які завдяки своїй гнучкості та популярності застосовуються у проектах різного масштабу. Однак, масштабованість додатків залежить не лише від їхньої функціональності, але й від правильного підходу до управління станом. Масштабовані Vue-додатки часто стикаються з проблемами, що ускладнюють їх розвиток і підтримку: змішування бізнес-логіки з логікою компонента, неконтрольоване зростання складності стану та труднощі в тестуванні. Наприклад, перевантажені компоненти ускладнюють тестування та повторне використання, а погано структурований глобальний стан створює труднощі з доступом до даних і збільшує ризик конфліктів. Для вирішення цих проблем необхідно чітко розділяти логіку управління станом, бізнес-логіку та UI. Використання інструментів, таких як Vuex чи Pinia, допомагає структурувати стан і спрощує роботу з додатками.

У сучасних Vue-додатках існує кілька підходів до управління станом. Найпоширеніші з них: Vuex, Pinia та локальний стан із використанням Composition API. Кожен підхід має свої особливості, які необхідно враховувати залежно від потреб проєкту. Важливо не лише обрати відповідний інструмент, а й правильно організувати його використання, уникаючи надмірної складності та забезпечуючи зручність роботи з даними. Наприклад, у великих проєктах доцільно структурувати глобальний стан за модулями, а для локального керування даними використовувати реактивність Composition API. Також варто враховувати майбутнє масштабування додатку, можливість розширення функціональності та підтримку сторонніх бібліотек, що можуть впливати на вибір підходу до управління станом.

Рисунок В.1 – Перша сторінка матеріалу (рисунок створено самостійно)

Vuex забезпечує централізоване управління станом із підтримкою одностороннього потоку даних, розподіляючи стан на модулі для впорядкованості [1-2]. Це знижує дублювання даних між компонентами, полегшує моніторинг і налагодження за допомогою інструментів, як-от Vue DevTools. Vuex складається з кількох основних частин: State містить глобальні дані, до яких звертаються компоненти, Mutations змінюють стан, однак вони повинні бути синхронними, Actions використовуються для асинхронних операцій, таких як запити до Backend API, після чого вони комітять мутації, а Vue Components викликають дії через dispatch і отримують оновлений стан для відображення (див. рис. 1).

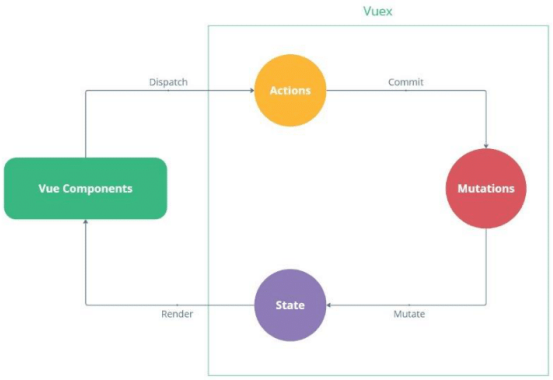


Рисунок 1 – Управління станом за допомогою Vuex

Однак Vuex вимагає багато шаблонного коду, що ускладнює його використання для новачків і робить менш гнучким у порівнянні з сучасними рішеннями, такими як Pinia.

Pinia, створений на базі Composition API, пропонує спрощений синтаксис, зменшений обсяг шаблонного коду та кращу інтеграцію з TypeScript [3]. Це робить його більш гнучким і зручним у використанні для сучасних проєктів. Pinia підтримує реактивність завдяки використанню ref та computed, що дозволяє напряму змінювати стан без потреби у мутаціях, як у Vuex. Його API мінімізує шаблонний код, дозволяючи визначати стан, дії та гетерів у простій та зрозумілій формі. Крім того, Pinia дозволяє створювати кілька сторів, що полегшує модульність та організацію логіки в масштабованих додатках. Завдяки офіційній підтримці Vue DevTools розробники отримують зручні інструменти для налагодження. Однак через новизну інструмента його екосистема менш розвинена, ніж у Vuex, що може ускладнити використання специфічних плагінів або інтеграцій.

Composition API забезпечує простий та ефективний підхід для управління даними в невеликих додатках [1]. Він не потребує сторонніх бібліотек і підходить для ізольованих компонентів. Відповідно до діаграми, управління станом у Composition API базується на трьох основних частинах: View відповідає за відображення інтерфейсу користувача, State зберігає дані

Рисунок В.2 – Друга сторінка матеріалу (рисунок створено самостійно)

та оновлюється при зміні інформації, а Actions виконують зміну стану та реагують на події (див. рис. 2).

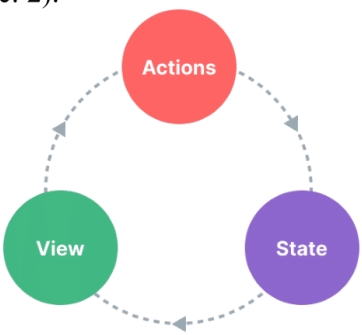


Рисунок 2 – Управління станом за допомогою Composition API

Цей підхід дозволяє зменшити складність додатку, зосереджуючись лише на необхідних функціях.

Запропоновані рекомендації висвітлюють різні підходи до організації управління станом у масштабованих Vue-додатках, враховуючи їхню специфіку та сферу застосування. Використання Vuex забезпечує перевірену часом централізовану модель управління станом, яка особливо добре підходить для великих проєктів із складною архітектурою та потребою в інтеграції з широкою екосистемою плагінів. Pinia, як сучасна альтернатива, пропонує більш простий синтаксис, кращу підтримку TypeScript і гнучкість у реалізації, що робить її ідеальним вибором для проєктів, орієнтованих на нові технології.

Натомість використання локального стану з Composition API є оптимальним рішенням для ізольованих компонентів або невеликих модулів, де централізація не є критичною. Такий підхід дозволяє уникнути зайвого ускладнення додатку, водночас залишаючись простим і інтуїтивним.

Поєднання цих підходів дозволяє збалансувати продуктивність, масштабованість і підтримуваність проєкту, даючи змогу адаптуватися до змінних потреб як на етапі розробки, так і у процесі підтримки додатку. Ретельний вибір інструменту залежить від масштабу проєкту, вимог до його функціональності та рівня експертизи команди.

Список використаних джерел:

1. Vue.js Official Guide. URL: https://vuejs.org. (дата звернення: 25.02.2024).

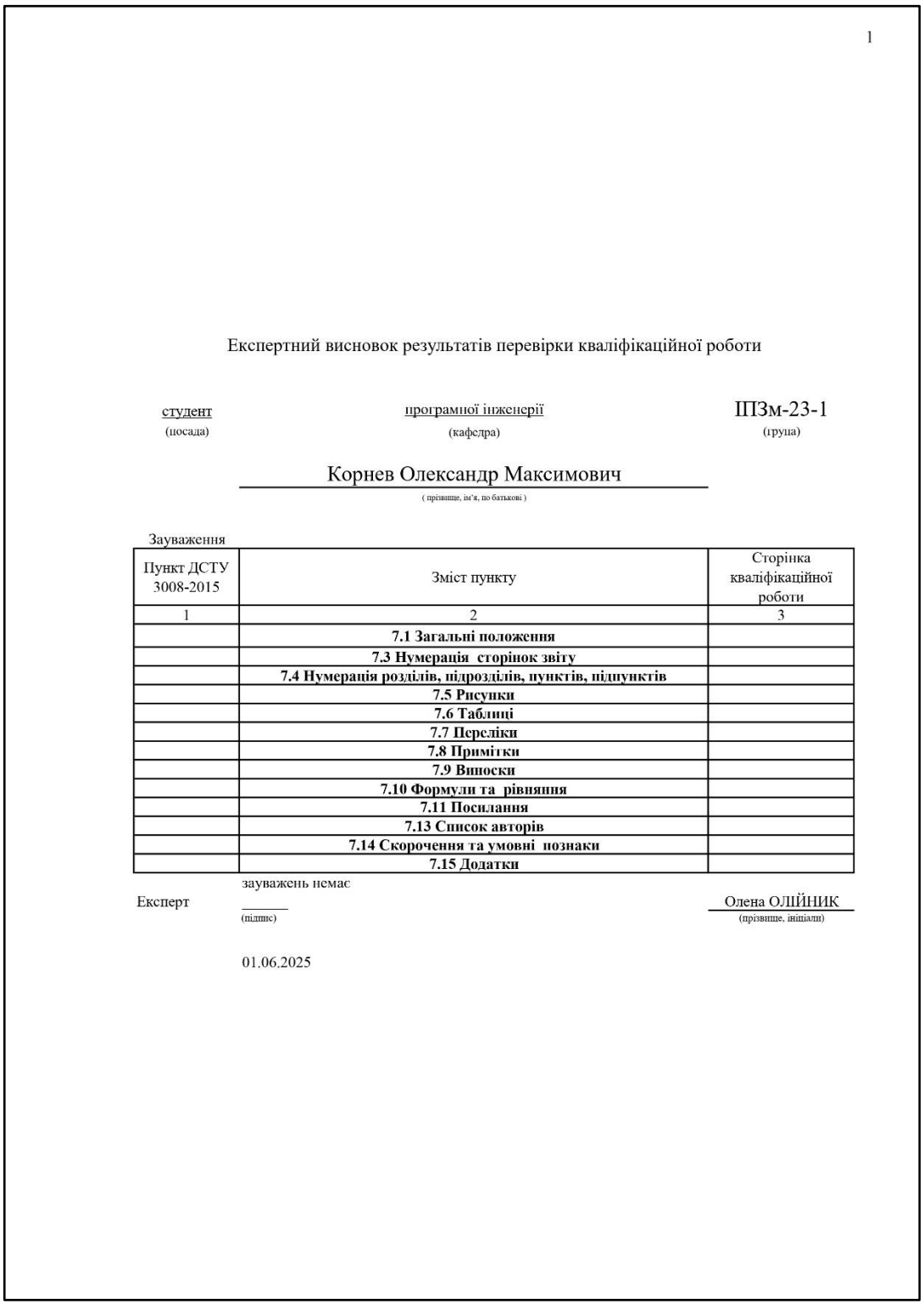
2. Vuex Documentation. URL: https://vuex.vuejs.org. (дата звернення: 25.02.2024).

3. Pinia Documentation. URL: https://pinia.vuejs.org. (дата звернення: 25.02.2024).

Рисунок В.3 – Третя сторінка матеріалу (рисунок створено самостійно)

Рисунок В.4 – Диплом переможця 29-го Молодіжного Форума (рисунок створено самостійно)

ДОДАТОК Г
ЕКСПЕРТНИЙ ВИСНОВОК РЕЗУЛЬТАТІВ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА ВІДПОВІДНІСТЬ ОФОРМЛЕННЯ ВИМОГАМ ДСТУ 3008: 2015



1

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-1 (група)

Корнев Олександр Максимович
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

01.06.2025

Рисунок Г.1 – Результати перевірки на нормоконтроль (рисунок створено самостійно)