Unified Core: кросплатформна бізнес-логіка на Rust/WASM

Денис Компанець[1], студент (ORCID: 0009-0002-2610-5129)
Олена Доля[1], канд. фіз.-мат. наук, доц. (ORCID: 0000-0003-2503-2634)
[1] *Київський національний університет будівництва і архітектури, Київ, Україна*

АНОТАЦІЯ

У роботі представлено результати аналізу та систематизації переваг запропонованого підходу, серед яких – майже нативна продуктивність, гарантії безпеки пам'яті, скорочення часу виходу продукту на ринок та зниження витрат на розробку. Розглянуто ключовий інструментарій (wasm-bindgen, UniFFI), що автоматизує міжмовну взаємодію, а також практичні виклики, пов'язані з серіалізацією даних та управлінням пам'яттю у WASM-середовищі. Висновки підкріплено аналізом кейсів із реальних проєктів (Figma, Cloudflare Workers, Mozilla), що підтверджує ефективність та перспективність архітектури для сучасних програмних систем.

Ключові слова: Unified Core, Rust, кросплатформеність, WebAssembly, кросплатформна розробка, ефективність.

1. ВСТУП

Сьогоднішній ринок програмного забезпечення ставить перед розробниками вимогу забезпечити доступність додатків на максимально широкому спектрі платформ. Розробка окремих нативних додатків для кожної операційної системи - Windows, macOS, Linux, iOS та Android - є надзвичайно витратним, ресурсоємним та складним завданням. Цей підхід призводить до необхідності підтримувати кілька кодових баз, що спричиняє значні проблеми з узгодженням функціональності, виправленням помилок та виведенням оновлень. Навіть найменші зміни в бізнес-логіці доводиться реалізовувати і тестувати багаторазово, що суттєво збільшує час та вартість розробки. Це створює нагальну потребу в нових архітектурних моделях, які дозволять уніфікувати високопродуктивний код, відокремлюючи його від платформно-специфічного інтерфейсу [4].

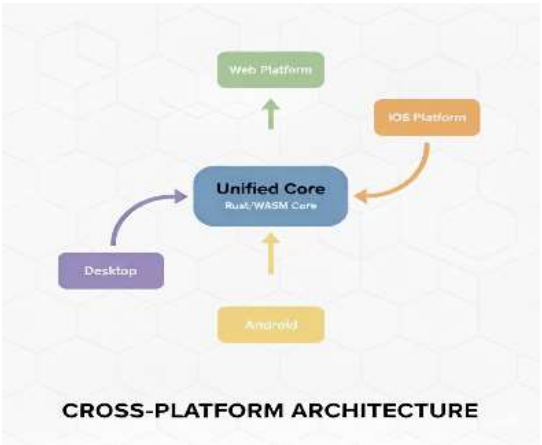

Рисунок 1. Загальна архітектура "Unified Core"

2. МЕТА

Метою даного дослідження є аналіз, систематизація та оцінка ефективності архітектури "Unified Core" на базі мови програмування Rust та технології WebAssembly як інструменту для створення кросплатформної бізнес-логіки.

3. КОНЦЕПЦІЯ "UNIFIED CORE" ЯК ПАРАДИГМА УНІФІКАЦІЇ БІЗНЕС-ЛОГІКИ

Термін "Unified Core" є метафорою, запозиченою з архітектури процесорів Intel, де інженери прагнуть створити єдине ядро, що поєднує високу продуктивність P-ядер та енергоефективність E-ядер для спрощення управління навантаженням. Аналогічно, у програмній архітектурі, "Unified Core" пропонує створити єдину, високопродуктивну кодову базу для бізнес-логіки, яка буде ефективно працювати в різних середовищах. Центральна ідея полягає у стратегічному розділенні архітектури на дві основні частини: "ядро" (core) та "оболонку" (shell). Ядро відповідає за всю критичну бізнес-логіку та обчислювально-інтенсивні завдання, такі як обробка даних, складні алгоритми та розрахунки. Оболонка, своєю чергою, є платформно-специфічною, відповідаючи за користувацький інтерфейс та взаємодію з нативними API [3]. Цей підхід кардинально відрізняється від традиційних гібридних фреймворків, які намагаються уніфікувати обидва шари. Завдяки зосередженню лише на уніфікації ядра, розробники можуть досягти продуктивності, що не поступається нативним рішенням, оскільки найскладніші обчислення виконуються в оптимізованому середовищі "ядра", тоді як UI залишається повністю нативним для кожної платформи. Це дозволяє поєднати переваги обох світів: уніфікацію коду, що забезпечує швидкість та економію ресурсів, і нативний UI/UX, що відповідає очікуванням користувачів [4].

4. ПЕРЕВАГИ RUST ТА WEB-ASSEMBLY

Однією з головних переваг Rust є безкомпромісна безпека пам'яті. Завдяки моделі володіння (ownership), запозичення (borrowing) та життєвих циклів (lifetimes), Rust запобігає цілому класу критичних помилок, що є поширеними у мовах, таких як C та C++ [1]. Компілятор Rust на етапі збирання проєкту виявляє та усуває витоки пам'яті, висячі покажчики та змагання даних (data races), що робить застосунки більш надійними та менш вразливими до атак. Окрім безпеки, Rust забезпечує високу продуктивність, порівнянну з C/C++, без накладних витрат на збирання сміття (garbage collection). Модель мови дозволяє писати безпечний багатопотоковий код, що є надзвичайно важливим для сучасних багатоядерних систем та додатків, що вимагають високого рівня паралелізму. WebAssembly ж

є портативним бінарним форматом, розробленим для ефективного виконання в браузерах з майже нативною швидкістю. Він усуває потребу в застарілих плагінах, таких як Flash, і забезпечує стабільну та передбачувану продуктивність на всіх сучасних веб-платформах. Ця технологія дозволяє переносити існуючий код, написаний на таких мовах, як C, C++ або Rust, у веб без необхідності його повного переписування. Завдяки цьому, команди розробників можуть повторно використовувати напрацьовані кодові бази та експертизу, скорочуючи цикл розробки та зберігаючи цінність існуючих проєктів.

5. ПРАКТИЧНІ КЕЙС-СТАДІ ТА ГАЛУЗЕВІ ТЕНДЕНЦІЇ

Використання уніфікованого ядра на Rust/WASM є не просто теоретичною концепцією, а перевіреним на практиці рішенням, що вже застосовується в індустрії. Досвід провідних компаній та фреймворків демонструє його життєздатність та ефективність. Одним із найяскравіших прикладів є Figma — провідний інструмент для веб-дизайну. Щоб досягти продуктивності на рівні десктопних застосунків, команда Figma реалізувала своє ядро на C++ та скомпілювала його у WebAssembly. Цей підхід дозволив забезпечити високопродуктивний рендеринг складної векторної графіки безпосередньо в браузері, що було неможливо для традиційних веб-технологій [2]. Досвід Figma демонструє, що WASM може служити «турбонаддувом» для веб-застосунків, дозволяючи виконувати інтенсивні обчислення та операції, які раніше вимагали нативних програм. Великі технологічні компанії, такі як Dropbox та Discord, також використовують Rust для критично важливих компонентів своїх систем [5]. Dropbox переписав свій механізм синхронізації з Python і C++ на Rust, що дозволило значно знизити навантаження на CPU і поліпшити багатопоточність. Discord використовує Rust у своїх високопродуктивних бекенд-сервісах, таких як відстеження статусу користувачів, що призвело до помітного зменшення затримок навіть під час пікових навантажень.[28] Ці приклади підтверджують, що Rust є надійним вибором для систем, що вимагають високої продуктивності, безпеки та масштабованості.

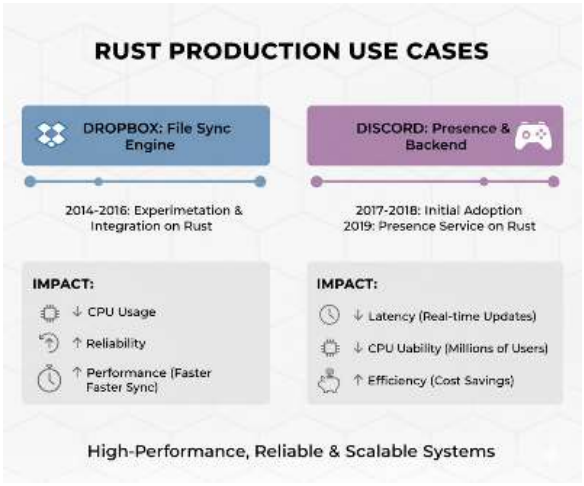

Рисунок 2. Індустріальні приклади використання Rust

Rinf, зокрема, заявляє про можливість 40-кратного прискорення бізнес-логіки порівняно з Dart, зберігаючи при цьому гнучкість та зовнішній вигляд нативного інтерфейсу, що забезпечується Flutter. Це дозволяє командам створювати високопродуктивні, але при цьому візуально багаті кросплатформені застосунки, мінімізуючи дублювання коду. Аналіз цих практичних кейс-стаді, від великих гравців до спеціалізованих фреймворків, показує, що архітектурний патерн уніфікованого ядра є перевіреним і життєздатним рішенням. Він є відповіддю на потребу в створенні високопродуктивних, безпечних та універсальних застосунків, які сьогодні не можуть бути реалізовані за допомогою класичних монолітних підходів.

6. ВИСНОВКИ

Уніфіковане ядро на Rust/WASM є ефективним підходом до подолання проблеми фрагментації та дублювання коду в кросплатформенній розробці. Воно забезпечує єдину базу бізнес-логіки, що спрощує підтримку, тестування та інтеграцію змін. Поєднання продуктивності та безпеки Rust із портативністю WebAssembly дозволяє створювати універсальні рішення для вебу й нативних середовищ. Зростаюча підтримка інструментів та індустрії підтверджує перспективність цієї архітектури для розробки сучасних, надійних і масштабованих програмних продуктів.

Список літератури

[1] Sebouai H. How Code Duplication Affects Software Quality and Maintainability - Axify Blog, 02.04.2025. - URL: https://axify.io/blog/code-duplication (дата звернення: 10.09.2025).

[2] The Dark Side of Code Reuse: Hidden Risks in Copy-Paste Programming. - Dark Tech Insights, 27.08.2025. - URL: https://darktechinsights.com/dark-side-of-code-reuse-copy-paste-programming/ (дата звернення: 10.09.2025).

[3] Palachi E. What is Software Architecture? A Comprehensive Guide - vFunction, 28.07.2025. - URL: https://vfunction.com/blog/what-is-software-architecture/ (дата звернення: 10.09.2025).

[4] Powell J.E. Q&A: Why You Should Separate Data from Business Logic - TDWI, 19.07.2019. - URL: https://tdwi.org/articles/2019/07/19/bi-all-why-separate-data-from-business-logic.aspx (дата звернення: 10.09.2025).

[5] Business Logic: Definition, Benefits, and Example. — Investopedia, оновлено 28.08.2020. — URL: https://www.investopedia.com/terms/b/businesslogic.asp (дата звернення: 10.09.2025).