3. Інформаційні технології. Методи захисту системи управління інформаційною безпекою. ДСТУ ISO/IEC 27001:2015. Київ, Держспоживстандарт України, 2016.
4. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування. Безпека інформації. 2018. 24 (2). С. 110 -117. Коваленко О. В.
5. Оцінка ефективності технології тестування безпеки. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. 2018. 29 (68). С. 137 - 141. Коваленко О. В.

ОБГРУНТУВАННЯ ВИБОРУ ФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ / JUSTIFICATION OF THE CHOICE OF FRAMEWORKS FOR DEVELOPING THE CLIENT PART OF A WEB APPLICATION

Кирієнко О. В., Пєчкурова ОМ. / O. Kyriienko, O. Pyechkurova
Національний університет «Києво-Могилянська Академія» / National University of "Kyiv-Mohyla Academy"
04070, Київ, вул. Григорія Сковороди, 2.
E-mail: o.kyriienko@ukma.edu.ua

This article examines the main criteria for selecting frameworks, compares popular solutions, and provides recommendations for their use depending on the project's specifics.

В сучасному світі веброзробки вибір фреймворку для створення клієнтської частини вебзастосунку є одним із найважливіших етапів розробки. Фреймворки надають набір інструментів і бібліотек, які дозволяють розробникам ефективно створювати інтерактивні, швидкі та зручні вебзастосунки. Правильний вибір фреймворку може суттєво вплинути на час розробки, продуктивність фінального продукту, а також на можливості його масштабування і підтримки.

Основні критерії вибору фреймворку для розробки клієнтської частини вебзастосунку.

У веброзробці фреймворки для клієнтської частини (frontend) відіграють ключову роль і вибір залежить від багатьох чинників, що впливають на продуктивність, зручність розробки, масштабованість та підтримку проєкту. Розглянемо основні критерії вибору.

Продуктивність. Важливо, щоб забезпечувалося швидке завантаження сторінок та рендеринг контенту. Фреймворк повинен ефективно використовувати пам'ять та процесорний час, щоб забезпечити стабільну роботу застосунку навіть при високих навантаженнях.

Легкість у використанні. Наявність детальної, зрозумілої та актуальної документації значно спрощує впровадження фреймворку у проєкт. Для команди важливо обрати фреймворк, який відповідає їхньому рівню знань та досвіду. Зручний та логічний API допомагає розробникам швидко освоїти фреймворк та ефективно використовувати його можливості.

Масштабованість. Фреймворк має забезпечувати можливість легкого розширення застосунку, бути здатним підтримувати великі та складні проекти без втрати продуктивності. Підтримка модульної архітектури сприяє спрощенню роботи над масштабними проєктами відповідно до потреб бізнесу.

Безпека. Фреймворк повинен мати вбудовані механізми захисту від поширених веб-вразливостей, таких як XSS, CSRF та SQL-ін'єкції. Регулярні оновлення забезпечують усунення потенційних проблем із безпекою, а документація та інструменти фреймворку повинні сприяти дотриманню безпечних практик розробки.

Інтеграція з іншими технологіями. Важливо, щоб фреймворк міг інтегруватися з іншими використовуваними бібліотеками чи бекенд-технологіями та забезпечував інтеграцію з інструментами для тестування, збірки та управління станом.

Розмір і продуктивність кінцевого застосунку. Важливо, щоб фреймворк дозволяв мінімізувати розмір кінцевого бандла (збірки) застосунку для забезпечення швидшого завантаження сторінок, особливо на мобільних пристроях та при повільному інтернет-з'єднанні. Також, фреймворк повинен підтримувати динамічне завантаження модулів (lazy loading) для завантаження тільки необхідні частини застосунку під час його використання, зменшуючи початковий розмір бандла. Та забезпечувати високу швидкість рендерингу компонентів, що зменшує час відгуку застосунку та покращує взаємодію з користувачем. Наявність ефективних механізмів управління станом застосунку дозволяють зменшити кількість перерендерів та покращити загальну продуктивність.

Інструменти для аналізу та оптимізації. Наявність інструментів для профайлінгу та моніторингу продуктивності дозволяє розробникам виявляти та усувати вузькі місця у застосунку. А інструменти для аналізу розміру бандла допомагають визначити, які частини коду займають найбільше місця, та оптимізувати їх.

Підтримка та спільнота. Наявність великої та активної спільноти розробників забезпечує швидке вирішення проблем та обмін досвідом. Популярні фреймворки (наприклад, React, Angular, Vue.js) мають активну спільноту, яка може надати підтримку та розробляє додаткові бібліотеки. Фреймворк має бути активно підтримуваним розробниками, щоб уникнути застарілості. Регулярні оновлення фреймворку свідчать про його підтримку та розвиток, що важливо для безпеки та додавання нових функцій.

Порівняння популярних фреймворків для розробки клієнтської частини вебзастосунку: React, Angular, Vue.js, Nuxt

Отже, вибір фреймворку є стратегічним рішенням, що впливає на ефективність розробки та довгостроковий успіх проєкту. Детальний аналіз вимог проєкту, досвіду команди та доступних технологій дозволить обрати фреймворк, який максимально відповідає потребам конкретного застосуну.

1. React[1] — *це бібліотека для побудови користувацького інтерфейсу, розроблена компанією Meta (Facebook).* Використовується **віртуальний DOM**, що забезпечує високу швидкість оновлення UI; має компонентну структуру, що дозволяє створювати повторно використовувані модулі; підтримує односторонній потік даних (unidirectional data flow).

- **Переваги:** гнучкість у виборі додаткових бібліотек для маршрутизації (React Router) чи управління станом (Redux, Zustand); велика спільнота та багата екосистема; швидкість рендерингу завдяки оптимізації оновлень.
- **Недоліки:** вимагає додаткових бібліотек для повноцінного функціонування; крива навчання може бути складною для початківців.

Підсумок: ідеально підходить для SPA (односторінкових застосунків), інтерактивних елементів або проєктів із кастомною архітектурою.

2. Angular [2]— це фреймворк від **Google**, побудований на TypeScript. Він орієнтований на розробку масштабованих вебзастосунків. Це повноцінний фреймворк із вбудованими рішеннями для маршрутизації, управління станом і тестування; використовує двосторонній зв'язок даних (two-way data binding); шаблонний синтаксис для створення інтерфейсів.

- **Переваги:** підтримка великих і складних проєктів завдяки строгій структурі; висока продуктивність завдяки AoT (Ahead-of-Time) компіляції; вбудовані інструменти для тестування та налагодження.
- **Недоліки:** строга структура може обмежувати гнучкість; вища складність для вивчення порівняно з React або Vue.js.

Підсумок: підходить для великих корпоративних застосунків, багатофункціональних вебсистем.

3. Vue.js[3] — це прогресивний фреймворк із відкритим кодом, розроблений **Еваном Ю**. Він зосереджений на простоті й доступності. Використовується декларативний стиль програмування для створення компонентів; підтримує двосторонній зв'язок даних; легко інтегрується в існуючі проєкти.

- **Переваги:** простота використання та легка крива навчання; гнучкість: можна використовувати як фреймворк або як бібліотеку; легка вага, швидке завантаження компонентів.
- **Недоліки:** менша спільнота в порівнянні з React чи Angular; не завжди підходить для дуже великих проєктів через менш структурований підхід.

Підсумок: Оптимальний вибір для невеликих і середніх проєктів, стартапів чи тих, хто впроваджує frontend поступово.

4. Nuxt.js[4] — це фреймворк, побудований на основі Vue.js, орієнтований на **SSR (Server-Side Rendering)** і створення SEO-оптимізованих вебзастосунків. Підтримує автоматичну генерацію маршрутизації; має підтримку статичних сайтів та серверної рендеризації; надає структуру проєкту «із коробки».

- **Переваги:** відмінна SEO-оптимізація завдяки SSR; простий старт і зручність налаштування; готові плагіни для інтеграції.
- **Недоліки:** може бути зайвим для проєктів, які не потребують SSR; залежність від Vue.js, що обмежує вибір для інших архітектур.

Підсумок: Чудово підходить для блогів, маркетингових сайтів, електронної комерції, де SEO є критично важливим.

Порівняльна таблиця

Критерій	**React**	**Angular**	**Vue.js**	**Nuxt.js**
Продуктивність	Висока	Висока	Висока	Висока (завдяки SSR)
Крива навчання	Середня	Висока	Низька	Низька
Гнучкість	Висока	Середня	Висока	Середня
Масштабованість	Висока	Висока	Середня	Висока
SEO-оптимізація	Обмежена	Обмежена	Обмежена	Висока
Екосистема	Широка	Вбудована	Розвивається	Спеціалізована

Загальний алгоритм вибору фреймворку

Визначте тип проєкту: SPA, MPA, SEO-залежний чи інший.

Оцініть досвід команди: Чи є спеціалісти з JavaScript, TypeScript, або Vue.js?

Поставте бізнес-цілі: Які критерії важливі: швидкість розробки, SEO, масштабованість?

Врахуйте довгострокові плани: Чи планується розширення застосунку та додавання нових функцій?

Перевірте екосистему: Чи є потрібні бібліотеки, плагіни та підтримка?

Приклади вибору

Мобільний додаток із високою взаємодією: React із React Native.

Корпоративна система із складною логікою: Angular.

Блог або маркетинговий сайт: Nuxt.js.

Прототип або MVP: Vue.js для швидкої розробки та гнучкості.

Ретельний аналіз специфіки проєкту та вимог допоможе зробити правильний вибір, який сприятиме ефективному виконанню завдань і підтримці застосунку в майбутньому.

Висновок

Серед безлічі доступних фреймворків, таких як React, Angular, Vue.js та Nuxt.js, важливо не лише враховувати їхню популярність, а й визначити, який із них найкраще відповідає специфіці конкретного проекту та вимогам замовника. Зважаючи на стрімкий розвиток технологій та зміну потреб користувачів, підхід до вибору фреймворків повинен бути ретельним і зваженим. У цій доповіді було надано практичні рекомендації для успішного вибору та використання фреймворків у реальних проектах.

1. React Developer Tools - Chrome Web Store https://chromewebstore.google.com/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en

2. Angular https://angular.dev/
3. Vue.js - The Progressive JavaScript Framework https://vuejs.org/
4. Nuxt: The Intuitive Vue Framework https://nuxt.com/

МЕТОДИ ЗАСТОСУВАННЯ МАШИН СТАНІВ У СИСТЕМАХ ОБРОБКИ ДАНИХ З ВИСОКИМ НАВАНТАЖЕННЯМ / METHODS FOR APPLYING STATE MACHINES IN HIGH-LOAD DATA PROCESSING SYSTEMS

Моголівський В.О. / Moholivskyi V.O.
Національний університет «Києво-Могилянська академія» / National University of Kyiv-Mohyla Academy
04070, Київ, вул. Г. Сковороди, 2, тел. (044) 425-60-59
E-mail: v.moholivskyi@ukma.edu.ua

Methods for applying state machines to the domain of high-load data processing systems are reported in this paper. This study focuses on improving existing optimization approaches for which the use of state machines is the most beneficial. The process's in-memory cache, data processing rate limiting, and background system maintenance jobs are studied. The paper suggests utilizing state machines to alleviate the problem of distributed in-memory cache partial invalidation. Furthermore, state machines are proposed as a central control unit for managing data processing rate-limiting and long-running background tasks. The suggested methods reduce the complexity of the studied approaches by adding more transparency and monitoring capabilities.

Вимоги до систем обробки даних у сучасному світі зростають з кожним днем [3]. Обсяг та складність даних збільшується, а бажаний час їх обробки залишається незмінним [3]. Для досягнення високої ефективності у таких системах використовуються певні архітектурні підходи, техніки масштабування та методи оптимізації обробки даних [3]. Дане дослідження зосереджене на оптимізаційних підходах для яких використання машин станів є найбільш корисним. А саме на кешуванні в оперативній пам'яті процесу, контрольованому обмеженні швидкості обробки даних та фонових процесах обслуговування системи. Метою даного дослідження є доповнення кожного зі згаданих підходів використанням машин станів [2, 4] задля збільшення прозорості та ефективності їх застосування. Машини станів дозволяють керувати складністю програми роблячи процеси більш структурованими та передбачуваними.

Машина станів може бути визначена як кортеж з п'яти елементів $(Q, \Sigma, \delta, q_0, F)$ [2], де Q – це скінченна множина станів, Σ – це скінченна множина вхідного алфавіту (вхідних символів або подій), δ – це функція переходу між станами $\delta: Q \times \Sigma \to Q$, яка приймає вхідний символ (подію) та визначає наступний стан, q_0 – це початковий стан ($q_0 \in Q$), F – це множина кінцевих станів ($F \subseteq Q$). У контексті розподіленої системи функція δ окрім визначення наступного стану може також ініціювати виконання процедур згідно з функціональними вимогами програми, а саме робити виклики до баз даних, викликати віддалені процедури, надсилати повідомлення у брокери повідомлень тощо. Така машина станів може бути застосована для розв'язання функціональних задач у розподіленому середовищі.

Одним із ключових методів оптимізації обробки даних у високонавантажених системах є кешування, зокрема кешування в оперативній пам'яті [1, 3]. Основною перевагою такого підходу є швидкість доступу до кешу, вона є найвищою у порівнянні з іншими видами кешів. Проте є і суттєві недоліки такі як обмеження у розмірі та складність точкового анулювання кеша за потребою у розподілених системах. Проблема точкового анулювання кеша може бути значно полегшена за допомогою використання машини станів. Якщо відбулося оновлення даних та необхідне анулювання певної групи записів у кешах розподілених в пам'яті багатьох процесів, має бути ініціалізована машина станів, яка послідовно чи паралельно здійснить координацію видалення неактуальних записів кеша. Машина станів може використовувати виклики віддалених процедур чи обмін подіями для комунікації з кожним із процесів. Даний підхід дозволяє чітко визначити процес анулювання кеша та забезпечити структуроване керування ним у розподілених системах.