**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФРАСТРУКТУРИ ТА ТЕХНОЛОГІЙ**

ІНСТИТУТ УПРАВЛІННЯ, ТЕХНОЛОГІЙ ТА ПРАВА

Факультет управління і технологій

Кафедра інформаційних технологій та дизайну

ЗВІТ

З ПЕРЕДДИПЛОМНОЇ ПРАКТИКИ

Спеціальність:  *121 «Інженерія програмного забезпечення»*

Місце проходження практики:  \_ФОП «Мартинюк Артем Віталійович»\_\_\_

Дата початку практики: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 року

Дата закінчення практики: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 року

Керівник практики від кафедри: Ткаченко О.І.                         \_\_\_\_\_\_\_\_\_\_\_\_\_\_

                                                                                                                                                                                                     (підпис)

Керівник практики від підприємства:                                         \_\_\_\_\_\_\_\_\_\_\_\_\_\_

                                                                                                                                                                                                     (підпис)

Студента

2 курсу (маг) д.ф.н.

Мартинюка Артема Віталійовича

Київ 2021

5ЗМІСТ

|  |  |
| --- | --- |
| ВСТУП | 3 |
| АНАЛІЗ ДЖЕРЕЛ ЩОДО ОБРАНОЇ НАУКОВОЇ ПРОБЛЕМИ | 7 |
| ОПИС ІСНУЮЧИХ ШЛЯХІВ ВИРІШЕННЯ ОБРАНОЇ НАУКОВОЇ ПРОБЛЕМИ | 15 |
| ОПИС ВЛАСНОГО ШЛЯХУ ВИРІШЕННЯ ОБРАНОЇ НАУКОВОЇ ПРОБЛЕМИ | 17 |
| ОПИС РОЗРОБЛЕНОЇ МОДЕЛІ ТА ПРОГРАМНОГО ПРОДУКТУ | 21 |
| ВИСНОВКИ |  |
| СПИСОК ДЖЕРЕЛ |  |

ВСТУП

JavaScript спочатку був створений як мова сценаріїв для Netscape. Потім він був відправлений в ESMA international для стандартизації. Це призвело до появи новогостандарту ESMAscript. Наступні версії JavaScript вже базувалися на стандарті ESMAscript.

ESMAscript є стандартом, а JavaScript є найпопулярнішою реалізацією цього стандарту.

Однак JavaScript все ще був «найбільш незрозумілою мовою програмування в світі». Серед його проблем були глобальні зони видимості змінних, однопоточність, відсутність суворої типізації та принципів ООП.

Але в червні 2015 року все змінилося. Був випущений новий стандарт - ES6, додавши наступні зміни:

• деструктивне призначення;

• стрілочні функції;

• шаблонні строки;

• локальні змінні та константи;

• ітератори та генератори;

• класи;

• проміси та async/await.

З тих пір щороку випускається новий стандарт ESMAscript.

*Актуальність*. Про попит на будь-яку мову можна судити по кількості написаних для неї бібліотек. NPM (node package manager) це головний репозиторій пакетів для середы node.js. Він є найбільшим у світі. Станом на 2017 рік в NPM було розміщено понад 380 тисяч пакетів. Для порівняння в репозиторії packagist(php) – 123 тисячі, в PyPI(Python) – 96 тисяч і в nuget (.NET) – 70 тисяч.

В середньому через NPM на місяць встановлюється понад 18 мільярдів пакетів.

*Мета дослідження*. Метою роботи є дослідження бібліотеки react.js та його можливості для створення клієнтської частини веб-додатку.

*Завдання дослідження*. Створення робочого зразка програмного забезпеченнях, який демонструє кращі практики створення SPA-додатку.

*Об'єкт дослідження*. Процес створення додатку за допомогою react.js та redux; порівняння різних методів та підходів створення.

*Предмет дослідження.* Ретроспектива; web-додаток, що надає змогу створювати задачі та управляти ними.

ОГЛЯД ТА АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ

* 1. Про веб-додатки

Веб-додаток – це веб-сайт, на якому розміщено сторінки з частково або повністю несформованим вмістом. Остаточний контент генерується тільки після того, як відвідувач сайту запитує сторінку з веб-сервера. Оскільки кінцевий зміст сторінки залежить від запиту на основі дій відвідувача, сторінка називається динамічною.

На відміну від веб-сайту, веб-додаток - це повноцінна програма, до якою користувач отримує доступ через інтернет, тобто не вимагає установки на пристрій. Веб-додаток є інтерактивним і дозволяє користувачам взаємодіяти з різними елементами: наприклад, залишити заявку на покупку товару, зробити покупку авіаквитка або прокоментувати пост друга.

Спектр використання веб-додатків дуже широкий. У цьому розділі розглядаються загальні міркування щодо використання веб-застосунків і наведено приклад простого веб-застосунку.

* 1. Типове використання веб-застосунків

Використання веб-додатків приносить певні переваги як відвідувачам веб-сайту, так і їх розробникам.

1. Веб-додатки дозволяють відвідувачам швидко і легко знаходити необхідну інформацію на сайтах з великим обсягом інформації.

Цей вид веб-додатків дозволяє шукати контент, впорядковувати контент і переміщатися по ньому зручним для відвідувачів способом. Прикладами таких додатків є внутрішні мережі компаній — Microsoft MSDN ([www.msdn.microsoft.com](http://www.msdn.microsoft.com/)) і Amazon ([www.amazon.com](http://www.amazon.com/)).

1. Веб-додатки дозволяють збирати, зберігати і аналізувати дані, отримані від відвідувачів сайту.

Довгий час використовувався метод, в якому введені дані HTML-форми, відправлялись на обробку CGI-додаткам або спеціально призначеним працівникам у вигляді повідомлень електронної пошти. Веб-додаток дозволяє зберігати дані безпосередньо в базі даних, а також отримувати дані і генерувати звіти на основі даних, отриманих для аналізу. Приклади включають інтерактивні сторінки банків, сторінки для контролю запасів, соціологічні дослідження та опитування, а також форми для зворотного зв'язку користувачів.

1. веб-застосунок можна використовувати для оновлення веб-сайтів, які мають вміст, який періодично змінюється.

веб-додаток звільняє веб-дизайнера від рутинної роботи постійного оновлення html-сторінок сайту. контент-провайдери, такі як редактори новин, несуть відповідальність за наявність свіжого матеріалу, а веб-додаток стежить за автоматичним оновленням сайту. Прикладом є веб-версія журналу «The Economist» ([www.economist.com](http://www.economist.com/)) и служба новин CNN ([www.cnn.com](http://www.cnn.com/)).

* 1. Як працюють веб-додатки

Будь-який веб-застосунок - це набір статичних і динамічних веб-сторінок. *Статична веб-сторінка* - це сторінка, яка завжди відображається користувачеві в незмінному вигляді. Веб-сервер відправляє сторінку на запит веб-браузера без будь-яких змін. На відміну від цього, сервер вносить зміни на *динамічну веб-сторінку* перед відправкою її в браузер.

Наприклад, можна створити сторінку, на якій відображаються результати оздоровчої програми. При цьому деяка інформація (наприклад, ім'я співробітника і його результати) буде визначатися в момент запиту сторінки співробітником.

У наступному розділі детально описано, як працюють веб-застосунки.

Обробка статичних веб-сторінок

Статичний веб-сайт містить набір відповідних HTML-сторінок і файлів, розміщених на комп'ютері, на якому встановлено веб-сервер.

Веб-сервер - це програмне забезпечення, яке надає веб-сторінки у відповідь на запити веб-браузерів. Зазвичай запит на сторінку створюється натисканням посилання на веб-сторінці, вибором закладки у браузері або введенням URL-адреси в адресному рядку браузера.

Остаточний вміст статичної веб-сторінки визначається розробником і залишається незмінним в процесі запиту сторінки. Приклад:

<html>

<head>

    <title>Trio Motors Information Page</title>

  </head>

  <body>

    <h1>About Trio Motors</h1>

    <p>Trio Motors **is** a leading automobile manufacturer.</p>

  </body>

</html>

Весь HTML-код генерується розробником до розміщення сторінки на сервері. Оскільки HTML-код не змінюється після розміщення сторінки на сервері, сторінка називається статичною.

Примітка.

Строго кажучи, "статична" сторінки насправді може такою і не бути. Наприклад, зображення заміщення або flash-контент (як файл SWF) дозволяє «оживити» статичну сторінку. Однак у цьому контексті сторінка статична, оскільки вона надсилається до браузера без змін.

Коли веб-сервер отримує запит на видачу статичної сторінки, то, проаналізувавши запит, сервер знаходить потрібну сторінку і відправляє її в браузер, як показано нижче.

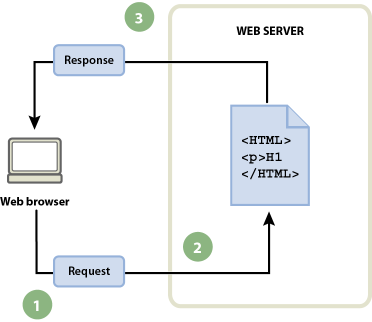


Рисунок 1.3.1 – *Обробка статичної веб-сторінки*

**A.** Веб-браузер запитує статичну сторінку. **B.** Веб-сервер знаходить сторінку. **C.** Веб-сервер надсилає сторінку до браузера, який її запросив.

У випадку веб-додатків деякі частини коду сторінки відсутні, поки відвідувач не запитає сторінку. Відсутній код генерується якимось механізмом, і тільки тоді сторінка може бути відправлена в браузер. Цей механізм генерації коду обговорюється в наступному розділі.

Обробка динамічних сторінок

Коли веб-сервер отримує запит на випуск статичної веб-сторінки, він надсилає сторінку безпосередньо до браузера. Однак, коли запитується динамічна сторінка, дії веб-сервера не настільки чіткі. Сервер передає сторінку в спеціальну програму, яка формує кінцеву сторінку. Така програма називається сервером додатків.

Сервер програми зчитує код, який знаходиться на сторінці, генерує кінцеву сторінку відповідно до прочитаного коду, а потім видаляє її зі сторінки. В результаті всіх цих операцій виходить статична сторінка, яка передається на веб-сервер, який в свою чергу відправляє її в клієнтський браузер. Всі сторінки, які отримує браузер, містять тільки HTML-код. Схематичне представлення процесу:

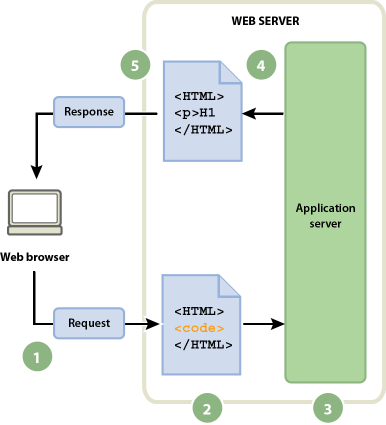


Рисунок 1.3.2 – *Обробка динамічних сторінок*

**A.** Веб-браузер запитує динамічну сторінку. **B.** Веб-сервер знаходить сторінку та передає її серверу застосунків. **C.** Сервер програми сканує сторінку на наявність інструкцій і створює її. **D.** Сервер застосунків повертає підготовлену сторінку на веб-сервер. **E.** Веб-сервер надсилає підготовлену сторінку до браузера, який її запросив.

* 1. Типи веб-додатків

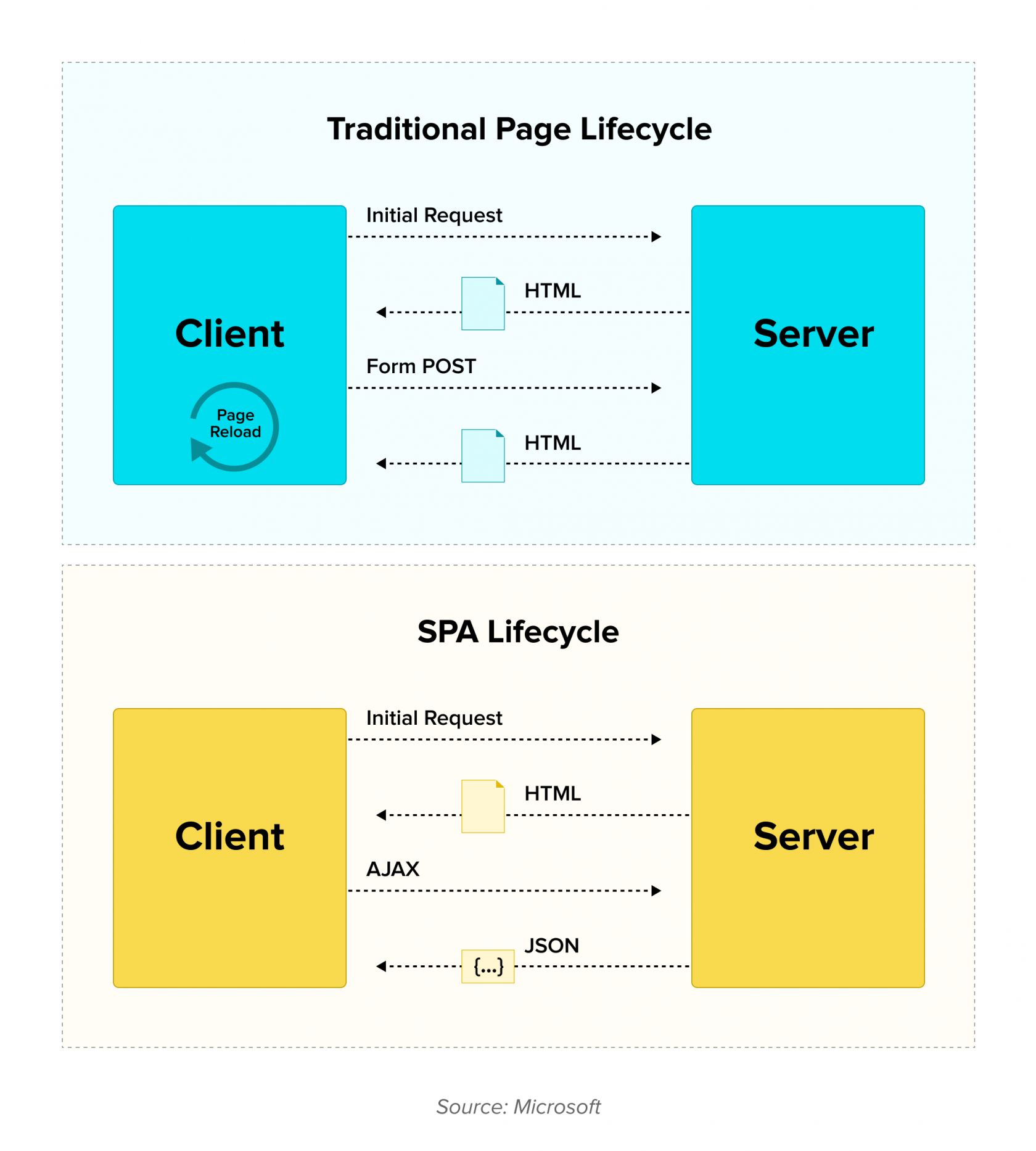
Веб-додатки можна класифікувати по-різному: в залежності від їх функціональності і призначення. Давайте детальніше розглянемо ці типи додатків, щоб краще зрозуміти, як вони працюють і який з них підходить для ваших бізнес-завдань.

Існує три основні шаблони для створення веб-сайтів:

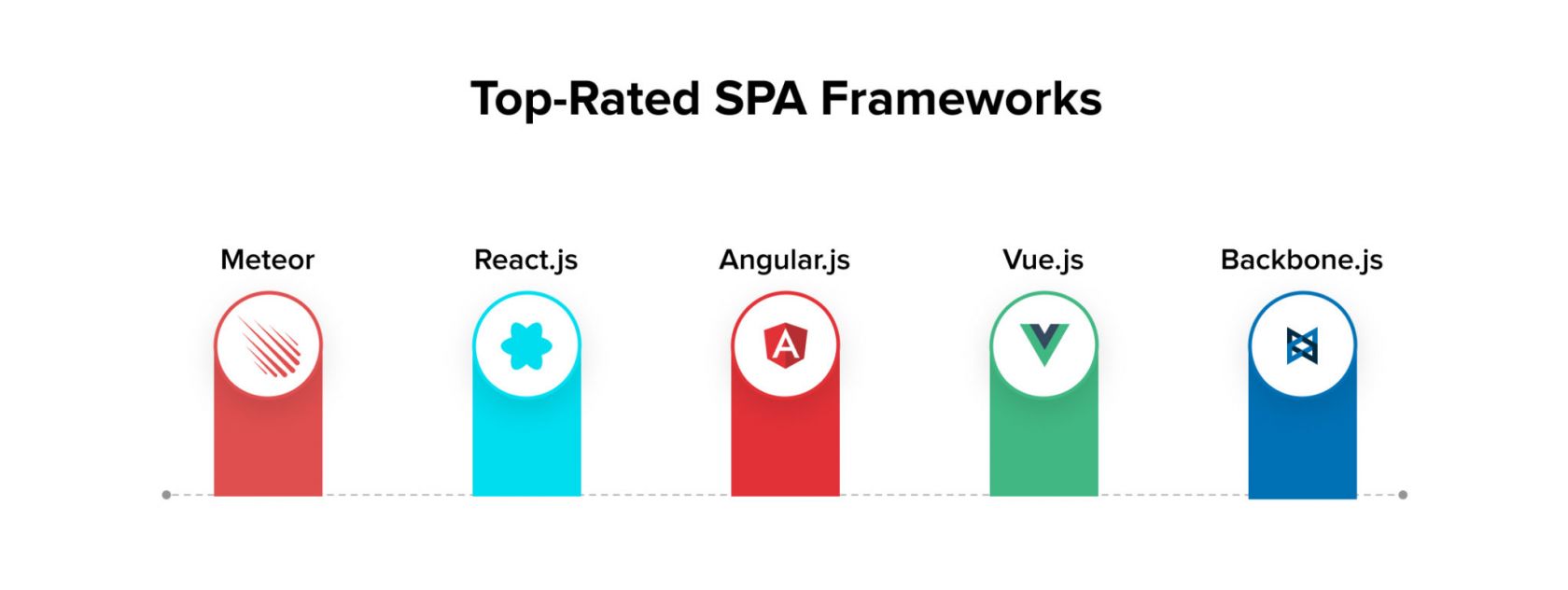
1. MPA (multi-page application): багатосторінковий застосунок, який надсилає запит на сервер і повністю оновлює сторінку під час виконання на ній дії.
2. SPA (single-page application): односторінковий застосунок, який містить HTML-сторінку, яка динамічно оновлюється на основі дій користувача без повного перезавантаження.
3. PWA (progressive web application): додаток, який користувач встановлює і може використовувати в автономному режимі.

Що таке SPA-додаток

Значення терміна SPA (Single Page Application) лежить в собі. SPA - це односторінковий додаток, який містить HTML-сторінку, яка динамічно (без повного перезавантаження) оновлюється у відповідь на дії користувача. Архітектура програми розроблена так, що при початковому запуску відвідувач бачить основний зміст сайту в браузері, а нові дані завантажуються в міру необхідності, наприклад, при прокручуванні або натисканні на іконку. Якщо ви коли-небудь перегорнули стрічку Facebook, Twitter або Instagtam, ви знаєте, про що йде мова.



Насправді односторінкові додатки багато в чому схожі на настільні: при переході на нову сторінку оновлюється тільки частина контенту, що дозволяє не завантажувати одні і ті ж елементи сайту багато разів. Подібний ефект для SPA забезпечують сучасні javascript фреймворки. Для невеликих проектів підійде jQuery, але для створення серйозного SPA вам знадобляться фронтенд-бібліотеки, такі як React, Vue або Angular (рис ).



рис

SPA-додатки використовуються в тих випадках, коли на сайт мало інформації і вона повинна бути ефективно продемонстрована користувачеві. Прикладами SPA є соціальні мережі та сайти з великою кількістю користувачів.

Переваги SPA

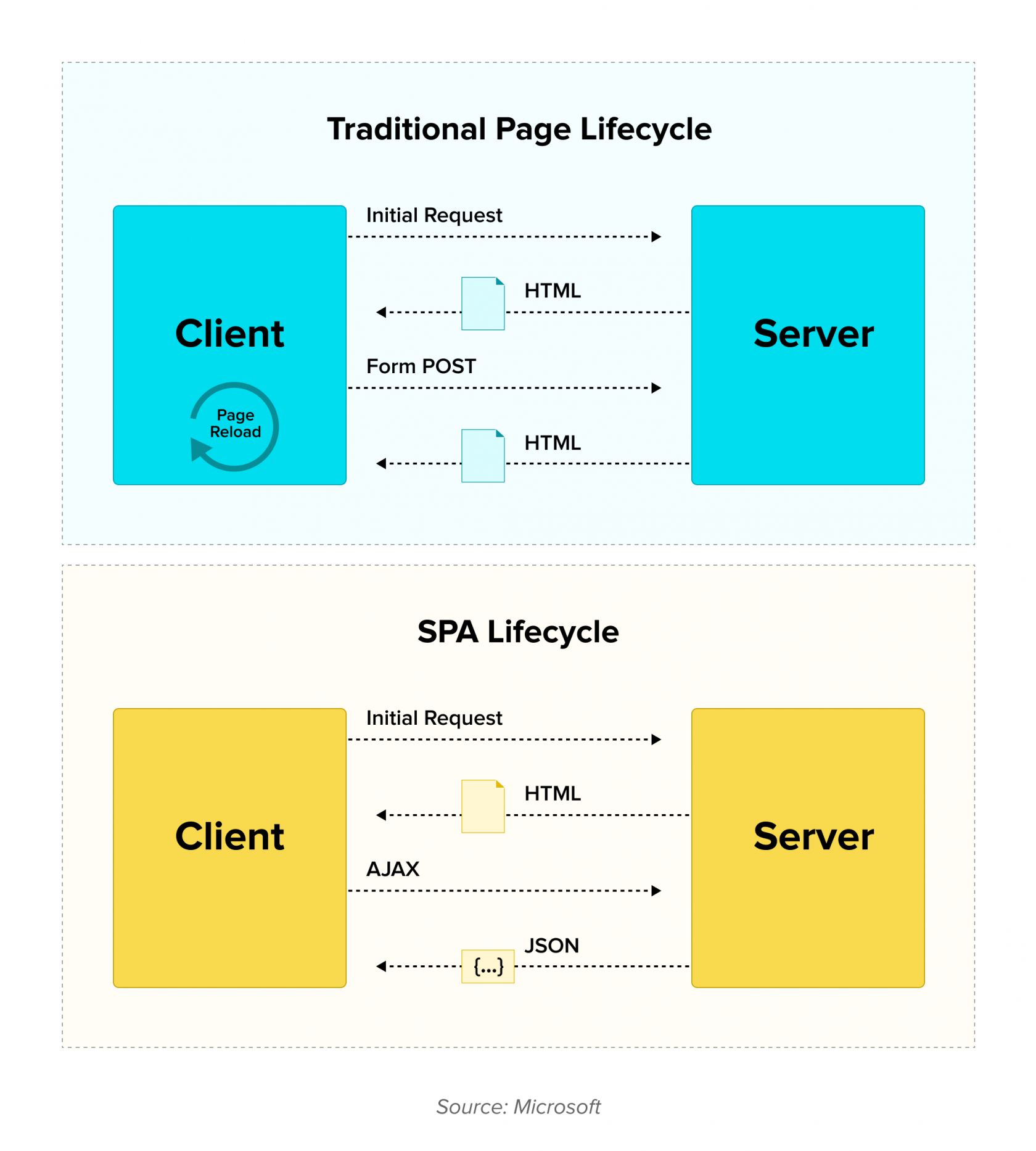
Дослідження показують, що користувачі чутливі до швидкості завантаження сторінок. У питанні швидкості завантаження SPA-сайти не мають рівних, що є їх головною перевагою. Так як левова частка ресурсів завантажується за один сеанс, при подальших діях SPA швидко оновлює необхідну частину даних, значно заощаджуючи час.

Однак висока швидкість - не єдиний плюс SPA-додатків. Крім того, односторінкові додатки мають наступні переваги:

1. Легкість створення. Бібліотеки та фреймворки вже готові до розробки SPA-додатків, робота над інтерфейсом та бекендом може проводитися паралельно. Крім того, на основі готового коду в майбутньому можна побудувати мобільний додаток;
2. гнучкість інтерфейсу користувача. Для однієї сторінки простіше розробити захоплюючий та інтерактивний дизайн інтерфейсу;
3. Просте кешування даних. Важливі дані завантажуються в одному запиті, і тоді користувач односторінкового додатка може працювати частково в автономному режимі, підключаючись до Інтернету для збереження прогресу.

Що таке MPA-додаток

MPA або Multi Page Applications за принципом роботи є зовсім протилежний SPA. MPA - це багатосторінковий додаток, який працює як веб-сайт, до якого ми звикли. Вони відправляють запит на сервер і повністю оновлюють сторінку при виконанні з нею будь-якої дії (переходячи на іншу сторінку, вводячи і змінюючи дані). Ця архітектура додатків має значний вплив на швидкість і продуктивність, оскільки більшість даних перезавантажується на кожному стрибку.



MPA підходить для великих компаній, що пропонують широкий спектр послуг і товарів: інтернет-магазини, сайти компаній, каталоги і торгові платформи.

В цілому, MPA є головною сторінкою і багато посилань на інші сторінки. Цей принцип використовується сайтами електронної комерції, такими як Ebay, Ozon і Amazon, де необхідно відображати велику кількість контенту.

Переваги MPA

* Проста SEO-оптимізації. MPA часто використовують сайти, для яких важливо потрапити в топи пошукових систем. Кожна зі сторінок має унікальну URL-адресу і є стабільною, що дозволяє пошуковим ботам адекватно її повзати.
* Масштабованість. Ви можете додати стільки інформації, скільки вам потрібно, до програми MPA, без обмежень на сторінку або функції.
* Перевірена класика. МПА працюють за тими ж принципами, що і веб-сайти з класичною навігацією, які знайомі користувачеві.

Недоліки МПА

* Повільна швидкість розвитку. MPA-додатки вимагають використання фреймворків з обох сторін: клієнта і сервера. Також через тісний зв'язок між інтерфейсом і бекендом робота розробників ускладнюється. Все це разом збільшує бюджет і час розвитку.
* Вартість створення мобільного додатку. Програми MPA не перетворюються добре на мобільні пристрої. Для цього в значній частині випадків буде потрібно розробка бекенда з нуля.

Що таке PWA-додаток

Progressive Web Application (PWA) це щось середнє між веб-сайтом і нативним додатком. Щоб почати роботу з PWA, користувач повинен завантажити та інсталювати сайт як додаток. Подальший доступ до сайту здійснюється через іконку на робочому столі користувача. При натисканні сайт швидко відкривається без посередника у вигляді браузера.

PWA додатки цікаві тим, що вони:

1. кросплатформні - робота відразу на декількох операційних системах;
2. встановлюються в один клік, не виходячи на ринки додатків;
3. завжди під рукою у користувача — сайт доступний за іконкою в смартфоні навіть в автономному режимі;
4. зниження конкуренції в пошукових системах;
5. дозволяють надсилати push-сповіщення як додаток.
6. легко конвертувати з існуючого сайту, просто змінити його трохи.

Прогресивні веб-додатки корисні для часто поновлювальних сервісів, до яких клієнти будуть продовжувати повертатися. Push-повідомлення дозволять періодично привертати увагу користувачів і заохочувати їх відвідувати додаток, збільшуючи прибуток.

* 1. Використання фреймворків

JS-framework’и — є інструментами для побудови динамічних веб / мобільних / настільних додатків на мові Javascript. Як і з будь-якими іншими інструментами, розробники вдаються до використання js-фреймворків там, де неможливо / дуже складно / дуже довго виконувати завдання звичайними засобами. У переважній більшості випадків фреймворки використовуються для написання так званих односторінкових додатків.

З їх допомогою можна розробити як повноцінні сайти, так і функціональні модулі (різні онлайн-інструменти). Звичайно, для першого завдання краще підходять повноцінні фреймворки, а для другого рекомендується використовувати більш легкі фреймворки або бібліотеки.

Frayworkі забезпечують чітку структуру програми і реалізуються за допомогою так званих «шаблонів проектування» (концепція більше мігрувала з бекенда, але явно має місце в зовнішній розробці з використанням js-фреймворків). Найбільш широко використовуються наступні шаблони: MVC (Model-View-Controller), MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel).

Переваги побудови додатку на фреймворку JS:

1. може бути легко реалізований SPA (Single Page Application);
2. використання фреймворку JS зобов'язує нас мати структуру програми (скажімо, сильне «ні» коду спагетті);
3. код стає помітно менше і він чистіший, що позитивно позначається на швидкості розробки, а також на підтримку і усунення помилок в коді заявки;
4. наявність структури має на увазі модульність програми, а це полегшує роботу над додатком кільком розробникам одночасно;
5. Наступна перевага більше випливає з використання самого JavaScript, але значно посилюється при використанні фреймворку: можливість швидкого створення мобільного і / або настільного кросплатформного додатку з веб-версії за допомогою таких систем, як PhoneGap або Apache Cordova.

Зі істотних недоліків можна виділити тільки тимчасово неповну підтримку пошуковими системами, однак це завдання рідко збігається з завданням реалізації SPA (Single Page Application), тим більше, що провідні пошукові системи (як мінімум, Google), вже майже повністю вирішили цю проблему.

Багато додатків вже випущені на JS-фреймворки і цей сегмент тільки набирає обертів. Для прикладів виконаних проектів на Angular.js наприклад, виділяється цілий сайт builtwith.angularjs.org, списки сайтів і додатків з використанням інших фреймворків можна знайти внизу на головних сторінках офіційних сайтів: backbone.js, ember.js.

Програми- збирачі

Зазвичай проект передбачає модульну структуру, тобто файлів буде багато. Оскільки імпорт не працює в браузері, а «власний синтаксис» може бути використаний, збирачу доведеться конвертувати вихідні файли в зрозумілий і робочий для браузера код.

Тому збирач (зазвичай webpack) повинен мати свої налаштування, які новачкові важко зрозуміти. Тут допомагає Vue CLI, який працює з коробки. Але, якщо завдання полягає в тому, щоб зробити щось інше, то вам доведеться виконати всі налаштування самостійно. А для цього потрібно вивчити Webpack і бажано робити це до або паралельно з вивченням js-фреймворку.

По-перше, веб-компоненти

Модульний або компонентний підхід до розробки вже давно використовується практично скрізь, крім JS. Все, що є в JS зараз, насправді досить примітивні речі. Веб-компоненти - це технологія майбутнього, яка може змінити ситуацію і зробити веб-сайти повноцінними додатками.

Прикладом веб-компонента є тег VIDEO, який відображає досить складний візуальний компонент з елементами управління. Тобто ми вказуємо тільки тег і його параметри, а браузер перетворює його в щось більш складне.

Якщо не зайти в технічну складову веб-компонентів, то, по суті, вони надають розробнику можливість створювати окремі сутності (компоненти) за допомогою звичайних HTML і CSS (in scope!), і писати логіку програми в JavaScript.

Наприклад, нам потрібно розмістити слайдер jQuery на сторінці. Тепер потрібно підключити js-файл, css-стилі, які при цьому не повинні конфліктувати з іншими стилями сайту, а також написати код ініціалізації. З веб-компонентами це трохи простіше - це просто якийсь конкретний тег HTML (наприклад, MY-SLIDER) і набір необов'язкових атрибутів.

Але на даний момент підтримка веб-компонентів слабка, тому єдиний спосіб отримати щось подібне - заздалегідь побудувати проект в зручному для браузера коді. І тут ми підходимо до питання - наскільки js-фреймворк відповідає поняттю веб-компонентів.

Друга причина - реальна підтримка сайту

Якщо ви хочете побачити приклади сайтів, зроблених в React, Angular або Vue, то, швидше за все, це будуть якісь жалюгідні агрегати - просто пил в порівнянні з повноцінними сайтами в PHP.

Шум навколо фреймворків JS явно не корелює з їх фактичним використанням у веб-розробці. Отже, які сайти роблять на цих рамках?

Як правило, це якісь закриті розробки для конкретного клієнта (b2b), який готовий заплатити за таку роботу чимала грошей. Але справа не тільки в тому, що клієнти, як правило, погано розбираються в технологіях, але і в тому, що замовнику важливо не тільки створити проект, але і підтримати його далі.

Php сайт складається з декількох частин. Перший - це робота з базою даних, тобто отримання зразка у вигляді структурованих даних. Це найпростіша частина роботи. Потім дані переходять на шаблонний рівень, де вже відбувається формування HTML-виведення. Тут це не стільки php-програмування, скільки html-макет. Після цього настала черга CSS довести вихід до потрібного дизайну. Якщо вам потрібно змінити / додати елемент сторінки, цикл PHP-HTML-CSS потрібно буде повторювати знову і знову.

Підтримкою такого проекту стануть повноцінні фахівці, які роздадуться відразу в декількох технологіях. І це в якійсь мірі проблема персоналу, тому потрібно постаратися побудувати ділянку так, щоб структурні частини взаємодіяли один з одним тільки за допомогою якихось примітивних методів.

Цей підхід використовується в фреймворках JS. На рівні PHP надається якийсь API, запити на які повертають дані json. Ці дані отримані в js через ajax, а це означає, що js-програміст може взагалі не розуміти, як вони виникли - йому дали URL і приклад використання, а цього вже достатньо. PHP програмісту не потрібно вигадувати html-код і навіть думати про те, як будуть виглядати ці дані на сайті.

Сам макет відбувається на рівні компонентів - це вже HTML і CSS з примітивними включеннями js.

(React || Angular || Vue) vs. jQuery

Насправді, дуже дивне твердження питання, оскільки порівнювати ці фреймворки з jQuery дещо некоректно. Але якщо ви google, це питання є найбільш частим.

Потрібно розуміти, що jQuery - це просто набір функцій, які зібрані в одну оболонку (бібліотеку). Це не краще або гірше, ніж Vue або React, які є їх власними наборами функцій.

Питання тут скоріше в загальній ідеології. jQuery можна використовувати в будь-якому місці на сторінці і з будь-якими елементами. Це такий «атомний» підхід, який дозволяє використовувати будь-які елементи. JS-фреймворки припускають більш комплексний підхід і розглядають сторінку вже не як простий набір html-тегів, а як пов'язані компоненти. І ніхто не заважає використовувати jQuery в поєднанні з ними.

Існує ряд завдань, які легше вирішити за допомогою фреймворку, особливо якщо вони «заточені» для цих завдань. Але також набагато простіше використовувати готовий плагін jQuery, замість того, щоб винаходити колесо.

Типи сайтів для javascript фреймворків

JS-фреймворки, у зв'язку з тим, що вони працюють на стороні клієнта, не можуть використовуватися нормально там, де багато динамічних даних і різних сторінок / адрес, наприклад, в блогах. Тому основною метою для них є SPA (односторінковий додаток), де вся дія відбувається на одній сторінці. І хоча для них вже є рішення у вигляді маршрутизації, це все одно виглядає дещо дивно.

Тому js-фреймворки будуть цікаві тільки там, де потрібен високий «динамізм», наприклад, в тій же gmail-пошті. Головна особливість тут полягає в тому, що інтерфейс змінюється без перезавантаження сторінки і виглядає як звичайна програма в цілому.

Js-фреймворк не повинен використовуватися на звичайних сайтах або цільових сторінках. Як правило, немає необхідності в декількох запитах Ajax, а значить, ви можете легко обійтися звичайним PHP / HTML / CSS / JS / jQuery.

**Розробка FLUX-АРХІТЕКТУРИ НА ПРИКЛАДІ КОНТЕЙНЕРУ СТАНІВ JAVA-SCRIPT ДОДАТКУ REDUX**

*Facebook розробив Flux-архітектуру як спробу вирішити проблеми, викликані шаблоном MVC (Model-View-Controller) у своїй масивній кодовій базі. Вони боролися з проблемами, коли дії призвели до каскадних оновлень, що призвело до непередбачуваних результатів і коду, який стало важко відлагоджувати. Це може здатися знайомим, якщо ви використовували MVC фреймворки раніше, оскільки більшість з них по суті дуже схожі. При додаванні спостерігачів і двосторонніх даних, що зв'язуються з сервером, і ви отримаєте справжній головний біль.*

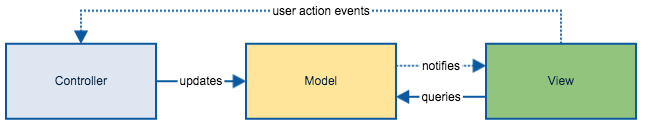
***Ключові слова:*** *Flux-архітектура, сховище станів додатку, Redux, reducer, actions, store, view, dispatcher.*

**Вступ.**Flux-архітектура — архітектурний підхід або набір патернів проєктування для побудови користувацького інтерфейсу веб-додатків, в поєднанні з реактивним програмуванням і побудований на односпрямованих потоках даних.

Згідно з планом творців, і незважаючи на те, що Facebook забезпечив реалізацію Flux на додаток до React, Flux - це не ще одна веб-база, а архітектурне рішення.

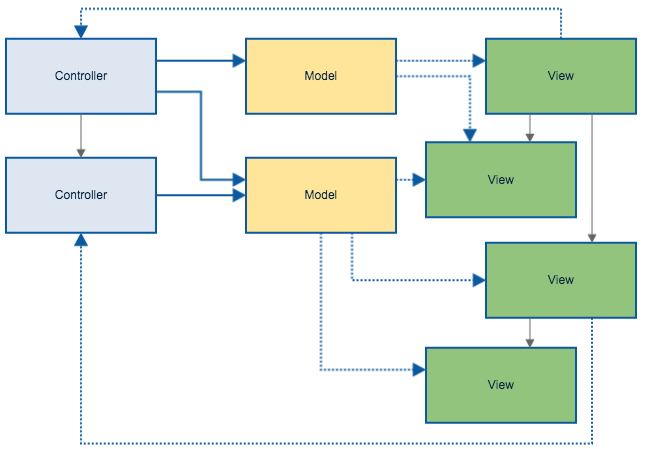
Головною відмінною рисою Flux є односторонній напрямок передачі даних між компонентами архітектури Flux. Архітектура накладає обмеження на потік даних, зокрема, виключаючи можливість оновлення стану компонентів самостійно. Такий підхід робить потік даних передбачуваним і полегшує відстеження причин можливих помилок в програмному забезпеченні.

**Порівняння Flux та MVC архітектур.** Щоб краще описати потік, ми порівняємо його з однією з провідних клієнтських архітектур: MVC. У клієнтському застосунку MVC  (рис.1) взаємодія користувача запускає код у контролері. Контролер знає, як координувати зміни в одній або декількох моделях, викликаючи методи на моделях. Коли моделі змінюються, вони повідомляють одне або кілька переглядів, які, в свою чергу, читають нові дані з моделей і оновлюють себе відповідно, щоб користувач міг бачити ці нові дані.



*Рис. 1.* **Схематичне зображення простого додатку MVC-архітектури.**

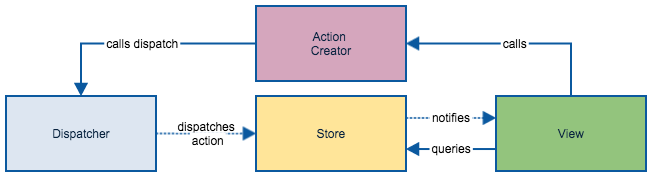
Оскільки програма MVC зростає, а контролери, моделі та подання додаються, залежності стають складнішими.



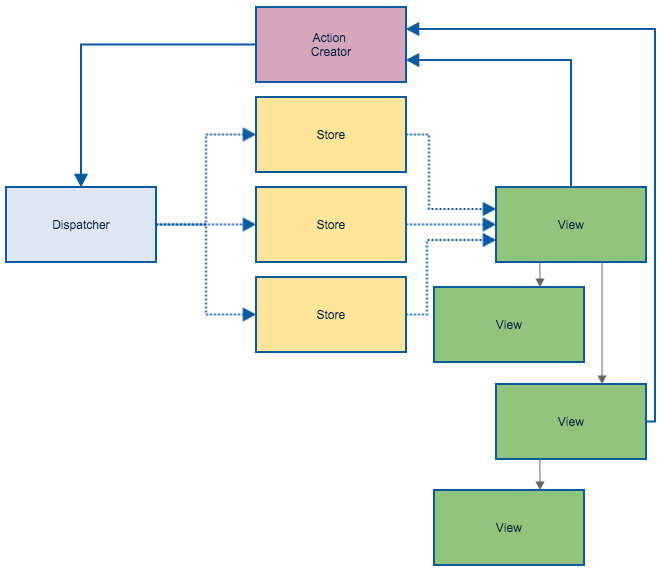
*Рис. 2.* **Схематичне зображення складного додатку MVC-архітектури.**

З додаванням всього трьох переглядів (рис. 2), одного контролера і однієї моделі, графік залежності вже важче простежити. Коли користувач взаємодіє з інтерфейсом користувача, виконуються кілька шляхів коду розгалуження, і проблеми з налагодженням у стані програми стають вправою для з'ясування того, який модуль (або модулі) в одному (або декількох) з цих потенційних шляхів коду містить помилку. У гіршому випадку взаємодія з користувачем спричинить оновлення, які, в свою чергу, спричиняють додаткові оновлення, що призводить до помилок та каскадних ефектів, схильних до помилок, уздовж кількох з цих, іноді перекриваються, шляхів.

Flux уникає цієї конструкції на користь односторонній потік даних. Усі взаємодії користувачів у поданні викликають *дію (Action Creator)*, що призводить до того, що подія *Action*  викликається диспетчером (*Dispatcher*). Диспетчер є однією точкою виклику для всіх дій у Flux-додатку. Дія відправляється від диспетчера в  *сховище (Store)*, які оновлюються у відповідь на дію (рис. 3).

 *Рис. 3.* **Схематичне зображення простого додатку Flux-архітектури.**

Потік істотно не змінюється для додаткових станів або виглядів (*Viev*). Диспетчер просто відправляє всі дії в *усі*  стани в додатку. Зверніть увагу, що він не містить знань про те, як насправді оновити стан – саме сховище містить цю бізнес-логіку. Кожен стан відповідає за домен програми, і оновлює себе тільки у відповідь на дії.



*Рис. 4.* **Схематичне зображення складного додатку Flux-архітектури.**

Коли стан оновлюється, він випускає подію зміни. У багатьох додатках React спеціальні перегляди (відомі іноді як "контролер-перегляди") відповідають за спостереження за цією подією змін, читання нових даних станів та передачу цих даних через властивості до дочірніх переглядів. Це не рідкість у програмі React для події зміни стану, щоб викликати повторний рендеринг подання верхнього рівня, ефективно ре-рендерить всю ієрархію подання (яка ефективно обробляє React). Це повністю дозволяє уникнути складних помилок і проблем з продуктивністю, які можуть виникнути через спроби спостерігати за конкретними змінами властивостей моделей і лише незначно змінювати частини переглядів.

## Ключові властивості. Архітектура потоку має кілька ключових властивостей, які роблять її унікальною і забезпечують важливі гарантії, всі з яких зосереджені навколо того, щоб зробити потік даних явним і легко зрозумілим і збільшити локалізацію помилок.

### Примусова синхронність. Дії (*Actions*) та їх обробники (*Handlers*) всередині сховища синхронні. Усі асинхронні операції мають викликати дію, яка повідомляє системі про результат операці. Хоча дії можуть робити асинхронні запити на сервер (API-call), обробники дій у сховищі в ідеалі цього не зроблять. Це правило робить потік інформації в додатку надзвичайно явним; налагодження помилок у стані застосунку просто включає в себе з'ясування того, яка дія викликала поганий стан, а потім знайти неправильну логіку, яка відповіла на цю дію.

### Інверсія контролю. Оскільки стани оновлюються всередині сховища у відповідь на дії (а не оновлюються ззовні контролером або подібним модулем), жодна інша частина системи не повинна знати, як змінити стан програми. Вся логіка оновлення стану міститься в самому сховищі. І, оскільки сховище тільки коли-небудь оновлюються у відповідь на дії, і тільки синхронно, тестування сховища просто стає питанням: поставити їх у початковий стан (INITIAL\_STATE), надіслати їм дію та перевірити, чи змінився правильно стан.

### Семантичні дії. Оскільки сховище повинно оновлювати себе у відповідь на дії, дії, як правило, семантично описові. Наприклад, у Flux-програмі форуму, щоб позначити повідомлення як прочитане, можна надіслати дію з відповідним типом . Дія (і компонент, що генерує дію) не знає, *як* виконати оновлення, але *описує,* що він хоче, щоб сталося. MARK\_THREAD\_READ. Через цю властивість рідко доводиться змінювати типи дій, лише те, як на них реагує сховище. До тих пір, поки ваша програма має концепцію "потоку", і у вас є кнопка або інша взаємодія, яка повинна позначати повідомлення як прочитане, тип дії семантично дійсний. MARK\_THREAD\_READ

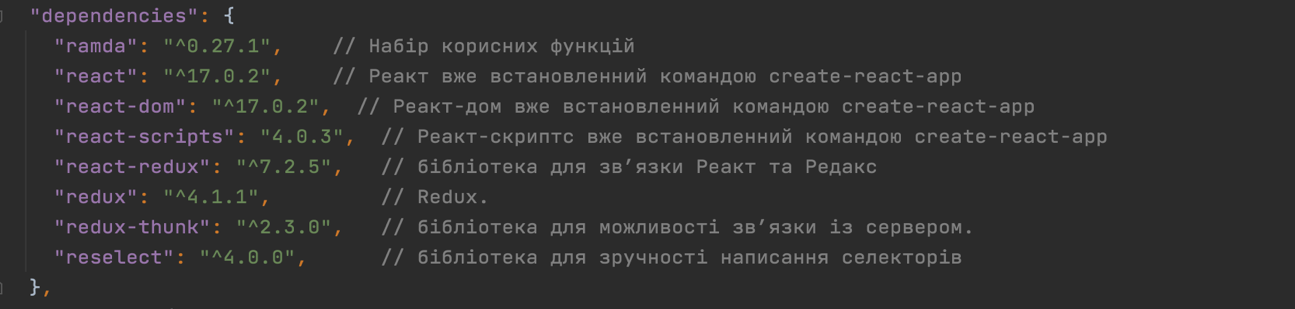
### Без каскадних дій. Потік забороняє відправку другої дії в результаті відправки дії. Це допомагає запобігти каскадним оновленням, які важко налагоджують, і допомагає вам думати про взаємодію у вашому додатку з точки зору семантичних дій.

**Реалізація Flux-архітектури***.* На сьогоднішній день є багато реалізацій Flux-архітектури. Ось найпоширеніші з них:

* Redux — це бібліотека JavaScript з відкритим кодом для керування станом програми. Найчастіше використовується в поєднанні з React або Angular для розвитку клієнтської частини. Містить ряд інструментів, які значно полегшують передачу даних зберігання через контекст.
* Fluxxor - це набір інструментів для полегшення побудови шарів даних JavaScript за допомогою Flux-архітектури шляхом об'єднання багатьох основних Flux-концепцій. Він особливо добре працює в поєднанні з React і містить кілька помічників, щоб полегшити інтеграцію з додатками React.
* MobX - це автономна бібліотека для управління front-end станом програми. MobX забезпечує узгодженість внутрішнього стану інтерфейсного додатку, надаючи зручні інструменти для його зміни. Спрощено, MobX дозволяє реалізувати ланцюжок: «Виконати дію» → «Змінити стан» → «Змінити вигляд». При цьому зміни відбуваються атомно і автоматично - в результаті гарантовано, що не буде моменту, коли стан буде не узгодженим.
* VueX - шаблон керування станом застосунку + бібліотека для Vue.js додатків. Він служить централізованим магазином для всіх компонентів у додатку, з правилами, які гарантують, що стан може бути мутований лише передбачуваним чином.
* NgRx – це фреймворк для побудови реактивних додатків в Angular. NgRx надає бібліотеки для: управління глобальним та локальним станом,  ізоляція побічних ефектів для просування більш чистої архітектури компонентів,  управління зборами сутностей,  інтеграція з маршрутизатором, інструмент розробника, який покращує досвід розробників при створенні багатьох різних типів додатків.

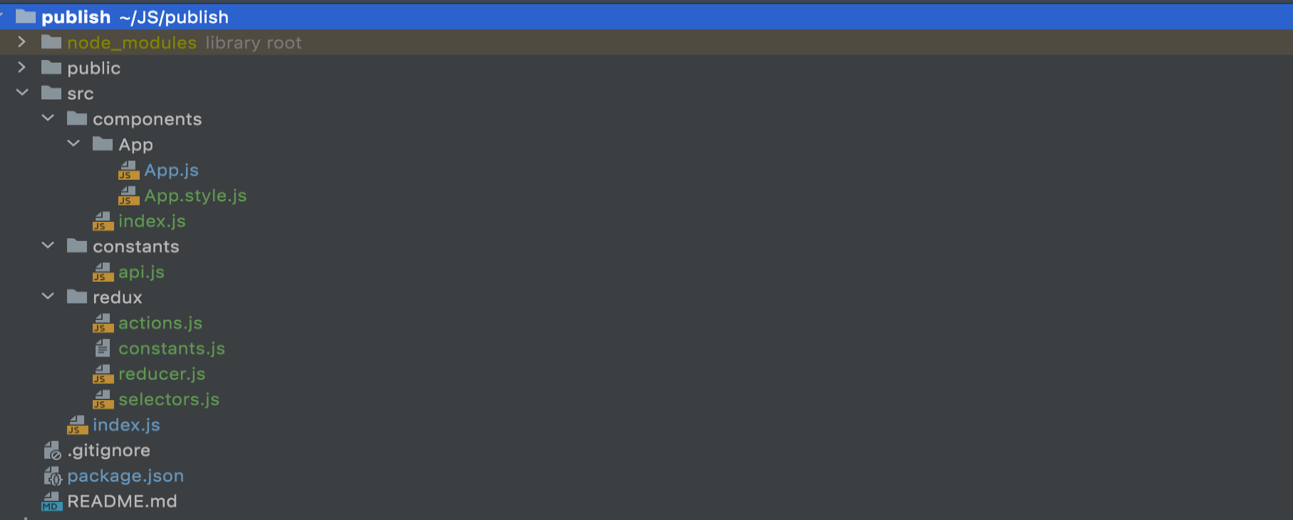
**Приклад***.* Створимо примітивний додаток. Візуальною частиною будемо нехтувати, так як це не є темою цієї статті.

* Створимо пустий проект, та виконаємо команду *create-react-app*
* За допомогою NPM інсталюємо ще низку бібліотек (рис. 5):



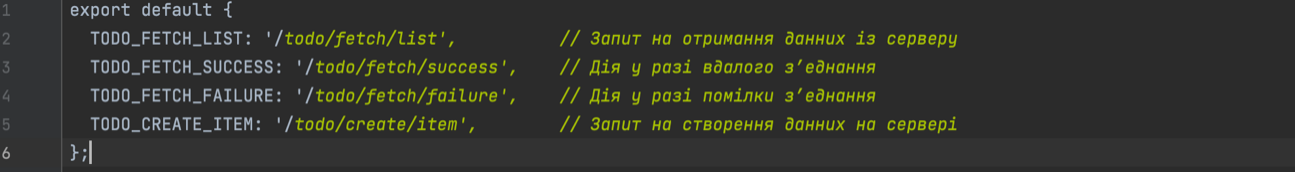
*Рис. 5.* **Список залежностей додатку.**

* Створимо всі файли, які нам знадобляться (рис. 6):



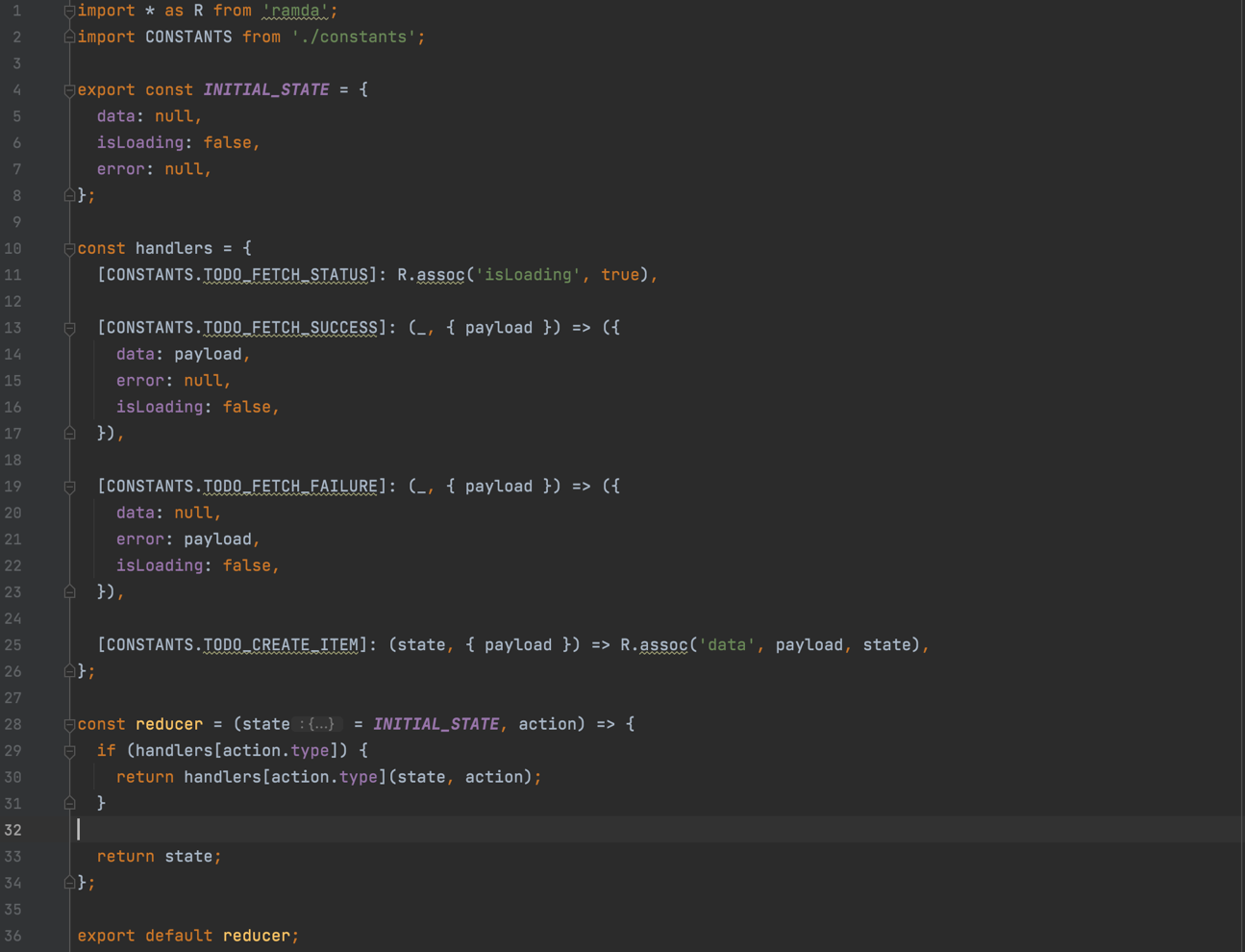
*Рис. 6.* **Структура проекту.**

* У файлі *constant.js* (рис. 7) опишемо усі потрібні нам типи дій.



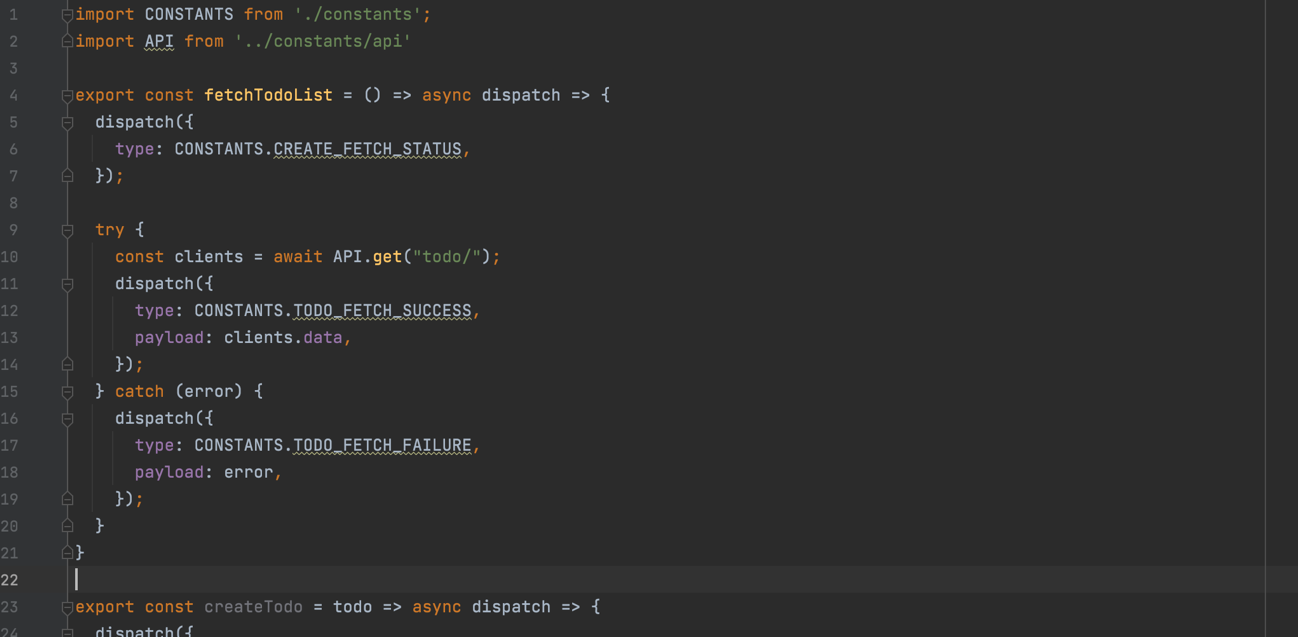
*Рис. 7.* **constants.js.**

* У файлі *reduser.js* (рис. 8)   Створимо сховище, де будуть зберігатися данні, помилки та флаг загрузки. Також опишемо маніпуляції згідно з типами. Коли пішов запит на сервер *isLoading: true*, коли сервер відповів коректно у *data* записуємо відповідь серверу, а якщо сталося помилка, тоді у *error* записуємо повідомлення помилки. У разі невідомої дії сховище повертається до первісного стану *(INITIAL\_STATE).*



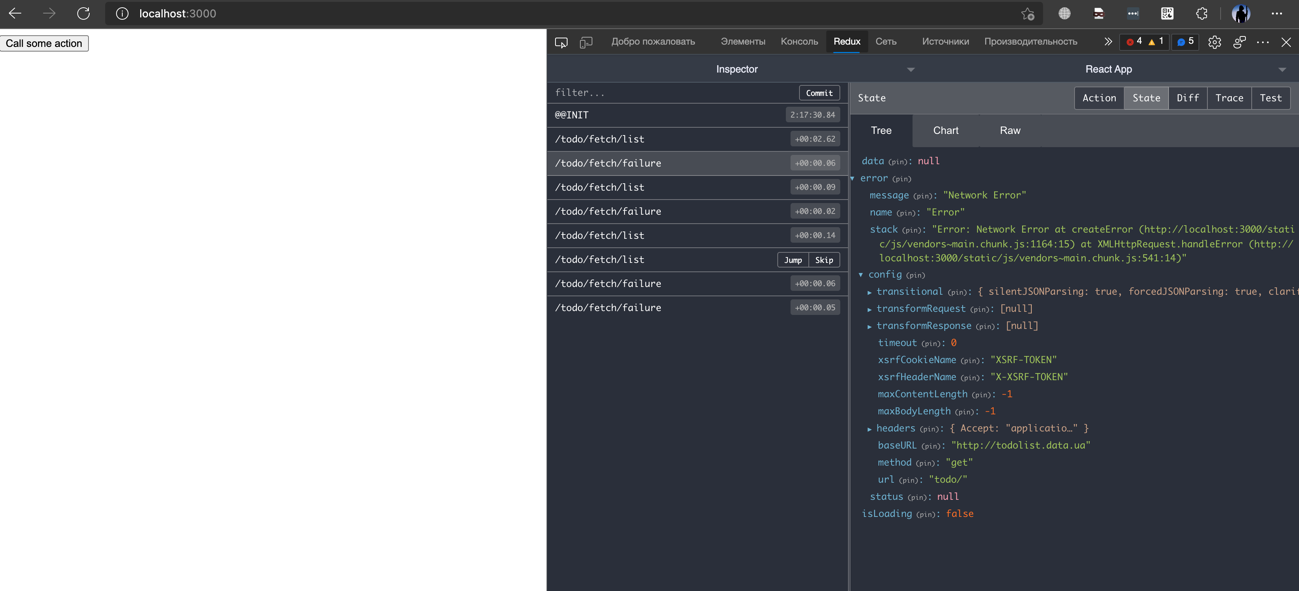
*Рис. 8.* **reducer.js.**

* У файлі  actions.js (рис. 9) опишемо дві дії. Запит сервер на отримання отримання, та на зміну.



*Рис. 9.* **actions.js.**

**Висновки.**За допомогою цих нескладних маніпуляцій ми маємо логічний веб-додаток із відокремленою графічною частиною та логічною.   За допомогою  *\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_* (рис. 10) ми можемо спостерігати за історією виклику дій, та зміною станів на кожному кроці життєвого циклу додатку. А це означає, що якщо десь розробник допустить помилку – побачити її буде у рази легше.  На скріншоті ми бачимо що після запиту на сервер викликається дія *'/todo/fetch/failure'* оскільки його не існує.



*Рис. 9.* **Середа розробки Redux у браузері.**

ВИСНОВНИ

JavaScript був створений як мова сценаріїв для Netscape. Потім він був відправлений в ESMA international для стандартизації. Це призвело до появи новогостандарту ESMAscript. Наступні версії JavaScript вже базувалися на стандарті ESMAscript.

ESMAscript є стандартом, а JavaScript є найпопулярнішою реалізацією цього стандарту.

Однак JavaScript все ще був «найбільш незрозумілою мовою програмування в світі». Серед його проблем були глобальні зони видимості змінних, однопоточність, відсутність суворої типізації та принципів ООП.

Але в червні 2015 року все змінилося. Був випущений новий стандарт - ES6, додавши наступні зміни:

• деструктивне призначення;

• стрілочні функції;

• шаблонні строки;

• локальні змінні та константи;

• ітератори та генератори;

• класи;

• проміси та async/await.

З тих пір щороку випускається новий стандарт ESMAscript.

Про попит на будь-яку мову можна судити по кількості написаних для неї бібліотек. NPM (node package manager) це головний репозиторій пакетів для середы node.js. Він є найбільшим у світі. Станом на 2017 рік в NPM було розміщено понад 380 тисяч пакетів. Для порівняння в репозиторії packagist(php) – 123 тисячі, в PyPI(Python) – 96 тисяч і в nuget (.NET) – 70 тисяч.

В середньому через NPM на місяць встановлюється понад 18 мільярдів пакетів.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Javascript фреймворки. [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://maxsite.org/page/javascript-framework.
2. Куріченко А. Javascript фреймворки [Електронний ресурс] / Олексій Куріченко. – 2015. – Режим доступу до ресурсу: https://www.reclamare.ua/blog/javascript-frejmvorki/.
3. Single Page Application: як працює сайт-додаток [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://thecode.media/spa/.
4. Що таке SPA або односторінковий портал? [Електронний ресурс] – Режим доступу до ресурсу: http://www.codenet.ru/webmast/js/spa/.
5. Що таке SPA-додатки? [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://wezom.com.ua/blog/chto-takoe-spa-prilozheniya.
6. Морозова Я. Что такое SPA, MPA и PWA [Електронний ресурс] / Яна Морозова. – 2021. – Режим доступу до ресурсу: https://www.azoft.ru/blog/spa-mpa-pwa/.
7. Флэнаган Д. JavaScript: Полное руководство / Дэвид Флэнаган., 2021. – (7). – (722).
8. Хавербеке М. Выразительный JavaScript / Марейн Хавербеке.. – 480 с.
9. Кайл С. ES6 и не только / Симпсон Кайл., 2018. – 336 с.
10. Пауэлл Д. Single Page Web Applications: JavaScript End-to-end / Д. Пауэлл, М. Миковский., 2013. – 432 с.
11. Abramov D. Getting Started with Redux [Електронний ресурс] / Dan Abramov. – 2021. – Режим доступу до ресурсу: https://redux.js.org/introduction/getting-started.
12. Tilley M. What is Flux [Електронний ресурс] / Michelle Tilley. – 2014. – Режим доступу до ресурсу: http://fluxxor.com/what-is-flux.html.
13. Flux и Redux [Електронний ресурс] // Metanit. – 2021. – Режим доступу до ресурсу: https://metanit.com/web/react/5.1.php.
14. Flux-архитектура [Електронний ресурс] // Википедия. – 2019. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Flux-архитектура
15. Catalin V. Изучаем архитектуру Flux в React [Електронний ресурс] / Vasile Catalin. – 2017. – Режим доступу до ресурсу: https://code.tutsplus.com/ru/tutorials/getting-started-with-the-flux-architecture-in-react--cms-28906.
16. Yangshun T. In-Depth Overview [Електронний ресурс] / Tay Yangshun. – 2019. – Режим доступу до ресурсу: https://facebook.github.io/flux/docs/in-depth-overview.
17. MobX [Електронний ресурс] – Режим доступу до ресурсу: https://mobx.js.org/README.html.
18. Библиотека MobX [Електронний ресурс] // ТМ «Web Creator» – Режим доступу до ресурсу: https://web-creator.ru/technologies/webdev/mobx.
19. What is Vuex [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://next.vuex.vuejs.org/.
20. What is NgRx [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://ngrx.io/docs>.
21. Бодух А. Flux Architecture / Адам Бодух., 2016. – 352 с.
22. Garreau M. Redux in Action / M. Garreau, W. Faurot., 2018. – 312 с.