МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Савченко А.С.

"____"_____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТРА"

ЗА СПЕЦІАЛІЗАЦІЄЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)"

Тема: "Веб додаток для візуалізації статистики захворювань на COVID-19"

Виконавець: Бондар Святослав Олександрович

Керівник: к.т.н., доцент Харченко Олександр Григорович

Нормоконтролер: _____ Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, спеціалізація: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології (за галузями)"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Савченко А.С.
" " 2020 р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Бондаря Святослава Олександровича

1. Тема роботи: "Веб додаток для візуалізації статистики захворювань на COVID-19".

Затверджена наказом ректора від «05» жовтня 2020 за №1891/ст.

2. Термін виконання роботи: з 5 жовтня 2020р. до 31 грудня 2020р.

3. Вихідні данні до роботи: додаток для моніторингу захворюваності на COVID-19 в світі. Використання данних з всесвітніх відкритих джерел. Створення веб-додатку за допомогою бібліотеки ReactJS для відображення статистики.

4. Зміст пояснювальної записки: 1) Огляд основних методів створення веб додатків. 2) Інструменти для розробки веб додатка. 3) Розробка веб додатка для моніторингу захворюваності на COVID-19 в світі. 4) Використання додатку клієнтом.

5. Перелік обов'язкового ілюстративного матеріалу: таблиця, рисунки, графіки, а також слайди презентації доповіді у PowerPoint.

6. Календарний план-графік

№ з/п	***Завдання***	***Термін виконання***	***Підпис керівника***
1.	Провести аналіз літератури за темою дипломної роботи.	05.10.20 – 08.10.20	
2	Написати розділ «Огляд основних методів створення веб додатків».	08.10.20 – 16.10.20	
3	Провести аналіз необхідних компонентів для реалізації веб додатку.	16.10.20 – 20.10.20	
4	Написати розділ «Інструменти для розробки веб додатка».	20.10.20 – 24.10.20	
5	Розробка веб додатка для моніторингу COVID-19.	24.10.20 – 12.11.20	
6	Написати розділ «Розробка веб додатка для моніторингу захворюваності на COVID-19 в світі».	12.11.20 – 18.11.20	
7	Написати розділ «Використання додатку клієнтом».	18.11.20 – 25.11.20	
8	Створення доповіді та презентації.	25.11.20 – 28.11.20	
9	Оформлення та друк пояснювальної записки дипломної роботи.	28.11.20 – 21.12.20	

7. Консультація з окремого(мих) розділу(ів) роботи:

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв

8. Дата видачі завдання «05» жовтня 2020р.

Керівник дипломної роботи Райчев І.Е.

Завдання прийняв до виконання Бондар С. О.

(підпис випускника) (ПІБ)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи "Веб додаток для візуалізації статистики захворювань на COVID-19" містить 86 сторінок, 15 рисунків, 41 наукове джерело.

Об'єкт дослідження: система моніторингу розповсюдження пандемії COVID-19 в світі.

Мета роботи: дослідити сучасні технології створення веб додатків. Побудувати веб додаток для моніторингу розповсюдження пандемії COVID-19 в світі.

Методи дослідження: є дослідження технологій створення веб-додатків на основі реактивних технологій та створення найсучаснішого SPA з використанням фреймворку React.js, розробки структури та вибору методів обробки даних і алгоритмів функціонування програмних модулів, забезпечення якісних показників роботи інформаційної системи та створення зручного для кінцевого користувача інтерфейсу.

Результати магістерської роботи: Створено сучасний SPA з використанням фреймворку React.js для відстеження статистики захворюваності на COVID-19 в світі. Було створено сучасний інтерфейс з інтерактивною мапою та графіками. Дані отримуються з відритих джерел та є актуальними і надійними.

Ключові слова: СУЧАСНИЙ ОДНОСТОРІНКОВИЙ ДОДАТОК ДЛЯ ВІДСТЕЖЕННЯ СТАТИСТИКИ НА COVID-19 В СВІТІ, SINGLE PAGE APPLICATION НА БАЗІ REACTJS. ВІДКРИТІ API, COVID-19 TRACKER, СИСТЕМА МОНІТОРИНГУ РОЗПОВСЮДЖЕННЯ ПАНДЕМІЇ COVID-19 В СВІТІ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTML – HyperText Markup Language

ПЗ – Програмне забезпечення

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

HTTP – HyperText Transfer Protocol

API – Application programming interface

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

XML – eXtensible Markup Language

SPA – Single Page Application

MVC – Model View Controller

DOM – Document Object Model

AJAX – Asynchronous JavaScript And XML

REST – Representational State Transfer

API – Application Programming Interface

ВСТУП

Всесвітнє павутиння не схоже на те, яким воно було 5 років тому, тим паче не схоже на те, яким воно було під час свого створення. За 30 років розвитку інтернету, технології та підходи до розробки кардинально змінилися.

Скриптова мова JavaScript надала можливість створення функціональних веб-додатків. За допомогою JavaScript було розроблено декілька підходів для створення Single Page Application (SPA). Зараз розробники мають достатньо технологій, щоб змінити поведінку традиційного веб-сайту й зробити його схожим на нативний додаток операційної системи. Серед них найпопулярнішою технологією є розробка веб-додатків на базі фреймворку React.js та платформи Node.js.

SPA – це повноцінні додатки у вікні браузера. Вони відрізняються від звичайних веб сторінок більшою функціональністю та швидкістю роботи. SPA працює без перезавантаження сторінки. Ідея SPA не нова, вона почала з’являтися як тільки з’явилися JavaScript та AJAX.

Незважаючи на назву, суть концепції односторінкового додатку полягає не в обмеженні сторінок веб-сайту, а в наданні сайту вигляду нативного додатка, шляхом позбавлення перезавантажень й перенесенні відтворюючої логіки з сервера до клієнта, тим самим зменшуючи залежність роботи сайту від сервера та інтернет з'єднання.

Оновлення відбувається частковим підвантаженням контенту у вже існуючу структуру, причому таке підвантаження не потребує великих витрат часу та пам’яті. Швидкість забезпечується тим, що браузер обробляє не HTML розмітку а JavaScript-об'єкти. Така заміна сприяє пришвидшенню роботи веб-додатку.

Новітні SPA розробляються за допомогою JS-фреймворків та бібліотек. Найкращим прикладом бібліотеки є React.js. Це відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових додатків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

РОЗДІЛ 1. ОГЛЯД ОСНОВНИХ МЕТОДІВ СТВОРЕННЯ ВЕБ-ДОДАТКІВ

1.1. Веб додаток

Веб-додаток – додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер. Браузер може бути реалізацією так званих тонких клієнтів – логіка додатку зосереджується на сервері, а функція браузера полягає переважно у відображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача.

Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є кросплатформними сервісами. Завдяки цієї універсальності і відносної простоти розробки веб-додатки стали широко популярними в кінці 1990-х – початку 2000-х років.

Істотною перевагою побудови веб-додатків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб писати різні версії для Windows, MacOS X, GNU/Linux й інших операційних систем, додаток створюється один раз для довільно обраної платформи і на ній розгортається .

Переваги веб-додатків:

- насичений функціоналом інтерфейс;
- швидка реакція інтерфейсу, завдяки відсутній необхідності звертатися до серверу при кожній дії;
- значне зменшення навантаження на сервер;
- значне спрощення логіки та складності серверу;
- схожість додатків з нативними додатками операційної системи.

Проте різна реалізація HTML, CSS, DOM й інших специфікацій в браузерах може викликати проблеми при розробці веб-додатків і подальшої

підтримки. Крім того, можливість користувача налаштовувати багато параметрів браузера (наприклад, розмір шрифту, кольори, відключення підтримки сценаріїв) може перешкоджати коректній роботі додатку .

1.2. Актуальність створення веб-додатків

Створення веб-додатків сьогодні вважається одним з перспективних напрямків діяльності для багатьох компаній, зайнятих у будь-якій сфері.

Відмова від традиційного програмного забезпечення і перенесення бізнес-інструментів на веб – це тренд, який не можна упускати. Але багато клієнтів, та й самі програмісти-консерватори не хочуть виходити з тіні минулого. Саме тому вони завжди будуть відставати від тих, хто не боїться реалізовувати креативні ідеї, шукати нові підходи, розробляти програми, які вже потрібні будуть в найближчому майбутньому.

Розробка веб-додатків стала актуальною темою для багатьох спеціалізованих компаній, і одночасно доступною для простих користувачів.

Причини збільшення ролі веб-додатків очевидні. Головною перевагою SPA додатків вважається їх універсальність. Вони однаково добре функціонують як на персональних комп'ютерах, так і на мобільних пристроях. Планшети, смартфони без проблем працюють з проектами, розробленими за принципом SPA.

Також, головною перевагою односторінкових додатків можна вважати широке охоплення пристроїв. Отже, створення односторінкового додатку дозволяє розраховувати на значно більшу цільову аудиторію, ніж при використанні стандартних методів веб-розробки .

Другою перевагою SPA вважається насичений інтерфейс. Ця перевага обумовлена тим, що на одній веб-сторінці набагато простіше створити широкофункціональний інтерфейс.

Такий підхід суттєво спрощує процеси зберігання інформації, а також управління станом, представленням та анімацією.

Ще одна перевага односторінкових додатків полягає у спрощенні процедури завантаження контенту. Якщо сайт працює з шаблонами, то разом із завантаженням будь-якої сторінки цього порталу користувач обов'язково завантажує і розмітку

шаблону. В SPA нічого кешувати, а це означає, що відбувається суттєва економія часу і ресурсів .

Від користувача, тепер не потрібно установки додаткового програмного забезпечення. SPA додатки більш керовані, знижені вимоги до пристрою клієнта та працюють з будь-якого пристрою. Вони створюються один раз для довільно обраної платформи, тому немає необхідності в різних версіях веб-додатку для різних ОС і специфічних серверних ОС, такі веб-додатки є кросплатформними.

1.3. Огляд принципів роботи веб-додатків

Як правило, веб-додатки не вимагають встановлення додаткового програмного забезпечення на стороні клієнта, а вся логіка, в основному, виконується на стороні сервера. Для відображення інтерфейсу користувача використовується браузер – програма, здатна розпізнавати мову розмітки HTML (і супутні технології – таблиці стилів CSS, клієнтську скриптову мову програмування JavaScript). Браузер зазвичай прийнято називати "тонким клієнтом", тобто клієнтом, який містить мінімальну кількість бізнес-логіки .

Найчастіше веб-додаток складається з трьох основних компонентів:

- клієнтська частина;
- серверна частина;
- база даних.

Клієнтська частина веб-додатку – це графічний інтерфейс, який відображається в веб-браузері. Користувач взаємодіє з веб-додатком саме через веб-браузер.

Серверна частина веб-додатку – це програма або сценарій на сервері, що обробляє запити користувача веб-браузера. Найчастіше серверна частина веб-додатку розробляється за допомогою JavaScript або PHP. При кожному переході користувача за посиланням веб-браузер відправляє запит до сервера.

Сервер обробляє цей запит, викликаючи деякий JavaScript-сценарій, який формує веб-сторінку, описану мовою HTML, і відсилає клієнтові по мережі. Веб-браузер відразу відображає отриманий результат у вигляді веб-сторінки. Цей процес схематично зображено на рисунку 1.1.

База даних веб-додатку – це програмне забезпечення на сервері, що займається зберіганням даних та їх видачою в потрібний момент. У випадку форуму або блогу, дані, що зберігаються в БД – це пости, коментарі, новини, і так далі. База даних розташовується на сервері .

Серверна частина веб-додатку звертається до бази даних, отримуючи дані, які необхідні для формування сторінки, запитаної користувачем .

Рис. 1.1 Життєвий цикл веб-сайту

Для програмування серверної частини веб-додатку може використовуватися мова скриптова JavaScript, але також можна використовувати практично будь-які сучасні мови програмування: PHP, Perl, Ruby, Java, Golang, TypeScript, платформа .NET.

Необхідні компоненти для роботи користувача з веб-додатком:

- браузер (тонкий клієнт),
- веб-сервер (серверна частина),
- протокол взаємодії клієнта і сервера (HTTP)
- мова розмітки для створення документів (HTML).

Для того, щоб веб-додаток став доступним в мережі, його необхідно розмістити в рамках веб-сервера (спеціальна програма, яка обробляє запити з мережі). Після цього додаток отримає свою унікальну адресу в рамках протоколу HTTP (наприклад, http://www.myapplication.com/page1.html).

Використовуючи цю адресу користувач може звернутися до додатка. Для цього він повинен запустити браузер (клієнтську програму) і ввести в адресному рядку адресу програми. У цей момент браузер згенерує запит до сервера і відправить його, використовуючи протокол HTTP. У момент, коли сервер прийме цей запит, він зможе розпізнати, що саме потрібно від нього на основі отриманого запиту.

Використовуючи ці дані, сервер згенерує відповідь і відправить її назад клієнтові також використовуючи протокол HTTP. Зазвичай відповідь містить гіпертекстову розмітку HTML, що містить структуру документа, яка передається користувачеві. Після того, як браузер отримає відповідь у вигляді HTML-документа, він негайно покаже його користувачеві. Таким чином, була виконана взаємодія клієнта і сервера.

Найчастіше документ HTML містить посилання на зображення, інші медіа-файли, таблиці стилів або клієнтські сценарії. В цьому випадку браузер генерує ще кілька аналогічних запитів до веб-сервера. Однак, в цьому випадку веб-сервер передає клієнтові вже не HTML-документ, а відповідні запитувані ресурси (зображення, таблиці стилів, клієнтські сценарії).

У найпростішому випадку на сервері може бути збережений готовий документ HTML, який за запитом буде передаватися користувачу. Це – так званий, статичний документ. У цьому випадку він просто зчитується з жорсткого диска і передається клієнтові. Однак, такий сценарій роботи в глобальній мережі стає все більш рідкісним.

Інший підхід – генерація коду HTML в процесі обробки запиту від клієнта. Цей підхід дозволяє зробити веб-додаток більш інтерактивним, чуйним на дії клієнта і створює враження цього додатка, а не простого завантаження HTML-документів. Таким чином, ми, маючи можливість генерувати HTML-код сторінки, можемо впливати на ту інформацію, елементи управління та інші аспекти інтерфейсу, які побачить користувач.

По суті, завдання веб-додатки і полягає в тому, щоб генерувати потрібний HTML-код, в залежності від дій користувача. Однак, можливості веб-додатків

можуть не обмежуватися тільки генерацією розмітки HTML – вони можуть генерувати зображення, клієнтські сценарії, таблиці стилів і інші ресурси, які можуть бути завантажені користувачем. Проте, основним сценарієм є все-таки генерація кінцевого документа HTML.

Як вже було зазначено, для взаємодії клієнта і сервера використовується протокол HTTP. Цей протокол працює за схемою "запит-відповідь". У момент, коли клієнт хоче звернутися до сервера, він генерує запит, який відправляється сервера. Сервер обробляє цей запит і готує ресурси, які будуть відправлені клієнту. Після цього сервер генерує відповідь, в якому містяться всі необхідні дані і відправляє клієнту.

Робота веб-додатків полягає в формуванні необхідних даних як раз в момент підготовки ресурсів на сервері. Зазвичай в цей момент запускається деякий програмний код, який містить певну бізнес логіку.

1.4. Порівняльний аналіз настільного додатку та веб-додатку

Веб-додатки істотно відрізняються від настільних додатків. Останні запускаються на комп'ютері клієнта і виконують свій код саме там. Тому настільні додатки часто мають багатший інтерфейс і дозволяють реалізовувати більшу кількість сценаріїв.

Перевага розробки веб-додатків з підтримкою стандартних функцій веб-браузера полягає в тому, що функції повинні виконуватися незалежно від ОС клієнта. Замість того щоб писати різні версії для Windows, MacOS X, GNU/Linux і інших ОС, веб-додаток створюється один раз для довільно вибраної платформи і на ній розгортається.

У порівнянні з настільними додатками, веб-додатки мають обмежені можливості щодо формування призначеного для користувача інтерфейсу і клієнтської функціональності. З цієї причини за останній час склався стереотип про те, що серйозні додатки (наприклад, бізнес-додатки) – це, як правило, настільні додатки.

Однак, розвиток веб-технологій довів, що веб-додатки також можуть реалізовувати багаті сценарії і успішно конкурувати з настільними додатками. Крім

того, за останні кілька років дуже активно розвиваються технології, що дозволяють зробити веб-додатки ще більш інтерактивними. До них відносяться технологія AJAX, яка на основі клієнтських сценаріїв JavaScript може зробити взаємодію більш інтерактивною.

Незважаючи на те, що існує ряд технологій, що спрощують створення динамічних веб-додатків, їх розробка залишається досить трудомістким завданням. Розробка веб-додатків істотно відрізняється від розробки настільних систем. Веб-додатки виконуються на сервері. Весь програмний код виконується в рамках веб-сервера, а клієнтові доставляється вже готова розмітка HTML, яка відображається всередині браузера. Веб-додатки не зберігають стан. По-суті, сервер "забуває" про користувача після того, як обробив його запит.

Обидва ці фактори суттєво впливають на процес розробки веб-додатків. Через це при побудові будь-якого веб-додатка доводиться вирішувати типові завдання – способи зберігання інформації про користувача, організація сеансів роботи користувача, способи переходів від сторінки до сторінки, механізми оптимізації ефективності (наприклад, кешування).

При реалізації кожного веб-додатку розробнику доведеться зіткнутися з цими проблемами і вирішити їх. Оскільки набір цих завдань є досить стандартним і однаково вирішується для більшості веб-додатків, то його реалізація винесена в окремі технології, які називаються технологіями для розробки веб-додатків.

До таких технологій належать Microsoft ASP.NET, PHP, Ruby on Rails і ін. В них, фактично, містяться всі компоненти, необхідні для реалізації веб-додатків і враховують їх специфіку.

Основна перевага веб-додатків полягає в процесі розгортання програми.. Якщо настільний додаток необхідно встановити на кожне робоче місце де воно буде використовуватися, то веб-додаток потрібно розмістити на сервері і дати посилання на нього всім користувачам.

Особливо актуальний цей аспект там, де присутня велика кількість робочих місць. Важливо зазначити, що, в разі оновлення програмного коду, веб-додатки також мають перевагу – для їх поновлення потрібно тільки оновити код на сервері.

При цьому настільний додаток треба було б оновлювати на кожному робочому місці.

1.5. Структура веб додатку

Зазвичай програми розбиваються на логічні фрагменти, які називаються "рівнями", де кожному рівню присвоюється роль. Традиційні програми складаються лише з 1 рівня, який розміщений на клієнтській машині, але веб-додатки за своєю природою піддаються багаторівневому підходу. Хоча можливих багато варіантів, найпоширенішою структурою є трирівнева програма. У своїй найпоширенішій формі три рівні називаються презентацією, застосуванням та зберіганням, у цьому порядку.

Веб-браузер - це перший рівень (презентація), механізм, що використовує певні технології динамічного веб-вмісту (наприклад, ASP, CGI, ColdFusion, Dart, JSP/Java, Node.js, PHP, Python або Ruby on Rails) - середній рівень (логіка програми), а база даних - це третій рівень (сховище). Веб-браузер надсилає запити середньому рівню, який обслуговує їх, роблячи запити та оновлення щодо бази даних та генеруючи користувальницький інтерфейс.

Для більш складних додатків 3-рівневе рішення може виявитися недостатнім, і може бути корисним використовувати підхід з ярусами, де найбільшою вигодою є розбиття бізнес-логіки, що знаходиться на рівні додатків, на більш дрібний модель.

Ще однією перевагою може бути додавання рівня інтеграції, який відокремлює рівень даних від решти рівнів, надаючи простий у використанні інтерфейс для доступу до даних. Наприклад, доступ до даних клієнта здійснюється шляхом виклику функції "list_clients ()" замість того, щоб робити запит SQL безпосередньо до таблиці клієнта в базі даних. Це дозволяє замінити базову базу даних, не вносячи жодних змін в інші рівні.

Є деякі, хто розглядає веб-додаток як дворівневу архітектуру. Це може бути "розумний" клієнт, який виконує всю роботу і запитує "німий" сервер, або "німий" клієнт, який покладається на "розумний" сервер. Клієнт обробляв би рівень презентації, сервер мав би базу даних (рівень зберігання), а бізнес-логіка (рівень додатків) знаходився б на одному з них або на обох. Хоча це збільшує

масштабованість додатків та відокремлює дисплей та базу даних, це все одно не дає можливості справжньої спеціалізації шарів, тому більшість програм переросте цю модель.

1.6. Переваги веб додатків

Економічна розробка: Головною перевагою розробки та використання веб-програм є доступність із меншими витратами. Загальний додаток можна розробити, враховуючи єдину операційну систему. Тестери програм виконують своє завдання та забезпечують його безперебійну роботу на всіх можливих платформах. Це нормально, якщо ви не протестуєте його на всіх можливих версіях. Користувачі можуть отримати доступ до системи через єдине середовище. Це вважається основною перевагою для бізнесу.

Доступ у будь-який час: Веб-програми настільки гнучкі, що ними можна керувати в будь-який час і в будь-якому місці. Для доступу до веб-сайту достатньо ПК з підключенням до Інтернету.

Налаштування: Розробники веб-додатків поділяють, що легко налаштувати веб-програми порівняно з настільними програмами. Змінення зовнішнього вигляду веб-програми полегшує роботу веб-програм. Розумні розробники можуть розробляти високо налаштовані продукти з меншими зусиллями завдяки розробці веб-додатків.

Підтримує різні пристрої: на ринку доступний ряд пристроїв та їх версій. Веб-програми легко підтримують усі пристрої, підключені до Інтернету. Керуйте веб-програмами просто на КПК та смарт-телефонах. Завдяки послугам розробки веб-додатків стало безпроблемно надсилати та отримувати інформацію з розширеною продуктивністю.

Покращена сумісність: Ізольовані настільні системи поки що не сумісні в порівнянні з веб-додатками. Цю заяву підтримали багато лідерів галузі, що за допомогою веб-додатків можна досягти більшого рівня взаємодії. Візьмемо приклад. Якщо ви хочете інтегрувати модуль кошика для покупок із пакетом бухгалтерського обліку, це означає, що розробник зручний для веб-програм.

Не обробляйте підвищене навантаження: У епоху цифрових технологій веб-програми часто зазнають змін. Іноді веб-програми потрібно розширити за допомогою ексклюзивних функцій. У цій ситуації здається простим завданням збільшити ємність процесора. У випадку, якщо веб-програма вимагає більше енергії, це питання легко вирішити шляхом оновлення апаратного забезпечення сервера. “Кластеризація” - це процес збільшення можливостей веб-програмного забезпечення. Експерт може також інтегрувати нові сервери у ситуації підвищеного навантаження.

Безпека є пріоритетом: це велике занепокоєння, коли відбувається трансформація високочутливих даних. Веб-програми є більш безпечними, ніж прості веб-сайти, оскільки вони, як правило, розміщені на виділених серверах. Це гарантує безпеку. Досвідчені адміністратори серверів відстежують веб-програми та підтримують їх для забезпечення чудового користувацького досвіду та безпеки.

Використовуйте основну технологію на ваш вибір: компанія, що розробляє веб-програми, може вибрати будь-яку з трьох доступних основних технологій. Перший варіант - вибір рішень на базі Java від Sun Microsystems, які включають такі технології, як JSP і сервлети. У розробників є інша можливість вибрати платформу Microsoft .NET, яка використовує сторінки активних серверів, мови SQL Server та .NET. Також доступні розроблені для розробників платформи з відкритим кодом, такі як PHP та MySQL. Численні плюси веб-розробки PHP роблять його ідеальною платформою для розробки веб-сайтів.

1.7. Недоліки веб додатків

Технологічні досягнення завжди впроваджуються з хорошою метою. Але кожен варіант має своє ставлення до деяких критичних умов. У кожній існуючій технології є як позитивні, так і негативні сторони. Кожна платформа має своє власне середовище, розроблене для розробки веб-додатків.

Накладні витрати та збільшений розмір: помічено, що, порівняно з рідними робочими програмами, веб-програми працюють і функціонують повільно. Інший фактор, який впливає на багато, полягає в тому, що під час розробки веб-додатків

розробник постачає весь веб-браузер із додатком. Отже, в результаті розмір програми збільшується.

Обмеження на стороні клієнта: усередині браузера клієнти не можуть читати файли, друкувати або надсилати електронні листи. Оскільки це розглядається як діра для безпеки веб-додатків.

Розглядаючи як позитивні, так і негативні аспекти, прходимо до висновку, що веб-додаткам слід віддавати перевагу для комерційних цілей. Переваги веб-додатків полягають у тому, що їх легко розробити, орієнтувати на користувача, легко встановлювати, легко обслуговувати та забезпечувати хороший захист.

1.8. Процесс створення веб додатку

Як і традиційні настільні програми, веб-програми мають різний рівень ризику. Персональна домашня сторінка набагато менш ризикована, ніж, наприклад, веб-сайт з біржової торгівлі. Для деяких проектів основними проблемами є безпека, помилки програмного забезпечення тощо. Якщо питання ринку чи технічна складність викликає занепокоєння, документація, планування випробувань, контроль змін, аналіз вимог, архітектурний опис та офіційні практики проектування та будівництва можуть зменшити ризик.

Дослідження, проведене за участю практики веб-інженерії, показало, що розробка веб-додатків має декілька характеристик, на які слід звернути увагу, і серед них: короткий час життєвого циклу розробки; різні бізнес-моделі; мультидисциплінарні команди розвитку; невеликі команди розробників, що працюють над подібними завданнями; бізнес-аналіз та оцінка з кінцевими споживачами; чіткі вимоги та суворе навчання проти вимог; та технічне обслуговування.

Час виходу на ринок, зростання компанії та зменшення вимог - три речі, на які наголошується у веб-бізнесі, збігаються з принципами практики Agile. Деякі гнучкі моделі життєвого циклу: Екстремальне програмування, Scrum.

Веб-програми проходять те саме модульне, інтеграційне та системне тестування, що і традиційні настільні програми. Вона має ті самі цілі, які передбачають:

1) визначення того, що програма працює коректно

2) виявлення помилок, які потребують виправлення.

Проте процес тестування веб-додатків має деякі особливі характеристики, що трохи відрізняється від тесту, що використовується для програмного забезпечення. Сюди входить той факт, що веб-програми, як правило, мають велику кількість інформації, яка може містити помилки, упущення, неправильні мітки, надмірність тощо. Вони також можуть спричиняти декілька шарів веб-додатків та декілька динамічних конфігурацій. Отже, тестування та пошук помилок передбачає більш складний процес, такий як включення операційного аналізу для кожного рівня або конфігурації.

Клієнти веб-додатків сильно різняться, отже, команди можуть провести додаткове тестування, таке як: Безпека, Продуктивність, Навантаження та Стрес, перевірка HTML / CSS, Доступність, Юзабіліті та Крос-браузер.

Багато видів тестів є автоматизованими. На рівні компонентів один з пакетів xUnit може бути корисним інструментом. Або організація може створити власну структуру модульного тестування. На рівні графічного інтерфейсу корисні Watir або iMacros.

У випадку з ASP.NET розробники можуть використовувати Microsoft Visual Studio для написання коду. Але, як і у більшості інших мов програмування, вони також можуть використовувати текстовий редактор (наприклад, Notepad ++). Сервер інтеграції WebORB для .NET можна використовувати для інтеграції служб .NET, даних та засобів масової інформації з будь-яким веб-клієнтом. Він включає інструменти для підвищення продуктивності розробника та API для віддаленого доступу, обміну повідомленнями та управління даними.

Для ColdFusion та відповідних механізмів CFML з відкритим кодом існує кілька інструментів для написання коду. Сюди входять Adobe Dreamweaver CS4, плагін CFEclipse для Eclipse (програмне забезпечення) та Adobe CF Builder. Також можна використовувати будь-який текстовий редактор, наприклад Notepad ++ або TextEdit.

Багато інструментів підтримують мову програмування Java. Найпопулярнішими є Apache Tomcat, GlassFish, JDeveloper та Netbeans, але є численні інші.

Для PHP середовище розробки Zend надає численні інструменти налагодження та багатий набір функцій, що спрощує розробку PHP. Сервер інтеграції WebORB для PHP можна використовувати для інтеграції класів та даних PHP з будь-яким веб-клієнтом. Він включає в себе інструменти для продуктивності розробників та API для віддаленого доступу, обміну повідомленнями та управління даними. Такі інструменти, як Hammerkit, абстрагують PHP у середовище візуального програмування та використовують програмні методи на основі компонентів для пришвидшення розробки.

Інші інструменти включають різні програми веб-розробки, браузери та FTP-клієнти.

1.9. Модель MVC як стандарт веб додатку

Сучасні сайти інтерактивні і динамічні, тобто вони реагують на дії користувача, обробляють його запити і видають результат. Так працюють онлайн-сервіси, наприклад, інтернет-банкінг або онлайн-кінотеатр. Для створення інтерактивних і динамічних сайтів зазвичай використовується архітектурний патерн MVC. У цій статті простими словами пояснюється суть цієї моделі.

Статична сторінка на HTML не вміє реагувати на дії користувача. Для двосторонньої взаємодії потрібні динамічні веб-сторінки. MVC - ключ до розуміння розробки динамічних веб-додатків, тому розробнику потрібно знати цю модель.

MVC розшифровується як модель-вигляд-контролер (від англ. Model-view-controller). Це спосіб організації коду, який передбачає виділення блоків, що відповідають за вирішення різних завдань. Один блок відповідає за дані додатки, інший відповідає за зовнішній вигляд, а третій контролює роботу додатка. Компоненти MVC:

Модель - це компонент відповідає за дані, а також визначає структуру програми. Наприклад, якщо ви створюєте To-Do додаток, код компонента model визначатиме список завдань і окремі завдання.

Вигляд - це компонент відповідає за взаємодію з користувачем. Тобто код компонента view визначає зовнішній вигляд програми і способи його використання.

Контролер - цей компонент відповідає за зв'язок між model і view. Код компонента controller визначає, як сайт реагує на дії користувача. По суті, це мозок MVC-додатки.

Концепція MVC була описана Трюгве Реенскаугом в 1978 році, який працював в науково-дослідному центрі «Xerox PARC» над мовою програмування «Smalltalk». Пізніше, Стів Бурбек реалізував шаблон в Smalltalk-80.

Остаточна версія концепції MVC була опублікована лише в 1988 році в журналі Technology Object.

Згодом шаблон проектування став еволюціонувати. Наприклад, була представлена ієрархічна версія HMVC; MVA, MVVM.

Подальший виток популярності привнесло розвиток фреймворків, орієнтованих на швидку розгортку, на мовах Python, PHP і Ruby: Django, Laravel і Ruby on Rails відповідно. На момент 2017 року, фреймворки з MVC зайняли помітні позиції по відношенню до решти фреймворками без цього шаблону .

З розвитком об'єктно-орієнтованого програмування і поняття про шаблони проектування - був створений ряд модифікацій концепції MVC, які при реалізації у різних авторів можуть відрізнятися від оригінальної. Так, наприклад, Еріан Верми в 2004 році описав приклад узагальненого MVC .

У передмові до дисертації «Naked objects» Річарда Поусона (Richard Pawson), - Трюгве Реенскауг згадує свою неопубліковану найбільш ранню версію MVC, згідно з якою:

- Модель ставилася до «розуму» користувача;
- Під поданням мався на увазі редактор, що дозволяє користувачеві переглядати і оновлювати інформацію;
- Контролер був інструментом для зв'язування уявлень воєдино і застосовувався користувачем для вирішення його завдань.

Основна мета застосування цієї концепції полягає в відділенні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого поділу підвищується

можливість повторного використання коду. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах і / або з різних точок зору. Зокрема, виконуються наступні завдання:

- До однієї моделі можна приєднати кілька видів, при цьому не зачіпаючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у вигляді електронної таблиці, гістограми і кругової діаграми;
- Не торкаючись реалізацію видів, можна змінити реакції на дії користувача (натискання мишею на кнопці, введення даних) - для цього досить використовувати інший контролер;
- Ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можливо добитися того, що програмісти, які займаються розробкою бізнес-логіки (моделі), взагалі не будуть обізнані про те, яке уявлення буде використовуватися.

РОЗДІЛ 2. ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ ВЕБ ДОДАТКА

2.1. HTML та CSS

Hypertext Markup Language (HTML) – це мова розмітки гіпертекстових документів. Стандартна мова розмітки для створення веб-сторінок і веб-додатків. З Cascading Style Sheets (CSS) і JavaScript, вона утворює тріаду основних технологій для World Wide Web.

Веб-браузери отримують HTML-документи з веб-сервера або з локальної пам'яті і передають документи в мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і спочатку включені сигнали для зовнішнього вигляду документа.

Елементи HTML є будівельними блоками сторінок HTML. За допомогою конструкцій HTML, зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані у візуалізовану сторінку. HTML надає засоби для створення структурованих документів, позначаючи структурну семантику тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML окреслені тегами, написаними з використанням кутових дужок. Теги, такі як і безпосередньо вводять вміст на сторінку. Інші теги надають інформацію про текст документа і можуть включати інші теги як під-елементи. Браузери не показують теги HTML, але використовують їх для інтерпретації вмісту сторінки.

Розмітка HTML складається з декількох ключових компонентів, включаючи ті, що називаються тегами (та їх атрибутами), типи даних на основі символів, посилання на символи та посилання на сутності. Теги HTML найчастіше приходять в парах, таких як <h1> та </h1>, хоча деякі представляють порожні елементи, тому вони не є парними, наприклад <img>. Перший тег у такій парі - це початковий тег, а другий - кінцевий тег (їх ще називають тегами відкриття та закриття тегів).

Іншим важливим компонентом є декларація типу документа HTML, яка ініціює рендеринг стандартного режиму.

Документи HTML мають на увазі структуру вкладених елементів HTML. Вони позначені в документі тегами HTML, укладеними в кутові дужки таким чином: .

У простому загальному випадку обсяг елемента позначається парою тегів: "початковий тег" і "кінцевий тег" . Текстовий вміст елемента, якщо такий є, розміщується між цими тегами.

Теги можуть також містити додаткову розмітку тегів між початком та кінцем, включаючи суміш тегів та тексту. Це вказує на подальші (вкладені) елементи як дочірні елементи батьківського елемента.

Початковий тег може також містити атрибути всередині тегу. Вони вказують на іншу інформацію, таку як ідентифікатори розділів у документі, ідентифікатори, що використовуються для прив'язки інформації про стиль до презентації документа, а для деяких тегів, таких як <img>, що використовується для вбудування зображень, посилання на ресурс зображення в такий формат: <img src = "example.com/example.jpg">

Деякі елементи, такі як розрив рядка
, або
 не дозволяють вбудовувати вміст, ні текст, ні інші теги. Для них потрібен лише один порожній тег (подібний до початкового тегу) і не використовується кінцевий тег.

Багато тегів, зокрема закриваючий кінцевий тег для дуже часто використовуваного елемента абзацу , є необов’язковими. Браузер HTML або інший агент може зробити висновок про закриття кінця елемента з контексту та структурних правил, визначених стандартом HTML. Ці правила є складними і не широко зрозумілі більшості кодерів HTML.

Отже, загальною формою елемента HTML є: <tag attribute1 = "value1" attribute2 = "value2"> " content " </tag>. Деякі елементи HTML визначаються як порожні елементи і мають вигляд <tag attribute1 = "value1" attribute2 = "value2">. Порожні елементи не можуть містити жодного вмісту, наприклад, тег
 або вбудований тег <img>. Ім'я елемента HTML - це ім'я, яке використовується в тегах. Зверніть увагу, що перед ім'ям кінцевого тегу перебуває коса риса, /, і що в порожніх

елементах кінцевий тег не є ні обов'язковим, ні дозволеним. Якщо атрибути не згадані, у кожному випадку використовуються значення за замовчуванням.

HTML може вбудовувати програми, написані на мові сценаріїв, наприклад JavaScript, що впливає на поведінку та вміст веб-сторінок. Включення CSS визначає вигляд і компонування вмісту. World Wide Web Consortium (W3C), які супроводжують як HTML і CSS стандартів, заохочує використання CSS над явним презентаційним HTML з 1997 року.

HTML впроваджує засоби для:

- створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації із Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

CSS – спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів. Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3.

Профілі – сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці веб-сторінок. Головна перевага блочної верстки – розділення змісту сторінки (даних) та їхньої візуальної презентації.

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних

переваг – можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS).

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою, у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах.

2.2. JavaScript

JavaScript – динамічна, об'єктно-орієнтована, прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

Історія створення мови програування починається з веб-браузеру Mosaic, який був випущений в 1993 році. Будучи першим браузером з графічним інтерфейсом користувача, доступним для нетехнічних людей, він зіграв помітну роль у швидкому зростанні зароджуваної Всесвітньої павутини. Тоді провідні розробники Mosaic заснували корпорацію Netscape, яка випустила більш досконалий браузер Netscape Navigator в 1994 році. Навігатор швидко став найбільш використовуваним браузером.

Протягом цих років Інтернету веб-сторінки могли бути лише статичними, не маючи можливості динамічної поведінки після завантаження сторінки в браузер. На зростаючій сцені веб-розробки було бажання зняти це обмеження, тому в 1995 році Netscape вирішив додати мову сценаріїв до Навігатора. Вони досягли двох шляхів досягнення цього: співпраця з Sun Microsystems для вбудування мови програмування Java, а також найняття Брендана Ейха для вбудування мови Scheme.

Незабаром керівництво Netscape вирішило, що найкращим варіантом було б для Eich розробити нову мову, із синтаксисом, подібним до Java, і менш схожим на Scheme або інші існуючі мови сценаріїв. Незважаючи на те, що нова мова та її інтерпретатор були офіційно названі LiveScript, коли вперше були поставлені як частина випуску Navigator у вересні 1995 року, назва була змінена на JavaScript через три місяці.

Вибір імені JavaScript викликав плутанину, іноді створюючи враження, що це виділення Java. Оскільки Java на той час була гарячою новою мовою програмування, Netscape це характеризує як маркетинговий трюк для створення власного кешу нової мови.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам, наприклад функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання (closures) – що додає мові додаткову гнучкість.

JavaScript має С-подібний синтаксис, але в порівнянні з мовою С має такі корінні відмінності.

Мова JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- створення односторінкових веб-додатків (React.js, AngularJS, Vue.js);
- програмування на стороні сервера (Node.js);
- стаціонарних додатків (Electron, NW.js);
- мобільних додатків (React.js Native, Cordova);

- сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів тощо.

JavaScript, наразі, є однією з найпопулярніших мов програмування. Але спочатку багато професійних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-любителів.

Поява AJAX змінила ситуацію та повернула увагу професійної спільноти до мови, а подальші модифікації мови за стандартами ES2015 та ES2017 внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

Сучасний JavaScript - це «безпечний» мову програмування. Він не надає низькорівневий доступ до пам'яті або процесору, тому що спочатку був створений для браузерів, які не потребують цього.

Можливості JavaScript сильно залежать від оточення, в якому він працює. Наприклад, Node.JS підтримує функції читання / запису довільних файлів, виконання мережевих запитів і т.д.

У браузері для JavaScript є все, що пов'язано з маніпулюванням веб-сторінками, взаємодією з користувачем і веб-сервером.

Наприклад, в браузері JavaScript може:

- Додавати новий HTML-код на сторінку, змінювати існуючий вміст, модифікувати стилі.
- Реагувати на дії користувача, клацання миші, перемістити вказівник, натискання клавіш.
- Відправляти мережеві запити на віддалені сервера, завантажувати і завантажувати файли (технології AJAX і COMET).
- Отримувати і встановлювати куки, задавати питання відвідувачеві, показувати повідомлення.
- Запам'ятовувати дані на стороні клієнта («local storage»).

Можливості JavaScript в браузері обмежені заради безпеки користувача. Мета полягає в запобіганні доступу недобросовісної веб-сторінки до особистої інформації або нанесення шкоди даними користувача.

Приклади таких обмежень включають в себе:

JavaScript на веб-сторінці не може читати / записувати довільні файли на жорсткому диску, копіювати їх або запускати програми. Він не має прямого доступу до системних функцій ОС.

Сучасні браузери дозволяють йому працювати з файлами, але з обмеженим доступом, і надають його, тільки якщо користувач виконує певні дії, такі як «перетягування» файлу в вікно браузера або його вибір за допомогою тега <input>.

Існують способи взаємодії з камерою / мікрофоном і іншими пристроями, але вони вимагають явного дозволу користувача. Таким чином, сторінка з підтримкою JavaScript не може непомітно включити веб-камеру, спостерігати за тим, що відбувається і відправляти інформацію в ФСБ.

Різні вікна / вкладки не знають один про одного. Іноді одне вікно, використовуючи JavaScript, відкриває інше вікно. Але навіть в цьому випадку JavaScript з однієї сторінки не має доступу до іншої, якщо вони прийшли з різних сайтів (з іншого домену, протоколу або порту).

Це називається «Політика однакового джерела» (Same Origin Policy). Щоб обійти це обмеження, обидві сторінки повинні погодитися з цим і містити JavaScript-код, який спеціальним чином обмінюється даними.

Це обмеження необхідно, знову ж таки, для безпеки користувача. Сторінка https://anysite.com, яку відкрив користувач, не повинна мати доступ до іншої вкладці браузера з URL https://gmail.com і красти інформацію звідти.

JavaScript може легко взаємодіяти з сервером, з якого прийшла поточна сторінка. Але його здатність отримувати дані з інших сайтів / доменів обмежена. Хоча це можливо в принципі, для чого потрібно явне згоду (виражене в заголовках HTTP) з віддаленої стороною. Знову ж таки, це обмеження безпеки.

Подібні обмеження не діють, якщо JavaScript використовується поза браузера, наприклад - на сервері. Сучасні браузери надають плагіни / розширення, за допомогою яких можна запитувати додаткові дозволи.

Що робить JavaScript особливим? Як мінімум, три сильні сторони JavaScript:

- Повна інтеграція з HTML / CSS.
- Прості речі робляться просто.
- Підтримується всіма основними браузерами і включений за замовчуванням.

JavaScript - це єдина браузерна технологія, що поєднує в собі всі ці три речі. Ось що робить JavaScript особливим. Ось чому це найпоширеніший інструмент для створення інтерфейсів в браузері. Хоча, звичайно, JavaScript дозволяє робити програми не тільки в браузерах, але і на сервері, на мобільних пристроях і т.п.

Синтаксис JavaScript підходить не під всі потреби. Різні люди хочуть мати різні можливості. Це природно, тому що проекти різні і вимоги до них теж різні.

Так, останнім часом з'явилося багато нових мов, які транспіліруются (конвертуються) в JavaScript, перш ніж запустяться в браузері.

Сучасні інструменти роблять транспіляцію дуже швидкою і прозорою, фактично дозволяючи розробникам писати код на іншій мові, автоматично перетворюючи його в JavaScript «під капотом».

Приклади таких мов:

- CoffeeScript додає «синтаксичний цукор» для JavaScript. Він вводить більш короткий синтаксис, який дозволяє писати чистий і лаконічний код. Зазвичай таке подобається Ruby-програмістам.
- TypeScript концентрується на додаванні «суворої типізації» для спрощення розробки та підтримки великих і складних систем. Розроблено Microsoft.
- Flow теж додає типізацію, але інакше. Розроблено Facebook.
- Dart стоїть осібно, бо має власний движок, що працює поза браузера (наприклад, в мобільних додатках). Спочатку був запропонований Google, як заміна

JavaScript, але на даний момент необхідна його транспіляція для запуску так само, як для перерахованих вище мов.

- Brython транспілірует Python в JavaScript, що дозволяє писати програми на чистому Python без JavaScript.

Є й інші. Але навіть якщо ми використовуємо один з цих мов, ми повинні знати JavaScript, щоб дійсно розуміти, що ми робимо.

Отже, JavaScript спочатку створювався тільки для браузера, але зараз використовується на багатьох інших платформах. Сьогодні JavaScript займає унікальну позицію в якості самого поширеного мови для браузера, який володіє повною інтеграцією з HTML / CSS. Багато мов можуть бути «транспіліровани» в JavaScript для надання додаткових функцій. Рекомендується хоча б коротко розглянути їх після освоєння JavaScript.

2.3. AJAX

AJAX (від англ. Asynchronous Javascript and XML - «асинхронний JavaScript і XML») - підхід до побудови призначених для користувача інтерфейсів веб-додатків, що полягає в «фоновому» обміні даними браузера з веб-сервером. В результаті при оновленні даних веб-сторінка не перезавантажується повністю, і веб-додатки стають швидше і зручніше. По-русски іноді вимовляється транслитом як «аякс». У абревіатури AJAX немає усталеного аналога на кирилиці.

У класичній моделі веб-додатки:

- Користувач заходить на веб-сторінку і натискає на який-небудь її елемент.
- Браузер формує і відправляє запит серверу.
- У відповідь сервер генерує абсолютно нову веб-сторінку і відправляє її браузеру і т. п., Після чого браузер повністю перезавантажує всю сторінку.

При використанні AJAX:

- Користувач заходить на веб-сторінку і натискає на який-небудь її елемент.
- Скрипт (на мові JavaScript) визначає, яка інформація необхідна для оновлення сторінки.

- Браузер відправляє відповідний запит на сервер.
- Сервер повертає лише ту частину документа, на яку прийшов запит.
- Скрипт вносить зміни з урахуванням отриманої інформації (без повного перезавантаження сторінки).

AJAX - не самостійна технологія, а концепція використання декількох суміжних технологій. AJAX базується на двох основних принципах:

використання технології динамічного звернення до сервера «на льоту», без перезавантаження всієї сторінки повністю, наприклад з використанням XMLHttpRequest (основний об'єкт);

- через динамічне створіння дочірніх фреймів;
- через динамічне створіння тега <script>.
- через динамічне створіння тега <img>, як це реалізовано в Google Analytics.
- використання DHTML для динамічної зміни змісту сторінки;

Дії з інтерфейсом перетворюються в операції з елементами DOM (англ. Document Object Model), за допомогою яких обробляються дані, доступні користувачеві, в результаті чого уявлення їх змінюється. Тут же проводиться обробка переміщень і клацань мишею, а також натискань клавіш. Каскадні таблиці стилів, або CSS (англ. Cascading Style Sheets), забезпечують узгоджений зовнішній вигляд елементів програми та спрощують звернення до DOM-об'єктів. Об'єкт XMLHttpRequest (або подібні механізми) використовується для асинхронного взаємодії з сервером, обробки запитів користувача і завантаження в процесі роботи необхідних даних.

Три з цих чотирьох технологій - CSS, DOM і JavaScript - складають DHTML (англ. Dynamic HTML). На думку деяких фахівців (книг), кошти DHTML, що з'явилися в 1997 році, подавали великі надії, але так і не виправдали їх.

Як формат передачі даних можуть використовуватися фрагменти простого тексту, HTML-коду, JSON або XML.

2.4. ReactJS

React.js – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових додатків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React.js дозволяє розробникам створювати великі веб-додатки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Мета фреймворку полягає в тому, щоб бути створити швидкий, простий, масштабований веб-додаток. React.js обробляє тільки користувацький інтерфейс у додатках.

Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Як бібліотеку для побудови інтерфейсу користувача, React.js найчастіше використовують разом з іншими бібліотеками, такими як Redux.

Ще однією помітною особливістю є використання віртуальної об'єктної моделі документа або віртуальної DOM. React створює кеш-структуру даних у пам’яті, обчислює отримані різниці, а потім ефективно оновлює відображуваний DOM браузера.. Цей процес називається примиренням. Це дозволяє програмісту писати код так, ніби вся сторінка відображається при кожній зміні, тоді як бібліотеки React відображають лише підкомпоненти, які насправді змінюються. Цей вибірковий візуалізація забезпечує значне підвищення продуктивності. Це економить зусилля з перерахунку стилю CSS, макета сторінки та рендеринга для всієї сторінки.

Схему роботи оновлення віртуального DOM зображено на рисунку 2.1. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

Компоненти React.js зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React.js. Розробники можуть так само писати на чистому JavaScript.

JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP. JSX надає ряд атрибутів елементів, призначених для відображення тих, що надаються у форматі HTML. Користувацькі атрибути також можуть бути передані компоненту. Всі атрибути будуть отримані компонентом як реквізит.

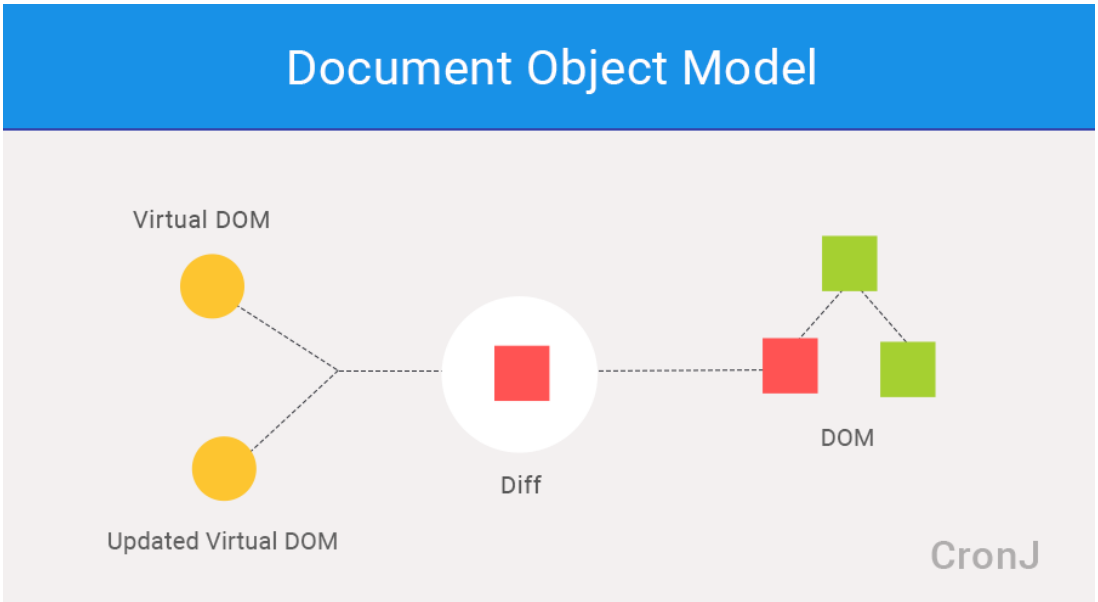

Рис. 2.1 Робота віртуального DOM в React.js

React.js використовують не лише для рендерингу HTML в браузері. Наприклад, Facebook має динамічні графіки які рендеряться в теги <canvas>, Netflix та PayPal використовують ізоморфне завантаження для рендерингу ідентичного HTML на сервері та клієнті.

Методи життєвого циклу – це різні методи, які вбудовуються за допомогою React.js. Послідовність методів життєвого циклу компонентів React.js зображено на рисунку 2.2. Вони дозволяють розробнику обробляти дані в різних точках життєвого циклу програми React.js.

Основні методи життєвого циклу React.js:

- shouldComponentUpdate – це метод життєвого циклу, який каже Javascript оновити компонент, використовуючи логічні змінні;
- componentWillMount – це метод життєвого циклу, який каже Javascript налаштувати певні дані перед монтуванням компонентів (вставлення у віртуальний DOM);

- componentDidMount – це метод життєвого циклу, подібний до компонента WillMount, за винятком того, що він працює після методу render, і може використовуватися для додавання JSON-даних, а також для визначення властивостей та станів;
- render є найважливішим методом життєвого циклу, необхідним у будь-якому компоненті. Метод render – це те, що з'єднується з JSX і відображати власний JSX.

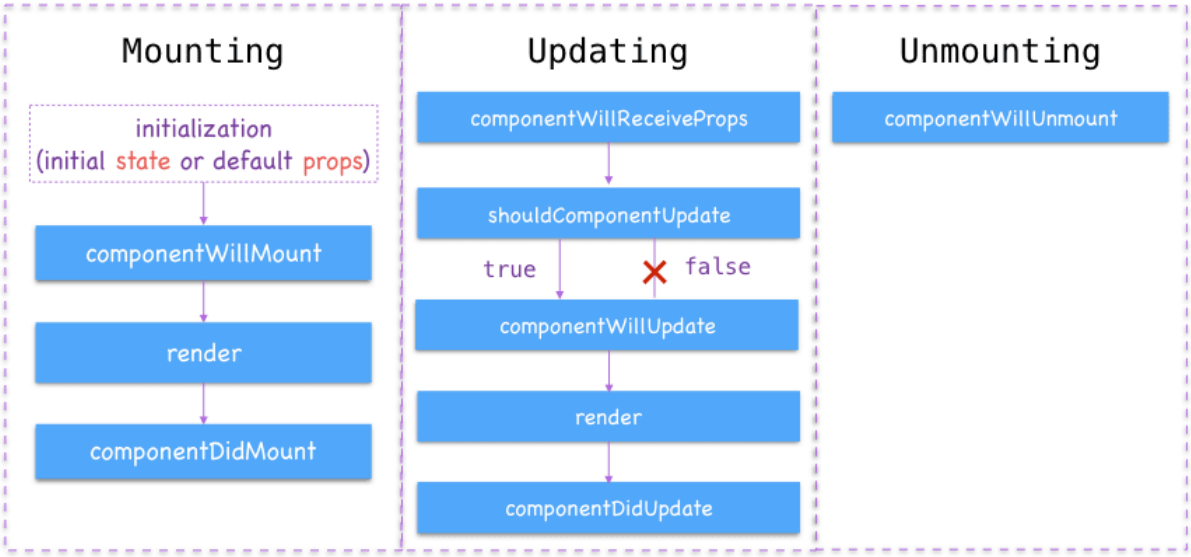

Рис. 2.2 Методи життєвого циклу в React.js

React.js не намагається надати повну «схему додатків». Він безпосередньо спрямований на побудову користувацьких інтерфейсів, і тому не включає в себе безліч інструментів, які деякі розробники вважають необхідними для створення програми. Це дозволяє вибрати будь-які бібліотеки, які розробник вважає за краще виконувати, щоб виконати певних завдань, таких як здійснення доступу до мережі або локальне зберігання даних.

Бібліотеку React.js створено Джорданом Волком (Jordan Walke), програмістом з Facebook. Автор працював над проектом під впливом XHP, фреймворку HTML для PHP. 2011-го року реліз з'явився у новинах Facebook, за рік – у блозі Instagram.

Також фреймворк був представлений як проект з відкритим початковим кодом на конференції розробників JSConf US, що проходила у Сполучених Штатах у

травні 2013 року. На конференції React.js Conf, влаштовану Фейсбуком у березні 2015-го, проект було представлено як відкрите програмне забезпечення.

Для підтримки концепції React про односпрямований потік даних (який може бути протиставлений двонаправленому потоку AngularJS), архітектура Flux представляє альтернативу популярній архітектурі модель-вигляд-контролер. Flux має дії, які надсилаються через центральний диспетчер до магазину, а зміни в магазині передаються назад до подання. При використанні з React це розповсюдження здійснюється завдяки властивостям компонентів. Потік можна вважати варіантом моделі спостерігача.

Компонент React під архітектурою Flux не повинен безпосередньо модифікувати будь-який переданий йому реквізит, але повинен передавати функції зворотного виклику, які створюють дії, які відправляються диспетчером для модифікації сховища. Дія - це об'єкт, відповідальність якого полягає в описі того, що відбулося: наприклад, дія, що описує одного користувача, "що слідує" за іншим, може містити ідентифікатор користувача, цільовий ідентифікатор користувача та тип USER_FOLLOWED_ANOTHER_USER. Магазини, які можна сприймати як моделі, можуть змінити себе у відповідь на дії, отримані від диспетчера.

Ця закономірність іноді виражається як "властивості стікають вниз, дії течуть вгору". З моменту його створення було створено багато реалізацій Flux, мабуть, найвідомішим є Redux, який має єдиний магазин, який часто називають єдиним джерелом істини.

Redux – відкрита JavaScript бібліотека призначена для управління станом програми. Найчастіше використовується разом з React.js або Angular для побудови інтерфейсів користувача.

Бібліотека являє собою контейнер станів для додатків JavaScript. Він допомагає розробникам оптимізувати код програми. Крім того, він забезпечує вдосконалення досвіду розробника, наприклад, редагування живого коду в поєднанні з відладчиком, що працює під час роботи.

Redux зберігає стан всього додатку в дереві об'єктів в одному сховищі. Одне дерево станів полегшує налагодження або перевірку програми. Також дерево станів

дозволяє зберігати стан додатку в процесі розробки, для прискорення циклу розробки.

Єдиний спосіб змінити стан – це виокремити дію, об'єкт, що описує те, що сталося. Це гарантує, що ні перегляди, ні зворотні виклики мережі ніколи не будуть змінювати стан. Натомість вони виражають тільки намір це зробити. Всі зміни – централізовані і відбуваються одна за іншою у чіткій послідовності.

Оскільки дії є простими об'єктами, вони можуть бути зареєстровані, серіалізовані, збережені та пізніше відтворені в тій же послідовності для налагодження або тестування.

Ред’юсери – це лише чисті функції, які приймають попередній стан і дію, і повертають наступний стан. В процесі розробки ред’юсери можуть бути розділені на дрібніші ред’юсери, які управляють певними частинами дерева станів. Оскільки ред’юсери – це лише функції, то є можливо контролювати порядок їх надсилання, передавати додаткові дані або навіть створювати повторювані ред’юсери для звичайних завдань.

2.5. Node.js

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих додатків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Якщо раніше Javascript застосовувався для обробки даних в браузері на сторонні користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їх виконання.

Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників. Для керування модулями використовується пакетний менеджер npm (node package manager).

Node.js має наступні властивості:

- асинхронна однопотокова модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8.

Платформа Node.js призначена для виконання високопродуктивних мережевих додатків, написаних мовою програмування JavaScript. Платформа окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських та серверних програм. В платформі використовується розроблений компанією Google рушій V8.

Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback).

2.6. REST

REST (Representational State Transfer) – це архітектурний стиль для розподілених гіпертекстових систем або підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Базову схему роботи REST API зображено на рисунку 2.3.

Стиль REST описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP .

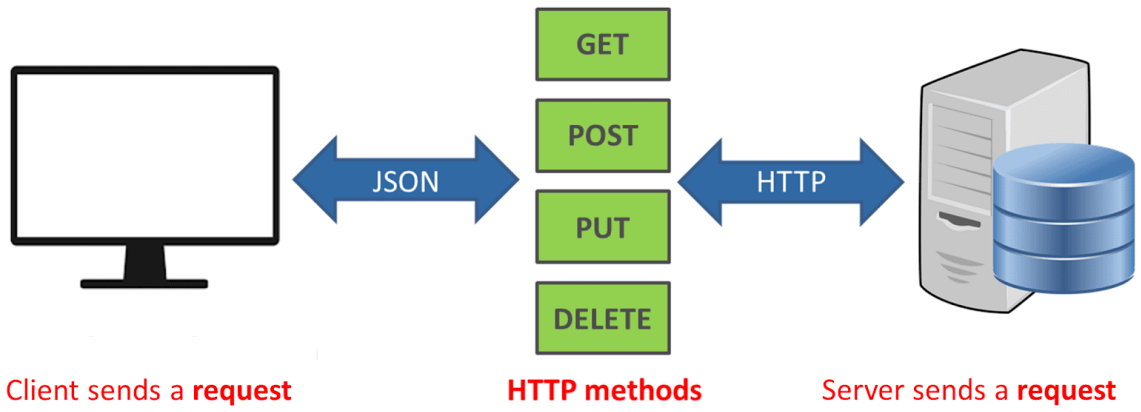

Рис. 2.3 Схема роботи REST API

В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Рой Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

За правилами архітектурного стилю, всі дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не

повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами .

Антиподом REST є підхід, заснований на виклику віддалених процедур RPC. Підхід RPC дозволяє використовувати невелику кількість мережевих ресурсів з великою кількістю методів і складним протоколом.

При підході REST кількість методів і складність протоколу суворо обмежені, що призводить до того, що кількість окремих ресурсів має бути достатньо великою .

REST, як і кожен архітектурний стиль відповідає ряду архітектурних обмежень. Це гібридний стиль який успадковує обмеження з інших архітектурних стилів.

Перша архітектура від якої він успадковує обмеження – це клієнт-серверна архітектура. Її обмеження вимагає розділення відповідальності між компонентами, які займаються зберіганням та оновленням даних (сервером), і тими компонентами, які займаються відображенням даних на інтерфейсі користувача та реагування на дії з цим інтерфейсом (клієнтом). Таке розділення дозволяє компонентам еволюціонувати незалежно .

Наступним обмеженням є те, що взаємодії між сервером та клієнтом не мають стану, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту.

Відсутність стану не означає що стану немає. Відсутність стану означає, що сервер не знає про стан клієнта. Коли клієнт, наприклад, запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта. Клієнт може залишити сторінку відкритою протягом кількох років, перш ніж натиснути посилання, і тоді сервер відповість на інший запит. Тим часом сервер може відповідати на запити інших клієнтів, або нічого не робити – для клієнта це не має значення .

Таким чином, наприклад дані про стан сесії (користувача, який аутентифікувався) зберігаються на клієнті, і передаються з кожним запитом. Це покращує масштабовність, бо сервер після закінчення обробки запиту може

звільнити всі ресурси, задіяні для цієї операції, без жодного ризику втратити цінну інформацію.

Також спрощується моніторинг дебаґінгу, бо для того аби розібратись, що відбувається в певному запиті, досить подивитись лише на той один запит. Збільшується надійність, бо помилка в одному запиті не зачіпає інші.

Мінусом цього обмеження є те, що знижується продуктивність через те, що в кожен запит тепер доводиться додавати дані сесії з клієнта. Також збереження стану на різних клієнтах важче підтримувати, бо реалізації клієнтів можуть відрізнятись, тоді як середовище сервера повністю під контролем розробника .

Додатковим обмеженням стилю REST є те, що системи, написані в цьому стилі, повинні підтримувати кешування, тобто дані, які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність, уникаючи зайвих запитів, але також зменшує надійність системи, через те, що дані в кеші можуть бути застарілими або неправильними.

Рання архітектура веб, створена Тімом Бернерсом-Лі відповідала цим трьом обмеженням – клієнт-сервер без стану з підтримкою кешування. Проте стиль REST додає ще додаткові обмеження .

Всі компоненти в архітектурі REST підтримують однорідний інтерфейс. Це зменшує зв'язність між компонентами і сервісами які вони надають і дозволяє нескладно змінювати компоненти при потребі. Це досягається кількома точнішими обмеженнями :

- ідентифікація ресурсів;
- маніпуляція ресурсами через представлення;
- самоописові повідомлення;
- гіпермедіа як рушій стану додатку.

Наступним обмеженням для REST є поділ на шари абстракції. Кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмеження знання системи одним шаром зменшує складність компонентів .

Останнім архітектурним обмеженням в REST є те що клієнти повинні дозволяти розширювати свою функціональність дозволяючи завантаження додаткового коду в формі аплетів чи скриптів. Щоправда, це необов'язкове обмеження, і якщо воно не дає переваг для конкретного застосування, то його не обов'язково реалізовувати. Наприклад, дозвіл завантаження стороннього коду може бути не бажаним з точки зору безпеки .

Компоненти REST системи спілкуються, передаючи один одному представлення ресурсу в форматі, що обирається з оновлюваного набору стандартних форматів даних. Формат обирається динамічно відповідно до бажань компонента-клієнта і можливостей сервера. Чи представлення має той самий формат, що й сам ресурс, чи є результатом якогось перетворення – це деталь реалізації, яка ховається за інтерфейсом .

Ресурс – це ключовий елемент даних в REST. Ресурсом може бути що завгодно що можна назвати: якийсь документ (наприклад зображення), динамічне значення (наприклад погода у Львові), щось з реального світу (наприклад працівник компанії). Але якщо точніше, то ресурс – це функція приналежності, що відображає моменти в часі на множину однотипних сутностей чи значень. Множина може бути порожньою, тобто REST дозволяє посилання на якийсь об'єкт якого ще не існує .

Ресурс може бути динамічним, наприклад ресурс «стаття про REST у вікіпедії» час від часу оновлює свій вміст, а може бути статичним, і після появи ніколи не змінювати свого значення, наприклад «перша версія статті про REST у вікіпедії». У REST такі два ресурси вважаються різними, хоча в певний момент часу вони можуть вказувати на одну й ту ж сутність. Єдине що важливо – семантика відображення імені ресурсу на його вміст .

Для того, щоб посилатись на ресурси, використовуються ідентифікатори ресурсів. Компонент, який надав ресурсу ідентифікатор і дозволяє звертатись до нього за цим ідентифікатором, відповідає за збереження функції приналежності незмінною. Якість ідентифікатора залежить від якості компонента, який цей ідентифікатор надає, тому деякі ідентифікатори стають «мертвими посиланнями»,

коли інформацію переміщують або знищують. Приклади ідентифікаторів ресурсу: URL, URN .

Представлення – це послідовність байтів та метадані представлення, для того щоб описати ці байти. Часто, представлення називають документом, файлом, повідомленням HTTP тощо. Приклади представлення: фотографія JPEG, документ HTML. Приклад метаданих представлення – тип медіа, час останньої зміни .

Метадані також можуть бути не лише в представлення ресурсу, а й в самого ресурсу. Прикладами метаданих ресурсу є посилання на джерело, <link rel="alternates" />, заголовок HTTP.

Контрольні дані в представленні описують ціль повідомлення між компонентами, наприклад прохання про дію (створити, змінити видалити ресурс), або значення відповіді (наприклад поточний стан ресурсу, чи значення якогось іншого ресурсу, наприклад опис помилки).

Також, контрольні дані, включені в запити чи відповіді, можуть керувати поведінкою кеша. Прикладом таких контрольних даних можуть бути заголовки HTTP if-modified-since та cache-control .

Формат даних представлення називають типом меді. Одні типи медіа краще підходять для автоматичної обробки, інші – для того щоб бути показаними користувачу. Композитні типи медіа можуть використовуватись для того, аби поєднати кілька видів представлення в одному .

Від формату даних дуже залежить латентність додатку, яку сприймає користувач. Наприклад, браузер може почати показувати сторінку ще до того, як завантажиться ввесь HTML, це збільшує видиму швидкість роботи.

Конектори надають інтерфейс для комунікації компонентів, приховуючи реалізації ресурсів та механізм комунікації. Конектори подібні на віддалений виклик процедур, але з певними нюансами щодо передачі параметрів та результату виклику. Параметри складаються з ідентифікатора ресурсу, контрольних даних та необов'язково, представлення. Результат – з контрольних даних відповіді і представлення .

Можна абстрагуватись і вважати такий виклик синхронним, але насправді передача даних відбувається потоково, тому обробку даних можна починати ще до того як отримані всі дані, таким чином зменшуючи латентність.

Двома найважливішими типами конекторів є клієнт і сервер. Відмінністю між ними є те що клієнт ініціює запит, в той час як сервер очікує запитів і відповідає на них даючи доступ до своїх сервісів. Компонент може містити одночасно як серверні так і клієнтські конектори.

Додатковим типом конектора є кеш. Кеш може бути як клієнтським, для уникнення зайвих запитів, так і серверним – для уникнення зайвого обчислення відповіді на запит. Тому що інтерфейс однорідний, кеш легко може дізнатись чи запит можна кешувати. За замовчуванням, відповідь на запит отримання ресурсу можна кешувати, а запити зміни ресурсу – ні. Проте ці замовчування можна перевантажити контрольними даними .

Резолвер – це ще один тип конектора, який перетворює ідентифікатори ресурсів в інформацію про мережеві адреси необхідну компонентам щоб отримати цей ресурс. Наприклад в URI міститься доменне ім'я, і для доступу до відповідного домена, потрібно дізнатись в DNS-сервера адресу. Тому в цьому випадку система DNS грає роль резолвера. Використання кількох резолверів може збільшити життєздатність ідентифікатора ресурсу, через те що він не вказує напряму на фізичне розташування ресурсу яке може змінитись .

Останньою формою конектора є тунель, який просто проводить запити через межу системи, наприклад, через фаєрвол. Причиною, через яку тунелі включено до архітектури REST, а не закрито абстракцією мережі, є те, що певні компоненти можуть перетворюватись на тунелі за запитом. Наприклад тунель HTTP активується при отриманні запиту з методом CONNECT .

Хоча ресурс може бути яким завгодно, дії, які можна виконувати над ресурсом, визначаються повідомленнями, які визначено стандартним протоколом. В системі WWW цей протокол – HTTP, але існують REST-архітектури на основі й інших протоколів. Стандарт HTTP визначає вісім типів основних повідомлень .

Найчастіше використовують 4 з них :

- GET – отримати представлення ресурсу;
- DELETE – знищити ресурс;
- POST – створити новий ресурс на місці даного використавши передане представлення;
- PUT – замінити стан поточного ресурсу станом, що описується переданим представленням.

Наступні типи повідомлень використовуються, щоб дослідити API:

- HEAD – отримати заголовки, які б відсилались разом з представленням цього ресурсу, але не саме представлення;
- OPTIONS – визначити список методів, на які цей ресурс відповідає.

Інші два методи, CONNECT та TRACE, використовуються лише для HTTP-проксі.

Існує також дев'ятий метод, щоправда, описаний не в HTTP, а в додатку RFC 5789:

- PATCH – змінити лише частину цього ресурсу на основі даного представлення. Якщо якась частина ресурсу не згадується в переданому представленні – не чіпати її. Це знижує кількість інформації, яку потрібно передавати.

Ще два методи описуються в пропозиції до стандарту «Internet-Draft snell-link-method»:

- LINK – прив'язати певний ресурс до цього;
- UNLINK – відв'язати ресурс від цього.

Методи GET, PUT та DELETE – ідемпотентні, що означає, що незалежно від того, скільки разів ви виконаєте операцію, яку вони просять, – ви отримаєте той самий результат. Звісно, спершу DELETE поверне 204 No Content, а потім 404 Not Found, але ресурсу не буде, що після одного видалення, що після десяти. Ідемпотентність є дуже важливою в мережі, де ви не знаєте, чи досяг запит успіху, і, не отримавши відповіді, надсилаєте його ще раз. POST – не ідемпотентний, тобто, відправивши POST на створення повідомлення кілька разів, ви отримаєте кілька повідомлень .

Для отримання інформації про фільми з відкритого джерела було використано The Movie Database API.

API або прикладни́й програмний інтерфейс – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено – це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек .

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані. Програмісти використовують переваги API у функціональності, таким чином їм не доводиться розробляти все з нуля. API є абстрактним поняттям – програмне забезпечення, що пропонує деякий API, часто називають реалізацією даного API. У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір може включати як API, так і інші інструменти, отже ці два терміни не є взаємозамінювані .

Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим .

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, ППІ визначається набором повідомлень запиту HTTP, також визначається структура повідомлень-відповідей, зазвичай у розширенні мови розмітки XML або в форматі об'єктного запису JavaScript (JSON) .

У той час як прикладний програмний інтерфейс у Web історично був практично синонімом для веб-служби, останнім часом тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA) на більш прямі передачі репрезентативного стану (REST) стилів веб-сайтів .

2.7. NPM

NPM (скорочення від Node Package Manager) - це менеджер пакетів для мови програмування JavaScript. npm, Inc. є дочірньою компанією GitHub, американської

транснаціональної корпорації, яка забезпечує розміщення програмного забезпечення та контроль версій за допомогою Git. Це менеджер пакетів за замовчуванням для середовища виконання JavaScript Node.js. Він складається з клієнта командного рядка, який також називається npm, та онлайн-бази даних загальнодоступних та платних приватних пакетів, яка називається реєстр npm. Доступ до реєстру здійснюється через клієнта, а доступні пакети можна переглядати та шукати через веб-сайт npm. Менеджером пакетів та реєстром керує npm, Inc.

NPM включений як рекомендована функція в програмі встановлення Node.js. npm складається з клієнта командного рядка, який взаємодіє з віддаленим реєстром. Це дозволяє користувачам споживати та розповсюджувати модулі JavaScript, доступні в реєстрі. Пакети в реєстрі мають формат CommonJS і містять файл метаданих у форматі JSON. Понад 477 000 пакетів доступні в основному реєстрі npm. Реєстр не має жодного процесу перевірки для подання, що означає, що знайдені там пакети можуть бути низької якості, незахищеними чи шкідливими. Натомість npm покладається на звіти користувачів про видалення пакунків, якщо вони порушують політику через низьку якість, небезпеку чи зловмисність . npm надає статистичні дані, включаючи кількість завантажень та кількість залежних пакунків, щоб допомогти розробникам оцінити якість пакетів.

У версії npm 6 була введена функція аудиту, яка допомагає розробникам виявляти та виправляти проблеми вразливості та безпеки в встановлених пакетах. Джерело проблем із безпекою було взято із звітів, знайдених на платформі Node Security Platform (NSP), і інтегровано з npm з моменту придбання npm NSP.

NPM може керувати пакетами, які є локальними залежностями певного проекту, а також глобально встановленими інструментами JavaScript. При використанні в якості диспетчера залежностей для локального проекту, npm може за допомогою однієї команди встановити всі залежності проекту через файл package.json. У файлі package.json кожна залежність може вказати діапазон допустимих версій, використовуючи семантичну схему версій, що дозволяє розробникам автоматично оновлювати свої пакети, одночасно уникаючи небажаних змін. npm також надає інструменти для встановлення версій для розробників, щоб

позначити свої пакети певною версією. npm також надає файл package-lock.json, який містить запис точної версії, яка використовується проектом після оцінки семантичного встановлення версій у package.json.

2.8. VS Code

VS Code являє собою безкоштовний засіб для створення, редагування і дебаґінгу сучасних веб-додатків і програм. Середовище розробки розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X .

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим крос-платформним продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js.

Редактор містить вбудований дебаґер, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як спрощене рішення, що дозволяє обійтися без повного інтегрованого середовища розробки .

Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, R, Objective-C, PowerShell, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS.

РОЗДІЛ 3. РОЗРОБКА ВЕБ ДОДАТКА ДЛЯ МОНІТОРИНГУ ЗАХВОРЮВАНОСТІ НА COVID-19 В СВІТІ

3.1. Пошук відкритого джерела данних захворюваності на COVID-19

Оскільки боротьба з COVID-19 триває, моніторинг прогресу пандемії став головним пріоритетом. Проінформованість є абсолютно важливою в умовах кризи - як для громадян, так і для науковців. На щастя, організації та розробники створили доступними ключові показники статистики для громадськості.

Підрахунок випадків, летальний результат, відновлення та географічні дані допомогли нам зрозуміти цю еволюційну загрозу. Центр системної науки та інженерії при Університеті Джона Хопкінса підтримує, мабуть, найпопулярніший на сьогодні інструмент моніторингу. Світова організація охорони здоров'я (ВООЗ) також має власну глобальну інформаційну панель, доповнену регіональною візуалізацією даних. Це легкозасвоювана інфографіка, від якої може отримати вигоду кожен.

Веб-сайти можуть збирати цю статистику COVID через API. Ми розглянули деякі з цих API та їх постачальників нижче, в надії на демократизацію цієї важливої інформації.

Apify - це ринок API, багато ресурсів якого нещодавно було присвячено боротьбі з COVID-19. Веб-сайт навіть пропонує власну інформаційну панель, яка періодично оновлюється. Однак основним внеском компанії є бібліотека API, що охоплює 42 мови. Сюди входять 40 країн, уніфікований набір даних "Світ", та Лос-Анджелес, Каліфорнія.

Ці API викликають числові підсумки для наступного:

- Обстежені особи
- Заражені особи (підтверджено)
- Одужалі особи

- Померлі особи
- Перевірені особи
- Інфіковані та померлі в регіоні (певні країни)
- Підозрювані випадки
- Карантинні особи

Apify зазначає, що багато офіційних веб-сайтів моніторингу COVID не пропонують API. Бібліотека COVID-19 - це спроба перетворити сайти відстеження пандемії на загальнодоступні ресурси. Розробники Marketplace розробили URL-адреси з відкритим кодом. Ці кінцеві точки на 100% безкоштовні. Це робить моніторинг спалаху доступним як для любителів, так і для організацій.

Найкраще, служба завжди приймає нові внески - ці ресурси, швидше за все, збільшаться протягом найближчих місяців. Таким чином, ми можемо розраховувати, що з часом додатки та веб-сайти розмістять дані відстеження.

Наступний кандидат - розроблений Хав'єром Авілесом, проект CovidAPI вилучає дані з основних джерел та компілює їх на вашій інформаційній панелі. API надає глобальну статистику, тоді як спеціалізовані кінцеві точки з легкістю збирають інформацію про конкретну країну. CovidAPI є форком старішого API - додає деякі виправлення коду та іонну зручність інтерфейсу.

CovidAPI також пропонує індивідуальні частоти вишкрібання. Він також безкоштовний та з відкритим кодом, завдяки чому ідеально підходить для численних проектів.

Наступний - розроблене Родріго Помбо, разом із 77 учасниками, рішення covid19 повертає підтверджені випадки, смерті та відновлення на національному рівні. Проект збирає інформацію про COVID-19, одночасно повертаючи результати у форматі JSON. Рішення Pombo є центральним стовпом чотирьох інших API:

- API COVID-19 Grafana для візуалізації статистики в Grafana
- API COVID-19 GraphQL
- API CovidAPI.info, RESTful рішення, створене для інформаційних панелей
- Самоцвіт COVID-19 Ruby

Шістдесят шість різних візуалізацій покладаються на covid19 - як на настільному, так і на мобільному телефоні. Вони варіюються від щоденних звітів, до глобальних звітів, порівнянь та кривих зараження. Проект covid19 ідеально підходить для тих, хто поєднує унікальні інформаційні панелі з надійною інформацією. Він безкоштовний і з відкритим кодом, що підвищує його популярність серед численних розробників.

Наступний, як і Apify, RapidAPI's Marketplace пропонує 13 API для коронавірусу для громадського споживання. П'ять з них - це інтерфейси API Freemium (оплата за функцією або кількістю дзвінків), тоді як решта вісім повністю безкоштовні. Незалежно від того, розпочати роботу слід порівняно просто. Зауважте, що найпопулярніші з них широко використовувались протягом останніх 30 днів, збільшуючи середній час затримки вгору. Ці API вимагають чогось із наступного:

- Загальна світова статистика
- Випадки за країнами
- Випадки за країнами по днях
- Дані для всіх країн
- Інструкція з використання маски
- Історії зараження за країнами
- Показники, характерні для Індії
- Північноамериканські діячі
- Дані за кодом ISO
- Статистика тестування
- Дані про транспортну інфекцію

Ці API функціонують добре в цілому, хоча показники успіху можуть сильно відрізнятися залежно від API. Ті, хто потребує максимуму в статистичній стабільності COVID-19, повинні це пам'ятати. Інді-розробники будуть тяжіти до безкоштовних опцій RapidAPI. Однак ті, у кого менші користувацькі бази, можуть

обійтись за допомогою інтерфейсів freemium. Кожен API часто оновлюється, хоча, як виявилось, Coronavirus (COVID-19) API припинено.

Нещодавно Харіх запустив те, що називається ініціативою API COVID19, орієнтованою на дослідників та розробників. Доступ до їх Covid19API безкоштовний - як і річне використання, обмежене 500 000 дзвінків на місяць. Оскільки ця пандемія непередбачувана, розширений доступ може виявитися безцінним. Ви також можете налаштувати кінцеві точки Covid19API для доступу до певних даних. Це включає:

- Загальний підрахунок справ
- Випадки за країнами
- Смерть та відновлення по країнах
- Смерть та одужання у всьому світі

Рішення навіть пов’язує з популярними даними інформаційної панелі Джона Хопкінса. Ми навіть можемо відстежувати унікальну інформацію, таку як лікарняні ліжка та визначні місця.

Початок роботи з Харіх вимагає реєстрації. Ви створите організацію, файл проекту та дослідите доступні кінцеві точки REST Covid19API. Публікація будь-якої вибраної кінцевої точки - це просто команда curl або PowerShell. Компанія також пропонує підтримку - ідеально підходить для нових розробників та компаній із надійною реалізацією API.

Послуга пропонує інформаційну панель кінцевих даних, відображення та аналітику. Легко побачити, як працює Covid19API, гарантуючи, що ви та ваші користувачі отримуєте найновіші події спалаху без суєти. Харіх також включає файли Swagger та OpenAPI для додаткового використання. Це ідеальний загальний варіант для додатків із великим трафіком.

Зберігаючи найповніший ресурс остаточно, ми маємо колекцію API COVID-19 від Postman. На сайті розміщена документація щодо 28 API COVID та колекцій API. Вони зібрані з офіційних джерел, таких як ВООЗ, CDC та ECDC. Сторінка містить численні API, специфічні для локалі, і при цьому надає способи доступу до

глобальної статистики. Розробники можуть запускати кожен API в середовищі Postmates для тестування.

Моніторинг коронавірусу стосується не лише метрик. Ми також повинні бути в курсі офіційних подій, оголошень та процедур. З урахуванням цього доступні наступні дані:

- YouTube, Twitter та RSS-канали для ВООЗ, CDC та ECDC
- Інформація про COVID-19 для всіх 50 штатів США, плюс п'ять територій
- Загальна інформація про поточні справи
- Інформація про випадки пневмонії, пов’язаної з COVID-19
- Глобальна статистика, статистика країн та відповідні терміни Індексація пошукових систем для понад 60 наукових сховищ
- Смерті
- Кількість досліджених зразків
- Трендові пошукові терміни в Twitter
- Облікові записи Twitter губернатора штату та управління охорони здоров’я
- Контактна інформація як державного, так і окружного управління охорони здоров’я

Це справді вражаючі ресурси, які пропонують набагато більше, ніж типові API, що керуються чисельністю. Рішення, пропоновані Postman, якісні та надійні. Вони чудово підходять для інфографіки, інформаційних панелей та підвищення рівня обізнаності громадськості на різних рівнях.

Останній кандидат - The COVID Tracking Project. У центрі проекту відстеження COVID знаходиться його API даних, який повертає корисну статистику у форматі JSON або CSV. Дані накопичувальні, тому ваші показники з часом не зменшаться автоматично. Ресурс зосереджений на американських показниках, пропонуючи інформацію на національному, штатному та окружному рівнях. Зусиллями керує Алексіс Мадрігал за підтримки 100 добровольців з різних дисциплін.

На детальному рівні користувачі можуть запитувати таке:

- Сукупні позитивні результати тестів
- Сукупні негативні результати тестів
- Розраховані диференціали між позитивними та негативними результатами
- Всього госпіталізовано
- Загальна кількість смертей
- Очікування результатів тесту
- Будь-які зміни у вищезазначених цифрах на національному, штатному та окружному рівнях
- Люди під слідством
- Показники CDC

Проект відстеження також пропонує унікальну інформацію. Він призначає оцінки літер відповідям і навіть оцінює джерела даних на основі їх надійності (або прозорості). Користувачі можуть запитувати хеші та перевіряти наявність оновлень. Ви навіть можете переглянути інформацію соціальних мереж департаменту охорони здоров’я. Нарешті, історичні дані легко отримати за допомогою виклику API. Проект відстеження COVID пропонує вражаючий вибір варіантів розслідування, що відображає вплив його архітекторів.

Перемога над COVID-19 шляхом оцінки даних. Така глобальна пандемія, як COVID-19, вимагає як якісного, так і кількісного аналізу. Ці API можуть з'єднати мільйони - можливо, навіть мільярди - користувачів із важливою інформацією про відстеження. Якщо стати доступними для кожного, це допоможе нам приймати розумніші рішення. Сюди входять звичайні громадяни, науковці, уряди, медичні працівники та розробники. Ці інструменти мають силу змінювати життя. Іноді обізнаності достатньо, щоб здійснити зміни та допомогти людям протистояти невідомому.

3.2. Створення каркасу React.js додатку

Останнім часом React.js завоював шалену популярність. Це потужна і в той же час елегантна бібліотека. Однак його істотним мінусом є складність старту зокрема через необхідність конфігурації babel і webpack.

На допомогу приходить модуль create-react-app. Він дозволяє створювати додатки на React.js без будь-яких додаткових конфігурацій.

React.js розробка полягає в описі того, що потрібно вивести на сторінку (а не в складанні інструкцій для браузера, присвячених тому, як це робити). Це, крім іншого, означає значне скорочення обсягів шаблонного коду.

У React.js використовується одностороння прив'язка даних. Це – великий плюс даної бібліотеки, так як виражається це в тому, що програміст завжди точно знає про те, що привело до зміни стану програми. Подібний підхід до прив'язки даних значно спрощує налагодження додатків.

Для створення основи React.js-додатка, було використано пакет create-react-app від Facebook. Ймовірно, це найпопулярніший підхід до налаштування робочого оточення, яке дозволяє приступити до розробки.

Завдяки пакету create-react-app програміст отримує в своє розпорядження безліч потрібних інструментів, що позбавляє його від необхідності самостійно їх підбирати.

Для того щоб глобально встановити пакет create-react-app, потрібно скористатися такою командою:

- npm i -g create-react-app

Потім, для створення шаблону додатки, потрібно виконати таку команду:

- npx create-react-app COVIDTracker

На цьому попередня підготовка закінчена. Для запуску програми виконайте наступні команди:

- cd COVIDTracker
- npm start

Після виконання цих команд середовище розробки перейде в папку проекту і npm запустить сервер розробки, який дозволяє відкрити створений React.js-додаток, перейшовши в браузері за адресою http://localhost:3000/.

Після створення шаблону проекту потрібно встановити всі необхідні npm залежності для правильного функціонування додатку. Встановлення виконується командою npm і «назва пакету».

Отже, готовий шаблон проекту доповнюється потрібними компонентами та викликами до нашого REST API для отримання інформації і миттєвого виводу її на екран користувача без перезавантаження сторінки за допомогою віртуального DOM.

3.3. Імплементація компонентів

Найважливішим поняттям, яке слід зрозуміти в React.js, є компонент. Компонент React може бути одного з двох типів. Це може бути як функціональний компонент, так і компонент класу. Іноді ви чуєте різні терміни для опису цих двох типів, наприклад, без громадянства та стану. Функціональні компоненти також часто асоціюються з презентаційною концепцією. У цій статті я буду називати їх функціональними компонентами та компонентами класу.

Функціональний компонент - це найпростіша форма компонента React. Це проста функція з простим контрактом:

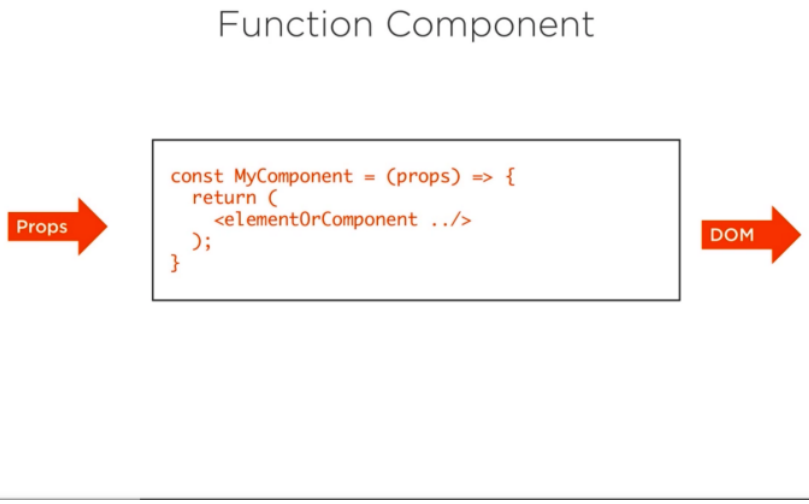

Рис. 3.1 Функціональній компонент React.js

Функціональний компонент отримує об'єкт властивостей, який зазвичай називається props. Він повертає те, що схоже на HTML, але насправді є спеціальним синтаксисом JavaScript, який називається JSX.

Компонент класу є більш функціональним способом визначення компонента React. Він також діє як функція, яка приймає реквізит, але ця функція також розглядає приватний внутрішній стан як додатковий вхід, який контролює повернутий JSX.

Цей приватний внутрішній стан - це те, що надає React його реактивну природу. Коли стан компонента класу змінюється, React повторно відображатиме цей компонент у браузері.

Об'єкти Держава та Реквізит мають одну важливу відмінність. Усередині компонента класу об'єкт State можна змінити, тоді як об'єкт Props представляє фіксовані значення. Компоненти класу можуть змінювати лише свій внутрішній стан, а не свої властивості. Це основна ідея, яку слід зрозуміти в React, і ця стаття матиме приклад цього.

Давайте розглянемо фактичний приклад компонента. Дуже простий, без будь-якого вводу і з простим h1 у виведенні div.

JSX дозволяє описувати наші користувальницькі інтерфейси (UI) у синтаксисі, дуже близькому до HTML, до якого розробники звикли. Однак це необов’язково. React можна використовувати без JSX. Насправді React просто компілює JSX, до чистого JavaScript. Потім він працює зі скомпільованим JavaScript у браузері.

Виклик React.createElement - це представлення JavaScript об’єктної моделі документа (DOM). React ефективно транслює його в операції DOM, які він виконує в браузері.

3.4. Інтеграція з REST API

Останнім часом спостерігається стрімке зростання мобільного інтернет-зв'язку. Переважна більшість людей використовує смартфони, планшети, нетбуки та інші підключені пристрої, які створюють необхідність віддавати відповідний вміст для кожного конкретного пристрою.

Крім того, інтернет стає доступним для нових регіонів і соціальних груп, постійно збільшуючи веб-трафік. З іншого боку, комп'ютери користувачів і браузери, а також сам JavaScript стають все більш і більш потужними, забезпечуючи більше можливостей для обробки даних в інтернеті через клієнтську сторону.

У цій ситуації найкращим рішенням часто може бути відправлення даних як відповідь, а не надсилання вмісту сторінки. Тобто, не потрібно перезавантажувати всю сторінку на кожен запит, а лише відправляти користувачу відповідні дані і дозволяти клієнтському пристрою обробляти дані та будувати новий візуальний інтерфейс на базі нових даних.

Приклади життєвих циклів таких веб-сайтів наведені на рисунку 3.2.

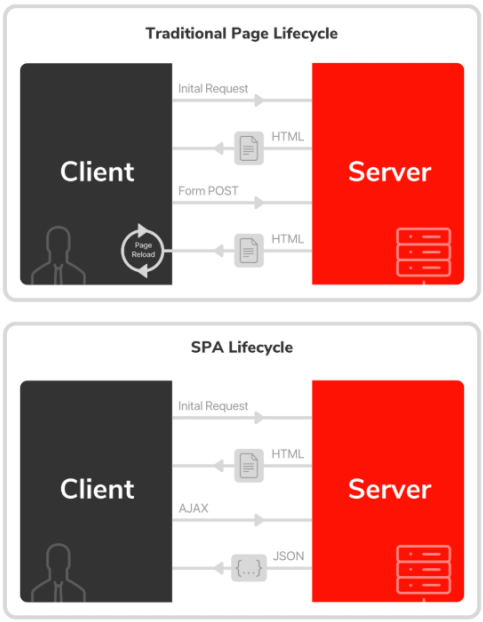

Рис. 3.2 Відмінності звичайного веб-сайту та SPA

Отже ціллю дипломної роботи було створити серверну програму, що описує віддалений API на основі протоколу REST і інтерфейсний JavaScript додаток, який зв'язується з API і надає всі дані на пристрої.

Бекенд дані споживаються людьми, тому необхідно розробити інтерфейс користувача (UI), щоб забезпечити користувачам можливість керувати даними.

Сучасні веб-програми повинні мати адаптивні та дружні до користувача інтерфейси, що забезпечують належний досвід використання додатку.

Крім того, сучасні користувальницькі інтерфейси можуть бути довільно складними, з багатопанельними, вкладеними макетами, сторінковими даними, індикаторами прогресу тощо. У цьому випадку модель компонента може бути правильним рішенням.

React.js – це легкий фреймворк JavaScript, який орієнтований на створення веб-інтерфейсів на основі компонентів. React.js не надає жодних засобів для спілкування з сервером, але можливо використовувати будь-яку бібліотеку зв'язку з компонентів React.js.

В результаті, було розроблено сучасний React.js додаток, який використовує API REST, створений напередодні. API надає способи використання програми керування онлайн-колекцією. Завданням було розробка веб-інтерфейсу для використання цих методів.

Гарною практикою є створення всіх пов'язаних функції в одному місці. Розміщення функціоналу в одному місці, який надає певні API, забезпечує більшу гнучкість і тестування для нашої програми.

Таким чином, було створено клас послуг зв'язку, який реалізує всі основні операції CRUD для обміну даними з сервером і виводить ці операції як методи для всіх компонентів React.js.

Щоб інтерфейс користувача був більш чутливим, було реалізовано ці методи як асинхронні. За умови, що API незмінний, існує можливість вільно змінити реалізацію, і жоден з користувачів не постраждає.

React.js підтримує ієрархії компонентів, де кожен компонент може мати стан, а стан може бути спільним між відповідними компонентами. Крім того, поведінка кожного компонента може бути налаштована шляхом передачі властивостей до нього. Таким чином, було розроблено головний компонент, який містить список елементів колекції і працює як заповнювач для відображення форм для відповідних дій CRUD.

Для керування станом додатку використовується методологія Redux. Замість збереження стану в кожному дочірньому компоненті і синхронізації станів, а отже і

появи пов'язаних компонентів, загальний стан передається до одного головного загального стану додатку (store).

Таким чином, зберігається стан у компоненті батьківського додатка з використанням функцій зворотного виклику, які передаються до дочірніх компонентів через властивості. Потім викликаються функції зворотного виклику всередині обробників подій у дочірніх компонентах. У цих функціях змінюється стан головного стану відповідно до дій користувача, запущених у дочірніх компонентах.

На підставі зміни головного стану додатку, React.js повторно передає дочірнім компонентам властивості, якщо це необхідно. Таким чином, існує одне єдине управління станом, що робить нашу модель інтерфейсу більш послідовною.

Хоча React.js не надає жодної вбудованої підтримки для надсилання запитів на сервер, є можливість використовувати будь-яку бібліотеку зв'язку в наших програмах React.js. Було вирішено використовувати бібліотеку axios, яка майже є стандартним способом надсилання HTTP-запитів і підтримується в всіх сучасних браузерів. Схему взаємодії React.js/Redux з REST API зображено на рисунку 3.3.

Отже, за умови, що інтерфейс комунікації визначений, є можливість легко замінити макетну реалізацію сервісу.

Також було дотримано принципу єдиної відповідальності і проставлено всі налаштування конфігурації запиту в один об'єкт, який можна імпортувати у всі відповідні компоненти.

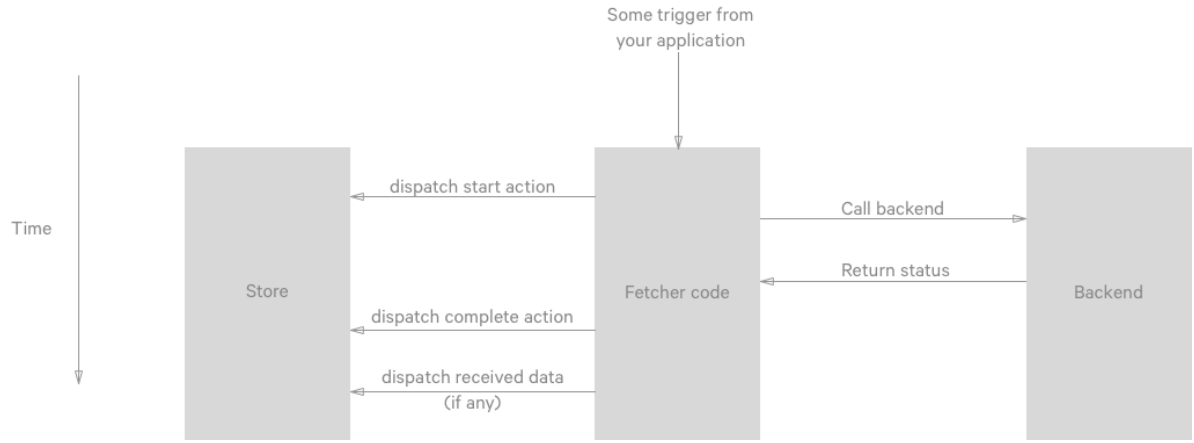

Рис. 3.3 Схема взаємодії React.js/Redux з REST API

Таким чином, було розроблено повністю функціональний веб-додаток, який підтримує основні операції з управління колекціями, тобто можливість додавати, переглядати, оновлювати і видаляти елементи.

Використовуючи модель компонента React.js, можливо створювати складні користувальницькі інтерфейси з вкладеними, багатосторінковими переглядами, забезпечуючи багатий досвід користувачів.

3.5. Структура веб додатку

Програмний продукт має є фронтенд додатком. Проект складається з наступних компонентів:

- .git (системна папка системи контролю версій Git);
- src (містить всю фронтенд частину. React.js та необхідні для його функціонування компоненти);
- public (містить базовий HTML документ та статичні данні)
- node_modules (містить всі встановлені залежності для проекту);
- .gitignore (системний файл Git для ігнорування деяких папок та файлів при збережені проекта на сервісі);
- package.json (список всіх залежностей та їх версій для проекту);

Загалом, сукупність всіх складових архітектури представляє собою працюючий програмний комплекс.

РОЗДІЛ 4. ВИКОРИСТАННЯ ДОДАТКУ КЛІЄНТОМ

4.1. Вимоги до програмного забезпечення

Функціональні вимоги – це вимоги до програмного забезпечення, які описують внутрішню роботу системи, її поведінку. Наприклад калькулювання даних, маніпулювання даними, обробка даних та інші специфічні функції, які має виконувати система.

Функціональні вимоги до веб-додатку «COVIDTracker»:

- Перегляд кількості хворих в світі
- Перегляд кількості хворих в певній країні
- Перегляд кількості померлих в світі
- Перегляд кількості померлих в певній країні
- Перегляд кількості людей що одужали в світі
- Перегляд кількості людей що одужали в певній країні
- Перегляд графіку з динамікой росту захворювання
- Перегляд інтеактивної карти захворюваності в світі

Вимоги для функціонування додатку діляться на дві категорії:

– локальні вимоги – рекомендований мінімальний набір апаратного та програмного забезпечення для функціонування на пристроях клієнта додатку для його коректного та швидкого функціонування, для зручності додавання та розширення функціоналу. Для кросплатформених додатків локальні вимоги включають в себе не тільки персональні комп’ютери, а й мобільні пристрої;

– мережеві вимоги – рекомендований мінімальний набір апаратного та програмного забезпечення для функціонування серверної частини, куди буде надходити великий потік даних, а також буде відбуватись обробка запитів клієнта.

Зазвичай мережеві вимоги до апаратного забезпечення відрізняються від

локальних, так як додаток на сервері працює з великою кількістю користувацьких підключень та великою кількістю вхідної інформації для обробки, але у нашому випадку, створений додаток використовує *API* для роботи з даними, яке працює з надпотужними віддаленими серверами.

Для того, щоб скористатись веб-додатком «COVIDTracker», необхідно щоб ПК відповідав рекомендованим вимогам для запуску сучасного браузера. При недотриманні рекомендованих вимог, правильна робота системи не гарантована: додаток може не запрацювати взагалі, або його робота може бути сповільнена (при запуску на слабких ПК або застарілих версіях браузера).

Рекомендовані системні вимоги веб-додатку «COVIDTracker»:

- ОС: Windows XP або вище, Linux, MacOS, IOS, Android, Chrome OS;
- процесор (CPU): 1.6 ГГЦ;
- відеокарта: не менше 256 МБ пам’яті;
- оперативна пам’ять: не менше 512 МБ ОЗУ;
- вільне місце на жорсткому диску: 5 МБ;
- браузер актуальної версії.

Авторський веб-додаток розроблений власноруч автором дипломної роботи. Назва додатку – «COVIDTracker».

В даному веб-додатку реалізована можливість перегляд кількості хворих в світі на COVID-19. Реалізована інтерактивна карта та динамічні графікі. Данні оновлюються при кожному запиті і є актуальними.

Авторський веб-додаток має значні переваги в швидкості та способі завантаження контенту в порівнянні з конкурентами. Основними перевагами є зручність, простота, швидкість та кросплатформність веб-додатку.

Веб-додаток «COVIDTracker» розроблений для роботи в різних ОС, таких як Windows, Linux, MacOS, iOS, Android з використанням мови програмування JavaScript та додаткових бібліотек, React.js, Node.js. Основна вимога – підтримка середовища веб-браузера.

Дане програмне забезпечення не є ліцензованим та патентованим, може вільно розповсюджуватись, та є безкоштовним.

4.2. Запуск додатку

React за допомогою якого описано додаток, запускає головний JavaScript файл index,js. Цей файл буде зібрано за допомогою Webpack.

Webpack - це модульний пакет. Його основна мета - згрупувати файли JavaScript для використання в браузері, але він також здатний трансформувати, групувати або упаковувати майже будь-який ресурс або актив. Файли, які будуть зібрані в бандл, розташовані в каталозі /build. Файли package.json, використовувані для управління додатком і його залежностями, розташовуються в каталозі / /package.json.

Рис. 4.1. Конфігурація package.json файлу

Кожен користувач додатку має свій файл package.json, в якому описано, в який час і які залежності включати до додатку від імені цього користувача. Для редагування файлу package.json використовується спеціальний редактор коду, який підтримує файли JSON формату, що дозволяє не переривати процес підключення залежностей на час редагування.

Файл package.json складається з ключів та значень, поділених пробілами або табуляторами. Тут можна задати основні налаштування додатку, такі як шлях до файлу, з якого починається збірка бандлеру, скрипти для запуску додатку. Об’єкт із залежностями додатку містить у собі інформацію про всі залежності у вигляді назв бібліотек та їх версій. У потрібний момент часу бібліотеки підключаються до додатку та виконують свої функції за допомогою менеджеру пакетів NPM.

Щоб запустити додаток, клієнт відкриває корінь додатку та вводить в термінал команду «npm start». Після завантаження всіх ресурсів додаток будет відкритий за адрессою http://localhost:3000/.

4.3. Інструкція користувача

При відкритті додатку користувача зустрічаюсь всі основні компоненти на головній сторінці. Головні елементі керування розміщені по головному екрані на інтуітивно зрозумілих місцях для користувача.

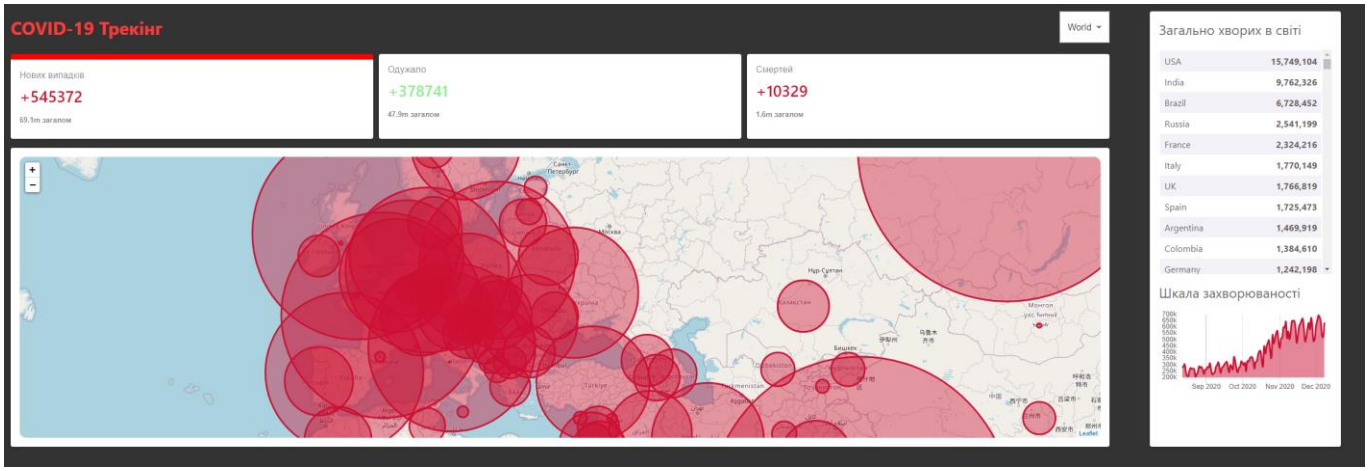

Рис. 4.2. Загальний вигляд додатку

Перше на що варто звернути увагу це світова статистика захвоюваності, яка зображена на рисунку 4.3.

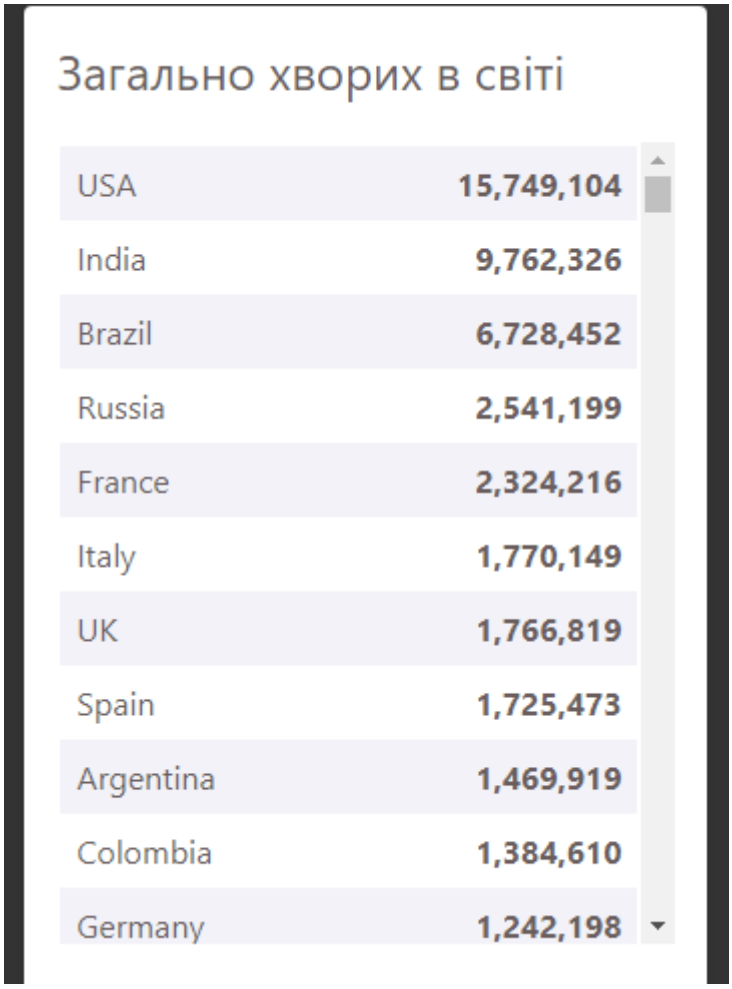

Рис. 4.3. Світова статистика захвоюваності

Тут ми бачимо відсортовану від більшого до меньшого статистику по всім країнам світу.

Наступний елемент це шкала захворюваності (див. Рис. 4.4).

Рис. 4.4. Шкала захворюваності

Шкала захворюваності дає можливість візуально оцінити маштаби та темпи росту захворюваності.

Для зручної навігаціє створено селектор країн зі списком всіх країн світу. Щоб переглянути статистику по певній країні – потрібно обрати її в селекторі і данні будут автоматично завантажені і дані на сторінці обновляться (див. Рис. 4.5).

Рис. 4.5. Селектор країн

В додтаку була розроблена інтерактивна карта, яка показую зони захворювання в світі. Є моєливість натиснути на країну та дізнатися детальну статистику про хворих (див. Рис. 4.6).

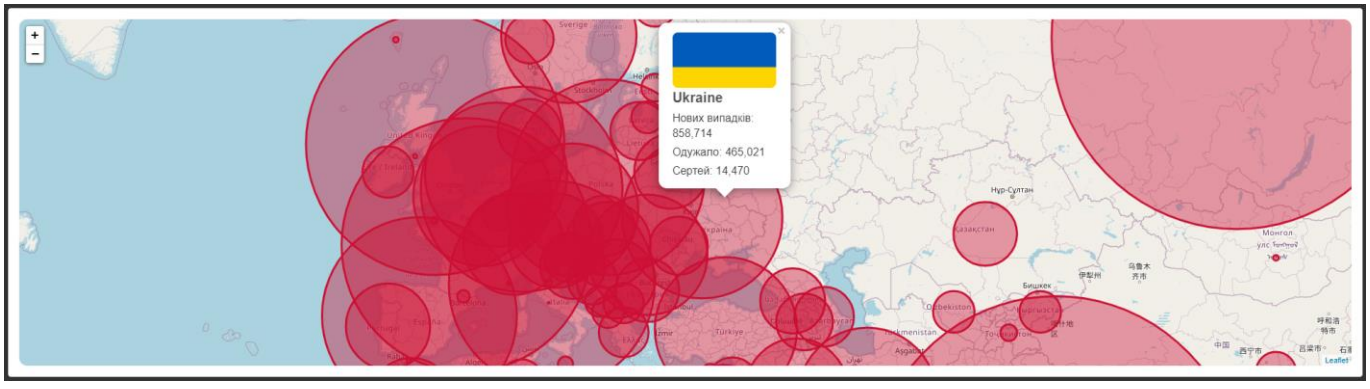

Рис. 4.6. Селектор країн

Над картою також розміщена більш детальна статистика по обранії крахїні або в світі в цілому. На кожен пункт можливо натиснути та інтерактивна карта

покаже візуалізовану статистику по обраному пункту (див. Рис. 4.7).

Рис. 4.7. Селектори статистики для інтерактивної карти

Програмний продукти було адаптовано під мобільні пристрої (див. Рис. 4.8).

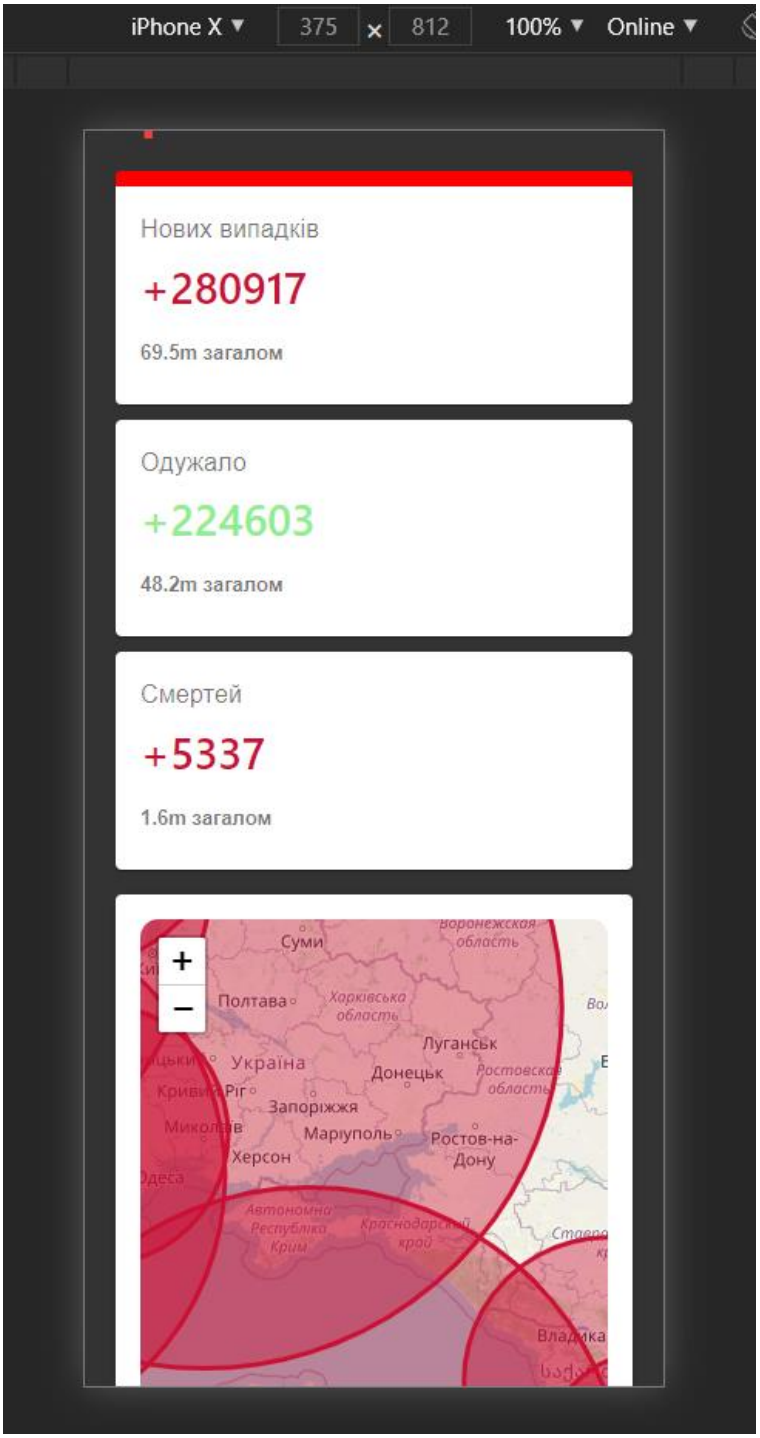

Рис. 4.8. Програмний продукт на iPhone X

ВИСНОВКИ

В ході даної дипломної роботи, було детально досліджено та проаналізовано популярну концепцію створення веб-додатків Single Page Application та ключові елементи при проектувані, наведені приклади сучасних односторінкових додатків.

Були розглянуті засоби і технології створення веб-застосувань, визначено мову веб-програмування JavaScript для створення кросплатформних веб-додатків з використанням фреймворку React.js, платформи Node.js та стандартної мови розмітки HTML з використанням каскадних стилів CSS.

Проаналізувано актуальність, недоліки та переваги SPA у порівнянні з традиційним веб-сайтом та десктопним додатком було встановлено, що на сьогодні десктопні додатки мають перевагу у продуктивності, мережевою незалежності та більших можливостях.

Односторінкові веб-додатки в свою чергу мають переваги в кросплатформності й відсутній необхідності встановлення. Також було встановлено що концепція набула популярності тільки нещодавно, оскільки з’явились інструменти, фреймворки і технології що дозволяють значно зменшити великий об’єм роботи, що необхідний для реалізації даної концепції та значно спростити процес розробки.

Також, було досліджено процес створення веб-додатку від концептуальної моделі до вибору шаблонів проектування та рішень що використовуються при розробці таких додатків.

Встановлено, що архітектура односторінкового додатку має певні необхідні елементи для функціонування, проте велика кількість підходів не дозволяють створити єдиний правильний варіант розробки. Вибір технології, архітектурних рішень та підходів залежить від задач які буде вирішувати додаток та його складності.

Детально розглянуто технології, які використовувалися при розробці веб-додатку «COVIDTracker». Проаналізовано їх переваги та способи взаємодії між собою.

Розглянуті фреймворки та технології дозволяють зробити висновок про зростання інтересу до розробки веб-додатків в останні декілька років. Нові фреймворки з’являються кожні декілька місяців й значно спрощують процес розробки зменшуючи об’єм коду, структуруючи його та адаптуючи відомі рішення з інших платформ.

Був повністю розглянутий процес створення односторінкового веб-додатку з використання обраного фреймворку React.js та платформи Node.js. Було розглянуто необхідне для роботи фреймворка програмне забезпечення та процес його встановлення. Розроблено REST API для взаємодії веб-додатку з сервером.

Також була розглянута переважна більшість основних можливостей фреймворка і досліджено як їх використовувати. Детально описана структура додатку, процес створення й запуску додатку. Розробка з використанням даного фреймворку була зручна й комфортна. Його можливостей цілком достатньо для створення досить складних додатків.

Набуті в ході даної роботи знання концепції, архітектури та практичні навички роботи з фреймворком React.js були використанні в розробці інтерфейсу у вигляді односторінкового веб-додатку, мета якого полягає у створенні найсучаснішого SPA, розробки структури та вибору методів обробки даних і алгоритмів функціонування програмних модулів, забезпечення якісних показників роботи інформаційної системи та створення зручного для кінцевого користувача інтерфейсу.

Отже, концепція односторінкових додатків має значні переваги і перспективу розвитку у майбутньому. Кількість інструментів, фреймворків та технологій, основним напрямом яких є розробка SPA, швидко зростає. Це суттєво спрощує створення нових додатків.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Amler V. ReactJS by Example - Building Modern Web Applications with React / V. Amler, P. Sonpatki., 2016. – 280 c. – (2nd Edition).
2. Banks A. Learning React: Learning React Functional Web Development with React and Redux / A. Banks, E. Porcello., 2017. – 350 c.
3. Bugl D. Learning Redux: Write maintainable, consistent, and easy-to-test web applications / Daniel Bugl., 2017. – 374 c.
4. Chinnathambi K. Learning React / Kirupa Chinnathambi., 2018. – 304 c. – (2nd Edition).
5. Create React App [Електронний ресурс] // Facebook. – 2019. – Режим доступу до ресурсу: https://github.com/facebook/create-react-app.
6. Dawson C. JavaScript's History and How it Led To ReactJS [Електронний ресурс] / Chris Dawson. – 2014. – Режим доступу до ресурсу: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/.
7. Elizabeth L. Should You Develop a Desktop or Web App? [Електронний ресурс] / Laura Elizabeth. – 2015. – Режим доступу до ресурсу: https://www.sitepoint.com/web-desktop-apps/.
8. Feldman R. Developing a React.js Edge: The Javascript Library For User Interfaces / R. Feldman, F. Bagnardi, S. Hojberg., 2018. – 232 c. – (2nd edition).
9. Flanagan D. JavaScript: The Definitive Guide / David Flanagan., 2011. – 1096 c. – (6th edition).
10. Freeman A. Pro ASP.NET Core MVC 2 / Adam Freeman., 2017. – 1017 c. – (7th ed. Edition).
11. Freeman E. Head First HTML and CSS / E. Freeman, E. Robson., 2012. – 768 c.
12. , and testing Node.js applications / Evan Hahn., 2016. – 256 c. – (1st Edition).
13. Haverbeke M. Eloquent JavaScript / Marijn Haverbeke., 2018. – 472 c. – (3rd edition).
14. Herron D. Node.js Web Development: Server-side development with Node 10 made easy / David Herron., 2018. – 492 c. – (4th Revised edition).

15. Jin B. Designing Web APIs: Building APIs That Developers Love / B. Jin, S. Sahni, A. Shevat., 2018. – 232 c. – (1st Edition).
16. JSX In Depth [Електронний ресурс] // Facebook. – 2019. – Режим доступу до ресурсу: https://reactjs.org/docs/jsx-in-depth.html.
17. Kononenko V. M. Multilevel intellectual approach to HTTP-requests legitimacy validation / V. M. Kononenko, S. O. Kravchuk, Yu. V. Ivlev, L. A. Kononenko // Telecommunication sciences. - 2013. - Vol. 4, no. 1. - С. 27-32. - Режим доступу: http://nbuv.gov.ua/UJRN/Telnau_2013_4_1_7.
18. Krill P. React: Making faster, smoother UIs for data-driven Web apps [Електронний ресурс] / Paul Krill. – 2014. – Режим доступу до ресурсу: https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html.
19. Ledyayev R. React: an architectural framework for the development of a software product line for dependable crisis management systems / R. Ledyayev, B. Ries, A. Gorbenko // Радіоелектронні і комп'ютерні системи. - 2012. - № 7. - С. 284–288. - Режим доступу: http://nbuv.gov.ua/UJRN/recs_2012_7_50.
20. Mikowski M. Single Page Web Applications: JavaScript end-to-end / M. Mikowski, J. Powell., 2013. – 432 c. – (1st Edition).
21. Moreau E. What Exactly Is a Web Application? [Електронний ресурс] / Elise Moreau. – 2019. – Режим доступу до ресурсу: https://www.lifewire.com/what-is-a-web-application-3486637.
22. Node.js in Action / [A. Young, B. Meck, M. Cantelon та ін.]., 2017. – 392 c. – (2nd Edition).
23. Prokopiak S. Багатосторінкові та односторінкові додатки, їх переваги та недоліки [Електронний ресурс] / Stepan Prokopiak // DOU. – 2018. – Режим доступу до ресурсу: https://dou.ua/forums/topic/25444/.
24. Purewal S. Learning Web App Development: Build Quickly with Proven JavaScript Techniques / Semmy Purewal., 2014. – 306 c. – (1st Edition).
25. Quick-Start Guide to mLab [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://docs.mlab.com/.

26. Resig J. Secrets of the JavaScript Ninja / J. Resig, B. Bibeault, J. Maras., 2016. – 464 c. – (2nd Edition).
27. Richardson L. RESTful Web APIs: Services for a Changing World / L. Richardson, M. Amundsen, S. Ruby., 2013. – 406 c. – (1st Edition).
28. Shen S. Differences Between Designing a Web App and a Desktop App [Електронний ресурс] / Shan Shen. – 2018. – Режим доступу до ресурсу: https://medium.muz.li/differences-between-designing-a-web-app-and-a-desktop-app-e7d65a7545eb.
29. Simpson K. You Don't Know JS [Електронний ресурс] / Kyle Simpson. – 2018. – Режим доступу до ресурсу: https://github.com/getify/You-Dont-Know-JS.
30. Systems W. Веб-розробка: етапи, стандарти, реальні проекти [Електронний ресурс] / Web Systems. – 2019. – Режим доступу до ресурсу: https://web-systems.solutions/blog/web-development-stages-standards-projects/.
31. Visual Studio Code FAQ [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://code.visualstudio.com/docs/supporting/faq.
32. Базурін В. М. Особливості навчання веб-програмування мовою JavaScript студентів-математиків / В. М. Базурін // Вісник Житомирського державного університету імені Івана Франка . - 2014. - Вип. 1. - С. 79-83. - Режим доступу: http://nbuv.gov.ua/UJRN/VZhDU_2014_1_17.
33. Джексон К. 9 корисних порад для ознайомлення з React.js [Електронний ресурс] / Кем Джексон. – 2019. – Режим доступу до ресурсу: http://echo.lviv.ua/dev/5608.
34. Клапчук Р. Г. Монолітні веб-сервіси та мікросервіси: порівняння та вибір / Р. Г. Клапчук, В. С. Харченко // Радіоелектронні і комп’ютерні системи. - 2017. - № 1. - С. 51–56. - Режим доступу: http://nbuv.gov.ua/UJRN/recs_2017_1_8.
35. Ковівчак Я. В., Пелешко Д. Д., Кінаш Ю. Є. Технологія створення програмних продуктів : Конспект лекцій з дисципліни «Технологія створення програмних продуктів» для студентів бакалаврського рівня підготовки зі спеціальності 0927 «Видавничо-поліграфічна справа– Львів: Національний університет «Львівська політехніка», 2008. – 122 с.

36. Косова К. О. Навчальний курс "Мова розмітки гіпертексту HTML. Розробка веб-сторінок" для бакалаврів напряму підготовки "Туризм" / К. О. Косова // Науковий часопис НПУ імені М. П. Драгоманова. Серія 2 : Комп'ютерно-орієнтовані системи навчання. - 2015. - №. 15. - С. 85-89. - Режим доступу: http://nbuv.gov.ua/UJRN/Nchnpu_2_2015_15_17.
37. Кутковий Б. Є. Оптимізація API запитів до хмарних систем управління базами даних / Б. Є. Кутковий, Н. Я. Павич // Міжнародний науковий журнал "Інтернаука" . - 2017. - № 14. - С. 88-91. - Режим доступу: http://nbuv.gov.ua/UJRN/mnj_2017_14_18.
38. Петрик М.Р. Моделювання програмного забезпечення : науково-методичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.
39. Попхадзе О. А. Розгляд перспективної концепції побудови композитних веб-додатків / О. А. Попхадзе // Системи обробки інформації. - 2016. - Вип. 5. - С. 137-141. - Режим доступу: http://nbuv.gov.ua/UJRN/soi_2016_5_29.
40. Щербакова М. Є. Функціональні особливості JavaScript-додатків / М. Є. Щербакова, Є. В. Щербаков // Вісник Східноукраїнського національного університету імені Володимира Даля. - 2014. - № 10. - С. 142-146. - Режим доступу: http://nbuv.gov.ua/UJRN/VSUNU_2014_10_30.
41. Яцишин В. В. SINGLE PAGE APPLICATION ЯК ТЕХНОЛОГІЯ FRONT END РОЗРОБКИ [Електронний ресурс] / Василь Васильович Яцишин // 2017 – Режим доступу до ресурсу: http://elartu.tntu.edu.ua/bitstream/lib/23102/2/CAZST_2017v2_Yatcyshyn_V_V-Single_Page_Application_128-129.pdf.

ДОДАТКИ

Додаток А. Лістинг программи

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));


import React, { useState, useEffect } from "react";
import "./App.css";
import {
  MenuItem,
  FormControl,
  Select,
  Card,
  CardContent,
} from "@material-ui/core";
import InfoBox from "./InfoBox";
import LineGraph from "./LineGraph";
import Table from "./Table";
import { sortData, prettyPrintStat } from "./util";
import numeral from "numeral";
import Map from "./Map";
import "leaflet/dist/leaflet.css";

const App = () => {
  const [country, setInputCountry] = useState("worldwide");
  const [countryInfo, setCountryInfo] = useState({});
  const [countries, setCountries] = useState([]);
  const [mapCountries, setMapCountries] = useState([]);
  const [tableData, setTableData] = useState([]);
  const [casesType, setCasesType] = useState("cases");
  const [mapCenter, setMapCenter] = useState({ lat: 49, lng: 32 });

  useEffect(() => {
    fetch("https://disease.sh/v3/covid-19/all")
      .then((response) => response.json())
      .then((data) => {
        setCountryInfo(data);
      });
  }, []);
```

```
useEffect(() => {
  const getCountriesData = async () => {
    fetch("https://disease.sh/v3/covid-19/countries")
      .then((response) => response.json())
      .then((data) => {
        const countries = data.map((country) => ({
          name: country.country,
          value: country.countryInfo.iso2,
        }));
        const sortedData = sortData(data);
        setCountries(countries);
        setMapCountries(data);
        setTableData(sortedData);
      });
  };

  getCountriesData();
}, []);

const onCountryChange = async (e) => {
  const countryCode = e.target.value;
  const isWorldwide = countryCode === "worldwide";

  const url = isWorldwide
    ? "https://disease.sh/v3/covid-19/all"
    : `https://disease.sh/v3/covid-19/countries/${countryCode}`;
  await fetch(url)
    .then((response) => response.json())
    .then((data) => {
      setInputCountry(countryCode);
      setCountryInfo(data);
      if (isWorldwide) {
        setMapCenter([34, -40]);
      } else {
        setMapCenter([data.countryInfo.lat, data.countryInfo.long]);
      }
    });
};

return (
  <div className="app">
    <div className="app__left">
```

```
<div className="app__header">
  <h1>COVID-19 Трекінг</h1>
  <FormControl className="app__dropdown">
    <Select
      variant="outlined"
      value={country}
      onChange={onCountryChange}
    >
      <MenuItem value="worldwide" selected>
        World
      </MenuItem>
      {countries.map((country) => (
        <MenuItem
          value={country.value}
          key={country.value + country.name}
        >
          {country.name}
        </MenuItem>
      ))}
    </Select>
  </FormControl>
</div>
<div className="app__stats">
  <InfoBox
    onClick={(e) => setCasesType("cases")}
    title="Нових випадків"
    isRed
    active={casesType === "cases"}
    cases={prettyPrintStat(countryInfo.todayCases)}
    total={numeral(countryInfo.cases).format("0.0a")}
  />
  <InfoBox
    onClick={(e) => setCasesType("recovered")}
    title="Одужало"
    active={casesType === "recovered"}
    cases={prettyPrintStat(countryInfo.todayRecovered)}
    total={numeral(countryInfo.recovered).format("0.0a")}
  />
  <InfoBox
    onClick={(e) => setCasesType("deaths")}
    title="Смертей"
    isRed
    active={casesType === "deaths"}
```

```
            cases={prettyPrintStat(countryInfo.todayDeaths)}
            total={numeral(countryInfo.deaths).format("0.0a")}
          />
        </div>
        <Map
          countries={mapCountries}
          casesType={casesType}
          center={mapCenter}
        />
      </div>
      <Card className="app__right">
        <CardContent className="app__information">
          <h3>Загально хворих в світі</h3>
          <Table countries={tableData} />
          <h3>Шкала захворюваності</h3>
          <LineGraph casesType={casesType} />
        </CardContent>
      </Card>
    </div>
  );
};

export default App;
```

```
.app__stats {
  display: flex;
  justify-content: space-between;
}

.app__left {
  flex: 0.9;
}

.app__information > h3 {
  color: #6a5d5d;
  font-weight: 400;
  font-size: 1.5rem;
  margin-bottom: 1rem;
}

.app__information > h3:last-of-type {
  margin-top: 1rem;
}
```

```
@media (max-width: 990px) {
  .app {
    flex-direction: column;
  }
  .app__stats {
    flex-direction: column;
  }
}

import React, { useState, useEffect } from "react";
import { Line } from "react-chartjs-2";
import numeral from "numeral";

const options = {
  legend: {
    display: false,
  },
  elements: {
    point: {
      radius: 0,
    },
  },
  maintainAspectRatio: false,
  tooltips: {
    mode: "index",
    intersect: false,
    callbacks: {
      label: function (tooltipItem, data) {
        return numeral(tooltipItem.value).format("+0,0");
      },
    },
  },
  scales: {
    xAxes: [
      {
        type: "time",
        time: {
          format: "MM/DD/YY",
          tooltipFormat: "ll",
        },
      },
    ],
```

```
    yAxes: [
      {
        gridLines: {
          display: false,
        },
        ticks: {
          // Include a dollar sign in the ticks
          callback: function (value) {
            return numeral(value).format("0a");
          },
        },
      },
    ],
  },
};

const buildChartData = (data, casesType) => {
  let chartData = [];
  let lastDataPoint;
  for (let date in data.cases) {
    if (lastDataPoint) {
      let newDataPoint = {
        x: date,
        y: data[casesType][date] - lastDataPoint,
      };
      chartData.push(newDataPoint);
    }
    lastDataPoint = data[casesType][date];
  }
  return chartData;
};

function LineGraph({ casesType }) {
  const [data, setData] = useState({});

  useEffect(() => {
    const fetchData = async () => {
      await fetch("https://disease.sh/v3/covid-
19/historical/all?lastdays=120")
        .then((response) => response.json())
        .then((data) => {
          const chartData = buildChartData(data, casesType);
          setData(chartData);
```

```
        });
    };

    fetchData();
  }, [casesType]);

  return (
    <div>
      {data?.length > 0 && (
        <Line
          data={{
            datasets: [
              {
                backgroundColor: "rgba(204, 16, 52, 0.5)",
                borderColor: "#CC1034",
                data: data,
              },
            ],
          }}
          options={options}
        />
      )}
    </div>
  );
}

export default LineGraph;

const Table = ({ countries }) => (
  <table className="table">
    <tbody>
      {countries.map((country) => (
        <tr key={country.country}>
          <td>{country.country}</td>
          <td>
            <strong>{numeral(country.cases).format("0,0")}</strong>
          </td>
        </tr>
      ))}
    </tbody>
  </table>
);
export default Table;
```

Додаток В. Лістинг мапи

```
import React from "react";
import { Map as LeafletMap, TileLayer } from "react-leaflet";
import "./Map.css";
import { showDataOnMap } from "./util";

function Map({ countries, casesType, center, zoom = 3.5 }) {
  return (
    <div className="map">
      <LeafletMap center={center} zoom={zoom}>
        <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" /
>
        {showDataOnMap(countries, casesType)}
      </LeafletMap>
    </div>
  );
}

export default Map;
.map {
  height: 500px;
  background-color: white;
  padding: 1rem;
  border-radius: 4px;
  margin-top: 16px;
  box-shadow: 0 0 8px -4px rgba(0, 0, 0, 0.5);
}

.map .leaflet-container {
  height: 100%;
  border-radius: 12px;
}

.info-flag img {
  width: 100px;
  border-radius: 5px;
}

.info-name {
  font-size: 20px;
  font-weight: bold;
  color: #555;
}
```

```
.info-container {
  width: 150px;
}

.info-flag {
  height: 80px;
  width: 100%;
  background-size: cover;
  border-radius: 8px;
}

.info-confirmed,
.info-recovered,
.info-deaths {
  font-size: 16px;
  margin-top: 5px;
}


import React from "react";
import { Card, CardContent, Typography } from "@material-ui/core";
import "./InfoBox.css";

function InfoBox({ title, cases, total, active, isRed, ...props }) {
  return (
    <Card
      onClick={props.onClick}
      className={`infoBox ${active && "infoBox--selected"} ${
        isRed && "infoBox--red"
      }`}
    >
      <CardContent>
        <Typography color="textSecondary" gutterBottom>
          {title}
        </Typography>
        <h2 className={`infoBox__cases ${!isRed && "infoBox__cases--
green"}`}>
          {cases}
        </h2>

        <Typography className="infoBox__total" color="textSecondary">
          {total} загалом
```

```
        </Typography>
      </CardContent>
    </Card>
  );
}

export default InfoBox;
```

```
.infoBox {
  flex: 1;
  cursor: pointer;
}

.infoBox:not(:last-child) {
  margin-right: 10px;
}

.infoBox--selected {
  border-top: 10px solid greenyellow;
}

.infoBox--red {
  border-color: red;
}

.infoBox__cases--green {
  color: lightgreen !important;
}

.infoBox__cases {
  color: #cc1034;
  font-weight: 600;
  font-size: 1.75rem;
  margin-bottom: 0.5rem;
}

.infoBox__total {
  color: #6c757d;
  font-weight: 700 !important;
  font-size: 0.8rem !important;
  margin-top: 15px !important;
}

@media (max-width: 990px) {
```

```
  .infoBox:not(:last-child) {
    margin: 0;
    margin-bottom: 10px;
  }
}
```