КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Веб-застосунок моніторингу рівня захворюваності та вакцинації на прикладі SARS-CoV-2»

Виконав _____
(Підпис)
Шабатін Павло Євгенійович
(прізвище, ім'я, по батькові)

Керівник доцент Бойко Юлія Петрівна
(прізвище, ім'я, по батькові)
_____ _____
(Резолюція «До захисту»)

Унікальність тексту 93,83 %
Автор _____ Шабатін П.Є.
(Підпис) (Прізвище, ініціали)

Попередній захист:

_____***23.05.2022***_____
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали)
(Дата)

Київ – 2022 року

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «Веб-застосунок моніторингу рівня захворюваності та вакцинації на прикладі SARS-CoV-2»

Освітня програма: Прикладне програмування

Спеціальність: Комп'ютерні науки

ПІБ	Підпис
Шабатін Павло Євгенійович	

Назва роботи українською та англійською мовами:

Веб-застосунок моніторингу рівня захворюваності та вакцинації на прикладі SARS-CoV-2
Web application for monitoring the state of SARS-CoV-2 vaccination and disease rates

Мета бакалаврської (курсової) роботи: надання актуальних даних про епідеміологічну ситуацію пов'язану з SARS-CoV-2 за допомогою веб-застосунку.
План роботи: 1. Сучасні підходи до розроблення і впровадження веб-сервісів 2. Аналіз архітектурних рішень і вибір програмних засобів для реалізації веб-системи 3. Програмна реалізація веб-сервісу для моніторингу ситуації з COVID-19

ПІБ, ступінь, звання наукового керівника роботи: доц. Бойко Юлія Петрівна

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	заява
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2022	виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2022	виконано
9.	Подання роботи у першому варіанті	28.04.2022	виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	23.05.2022	виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	виконано
14.	Захист кваліфікаційної роботи бакалавра	23.06.2022	виконано

Здобувач вищої освіти _____

(підпис)

Керівник _____

(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Заява	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	2
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	6
2	13
3	24
Висновки	2
Перелік використаних джерел	3
Додатки	5

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розробн.</td><td>Шабатін П.Є.</td><td></td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Бойко Ю.П.</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Базилюк А.М.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Дипломна робота: 88 с., 31 рис., 5 табл., 30 джерел, 10 додатків.

Ця дипломна робота присвячена розробці веб-застосунку для моніторингу рівня захворюваності та вакцинації на прикладі епідемії SARS-CoV-2.

Метою дипломної роботи є надання актуальних даних про епідеміологічну ситуацію пов’язану з SARS-CoV-2 за допомогою веб-застосунку.

Для досягнення поставленої мети треба вирішити такі **завдання**:

- Дослідити сучасні підходи до розроблення і впровадження веб-сервісів;
- Проаналізувати архітектурні рішення і обрати програмні засоби для реалізації веб-системи;
- Реалізувати програмно веб-сервіс для моніторингу рівня захворюваності та вакцинації епідемії SARS-CoV-2.

Об’єкт дослідження. Процеси моніторингу епідеміологічної ситуації пов’язаної з SARS-CoV-2.

Предмет дослідження. Програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-застосунку для моніторингу рівня захворюваності та вакцинації на прикладі епідемії SARS-CoV-2.

Методи дослідження.

Теорія управління для дослідження теоретичних аспектів побудови веб-застосунків, емпіричний аналіз і синтез систем, що застосовувався при вивченні прикладів сучасних методів побудови веб-застосунків для моніторингу предметів та явищ, моделювання, побудова діаграм, проектування структур даних.

Ключові слова: веб-застосунок, SARS-CoV-2, моніторинг.

ABSTRACT

Thesis: 88 pages, 31 figures, 5 tables, 30 sources, 10 appendices.

This thesis is dedicated to the development of a web application for monitoring the level of morbidity and vaccination on the example of the SARS-CoV-2 epidemic.

The purpose of the thesis is to provide up-to-date information on the epidemiological situation related to SARS-CoV-2 through a web application.

To achieve this goal you need to solve the following **tasks:**

- Explore modern approaches to the development and implementation of web services;
- Analyze architectural solutions and choose software tools for web system implementation;
- Implement a software web service for monitoring the incidence and vaccination of the SARS-CoV-2 epidemic.

Object of study.

Processes of monitoring of the epidemiological situation.

Subject of study.

Software and technical, organizational principles, principles, approaches to building a web application for monitoring the incidence and vaccination on the example of the SARS-CoV-2 epidemic.

Research methods.

Management theory for research of theoretical aspects of web application construction, empirical analysis and synthesis of systems used in studying examples of modern methods of web application construction for monitoring objects and phenomena, modeling, diagramming, data structure design.

Key words: web application, SARS-CoV-2, monitoring.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

Фреймворк – інфраструктура програмних рішень, що полегшує розробку.

Бібліотека – збірка об’єктів чи підпрограм для вирішення близьких за тематикою задач.

Модуль – функціонально завершений фрагмент програми, оформлений у вигляді окремого файлу з сирцевим кодом або його іменованої частини, призначений для використання в інших програмах.

Деплой – розгортання програмного забезпечення, усі дії, що роблять програмну систему готовою до використання.

Інтрефейс – сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача.

HTTP – HyperText Transfer Protocol.

SSH – Secure Shell.

HTTPS – HyperText Transfer Protocol Secure.

HTML – HyperText Markup Language.

CSS – Cascading Style Sheets.

API – Application Programming Interface.

Бекенд – серверна частина веб-сервісу.

Фронтенд – безпосередньо браузерна частина веб-сервісу, яка включає в себе HTML, CSS, JavaScript.

ВСТУП

Україна і весь світ зараз переживають складні часи у зв’язку з масовою пандемією, яку спричинив вірус COVID-19. Для того, щоб кожна людина мала можливість в будь-який момент подивитися на загальну ситуацію у світі та в окремих країнах, потрібне місце, де буде в зручному вигляді зібрана статистика з найважливішою інформацією стосовно захворювання.

Актуальність цієї теми зумовлено тим, що пандемія SARS-CoV-2 показала, наскільки важливим є швидке отримання актуальних статистичних даних для упередження або вдосконалення методів запобігання шкідливим явищам, огляду поточної ситуації та розробки дій та рішень на основі отриманих даних.

Метою кваліфікаційної роботи бакалавра є надання актуальних даних про епідеміологічну ситуацію пов’язану з SARS-CoV-2 за допомогою веб-застосунку.

Завдання дослідження:

- Дослідити сучасні підходи до розроблення і впровадження веб-сервісів;
- Проаналізувати архітектурні рішення і обрати програмні засоби для реалізації веб-системи;
- Реалізувати програмно веб-сервіс для моніторингу рівня захворюваності та вакцинації епідемії SARS-CoV-2.

Об’єктом дослідження кваліфікаційної роботи бакалавра є процеси моніторингу епідеміологічної ситуації пов’язаної з SARS-CoV-2.

Предметом дослідження кваліфікаційної роботи бакалавра є програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-застосунку для моніторингу рівня захворюваності та вакцинації на прикладі епідемії SARS-CoV-2.

Методи дослідження: теорія управління для дослідження теоретичних аспектів побудови веб-застосунків, емпіричний аналіз і синтез систем, що

застосовувався при вивченні прикладів сучасних методів побудови веб-застосунків для моніторингу предметів та явищ, моделювання, побудова діаграм, проектування структур даних.

Практичне значення одержаних результатів полягає у тому, що розроблена прикладна система може стати корисним, зручним та сучасним рішенням для отримання статистичних даних про епідеміологічну ситуацію SARS-CoV-2.

Структура роботи:

Кваліфікаційна робота бакалавра складається із вступу, трьох розділів, висновку, додатків і списку джерел.

Розділ 1. СУЧАСНІ ПІДХОДИ ДО РОЗРОБЛЕННЯ І ВПРОВАДЖЕННЯ ВЕБ-СЕРВІСІВ

1.1 Веб-сервіс

Веб-сервіс або веб-служба – це програмна система, яка має стандартизовані інтерфейси, за допомогою яких ця система може проводити взаємодію з зовнішніми застосунками за допомогою відправки та отриманні повідомлень, базованих на спеціальних протоколах [1].

Веб-сервіси мають в своєму складі не один протокол, а скупчення відразу декількох, кожен з яких відповідає за окрему частину роботи веб-служби. Ці протоколи розрізняються за такими рівнями:

- Фізичний рівень, до якого відносяться ISDN та RS-232
- Канальний рівень, до якого відносяться Ethernet, Fibre channel та Token ring
- Мережевий рівень, до якого відносяться IP, IPX та ICMP
- Транспортний рівень, до якого відносяться TCP, UDP та SPX
- Прикладний рівень, до якого відноситься найбільша кількість протоколів, серед яких є HTTP, HTTPS, SSH, FTP, SMTP та багато інших.

Також вирізняють поняття «стеків протоколів», які являють собою систематизований набір протоколів, який є достатнім для організації роботи вузлів мережі. Найбільш популярним на даний момент є стек протоколів TCP/IP [2], назва якого походить від двох основних протоколів стеку. Окрім нього, також використовуються IPX/SPX, NetBIOS/SMB, SNA, DECnet, AppleTals. Стек протоколів TCP/IP також корелює з моделлю OSI (вона теж має відповідний стес OSI), яка описує абстрактну еталонну модель для розробки мережевих протоколів. Ця модель описує більшу кількість рівнів, ніж стек TCP/IP, але їх можна зіставити одне з одним.

Таблиця 1.1 Порівняння стеку протоколів TCP/IP з еталонною моделлю OSI

TCP/IP	**OSI**	**Приклад протоколів**
Прикладний	Прикладний	HTTP, FTP
	Представлення	SSL, TLS
	Сеансовий	ASP, RPC
Транспортний	Транспортний	TCP, UDP
Мережевий	Мережевий	IP, ICMP, IPX
Канальний	Канальний	Ethernet, ATM
	Фізичний	Проводи, радіозв’язок

При побудові базового веб-сервісу розробники зазвичай працюють з протоколами прикладного рівня, тому розглянемо деякі з них детальніше:

HTTP – головний протокол всього сучасного інтернету, відповідає за передачу даних, а саме гіпертекстових документів, які в більшості випадків мають розмітку HTML, хоча це протокол має змогу передавати і складніші об’єкти, наприклад зображення [3].

SSH – протокол, який у сучасному інтернеті відповідає за шифрування даних. Цей протокол має декілька версій, але рекомендується використовувати останню (на данний момент це SSH-2), а також

FTP – протокол, який відповідає за передачу файлів між клієнтом та сервером. Для захисту передачі даних в цьому протоколі використовують надбудову над протоколом SSH, криптографічний протокол SSL.

DNS – протокол, який здійснює перетворення імені хоста в його IP-адресу.

POP3 – протокол, який відповідає за доступ клієнта до повідомлень електронної пошти, які знаходяться на сервері.

VoiP – протокол, який дозволяє здійснювати передачу голосових повідомлень через мережу інтернет.

1.2 Протоколи реалізації веб-сервісів

Після дослідження різних протоколів веб-сервісів можна прийти до висновку, що для безпосередньої побудови веб-застосунку потрібно обрати стандартизований програмний інтерфейс, на базі якого буде побудований бекенд застосунок. Такі інтерфейси мають назву API. Є велика кількість описаних стандартів API, але тих, які користуються популярністю, не так вже й багато. Слід зазначити, що можлва розробка і без стандартизованого стилю, але це призведе до проблем як під час розробки, так і після, при спробі розширити веб-застосунок. Розглянемо деякі архітектурні стилі, які використовують для побудови веб-застосунків.

RPC – стиль, який працює на віддалених процедурах. В цьому контексті віддалені процедури – це повідомлення з параметрами і додатковою інформацією, які відправляється від клієнта на сервер, десеріалізується там, після чого сервер, при валідності операції, виконує її та відправляє результат клієнту.

SOAP – це жорстко стандартизований протокол, який використовує формат XML. SOAP вимагає певний формат повідомлень, якими обмінюються клієнт та сервер. Через високий рівень стандартизації та жорсткі вимоги до структури повідомлень SOAP є найбільш детальним стилем API. Протокол є достатньо гнучким для використання різних транспортних протоколів.

REST – найбільш розповсюджений архітектурний стиль на сьогоднішній день, також є одним з найпростіших для вивчення та використання, хоча і має певні обмеження через це. REST, як і SOAP, користується системою передачі повідомлень між клієнтом та сервером, але він не настільки стандартизований і структурований, як його попередник.

GraphQL – синтаксис для відправки точного запиту на конкретні дані. Він являє собою описану схему, яка має вигляд графа, і являє собою інформацію про всі запити та типи даних, які вони повертають. Побудова такої схеми – непростий

процес, але вона дозволяє досить сильно оптимізувати процес пошуку користувачем потрібних даних, але водночас це накладає певні обмеження на можливість веб-кешування. Підтримує підписування на зміну даних в реальному часі за допомогою веб-хуків.

Таблиця 1.2 Порівняння архітектурних стилів [4]

	RPC	SOAP	REST	GraphQL
Організація у вигляді	локальних процедур	обгорнутого повідомлення з жорсткою сруктурою	відповідності шести архітектурним принципам	схеми та типів даних
Формат даних	JSON, XML, Protobuf, Thrift, FlatBuffers	лише XML	XML, JSON, HTML, текст	JSON
Простота вивчення	Просто	Складно	Просто	Посередньо
Ком'юніті	Велике	Мале	Велике	Середнє
Використання	Командні і подіє-орієнтовані API; внутрішні мікросервіси з високою продуктивність	Білінгові системи, фінансові сервіси, сервіси, які вимагають велиої захищенності	Просто публічні застосунки	Мобільні API, складні системи, мікросервіси

Для потреб мого веб-застосунку відразу не підходить SOAP, адже мої дані не потребують серйозного захисту, тому немає сенсу ускладнювати процес. RPC теж не підходить, адже мій застосунок призначений для зовнішнього використання, а RPC краще всього показує себе у мікросервісах.

Залишається вибір між REST та SOAP, і цей вибір можна зробити, оглянувши складність даних. В моєму випадку дані будуть досить простими (описано в наступних розділах), тому немає потреби створювати складні схеми для оптимізації пошуку користувачем. Таким чином, очевидним вибором для розробки є REST API.

1.3 Принципи роботи REST API

REST – це архітектурний стиль побудови стандартизованих веб-інтерфейсів. Для того, щоб архітектуру можна було назвати RESTful, веб-застосунок повинен відповідати шести критеріям [5]:

- єдиний інтерфейс, який дозволяє однаково взаємодіяти з сервером на будь-якому пристрої;
- відсутність станів: сервер не зберігає стан застосунку, вся необхідна інформація зберігається в тілі і параметрах самого запиту клієнта;
- кешування;
- клієнт-серверна архітектура, яка дозволяє незалежний розвиток веб-застосунку з обох сторін;
- багаторівнева система застосунку;
- можливість для серверу забезпечувати код для виконання на стороні клаєнта.

Побудову архітектури REST API можна звести до чотирьох базових операцій (самих операцій може бути більше, однак 4 – це мінімум для того, щоб веб-сервіс можна було вважати повноцінним). Ці операції корелюють з методами HTTP-запитів:

- GET – отримання даних з сервера, зазвичай у форматі JSON.
- POST – додавання на сервер нових даних, викликає зміну стану на сервері окремих сутностей, може привести до побічних ефектів.
- PUT – модифікація даних на сервері, замінює дані певного ресурсу на нові, відправлені клієнтом.
- DELETE – видалення даних з сервера, видаляє вказаний клієнтом ресурс.

Схему роботи REST API зображено на рисунку 1.1.

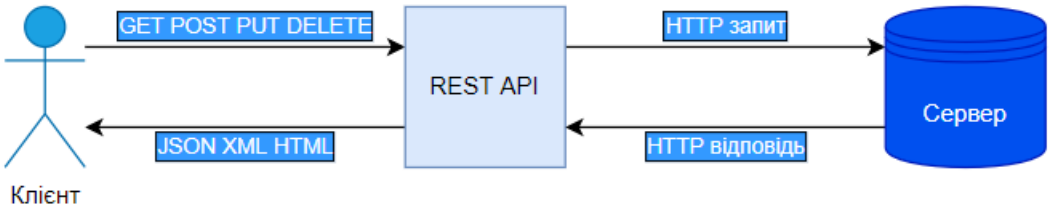

Рисунок 1.1 – Схема роботи REST API

REST в кожній відповіді повинен надавати клієнту метадані, які описують всю інформацію про поточне використання API. Наявність метаданих дозволяє у REST відрізнити клієнта від сервера, в результаті чого досягається принцип незалежного розвитку клієнта та сервера без ускладнення комунікації між ними. Існує декілька рівнів зрілості API в залежності від способу розрізняти клієнт та сервер (рисунок 1.2)

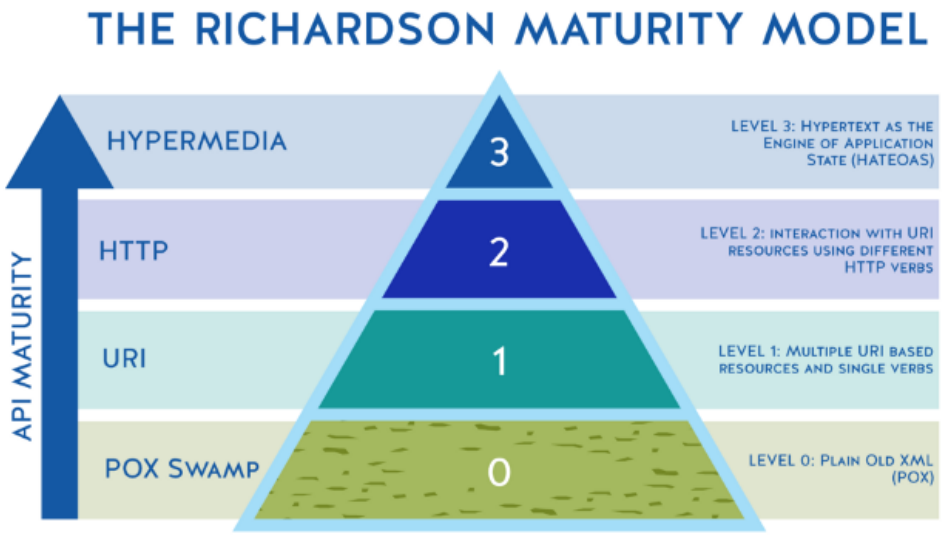

Рисунок 1.2 – Схема зрілості Річардсона [6], яка слугує орієнтиром для створення корисних та наповнених API. REST досягає найвищого рівня за рахунок метаданих.

Висновки до розділу

Отже, в першій частині дослідження мною було досліджено сучасні підходи до розроблення та впровадження веб-сервісів.

РОЗДІЛ 2. АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ І ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СИСТЕМИ

2.1 Бекенд фреймворк

Для реалізації бекенду мого веб-ресурсу потрібно створити власний REST API, де будуть виконуватися серверні операції, такі як оновлення даних, передача даних клієнту, фільтрація даних, адаптація даних. REST API можна створити власноруч з нуля, використовуючи будь-яку мову програмування, але для спрощення робити існують фреймворки та бібліотеки, які мають в собі велику кількість готових рішень.

Найпопулярнішими фреймворками для розробки REST API на сьогоднішній день є Django, Laravel, Ruby on Rails, ExpessJS, Flask, ASP .NET Core, Spring.

Такі фреймворки, як Spring, ASP .NET Core та Ruby on Rails частіше всього використовуються для великих enterprise-застосунків, надають дуже велику кількість готових рішень та модулів, але через це займають багато дискового простору та ресурсів, тому їх використання для створення простого REST API не є виправданим, адже більша частина їх функціоналу не буде застосована.

Враховуючи, що я маю більше всього досвіду роботи з мовою програмування JavaScript, серед інших фреймворків найбільш зручним буде використання ExpressJS на базі NodeJS.

ExpressJS – це простий та гнучкий фреймворк розробки веб-застосунків на базі NodeJS. Він надає велику кількість функцій, які спрощують розробку веб-сервісів. Приклади функцій [7]:

- get(path, callback) – реалізація методу GET за певним шляхом з виконанням коду всередині callback-функції;
- post(path, callback) – реалізація методу POST за певним шляхом з виконанням коду всередині callback-функції;

- use(callback) – функція проміжної обробки, яка виконує певні команди при кожному запиті на сервер;
- listen(port, host, callback) – задає порт і хост, на якому потрібно виконувати застосунок, а також опціональний колбек, який виконується при старті роботи сервера.

ExpressJS використовує для роботи локально обгортку у вигляді NodeJS, фреймворка, який дозволяє запускати код JavaScript на сервері, а не лише у браузері (JavaScript був створений виключно для роботи у браузері з використанням движка V8).

NodeJS має багато відмінностей від звичайних засобів, серед яких є [8]:

- однопотокова асинхронна модель для виконання запитів
- неблокуючий ввід та вивід
- система модулів CommonJS
- движок JS V8 від Google

Подібна структура робить NodeJS ідеальним рішенням для роботи над невеликими застосунками, в яких іде постійне спілкування клієнта та сервера. Завдяки одному неблокуючому потоку клієнт може зробити відразу декілька запитів і не чекати розблокування потоку для продовження роботи, а розробник не повинен займатися розбиванням задач на потоки.

Зокрема великим плюсом є те, що сервер може асинхронно виконувати власні методи, не зупиняючи роботу застосунку для користувача, що дозволяє уникнути розподілу серверу на велику кількість мікросервісів.

Це, в скупченні з методами для розробки сервісної архітектури ExpressJS робить стек NodeJS + ExpressJS ідеальним для розробки мого застосунку.

2.2 Вибір бази даних

Необхідністю для роботи мого застосунку є база даних, де буде зберігатися необхідна для клієнта інформація, в моєму випадку це дані по епідемії SARS-Cov-2. Існують два види баз даних, реляційні та нереляційні.

Дані у реляційних базах даних організовані у виді таблиць, між якими існують зв’язки по певних полях. Організація великої кількості даних у такому вигляді може привести до проблем з швидкістю обробки запитів, тому були розроблені так звані «нормальні форми», які задають правила для побудови та групування таблиць. У нижче наведеному списку представлені нормальні форми по ступені збільшення їх досконалості (крім ДКНФ, це форма зі специфічними випадками для використанні) [9]:

- Перша нормальна форма
- Друга нормальна форма
- Третя нормальна форма
- Четверта нормальна форма
- П'ята нормальна форма
- Доменно-ключова нормальна форма.

До реляційних відносяться такі системи керування базами даних, як PostgreSQL, MySQL, Microsoft SQL Server, Oracle Database etc. Для виконання вони використовують декларативну мову програмування SQL.

Нереляційні бази даних – це бази даних, які використовують відмінні від використання таблиць-зв’язків механізми зберігання та роботи з даними. Серед них можна виділити такі класи моделей даних [10]:

- Ключ-значення: Aerospike, ArangoDB, Redis;
- Стовпчик: Cassandra, Vertica, Druid;
- Граф: OrientDB, ArangoDB, MarkLogic;
- Документ: ArangoDB, MongoDB, OrientDB;
- Мульти-модель: MarkLogic, ArangoDB, OrientDB.

Як видно, класифікація не є ідеальною, адже багато систем керування повторюються у різних категоріях.

Вибір системи керування базою даних для мого веб-сервісу не був складним, адже для роботи у середовищі NodeJS існує бібліотека mongoose, яка дозволяє просто та швидко працювати з MongoDB. Розглянемо її детальніше.

MongoDB – система керування базами даних, яка відноситься до документо-орієнтованих. Це означає, що дані в ній зберігаються у форматі документів, зокрема для MongoDB, ці документи мають JSON-подібну структуру. MogoDB є збалансованим вибором між швидкими, але обмеженими системами формату ключ-значенні, та реляційними базами даних.

До переваг MongoDB можна віднести зручний формат зберігання, гнучку мову запитів, автоматичне створення індексів, журнал змін.

Приклад документу зображено на рисунку 2.1:

```
{
    "name" : "John",
    "address" : {
      "street" : "Stepana Bandery 9",
      "city" : "Lviv",
      "state" : "Lvivska oblast"
    }
 }
```

Рисунок 2.1 – Документ MongoDB

Приклад запиту з використанням mongoose [11] наведено на рисунку 2.2:

```
db.users.find({"address.state" : "Lvivska oblast"})
db.meal.insert({"vegetable" : ["carrot", "potato",
"tomato"]})
db.meal.delete({"vegetable" : "tomato"})
```

Рисунок 2.2 – Приклад роботи mongoose

2.3 Single-page application

Окрім серверної частини, мій застосунок повинен мати також клієнтську частину. Ця частина буде включати в себе роботу з такими головними складовими фронтенд розробки, як HTML, CSS, JavaScript.

Враховуючи, що клієнт буде постійно отримувати нові дані, мені не підходить стандартна модель фроненду, коли до користувача приходять цілі сторінки, адже це буде призводити до постійного перезавантаження вікна браузеру та втрати даних після кожної взаємодії з ресурсом. Це означає, що потрібно використати іншу модель, коли дані будуть приходити частинами і буде перезавантаження лише окремих частин застосунку, а не всього ресурсу. Такою моделлю є SPA.

SPA – це фактично робота користувача з однією сторінкою, в якій змінюються лише певні елементи, що дозволяє динамічно переписувати окремі частини сторінки, а не кожного разу отримувати нову [12]. До прикладів ресурсів, які користуються цим підходом відносяться такі веб-застосунки, як Facebook, Trello, YouTube, Gmail etc.

SPA дуже доцільно використовує модель REST та маршрутизацію, адже доступ до контенту виконується за допомогою різних параметрів у адресному рядку, що дозволяє отримати доступ до певної статичної частини ресурсу за простим запитом. Це дозволяє користувачам ділитися одне з одним інформацією, хоч і ресурс все одно являє собою одну сторінку.

SPA зазвичай працюють швидше багатосторінкових сайтів, бо динамічна зміна контенту відбувається з більшою швидкістю, ніж обмін повноцінними сторінками. Виключенням може бути стартова розмітка сторінки, адже там досить часто зберігається скрипт всього фреймворку, який дозволяє функціонувати SPA.

Розробити SPA з нуля цілком можливо, але для спрощення розробки були створені різні фреймворки. Найпопулярнішими є Angular, Vue.js та React.

2.3.1 Angular

По праву фреймворком серед вище перелічених засобів можна назвати тільки Angular. Він включає в себе велику кількість готових модулів, зокрема тих, які використовуються для роботи з сервером, маршрутизації, контролю станів, створення форм, передачі даних між компонентами тощо.

Переваги Angular над іншими рішеннями:

- Двостороння робота з даними. Angular був розроблений з використанням моделі Model-View-Controller, що дозволяє синхронізувати роботу Model та View.
- Директиви: директиви дозволяють розробникам призначати особливу поведінку DOM, за допомогою чого створюється динамічний вміст HTML.
- Ін'єкція залежностей [13]: залежності визначають, як різні фрагменти коду взаємодіють один з одним і як зміни в одному компоненті впливають на інші. Зазвичай залежності визначаються безпосередньо в самих компонентах, тому кожна зміна залежності вимагає також зміни компонентів. З Angular можна використовувати «інжектори», які визначають залежності як зовнішні елементи, що відокремлюють компоненти від їхніх залежностей. Ін’єкція залежностей зробила компоненти більш придатними для повторного використання.
- Ком’юніті: Angular є одним з найпопулярніших фреймворків, тому існує велика кількість документації, навчальних матеріалів, дискурсів та готових рішень.
- Підтримка від Google [14] та Microsoft
- Підтримка RxJS прямо з «коробки»: ця бібліотека спрощує роботу з асинхронним кодом та колбек-функціями за допомогою реактивного підходу
- Шаблони HTML
- Підтримка TypeScript з «коробки»

- Ліниве завантаження
- Підтримка i18n

Недоліки Angular:

- Швидкодія: серед інших рішень Angular дає найбільший функціонал, але це приводить до того, що сам фрейморк досить багато важить і сповільнює роботу ресурсу.
- Складність: знову впираючись у великі можливості, цей фреймворк досить складний для вивчення порівняно з іншими, адже надає так багато готових рішень, що новачкам буде дуже просто заплутатися [15].
- Немає рішення для керування станами: необхідно додатково вивчати та вручну додавати бібліотеки, які допомагають працювати зі станами, наприклад Redux, Mobx
- Обмеженість вибору модулів: через принцип надання всього з «коробки» є тільки невеликий простір для кастомізації проекту за допомогою сторонніх модулів.
- Велика кількість несумісних версій
- Зменшення популярності
- Лімітовані можливості для CEO

Підсумовуючи всі зазначені вище переваги та недоліки, я прийшов до висновку, що мені не підходить Angular, адже більша частина його функцій мені буде зайвою, а та частина, яка потрібна, у інших рішеннях виконана набагато простіше і зручніше.

Цей фреймворк ідеально підходить для створення великих enterprise-застосунків, які будуть використовувати більшу частину наданого функціоналу, не вимагатимуть значної оптимізації, матимуть велику та досвідчену команду для розробки проекту. В моєму випадку, жоден з цих критеріїв не відповідає дійсності.

2.3.2 Vue.js

Серед вказаних мною рішень Vue.js є наймолодшим і довгий час він не мав ніякої підтримки від корпорацій, тому до нього ставилися з насторогою, але при цьому він все ж зміг конкурувати з React та Angular, хоч і лише в малих проектах.

Vue.js – це скоріше бібліотека, ніж фреймворк, тому що він не надає велику кількість функціоналу в базовому пакеті, однак це виправляється тим, що під нього підлаштована велика кількість допоміжних модулів, які додаються по мірі необхідності в проекті, що робить веб-застосунки, побудовані на Vue.js легкими та швидкими.

Однак досить часто навіть досвідчені розробники можуть додати Vue.js до фреймворків, і це не дивно, адже він, хоч і за допомогою сторонніх пакетів, може задовольнити всі потреби, необхідні для створення якісного SPA. Все вищесказане також є правдивим і для React.

У Vue.js є ще одна відмінність від інших популярних рішень – його розробкою займається лише одна людина, Еван Ю. Він займається цим проектом та отримує фінансування від інших людей на платформі Patreon, а також його спонсорами виступають деякі компанії, однак кінцевий продукт все одно належить Евану. Це створює певні обмеження, особливо у enterprise-секторі, адже подібна модель є дуже ненадійною, але водночас це дає творчу свободу та вільний графік творцю, що несе в собі переваги для ентузіастів та стартапів. Саме вони є основними користувачами цього фреймворку.

Переваги Vue.js [16]:

- Невеликий розмір
- Наявність віртуального DOM
- Оптимізація та швидкодія
- Реактивна двостороння робота з даними
- Комопнування всього коду компонента в один файл, що відкриває такі додаткові переваги:

    - Простота повторного використання компонентів
    - Читабельність коду
    - Зручність тестування
- Простота інтеграція та гнучкість в розробці
- Підтримка ком’юніті
- Простота вивчення
- Велика екосистема

Недоліки Vue.js:

- Складність реактивного програмування
- Відсутність підтримки для enterprise-проектів
- Ризик надвеликої гнучкості
- Лімітована кількість ресурсів
- Недостатня кількість досвідчених розробників

Виходячи з цього списку переваг та недоліків, можна дійти до висновку, що в цілому Vue.js задовольняє потреби мого проекту, однак якщо розібратися у всьому детальніше, то виявиться, що у розробки на Vue.js є одна велика проблема – недостатня кількість модулів. Для мого веб-застосунку вкрай необхідно мати бібліотеку, яка спрощує роботу з інтерактивною картою, статистикою та графіками. І хоча на Vue.js є деякі рішення, які частково задовольняють мої потреби, у інших фреймворків набагато більший вибір.

Можливості Vue.js ідеально підходять для простих проектів, де вкрай важлива оптимізація та краса інтерфейсу, в цій області у даного фреймворку немає рівних. Однак для більш специфічних застосунків краще будуть працювати інші рішення.

Також наостанок можна відзначити простоту інтеграції проектів на Vue.js в готові проекти, яким не вистачає певної інтерактивності. Однак ця перевага теж не є потрібною для моєї задачі, адже весь веб-застосунок буде робитися з нуля.

2.3.3 React

На даний момент React є найпопулярнішим рішенням для розробки фронтенд частини веб-застосунків. React, на час створення, пропонував геть новий підхід до візуалізації об’єктів та створення динамічного контенту на сайтах.

Компоненти цього інструменту були розроблені Facebook. Він був запущений як інструмент JavaScript з відкритим вихідним кодом в 2013 році. В даний час React випереджає своїх основних конкурентів, таких як Angular і Vue.js.

На основі React побудовані такі відомі сервіси, як Netflix, WhatApp, eBay, Facebook, Instagram etc. Таке широке використання вказує на популярність та гнучкість даної бібліотеки.

Щоб почати розроблювати веб-застосунки на React, достатньо вчити його лише декілька днів, якщо ви до цього вже мали досвід з JavaScript. Однак слід зазначити, що тут мається на увазі лише основна бібліотека. Чим більший проект, тим більша кількість різних додаткових модулів потрібно знати, і в якийсь момент React перестає бути простим у вивченні

Переваги React:

- Простота у вивчанні та використанні
- Зручне повторне використання компонентів
- Чудова оптимізація
- Побудова динамічного HTML дуже проста через наявність JSX
- Підтримка великої кількості сторонніх модулів (Redux, React Dev Tools. React Router etc)
- Односторонній потік дани: хоча двосторонній потік даних дає гнучкість в розробці, він також порушує стабільність застосунку. Завдяки рішенню React відмовитися від нього компоненти-нащадки на впливають на роботу батьківських компонентів.

- Ком’юніті
- Просте SEO
- Простота тестування
- Віртуальний DOM: являє собою віртуальну копію реального DOM-дерева, через що дозволяє маніпулювати та змінювати навіть найменші компоненти і лише потім порівнювати результат з реальним DOM та вносити зміни [17]
- React Native: фреймворк для побудови мобільних застосунків на базі React. Через це портувати веб-застосунок у формат мобільного дуже просто для розробників, які мають досвід з React.

Недоліки React:

- Занадто стрімкі темпи розробки [18]
- Погана документація в порівнянні з конкурентами
- JSX іноді може бути перешкодою [19]: тут досить спірно, адже для багатьох JSX є перевагою, однак слід відзначити, що через нього розробники часто жертвують гнучкістю заради простоти, і тому на багатьох проектах від нього відмовляються в користь простого HTML.

Підсумовуючи інформацію про переваги та недоліки даної бібліотеки, як прийшов до висновку, що вона краще всього підходить для мого проекту. React зручно взаємодіє з різними модулями, не настільки важкий та складний, як Angular, та надає значно більше можливостей, ніж Vue.js. Що стосується його недоліків, то в межах мого завдання вони не грають визначну роль, хоча відсутність гарно структурованої документації може стати перешкодою на перших етапах розробки. Однак цей недолік компенсується великою кількістю інформації, навчальних матеріалів та дискурсів від ком’юніті, яке у React найбільше серед всіх фронтенд рішень [20].

2.4 Вибір протоколу реалізації

В першому розділі мною було розглянуто різні архітектурні стилі реалізації веб-сервісів та обрано REST API. На початку другого розділу я описав інструментарій, за допомогою якого планую створювати власний бекенд для веб-застосунку замість того, щоб використовувати готовий. Однак, щоб зібрати дані по статистиці захворюваності та вакцинації SARS-Cov-2, все ж доведеться використати сторонні ресурси.

Я провів дослідження у всесвітній мережі та знайшов велику кількість готових REST API, які пропонують готові дані, однак їх функціоналу для реалізації мого задуму виявилося недостатньо, тому, після деякого часу тестування та перевірки можливостей різних сервісів, я прийшов до висновку, що мені необхідні дані з двох веб-сервісів – Coronavirus.app та disease.sh.

Обом ресурсам притаманні такі риси:

- регулярне оновлення даних;
- відправка у форматі json;
- зручна і детально написана документація по використанню;
- можливість імпортувати запити в десктопний застосунок Postman;
- відсутність ліміту відправки запитів;
- можливість сортувати отримані дані окремо по країнам;
- можливість отримувати інформацію за минулі дні, аж до початку пандемії;

Стосовно Coronavirus.app, то у нього є такі обмеження:

- платний доступ, мінімальна сума підписки 3 долари
- мала кількість запитів, корисних лише два

У disease.sh в свою чергу існують такі проблеми:

- погана структуризація даних
- перенасичення функціоналу

- відсутність можливості отримати глобальні дані про вакцинацію
- неочевидні дані за минулі дні.

Як видно, у кожного з цих ресурсів є свої недоліки, тому мені довелося об’єднати їх функціонал разом, щоб досягнути задовільного результату для реалізації мого функціоналу.

Інші REST API також мали свої переваги, але для реалізації мого веб застосунку функціонал цих двох ресурсів виявився найбільш повним. Хоча слід зазначити, що по деяким пунктам інші REST API все ж виявилися кращими, ніж обраний мною ресурс. Наприклад, CoronaNinja має більш точні дані, але перестав оновлюватися рік тому, а Coronavirus Smartable, хоч і є повністю безкоштовним, не надає достатню кількість даних для розробки мого веб-застосунку.

Висновки до розділу

Отже, в другій частині мого дослідження було проаналізовано архітектурні рішення, які наразі використовуються для побудови веб-застосунків та обрано програмні засоби для реалізації як серверної частини веб-застосунку (ExpressJS), так і клієнтської (ReactJS)

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ДЛЯ МОНІТОРИНГУ СИТУАЦІЇ З COVID-19

3.1 Структура бази даних

Для початку потрібно підготувати схему документу в MongoDB. Створимо базу даних під назвою covid. Всередині неї створимо дві колекції: datas та histories.

Колекція datas буде мати глобальну інформацію про статистику захворюваності, смертності та вакцинації у країнах світу, а також буде мати статистику за деякі проміжки часу: за місяць, за півроку та за рік.

Структура колекції datas (рисунок 3.1)

```
{
  "_id": "Унікальний ідентифікатор",
  "id": "Унікальний ідентифікатор від coronavirus.app (якщо дані беруться звідти)",
  "country": "Назва країни",
  "province": "Назва провінції (якщо доступно)",
  "__v": "Версія документу",
  "coordinates":[
    {
      "latitude": "Координати широти",
      "longitude": "Координати довготи"
    }
  ],
  "stats":[
    {
      "confirmed": "Загальна кількість інфікованих",
      "deaths": "Загальна кількість смертей",
      "recovered": "Загальна кількість одужань",
      "vaccine": "Загальна кількість вакцинацій"
    }
  ],
  "monthStats":[...],
  "halfyearStats":[...],
  "yearStats":[...]
}
```

Рисунок 3.1 – Приклад документу з колекції datas

Колекція histories буде мати в собі дані історії протікання епідемії.

Структура колекції histories (рисунок 3.2):

```
{
  "_id": "Унікальний ідентифікатор",
  "id": "Унікальний ідентифікатор від coronavirus.app (якщо дані беруться звідти)",
  "__v": "Версія документу",
  "country": "Назва країни",
  "province": "Назва провінції (якщо доступно)",
  "stats":[
    {
      "confirmed": "Загальна кількість інфікованих",
      "deaths": "Загальна кількість смертей",
      "recovered": "Загальна кількість одужань",
      "vaccine": "Загальна кількість вакцинацій"
    }
  ],
  "timeline": [
    {
      "day": "Дата статистичних даних",
      "confirmed": "Загальна кількість інфікованих на даний момент",
      "deaths": "Загальна кількість смертей на даний момент",
      "recovered": "Загальна кількість одужань на даний момент",
      "vaccine": "Загальна кількість вакцинацій на даний момент",
    },
    ...,
    {"day": "Дата початку епідемії статистичних даних"...}
  ]
}
```

Рисунок 3.2 – Приклад документу з колекції histories

Після того, як була визначена структура бази даних, можна переходити до наступного етапу – розробки серверу веб-застосунку.

3.2 Розробка бекенд-частини веб-застосунку

Express.js працює за патерном MVC, який розшифровується як Model-View-Controller [21]. Спрощена схема роботи цього патерну в реалізації Express.js зображено на рис 3.3.

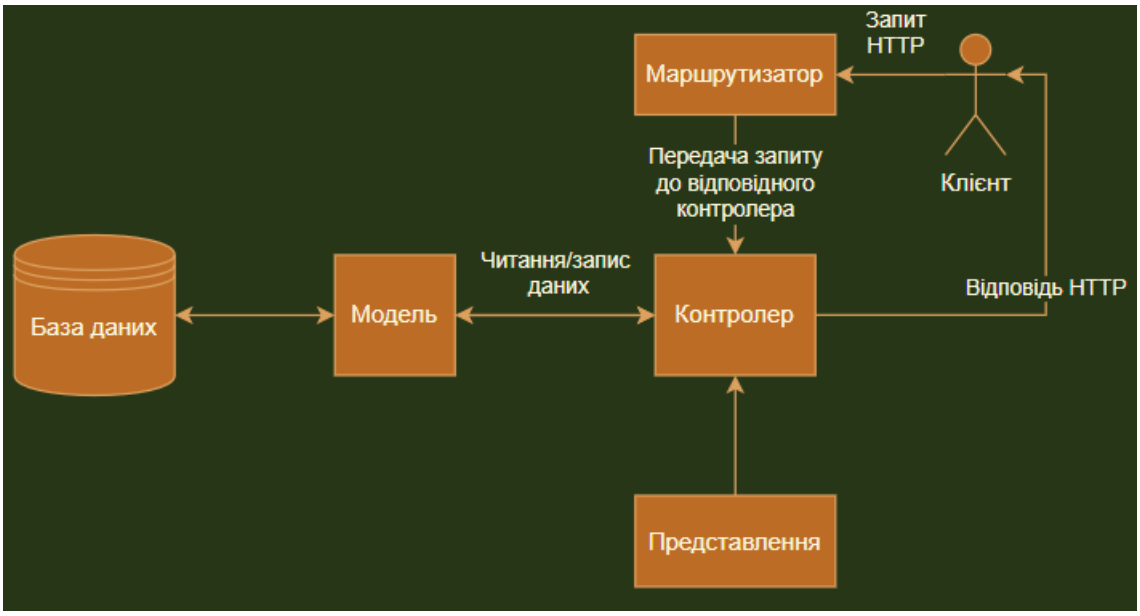

Рисунок 3.3. – Спрощена схема роботи патерну MVC

В нашому випадку частина представлення буде реалізована у фронтенд-частині веб-сервісу, тому на сервері нам треба буде реалізувати лише контролер, моделі та маршрутизацію.

3.2.1 Реалізація моделей

Для початку потрібно за допомогою засобів пакету mongoose створити моделі, які будуть працювати з визначеною структурою документу MongoDB.

Модель Data представлена в ДОДАТКУ А.

Модель History представлена в ДОДАТКУ Б.

3.2.2 Створення маршрутів

Наступним етапом буде реалізація маршрутів, по яким клієнт зможе отримати дані, необхідні для роботи з веб-застосунком. Так як не передбачено ніяких змін даних самим клієнтом, для роботи веб-застосунку буде достатньо лише методу GET.

Після дослідження даних, які надходять з інших API, а також перегляду аналогів ресурсу, який я розробляю, я дійшов до висновку, що кінцевому користувачу буде достатньо трьох маршрутів:

1. /v1/all-data, де v1 – версія API, a all-data – шлях до вибірки даних. Цей маршрут буде повертати найбільш повні дані по всім країнам та їх провінціям, основна частина даних буде надходити з disease.sh, тому тут існує проблема з даними за певні відрізки часу
2. /v1/specific-data, де v1 – версія API, a specific-data – шлях до вибірки даних. Цей маршрут буде повертати дані по країнам та їх провінціям, але зі статистикою по відрізкам часу. Дані будуть надходити з coronavirus.app, де немає такого розподілу по провінціях, як у disease.sh, тому кількість даних буде дещо менша
3. /v1/history, де v1 – версія API, a history – шлях до вибірки даних. Цей маршрут буде повертати статистичну історію захворюваності по конкретній країні. Країна буде визначатися по id, який користувач передасть як параметр в адресному рядку.

Реалізація маршрутів (рисунку 3.4):

```
router.get( path: '/v1/all-data', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) => {
    Data.find( filter: {'id': null}).lean().exec( callback: (err : CallbackError , data : (LeanDocument<any>)[] | any[] ) => {
        res.setHeader( name: 'Access-Control-Allow-Origin', value: 'https://covid-map-nk.netlify.app')
        return res.end(JSON.stringify(data))
    })
})

router.get( path: '/v1/specific-data', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) => {
    Data.where( path: 'id').ne( val: null).lean().exec( callback: (err : CallbackError , data : (LeanDocument<any>)[] | any[] ) => {
        res.setHeader( name: 'Access-Control-Allow-Origin', value: 'https://covid-map-nk.netlify.app')
        return res.end(JSON.stringify(data))
    })
})

router.get( path: '/v1/history', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) => {
    History.findOne( filter: { id : req.query.countryId } ).lean().exec( callback: (err : CallbackError , data) => {
        res.setHeader( name: 'Access-Control-Allow-Origin', value: 'https://covid-map-nk.netlify.app')
        return res.end(JSON.stringify(data))
    })
})
```

Рисунок 3.4. – Реалізація маршрутизації

У всіх маршрутах необхідно виставити заголовок, який буде давати доступ до цього маршруту з майбутньої фронтенд-частини веб-застосунку.

3.2.2 Адаптація даних

Так як дані приходять відразу з двох API, формати яких відрізняються одне від одного, необхідно перетворити їх у необхідний нам вигляд, використовуючи патерн адаптер [22]. Для початку визначимося з тими запитами на сторонні API, які нам потрібні, та оглянемо їх структуру.

https://coronavirus.app/get-places

Цей запит повертає загальні дані по всіх доступним країнам та їх провінціям, якщо такі наявні. Формат даних (рисунок 3.5):


```
"data": [
{
  "vaccinated": 54146,
  "latitude": 49.4482,
  "id": "0FOJefeLxQYXQ1CnJm9o",
  "dead": 17,
  "pop": 63196,
  "country": "GG",
  "longitude": -2.5895,
  "name": "Guernsey",
  "lastUpdated": "2022-05-15T01:09:54.515Z",
  "infected": 13503,
  "recovered": 12413,
  "sick": 1073,
  "invisible": false
},...]
```


Рисунок 3.5. – Приклад даних, що повертає сервіс coronavirus.app для загальної статистики

Нам з нього необхідні поля latitude, longitude, dead, name, infected, recovered, vaccinated, country, id.

https://coronavirus.app/get-history?id={id}

Цей запит повертає історію захворюваності по конкретній країні, приймає як параметр id країни, який ми отримали через попередній запит.

Формат даних (рисунок 3.6):

Рисунок 3.6 – Приклад даних, що повертає сервіс coronavirus.app для окремої країни

Нам з нього необхідні поля latitude, longitude, day, dead, name, infected, recovered, vaccinated, country.

https://disease.sh/v3/covid-19/jhucsse

Цей запит повертає загальну статистику по захворюваності в різних країнах та їх провінціях.

Формат даних (рисунок 3.7):

```
[
  {
    "country": "Afghanistan",
    "county": null,
    "updatedAt": "2022-05-15 04:20:53",
    "stats": {
      "confirmed": 179242,
      "deaths": 7687,
      "recovered": null
    },
    "coordinates": {
      "latitude": "33.93911",
      "longitude": "67.709953"
    },
    "province": null
  },...]
```


Рисунок 3.7 – Приклад даних, що повертає сервіс disease.sh по загальній статистиці

З цього запиту нам необхідні поля country, province, coordinates.latitude, coordinates.longitude, stats.confirmed, stats.deaths. Однак є проблема, і вона полягає в тому, що в цьому запиті відсутні дані по вакцинації, тому потрібно їх додати на сервері, користуючись іншим запитом.

https://disease.sh/v3/covid-19/vaccine/coverage/countries/?lastdays=1

Цей запит повертає виключно дані про вакцинацію по країнах, без провінцій. Він приймає параметр, який визначає, за який проміжок часу повертати дані. В моєму випадку необхідний лише останній день.

Формат даних (рисунок 3.8):

```
[
  {
    "country": "Afghanistan",
    "timeline": {
      "5/15/22": 6029737
    }
  },...]
```


Рисунок 3.8 – Приклад даних, що повертає сервіс disease.sh по історії вакцинації

Звідси нам достатньо отримати число вакцин з масиву timeline.

Схему адаптації даних можна побачити на рисунок 3.9.

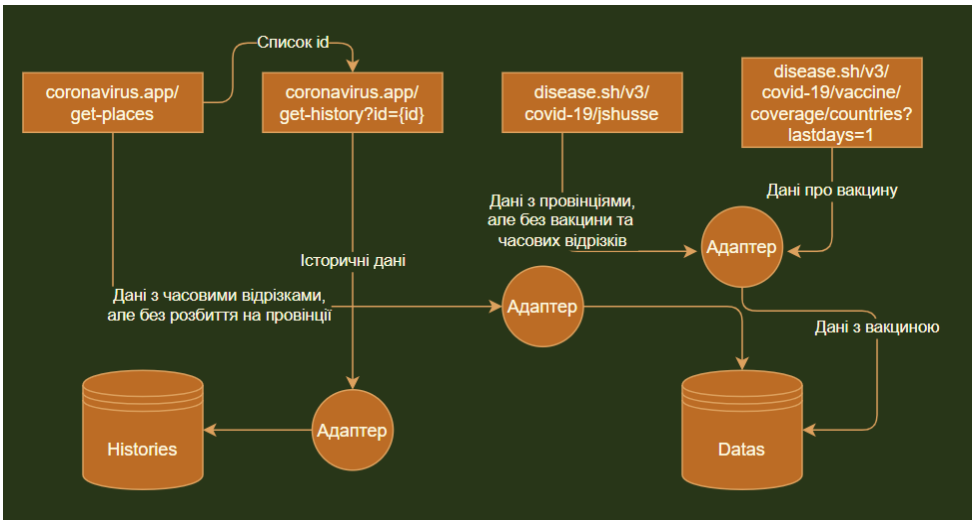

Рисунок 3.9 – Схема адаптації даних, отриманих зі сторонніх API

Окремо слід зазначити адаптації даних за часовими відрізками, адже для цього виконуються додаткові операції на сервері. Для того, щоб отримати дані за місяць, з масиву history беруться дані за останній день, та за день, що був 30 днів тому, та робиться операція віднімання. Так ж саме відбувається зі статистикою за півроку та рік, але там віднімається дні, які були 180 та 365 діб тому відповідно. Для обробки цих даних була створена окрема функція createPeriodicStats(history) (рисунок 3.10), яка приймає масив history та повертає об’єкт з адаптованими даними.

```
async function createPeriodicStats(history){
    return {
        month: {
            confirmed: history[0].infected - history[30].infected,
            deaths: history[0].dead - history[30].dead,
            recovered: history[0].recovered - history[30].recovered,
            vaccine: history[0].vaccinated - history[30].vaccinated
        },
        halfyear: {
            confirmed: history[0].infected - history[180].infected,
            deaths: history[0].dead - history[180].dead,
            recovered: history[0].recovered - history[180].recovered,
            vaccine: history[0].vaccinated - history[180].vaccinated
        },
        year: {
            confirmed: history[0].infected - history[365].infected,
            deaths: history[0].dead - history[365].dead,
            recovered: history[0].recovered - history[365].recovered,
            vaccine: history[0].vaccinated - history[365].vaccinated
        }
    }
}
```

Рисунок 3.10 – Код функції createPeriodicStats

Також потрібно звернути увагу на процес створення даних за всю історію. Жоден з цих API не надає глобальні дані про статистику епідемії, тому його слід зробити на сервері. Для початку було створено нову структуру даних (рисунок 3.11):

```
const globalHistory = {
    country: 'World',
    province: 'none',
    id: '1aVVar2d0',
    updatedAt: Date.now(),
    stats: {
        confirmed: 0,
        deaths: 0,
        recovered: 0,
        vaccine: 0
    },
    history: []
}
```

Рисунок 3.11 – Змінна globalHistory для заповнення глобальної статистики

Потім до статистики додаються загальні дані по кожній країні (рисунок 3.12):

```
globalHistory.stats.confirmed += country.infected
globalHistory.stats.deaths += country.dead
globalHistory.stats.vaccine += country.vaccinated
globalHistory.stats.recovered += country.recovered
```

Рисунок 3.12 – Додавання статистики по країнам в глобальну історію

Далі необхідно додати історію. Це робиться за допомогою методу updateWorldStats(history), де history – це історична статистика поточної країни. Далі необхідно зробити додавання даних, орієнтуючись на поле day. Якщо в масиві історії вже є день, то ми просто додаємо туди дані, а якщо його немає, то створюємо новий об’єкт (рисунок 3.13).

```
async function updateWorldStats(history){
    history.map(day => {
        if(globalHistory.history.find(d => d.day === day.day)){
            const index = globalHistory.history.findIndex(d => d.day === day.day)
            globalHistory.history[index].confirmed += day.confirmed
            globalHistory.history[index].deaths += day.deaths
            globalHistory.history[index].vaccine += day.vaccine
            globalHistory.history[index].recovered += day.recovered
        }
        else{
            globalHistory.history.push(day)
        }
    })
}
```

Рисунок 3.13 – Додавання історії захворюваності в глобальну історію

В результаті ми отримаємо статистику по епідемії у всьому світі разом з історично статистикою.

3.2.3 Оновлення даних

Після реалізації всього функціоналу ми отримаємо наповнену базу даних, три маршрути, по яким клієнт зможе отримувати необхідні дані, а також функціонал по адаптації даних. Однак, залишилася ще одна проблема: після запуску сервера дані будуть додані лише один раз, через що для клієнта будуть надіслані тим більш неактуальна статистика, чим більше часу пройде з запуску серверу. Щоб виправити цю проблему, потрібно додати backgoundWorker –

функцію, яка буде запускати оновлення даних через певний проміжок часу. Для цього скористаємося модулем cron.

Згідно з визначення, cron – це утиліта, яка дає змогу користувачам виконувати скрипти та команди автоматично в заданий час. Для задання цього часу використовується спеціальна таблиця під назвою crontab, де кожний стовпчик відповідає за свою одиницю виміру часу [23]. Таблиця зображена на рисунку 3.14:

```
# .---------------- хвилина (0 - 59)
# |  .------------- година (0 - 23)
# |  |  .---------- день місяця (1 - 31)
# |  |  |  .------- місяць (1 - 12) АБО jan,feb,mar,apr ...
# |  |  |  |  .---- день тижня (0 - 6) (неділя=0 чи 7) АБО
sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
  *  *  *  *  *  виконувана команда
```

Рисунок 3.14. – Таблиця визначення періодичності cron

У моєму випадку необхідно, щоб дані оновлювалися кожну годину, це є збалансований проміжок часу, в який сервер не буде перевантажуватися, але при цьому давати актуальні дані. Для цього підходить такий вираз cron: 0 * * * *. Він означає, що скрипт повинен запускатися кожну нульову хвилину наступних годин. Помістимо всередину цього воркера функції адаптації даних (рисунок 3.15):

```
cron.schedule( expression: '0 * * * *', func: async () => {
    console.log('in cron')
    let covGetPlaces = await getApi( url: 'https://coronavirus.app/get-places', params: { headers })
    await insertHistoryData(covGetPlaces.data, headers)
    await insertPeriodicData(covGetPlaces.data, headers)
    let countryData = await getApi( url: 'https://disease.sh/v3/covid-19/jhucsse')
    let vaccineTotal = await getApi( url: `https://disease.sh/v3/covid-19/vaccine/coverage/countries/?lastdays=1`)
    await insertTotalData(countryData, vaccineTotal)
})
```

Рисунок 3.15 – Реалізація автоматизації оновлення даних за допомогою cron

3.2.4 Деплой серверу

Останнім етапом у створенні серверної частини веб-застосунку є його деплой у мережу інтернет. Це необхідно для того, щоб він зберігався на

віддаленому сервері і не залежав від моєї локальної машини. Таким чином, доступ до серверу завжди буде надійним та швидким.

Для деплою застосунків на базі NodeJS ідеально підходить хмарна платформа Heroku. Вона надає можливість безкоштовно розгортувати в своєму середовищі прості веб-застосунки з невеликим трафіком.

Для розгортування застосунку необхідно встановити дві утиліти, Git та Heroku CLI (Command Line Interface). За допомогою CLI можна налаштувати поточний локальний проект на зв’язок з сервісами Heroku. Це робиться за допомогою команди heroku create. Ця команда створить пустий проект на серверах Heroku та одночасно з тим проініціалізує там пустий гіт-репозиторій.

Далі все що буде необхідно, так це додати туди свій код. Це робиться за допомогою команди git push heroku main. Однак перед цим також необхідно створити локальний репозиторій. Щоб це зробити, потрібно виконати команду git init. Потім додати файли до коміту за допомогою git commit -am «Commit name» [24]. Перед цим рекомендується виключити з завантажуваних файлів папку node_modules, адже вона є досить важкою та необов’язковою для роботи серверу. Щоб це зробити, потрібно створити файл .gitignore в корені локального репозиторію та додати туди такі правила:

```
/node_modules
npm-debug.log
.idea
.DS_Store
/*.env
```

Це виключить всі непотрібні файли з коміту.

Після виконання всіх команд Heroku створить та запустить сервер, надавши йому доменне ім’я, за яким можна буде отримати доступ до ресурсу. Адреса серверу: https://calm-headland-93075.herokuapp.com/

3.3 Розробка фронтенд частини веб-застосунку

Для створення React-застосунку необхідно встановити сам react за допомогою команди npx create-react-app {назва} [25]. Після цього npx створить файлову структура та завантажить модулі, потрібні для роботи веб-застосунку.

3.3.1 Структура інтерфейсу веб-застосунку

На даний момент є багато підходів до розробки інтерфейсу для веб-застосунків. Однак, для цього проекту не є основним мати професійний дизайн з безліччю красивих і плавних анімації. Основна задача — якомога швидше надати необхідну користувачу інформацію. Ось чому я вирішив підійти до розробки інтерфейсу з функціональної і практичної точки зору. Інтерфейс складається з таких модулів:

- навігація
- сторінка з картою (рисунок 3.16)
    - карта
    - кнопки керування
- сторінка зі статистикою (рисунок 3.17)
    - назва країни
    - загальні статистичні дані
    - графік прогресу епідемії
- сторінка з інформацією (рисунок 3.18)
    - корисна інформація про SARS-Cov-2

Перед самою розробкою варто створити приблизний макет інтерфейсу для кожної сторінки:

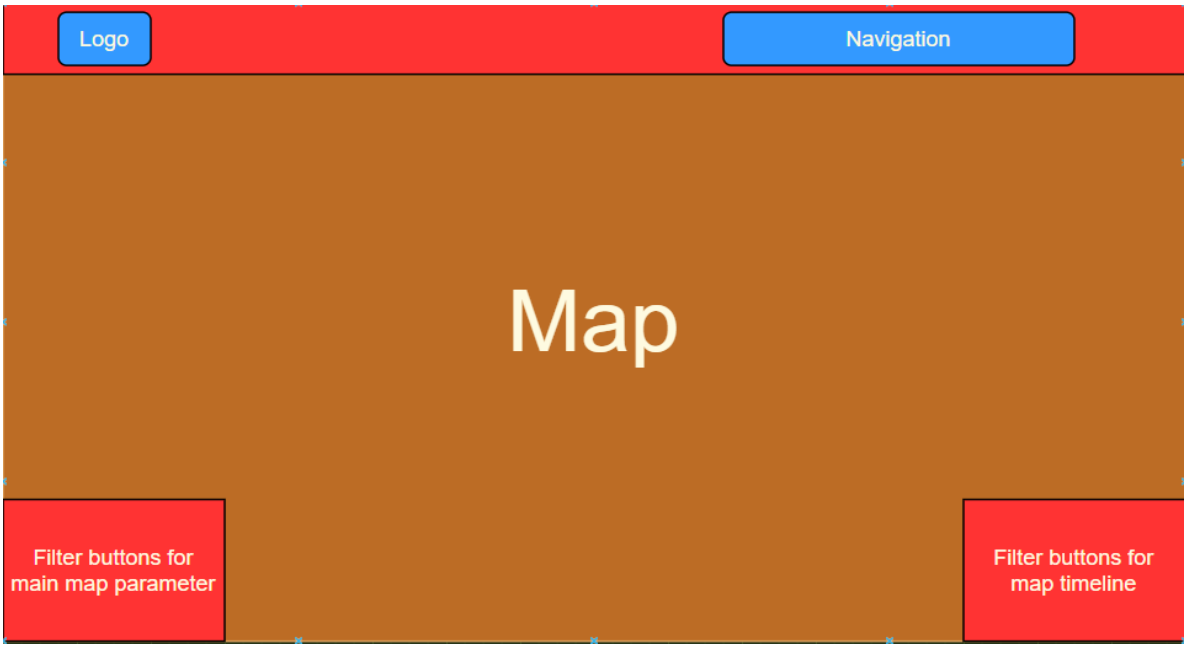

Рисунок 3.16 – Структура сторінки з картою

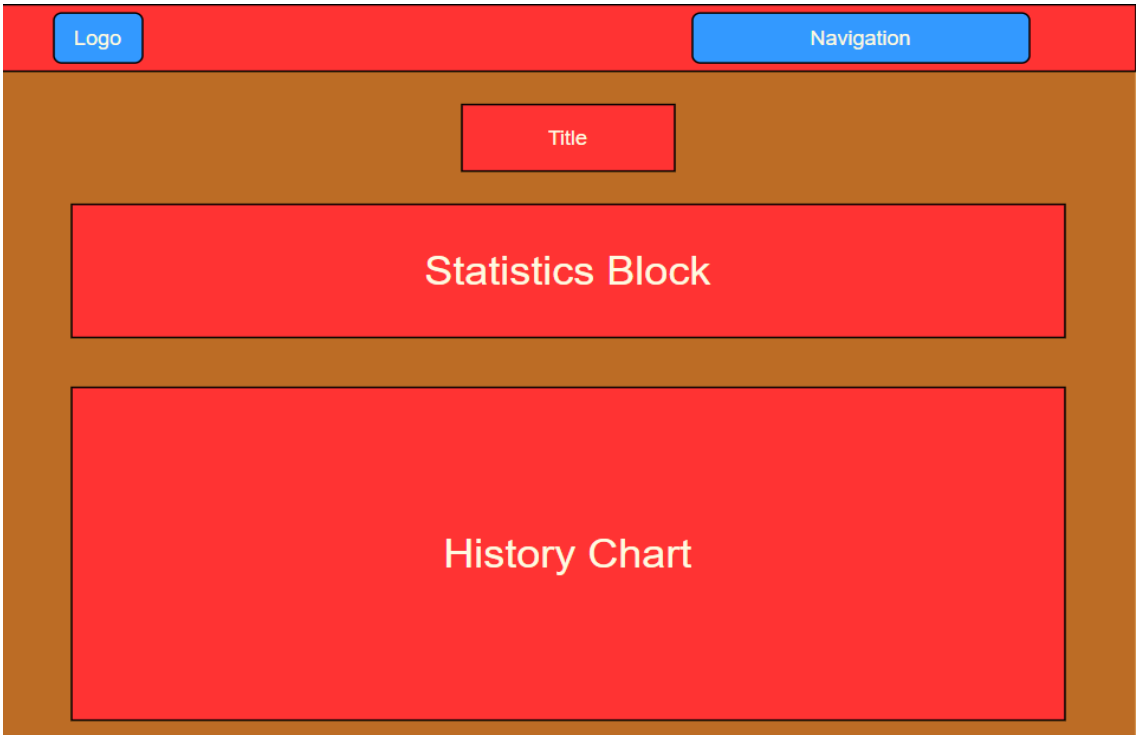

Рисунок 3.17 – Структура сторінки зі статистикою

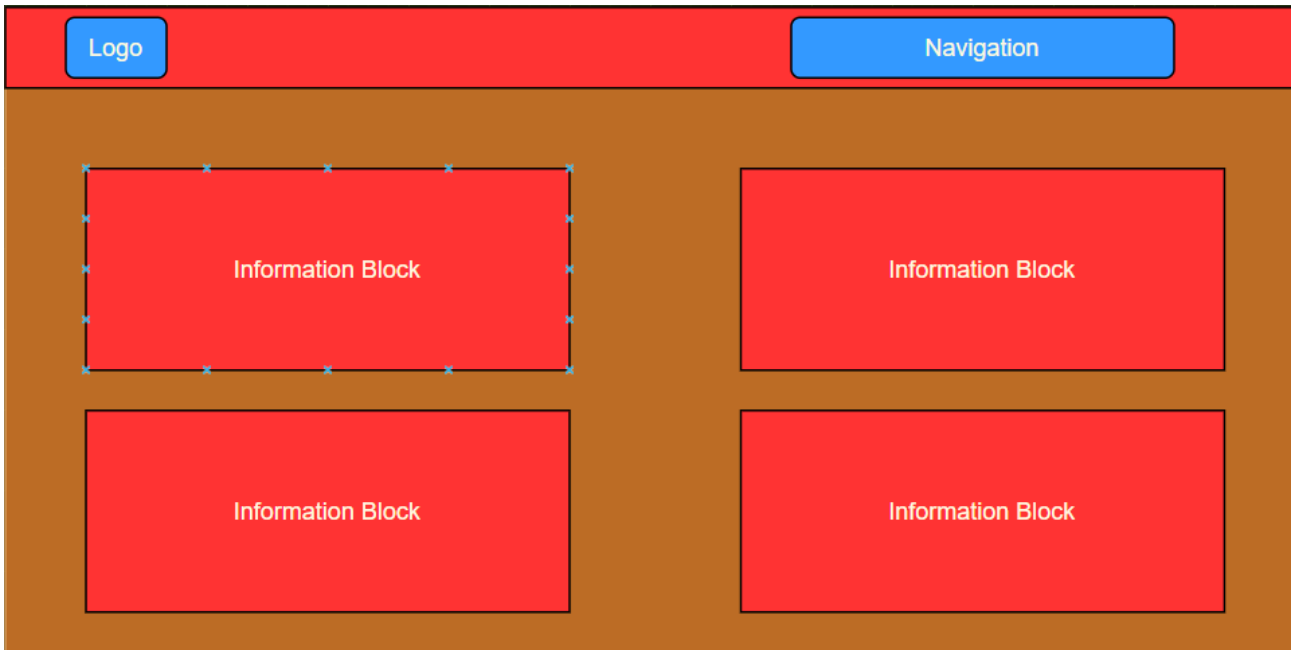

Рисунок 3.18 – Структура сторінки з інформацією

3.3.2 Сторонні модулі, використанні у веб-застосунку, їх опис та призначення

Для простішої та швидшої розробки веб-застосунку необхідно встановити деякі додаткові модулі, які допоможуть створити необхідний функціонал. Слід зазначити, що всі модулі пристосовані до роботи в браузерах на базі Chromium, тому гарантувати коректну роботу у всіх браузерах не можу.

Таблиця 3.1 – Сторонні модулі, використані у веб-застосунку, їх опис та призначення

Назва модуля	Опис модуля
mapbox-gl	Модуль з готовими методами для роботи з інтерактивною картою. Дозволяє швидко інтегрувати дані формату json в об’єкти на карті, задати їм потрібні параметри та оновлювати їх динамічно, за допомогою системи окремих слоїв [25]

Продовження таблиці 3.1

react-loader-spinner	Бібліотека, яка надає для використання готові рішення для утворення лоадерів, які необхідні для застосунку в той час, як він отримує дані з сервера
swr	Модуль, який додає до функціоналу React новий хук, який полегшує роботу з сервером [26]
recharts	Модуль, який додає велику кількість готових рішень для візуалізації даних у вигляді графіку [27]

3.3.2 Опис роботи з API

Робота з API на боці фронтенду стає досить простою задачею з використанням хука useSWR, але все одно є необхідність в адаптації даних для додавання на карту у вигляді кружечків.

Хуки дозволяють компонентам функцій мати доступ до стану та інших функцій React. Через це компоненти класу, як правило, більше не потрібні [28].

На карті використовується два запити на сервер. Перший отримує глобальні дані і повертає більшу кількість провінції, а інший повертає меншу кількість, але при цьому саме він використовується для фільтрації за часом.

Адаптація даних відбувається у окремих функціях. Функція totalDataFetcher(url) приймає в себе url та повертає адаптовані дані для карти. specificDataFetcher працює так само, тільки з тою різницею, що у вихідних даних будуть поля зі статистикою за часові проміжки.

Приклад функції (рисунок 3.19):

```
export const totalDataFetcher = (url) =>
    fetch(url) Promise<Response>
        .then((r : Response ) => r.json()) Promise<any>
        .then((data) =>
            data.map((point, index) => ({
                type: 'Feature',
                geometry: {
                    type: 'Point',
                    coordinates: [point.coordinates[0].longitude, point.coordinates[0].latitude]
                },
                properties: {
                    _id: index, // unique identifier in this case the index
                    country: point.country,
                    province: point.province,
                    cases: point.stats[0].confirmed,
                    deaths: point.stats[0].deaths,
                    vaccine: point.stats[0].vaccine
                }
            }))
        )
```

Рисунок 3.19. – Реалізація адаптації даних, отриманих від сервера

Додатково, для карти мені знадобився сторонній API, дані з якого немає необхідності додавати в базу даних. Назва цього API https://countryflagsapi.com, і, як можна судити з назви, він повертає прапори країн для відображення на карті.

Для сторінки зі статистикою адаптації даних не потрібно.

В статистиці використовується запит на історичні дані, щоб побудувати графік. Схему роботи веб-застосунку з сервером зображено на рисунку 3.20.

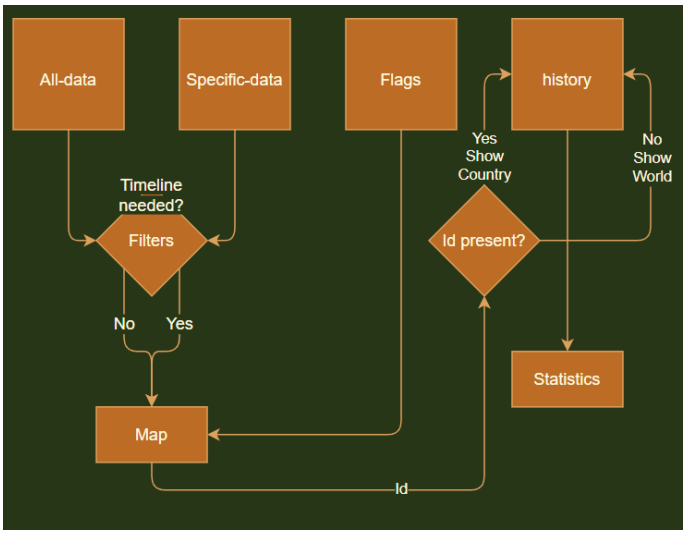

Рисунок 3.20. – Схема роботи веб-застосунку з сервером

3.3.3 Компоненти веб-застосунку

Під час розробки веб-застосунку я розробив деякі додаткові компоненти та допоміжні файли, щоб розбити застосунок по функціоналу. Нижче представлено таблицю, з описом цих компонентів.

Таблиця 3.2 – Опис компонентів, розроблених у веб-застосунку

Назва	Опис
App.js	Головний компонент, в якому збираються всі інші. Він є останньою ланкою між перетворенням JSX-синтаксису в HTML-розмітку
helper.js	Компонент, в якому зберігаються всі допоміжні функції, які можливо винести для використанна в окремі файли
Statistics.js	Компонент, призначений для відображення статистичних історичних даних по світу або по конкретній країна. Використовує запит на маршрут API v1/history
Map.js	Компонент, який відповідає за роботу з інтерактивною картою. Отримує дані з серверу та з API про прапори країн, оброблює їх та виставляє на інтерактивну карту, з якою може взаємодіяти користувач
Information.js	Компонент, який показує корисну інформацію про SARS-Cov-2
Navbar.js	Компонент, який відповідає за перехід між сторінками
mapDataToLayer(filter, timeline)	Допоміжна функція, яка видаляє минулий слой з карти та додає новий, з тими параметрами, які користувач обрав на кнопках фільтрації. Створює для об’єкта на карті popup та перенаправлення на статистику країни. Приймає аргументи filter, timeline.

Продовження таблиці 3.2

createLayer()	Допоміжна функція, яка повертає параметри поточного слою в залежності від фільтрації, обраної користувачем. Приймає аргументи про максимальне, середнє та мінімальне число випадків, назву, тип та кольорову схему слоя.
yAxisFormatter, dateFormatter	Функції-форматери даних для відображення на графіку в компоненті Statistics.js
specificDataFetcher, totalDataFetcher, chartDataFetcher	Допоміжні функції, які адаптують дані, отримані з сервера, у вигляд, необхідний для роботи в інших компонентах
specSource	Допоміжна функція, яка визначає, дані з якого джерела потрібно використовувати на карті, з coronavirus.app чи з disease.sh

3.3.4 Опис структури веб-застосунку

Розроблений веб-застосунок складається з файлів, що розташовані в певній ієрархії та працюють між собою в користувальницькому режимі. У схемі на рисунку 3.7 представлено внутрішню файлову структуру програми на етапі розробки, коли проект ще не було «збудовано» у повноцінний, готовий до деплою веб-застосунок.

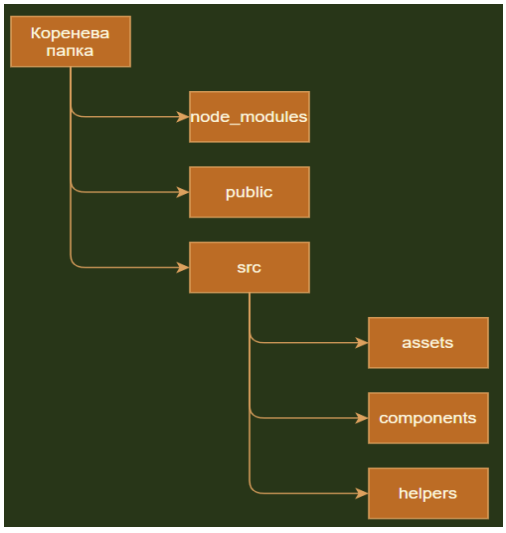

Рисунок 3.21 – Схема файлової структури веб-застосунку

Таблиця 3.3 – Внутрішня файлова структура програми

Назва	Опис
Коренева папка	Папка, в якій знаходяться всі файли веб-застосунку
node_modules	Папка, що містить всі необхідні для роботи веб-застосунку модулі. Це включаю в себе як і чисту бібліотеку React, так і будь-які сторонні модулі, встановлені розробником
public	Папка, що містить кінцеву сторінку з розміткою HTML, іконки, які будуть використовуватися на вкладці сайту, логотип, а також певні конфігурації файлу
src	Папка, в якій знаходиться весь основний код веб-застосунку
components	Папка, в якій зберігаються код всіх компонентів веб-застосунку, включаючи файли стилів
assets	Папка, в якій містяться зображення, необхідні всередині веб-застосунку
helpers	Папка, в які зберігаються файли-хелпери, які призначені для роботи всередині інших компонентів

3.3.5 Деплой застосунку в мережу інтернет

Як і серверну частину веб-застосунку, фронтенд теж необхідно завантажити в хмарне сховище, щоб він мав свою власну адресу і до нього мали б доступ всі користувачі всесвітньої павутини.

Для розгортання React-застосунків не підійде Heroku, адже він пристосований до бекенд-застосунків. Тому для цього скористаємося сервісом Netlify. Netlify надає можливість безкоштовно розгортати веб-застосунки, регулярно оновлювати їх, при умові, що проект буде простий, з невеликим трафіком та автоматично згенерованим доменним іменем.

Розгортати веб-застосунок настільки ж просто, як і за допомогою Heroku. Достатньо лише встановити netlify-cli, де виконати команду всередині проекту netlify deploy. Попередньо необхідно, у випадку React, запустити команду npm run build, адже саме файли в папці build ми будемо використовувати на ресурсі [29].

Після виконання всіх команд ми можемо отримати доступ до веб-застосунку за такою адресою: https://covid-map-nk.netlify.app/stats/1aVVar2d0

Оглянемо інтерфейс веб-застосунку (рисунок 3.22):

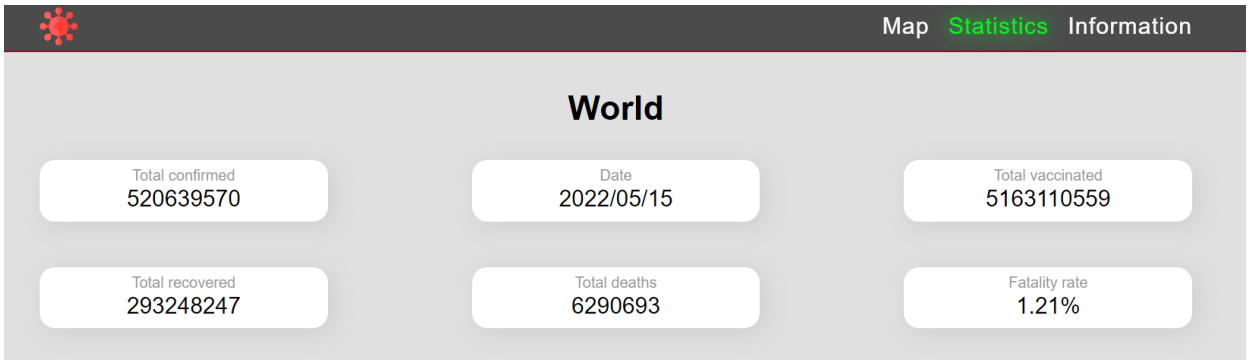

Рисунок 3.22 – Сторінка зі статистикою

Ця сторінка є стартовою та перша зустрічає користувача. На ній містяться загальні дані про епідемію та графік, який можна побачити на рисунку 3.23.

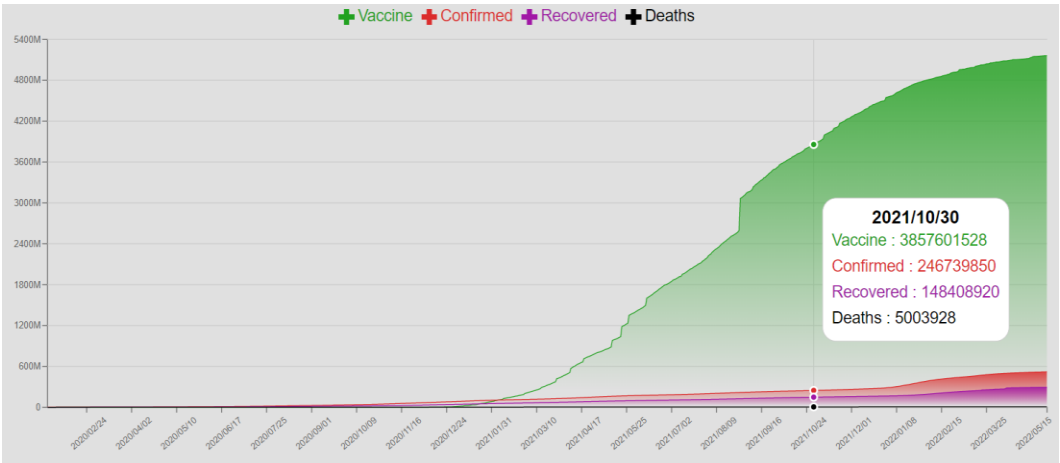

Рисунок 3.23 – Графік історичної статистики

Результат переходу на карту можна побачити на рисунку 3.24.

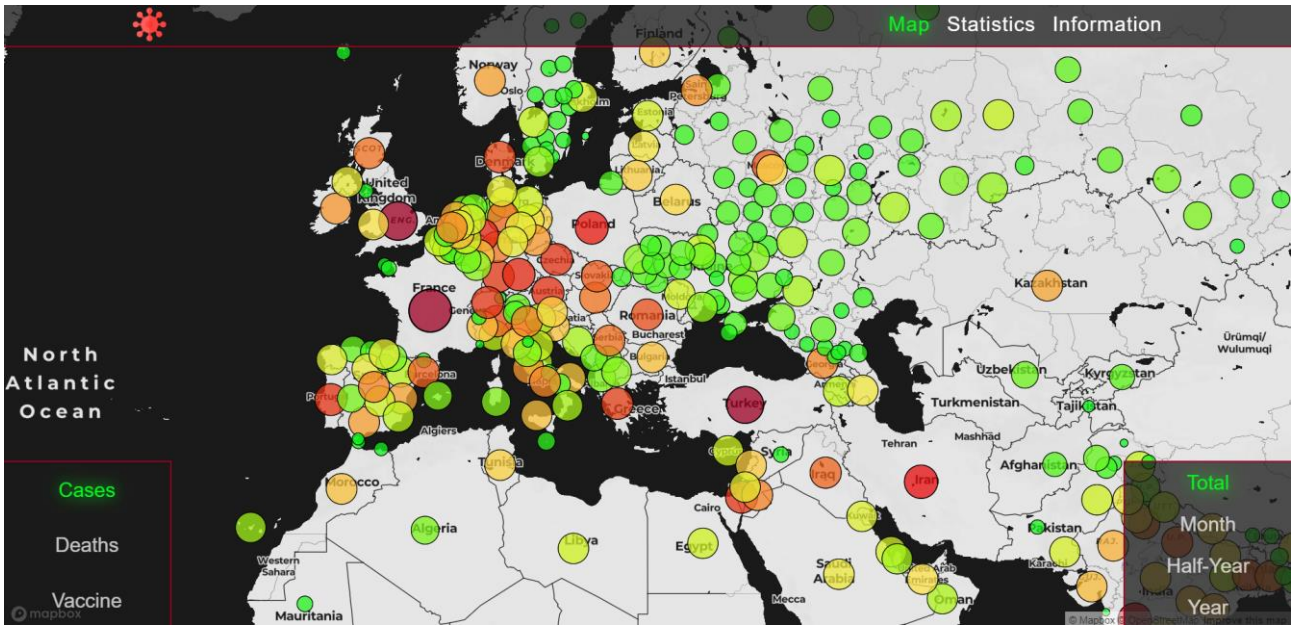

Рисунок 3.24 – Карта з фільтрами по випадках за весь час

При виборі інших фільтрів контент на карті буде змінено, див. рисунок 3.25:

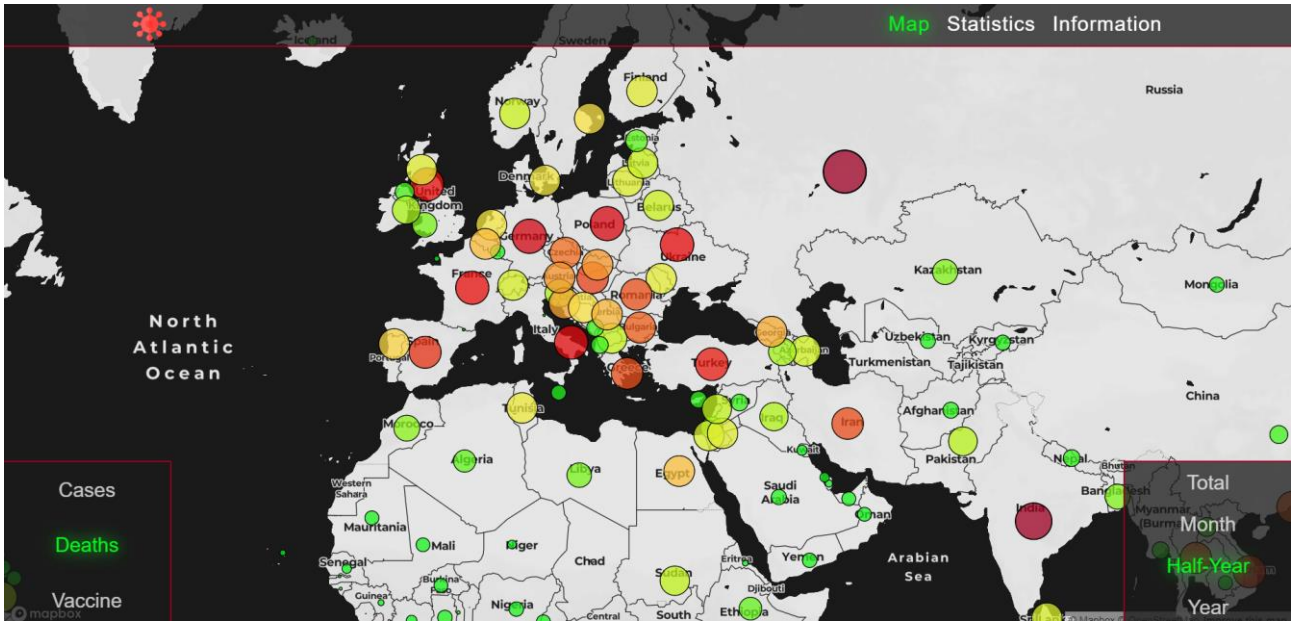

Рисунок 3.25 – Карта з фільтрами по смертях за півроку

Останньою сторінкою є сторінка з корисною інформацією про хворобу, див. рисунок 3.26. Поради були взяті з сайту Міністерства здоров'я України [30].

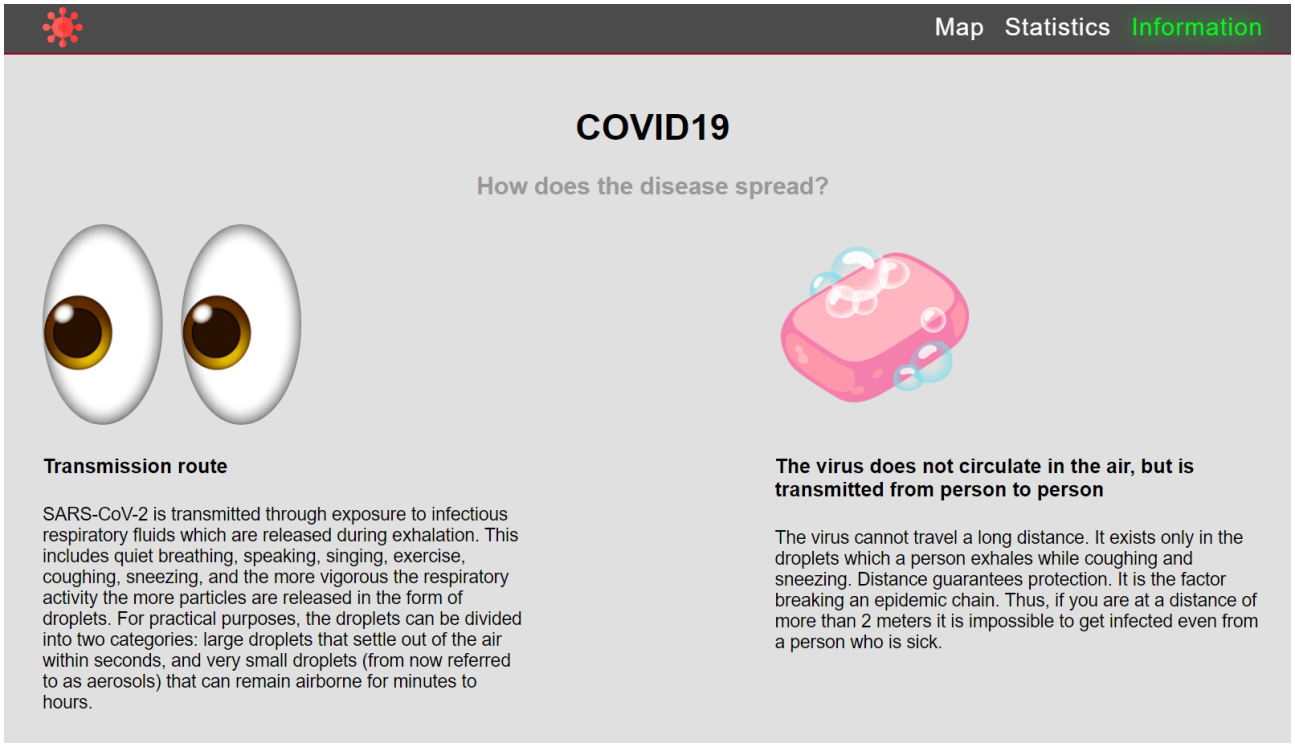

Рисунок 3.26 – Сторінка з корисною інформацією про хворобу

Висновки до розділу

Отже, мною було реалізовано програмно веб-сервіс для моніторингу рівня захворюваності та вакцинації епідемії SARS-CoV-2. Для цього було використано сучасні бібліотеки, модулі та пакети, призначені для обробки та візуалізації даних, а також найкращі практики та рішення в сфері побудови веб-застосунків.

ВИСНОВОК

У результаті виконання бакалаврської кваліфікаційної роботи було спроектовано та розроблено веб-застосунок для моніторингу епідеміологічної ситуації з SARS-CoV-2 та виконано такі завдання:

- Досліджено сучасні підходи до розроблення і впровадження веб-сервісів;
- Проаналізовано архітектурні рішення та обрано програмні засоби для реалізації веб-системи;
- Програмно реалізовано веб-сервіс для моніторингу рівня захворюваності та вакцинації епідемії SARS-CoV-2.

Зокрема, було досліджено аналоги побудованої системи, підібрано найкращі практики та формат веб-застосунку, розглянуто різні технології, які на сьогодні використовуються для побудови сучасних веб-застосунків, оглянуто джерела статистичних даних про епідеміологічну ситуацію з SARS-CoV-2 та обрано найкращі з них, які після цього були згруповані та об’єднані в єдиний формат.

Велика частина часу була виділена в першу чергу на проектування системи, бази даних, окремих компонентів та їх взаємодії, куди також входила розробка дизайну веб-застосунку. Досліджено сучасні підходи до розробки користувацького інтерфейсу, серверної частини веб-застосунків. Зроблено огляд та вибір бібліотек, модулів, пакетів, фреймворків, необхідних для створення системи для моніторингу за предметами та явищами, де необхідне використання та візуалізація статистичних даних.

Для проектування та розробки системи кваліфікаційної бакалаврської роботи був використаний стек технологій MERN на мові програмування JavaScript, куди входять такі технології, як MongoDB, ExpressJS, React, NodeJS. Для серверної частини веб-застосунку було використано ExpressJS та NodeJS, код було розгорнуто в мережі інтернет за допомогою хмарного сервісу Heroku.

База даних спроектована та розроблена за допомогою засобів MongoDB та бібліотеки mongoose. Для створення користувацького інтерфейсу використовувалася бібліотека React, до допоміжних засобів відносяться HTML, CSS, Sass. Код фронтенд-частини веб-застосунку було розгорнуто в мережі інтернет за допомогою засобів Netlify. Розробка виконувалася в IDE WebStorm від JetBrains, яке було безкоштовно надане мені за студентською ліцензією.

В результаті було створено веб-застосунок для моніторингу епідеміологічної ситуації, пов’язаної з SARS-CoV-2 з актуальною інформацією про протікання хвороби, рівень смертності, вакцинації, поширення інфекції в країнах світу та їх регіонах.

Головними особливостями веб-застосунку є інтерактивна карта, щогодинне оновлення, інфографіка, а також доступ у мережі інтернет, завдяки чому сайт доступний на всіх платформах.

Список використаних джерел

1. Pamela Fox. The World Wide Web. URL: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:web-protocols/a/the-world-wide-web
2. Телекомунікаційні системи та мережі / В. В. Поповський, О. В. Левешко, М. Д. Плотніков та ін. ; Львів : СМІТ, 2018. 134 с.
3. Pavan Podila. HTTP: Протокол, який повинен розуміти кожний веб-розробник (Частина 1). URL: https://code.tutsplus.com/uk/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177
4. MoesiF. Comparisons of API Architectural Styles. 22.01.2022 p. URL: https://www.moesif.com/blog/api-guide/comparisons-of-api-architectural-styles/
5. Lokesh Gupta. REST Architectural Constraints. 09.03.2022 p. URL: https://restfulapi.net/rest-architectural-constraints/
6. Martin Fowker. Richardson Maturity Model. 18.03.2010 p. URL: https://martinfowler.com/articles/richardsonMaturityModel.html
7. Express. Routing. URL: https://expressjs.com/en/guide/routing.html
8. NodeJS. Документація. URL: https://nodejs.org/uk/docs/
9. Date C. J. An Introduction to Database Systems. London : Pearson, 2003. 520 c.
10. Lauren Schaefer. What is NoSQL? URL: https://www.mongodb.com/nosql-explained
11. Mongoose. Mongoose API Docs. URL: https://mongoosejs.com/docs/api.html
12. MDN. SPA (Single-page application). URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA
13. Utkarsh Sidana. What are the Advantages and Disadvantages of Angular? 22.11.2021 p. URL: https://www.edureka.co/blog/advantages-and-disadvantages-of-angular/

14. KnowlegeHut. Advantages and disadvantages of Angular. 18.07.2020 p. URL: https://www.knowledgehut.com/blog/web-development/advantages-and-disadvantages-of-angular
15. Alexsoft. The Good and the Bad of Angular Development? 28.05.2022. URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/
16. Alexsoft. The Good and the Bad of Vue.js Framework Programming. 11.09.2019 p. URL: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/
17. Wojciech Baranowski. React JS: Advantages and Disadvantages? 04.10.2021 p. URL: https://massivepixel.io/blog/react-advantages-disadvantages/
18. Javatpoint. Pros and Cons of ReactJS. URL: https://www.javatpoint.com/pros-and-cons-of-react (дата звернення: 20.03.2022)
19. Alvaro Insignares. React Pros and Cons: What are the Advantages and Disadvantages of ReactJS? 10.03.2021 p. URL: https://www.koombea.com/blog/react-pros-and-cons-what-are-the-advantages-and-disadvantages-of-reactjs/ (дата звернення: 20.03.2022)
20. Aris Pattakos. Angular vs React vs Vue 2022. 04.01.2022 p. URL: https://athemes.com/guides/angular-vs-react-vs-vue/
21. Tutorialspoint. MVC Pattern. URL: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm (дата звернення: 21.03.2022)
22. Refactoring Guru. Адаптер. URL: https://refactoring.guru/uk/design-patterns/adapter (дата звернення: 21.03.2022)
23. Pair. What is Cron and How do I Use It? URL: https://www.pair.com/support/kb/configuring-cron/
24. Heroku Dev Center. Deploying with Git. URL: https://devcenter.heroku.com/articles/git
25. Mapbox. API Reference. URL: https://docs.mapbox.com/mapbox-gl-js/api/

26. Vercel. SWR React Hooks for Data Fetching. URL: https://swr.vercel.app/
27. Recharts Group. Rechars. URL: https://recharts.org/en-US/
28. W3School. React Hooks. URL: https://www.w3schools.com/react/react_hooks.asp
29. Netlify. Deploying with Netlify. URL: https://www.netlify.com/blog/tags/deploy/
30. Ministry of health of Ukraine. COVID-19 pandemic in Ukraine. URL: https://covid19.gov.ua/en

Додатки

ДОДАТОК А

Код сервера, файл History.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
    country: {
        type: String,
        required: true
    },
    updatedAt: {
        type: String
    },
    id: {
        type: String,
        required: true
    },
    stats: [{
        confirmed: {
            type: Number
        },
        deaths: {
            type: Number
        },
        recovered: {
            type: Number
        },
        vaccine: {
            type: Number
        }
    }],
    province: {
        type: String
    },
    timeline: [{
        day: {
            type: String
        },
        confirmed: {
            type: Number
        },
        deaths: {
            type: Number
        },
        recovered: {
            type: Number
        },
        vaccine: {
            type: Number
        }
    }]
})

export default mongoose.model('History', schema)
```

ДОДАТОК Б

Код сервера, файл Data.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
    country: {
        type: String,
        required: true
    },
    updatedAt: {
        type: String
    },
    id: {
        type: String
    },
    stats: [{
        confirmed: {
            type: Number
        },
        deaths: {
            type: Number
        },
        recovered: {
            type: Number
        },
        vaccine: {
            type: Number
        }
    }],
    coordinates: [{
        latitude: {
            type: String
        },
        longitude: {
            type: String
        }
    }],
    province: {
        type: String
    },
    monthStats: [{
        confirmed: {
            type: Number
        },
        deaths: {
            type: Number
        },
        recovered: {
            type: Number
        },
        vaccine: {
            type: Number
        }
    }],
    halfyearStats: [{
        confirmed: {
            type: Number
        },
        deaths: {
            type: Number
```

```
        },
        recovered: {
            type: Number
        },
        vaccine: {
            type: Number
        }
    }],
    yearStats: [{
        confirmed: {
            type: Number
        },
        deaths: {
            type: Number
        },
        recovered: {
            type: Number
        },
        vaccine: {
            type: Number
        }
    }]
})

export default mongoose.model('Data', schema)
```

ДОДАТОК В

Код сервера, файл covid.js

```
import { Router } from 'express'
import path from 'path';
import {fileURLToPath} from 'url';
import Data from '../models/Data.js'
import History from "../models/History.js";

const router = new Router()
const __filename = fileURLToPath(import.meta.url)
const __dirname = path.dirname(__filename)

router.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, '..', 'index.html'))
})

router.get('/v1/all-data', (req, res) => {
    Data.find({'id': null}).lean().exec((err, data) => {
        res.setHeader('Access-Control-Allow-Origin', 'https://covid-map-
nk.netlify.app')
        return res.end(JSON.stringify(data))
    })
})

router.get('/v1/specific-data', (req, res) => {
    Data.where('id').ne(null).lean().exec((err, data) => {
        res.setHeader('Access-Control-Allow-Origin', 'https://covid-map-
nk.netlify.app')
        return res.end(JSON.stringify(data))
    })
})

router.get('/v1/history', (req, res) => {
    History.findOne({ id : req.query.countryId } ).lean().exec((err, data) => {
        res.setHeader('Access-Control-Allow-Origin', 'https://covid-map-
nk.netlify.app')
        return res.end(JSON.stringify(data))
    })
})

export default router
```

ДОДАТОК Г

Код сервера, файл index.js

```
import fetch, { Headers } from 'node-fetch'
import cors from 'cors'
import express from 'express'
import mongoose from 'mongoose'
import cron from 'node-cron'
import router from './routes/covid.js'
import Data from './models/Data.js'
import History from "./models/History.js";

const PORT = process.env.PORT || 3000

const app = express()

const globalHistory = {
    country: 'World',
    province: 'none',
    id: '1aVVar2d0',
    updatedAt: Date.now(),
    stats: {
        confirmed: 0,
        deaths: 0,
        recovered: 0,
        vaccine: 0
    },
    history: []
}

app.use(router)

async function start() {
    try {
        await
mongoose.connect('mongodb+srv://NeroKrase:k3fqZ5WuDUS3rQuR@cluster0.jmjpz.mongod
b.net/covid', {
            useNewUrlParser: true
        })
        const type = ['Content-Type', 'application/json']
        const key = ['Authorization', 'Bearer
SLpKfTVXOcgyPv3bKxvKcs55RV4XtZDOUZXvZNugjmDIMmFEb0upaJcas2oDgaUM']
        const headers = new Headers([type, key])
        app.listen(PORT, () => {
            console.log('Started')
        })
        cron.schedule('0 * * * *', async () => {
            console.log('in cron')
            let covGetPlaces = await getApi('https://coronavirus.app/get-
places', { headers })
            await insertHistoryData(covGetPlaces.data, headers)
            await insertPeriodicData(covGetPlaces.data, headers)
            let countryData = await getApi('https://disease.sh/v3/covid-
19/jhucsse')
            let vaccineTotal = await getApi(`https://disease.sh/v3/covid-
19/vaccine/coverage/countries/?lastdays=1`)
            await insertTotalData(countryData, vaccineTotal)
        })
    }
    catch (e) {
        console.log(e)
```

```
    }
}

async function getApi(url, params = {}){
    const response = await fetch(url, params)
    let data = await response.json()
    return data
}

async function insertPeriodicData(data, headers){
    for (const country of data){
        console.log(country.name)
        const timeline = await getApi(`https://coronavirus.app/get-
history?id=${country.id}`, { headers })
        const periodicStats = await createPeriodicStats(timeline.data.history)
        if(!Object.keys(timeline).includes('error')){
            await Data.updateOne({
                country: country.country === 'US' || country.country === 'GB' ?
country.country : country.name,
                province: country.country === 'US' || country.country === 'GB' ?
country.name : 'none'
            }, {
                country: country.country === 'US' || country.country === 'GB' ?
country.country : country.name,
                updatedAt: country.updatedAt,
                id: country.id,
                stats: [{
                    confirmed: country.infected,
                    deaths: country.dead,
                    vaccine: country.vaccinated,
                    recovered: country.recovered
                }],
                coordinates: [{
                    latitude: country.latitude,
                    longitude: country.longitude
                }],
                province: country.country === 'US' || country.country === 'GB' ?
country.name : 'none',
                monthStats: [{
                    confirmed: periodicStats.month.confirmed,
                    deaths: periodicStats.month.deaths,
                    recovered: periodicStats.month.recovered,
                    vaccine: periodicStats.month.vaccine
                }],
                halfyearStats: [{
                    confirmed: periodicStats.halfyear.confirmed,
                    deaths: periodicStats.halfyear.deaths,
                    recovered: periodicStats.halfyear.recovered,
                    vaccine: periodicStats.halfyear.vaccine
                }],
                yearStats: [{
                    confirmed: periodicStats.year.confirmed,
                    deaths: periodicStats.year.deaths,
                    recovered: periodicStats.year.recovered,
                    vaccine: periodicStats.year.vaccine
                }]
            }, { upsert: true })
        }
    }
}

async function insertHistoryData(data, headers){
    for(const country of data){
```

```
        console.log(country.name)
        globalHistory.stats.confirmed += country.infected
        globalHistory.stats.deaths += country.dead
        globalHistory.stats.vaccine += country.vaccinated
        globalHistory.stats.recovered += country.recovered
        const timeline = await getApi(`https://coronavirus.app/get-
history?id=${country.id}`, { headers })
        const history = await mapHistory(timeline.data.history)
        await updateWorldStats(history)
        if(!Object.keys(timeline).includes('error')){
            await History.deleteOne({id: country.id})
            await History.updateOne({id: country.id}, {
                country: country.country === 'US' || country.country === 'GB' ?
country.country : country.name,
                updatedAt: country.updatedAt,
                id: country.id,
                stats: [{
                    confirmed: country.infected,
                    deaths: country.dead,
                    vaccine: country.vaccinated,
                    recovered: country.recovered
                }],
                province: country.country === 'US' || country.country === 'GB' ?
country.name : 'none',
                $push: {
                    timeline: {
                        $each: history
                    }
                }
            }, {upsert: true})
        }
    }
    await History.deleteOne({id: globalHistory.id})
    await History.updateOne({id: globalHistory.id}, {
        country: globalHistory.country,
        updatedAt: globalHistory.updatedAt,
        id: globalHistory.id,
        stats: [{
            confirmed: globalHistory.stats.confirmed,
            deaths: globalHistory.stats.deaths,
            vaccine: globalHistory.stats.vaccine,
            recovered: globalHistory.stats.recovered
        }],
        province: globalHistory.province,
        $push: {
            timeline: {
                $each: globalHistory.history
            }
        }
    }, {upsert: true})
}

async function mapHistory(history){
    return history.map(day => {
        return {
            day: day.day,
            confirmed: day.infected,
            deaths: day.dead,
            vaccine: day.vaccinated,
            recovered: day.recovered
        }
    })
}
```

```
async function updateWorldStats(history){
    history.map(day => {
        if(globalHistory.history.find(d => d.day === day.day)){
            const index = globalHistory.history.findIndex(d => d.day ===
day.day)
            globalHistory.history[index].confirmed += day.confirmed
            globalHistory.history[index].deaths += day.deaths
            globalHistory.history[index].vaccine += day.vaccine
            globalHistory.history[index].recovered += day.recovered
        }
        else{
            globalHistory.history.push(day)
        }
    })
}

async function insertTotalData(data, vaccineTotal){
    for (const country of data) {
        console.log(country.country)
        await Data.updateOne({country: country.country, province:
country.province, id: null}, {
            country: country.country,
            updatedAt: country.updatedAt,
            stats: [{
                confirmed: country.stats.confirmed,
                deaths: country.stats.deaths,
                recovered: country.stats.recovered,
                vaccine: filterVaccineTimeline(vaccineTotal, country.country)
            }],
            coordinates: [{
                latitude: country.coordinates.latitude,
                longitude: country.coordinates.longitude
            }],
            province: country.province
        }, { upsert: true })
    }
}

async function createPeriodicStats(history){
    return {
        month: {
            confirmed: history[0].infected - history[30].infected,
            deaths: history[0].dead - history[30].dead,
            recovered: history[0].recovered - history[30].recovered,
            vaccine: history[0].vaccinated - history[30].vaccinated
        },
        halfyear: {
            confirmed: history[0].infected - history[180].infected,
            deaths: history[0].dead - history[180].dead,
            recovered: history[0].recovered - history[180].recovered,
            vaccine: history[0].vaccinated - history[180].vaccinated
        },
        year: {
            confirmed: history[0].infected - history[365].infected,
            deaths: history[0].dead - history[365].dead,
            recovered: history[0].recovered - history[365].recovered,
            vaccine: history[0].vaccinated - history[365].vaccinated
        }
    }
}

function filterVaccineTimeline(timeline, country){
```

```
    const currentCountry = timeline.find(c => c.country === country)
    if(currentCountry === undefined){
        return null
    }
    const vaccination = Object.values(currentCountry.timeline)
    if(vaccination.length === 1){
        return vaccination[0]
    }
    else{
        return vaccination[vaccination.length - 1] - vaccination[0]
    }
}

// noinspection JSIgnoredPromiseFromCall
start()
```

ДОДАТОК Ґ

Код клієнта, файл App.js

```
//Packages
import React, {useEffect} from 'react';
import mapboxgl from 'mapbox-gl';
import { Routes, Route, useNavigate } from "react-router-dom";
//Components
import Navbar from "./components/Navbar/Navbar";
import Map from "./components/Map/Map";
import Statistics from "./components/Statistics/Statistics";
import Information from "./components/Information/Information";
//Styles
import './App.scss';
// Mapbox css - needed to make tooltips work later in this article
import 'mapbox-gl/dist/mapbox-gl.css';

mapboxgl.accessToken =
'pk.eyJ1IjoibmVyb2tyYXNlIiwiYSI6ImNrbXJvOWZobjA1bzgycHBjNGVwc3g4Y2UifQ.ebiYBezLv
sllg2i7WW6QVA';

const App = () => {
    const navigate = useNavigate()
    useEffect(() => {
        const href = window.location.pathname
        if(href === '/'){
            navigate('/stats/1aVVar2d0')
        }
    }, [])

    return (
        <div className='App'>
            <Navbar/>
            <Routes>
                <Route path='/' element={<Map/>}/>
                <Route path='/map' element={<Map/>}/>
                <Route path='/stats/:id' element={<Statistics/>}/>
                <Route path='/info' element={<Information/>}/>
            </Routes>
        </div>
    )
}

export default App;
```

ДОДАТОК Д

Код клієнта, файл mapHelper.js

```
export const specSource = (timeline, filter) => {
    return timeline === 'total' && filter !== 'vaccine'
}

export const chartDataFetcher = (url) =>
    fetch(url)
        .then((r) => r.json())


export const specificDataFetcher = (url) =>
    fetch(url)
        .then((r) => r.json())
        .then((specificData) =>
            specificData.map((point, index) => ({
                type: 'Feature',
                geometry: {
                    type: 'Point',
                    coordinates: [point.coordinates[0].longitude,
point.coordinates[0].latitude]
                },
                properties: {
                    _id: index, // unique identifier in this case the index
                    id: point.id,
                    country: point.country,
                    province: point.province,
                    cases: point.stats[0].confirmed,
                    deaths: point.stats[0].deaths,
                    vaccine: point.stats[0].vaccine,
                    monthCases: point.monthStats[0].confirmed,
                    monthDeaths: point.monthStats[0].deaths,
                    monthRecovered: point.monthStats[0].recovered,
                    monthVaccine: point.monthStats[0].vaccine,
                    halfyearCases: point.halfyearStats[0].confirmed,
                    halfyearDeaths: point.halfyearStats[0].deaths,
                    halfyearRecovered: point.halfyearStats[0].recovered,
                    halfyearVaccine: point.halfyearStats[0].vaccine,
                    yearCases: point.yearStats[0].confirmed,
                    yearDeaths: point.yearStats[0].deaths,
                    yearRecovered: point.yearStats[0].recovered,
                    yearVaccine: point.yearStats[0].vaccine
                }
            }))
        )

export const totalDataFetcher = (url) =>
    fetch(url)
        .then((r) => r.json())
        .then((data) =>
            data.map((point, index) => ({
                type: 'Feature',
                geometry: {
                    type: 'Point',
                    coordinates: [point.coordinates[0].longitude,
point.coordinates[0].latitude]
                },
                properties: {
                    _id: index, // unique identifier in this case the index
                    country: point.country,
```

```
                    province: point.province,
                    cases: point.stats[0].confirmed,
                    deaths: point.stats[0].deaths,
                    vaccine: point.stats[0].vaccine
                }
            }))
        )

export const createLayer = (max,
                            min,
                            avg,
                            name,
                            type = 'cases',
                            colors = mainColors) => {
    return {
        id: name,
        source: 'points', // this should be the id of the source
        type: 'circle',
        // paint properties
        paint: {
            'circle-opacity': 0.75,
            'circle-stroke-width': [
                'interpolate',
                ['linear'],
                ['get', type],
                1, 1,
                max, 1.75
            ],
            'circle-radius': [
                'interpolate',
                ['linear'],
                ['get', type],
                1, min, avg / 32,
                8, avg / 8,
                10, avg / 4,
                14, avg / 2,
                18, max, 25
            ],
            'circle-color': [
                'interpolate',
                ['linear'],
                ['get', type],
                min, colors[0],
                max / 64, colors[1],
                max / 32, colors[2],
                max / 16, colors[3],
                max / 8, colors[4],
                max / 4, colors[5],
                max / 2, colors[6]
            ]
        }
    }
}

export const LAYERS = [
    'casesLayer',
    'deathsLayer',
    'vaccineLayer',
    'yearCasesLayer',
    'yearDeathsLayer',
    'yearVaccineLayer',
    'halfyearCasesLayer',
    'halfyearDeathsLayer',
```

```
    'halfyearVaccineLayer',
    'monthCasesLayer',
    'monthDeathsLayer',
    'monthVaccineLayer'
]

const mainColors = [
    '#0af520',
    '#c7f61c',
    '#fee64c',
    '#f67a25',
    '#e53b18',
    '#ea0d0f',
    '#9f0125'
]
```

ДОДАТОК Е

Код клієнта, файл Navbar.js

```
import React from "react"
import { NavLink } from "react-router-dom";
import './Navbar.scss'
import Logo from '../../assets/images/logo.svg'


const Navbar = () => {
    return (
        <nav className={'navbar'}>
            <div className={'content_wrapper'}>
                <img src={Logo} alt="logo" className={'logo'}/>
                <ul className={'navlist'}>
                    <NavLink
                        to='/map'
                        className={({ isActive }) => (isActive ?
`navlist_element active` : `navlist_element`)}
                    >
                        Map
                    </NavLink>
                    <NavLink
                        to='/stats/1aVVar2d0'
                        className={({ isActive }) => (isActive ?
`navlist_element active` : `navlist_element`)}
                    >
                        Statistics
                    </NavLink>
                    <NavLink
                        to='/info'
                        className={({ isActive }) => (isActive ?
`navlist_element active` : `navlist_element`)}
                    >
                        Information
                    </NavLink>
                </ul>
            </div>
        </nav>
    )
}

export default Navbar
```

Код клієнта, файл Navbar.scss

```
.navbar{
  position: fixed;
  background-color: rgba(26, 26, 26, 0.75);
  font-family: 'Baloo Thambi 2', sans-serif;
  font-size: 1.5rem;
  width: 100%;
  height: 50px;
  color: #fff;
  border-bottom: 2px solid #9f0125;
  z-index: 2;
}

.logo{
  height: 40px;
}
```

```
.content_wrapper{
  margin: 0 auto;
  width: 80%;
  height: 100%;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.navlist{
  display: inline-flex;
}

.navlist_element{
  color: #fff;
  text-decoration: none;
  list-style: none;
  transition: 0.3s;
  font-weight: 300;
  letter-spacing: 1px;
}

.navlist_element.active{
  color: #0AF520FF;
  text-shadow: 0 0 20px #0AF520FF;
}

.navlist_element:hover{
  color: #9f0125;
  text-shadow: 0 0 20px #9f0125;
}

.navlist_element + .navlist_element{
  margin-left: 20px;
}
```

ДОДАТОК Є

Код клієнта, файл Map.js

```
import React, {useEffect, useRef, useState} from "react"
import useSWR from "swr"
import { useNavigate } from "react-router-dom";
import 'mapbox-gl/dist/mapbox-gl.css';
import mapboxgl from 'mapbox-gl';
import lookup from "country-code-lookup"
import { totalDataFetcher, specificDataFetcher, specSource, createLayer, LAYERS
} from '../../heplers/mapHelper'
import './Map.scss';
import {Rings} from "react-loader-spinner";

// @ts-ignore
// eslint-disable-next-line import/no-webpack-loader-syntax
mapboxgl.workerClass = require('worker-loader!mapbox-gl/dist/mapbox-gl-csp-
worker').default;

const Map = () => {
    const mapboxElRef = useRef(null);
    const map = useRef(null);
    const [show, setShow] = useState(false)
    const [filter, setFilter] = useState('cases')
    const [timeline, setTimeline] = useState('total')
    const [activeBtn, setActiveBtn] = useState('total-cases')

    const navigate = useNavigate();

    const navigateToStats = (id) => {
        navigate(`/stats/${id}`);
    }

    const mapDataToLayer = (filter, timeline) => {
        setFilter(filter)
        setTimeline(timeline)
        setActiveBtn(`${timeline}-${filter}`)

        let avg, min, max;
        let currentFilter = filter;

        if(timeline !== 'total'){
            filter = `${timeline}${filter.charAt(0).toUpperCase() +
filter.slice(1)}`
        }

        if(specSource(timeline, currentFilter)){
            avg = data.reduce((total, next) => total + next.properties[filter],
0) / data.length

            min = Math.min(...data.map((item) => item.properties[filter]))

            max = Math.max(...data.map((item) => item.properties[filter]))

            map.current.getSource('points').setData({ type: 'FeatureCollection',
features: data})
        }
        else{
            avg = specificData.reduce((total, next) => total +
next.properties[filter], 0) / specificData.length
```

```
            min = Math.min(...specificData.map((item) =>
item.properties[filter]))

            max = Math.max(...specificData.map((item) =>
item.properties[filter]))

            map.current.getSource('points').setData({ type: 'FeatureCollection',
features: specificData})
        }

        // Remove previous layer
        map.current.getStyle().layers.forEach((layer) => {
            if(LAYERS.includes(layer.id)){
                map.current.removeLayer(layer.id)
            }
        })

        // Add our layer
        map.current.addLayer(createLayer(max, min, avg, `${filter}Layer`,
filter))

        // Create a mapbox popup
        const popup = new mapboxgl.Popup({
            closeButton: false,
            closeOnClick: false
        })

        // Variable to hold the active country/province on hover
        let lastId

        // Mouse move event
        map.current.on('mousemove', `${filter}Layer`, (e) => {
            // Get the id from the properties
            const id = e.features[0].properties._id

            // Only if the id are different we process the tooltip
            if (id !== lastId) {
                lastId = id

                // Change the pointer type on move move
                map.current.getCanvas().style.cursor = 'pointer'

                let cases, deaths, vaccine, country, province;

                switch (timeline){
                    case 'total':
                        cases = e.features[0].properties.cases;
                        deaths = e.features[0].properties.deaths;
                        vaccine = e.features[0].properties.vaccine;
                        country = e.features[0].properties.country;
                        province = e.features[0].properties.province;
                        break;
                    case 'month':
                        cases = e.features[0].properties.monthCases;
                        deaths = e.features[0].properties.monthDeaths;
                        vaccine = e.features[0].properties.monthVaccine;
                        country = e.features[0].properties.country;
                        province = e.features[0].properties.province;
                        break;
                    case 'halfyear':
                        cases = e.features[0].properties.halfyearCases;
                        deaths = e.features[0].properties.halfyearDeaths;
                        vaccine = e.features[0].properties.halfyearVaccine;
```

```
                        country = e.features[0].properties.country;
                        province = e.features[0].properties.province;
                        break;
                    case 'year':
                        cases = e.features[0].properties.yearCases;
                        deaths = e.features[0].properties.yearDeaths;
                        vaccine = e.features[0].properties.yearVaccine;
                        country = e.features[0].properties.country;
                        province = e.features[0].properties.province;
                        break;
                    default:
                        cases = e.features[0].properties.cases;
                        deaths = e.features[0].properties.deaths;
                        vaccine = e.features[0].properties.vaccine;
                        country = e.features[0].properties.country;
                        province = e.features[0].properties.province;
                        break;
                }
                const coordinates = e.features[0].geometry.coordinates.slice()

                // Get all data for the tooltip
                if(country === 'GB'){
                    country = 'United Kingdom'
                }
                const countryISO = lookup.byCountry(country)?.iso2 ||
lookup.byInternet(country)?.iso2

                const provinceHTML = typeof province !== 'undefined' && province
!== 'none' ? `<p>Province: <b>${province}</b></p>` : ''

                const mortalityRate = ((deaths / cases) * 100).toFixed(2)

                const countryFlagHTML = Boolean(countryISO)
                    ? `<img
src="https://countryflagsapi.com/png/${countryISO}"></img>`
                    : ''

                const HTML = `<p>Country: <b>${country}</b></p>
            ${provinceHTML}
            <p>Cases: <b>${cases}</b></p>
            <p>Deaths: <b>${deaths}</b></p>
            <p>${timeline === 'total' && country !== 'US' ? 'Vaccine (for
country): ' : 'Vaccine'}<b>${vaccine ? vaccine : 'No Info'}</b></p>
            <p>Mortality Rate: <b>${mortalityRate}%</b></p>
            ${countryFlagHTML}`

                while (Math.abs(e.lngLat.lng - coordinates[0]) > 180) {
                    coordinates[0] += e.lngLat.lng > coordinates[0] ? 360 : -360
                }

                popup.setLngLat(coordinates).setHTML(HTML).addTo(map.current)
            }
        })

        // Mouse leave event
        map.current.on('mouseleave', `${filter}Layer`, function () {
            // Reset the last Id
            lastId = undefined
            map.current.getCanvas().style.cursor = ''
            popup.remove()
        })

        // Mouse click event
```

```
        map.current.on('click', `${filter}Layer`, (e) => {
            const id = e.features[0].properties.id;
            if(id){
                navigateToStats(id)
            }
        })

        if(!specSource(timeline, filter)){
        }
    }

    // Fetching our data with swr package
    const { data } = useSWR('https://calm-headland-93075.herokuapp.com/v1/all-
data', totalDataFetcher)
    const specificData = useSWR('https://calm-headland-
93075.herokuapp.com/v1/specific-data', specificDataFetcher).data

    // Initialize our map
    useEffect(() => {
        if (data && specificData) {
            let timer = setTimeout(() => {
                setShow(true)
                map.current = new mapboxgl.Map({
                    container: mapboxElRef.current,
                    style: 'mapbox://styles/mapbox/cj3kbeqzo00022smj7akz3o1e',
                    center: [-98, 37], // initial geo location
                    zoom: 3 // initial zoom
                })

                // Call this method when the map is loaded
                map.current.once('load', function () {
                    // Add S
                    map.current.addSource('points', {
                        type: 'geojson',
                        data: {
                            type: 'FeatureCollection',
                            features: data
                        }
                    })

                    mapDataToLayer(filter, timeline)
                })
            }, 0)

            return () => {
                clearTimeout(timer)
            }
        }
    }, [data, specificData])

    if (show){
        return (
            <div className='mapContainer'>
                <div className='mapBox' ref={mapboxElRef} />
                <div className="filterBtns" id='filterBtns'>
                    <button
                        className={activeBtn.includes('cases') ? 'active' :
null}

                        onClick={() => {mapDataToLayer('cases', timeline)}}
                    >
                        Cases
                    </button>
                    <button
```

```
                        className={activeBtn.includes('deaths') ? 'active' :
null}
                        onClick={() => {mapDataToLayer('deaths', timeline)}}
                    >
                        Deaths
                    </button>
                    <button
                        className={activeBtn.includes('vaccine') ? 'active' :
null}
                        onClick={() => {mapDataToLayer('vaccine', timeline)}}
                    >
                        Vaccine
                    </button>
                </div>
                <div className="timeBtns">
                    <button
                        className={activeBtn.includes('total') ? 'active' :
null}
                        onClick={() => {mapDataToLayer(filter, 'total')}}
                    >
                        Total
                    </button>
                    <button
                        className={activeBtn.includes('month') ? 'active' :
null}
                        onClick={() => {mapDataToLayer(filter, 'month')}}
                    >
                        Month
                    </button>
                    <button
                        className={activeBtn.includes('halfyear') ? 'active' :
null}
                        onClick={() => {mapDataToLayer(filter, 'halfyear')}}
                    >
                        Half-Year
                    </button>
                    <button
                        className={activeBtn.includes('year') &&
!activeBtn.includes('halfyear') ? 'active' : null}
                        onClick={() => {mapDataToLayer(filter, 'year')}}
                    >
                        Year
                    </button>
                </div>
            </div>
        )
    }
    else {
        return (
            <div className='loader'>
                <Rings height='100' width='100' color='black'
ariaLabel='loading'/>
            </div>
        )
    }
}

export default Map
```

Код клієнта, файл Map.scss

```
.filterBtns, .timeBtns{
  position: absolute;
```

```
  z-index: 5;
  display: flex;
  flex-direction: column;
  width: 200px;
  height: 200px;
}

.filterBtns, .timeBtns{
  button{
    height: 100%;
    cursor: pointer;
    background-color: rgba(26, 26, 26, 0.75);
    font-size: 1.5rem;
    color: rgba(255, 255, 255, 0.75);
    border: none;
    transition: 0.3s;
    padding: 10px 0;
    outline: none;
  }

  button:nth-child(1){
    border-top: 2px solid #9f0125;
  }

  button.active{
    color: #0AF520FF;
    text-shadow: 0 0 20px #0AF520FF;
  }

  button:hover{
    color: #9f0125;
    text-shadow: 0 0 20px #9f0125;
  }

  button + button{
    border-top: none;
  }
}

.filterBtns{
  button{
    border-right: 2px solid #9f0125;
  }
}

.timeBtns{
  button{
    border-left: 2px solid #9f0125;
  }
}

.filterBtns{
  top: 100%;
  left: 0;
  transform: translateY(-100%);
}

.timeBtns{
  top: 100%;
  left: 100%;
  transform: translateY(-100%) translateX(-100%);
}
```

```
.loader{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

ДОДАТОК Ж

Код клієнта, файл Statistics.js

```
import React, {useEffect, useState} from "react"
import { chartDataFetcher } from '../../heplers/mapHelper'
import { useParams } from "react-router-dom";
import {AreaChart, Area, XAxis, YAxis, Tooltip, Legend, CartesianGrid,
ResponsiveContainer} from 'recharts'
import './Statistics.scss'
import { Rings } from 'react-loader-spinner'
import useSWR from "swr";


const yAxisFormatter = (num) => {
    if (num >= 1000000) {
        return (num / 1000000).toFixed(1).replace(/\.0$/, '') + 'M';
    }
    if (num >= 1000) {
        return (num / 1000).toFixed(1).replace(/\.0$/, '') + 'K';
    }
    return num;
}

const dateFormatter = (date) => {
    return `${date.day.slice(0, 4)}/${date.day.slice(4, 6)}/${date.day.slice(6,
8)}`
}

const Statistics = () => {
    const { id } = useParams()
    const [charts, setCharts] = useState(false)
    const query = `https://calm-headland-
93075.herokuapp.com/v1/history?countryId=${id}`
    const { data } = useSWR(query, chartDataFetcher)

    useEffect(() => {
        if(data){
            if(data.timeline[0].day.includes(new Date().getFullYear())){
                data.timeline.reverse()
            }
            setCharts(true)
        }
    }, [data])

    if (charts){
        return (
            <div className='stats-wrapper'>
                <div className='stats-content-wrapper'>
                    <h1 className='stats-title'>{`${data.country}${data.province
=== 'none' ? '' : ` (${data.province})`}`}</h1>
                    <div className='stats-elements-wrapper'>
                        <p className='stats-element'><span className='stats-
element-title'>Total confirmed</span><br/>{data.stats[0].confirmed}</p>
                        <p className='stats-element'><span className='stats-
element-title'>Date</span><br/>{dateFormatter(data.timeline[data.timeline.length
- 1])}</p>
                        <p className='stats-element'><span className='stats-
element-title'>Total vaccinated</span><br/>{data.stats[0].vaccine}</p>
                        <p className='stats-element'><span className='stats-
element-title'>Total recovered</span><br/>{data.stats[0].recovered}</p>
                        <p className='stats-element'><span className='stats-
```

```
element-title'>Total deaths</span><br/>{data.stats[0].deaths}</p>
                        <p className='stats-element'><span className='stats-
element-title'>Fatality
rate</span><br/>{(data.stats[0].deaths/data.stats[0].confirmed*100).toFixed(2)}%
</p>
                    </div>
                    <ResponsiveContainer width='100%' height={500}>
                        <AreaChart data={data.timeline}>
                            <defs>
                                <linearGradient id="colorConfirmed" x1="0"
y1="0" x2="0" y2="1">
                                    <stop offset="5%" stopColor="#db2c2c"
stopOpacity={0.8}/>
                                    <stop offset="95%" stopColor="#db2c2c"
stopOpacity={0}/>
                                </linearGradient>
                                <linearGradient id="colorDead" x1="0" y1="0"
x2="0" y2="1">
                                    <stop offset="5%" stopColor="#000000"
stopOpacity={0.8}/>
                                    <stop offset="95%" stopColor="#000000"
stopOpacity={0}/>
                                </linearGradient>
                                <linearGradient id="colorRecovered" x1="0"
y1="0" x2="0" y2="1">
                                    <stop offset="5%" stopColor="#a117a6"
stopOpacity={0.8}/>
                                    <stop offset="95%" stopColor="#a117a6"
stopOpacity={0}/>
                                </linearGradient>
                                <linearGradient id="colorVaccinated" x1="0"
y1="0" x2="0" y2="1">
                                    <stop offset="5%" stopColor="#22a120"
stopOpacity={0.8}/>
                                    <stop offset="95%" stopColor="#22a120"
stopOpacity={0}/>
                                </linearGradient>
                            </defs>
                            <XAxis height={45} angle={-40} tickMargin={20}
dataKey={(data) => `${data.day.slice(0, 4)}/${data.day.slice(4,
6)}/${data.day.slice(6, 8)}`} />
                            <YAxis tickFormatter={yAxisFormatter}
tickCount={10}/>
                            <CartesianGrid vertical={false}/>
                            <Legend
                                iconSize={18}
                                iconType='cross'
                                align='center'
                                verticalAlign='top'
                                height={36}
                                formatter={(text) =>
`${text[0].toUpperCase()}${text.slice(1)}`}
                            />
                            <Tooltip
                                itemStyle={{fontSize: '18px', textTransform:
'capitalize'}}
                                labelStyle={{fontSize: '20px', textAlign:
'center', fontWeight: 700}}
                                contentStyle={{borderRadius: '16px'}}
                            />
                            <Area type="linear" dataKey="vaccine"
stroke="#22a120" fillOpacity={1} fill="url(#colorVaccinated)" />
                            <Area type="linear" dataKey="confirmed"
```

```
stroke="#db2c2c" fillOpacity={1} fill="url(#colorConfirmed)" />
                            <Area type="linear" dataKey="recovered"
stroke="#a117a6" fillOpacity={1} fill="url(#colorRecovered)" />
                            <Area type="linear" dataKey="deaths"
stroke="#000000" fillOpacity={1} fill="url(#colorDead)" />
                        </AreaChart>
                    </ResponsiveContainer>
                </div>
            </div>
        )
    }
    else {
        return (
            <div className='loader'>
                <Rings height='100' width='100' color='black'
ariaLabel='loading'/>
            </div>
        )
    }
}

export default Statistics
```

Код клієнта, файл Statistics.scss

```
.stats-wrapper{
  padding-top: 80px;
  padding-bottom: 100px;
  background-color: #e0e0e0;
  font-family: 'Baloo Thambi 2', sans-serif;
  min-height: 100vh;
}

.stats-content-wrapper{
  width: 80%;
  margin: 0 auto;
}

.stats-title{
  font-size: 36px;
  text-align: center;
  margin: 10px 0;
}

.stats-elements-wrapper{
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
  margin-bottom: 50px;
}

.stats-element{
  display: block;
  text-align: center;
  font-size: 24px;
  background-color: #fff;
  border-radius: 16px;
  width: 250px;
  flex-basis: 25%;
  margin-right: 50px;
  padding: 0 0 10px 0;
  box-shadow: 5px 5px 25px #cecdcd;
}
```

```
.stats-element:nth-child(3){
  margin-right: 0;
}

.stats-element:nth-child(6){
  margin-right: 0;
}

.stats-element-title{
  font-size: 16px;
  color: #999999;
}

.recharts-legend-item{
  font-size: 18px;
  vertical-align: middle;
  line-height: normal;
}

.loader{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

.chart{
  width: 90%;
  height: 400px;
  margin: 0 auto;
}
```

ДОДАТОК З

Код клієнта, файл Information.js

```
import React from "react"
import styles from './Information.module.scss'
import eyes_emoji from '../../assets/images/eyes-emoji.png'
import soap_emoji from '../../assets/images/soap_emoji.png'
import cat_emoji from '../../assets/images/cat_emoji.png'
import fruit_emoji from '../../assets/images/apple_emoji.png'
import antiseptic_emoji from '../../assets/images/antiseptic_emoji.webp'
import granny_emoji from '../../assets/images/granny_emoji.png'

const Information = () => {
    return (
        <div className={styles.wrapper}>
            <div className={styles.content_wrapper}>
                <h1 className={styles.title}>COVID19</h1>
                <h3 className={styles.subtitle}>How does the disease
spread?</h3>
                <div className={styles.info_blocks}>
                    <div className={styles.block}>
                        <img className={styles.block_img} src={eyes_emoji}
alt=""/>
                        <h4 className={styles.block_title}>Transmission
route</h4>
                        <p className={styles.block_text}>SARS-CoV-2 is
transmitted through exposure to infectious respiratory fluids which are released
during exhalation. This includes quiet breathing, speaking, singing, exercise,
coughing, sneezing, and the more vigorous the respiratory activity the more
particles are released in the form of droplets. For practical purposes, the
droplets can be divided into two categories: large droplets that settle out of
the air within seconds, and very small droplets (from now referred to as
aerosols) that can remain airborne for minutes to hours.</p>
                    </div>
                    <div className={styles.block}>
                        <img className={styles.block_img} src={soap_emoji}
alt=""/>
                        <h4 className={styles.block_title}>The virus does not
circulate in the air, but is transmitted from person to person</h4>
                        <p className={styles.block_text}>The virus cannot travel
a long distance. It exists only in the droplets which a person exhales while
coughing and sneezing. Distance guarantees protection. It is the factor breaking
an epidemic chain. Thus, if you are at a distance of more than 2 meters it is
impossible to get infected even from a person who is sick.</p>
                    </div>
                    <div className={styles.block}>
                        <img className={styles.block_img} src={cat_emoji}
alt=""/>
                        <h4 className={styles.block_title}>Can a person get
infected from animals?</h4>
                        <p className={styles.block_text}>At the moment there is
no official proof that pets, such as dogs and cats, can transmit the novel
coronavirus. So, after you contact pets, it is recommended to wash your hands
with soap. This will protect you from spreading the bacteria from pets.</p>
                    </div>
                    <div className={styles.block}>
                        <img className={styles.block_img} src={fruit_emoji}
alt=""/>
                        <h4 className={styles.block_title}>Is the virus
transmitted through fruit?</h4>
                        <p className={styles.block_text}>Fruit is the
```

```
environment that cannot provide lasting survival of the virus. There are no data
on transmitting the virus through fruit. We remind that it is necessary to wash
fruit thoroughly, but that is a standard recommendation.</p>
                    </div>
                    <div className={styles.block}>
                        <img className={styles.block_img} src={antiseptic_emoji}
alt=""/>
                        <h4 className={styles.block_title}>How long does the
virus survive on surfaces? What can kill the virus?</h4>
                        <p className={styles.block_text}>The virus can survive
on surfaces from 3 hours to several days. The term depends on a set of
conditions. For example, the type of surface, temperature and humidity of the
environment. That is why it is necessary to disinfect surfaces, handles,
equipment, etc.</p>
                    </div>
                    <div className={styles.block}>
                        <img className={styles.block_img} src={granny_emoji}
alt=""/>
                        <h4 className={styles.block_title}>Is it true that the
virus is transmitted only to old people?</h4>
                        <p className={styles.block_text}>People of any age can
be infected, so everyone should follow the virus protection measures. Elderly
people and patients with chronic diseases are at increased risk.</p>
                    </div>
                </div>
            </div>
        </div>
    )
}

export default Information
```

Код клієнта, файл Information.scss

```
.wrapper{
  padding-top: 80px;
  background-color: #e0e0e0;
  font-family: 'Baloo Thambi 2', sans-serif;
  min-height: 100vh;
}

.content_wrapper{
  width: 80%;
  margin: 0 auto;
}

.title{
  font-size: 36px;
  text-align: center;
}

.subtitle{
  font-size: 24px;
  text-align: center;
  color: #999999;
}

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

```
.block{
  flex-basis: 40%;
  margin-bottom: 30px;
}

.block_img{
  height: 200px;
}

.block_title{
  font-size: 20px;
}

.block_text{
  font-size: 18px;
}
```