Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка та дослідження системи моніторингу захворювань на COVID-19 на основі еволюційних моделей та поведінкових архітектурних рішень

Виконав: студент 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Ліщук Ю.І.
(підпис) (прізвище та ініціали)

Керівник _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Фриз М.Є.
(підпис) (прізвище та ініціали)

Тернопіль
2021

РЕФЕРАТ

Атестаційна робота магістра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2021. Сторінок 67, таблиць 0, рисунків 25, презентація.

Тема: Розробка та дослідження системи моніторингу захворювань на COVID-19 на основі еволюційних моделей та поведінкових архітектурних рішень.

В атестаційній роботі магістра висвітлено принципи роботи систем моніторингу та приклади систем відстеження захворюваності на COVID-19. Проаналізовано та створено еволюційну модель продукту. Досліджено мови програмування, підібрані для розробки програмного забезпечення. Зображено діаграму варіантів використання для користувача системи. Розроблено архітектуру продукту і протестовано кінцеву версію продукту на локальному хості.

Ключові слова: система моніторингу, еволюційна модель, діаграма варіантів використання, розробка front-end, поведінкова архітектура

ANNOTATION

Master's certification work. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2021. Pages 67, tables 0, figures 25, presentation.

Topic: Development and research of the monitoring system of COVID-19 cases based on evolutionary models and behavioral architectural solutions.

The master's attestation work highlights the principles of operation of monitoring systems and examples of tracking systems about COVID-19 morbidity. The evolutionary model of the product is analyzed and created. The programming languages selected for software development are studied. A use-case diagram based on user’s interaction with the system is shown. The product’s architecture was developed and the final version of the product was tested on the local host.

Keywords: monitoring system, evolutionary model, use-case diagram, front-end development, behavioral architecture

ЗМІСТ

ВСТУП

На даний момент навряд чи можна зустріти людину, яка не знала чи не читала новин про спалах коронавірусу штаму COVID-19, і похідної від поширення даного вірусу пандемії, що охопила земну кулю. Те, що спершу сприймалось вченими, як локальний випадок спалаху, який можна нейтралізувати в осередку зародження, перетворилось на загрозу для людських життєвих процесів, подекуди докорінно змінило порядки денні багатьох мешканців планети і посприяло розвитку раніше незнаних та новостворених професій та видозміні наявних. Більшість працівників у сфері програмування та інформаційних технологій працюють дистанційно, не втрачаючи продуктивності; студенти та викладачі використовують необхідне програмне забезпечення для віддаленого навчання, внаслідок чого з’явився попит на раніше мало використовувані застосунки.

З’явився попит і на апаратне забезпечення – окрім засобів для дистанційної діяльності, також використовуються прилади, що допомагають тяжко зараженим, що потрапили в реанімацію і перебувають в критичному стані, полегшити і – за успішного лікування – усунути симптоми, спричинені вірусом на кшталт апарату штучної вентиляції легень.

Спалах коронавірусу, попри свою загадковість і швидкість поширення, а тому й цілковиту необізнаність вчених, став серйозним рушієм наукового прогресу, складовими якого на чинному етапі є створення нового типу вакцин – мРНК-вакцин, котрі найбільш ефективно діють проти штамів вірусу, наявних зараз, а також було розроблено десятки інформаційних систем та програмних забезпечень, які мають за мету запобігати поширенню пандемії, допомагати лікарям та взаємодіяти із пацієнтами задля стеження за станами здоров’я, здійснювати інформаційно-виховну роботу з метою надання правил використання засобами індивідуального захисту, запобігання користувачів від зараження, оповіщення населення про необхідність захисту організму від COVID-19 та – за умови підхоплення вірусу – безпечного лікування симптомів без нагальної

потреби в стаціонарі та реанімації, і також надання населенню поточних даних про захворюваність на COVID-19.

Одним з таких нових видів програмного забезпечення є статистично-графічні системи стеження за захворюваністю на COVID-19.

Вказані вище системи дають змогу користувачам мережі Інтернет та працівникам державних структур оцінити щоденно оновлювані дані про кількість заражених, тих, хто одужав та мертвих внаслідок летальних ускладнень симптомів. Завдяки даним, які надає системи, кожен має можливість оцінити становище держави чи окремого регіону із заразністю і внаслідок цього реалізовувати заходи запобігання поширення вірусу на кшталт посилення чи послаблення заходів карантину, проведення процедури одинарної чи масової вакцинації населення, тощо.

Якщо говорити про дисципліну інженерії програмного забезпечення, кінцевий програмний продукт повинен надавати дані про захворюваність для оповіщення населення про світовий стан поширення вірусу та ефективно працювати на особистих персональних комп’ютерах. Власне, питання вирішення цих проблем і вирішується безпосередньо у дипломній роботі.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні відомості про системи моніторингу. Огляд систем моніторингу захворюваності на COVID-19

Система моніторингу – це різновид програмного забезпечення, що допомагає системним адміністраторам слідкувати за власною сферою діяльності та за потреби відображати дані вказаної сфери для користувачів (рис.1). Завдяки цим засобам є можливість стежити за перебігом даних, станом системи та вчасно реагувати на випадок можливих збоїв. Системи моніторингу ранжуються від систем з відкритим доступом до професійних, що не надають даних до загального відома.

Склад системи моніторингу визначається такими чинниками:

- функціональне призначення системи моніторингу;
- ціль та область використання;
- функції обробки інформації, покладені на систему моніторингу і визначені користувачем.

Перевагами системи моніторингу є:

- можливість стежити за власною системою в режимі реального часу через єдиний пристрій;
- миттєве оповіщення адміністратора у випадку наявності помилки і поступове налагодження роботи системи;
- низька ресурсозатратність, що дозволяє зекономити вдосталь часу і засобів адміністраторові.

Прикладами загальних систем моніторингу є програмні забезпечення:

- Atera
- LogicMonitor
- Microsoft Endpoint Manager
- PRTG Network Monitor

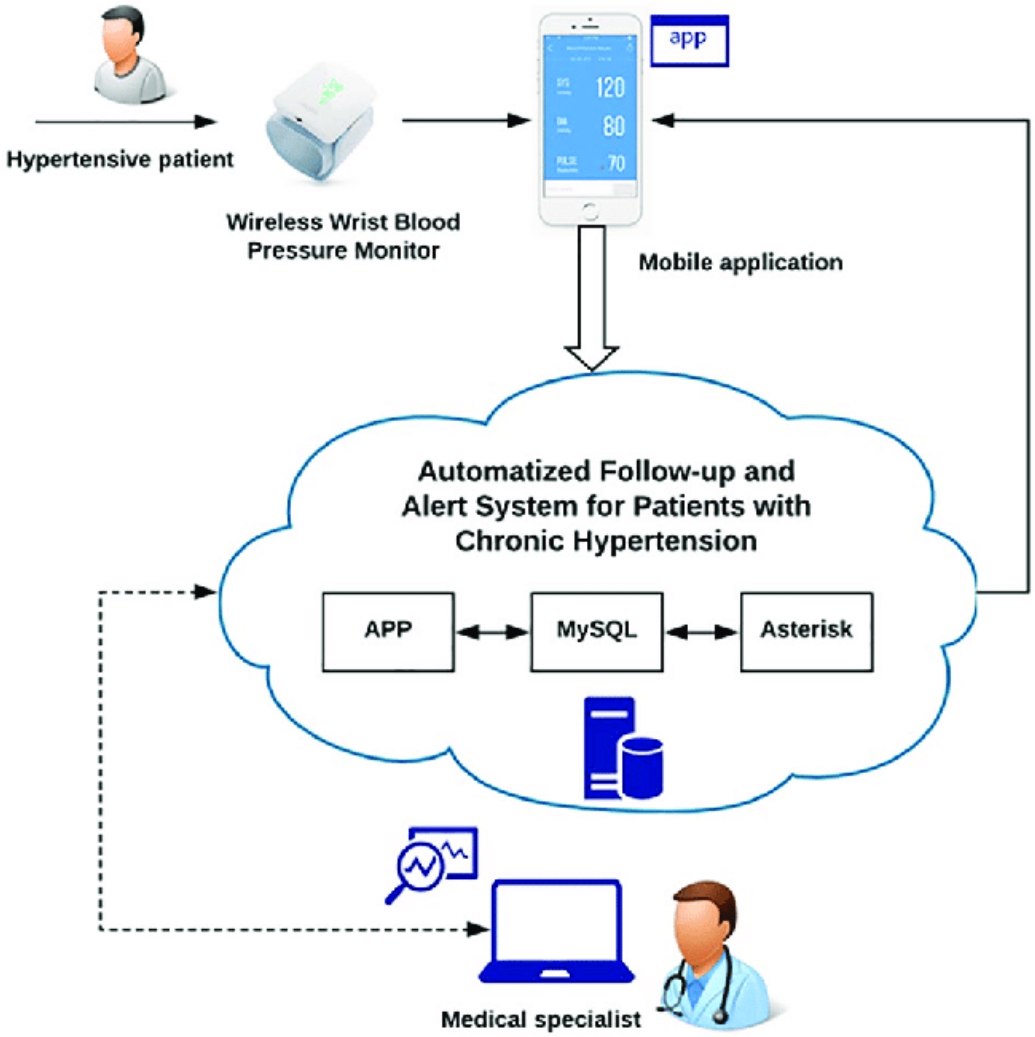

Рисунок 1 – Зразок системи моніторингу на прикладі механізму оповіщення пацієнтів з хронічним високим тиском

Системи контролю зазвичай розділяються на два типи:

- Активний моніторинг – стеження за інфраструктурою, даними і сервісами наживо, можливість визначення невідповідності параметрів роботи компонентів відносно чинних показників;
- Проактивний моніторинг – можливість побудови інфраструктури, оцінки внесення змін, прогнозу і запобігання недолікам завчасно відносно історичних даних [1].

В наш час тяжко уявити підприємства чи компанії, які би не використовували технології. Проте будь-яка установа, котра містить в собі певні технологічні устаткування, зобов'язана керувати операціями і слідкувати за здійсненням робіт задля недопущення помилок, що призведуть до негативного впливу на користувачів – або ж за умови наявності великої бази даних з'являється необхідність їхнього пропорційного розподілу, сортування заради зручності використання та відображення. Для згаданих вище випадків і існують подібні різновиди програм.

Системи моніторингу захворюваності на COVID-19 – це різновид програмного забезпечення, завдяки якому системний адміністратор надає можливість користувачам оцінювати завдяки статистичним та графічним даним ситуацію відносно поширення COVID-19 у світі чи в межах окремої країни/регіону. Переважно це – мережеве програмне забезпечення зі зображеними статистикою та графіками для кращого відображення ситуації, іноді до складу ПЗ додаються інтерактивні карти.

З початком пандемії в мережі з'явилося безліч подібних різновидів програмного забезпечення, які використовують спеціально створені для збору інформації звідти обширні бази даних. Дані системи використовуються насамперед державними та місцевими урядовими установами задля вирішення питань відносно реалізації заходів щодо стримування зараження від COVID-19 в межах адміністративних регіонів. Наприклад, під час осінньо-зимової хвилі в 2020 році було прийнято рішення ділити адміністративно-територіальні одиниці України на чотири зони зараження, кожна з яких мала власні критерії для попадання території під певну категорію та обмеження чи полегшення, які накладались на відповідний регіон, якщо показники даних критеріїв зростали або спадали.

Одним із яскравих прикладів програмного забезпечення для надання світових та локальних даних по захворюваності на COVID-19 задля відстеження кількості заражених є система моніторингу поширення епідемії COVID-19 на сайті Ради національної безпеки та оборони України (рис.2):

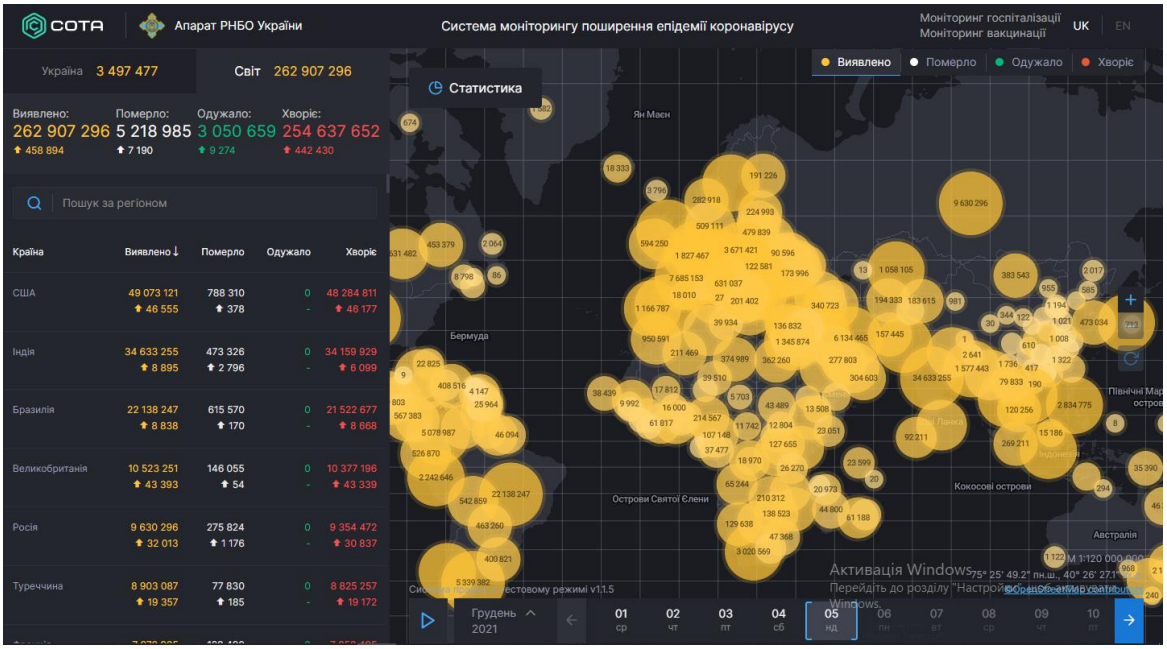

Рисунок 2 – Система моніторингу поширення епідемії COVID-19 Ради національної безпеки та оборони України

Вказана система дає користувачу можливість ознайомитись із статистичними даними відносно кожної із світових країн, а також України – як цілком, так і кожної з обласних адміністративно-територіальних одиниць держави – за винятком окупованих Росією територій Луганської та Донецької областей і Автономної Республіки Крим. База даних забезпечення оновлюється щоденно і оперативно.

Система стеження надає дані щодо:

- загальної кількості захворюваних на країну;
- кількості мертвих;
- кількості тих, хто одужав – діє тільки в Україні;
- кількості хворих – світова спільнота в кожній із країн – щонайменше з весни 2021 року – надає дані в рамках хворих за рахунок об’єднання показників чинних хворих і людей, що перехворіли симптомами коронавірусу.

Портал Ради національної безпеки і оборони України також надає можливість ознайомитись із системами спостереження за темпами госпіталізації та вакцинації мешканців України. Дані системи перебувають в тестовому режимі, і попри це надають щоденно оновлювану інформацію про стан медичної інфраструктури, спрямованої на боротьбу з коронавірусом і лікуванням жителів областей, зокрема – показом темпів вакцинації і кількості вакцинованих з урахуванням виду вакцини і відсоткового співвідношення видів вакцини до кількості вакцинованих – першою дозою, другою дозою чи загальної. Дані про госпіталізацію та вакцинацію надаються у форматах карт, графіків і таблиць (рис.3):

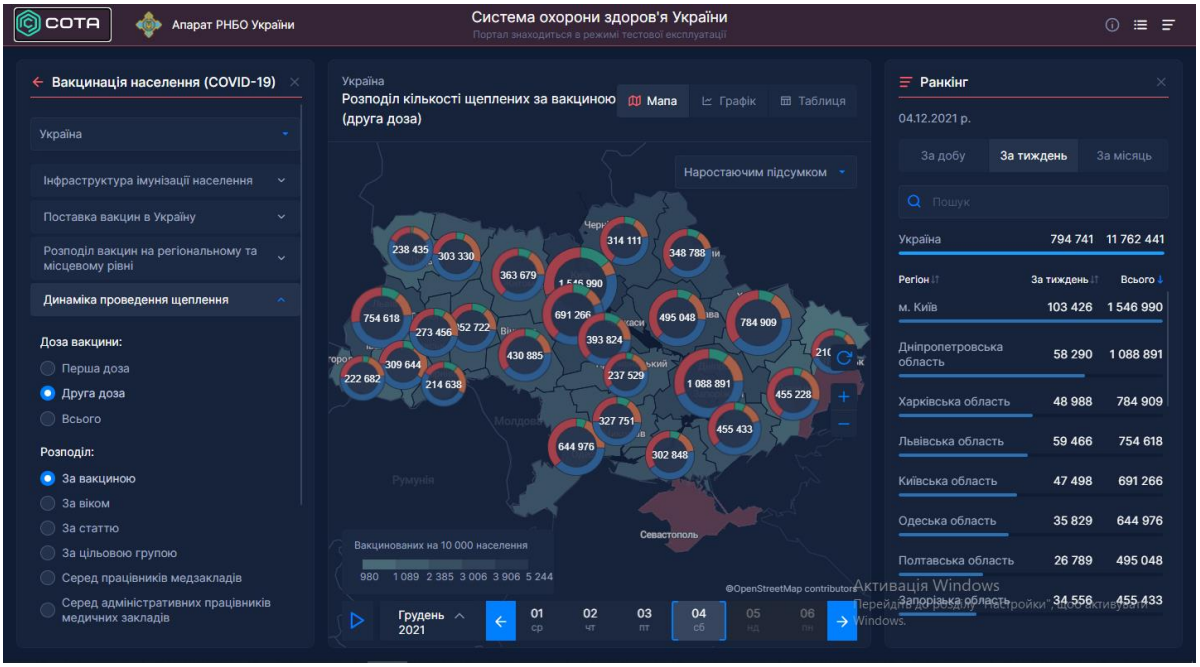

Рисунок 3 – Система моніторингу вакцинації від COVID-19 Ради національної безпеки та оборони України

1.2 Вибір моделі розробки системи моніторингу

Комп'ютерна система, неначе живий організм, має власний життєвий цикл, протягом якого продукт проходить відповідні етапи, найбільш вагомими з яких є:

- аналізу вимог;
- проєктування;
- програмування;
- тестування/налагодження;
- експлуатація, супровід і підтримка [2].

Проте існує певна варіативність у проходженні даних етапів і для того, аби їх відображати і краще ними керувати, було сформульовано моделі життєвого циклу – структури, що складаються із процесів, робіт та задач, які охоплюють життя системи від визначення вимог до неї до припинення її використання [3]. До найбільш відомих моделей відносять каскадну, спіральну та еволюційну моделі.

В каскадній моделі (рис.4) реалізовується послідовне і одноразове виконання всіх етапів проекту з попереднім плануванням і визначенням вимог. Перехід поміж етапами відбувається лише за умови повноцінного і безпомилкового завершення робіт на попередньому етапі. Кожен етап завершується випуском документації, якої вистачає для того, щоб обробку могли продовжити інші команди розробників.

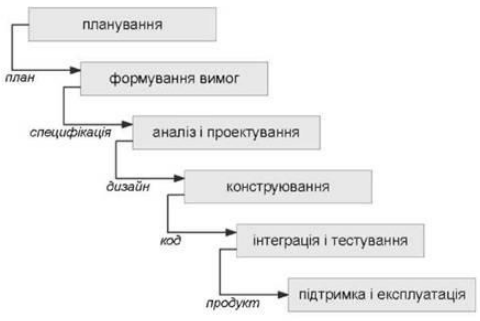

Рисунок 4 – Схема каскадної моделі

Еволюційна (ітераційна) модель (рис.5) передбачає розбиття проекту на частини (етапи, ітерації) і проходження етапів життєвого циклу на кожному з них. Кожен етап є закінченим сам по собі, а сукупність етапів формує кінцевий результат. Тестування в такому випадку можна починати на ранніх стадіях життєвого циклу [4].

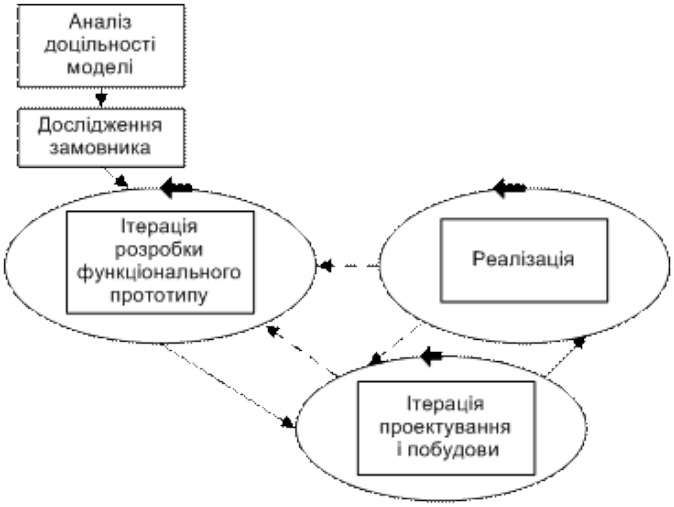

Рисунок 5 – Схема еволюційної моделі

Спіральна модель фактично є одним із видів еволюційної моделі. При реалізації даного типу моделі кожен з етапів життєвого циклу відповідає за один виток спіралі, і на кожному з них відбуваються проектування, конструювання, дизайн, тестування, тощо. Даний процес відображає суть назви: продукт проходить по одному витку (цикл) спіралі задля досягнення остаточного результату і кінцевої реалізації. Але в даному випадку не обов'язково, щоб один і той же перелік процесів повторявся кожного витка (рис.6).

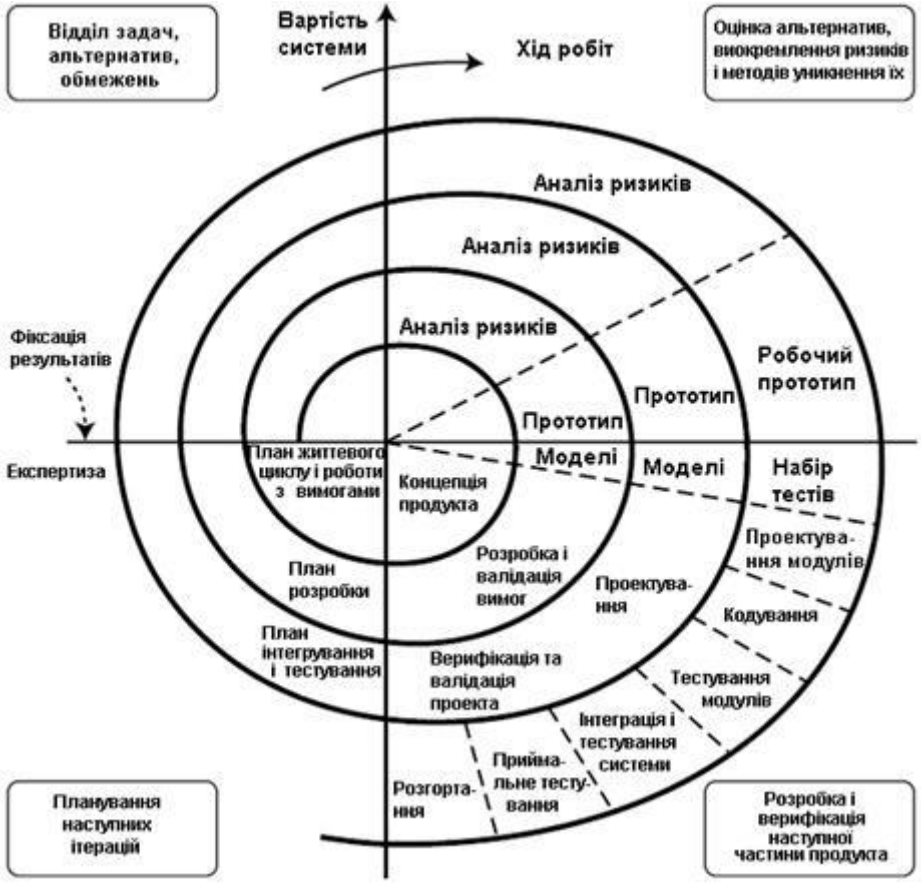

Рисунок 6 – Схема спіральної моделі

Як і у вище зазначеній спіральній моделі, у еволюційній кожен наступний етап наближає до бажаного результату або протягом вказаної ітерації уточнюються вимоги, і тому будь-який поточний етап може виявитися останнім або черговим на шляху до завершення – в цьому є велика перевага еволюційної моделі над каскадною, де користувач отримає доступ до продукту лише на останньому етапі життєвого циклу.

Також перевагами еволюційної моделі є можливість користувача випробовувати продукт, розроблений не до кінця – відповідно, можливість здійснювати за необхідності оперативні зміни до структури проєкту або надавати кінцеву версію на стадії, де користувач буде задоволеним результатом роботи; мінімізація помилок, оскільки всі частини протягом здійснення етапів життєвого циклу системи проходять постійне і ретельне тестування, а також зниження вартості розробки.

Еволюційна модель має і свої недоліки. До недоліків відноситься запізнення з виходом цілісного програмного забезпечення, пов’язане із пропозиціями змін від користувачів протягом розробки, незручність для малих проєктів, і складність розбиття проблеми на частини, що зможуть задовольнити користувача.

Еволюційна модель корисна для розробки в багатьох випадках, найбільш вагомими з яких є:

- використання моделі у великих проєктах, де можна легко знайти частину для покрокового виконання; еволюційна модель використовується для того, аби користувачі могли використовувати почастинно особливості програми замість очікування на готовий продукт;
- еволюційна модель також є корисною для об’єктно-орієнтованого програмування, адже вся розробка розбита на різні частини/об’єкти.

Для розвитку нашої системи, якій завжди буде необхідно оновлюватись протягом власного життєвого циклу, і якій не необхідно мати єдину останню версію, а завше розвиватись, вдосконалюватись, задовольняти зростаючі потреби користувачів, допоки існує потреба в програмному забезпеченні, еволюційна модель підходить найкраще.

1.3 Опис і аналіз предметної області

Створений проєкт буде містити реалізацію системи, спроєктованої для вирішення поставленого завдання – стеження та обробки даних щодо захворюваності на COVID-19 (рис.7):

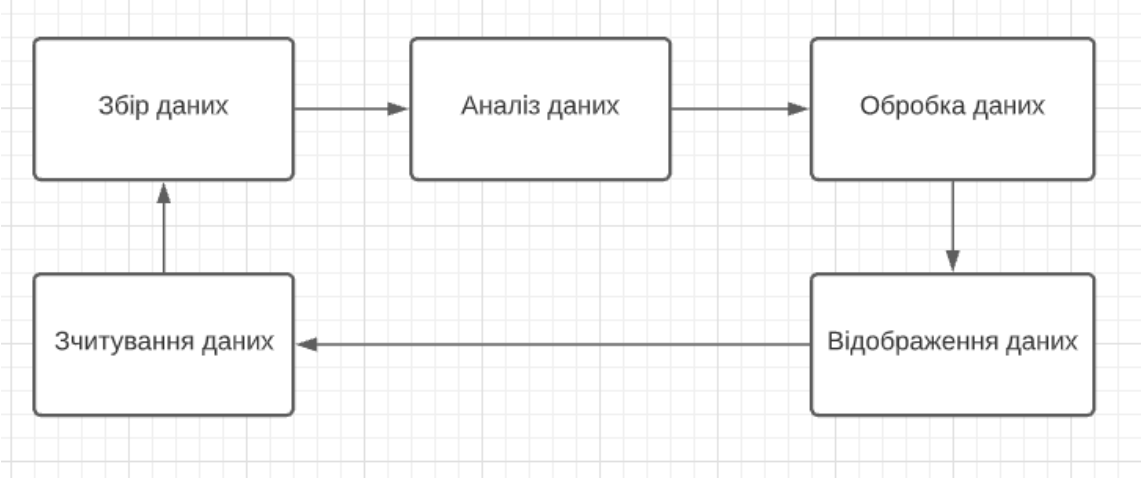

Рисунок 7 – Схема проєкту

Перший крок – це збір вхідних даних для реалізації системи. Під час даного кроку ми збираємо базу даних, яка може містити різноманіття об’єктів на кшталт кількості захворюваних за останні двадцять чотири години, число вакцинованих чи кількість заражених суто штамом «Дельта».

Другий крок – це аналіз даних. Протягом цього кроку ми відбираємо найбільш важливі для нас об’єкти даних, які знадобляться при розробці програмного забезпечення. Зазвичай такими об’єктами є загальна кількість захворюваних, кількість смертей та кількість тих, хто одужав. Варто відзначити, що станом на сьогодні бази даних щодо хворих на симптоми коронавірусу, які знаходяться у відкритому доступі, на зовнішніх джерелах, в якості прикладних програмних інтерфейсів – API, більше не відображають показник кількості тих, хто одужав, поєднуючи даний параметр із кількістю наразі хворих, і відповідно об’єднавши обидва параметри в один – кількість живих носіїв вірусу. Починаючи з березня 2021 року, дана практика стала поширеною в світових базах даних, тому при розробці програмного забезпечення ми будемо використовувати нижче перелічені параметри:

- кількість глобальних підтверджених заражень;
- кількість глобальних смертей;
- кількість глобальних живих носіїв вірусу;
- країна/регіон з даними щодо захворюваності:

- кількість підтверджених заражень в межах країни;
- кількість живих носіїв вірусу в межах країни;
- кількість смертей в межах країни.

Третій крок – обробка даних – передбачає власне розробку та верстку системи з використанням видобутих із зовнішнього джерела даних. Протягом згаданого вище кроку ми розписуємо кістяк програмного забезпечення.

Четвертий крок – відображення даних – передбачає компіляцію і розгортання системи для зовнішнього використання, під час якого опрацьовані нами дані відображаються в різного роду форматах: статистичному або за допомогою інтерактивної карти.

П’ятий крок – зчитування даних – є завершальним в даному циклі і передбачає експлуатацію і використання системи моніторингу захворюваності з ймовірним подальшим опрацюванням і поновленням системи до моменту повноцінного завершення життєвого циклу програмного забезпечення.

Враховуючи щоденне оновлення даних, що надаються із зовнішніх джерел, надана вище схема буде циклічною і буде діяти протягом всього життєвого циклу системи.

1.4 Поведінкове моделювання та аналіз вимог

Поведінкове моделювання допомагає розробникам вияснити, як програмне забезпечення буде взаємодіяти в тому чи іншому випадку із користувачем.

Поведінка системи за даного підходу буде проявлятись у певній дії в певний період часу. Однією із складових поведінкового моделювання є побудова діаграми варіантів використання.

Основними елементами діаграми є учасник (актор) і варіант використання.

Учасник – це роль або зв’язаність ролей для взаємодії із системою.

Учасником може бути як людина, так і інша система. Графічно на діаграмі варіантів використання позначається «людиною».

Варіант використання (ВВ) – опис події, яку виконує система і призводить до необхідного учасникові результату. ВВ специфікує очікувану поведінку системи, описує послідовності дій, які вона здійснює для досягнення дійовою особою певного результату. ВВ застосовується для вираження необхідної поведінки розроблювальної системи, без описування реалізації цієї поведінки [5].

Графічно варіант використання на діаграмі позначається еліпсом із вписаним всередину описом.

Цільовий користувач даного програмного забезпечення – це пересічний громадянин, що зазвичай тяжко вбирає інформацію, тому дуже важливо, щоб користувацький інтерфейс був зручним, інтуїтивно зрозумілим.

Система моніторингу має відповідати таким нефункціональним вимогам:

- має працювати у веб-браузерах серії Google Chrome, Microsoft Edge найновіших версій;
- повинна мати простий та зрозумілий інтерфейс;

Система моніторингу має відповідати таким функціональним вимогам:

- зчитування даних у статистичному вигляді;
- взаємодія з даними за допомогою інтерактивної карти;
- відображення коректних даних в інтерфейсі;
- вибірка даних за назвою країни.

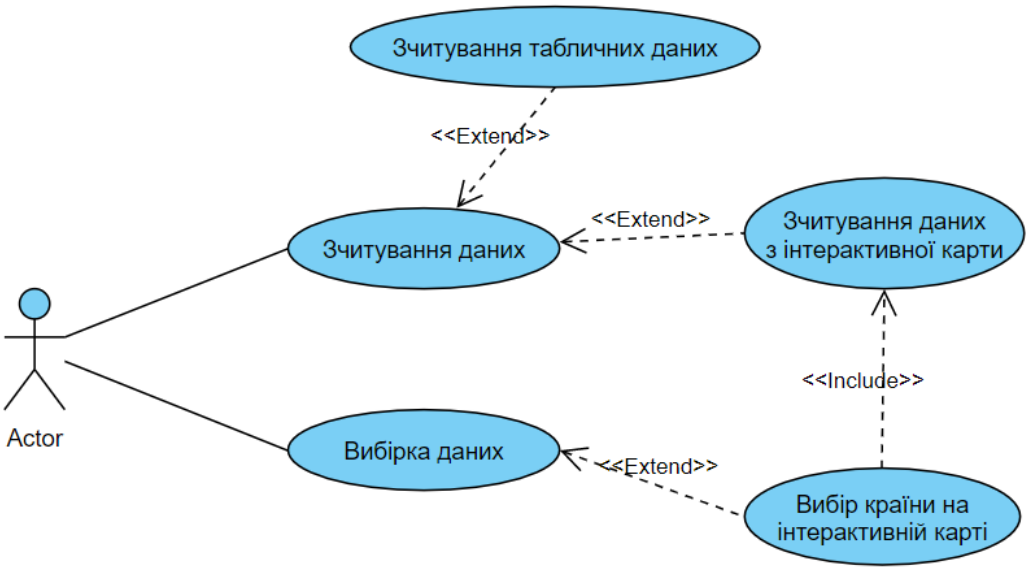

Рисунок 6 – Діаграма варіантів використання

Система стеження передбачає тільки одного учасника – особу, якою може бути будь-який відвідувач мережі Інтернет. Програмне забезпечення задля зрозумілості та простоти не навантажуватиме користувача зайвими деталями, а надаватиме корисний мінімум, якого цілком буде достатньо для того, аби ознайомити з чинною ситуацією зі світовим поширенням коронавірусу – зчитування даних в статистичному (у формі таблиці) та інтерактивному (за допомогою карти) виглядах, а також вибір на карті бажаної країни, після чого в інформаційному вікні буде подано найбільш достовірні дані про стан зараженості.

1.5 Обґрунтування вибору середовища розробки

При виборі засобів для вирішення задач в галузі програмування найчастіше використовують текстові редактори коду. Редактори можуть бути як окремими додатками, так і інтегрованими середовищами розробки – IDE. Є зручними

засобами для компіляції, налагодження, перетворення коду та інших взаємодій, необхідних при конструюванні програмного забезпечення.

При виборі середовища розробки було здійснено опору на можливості апаратного забезпечення, швидкодію і легкість при майбутній роботі в рамках середовища, внаслідок чого після аналізу було обрано редактор Visual Studio Code.

Visual Studio Code було представлено компанією Microsoft у квітні 2015. Програма стала першим кроссплатформним продуктом лінійки Visual Studio – тобто могла працювати як на Windows, так і на Linux чи Mac OS X (рис. 7). Вагомою перевагою даного середовища також є безкоштовне розповсюдження, що робить його доступним для кожного розробника, незалежно від рівня чи досвіду програмування – це підтверджується наступним фактом: в рамках опитування розробників на сайті Stack Overflow у 2021 році Visual Studio Code був визнаним найбільш популярним для користування середовищем серед розробників, де 70% із 82000 респондентів вказали на користування редактором [6].

Рисунок 7 – Логотип Visual Studio Code

Visual Studio Code є редактором коду, який взаємодіє з багатьма мовами програмування на кшталт Java, JavaScript, Go, Python і C++. Середовище дає можливість користувачам розширювати себе крізь центральний репозиторій завдяки додаткам до редактора чи підтримкам мов програмування, містить розширення, що робить редактор альтернативою для середовищ веб-розробки, адже у Visual Studio Code можна синхронізувати код між редактором і сервером без під’єднання додаткових програмних забезпечень.

Також Visual Studio Code від конкурентів відрізняє наявність вбудованого відлагоджувача (дебаґера), завдяки чому можна виправляти помилки в коді в межах самого редактора. Окрім цього, наявна і вбудована консоль, де буде виводитись результат роботи чи повідомлення з помилкою за умови збоїв у роботі коду. Дебаґер можна налаштовувати під різні мови і поставлені задачі.

Середовище також містить засоби для взаємодії з Git і можливість встановлювати розширення в рамках власне середовища, що дозволяє збільшувати кількість мов програмування для взаємодії, і відповідно надає більше опцій для функціонування Visual Studio Code, що вкупі з його ергономічним і простим у використанні інтерфейсом робить його популярним як серед досвідчених розробників, так і серед початківців.

Для роботи з Visual Studio Code необхідно базове знання англійської мови і – за потреби – доступ до мережі Інтернет.

1.6 Огляд мови програмування і технологій розробки

Враховуючи вище зазначені вимоги до програмного забезпечення, було вирішено використати для написання системи мову JavaScript з похідним залученням мов розмітки HTML і CSS.

HTML (HyperText Markup Language) — стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та подається на екрані у стандартному для людини вигляді. Попри те, що HTML — штучна комп'ютерна мова, вона не є мовою програмування [7].

HTML дає можливість надавати форму структурі документа в рамках Всесвітньої мережі. Фінальна версія документу може містити різного роду

елементи: ілюстрації, аудіо, відео, гіперпосилання, інтерактивні об’єкти. Мова HTML містить у собі засоби для визначення декількох рівнів заголовків, виділень шрифта та багато додаткових можливостей; мова також дає можливість отримувати інформацію із зовнішніх джерел в межах мережі Інтернет через гіперпосилання.

Кожен HTML документ складається з двох частин - імені та розширення (.html чи .htm). Наприклад: page.html, index.htm тощо [8]. За основу моделі розмітки документів у HTML прийнята тегова модель, що описує документ як сукупність контейнерів, кожен з яких починається і закінчується тегами.

Теги HTML-документів в основному є простими і зрозумілими для використання, оскільки вони створені за допомогою загальновживаних слів англійської мови, зрозумілих скорочень і позначень [9].

Чотири основні компоненти є базовими при верстці розмітки HTML: елементи включно з їхніми типами атрибутів, базові типи даних, символьні об’єкти та декларація типу документа – тег <!DOCTYPE>, що вказується на початку HTML-документа.

Найбільш ключовими базовими компонентами мови HTML, завдяки яким і існує кістяк веб-документу, є елементи. Будь-який елемент містить три основні складові: теги – початковий і завершальний, вміст та атрибути.

Існують декілька типів елементів розмітки HTML, серед яких виділяються:

Елементи структурної розмітки – використовуються для опису семантики тексту і не змінюють візуальне відтворення тексту, хоча більшість браузерів містять стандартні стилі форматування для кожного елемента.

Елементи візуальної розмітки – застосовуються для опису візуальних ефектів тексту, не зазначаючи під час цього функції тексту вмісту. Остання чинна специфікація HTML визначає більшість цих елементів не рекомендованими для застосування у розмітці.

Елементи розмітки гіпертексту – застосовуються задля поєднання частин документа з іншими, переважно зовнішніми документами [10].

CSS (Cascading Style Sheets) — спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних. Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

CSS використовується авторами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки [11].

У такого способу форматування є кілька переваг:

- теги не дублюються;
- документ легше обслуговувати;
- зовнішній вигляд сайту можна змінювати централізовано, а не редагувати сайти посторінково.

Якщо HTML структуризує вміст на сторінці, то CSS дозволяє його форматувати, аби зробити більш привабливим і цікавим для користувачів. Насправді веб-розробники використовували тільки HTML для розмітки оформлення на початку розвитку мережевих технологій верстки – можна було виділити параграф, заголовок, змінити стиль тексту – і більш нічого не було потрібно. Та дані можливості і потреби поступово розширювались, а з еволюцією Інтернет-технологій ставало тяжче вміщувати атрибути оформлення в межах HTML, і з часом CSS вирішив цю проблему, вмістивши і уніфікувавши у власній структурі єдину базу розмітки, що полегшила життя верстальщикам і спростила роботу з веб-документами [12].

Мова CSS дозволяє адаптувати вміст до відображення на різних пристроях і в різних умовах, себто робить документ респонзитивним – наприклад, один і той же файл може відображатись на екрані монітора ПК чи мобільного пристрою без втрати у функціональності.

Мова CSS швидко стала стандартною у веб-розробці, оскільки дозволяє швидко змінити вигляд сайту, не змушуючи при цьому використовувати для верстки більш складні мови програмування. Варто лише ознайомитись із

найпростішими правилами CSS, і можна легко зібрати цілком симпатичний сайт з необхідним вмістом. Простота забезпечується за рахунок зрозумілого і зручного синтаксису, який використовує можливий мінімум слів англійської мови для подання складових стилю. Кожен стиль містить у своєму тілі список правил. А кожне із правил у відповідь містить від одного до кількох селекторів – посилань на елементи HTML, над якими буде вестись робота – та блок визначення, що складається із оточених фігурними дужками властивостей – характеристик елемента, який потрібно видозмінити – і значень – цифрового чи текстового позначення для вибраної властивості. Синтаксис правила в CSS виглядає приблизно так (рис.8):

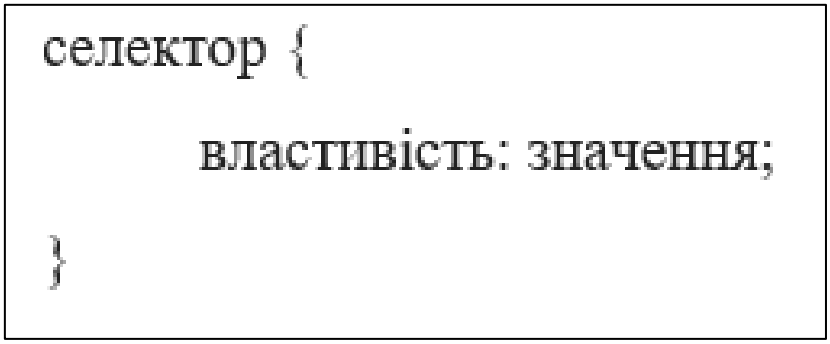

Рисунок 8 – Зразок синтаксису правила CSS

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки [13].

З появою Node.js мова програмування JavaScript стала також використовуватись для програмування на боці сервера (подібно до таких мов програмування, як Java, PHP), розробки ігор, стаціонарних та мобільних додатків, сценаріїв в прикладному програмному забезпеченні, всередині PDF-документів тощо.

JavaScript містить набір вбудованих операцій, які не обов'язково є функціями або методами, а також набір вбудованих операторів, що управляють логікою виконання програм [14].

JavaScript на даний момент є однією з найбільш популярних мов серед розробників програмного забезпечення. Хоча спочатку багато досвідчених розробників скептично відносилися до мови, аудиторія якої раніше переважно складалася з програмістів-любителів і новачків, але поява технології AJAX («Asynchronous JavaScript And XML», себто «асинхронний JavaScript і XML») змінила ситуацію та змусила професійну спільноту звернути увагу на мову програмування. Внаслідок цього були розроблені та покращені безліч практик використання JavaScript (зокрема тестування та налагодження), створено бібліотеки та фреймворки. І станом на 2021 рік, згідно з даними опитування розробників на сайті Stack Overflow, JavaScript було обрано найбільш популярною мовою у середовищі програмної інженерії з показником в 65% з-поміж близько 83000 учасників [6].

JavaScript відкриває безліч можливостей для розробників та використовується в більшості Інтернет-проектів, браузерних додатках, мобільних програмах та іграх, дозволяє створювати унікальні інтерактивні веб-сторінки з надзвичайно широкими можливостями. На сьогоднішній день розробка сучасних веб-сайтів не обходиться без використання JavaScript, а затребуваність ринку у спеціалістах, які володіють програмуванням JavaScript, стабільно зростає. Варто зазначити, що мову JavaScript вивчити не так важко, як може здатись на перший погляд – і завдяки своєму синтаксису, і завдяки легкості при верстці.

Node.js являє собою платформу для коду, написаного на JavaScript, спроектовану для побудови масштабованих мережевих додатків, і яке надає можливість одразу на серверах виконувати команди, прописані в коді JavaScript [15]. Платформа, окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських та серверних програм [16]. Node.js вміє також взаємодіяти із зовнішніми бібліотеками і виконувати роль веб-сервера, що й було застосовано під час роботи над програмним забезпеченням в рамках написання кваліфікаційної роботи. І оскільки під час розробки програм необхідно забезпечувати аналіз великого переліку паралельних запитів, у Node.js для цього використовується модель асинхронного вводу-виводу.

Node.js схожий по суті на фреймворки AnyEvent – мова Perl, Event Machine – мова Ruby і Twisted – мова Python, але цикл обробки подій у Node.js не

надається на показ розробникові і своєю поведінкою нагадує обробку подій у браузерному веб-застосунку [16].

Node.js керується спеціально створеним Технічним керівним комітетом (Technical Steering Committee (TSC)), що є відповідальним за найвищий рівень координування проектом.

TSC має остаточне авторство над цим проектом, включаючи:

- технічне спрямування;
- управління проектом та процесом;
- політику співпраці;
- хостинг GitHub-репозиторіїв;
- керівництва з поведінки;
- підтримку списку додаткових співавторів.

ReactJS, або React – це декларативна і гнучка JavaScript-бібліотека з відкритим програмним кодом, що використовується для створення компонентів користувацького інтерфейсу з повторним використанням, що представляють дані, змінні з плином часу [17].

Станом на 2021 рік React є не тільки найбільш популярною бібліотекою в межах JavaScript, а й найбільш використовуваним веб-фреймворком, обійшовши jQuery – згідно з даними опитування розробників Stack Overflow, з-поміж 67500 респондентів, 50000 з яких – професіонали, 40% обрали React[6]. Відомість і вживаність бібліотеки можна пояснити хоча би назвами великих компаній і стартапів, коди систем яких містять React – Netflix, Airbnb, Instagram, New York Times, тощо. Бібліотека надає безліч переваг розробникам, завдяки чому її продовжують обирати більше й більше осіб, зацікавлених у програмуванні – і в цьому грає роль ряд ключових особливостей:

- легке створення динамічних програм: при використанні React код хоч і стає візуально меншим, але при цьому надає більше функціональних можливостей;
- покращення продуктивності: React використовує віртуальну об’єктну модель документа (Virtual DOM), завдяки чому веб-застосунки створюються швидше [18];

- повторне використання компонентів, що кардинально зменшує на краще час розробки програм;
- одностороння передача даних: завдяки даній особливості легше виявляти помилки в додатку;
- React легко вивчити, оскільки він переважно вміщує в собі базиси HTML і JS;
- бібліотеку можна використовувати для розробки як мережевих, так і мобільних програм.

Однією з основних характеристик React є JSX – синтаксне розширення JavaScript, що використовується спільно з бібліотекою для опису вигляду користувацького інтерфейсу – це досягається вкладенням HTML-структур в JavaScript-синтаксис (рис.9) у тому ж документі, що містить код даної мови програмування.

```
const name = 'Simplilearn';
const greet = <h1>Hello, {name}</h1>;
```

Рисунок 9 – Приклад реалізації JSX

2 РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ЗАХВОРЮВАНЬ

2.1 Огляд середовища розробки

Редактор Visual Studio Code є безкоштовним і дуже зручним у використанні – і неодмінно допоможе розробникам та верстальникам будь-якого рівня, адже він підійде і початківцям: інтерфейс середовища є інтуїтивно простим і зрозумілим; і досвідченим програмістам: Visual Studio Code містить велику кількість додаткових опцій, а тому надає більше можливостей при розробці на різних мовах програмування.

Після установки програмного середовища при відкритті Visual Studio Code виводиться вітальна вкладка, за допомогою якої можна по-різному взаємодіяти із редактором: почати новий файл, створити чи відкрити папку з проектом або клонувати репозиторій Git; обрати необхідний з переліку нещодавно виконуваних проектів, а також переходити через корисні посилання, налаштовувати зовнішній вигляд Visual Studio Code на власний розсуд чи дізнаватись про комбінації клавіш, які полегшать розробникам роботу і зекономлять час при верстці (рис.10). Якщо вам не буде більше необхідна стартова вкладка, варіант з показом можна вимкнути, натиснувши на галочку біля опції «Показувати стартову вкладку при запуску» внизу програми («Show welcome page on startup»). Змінити параметр можна завжди за допомогою команди Help > Get Started.

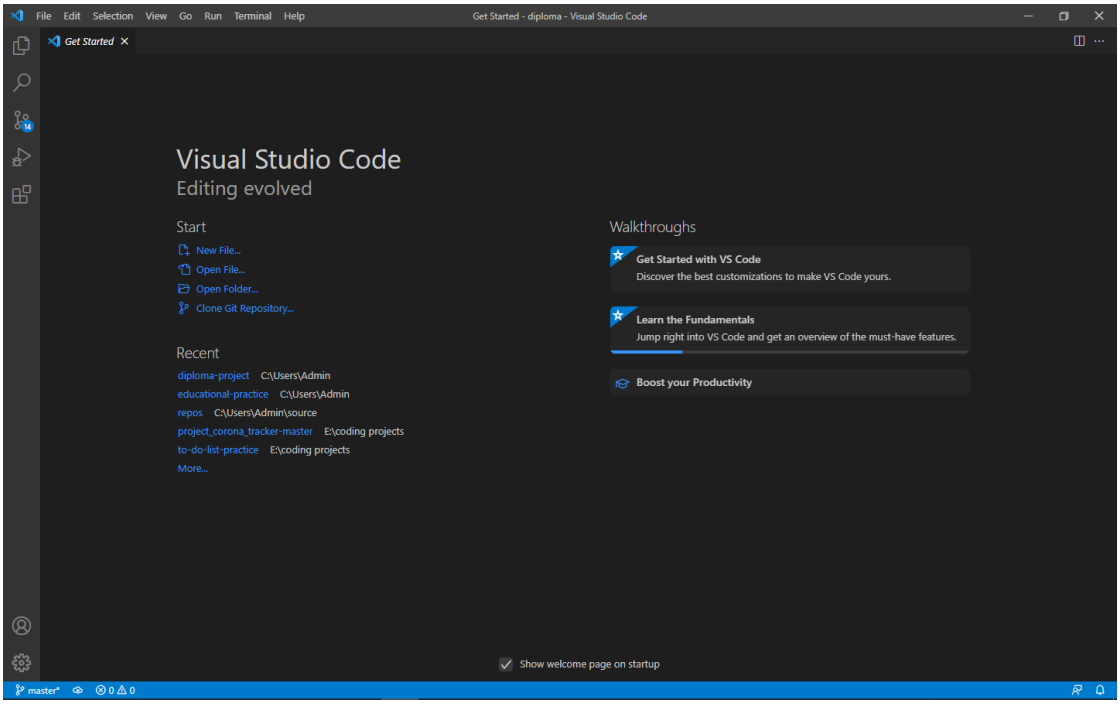

Рисунок 10 – Стартова вкладка Visual Studio Code

По лівий бік екрану можна побачити бічне активне меню, що складається із п’яти частин:

- провідника – дана частина відкривається за замовчуванням при запуску програми. В ній виводяться список відкритих файлів і каталог чинної папки;
- пошук – тут можна вводити частини коду, а редактор покаже, де саме у вихідному коді вони знаходяться. Також є наявною функція заміни необхідної частини коду за умови внесення змін в проекті чи виправлення помилок через поле заміни;
- Git-панель контролю, де відображається список змін проекту з часу останнього внесення правок – кількість змін видається в синьому колі в правому нижньому куті іконки;
- запуск і дебаґ – наявність вбудованого відлагоджувача відрізняє Visual Studio Code від конкурентів, що дозволяє середовищу вишукувати помилки одразу в редакторі. Окрім цього, є у редакторі також вбудована консоль, де можуть виводитись результат роботи чи повідомлення про помилку, якщо сталися збої в роботі коду.

Дебаґер можна використовувати під різного роду мови і задачі;

- огляд розширень для Visual Studio Code – у редакторі є наявними розширення для більшості мов програмування, що дозволяє Visual Studio Code бути варіативним у допомозі при розробці програмного продукту.

2.2 Конструювання системи моніторингу

Після ознайомлення із загальними налаштуваннями середовища Visual Studio Code почалося конструювання системи моніторингу – першою стадією якого була побудова програмного кістяку на базі бібліотеки React. Оскільки система буде односторінковою і не потребуватиме великих ресурсних затрат, створити її можна за допомогою команди npx create-react-app *, де:

- npx – це командний засіб в рамках Node.js, який дозволяє встановлювати пакети технології. npx перевіряє, чи існує необхідна розробникові команда відносно пакету даних і виконує її в межах редактора, особливо якщо необхідний пакет даних не був встановлений раніше;
- create-react-app – це застосунок, що налаштовує середовище розробки під взаємодію із бібліотекою ReactJS і забезпечує зручність при використанні. create-react-app створює каркас для розробки front-end, який можна легко використовувати під час роботи з будь-яким back-end.
- * - за бажанням можна дати назву своєму застосунку. За замовчуванням – якщо не вказано назви – дається найменування repos.

Для цього запускається вбудований у редактор термінал (команда Terminal > New Terminal або комбінація клавіш Ctrl+Shift+`), після чого у командному

рядку, що з’явиться після запуску, навпроти шляху, за яким вказано майбутній проект, вводиться команда npx create-react-app (рис.11):

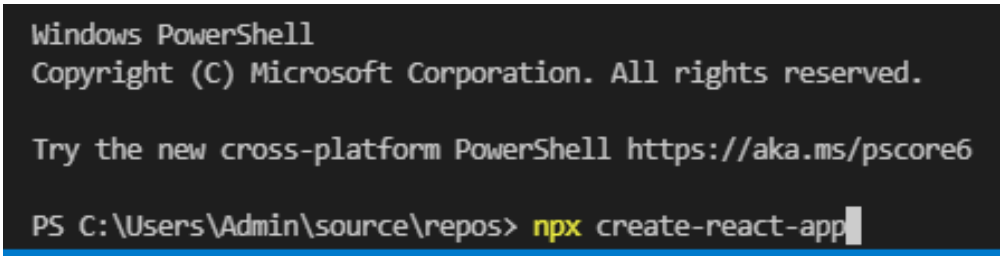

Рисунок 11 – Створення застосунку завдяки команді npx create-react-app

Після завершення процесу формування кістяка програмного забезпечення в теку проекту було додано декілька файлів, що відповідатимуть за функціонування окремих компонентів системи, а саме:

- InfoBox.js
- MapUI.js
- Table.js
- util.js

Загальна файлова структура програми виглядає наступним чином (рис.12):

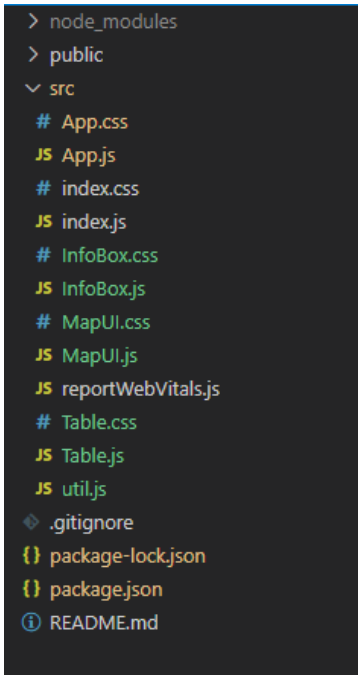

Рисунок 12 – Файлова структура системи

2.2.1. Конструювання App.js

У структурі програми App.js є основним файлом, що відповідає за взаємодію програмного забезпечення з користувачем.

Для того, аби зробити код більш зручним для прочитання і редагування, а також використати на повну всі можливості React, у зверстаному коді програми було використано хуки.

Хуки (Hooks) – це різновид API, впроваджений в лютому 2019 року в межах виходу версії React 16.8. Говорячи простими словами, хуки забезпечують написання функціональних компонентів зі станом та використання інших, більших можливостей React без потреби розписувати класи. Більше того, хуки можна використовувати тільки в функціях, тому потреба в класах відпадає ще більше. Завдяки використанню вказаних API верстка коду здійснюється швидше, що в свою чергу зменшує часові витрати на розробку програмних забезпечень.

Розробники можуть розписувати власні хуки, але бібліотека React станом на написання кваліфікаційної роботи вже містить за замовчуванням ряд вбудованих хуків, серед яких найбільше було необхідно два наступних: useState і useEffect.

useState – один із трьох базових хуків, який повертає значення зі станом і функцію для оновлення даного значення (рис.13):

```
const [state, setState] = useState(initialState);
```

Рисунок 13 – Зразок оголошення useState

Перше значення в масиві, себто state, являється чинним (на кшталт this.state у класах). Друга змінна – setState – це функція, що використовується для оновлення state (this.setState у класах). initialState в даному випадку – це початковий аргумент, котрий з кожним новим викликом протягом дії програми буде повторно рендеритись і від того змінюватись – залежно від дії користувача.

useState у функції може бути оголошеним через будь-який тип значення.

В свою чергу, useEffect призначений для перехоплення різноманітних змін в компонентах, не придатних для внутрішньої обробки (рис.14). Говорячи простіше, useEffect запускає введену всередину частину коду на основі заданої умови. Хук useEffect буде викликатись декілька разів протягом життєвого циклу компонента — один раз після монтування (одразу як компонент буде додано) та кожного разу, коли є необхідність оновлення [19].

```
let [names, setNames] = useState([]);

useEffect(() => {
  fetch("https://uinames.com/api/?amount=25&region=nigeria")
    .then(response => response.json())
    .then(data => {
      setNames(data);
    });
}, []);
```

Рисунок 14 – Зразок оголошення useEffect

На вказаному зразку, де окрім useEffect, оголошено водночас і useState, показано, що хук викликає гіперпосилання і бажає вивільнити звідти необхідні йому дані, тоді повертає response, конвертує базу в JSON, і тоді виконується функція оновлення – в даному випадку setNames – для установки нового значення.

Якщо масив наприкінці useEffect – порожній, то код, що всередині хука, буде запущено лиш одноразово для завантаження компонентів. Якщо ж матимемо в масиві значення – names, наприклад – то код також буде запускатись одноразово і також – кожного разу за умови зміни значення names.

Також під час верстки програмного коду було використано елементи Material UI – це бібліотека в межах React, що дозволяє імпортувати і верстати компоненти для користувацького інтерфейсу. Оскільки в бібліотеці наявна база для розробки, це зберігає вдосталь часу програмістам, адже більше не потрібно розписувати структуру частин з нуля. Великою перевагою Material UI є детальна і зрозуміла документація, завдяки чому розробники можуть легко читати код фреймворку.

У документ було імпортовано дані з бібліотеки React, зокрема вище зазначені хуки useState і useEffect; компоненти Card, CardContent, MenuItem, FormControl і Select з бібліотеки MaterialUI, а також дані із документів InfoBox.js, Table.js, MapUI.js, util.js, App.css і leaflet/dist/leaflet.css – останній відповідає за зручний вигляд інтерактивної карти.

В тілі функції було реалізовано десять хуків: перші вісім повертають значення – списку країн; однієї країни – за замовчуванням змінна встановлена на загальносвітове значення; інформації щодо кожної країни; відображення табличних даних; центрування карти; приближення/віддалення карти; перелік і відображення країн на карті; типів даних по захворюваності. Інші два хуки відповідають за виклик гіперпосилань, що містять зовнішні дані в API, розміщених в мережі Інтернет: перший – за видобування інформації про світову захворюваність, другий – даних по всім країнам і їхнього похідного сортування, розміщення у вигляді таблиці та на інтерактивній карті.

Також в коді було оголошено асинхронну функцію зміни країни при виборі на карті. За замовчуванням, якщо користувач не наводив курсор і не обирав необхідну йому наразі державу, встановлюється значення світової захворюваності і лишається добування даних з відповідного API, в іншому випадку – береться значення певної країни, вивільняється база, опрацьовується і подається користувачеві на екран.

Оголошена функція App повертає налаштований графічний інтерфейс у складі: компонентів Material UI для видавання списку країн при огляді даних захворюваності, інформаційних вікон, інтерактивної карти і табличного блоку.

Повний лістинг коду (рис.15):

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

function App() {

  const [countries, setCountries] = useState([]);
  const [country, setCountry] = useState('worldwide');
  const [countryInfo, setCountryInfo] = useState({});
  const [tableData, setTableData] = useState([]);
  const [mapCenter, setMapCenter] = useState({ lat: 25.0, lng: 11.0 });
  const [mapZoom, setMapZoom] = useState(4);
  const [mapCountries, setMapCountries] = useState([]);
  const [casesType, setCasesType] = useState("cases");

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

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

        let sortedData = sortData(data);
        setTableData(sortedData);
        setMapCountries(data);
        setCountries(countries);
      })
    }
    getCountriesData();
  }, []);

  const onCountryChange = async (e) => {
    const countryCode = e.target.value;
    setCountry(countryCode);

    const url =
      countryCode === "worldwide"
      ? "https://disease.sh/v3/covid-19/all"
      : `https://disease.sh/v3/covid-19/countries/${countryCode}`;

      await fetch(url)
      .then((response) => response.json())
      .then((data) => {
        setCountry(countryCode);
```

```
        setCountryInfo(data);
        setMapCenter([data.countryInfo.lat, data.countryInfo.long]);
        setMapZoom(4);
      });
  }

  return (
    <div className="app">
      <div className="app__left">
      <div className="app__header">
      <h1>COVID-19 Monitoring System</h1>
      <FormControl className="app__dropdown">
        <Select variant="outlined" onChange={onCountryChange} value={country}>
          <MenuItem value="worldwide">Worldwide</MenuItem>
          {countries.map((country) => (
            <MenuItem value={country.value}>{country.name}</MenuItem>
          ))}
        </Select>
      </FormControl>
      </div>

      <div className="app__stats">
        <InfoBox
            title="COVID Cases"
            isRed
            active={casesType === "cases"}
            onClick={(e) => setCasesType("cases")}
            total={prettyPrintStat(countryInfo.cases)}/>
        <InfoBox
            title="Recovered"
            active={casesType === "recovered"}
            onClick={(e) => setCasesType("recovered")}
            total={prettyPrintStat(countryInfo.recovered)}/>
        <InfoBox
            title="Deaths"
            isRed
            active={casesType === "deaths"}
            onClick={(e) => setCasesType("deaths")}
            total={prettyPrintStat(countryInfo.deaths)}/>
      </div>

      <MapUI countries={mapCountries} casesType={casesType} center={mapCenter} zoom={mapZoom}/>

    </div>
    <Card className="app__right">
      <CardContent>
        <h3>Live Cases by Country</h3>
        <Table countries={tableData}/>
      </CardContent>
    </Card>
    </div>
  );
}

export default App;
```

Рисунок 15 – Лістинг коду

2.2.2. Конструювання InfoBox.js

У структурі програми документ InfoBox.js відповідає за опис і вміст інформаційного вікна. У документ було імпортовано дані з бібліотеки React, компоненти Card, CardContent і Typography з бібліотеки MaterialUI, а також дані з документу InfoBox.css.

Оголошена однойменна функція містить кілька параметрів, основними з яких є заголовок – title – і значення захворюваних – total – залежно від того, яку категорію даних бажає обрати користувач, і повертає компонент Card, при натисканні на який будуть видаватися відповідні дані як в інформаційному вікні, так і на карті.

Повний лістинг коду (рис.16):

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

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

export default InfoBox;
```

Рисунок 16 – Лістинг коду

2.2.3. Конструювання MapUI.js

В межах загального програмного коду документ MapUI.js відповідає за опис і вміст інтерактивної карти. У документ було імпортовано дані з бібліотеки React, компоненти TileLayer і MapContainer з бібліотеки React Leaflet, showDataOnMap з файлу util.js, а також дані з документу MapUI.css.

Оголошена однойменна функція містить параметри переліку країн – countries, типу випадків захворюваності – casesType, і два параметри відносно карти: центрування – center – і масштабування – zoom. MapUI повертає блок карти, призначений для користувацького процесу, з вкладеними туди компонентами MapContainer і TileLayer.

Говорячи про React Leaflet, це – бібліотека для створення динамічних інтерактивних карт. Насправді це – та ж бібліотека Leaflet в межах JavaScript, тільки з вкрапленнями від React, завдяки чому конструювання мап стає більш різностороннім і надає більше можливостей розробникам. Для показу карти в рамках системи використовуються компоненти MapContainer – відповідає за контейнер, що містить шар карти – і TileLayer – надає можливість вкласти поверх MapContainer шар земної поверхні завдяки виклику із зовнішнього джерела.

Повний лістинг коду (рис.17):

```
import React from "react";
import { TileLayer, MapContainer } from "react-leaflet";
import "./MapUI.css";
import { showDataOnMap } from "./util";

function MapUI( { countries, casesType, center, zoom } ) {
    return(
        <div className="map">
        <MapContainer center={center} zoom={zoom} maxZoom={18}>
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
        />
        {showDataOnMap(countries, casesType)}
      </MapContainer>
        </div>
    )
}

export default MapUI;
```

Рисунок 17 – Лістинг коду

2.2.4. Конструювання Table.js

У структурі програми документ Table.js відповідає за опис і вміст таблиці для виведення загальних даних по світовій захворюваності. У файл було імпортовано дані з бібліотек React, Numeral, а також з документу Table.css.

Оголошена однойменна функція містить лише одне значення – countries – і відповідає за базу даних по країнах, і повертає таблицю із виведеними на екран даними, приведеними в числовому форматі за допомогою бібліотеки Numeral, завдяки якій можна форматувати і керувати цифровими значеннями.

Повний лістинг коду (рис.18):

```
import React from "react";
import numeral from "numeral";
import "./Table.css";

function Table({ countries }) {
  return (
    <div className="table">
      {countries.map((country, cases) => (
        <tr>
          <td>{country.country}</td>
          <td>
            <strong>{numeral(country.cases).format("0,0")}</strong>
          </td>
        </tr>
      ))}
    </div>
  );
}

export default Table;
```

Рисунок 18 – Лістинг коду

2.2.5. Конструювання util.js

В межах загального програмного коду документ util.js відповідає за додаткові утиліти, необхідні для повноцінного функціонування системи моніторингу. У документ було імпортовано дані з бібліотек React і Numeral, і компоненти Circle і PopUp з бібліотеки React Leaflet.

Додатковими утилітами, які було реалізовано в коді, стали:

- casesTypeColors: відповідає за відображення кольорів типів випадків захворювання;
- sortData: автоматично сортує дані по загальній кількості хворих від зростання до спадання і змінюється за умови перевищення показника однієї країни над іншою;
- prettyPrintStat: декоративна утиліта, що відповідає за показ статистики в інформаційних блоках у вписаному всередину форматі;
- showDataOnMap: відображає дані на інтерактивній карті за допомогою двох компонентів React Leaflet:
    - Circle: показ кількості захворюваних у вигляді кола; чим більший розмір кола на карті, тим більше заражених містить країна;
    - PopUp: відповідає за відображення інформаційного вікна зі статистичними даними по кожній країні. PopUp також містить декоративне оголошення формату чисел заражених;

Повний лістинг коду (рис.19):

```
import React from "react";
import numeral from "numeral";
import { Circle, Popup } from "react-leaflet";

export const casesTypeColors = {
  cases: {
    rgb: "rgb(204, 16, 52)",
    half_op: "rgba(204, 16, 52, 0.5)",
    multiplier: 800,
  },
  recovered: {
    rgb: "rgb(125, 215, 29)",
    half_op: "rgba(125, 215, 29, 0.5)",
    multiplier: 1200,
  },
  deaths: {
    rgb: "rgb(251, 68, 67)",
    half_op: "rgba(251, 68, 67, 0.5)",
    multiplier: 2000,
  },
};

export const sortData = (data) => {
  let sortedData = [...data];
  sortedData.sort((a, b) => {
    if (a.cases > b.cases) {
      return -1;
    } else {
      return 1;
    }
  });
  return sortedData;
};

export const prettyPrintStat = (stat) =>
  stat ? `+${numeral(stat).format("0.0a")}` : "+0";

export const showDataOnMap = (data, casesType = "total") =>
  data.map((country) => (
    <Circle
      center={[country.countryInfo.lat, country.countryInfo.long]}
      color={casesTypeColors[casesType].rgb}
      fillColor={casesTypeColors[casesType].rgb}
      fillOpacity={0.4}
      radius={
        Math.sqrt(country[casesType]) * ((casesTypeColors[casesType].multiplier)/5)
      }
    >
      <Popup>
        <div className="info-container">
          <div
            className="info-flag"
            style={{ backgroundImage: `url(${country.countryInfo.flag})` }}
          ></div>
          <div className="info-name">{country.country}</div>
          <div className="info-confirmed">
            Cases: {numeral(country.cases).format("0,0")}
          </div>
          <div className="info-recovered">
            Recovered: {numeral(country.recovered).format("0,0")}
          </div>
          <div className="info-deaths">
            Deaths: {numeral(country.deaths).format("0,0")}
          </div>
        </div>
      </Popup>
    </Circle>
  ));
```

Рисунок 19 – Лістинг коду

2.3 Розгортання системи моніторингу

Після завершення конструювання кодової частини проєкту для його розгортання і перегляду потрібно було його показати на сервері. Для цього в редакторі Visual Studio Code, що дозволяє відображати веб-застосунки завдяки Node.js, запускаємо термінал і прописуємо скрипт npm start (рис.20):

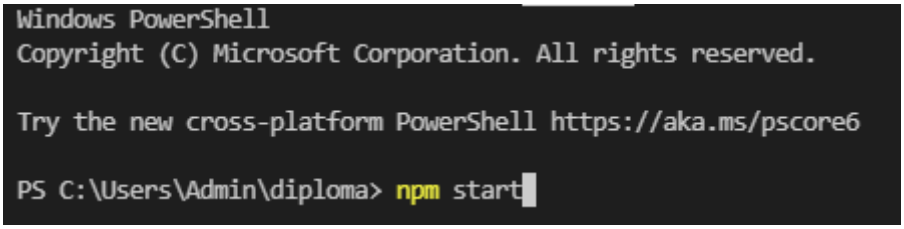

Рисунок 20 – Запуск розгортки системи завдяки скрипт npm start

npm start використовується для того, аби виконати розгортання системи на локальному чи іншому сервері.

Після прописання скрипта очікуємо на відповідь сервера, після отримання якої буде відображено повідомлення про успішне завершення компіляції в редакторі (рис.21) і результат роботи – веб-систему моніторингу – на сервері (рис.22):

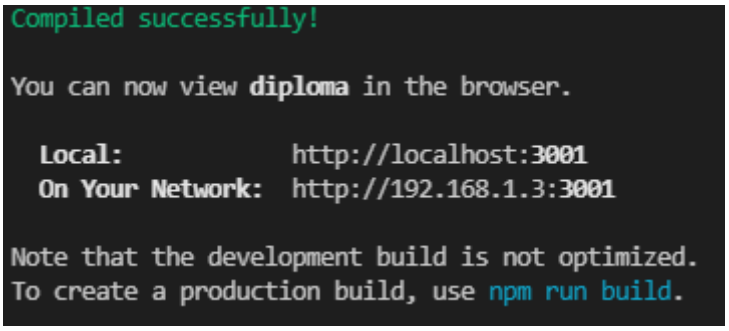

Рисунок 21 – Повідомлення про завершення компіляції

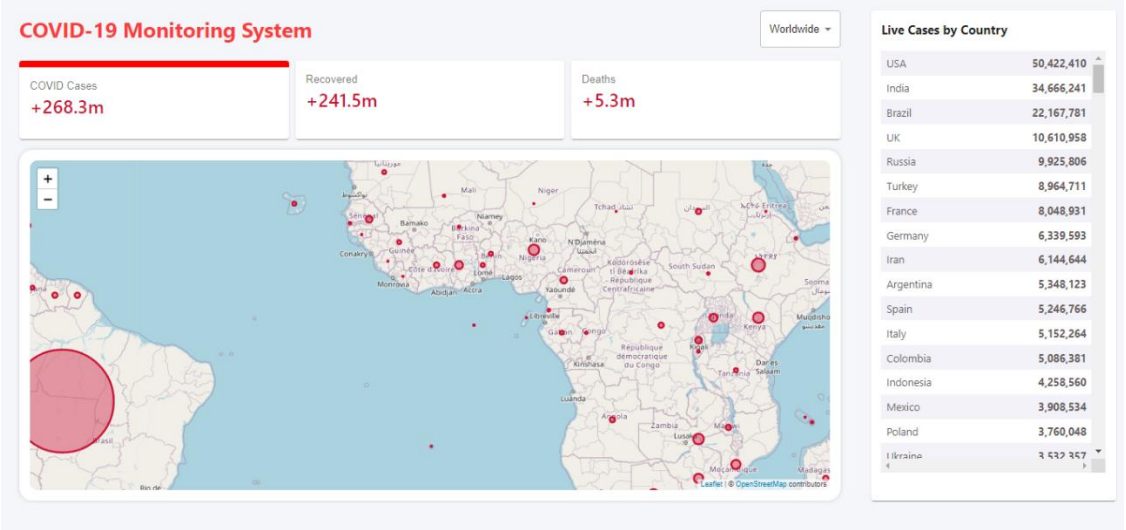

Рисунок 22 – Основна сторінка системи

На основній сторінці відображено:

- три інформаційні вікна із загальною світовою статистикою – «COVID Cases», тобто випадки зараження коронавірусом, «Recovered», себто кількість тих, хто одужав і «Deaths» - кількість тих, хто зазнав летальних наслідків від симптомів;
- таблицю з даними по кількості захворюваних в кожній країні світу, відсортовану від найбільш зараженої до найменш;
- інтерактивну карту, яка дозволяє у візуальній формі оцінити становище держав і за потреби вибрати одну з них, натиснувши на коло, що відповідає тій чи іншій країні;

Може бути випадок, що користувач хоче подивитись на кількість заражених в інформаційному вікні. Для цього обираємо меню-список вгорі, лівіше від таблиці, і замість значення «Worldwide» - світова зараженість – оберемо довільну державу. Наприклад, Україну – «Ukraine» (рис.23):

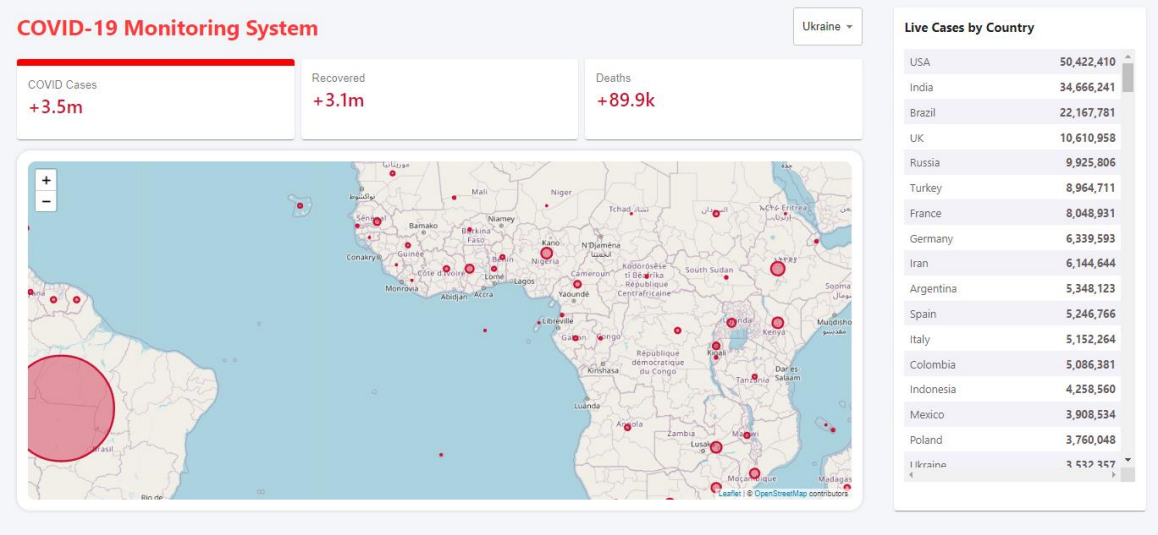

Рисунок 23 – Відображення даних по Україні в інформаційних вікнах

Якщо ж користувач бажає подивитись дані по тій же Україні на інтерактивній карті, він може прогортати карту, знайти необхідну йому державу і натиснути на відповідне до неї графічне коло (рис.24):

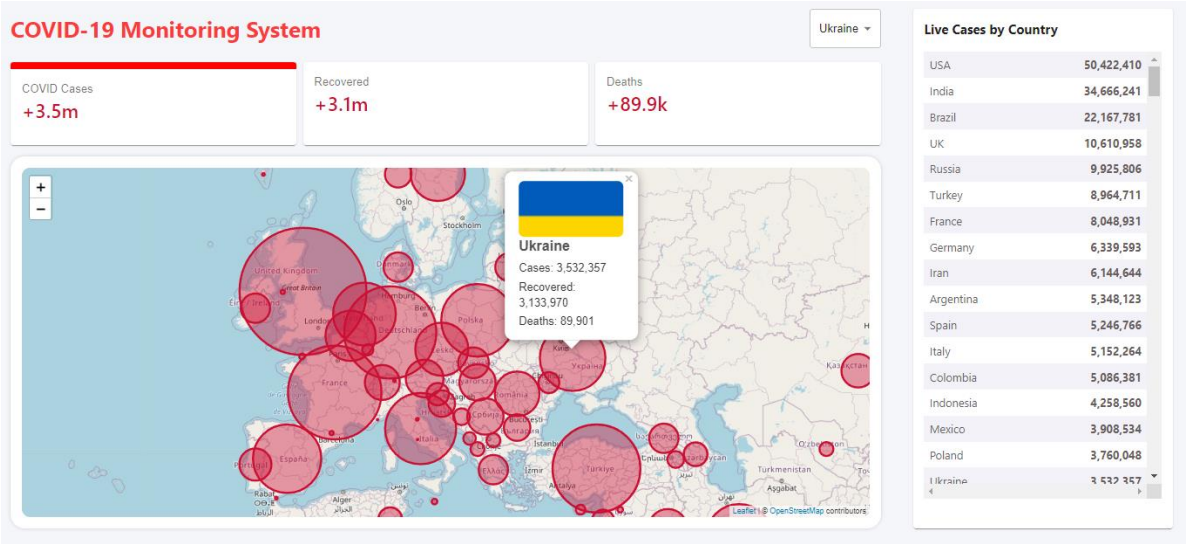

Рисунок 24 – Відображення даних по Україні на інтерактивній карті

Інформаційні вікна мають також кнопковий функціонал – коли користувач натискає на одне із вікон, на карті відображаються графічні моделі кількості захворювань по країнах – залежно від того, які із даних переважають по кількості, випадуть більші чи менші. Наприклад, користувач замість опції «COVID Cases» вибрав опцію «Deaths» - і на екрані буде виведено наступний результат (рис.25):

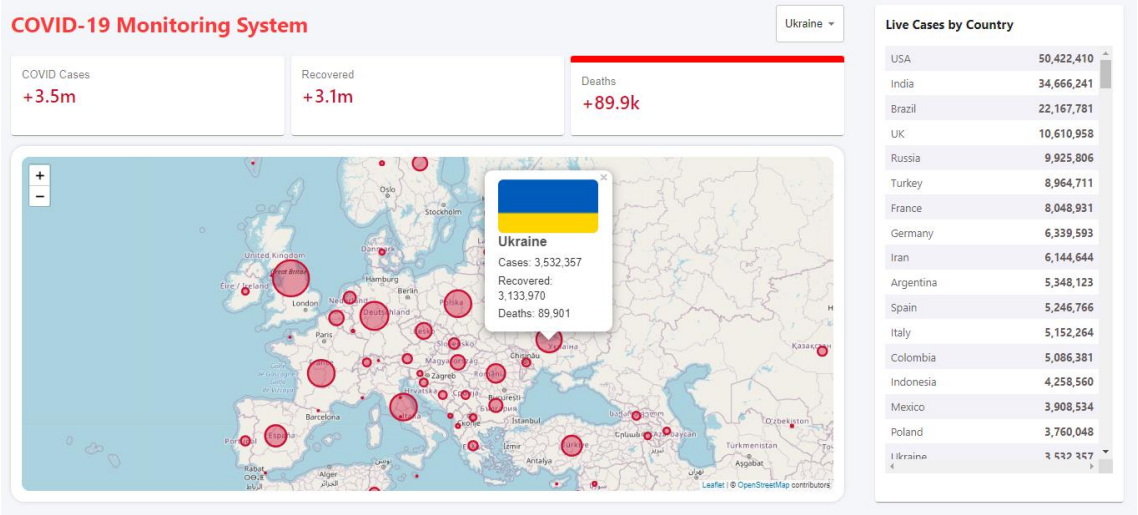

Рисунок 25 – Відображення даних по смертності на інтерактивній карті

Надалі планується розробка і реалізація компоненту лінійних графіків і додання відображення даних по регіонах України.

Тестування системи відбувалось протягом процесу конструювання і переважно ставались механічні помилки, які миттєво лагодились. Остаточна версія програмного забезпечення працює безпомилково і жодних багів не було виявлено.

3 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Охорона праці

Розробка системи моніторингу захворюваності на COVID-19 здійснювалась з використанням вектора метрики якості з використанням портативних пристроїв та персональних комп'ютерів. Відповідно, в ході виконання експлуатації необхідно працювати з комп'ютерною технікою. Тому надзвичайно важливим фактором безпеки праці є дотримання правил користування технікою, норм та правил охорони праці.

Питання охорони праці регулюються певними законодавчими та нормативно-правовими актами, які, зокрема, визначають обов'язки роботодавця із забезпечення робітникам комфортних та безпечних умов для здійснення роботи. Ці обов'язки, а також права робітників на таких умовах праці передбачені частиною 2 ст.2 і частиною 1 ст.21 КЗпП, а також ст.13 Закону України «Про охорону праці»[20], у яких визначаються основні положення з реалізації конституційного права робітників. Існує цілий ряд вимог, які визначають специфіку заходів з охорони праці при роботі з персональним комп'ютером. Законодавчі та нормативно-правові акти, які за участі відповідних органів державної влади регулюють відносини між роботодавцем та робітником з питань безпеки, гігієни праці та виробничого середовища, а також встановлюють єдиний порядок організації охорони праці в Україні. На їх основі розроблені чисельні документи: правила, інструкції, норми, державні санітарні правила та інші нормативно-правові документи, якими мають керуватись роботодавці та які регламентують певні питання щодо конструкції електронно-обчислювальної техніки, та особливостей їх розміщення.

Наразі основними документами, що регламентують питання охорони праці при використанні працівниками персональних комп'ютерів, можна вважати наступні підзаконні акти:

- НПАОП 0.00-7.15-18 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [21];
- ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [22].

У відповідності з цими актами, при розробці будь-якого програмного забезпечення, в тому числі при розробці системи моніторингу захворюваності на COVID-19, необхідно вжити всіх необхідних заходів з охорони праці та розробити відповідні документи.

Окрім цього, в залежності від кількості працівників в ІТ компанії та від кількості працівників задіяних в проекті по розробці системи моніторингу захворюваності на COVID-19 служба охорони праці виглядає наступним чином:

- В ІТ компанії або проекті з кількістю працюючих 50 і більше осіб, роботодавець створює службу охорони праці відповідно до типового положення, що затверджується центральним органом виконавчої влади, що забезпечує формування державної політики у сфері охорони праці.
- В ІТ компанії або проекті з кількістю працюючих менше 50 осіб, функції служби охорони праці можуть виконувати в порядку сумісництва особи, які мають відповідну підготовку.
- В ІТ компанії або проекті з кількістю працюючих менше 20 осіб, для виконання функцій служби охорони праці можуть залучатися сторонні спеціалісти на договірних засадах, які мають відповідну підготовку.

Підпорядковується служба охорони праці згідно із законодавством безпосередньо роботодавцеві [20]. Проте, роботодавець може доручити функціональне управління (кураторство) діяльністю служби іншій посадовій особі, скажімо, головному інженерові, заступникові директора з охорони праці тощо. Покладення таких обов’язків потрібно закріпити наказом або відобразити в посадовій інструкції уповноваженої особи. Робота служби охорони

праці підприємства має здійснюватися відповідно до плану роботи та графіків обстежень, затверджених роботодавцем. Функції служби охорони праці:

- Підготовка проектів наказів (розпоряджень) з питань охорони праці і внесення їх на розгляд роботодавцю. Проведення перевірок дотримання працівниками вимог нормативно- правових актів з охорони праці.
- Проведення з працівниками вступного інструктажу з питань охорони праці та супутніх інструктажів.
- Ведення обліку та проведення аналізу причин виробничого травматизму, професійних захворювань, аварій на виробництві, заподіяної ними шкоди.
- Забезпечення належного оформлення і зберігання документації з питань охорони праці.
- Складання звітності з охорони праці за встановленими формами.
- Складання за участю керівників підрозділів підприємства переліків професій, посад і видів робіт, на які повинні бути розроблені інструкції з охорони праці, що діють в межах підприємства, надання методичної допомоги під час їх розроблення.
- Інформування працівників про основні вимоги законів, інших нормативно-правових актів та актів з охорони праці, що діють в межах підприємства.
- Розгляд питань про підтвердження наявності небезпечної виробничої ситуації, що стала причиною відмови працівника від виконання дорученої роботи відповідно до законодавства (у разі необхідності).
- Організація - забезпечення підрозділів нормативно-правовими актами з охорони праці та актами з охорони праці, що діють в межах підприємства, посібниками, навчальними матеріалами з цих питань [20].

Дотримання даних пунктів є необхідним при роботі з системою моніторингу захворюваності на COVID-19.

3.2 Підвищення стійкості роботи лікарень у воєнний час

Стійкість роботи об’єкта господарської діяльності – це здатність його в умовах надзвичайних ситуацій (далі – НС) випускати продукцію у запланованому обсязі та визначеної номенклатури, а у разі слабких та середніх руйнувань або порушення матеріального постачання – відновлювати виробництво власними силами у короткий термін.

На стійкість роботи об’єкта впливають такі фактори:

- захищеність робітників та службовців від вражальних факторів у НС;
- здатність інженерно-технічного комплексу об’єкта (будівель, споруд, обладнання та комунально-енергетичних мереж) протистояти руйнівній дії вражальних факторів аварій, катастроф, стихійного лиха та сучасної зброї;
- надійність постачання об’єкта електроенергією, водою, паливом, комплектуючими та сировиною;
- підготовленість об’єкта до проведення аварійно-рятувальних та відновлюваних робіт;
- оперативність здійсненням заходів закладу під час НС.

Стійкість функціонування системи охорони здоров'я міста, району, регіону і держави в цілому в умовах НС забезпечується шляхом завчасного вирішення конкретних завдань для кожного окремого об'єкту охорони здоров'я. Заходи щодо підвищення стійкості роботи об'єктів охорони здоров'я не можуть бути проведені під час виникнення НС. Вони повинні мати чіткий поділ на заходи першої і другої черги з врахуванням специфіки кожного об'єкту охорони здоров'я.

Для всіх об'єктів охорони здоров'я можна сформулювати загальні і спеціалізовані питання, за якими оцінюється стійкість їх функціонування в екстремальних умовах НС мирного і воєнного часу. До них належать:

- аналіз вихідних даних об'єкту охорони здоров'я, які обумовлюють стан стійкості його роботи;

- прогнозування можливого впливу на об'єкти вражаючих чинників НС мирного і воєнного часу;
- оцінка готовності об'єкту до роботи в екстремальних умовах мирного і воєнного часу з врахуванням особливостей регіону, міста і прогнозованої обстановки при виникненні НС в мирний і воєнний час;
- визначення переліку заходів, що підвищують стійкість функціонування об'єкту і термінів їх проведення.

Також до переліку питань підвищення стійкості функціонування лікарень входить ряд і інших технічних вимог:

- передбачається надійність системи енергопостачання і електроосвітлення лікарень, варіанти аварійного освітлення за допомогою рухомих електростанцій для освітлення операційних (родових), перев'язувальних, реанімаційних, палат інтенсивної терапії, стерилізаційних, а також освітлення приймальних відділень, коридорів;
- передбачається аварійне теплопостачання шляхом створення запасів газу в балонах і інших видів палива (для котелень або печей) на період відновлення основного джерела теплопостачання;
- передбачається аварійне водопостачання шляхом створення запасів питної води з розрахунку 2 л / добу на одного хворого і технічної води з розрахунку по 10 л / добу на койку з аварійних ємностей, встановлених у верхній частині будівлі з можливістю подачі води за допомогою трубопроводів (гнучких шлангів) від зовнішніх мереж;
- каналізаційна система в лікарні повинна забезпечувати проведення дезактивації, обладнується спеціальними відстійниками в системі очисних споруд;
- система внутрішньолікарняної безпеки від вражаючих факторів (пожежо- і вибухонебезпечні речовини, наприклад) забезпечується раціональним розподілом потоку хворих і обслуговуючого персоналу, а також раціональним розміщенням і обладнанням відповідних приміщень лікарні;

- для захисту хворих в стаціонарних установах передбачається будівництво захисних споруд (сховищ або протирадіаційних укриттів);
- засоби зв'язку в лікарні повинні забезпечувати постійну можливість швидкої подачі сигналу тривоги в усі приміщення, де перебувають хворі і персонал через радіомережа або іншу систему гучного зв'язку, а черговий персонал забезпечується портативними переносними засобами зв'язку для роботи всередині будівлі;
- у великих лікарнях повинна бути автоматизована система реєстрації уражених і банк даних про історії хвороби для їх швидкої статистичної обробки;
- передбачається система екстреної евакуації хворих, доповнена індивідуальними рятувальними пристроями при порушеннях евакуації звичайним порядком: через вікна на першому поверсі, а починаючи з другого поверху і вище – з використанням трапів, запасних сходів, спеціальних мереж та ін.
- передбачається створення резерву медичного майна на випадок НС: лікарських засобів, антидотів, виробів медичного призначення, дезінфекційних засобів та інших витратних матеріалів, а для їх зберігання – спеціальні складські приміщення, в тому числі приміщення з холодильними камерами для зберігання препаратів вимагають дотримання температурного режиму.

Для того, щоб забезпечити захист лікарень та надати їм можливість працювати в умовах НС необхідно провести планування та підготовку до роботи в цих умовах. Це дозволить лікарні не припиняти свою роботу і надавати медичну допомогу постраждалим внаслідок НС, а також мати у самій лікарні план дій на випадок виникнення НС і підготувати медичний персонал для продовження роботи в екстремальних умовах для задоволення зростаючих вимог і потреб, що виникають в умовах ліквідації наслідків НС. З точки зору витрат на підготовку лікарень до роботи в екстремальних умовах НС є будівництво самих лікарень за виваженими проектами або модифікацію вже побудованих.

Також слід зазначити, що найбільшу цінність лікарні мають неструктурні елементи, такі як обладнання життєзабезпечення будівлі (енергоустановки, водопостачання, системи каналізації, вентиляції, холодильні установки), та саме пошкодження цих систем призводить до припинення роботи лікарні в умовах НС.

Медична допомога за умов НС — це особливий вид медичної допомоги, коли за умов дефіциту часу й засобів, а також високої психічної напруги лікар та його помічники повинні постійно проявляти не тільки високий професіоналізм, але й співчуття і вміння працювати у несприятливих умовах. Однією з найважливіших умов ефективної роботи в НС є забезпечення повної автономності мобільних формувань в усіх напрямах (енерго- та водопостачання, зв'язок), не кажучи вже про все необхідне медичне майно. Пріоритет медика повинен бути обов'язковим для всіх учасників ліквідації наслідків НС будь-якого характеру.

Слід також наголосити на значущості наявності необхідного медичного і спеціального майна. Відсутність у необхідній кількості медикаментів, перев'язувальних засобів, тобто найнеобхіднішого, особливо в перші години і дні після катастрофи, а також проблеми з транспортом, що заважають евакуації постраждалих, нечітка організація першої медичної допомоги безпосередньо в зоні НС і низька оснащеність медичних формувань, розгубленість і паніка в роботі органів управління, психологічна пригніченість місцевих мешканців — все це створює великі труднощі при наданні медичної допомоги.

Не менш вагомими у підвищенні ефективності ліквідації медико-санітарних наслідків НС є організаційні аспекти, які мають бути комплексними і повинні реалізовуватись не тільки за умов НС, але й у період підготовки до НС, у режимі підвищеної готовності та ліквідації її наслідків.

З перших хвилин після виникнення НС робота органів охорони здоров'я повинна бути спрямована на надання екстреної медичної допомоги (ЕМД): мобілізований персонал; до місць руйнувань направляються мобільні формування; налагоджується постачання медикаментів і медичного майна, транспортування постраждалих; забезпечується порядок на дорогах та безперебійна робота транспорту.

При підготовці до роботи в умовах виникнення НС керівництво лікувального закладу (ЛЗ) вирішує два основні завдання.

Перше завдання: якщо ЛЗ зазнає впливу вражаючих факторів НС, необхідно, перш за все, забезпечити захист хворих, персоналу, обладнання та інших матеріальних засобів. Далі, залежно від обставин, приступити до надання кваліфікованої та спеціалізованої ЕМД постраждалому населенню, у тому числі персоналу ЛЗ, а також хворим, які можуть зазнавати впливу вражаючих факторів НС. Явним є те, що ЛЗ може приступити до роботи, опинившись у зоні НС, тільки за певних умов.

Друге завдання: якщо ЛЗ не зазнає впливу вражаючих факторів НС, то, відповідно до плану, перепрофілює ліжкофонд окремих лікувальних відділень, створює медичний персонал і забезпечує ним безперебійну діяльність приймально-сортувально-діагностичного відділення ЛЗ, надає ураженому населенню кваліфіковану та спеціалізовану ЕМД, приводить у готовність створені на його базі медичні формування ДСМК — спеціалізовані бригади постійної готовності другої черги. Медичні формування (медичні бригади та медичні загони), створені у ЛЗ, використовуються відповідно до сформованої обстановки й отриманих керівних розпоряджень територіального органу управління охорони здоров'я.

Дотримання усіх вимог і оперативне виконання поставлених завдань з урахуванням особливостей установи багато в чому підвищить стійкість функціонування лікарень під час НС, зокрема – війни.

ВИСНОВКИ

У дипломній роботі представлено дослідження системи моніторингу захворюваності на COVID-19, засноване на базі еволюційної моделі життєвого циклу системи і з прийняттям поведінкових архітектурних рішень. З цією метою була створена програмна система. Вона базується на представленні статичних та інтерактивних даних відносно стану держав з наявними захворюваними на симптоми коронавірусу. Реалізована на основі еволюційної моделі життєвого циклу система показує достовірні дані, надані із зовнішніх джерел, які щоденно оновлюються. Програмний комплекс складається з декількох модулів, які взаємодіють одним з одним за допомогою обраного для розробки поведінкового паттерну, але також може доповнюватись за потреби, і завдяки такому «структуруванню» програмне забезпечення може покращуватись і збільшувати свою функціональність за необхідності кращого висвітлення даних.

Всі розділи і методи проілюстровані результатами розробки, щоб показати, як розроблені засоби застосовуються для інформування населення. Розробка системи починається з структури та побудови кістяку, на якому буде базуватись програмне забезпечення. Потім ми описуємо варіанти надання даних користувачам і реалізовуємо процес роботи.

Робота також містить презентацію графічного інтерфейсу програми з докладним описом результатів розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Системи моніторингу та керування | IT-Solutions, Україна. [Електронний ресурс] URL: https://it-solutions.ua/uk/servisi/sistemi-monitoringu-ta-keruvannya
2. Розробка ПЗ: моделі життєвого циклу, методи та принципи. [Електронний ресурс] URL: https://evergreens.com.ua/ua/articles/software-development-metodologies.html
3. Життєвий цикл програмного забезпечення. Матеріал з Вікіпедії — вільної енциклопедії. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Життєвий_цикл_програмного_забезпечення
4. Моделі життєвого циклу програмного забезпечення ІС. [Електронний ресурс] URL: https://pidru4niki.com/10050711/informatika/modeli_zhittyevogo_tsiklu_programnogo_zabezpechennya
5. Моделювання програмного забезпечення : науково-методичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.
6. Stack Overflow Developer Survey 2021. [Електронний ресурс] URL: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-integrated-development-environment
7. MSDN – Основи HTML. [Електронний ресурс] URL: https://msdn.microsoft.com/ru-ru/library/wx4hctt4.aspx
8. Електронний HTML і CSS довідник українською мовою. [Електронний ресурс] URL: https://html-css.co.ua/
9. Вивчаємо веб-дизайн дистанційно – Лекція 3. Основні поняття мови HTML та структура документів. [Електронний ресурс] URL: https://sites.google.com/site/vivcaemowebdizajndistancijno/html/lekcia-3-osnovni-ponatta-movi-html-ta-struktura-dokumentiv

10. Елементи HTML. Матеріал з Вікіпедії — вільної енциклопедії. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B8_HTML
11. MSDN – Working with CSS Overview. [Електронний ресурс] URL: https://msdn.microsoft.com/en-us/library/bb398931(v=vs.110).
12. Що таке CSS: пояснюємо простими словами. [Електронний ресурс] URL: https://gb.ru/posts/chto-takoe-css-obyasnyaem-prostymi-slovami
13. MSDN – Справочник по мові Java Script. [Електронний ресурс] URL: https://msdn.microsoft.com/ru-ru/library/d1et7k7c(v=vs.94).aspx
14. JavaScript – JavaScript-довідка. [Електронний ресурс] URL: http://xn--80adth0aefm3i.xn--j1amh/javascript
15. Про проєкт | Node.js. [Електронний ресурс] URL: https://nodejs.org/uk/about/
16. Node.js. Матеріал з Вікіпедії — вільної енциклопедії. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Node.js
17. Getting Started – React. [Електронний ресурс] URL: https://reactjs.org/docs/getting-started.html
18. What is React: Definition, Why ReactJS, its Features and Installation. [Електронний ресурс] URL: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs
19. React Hooks — огляд можливостей нового API. [Електронний ресурс] URL: https://dou.ua/lenta/articles/react-hooks-guide/
20. Закон України «Про охорону праці». [Електронний ресурс] URL: https://zakon.rada.gov.ua/laws/show/2694-12
21. Закон України «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». [Електронний ресурс] URL: https://zakon.rada.gov.ua/laws/show/z0508-18

22. Закон України «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». [Електронний ресурс] Режим доступу – https://zakon.rada.gov.ua/rada/show/v0007282-98
23. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с.

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку кваліфікаційної роботи

«Розробка та дослідження системи моніторингу захворювань на COVID-19 на основі еволюційних моделей та поведінкових архітектурних рішень»

Розробники:

виконавець ст. гр. СПм-61

Ліщук Юрій Ігорович

_____

(підпис)

керівник кваліфікаційної роботи

Стоянов Юрій Миколайович

_____

(підпис)

Тернопіль 2021

ЗВІТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки бакалаврів за спеціальністю 121 «Інженерія програмного забезпечення».

Тема кваліфікаційної роботи: «Розробка та дослідження системи моніторингу захворювань на COVID-19 на основі еволюційних моделей та поведінкових архітектурних рішень».

Термін виконання: до «___»_____2021р.

2 ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Інформаційна система призначена для моніторингу стану захворюваності на COVID-19.

Інформаційна система буде корисною в сфері медицини та держуправління.

Інформаційна система дозволить моделювати стан захворюваності в державах.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система повинна передбачати одну роль:

користувач

Для користувачів система надає доступ до повного переліку функцій.

Кінцевий програмний продукт повинен надавати дані про захворюваність для оповіщення населення про світовий стан поширення вірусу та ефективно працювати на особистих персональних комп’ютерах. Власне, питання вирішення цих проблем і вирішується безпосередньо у дипломній роботі.

Набір даних функцій дозволяє користувачу використовувати програму для оцінки становища із поширенням COVID-19.

3.2 Технічні вимоги

Вимоги до клієнтської частини: OS Windows, інтуїтивно зрозумілий, не перевантажений інтерфейс.

3.3 Програмні вимоги

Розробка серверної частини: Node.js

Розробка клієнтської частини: HTML/CSS/JavaScript

Додаткові вимоги: зручний інтерфейс – бібліотека React на базі мови JavaScript

4. ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться в наступному порядку:

аналіз предметної області, аналіз конкурентів та основних алгоритмів роботи програмної системи;

вибір засобів розробки;

розробка архітектури та складових системи;

оформлення супровідної документації;

здача проекту.

Результати виконання кожного етапу проекту погоджуються з керівником проекту.

5. СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

завдання

пояснювальна записка до кваліфікаційної роботи;

презентація проекту;

рецензія на проект;

диск з проектом.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК ЗДАЧІ ПРОЕКТУ

Розроблена інформаційна системи повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	
Архітектура системи	
Проектування системи	
Використання системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

Публікація в науковому виданні

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

IX НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

ДОДАТОК В