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 з підтримки комунікації для пошуку втрачених речей у громадських закладах»

Виконав _____
(Підпис)

Глинка Олег Ярославович
(прізвище, ім'я, по батькові)

Керівник Броварець Олександр Олександрович
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали)

Унікальність тексту : 82%

Київ – 2024

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	01.11.2023	виконав
2.	Видача завдання кваліфікаційної роботи бакалавра	15.11.2023	виконав
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	27.11.2023	виконав
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.12.2023	виконав
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.12.2023	виконав
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	31.12.2023	виконав
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2024	виконав
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	29.03.2024	виконав
9.	Подання роботи у першому варіанті	29.04.2024	виконав
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	02.05.2024	виконав
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	15.05.2024	виконав
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	20.05.2024	виконав
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	06.06.2024	виконав
14.	Захист кваліфікаційної роботи бакалавра	17.06.2024	Виконано

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою – англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
Розділ 1	15
Розділ 2	10
Розділ 3	14
Висновки	2
Перелік використаних джерел	5
Додатки	15

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Глинка О. Я.			Відомість дипломної роботи	Лист	Листів
Керівн.	Броварець О. О					
Н/контр.						
Зав. каф.	Плескач В. Л					

АНОТАЦІЯ

Дипломна робота: 70 с., 27 рис., 8 табл., 39 джерел, 7 дод.

Метою кваліфікаційної роботи бакалавра є підвищення ефективності комунікації між людьми, які загубили речі, і тими, хто їх знаходить, таким способом, збільшуючи шанси на повернення загублених речей їхнім законним власникам.

Для виконання поставленої мети необхідно вирішити наступні завдання:

- провести огляд існуючих цифрових платформ, що використовуються для повідомлення про загублені речі та їх пошуку, з метою виявлення найкращих практик та сфер, що потребують вдосконалення;
- проаналізувати сучасні технології та методології розробки веб-застосунків підтримки комунікації, що застосовують геокодування;
- розробити зручний веб-застосунок підтримки комунікації, пристосований до особливих потреб тих, хто втратив чи знайшов речі у громадських закладах.

Об'єктом дослідження кваліфікаційної роботи бакалавра є процеси підтримки комунікації між людьми для пошуку та повернення втрачених речей у громадських закладах.

Предметом дослідження роботи є програмно-технічні засади для побудови веб-застосунку для підтримки комунікації.

Практичне значення полягає у тому, що розроблений веб-застосунок може бути використаний для надання зручного засобу комунікації людям, метою яких є безпечний пошук та швидке повернення втрачених речей у громадських місцях.

Ключові слова: комунікація, веб-застосунок, serverless, втрачені речі, громадські місця.

ABSTRACT

The *purpose* of the bachelor's thesis is to increase the effectiveness of communication between people who have lost things and those who find them, thus increasing the chances of returning lost items to their rightful owners

To achieve this goal, the following tasks need to be solved:

- conduct a review of existing digital platforms used to report and locate lost and found items to identify best practices and areas for improvement;
- analyze modern technologies and methodologies for developing web-based communication support applications that use geocoding;
- develop a user-friendly web-based communication support tool adapted to the specific needs of those who have lost or found things in public places.

The *object* of study of the bachelor's thesis is the processes of supporting communication between people to find and return lost items in public institutions.

The *subject* of the research is the software and hardware foundations for building a web application that increases the efficiency of finding lost items in public places.

The *practical significance* lies in the fact that the developed web application can be used to provide a convenient means of communication for people whose goal is to safely search for and quickly return lost items in public places.

Keywords: communication, web application, serverless, lost items, public places.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AJAX (англ. Asynchronous JavaScript and XML) – набір методів веб-розробки, який використовує різні веб-технології на стороні клієнта для створення асинхронних веб-застосунків.

WWW (англ. World Wide Web) – інформаційна система, яка дозволяє обмінюватися контентом через Інтернет у зручний для користувача спосіб, розрахований не лише на IT-спеціалістів та аматорів, а й на широке коло користувачів.

API (англ. Application Programming Interface) – це механізми, які дозволяють двом програмним компонентам взаємодіяти один з одним за допомогою набору визначень і протоколів.

JWT (англ. JSON Web Token) – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON.

AWS (англ. Amazon Web Services) – провайдер, що надає хмарні обчислювальні платформи та API на вимогу приватним особам, компаніям та урядам на основі оплати за фактом використання.

БД – база даних.

S3 (Simple Storage Service) – це масштабований хмарний сервіс для зберігання об'єктів.

SQS (Simple Queue Service) - масштабований сервіс черги повідомлень для декомпозиції та забезпечення координації розподілених програмних систем і компонентів.

ВСТУП

У сучасному сповненому стресу світі, втрата особистих речей у громадських місцях стала поширеною незручністю, яка може порушити повсякденну діяльність і спричинити додатковий стрес. Смартфони і гаманці, ключі і особисті документи - список речей, які можна загубити, дуже довгий. Проте, поширеність та доступність Інтернету і електронних девайсів дають можливість людям економити час та зусилля для вирішення як повсякденних справ, так і для непередбачуваних. Таким чином, потреба в розробці веб-застосунку для підтримки комунікації та полегшення пошуку загублених речей, ґрунтується на постійно зростаючій потребі в більш простому і швидкому вирішенні цієї все більш поширеної проблеми.

Актуальність роботи полягає в тому, що представлення платформи, яка надасть спосіб комунікації в реальному часі тим, хто втратив особисті речі в громадських місцях, посприяє загальному покращенню самопочуття та впевненості користувачів.

Метою кваліфікаційної роботи є підвищення ефективності комунікації між людьми, які загубили речі, і тими, хто їх знаходить, таким способом, збільшуючи шанси на повернення загублених речей їхнім законним власникам.

Завдання дослідження:

- провести огляд існуючих цифрових платформ, що використовуються для повідомлення про загублені речі та їх пошуку, з метою виявлення найкращих практик та сфер, що потребують вдосконалення;
- проаналізувати сучасні технології та методології розробки веб-застосунків підтримки комунікації, що застосовують геокодування;
- розробити зручний веб-застосунок підтримки комунікації, пристосований до особливих потреб тих, хто втратив чи знайшов речі у громадських закладах.

Об'єктом дослідження кваліфікаційної роботи бакалавра виступають процеси підтримки комунікації між людьми для пошуку та повернення втрачених речей у громадських закладах.

Предметом дослідження є програмно-технічні засади для побудови веб-застосунку для підтримки комунікації.

Структура даної кваліфікаційної роботи бакалавра складається із вступу, трьох розділів, висновку, списку джерел і додатків.

РОЗДІЛ 1
ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО РОЗРОБЛЕННЯ І ВПРОВАДЖЕННЯ ВЕБ-ЗАСТОСУНКІВ

1.1 Роль веб-застосунків у світі

Для людини XXI століття веб-застосунки стали невід’ємною частиною життя, так як вони слугують основою для сучасного зв'язку та цифрової взаємодії. Це, в певному сенсі, комп’ютерні програми, доступ до яких і взаємодія з якими здійснюється через веб-браузер з підключенням до мережі Інтернет. Іншими словами, вони не запускаються операційною системою користувача, як традиційні десктопні програми, а доступні через спеціальну комп’ютерну програму - веб-браузер, що свідчить про їхню залежність як від браузера, так і веб-сервера для управління запитами користувачів [35]. Це означає, що більша частина навантаження прилягає на сторону сервера, відповідального за обробку даних та їх повернення.

Історія веб-додатків бере свій початок з перших днів існування Всесвітньої павутини, яка спочатку повністю складалася зі статичних сторінок (рис. 1.1) [38].

Рисунок 1.1 – Перша у світі веб-сторінка

Впровадження CGI (англ. Common Gateway Interface, загальний інтерфейс шлюзу) та інших технологій на початку 1990-х років почало змінювати цю ситуацію.

Пізніші розробки JavaScript і AJAX дозволили зробити сторінки динамічними і швидко реагуючими, фактично перетворивши їх на інтерактивні веб-додатки, якими ми звикли користуватися сьогодні [37]. Вищезгадані технології дозволили з'явитися онлайн-формам, системам управління контентом і платформам електронної комерції, і незабаром веб-додатки змінили те, як ми робимо покупки, здійснюємо банківські операції або навчаємося. Завдяки цьому веб-додатки вирішили незліченну кількість практичних проблем і спростили операції, що вимагали великих ресурсів для виконання. Вони усунули географічні бар'єри для здійснення покупок, навчання та спілкування в режимі реального часу з людьми по всьому світу. Мобільні технології та адаптивний дизайн також зробили веб-додатки доступними і придатними для використання на різних пристроях, що покращило доступ до них і користувацький досвід загалом. Один з найяскравіших прикладів - це сфера електронної комерції. Без веб-додатків покупки вимагали поїздок до магазинів, що займало багато часу і обмежувало людей в більшості місцем їхнього проживання та графіком роботи магазинів. З приходом таких сайтів, як Amazon та eBay, покупці отримали можливість переглядати, порівнювати та купувати товари з усього світу в будь-який зручний для них час. Ці веб-сервіси не лише дозволити зберегти час і зусилля, але й надали ширший асортимент товарів за більш конкурентоспроможними цінами, до яких користувач може отримати доступ коли і де завгодно.

Не можна не згадати роль, яку веб-застосунки відіграли у розвитку соціальної сфери життя. Соціальні мережі, такі як Facebook, Twitter та Instagram, переосмислили спосіб, у який ми встановлюємо зв'язки, спілкуємося та обмінюємося інформацією. Веб-застосунки дозволили спілкуватись з людьми по всьому світу, вести дискусії з громадою, а також організовувати чи брати

участь у віртуальних заходах, які раніше вимагали фізичної присутності чи фінансових вкладень [39].

У секторі охорони здоров'я веб-застосунки сприяли розширенню доступу до медичної інформації та послуг. За допомогою телемедичних застосунків пацієнти можуть консультуватися з лікарями за допомогою відеодзвінків, що особливо корисно для тих, хто знаходиться далеко або має проблеми з мобільністю. З іншого боку, веб-застосунки спростили управління медичною документацією, планування зустрічей і роботу з рецептами, скорочуючи час очікування і пов'язані з цим адміністративні витрати.

Проблема загублених речей, залишаючись неприємною, отримала більше шансів бути вирішеною з появою веб-сервісів. Сервіси, такі як Tile і Chipolo, надають невеликі Bluetooth-пристрої, які можна прикріпити до предметів - ключів, гаманців або навіть домашніх тварин, - місцезнаходження яких можна відстежувати за допомогою сервісів в Інтернеті. Часто ці програми покладаються на спільноту користувачів, які допомагають знайти предмети, що знаходяться поза зоною дії Bluetooth-пристроїв, сповіщаючи власника, коли інший девайс виявляє позначений предмет. Поряд з цим, соціальні мережі стали місцем утворення онлайн-спільнот для пошуку загублених речей, де користувачі можуть розміщувати інформацію про загублені чи знайдені речі, домашніх тварин або навіть людей у разі надзвичайних ситуацій. І саме тут Інтернет та веб-технології проявляють свою силу, збільшуючи масштаби пошуку з локальних на глобальні.

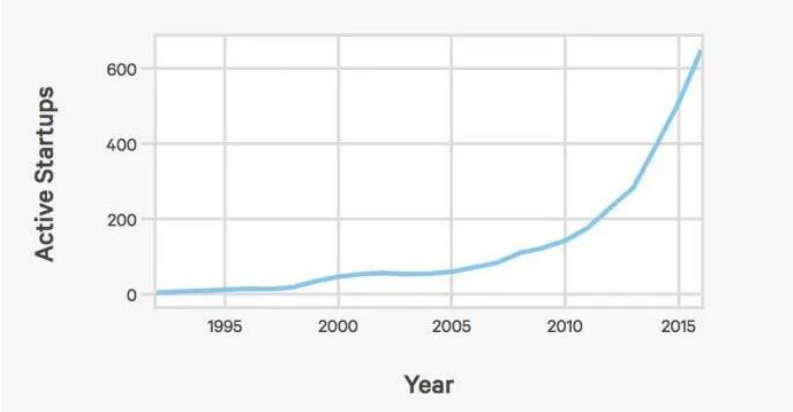

Рисунок 1.2 – Діаграма використання штучного інтелекту стартапами [36]

У майбутньому веб-додатки продовжуватимуть розвиватися, оскільки в них інтегрується все більше і більше нових технологій, таких як штучний інтелект (рис. 1.2) і машинне навчання, розширюючи межі можливого завдяки автоматизації та оптимізації [36]. Цей прогрес принесе ще більше зручності та підвищить ефективність виконання повсякденних завдань, водночас вимагаючи від веб-застосунків значних покращень персоналізації, доступності та функціональності.

1.2 Підходи до розроблення сучасних веб-застосунків

Сьогодні сфера веб-розробки характеризується великою різноманітністю і перебуває у стані постійної трансформації. Тому від сучасних веб-застосунків очікується, що вони повинні бути адаптивними на різних пристроях і з різним доступом до мережі, а також забезпечувати приємний користувацький досвід, який можна порівняти з нативними застосунками. Цей фактор спричинив появу цілої низки практик і архітектур розробки - від серверних додатків до клієнтських односторінкових застосунків і прогресивних веб-застосунків, основні відмінності яких наведено в таблиці 1.1 [22].

Таблиця 1.1 – Порівняння односторінкових та прогресивних веб-застосунків

Характеристика	Односторінковий веб-застосунок	Прогресивний веб-застосунок
Ключова концепція	Динамічне оновлення веб-сторінки без перезавантаження	Використання сучасних веб-технологій для отримання ефекту, схожого на нативний застосунок

Продовження таблиці 1.1

Досвід користувача	Плавна, безперервна взаємодія, подібна до десктопного додатку	З ефектом занурення, взаємодією та навігацією, схожою на додаток
Офлайн-можливості	Обмежені	Високі, з використанням Service Workers для кешування ресурсів і даних додатків
Встановлення	Доступ через веб-браузери без встановлення	Може бути встановлений на головні екрани пристроїв за допомогою іконки
Розробка та супровід	Простіша, орієнтована на веб-технології	Включає керування веб-функціями та нативними функціями, включно з офлайн-функціоналом

Основою будь-якого веб-додатку є його базові технології: HTML5, CSS3 та JavaScript. HTML5 - це остання версія мови гіпертекстової розмітки, яка має більше семантичних елементів для пояснення зміст у набагато простіший спосіб, таких як: <article>, <section>, <nav> і <footer>, що підвищує доступність і робить додатки набагато надійнішими і зрозумілішими як для розробників, так і для пристроїв. Доповнена потужним синтаксисом CSS3, мова стилів надає розробникам потужний засіб для створення візуально привабливих та адаптивних користувацьких інтерфейсів за допомогою розширених селекторів, анімації та функцій адаптивного дизайну [30]. JavaScript, який колись був інструментом для написання простих сценаріїв на стороні клієнта, став основою

для інтерактивних і динамічних функцій у веб-додатках. Набагато більше функціональності було додано з впровадженням AJAX, а пізніше фреймворків, які дозволили створювати складні додатки у більш керований та ефективний спосіб.

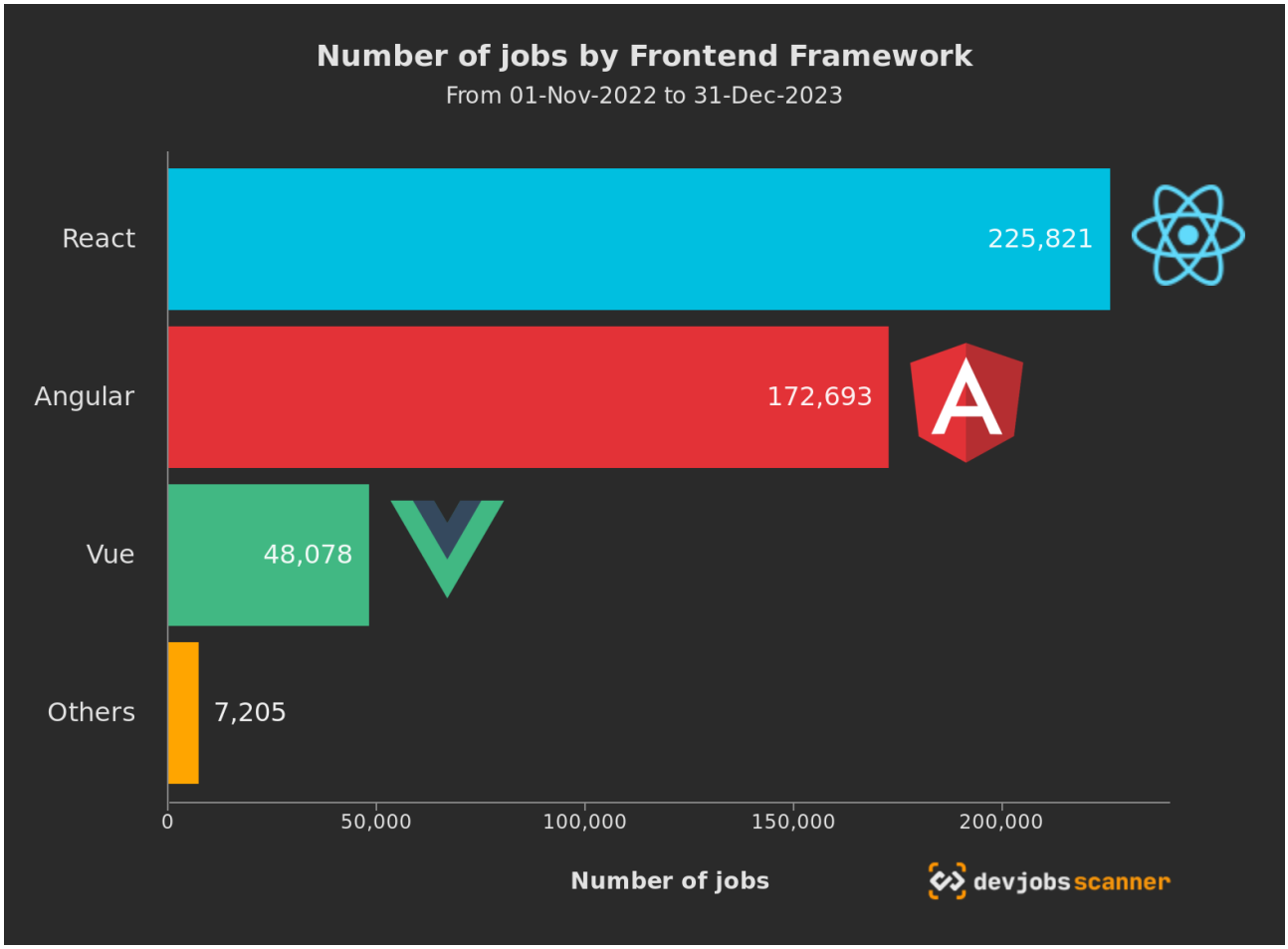

Рисунок 1.3 – Діаграма популярності фронтенд фреймворків [29]

React, Angular та Vue.js вважаються найпопулярнішими інструментами для створення інтерфейсів користувача (рис. 1.3). React, розроблений компанією Facebook, має компонентну архітектуру, що забезпечує простоту управління складним станом і динамічним контентом, яким так славляться сучасні веб-додатки. Він відомий своєю функцією віртуального DOM, яка оптимізує процеси повторного рендерингу для підвищення продуктивності.

Angular, фреймворк від Google, пропонує велику кількість інтегрованих функцій, таких як обробка форм, маршрутизація та управління станами, які

можуть швидко прискорити процес розробки, але мають більш круту криву навчання. Він використовує TypeScript, який привносить статичну типізацію в JavaScript і дозволяє розробляти більш складні та стійкі до помилок додатки.

Vue.js також відомий своєю легкістю та гнучкістю, поєднуючи в собі функції React та Angular. Це означає, що він не тільки доступний для початківців, але й достатньо потужний для просунутих додатків. Vue.js пропонує реактивну модель компонентів, яку можна комбінувати. Останнім часом фреймворк набув популярності через низький поріг входу та екосистемі, що поступово адаптується [29].

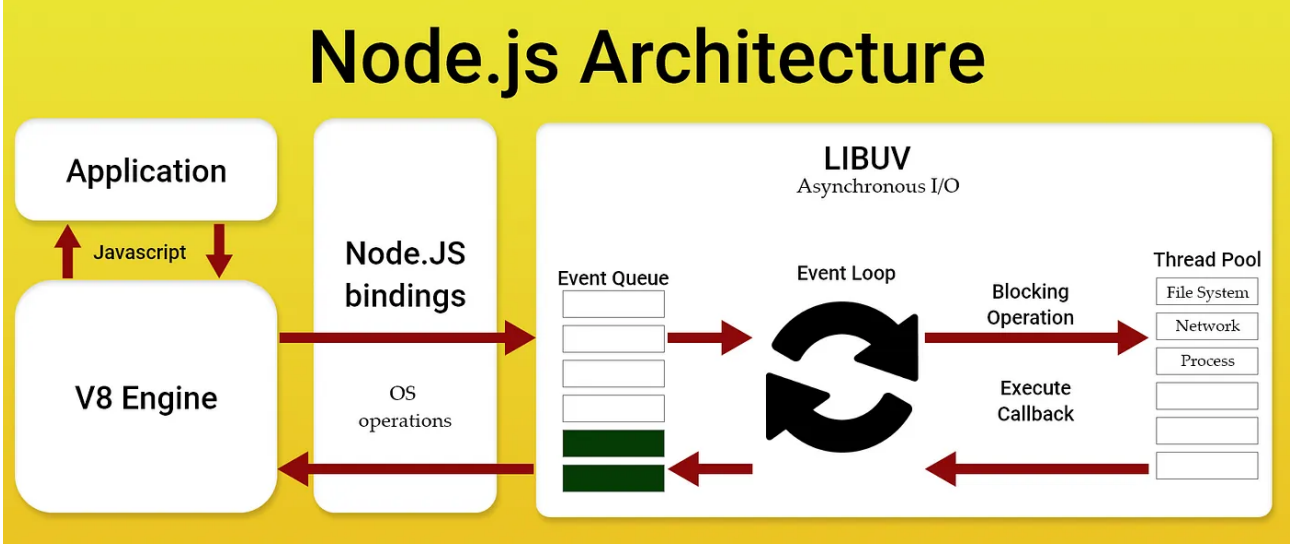

Рисунок 1.4 – Архітектура бекенд фреймворку Node.js [26]

На стороні сервера Node.js зробив революцію у використанні JavaScript, дозволивши розробникам запускати JavaScript на сервері. Його неблокуюча архітектура, керована подіями, ідеально підходить для створення масштабованих мережевих додатків (рис. 1.4) [26]. Для мови Python, існують такі фреймворки, як Django та Flask. Django - це високорівневий фреймворк з повним стеком, який робить акцент на швидкій розробці та чистому, прагматичному дизайні, пропонуючи адміністративний інтерфейс «з коробки» і абстрагуючись від більшої частини інфраструктури сайту, щоб розробники могли зосередитися на написанні програми, не вигадуючи велосипед. Flask, навпаки, пропонує набагато легший і модульний спосіб розробки, дозволяючи розробникам

використовувати будь-які інструменти та бібліотеки, які вони вважають за краще для створення додатків [15].

У сфері сучасної веб-розробки перехід від монолітних архітектур до мікросервісів став значною тенденцією, зумовленою потребою в більш адаптивних і масштабованих структурах застосунків. Традиційно монолітні рішення розроблялися як єдиний, неподільний блок, де всі компоненти - від обробки вводу до обробки даних і рендерингу - були тісно інтегровані і розгорнуті разом. Такий підхід, хоч і простий, часто призводить до створення складних і громіздких кодових баз, які важко масштабувати та оновлювати в міру зростання [16]. Перехід до мікросервісів став способом вирішення таких проблем, де рекомендується декомпозиція великих застосунків на менші сервіси, що розгортаються незалежно один від одного. Сервіс в рамках архітектури мікросервісів працює незалежно і взаємодіє з легким, чітко визначеним механізмом для досягнення бізнес-цілі. Детальніше порівняння цих двох підходів наведено в таблиці 1.2. Ця архітектура пов'язана з принципами розробки API-first, наголошуючи на проектуванні API на початку процесу розробки. Прийняття підходу, що базується на API, має переваги у гнучкості та масштабованості. Завдяки попередньому проектуванню, API гарантує, що кожна мікропослуга будується на основі узгодженого контракту щодо комунікацій, які необхідно здійснити (рис. 1.5) [34]. Це полегшує інтеграцію та взаємодію між сервісами, а також спрощує оновлення або заміну окремих компонентів, не порушуючи роботу всієї системи. API визначають межі для кожного сервісу, і робота над різними аспектами програми може легко виконуватися паралельно різними командами, що значно прискорює цикли розробки [33].

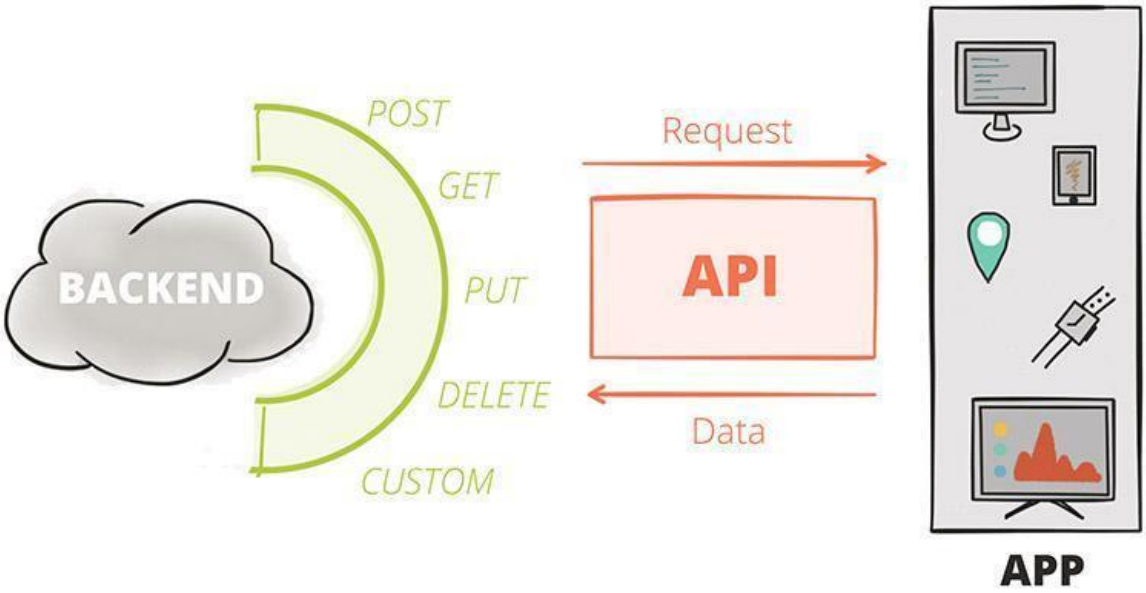

Рисунок 1.5 – Структура REST API [40]

Крім того, дизайн, орієнтований на API, полегшує розширення функціональності, доступної зовнішнім партнерам і стороннім розробникам, таким чином збільшуючи можливості та охоплення додатку. Це заохочує екосистему, в якій сервіси можна повторно використовувати і компонувати по-новому, щоб підвищити загальну цінність і функціональність додатку. Що стосується масштабованості, то мікросервіси мають перевагу, дозволяючи масштабувати певні області додатку на основі попиту, не впливаючи на продуктивність інших сервісів [23]. Цілеспрямоване масштабування є більш економічно вигідним та ефективним, ніж масштабування монолітного додатку, оскільки монолітний додаток вимагає масштабування всього додатку, навіть якщо лише одна функція має високий попит.

Така зміна парадигми слугує не лише технічній масштабованості веб-додатків, але й зростанню та адаптації бізнесу. Потреби програмного застосунка швидко змінюються в міру розвитку бізнесу, а мікросервісна архітектура, що підтримується підходом API-first, дозволяє швидко вносити зміни та корективи, не збільшуючи час простою або переробки (табл.1.2). Це надає архітектурі гнучкості, яка допомагає будь-якому бізнесу бути настільки

конкурентоспроможним, наскільки це необхідно цифровому ринку, що безупинно змінюється та розвивається.

Таблиця 1.2 – Порівняння монолітної та мікросервісної архітектур

Параметр	Монолітна архітектура	Мікросервісна архітектура
Масштабованість	Масштабується як єдине ціле, що може бути неефективним	Кожен сервіс можна масштабувати незалежно, що підвищує ефективність
Розгортання	Розгортання всього додатку необхідне для оновлень	Сервіси можна розгортати незалежно, не впливаючи на інші
Розробка	Простіша на початковому етапі, але може стати складною	Складніша у налаштуванні, але пропонує більшу гнучкість і маневреність
Ізоляція збоїв	Несправність в одному компоненті може вплинути на всю систему	Несправності в одному сервісі зазвичай не впливають на інші

1.3 Огляд існуючих рішень підтримки комунікації для пошуку втрачених речей

Втрата та пошук речей у громадських місцях була і залишається поширеною проблемою, з якою стикаються люди по всьому світу. Традиційно повернення загублених речей до їхніх власників відбувалося за допомогою

пунктів знахідок, дощок оголошень або місцевих органів влади. Ці методи, хоча й були певною мірою ефективними, часто були повільними і незручними, що вело до низького рівня успішності повернення загублених речей. Поява технологій, зокрема розвиток інтернет-сервісів і соціальних мереж, докорінно змінила підхід до роботи з загубленими речами, зробивши цей процес ефективнішим і зручнішим для користувачів. З поширенням смартфонів та інтернету з'явилося багато веб-сервісів і платформ, які допомагають подолати розрив між тими, хто втрачає речі, і тими, хто їх знаходить.

Одним з таких веб-сервісів є iLost.com, що надає глобальну платформу з великою базою даних знайдених та втрачених речей та функціоналом для встановлення зв'язку між тими, хто загубив і знайшов речі (рис. 1.6).

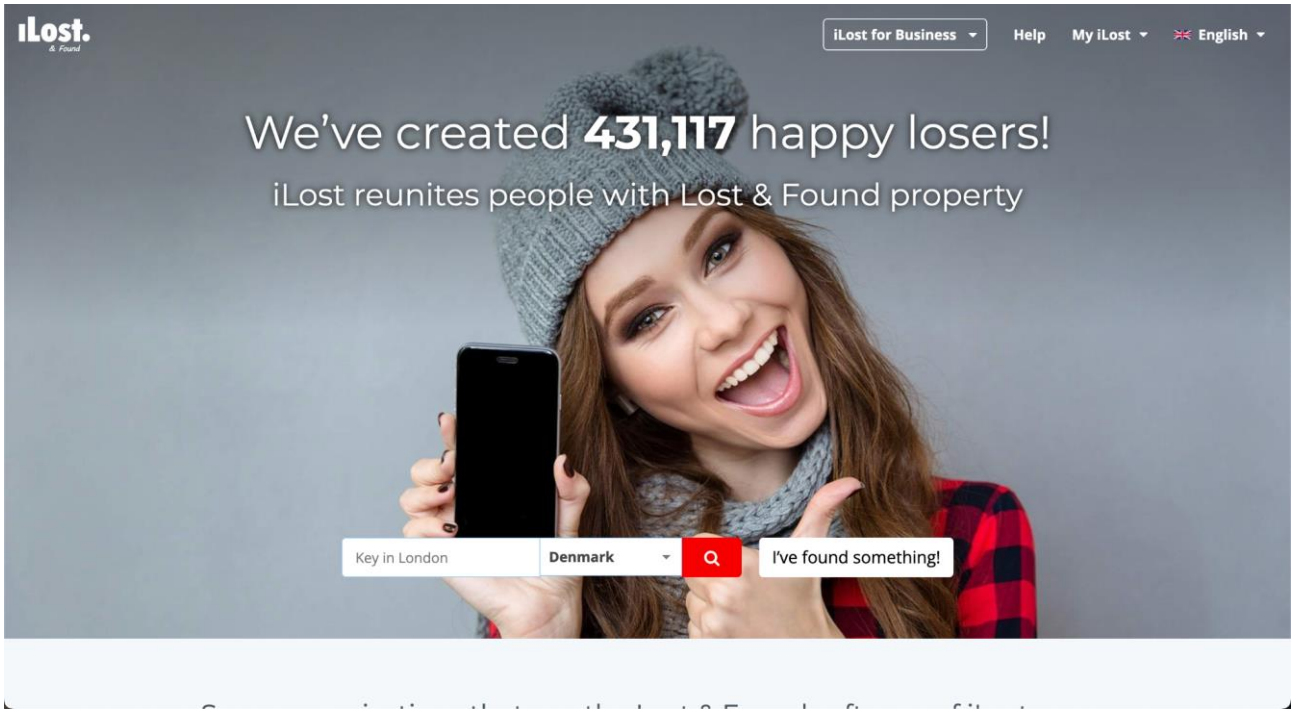

Рисунок 1.6 – Головна сторінка сервісу iLost.com

Цей сервіс розроблений таким чином, щоб бути зручним для користувачів і доступною по всьому світу, забезпечуючи оновлення в режимі реального часу інформації про загублені та знайдені речі які від користувачів, так і від громадських закладів, таких як аеропорти, готелі, громадський транспорт та розважальні заклади. Користувачі можуть просто ввести дані про загублену річ і місце, де вона була загублена, та здійснити пошук в онлайн списках. Так само

звичайний користувач може створити пост про знайдену річ, вказавши всю детальну інформацію про річ, а саме: завантажити зображення загубленої речі; місце, де її було знайдено; а також надати свої контакти для зворотного зв'язку (рис. 1.7).

Сервіс є безкоштовним і широко співпрацює з громадськими закладами. В свою чергу, основним недоліком є те, що, незважаючи на велику кількість країн у яких застосунок офіційно працює, все ж він досить локальний, адже охоплює тільки країні Європейського Союзу, США, а також кілька азійських країн [14].

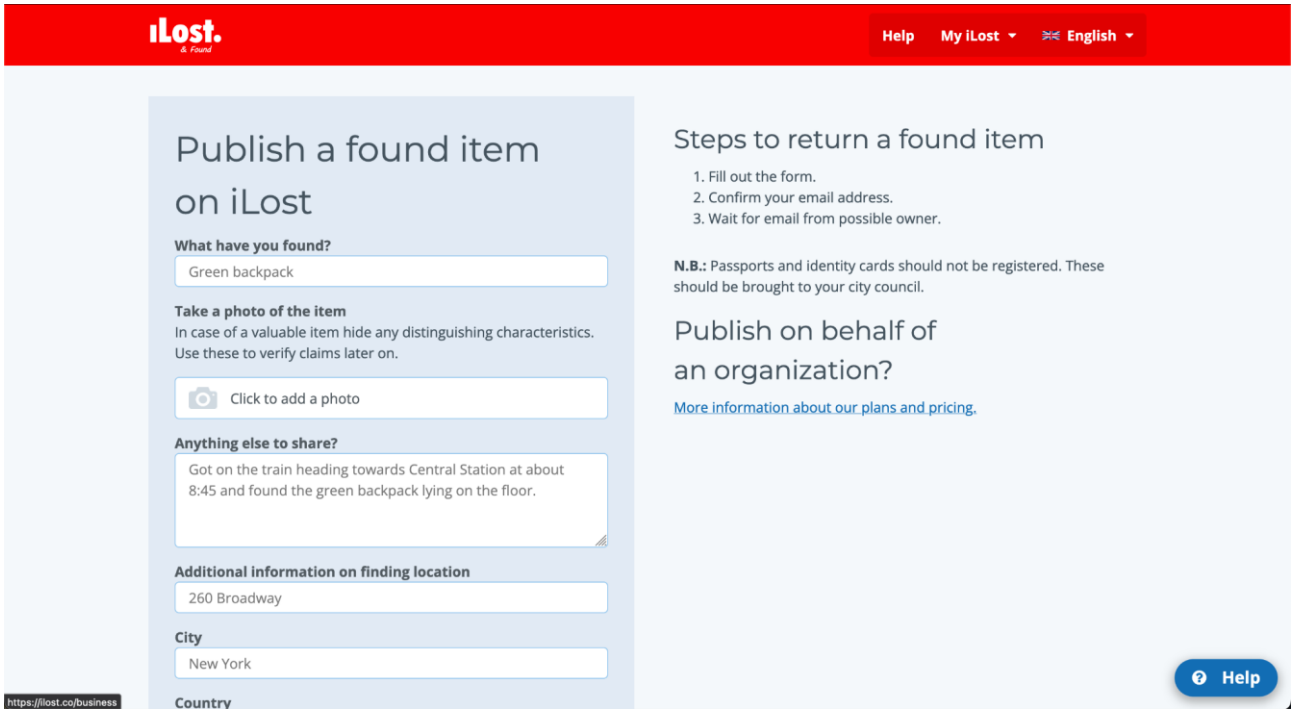

Рисунок 1.7 – Сторінка створення посту про знайдену річ на iLost.com

Пошук у застосунку iLost надається тільки для постів про вже знайдені речі. Користувач може вказати ключові слова , вибрати країну пошуку, вказати дату, коли приблизно було загублено річ, а також вибрати афілійовану організацію (рис. 1.8). Надається можливість сортувати пости в алфавітному порядку, відносно дати, вказаної при створенні поста, та по релевантності.

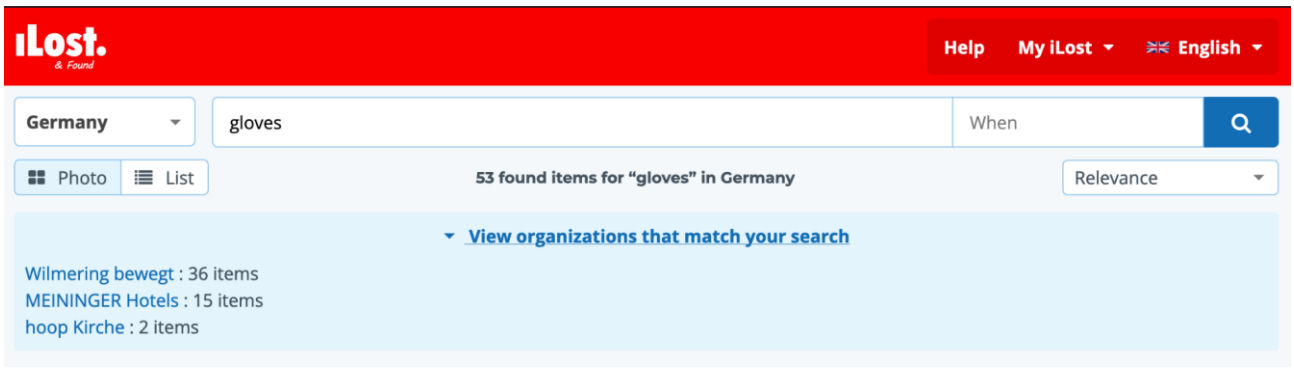

Рисунок 1.8 – Фільтри пошуку постів про знайдені речі

Після того, як користувач ідентифікує свою втрачену річ на веб-сайті iLost.com та ініціює заяву, він повинен надати конкретні дані або докази для підтвердження права власності, наприклад, детальний опис або серійний номер. У свою чергу, особа, яка розмістила пост, перевіряє цю інформацію, щоб підтвердити право власності. iLost полегшує цю взаємодію через свою платформу, забезпечуючи конфіденційність і безпеку комунікації, надаючи можливість залишати приватні коментарі під постами (рис. 1.9).

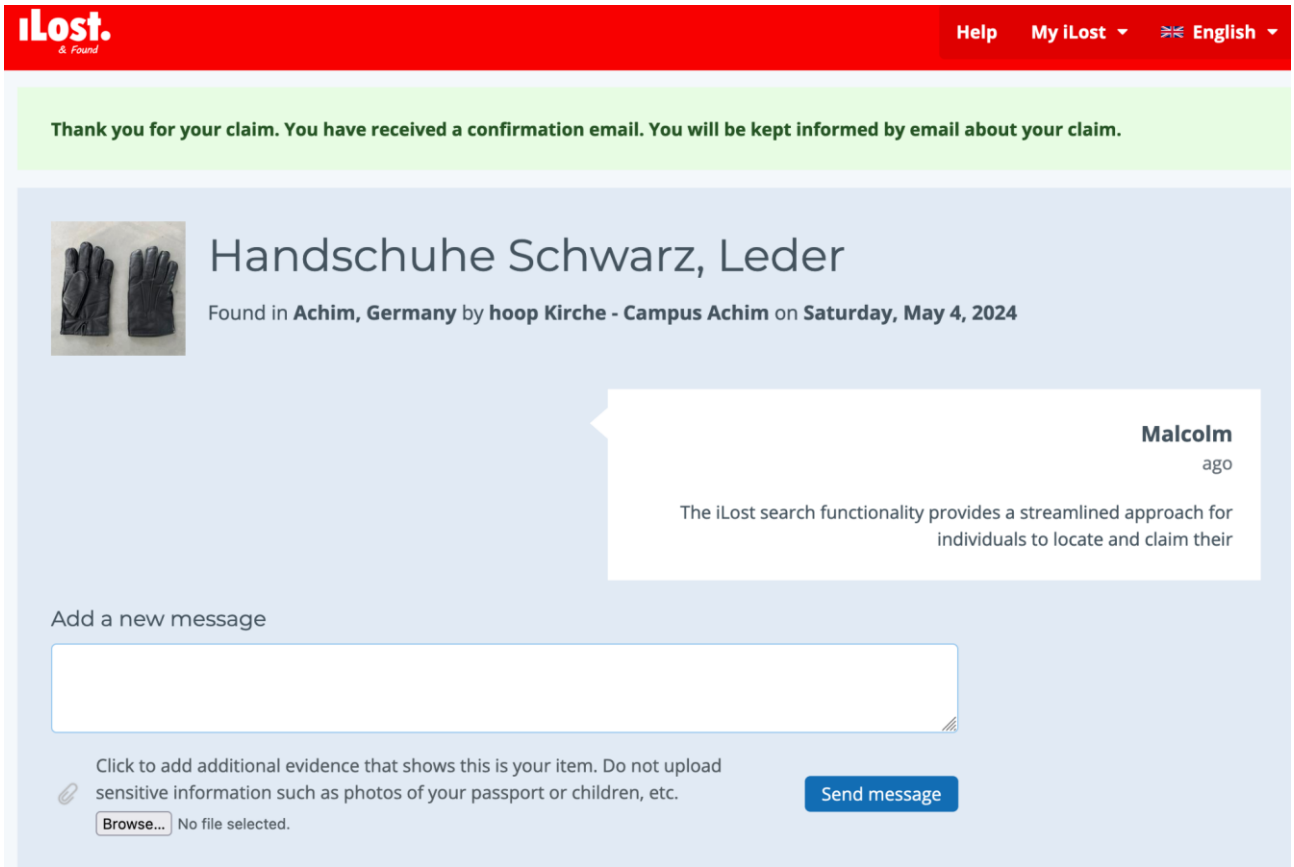

Рисунок 1.9 – Сторінка комунікації між користувачами

Веб-сервіс LostAndFound.com слугує подібній меті, пропонуючи комплексне онлайн-рішення для пошуку загублених речей, яке охоплює широкий спектр предметів. Завдяки зручному інтерфейсу, люди можуть повідомляти про загублені речі або шукати їх за допомогою організованої системи (рис. 1.10). Платформа також включає мобільний додаток, що підвищує доступність і зручність для користувачів, які перебувають у дорозі.

Інтеграція онлайн та мобільних платформ гарантує, що користувачі завжди можуть залишатися на зв'язку та бути проактивними у своєму пошуку.

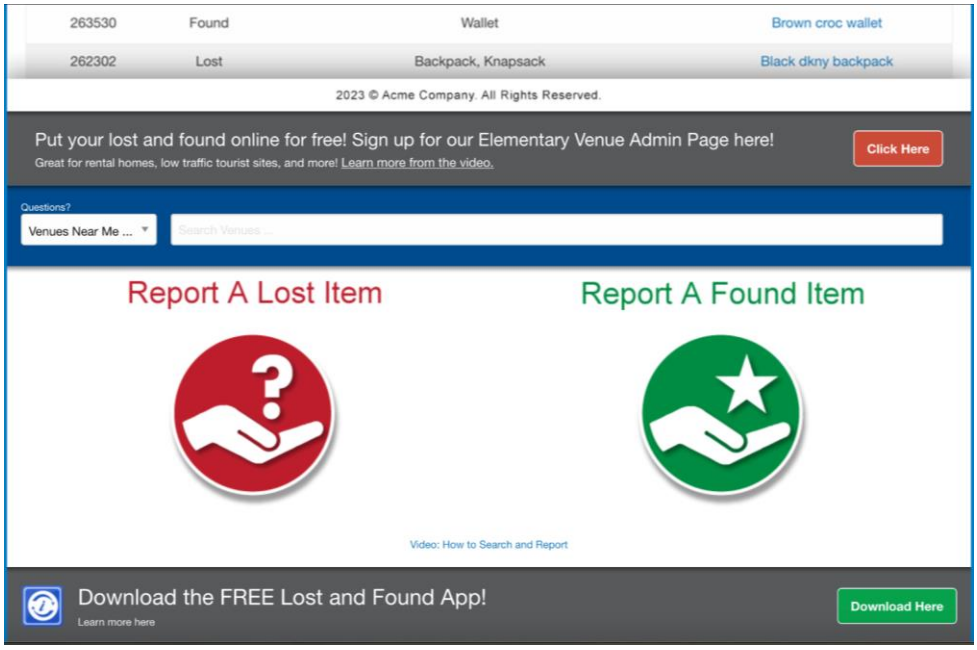

Рисунок 1.10 – Головна сторінка сервісу LostAndFound.com

Користувачі можуть створювати повідомлення як про загублені, так і про знайдені речі. Ці повідомлення можуть містити детальний опис, фотографії та конкретну інформацію про предмет, наприклад, час і місце, де він був загублений або знайдений. Платформа має функцію розширеного пошуку, яка дозволяє користувачам шукати речі за кількома критеріями, включаючи місцезнаходження, категорію і часовий проміжок [28]. Це допомагає звузити результати і збільшити ймовірність збігу.

Користувачі мають доступ до особистої інформаційної панелі, де вони можуть керувати своїми публікаціями, відстежувати пошукові запити та оновлювати інформацію про свій профіль. LostAndFound.com має функціонал

сповіщеннь, щоб тримати користувачів в курсі потенційних збігів або нових повідомлень, які відповідають їхнім критеріям пошуку.

РОЗДІЛ 2
АНАЛІЗ ВИМОГ ТА АРХІТЕКТУРИ ВЕБ-ЗАСТОСУНКІВ

2.1 Аналіз способів аутентифікації у веб-застосунках

У веб-розробці концепція авторизації, що включає в себе захист додатку і забезпечення доступу користувачів лише до тих ресурсів, до яких вони мають дозвіл, має першочергове значення. Авторизація відрізняється від процесу аутентифікації, який насправді відноситься до перевірки особи, тому що це специфікація того, що ці автентифіковані користувачі можуть робити. Концепція авторизації може бути реалізована різними способами, залежно від вимог програми та архітектури. Популярними методами є управління доступом на основі ролей, OAuth та більш динамічні підходи, такі як управління доступом на основі атрибутів [31]. Основою авторизації в сучасному світі веб-розробки є JWT, а AWS Cognito з API Gateway доповнюють його та спрощують реалізацію авторизаційної логіки для розробників, залишаючись при цьому безпечними та надійними.

JSON Web Tokens (JWT) визначають відкритий стандарт (RFC 7519) для компактної та автономної захищеної передачі інформації між двома сторонами у вигляді JSON-об'єктів. Ці дані підписуються цифровим підписом, а отже, їм можна довіряти і їх можна перевірити. Як правило, JWT можуть бути підписані за допомогою секретного (за допомогою алгоритму HMAC) або відкритого/закритого ключа з використанням RSA або ECDSA. Вони є потужним інструментом для авторизації, оскільки дозволяють власнику токена отримати доступ до ресурсу без численних перевірок облікових даних користувача кожного разу.

Рисунок 2.1 – Структура заголовка JWT токена

Веб-токен JSON складається з трьох частин: заголовка, корисного навантаження і підпису. Заголовок зазвичай складається з двох частин: тип токена - тобто JWT - і алгоритму підпису, який використовується, наприклад, HMAC SHA256 або RSA (рис. 2.1). Друга частина токена, яка містить твердження, називається корисним навантаженням. Реквізити - це твердження про сутність, зазвичай користувача, і додаткові метадані. Існує три типи реквізитів: зареєстровані, публічні та приватні реквізити [21]. Підпис призначений для захисту токена від підробки. Наприклад, токен, підписаний алгоритмом HMAC SHA256, буде згенерований таким чином, щоб зашифрувати заголовок і корисне навантаження.

Cognito є комплексним рішенням для аутентифікації та авторизації користувачів в екосистемі AWS, а отже він безперешкодно працює з іншими сервісами AWS, такими як API Gateway. AWS Cognito підтримує реєстрацію та вхід користувачів, а також надає функції об'єднаної ідентичності, які дозволяють розробникам створювати пули ідентичностей, де користувачі можуть автентифікуватися з допомогою сторонніх провайдерів, таких як Google, Facebook та Amazon, або безпосередньо через пул користувачів Cognito [17].

Інтеграція між Cognito та API Gateway дозволяє розробникам захистити доступ до своїх API. Щоразу, коли клієнт надсилає запит на кінцеву точку, захищену Cognito, API-шлюз спочатку перевіряє наявність дійсного JWT у запиті. Якщо токен дійсний, запит буде перенаправлено на внутрішній сервіс. В іншому випадку цей запит буде відхилено. По суті, така інтеграція відокремлює механізм авторизації від логіки додатку, що робить його дуже простим в управлінні безпекою і масштабованістю.

У цьому відношенні Serverless Framework лише продовжує сучасну тему безшовної інтеграції веб-додатків, чому сприяє потужність даного інструменту, який спрощує розгортання хмарних додатків, абстрагуючись від значної частини управління інфраструктурою. Це виявляється дуже зручним при інтеграції сервісів аутентифікації та авторизації, таких як AWS Cognito, оскільки дозволяє

більше зосередитися на написанні логіки додатку, а не витрачати час на базову інфраструктуру [25].

```
httpApi:
  cors: true
  authorizers:
    ApiGatewayAuthorizer:
      type: jwt
      identitySource: $request.header.Authorization
      issuerUrl:
        Fn::Sub:
          - 'https://cognito-idp.${aws:region}.amazonaws.com/${UserPoolId}'
          - UserPoolId: !Ref UserPool
      audience:
        - Ref: UserPoolClient
```

Рисунок 2.2 – Налаштування авторизації ендпоінтів з використанням Cognito в Serverless Framework [25]

Serverless Framework підтримує просту інтеграцію з AWS Cognito, що дозволяє легко налаштувати авторизацію для застосунку. Лише кілька конфігурацій у файлі serverless.yml (див. рис. 2.2) дозволяють встановити, які ресурси слід створити для пулу користувачів Cognito і як він повинен взаємодіяти з лямбда-функціями AWS та API-шлюзом. Налаштування не лише керує потоками автентифікації, але й чітко інтегрується з функціями авторизації, що надаються AWS, використовуючи надійність Cognito.

Розглянемо найпростішу архітектуру сервісу авторизації (рис. 2.3), що складатиметься з таких ендпоінтів:

- sign-up - ініціалізація процесу створення користувача в пулі користувачів Cognito;
- confirm - підтвердження емейлу способом введення надісланого на email коду;
- sign-in - аутентифікація користувача та отриманням JWT.

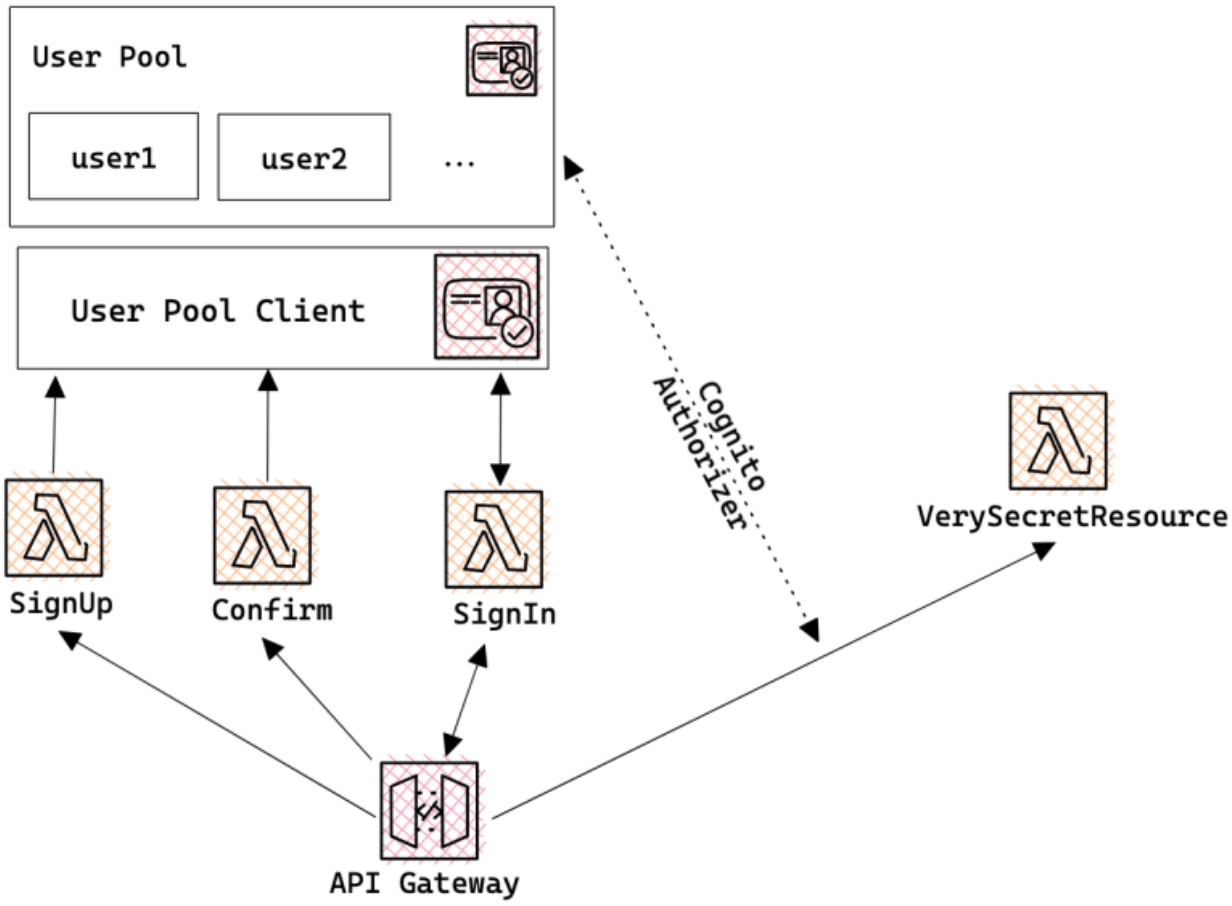

Рисунок 2.3 – Приклад простої інфраструктури аутентифікації в AWS [17]

Процес аутентифікації починається з пулу користувачів, де зберігаються облікові дані та інформація про профілі всіх зареєстрованих користувачів. Клієнт пулу користувачів виконує такі операції, як реєстрація, підтвердження та вхід до пулу користувачів.

На етапі реєстрації нові користувачі реєструються з необхідними даними, такими як ім'я користувача та пароль. Ці дані надсилаються від клієнта до пулу користувачів Cognito [32]. Наступне, що зазвичай відбувається, це те, що зареєстровані користувачі повинні підтвердити активацію свого облікового запису, зазвичай за допомогою коду підтвердження, надісланого на їхню електронну пошту або телефон, який вони вводять для завершення процесу реєстрації. Після того, як користувачі підтверджені, вони можуть увійти за допомогою своїх облікових даних. Якщо облікові дані підтверджені AWS Cognito, повертається токен автентифікації у вигляді JWT. У випадку, якщо

користувач намагається отримати доступ до захищеного ресурсу, в даному випадку «VerySecretResource», API-шлюз буде використовувати Cognito Authorizer для перевірки токена під час входу в систему [20].

Якщо токен дійсний і користувач має відповідні дозволи на доступ, то Cognito Authorizer надає доступ. Якщо ні, то запит відхиляється. Таким чином, AWS Cognito обробляє перевірку особи користувача та авторизацію за допомогою токенів автентифікації, так що тільки автентифіковані та авторизовані користувачі можуть отримати доступ до захищених ресурсів завдяки використанню шлюзу API - надійного методу захисту кінцевих точок API в безсерверному середовищі [19].

2.2 Оцінка технологічних стеків та їх впливу на продуктивність

Для створення веб-застосунку розглянемо наступний технологічний стек: TypeScript, Node.js, React, MaterialUI, MongoDB, AWS S3, AWS Lambda, AWS Cognito, AWS SQS, AWS SES, Serverless Framework та Visual Studio Code.

React - одна з найпопулярніших бібліотек для фронтенд розробки, яка дуже добре підходить для створення інтерфейсів користувача, особливо для односторінкових додатків, де необхідна динамічна взаємодія.

```
import React from 'react';
import ReactDOM from 'react-dom';

function Message({ name }) {
  return <h1>Hello {word}!</h1>;
}

ReactDOM.render(<Message word="world" />, document.getElementById('app'));
```

Рисунок 2.4 – Створення простого компонента в React

Вона була створена компанією Facebook і дозволяє створювати інтерфейси за допомогою багаторазово використовуваних фрагментів коду, які називаються компонентами (рис. 2.4). Ці компоненти керують своїм станом і комбінуються, дозволяючи будувати складні інтерфейси користувача. Однією з ключових

особливостей React є віртуальний DOM, який оптимізує оновлення веб-сторінки, перемальовуючи лише ті компоненти, що змінилися, а не всю сторінку, що відображається на швидкості взаємодії з користувачами [27]. Використання React з TypeScript додає в розробку статичні типи, що допомагає виявляти помилки на ранній стадії циклу розробки, роблячи кодову базу більш надійною та легкою для рефакторингу [2].

Бібліотека компонентів MaterialUI надає багатий набір попередньо розроблених компонентів, які відповідають основним принципам Material Design, забезпечуючи красивий вигляд і даючи відчуття ідеальності інтерфейсу [24]. Це особливо зручно для створення прототипів або розробки застосунків, які вимагають професійного вигляду без зайвих зусиль.

На стороні сервера Node.js надає легку, ефективну платформу для створення мережевих застосунків. Серед найважливіших переваг технології - неблокуюча, керована подіями архітектура, яка дозволяє підтримувати велику кількість з'єднань одночасно, що робить її ідеальною для веб-додатків, які потребують роботи в реальному часі, наприклад, чат-додатків або оновлень в реальному часі. Особливістю середовища є використання асинхронної моделі, яка показує свою перевагу, коли має справу з такими операціями, як читання або запис у файлову систему, мережевими операціями або будь-якими завданнями вводу-виводу [18]. Така архітектура надає можливість паралельно виконувати інші завдання, поки він чекає на завершення операцій вводу-виводу, роблячи застосунок більш ефективним і продуктивним.

Наприклад, у традиційному синхронному виконанні, коли сервер читає файл або обробляє запит до бази даних, він буде чекати завершення кожного читання або запиту, перш ніж перейти до наступного завдання. На відміну від цього, Node.js відправляє запит на читання файлу, а потім продовжує виконувати інші запити. Коли читання файлу завершено, середовище отримує результати і продовжує їх обробку - і все це без блокування основного потоку. У свою чергу AWS Lambda широко інтегрується з Node.js, щоб запускати код без виділення серверів або керування ними. Це дозволяє додаткам масштабуватися на

найвищому рівні і бути дуже економічно ефективними, оскільки Lambda автоматично масштабує виконання коду відповідно до вхідних запитів. Наприклад, функція Node.js на Lambda може запускатися за HTTP-запитами через Amazon API Gateway, обробляти дані з інших сервісів AWS, таких як S3 або DynamoDB.

MongoDB - це документно-орієнтована NoSQL-база даних, що використовується для зберігання великих обсягів даних. Подібно до того, як реляційна база даних зберігає дані в рядках і стовпцях, MongoDB зберігає дані в JSON-подібному документі з динамічною схемою. Така концепція робить інтеграцію даних простішою та швидшою завдяки гнучкості схем документів, що означає те, що в одній колекції можуть бути документи з дуже різними структурами. База даних розроблена для масштабування та продуктивності за допомогою таких функцій, як шардінг, який розподіляє дані між кластером серверів [11]. Вона також підтримує реплікацію, що забезпечує високу доступність завдяки наборам реплік з двох або більше вузлів.

```
db.orders.aggregate([
  {
    $group: {
      _id: "$customer_id",
      totalAmount: { $sum: "$amount" }
    }
  }
]);
```

Рисунок 2.5 – Приклад запиту в MongoDB з функцією агрегації

Основні можливості MongoDB наступні:

- зберігання різних типів контенту, таких як коментарі, профілі користувачів і публікації в блогах, в одній базі даних без необхідності налаштовувати окремі таблиці або зв'язки;
- фреймворк агрегації дозволяє виконувати складні операції агрегації даних (рис. 2.5) [11];

- мова запитів MongoDB підтримує пошук за полем, діапазонні запити та пошук за регулярними виразами, що допомагає ефективно реалізовувати такі функції, як пошукові фільтри та оновлення в режимі реального часу.

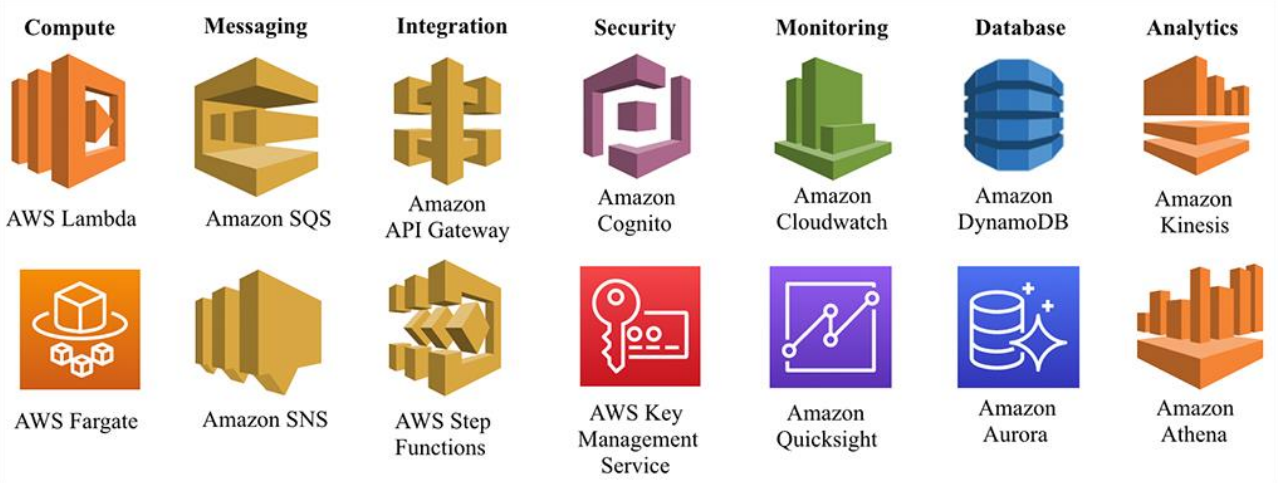

Рисунок 2.6 – Основні сервіси AWS для безсерверної розробки [41]

Сервіси AWS ще більше розширюють стек. Вони надають масштабовані та економічно ефективні рішення для обчислень, зберігання та безпеки, серед інших функцій, що дозволяє створювати складні додатки з підвищеною гнучкістю, масштабованістю та надійністю (рис. 2.6) [13].

AWS S3 є сервісом для зберігання об'єктів, що відомий своєю масштабованістю, доступністю даних, безпекою та ефективністю. Основними цілями для використання даного сервісу є зберігання та захист будь-яких обсягів даних для веб-сайтів, мобільних застосунків, для резервного копіювання та відновлення, архівування, для пристрої Інтернету речей та для аналізу великих даних. Наприклад, медіакомпанії використовують S3 для зберігання та розповсюдження великих відеофайлів по всьому світу [10].

AWS Lambda дозволяє запускати код без виділення ресурсів або керування серверами. Автоматичне масштабування досягається виконанням коду у відповідь на тригери від інших сервісів AWS, таких як S3 і SQS, що полегшує створення додатків, які швидко реагують на нову інформацію [12].

Автентифікація та контроль доступу з можливістю масштабуватись для підтримки мільйонів користувачів надається сервісом AWS Cognito. Простіше кажучи, він спрощує процес імплементації функціоналу реєстрації, входу та контролю доступу користувачів до застосунків. Користувачі можуть входити безпосередньо за допомогою імені користувача та пароля або через треті сторони, такі як Facebook, Amazon або Google.

AWS SQS є службою обміну повідомленнями, що дозволяє декомпозувати та масштабувати мікросервіси, розподілені системи та безсерверні застосунки. За допомогою SQS можна надсилати, зберігати та отримувати повідомлення між програмними компонентами без втрати повідомлень і без необхідності доступу до інших сервісів [9]. Це гарантує, що різні компоненти програми взаємодіють безперебійно та ефективно, навіть під час високого навантаження. Наприклад, в сервісах підтримки комунікації SQS можна використовувати для асинхронної обробки сповіщень через окрему систему, ефективно керуючи повідомленнями навіть у випадках пікового навантаження [8].

Разом ці хмарні сервіси формують надійну основу для багатьох типів застосунків, підтримуючи широкий спектр робочих навантажень і надаючи інструменти, необхідні для ефективного і безпечного створення, розгортання та управління безсерверними застосунками.

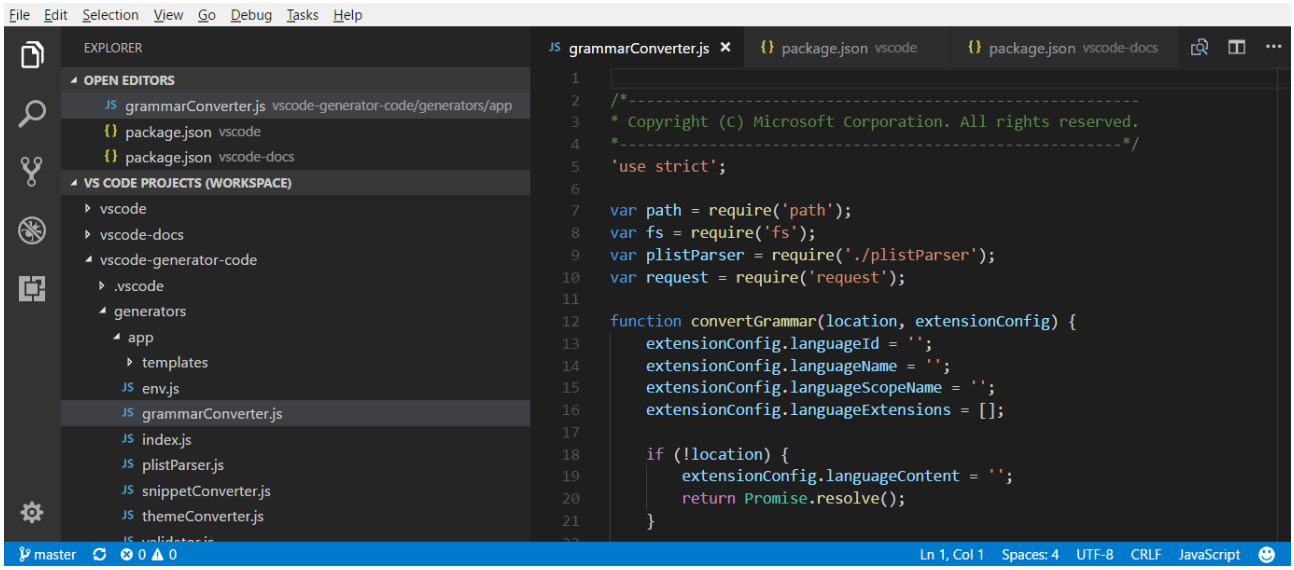

Рисунок 2.7 – Вигляд робочого середовища Visual Studio Code

Для розробки веб-застосунків одним з найкращих інструментів для написання коду є Visual Studio Code - надійне та гнучке інтегроване середовище розробки з потужною підтримкою JavaScript і TypeScript (рис. 2.7). Однією з основних переваг VSCode є велика бібліотека розширень, таких як Prettier для форматування коду та ESLint для виявлення проблемних патернів у TypeScript-коді, які розширюють його функціональність та адаптивність до різних робочих процесів. VSCode також має вбудовані інтеграції з Git, які полегшують контроль версій безпосередньо з IDE, спрощуючи процес розробки. Функція IntelliSense забезпечує інтелектуальне завершення на основі типів змінних, визначень функцій та імпортованих модулів, підвищуючи ефективність і точність кодування. Інтерфейс IDE, що налаштовується, дозволяє пристосовувати середовище розробки до різних потреб, підвищуючи продуктивність програміста [5].

Підсумовуючи, даний стек ідеально підходить для створення простого додатку, забезпечуючи збалансоване поєднання практичності, продуктивності та масштабованості, а масштабованість, економічна ефективність та надійна екосистема - здебільшого переважають решту недоліків.

РОЗДІЛ 3
ПРОЕКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ЕКСПЛУАТАЦІЯ ВЕБ-ЗАСТОСУНКУ

3.1 Проектування веб-застосунку

Зважаючи на функціонал наявних веб-застосунків з підтримки комунікації для пошуку втрачених речей у громадських місцях та їх аналізу, проведеного у розділі 1.3, визначимо наступні функціональні вимоги до веб-застосунку, що розробляється:

- користувач повинен мати змогу зареєструватися для користування веб-застосунком;
- користувач повинен мати змогу увійти у вже існуючий обліковий запис;
- користувач повинен мати змогу переглянути пости, які він опублікував;
- користувач повинен мати змогу переглянути список усіх постів про знайдені чи втрачені речі;
- користувач повинен мати змогу шукати пости за співпадінням слів в заголовку чи описі, за радіусом від певної точки на карті, а також типом поста;
- користувач повинен мати змогу створити пост з типами “Втрачено” або “Знайдено”, додати заголовок, опис, геолокацію, приблизну дату події, а також завантажити фото;
- користувач повинен мати змогу залишати коментарі під будь-яким постом;
- користувач повинен мати змогу редагувати опублікований пост;
- користувач повинен мати змогу оновити своє ім’я та аватар;
- користувач повинен мати змогу видалити опублікований пост;
- користувач повинен мати змогу видаляти свої коментарі, а також будь-які коментарі під своїм постом;

- користувач повинен мати можливість налаштовувати сповіщення про нові коментарі від інших користувачів під його власними постами.

Варто підкреслити, що дизайн веб-додатку повинен бути мінімалістичним, без зайвих елементів, які можуть відволікати увагу користувача. Користувацький інтерфейс повинен бути простим та інтуїтивно зрозумілим, включати знайомі, стандартизовані компоненти, які відображають дизайн часто відвідуваних веб-сайтів. Враховуючи, що користувачі можуть отримувати доступ до сайту в різних умовах, дуже важливо забезпечити безперебійну роботу і коректне відображення сайту на всіх основних пристроях, включаючи смартфони, планшети, нетбуки, ноутбуки і стаціонарні комп'ютери. Така крос-платформна сумісність підвищить доступність і забезпечить однаковий користувацький досвід незалежно від використовуваного пристрою. Цього можна досягти, використовуючи бібліотеку компонентів MaterialUI для React.

Для виконання поставлених вище вимог, спроектуємо високорівневу архітектуру застосунка з використанням сервісів AWS як на рисунку 3.1.
Веб-застосунок базуватиметься на сучасному, високопродуктивному інтерфейсі, розробленому за допомогою React.

Коли користувачі взаємодіятимуть із застосунко, відправляютимуться HTTP/2.0 запити. Використання сучасного протоколу HTTP/2.0 має важливе значення, оскільки він забезпечує швидший зв'язок між клієнтом і сервером, покращуючи час завантаження і можливості передачі даних на сервер [1]. Ці запити спочатку будуть спрямовані до AWS API Gateway, який виступає в якості оркеструючого рівня в архітектурі. API Gateway спрощує управління та моніторинг API, спрямовує вхідні запити до цільових сервісів та керує масштабуванням паралельних викликів API.

Для забезпечення безпеки та контролю доступів інтегруємо AWS Cognito безпосередньо з ApiGateway. Cognito аутентифікує користувачів і гарантуватиме, що їхні запити мають відповідні облікові дані та дозволи.

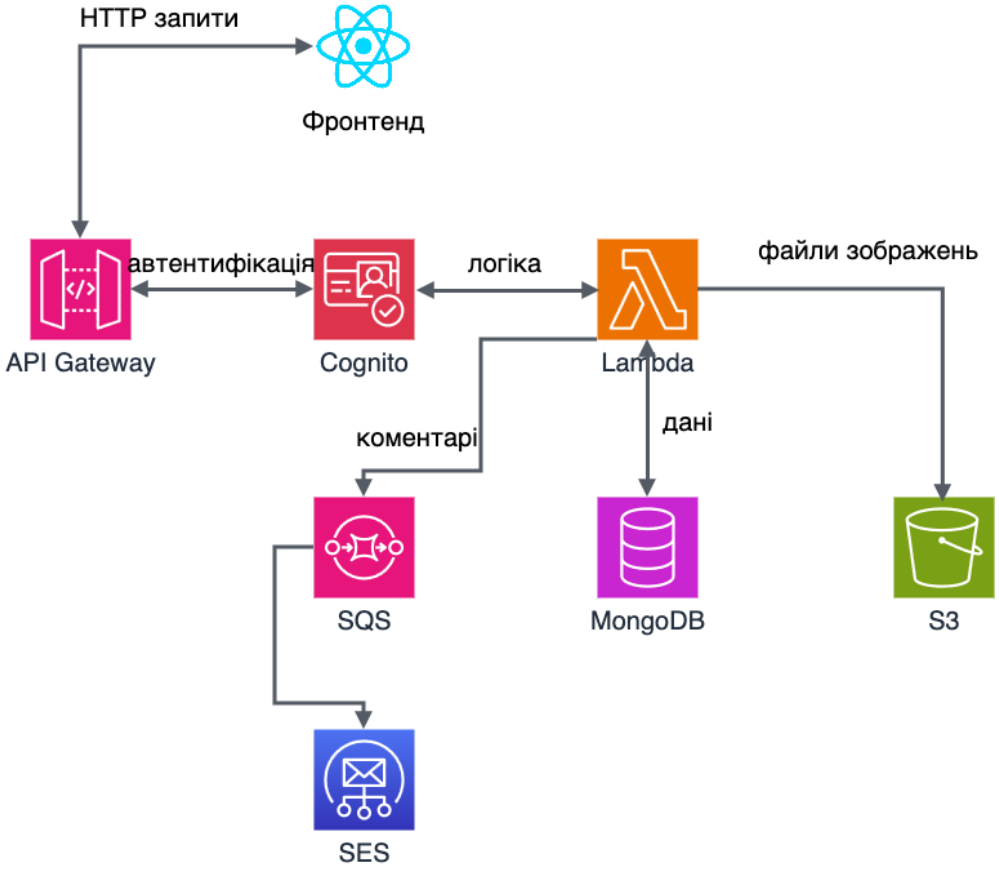

Рисунок 3.1 – Схема ресурсів веб-застосунку та зв’язку між ними

Після успішної автентифікації запити перенаправляються до AWS Lambda, безсерверного обчислювального компонента, який виконуватиме код з бізнес-логікою у відповідь на події, а саме повідомлення від API Gateway. Інтеграція Lambda дозволяє зменшити витрати на підтримку сервера і автоматично масштабується, обробляючи багато запитів паралельно. В основному лямбда-функції використовуватимуться для роботи з базою даних MongoDB.

Крім обробки даних, одна з лямбда-функцій, що відповідатиме за створення записів про коментарі до постів, буде зв’язана з сервісом AWS SQS. За умови успішного створення нового запису в базі даних, в чергу SQS прощатиметься повідомлення, які очікуватимуть до того моменту, поки їх не візьме в обробку наступний сервіс - AWS SES. Такий підхід відокремлює процес отримання даних від процесу їх обробки, що підвищує продуктивність і надійність, адже за умови виникнення помилки при надсиланні електронного листа - це не вплине на запис даних до бази. AWS SES не може бути напряму

зв’язана з чергою SQS, тому буде використана проміжна лямбда-функція, що формуватиме структуру листа, що надсилається, викликатиме сервіс SES, а також створюватиме запис у сервісі AWS CloudWatch про деталі виконання.

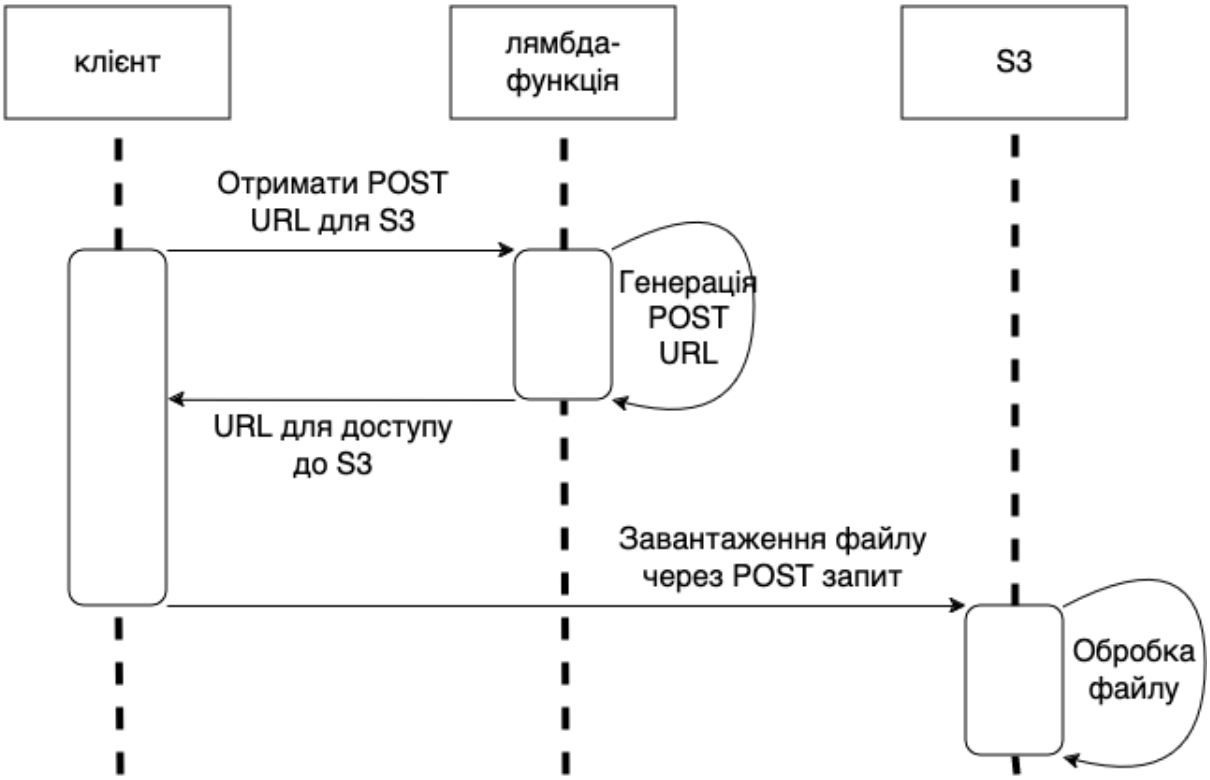

Рисунок 3.2 – Діаграма процесу збереження файлів

Для завантаження файлів, таких як зображення, у застосунку використаємо лямбда-функцію для генерації URL-адрес для безпосереднього тимчасового доступу для запису в AWS S3. Така URL-адреса надаватиме користувачам безпечний метод завантаження файлів зображень до S3 без прямої взаємодії з бекенд-сервісами [3]. До переваг цього підходу можна віднести підвищення безпеки бекенду, оскільки файл не буде оброблятись на серверній стороні, а напряму завантажуватиметься на S3, де AWS відповідальні за контроль процесу. До того ж це розвантажить бекенд від роботи з передачею об’ємних даних, оптимізуючи роботу.

Великою перевагою використання цієї архітектури з Serverless Framework є те, що вона спрощує розгортання та управління сервісами AWS, такими як Lambda, ApiGateway та SQS. Фреймворк спрощує конфігурацію і розгортання

безсерверних компонентів, що значно полегшує розробку і знижує ймовірність випадкових помилок. Він забезпечує рівень абстракції, який керує інфраструктурою з мінімально необхідною конфігурацією, звільняючи розробника від складнощів налаштування базової інфраструктури для написання логіки додатку. Фреймворк допомагає в управлінні кількома розгортаннями, наприклад, для різних середовищ, таких як середовище тестування та продакшн-середовище. А поєднання з інтегрованими інструментами моніторингу гарантує підвищує надійність застосунку. Іншими словами, Serverless Framework не тільки прискорює цикл розробки, але й удосконалює архітектуру застосунка, щоб задовольнити сучасні вимоги до веб-застосунків з високою ефективністю з точки зору вартості та експлуатаційної надійності.

3.2 Програмна реалізація веб-застосунку

Отже, для написання даного веб-сервісу було використано мову TypeScript, оскільки вона є досить зручною для написання застосунків із використанням фреймворку React [7].

Цей фреймворк є також зручним для написання фронтенд-частини, оскільки він пропонує використовування такого синтаксичного розширення як JSX – тобто надає можливість написання HTML-коду всередині JavaScript-файлу. Завдяки такому функціоналу можна легше розробляти інтерактивні компоненти фронтенд-частини. Однак, при написані компонентів з використанням JSX варто бути уважнішим, оскільки правила написання JSX-компонентів дещо строгіші ніж правила написання HTML.

Загальну структуру розробленого веб-сервісу можна представити у вигляді наступної схеми, як показано на рисунку 3.2.

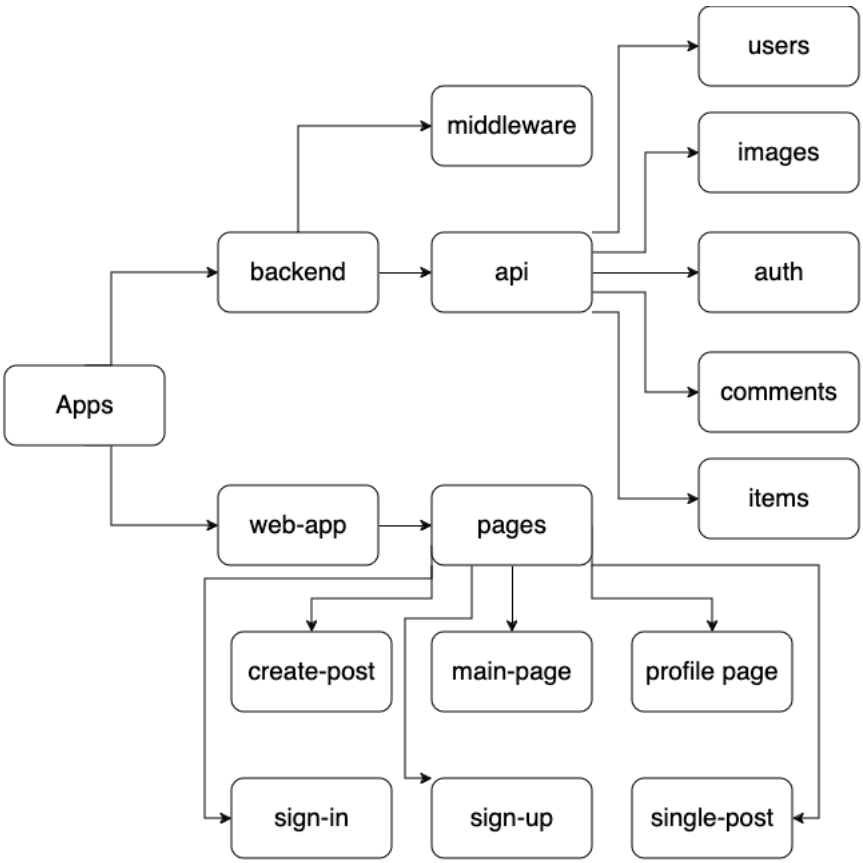

Рисунок 3.2 – Загальна структура веб-сервісу

У папці «api», зберігаються файли, що відповідають за бізнес логіку застосунку. В той час як папка «pages» формує структуру маршрутів та навігації користувача між сторінками веб-сервісу.

Папка «auth» відповідає за взаємодію з сервісом авторизації та автентифікації AWS Cognito, а також власною базою даних користувачів. Саме тут знаходиться логіка генерації JWT-токенів, що потім використовуються сервісом API Gateway для обробки HTTP-запитів.

Папка «pages» містить фронтенд компоненти та логіку, що відповідає за обробку отриманих з ендпоінтів даних, їх відображення користувачу, а також надання функціоналу для зміни - оновлення цих даних.

У папці «middleware» знаходяться компоненти проміжного програмного забезпечення такі, як логіка стандартизації виведення помилок та результатів HTTP-запитів, що є спільною для усіх ендпоінтів, а також код підключення до

бази даних. Більш детальну структуру із системними елементами та їхнім призначенням можна побачити у таблиці 3.1.

Таблиця 3.1 Авторизаційні ендпоінти

Ім’я	Призначення
auth/sign-up	Ініціалізація процесу реєстрації користувача
auth/confirm	Підтвердження емейлу користувача
auth/sign-in	Авторизація користувача

Таблиця 3.2 Ендпоінти коментарів

Ім’я	Призначення
comments/add-comment	Додавання коментаря до поста
comments/delete-comment	Видалення коментаря
comments/edit-comment	Зміна тексту коментаря

Таблиця 3.3 Ендпоінти постів

Ім’я	Призначення
items/delete-item	Видалення поста
items/get-item	Отримання конкретного поста
items/get-items	Отримання масиву постів за замовчуванням
items/search-item	Функціонал гнучкого пошуку постів
items/update-item	Зміна параметрів поста

Таблиця 3.4 Ендпоінти керування користувачами

Ім’я	Призначення
users/delete-user	Видалення користувача
users/update-user	Модифікація параметрів користувача
users/get-user	Отримання всіх даних про юзера

Таблиця 3.4 Ендпоінти для роботи з зображеннями

Ім’я	Призначення
images/get-upload-url	Функціонал зберігання зображення

У ході розроблення веб-застосунку для підтримки комунікації, було створено ряд компонентів, які описані у таблиці 3.5. Також було додатково розроблено менш універсальні компоненти для реалізації функціоналу на сторінці постів.

Таблиця 3.5 - Розроблені компоненти

Назва компонента	Призначення
advanced-filter.tsx	Модальне вікно розширеного фільтру пошуку
comment-form.tsx	Форма створення коментаря під постом
delete-account-modal.tsx	Модальне вікно підтвердження видалення аккаунта
header.tsx	Хедер веб-застосунку
map-circle.tsx	Компонент карти
post-image-input.tsx	Форма завантаження зображень
search-bar.tsx	Форма гнучкого пошуку постів

Для зберігання даних було використано NoSQL базу даних MongoDB. Створена база даних містить дві колекції: *items* та *users*. Колекція *items* слугує

для збереження та роботи з об’єктами постів, їхньою геолокацією, а також коментарями. Для збереження інформації про користувачів було створено колекцію *users*. Структуру бази даних та зв’язки між колекціями показано на рисунку 3.3.

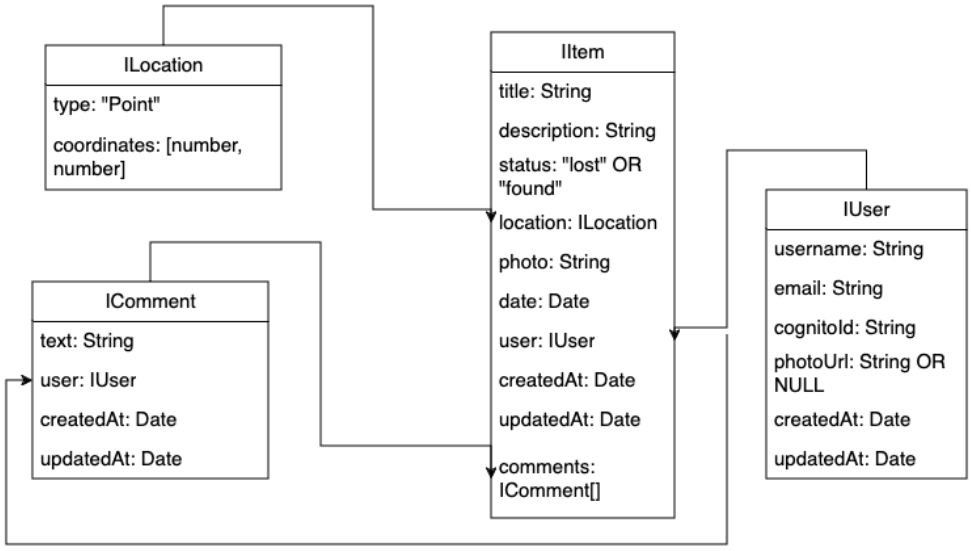

Рисунок 3.3 – Схема бази даних веб-сервісу

Зареєструватися можна за використанням електронної пошти, з обов’язковою умовою - підтвердити кодом вказану в реєстрації пошту. Для реєстрації із електронною поштою необхідно ввести своє ім’я, електронну пошту та придумати пароль, що складається із мінімум 8 символів. За бажанням, пізніше можна змінити ім’я чи електронну пошту.

При реєстрації та авторизації за електронною поштою, веб-сервіс не зберігає та не використовує паролі користувачів. Натомість, використовуються JWT-токени, що містять усю необхідну інформацію про користувача.

3.3 Експлуатація веб-застосунку

Для використання веб-застосунку, кожен користувач зобов’язаний створити власний аккаунт. Це обумовлено тим, що наданий користувачу функціонал з можливістю створювати пости, та залишати коментарі вимагає тісної прив’язки до власника даних. Таким чином, користувач відразу

перенаправляється на сторінку для автентифікації чи реєстрації аккаунта (рис.3.4).

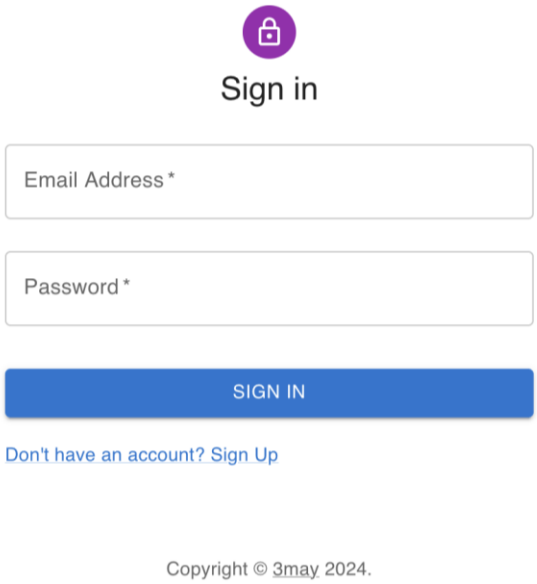

Рисунок 3.4 – Сторінка входу в обліковий запис та реєстрації

Обидві сторінки містять підказки для кожного поля, а також обов’язково валідують інформацію, введену користувачем для створення облікових записів із коректними даними.

Також є сторінка для перегляду інформації про профіль користувача, видалення облікового запису, тощо (рис. 3.5). На цій ж сторінці, користувач має можливість встановити або оновити зображення профілю та ім’я, що відображатиметься під його постами та коментарями.

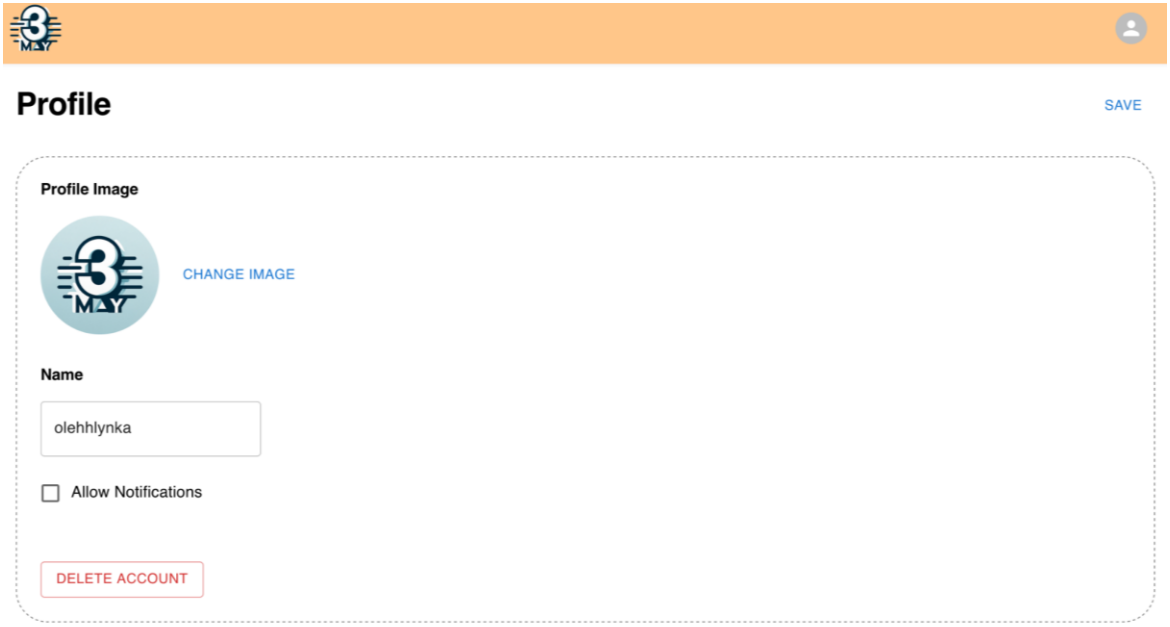

Рисунок 3.5 – Сторінка редагування профілю

На головній сторінці користувач може переглянути перелік постів, що знаходяться близько до його геолокації (рис. 3.6). Користувач за замовчуванням отримує пости в радіусі 5 км від його місцезнаходження. Якщо потрібно проводити пошук в більш віддаленому місці, надається можливість налаштувати як точку відліку, так і задати радіус пошуку.

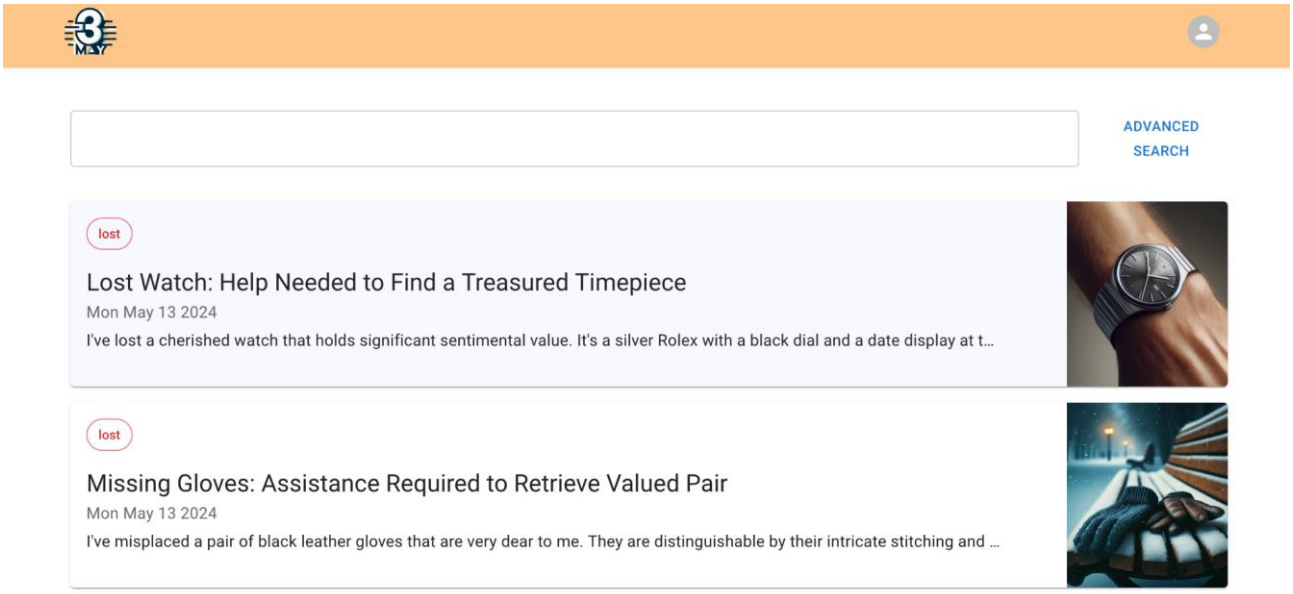

Рисунок 3.6 – Головна сторінка користувача

Після обрання посту користувач перенаправляється на сторінку посту (рис. 3.7), де можна детально ознайомитись з описом речі, переглянути коментарі, прикріплене фото та карту з точними координатами місця, де було загублено чи знайдено річ.

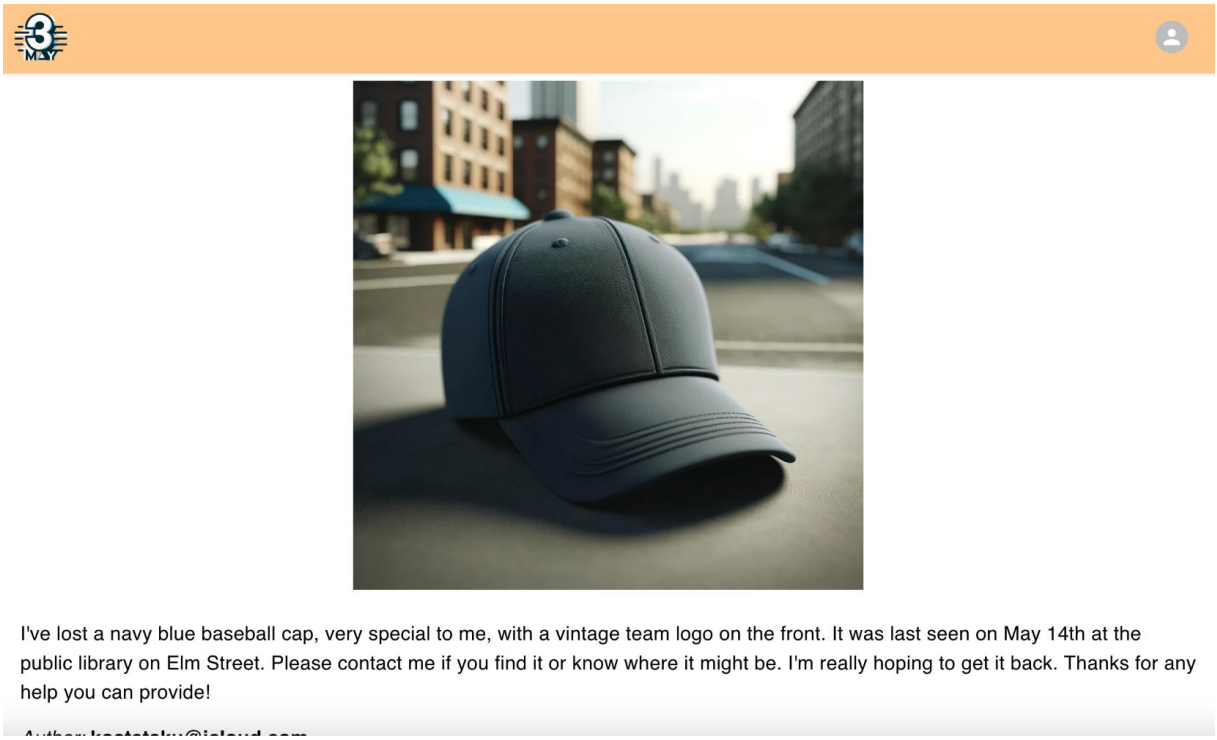

Рисунок 3.7 – Сторінка з окремим постом

Також користувач може переглянути список коментарів посту та додати свій коментар (рис 3.8). Якщо власник поста надав дозвіл на отримання сповіщень про нові коментарі під його постами, кожного разу, коли під постом залишатимуть коментар - власнику поста приходитиме електронний лист з текстом коментаря на адресу, що була вказана при реєстрації профілю.

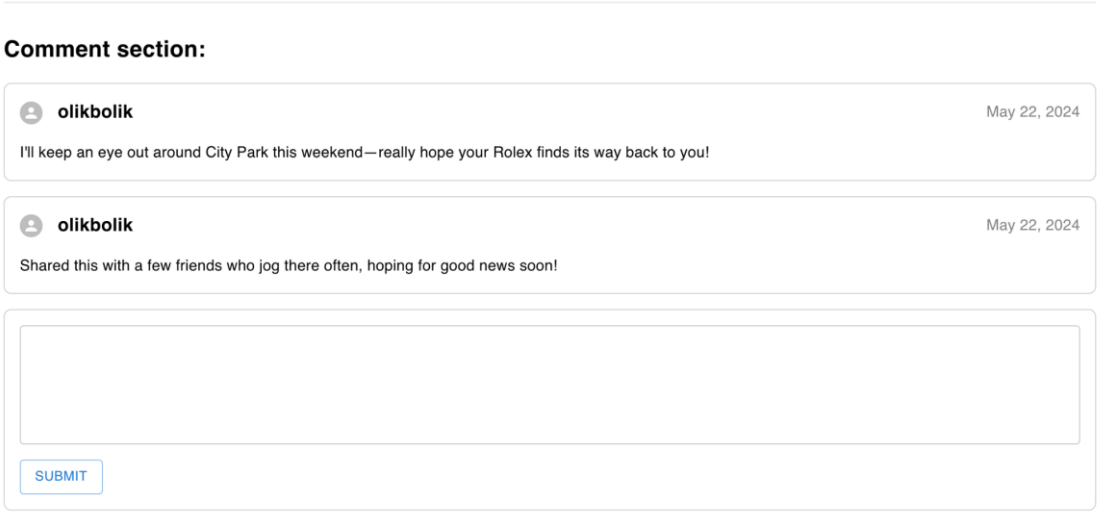

Рисунок 3.8 – Секція коментарів з формою створення нового коментаря

Для створення посту користувачу необхідно вказати заголовок, опис речі, додати фото речі, та статус разом з геолокацією (рис. 3.9). Власник посту може редагувати свій пост, власні коментарі а також видаляти коментарі інших користувачів.

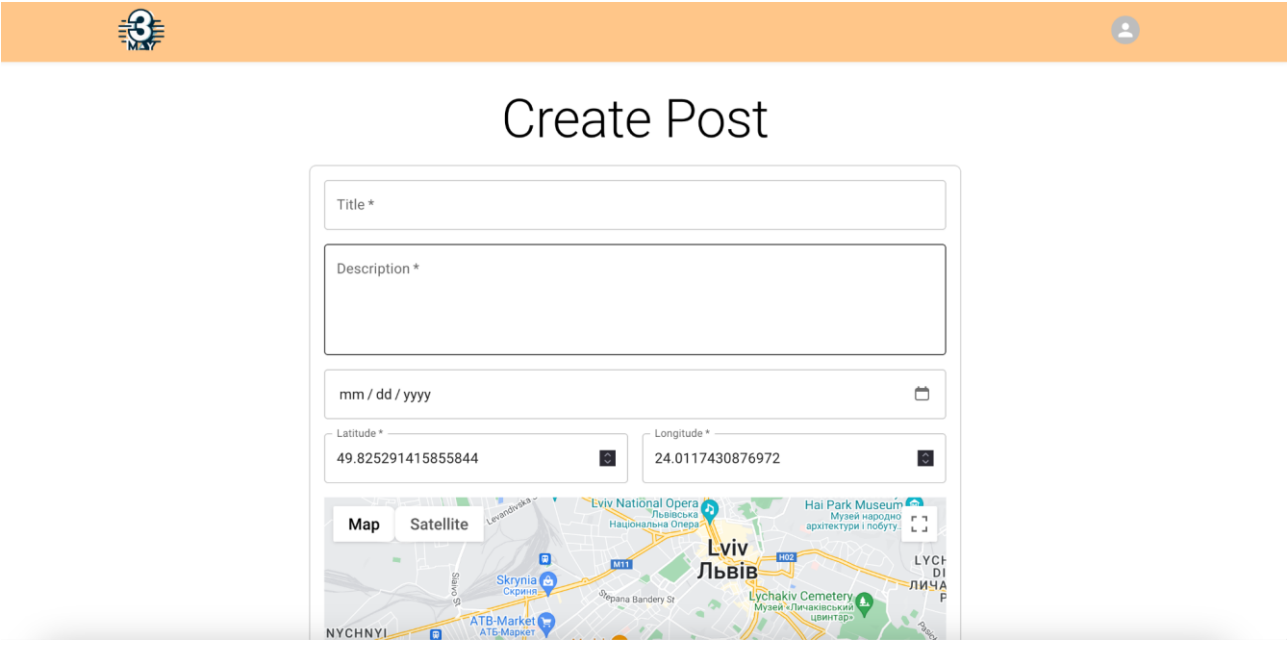

Рисунок 3.9 – Сторінка створення поста

Якщо користувач вирішує, що хоче завершити роботу з застосунком, з метою збереження особистих даних йому надається можливість вийти за профілю, вибравши пункт «Log Out» з меню дій в заголовку (рис. 3.10).

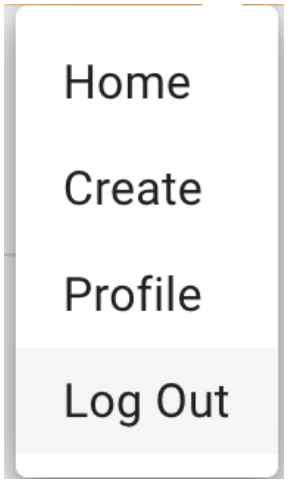

Рисунок 3.10 – Меню дій користувача

ВИСНОВОК

Отже, у результаті виконання кваліфікаційної роботи бакалавра було розроблено веб-застосунок з підтримки комунікації для пошуку втрачених речей у громадських закладах, а саме було здійснено:

- аналіз існуючих цифрових платформ, що використовуються для організації комунікації щодо загублених речей у громадських місцях та їх пошуку;
- аналіз сучасних технологій та методів розробки веб-застосунків;
- спроектовано та реалізовано зручний веб-застосунок підтримки комунікації, пристосований до особливих потреб тих, хто втратив чи знайшов речі у громадських закладах.

Зокрема було розглянуто основні компоненти веб-сервісів та механізми їх взаємодії, основні протоколи, що стосуються веб-сервісів.

Також наведено основні переваги та недоліки веб-сервісів та проаналізовано приклади використання веб-сервісів у різних сферах людської діяльності.

У роботі розглянуто найпопулярніші веб-сервіси підтримки комунікації та спрощення пошуку втрачених речей. Проведено порівняльний аналіз функціональності, що вони пропонують та їхніх основних характеристик.

У другому розділі розглянуто такі архітектурні моделі веб-сервісів як сервісно-орієнтована архітектура, REST-архітектура, монолітна архітектура, мікросервісна архітектура та їхні характеристики, випадки застосування.

Окрім цього, на основі проведеного дослідження та аналізу було реалізовано безпосередньо сам веб-застосунок на основі клієнт-серверної архітектури. Для цього були використані такі технології, як MaterialUI, TypeScript, Node.js, React, MongoDB, AWS S3, AWS Lambda, AWS Cognito, AWS SQS, AWS SES, Serverless Framework та Visual Studio Code.

У подальшому до веб-сервісу можна додати більше різноманітних сповіщень, підтримку аналізу та пошуку постів штучним інтелектом, а також приватні чати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Протокол HTTP/2: що це, переваги та як ним користуватися | Блог VPS.ua. *Блог хостинговой компании VPS.ua*. URL: https://vps.ua/blog/ukr/protocol-http-2-benefits/ (дата звернення: 07.05.2024).
2. Adams D. Learn TypeScript – The Ultimate Beginners Guide. *freeCodeCamp.org*. URL: https://www.freecodecamp.org/news/learn-typescript-beginners-guide/ (дата звернення: 11.05.2024).
3. Engineering G. A Complete Guide to S3 File Upload using pre-signed POST URLs. *Medium*. URL: https://medium.com/@Games24x7Tech/a-complete-guide-to-s3-file-upload-using-pre-signed-post-urls-9cb2d6cfc0ab (дата звернення: 24.05.2024).
4. Get Started | Geocoding API | Google for Developers. *Google for Developers*. URL: https://developers.google.com/maps/documentation/geocoding/start (дата звернення: 20.05.2024).
5. Introduction to Visual Studio - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/introduction-to-visual-studio/ (дата звернення: 04.05.2024).
6. MongoDB Indexes | A Complete Guide and Tutorial | Studio 3T. *Studio 3T*. URL: https://studio3t.com/knowledge-base/articles/mongodb-index-strategy/ (дата звернення: 18.05.2024).
7. Programming TypeScript: Making Your JavaScript Applications Scale. O'Reilly Media, 2019. 322 с.
8. Amazon Web Services, Inc. Serverless application model (developer guide) [Текст] / Amazon Web Services, Inc. – Режим доступу: https://docs.aws.amazon.com/pdfs/serverless/latest/devguide/serverless-core.pdf – Назва з екрана.
9. Amazon Web Services, Inc. Amazon SQS developer guide [Текст] / Amazon Web Services, Inc. – Режим доступу: https://s3.cn-north-1.amazonaws.com.cn/aws-dam-prod/china/pdf/sqs-dg.pdf – Назва з екрана.

10. Amazon Web Services, Inc. Amazon S3 developer guide [Текст] / Amazon Web Services, Inc. – Режим доступу: http://awsdocs.s3.amazonaws.com/S3/20060301/s3-dg-20060301.pdf – Назва з екрана.
11. What Is NoSQL? NoSQL Databases Explained. *MongoDB*. URL: https://www.mongodb.com/resources/basics/databases/nosql-explained (дата звернення: 11.05.2024).
12. What is serverless?. *Red Hat - We make open source technologies for the enterprise*. URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless (дата звернення: 11.05.2024).
13. Amazon Web Services, Inc. AWS overview [Текст] / Amazon Web Services, Inc. – Режим доступу: https://media.amazonwebservices.com/AWS_Overview.pdf – Назва з екрана.
14. iLost. Support center [Електронний ресурс] / iLost. – Режим доступу: https://support.ilost.co/hc/en-us?utm_campaign=b2c_help_menu&utm_medium=link&utm_source=my_ilost – Назва з екрана.
15. An ultimate guide to web development in Python | BrowserStack. *BrowserStack*. URL: https://www.browserstack.com/guide/web-development-in-python-guide (дата звернення: 14.05.2024).
16. Beznos M. Microservices vs monolithic architecture: How each of them can impact your business?. *Software Development Company - N-iX*. URL: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ (дата звернення: 20.05.2024).
17. Chollet P. Getting started with AWS serverless - Authentication. *DEV Community*. URL: https://dev.to/slsbytheodo/learn-serverless-on-aws-authentication-with-cognito-19bo (дата звернення: 11.05.2024).
18. Cook F. Node. js Essentials. Packt Publishing, Limited, 2015.
19. Integrate a REST API with an Amazon Cognito user pool - Amazon API Gateway. URL:

https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-enable-cognito-user-pool.html (дата звернення: 07.05.2024).

20. Janvi thakkar. Implementing Serverless API with Amazon API Gateway, Lambda Function, and Amazon Cognito User Pool. *Medium*. URL: https://medium.com/@janvithakkar.583/implementing-serverless-api-with-amazon-api-gateway-lambda-function-and-amazon-cognito-user-pool-92ff44cc4949 (дата звернення: 18.05.2024).
21. JWT.IO - JSON Web Tokens Introduction. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction (дата звернення: 24.05.2024).
22. Lyngenda D. Single Page Application Vs. Progressive Web App: A Comparison. *Microverse â Where talent connects with global opportunities*. URL: https://www.microverse.org/blog/single-page-application-vs-progressive-web-apps-a-comparison (дата звернення: 11.05.2024).
23. Monolithic vs Microservices - Difference Between Software Development Architectures- AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/ (дата звернення: 20.05.2024).
24. Overview - Material UI. *MUI: The React component library you always wanted*. URL: https://mui.com/material-ui/getting-started/ (дата звернення: 21.05.2024).
25. Serverless Framework - AWS Lambda Events - Cognito User Pool. *Serverless: Zero-Friction Serverless Apps On AWS Lambda & Beyond.* URL: https://www.serverless.com/framework/docs-providers-aws-events-cognito-user-pool (дата звернення: 18.05.2024).
26. Shikha R. Understanding Node.js Ecosystem and Tooling. *Medium*. URL: https://medium.com/@shikha.ritu17/understanding-node-js-ecosystem-and-tooling-fe7466d686c9 (дата звернення: 07.05.2024).
27. Stefanov S. React : up and Running: Building Web Applications. O'Reilly Media, Incorporated, 2021. 222 с.

28. The Internet Lost and Found. *LostAndFound.com: The Internet Lost and Found*. URL: https://lostandfound.com/aboutusnew (дата звернення: 09.05.2024).
29. The Most Demanded Frontend Frameworks in 2023. *Devjobsscanner*. URL: https://www.devjobsscanner.com/blog/the-most-demanded-frontend-frameworks/ (дата звернення: 07.05.2024).
30. Types of Web Development for Beginners. *Maryville University Online*. URL: https://online.maryville.edu/online-bachelors-degrees/computer-science/careers/types-of-web-development/ (дата звернення: 04.05.2024).
31. Web Application Authentication: How It Works and How to Implement It - Authgear. *Simplified Identity & Access Management, Built for Dev with Security - Authgear*. URL: https://www.authgear.com/post/web-application-authentication-guide (дата звернення: 24.05.2024).
32. What is Amazon Cognito? - Amazon Cognito. URL: https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html (дата звернення: 23.05.2024).
33. What is API-first? The API-first Approach Explained | Postman. *Postman API Platform*. URL: https://www.postman.com/api-first/ (дата звернення: 10.05.2024).
34. What is RESTful API? - RESTful API Explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/restful-api/ (дата звернення: 10.05.2024).
35. A short history of the Web. *CERN*. URL: https://home.cern/science/computing/birth-web/short-history-web (дата звернення: 02.05.2024).
36. Columbus L. 10 Charts That Will Change Your Perspective On Artificial Intelligence's Growth. *Forbes*. URL: https://www.forbes.com/sites/louiscolumbus/2018/01/12/10-charts-that-will-change-your-perspective-on-artificial-intelligences-growth/?sh=5ff264214758 (дата звернення: 02.05.2024).

37. Flanagan D. JavaScript: The Definitive Guide. O'Reilly Media, Inc., 2006. 1018 с.
38. Shontell A. FLASHBACK: This Is What The First-Ever Website Looked Like. *Business Insider*. URL: https://www.businessinsider.com/flashback-this-is-what-the-first-website-ever-looked-like-2011-6 (дата звернення: 02.05.2024).
39. The Influence of Social Media on Web Development Trends. *Custom software development company | MoldStud.com*. URL: https://moldstud.com/articles/p-the-influence-of-social-media-on-web-development-trends (дата звернення: 07.05.2024).
40. What is an api: how does it work and why do you need it. *Lvivity*. URL: https://lvivity.com/what-is-an-api-and-how-does-it-work (дата звернення: 10.05.2024).
41. Serverless services on AWS. *Itemis.com*. URL: https://blogs.itemis.com/en/serverless-services-on-aws (дата звернення: 11.05.2024)

ДОДАТКИ

ДОДАТОК А

Конфігурація інфраструктури застосунку в Serverless Framework

```
service: threemay
useDotenv: true

plugins:
 - serverless-esbuild

resources: ${file(./resources.yaml)}

provider:
 name: aws
 runtime: nodejs20.x
 region: eu-central-1
 stage: ${opt:stage, 'dev'}
 memorySize: 256
 timeout: 30
 environment:
   USER_POOL_CLIENT_ID:
     Ref: UserPoolClient
   USER_POOL_ID:
     Ref: UserPool
   MONGO_URL: ${env:MONGO_URL}
   STAGE: ${self:provider.stage}
   IMAGES_BUCKET_NAME:
     Ref: ImagesBucket
   SES_EMAIL: ${env:SES_EMAIL}
   NOTIFICATIONS_SQS_URL:
     Ref: EmailNotificationQueue
 iam:
   role:
     statements:
       - Effect: Allow
         Action:
```

```
            - cognito-idp:*
            - s3:*
            - ses:*
            - sqs:*
          Resource: '*'
 httpApi:
   cors: true
   authorizers:
     ApiGatewayAuthorizer:
       type: jwt
       identitySource: $request.header.Authorization
       issuerUrl:
         Fn::Sub:
           - 'https://cognito-
idp.${aws:region}.amazonaws.com/${UserPoolId}'
           - UserPoolId: !Ref UserPool
       audience:
         - Ref: UserPoolClient

custom:
 servicePrefix: ${self:service}-${self:provider.stage}
 esbuild:
   bundle: true
   minify: true
   packager: pnpm
```

ДОДАТОК Б

React-компонент карти

```
import {
 forwardRef,
 useContext,
 useEffect,
 useImperativeHandle,
 useRef,
} from 'react';

import type { Ref } from 'react';
import { GoogleMapsContext, latLngEquals } from '@vis.gl/react-
google-maps';

type CircleEventProps = {
 onClick?: (e: google.maps.MapMouseEvent) => void;
 onDrag?: (e: google.maps.MapMouseEvent) => void;
 onDragStart?: (e: google.maps.MapMouseEvent) => void;
 onDragEnd?: (e: google.maps.MapMouseEvent) => void;
 onMouseOver?: (e: google.maps.MapMouseEvent) => void;
 onMouseOut?: (e: google.maps.MapMouseEvent) => void;
 onRadiusChanged?: (r:
ReturnType<google.maps.Circle['getRadius']>) => void;
 onCenterChanged?: (p:
ReturnType<google.maps.Circle['getCenter']>) => void;
};

export type CircleProps = google.maps.CircleOptions &
CircleEventProps;

export type CircleRef = Ref<google.maps.Circle | null>;

function useCircle(props: CircleProps) {
 const {
   onClick,
```

```
    onDrag,
    onDragStart,
    onDragEnd,
    onMouseOver,
    onMouseOut,
    onRadiusChanged,
    onCenterChanged,
    radius,
    center,
    ...circleOptions
  } = props;
  const callbacks = useRef<Record<string, (e: unknown) =>
void>>({});
  Object.assign(callbacks.current, {
    onClick,
    onDrag,
    onDragStart,
    onDragEnd,
    onMouseOver,
    onMouseOut,
    onRadiusChanged,
    onCenterChanged,
  });

  const circle = useRef(new google.maps.Circle()).current;
  circle.setOptions(circleOptions);

  useEffect(() => {
    if (!center) return;
    if (!latLngEquals(center, circle.getCenter()))
circle.setCenter(center);
  }, [center]);

  useEffect(() => {
    if (radius === undefined || radius === null) return;
    if (radius !== circle.getRadius()) circle.setRadius(radius);
```

```
 }, [radius]);

 const map = useContext(GoogleMapsContext)?.map;

 useEffect(() => {
   if (!map) {
     if (map === undefined)
       console.error('<Circle> has to be inside a Map
component.');

     return;
   }

   circle.setMap(map);

   return () => {
     circle.setMap(null);
   };
 }, [map]);

 useEffect(() => {
   if (!circle) return;

   const gme = google.maps.event;
   [
     ['click', 'onClick'],
     ['drag', 'onDrag'],
     ['dragstart', 'onDragStart'],
     ['dragend', 'onDragEnd'],
     ['mouseover', 'onMouseOver'],
     ['mouseout', 'onMouseOut'],
   ].forEach(([eventName, eventCallback]) => {
     gme.addListener(circle, eventName, (e:
google.maps.MapMouseEvent) => {
       const callback = callbacks.current[eventCallback];
       if (callback) callback(e);
```

```
      });
    });
    gme.addListener(circle, 'radius_changed', () => {
      const newRadius = circle.getRadius();
      callbacks.current.onRadiusChanged?.(newRadius);
    });
    gme.addListener(circle, 'center_changed', () => {
      const newCenter = circle.getCenter();
      callbacks.current.onCenterChanged?.(newCenter);
    });

    return () => {
      gme.clearInstanceListeners(circle);
    };
  }, [circle]);

  return circle;
}

export const MapCircle = forwardRef((props: CircleProps, ref:
CircleRef) => {
  const circle = useCircle(props);

  useImperativeHandle(ref, () => circle);

  return null;
});
```

ДОДАТОК В

React-компонент для завантаження зображень

```
import React from 'react';
import { useDropzone } from 'react-dropzone';
import Button from '@mui/material/Button';
import Box from '@mui/material/Box';
import Typography from '@mui/material/Typography';

interface IProps {
 filePath: string;
 setFilePath: (val: string) => void;
 token: string | null;
 file: File | null;
 setFile: (val: File | null) => void;
 isSubmitting?: boolean
}
const PostImageInput: React.FC<IProps> = ({ setFile, file,
isSubmitting }) => {

 const setFileImageHandler = React.useCallback((file: File) => {
   try {
     const url = URL.createObjectURL(file);
     const img = new Image();

     img.src = url;

     setFile(file);
   } catch (e) {
     console.error('There is image loading error');
   }
 }, []);

 const imageDropHandler = React.useCallback(
   (acceptedFiles: Array<File>) => {
     setFileImageHandler(acceptedFiles[0]);
```

```
  },
  [setFileImageHandler],
);

const { getInputProps, getRootProps } = useDropzone({
  onDrop: imageDropHandler,
  accept: {
    'image/jpeg': ['.jpg'],
    'image/png': ['.png'],
  },
  multiple: false,
  disabled: isSubmitting
});

const imageReset = React.useCallback(() => {
  setFile(null);
}, []);

return (
  <>
    {file ? (
      <Box sx={{
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'space-between',
        padding: '0.5rem',
        border: '1px solid #ccc',
        borderRadius: '5px',
        marginBottom: '1rem',
      }}>
        <div>
          <Typography sx={{
            fontSize: '1rem',
            fontWeight: 'bold',
            color: "black",
            marginBottom: '0.5rem',
```

```
            }}>{file?.name}</Typography>
            <Typography sx={{
              fontSize: '0.8rem',
              color: "gray",
            }}>
              {Math.round(file?.size * 0.001 * 10) / 10} KB
            </Typography>
          </div>
          <Button
            onClick={imageReset} variant="outlined" color={"error"}
          >
            Remove
          </Button>
        </Box>
      ) : (
        <Box {...getRootProps()} sx={{
          display: 'flex',
          alignItems: 'center',
          justifyContent: 'center',
          padding: '1rem',
          border: '1px dashed #ccc',
          borderRadius: '5px',
          marginBottom: '1rem',
          color: "gray",
          cursor: "pointer",
        }}>
          <input {...getInputProps()} />
          <p>Drag 'n' drop an image here, or click to select a
file</p>
        </Box>
      )}
    </>
  );
};

export default PostImageInput;
```

ДОДАТОК Г

Тіло лямбда-функції створення користувача

```
const client = new CognitoIdentityProviderClient({});

const main = getHandler(signUpContract, { ajv })(async (event) =>
{
 const { password, email, login } = event.body;

 await client.send(
   new SignUpCommand({
     ClientId: process.env.USER_POOL_CLIENT_ID!,
     Username: email,
     Password: password,
     UserAttributes: [{ Name: 'email', Value: email }],
   }),
 );

 return httpResponse({ login });
});

export const handler = middy(main)
 .use(doNotWaitForEmptyEventLoop())
 .use(cors())
 .use(dbConnection())
 .use(errorHandlingMiddleware());
```

ДОДАТОК Д

Тіло лямбда-функції для генерації URL-адрес для завантаження зображень на S3

```
const { IMAGES_BUCKET_NAME } = process.env;
const SUPPORTED_EXTENSIONS = /\.jpg|\.png|\.jpeg/;

const client = new S3Client();

const main = getHandler(getImageUploadUrlContract, {
 ajv,
 validateOutput: false,
})(async (event, context) => {
 const { db } = context as DbConnectionContext;
 const { fileName } = event.pathParameters;
 const { profile } = event.queryStringParameters;
 const { sub: cognitoId } =
event.requestContext.authorizer.jwt.claims;

 const user = await db
   .collection<UserType>(USERS_COLLECTION)
   .findOne({ cognitoId });

 if (!user) {
   throw new Error('User not found', { cause:
HttpStatusCodes.FORBIDDEN });
 }

 const extension =
fileName.toLowerCase().match(SUPPORTED_EXTENSIONS)?.[0];

 if (!extension) {
   throw new Error('Unsupported file extension', {
     cause: HttpStatusCodes.BAD_REQUEST,
   });
 }
```

```
 const key = `${user._id}/${profile ? 'profile/' :
''}${uuidv4()}${extension}`;
 const imageUrl =
`https://${IMAGES_BUCKET_NAME}.s3.amazonaws.com/${key}`;

 const presignedPost = await createPresignedPost(client, {
   Bucket: IMAGES_BUCKET_NAME!,
   Key: key,
   Conditions: [
     { bucket: IMAGES_BUCKET_NAME! },
     ['content-length-range', 1024, 5242880],
   ],
   Fields: { key },
   Expires: 300,
 });

 return httpResponse({ url: imageUrl, presignedPost });
});

export const handler = middy(main)
 .use(doNotWaitForEmptyEventLoop())
 .use(cors())
 .use(dbConnection())
 .use(errorHandlingMiddleware());
```

ДОДАТОК Е

Визначення контракту для ендпоінту створення постів

```
export const postNewItemContract = new ApiGatewayContract({
 id: 'postNewItem',
 path: '/items/{status}',
 method: 'POST',
 integrationType: 'restApi',
 authorizerType: 'cognito',
 requestContextSchema: requestContextSchemaCustom,
 bodySchema: {
   type: 'object',
   properties: {
     title: { type: 'string' },
     description: { type: 'string' },
     photo: { type: 'string', minLength: 10 },
     lng: { type: 'number', minimum: -180, maximum: 180 },
     lat: { type: 'number', minimum: -90, maximum: 90 },
     date: { type: 'string' },
     tags: { type: 'array', items: { type: 'string' } },
   },
   additionalProperties: false,
   required: ['title', 'description', 'date', 'lng', 'lat'],
 } as const,
 pathParametersSchema: {
   type: 'object',
   properties: {
     status: { type: 'string', enum: ['lost', 'found'] },
   },
   additionalProperties: false,
   required: ['status'],
 } as const,
 outputSchemas: {
   [HttpStatusCodes.OK]: {
     type: 'object',
     properties: {
```

```
        id: { type: 'string' },
      },
      required: ['id'],
      additionalProperties: false,
    } as const,
    [HttpStatusCodes.BAD_GATEWAY]: errorSchema,
  } as const,
});
```

ДОДАТОК Є

Проміжне програмне забезпечення для підключення до бази даних

```
let cachedClient: MongoClient | null = null;
let cachedDb: Db | null = null;

export interface DbConnectionContext extends Context {
 mongoClient: MongoClient;
 db: Db;
}

const dbConnection = (): middy.MiddlewareObj<
 unknown,
 unknown,
 unknown,
 DbConnectionContext
> => {
 const before: middy.MiddlewareFn<
   unknown,
   unknown,
   unknown,
   DbConnectionContext
 > = async (handler) => {
   if (!cachedClient || !cachedDb) {
     cachedClient = await
MongoClient.connect(process.env.MONGO_URL!);

     cachedDb = cachedClient.db(process.env.DB_NAME!);
   }
   Object.assign(handler.context, { mongoClient: cachedClient, db:
cachedDb });
 };
 return { before };
};

export { dbConnection };
```