Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ПОБУДОВА БАГАТОРІВНЕВОГО ВЕБ-ЗАСТОСУВАННЯ НА ПЛАТФОРМІ AMAZON WEB SERVICES (AWS)

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення»

Керівник курсової роботи

Кандидат технічних наук

Черкасов Д. І.

Виконала студентка 4 курсу

Гаращук Д. С.

Київ 2022

Тема: Побудова багаторівневого веб-застосування на платформі Amazon Web Services(AWS)

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	Жовтень 2022р.	
2.	Пошук тематичної літератури	Березень-квітень 2022р.	
3.	Ознайомлення з літературою	Березень-квітень 2022р.	
4.	Ознайомлення та аналіз існуючих рішень	Березень-квітень 2022р.	
5.	Планування веб-сайту	Березень-квітень 2022р.	
10.	Структурна розробка власного рішення	Травень 2022р.	
12.	Додавання стилів до сторінок	Травень 2022р.	
13.	Написання текстової частини	Травень 2022р.	
14.	Захист роботи	Травень 2022р.	

Студентка Гаращук Д.С.

Керівник Черкасов Д. І.

“____” _____

Перелік умовних позначень

1. P2P – (Peer-to-peer) - модель взаємодії рівноправних систем, де кожна може виступати і клієнтом, і сервером.
2. ISP – (Internet Services Provider), компанія, що надає доступ до Інтернету.
3. API – (Application Programming Interface) - сукупність процедур та функцій для взаємодії кількох програм.
4. Фреймворк – шаблон, який дозволяє на своїй основі реалізовувати власний програмний код
5. Cloud-native – технології, які дають можливість організаціям створювати і запускати програми у сучасних, динамічних середовищах, таких як публічні, приватні та гібридні хмари.
6. AWS – (Amazon Web Services) - хмарна платформа для обробки великої кількості даних.

ВСТУП

Сьогодення диктує нам свої правила, настав час, коли інформація може коштувати комусь життя. І саме за допомогою інтернету вона може швидко поширюватися. Актуальність новинних сервісів сильно зросла під час воєнного стану, до бурхливого інформаційного поля прикуті очі мільйонів українців.

Проаналізувавши доступні новинні джерела та потреби людей, які зараз знаходяться в Україні, було вирішено розробити сайт новин. Новий новинний сервіс має забезпечувати безперебійну роботу без простоїв, масштабованість до мільйонної кількості користувачів, безпеку даних та офлайн доступ, адже користувачі у гарячих точках можуть мати проблеми з інтернетом.

Усі ці потреби забезпечує платформа AWS, за допомогою більше ніж 200 повнофункціональних сервісів відбувається швидка та гнучка розробка. AWS є економічно вигідною, що теж наразі дуже важливо. Зберігання файлів на AWS S3 коштує копійки, завдяки AWS Amplify застосунок реагує на кількість запитів і автоматично масштабується за потреби, за невикористовувані частини функціоналу AWS не братиме кошти.

Метою роботи є створення новинного сервісу на платформі AWS.

Завдання: розробити потужний новинний сервіс, де користувачі зможуть публікувати свої новини, переглядати стрічку новин, відфільтровувати новини за категоріями та геолокацією.

Об’єктом дослідження є створення новинного веб-застосунку.

Предметом дослідження є огляд існуючих архітектур мережевих застосувань, платформи AWS та її сервісів.

Веб-застосунок розроблений на мові Javascript за допомогою фреймворків React та AWS Amplify. Для збереження даних використано

хмарне сховище AWS S3 та базу даних DynamoDB. За допомогою Amazon Cognito створено систему контролю доступу користувачів. AppSync став посередником між усіма сервісами AWS, також за його допомогою розроблено GraphQL API. Середовище розробки – Visual Studio Code.

У навчальних цілях було прочитано багато офіційної документації AWS та пройдено курс «AWS AppSync & Amplify with React & GraphQL».

Курсова робота складається зі вступу, двох розділів та їх підрозділів, висновків, списку використаної літератури.

РОЗДІЛ 1: Огляд існуючих рішень

1.1. Архітектура мережевих застосувань

Застосування для доступу до якого використовується мережа передачі даних називається мережевим застосуванням. Мережева програма — це будь-яка програма, що працює на одному хості, і забезпечує зв’язок з іншою програмою, що працює на іншому хості. Ці програми комунікують одна з одною, підключаючись до мережі. Наприклад, коли користувач відвідує google.com, його браузер виступає у ролі мережевої програми, яка використовує інтернет для спілкування з іншою мережевою програмою, що працює на комп’ютері Google. Мережеві застосування є розподіленими та багатокомпонентними. Кількість користувачів мережевого застосування нелімітована.

Роботу, яку виконує будь-яке мережеве застосування, можна розділити на чотири загальні функції. Це зберігання даних, логіка доступу до даних та їх обробка, бізнес-логіка та взаємодія з користувачем. Існує багато способів, за допомогою яких ці чотири функції можуть бути розподілені між клієнтськими комп’ютерами та серверами в мережі.

Сьогодні виділяють чотири типи комунікаційних моделей мережевих застосувань:

- одноріvнева модель
- модель симетричної взаємодії (peer-to-peer)
- модель “клієнт-сервер” (client-server)
- багаторівнева модель (є результатом розвитку клієнт-серверної моделі)

1.2. Види серверів та клієнтів

Існує велика кількість видів серверів та клієнтів, які можуть бути частиною мережі, з часом різниця між ними тільки ускладняється.

Можна виділити чотири типи комп'ютерів, які зазвичай використовуються як сервери:

1) Мейнфрейм – універсальний високопродуктивний сервер, який здатний виконувати інтенсивні обчислювальні роботи, підтримувати велику кількість користувачів одночасно і зберігати величезні обсяги даних.
2) Мікрокомп'ютер – це звичайний персональний комп'ютер. Також до мікрокомп'ютерів, які використовуються як сервери, відносять комп'ютери мікро розмірів. Наприклад, «Raspberry Pie», може використовуватися як домашній сервер чи керуючий комп'ютер невеликого некомерційного проекту.
3) Кластер – це група серверів (часто мікрокомп'ютерів), пов'язаних в одну систему, щоб вони діяли як один сервер. Кластери масштабовані, тому що в кластер завжди можна додавати додаткові комп'ютери. Кластери складніші, ніж окремі сервери, оскільки запити необхідно швидко розподіляти між окремими комп'ютерами та координувати розподіл. Якщо один комп'ютер виходить з ладу, кластер просто обходить його, тому що кожен комп'ютер працює окремо.

   Існують такі типи кластерів:

   а) Відмовостійкі (high-availability) – декілька серверів об'єднують з метою дублювання один одного. При відмові одного сервера його задачі переймають інші комп'ютери. Завдяки апаратній надмірності сервіси продовжують працювати без простоїв та зупинок.

   Забезпечує захист від збоїв різного характеру, а саме:

i) програмні збої – впливають на критично важливі служби в мережі;

ii) збої датацентру – частіше за все викликані стихійними лихами (повінь, пожежа), призводять до відключень електроенергії, навіть до необоротної втрати обладнання;

iii) системні та апаратні збої – уражують процесори, адаптери, пам'ять та джерела живлення;

b) Балансувальники (load) – у межах одного кластера навантаження рівномірно розподіляється між декількома нодами. Таким чином виходить уникнути ситуацій, коли один комп'ютер працює на межі можливостей, а інші у простої. Підвищується оптимізація обробки запитів. Для розподілу завдань у кластері використовується один або кілька вхідних обчислювальних вузлів. Через них завдання перенаправляються з однієї машини на іншу. Головною ціллю балансування є підвищення продуктивності системи, проте часто використовують і методи, що покращують надійність. Такі платформи називають серверними фермами.

c) Високопродуктивні (high-performance computing cluster) – кластер побудований на основі групи машин з метою підвищення продуктивності у ресурсномістких завданнях. Комп'ютери об'єднані високошвидкісними лініями передач. Єдина система комплексних обчислень автоматично розподіляє та надсилає дані для обробки із загального пулу завдань. Учасники кластеру можуть виконувати паралельні обчислення. Високопродуктивні кластери знайшли застосування в аналітиці, системах штучного інтелекту, збиранні та обробці даних для Big Data.

Загалом, переваги кластера – це забезпечення користувачам більш високої доступності, підвищення продуктивності роботи системи, горизонтальна масштабованість, скорочення часу простою, шляхом

перерозподілу робочого навантаження, уникнення збоїв і їх негативних наслідків.

4) Віртуальний сервер (virtual server) – це одна машина (часто мікрокомп’ютер), яка емулює роботу фізичного сервера. На одному комп’ютері може бути запущено багато віртуальних серверів за допомогою спеціального програмного забезпечення. На одному комп’ютері встановлюється декілька операційних систем. У мережі така машина буде відображатися як кілька окремих серверів. Віртуальні сервери надають незалежний та повний контроль, можуть виконувати як однакові, так і окремі функції. Кожна віртуальна машина має окреме адміністрування, свої процеси, ресурси та конфігурацію, як фізичні сервери. Віртуальний сервер ідеально підходить для невеликих проектів, бо дає можливість уникнути необхідності купувати окреме обладнання для кожного сервера, у той самий час має переваги кластерного сервера.

Існує чотири поширених типів клієнтів:

1) Мікрокомп'ютер та портативний комп’ютер – на сьогодні є найпоширенішим типом клієнта. До мікрокомп’ютерів відносять персональні комп’ютери, до портативних – ноутбуки, планшети, смартфони.

2) Термінал – це монітор під’єднаний до клавіатури. Цей простий пристрій для вводу і виводу, він не має центрального процесора. Будь-яка операція введена через термінал одразу потрапляє на сервер, де і оброблюється.

3) Термінал для транзакцій – термінал призначений для підтримки конкретних бізнес-транзакцій. Прикладом можуть бути банкомати та термінали для розрахунку в супермаркеті.

4) Мережевий комп’ютер – це комп'ютер без жорсткого диску, який не може зберігати інформацію. Також називається тонким клієнтом та бездисковим вузлом. Мережевий комп’ютер має завантажуватися з

центрального сервера при кожному запуску, бо повністю від нього залежить.

1.3. Однорівнева архітектура

Однорівнева архітектура передбачає розміщення як клієнтської, так і серверної частини, у вигляді єдиної системи, на одній машині.

Така система підходить лише для невеликих додатків з низьким трафіком, також підходить для перевірки мережевого застосунка на етапі розробки. Ця модель має низку недоліків, а саме – низьку масштабованість, створює ризик безпеки та доступності. Якщо зловмисник атакує сервер, усі данні будуть втрачені, систему буде неможливо відновити, бо однорівнева архітектура не передбачає створення репліки бази даних. Якщо сервер вийде з ладу, застосунок не зможе оброблювати запити, бо зв’язок з базою даних буде втрачено. У той самий час однорівнева модель забезпечує простоту розгортання, розробки, зручність та швидкість розробки.

1.4. Архітектура симетричної взаємодії

Архітектура симетричної взаємодії передбачає прямий зв’язок між одноранговими вузлами, без проходження даних через спеціалізований сервер у дата центрі. Вузли здатні виконувати будь-які функції застосунку, як клієнтської, так і серверної частини, для інших вузлів мережі. Оскільки архітектура використовує лише користувацькі персональні комп’ютери, які періодично підключаються до мережі, всі учасники архітектури є рівними, тому називаються одноранговими. Оскільки однорангові спілкуються без проходження через виділений сервер, архітектура називається одноранговою.

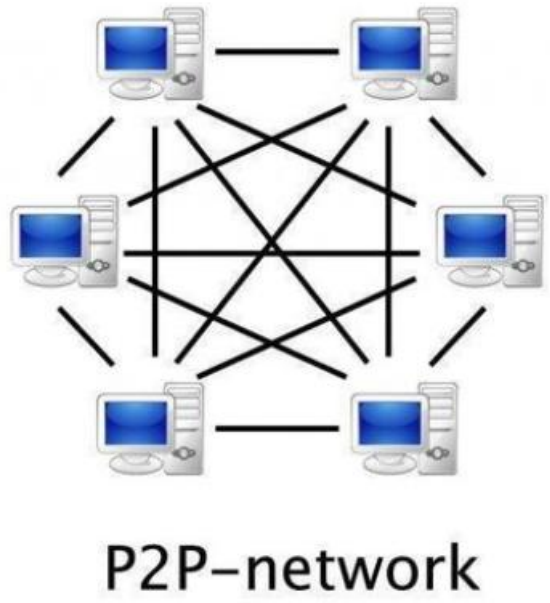

Рисунок 1 – Peer-to-peer архітектура

Модель симетричної взаємодії стала популярною на початку 2000-х років із появою P2P програм обміну файлами, наприклад, BitTorrent. Використовуючи логіку презентації та доступу до даних протоколу Torrent, користувач завантажує файл з пристрою клієнта-джерела, отримуючи частини файлу користувач сам починає віддавати їх іншим клієнтам. Кількість користувачів, які роздають один і той самий файл увесь час збільшується. Таким чином знижується навантаження у мережі, збільшується продуктивність та доступність. Файл роздають усі користувачі, які його завантажили та є під'єднаними до мережі.

Деякі програми мають гібридні архітектури, у них є клієнт-серверна частина та елементи P2P. Дані в моделі P2P можуть знаходитися у будь-якому місці мережі, тому потрібен центральний сервер, який буде допомагати вузлам знайти одне одного. Компанія Spotify використовувала гібридну архітектуру до 2014 року, відмову від використання P2P частини в архітектурі пояснили тим, що компанія мала достатню кількість серверів по всьому світу. Ще одна всесвітньовідома компанія, яка використовує

гібридну архітектуру з 2003 року до сьогодення – це Skype. Архітектура Skype складається з трьох сутностей: звичайні вузли, супер вузли та сервер входу (логіну). Окрім серверу входу, Skype не має інших централізованих серверів. Інформація про користувачів зберігається децентралізовано, однорангові вузли використовуються для голосових дзвінків і текстових повідомлень. Супер вузол виступає ендпоінтом у архітектурі Skype, при чому супер вузлом може стати будь-який пристрій у мережі з публічною IP-адресою та достатніми параметрами системи.

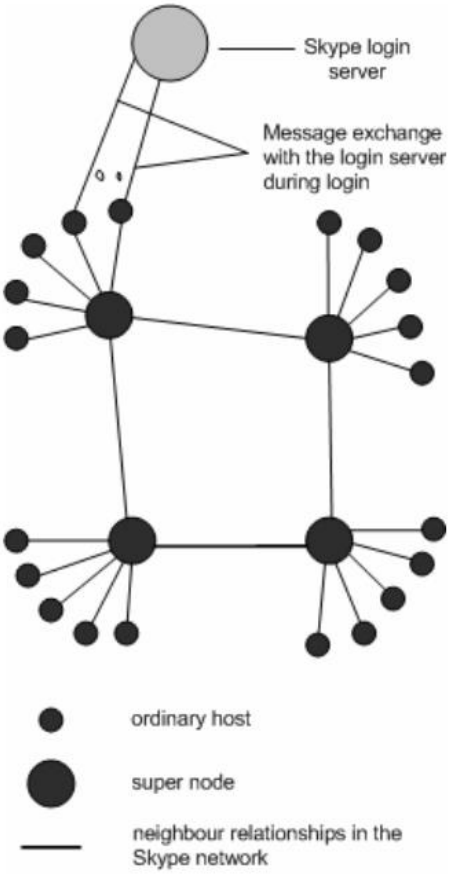

Рисунок 2 – будова мережі Skype

Основні переваги P2P архітектури наступні:

- Відмовостійкість – стійкість до збоїв;
- Самомасштабованість – кожен одноранговий партнер, хоча і створює робоче навантаження, проте також додає сервісну потужність до системи;

- Економічна ефективність – архутектура не потребує дорогої серверної інфраструктури, робота розподіляється між вузлами-учасниками системи;

У той самий час P2P стикається з наступними викликами:

- підтримка користувачів
- обмеження інтернет-провайдерів
- забезпечення безпеки

Майбутнє P2P архітектури напряму залежить від підтримки користувачів, чи будуть вони згодні надавати сховище, обчислювальні ресурси та пропускну здатність програмам.

Модель симетричної взаємодії також іде в розріз з апаратними можливостями ISP. Адже більшість сучасних домашніх інтернет провайдерів надають асиметричну пропускну здатність. Вони розраховані на те, щоб доносити великі потоки трафіку до користувача (низхідний потік), і значно менші від нього (висхідний). Але програми з P2P архітектурою перекладають відповідальність за потік висхідного трафіку від потужних серверів до інтернет провайдерів, чим створюють для них майже катастрофічний стрес.

Безпека мережевого застосунку також знаходиться під питанням через розгалужену та відкриту архітектуру. Не кожен мережевий застосунок може нехтувати безпекою даних користувачів чи іншою конфіденційною інформацією задля переваг P2P архітектури.

1.4. Клієнт-серверна архітектура

Клієнт-серверна архітектура передбачає використання двох фізичних машин, клієнта та сервера, та розподілення між ними функцій зберігання

даних, логіки доступу до даних та їх обробку, бізнес-логіку та користувацький інтерфейс.

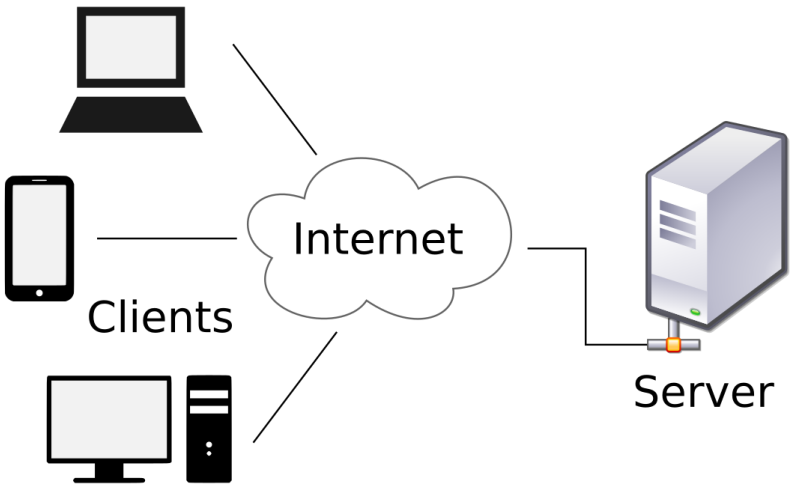

Рисунок 3 – клієнт-серверна архітектура

Серед дволанкових архітектур виділяють два види, в залежності від того, як між ними розподіляються наведені вище функції:

- Тонкий клієнт – вся бізнес-логіка застосунку та управління даними знаходиться на сервері, клієнт відповідає тільки за представлення даних;
- Товстий клієнт – сервер відповідає тільки за управління даними, представлення, логіка та обробка інформації відбувається на клієнті;

Серверів в одній архітектурі може бути багато (кластер серверів), тоді використовується ще одна проміжна ланка – балансувальник навантаження, який відправить запит на менш навантажений сервер. Додаткові сервери використовуються для збільшення доступності інформації та підвищення швидкості обробки запитів.

Клієнт-серверна архітектура має наступні переваги:

- Інклюзивність –мережеве застосування з клієнт-серверною архітектурою є доступним для більшої кількості користувачів, бо його

використання не потребує надпотужної машини та не витрачає багато трафіку на обробку даних;

- Відсутність дублювання коду — основний код зберігається на сервері, клієнт відповідає тільки за відмальовку клієнтського інтерфейсу та валідацію полів;

І мінуси:

- Безпека – існують потенційні проблеми з безпекою, оскільки всі дані знаходяться на центральному сервері;
- Низька доступність – можливі простої, якщо один сервер вийде з ладу, уся система впаде, проте цього можна уникнути, якщо використовувати кластер серверів;

Економічна ефективність клієнт серверної архітектури стоїть під питанням. З обного боку, вигідніше придбати один потужний сервер ніж багато потужних клієнтських пристроїв. З іншого, сервер все ж є дуже дорогим обладнанням, до того ж, доведеться найняти сісадміна, щоб він слідкував за серверною.

1.5. Багаторівнева архітектура

N-рівнева архітектура передбачає використання трьох і більше ланок. Багаторівневав модель наразі є найпоширенішою. Система поділяється на рівні, кожен з яких має змогу спілкуватися лише зі своїми сусідами. Клієнт відповідає за логіку презентації, сервер бази даних відповідає за логіку доступу та зберігання даних, інші функції мережевого застосунку поширюються на інші компоненти, кожен з яких реалізує специфічний набір сервісів.

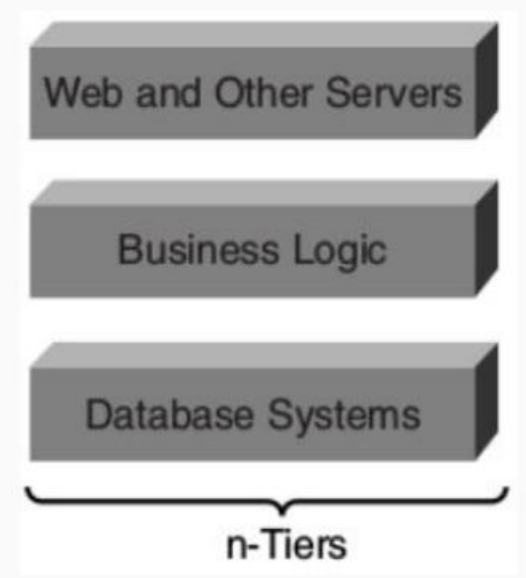

Рисунок 4 – приклад багаторівневої архітектури

Вирізняють таки види серверів за їхньою роллю у системі: веб-сервер, сервер застосунків, сервер баз даних, сервер друку, файловий, поштовий, файловий, термінальний, стрімінг, ігровий сервери, сервер віддаленого доступу, DNS сервер, DHCP сервер.

Багаторівнева архітектура має наступні переваги:

- Висока гнучкість
- Масштабованість
- Безпека – захист можна встановити на кожному рівні
- Продуктивність – завдання розподілені між рівнями
- Популярність – для створення багаторівневої архітектури розроблено багато генераторів шаблонів та написано велику кількість літератури

І мінуси:

- Погана організація коду – при хаотичному та непродуманому додаванні нових рівнів до системи, гнучкість архітектури може зіграти негативну роль;

- Низька швидкість – надлишкові рівні у архітектурі можуть спричинити ситуацію, коли кількість непотрібних операцій починає переважати над корисними, адже архітектура вертикально масштабована, інформація може марно проходити від рівня до рівня;
- Ускладнений дебагінг коду – якщо дані випадково ушкоджуються на одному з рівнів, де вони б не мали оброблюватися, доведеться перевірити всі рівні окремо;

Загалом багаторівнева архітектура стала звичним природнім вибором для багатьох мережевих застосувань. Так склалося, що у компаніях існує поділ на окремі команди, одна відповідальна за фронтенд, інша за бекенд. Розробляючи мережевий застосунок компанії вимушені дотримуватися схеми, яка відтворює організацію комунікації та розподілу обов’язків усередині компанії.

.

РОЗДІЛ 2: Структура розробки власного рішення

2.1. Використаний стек технологій

2.1.1 AppSync

AWS AppSync — це AWS сервіс, посередник між клієнтськими застосуваннями та іншими сервісами AWS. На офіційній схемі (див. Рисунок 5) ми можемо побачити, що всі серверні елементи Amazon (Amazon Aurora, ElasticSearch Service, Lambdas і т.д.) підключаються до застосунку через AppSync.

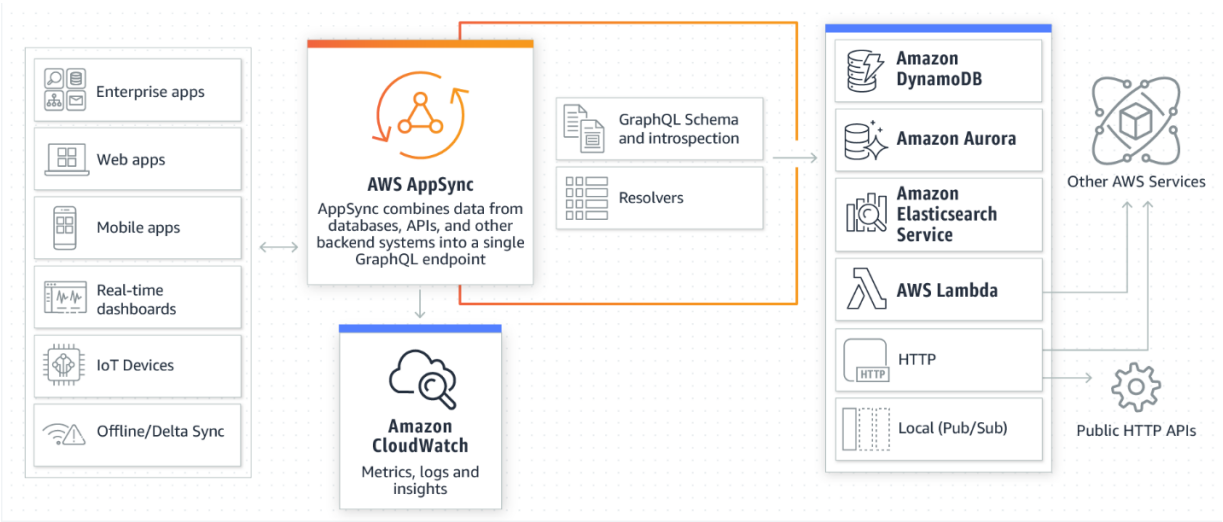

Рисунок 5 – офіційна схема використання AppSync

AppSync полегшує розробку GraphQL API, вирішуючи проблему безпечного підключення до баз даних, таких як AWS DynamoDB, Lambda тощо. Завдяки AppSync розробники мережевого застосунку мають можливість робити запити до багатьох джерел даних, мікросервісів та API з одного ендпоінта GraphQL. Для підвищення продуктивності можна підключити кешування даних, підписки для підтримки оновлень у

реальному часі, тимчасове збереження даних на стороні клієнта для його синхронізації після перепідключення до інтернету.

Так як AppSync, окрім швидкої, гнучкої розробки, надає наступні переваги:

- уніфікований захищений доступ до даних
- самомасштабованість в залежності від кількості користувачів і запитів
- підписки в режимі реального часу та офлайн-доступ

він є ідеальним рішенням при створенні соціальної мережі.

2.1.2 AWS Amplify

AWS Amplify – це Javascript бібліотека (фреймворк), яка надає безліч інструментів та готових компонентів для пришвидшення створення застосунків у середовищі AWS.

При створенні безсерверних застосувань, розробникам доводиться витрачати багато часу на конфігурування та керування сервісами. Amplify допомагає і з цією задачею. Безсерверна модель розробки є cloud-native, що дозволяє розробникам збирати та запускати програми без необхідності піклуватися про фізичний сервер. Звичайно, в цій моделі все ще є сервери, проте вони абстраговані від процесу розробки. Cloud провайдер бере на себе всю рутинну роботу з надання, підтримки та масштабування інфраструктури сервера. Розробникикам залишається лише написати код та запакувати його у контейнери для розгортання. Задеплоєні безсерверні застосунки реагують на кількість запитів і автоматично масштабуються за потреби. Якщо частина програми не використовується, постачальник хмари не бере за неї кошти, тому що серверні потужності надаються за дію-орієнтованою моделлю.

Amplify також надає готові рішення, модулі для авторизації, кешування, сховище, аналітика, API, UI компоненти тощо.

2.1.3 DynamoDB

Amazon DynamoDB — це повністю керована безсерверна база даних. DynamoDB є нереляційною базою даних, дані представлені у вигляді «ключ-значення».

Цей продукт ідеально підходить для запуску високопродуктивних застосунків будь яких масштабів, з великою кількістю даних та жорсткими вимогами до затримок. Не дарма його обрали такі компанії-гіганти, як Snapchat, Zoom, Dropbox.

DynamoDB надає безперервне резервне копіювання, вбудований захист, автоматичну реплікацію, кешування в пам’яті та інструменти експорту даних.

2.1.4 Amazon Cognito

Amazon Cognito надає можливість у пару кліків додати контроль доступу користувачів, форми реєстрації та входу. Він масштабується до мільйонів користівачів. Також підтримує вхід через соціальні мережі, такі як Facebook, Google, Apple, Amazon.

За допомогою Amazon Cognito легко створити контроль доступу користувачів та ресурсів. Можна визначити ролі і надавати їх певним користувачам, щоб вони мали доступ лише до певних авторизованих ресурсів.

2.1.5 Amazon S3

Три S у назві сервісу розшифровується як Simple Storage Service. Amazon S3 — це хмарне сховище об’єктів, яка пропонує провідну в галузі масштабованість, безпеку, продуктивність та доступність даних.

Amazon S3 має різноманітні функції, які можна використовувати для організації, керування даними, проводити аналітику. Розмір одного об’єкта може становити до 5 терабайт.

2.2 Технічні характеристики застосунку

У створеному сайті новин наявні наступні групи користувачів: зареєстрований та незареєстрований користувачі. Незареєстрований користувач має право:

- створити власний акаунт;
- переглядати загальну стрічку з прев’ю новин;
- відкривати новини;
- переглядати коментарі;

У той самий час після реєстрації користувачу відкривається більше можливостей, окрім зазначених вище:

- перегляд даних профілю;
- створення публікацій з новинами;
- прикріплення фото до постів;

- у навігаційному меню з’являється розділ «Опубліковані новини», де можна переглянути, відредагувати та видалити раніше створені новини;
- на сторінках з новинами від інших користувачів можна залишати коментарі;

Інтерфейс сайту є простим та інтуїтивно-зрозумілим. Веб-сайт налічує чотири основні сторінки:

«Стрічка» – загальна стрічка з прев’ю новин, які оновлюються у режимі реального часу, коли якийсь з користувачів створює нову публікацію. На кожну публікацію можна натиснути, аби прочитати повністю та залишити коментарі на наступній сторінці.

«Створити» – сторінка, яка надає безліч інструментів для створення публікації, дозволяє прикріпити фото, можна використовувати маркдаун розмітку, після створення публікації переадресовує на сторінку публікації.

«Профіль» – сторінка, на якій зареєстрований юзер може переглянути інформацію профілю та вийти з системи. Незареєстрований навпаки може пройти процедуру реєстрації та/або увійти у існуючий акаунт.

«Мої публікації» – сторінка, де можна переглянути усі свої публікації, внести до них зміни та видалити.

2.2 Обгрунтування вибору засобів розробки

Сайт новин написано на Javascript та React. В основі проекту – взаємодія користувацького інтерфейсу з сервісами AWS. AWS дозволяє швидко розроблювати та деплоїти застосунки і надає переваги, вкрай важливі для подібного застосунку.

Завдяки використанню AWS сервісів застосунок може легко розширюватися та масштабуватися до будь-якої кількості користувачів. Це важливо для сайту новин, де кожен користувач, може публікувати новини. Відкритий простір, де кожен може ділитися інформацією, в часи, коли кожна людина втягнута в інформаційну війну, подібний застосунок може привабити багато користувачів і має бути готовий до різкого збільшення трафіку. Завдяки AWS Amplify задеплоєні безсерверні застосунки реагують на кількість запитів і автоматично масштабуються за потреби.

Забпечує безпечний уніфікований доступ до всіх доступних даних. Анонімність на такій платформі також відіграє важливу роль, потрібно забезпечити збереження даних усіх користувачів.

Ще для сайту новин важлива можливість офлайн доступу і подальша синхронізація у мережі. Адже є кореспондентів і люди, які звітують у гарячих точках, де можливі проблеми з інтернетом.

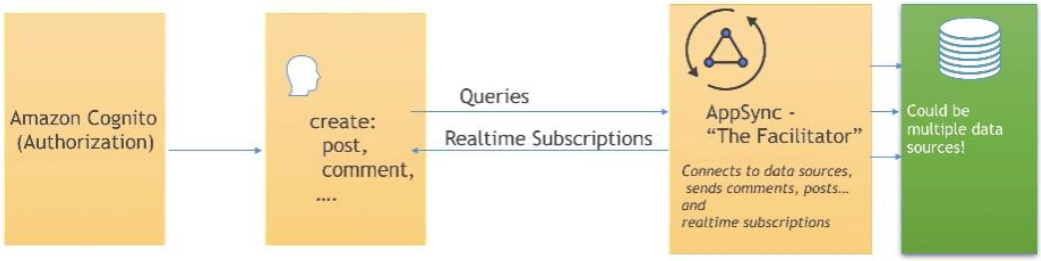

Рисунок 6 – архітектура створеного застосунку

Застосунок було створено за допомогою фреймворка Amplify. AppSync – посередник між інтерфейсом та сервісами від Амазон, за його допомогою було створено GraphQL схему, мутації, запити та підписки для бродкаст івентів.

schema.graphql U

amplify > backend > api > newsapp > schema.graphql

```
input AMPLIFY {
  globalAuthRule: AuthRule = { allow: public }
}
type Post
  @model
  @auth(
    rules: [
      { allow: owner, ownerField: "username" }
      { allow: public, operations: [read] }
    ]
  ) {
  id: ID!
  title: String!
  content: String!
  username: String
    @index(name: "postsByUsername", queryField: "postsByUsername")
  coverImage: String
  comments: [Comment] @hasMany(indexName: "byPost", fields: ["id"])
}

type Comment
  @model
  @auth(
    rules: [
      { allow: owner, ownerField: "createdBy" }
      { allow: public, operations: [read] }
    ]
  ) {
  id: ID!
  message: String
  post: Post @belongsTo(fields: ["postID"])
  postID: ID @index(name: "byPost")
}
type Subscription {
  newOnCreatePost: Post @aws_subscribe(mutations: ["createPost"])
}
```

Рисунок 7 – схема graphQL

За допомогою цієї схеми AWS AppSync згенерував API з наступними методами:

Queries: getComment, getPost, listComments, listPosts, postsByUserName

Mutations: createComment, createPost, deleteComment, deletePost, updateComment, updatePost

Subscriptions: newOnCreatePost, onCreate, onDelete, onUpdate

AWS Cognito – сервіс, який дозволив швидко додати засоби аутентифікації, він також зберігає інформацію користувачів. За його допомогою було створено пул користувачів.

DynamoDB база даних в якій було створено дві таблиці «Пости» та «Коментарі» зі зв’язком один до багатьох.

Amazon S3 став сховищем для великих зображень, які можна прикріпити до новин.

TailwindCss – бібліотека стилів, за допомогою якої було створено дизайн прямо у HTML тегах.

3.5. Тестування програми і результати її виконання

Тестування програми проводилося в браузері Opera.

Спочатку користувач потрапляє на основну сторінку, нескінченно проскролювану стрічку останніх новин. Користувач може натиснути на них, щоб відкрити контент публікації, переглянути та залишити коментарі.

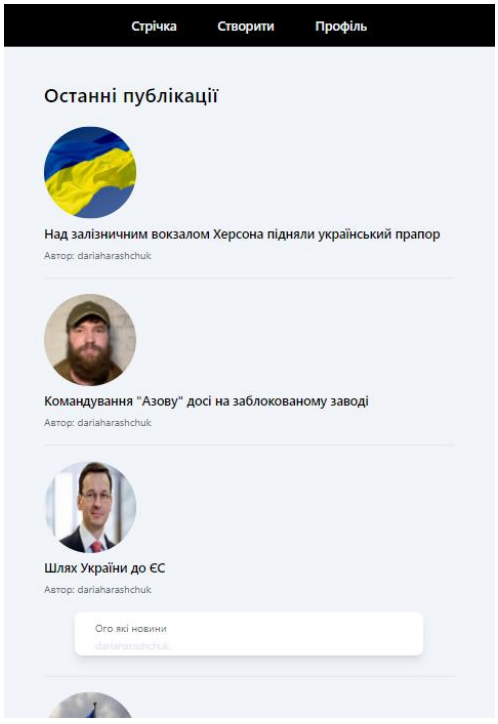

Рисунок 7 – сторінка «Стрічка» останніх новин

Сторінки «Створити» та «Профіль» відображують форму входу на сайт, тому що незареєстрований юзер має увійти у систему, щоб отримати до них доступ. Процес створення нового акаунту наступний:

1) Натиснути «Створити акаунт»
2) Ввести валідні дані (Юзернейм, пошту, пароль, номер телефону)
3) Отримати номер підтвердження реєстрації на пошту
4) Вставити номер у відповідний рядок

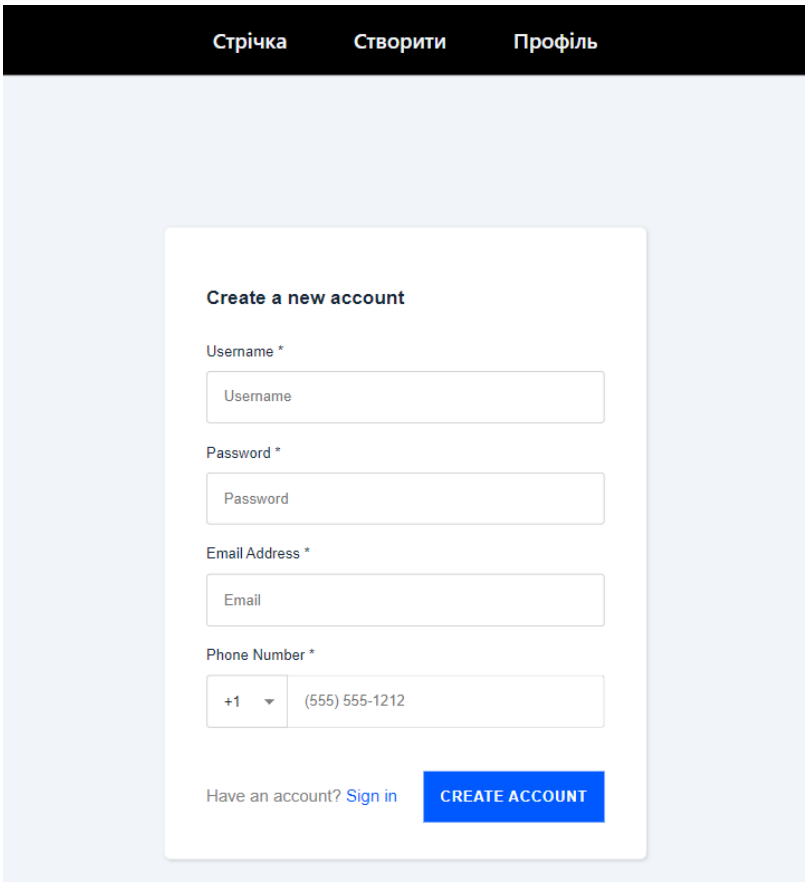

Рисунок 8 – форма реєстрації нового користувача

Після реєстрації користувач потрапляє на сторінку свого профілю, де може переглянути свої дані та за бажанням вийти з системи.

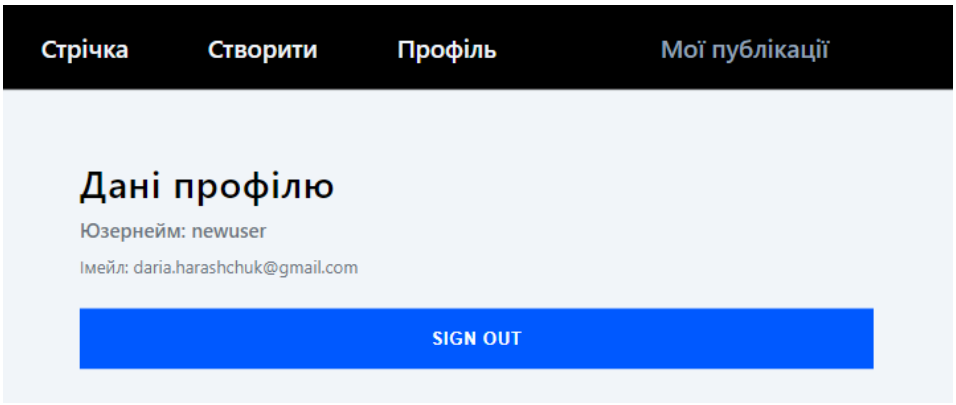

Рисунок 9 – сторінка «Профіль»

Зареєстрованому юзеру також доступні сторінки «Створити» та «Мої публікації». На сторінці створення нових публікацій доступно безліч інструментів для цього, форма підтримує основні інструменти для редагування тексту, маркдаун розмітку, можна також додати обкладинку до публікації.

Рисунок 10 – сторінка «Створити»

Тепер у розділі «Мої публікації» з’явилася щойно додана публікація. Там її можна редагувати, переглянути та видалити. Нова публікація також з’явилася в основній стрічці новин.

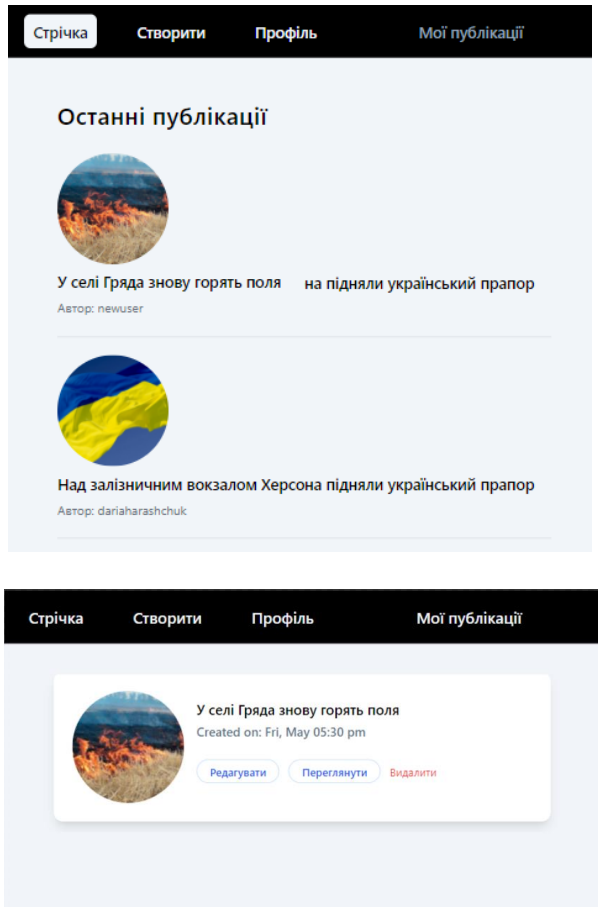

Рисунок 11 – новостворена публікація на сторінках «Стрічка» та «Мої публікації»

У зареєстрованого користувача також є можливість ознайомитися детальніше з кожною публікацією та залишити коментар.

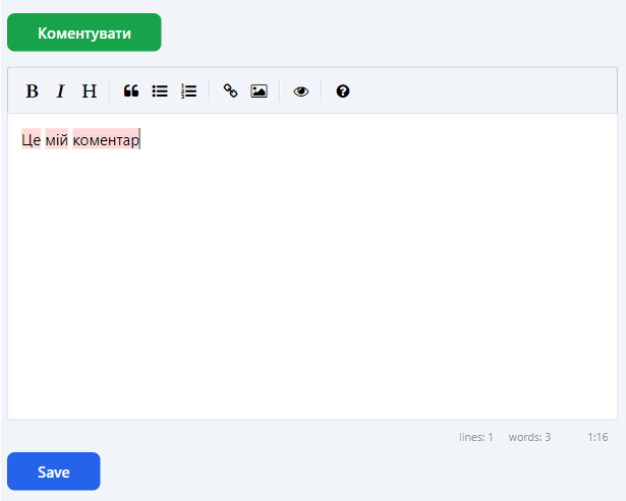

Рисунок 12 – форма для додавання коментарів

Висновки

Отже, поставлена задача була виконана: проведено детальний аналіз моделей мережевих застосувань. Було розглянуто однорівневу, архітектуру симетричної взаємодії, клієнт-серверну та багаторівневу архітектури, визначено їх плюси та мінуси.

Для розроблюваного застосунку «Сайту Новин», було обрано AWS платформу та її сервіси. Адже AWS сервіси закривають ключові потреби даного застосунку, а саме: забезпечують самомасштабованість і гнучкість системи, безпеку даних, офлайн доступ та подальшу синхронізацію даних.

У результаті виконання роботи було створено мережевий застосунок, завдяки якому користувачі можуть публікувати та переглядати новини в режимі реального часу. Окрім цього, завдяки гнучкості фреймворку Amplify можна з легкістю розширювати функціонал даного мережевого застосунку.

Список використаної літератури

1. Архітектура мережевих застосувань [Електронний ресурс] http://what-when-how.com/data-communications-and-networking/application-architectures-data-communications-and-networking/
2. Серверний кластер [Електронний ресурс] https://server-shop.ua/understanding-clustering-for-servers.html
3. Baset S. A., Schulzrinne H. An analysis of the skype peer-to-peer internet telephony protocol //arXiv preprint cs/0412017. – 2004.
4. Клієнт-серверна та P2P архітектури [Електронний ресурс] https://electronicspost.com/network-application-architectures/
5. Сервіси Amazon [Електронний ресурс] https://aws.amazon.com
6. Офіційна документація по Amplify [Електронний ресурс] https://docs.amplify.aws
7. Моделювання даних та створення зв’язків [Електронний ресурс] https://docs.amplify.aws/cli/graphql/data-modeling/#many-to-many-relationship