Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Побудова багаторівневого веб-застосунку на платформі Google Cloud Platform

Текстова частина до курсової роботи

за спеціальністю «Комп’ютерні науки» - 122

Керівник курсової роботи

ст. викладач

Черкасов Д.І.

_____

(підпис)

“__” _____ 2024 року

Виконав студент КН-3

Максімовіч Д.М.

“__” _____ 2024 року

Київ 2024

ЗМІСТ

Анотація

У цій роботі розглядається побудова багаторівневого веб-застосування на хмарній платформі Google Cloud Platform (GCP). Описано переваги та недоліки такого рішення. У роботі обґрунтовується вибір архітектури та розміщення. Також для практичної демонстрації реалізовується багаторівневий веб-застосунок на платформі GCP, принципи і функціональність якого детально описуються.

ВСТУП

Веб-застосунки у сучасному світі є невід’ємним елементом розвитку бізнесу, освіти та багатьох інших сфер діяльності. Через постійний розвиток технологій та потребу у продуктивних та ефективних додатках виникає необхідність у побудові таких програмних продуктів, що є гнучкими, надійними, захищеними, дозволяють швидко масштабовуватись та відповідають багатьом іншим критеріям. Таку задачу вирішує побудова багаторівневого веб-застосунку.

Багаторівневий веб-застосунок – це архітектурний підхід до розробки програмного забезпечення. Він передбачає розподіл системи на велику кількість компонентів на різних рівнях функціональності. Це забезпечує модульність, гнучкість та масштабованість.

Мета цієї курсової – проаналізувати переваги та недоліки багаторівневої архітектури на основі побудови веб-застосунку – сайту новин. Порівняти різні види архітектур, продемонструвавши аспекти багаторівневості, що забезпечують швидшу та ефективнішу масштабованість, гнучкість, підвищену безпеку, надійність та відмовостійкість. Також дослідити види інфраструктурних розміщень, показавши перевагу хмарної платформи, а саме Google Cloud Platform. Основна увага буде приділена структурній розробці свого рішення, щоб показати те, як саме будується та функціонує багаторівнева архітектура, зокрема найважливіші його компоненти: frontend, middleware та backend.

Робота складається з трьох розділів. У першому буде огляд можливих рішень, опис вимог сучасного веб-застосунку, вибір його архітектури та розміщення. У другому розділі буде структурна розробка, у якій оглянуто кожен компонент системи детально. А у третьому – розробка компоненти

веб-застосунку, а саме сайту новин, використовуючи багаторівневу архітектуру, зробивши акцент на її backend частину.

Результати дослідження можуть бути корисними для розробників програмного забезпечення, архітекторів систем, а також для всіх, хто цікавиться сучасними тенденціями веб-розробки та архітектурними підходами до створення високопродуктивних веб-застосунків.

РОЗДІЛ 1. ОГЛЯД МОЖЛИВИХ РІШЕНЬ

Важливою частиною реалізації будь-якої програми є вибір архітектури. Для веб-застосунків це має ключову роль ще й з позиції інфраструктури. Має вагу вибір саме такої архітектури, що оптимально підійде під проєкт, зважаючи на поставлені цілі та мету. Адже навіть коли вибрані правильні стратегії у написанні програмного коду, це може нівелюватись обмеженнями обраної інфраструктурної архітектури, яка є фізичною компонентою, а тому відіграє стратегічну роль. Рішення про використання тієї чи іншої архітектури потрібно приймати виходячи з потреб самого проєкту. І дуже важливо також прогнозувати його розвиток, адже деякі архітектурні рішення не дозволяють швидко та легко масштабовуватись, на відміну від, наприклад, багаторівневої архітектури, де завдяки розподіленій функціональності можна ефективно розширитись. Але і не потрібно також забувати про недоліки, які притаманні усім рішенням.

1.1 Вимоги розробки сучасних застосунків

Сучасні застосунки зобов’язані виконувати ряд вимог і відповідати багатьом критеріям. Веб-додатки не виключення, а тому при виборі відповідної архітектури потрібно зважати на такі рекомендації. Зокрема, застосунки повинні бути надійними, безпечними, мобільними, сумісними, продуктивними/швидкими, мати легкий та зрозумілий інтерфейс та мати можливість ефективно розширюватись. Від цих критеріїв залежить розвиток продукту, його потенціал, привабливість та популярність, вони полегшують супровід.

1.1.1 Масштабованість

Можливість застосунку швидко та легко розширюватись є однією з ключових. Вона забезпечує інтеграцію з іншими програмними продуктами. У сучасному світі дуже швидко з’являються нові застосунки і часто є необхідним створювати комунікації між ними. Легкодоступність розширення сильно залежить від вибраної архітектури веб-застосунку. Якщо компонентів вибраної інфраструктури для продукту є багато і логіка зв’язків між ними налагоджена належним чином, то зникає більшість проблем з масштабованістю. Якщо ж компонентів доволі мало, або він тільки один, то розширюваність стає проблемною і трудомісткою. Масштабованість визначає гнучкість веб-застосунку.

1.1.2 Надійність

Застосунок є надійним, якщо навіть у дуже складних і непередбачуваних ситуаціях програма видає хоча б якісь повідомлення, визначені програмістом за замовчуванням. Тобто потрібно вибирати архітектуру таким чином, щоб у будь-якому випадку застосунок функціонував коректно, а саме так, як це прогнозувала команда розробників. Якщо веб-додаток розміщено на одному сервері, то при якихось поламках цього елементу перестане працювати увесь веб-застосунок. Якщо ж інфраструктура, на якій є програма, складається з кількох елементів, то при поламці однієї такої частини у коді можна відслідкувати такі проблеми і додати відповідну логіку, щоб робота застосунку у таких важких ситуаціях екстрено не завершувалась. Ще краще, коли є ідентичні резервні компоненти серверів та інших частин інфраструктури, що можуть взяти на себе роботу тих частин, що перестали працювати через якісь неполадки. Тому більше компонентів означають більшу надійність при їх

правильних налаштуванні та логіці, що забезпечуватимуть неперервну роботу застосунку.

1.1.3 Безпека

Важливим елементом забезпечення коректної роботи застосунку є його захищеність. Безпеку можна розділити на два типи: внутрішню та зовнішню. Зовнішню захищеність від глобальної мережі можна забезпечити за допомогою міжмережевих екранів для контролю мережевого трафіка, щоб його можна було шифрувати, пропускати або ж відмовляти, залежно від того як він налаштований і за якими правилами вважає його безпечним або ні. Це створює уже суттєву безпеку застосунку. Таку ж захищеність ще може гарантувати NAT – інша технологія, що також дозволяє контролювати трафік на вході з глобальної мережі. Якщо зовнішня безпека не дуже сильно залежить від вибору архітектури, то внутрішня суттєво змінюється при зменшенні або збільшенні кількості компонентів, таких як різні сервери. Якщо інфраструктура має тільки одну компоненту, то важко забезпечити або навіть створити умови внутрішньої захищеності. Команді розробників доволі важко розробляти елементи безпеки в таких умовах. Якщо на цій єдиній компоненті буде якась вразливість тільки в одній частині, то може бути пошкоджено усі системи застосунку, що розміщені разом. Також при можливому проходженні небезпечного трафіка через міжмережевий екран, що деколи може відбуватись, також буде уражено усю програму. Якщо ж інфраструктура складається з декількох компонентів, то при пошкоджені однієї частини від якоїсь вразливості можна забезпечити безпеку інших компонентів. Також у такому випадку командам розробників легше планувати і розробляти систему захисту усього застосунку не разом, а по частинах, що дозволяє швидше та ефективніше справлятись з вразливостями, що з’являються. Це дозволяє працювати незалежно усім компонентам і налаштовувати безпеку зв’язків цих елементів і гарантувати, що вразливість

однієї частини не буде шкодити іншим. Таким методом можна і легше відслідковувати проблеми у захисті застосунку, і в результаті ефективніше їх виправляти.

1.1.4 Моніторинг

Моніторинг є важливим елементом, зокрема і для забезпечення безпеки застосунку. Його наявність дозволяє відслідковувати вразливості, інші помилки, неполадки та не тільки. Глобальні ситуації у програмі легше моніторити, коли компонентів інфраструктури мало, але на локальному рівні це краще тоді, коли їх більше, адже можна моніторинг розділити по частинах і відразу зрозуміло за яким компонентом ми спостерігаємо. Таким чином немає проблем у тому, щоб зрозуміти, де саме знайдено, наприклад, вразливість. А з глобальними проблемами при багатьох компонентів доволі важко зрозуміти, де і в чому причина.

1.2 Вибір архітектури

Застосунки можна розрізняти за багатьма параметрами, зокрема типом архітектури інфраструктури, на якій вони розміщені. Його вид визначають за кількістю різних компонентів, зокрема такі як сервери, наприклад, для бази даних, тощо. Ці елементи утворюють різні рівні, а тому застосунки поділяють на однорівневі, дворівневі, трирівневі та багаторівневі, кожен з яких має свої переваги та недоліки. Забезпечення надійності, масштабованості та безпеки напряму залежить від вибору такої архітектури.

1.2.1 Однорівнева архітектура

У однорівневій архітектурі усі частини системи, такі як бізнес-логіка, сервери, бази даних та інші, знаходяться на одній машині. Такий тип є

найпростішим. Це ефективно і оптимально у тому випадку, коли функціональних частин дуже мало у своєму обсягу. Перевагами такої архітектури є її швидкодія та простота. Якщо ж система величезна, або поки мала, але планується масштабування, то це абсолютно неоптимально через ряд недоліків: відсутність гнучкості, ускладнення моніторингу та масштабованості. Також навіть при малих помилках під час виконання програми може перестати працювати уся система, а не тільки її частина. Здебільшого така архітектура підходить для десктопних застосунків. У випадку веб-додатків і frontend, і backend компоненти також знаходитимуться на одній машині, а тому таке рішення для них є некоректним.

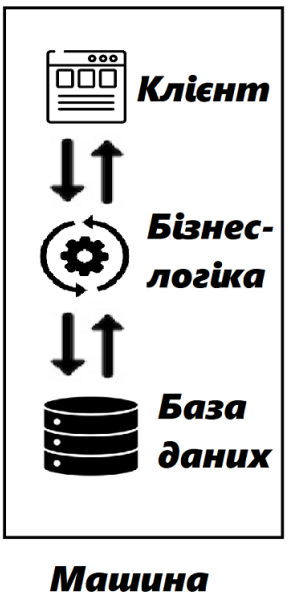

Рисунок 1.1 – Схема одноріневої архітектури

1.2.2 Дворівнева архітектура

Дворівнева архітектура представляє собою інфраструктуру, яка складається з двох компонентів. По-суті, вона включає в себе клієнт-серверну взаємодію. Тобто тут є два рівні. Клієнтський рівень представляє

користувацький інтерфейс, за допомогою якого відбувається представлення застосунку. А іншим рівнем виступає сервер, що бере на себе роль і сервера бази даних, звідки клієнт може взяти інформацію. При такій архітектурі бізнес-логіка реалізована на стороні клієнта, а тому навантаження на сервер з БД зменшується. Таке рішення уже може бути використане для реалізації невеликих веб-застосунків, забезпечуючи кращу масштабованість, надійність та безпеку. Проте недоліком є перебування сервера із базою даних на одному рівні. У цій та наступних архітектурах з'являється комунікація між компонентами, а тому і потреба у її захисті.

Веб-застосунки, реалізовані за такою архітектурою, розділяють на товстий та тонкий клієнти. У товстому клієнті бізнес-логіка у своїй переважній більшості реалізована на стороні клієнта, який потребуватиме більше ресурсів для функціонування веб-застосунку, а сервер у свою чергу не дуже завантажений. У тонкому ж клієнті усе навпаки: логіка знаходиться на сервері, що у цьому випадку навантажений, а на стороні клієнта може бути непотужна машина (комп'ютер).

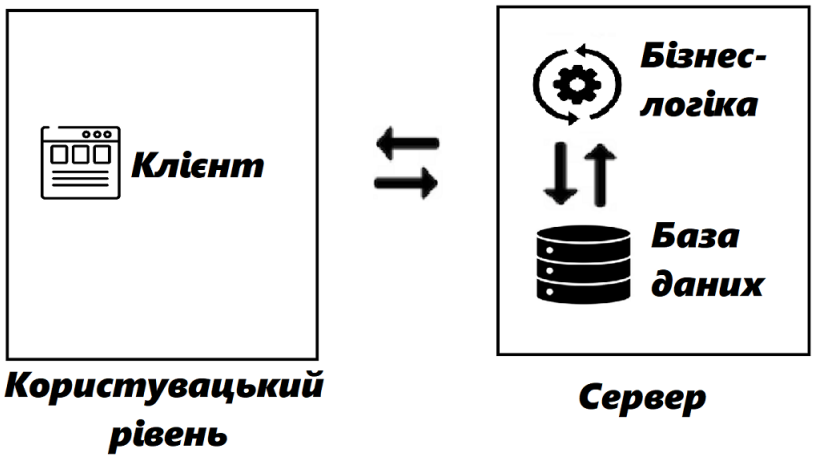

Рисунок 1.2 – Схема дворівневої архітектури

1.2.3 Трирівнева архітектура

Трирівнева архітектура є найтиповішою і найвживанішою. Вона складається з таких рівнів: користувацький інтерфейс (frontend), бізнес-логіка (middleware) та база даних (backend). Тобто між БД та користувачем з’являється ще один – проміжний. У такому випадку бізнес-логіка відділена окремо і вона взаємодіє з базою даних і користувацьким інтерфейсом, які прямий зв’язок між собою не мають. Це спрощує масштабування, але ускладнює систему, також покращує безпеку та надійність. Через те, що існує мережа взаємодії між компонентами для передачі даних, є потреба у забезпеченні безпеки такого зв’язку. Клієнт отримує дані через запит до сервера з бізнес-логікою, яка у свою чергу виконує якийсь ряд дій і звертається до віддаленої бази даних. Після цього БД повертає інформацію назад на сервер з логікою, який може провести якісь маніпуляції з ними, і поверне результат на користувацький рівень. Більшість веб-застосунків реалізовані таким чином.

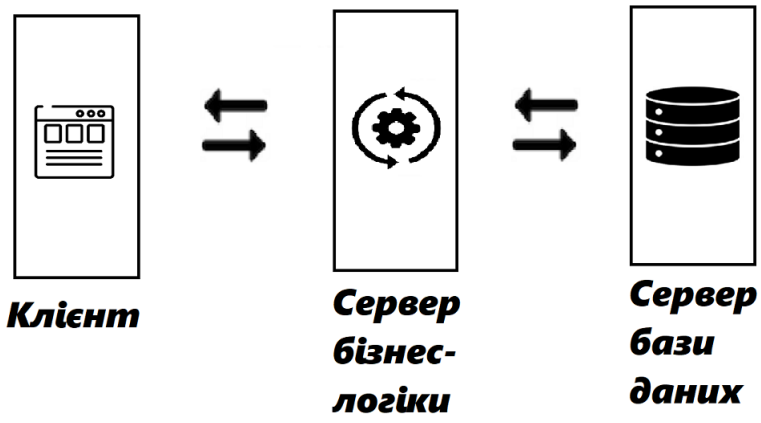

Рисунок 1.3 – Схема трирівневої архітектури

1.2.4 Багаторівнева архітектура

Багаторівнева архітектура у своєму складі має трирівневу архітектуру та один або більше додаткових рівнів. Інколи бізнес-логіку можуть розділяти на більше частин, кожну з яких розмістять на різних серверах, таким чином реалізувавши таку систему. Також у такій архітектурі може бути декілька окремих віддалених одна від одної баз даних для різних цілей. Може бути і окремий віддалений сервер для зберігання резервних копій БД. Однією з головних можливостей є забезпечення надійності через резервні сервери, які можуть взяти на себе увесь функціонал, якщо якийсь із основних компонентів поламався. Тобто вихід однієї частини з ладу не зупиняє роботу усієї системи. Для підвищення безпеки застосовуються міжмережеві екрани для фільтрації трафіку на вхід у застосунок та вихід з нього. Брандмауер можна вважати ще одним рівнем. У цій архітектурі також можуть бути і інші окремі елементи, наприклад, для фільтрації запитів, забезпечення доступу, кешування і не тільки. Це все забезпечує величезну надійність та безпеку, можливість швидкого масштабування, хоч і ускладнює розробку та моніторинг, а тому сучасні веб-застосунки орієнтуються саме на таку архітектуру. У табл. 1 показано переваги та недоліки різних рішень, де наглядно видно перевагу багаторівневості над іншими.

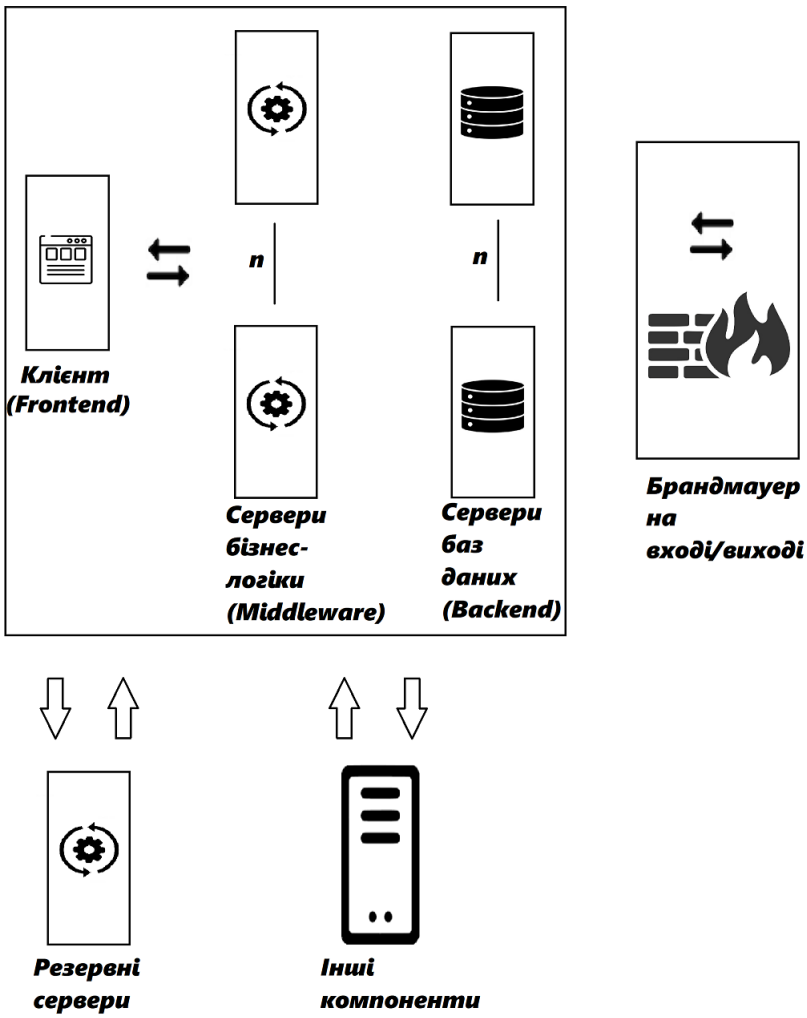

Рисунок 1.4 – Схема багаторівневої архітектури

1.2.5 Мікросервісна архітектура

Мікросервісну архітектуру можна розглядати як підтип багаторівневої. У її суть покладено розбиття системи на дуже багато малих елементів, між якими існує багато зв’язків. Мікросервіси мають такі ж рівні, як і багаторівнева архітектура, але на відміну від неї, усі ці компоненти дуже сильно розділені. Це дозволяє краще розподілити роботу між членами команди розробників. Найважливішими перевагами такої архітектури є масштабованість та надійність. При виході з ладу однієї компоненти решта продовжує працювати, а

через величезну кількість рівнів легше розширювати функціонал веб-застосунку, просто оновлюючи ту чи іншу частину системи. Це все полегшує розвиток додатка поелементно, але ускладнює бізнес-логіку, мережеву струтуру та моніторинг.

1.2.6 Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура (СОА) схожа до мікросервісної, але має набагато менше зв’язків між компонентами системи. СОА доволі специфічна і не підходить для ряду задач. ЇЇ легше тестувати та моніторити через спрощену мережеву структуру, але через це і важче масштабовувати.

	1-рівневе	2-рівневе	3-рівневе	Багаторівневе	Мікросервіси
Спрощеність розробки та підтримки	5	4	3	2	1
Безпека системи	1	2	3	5	4
Надійність	1	2	3	5	4
Відмовостійкість	1	2	3	4	5
Масштабування	1	2	3	4	5
Моніторинг	5	4	3	2	1
Тестування	5	4	3	2	1
Швидкість оновлення	5	4	3	1	2

Таблиця 1. Порівняльна характеристика архітектур за 5-бальною шкалою (1 - найгірше, 5 - найкраще)

1.3 Вибір розміщення

Після вибору архітектури постає питання правильного розміщення застосунку, яке забезпечить його надійність, безпеку і не тільки.

1.3.1 Розміщення на власних ресурсах

Розміщення на власних ресурсах означає повну підтримку усіх процесів з моменту розробки силами розробника застосунку. Власник сам повністю веде технічне обслуговування, зокрема відповідає за безпеку, розширення, оновлення та цілодобову роботу інфраструктури. У такому випадку усі дані точно під захистом через те, що власник має повний контроль над системою. Але підтримка власного дата-центру є дороговартіснішим за інші види розміщень. Таке рішення є правильним для гігантських корпорацій, що мають великі програмні проєкти, багато капіталу та необхідність забезпечення максимальної безпеки. Розміщення на власних ресурсах дозволяє контролювати не тільки сервери з даними, але й усю інфраструктуру. Це забезпечує можливість самостійно вибудовувати потрібну архітектуру, а також оновлювати різне обладнання до власних потреб. Але через збільшення гнучкості з’являється і проблема, що налаштування такої системи потребує вмінь, зусиль, часу і коштів, а потім постійну перевірку усіх частин.

1.3.2 Розміщення на орендованих ресурсах

Розміщення на орендованих ресурсах здебільшого полягає у дата-центрах (центрах даних). Центри даних призначені для зберігання та збирання інформації. Вони складаються з багатьох комп’ютерів та обладнанням для безпечного зовнішнього зв’язку. Дата-центри гарантують безпеку даних, що на них зберігаються. Для вибору центру зберігання та обробки даних (ЦЗОД) надається величезний асортимент, їх можна розрізнити за розміром,

надійністю і так далі. Тобто при рішенні з орендою немає проблем з вибором дата-центру. У цьому і полягає їхня перевага. Вони дешевші, аніж розміщення на власних ресурсах, не потребують постійної підтримки і перевірки з боку розробників застосунку. Також дата-центри не потрібно налаштовувати з нуля – вони уже готові, їх потрібно тільки орендувати. Тобто можна дуже швидко почати підтримку роботи веб-застосунку з таким рішенням. Але недоліком є безпека, оскільки не можна бути впевненими у тому, що дані гарантовано знаходяться під повним захистом. Також таке рішення надає менше гнучкості з налаштуванням інфраструктури. Тобто розміщення на орендованих ресурсах є гарним вибором для швидкого старту і підійде для невеликих компаній. Дата-центри потрібно обирати за рівнем доступності – якщо він вищий за 99.99%, то це надійний ЦЗОД.

1.3.3 Розміщення на хмарних ресурсах

Найбільшої популярності у сучасному світі набуває розміщення на хмарних ресурсах. У такій ситуації більшість роботи беруть на себе постачальники послуг. Хмарні сховища мають ряд переваг, зокрема те, що дані зберігаються не в одному місці на одному сервері, а у різних точках по всьому світу. Це гарантує збереження даних, якщо відбудеться якийсь природний катаклізм, до прикладу. Також велика перевага у тому, що потрібно платити за послуги тільки тоді, коли ними користуєтесь. Хмарні ресурси дозволяють швидко збільшити або зменшити потужність системи, залежно від потреб. Окрім того, таке розміщення є дуже гнучким через scaling, воно дозволяє ефективно і в найкоротші терміни оновлювати застосунок, зокрема за допомогою контролю версій. У табл. 2 наглядно показано плюси та мінуси різних ресурсів. Відповідно до неї можна впевнено сказати про значні переваги хмірного розміщення.

	Власні ресурси	Орендовані ресурси	Хмарні ресурси
Контроль	+	+-	-
Вартість	-	+-	+
Надійність	+	-	+-
Безпека	+	-	+-
Відмовостійкість	+-	-	+
Гнучкість / масштабованість	+-	-	+
Оновлення	+-	-	+
Розгортання	-	+	+-

Таблиця 2. Порівняльна характеристика розміщень

Таким чином можна сказати, що багаторівнева архітектура та хмарне розміщення має найбільшу кількість переваг, а тому є найкращим рішенням.

РОЗДІЛ 2. СТРУКТУРНА РОЗРОБКА ВЕБ-ЗАСТОСУВАННЯ

2.1 Опис компонентів веб-застосування

У цьому розділі буде розглянуто структурну розробку власного рішення, а саме побудови багаторівневого веб-застосунку сайту новин. Головними компонентами у такій архітектурі є frontend (користувацький рівень), middleware (бізнес-логіка) та backend (сервер бази даних). Також огляд структурної розробки і решти рівнів.

2.1.1 Frontend

Frontend (користувацький рівень) відіграє важливу роль, адже саме він відповідає за зовнішній вигляд веб-застосунку, інтерфейс та взаємодію програми з користувачем. Тобто він звертає на себе увагу клієнта та впливає на перше враження, від того, наскільки правильно він реалізований, залежить успіх продукту.

Основними складовими frontend компоненти є Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) та мова програмування JavaScript (JS). HTML відповідає за структуру веб-сторінок, усі їхні елементи, наприклад, поля вводу, кнопки, зображення, форми, тексти та решту, частково він відповідає і за стилі. CSS призначений для опису зовнішнього вигляду продукту. Він задає стилі елементів HTML, зокрема визначає їм кольори, розміри, шрифти і не тільки. JavaScript є динамічною ООП мовою, зокрема використовується для створення сценаріїв вебсторінок для можливої логічної взаємодії користувача з програмою, а тому це більше відноситься до бізнес-логіки (middleware частини) і у підпункті 2.1.2 буде детально розглянуто JS.

Для належної реалізації frontend використовуються різноманітні інструменти, зокрема і фреймворки, найвідоміші з них: React, Vue, Angular. Такий підхід дозволяє пришвидшити розробку. Фреймворки забезпечують масштабованість, кросплатформність та ефективність, також мають велику спільноту для підтримки і вирішення проблем. Головна їхня суть – готові рішення для багатьох типових задач, що полегшує розробку.

2.1.2 Middleware

Middleware відповідає за бізнес-логіку. Саме тут головним елементом є популярна мова JavaScript, хоча і можливе використання інших, до прикладу, TypeScript. За допомогою неї будується зв’язок між користувацьким рівнем та бізнес-логікою. Наприклад, створюється функція і через подію (event) onclick якоїсь кнопки прив’язується цей метод. Тоді при натисканні на кнопку буде викликатись відповідна функція. Це тільки один з можливих прикладів. Якщо говорити про сайт новин, то зокрема при натисканні на кнопку типу “Подивитись усі новини” має виконуватись відповідна функція, що робить запит до бази даних через fetch, що поверне список новин, який потім буде відсортовано за їхньою датою, або іншим принципом, це і є частиною бізнес-логіки. Ці дані будуть повернуті на frontend рівень, де вони будуть відтворені користувачу у якомусь вигляді. Таким чином цей проміжний етап обробляє запити до бази даних. На цьому рівні реалізується аутентифікація/авторизація, обробка помилок, логування, кешування і не тільки.

2.1.3 Backend

Backend частина відповідає за сервер бази даних. Зокрема на цьому рівні знаходяться функції для того, щоб повернути дані з БД. Для цього може використовуватись мова програмування JavaScript. Для створення сервера

можна використати бібліотеку express, а для бази даних – MySQL, PostgreSQL, Oracle, MSSQL, SQLite або інші, зокрема бібліотеку Mongoose, яка зручна для використання на JS. Для доступу до БД можна створювати HTTP-ендпоїнти з різними HTTP-методами, у табл. 3 продемонстровано основні їхні типи і призначення. Здебільшого такі функції повертають дані з сервера, обгорнуті у об’єкти Promise. З middleware можна до них звернутись асинхронно через раніше згаданий fetch з відповідною обробкою даних, або їхнім надсиланням на сервер.

HTTP-метод	**Призначення**
GET	Отримання даних із сервера
POST	Передача своїх даних на сервер БД, створення нового ресурсу на сервері
PUT	Передача даних на сервер БД, оновлення наявного ресурсу або створення нового, якщо такого не існувало
PATCH	Передача даних на сервер. Використовується як PUT-метод, але на відміну від нього тільки частково оновлює наявний ресурс
DELETE	Видалення наявного ресурсу на сервері
HEAD	Ідентичний до GET-запиту, але не має тіла відповіді. Використовується для отримання метаданих про ресурс без зайвих даних
OPTIONS	Отримання інформації про можливості сервера або параметри, які підтримуються для ресурсу. Такий запит застосовується для отримання доступних HTTP-методів та іншої інформації, що необхідна для взаємодії з сервером

Таблиця 3. Основні HTTP-методи і їхні призначення

2.2 Вибір хмарної платформи та огляд Google Cloud Platform

Як один з варіантів хмарного розміщення можна обрати Google Cloud Platform (GCP). Цю службу використовують дуже популярні продукти, зокрема Youtube та Google Search. Google Cloud Platform конкурує з Amazon Web Services, Microsoft Azure та Oracle Cloud. Вона надає можливості для керування, а також інструменти для обчислення, аналізу та зберігання даних. Ще є підтримка з машинним навчанням та багато інших сервісів. Google Cloud Platform пропонує такі оточення як інфраструктура як послуга, платформа як послуга та безсерверні обчислення. Має ряд переваг: швидке розгортання, легка масштабованість, високі надійність, безпека та доступність, аналітичні можливості, гнучкі оновлення і управління.

Google App Engine – платформа для хостингу застосунків. Однією з переваг цього сервісу є наявність безкоштовного розміщення, хоч і обмеженого.

Computer Engine – інфраструктура як послуга, що дає змогу запускати віртуальні машини на потужностях Google. Має багато переваг, зокрема масштабованість, доступність, багато налаштувань, аналітика та зручне управління.

Kubernetes Engine – сервіс атоматичного розгортання застосунків у контейнерах для Kubernetes, що надає можливість масштабування та управління.

Cloud functions – сервіс, що надає можливість викликати безсерверний код, зокрема функції. Підтримує велику кількість мов програмування.

Cloud Run – схожий до Kubernetes Engine, але на відміну від нього цей сервіс проводить безсерверні обчислення.

Це були приклади платформ та інфраструктур, що надають Google Cloud Platform. Тепер розглянемо сховища та бази даних цієї служби.

Storage – програма як послуга, сховище об’єктів, у якому можна зберігати файли. Доступ надається через RESTful API. Цей сервіс також забезпечує масштабованість, управління, надійність, продуктивність та зберігання безмежної кількості даних.

Cloud SQL – хмарна база даних. Сервіс реалізований для дуже популярних керованих реляційних БД MySQL, PostgreSQL та Microsoft SQL Server.

Bigtable – NoSQL база даних, що побудована за принципом ключ-значення. Система є надзвичайно продуктивною і швидкою, що робить її дуже популярною.

Cloud Spanner – масштабована багатоваріантна база даних, що є географічно розподіленою та підтримує розподілені транзакції. Його попередником є Google Bigtable.

Cloud Datastore – повністю керована NoSQL база даних, що є високо масштабованою.

AlloyDB – хмарна база даних, що є сумісною з PostgreSQL.

BigQuery – інфраструктура як послуга. Це безсерверне сховище даних, що дозволяє широко та масштабно аналізувати величезні набори даних. Має свої особливості щодо управління та контролю.

Cloud AutoML – сервіси для машинного навчання.

Filestore – мережеве сховище файлів.

Persistent Disk – дисковий простір, що є віртуальним і потрібен для хмарних віртуальних машин.

У GCP доступно і дуже багато інших сервісів, таких як Cloud Vision, Google Video Intelligence, Cloud Machine Learning Engine, Google Genomics, Cloud Pub/Sub і не тільки.

Таким чином Google Cloud Platform є дуже ефективною хмарною службою, що забезпечує швидке розгортання, масштабованість, надійність, безпеку та доступність.

РОЗДІЛ 3. РОЗРОБКА ВЛАСНОГО РІШЕННЯ

У цьому розділі буде детально реалізовано одну компоненту багаторівневого веб-застосунку сайту новин – backend, та частково – middleware і frontend. Перші два рівні реалізовані на мові програмування JavaScript.

3.1 Код веб-застосунку

Для реалізації бекенд-сервера використовується бібліотека express, а для бази даних – бібліотека Mongoose. У БД створено схему для представлення новини.

Для сервера бекенду реалізовано RESTful API (Representational State Transfer Application Programming Interface), що надає великі можливості для застосунку. Така реалізація дозволяє маніпулювати з даними усіма необхідними операціями, такими як CRUD (Create Read Update Delete). Ці методи дозволяють створювати, читати, оновлювати та видаляти ресурси. Нижче перелічено усі реалізовані HTTP-методи:

- GET /news – для отримання масиву усіх новин у базі даних через GET
- GET /new/{id} – для отримання інформації про одну новину за параметром id через GET
- POST /new – для передачі даних на сервер, щоб створити нову новину через POST
- PUT /new/{id} – для передачі даних на сервер, щоб оновити наявну новину через PUT. id тут виступає у ролі параметра, а саме ідентифікатора новини, що і буде змінено

• DELETE /new/{id} – для видалення наявної новини із бази даних за допомогою HTTP-метода DELETE, де id – параметр, а саме ідентифікатор новини, що буде видалено

Для PUT і DELETE методів наявні параметри, а саме id – ідентифікатор новин. Для передачі даних для створення новини через POST та оновлення через PUT також використовується body, у якому і передається решта інформації про новини, зокрема їхні назви, теми, автори, дати і вмісти. Ці дані представлені у JSON-форматі. Якщо у ході звернення до якогось з цих методів виникають якісь помилки, то вони обробляються відповідним чином.

Також використовується бібліотека cors (Cross-Origin Resource Sharing). Це дозволяє контролювати з яких джерел дозволено взаємодіяти з сервером, що забезпечує кращий захист користувачів від нападів типу Cross-Site Request Forgery (CSRF) та Cross-Site Script Inclusion (XSSI).

3.2 Інтерфейс веб-застосунку

Користувацький рівень багаторівневого веб-застосунку цієї курсової роботи несе більше демонстраційний сенс. У ньому є 2 головні частини: початкова і основна. На рисунку 3.1 можна побачити вигляд другої. На ній можна побачити виведення карток з усіма новинами. На кожній з них відображаються назва і текст новини, а також є кнопка “More” для перегляду усієї новини з повною інформацією. Також нижче є кнопка “News handling” для додавання, видалення та оновлення новин. Ці операції доступні тільки для адміністратора сайту, який має знати пароль для входу. Відповідні поля з’являються при натисканні на кнопку “News handling”. При некоректному проходженні авторизації виводиться сповіщення про помилку. Якщо ж вхід успішний, то з’являється можливість виконання усіх CRUD операцій.

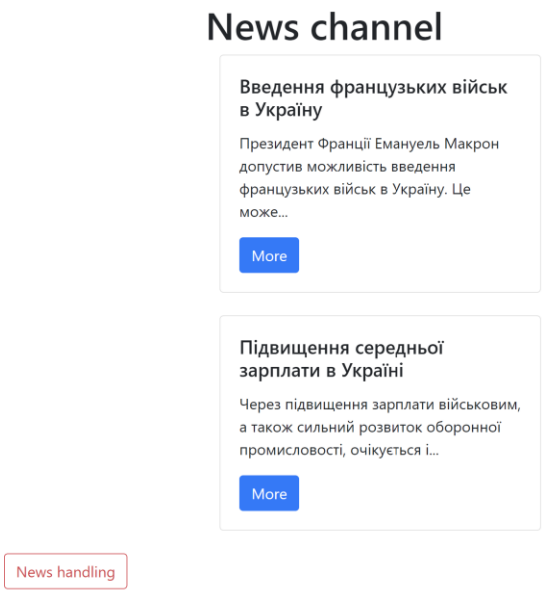

Рисунок 3.1 – Основна частина інтерфейсу

Можливість CRUD операцій показана у вигляді відповідних кнопок “Add news”, “Delete news”, “Update news” та “Cancel”. Остання кнопка дозволяє скасувати операцію і повернутись на основну сторінку. При натисканні на одну з перших трьох кнопок з’являються поля для вводу інформації для додавання, видалення або оновлення новин. Можливість скасування операції все ще залишатиметься. На рисунку 3.2 продемонстровано події на усі кнопки, тобто вигляд інтерфейсу при натисканні на відповідні кнопки, зокрема для додавання (add), видалення (delete) та оновлення (update) новин.

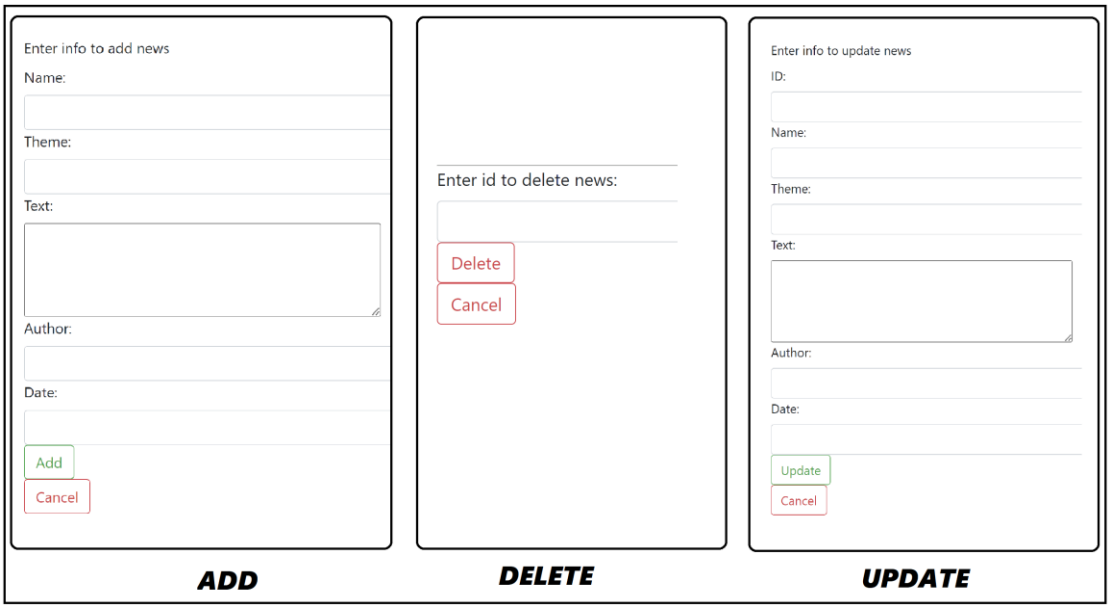

Рисунок 3.2 – CRUD-операції веб-застосунку

На стороні бізнес-логіки, що реалізована у файлі logic.js, який наданий у додатку Б, реалізовано багато функцій для зміни вигляду веб-інтерфейсу. Серед них є і ті, в яких реалізовано звернення до сервера бекенду. Вони використовують fetch для такої функціональності. Функція main звертається за допомогою HTTP-метода GET, що повертає увесь список новин, а потім відображає їх згідно з реалізованим інтерфейсом, тобто в картках. Наявні також функції add_news, update_news та delete_news, що використовують POST, PUT та DELETE методи відповідно. Це і надає можливість функціонування RESTful API, що дозволяє читати, видаляти, оновлювати та додавати новини на сайт новин.

Для html елементів використовується бібліотека стилів Bootstrap, що дозволяє надати кращого вигляду кнопкам та не тільки, а також реалізувати картки для відображення новин.

3.3 Розміщення веб-застосунку на хмарній платформі

Для розміщення веб-застосунку на хмарній платформі використано Google App Engine, оскільки програму реалізовано на Node.js, яка підтримується цією службою. Цей сервіс надає два середовища: стандартний та гнучкий, що надає більше переваг. Обрано стандарний варіант, який може бути безкоштовним, але з дуже обмеженими можливостями. Для реалізації розміщення потрібно:

1) Створити проєкт на GCP.
2) Встановити Google Cloud SDK.
3) Створити файл app.yaml та налаштувати проєкт відповідно до усіх потреб.
4) Запустити команду gcloud app deploy через термінал.
5) Перевірити чи веб-застосунок коректно працює за відповідною URL-адресою.

Завдяки таким діям Node.js сервер розгорнеться/розміститься на Google App Engine.

У реалізованому веб-застосунку для бази даних використовується MongoDB, а тому додаткового розміщення БД не потрібно робити. Але якби це була якась інша, то можна було б розмістити її на Google Cloud SQL, або насхожих до неї.

Також для автоматизації процесу збирання, тестування і розгортання програмного забезпечення використовується Google Cloud Build. Для його налаштування потрібен файл cloudbuild.yaml, у якому описано усі необхідні кроки, що виконуватимуться при кожному збиранні веб-застосунку.

Перелічені рішення на хмарній платформі забезпечують надійність, безпеку, масштабованість, гнучкість і продуктивність розробленого багаторівневого веб-застосунку.

Висновки

У цій курсовій роботі було проаналізовано різні архітектури побудови веб-застосунків та варіанти їхніх розміщень, а також описано переваги багаторівневості та хмарної платформи. За цими принципами реалізовано сайт новин, що дозволяє ними оперувати. Використано технології Google Cloud Platform для хмарного розміщення, зокрема Google App Engine.

У результаті було розроблено веб-застосунок, який є практичним прикладом, що демонструє функціонування багаторівневості та взаємодію з хмарними сервісами Google Cloud Platform.

Список використаної літератури

1. Google Cloud Platform [Електронний ресурс] – Режим доступу: https://cloud.google.com
2. Youtube Channel Google Cloud Tech [Електронний ресурс] – Режим доступу: https://www.youtube.com/user/googlecloudplatform/videos
3. Youtube Video Google Cloud Platform Full Course 2023 [Електронний ресурс] – Режим доступу: https://youtu.be/e385hDgXaAQ?si=kW6OyLowDPP0jQbW
4. GCP App Engine Course [Електронний ресурс] – Режим доступу: https://www.gopomelo.com/training/gcp-app-engine
5. Microsoft Learn – Deployment Patterns [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff646997(v=pandp.10)
6. Fowler, Martin "Patterns of Enterprise Application Architecture" (2002). Addison Wesley.
7. Bill Wilder. Cloud Architecure Patterns — O'Reilly Media, Inc., 2012
8. Stackpath – Web Application [Електронний ресурс] – Режим доступу: https://www.stackpath.com/edge-academy/what-is-a-web-application/
9. Node.js Site [Електронний ресурс] – Режим доступу: https://nodejs.org/
10. W3Schools JS Course [Електронний ресурс] – Режим доступу: https://www.w3schools.com/js/
11. Medium – Multi-tier Architecture [Електронний ресурс] – Режим доступу: https://medium.com/ayokoding/system-design-multi-tier-architecture-3213ea3e206a
12. Juniper – Deploying a Multi-Tier Web Application [Електронний ресурс] – Режим доступу: https://www.juniper.net/documentation/us/en/software/contrail-

networking19/contrail-networking-security-user-guide/topics/task/web-use-case-vnc.html

13. TechTarget – Cloud Infrastructure [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchcloudcomputing/definition/cloud-infrastructure
14. Folio3Dynamics – On Premise vs. Cloud: Key Differences, Benefits and Risks [Електронний ресурс] – Режим доступу: https://dynamics.folio3.com/blog/on-premise-vs-cloud-erp-software-difference/
15. IBM – What are Iaas, Paas and Saas? [Електронний ресурс] – Режим доступу: https://www.ibm.com/topics/iaas-paas-saas
16. Tutorialspoint – JS Course [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/javascript/index.htm

Додаток А (backend)

Файл backend_server.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const app = express();
const PORT = process.env.PORT || 3000;

mongoose.connect("mongodb://127.0.0.1:27017/news", {});
const db = mongoose.connection;
db.on("error", (error) => console.log(error));
db.once("open", () => console.log("DB connected"));
app.use(express.json());

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  next();
});

app.use(cors());

const server = app.listen(PORT, () => {
  console.log(`Server is running on port ${server.address().port}`);
});

app.options("*", cors());

const newsSchema = new mongoose.Schema({
  name: { type: String },
  theme: { type: String },
  text: { type: String },
  author: { type: String },
```

```
  date: { type: String },
});

const News = mongoose.model("News", newsSchema);

app.get("/news", async (req, res) => {
  try {
    await News.find({})
      .then(function (em) {
        res.json(em);
      })
      .catch(function (err) {
        console.log(error);
      });
    console.log("get /");
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

app.get("/new/:id", async (req, res) => {
  try {
    const news = await News.findById(req.params.id);
    if (!news) {
      return res.status(404).json({ message: "News are not found" });
    }
    res.json(news);
    console.log("get /new/:id");
  } catch (err) {
    console.error(err.message);
    res.status(500).json({ message: "Server error" });
  }
});

app.post("/new", async (req, res) => {
  console.log("post /");
```

```
  const newNews = new News({
    name: req.body.name,
    theme: req.body.theme,
    text: req.body.text,
    author: req.body.author,
    date: req.body.date,
  });
  try {
    const newN = await newNews.save();
    res.status(201).json(newN);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});

app.delete("/new/:id", async (req, res) => {
  try {
    console.log("delete /:id");
    const id = req.params.id;
    await News.findByIdAndDelete(id);
    res.status(200).json({ message: "Deleted" });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

app.put("/new/:id", async (req, res) => {
  try {
    console.log("put /:id");
    const id = req.params.id;
    const { name, theme, text, author, date } = req.body;

    const updatedNews = await News.findByIdAndUpdate(
      id,
      {
        name,
```

```
        theme,
        text,
        author,
        date,
      },
      { new: true }
    );

    res
      .status(200)
      .json({ message: "News updated successfully", updatedEmail });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});
```

Додаток Б (middleware)

Файл logic.js

```
function main() {
  document.getElementById("news_handling").style.display = "block";
  document.getElementById("CRUD").style.display = "block";
  document.getElementById("main").style.display = "none";
  document.getElementById("prime").style.display = "block";

  fetch("http://localhost:3000/news")
    .then((response) => response.json())
    .then((data) => {
      console.log(data);
      const newsRow = document.getElementById("row_news");
      newsRow.innerHTML = "";
      document.getElementById("row_news").style.display = "block";

      for (let news of data) {
        const firstDiv = document.createElement("div");
        firstDiv.setAttribute("class", "col-md-4");
        const secondDiv = document.createElement("div");
        secondDiv.setAttribute("class", "card mb-4");
        const thirdDiv = document.createElement("div");
        thirdDiv.setAttribute("class", "card-body");
        const h5 = document.createElement("h5");
        h5.setAttribute("class", "card-title");
        h5.textContent = news.name;
        const p = document.createElement("p");
        p.setAttribute("class", "card-text");
        if (news.text.length <= 100) {
          p.textContent = news.text;
        } else {
          p.textContent = news.text.slice(0, 100) + "...";
        }

        const a = document.createElement("a");
        a.setAttribute("class", "btn btn-primary");
```

```
        a.textContent = "More";

        a.onclick = function () {
          showNews(news);
        };

        thirdDiv.appendChild(h5);
        thirdDiv.appendChild(p);
        thirdDiv.appendChild(a);
        secondDiv.appendChild(thirdDiv);
        firstDiv.appendChild(secondDiv);
        newsRow.appendChild(firstDiv);
      }
    })
    .catch((error) => {
      console.error("Помилка:", error);
    });
}

function returnToMainPage() {
  document.getElementById("new").style.display = "none";
  document.getElementById("return_button").style.display = "none";
  main();
}

function showNews(news) {
  document.getElementById("CRUD").style.display = "none";
  let piece_news = news;
  document.getElementById("row_news").style.display = "none";
  document.getElementById("new").style.display = "block";
  document.getElementById("name").textContent = piece_news.name;
  document.getElementById("author").textContent =
    "Author: " + piece_news.author;
  document.getElementById("date").textContent = "Date: " + piece_news.date;
  document.getElementById("theme").textContent = piece_news.theme;
  document.getElementById("text").textContent = piece_news.text;
```

```
  document.getElementById("id").textContent = "Id: " + piece_news._id;
  document.getElementById("return_button").style.display = "block";
}

function admin_auth() {
  document.getElementById("news_handling").style.display = "none";
  document.getElementById("main").style.display = "none";
  document.getElementById("prime").style.display = "none";
  document.getElementById("Enter").style.display = "block";
  document.getElementById("cancel").style.display = "block";
  document.getElementById("continue").style.display = "block";
}

function cancel() {
  main();
  document.getElementById("Enter").style.display = "none";
  document.getElementById("cancel").style.display = "none";
  document.getElementById("continue").style.display = "none";
  document.getElementById("operations").style.display = "none";
  document.getElementById("deletebtn").style.display = "block";
  document.getElementById("addbtn").style.display = "block";
  document.getElementById("updatebtn").style.display = "block";
  document.getElementById("del").style.display = "none";
  document.getElementById("update").style.display = "none";
}

function crud() {
  const password = document.getElementById("pass").value;
  if (password === "66") {
    document.getElementById("Enter").style.display = "none";
    document.getElementById("continue").style.display = "none";
    document.getElementById("operations").style.display = "block";
    document.getElementById("addbtn").style.display = "block";
    document.getElementById("add").style.display = "none";
  } else {
    alert("Incorrect password");
```

```
  }
}

function hideCRUDButtons() {
  document.getElementById("addbtn").style.display = "none";
  document.getElementById("deletebtn").style.display = "none";
  document.getElementById("updatebtn").style.display = "none";
}

function addNews() {
  hideCRUDButtons();
  document.getElementById("add").style.display = "block";
}

function updateNews() {
  hideCRUDButtons();
  document.getElementById("update").style.display = "block";
}

function deleteNews() {
  hideCRUDButtons();
  document.getElementById("del").style.display = "block";
}

function add_news() {
  const newNews = {
    name: document.getElementById("namea").value,
    theme: document.getElementById("themea").value,
    text: document.getElementById("texta").value,
    date: document.getElementById("datea").value,
    author: document.getElementById("authora").value,
  };
  fetch("http://localhost:3000/new", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
```

```
    },
    body: JSON.stringify(newNews),
  })
    .then((response) => response.json())
    .then((data) => {
      cancel();
      console.log("Новину додано:", data);
    })
    .catch((error) => {
      console.error("Помилка під час додавання новини:", error);
    });
}

function delete_news() {
  const news_id = document.getElementById("input_delete").value;
  fetch(`http://localhost:3000/new/${news_id}`, {
    method: "DELETE",
  })
    .then((response) => {
      if (!response.ok) {
        throw new Error("Помилка видалення новини");
      }
      cancel();
      console.log("Новину успішно видалено");
    })
    .catch((error) => {
      console.error("Помилка видалення новини:", error);
    });
}

function update_news() {
  const newNews = {
    name: document.getElementById("nameu").value,
    theme: document.getElementById("themeu").value,
    text: document.getElementById("textu").value,
    date: document.getElementById("dateu").value,
```

```
    author: document.getElementById("authoru").value,
  };
  const newsId = document.getElementById("idu").value;
  fetch(`http://localhost:3000/new/${newsId}`, {
    method: "PUT",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newNews),
  })
    .then((response) => response.json())
    .then((data) => {
      cancel();
      console.log("Новину оновлено:", data);
    })
    .catch((error) => {
      console.error("Помилка під час оновлення новини:", error);
    });
}
```

Додаток В (frontend)

Файл index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>News channel</title>
    <script src="logic.js" defer></script>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
    />
  </head>
  <body>
    <div id="main">
      <h1>News</h1>
      <button
        type="button"
        class="btn btn-outline-success"
        id="main_page"
        onclick="main()"
      >
        See all news
      </button>
    </div>

    <div id="prime" class="container" style="display: none">
      <h1>News channel</h1>
      <div class="row_news" id="row_news"></div>
    </div>

    <div id="CRUD" style="display: none">
      <button
        type="button"
```

```
  class="btn btn-outline-danger"
  id="news_handling"
  onclick="admin_auth()"
>
  News handling
</button>
<div style="display: none" id="Enter">
  <label>Enter the admin password:</label>
  <input
    class="form-control"
    type="password"
    name="Password"
    id="pass"
  /><!--Password: 66-->
</div>

<div id="operations" style="display: none">
  <div id="addNews">
    <button
      type="button"
      class="btn btn-outline-primary"
      id="addbtn"
      onclick="addNews()"
    >
      Add news
    </button>
    <div id="add" style="display: none">
      <label style="display: block" for="">Enter info to add news</label>
      <label for="">Name:</label>
      <input id="namea" class="form-control" type="text" />
      <label for="">Theme:</label>
      <input id="themea" class="form-control" type="text" />
      <label for="">Text:</label>
      <textarea style="display: block" id="texta" rows="4" cols="50">
      </textarea>
      <label for="">Author:</label>
```

```
    <input id="authora" class="form-control" type="text" />
    <label for="">Date:</label>
    <input id="datea" class="form-control" type="text" />
    <button
      type="button"
      class="btn btn-outline-success"
      onclick="add_news()"
    >
      Add
    </button>
  </div>
</div>
<div id="deleteNews">
  <button
    type="button"
    class="btn btn-outline-danger"
    id="deletebtn"
    onclick="deleteNews()"
  >
    Delete news
  </button>
  <div id="del" style="display: none">
    <label for="">Enter id to delete news:</label>
    <input
      id="input_delete"
      class="form-control"
      type="text"
      name="delete"
    />
    <button
      type="button"
      class="btn btn-outline-danger"
      id="deletebtn"
      onclick="delete_news()"
    >
      Delete
```

```
    </button>
  </div>
</div>
<div id="updateNews">
  <button
    type="button"
    class="btn btn-outline-success"
    id="updatebtn"
    onclick="updateNews()"
  >
    Update news
  </button>

  <div id="update" style="display: none">
    <label style="display: block" for=""
      >Enter info to update news</label
    >
    <label for="">ID:</label>
    <input id="idu" class="form-control" type="text" />
    <label for="">Name:</label>
    <input id="nameu" class="form-control" type="text" />
    <label for="">Theme:</label>
    <input id="themeu" class="form-control" type="text" />
    <label for="">Text:</label>
    <textarea style="display: block" id="textu" rows="4" cols="50">
    </textarea>
    <label for="">Author:</label>
    <input id="authoru" class="form-control" type="text" />
    <label for="">Date:</label>
    <input id="dateu" class="form-control" type="text" />
    <button
      type="button"
      class="btn btn-outline-success"
      onclick="update_news()"
    >
      Update
```

```
        </button>
      </div>
    </div>
  </div>

  <button
    type="button"
    class="btn btn-outline-success"
    id="continue"
    onclick="crud()"
    style="display: none"
  >
    Continue
  </button>

  <button
    style="display: none"
    type="button"
    class="btn btn-outline-danger"
    id="cancel"
    onclick="cancel()"
  >
    Cancel
  </button>
</div>

<div id="new" style="display: none">
  <h2 id="name">Name</h2>
  <h4 id="theme">Theme</h4>
  <div id="text">Text</div>
  <div id="author">Author</div>
  <div id="date">Date</div>
  <div id="id">Id</div>
</div>

<div>
```

```
      <button
        class="btn btn-primary"
        style="display: none"
        id="return_button"
        onclick="returnToMainPage()"
      >
        Return to main page
      </button>
    </div>
  </body>
</html>
```

Додаток Г

Індивідуальне завдання до курсової роботи

«___» _____ 2024 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студенту факультету інформатики спеціальності «Комп’ютерні науки» Максімовіч Давиду Матійовичу

ТЕМА: Побудова багаторівневого веб-застосунку на платформі Google Cloud Platform

Вихідні дані:

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Анотація

Вступ

1 Огляд можливих рішень

2 Структурна розробка веб-застосунків

3 Розробка власного рішення

Висновки

Додатки

Список використаної літератури

Дата видачі «_____» _____ 2024 р.

Керівник Черкасов Д. І. _____

(підпис)

Завдання отримав _____

Додаток Ґ

Календарний план виконання курсової роботи

Тема: Побудова багаторівневого веб-застосунку на платформі Google Cloud Platform

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Приміт ка
1.	Отримання завдання на курсову роботу	30.10.2023	
2.	Ознайомлення з предметною областю	29.12.2023	
3.	Пошук літератури та написання першого розділу	10.03.2024	
4.	Написання другого розділу	25.03.2024	
5.	Початок виконання практичної частини	10.04.2024	
6.	Написання третього розділу	15.04.2024	
7.	Створення презентації	07.05.2024	
8.	Корегування роботи	08.05.2024	
9.	Здача роботи на перевірку на плагіат		
10.	Захист курсової роботи	20.05.2024-21.05.2024	

Студент **Максімовіч Д. М.**

Керівник **Черкасов Д. І.**

“_____”_____