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

ПОБУДОВА БАГАТОРІВНЕВОГО ВЕБ-ЗАСТОСУВАННЯ НА ПЛАТФОРМІ GOOGLE CLOUD PLATFORM (GCP)

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення»

Керівник курсової роботи
к.т.н., стар.викл. Черкасов Д.І

Виконав студент 3-го курсу
Ванін Д. О.

Київ 2021

ЗМІСТ

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
проф., д.ф.-м.н.
_____А. М. Глибовець
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту 3-го курсу факультету інформатики спеціальності «Інженерія програмного забезпечення» Ваніну Данилу Олеговичу

Вихідні дані:
Зміст ТЧ до курсової роботи:

- Індивідуальне завдання
- Календарний план
- Анотація
- Вступ
- Розділ 1.Огляд існуючих рішень
- Розділ 2. Особливості та можливості платформи GCP
- Розділ 3. Структурна розробка власного рішення
- Розділ 4. Детальна розробка компонента
- Висновки
- Список літератури

Дата видачі «___» _____ 2021 р.
Керівник Черкасов Д. І. _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план

Тема: Побудова багаторівневого веб-застосування на платформі Google Cloud Platform (GCP)

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2020р.	
2.	Огляд літератури за темою роботи	листопад-грудень 2020р.	
3.	Огляд існуючих рішень	січень 2021р.	
4.	Структурна розробка власного рішення	лютий 2021р.	
5.	Створення практичної частини роботи	лютень-квітень 2021р.	
6.	Написання текстової частини роботи	квітень-травень 2021р.	
6.	Надання роботи керівнику на перевірку	травень 2021р.	
7.	Корегування роботи згідно з результатами перевірки керівником	травень 2021р.	
8.	Створення та оформлення доповіді	травень 2021р.	
9.	Захист курсової роботи	травень 2021р.	

Студент _____

Керівник _____

“____“ _____ _____ р.

Анотація

Метою роботи розробка багаторівневого веб-застосування на хмарній платформі Google Cloud Platform, який би відповідав критеріям оцінки якості сучасних застосунків. Розроблений застосунок реалізує функції інформаційного ресурсу для музикантів.

У роботі аналізуються та порівнюються різні архітектурні рішення для веб-застосунку та можливі платформи для його розміщення. У першому розділі розглянуті вимоги до сучасних програмних проектів, різні моделі розгортання застосувань та варіанти побудови архітектури. У другому розділі описані можливості платформи GCP для вирішення задач роботи та розглядаються використані продукти платформи та їх переваги. У третьому розділі наведені результати структурної розробки застосунку. Четвертий розділ присвячений детальній розробці застосунку.

Вступ

Сучасні мережеві застосунки на етапі планування, розробки та використання мають відповідати багатьом критеріям. Серед них можна виділити відмовостійкість, оптимальне розміщення апаратних компонентів , збалансованість навантаження, можливість гнучкого масштабування, інтеграцію з ефективними процесами розробки застосунків («неперервна розробка/інтеграція»). Окрім того, сучасні програмні системи потребують моніторингу свого стану, процедур автоматизованого початкового розгортання та оновлення, забезпечувати авторизований безпечний доступ різним категоріям користувачів та бути зручними для подальшого розширення функціоналу чи сервісів. Відповідність застосунку більшості з перелічених критеріїв можуть забезпечити хмарні платформи, які пропонують численні засоби та інструменти для кожного етапу життєвого циклу програмного мережевого застосунку.

Однією з таких платформ є Google Cloud Platform, яка використовується зокрема для роботи інших продуктів компанії, як Google Search, YouTube та Gmail.

Роботу присвячено створенню багаторівневого веб-застосунку, який реалізуватиме функції інформаційного ресурсу для музикантів (бібліотека текстів пісень з акордами та статей на музичну тематику). У якості базової платформи використовується Google Cloud Platform.

Основними вимогами до застосунку є наступні:

- гнучка масштабованість в залежності від навантаження на застосунок
- оптимальне розподілення навантаження на обчислювальні компоненти

- моніторинг поточного стану веб-застосунку та генерація повідомлень щодо подій, які потребують уваги
- багаторівнева архітектура
- відмовостійкість
- підтримка автоматизованого розгортання застосунку та додавання нових компонентів
- інтеграція з процесом неперервної розробки/інтеграції

Хмарні платформи стають дедалі більш поширеною базою для мережевих веб-застосувань та перебувають у стані постійного розвитку функціонала. Переваги хмарних сервісів та можливість постійного розширення сфери їх використання завдяки додаванню нових продуктів дозволяють вважати розроблене застосування та проведене під час розробки дослідження роботи таких сервісів актуальними та корисними для практичних проектів.

1. Огляд існуючих рішень

1.1 Огляд вимог до розробки сучасних застосунків

Із розвитком програмної індустрії з’являється нове розуміння про те, яким чином мають бути побудовані процеси розробки, налагодження та розгортання рішень. Разом із новим розумінням з’являються нові вимоги та відповідно нові інструменти, які мають допомогти ці вимоги задовольнити. Серед таких сучасних умов **є відповідність практиці неперервної інтеграції та розгортанню**, яка дозволяє швидко та автоматизовано перевіряти якість коду та розгортати застосунок у відповідному середовищі. Для покращення параметрів системи та зменшення операційних та капітальних витрат важливою є **гнучка масштабованість**, тобто можливість за потреби збільшувати чи зменшувати кількість елементів системи. Окрім того, надлишковість компонентів, яку легко забезпечити в масштабованій системі забезпечують більш надійну роботу усієї системи. Успішність масштабованості тісно пов’язана з розвитком технології контейнерів та появою зручних інструментів для роботи як з окремим контейнером, так і з багатьма (оркестрація). Враховуючи можливість швидко запускати нові елементи системи більша увага починає приділятися вільно (loosely) зв’язаним системам з багатьма компонентами. Такі системи легше розроблювати, адже є можливість розподілити логіку системи між різними компонентами, а отже і розподілити розробку усієї системи між маленькими командами усередині компанії. Невеликий компонент легше додати до системи, змінювати, тестувати, відстежувати поведінку та розгортати у кінцевому середовищі. Проблеми взаємодії між багатьма компонентами різних типів також вирішуються завдяки балансувальникам навантаження,

окремим підмережам для різних функціональних компонентів і спрощеною роботою з мережею на хмарних платформах. У розподілених системах стає ще важливішим **питання моніторингу** та відслідковування поточного стану та дій, які відбуваються у системі, а тому логування у компонентах та його аналіз також посідає важливе місце у сучасних застосунках. Як завжди актуальним є питання швидкодії та зручності застосування для користувача, що стало ще легшим завдяки новим популярним моделям розгортання застосунку, які підтримують розміщення компонентів у різних частинах світу (ближче до кінцевого користувача), глобальне кешування та системи доставки контенту. Окремим пунктом є **безпека**. Безпеку від зовнішніх загроз забезпечує розподілення компонентів застосунку в окрему закриту систему, з використанням NAT та Firewall на межах виходу в глобальну мережу. Внутрішню безпеку підтримує жорсткий контроль доступу та можливості користуватися ресурсами, а також захищені можливості під’єднання до внутрішньої системи.

1.1.1 Неперервна розробка/інтеграція

Задля покращення якості програмного коду та забезпечення перевірки його коректної роботи ще до початку розгортання більшість сучасних сервісів для спільної розробки та керування версіями (як Github, Gitlub, Bitbucket), пропонують конвеєри(pipelines), які частково або повністю реалізують практику неперервної інтеграції та неперервного розгортання. Перед тим, як поєднувати зміни, новий код перевіряється на відповідність заданим умовам, як наприклад відповідність встановленому форматуванню та виконання усіх модульних тестів. Після вдалого проходження початкового тестування користувач, який має права на поєднання гілок може підтвердити зміни. Після цього запускається наступний етап конвеєру, наприклад розгортання

оновленого застосунку в тестовому середовищі. Після злиття з якоюсь з гілок (наприклад development, stage, production) автоматично викликається процес розгортання оновленого застосунку у відповідному середовищі. Це дозволяє провести інтеграційне тестування та тестування всієї системи з оновленим компонентом. Після успішного проходження і цього етапу тестування, зміни вносяться до гілок, які відповідають за наступне у процесі розгортання середовище. Після проходження усіх тестів та можливо додаткового підтвердження від керівника проекту застосунок автоматично розгортається у фінальному середовищі. Неперервна розробка передбачає автоматизацію тестування та розгортання застосунку у різних середовищах, після додавання нового коду до системи контролю версій.

1.1.2 Гнучка масштабованість

Масштабування – це процес збільшення або зменшення кількості ресурсів застосунку в залежності від потреб чи навантаження. Масштабованість системи прямо впливає на якість обслуговування та вартість використання ресурсів для компанії. За відсутності можливості гнучко змінювати кількість ресурсів великі навантаження на систему приводять до її сповільненої роботи. В свій час затримка всього у півсекунди для Google приводить до зниження трафіку на 20% [2]. Розрізняють вертикальне і горизонтальне масштабування. Вертикальне відповідає за якісне збільшення показників пам’яті та потужності окремих компонентів, але воно обмежене максимальними можливостями конкретного фізичного засобу. Таке масштабування може коштувати дорого, не мати можливості швидко зменшити ресурси за потреби та включати певний час простою. Окрім того, у випадку відмови одного компоненту, який був масштабований вертикально у системі з’являється серйозна проблема, адже швидко замінити такий дорогий

та складний компонент неможливо. У свою чергу горизонтальне масштабування забезпечується додаванням чи видаленням нових компонентів системи. Воно є ефективним у випадку коли масштабовані компоненти є взаємозамінними та існують автоматизовані процеси збільшення та зменшення їх кількості. Масштабованість дозволяє покращити показники ефективності та можливого навантаження системи, а також є основою відмовостійкістю системи, адже забезпечує надлишковість компонентів.

1.1.3 Розподілення навантаження

У розподілених системах з горизонтальною масштабованістю та відмовостійкістю за рахунок надлишковості є сенс розподіляти навантаження між компонентами системи. Таке розподілення найчастіше відбувається ще на рівні трафіку, де балансувальник навантаження рівномірно розподіляє навантаження між вузлами. Відповідно кожен з вузлів опрацьовує лише частку запитів і несе лише частку загального навантаження. Метою є уникнення перевантаження окремих компонентів та забезпечення високої доступності.

1.1.4 Моніторинг

У розподілених системах з великою кількістю незалежних компонентів стає складніше виявляти конкретні помилки чи проблеми застосунку. Моніторинг дозволяє збирати метрики, які відповідають за ефективність, доступність та якість роботи застосунку та інфраструктури. Ці метрики включають дані логування, результати запитів для перевірки стану застосунку, інформацію про обмін даними між елементами системи. Колекція вчасно зібраних даних дозволяє адаптувати систему і застосунок, відстежувати поточний стан та причину помилок, а також швидко реагувати на зміни.

1.1.5 Безпека

Для сучасних застосунків важливо підтримувати високий рівень безпеки на різних рівнях. Надійність та відмовостійкість застосування в цілому та окремих компонентів є важливим як для коректної роботи системи, так і для окремого користувача. Обмежений та контрольований доступ до ресурсів розробників та членів команди зменшують вірогідність помилок та забезпечують краще розподілення сфер відповідальності. Захист від зовнішніх загроз встановлений на межі зєдання локальної мережі застосунку та інтернету дозволяє зменшити вплив атак і шкідливих дій на ефективність роботи застосунку та забезпечують кращий захист особистих даних користувачів.

1.2 Порівняння архітектури мережевих застосувань

З часом типова архітектура програмного застосунку пройшла шлях від одного рівня до багатьох. Кожен з цих етапів доповнював попередній і має певні переваги та недоліки. У кожній архітектурі функціонально існує як мінімум три рівні: передній план (інтерфейс доступу користувача, frontend), бізнес-логіка (проміжний рівень, middleware) та рівень даних (бази даних). В загальному зі збільшенням кількості рівнів покращуєтеся розподіленість системи, а отже з’являється більше можливостей для масштабування, розподілення відповідальності та розробки, забезпечення ефективності та відмовостійкості. Водночас ускладнюється керування процесом оновлення та розгортання окремих компонентів, а також їх мережева взаємодія.

1.2.1 Однорівнева архітектура

У випадку однорівневої архітектури усі функціональні компоненти системи знаходяться на єдиному сервері. За цієї архітектури зручно контролювати усі

процеси, які відбуваються в програмі, але існують значні проблеми з відмовостійкістю, адже вся інформація застосунку знаходиться на одній машині. Окрім того, ця архітектура не масштабується і значно ускладнює розробку у випадку, коли над проектом працює багато команд розробників. У такому випадку багато часу витрачатиметься на злиття змін та погодження їх з іншими командами. Важливим недоліком є складність процесу оновлення застосунку, адже за завантаження коректної версії застосунку відповідає сам користувач. Розподілення та масштабування також ускладнюються, неможливо окремо масштабувати певний функціональний компонент, якого не вистачає, адже з ним архітектурно пов’язані всі інші.

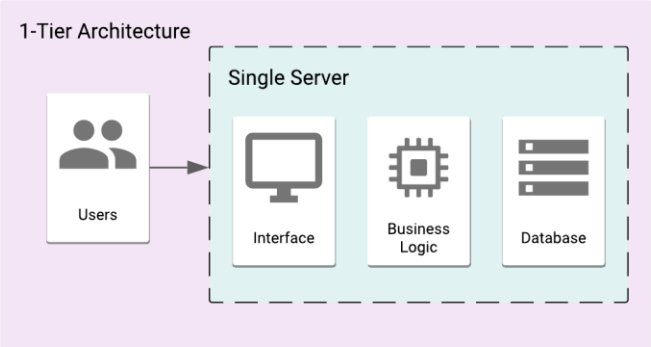


Рисунок 1.1 – Приклад схеми застосунку з однорівневою архітектурою

1.2.2 Дворівнева архітектура

Дворівнева архітектура являє собою просту клієнт-серверну модель застосунку, при якій клієнт найчастіше бере на себе рівень представлення та інтерфейсу, а за даними звертається до окремого сервера. В залежності від того, хто виконує більшість складної бізнес-логіки розрізняють архітектуру з «товстим» і «тонким» клієнтом. Така архітектура підходить для невеликих застосунків, основною метою яких є нескладна робота з даними. Вона

забезпечує трохи більше свободи у розробці, через наявність розподілених компонентів.

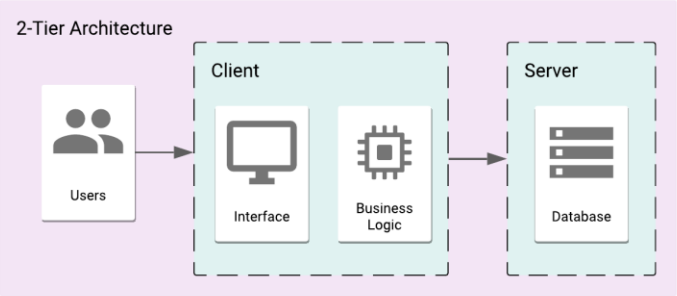


Рисунок 1.2 - Приклад схеми застосунку з дворівневою архітектурою

1.2.3 Трирівнева архітектура

Трирівнева архітектура складається з трьох згаданих рівнів – представлення, бізнес-логіки та даних, розміщених відділених один від одного. Це типова модель більшості веб-застосунків, яка дозволяє відділити дані від представлення, гнучко масштабувати окремі рівні, за потреби забезпечуючи збільшення продуктивності, відмовостійкість, поступове оновлення версій кожного з рівнів та оптимальне територіальне розміщення. Зі збільшенням кількості рівнів система стає складнішою та більше уваги приділяється безпечній передачі даних по мережі.

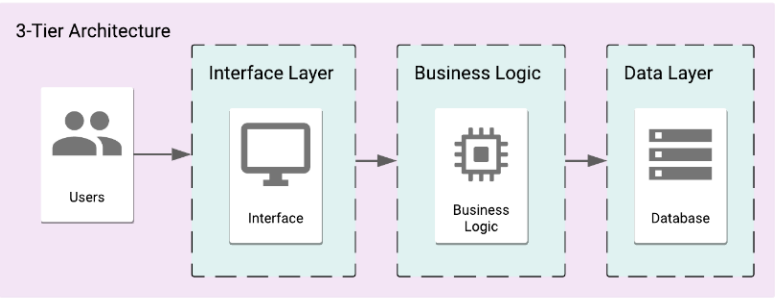


Рисунок 1.3 - Приклад схеми застосунку з трирівневою архітектурою

1.2.4 Багаторівнева архітектура

Системи з багаторівневою архітектурою отримують додаткові рівні порівняно з трирівневою архітектурою за рахунок розподілення рівня бізнес-логіки на декілька функціональних компонентів або за рахунок додавання проміжних рівнів обробки – наприклад елементів, які будуть відповідати за фільтрацію запитів, захист, балансування навантаження, кешування відповідей, адміністративний доступ.

1.2.5 Мікросервісна архітектура

Мікросервісна архітектура є прикладом архітектури створеної для використання переваг горизонтальної масштабованості та розподіленості. Мікросервісне застосування розкладається на множину дрібніших сервісів залежно від функціональності. Взаємодія між мікросервісами відбувається через мережу, кожен окремий сервіс є самодостатнім, може оновлюватися та змінюватися незалежно від інших. Така архітектура дозволяє розподілити логіку розробки між багатьма компонентами та командами розробників та пришвидшити випуск продукту. При цьому ускладнюється мережева структура, більше уваги має бути приділено моніторингу, автоматизації розгортання мікросервісів та тестуванню. Велика частина роботи покладається на DevOps фахівців.

1.3 Типи інфраструктур для розгортання застосувань

1.3.1 Власні ресурси володаря застосувань (On-Premise)

У випадку використання ресурсів власника застосування, усі процеси від розробки до розгортання проекту відбуваються внутрішньо. Так само власник повністю відповідає за безпеку, оновлення, технічне обслуговування програмного застосунку. Кошти за роботу та підтримку системи на власних

ресурсах часто будуть вищими ніж на хмарній платформі. З іншого боку, така інфраструктура пропонує потенційно найвищий рівень безпеки, адже власнику повністю підконтрольні усі апаратні ресурси і дані не передаються між датацентрами.

1.3.2 Хмарна платформа (Cloud)

За умови використання хмарної платформи для розробки та розгортання застосування велика частина відповідальності перекладається на поставника хмарних послуг. Хмарна платформа регулює мережу, апаратне та програмне забезпечення, ліцензії, безпеку та керування ресурсами самостійно. Велика кількість часто використовуваних елементів пропонується хмарним провайдером у якості сервісів, що значно полегшує процес розробки та розгортання. Окрім того, хмарні платформи беруть кошти лише за використовувані ресурси і дозволяють як швидко збільшити кількість та потужність системи, так і зменшити.

1.3.3 Порівняння інфраструктур

У таблиці 1.1 наведено порівняння розміщення застосунку на власних ресурсах та хмарній платформі.

Критерій оцінки	**Власні ресурси (On Premise)**	**Хмарна платформа (Cloud)**
Розгортання	власник відповідає за розгортання, технічне обслуговування та забезпечення потрібною інфраструктурою	хмарний провайдер забезпечує усі процеси пов’язані з інфраструктурою та всі потрібні ресурси за запитом
Контроль	власник повністю контролює систему та може забезпечити повну приватність даних	дані застосунку зберігаються на розподілених ресурсах та не дивлячись на високий рівень надійності можуть

		бути не доступні у випадку неочікуваних ситуацій
Вартість	власник платить за приміщення, витрати електроенергії, ремонт та обслуговування апаратних засобів та необхідне програмне забезпечення, можливі високі операційні та капітальні витрати	плата залежить від використовуваних ресурсів, не вимагає капітальних витрат, краще підходить для малих проектів
Безпека	розміщення даних на ресурсах власника може бути безпечнішим, за умови виконання певних заходів щодо забезпечення цієї безпеки	дані розташовані на спільних ресурсах, хоча інші сторони не мають доступу до чужих ресурсів, при обході безпеки провайдера вразливі дані застосунку та користувачів можуть бути під загрозою
Налаштовуваність	дані користувачів у деяких застосунках підпадають під нормативний контроль, а отже вимагають високий рівень безпеки, який легше забезпечити на ресурса власника	компанії мають перевіряти наскільки рівень безпеки конкретного провайдера відповідає нормативнивним регулюванням для їх застосунку
Мобільність	для безпечного віддаленого доступу потрібні відповідні налаштування та наявність сторонньої допомоги для підключення робітника дистанційно	швидкий доступ через мережу та можливість отримання послуги в довільному зручному для користувача місці
Масштабованість	масштабованість та її швидкість залежить від власника, при цьому потрібно автоматизувати чи мати окремих спеціалістів, які б відповідали за масштабування застосунку	самообслуговування по вимозі, масштабованість та еластична зміна ресурсів залежно від навантаження
Додаткові можливості	можливість фізичного доступу до апаратної частини та даних	велика кількість сервісів від провайдера, які полегшують розробку

Таблиця 1.1 – Порівняння розміщення застосунку на власних ресурсах та хмарній платформі

1.3.4 Висновок

Розміщення застосунку на власних ресурсах і на ресурсах хмарного провайдера мають свої переваги та недоліки. Зокрема розміщення даних у власника дозволяють мати вищий рівень безпеки, а хмарний провайдер

забезпечить менші витрати для невеликих проектів та зручнішу масштабованість. Варто також згадати, що інфраструктура для певного проекту може поєднувати власні та хмарні ресурси, створюючи гібридну хмару. Це дозволяє поєднати масштабованість та безпеку, але призводить до додаткової складності у керуванні системою.

Для невеликого сучасного багаторівневого застосунку без чутливих даних розміщення ресурсів на хмарному провайдері є оптимальним.

2. Особливості та можливості платформи GCP

2.1 Особливості хмарних платформ

Хмарні платформи задовольняють певні характеристики, які роблять розробку застосунків для цих платформ можливою та зручною [2]:

- через наявність нескінченних ресурсів (або ілюзію їх наявності) та обмежену апаратну можливість окремих віртуальних машин застосування для хмарних платформ націлене на горизонтальну масштабованість
- через націленість на короткострокову аренду ресурсів, хмарні платформи дозволяють легко звільняти та отримувати нові ресурси
- завдяки моделі оплати тільки за те, що використовується, хмарні застосунки витрачають тільки кошти, які потрібні для їх безпосередньої роботи, а інформація про всі витрати є доступною та прозорою
- орієнтовані на самообслуговування за потребою, автоматичне налаштування та виділення ресурсів, хмарні платформи дозволяють просте автоматичне масштабування
- хмарні застосунки оптимізовані для витрати коштів, а не надійності; проблеми та зупинка роботи окремих компонентів є очікуваною, але завдяки надлишковості час простою рідко стаж проблемою
- широка екосистема різноманітних сервісів, які пропонує платформа (збереження даних, черги повідомлень, віртуалізація серверів та мережевих функцій) розробка застосунків значно спрощується

2.2 Короткий огляд платформи GCP

Google Cloud Platform – набір хмарних служб від компанії Google, запущений у 2013 році. Входить до трьох найпопулярніших хмарних провайдерів наряду з AWS (Amazon Web Services) та Microsoft Azure. Хмарна платформа пропонує понад 100 сервісів у таких категоріях як Хмарні обчислення, Зберігання даних, Мережеві сервіси, Великі дані, Штучний інтелект і тд. Для нових користувачів, які підлючають до платформи кредитну картку платформа пропонує безкоштовний бюджет у 300 долларів на період трьох місяців, а тому є зручною для невеликих дослідницьких і навчальних проектів.

2.3 Використані продукти

Для розробки багаторівневого веб-застосування та у рамках дослідження хмарної платформи були використані наступні сервіси та продукти GCP.

2.3.1 Cloud Run

«Безсерверний» сервіс, який дозволяє швидко розгорнути додаток з готового Docker-образу або прямо з репозиторію, у випадку, якщо застосування написане на одній з підтримуваних мов. Цей сервіс побудований з Knative і підтримує автоматичне масштабування у залежності від навантаження на систему. Цей сервіс повністю інтегрований з іншими продуктами платформи, як CloudCode, CloudBuild, CloudMonitoring, CloudLogging.

2.3.2 Cloud CDN

Сервіс, який реалізує мережу розповсюдження контенту і являє собою розподілену мережеву інфраструктуру, яка автоматично надсилає потрібні клієнту файли з найближчого до нього місця доступу та кешує дані. Цей

сервіс дозволяє пришвидшити доставку контенту кінцевому користувачу і легко інтегрується з іншими сервісами платформи.

2.3.3 Cloud Load Balancing

Сервіс, який може реалізувати як зовнішнє так і внутрішнє балансування навантаження. Дозволяє фільтрувати та балансувати запити залежно від протоколів повідомлень, перевірки на відповідність регулярним виразам http-адрес, вхідних адрес. Відбалансовані запити можуть направлятися на різні кінцеві ресурси, а сам сервіс балансування може автоматично масштабуватися в залежності від навантаження на нього.

2.3.4 Cloud Storage

Хмарне сховище даних з необмеженим обсягом, високою безпекою даних та можливістю розміщення у декількох регіонах для підвищення відмовостійкості та зниження часу доступу для клієнтів. Має декілька класів сховищ, які відрізняються вартістю послуг в залежності від регулярності доступу до даних, часу зберігання та наявності резервного копіювання.

2.3.5 Cloud Build

«Безсерверний» сервіс, який дозволяє будувати, тестувати, розгортати застосунки і налаштовувати процес неперервної розробки та інтеграції. Має можливість використання з найпопулярнішими сервісами для контролю версій (Github, Bitbucket, Gitlab). Автоматично масштабується в залежності від кількості процесів, які має виконувати, та витрачає кошти лише під час роботи.

2.3.6 IAM (Identity and Access Management)

Сервіс, який дозволяє обмежувати та контролювати доступ до ресурсів платформи, як живих користувачів, так і застосунки. Інтегрований зі

структурою проектів на Google Cloud Platform і є безкоштовним для користувачів платформи.

2.3.7 Cloud Armor

Сервіс, який забезпечує зовнішню безпеку від розподілених атак на відмову в обслуговуванні(DDoS) та 10 іншим вразливостям визначеним Відкритим проектом з безпеки веб-застосунків (OWASP). Дозволяє фільтрувати доступ до ресурсів в залежності від географічного положення та адреси клієнта.

2.3.8 Container Registry

Приватний Docker-репозиторій з розподіленим розміщенням та автоматичним пошуком вразливостей у образах, які зберігаються на сервісі. Підтримує інтеграцію з багатьма сервісами Google Cloud Platform, завдяки чому спрощується процес розгортання застосувань.

2.3.9 Logging

Сервіс, який відповідає за збирання, аналіз, пошук по даним логувань застосунків. Дозволяє встановлювати сповіщення на визначені події логування, виконувати складні пошукові запити по логованим даним та зберігати дані логувань у інших сервісах платформи.

2.3.10 Cloud Monitoring

Сервіс, який дозволяє візуалізувати дані про роботу застосунків та логувань, контролювати відповідність показників SLI заданим SLO та підтримує інтеграцію з месенджерами та гібридними або мультихмарними застосунками.

2.3.11 Serverless VPC Access

Сервіс, який вирішує проблему доступу безсерверних компонентів до ресурсів, які знаходяться усередині VPC(Virtual Private Cloud) та не мають

доступної ззовні адреси. Сервіс під’єднується до приватної хмари і перенаправляє запити з безсерверних компонентів.

2.3.12 App Engine

“Безсерверний” сервіс, який дозволяє розгортати застосунки на найпопулярніших мовах програмування або розгортати власні контейнери. Автоматично масштабується в залежності від навантаження, підтримує A/B тестування та поступове оновлення коду.

3. Структурна розробка власного рішення

3.1 Архітектура рішення

Розроблене рішення на хмарній платформі логічно поділене на дві частини. Перша – власне застосунок, відповідає за обробку запитів клієнта та забезпечення роботи системи. Вона складається з сервісів обробки, фільтрації та перенаправлення запитів, хмарного сховища, де зберігаються статичні файли застосунку та трьох рівнів застосунку – рівню представлення, бізнес логіки та даних. Друга частина складається з сервісів, які забезпечують неперервну розробку та інтеграцію застосунку, контроль доступу та моніторинг стану.

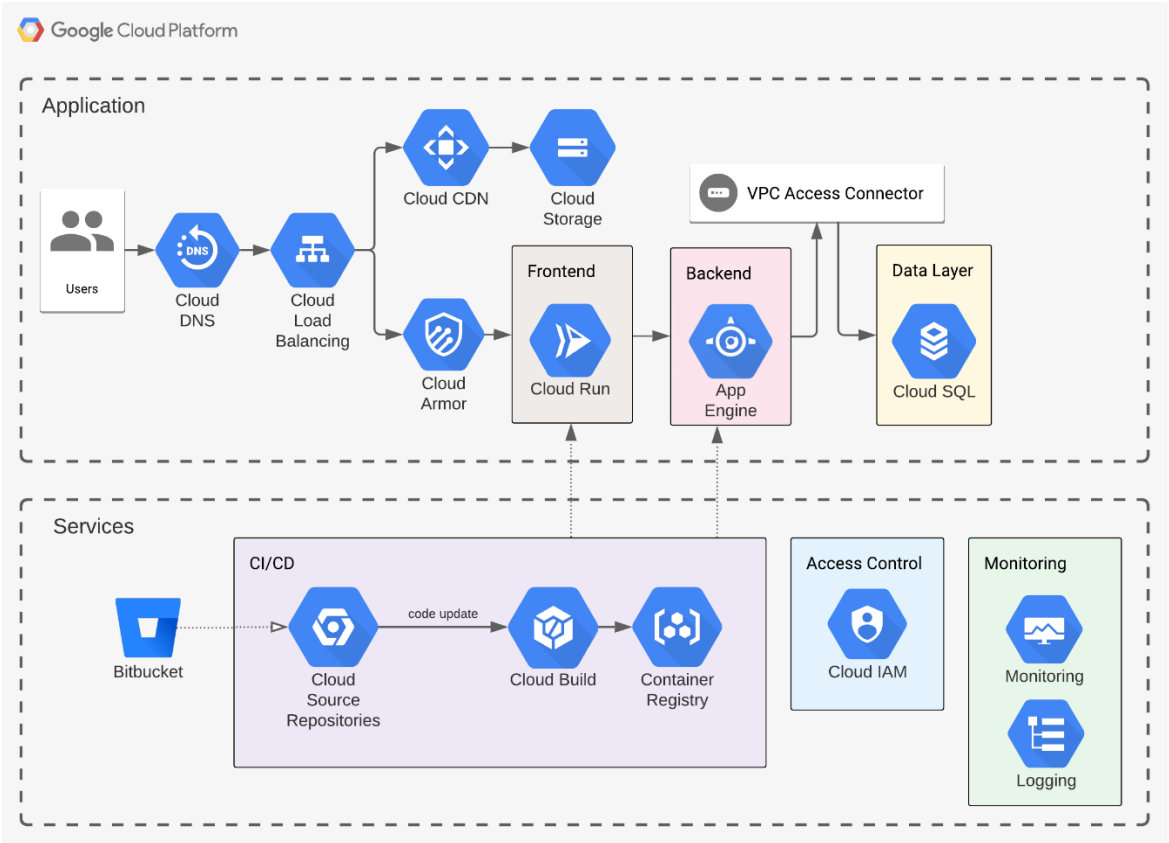


Рисунок 2 - Архітектура застосунку на хмарній платформі

3.2 Робота застосунку

Клієнт доступається до застосунку через зареєстровану у реєстратора доменних імен Namecheap адресу. Окрім DNS серверів реєстратора використовується сервіс Cloud DNS, який дозволяє додавати записи для перекладу доменного імені у адреси та поширювати їх на DNS серверах Google. Запити клієнтів направляються на зовнішній балансувальник навантаження, який аналізує адресу запиту. Адреси, які відповідають статичним файлам направляються на сервіс Cloud CDN, який займається кешуванням цих файлів та отриманням їх з хмарного сховища Cloud Storage, яке має публічний доступ. Запити з іншими адресами балансувальник надсилає на сервіс, де розміщений рівень представлення застосунку. Рівень представлення опрацьовує запити та надсилає клієнту згенеровані HTML сторінки. За додатковими даними рівень представлень звертається до рівню бізнес логіки за зовнішньою адресою. Рівень бізнес логіки доступається до бази даних, розміщеної на сервісі Cloud SQL. База даних не має публічної адреси, а тому для зв’язку рівня бізнес логіки та даних використовується сервіс VPC Access, який перенаправляє зовнішні запити до приватної мережі.

3.2 Робота додаткових сервісів

Додаткові сервіси забезпечують оновлення коду, моніторинг та обмеження доступу розробників. Код застосунку та його версії зберігаються на зовнішньому сервісі Bitbucket. Усі зміни на оригінальному репозиторії повторюються на віддзеркаленому внутрішньому репозиторії на сервісі Cloud Source Repositories. Оновлення хмарного репозиторію викликає процес перевірки та побудови застосунку сервісом Cloud Build, у результаті чого контейнер з відповідним компонентом буде збережений у іншому репозиторії

для контейнерів на сервісі Container Registry. Після цього завантажений контейнер розгортається на відповідному безсерверному сервісі платформи. Моніторинг застосунків відбувається автоматично за допомогою сервісу Cloud Logging та може бути візуалізований іншим сервісом Cloud Monitoring. Контроль доступу забезпечується сервісом IAM, який дозволяє визначати ролі з правами та застосовувати їх до акаунтів розробників та сервісів.

4. Детальна розробка компонента

4.1 Рівень інтерфейсу користувача

Для реалізації рівня представлення була використана мова Typescript з використанням фреймоворку React. Для побудови вихідного коду та проміжного тестування використовується пакувальник Webpack. Такий набір технологій був обраний з огляду на їх популярність та високу частоту використання у сучасних застосунках. Інформація про поточного користувача, обрану мову локалізації та спільний стан застосунку зберігаються та управляються за допомогою інструмента React Redux. Кожна сторінка застосунку складається з декількох React-компонетів, які можна повторно використовувати у подальшій розробці. Запити до серверу бізнес-логіки виконуються за допомогою бібліотеки axios у компонентах позначених reducer. Досвід розробки та розгортання застосунків з цими технологіями буде актуальним і надалі. Застосунок автоматично розгортається на «безсерверному» хмарному сервісі Cloud Run завдяки налаштованій безперервній інтеграції та розгортанню за домогою підключення сервісу CloudBuild до репозиторію BitBucket. Розгортання відбувається за рахунок використання Docker-контейнерів, описаних у відповідному Dockerfile, розміщеному разом з файлами застосунку.

```
# Pull official base image
FROM node:13.12.0-alpine AS builder
# Set working directory
WORKDIR /app
# Add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH
# Install app dependencies
COPY package.json ./
COPY package-lock.json ./
RUN npm install
# Copy app assets
COPY . ./
RUN npm run webapp:prod
# Define environment variables for Cloud Run
ENV PORT 8080
ENV HOST 0.0.0.0
EXPOSE 8080
# Start app
CMD ["node", "./server.js"]
```

Рисунок 3 - Вміст Docker-file для застосунку рівня представлень

4.2 Рівень бізнес логіки

Рівень бізнес логіки був написаний з використанням фреймворку Spring для мови програмування Java. Застосунок автоматично розгортається на сервісі App Engine, не потребує явного файлу з описом контейнеру, але вимагає декларативний файл app.yml для опису додаткових налаштувань сервісу. Для авторизації та аутентифікації запитів використовується Jwt. Підтримка зміни стану бази даних забезпечується бібліотекою Liquibase. Для зміни структури або даних достатньо внести дані про зміну у файл master.xml і при запуску програми база даних автоматично буде змінена. Кожна зміна має свій ідентифікатор, а інформація про виконані зміни зберігаються у базі даних. Таким чином однакові зміни не відбуваються при декількох повторних запусків застосунку. Рівень бізнес логіки визначає прикладний програмний інтерфейс для взаємодії з більшістю сутностей рівня даних. Застосунок архітектурно поділений на рівень моделей, репозиторіїв, сервісів та

контроллерів. Моделі представляються собою реалізацію сутностей бази даних, а репозиторії реалізують запити до баз даних. Сервіси слугують або проміжним рівнем між контролерами та репозиторіями, або містять складну логіку для роботи з декількома репозиторіями. Контролери описують інтерфейс зовнішнього доступу до програми і викликають методи сервісів. Розробка рівня моделей та репозиторіїв значно спрощується завдяки використанню інструментів Hibernate та Lombok, які автоматично генерують стандартний код для доступу до полів класу та методи для виконання часто використовуваних запитів до бази даних. Більшість налаштувань та дані для підключення до баз даних задаються у файлах налаштувань (application.properties або application.yml), але можуть передаватися і як параметри командного рядка. Застосунок містить різні профілі налаштувань для кінцевого середовища та середовища розробки, які також задаються у налаштуваннях. Оскільки сервіс виконує функцію REST API для документації визначений програмний інтерфейс автоматично документується за допомогою OpenAPI та SwaggerUI.

```
# Define runtime
runtime: java11
# Define instance class
instance_class: F4
# Needed for Cloud SQL access
vpc_access_connector:
  name: "projects/toneup-ci-cd/locations/europe-west3/connectors/database-connector"
```

Рисунок 4 - Вміст файлу app.yml для розгортання на сервісі App Engine

4.3 Рівень даних

Рівень даних реалізований за допомогою сервісу Cloud SQL з рушієм PostgreSQL. Сервіс розміщений одночасно у декількох регіонах та автоматично робить копії бази даних, що забезпечує високу доступність та надійність. Схема бази даних складається з 21 сутності, які потрібні для

реалізації початкової версії музичної платформи з статтями та акордами пісень. Для створення надійного серверу з базами даних достатньо обрати режим доступ до сервера (публічна або приватна адреса), дані для доступу, тип розміщення (один або декілька регіонів), рушій бази даних, резервне копіювання та апаратні характеристики сервера.

4.4 Оновлення коду

Безперервна інтеграція та розгортання забезпечуються сервісом Cloud Build, який може перевіряти правильність збирання, запускати модульні тести, будувати образи контейнерів та ініціювати розгортання застосування. Кроки виконання всього процесу описуються у файлах cloudbuild.yml.

```
steps:
# Clean
- name: maven:3-jdk-11
  entrypoint: mvn
  args: ["clean"]
# Build
- name: maven:3-jdk-11
  entrypoint: mvn
  args: ["package", "-Dmaven.test.skip=true"]
# Deploy to App Engine
- name: "gcr.io/cloud-builders/gcloud"
  args: ["app", "deploy"]
timeout: "1600s"
```

Рисунок 5 - Приклад файлу з описом процесу розгортання застосунку сервісом Cloud Build

4.5 Середовища розгортання

Для розгортання застосувань у різних середовищах достатньо створити окремий проект у межах організації, де розгорнути відповідну до середовища архітектуру. На хмарній платформі GCP усі ресурси в межах проекту ізольовані від ресурсів іншого проекту, а тому забезпечується незалежність середовищ. Окрім того IAM дозволяє обмежити доступ до кожного проекту в

залежності від ролі учасника. Так розробник не матиме доступу до кінцевого середовища, але матиме можливість працювати з середовищем для розробки. У ході роботи для перевірки роботи даної системи початково було розгорнуто два середовища – кінцеве і середовище для розробки, але з метою економії коштів у межах навчального проекту було вирішено залишити тільки середовище для розробки.

Висновки

Результатом курсової роботи є багаторівне веб-застосування на платформі Google Cloud Platform. У ході написання роботи вдалося дослідити та порівняти різні типи інфраструктур для розгортання застосувань та архітектур мережевих застосувань. Був проведений їх аналіз відповідно до вимог до розробки сучасних застосувань. Також були досліджені та використані під час розробки застосування багато сервісів хмарної платформи Google Cloud Platform. Розміщення застосунку та окремі реалізація окремого функціоналу системи відбувається за допомогою сервісів Cloud Run, App Engine, Cloud Load Balancing, Cloud Build та інших. Досвід використання цих сервісів є актуальним, адже більшість з них мають аналоги у інших хмарних провайдерів та будуть популярними з огляду на тенденцію розширення використання хмарних платформ. Додатково вдалося покращити навички та отримати досвід використання таких інструментів розробки застосувань як Typescript, Javascript, React, React Redux, Liquibase, Spring, PostgreSQL та Docker.

Застосунок та побудована архітектура демонструє принципи створення багаторівневих застосувань та їх розміщення на хмарній платформі.

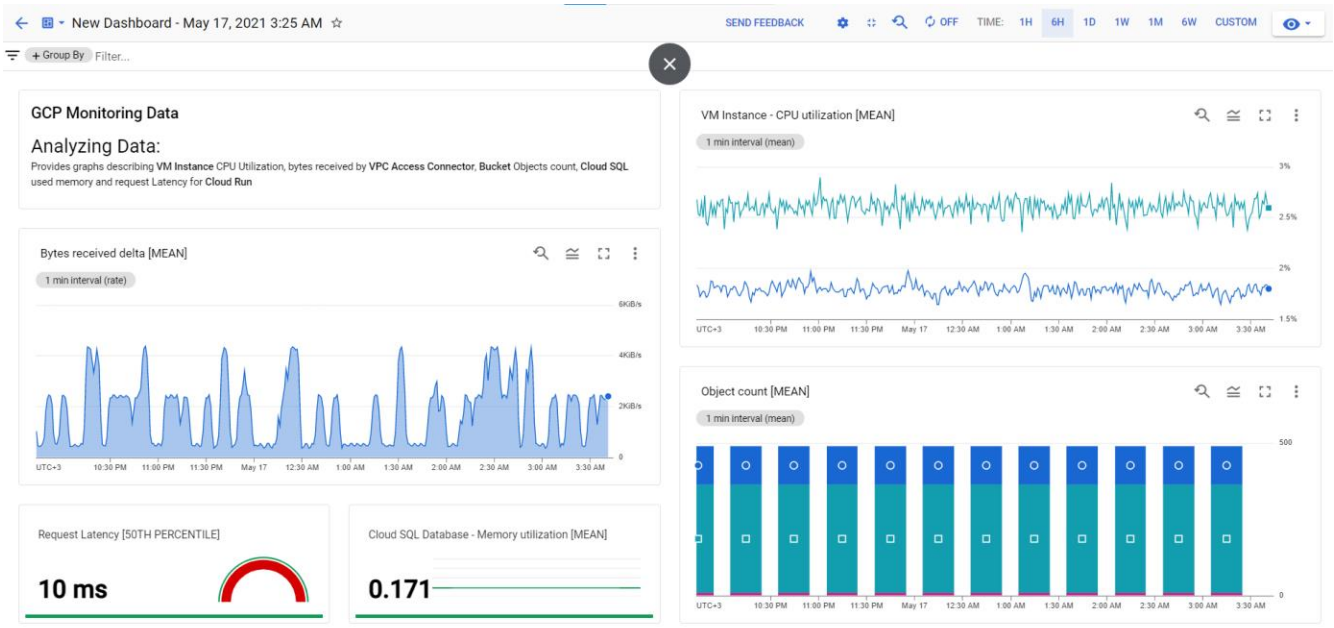
Список літератури

1. GCP Training Labs [Електронний ресурс]. — Режим доступу: https://www.qwiklabs.com/
2. Bill Wilder. Cloud Architecure Patterns — O'Reilly Media, Inc., 2012
3. Site Reliability Engineering: How Google Runs Production Systems / [Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy] — O'Reilly Media, Inc., 2018
4. Content Delivery Network [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Content_delivery_network
5. Навчальний курс Google Cloud Platform Fundamentals: Core Infrastructure [Електронний ресурс]. — Режим доступу: https://www.coursera.org/learn/gcp-fundamentals
6. Стаття «6 ключових відмінностей розміщення на власних ресурсах та на хмарній платформі» [Електронний ресурс]. — Режим доступу: - https://dynamics.folio3.com/blog/on-premise-vs-cloud-erp-software-difference/
7. Knowledge Base On Premise vs. Cloud [Електронний ресурс]. – Режим доступу до ресурсу: https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud
8. Wikipedia GCP Page [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Google_Cloud_Platform
9. Google Cloud Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://cloud.google.com/docs
10. React – a JavaScript framework [Електронний ресурс]. – Режим доступу до ресурсу: https://reactjs.org/
11. Google Cloud Tech YouTube channel - [Електронний ресурс]. – Режим доступу до ресурсу: https://www.youtube.com/user/googlecloudplatform/videos
12. Google Cloud YouTube channel - [Електронний ресурс]. – Режим доступу до ресурсу: https://www.youtube.com/c/googlecloud
13. TypeScript – open-source language build on Javascript [Електронний ресурс]. – Режим доступу до ресурсу: https://www.typescriptlang.org/
14. Baelung – website with Java-related tutorials [Електронний ресурс]. – Режим доступу до ресурсу: https://www.baeldung.com/
15. JHipster – development platform [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jhipster.tech/
16. Project Lombok library [Електронний ресурс]. – Режим доступу до ресурсу: https://projectlombok.org/

17. Hibernate documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://hibernate.org/orm/documentation/5.4/

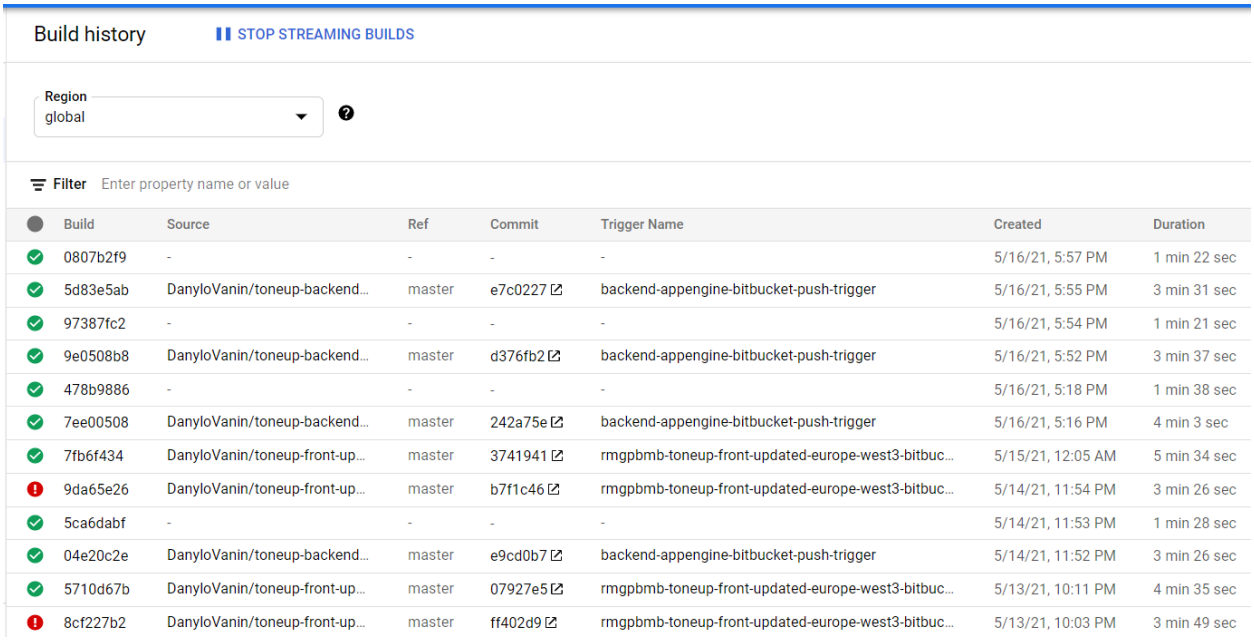
Додаток А

(обов’язковий)
Скріншот налаштованого інтерфейсу моніторингу Cloud Monitoring



Додаток Б

(обов’язковий)
Скріншот історії побудови застосунку сервісом Cloud Build



Build history STOP STREAMING BUILDS

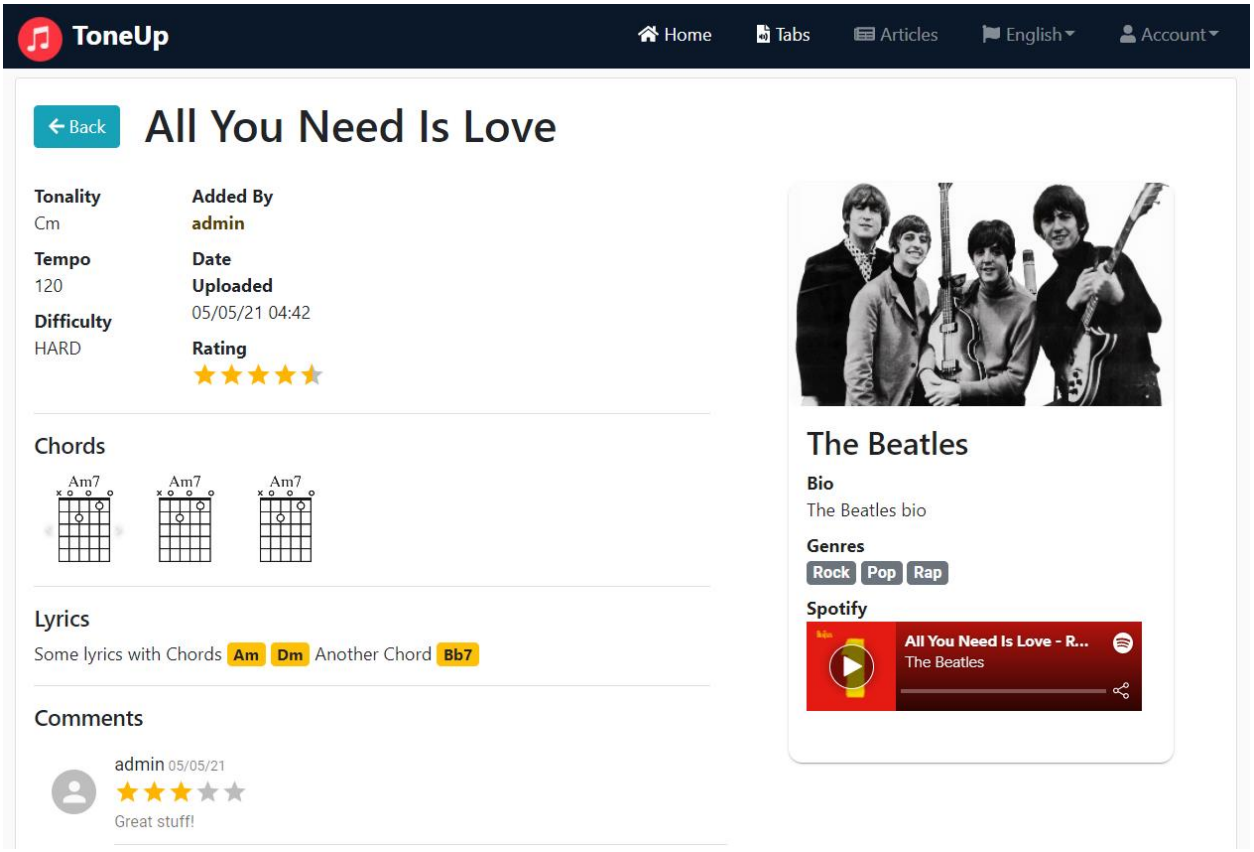
Region: global

Filter: Enter property name or value

Build	Source	Ref	Commit	Trigger Name	Created	Duration
0807b2f9	-	-	-	-	5/16/21, 5:57 PM	1 min 22 sec
5d83e5ab	DanyloVanin/toneup-backend...	master	e7c0227	backend-appengine-bitbucket-push-trigger	5/16/21, 5:55 PM	3 min 31 sec
97387fc2	-	-	-	-	5/16/21, 5:54 PM	1 min 21 sec
9e0508b8	DanyloVanin/toneup-backend...	master	d376fb2	backend-appengine-bitbucket-push-trigger	5/16/21, 5:52 PM	3 min 37 sec
478b9886	-	-	-	-	5/16/21, 5:18 PM	1 min 38 sec
7ee00508	DanyloVanin/toneup-backend...	master	242a75e	backend-appengine-bitbucket-push-trigger	5/16/21, 5:16 PM	4 min 3 sec
7fb6f434	DanyloVanin/toneup-front-up...	master	3741941	rmgpbmb-toneup-front-updated-europe-west3-bitbuc...	5/15/21, 12:05 AM	5 min 34 sec
9da65e26	DanyloVanin/toneup-front-up...	master	b7f1c46	rmgpbmb-toneup-front-updated-europe-west3-bitbuc...	5/14/21, 11:54 PM	3 min 26 sec
5ca6dabf	-	-	-	-	5/14/21, 11:53 PM	1 min 28 sec
04e20c2e	DanyloVanin/toneup-backend...	master	e9cd0b7	backend-appengine-bitbucket-push-trigger	5/14/21, 11:52 PM	3 min 26 sec
5710d67b	DanyloVanin/toneup-front-up...	master	07927e5	rmgpbmb-toneup-front-updated-europe-west3-bitbuc...	5/13/21, 10:11 PM	4 min 35 sec
8cf227b2	DanyloVanin/toneup-front-up...	master	ff402d9	rmgpbmb-toneup-front-updated-europe-west3-bitbuc...	5/13/21, 10:03 PM	3 min 49 sec

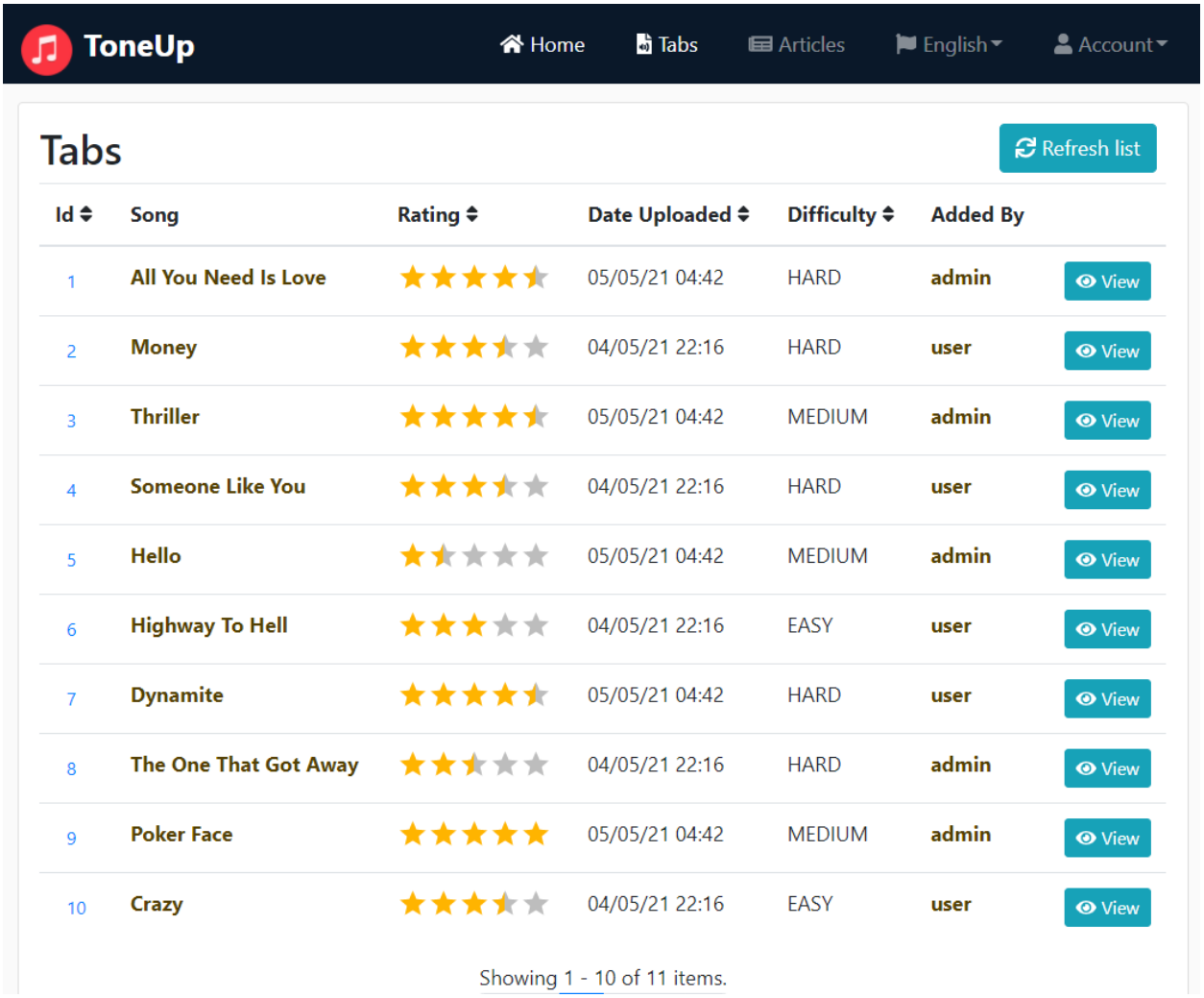
Додаток В

(обов’язковий)
Приклад сторінки застосунку з табами однієї пісні



Додаток Г

(обов'язковий)
Приклад сторінки застосунку з переліком табів



Додаток Д

(обов’язковий)
Приклад документації API за допомогою Swagger UI