МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет ________________Комп'ютерних наук_______________________
(повна назва)

Кафедра_________________Програмної інженерії_________________________
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти - другий (магістерський)

Дослідження методів генерації серверного коду у хмарі
(тема)

Виконав: Випускник 2 курсу, групи ІПЗм-19-3___

___________________Шабанов Д.В.______________
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

__Освітньо-наукова програма_
(тип програми)

___Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник ______________проф. каф. ПІ Каук В.І .
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____________________ З.В.Дудар

2021 р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет_Комп’ютерних наук______________________________________

Кафедра_Програмної інженерії____

Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_________________

Тип програми _освітньо-наукова програма____________________________

Освітня програма_Інженерія програмного забезпечення ___________________

ЗАТВЕРДЖУЮ:
Зав. кафедри ______________
(підпис)
«26»березня 2021 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Шабанову Дмитру Валерійовичу__________________________________

1. Тема роботи Дослідження методів генерації серверного коду у хмарі затверджена наказом університету від “26” березня 2021 р № 385 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 14 травня 2021 р.
3. Вихідні дані до роботи *методи генерації серверного коду у хмарі, AWS, C#, Visual Studio, Serverless, GraphQL, APP Sync , Databases , Інфраструктура*_______
4. Перелік питань, що потрібно опрацювати в роботі _*аналіз предметної галузі, формування функціональних вимог та не функціональних вимог, архітектура програмного забезпечення, генерація коду у хмарі.*
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, ілюстрацій (слайдів) мета завдання, обґрунтування доцільності розроблення, постановка задачі, методи і архітектурні підходи, структурно-логічна схема взаємодії даних, опис отриманих результатів, демонстраційні матеріали

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	Каук В.І.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	26.01.2021	виконано
2.	Постановка задачі і цілі дослідження	2.02.2021	виконано
3.	Аналіз предметної області	13.02.2021	виконано
4.	Постановка задачі	11.03.2021	виконано
5.	Формування вимог до програмної системи	10.04.2021	виконано
6.	Архітектура та проектування програмної системи	17.04.2021	виконано
7.	Підготовка пояснювальної записки	19.04.2021	виконано
9.	Нормоконтроль, рецензування	12.05.2021	виконано
10.	Занесення диплома в електронний архів	14.12.2021	виконано
11	Попередній захист	15.05.2021	виконано
12.	Допуск до захисту у зав. кафедри	16.05.2021	виконано

Дата видачі завдання __25 січня____2020 р.

Студент ____
(підпис)

Керівник роботи __________________________ __доц. Каук В.І.______
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 61 с., 15 рис., 14 дж.

Об’єктом дослідження є способи генерування серверного коду у хмарі.

Метою роботи є проектування системи генерації серверного коду у хмарі.

Методи проектування базуються на технологіях C# та Serverless.

У результаті роботи здійснено проектування архітектури системи для генерування серверного коду у хмарі.

ГЕНЕРУВАННЯ, C#, ХМАРА, СЕРВЕР, КОД, SERVERLESS.

The object of research is ways to generate server code in the cloud.

The purpose of the work is to design a server code generation system in the cloud.

Design methods are based on C # and Serverless technologies.

As a result of the work, the system architecture was designed to generate server code in the cloud.

GENERATION, C#, CLOUD, SERVER, CODE, SERVERLESS.

Я, Шабанов Дмитро Валерійович, студент групи ІПЗм-19-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів генерації серверного коду у хмарі», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання. Я ознайомлений(а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface;

ПЗ – Програмне забезпечення;

ПП – Програмний продукт.

ЗМІСТ

ВСТУП ... 8

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАВДАННЯ ... 10

1.1 Загальна характеристика систем генерації серверного коду ... 10

1.2 Огляд не функціональних вимог систем генерації серверного коду ... 14

1.3 Постановка задачі ... 16

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ ... 17

2.1 Загальні вимоги ... 17

2.2 Функціональні вимоги ... 17

2.3 Нефункціональні вимоги ... 18

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ... 19

3.1 UML проєктування ПЗ ... 19

3.2 Проєктування архітектури ПЗ ... 23

3.3 Неперервна інтеграція та тестування ... 25

3.4 Неперервна безпека ... 25

3.5 Інфраструктура як код та неперервне знищення ... 26

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ... 27

4.1 Загальний опис ... 27

ВИСНОВКИ ... 41

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ ... 42

ДОДАТОК А ... 44

ДОДАТОК Б ... 45

ДОДАТОК В ... 46

ДОДАТОК Г ... 50

ДОДАТОК Д ... 58

ДОДАТОК Е ... 60

ВСТУП

Останнім часом можна побачити інтерес до платформ без програмного коду які дозволяють створювати продукти або окрему функціональність завдяки програмному інтерфейсу аніж певній мові програмування напряму. Метою таких програмних систем є пришвидшення процесу розробки програмного забезпечення або створення прототипів програмного забезпечення, а також зменшення людських ресурсів для створення програмного забезпечення.

Зараз можна виділити два типа таких платформ, перші радше платформи для генерації певного функціоналу програмного забезпечення, наприклад логування або повідомлення, такі системи можна віднести до моделі програмний інтерфейс як сервіс, другі мають на меті зменшити поріг входження для створення програмних продуктів і дозволити виробляти програмні продукти без вміння писати програмний код. Зараз великі корпорації такі як Google та Amazon мають власні платформи які дозволяють робити веб та мобільні додатки з серверною логікою.

Дана робота має на меті дослідити способи генерування серверної логіки для платформ такого типу та виправити загальні недоліки у існуючих реалізаціях. Необхідно визначити, загальні недоліки та можливість створення за допомогою таких систем комплексних рішень, а також наскільки можливо інтегрувати існуючі рішення у вже створені програмні системи.

Об’єкт дослідження – системи створення серверного коду у хмарі. Предмет дослідження – динамічний серверний код, який віршує проблему створення програмних додатків без навичків програмування.

Проаналізовано схожі системи та на основі їх недоліків було побудовано програмну систему, зроблено тестування програмної імплементації, та інтеграцію її у мобільний додаток.

В результаті дослідження також були порівняні системи у вартості використання у залежності від навантаження та кількості користувачів.

В результаті дослідження потрібно довести застосовність систем генерації серверного коду в хмарі у повсякденних задачах таких як створення api для запитів у базу даних відповідно до моделей, можливість створення гнучких запитів з динамічною вибіркою та фільтрацією, також потрібно порахувати вартість користування сервісами і дослідити можливість створення калькулятору вартості використання в залежності від навантаження нової системи з можливістю приблизною оцінки за наступні сім, десять або календарний місяць. Також було досліджено способи створення довільних схем баз даних з відношеннями один до одного та один до багатьох.

Об’єктом дослідження є способи генерації серверного коду.

Предметом дослідження є генерування серверних веб інтерфейсів з логікою роботи з базою даних.

За результатами атестаційної роботи є публікація тез конференції. Також опубліковано тези, в яких описано використання систем генерацій серверного коду в хмарі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальна характеристика систем генерації серверного коду

Перш ніж перейти до систем генерації серверного коду потрібно зрозуміти у чому їх особливість та навіщо вони використовуються. На даний момент ці системи набувають все більшу популярність як серед людей які знають одну або декілька мов програмування так і серед людей які не вміють писати програмний код з використанням будь-якої мови програмування.

Розглянемо першу групу людей які вже вміють програмувати у все одно обирають для створення свого програмного продукту системи генерацій серверного коду, також слід відзначити що є велика тенденція до збільшення впливу код генерації адже у сучасному світі навіть для створення власного проекту обираються вже існуючи шаблони у випадку розробки з нуля які до розробників програмного забезпечення постачає сама мова або фреймворк[1] на якому вони планують розробляти програмний продукт. Тож чому люди які вміють програмувати обирають вже готові системи? Однією с головних причин є пришвидшення розробки та можливість розробити прототип програмного продукту за відносно невеликий час. Також чимось дуже схожим до цього є використання вже готових компонентів які генеруються за допомогою спеціальних команд у обраній мові програмування. З певною обережністю можна сказати що технологія контейнеризації також допомагає у генеруванні деяких частин програмних систем наприклад системи збору метричних даних та логування які дозволяють у пару простих дій згенерувати систему на написання якої можна було потратити місяці, а може навіть і роки.

Розглянемо другу групу людей які не мають знань будь-яких мов програмування та навіщо це їм. По-перше ця група людей має ідею, але немає навиків для її втілення саме тому їм або потрібно знайти команду яка втілить ідею продукту у життя або вивчити мову програмування і втілити її самостійно і тут стає зрозумілим що вивчити мову програмування з нуля може бути перепоною та

тільки віддалити від кінцевої мети. Звісно можна знайти команду, але як знайти команду та перевірити її на природність якщо в тебе немає безпосереднього досвіду, і що робити якщо немає достатнього бюджету для втілення кінцевого продукту? У цьому випадку велика кількість людей використовують різні системи генерації програмного коду, наприклад компанія Roastery Games для свого продукту Smartphone Tycoon які використали конструктор для створення своєї гри і як результат гра має більш ніж мільйон завантажень на платформах IOS та Android. Якщо брати системи генерації саме серверного коду то можна привести Bank of Etihad який використовує платформу Backendless для власного програмного продукту. Взагалі в цілому можна сказати що переважна більшість систем використовують системи генерації коду у своїх програмних продуктах і тій чи іншій мірі.

Розглянемо існуючі системи систем генерацій серверного коду, але варто відзначити що їх існує дуже вилка кількість, саме тому було обрано тільки деякі з них.

Першою системою буде платформа Backendless яка працює вже протягом восьми років та має сотні користувачів та працює з великими корпораціями.
До переваг платформи слід відзначити що за її допомогою можливо генерувати не тільки серверний код у хмарі, а також можливість створення UI[2] частини додатку під браузер, або створити мобільний додаток та використовувати створений серверний код у цьому мобільному додатку.
В цілому платформа дозволяє розробляти додатки будь-якої складності та має величезну кількість інтеграцій з платіжними системами, системами сповіщення за допомогою СМС повідомлень, листів на електронну пошту. Також великою перевагою цієї системи є можливість кінцевої генерації на різних мовах програмування наприклад Java, Python, Javascript, C#.

Якщо говорити про недоліки системи можна відзначити що через велику кількість функціоналу дуже важко орієнтуватись, тобто для входження у цю платформу потрібно пройти тренінги які наявні на платформі YouTube у профілі цієї компанії, також слід відзначити, що платформа більше орієнтована не на

людей які не вміють програмувати, а на людей які мають певні навички та мають розуміння що таке псевдокод, що таке база даних, та як будувати запроси на мові SQL, тобто платформа створена перш за все для користувачів з певним опитом досвідом програмування.

Також варто відзначити як формується ціна за використання програмного продукту, є три варіанти для кінцевого користувача. Першим варіантом є використання серверів компанії Backendless, та у цьому випадку потрібно піти до служби підтримки та з'ясувати плановане навантаження та сформувати вимоги до серверів та баз даних. У цьому випадку все залежить тільки від розміру програмного продукту та кількості користувачів та функціоналу. Другий варіант під собою розуміє покупку ліцензії на програмний продукт та встановлення платформи на власті сервери, але у цьому випадку моніторинг, підтримка та оновлення лягає на кінцевого користувача, ціна на ліцензію також встановлюється тільки через комунікацію з відділом підтримки, що досить не прозоро та не дає можливість оцінити чи є ціна великою або малою для створення програмного продукту за допомогою платформи Backendless. Третій підхід це використання клауд платформ як серверів і у цьому випадку є три тарифних плани SPRINGBOARD, CLOUD 9, CLOUD 99. Перший план це план для ознайомлення і робити на ньому комерційні системи неможливо, але він безкоштовний. Другий та третій коштують двадцять п'ять доларів США та дев'яносто дев'ять доларів США і відрізняє ці плани максимальна кількість користувачів 25 та 50 одночасно, кількість таблиць баз даних 100 та 200 відповідно, а також ліміт запитів до серверу на місяць десять мільйонів та сорок відповідно.

Платформу Backendless можна віднести до платформ які схожі на візуальні мови програмування або конструктор, де кінцевий користувач створює блоки логіки програмного коду та на основі цих блоків подалі будується вихідний код програмного продукту.

Другою розглянутою платформою буде Amazon HoneyCode від компанії Amazon Web Services. Ця платформа досить нова та знаходиться у бета версії і

доступна з 2020 року. Основними перевагами цієї платформи є великий вендор, що її підтримує та можливість легко інтегрувати різні частини платформи з сервісами Amazon Web Services[3]. Незважаючи на досить невеликий час на ринку продуктів для генерації програмного коду платформа вже має реальних клієнтів. Сама платформа побудована на тому що є користувачі та таблиці і чимось дуже схожа на Excel. Таблиці є нічим іншим як джерелом даних для програми та використовують роль бази даних до якої можливо будувати запроси за допомогою графічного інтерфейсу, а також відображати дані з цих джерел будуючи відповідні запити.

Інтерфейс Amazon HoneyCode більш інтуїтивний для кінцевого користувача, ніж інтерфейс розглянутої вище платформи Backendless. У ньому не потрібно будувати блоки коду за допомогою псевдокоду або проходити навчаючі відео щоб зрозуміти як користуватися платформою. Також до переваг можна відзначити велику кількість шаблонів для проектів таких як репортінг часу для команд розробників, списки справ, системи сповіщень, системи підтримки користувачів та системи менеджменту контенту, а також ще багато інших. Тобто якщо користувач має ідею для створення продукту схожого на вже існуючі він може це зробити за пару хвилин та зовсім не витрачати час на розробку та почати використовувати кінцевий програмний продукт. Також слід відзначити те що платформа дозволяє генерувати не тільки серверні додатки, а також і мобільні за допомогою конструктору так само як і платформа Backendless. Якщо говорити про ціни за користування програмного продукту то існує три плани, безкоштовний та два для реального бізнесу і ціна формується відповідно до кількості записів у базі даних. До недоліків платформи можна віднести значну орієнтованість на продукти для менеджменту табличних систем і недостатню гнучкість у випадку динамічної моделі, а також проекту з високими навантаженнями на систему.

1.2 Огляд не функціональних вимог систем генерації серверного коду

Варто відзначити що у таких системах важливі наступні не функціональні вимоги:

– надійність;

– конфігурування;

– масштабування;

– вартість;

– цілісність інформації;

– портативність;

– доступність;

– безпечність;

– здатність до розширення.

Можна побачити що у таких системах є надважливим доступність системи та її надійність адже якщо система впала то від цього залежить робота інших систем, що можуть бути досить великими.

Також варто відзначити, що у таких системах дуже активно використовується підхід описання програмного інтерфейсу за допомогою технології Swagger, яка дозволяє документувати програмний інтерфейс серверного коду у форматах .yml та .json, на данний момент Swagger[4] є стандартом для документування програмних інтерфейсів на сервері і як це виглядає можна побачити на рисунку 1.

Дана технологія надає можливість взаємодіяти з вже існуючим програмним інтерфейсом бачити його вхідні та вихідні контракти, також давати можливість показувати її як документацію проекту.

Якщо проаналізувати цінові плани усіх наявних проектів для генерації серверного коду у хмарах то можна побачити що переважна більшість для ціноутворення використовує так звані транзакції якими можуть бути як запити до

базі так і запити до програмного інтерфейсу вихідної системи, що дозволяє використовувати технології які підтримують pay-as-you-go плани.

Аналізуючи програмні продукти можна побачити велику кількість способів авторизації та аутентифікації користувачів, використовуючи різні протоколи, чи різні системи наприклад Gmail, Facebook.

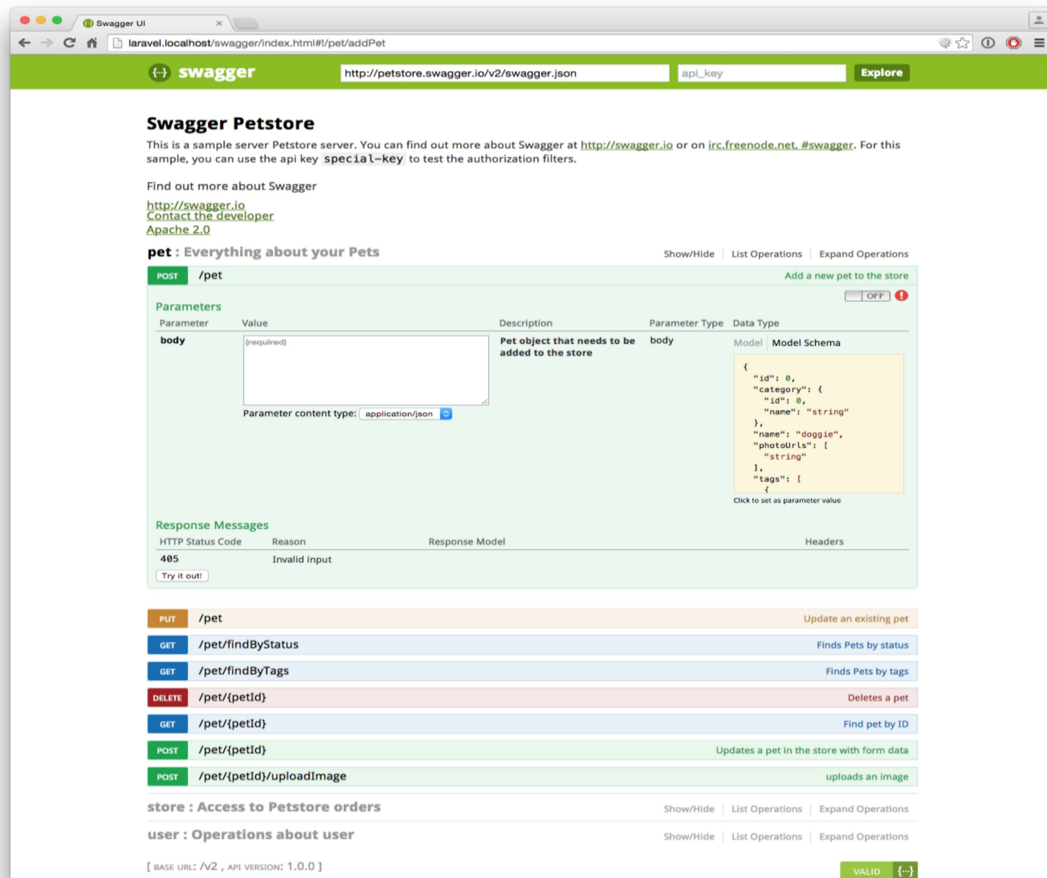


Рисунок 1 – графічний інтерфейс Swagger

Якщо казати про недоліки подібних систем можна відзначити досить непрозоре формування кінцевої ціни та відсутність можливості підрахувати реальну вартість системи, також варто відзначити відсутність можливості реагувати на будь які зміни у системі, всі механізми на які можливо реагувати чітко визначені існуючим програмним інтерфейсом.

1.3 Постановка задачі

Дана робота полягає у розробці програмної системи для створення серверного коду у хмарному середовищі, дане рішення повинно виділятися серед конкурентів за рахунок наступних чинників таких як документованість рішення, гнучкість та можливість розраховувати вартість в залежності від зовнішніх факторів.

У ході дослідження було поставлено такі задачі:

а) провести аналіз предметної області, виявити проблемні місця та актуалізувати рішення, порівняти їх;

б) сформувати вимоги до програмної системи генерації серверного коду;

в) спроектувати архітектуру програмного забезпечення;

г) змоделювати необхідні UML-діаграми;

д) розпланувати систему генерації серверного коду у хмарі.

У ході дослідження було проведено аналіз предметної області, сформульовано основну мету та визначено задачі, які необхідно виконати для успішного завершення роботи.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальні вимоги

Для проектування програмної системи необхідно сформулювати певні вимоги такі як:

– перелік функціоналу, що повинна містити система;

– архітектура кінцевої програмної системи;

– частини, з яких вона буде складатися;

– технології.

2.2 Функціональні вимоги

Отже потрібно реалізувати систему яка дозволить генерувати код для взаємодії з базою даних, також код який дозволить реагувати на ті чі інші модифікації за допомогою технологій вебхуків, мати можливість авторизуватися за допомогою технології OAuth. Також мати програмний інтерфейс, що дозволяє генерувати код для взаємодії за програмним інтерфейсом на обраній мові програмування (Java, C#, Javascript, Python, PHP). Також продукт потрібен вміти калькулювати вартість використання в залежності від навантажень та кількості запитів до системи.

Для реалізації подібної задачі потрібно виконати наступне:

а) описати формат специфікації для створення програмних інтерфейсів;

б) створити модуль налаштування авторизації;

в) створити модуль генерування документації;

г) створити модуль сповіщення про події на задані адреси;

д) створити модуль калькулювання вартості програмного продукту в залежності від навантажень.

2.3 Нефункціональні вимоги

Окрім зазначених вище функціональних вимог, варто розглянути також нефункціональні вимоги до системи, а саме вимоги до мови програмування, хмарної платформи та фреймворків і інструментів:

а) автоматичний генератор коду взаємодії за базою даних буде написаний на мові C#.

б) модуль авторизації буде розроблений на мові C#;

в) як специфікацію API було обрано GraphQL;

г) хмарною платформою є AWS;

д) механізм API має підтримувати технологію вебхуків.

Також до нефункціональних вимог можна віднести використання технології AWS Lambda та фреймворку Serverless[5], як базу використовувати DynamoDB, також потрібно налагодити процес безперервного постачання, інтеграції, тестування та безпека.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

В ході аналізу вимог важливим етапом їх формування було відображення станів, взаємодій даних та компонентів тощо у наглядному вигляді. Для цього було обрано UML діаграми. Для проектування програмної системи було використано безкоштовний засіб проектування draw.io та спроектовано наступні діаграми UML:

а) діаграма компонентів, що показує основні компоненти системи та зв’язок між ними, її можна побачити на рисунку 2;

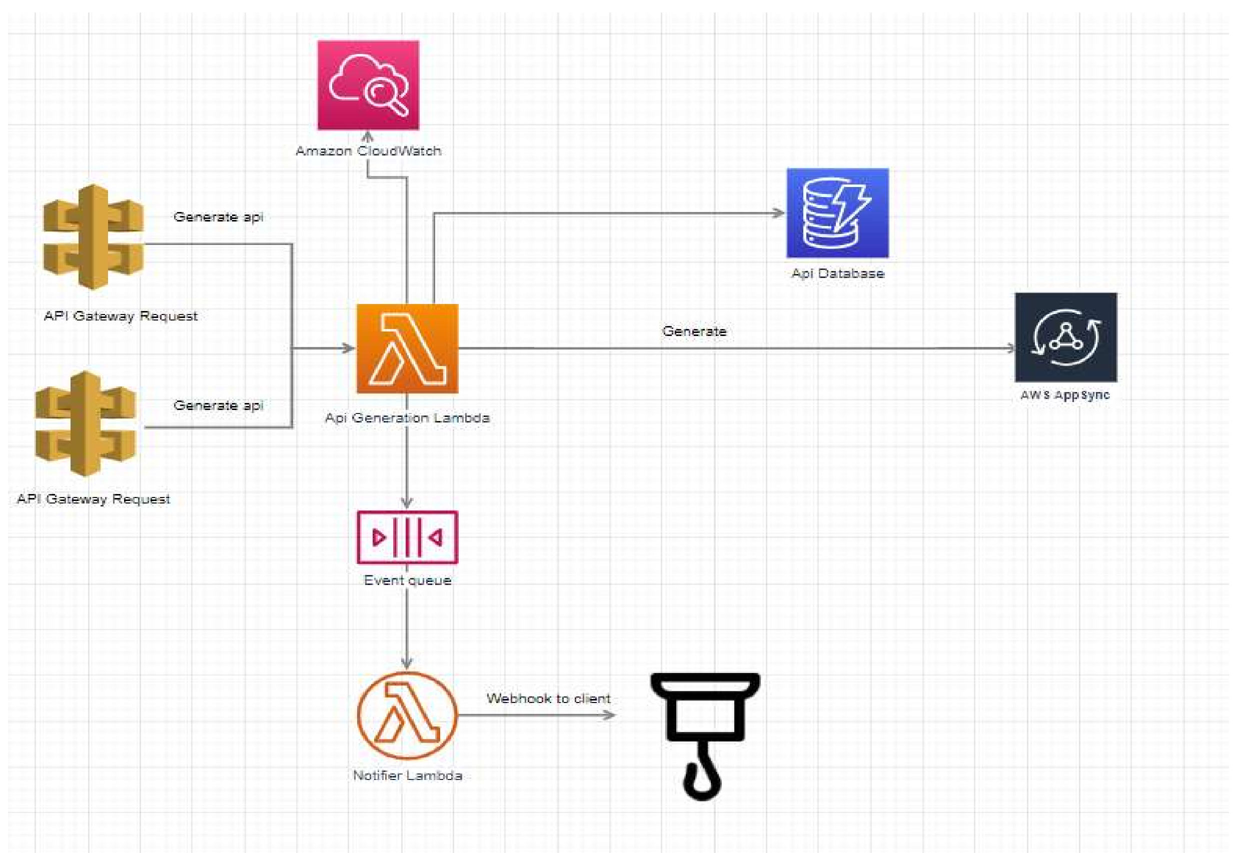


Рисунок 2 – Діаграма компонентів

На рисунку зображено основні компоненти такі як API Gateway для обробки клієнтських запитів, а також авторизація користувачів, функція яка генерує

програмний api та додає його у базу даних DynamoDB, після генерування створює AWS app sync додаток з використанням власних налаштувань схеми та її приведень після чого генерує подію у SQS[6] і вже на це реагує функція що сповіщає про успішне виконання запиту клієнта.

б) діаграма послідовностей показує послідовність дій для клієнта ціль якого згенерувати серверний код, відображена вона на рисунку 3;

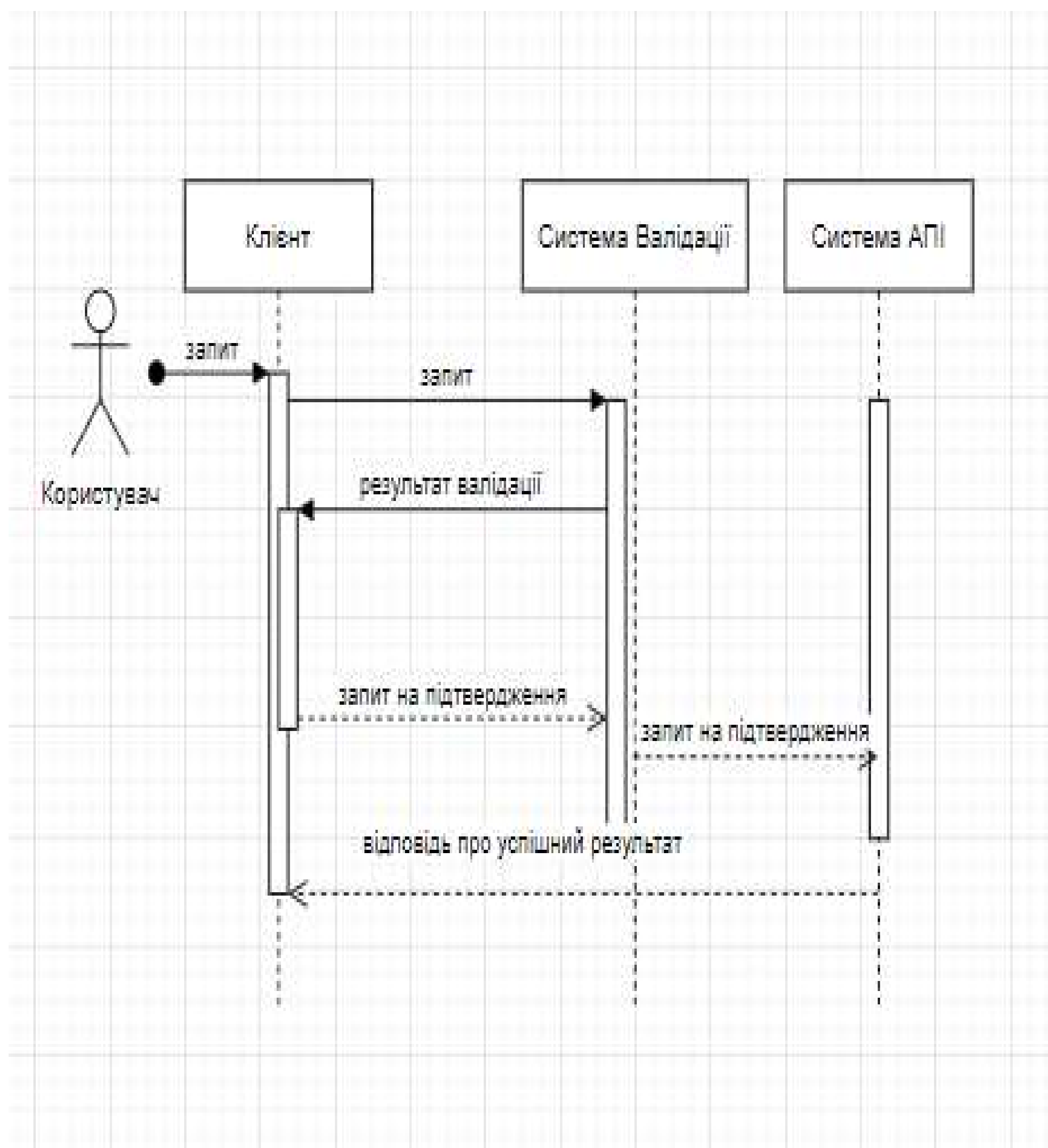


Рисунок 3 – діаграма послідовностей

На рисунку відображено послідовність дій для користувача щоб зробити серверний код, спочатку формується запит зі схемою яка потім приходить до

сервісу валідації запитів генерування, який після цього відповідає можливо зробити таке api чи ні, після цього користувач у разі успішної валідації має можливість запустити процес створення api, після цього починається процес створення api який у разі успіху повідомляє клієнта про успішну операцію та висилає посилання на документацію.

в) діаграма класів для функції генерування api яка зображена на рисунку 4;

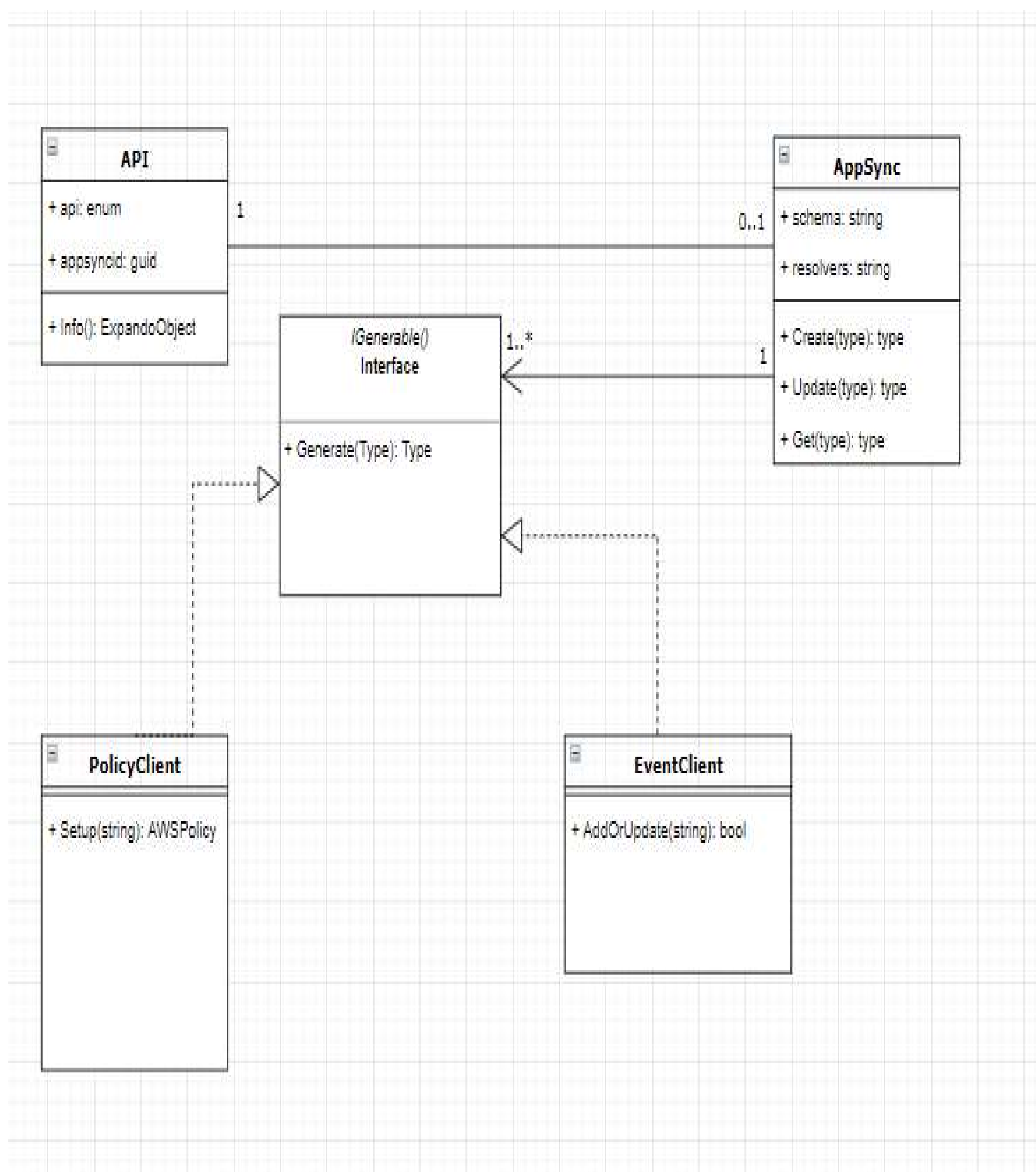


Рисунок 4 – Діаграма класів для функції генерування api

Як можна побачити на діаграмі існує 4 компонента які активно взаємодіють між собою перший API[7] зберігає тип та зв’язок з класом AppSync який зберігає у собі схему api та його приведення, також додатково є два класи PolicyClient та EventClient які допомагають встановлювати безпеку та права api і події на які потрібно реагувати або які мають виникати під час користування api.

г) діаграма класів функції валідації схеми api контракту зображено на рисунку 5.

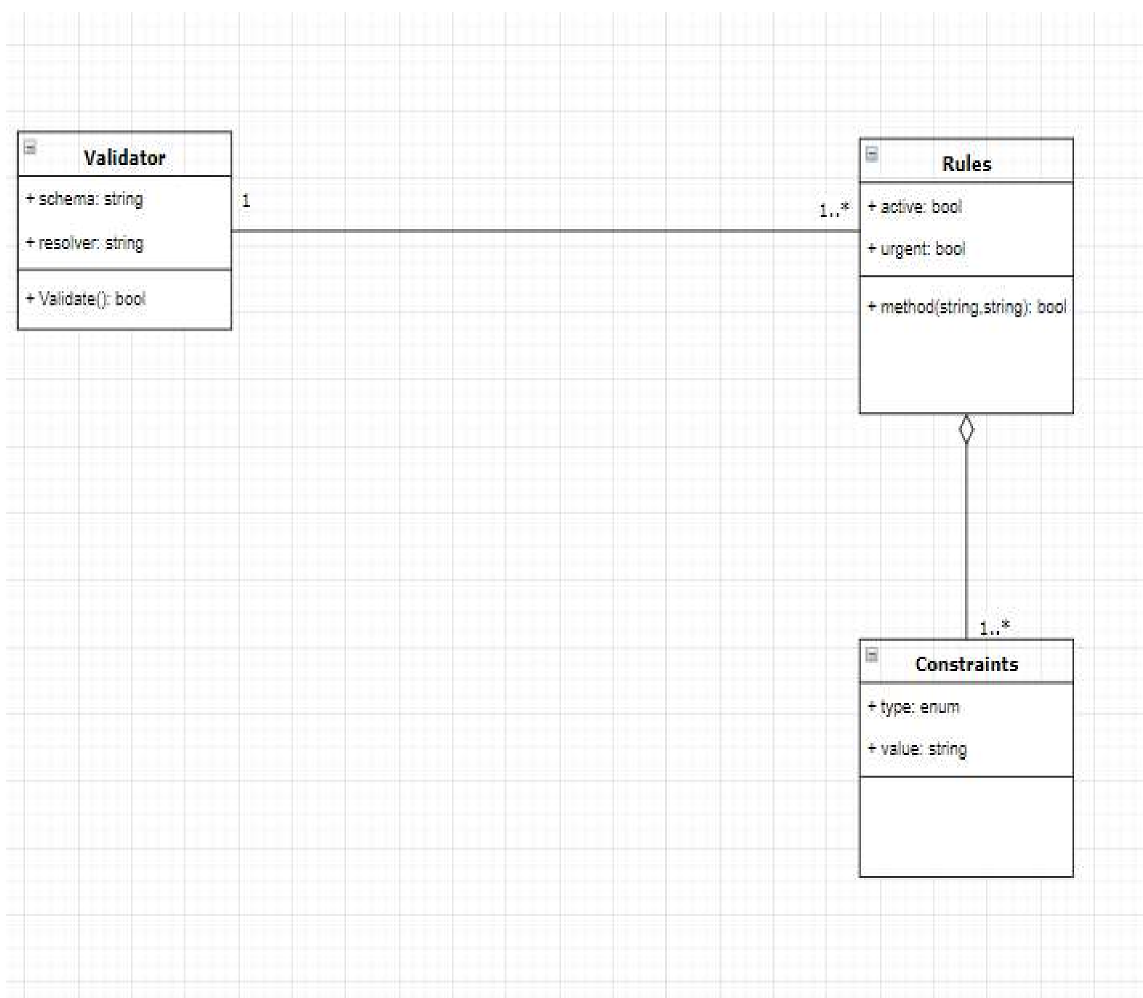


Рисунок 5 – Діаграма класів функції валідації

На діаграмі класів відображено основні компоненти валідатору, а саме клас Validator, який використовує схему та вирішувач як основу валідації, та має один або більше правил валідації які можуть бути активними і бути критичними для виконання або ні, і кожен з яких у той час має один чи більше додаткові

обмеження які використовуються для внутрішньої логіки валідації схеми або проекцій програмного контракту.

3.2 Проєктування архітектури ПЗ

На базі згаданих вище умов та функціональних та нефункціональних умов було побудовано архітектуру системи.

Як хмарне середовище була обрана платформа AWS. Сама система буде збудована на основі мікросервісної архітектури що дозволить розробляти компоненти незалежно один від одного та дозволить їх масштабувати незалежно один від одного використовуючи індивідуальні метрики, а також розгортати їх і більш швидкий час через більш менший розмір вихідних файлів[8]. Кожен з мікросервісів буде мати власний контекст і базу даних.

Для відображення корисної інформації було обрано технологію Grafana, інтерфейс якої можна побачити на рисунку 6.

Рисунок 6 – Графічний інтерфейс Grafana

Для збереження даних було обрану базу даних DynamoDB яку постачає платформа AWS, до переваг цієї бази даних можна віднести її ціну та зручність реплікації або партиції.

Для збору метрик з мікросервісів було обрано Prometheus[9] який дозволяє виконувати довільні вибірки по метрикам сервісу та періодично збирає їх, графічний інтерфейс зображено на рисунку 7.

Рисунок 7 – Графічни інтерфейс Prometheus

Як основну технологію для мікросервісів було обрано AWS Lambda[10] через її здатність до швидкого розгортання та можливість платити тільки за користування, що в свою чергу надає змогу зменшити вартість використання додатком у порівнянні з іншими моделями такими як EC2 та ECS[11].

Для проксювання та авторизації користувачів було обрано технологію Api Gateway яка дозволяє підтримувати різні середовища для розробників, та тестування. Також ця технологія є единою точкою входу у програмне забезпечення і ізолює клієнта від реального серверу.

3.3 Неперервна інтеграція та тестування

Задля поліпшення якості коду було вирішено використовувати процес безперервної інтеграції та робити регулярні процеси побудови вихідного коду, а також робити статичний аналіз за допомогою проекту SonarCube який регулярно перевіряє джерело вихідного коду та робить звіти щодо поліпшення або погіршення якості коду.

Для поліпшення забезпечення якості вихідного продукту було обрано процес неперервного тестування який полягає у неперервній перевірці продукту за допомогою автоматичних тестів та генерування звітів як результат цих перевірок, самі тести було розроблено за допомогою середовища Postman та виконувались вони за допомогою newman-cli автоматично.

3.4 Неперервна безпека

Для забезпечення безпеки кінцевого додатку було обрано підхід неперервної безпеки та використання CloudRemedy яка дозволяє перевіряти

доступи існуючих додатків на платформі AWS та на основі індустріальних стандартів дає рекомендації щодо безпеки платформи.

3.5 Інфраструктура як код та неперервне знищення

Для забезпечення можливості переносити продукт та швидко його відновлювати від падінь було обрано процес неперервного знищення який полягає у тому що раз у неділю додаток знищується з платформи разом з усіма доступами та відновлюється використовуючи існуючі механізми. Даний підхід є можливим через підхід до інфраструктури як до коду, тобто усі налаштування на платформі AWS[12] є у вигляді інструкцій які використовують інструкції написані за допомогою yml файлів які потім конвертують ці файли до json, що сумісний з технологією CloudFormation і є собою переліком ресурсів, правил доступів, обмежень які потрібні для роботи системи. Всі інструкції зберігаються у репозіторії на GitHub та мають версіювання відповідно до версій коду.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Загальний опис

Програмний додаток створено на мові програмування C#. Дана мова підтримує кілька парадигм програмування, зокрема об'єктно-орієнтовану та функціональну. Структури даних високого рівня разом із статичною типізацію роблять її привабливою для розробки надійних програмних систем, а також зменшують час виконання у порівнянні через те що платформа компілює а не інтерпретує весь вихідний код. Також через статичну типізацію створення контрактів програмного інтерфейсу стає зручнішим та надає можливість модульної та розподіленої розробки. Модульна платформа для розробки .NET CORE дозволяє використовувати не тільки Windows як кінцеву систему для розгортання що дозволяє пришвидшити та зменшити ціну додатку.

Додаток створено в середовищі Visual Studio Code. Перевагами цього інструменту є наявність спеціальних інструментів для візуалізації даних, інтелектуальні засоби рефакторингу коду та вбудована консоль, а також те що середовище безкоштовне, до перевах середовища можна віднести велику кількість розширень на будь який смак, а також під будь яку мову. Середовище підтримує різні операційні системи такі як Windows, Linux, MacOS. Також слід відзначити шо середовище більш швидке ніж стандартні інструменти для розробки на мові програмування C# такі як Visual Studio.

Для створення програмного додатку було обрано хмарну платформу AWS та заведено там аккаунт.

Для проекту було вирішено використовувати Serverless архітектуру через такі важливі фактори як швидкість розробки, швидкість розгортання, можливість ізоляції, а також можливість реагуваті на перепади навантажень на кінцеву систему, також було прораховано кінцеву вартість використання у порівнянні з системою яка була розгорнута на віртуальних машинах. Сама архітектура позволяє використовувати сучасні підходи до постачання та тестування продукту

і його безпеки такі як Continuous Deployment, Continuous testing, Continuous security. Через те що система має необхідність працювати для кожного користувача окремо то було обрано модель pay-as-you-go яка дозволяє зменшити вартість кінцевого продукту та платити тільки за безпосередне використання програмного продукту.

Для поліпшення розробки було вирішено обрати фреймворк Serverless. Якщо говорити про фреймворк Serverless це безкоштовний фреймворк для створення проектів які використовують Serverless архітектуру, та розгортаются у хмарних середовищах Amazon Web Services, AZURE, Google Cloud Platfrom.

Для розробки програмного додатку підключені наступні модулі:

− Newtonsoft.JSON – модуль для роботи с JSON, відповідає за серіалізацию та десереалізацію запитів користувача.

− AWS.APPSYNC – модуль для роботи с технологією AWS APPSYNC яка дозволяє побудовувати програмні інтерфейси на основі GraphQL, а також відстежувати їх;

− AWS.SDK – модуль для роботи з технологіями AWS та можливість створення ресурсів на стороні AWS використовуючі програмні виклики бібліотеки;

− Logging – модуль для ведення журналу операцій додатку;

− Queues – модуль для роботи з асинхронними операціями які використовують черги;

− ApiEvents – модуль для роботи з викликами API;

− DatabaseEvents – модуль для керування подіями у базі данних, зміна, додавання, видалення;

− Organizations – модуль для керування обліковими записами користувачів, додавання нових, внесення змін у старі;

Програмний додаток складається із модулів, які за своїм призначенням об’єднані в пакети:

− Business – пакет, який містить бізнес-логіку;

− Models – пакет, який містить моделі сутностей предметної області;

− Utils – пакет із допоміжними модулями.

Файл Handler.cs – точка входу у програму. serverless.yml – файл, в якому задається конфігурація функції.

На рисунку 8 наведений приклад файлу serverless.yml для модулю організацій.

```yaml
service: Organizations
frameworkVersion: '1'

provider:
  name: aws
  runtime: dotnetcore3.1
  environment:
    LAMBDA_NET_SERIALIZER_DEBUG: true
package:
  individually: true

functions:
  Organizations:
    handler: CsharpHandlers::AwsDotnetCsharp.Handler::Organizations
    events:
      - http:
          method: any
          path: /
      - http:
          method: any
          path: '{proxy+}'

    package:
      artifact: bin/Release/netcoreapp3.1/Organizations.zip
```

Рисунок 8 – Програмний код файлу serverless.yml

Весь файл має чітку схему, та декілька ключових секцій.

Перша секція provider, описує цільову хмарну платформу та середовище і фреймворк на якому буде виконуватися додаток.

Секція functions описує перелік можливих функцій для заданого пакету-артефакту, також визначає способи виклику у секції events, на заданому прикладі це http виклик який проксюється до додатку, також у цих секціях можна задавати ресурси які функція буде використовувати та обмеження по використанню, налаштовувати логування.

Артефакт для функції робиться у декілька кроків командної строки, тому був створенний допоміжний файл build.sh, який дозволяє використовувати його замість цих кроків, послідовність кроків є такою: кількість ітерацій;

− виклик dotnet restore який підтягує залежності проєкту;

− виклик dotnet tool install –g Amazon.Lambda.Tools бібліотека що дозволяє працювати з серверлесс технологіями у хмарному середовищі;

− dotnet lambda package пакує проект у zip файл.

Після того як є serverless.yml файл можливо перенести функцію у AWS. Для цього потрібно запустити чергу команд:

− ./build.sh для Linux або MacOS чи .\build.cmd для Windows;

− sls deploy.

Розглянемо більш детально, що відбувається під час останньої команди sls deploy. Перш за все перевіряється чи встановлені всі залежності та чи є на місці заданий артефакт. Надалі фреймворк serverless використовуючі файл який називаеться serverless.yml та містить вказівки для фреймворку та конфігурацію нашої функції генерує інструкції CloudFormation. CloudFormation – це сервіс Amazon, який дозволяє створити інфраструктуру в Amazon, описавши її тестом у форматі YAML, JSON. У нашому випадку це JSON. Далі завантажує наш артефакт на S3. S3 — сервіс-сховище даних, один з Amazon Web Services. Сервіс надає можливість для зберігання й отримання будь-якого обсягу даних, у будь-який час з будь-якої точки мережі, тобто так званий файловий хостинг. Також у вихідному CloudFormation файлі шлях до артефакту посилається на завантаженний артефакт до S3.

Перший єтап тільки створює необхідні артефакти для Amazon, інструкції та артефакт на стороні Amazon.

Далі фреймворк створює так званий Stack, який в цілому схожий на структуру данних і виконує спочатку перші команди cloudformation, у випадку успішного виконання переходить до других, третіх і далі. У випадку невдачі запускається зворотній процесс який робить компенсуючі операції з ресурсами на стороні AWS та весь процесс можливо побачити на веб-сторінці у AWS порталі.

Після успішного виконання всіх інструкцій можна побачити те що Stack стає зеленим та як результат створено AWS Lambda, Api Gateway для неї, графічний інтерфейс можна побачити на рисунку 9.

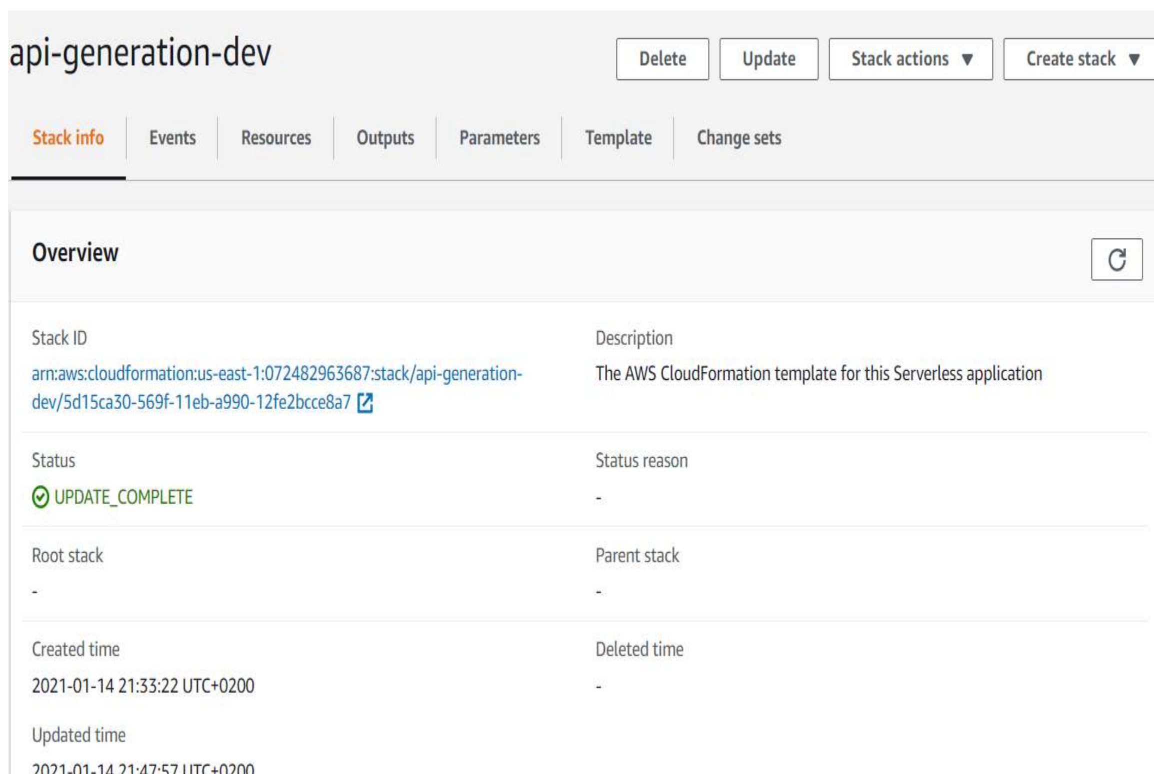


Рисунок 9 – приклад стеку cloudformation

На графічному інтерфейсі зображено загальну інформацію, список подій та ресурсів які виникають у результаті, вхідні параметри та список змін.

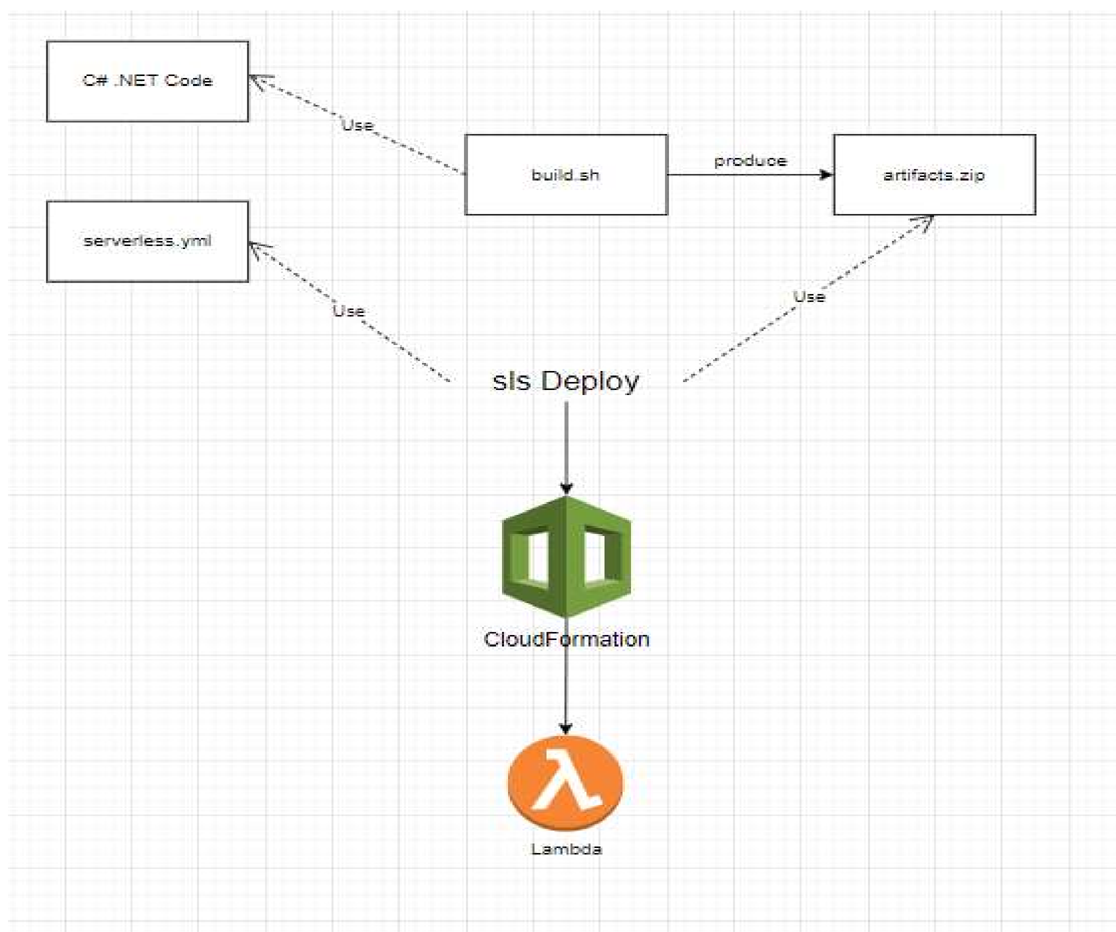
Згенеровану функцію можливо перевірити за допомогою команди sls invoke, або викликати за допомогою сконфігурованої події для виклику.

Занальну діаграму постачання з локального середовища до реального можна побачити на рисунку 10.

Під кожного користувача создається власне середовище та аккаунт у AWS за допомогою API AWS Organizations.

AWS Organizations допомогає централізовано керувати великою кількістю аккаунтів, дозволяє налаштувати ресурсі які вони використовують, виставляти рахунки за інфраструктуру для кожного аккаунту окремо.

Під час створення аккаунту, створюється окремий аккаунт у AWS за допомогою методу AmazonOrganizationsClient.CreateAccount(request), після створення аккаунту, надсилається письмо на електронну пошту з логіном та паролем для входу у консоль AWS.



.Рисунок 10 – загальна діаграма розгортання

Після цього можливо використовувати функції системи для генерування програмного коду.

Для генерування api, потрібно викликати метод POST api-generation, який для заданої GraphQL схеми згенерує AWS AppSync додаток.

Для додатку буде згенерован AppSync API за допомогою методу AmazonAppSyncClient.CreateGraphqlApi. Після цього буде згенерована функція за допомогою AmazonAppSyncClient.CreateFunction. Для бази даних буде згенеровано DynamoDB або ElasticSearch або RDS за допомогою функції

AmazonAppSyncClient.CreateDataSource, з заданими параметрами та додано резолвери Graphql.

Після цього API буде доступне для використання за допомогою мови запитів Graphql.

Для підписки на події у DynamoDB використовується технологія DynamoDB Streams, загальний принцип можна побачити на рисунку 11.

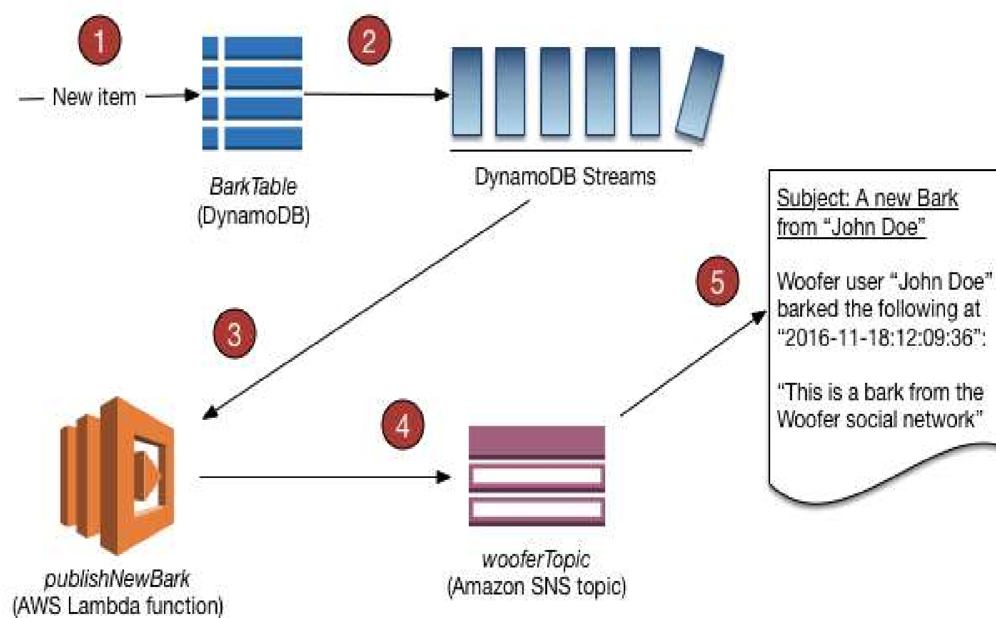


Рисунок 11 – загальний принцип роботи DynamoDB Streams

Також можливо додавати генерування подій після виклику заданого API у вигляді вебхуків або повідомлення до SQS. SQS це розподілена система повідомлень яка може працювати у двох режимах General Queue та FIFO Queue у останньому випадку першим оброблюється повідомлення яке було отримано першим.

Для відстеження роботи додатку було обрано стек Grafana, Prometheus.

Для налаштування було обрано технологію Docker та були використані docker images для Grafana grafana/grafana:7.5.5 та для Prometheus

prom/prometheus:v2.26.0. Надалі вони були поставлені на AmazonEC2. EC2 це виділенний сервер у хмарному середовищі AWS.

Весь код був завантаженний на платформу Github.

Для процессу неперервньої інтеграції було обрано Github Actions. Вони дозволяють налаштовувати різні процесси під час роботи з гілками у Github за допомогою yml інструкцій.

Було виділено наступні інструкції для пул реквестів у гілку main, перевірка що вихідний код компілюеться за допомогою команди dotnet build. Перевірка що файл serverless.yml має валідний формат за допомогою команди sls deploy –noDeploy, noDeploy позначає що результатом роботи команди не буде створення CloudFormation стеку а тільки показ потрібних ресурсів та перевірка на те що подальше завантаження можливе.

Для перевірки покриття тестами було використано CodeCov.

Для процесу неперервного тестування було обрано фреймворк xUnit для unit тестування. Для запуску тестів потрібно запустити команду dotnet test. Здебільшого unit тестування охоплювало Business модуль додатку, для інших залежностій було використано техологію Mocks та Stubs.

Mock об’єкти імітують реальну поведінку певних залежностей за допомогою реалізування інтерфейсів реальних об’єктов без їх безпосереднього створення.

Stub об’єкти були використані для перевірки що задані функції Амазону викликаются.

Успішне виконання тестів було обов’язковою ланкою для додавання коду до main гілки.

У разі якщо тести не проходили у інтерфейсі Github був репорт про тести які не пройшли та загальна кількість тестів які пройшли або не пройшли з детальним логуванням.

Також для виконання e2e тестів було розроблено стратегію sandbox середовища, загальну діаграму можна побачити на рисунку 12.

Під час створення пул реквесту до гілки main код завантажується до середовища sandbox за допомогою команди sls deploy –stage sandbox який заздалегіть був створенний, який є зеркальною копією гілки main з усіма ресурсами та залежностями

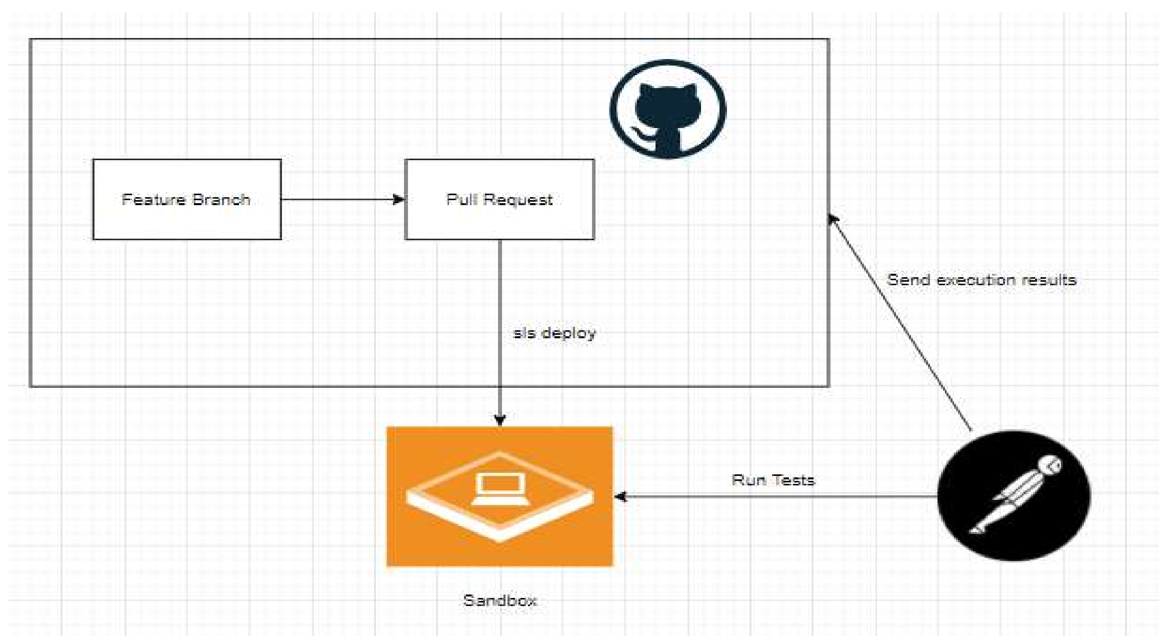


Рисунок 12 – діаграма тестування на sandbox середовищі

Після завантаження запускаються postman тести для http ендпоінтів за допомогою newmancli та генерувався репорт про виконання який можна побачити на рисунку 13.

Після успішного виконання e2e тестів код додається до гілки main автоматично та після цього запускається команда sls deploy яке доставляє код функцій до production середовища.

Для документування програмних контрактів було обрано специфікацю OpenAPI та Swagger який дозволяє зручно відображати програмні контракти а також викликати їх безпосередньо з веб сторінки.

Кожен самостійний модуль мав власний Github репозиторій, власний пайплайн для unit та e2e тестування, власну документацію.

Кожен репозиторій мав дві папки src та tests. У src зберігається вихідний код у папці за іменем репозиториію та serverless файл. У папці за іменем

репозиторію стуктура була однакова та поділена на чотири папки, Api, Business, Data, Utils, тобто класична трьох рівнева архітектура.

У папці tests було дві під папки e2e та unit, перша для postman тестів та друга для юніт тестів.

Обрана структура для репозиторіїв дозволила розробити загальні Gihub Actions та не писати під кожен с репозиторіїв свої що пришвидшило час роботи та покращило підтримку.

	executed	failed
iterations	1	0
requests	1	0
test-scripts	1	0
prerequest-scripts	0	0
assertions	1	1
total run duration: 1917ms		
total data received: 14B (approx)		
average response time: 1411ms		

Рисунок 13 – приклад звіту e2e у постман

Було створено репозиторії Organizations, Billing, Api-Generation, Monitoring, Events, Admin, Notifications, Auth.

Auth відповідає за авторизацію та аутентифікацію користувача у системі а також доступ до системи.

Organizations відповідає за створення аккаунтів для користувачів системи, керування їх середовищами та змінами.

Api-Generation допомогає користувачу створювати програмні api у системі а також бачити активні api, змінювати та видаляти їх. Створення API зроблено за допомогою AWS StepFunctions, процес створення можна побачити на рисунку 14.

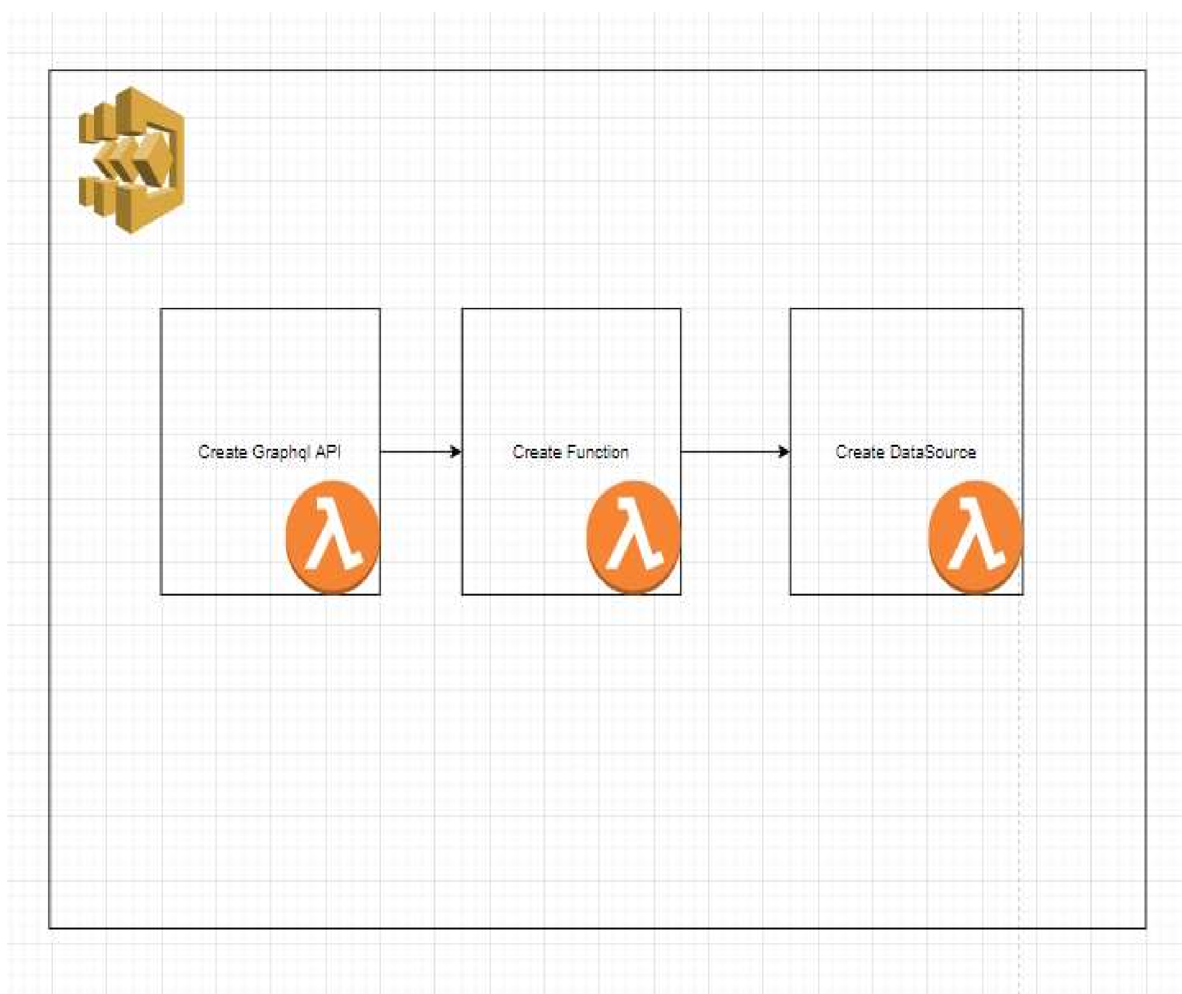


Рисунок 14 – процес створення Api

Billing дозволяє керувати оплатою за використання для кожного користувача системи, підрахунок вартості а також проєктування можливого використання, можливість прорахування потенційної вартості api у системі.

Monitoring вміщує логіку по загальному використанню api для всіх користувачів збирає статистику, а також дозволяє користувачу бачити власту статистику у режимі реального часу. Сервіс дописує метрики у сервіс AWS CloudWatch. AWS CloudWatch це сервіс моніторінгу за ресурсами та додатками у режимі реального часу. Діаграма Monitoring компоненту відображена на рисунку 15.

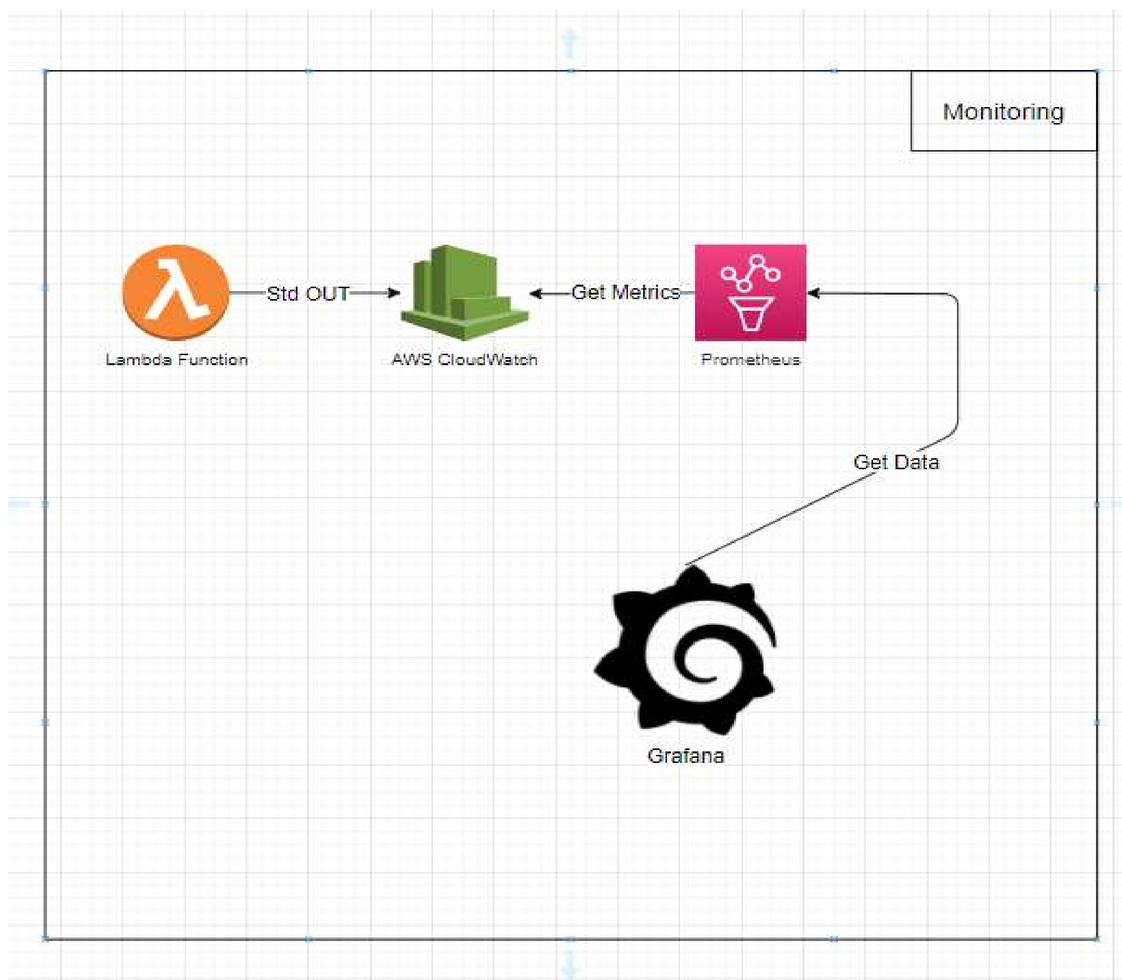


Рисунок 15 – діаграма Monitoring компоненту

Events дозволяє налаштовувати користувачу можливі події які він хоче відслідковувати як програмно так і за допомогою пошти.

Для перевірки безпеки проєкту використовується платформа CloudRemedy яка на основі заданих Policy та загальних рекомендацій для побудови додатків у

AWS сканує security[13] policy та надає рекомендації що потрібно змінити, також можливо налаштовувати власні правила.

Також варто відзначити що дана архітектура накладає певні обмеження та має підводні камні при імлементації такі як cold-start. Cold-Start це процес виконання лямбда функції коли вона не активна, через це можливі затримки під час першого запуску, або запуску функції яка давно не викликалась. Існує декілько можливих рішень, для вирішення цієї проблеми було обрано стратегію зменьшення кількості пакетів, що надає змогу збирати функцію швидше, а також періодичний ping лямбда функій щоб вони залишалися розгорнутими на AWS.

Для цього кожна функція мала /ping ендпоінт який повертав успішний статус код та викликався з періодичністю одна хвилина.

Також варто відзначити наступні ліміти:

– пам’ять жорсткого носія 512мб;

– розмір вихідного пакету не більше 50мб;

– максимальна оперативна пам’ять на функцію 256мб;

– максимальний час виконання функції 15 хвилин;

– максимальний розмір запиту 6мб;

– максимальне розмір відповіді 6мб;

– максимальна кілкість одночасних запитів 1000.

За запитом до AWS можна збільшити потрібні для функції ліміти, але у більшості випадків їх вистачає.

Загальна архітектура serverless дозволила використовувати підхід PayAsYouGo, що дозволило суттєєво зменшити плату за використання та надати можливість динамично масштабувати додаток через низьку зв’язність, також навіть якщо один з компонентів перестане працювати це не вплине на роботу системи в цілому.

Панель керування була розроблена за допомогою AWS Lambda[14], та на клієнтській стороні використовувався Angular. Додаток Angular був завантаженний на AWS Amplify. AWS Amplify це сервіс для розгортання frontend додатків написаних за допомогою Angular, React або VueJS.Для кожного

функціоналу був розроблений свій компонент. Всі компоненти були об’єднані у один SPA додаток що позитивно вплинуло на зручність використання додатку, а також надало можливість у парі з signalr бачити оновлення даних інтерактивно без перезавантаження сторінки. Для Admin сервісу був використан підхід Backend For Frontend, що дозволяє вносити зміни до адмін панелі незалежно від змін клієнського коду. Також даний підхід допоміх відокремити функціонал адміністратору від функціоналу користувача і був закритий для них.

В цілому підхід з serverless архітектурою та розбиттям компонентів програми на незалежні модулі гарно себе зарекомендував та дозволив зменьшити вартість у порівняні з традиційним підходом до мікросервісної або сервіс орієнтованої архітектури і не витрачати велику кількість серверів під кожен з компонентів.

ВИСНОВКИ

В ході виконання атестаційної роботи була досліджена предметна область систем генерацій серверного коду у хмарних середовищах, було розглянуто та порівняно існуючі системи на основі інформації щодо загального функціоналу, підходам до створення api та документації а також недоліків були сформовані функціональні та не функціональні вимоги. Було створено архітектуру нової системи враховуючі недоліки у існуючих системах використовуючи мікросервісну та серверлес архітектуру на базі хмарного провайдеру AWS.

В результаті роботи вдалося створити програмну систему яка дозволяла створювати серверний код для роботи з базою даних та який мав підтримку мови запитів GraphQL також було створено систему неперервного моніторингу. Були використані практики неперервної інтеграції, розгортання та тестування. Також в результаті дослідження вдалося створити прозору цінову модель за використання і можливо було розраховувати з невеликою погрішністю ціну за інфраструктуру серверного коду.

В цілому можна сказати, що результат роботи виявив доцільність використовування технології GraphQl для автогенерованного серверного коду на початкових єтапах, а також використання серверлес підходу для створення ізольованих середовищ у хмарі через низьку вартість у порівнянні з традиційними підходами. Як результат були опубліковані тези на конференції.

Системи генерування серверного коду досить нові та набувають популярність, але все ж таки можна їх віднести до технологій які потрібні здебільшого стартапам або невеликим продуктам. Також варто відзначити складність цих систем у випадку великої кількості користувачів такими системами та у випадку великого додатку. Також варто відзначити що авто генерований код досить негативно впливає на підтримку розробником сгенерованого коду і є чорним ящиком для нього, але незважаючи на це сегмент таких систем буде зростати.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. История развития автоматизации: Материалы VI Международной студенческой научной конференции [«Студенческий научный форум»], (Москва, 2014) / Российская Академия Естествознания, 2014. – 25 с.

2. Groover, Mikell. Fundamentals of Modern Manufacturing: Materials, Processes, and Systems [Текст]. / Mikell Groover. – Wiley, 2012. – 1128 c.

3. Fischer, Bernd. Logical Foundations for Automated Code Generation / Bernde Fischer. // Estonian Summer School in Computer and Systems Science. – 2006. – №5.

4. Evans, Eric. Domain-Driven Design. Tackling the Complexity in the Heart of Software [Текст]. / Eric Evans. – Addison-Wesley, 2004. – 529 c.

5. Natalia Kravets, Khrystova A. (2020). Using lambda architecture for big data analysis // Abstracts of VI International Scientific and Practical Conference. Milan, Italy 2020. pp. 491-494 pp. Available at : DOI: 10.46299/ISG.2020.II.VI URL: https://isg-konf.com.

6. Sergiy Zagorodnyuk, Bohdan Sus, Oleksandr Bauzha, Ilona Revenchuk. (2020). Information Security of Users Rights Assignment via the Software Solutions Based on LDAP. Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020).Kharkiv, Ukraine- 6-9 October 2020.

7. Reinhard Wilhelm, Helmut Seidl. Compiler Design: Virtual Machines. [Текст]. – Springer, 2011. – 187 c

8. Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. [Текст]. –Addison-Wesley, 2015. – 368 c

9. Gabor Cselle, “Tales of Creation”. URL: http://blog.gaborcselle.com/2012/10/every-step-costs-you-20-of-users.html (дата звернення: 02.03.2021)

10. AWS documentation [Електронний ресурс]. - 2020р. - Режим доступу до ресурсу https://docs.aws.amazon.com/index.html

11. Docker [Електронний ресурс]. - 2020р. - Режим доступу до ресурсу https://www.docker.com/products/container-runtime

12. Docker – практичне керівництво [Електронний ресурс]. - 2020р. - Режим доступу до ресурсу https://habr.com/ru/post/310460/

13. Sergiy Zagorodnyuk, Bohdan Sus, Oleksandr Bauzha, Ilona Revenchuk. (2020). Information Security of Users Rights Assignment via the Software Solutions Based on LDAP. Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020).Kharkiv, Ukraine- 6-9 October 2020.

14. Natalia Kravets, Khrystova A. (2020). Using lambda architecture for big data analysis // Abstracts of VI International Scientific and Practical Conference. Milan, Italy 2020. pp. 491-494 pp. Available at : DOI: 10.46299/ISG.2020.II.VI URL: https://isg-konf.com.