МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ДЛЯ АНАЛІЗУ ПРОДУКТИВНОСТІ СПІВРОБІТНИКІВ ІТ-КОМПАНІЙ

Здобувач освіти гр. ІН – 81 Владислав БОЖЕНКО

Науковий керівник,
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую
Зав. кафедри Анатолій ДОВБИШ
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-81 спеціальності «122 – Комп'ютерні науки» денної форми навчання Боженка Владислава Сергійовича..

Тема: « »

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) аналітичний огляд існуючих рішень; 2) аналіз архітектури, сховищ даних, фреймворків для візуалізації метрику 3) проектування архітектури системи 4)програмна реалізація; 5) аналіз отриманих результатів

Дата видачі завдання «_____»_____ 2022 р.

Керівник роботи _____ Ігор ШЕЛЕХОВ

Завдання прийняв до виконання _____ Владислав БОЖЕНКО

РЕФЕРАТ

Записка: 73 сторінки, 27 рис., 4 таблиці, 13 джерел, 3 додатки.

Об’єкт дослідження — процес проєктування інформаційно-аналітичної системи

Мета роботи — розробка системи для аналізу продуктивності IT-робітників

Методи дослідження — інформації аналіз, моделювання архітектури, аналіз технологій, розробка кінцевого продукту

Результати — проведено аналіз конкурентів та необхідних технологій для реалізації системи. Спроектовано мікросервісну архітектуру з використанням сучасного стеку технологій. Реалізовано та наведено приклад використання системи в умовах IT-середовища.

МІКРОСЕРВІСНА АРХІТЕКТУРА, БАЗИ ДАНИХ,
ВІЗУАЛІЗАЦІЯ ДАНИХ, ІНФОРМАЦІЙНИЙ ПОРТАЛ

ЗМІСТ

ВСТУП

Аналіз ефективності робітників допомагає визначити сильні та слабкі сторони співробітника, забезпечує зворотний зв’язок з керівництвом, допомагає оцінити корпоративні та професійні якості. Все це можна описати спеціальними показниками, які можна розрахувати як основні характеристики, які допомагають зрозуміти, як оцінити компетенцію – і отримати міру тієї ж комунікативності чи автономності в числовому вираженні. Такий вид оцінки важливий для того, щоб виявити активних і результативних працівників, які є важливими для роботи команди та готові на певного роду підвищення. Він також необхідний для розрахунку найбільш продуктивного персоналу та допомагає визначити, який відділ має недостатньо потужності, щоб впоратися з поставленими завданнями завданням.

Мета роботи - розробка системи аналізу показників ефективності співробітників IT-компаній. Технічна частина системи має складатися з інтеграції з популярними системами IT-сфери (JIRA, Confluence, Github) та використання всіх корисних та необхідних даних для створення графіків, діаграм, розрахунків показників для створення візуалізацій показників ефективності працівників залежно від посад та обов'язків. Такі системи є унікальними для IT-ринку, дуже часто компанії намагаються розробити подібні системи самостійно. Цей факт доводить важливість і актуальність розглянутого проекту.

Завдання роботи - розробити систему, яка буде збирати всю необхідну інформацію з найпопулярніших систем менеджменту та контролю версій програмного забезпечення та надасть можливість аналізувати зібрану інформацію в зручному для користувача режимі, а також реалізувати можливість динамічно вираховувати метрики в залежності від критичних для компанії показників. Після закінчення розробки необхідно провести повну демонстрацію системи з чітким й зрозумілим поясненням всього наявного функціоналу, а

також навести документацію з конфігурації та налаштування системи для майбутніх користувачів.

1 ОГЛЯД ВІДОМИХ РІШЕНЬ

Аналіз конкурентів — це процес визначення конкуруючих систем у галузі та дослідження їхніх маркетингових стратегій та різних функціональних бенефітів. Цю інформацію можна використовувати як точку для порівняння, щоб визначити сильні та слабкі сторони нашої системи щодо кожного конкурента.[1] В цьому розділі ми плануємо провести аналіз конкурентів на абстрактному рівні та зосередити увагу на розгляданні існуючих порталів, які мають схожий функціонал: систем аналізу показників ефективності співробітників, що є популярними на сьогоднішній день. Перш за все необхідно проаналізувати їх недоліки та переваги, оцінивши актуальність та конкурентоспроможність нашої системи в умовах сучасного ринку. Після чого приведемо висновки щодо кожної системи, підкреслимо ключові характеристики на яких необхідно зосередити увагу.

1.1 Аналіз існуючих інформаційних порталів

1.1.1 Trakstar

Trakstar — це повністю автоматизоване хмарне рішення для оцінки продуктивності, яке спрощує процес перевірки працездатності співробітників. Це допомагає організації налаштовувати оцінки ефективності як для команд, так і для окремих співробітників, що надає процесу персоналізованої привабливості, відображаючи дії щодо огляду та надсилаючи автоматичні нагадування електронною поштою, Trakstar забезпечує регулярне сповіщення оцінювачів, щоб вчасно надавати відгуки співробітникам. Програмне забезпечення також надає користувачам інформацію у вигляді вичерпних і візуальних звітів та аналітики. У звітах містяться дані в режимі реального часу, які допомагають визначити основні сильні сторони співробітників і напрямки вдосконалення. Звіти можуть виявляти упередження оцінювачів, знаходити найкращих виконавців, порівнювати результати діяльності співробітників та переглядати історію роботи співробітника.

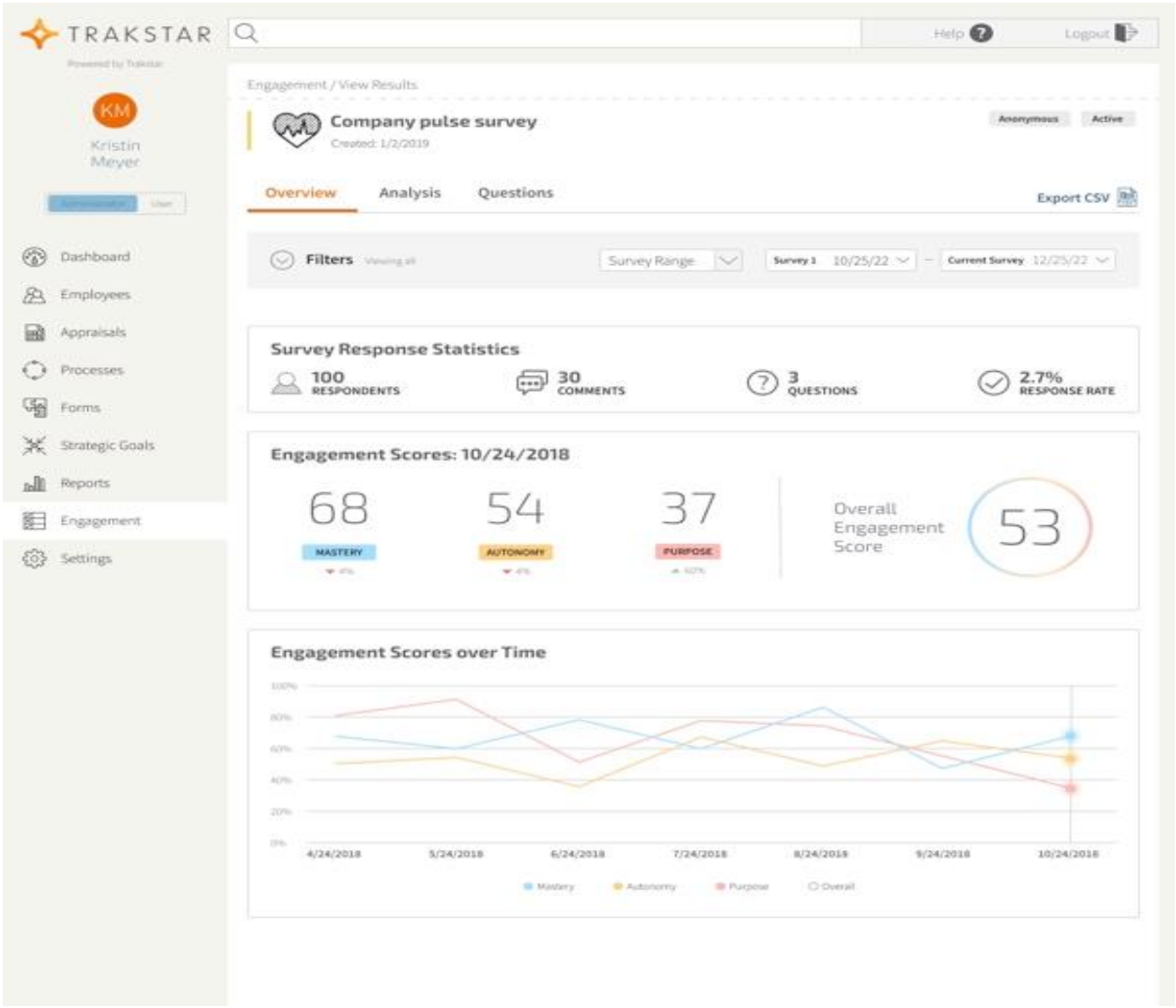


Рисунок 1.1.1 – Зовнішній вигляд веб-сайту Trakstar

1.1.2 BambooHR

BambooHR допомагає підприємствам отримати точне уявлення про свою робочу силу. Програмне забезпечення сприяє експертним оцінкам, що означає, воно надає користувачам детальні звіти про продуктивність, які допомагають менеджменту порівняти продуктивність співробітників і перфоменс в режимі реального часу. Програмне забезпечення також дозволяє менеджерам встановлювати цілі та контролювати їх прогрес за допомогою оглядових приміток, вбудованих у звіти. Ці візуальні звіти можна використовувати для пошуку областей покращення, визначення основних проблем і запобігання помилкам до того, як вони виникнуть у окремих осіб, відділів та організації вцілому.

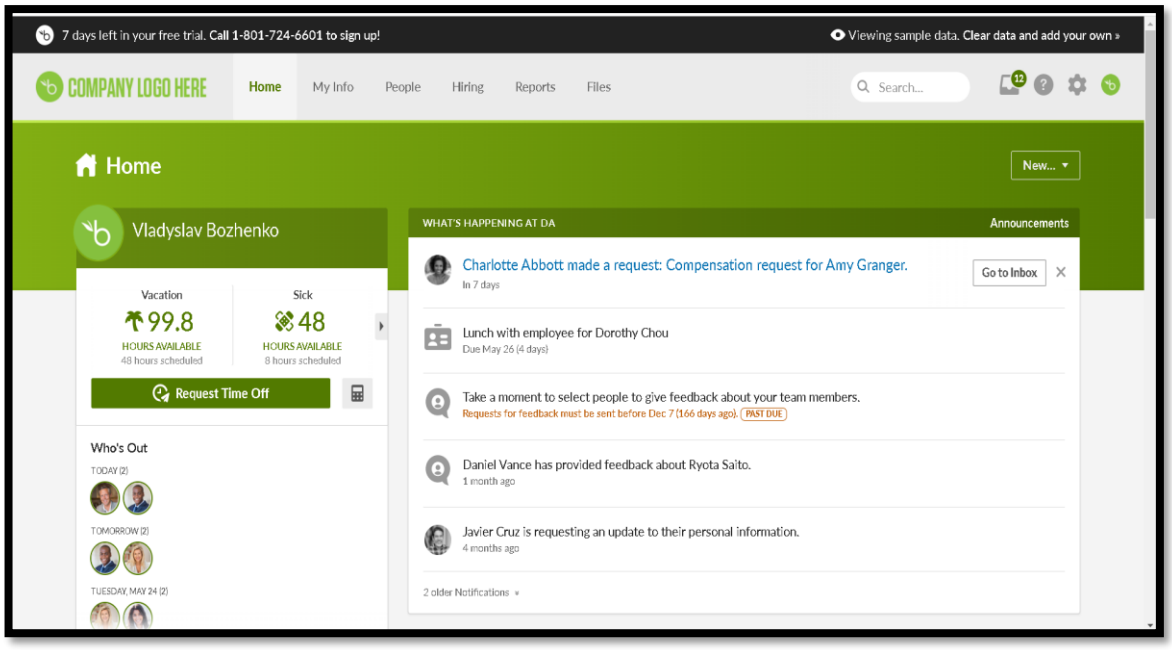


Рисунок 1.1.2 – Зовнішній вигляд системи BambooHR

1.1.3 Jira work management

Jira Work Management — це структурований інструмент керування роботою IT-команди, який дозволяє організаціям планувати й відстежувати роботу між командами та проектами в чітко визначеній структурі, що можна налаштувати. Jira Work Management — це простий та зручний вибір для організацій, які вже використовують програмне забезпечення Jira або керування послугами Jira або Atllassian вцілому.[13]

Рисунок 1.3 – Зовнішній вигляд системи Jira Work management

1.2 Порівняльна характеристика існуючих інформаційних порталів.

Під час оглядів інформаційних порталів було створена таблиця переваг та недоліків, що проаналізувати її та результаті отриманих даних зробити власний інформаційний портал.

Таблиця 1.1 – Результати порівняльного аналізу інформаційних порталів

Інформаційний додаток	Переваги	Недоліки
Trakstar	1. Зручний інтерфейс 2. Має можливість зручно будувати звіти та графіки . 3. Синхронізація та створення звітів у реальному часі	1. Платформа повністю на англійській мові, немає української локалізації 2. Платформа немає чіткої орієнтації на IT напрямок. 3. Відсутня інтеграція с популярними IT платформами та немає можливості динамічно конфігурувати звіти.
BambooHR	1. Можливість обирати план розвитку для кожного робітника визначати пріоритетні метрики в залежності від позиції 2. Частково Оптимізовано під айті сектор	1. Має чіткий пріоритет на HR напрямок 2. Відсутня можливість будувати більш технічні метрики
Jira work management	1. Платформа повністю оптимізована під айті сектор 2. Є інтеграції з сервісами atllassian користувачів 3. Присутні основні необхідні звіти та вирахування метрик	1. Відсутня можливість використовувати всі дані присутні в системах Atlassian, для створення звітів та вираховування метрик. 2. Відсутня інтеграція з системами контролю версій, що є критичним для оцінки продуктивності розробників.

Отримавши результат аналізу, наведений вище, було вирішено, що розробити власну систему для оцінки перфоменсу робітників галузі інформаційних технологій є доцільним, оскільки наша система зможе покрити недоліки зазначенних систем та буде конкурентноздатною.

1.3 Постановка задачі

Нами було поставлено задачу розробити систему, яка буде збирати всю необхідну інформацію з найпопулярніших систем менеджменту(Jira, Confluence) та контролю версій програмного забезпечення, а також реалізувати можливість динамічно вираховувати метрики в залежності від критичних для компанії показників. Таким чином, основними функціональними вимогами для нашої системи є:

1) IAC повинна інтегруватися з Jira, Confluence, Github системами.
2) Додаток має виконувати аналіз активності кожного робітника на основі створених метрик.
3) Створені метрики повинні підраховувати KPI в залежності від позиції та ролі співробітника.
4) Додаток повинен містити метрики за замовчування та надавати можливість користувачу створити власні.
5) Додаток повинен підтримувати як можливість підрахунку метрик в фоновому режимі (кожен день – місяць), так і розрахувати метрику на вимогу користувача.
6) Результати калькуляцій повинні відображатися в вигляди діаграм та графіків.
7) Система повинна надавати можливість створювати звіт про роботу конкретного робітника.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Мікросервісна архітектура

Як і для будь-якого іншого аспекту розробки, все починається з формалізованих вимог. В даному розділі ми плануємо описати цей процес у рамках трьох кроків. Першим кроком у визначенні архітектури стане формування ключових запитів на основі вимог до додатка. Але замість того, щоб описувати запити у вигляді конкретних технологій взаємодії між процесами, дуже часто використовуються більш абстрактні підходи до проектування системних операцій.[6] Системна операція являє собою запит, який додаток повинен обробити. Поведінка кожної команди визначається у вигляді абстрактної доменної моделі, яка теж формулюється на основі зазначених вимог. Системні операції стають архітектурними сценаріями, що ілюструють взаємодію між різними сервісами системи. Другим кроком у цьому процесі буде розбиття на сервіси. Існує кілька стратегій. Одна з них бере початок в сфері бізнес-архітектури і полягає в тому, що сервіси повинні відповідати певній бізнес-функції. Інша передбачає організацію сервісів навколо під-областей в контексті предметно-орієнтованого проектування. В результаті сервіси будуть засновані на бізнес концепціях, а не на технічних аспектах.[5]

Третій крок у визначенні архітектури додатку полягає в описі API для кожного сервісу. Для цього сервісам призначаються всі системні операції, визначені на першому кроці. Операцію можна реалізувати у вигляді одного або декількох сервісів. В останньому випадку потрібно вирішити, як вони будуть взаємодіяти між собою, що зазвичай вимагає підтримки додаткових операцій з їхнього боку.[5]

Під час декомпозиції предметної області системи можуть виникнути певні труднощі, які нам необхідно вирішити в рамках проектування нашої інформаційної системи аналізу данних.

По-перше, при неправильному проектування можуть виникнути мережеві затримки. Після певного роду аналізу ми можем зробоити висновок, що певна схема розбиття непрактична через занадто частого обміну даними між сервісами.

По-друге, синхронна взаємодія знижує доступність системи. Можливо, доведеться вдатися до концепції автономних сервісів.

Третьою проблемою може статати необхідність підтримки узгодженості даних між сервісами. Останню перешкоду на шляху до декомпозиції пов'язано з так званими спільними класами, застосовуваними в різних частинах програми. На щастя, для їх усунення можна скористатися концепціями з предметно-орієнтованого проектування.[4]

2.2 Вибір Бази Даних

Одним з перших етапів проектування системи є вибір бази даних для зберігання інформації, пов’язаної з метриками та необхідних для побудови оцінки перфоменсу значень. Перш за все, треба зрозуміти та проаналізувати з якими даними ми працюємо(їх вид, структуру) і яку обробку ми маємо намір проводити з цими даними. Тож розглянемо типові рішення для зберігання даних[12]:

Рішення для кешування — є доцільним у випадку розробки системи з великим навантаженням на зчитування, щоб покривати вимоги запитів для зберігання великої кількості даних(проблема повних часові рамок) та задовольнити вимогу щодо низької затримки. Типовими прикладами є Redis або Memcached.

Сховище файлової системи — підходять для зберігання зображень або аудіо-/відеофайлів. Надзвичайно популярним прикладом є Amazon S3 або Azure Blob Storage.

Текстові пошукові систем та аналітичні механізми – використовуються для реалізації full-text пошуку, а також для побудови систем для аналітики. Популярним прикладом є Elasticsearch.

Elasticsearch — це розподілений пошуковий та аналітичний механізм, побудований на Apache Lucene. З моменту свого випуску в 2010 році Elasticsearch швидко став найпопулярнішою пошуковою системою і зазвичай використовується для аналітики логів, повнотекстового пошуку, аналізу безпеки, бізнес-аналітики та операційної аналітики.[17]

Тип сховища в першу чергу залежить від типу даних. Якщо інформація структурована і може бути представлена у вигляді таблиці, і якщо потрібно, щоб наші транзакції були атомарними, послідовними, ізольованими та довговічними (ACID), ми можемо використати реляційну базу даних. Наприклад, PostgreSQL.

Якщо властивості ACID не потрібні, то все одно можна спробувати використовувати реляційну базу даних або використовувати альтернативу NoSQL. Але якщо дані не мають чіткої структури, їх не можна представити у вигляді таблиці, то доцільним буде розглянути базу даних NoSQL: MongoDB, Cassandra, HBase, Couchbase тощо. І саме тут шаблон запиту стає вирішальним фактором. Слід зазначити, що ElasticSearch є окремим випадком документо-орієнтованої NOSQL бази даних.

Нами було вирішено зупинитися на ElasticSearch, тому що:

1. Нам не дуже принципово мати гарантії виконання ACID принципів, оскільки ми плануємо використовувати базу тільки для реплікації даних з інших систем, а в контексті роботи системи в цілому ElasticSearch не буде єдиним і головним сховищем даних.
2. ElasticSearch пропонує вбудовану платформу для відображення метрик – Kibana, саме ця платформа буде розглянута при виборі фреймворку для віртуалізації даних у наступних розділах.
3. ElasticSearch чудово справляється с великими об’ємами даних та чудово маштабується у випадку необхідності.
4. ElasticSearch є популярною пошуковою системою та аналітичним механізмом, тому має чудову підтримку в спільноті, а також просту і структуровану документацію.

5. ElasticSearch перевірена часом технологія, є класичною технологією у сфері аналізу данних.

Таблиця 2.2.1 – Результати порівняльного аналізу інформаційних додатків

Інформаційний додаток	Переваги	Недоліки
ElasticSearch	1. Популярне для аналітики рішення 2. Має вбудовані технології для відображення метрик 3. Чудово масштабується як вертикально, так і горизонтально 4. Має чітку документацію 5. Може чудово працювати в тандемі з іншими базами даними	1. Не підходить для використання у вигляді головного сховища даних 2. Потрібен певний час для того щоб розібратися в технології та зробити найпростіше рішення 3. Не покриває ACID принципи
PostgreSQL	1. Підходить для даних з чіткою структурою 2. Покриває ACID принципи 3. Є популярним опенсорс рішенням з величезною підтримкою зі сторони громади 4. Може інтегруватись з інструментами візуалізації даних та бути використано для аналізу бізнес даних	1. Погано масштабується горизонтально 2. Не має вбудованих технологій для візуалізації даних
MongoDB	1. Чудово масштабується як вертикально, так і горизонтально 2. На відмінну від ElasticSearch дане NoSQL чудово підходить для основного сховища даних 3. Може інтегруватись з інструментами візуалізації даних та бути використано для аналізу бізнес даних	1. Не має вбудованих технологій для візуалізації даних 2. Не покриває ACID принципи

2.3 Вибір фреймворку для візуалізації даних

У сучасному середовищі майже кожній компанії потрібна власна система інформаційних технологій (IT), щоб збирати величезну кількість даних, відстежувати їх, зберігати і, нарешті, аналізувати, щоб отримати можливість

оцінити дані візуально. Незалежно від того, як буде використовуватись отримана інформація, інструмент для візуалізації даних є вирішальним для більшості проектів. Візуальна обробка результатів моніторингу певних джерел даних називається візуалізацією інформації. Візуалізація надає змогу відображати інформацію, отриману за допомогою інструментів агрегації, у зручному для людини форматі.

Три популярних рішення для візуалізації інформації — Kibana, Grafana і Prometheus. У кожного з них є свої сильні та слабкі сторони. Загалом, рішення для візуалізації інформації варіюються від плагінів, які покращують існуючі звіти, до повних рішень для ведення звітів всебічними і вже привабливими графічними функціями. Розглянемо кожен із інструментів окремо.

Kibana — це рішення з відкритим кодом, яке дозволяє візуалізувати дані Elasticsearch. У цьому випадку ми будемо посилатися саме на дані цієї бази даних. Доступно багато різних типів графіків, включаючи традиційні стовпчасті, лінійні та кругові діаграми. Якщо є необхідність розширити існуючу функціональність, то це її додати це власноруч. Ця можливість робить Kibana гнучким варіантом, який можна використовувати без особливих налаштувань і налаштовувати в міру зростання ваших потреб. Якщо організація має кілька різних місцеположень, то треба використовувати будь-які геопросторові дані, пов’язані з вашими даними, для створення звітів на основі місцезнаходження. Наприклад, можна використовувати карту США, щоб відобразити, скільки високо-пріоритетних сповіщень отримав системний адміністратор компанії за останній місяць у кожному місці. Візуалізація інформації у файлі логів географічно може дати додаткове уявлення про те, як найкраще розгорнути ресурси та обладнання. Географічні моделі також можуть допомогти в оцінці майбутніх потреб у співробітниках в кожному місці, якщо відстежується збільшення або зменшення середньої кількості високо пріоритетних подій з часом. Kibana підтримує кілька інших функцій звітності. Можна візуалізувати, як важливі події відбуваються протягом днів, а також у різний час доби. Це ще

один важливий спосіб визначити тенденції, що впливають на потреби в ресурсах і персоналі. Також можливо проаналізувати зв'язки між даними, об'єднавши можливості релевантності пошукової системи з функціями дослідження графіків Kibana.[16]

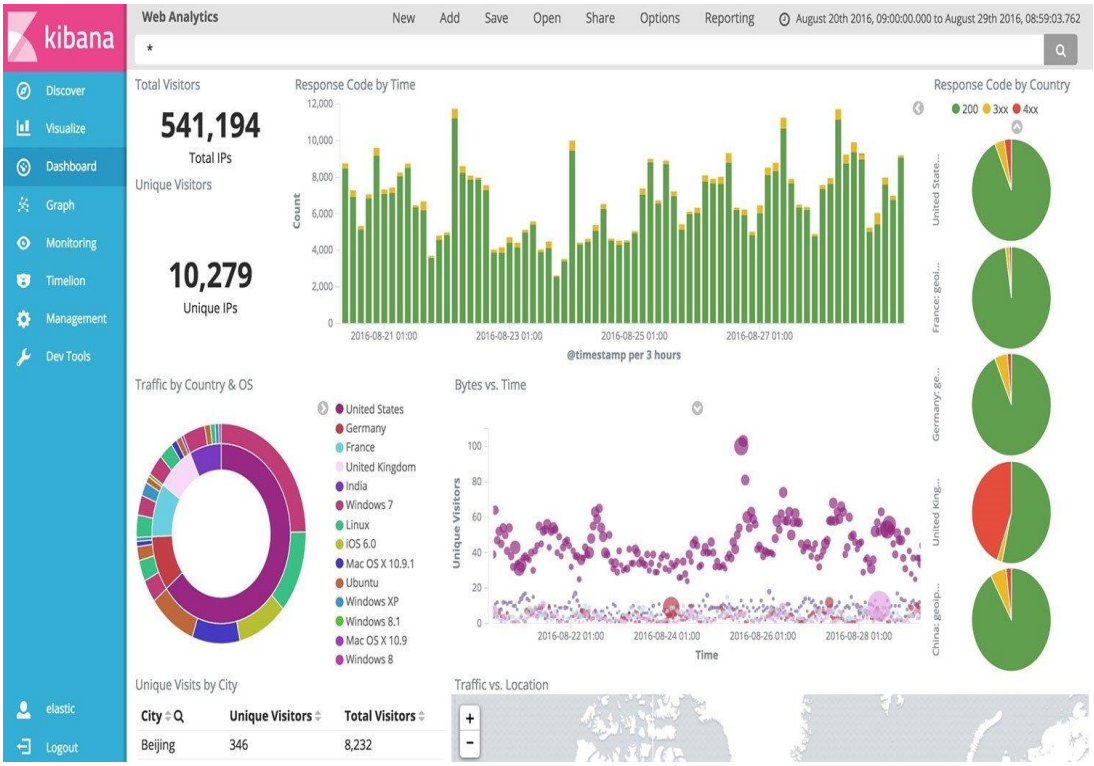


Рисунок 2.3.1 – Зовнішній вигляд системи BambooHR

Grafana — це платформа з відкритим вихідним кодом, яка використовується для побудови метрик, візуалізації даних, моніторингу та аналізу. Його мета — надати інформаційну панель візуалізації для відображення показників різного виду. Таким чином, це схоже на синергію між Kibana та Elasticsearch, оскільки Graphite є джерелом даних, а Grafana — програмним забезпеченням для візуальних звітів. Основна відмінність між цими двома зв'язками полягає в тому, що Grafana надає можливість отримання джерел з широкого діапазону сховищ даних, включаючи Elasticsearch. Це важлива функція для корпоративних організацій із широким спектром різноманітних рішень для зберігання даних. Основною метою Grafana є аналітика часових

рядів. Ці аналітичні дані обробляються з метою відображення їх на широкому спектрі параметрів. Теплові карти, гістограми та лінійні графіки – усі доступні варіанти для відображення даних у зрозумілий спосіб. Доступна система сповіщень, щоб повідомити, коли показники, які активно відстежуються, виходять за межі заздалегідь визначених порогових значень.

Рисунок 2.3.2 – Зовнішній вигляд системи BambooHR

Таблиця 2.3.1 – Результати аналізу інформаційних додатків

Інформаційний додаток	Переваги	Недоліки
Kibana	1. Дуже проста інтеграція з ElasticSearch 2. Велика кількість інструментів для візуалізації даних 3. Зручний інтерфейс та мова для для фільтрації даних	1. Інтегрується тільки з ElasticSearch
Grafana	1. Розроблялась на основі Kibana, тому має майже весь функціонал, що наявний у форк проекті 2. Інтегрується з більшістю наявних сховищ даних	1. Інтеграція зі сховищем даних більш складна ніж у Kibana

2.4 Вибір системи для потокової передачі даних

Потокова передача даних в режимі реального часу є критично важливою в області аналітики великих даних, а отже, є необхідною для розгляду в нашій системі. Потокова передача даних – це процес передачі безперервного потоку даних (також відомих як stream), який зазвичай подається в програмне забезпечення для обробки даних для отримання цінної інформації в режимі реального часу. Потік даних складається з серії елементів даних, упорядкованих у часі. Дані представляють собою «подію» або зміну стану, яку необхідно проаналізувати. Деякі приклади потоків даних включають дані датчиків, журнали активності з веб-браузерів і журнали фінансових транзакцій. Потік даних можна уявити як нескінченну конвеєрну стрічку, яка переносить елементи даних і безперервно подає їх у відповідний обробник. Зі зростанням популярності Інтернету речей (IoT) зросла значущість потокової передачі даних і обробки потоків.

Apache Kafka є класичним інструментом потокової передачі даних у реальному часі. Apache Kafka можна використовувати для керування піковими навантаженнями на прийом даних, а також як шину для трансляції великих об’ємів даних. Здатність Apache Kafka керувати піковими навантаженнями при прийомі даних є унікальною і значною перевагою перед звичайними механізмами зберігання даних. Загальне застосування Kafka знаходиться у серверній частині для інтеграції мікросервісів. Більшість платформ потокової передачі даних в реальному часі можуть ефективно інтегруватися з Kafka, щоб забезпечити аналітику потоків і обробку потоків. Kafka також може надсилати дані на інші платформи для потокової аналітики з метою аналізу. Оскільки Kafka є порівняно новою технологією, ніж інші, користувачам може бути трохи важко працювати з нею. Проте функції надмірності даних та відмовостійкості запропонували надійний підйом репутації Kafka, серед інших інструментів, що використовуються для потокової передачі даних.[14]

Redis Streams – один із популярних представників стріминг платформ. Концептуально потік у Redis — це список, до якого можна додавати записи. Кожен запис має унікальний ідентифікатор і значення. Ідентифікатор генерується автоматично за замовчуванням та містить позначку часу. Це дозволяє запитувати діапазони або використовувати команди блокування для читання записів у міру їх надходження. Потоки Redis ідеально підходять для створення брокерів повідомлень, що зберігають історію, черг повідомлень, уніфікованих журналів і систем чату. На відміну від повідомлень Pub/Sub, які видаляються через певний час, потоки Redis зберігають повідомлення назавжди. Redis Streams реалізують групи споживачів, функцію, яка дозволяє групі клієнтів співпрацювати під час споживання елементів із потоку[15].

Розглянемо основні переваги і недоліки обох стрімингових платформ в наступній таблиці:

Таблиця 2.4.1 – Результати порівняльного аналізу

Стрімингова система	Переваги	Недоліки
Kafka Streams	1. Висока продуктивність обробки повідомлень 2. Можливість масштабувати топіки як горизонтально так і вертикально 3. Наявність можливості паралельної обробки даних (Consumer Group)	1. Для конфігурації Kafka кластеру потрібно набагато більше часу ніж для Redis 2. Kafka використовує набагато більше ресурсів 3. Офіційний docker образ кафки не функціонує на платформі процесорів M1
Redis Streams	1. Висока продуктивність обробки повідомлень 2. Процес стрімингу відбуватиметься швидше відносно одніє партиції Kafka, оскільки Redis не зберігає дані на диск. 3. Наявність можливості паралельної обробки даних (Consumer Group) 4. Легкість конфігурації	1. Система реплікації master-slave допускає теоретично можливі втрати меседжів 2. Масштабування одного stream відбувається тільки вертикально

2.5 Проектування архітектури системи

Щоб побудувати ERD діаграму буде використано CaseStudioPortable, оскільки дана програма має доволі широкий функціонал та зручний для використання інтерфейс[9]. Однією з гарних частин даної програми є можливість генерувати SQL скрипт по побудованій ERD діаграмі. Загальний вигляд вікна програми можна побачити на рисунку 2.4.1:

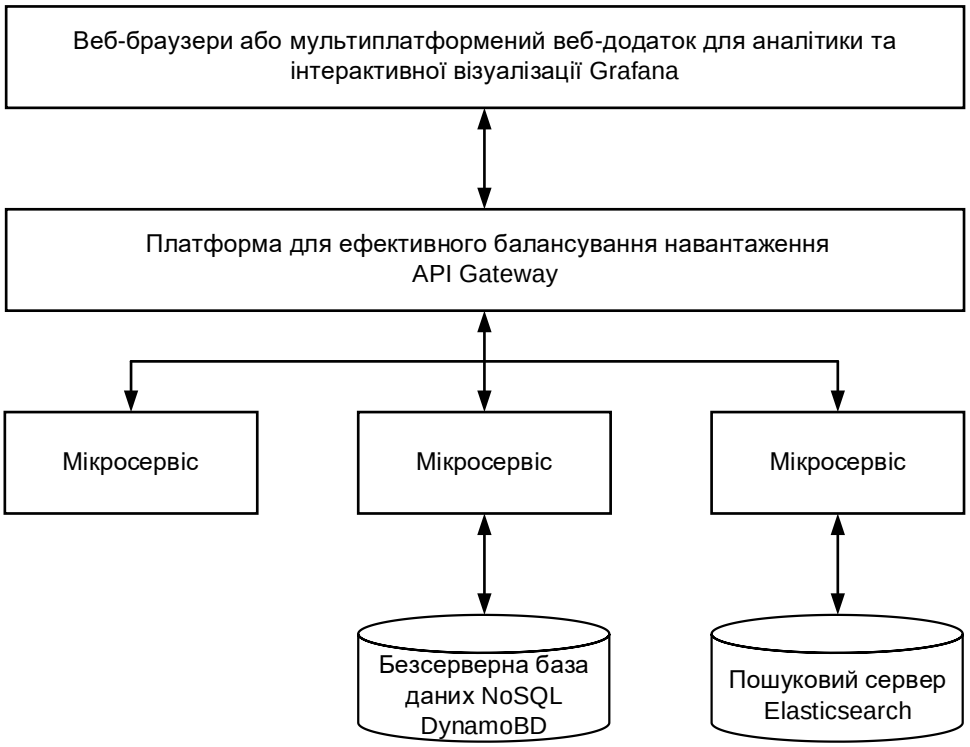


Рисунок 2.5.1 – Абстрактна діаграма компонентів системи

Діаграма компонентів, також відома як діаграма компонентів UML, описує організацію та підключення фізичних компонентів у системі.[5] Нами було створено діаграми компонентів, щоб проаналізувати деталі реалізації моделі та перевірити, чи кожен аспект необхідних функцій системи є охоплений запланованою розробкою. Було створено наступну Діаграму компонентів (рис. 2.6.2), на якій присутні всі наявні мікросервіси та технології, що використовуються в процесі роботи системи.

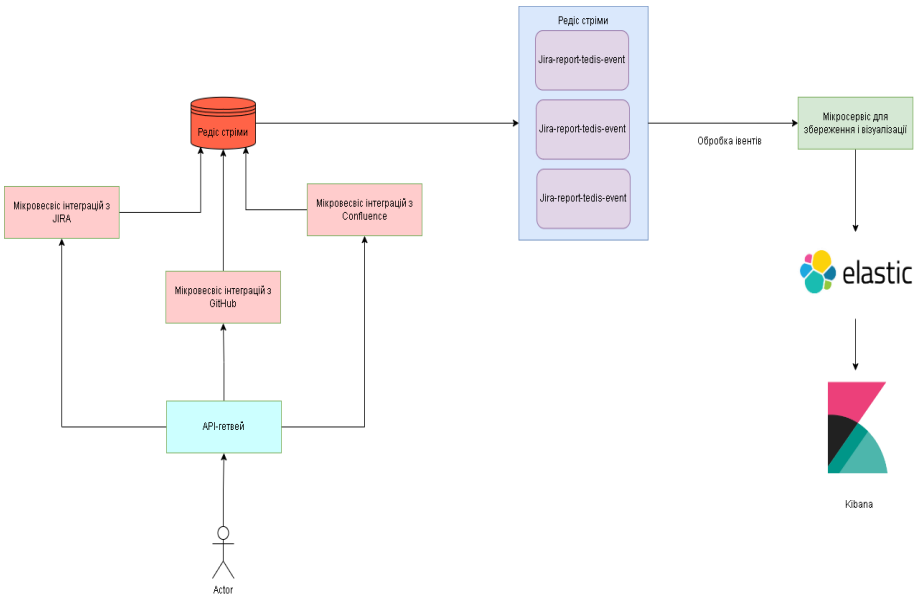


Рисунок 2.5.2 – Абстрактна діаграма компонентів системи

Діаграма послідовності демонструє взаємодію об’єктів, упорядкованих у часовій послідовності (див. Рис. 2.6.3), що допомагає визначити, які об’єкти та взаємодії необхідні для виконання функціональності. Діаграми послідовності є чудовими інструментами на початку проекту, оскільки вони крок за кроком показують, що і як має реалізовуватись на рівні взаємодії між компонентами системи[11]. Нами було створено діаграма послідовності до основної частини функціоналу системи.

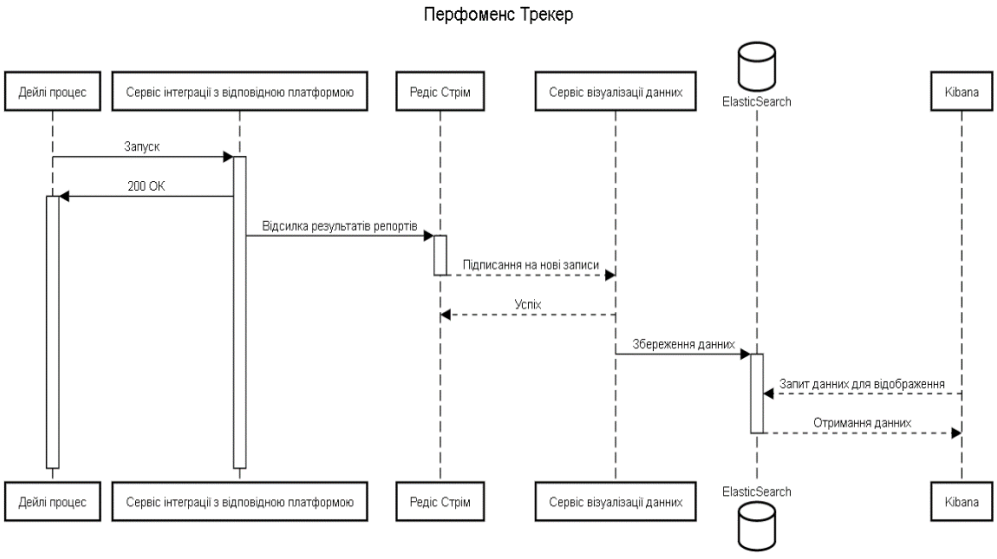


Рисунок 2.5.3 – Діаграма послідовностей системи

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Підготовка середовища

Найпершим етапом практичної реалізації є підготовка середовища. Вона включає підготовку всіх необхідних технологій, залежностей, даних, що є критично необхідними для функціонування та розробки програмного забезпечення. Для нашої системи такими речами є:

1 Створення Github аксес токена

2 Конфігурація Atlassian аккаунту, а також створення аксес токена

3 Наповнення даних для Atlassian систем

4 Запуск та розгорнення необхідних технологій

Перейдемо в налаштування аккаунту Github та створимо Access Token, обравши права тільки для читання:

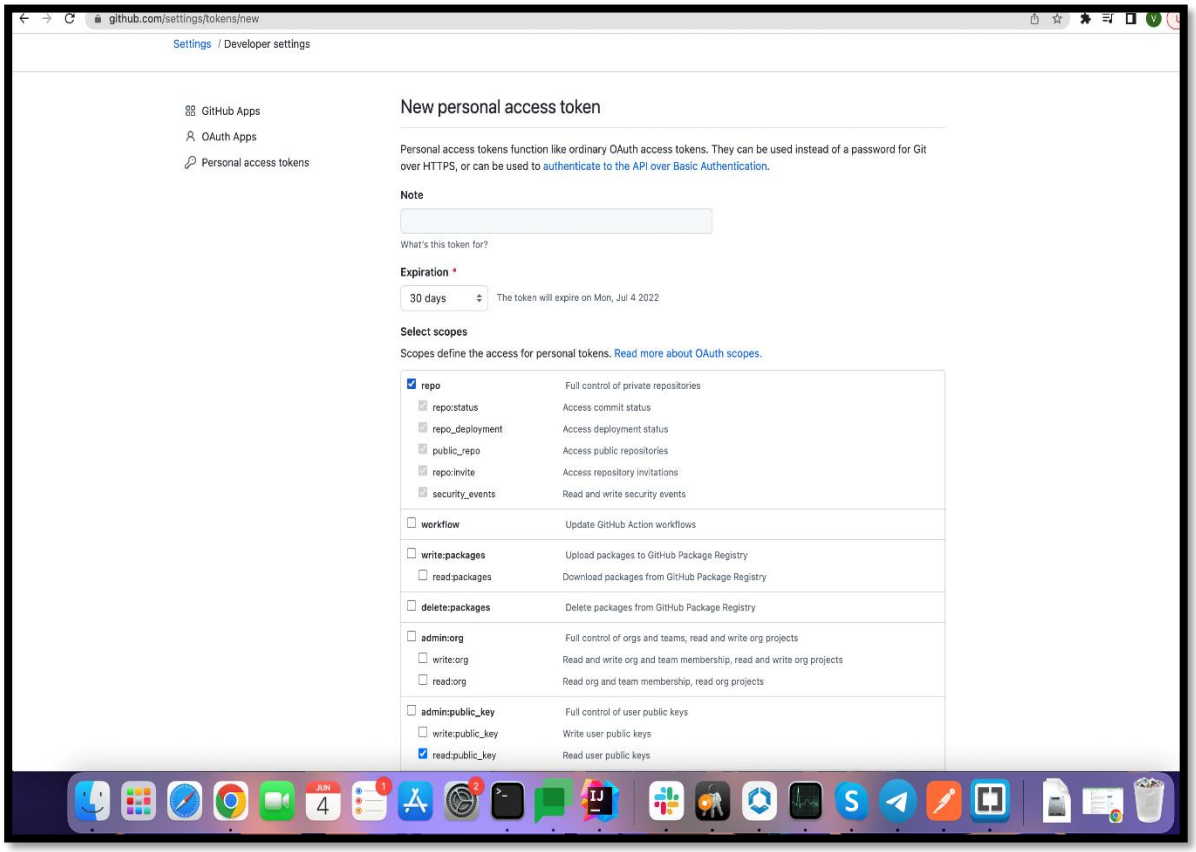


Рисунок 3.1.1 – Обрання прав нашої системи для Github

API Github — це API, яке можна використовувати для взаємодії з GitHub. Воно дозволяє створювати репозиторії, гілки, тікети, запити на витяг даних, що є найбільш цікавою опцією для нас. Для цих дій потрібно надати автентифікований маркер.

Нам потрібно пройти автентифікацію на сервері GitHub, але ми не хочемо розкривати наш пароль, який використовується з нашим обліковим записом GitHub, тому ми згенеруємо особистий маркер доступу, який буде використовуватися з командним рядком для автентифікації на GitHub.

Цей аккаунт гітхабу використовувався для багатьох пет-проектів в різних командах, а також в ньому відбувалась розробка нашої системи, тому ми можемо сміливо використовувати ці дані для подальшої обробки та аналізу перфоменсу робітників. Інтеграція з системою контролю версій є важливою, оскільки надає можливість слідкувати за прогресом програмістів, технічних лідів та інших технічних спеціальностей.

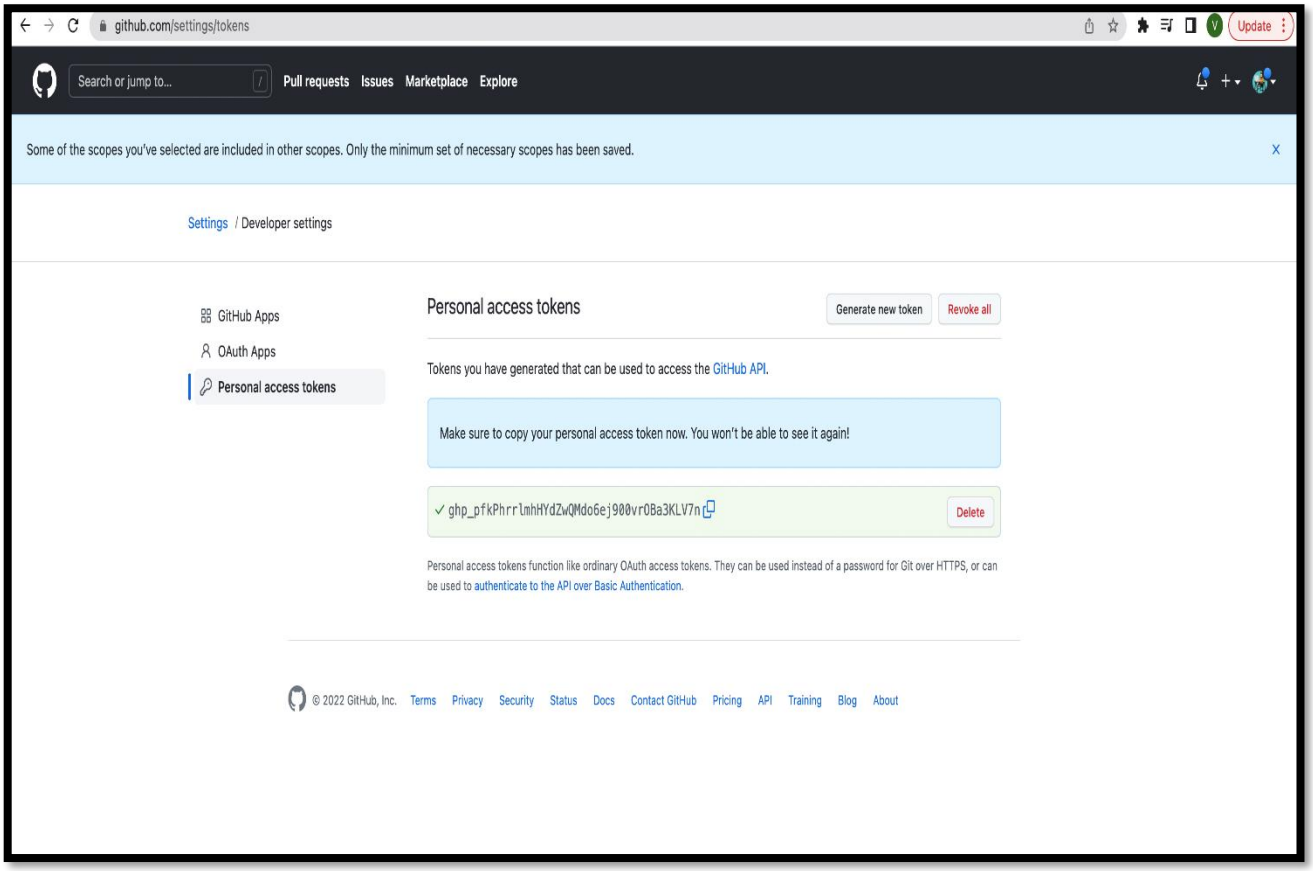


Рисунок 3.1.2 – Успішна генерація токену для Github

Нами було створено аккаунт Atlassian, а також Access Token для надання нашій системі доступів до даних Atlassian. Цей токен використовується для того, щоб надати сторонній системі доступ до даних з серверів Atlassian. Його створення э першим кроков щодо інтеграції нашої системи з такими системами як Jira, Confluence, Trello. Ці системи є одними з найважливіших джерел даних, бо є найпопулярнішими в IT суспільстві та використовуються кожною компанією. Вони позволять оцінити перфоменс будь-яких IT робітників та застосовуються для будь – яких позицій та рівнів. Саме тому ці інтеграції є критично важливими.

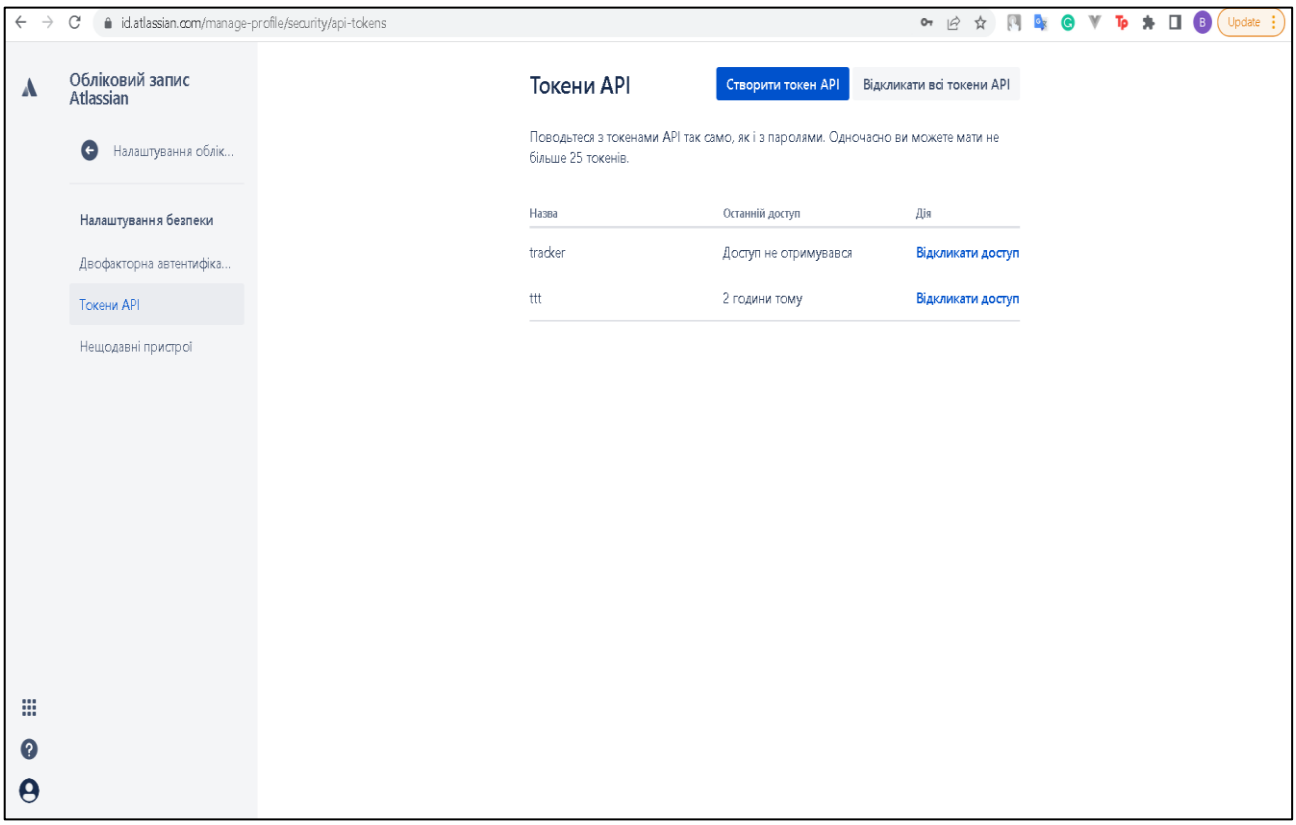


Рисунок 3.1.3 – Успішна генерація токену для Atlassian

В рамках цього аккаунту ми створювали тікети задачі та баги, стосовно нашої системи. Саме на основі цих даних ми будемо будувати звіти в наступних розділах. Дані можна вважати валідними, оскільки ми намагались імітувати процеси реальних команд в рамках створення цих активностей.

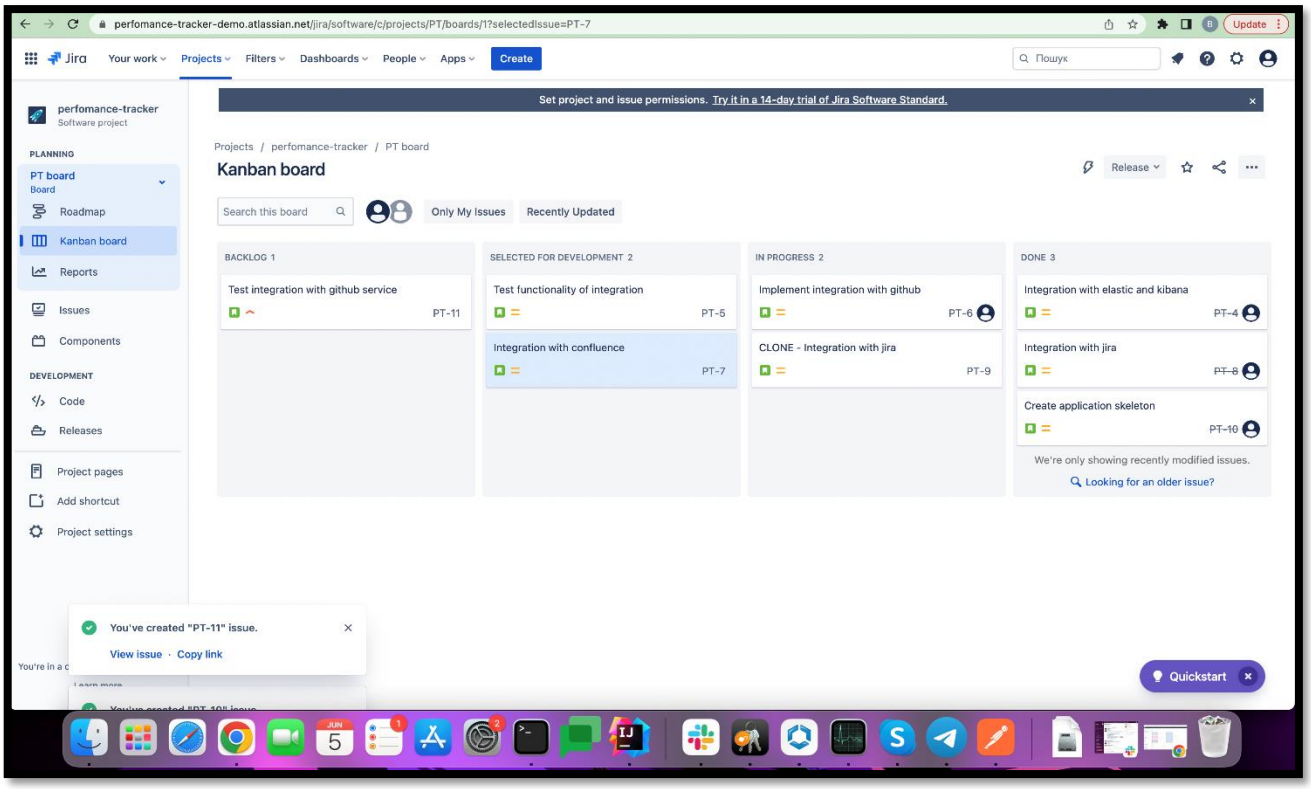
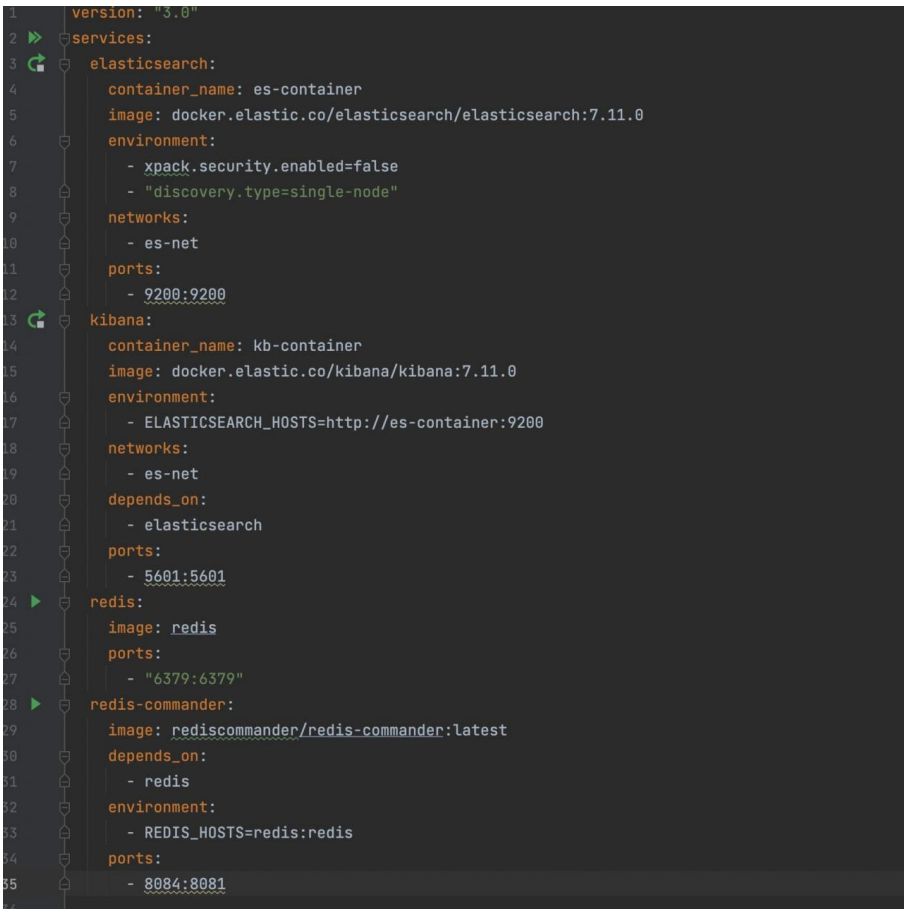


Рисунок 3.1.4 – Створені тікети для подальшого аналізу

Щодо запуску та розгорення необхідних технологій, то нами було створено наступний docker-compose файл, метою якого є запуск всіх необхідних залежностей для девелопменту в одну команду. В цей файл увішли інструменти, які було розглянуто у попередніх розділах, а саме ElasticSearch, Kibana, Redis, Redis-comander. Ці технології є важливою складовою системи, тому ми завчасно сконфігурували все необхідне для їх запуску у локальному середовищі. Для запуску цих інструментів будемо використовувати докер. Docker — це платформа для контейнерування з відкритим кодом. Це дозволяє розробникам упаковувати програми в контейнери — стандартизовані виконувані компоненти, що поєднують вихідний код програми з бібліотеками операційної системи (ОС) і залежностями, необхідними для виконання цього коду в будь-якому середовищі. Контейнери спрощують доставку розподілених додатків і стають дедалі популярнішими, оскільки організації переходять на власну хмарну розробку та гібридні мультихмарні середовища. Нами було створено наступну конфігурацію для контейнерів:



```yaml
version: "3.8"
services:
  elasticsearch:
    container_name: es-container
    image: docker.elastic.co/elasticsearch/elasticsearch:7.11.0
    environment:
      - xpack.security.enabled=false
      - "discovery.type=single-node"
    networks:
      - es-net
    ports:
      - 9200:9200
  kibana:
    container_name: kb-container
    image: docker.elastic.co/kibana/kibana:7.11.0
    environment:
      - ELASTICSEARCH_HOSTS=http://es-container:9200
    networks:
      - es-net
    depends_on:
      - elasticsearch
    ports:
      - 5601:5601
  redis:
    image: redis
    ports:
      - "6379:6379"
  redis-commander:
    image: rediscommander/redis-commander:latest
    depends_on:
      - redis
    environment:
      - REDIS_HOSTS=redis:redis
    ports:
      - 8084:8081
```

Рисунок 3.1.4 – Файл конфігурації докер-контейнерів для локального старту

3.2 Запуск інтеграцій та вивантаження даних

API для експорту даних надає можливість експортувати всі необхідні дані для майбутнього імпорту до сторонніх інструментів аналітики (наприклад, до нашої системи). Дані будуть зберігатись в нашій базі даних, яка є центральним джерелом інформації нашої системи візуалізації даних та створення репортів. API експорту Jira простий та має гарну документацію, яка відповідає на питання способу інтеграції нашої системи з серверами Atlassian. Дотримуючись методів API REST, в Jira можливо отримати всю інформацію про задачі за допомогою певної мови для вибірки даних - JQL. Ця мова використовується нами для того, щоб уточнити які дані(поля) та за який період нам необхідні. Оскільки наша команда достатньо добре знайома з розробкою Java, тож нами було використано attlassian-rest-api бібліотеку, оскільки це відмінний варіант передати запит JQL

через Java API. Розглянемо як можна запустити інтеграцію з Jira та Github використовуючи наш сервіс.

Нами було створено декілька основних енд-поінтів у відповідних мікросервісах , які відповідають за інтеграцію за отримання даних з необхідних систем. Перший з них /jira, що основна функція – запустити інтеграцію з Jira на основі певного періоду часу.

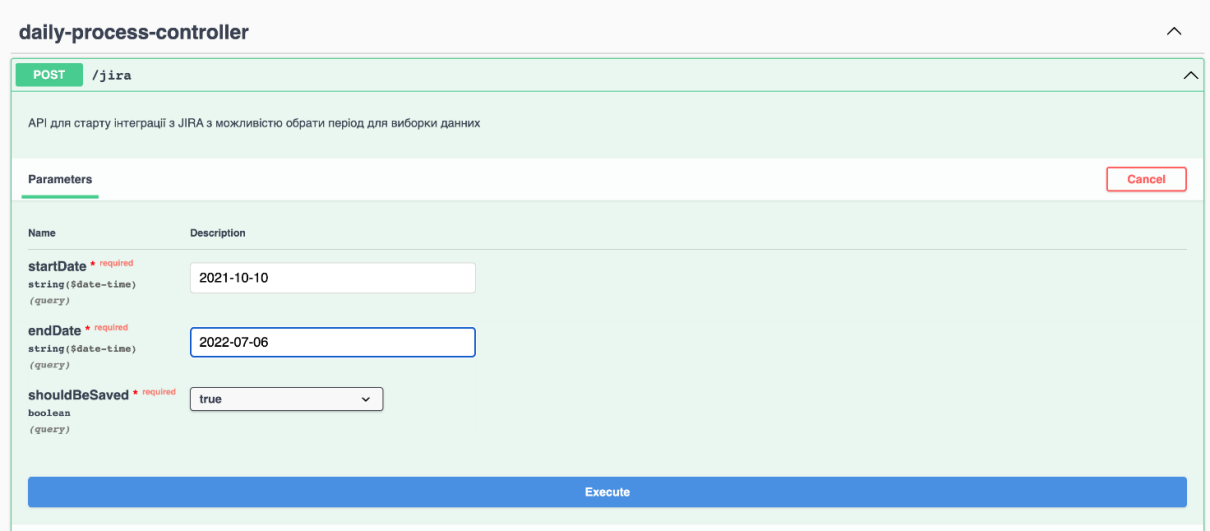


Рисунок 3.2.1 – Контроллер, надає можливість запустити інтеграцію з Jira

Схожий end-point було створено для інтеграції з github. API Github — це API, яке можна використовувати для взаємодії з GitHub. Воно дозволяє створювати репозиторії, гілки, проблеми, запити на витяг даних, що є найбільш цікавою опцією для нас. Розглянемо наше API для інтеграції з github:

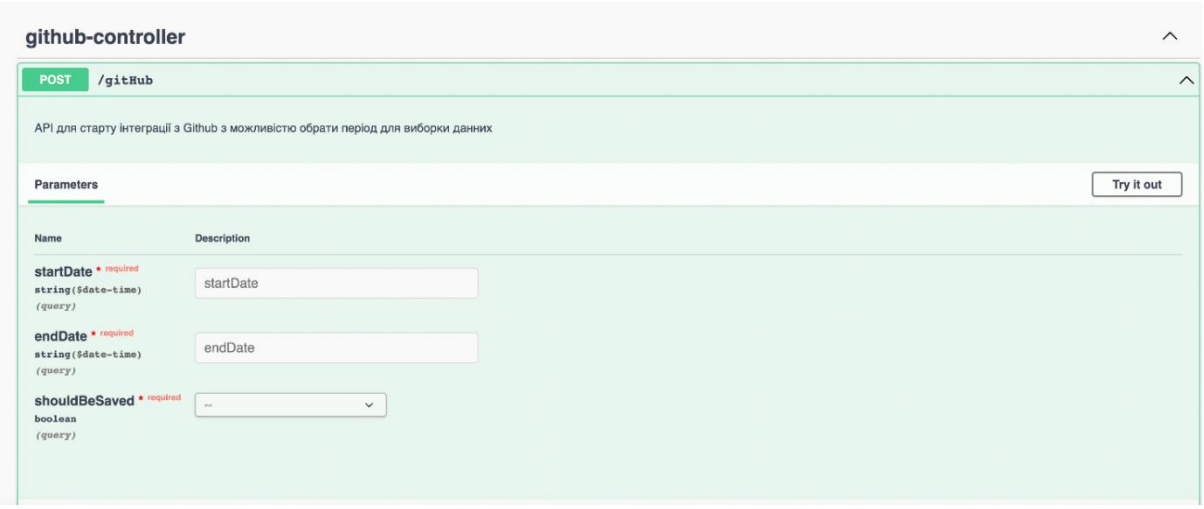


Рисунок 3.2.2 – Контролер для інтеграції з Github

Kibana потрібен шаблон індексу для доступу до даних Elasticsearch, які ми хочемо проаналізувати. Шаблон індексу вибирає дані для використання та дозволяє визначити властивості полів. Якщо дані було зібрано за допомогою одного з варіантів введення Kibana, завантаживши файл або додавши зразки даних, шаблон індексу буде створено автоматично, що надасть можливість одразу приступити до аналізу. Так як ми завантажили дані за допомогою нашої системи, то необхідно створити шаблони власноруч. Наприклад, наступний шаблон індексу було створено для github репортів:

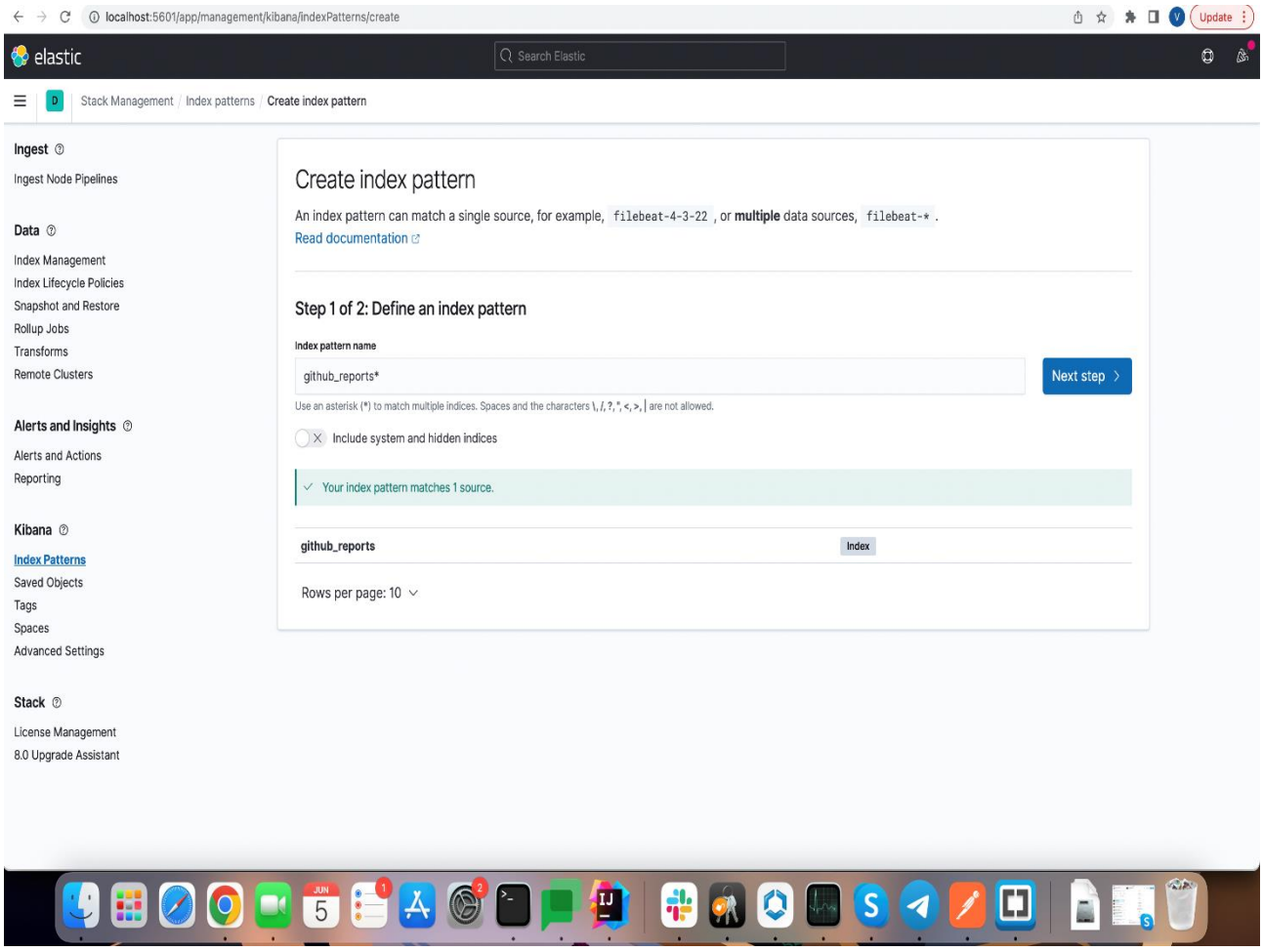


Рисунок 3.2.3 – Створення шаблону індексу

Тепер ми можемо переглянути дані, які зберігаються в нашій базі даних та переконатися, що інтеграція дійсно була завершена успішно:

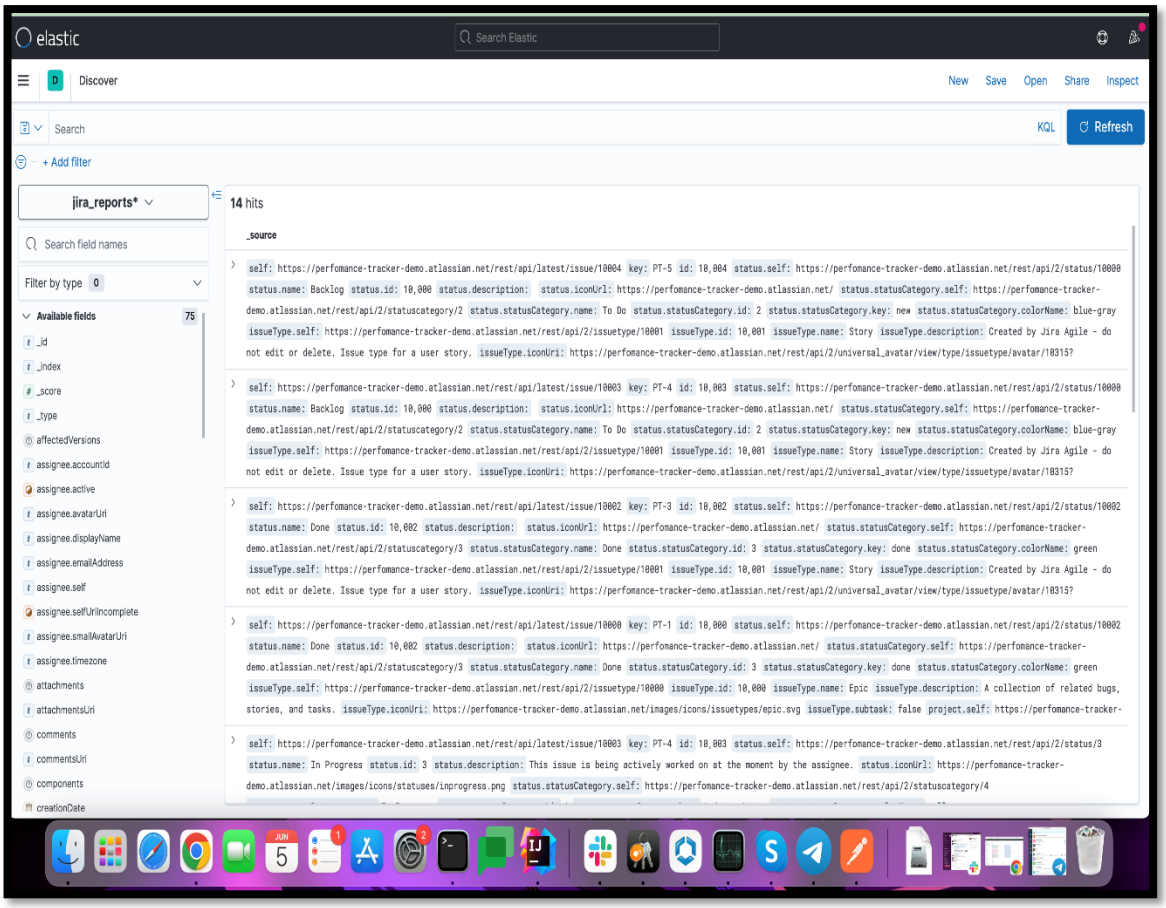


Рисунок 3.2.3 – Створення шаблону індексу

3.3 Візуалізація отриманих даних

Kibana має багато різних інструментів візуалізації даних з метою їх подальшої аналітики. В рамках огляду нашої системи ми будемо використовувати функцію Lens, оскільки вона є найбільш універсальною та зручною. Також є багато опцій візуалізацій на основі агрегації, для нашої системи можливо дуже багато сценаріїв, коли нам необхідно агрегувати за якимсь ключем. Kibana також дає змогу виявляти й аналізувати зв’язки в даних Elasticsearch, а також виявляти й досліджувати аномалії в даних Elasticsearch за допомогою функцій машинного навчання. Однією з переваг є те, робітники компаній, які відповідають за аналіз і збирання даних, мають можливість зручно ділитися зібраною інформацією з керівниками, колегами та клієнтами завдяки параметрам спільного доступу до інформаційної панелі Kibana. Також це надає

можливість створити базовий набір найпопулярніших метрик, який буде доступний одразу при старті системи у своєму середовищі.

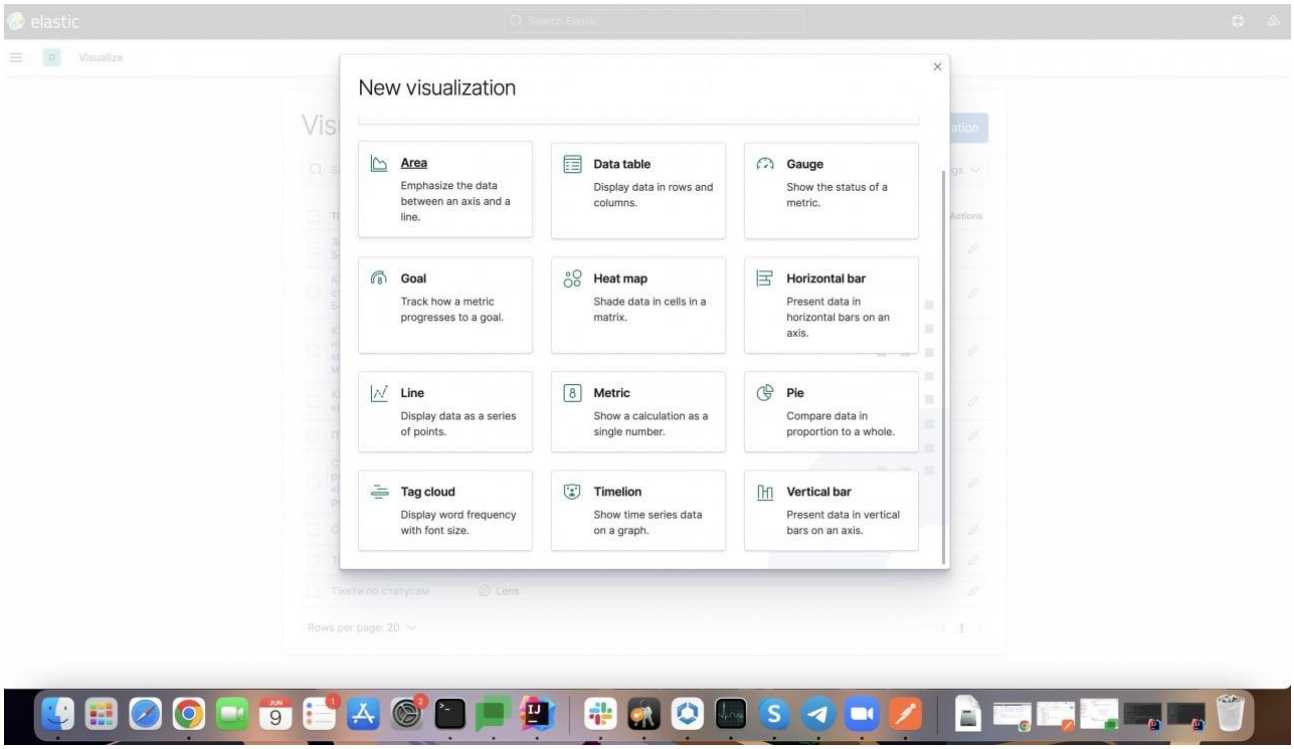


Рисунок 3.3.1 – Кількісних аналіз зробленої роботи з пов’язаної з кодом для кожного розробника

Розглянемо приклади сценаріїв використання візуалізації даних, які були зібрані нашими мікросервісами у попередньому розділі. Нами буде наведено базові сценарії генерації метрик та репортів, проте їх можна легко розширити створенням більш складних агрегаціях та розрахунків.

У всіх наших репозиторіях github використовувалось декілька користувачів (уявна команда з 4 людей). Цей аккаунт існує протягом довгого часу, тому в ньому є багато репозиторіїв різних пет-проектів, дані про які можна сміливо використовувати для аналізу Розглянемо та дослідимо, яку кількість модифікацій існуючого коду, додавання нового коду або ж видалення існуючого коду було зроблено кожним членом команди:

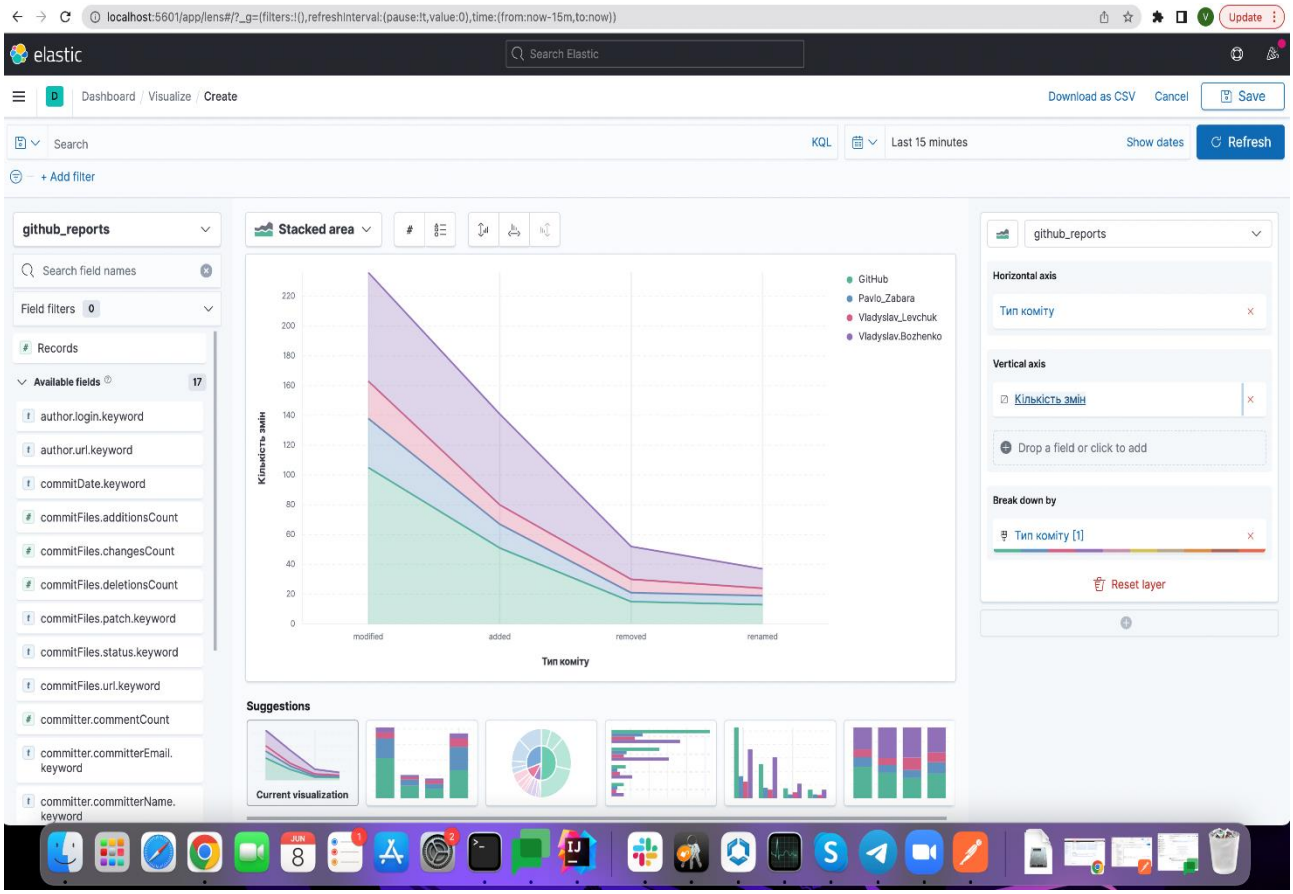


Рисунок 3.3.1 – Кількісних аналіз зробленої роботи з пов’язаної з кодом для кожного розробника

Тепер розглянемо певні метрики, використовуючи зібрані з системи Jira дані. В рамках Jira ми створювали тікети для нашого проекту на різних етапах розробки та намагались відстежувати кожну задачу. В рамках тестування нашої системи ці дані можуть використовуватись для оцінки пророблені роботи нашою командую в рамках. Давайте розглянемо обсяг роботи, який був закінчений/знаходиться в процесі або ж планується бути зробленим в майбутньому. Для цього побудуємо гістограму на основі статусів існуючих тікетів та прив’язки нашого імені до історії тікета, а також додамо фільтрацію по нашому логіну для того, щоб позбутися некоректних для нашого сценарію записів.

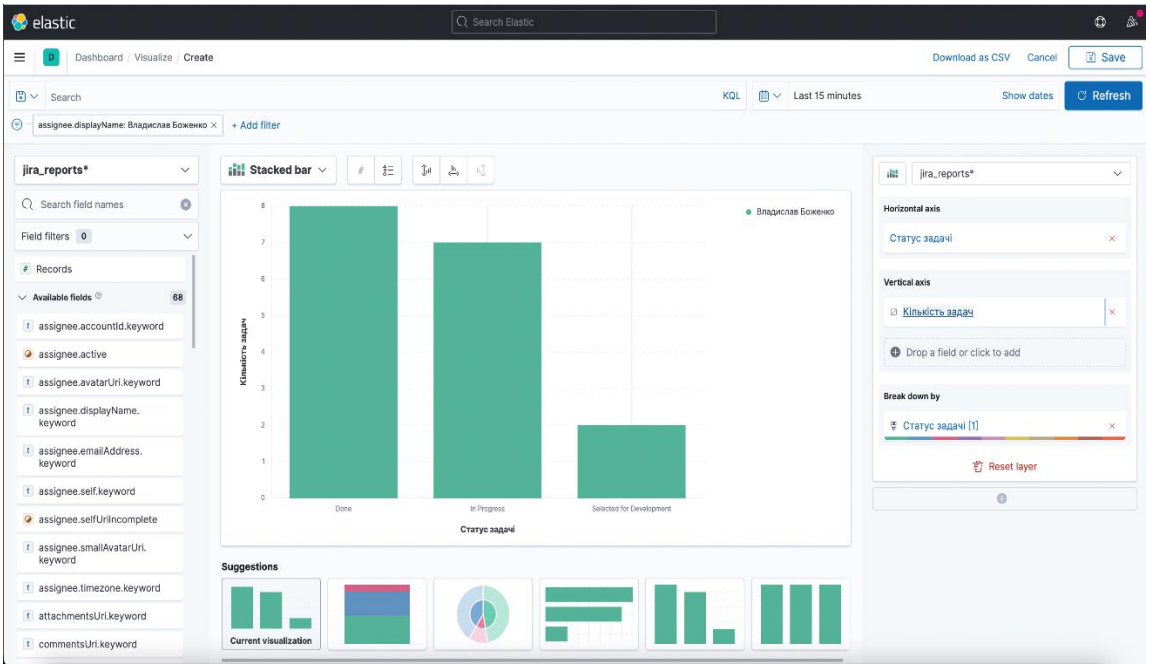


Рисунок 3.3.2 – Кількість виконаних задач, запланованих та задач, які виконуються зараз Владиславом Боженко

Наступим прикладом буде дослідження кількісті нашої команди, яку було створено для різних дат і погрупуємо їх по пріоритетам та по автору задачі:

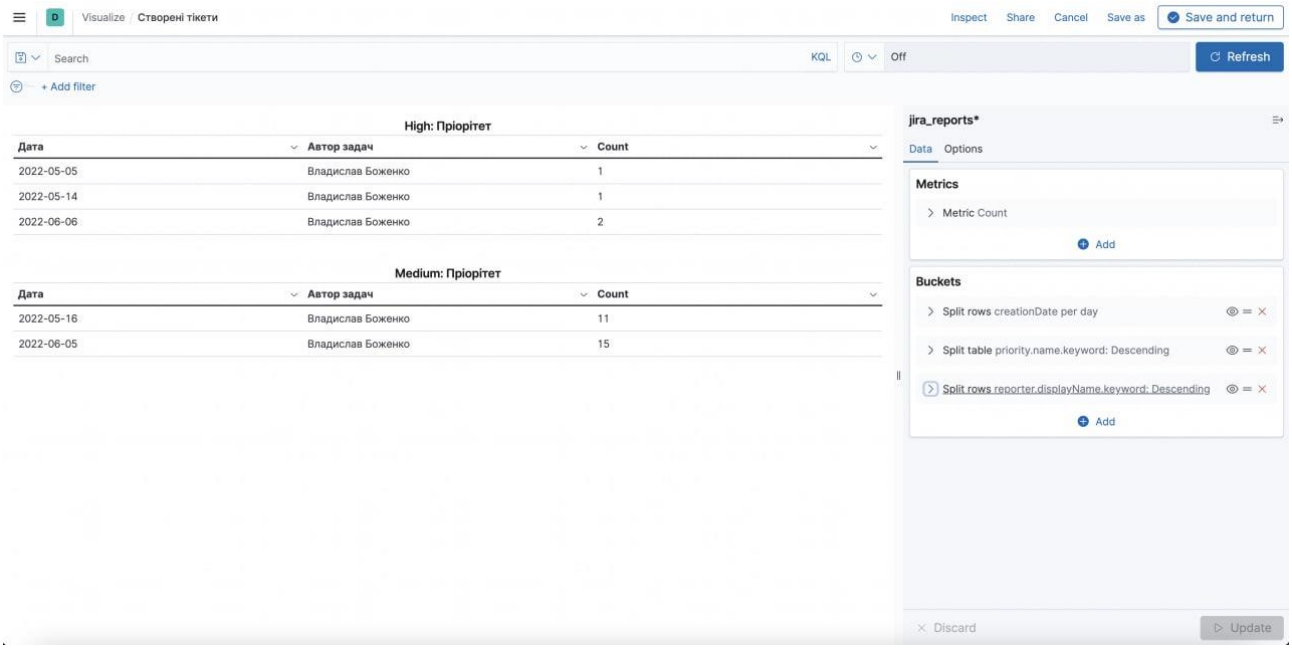


Рисунок 3.3.2 – Кількість створенних задач, погрупованних по даті створення, автору та пріоритету задачі

Розглянемо інший випадок. Наприклад нам необхідно оцінити навантаження для кожної компоненти (команди) нашого проекту. Для цього ми хочемо зрозуміти, яка загальна кількість задач була зроблена кожною командою:

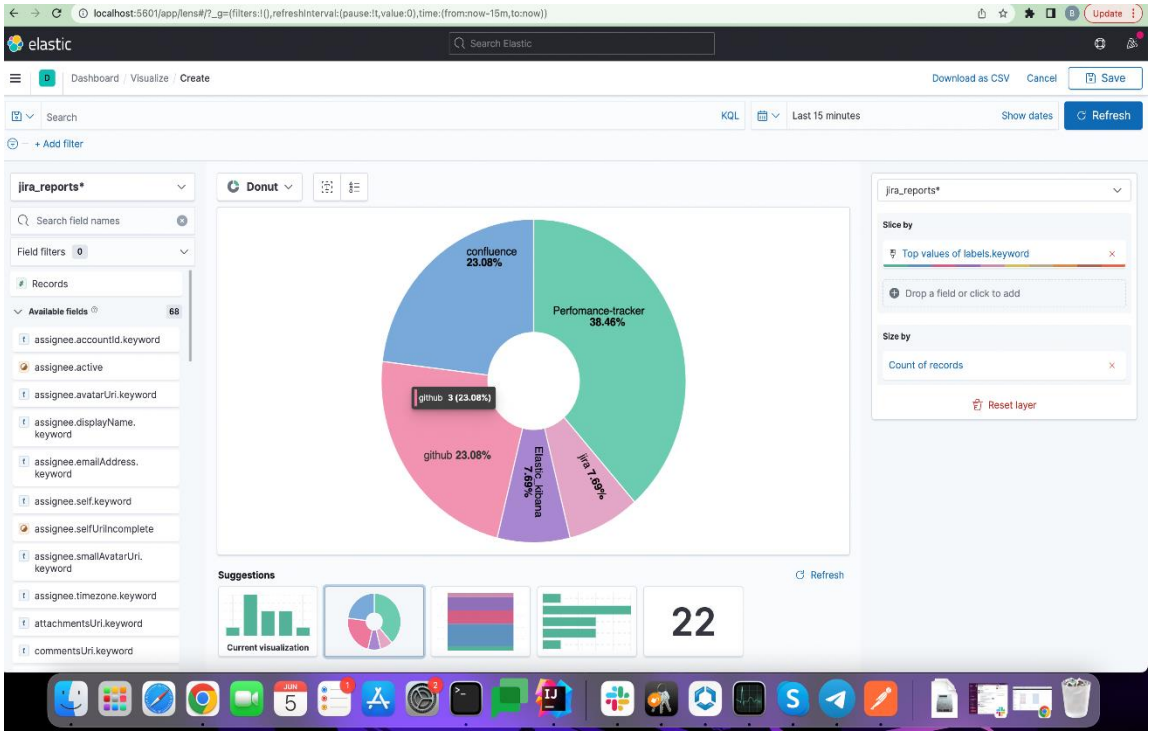


Рисунок 3.3.3 – Відсоток задач, виконаний кожною командою окремо

Схожу статистику можемо отримати для кожного мікросервіса окремо:

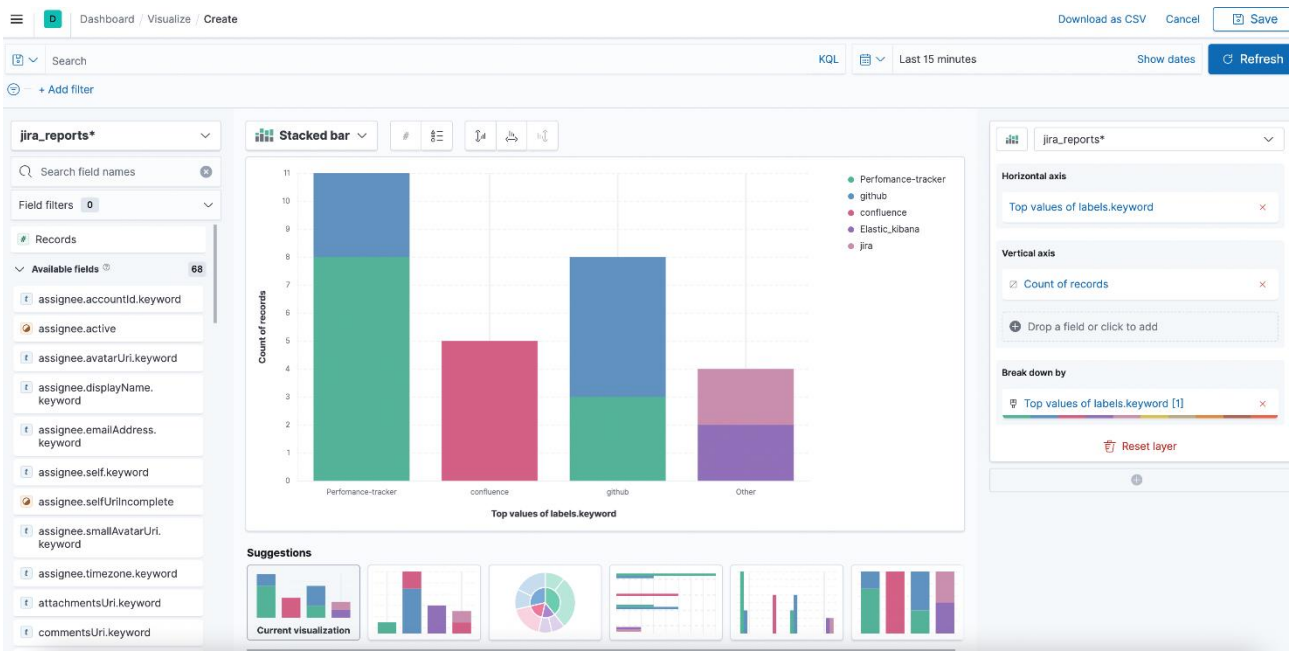


Рисунок 3.3.4 – Статистика задач по мікросервісах

Також є можливість оцінити кількість виконаної роботи з прив’язкою до дати та часу, по ним можна фільтрувати отримані результати або зробити графік, який відображає кількість зробленої роботи відносно конкретних дат:

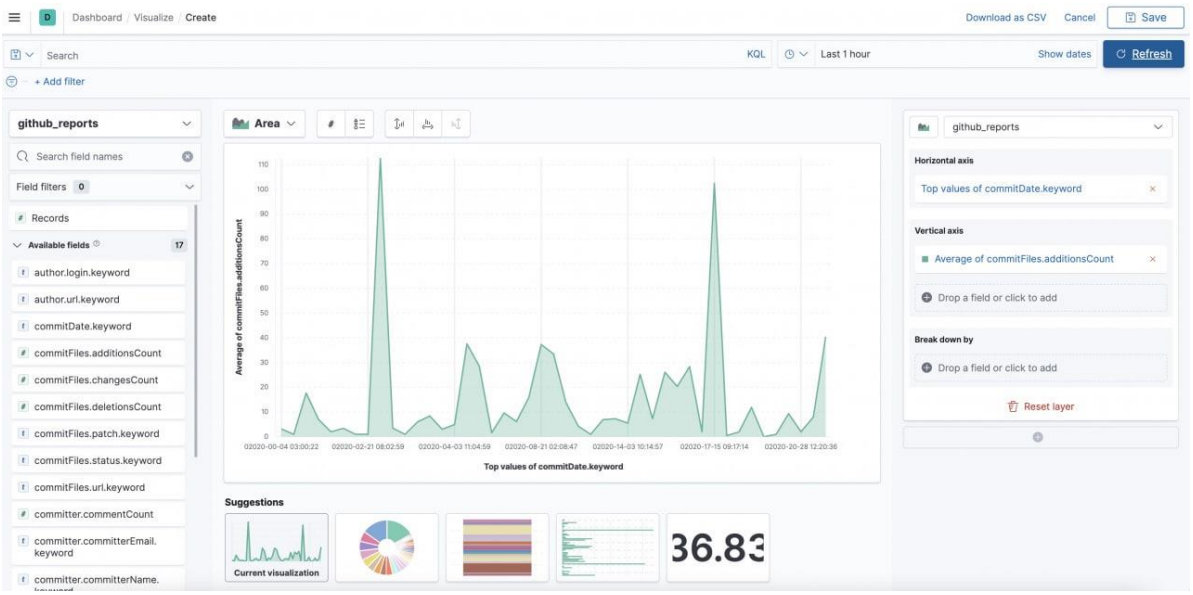


Рисунок 3.3.4 – Графік зробленої роботи відносно дати і часу

Отримані метрики можна зберегти для подальшого перегляду (на основі нових даних) або ж для того, щоб розповсюдити їх серед колег:

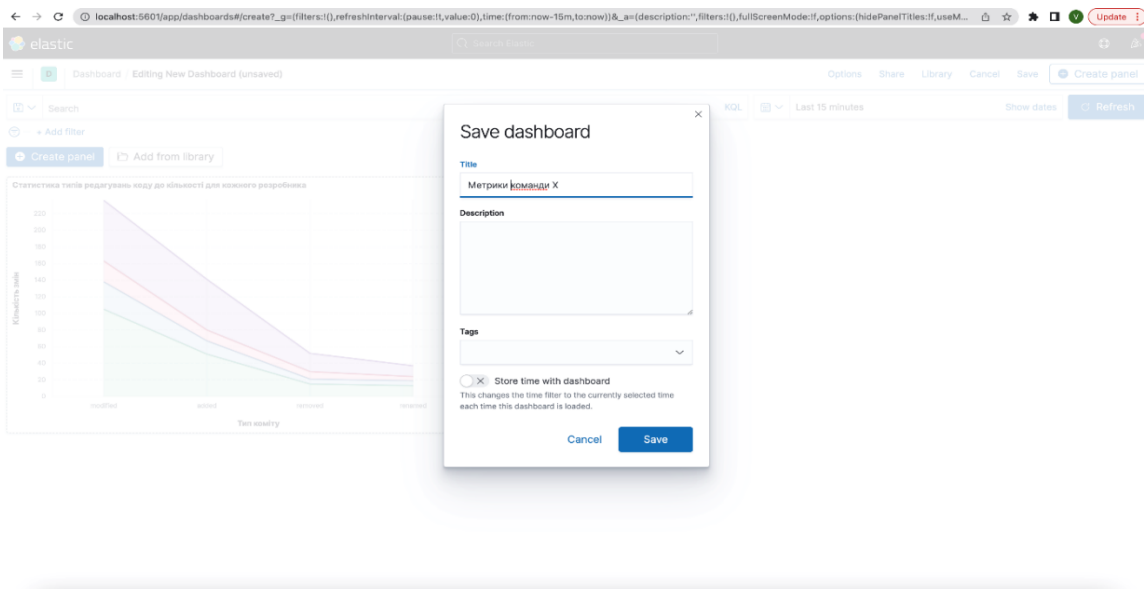


Рисунок 3.3.5 – Приклад збереження конкретної метрики

Тепер можемо переглянути всі візуалізації одночасно:

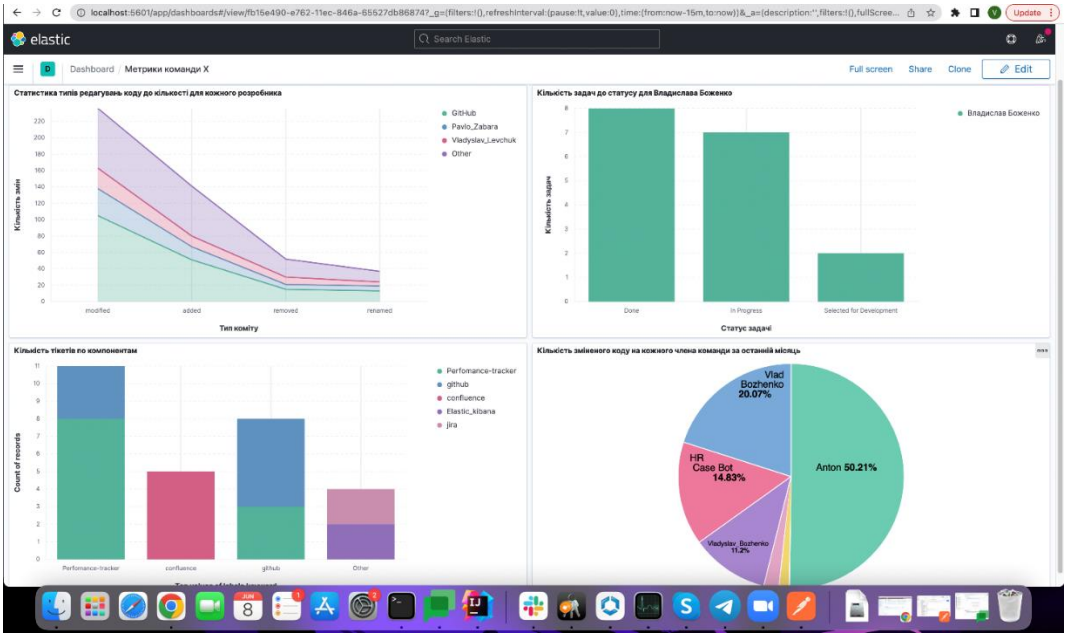


Рисунок 3.3.5 – Збережені метрики

Також система дозволяє вираховувати кількісний результат за наведеними умовами, лімітами або ж формулами. В даному випадку ми порахували кількість відкритих задач за останній місяць для Боженко Владислава:

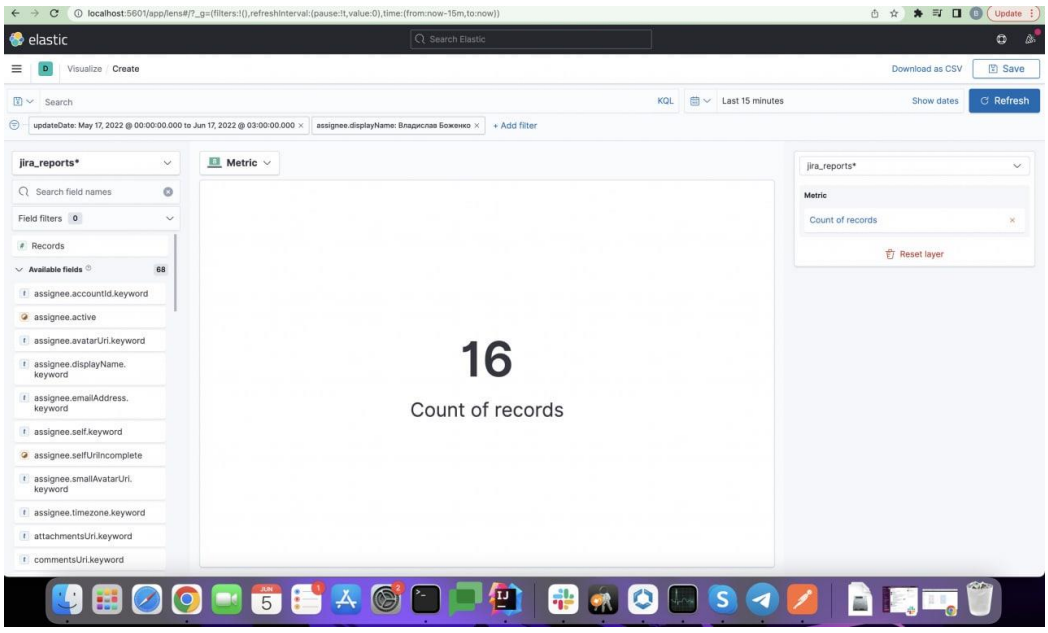


Рисунок 3.3.6 – Тестування розрахунку метрики

Таким чином, проведене тестування та демонстрація доводить працездатність розробленої інформаційно-аналітичної системи та її основних складових.

ВИСНОВКИ

У результаті виконання даної роботи було проаналізовано актуальність нашої системи, зроблено аналіз існуючих конкурентів, спроектовано архітектуру системи відносно зіставленого списку основних вимог . Нами було розглянуто можливе використання різних технологій: фреймворку для візуалізації даних, для потокової передачі даних, сховища даних, та підходу до архітектури в цілому. Оцінку актуальності кожної обраної технології було зроблено на основі порівняльного аналізу з конкурентами. Ми побудували основні архітектурні діаграми на основі яких було розпочато розробку системи для аналізу продуктивності робітників, що включає реалізацію всіх зазначених функціональних вимог. Нами було розроблено систему для аналізу продуктивності співробітників IT-компаній, а також наведено приклад експлуатації нашої системи, що включає основні необхідні етапи: конфігурація системи, її запуск, отримання даних з необхідних джерел інформації, а також приклади створення візуалізацій, підрахунку метрик. Отримані результати повністю задовольняють мету нашої роботи. Поставлене завдання було виконане повністю та покриває всі зазначені нами функціональні вимоги та цілі.

СПИСОК ЛІТЕРАТУРИ

1. Lehmann D., Winer R. Analysis for Marketing Planning - McGraw-Hill/Irwin, 2007. — 314 p
2. Stackoverflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com.
3. Draw.io [Електронний ресурс] – Режим доступу до ресурсу: https://app.diagrams.net/
4. Carneiro Cloves, Schmelmer Tim. Microservices From Day One, 2017. – 258p.
5. Richardson Chris. Microservices Patterns: With examples in Java. Manning Publications, 2019. — 520p.
6. Sharma Rahul, Mathur Akshay. Traefik API Gateway for Microservices: With Java and Python Microservices Deployed in Kubernetes. Apress Media LLC., 2020. — 269p.
7. Кисельов. Є.Л. Комбінований метод неперервної інтеграції мікросервісів. — Конференція «Прикладна математика та комп'ютинг». — Київ, 2017.
8. Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools – Iuliana Cosmina, 2017. – 849p
9. Nadareishvili. Microservice Architecture: Aligning Principles, Practices, and Culture / I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen – O'Reilly Media, 2016 – 146 p.
10. Kuruvilla Jobin. JIRA Development Cookbook - Packt Publishing, 2016. – 598 p
11. Ansari Hasanraza. Learn Spring Cloud: Deep dives into various components that make Spring Cloud a very useful framework - Amazon, 2021. — 77 p
12. Altoros Engineering Team. The NoSQL Technical Comparison Report - Sunnyvale, CA: Altoros, 2017. — 52 p
13. Wright Rachel. JIRA Strategy Admin Workbook: Templates for the application administrator to set up, clean up, and maintain JIRA - Industry Templates, LLC, 2017. — 295 p.

14. Seymour Mitch. Mastering Kafka Streams and ksqlDB: Building Real - Time Data Systems by Example - O'Reilly Media, 2021. — 435 p.
15. Carlson Josiah L. Redis in Action - Manning, 2013. — 293 p.
16. Srivastava Anurag. Mastering Kibana 6 - Packt Publishing, 2018. — 425 p.
17. Srivastava Anurag. Learning Elasticsearch 7.x: Index, Analyze, Search and Aggregate Your Data Using Elasticsearch - BPB Publishing, 2021 — 310 p.
18. Spring Framework Documentation[Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview.

Додаток 1. Лістинг мікросервісу metrics-reader

Docker-compose.yml

```
version:
"3.0"
          services:
          elasticsearch:
          container_name: es-container
          image: docker.elastic.co/elasticsearch/elasticsearch:7.11.0
          environment:
          - xpack.security.enabled=false
          - "discovery.type=single-node"
          networks:
          - es-net
          ports:
          - 9200:9200
          kibana:
          container_name: kb-container
          image: docker.elastic.co/kibana/kibana:7.11.0
          environment:
          - ELASTICSEARCH_HOSTS=http://es-container:9200
          networks:
          - es-net
          depends_on:
          - elasticsearch
          ports:
          - 5601:5601
          redis:
          image: redis
          ports:
          - "6379:6379"
          redis-commander:
          image: rediscommander/redis-commander:latest
          depends_on:
          - redis
          environment:
          - REDIS_HOSTS=redis:redis
          ports:
          - 8084:8081

          networks:
          es-net:
          driver: bridge
```

Pom.xml

```
<?xml
version="1.0"
encoding="UTF
-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.6.7</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.bramix.perfomance.tacker</groupId>
	<artifactId>metrics-reader</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>metrics-reader</name>
	<description>metrics-reader</description>
	<properties>
		<java.version>17</java.version>
		<kotlin.version>1.6.21</kotlin.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.elasticsearch.client</groupId>
			<artifactId>elasticsearch-rest-client</artifactId>
			<version>7.7.1</version>
		</dependency>
		<dependency>
			<groupId>org.elasticsearch.client</groupId>
			<artifactId>elasticsearch-rest-high-level-
client</artifactId>
			<version>7.7.1</version>
		</dependency>
		<dependency>
			<groupId>org.elasticsearch</groupId>
			<artifactId>elasticsearch</artifactId>
			<version>7.7.1</version>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-redis</artifactId>
		</dependency>
```

```
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                        <groupId>com.fasterxml.jackson.module</groupId>
                        <artifactId>jackson-module-kotlin</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.jetbrains.kotlin</groupId>
                        <artifactId>kotlin-reflect</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.jetbrains.kotlin</groupId>
                        <artifactId>kotlin-stdlib-jdk8</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                        <optional>true</optional>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
                <dependency>
                        <groupId>org.kohsuke</groupId>
                        <artifactId>github-api</artifactId>
                        <version>1.303</version>
                </dependency>
        </dependencies>

        <build>

        <sourceDirectory>${project.basedir}/src/main/kotlin</sourceDirectory>

        <testSourceDirectory>${project.basedir}/src/test/kotlin</testSourceDire
ctory>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                                <configuration>
                                        <excludes>
                                                <exclude>
```

```
            <groupId>org.projectlombok</groupId>

            <artifactId>lombok</artifactId>
                                        </exclude>
                                </excludes>
                        </configuration>
                </plugin>
                <plugin>
                        <groupId>org.jetbrains.kotlin</groupId>
                        <artifactId>kotlin-maven-plugin</artifactId>
                        <configuration>
                                <args>
                                        <arg>-Xjsr305=strict</arg>
                                </args>
                                <compilerPlugins>
                                        <plugin>spring</plugin>
                        <configuration>
                                <args>
                                        <arg>-Xjsr305=strict</arg>
                                </args>
                                <compilerPlugins>
                                        <plugin>spring</plugin>
                                </compilerPlugins>
                                </dependency>
                        </dependencies>
                </plugin>
        </plugins>
    </build>

</project>
```

MetricsReaderApplication

```
package com.bramix.perfomance.tracker.metricsreader

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class MetricsReaderApplication

fun main(args: Array<String>) {
        runApplication<MetricsReaderApplication>(*args)
}

package com.bramix.perfomance.tracker.metricsreader.config

import org.apache.http.HttpHost
import org.elasticsearch.client.RestClient
import org.elasticsearch.client.RestHighLevelClient
import org.springframework.context.annotation.ComponentScan
import org.springframework.context.annotation.Configuration
```

ElasticsearchClientConfig

```
@Configuration
@ComponentScan(basePackages = ["com.bramix.perfomance.tracker.metricsreader"])
class ElasticsearchClientConfig {
   fun client(): RestHighLevelClient? {
      return RestHighLevelClient(
         RestClient.builder(HttpHost("localhost"))
      )
   }
}
```

ObjectMapperConfiguration

```
package com.bramix.perfomance.tracker.metricsreader.config;
```

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.text.SimpleDateFormat;

@Configuration
public class ObjectMapperConfiguration {

   private final static String DATE_FORMAT = "yyyyy-mm-dd hh:mm:ss";

   @Bean
   public ObjectMapper createObjectMapper() {
      var objectMapper = new ObjectMapper();
      objectMapper.setSerializationInclusion(JsonInclude.Include.NON_NULL);
      objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);
      objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false);
      objectMapper.setDateFormat(new SimpleDateFormat(DATE_FORMAT));

     return objectMapper;
   }
}
```

RedisConsumerConfig

```
package com.bramix.perfomance.tracker.metricsreader.config;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.RedisSystemException;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.stream.Consumer;
```

```
import org.springframework.data.redis.connection.stream.ObjectRecord;
import org.springframework.data.redis.connection.stream.ReadOffset;
import org.springframework.data.redis.connection.stream.StreamOffset;
import org.springframework.data.redis.stream.StreamListener;
import org.springframework.data.redis.stream.StreamMessageListenerContainer;
import org.springframework.data.redis.stream.Subscription;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.time.Duration;

@Configuration
@Slf4j
public class RedisConsumerConfig {
    @Value("${stream.jira.key}")
    private String jiraStreamKey;

    @Value("${github.stream.name}")
    private String githubStreamKey;
    @Autowired
    private StreamListener<String, ObjectRecord<String, String>> jiraReportRedisConsumer;
    @Autowired
    private StreamListener<String, ObjectRecord<String, String>> githubReportRedisConsumer;



    @Bean
    public Subscription subscription(RedisConnectionFactory redisConnectionFactory) throws
UnknownHostException {
        StreamMessageListenerContainer.StreamMessageListenerContainerOptions<String,
ObjectRecord<String, String>> options = StreamMessageListenerContainer

.StreamMessageListenerContainerOptions.builder().pollTimeout(Duration.ofSeconds(1)).targetType(String.c
lass).build();
        StreamMessageListenerContainer<String, ObjectRecord<String, String>> listenerContainer =
StreamMessageListenerContainer
                .create(redisConnectionFactory, options);
        try {
```

```
            redisConnectionFactory.getConnection()
                    .xGroupCreate(jiraStreamKey.getBytes(), jiraStreamKey, ReadOffset.from("0-0"), true);
        } catch (RedisSystemException exception) {
            log.warn(exception.getCause().getMessage());
        }
        Subscription subscription = listenerContainer.receive(Consumer.from(jiraStreamKey,
InetAddress.getLocalHost().getHostName()),
                StreamOffset.create(jiraStreamKey, ReadOffset.lastConsumed()), jiraReportRedisConsumer);
        listenerContainer.start();
        return subscription;
    }

    @Bean
    public Subscription gitHubSubscription(RedisConnectionFactory redisConnectionFactory) throws
UnknownHostException {
        StreamMessageListenerContainer.StreamMessageListenerContainerOptions<String,
ObjectRecord<String, String>> options = StreamMessageListenerContainer

.StreamMessageListenerContainerOptions.builder().pollTimeout(Duration.ofSeconds(1)).targetType(String.c
lass).build();
        StreamMessageListenerContainer<String, ObjectRecord<String, String>> listenerContainer =
StreamMessageListenerContainer
                .create(redisConnectionFactory, options);
        try {
            redisConnectionFactory.getConnection()
                    .xGroupCreate(githubStreamKey.getBytes(), githubStreamKey, ReadOffset.from("0-0"), true);
        } catch (RedisSystemException exception) {
            log.warn(exception.getCause().getMessage());
        }
        Subscription subscription = listenerContainer.receive(Consumer.from(githubStreamKey,
InetAddress.getLocalHost().getHostName()),
                StreamOffset.create(githubStreamKey, ReadOffset.lastConsumed()),
githubReportRedisConsumer);
        listenerContainer.start();
        return subscription;
    }
}
```

GithubReportRedisConsumer

```
package com.bramix.perfomance.tracker.metricsreader.consumer;

import com.bramix.perfomance.tracker.metricsreader.service.GithubReportService;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.connection.stream.ObjectRecord;
import org.springframework.data.redis.stream.StreamListener;
import org.springframework.stereotype.Component;

@Component
@Slf4j
@AllArgsConstructor
public class GithubReportRedisConsumer implements StreamListener<String, ObjectRecord<String, String>> {

    private final GithubReportService githubReportService;

    @Override
    public void onMessage(ObjectRecord<String, String> message) {
        githubReportService.saveReport(message.getValue());
    }
}
```

JiraReportRedisConsumer

```
package com.bramix.perfomance.tracker.metricsreader.consumer;

import com.bramix.perfomance.tracker.metricsreader.service.JiraReportService;
import lombok.AllArgsConstructor;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.connection.stream.ObjectRecord;
import org.springframework.data.redis.core.ReactiveRedisTemplate;
import org.springframework.data.redis.stream.StreamListener;
import org.springframework.stereotype.Service;
```

```
import java.net.InetAddress;

@Service
@Slf4j
@AllArgsConstructor
public class JiraReportRedisConsumer implements StreamListener<String, ObjectRecord<String, String>> {

    private final ReactiveRedisTemplate<String, String> redisTemplate;
    private final JiraReportService jiraReportService;

    @Override
    @SneakyThrows
    public void onMessage(ObjectRecord<String, String> record) {
        log.info(InetAddress.getLocalHost().getHostName() + " - consumed :" + record.getValue());
        jiraReportService.saveReport(record.getValue());
    }
}
```

AuthorDto

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class AuthorDto {
    String login;
    String url;
}
```

CommitFileDto

```
package com.bramix.perfomance.tracker.metricsreader.model;
```

```
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.Value;

@Data
@NoArgsConstructor
public class CommitFileDto {
   String status;
   Integer changesCount;
   Integer additionsCount;
   Integer deletionsCount;
   String url;
   String patch;
}
```

CommitInfoDto

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class CommitInfoDto {
   String message;
   Integer commentCount;
   String committerName;
   String committerEmail;
}
```

GithubCommitDto

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
import lombok.NoArgsConstructor;
```

```
import lombok.Value;

import java.util.Date;
import java.util.List;

@Data
@NoArgsConstructor
public class GithubCommitDto {
    AuthorDto author;
    CommitInfoDto committer;
    OwnerDto owner;
    Date commitDate;
    List<CommitFileDto> commitFiles;
}
```

GithubReport

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.List;

@Data
@NoArgsConstructor
public class GithubReport implements Serializable {
    List<GithubCommitDto> ghCommits;
}
```

OwnerDto

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
```

```
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class OwnerDto {
    String name;
    String fullName;
    String language;
    String visibility;
}
```

GithubRepository

```
package com.bramix.perfomance.tracker.metricsreader.repository;

import lombok.AllArgsConstructor;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.xcontent.XContentType;
import org.springframework.stereotype.Repository;


@AllArgsConstructor
@Repository
@Slf4j
public class GithubRepository {

    private final RestHighLevelClient restHighLevelClient;

    @SneakyThrows
    public void saveGithubReport(String content) {
        IndexRequest request = new IndexRequest("github_reports");
```

```
        request.source(content, XContentType.JSON);
        IndexResponse response = restHighLevelClient.index(request, RequestOptions.DEFAULT);
        log.info("Response from elastic has been received", response);
    }
}
```

ReportRepository

```
package com.bramix.perfomance.tracker.metricsreader.repository;

import lombok.AllArgsConstructor;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.xcontent.XContentType;
import org.springframework.stereotype.Repository;

@Repository
@AllArgsConstructor
@Slf4j
public class ReportRepository {
    private final RestHighLevelClient restHighLevelClient;

    @SneakyThrows
    public void saveJiraReport(String content) {
        IndexRequest request = new IndexRequest("jira_reports");
        request.source(content, XContentType.JSON);
        IndexResponse response = restHighLevelClient.index(request, RequestOptions.DEFAULT);
        log.info("Response from elastic has been received", response);
    }
}
```

GithubReportService

```
package com.bramix.perfomance.tracker.metricsreader.service;

import com.bramix.perfomance.tracker.metricsreader.model.GithubReport;
import com.bramix.perfomance.tracker.metricsreader.repository.GithubRepository;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.AllArgsConstructor;
import lombok.SneakyThrows;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
public class GithubReportService {
    private final ObjectMapper objectMapper;
    private final GithubRepository githubRepository;

    @SneakyThrows
    public void saveReport(String reportDataJson) {
        objectMapper.readValue(reportDataJson, GithubReport.class).getGhCommits()
                .forEach(reportItem -> {
                    try {
                        githubRepository.saveGithubReport(objectMapper.writeValueAsString(reportItem));
                    } catch (JsonProcessingException e) {
                        throw new RuntimeException(e);
                    }
                });
    }
}
```

Додаток 2. Лістинг мікросервісу jra-service

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.6.7</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.example</groupId>
   <artifactId>perfomance-tracker</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>perfomance-tracker</name>
   <description>perfomance-tracker</description>
   <properties>
      <java.version>17</java.version>
      <jira.rest.client.version>5.2.2</jira.rest.client.version>
   </properties>

   <repositories>
      <repository>
         <id>atlassian-public</id>
         <url>https://packages.atlassian.com/maven/repository/public/</url>
         <releases>
            <enabled>true</enabled>
            <checksumPolicy>warn</checksumPolicy>
         </releases>
      </repository>
   </repositories>

   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-redis</artifactId>
      </dependency>
      <dependency>
```

```
            <groupId>com.atlassian.jira</groupId>
            <artifactId>jira-rest-java-client-app</artifactId>
            <version>5.2.0</version>
        </dependency>
        <dependency>
            <groupId>com.atlassian.jira</groupId>
            <artifactId>jira-rest-java-client-api</artifactId>
            <version>5.2.2</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

PerfomanceTrackerApplication

```
package com.bramix.perfomance.tracker;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PerfomanceTrackerApplication {

    public static void main(String[] args) {

        SpringApplication.run(PerfomanceTrackerApplication.class, args);
    }

}
```

JiraClientConfig

RedisConfig

```
package com.bramix.perfomance.tracker.configuration;

import com.atlassian.jira.rest.client.api.SearchRestClient;
import com.atlassian.jira.rest.client.auth.BasicHttpAuthenticationHandler;
import com.atlassian.jira.rest.client.internal.async.AsynchronousJiraRestClientFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.net.URI;
import java.net.URISyntaxException;

@Configuration
public class JiraClientConfig {

    @Value("${jira.uri}")
```

```
    private String uri;
    @Value("${jira.username}")
    private String username;
    @Value("${jira.token}")
    private String token;

    @Bean
    public SearchRestClient createJiraSearchClient() {
        var authenticationHandler = new BasicHttpAuthenticationHandler(username, token);
        URI serverUri;
        try {
            serverUri = new URI(uri);
        } catch (URISyntaxException e) {
            throw new RuntimeException("Unable to parse url + " , e);
        }

        return new AsynchronousJiraRestClientFactory()
                .createWithAuthenticationHandler(serverUri, authenticationHandler)
                .getSearchClient();
    }}
```

ObjectMapperConfiguration

```
package com.bramix.perfomance.tracker.configuration;

import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.map.SerializationConfig;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ObjectMapperConfiguration {

    @Bean
    public ObjectMapper createObjectMapper() {
        return new ObjectMapper()
                .setSerializationInclusion(JsonSerialize.Inclusion.NON_NULL)
                .disable(SerializationConfig.Feature.FAIL_ON_EMPTY_BEANS);
```

```
    }
}
```

```
package com.bramix.perfomance.tracker.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
public class RedisConfig {
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory) {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate();
        redisTemplate.setConnectionFactory(connectionFactory);
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(new StringRedisSerializer());
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(RedisSerializer.string());
        return redisTemplate;
    }
}
```

JiraDailyProcessService

```
package com.bramix.perfomance.tracker.service;

import com.atlassian.jira.rest.client.api.SearchRestClient;
import com.atlassian.jira.rest.client.api.domain.Issue;
import com.atlassian.jira.rest.client.api.domain.SearchResult;
import com.bramix.perfomance.tracker.redis.producer.JiraRedisEventProducer;
import com.bramix.perfomance.tracker.utility.JqlUtility;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Set;
import java.util.stream.IntStream;
import java.util.stream.StreamSupport;

@Service
@AllArgsConstructor
@Slf4j
public class JiraDailyProcessService implements DailyProcessService {
    private final SearchRestClient jiraClient;
    private final JiraRedisEventProducer jiraRedisEventProducer;

    private static final int BATCH_SIZE = 50;
    private static final int THREAD_SIZE = 5;
    private static final Set<String> FIELDS_TO_SEARCH = Set.of("*all");


    @Override
    public void startDailyProcess() {
        String searchQuery =
JqlUtility.createSearchByUpdatedDateDateQuery(LocalDateTime.now().minusDays(1),
LocalDateTime.now().plusDays(1));

        IntStream.iterate(0, i -> i + BATCH_SIZE * THREAD_SIZE )
                .map(startFrom -> {
                    List<Issue> resultOfIteration = IntStream.iterate(startFrom, i -> i +
BATCH_SIZE).limit(THREAD_SIZE)
                            .parallel()
                            .mapToObj(batchIndex -> doSearch(batchIndex, searchQuery))
                            .map(SearchResult::getIssues)
                            .filter(tickets -> tickets.iterator().hasNext())
                            .flatMap(issues -> StreamSupport.stream(issues.spliterator(), Boolean.FALSE))
                            .toList();
```

```
                jiraRedisEventProducer.send(resultOfIteration);

                return resultOfIteration.size();
            })
            .anyMatch(countOfFullBatches -> countOfFullBatches != THREAD_SIZE);

        log.info("Processing has been finished");
    }

    private SearchResult doSearch(int start, String searchQuery) {
        return jiraClient.searchJql(searchQuery, BATCH_SIZE, start, FIELDS_TO_SEARCH)
                .claim();
    }
}
```

JqlUtility

```
package com.bramix.perfomance.tracker.utility;

import lombok.experimental.UtilityClass;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@UtilityClass
public class JqlUtility {
    private final static DateTimeFormatter DATE_FORMATTER = DateTimeFormatter.ofPattern("yyyy-MM-dd");


    public String createSearchByUpdatedDateDateQuery(LocalDateTime startDate, LocalDateTime endDate) {
        return String.format("updatedDate >= %s and updatedDate <= %s", DATE_FORMATTER.format(startDate), DATE_FORMATTER.format(endDate));
    }

}
```

Додаток 3. Лістинг мікросервісу github-service

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>github-metrics-reader</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>github-metrics-reader</name>
    <description>github-metrics-reader</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.kohsuke</groupId>
            <artifactId>github-api</artifactId>
```

```
            <version>1.303</version>
        </dependency>
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-ui</artifactId>
            <version>1.6.4</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Pom.xml

```
package com.bramix.metrics.reader;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class GithubMetricsReaderApplication {

    public static void main(String[] args) {
        SpringApplication.run(GithubMetricsReaderApplication.class, args);
    }

}
```

ObjectMapperConfiguration

```
package com.bramix.metrics.reader.config;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.text.SimpleDateFormat;

@Configuration
public class ObjectMapperConfiguration {

    private final static String DATE_FORMAT = "yyyyy-mm-dd hh:mm:ss";

    @Bean
    public ObjectMapper createObjectMapper() {
        var objectMapper = new ObjectMapper();
        objectMapper.setSerializationInclusion(JsonInclude.Include.NON_NULL);
        objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);
        objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false);
        objectMapper.setDateFormat(new SimpleDateFormat(DATE_FORMAT));

        return objectMapper;
    }
}
```

RedisConfig

```
package com.bramix.metrics.reader.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
```

```
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
public class RedisConfig {
   @Bean
   public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory) {
      RedisTemplate<String, Object> redisTemplate = new RedisTemplate();
      redisTemplate.setConnectionFactory(connectionFactory);
      redisTemplate.setKeySerializer(new StringRedisSerializer());
      redisTemplate.setValueSerializer(new StringRedisSerializer());
      redisTemplate.setHashKeySerializer(new StringRedisSerializer());
      redisTemplate.setHashValueSerializer(RedisSerializer.string());
      return redisTemplate;
   }
}
```

CommitInfoDto

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class CommitInfoDto {
   String message;
   Integer commentCount;
   String committerName;
   String committerEmail;
}
```

GithubCommitDto

```
package com.bramix.perfomance.tracker.metricsreader.model;
```

```
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.Value;

import java.util.Date;
import java.util.List;

@Data
@NoArgsConstructor
public class GithubCommitDto {
    AuthorDto author;
    CommitInfoDto committer;
    OwnerDto owner;
    Date commitDate;
    List<CommitFileDto> commitFiles;
}
```

GithubReport

```
package com.bramix.perfomance.tracker.metricsreader.model;

import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.List;

@Data
@NoArgsConstructor
public class GithubReport implements Serializable {
    List<GithubCommitDto> ghCommits;
}
```

```
package com.bramix.perfomance.tracker.metricsreader.model;
```

GithubController

```
package com.bramix.metrics.reader.controller;


import com.bramix.metrics.reader.service.GitHubIntegrationService;
import io.swagger.v3.oas.annotations.Operation;
import lombok.AllArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.Date;

@RestController
@AllArgsConstructor
public class GithubController {
   private final GitHubIntegrationService gitHubIntegrationService;

   @Operation(description = "API для старту інтеграції з Github з можливістю обрати період для виборки даних")
   @PostMapping("/gitHub")
   public void retrieveCommits(@RequestParam("startDate") @DateTimeFormat(pattern="yyyy-MM-dd") Date startDate,
                               @RequestParam("endDate") @DateTimeFormat(pattern="yyyy-MM-dd") Date endDate,
                               @RequestParam boolean shouldBeSaved) {
      if (shouldBeSaved) {
         gitHubIntegrationService.retrieveCommitsByDateAndSendToKibana(startDate, endDate);
      }
      gitHubIntegrationService.retrieveCommitsByDate(startDate, endDate);
   }}
```

GithubCommitMapper

```
package com.bramix.metrics.reader.mapper;

import com.bramix.metrics.reader.model.*;
```

```
import lombok.SneakyThrows;
import org.kohsuke.github.GHCommit;
import org.kohsuke.github.GHUser;
import org.springframework.stereotype.Component;

import java.util.stream.Collectors;

@Component
public class GithubCommitMapper {

    private static final String UNDEFINED = "undefined";

    @SneakyThrows
    public GithubCommitDto map(GHCommit ghCommit) {

        AuthorDto author = getAuthor(ghCommit.getAuthor());

        var commitShortInfo = ghCommit.getCommitShortInfo();

        var committer = new CommitInfoDto(commitShortInfo.getMessage(),
                commitShortInfo.getCommentCount(),
                commitShortInfo.getCommitter().getName(),
                commitShortInfo.getCommitter().getEmail());

        var owner = new OwnerDto(ghCommit.getOwner().getName(),
                ghCommit.getOwner().getFullName(),
                ghCommit.getOwner().getLanguage(),
                ghCommit.getOwner().getVisibility().name()
        );

        var commitFiles = ghCommit.getFiles()
                .stream()
                .map(file -> new CommitFileDto(file.getStatus(),
                        file.getLinesChanged(),
                        file.getLinesAdded(),
                        file.getLinesDeleted(),
                        file.getRawUrl().toString(),
```

```
                file.getPatch())
        ).collect(Collectors.toList());

    return new GithubCommitDto(author, committer, owner, commitShortInfo.getCommitDate(),
commitFiles);
  }

  private AuthorDto getAuthor(GHUser ghUser) {
    if (ghUser == null) {
      return new AuthorDto(UNDEFINED, UNDEFINED);
    }
    return new AuthorDto(ghUser.getLogin(),
            ghUser.getUrl().toString());
  }
}
```

GithubRedisProducer

```
package com.bramix.metrics.reader.redis.producer;

import com.bramix.metrics.reader.mapper.GithubCommitMapper;
import com.bramix.metrics.reader.model.GithubReport;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.kohsuke.github.GHCommit;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.redis.connection.stream.ObjectRecord;
import org.springframework.data.redis.connection.stream.RecordId;
import org.springframework.data.redis.connection.stream.StreamRecords;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.stream.Collectors;
```

```
@Slf4j
@Component
@RequiredArgsConstructor
public class GithubRedisProducer {

    @Value("${github.stream.name}")
    private String streamKey;

    private final ObjectMapper objectMapper;
    private final RedisTemplate<String, Object> redisTemplate;
    private final GithubCommitMapper githubCommitMapper;

    @SneakyThrows
    public void send(List<GHCommit> ghCommits){

        var commits = ghCommits.parallelStream()
                .map(githubCommitMapper::map)
                .collect(Collectors.toList());

        ObjectRecord<String, String> record = StreamRecords.newRecord()
                .in(streamKey)
                .ofObject(objectMapper.writeValueAsString(new GithubReport(commits)))
                .withId(RecordId.autoGenerate());

        redisTemplate.opsForStream()
                .add(record);
    }

}
```

GithubRedisProducer

```
package com.bramix.metrics.reader.service;

import com.bramix.metrics.reader.redis.producer.GithubRedisProducer;
import lombok.AllArgsConstructor;
```

```
import lombok.SneakyThrows;
import org.kohsuke.github.GHCommit;
import org.kohsuke.github.GitHub;
import org.springframework.stereotype.Service;

import java.io.IOException;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@Service
@AllArgsConstructor
public class GitHubIntegrationService {
    private final GitHub gitHubClient;
    private final GithubRedisProducer githubRedisProducer;

    @SneakyThrows
    public List<GHCommit> retrieveCommitsByDate(Date startDate, Date endDate) {
        return gitHubClient.getMyself()
                .getAllRepositories().values()
                .parallelStream()
                .map(repo -> repo.queryCommits()
                        .since(startDate)
                        .until(endDate)
                        .list()
                )
                .flatMap(commmitsList -> {
                    try {
                        return commmitsList.toList().stream();
                    } catch (IOException e) {
                        return Stream.empty();
                    }
                })
                .collect(Collectors.toList());
    }
```

```
    @SneakyThrows
    public List<GHCommit> retrieveCommitsByDateAndSendToKibana(Date startDate, Date endDate) {
        List<GHCommit> ghCommits = retrieveCommitsByDate(startDate, endDate);
        githubRedisProducer.send(ghCommits);
        return ghCommits;
    }
}
```