МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Інститут прикладного системного аналізу

Кафедра Системного проектування

До захисту допущено

Завідувач кафедри

_____ А.І. Петренко
(підпис) (ініціали, прізвище)

“ ” 2017 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) освітньо-кваліфікаційного рівня

“Бакалавр”

з напряму підготовки (спеціальності) 6.050101-Компютерні науки

Кваліфікація – Фахівець з інформаційних технологій

на тему: Розробка інформаційної системи реєстрації на базі мікросервісної архітектури

Студент групи ДА-31 Кутовий Олександр Олександрович
(шифр групи) (прізвище, ім’я, по батькові) (підпис)

Керівник проекту(роботи) доц., к.т.н. Харченко К. В.
(вчені ступінь та звання, прізвище, ініціали) (підпис)

Консультанти:

Економіка доц., к. т. н. Рощина Надія Василівна .
(назва розділу ДР) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Нормоконтроль старший викладач Британ О.А. _____
(назва розділу ДР) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2017

Національний технічний університет України

“Київський політехнічний інститут”

Факультет (інститут) Інститут прикладного системного аналізу

Кафедра Системного проектування

Напрям підготовки 6.050101-Компютерні науки

Спеціальність 7.05010102,8.05010102-Інформаційні технології проектування

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ А.І.Петренко

(підпис) (ініціали, прізвище)

“ ” 2017 р.

ЗАВДАННЯ

на дипломний проект (роботу) освітньо-кваліфікаційного рівня

“Бакалавр”

Кваліфікація – Фахівець з інформаційних технологій

студенту _____Кутовому Олександру Олександровичу_____

(прізвище, ім’я, по батькові)

1. **Тема проекту** (роботи) _____Розробка інформаційної системи

_____реєстрації на базі мікросервісної архітектури_____

_____

затверджена наказом по університету від “___”_____ 2017 р. №_____

2. **Термін здачі** студентом закінченого проекту (роботи) “10” 06 2017 р.

3. **Вихідні дані до проекту** (роботи)

Мікроконтроллер Raspberry PI 3, зчитувач NFC карток RFID RC522, Docker container, тестова NFC картка, MySQL база даних, SQLite база даних.

4. **Перелік питань, які повинні бути розроблені**

1. Проаналізувати існуючі рішення.
2. Проаналізувати доцільність використання мікросервісної архітектури.
3. Реалізувати мікросервіс до контролю роботи інших мікросервісів
4. Реалізувати мікросервіс для експорту даних в Google Sheets
5. Провести функціонально вартісний аналіз проекту
6. Проаналізувати результати роботи, зробити висновки.

5. **Перелік графічного (ілюстративного) матеріалу**

Плакати:

- Архітектура системи - плакат;
- Схема роботи інформаційної системи;
- Схема декомпозиції на мікросервіси;
- Презентація.

6. **Консультанти розділів проекту (роботи)***:*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н. В., доцент		

7. **Дата видачі завдання** “ 01” 04 2017 р.

Керівник дипломного проекту (роботи) _____ Харченко К. В.

(підпис) (ініціали, прізвище)

Завдання прийняв до виконання _____ О. О. Кутовий

(підпис) (ініціали, прізвище)

ЗАТВЕРДЖУЮ
Керівник дипломного проекту (роботи)
_____ К.В. Харченко
(підпис) (ініціали, прізвище)

“ ” _____ 2017 р.

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК
виконання дипломного проекту (роботи)

студентом Кутовим О.О. .
(прізвище, ініціали)

№ з/п	Назва етапів роботи та питань, які повинні бути розроблені відповідно до завдання	Термін виконання	Позначки керівника про виконання завдань
1	Ознайомлення з технічною літературою і підготовка теоретичної частини роботи	17.04.2017 – 21.04.2017	
2	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	22.04.2017	
3	Проектування архітектури, складання базової документації	30.04.2017	
4	Перевірка розроблених прототипів. Перевірка відповідності завданню.	05.05.2017	
5	Підготовка графічного матеріалу, оформлення пояснювальної записки, підготовка до захисту	15.05.2017	
6	Проходження нормоконтролю, отримання рецензії, подача роботи в ДЕК	10.06.2017	
7	Захист дипломної роботи	19.06.2017	

Студент _____
(підпис)

Керівник дипломного проекту (роботи) Харченко К. В.
(підпис) (ініціали, прізвище)

АННОТАЦІЯ

бакалаврської дипломної роботи Кутового Олександра Олександровича

на тему :

«Розробка клієнтської частини інформаційних систем з застосуванням мікросервісної архітектури на мікроконтроллерах»

В даній дипломній роботі розглянуто процесс створення серверної частини інформаційної системи на базі мікросервісної архітектури. Інформаційна система обрана до розробки полягає в способі автоматизації реєстрації відвідувань студентами годин навчання, та автоматичному обліку набраної інформації.

В ході роботи на базі наведених теоретичних відомостей про мікросервісний стиль програмування, обрано варіант декомпозиції, орієнтуючись на існуючі способи його примінення в індустрії. Такі рішення, в результаті, втілені в прототипі системи призначенної для незалежного зберігання та захисту даних зібраних з окремих клієнтських пристроїв.

В висновках до дипломного проекту приведено спостереження що до перспектив використання мікросервісної архітектури, а також можливі переваги такого підходу.

Роботу рекомендовано до використання в якості прикладного рішення організаційних задач в структурах з численною кількістю користувачів, та необхідністю обліку їх присутності в регламентовані проміжки часу.

Загальний обсяг роботи : 81 сторінок, 2 додатки на 29сторінок, 17 рисунків, 4 таблиць, 11 посилання.

АННОТАЦИЯ

бакалаврской дипломной работы Кутового Александра Александровича

на тему:

«Разработка клиентской части информационных систем с применением микросервиснои архитектуры на микроконтроллерах»

В данной дипломной работе рассмотрены процесс создания серверной части информационной системы на базе микросервиснои архитектуры. Информационная система выбраная к разработке заключается в способе автоматизации регистрации посещений студентами часов обучения, и автоматическом учете накопленной информации.

В ходе работы на базе приведенных теоретических сведений о микросервисном стиле программирования, выбран вариант декомпозиции системы, ориентируясь на существующие способы его приминення в индустрии. Такие решения, в результате, воплощены в прототипе системы предназначеном для независимого хранения и защиты данных собранных с отдельных клиентских устройств.

В выводах к дипломному проекту приведены наблюдения относительно перспектив использования микросервиснои архитектуры, а также возможные преимущества такого подхода.

Работа рекомендуется к использованию в качестве прикладного решения организационных задач в структурах с большим количеством пользователей, и необходимостью учета их присутствия в регламентированых промежутках времени.

Общий объем работы: 1413132 страниц, 2 приложения на 29 страниц, 17 рисунков, 4 таблиц, 11 ссылки.

ANNOTATION

to Alexander Alexandrovich Kutobiy’s Bachelor's degree thesis on the topic: "Attendence system development based on microservices architecture"

In this thesis is described the process of creating a server part of the information system based on micro-service architecture. The information system chosen for development consists in a method of automating the registring of student visits to study hours and automatic recording of accumulated information.

In the course of work, based on gathered and demonstrated theoretical information on the microservice style of programming, a variant of the decomposition of the system was chosen, focusing on the existing ways of its application in the industry. Such solutions, as a result, are embodied in a prototype system designed for independent storage and protection of data collected from individual client devices.

In the conclusions to the diploma project, observations are made regarding the prospects of the micro-service architecture, and also the possible advantages of such approach.

Work is recommended to use as an application solution organizational tasks in structures with a large number of users, and the need to take into account their presence in the regulated intervals of time.

The total amount of work: 81 pages, 2 appendecies for 1231 pages, 17 pictures, 4 tables, 11 references.

ЗМІСТ

ВСТУП

Бакалаврська робота присвячена розробці серверної частини інформаційної системи, що повинна забезпечувати спрощений облік відвідування лекцій студентами. Полягає необхідний до розробки функціонал в розбитій на 3 мікросервіси системи, що відповідає за форматування даних отриманих з пристроїв обліку присутності, для поміщення цих даних в хмарне сховище Google Cloud.

Для гнучкості налаштування, та спрощення задач подальших ітерацій розробки інформаційної системи, що можуть включати розширення можливостей наочного експорту даних, або ж збільшення варіантів інтерпретації розкладу, або ж факту присутності студента, система проектується в рамках мікросервісної архітектури проекту. В порядку дослідження необхідні до розгляду аспекти мікросервісної архітектури включають : можливість налаштування проекту прямими маніпуляціями над процесами, на рівні файлової системи, спрощена імплементація та заміна функціональних блоків інформаційної системи.

В результаті розробки очікується реалізація додатку, що зможе забезпечити стабільне спілкування з пристроями обліку студентів, що дозволить розгрузити задачі переносного пристрою, і позбавити розробників необхідності розширення об’єму постійної пам’яті мікроконтроллеру.

ПОСИЛАННЯ НА МАТЕРІАЛИ, ЩО ВИКОРИСТАНІ В РОБОТІ

У роботі наведені 8 рисунків, та лістинги 2х файлів програм.

Тип	Назва	Пункт
Рисунок 1.1	Загальна DFD діаграма функції	1.2.1
Рисунок 1.2	Функціональна діграма другого рівня .	1.2.2
Рисунок 1.3	Порівняння мікросервісної та монолітної архітектури у відношенні до базової складності системи	1.2.3
Рисунок 2.1	Сутність централізованого та децентралізованого керування даними.	2.1.1
Рисунок 2.2	Формат структури, що отримується на вхід серверної чатисни	2.1.2.2
Рисунок 2.3	Загальні сегменти системи обліку відвідувань.	2.2.1
Рисунок 2.4	Ключові функціональні сегменти периферії інформаційної системи	2.2.4
Рисунок 2.5	Загальна Use-Case UML діарграмма для комплексного.	2.3.2
Рисунок 2.6	Детальна Use-Case UML діаграмма	2.2
Рисунок 3.1	Логотип сервісу Gitlab	3.4
Рисунок 3.2	Вигляд інтерфейсу клієнтської программи сервісу Gitlab	3.4
Рисунок 3.3	Результуючий механізм CI на базі Gitlab	3.4
Рисунок 4.1	Морфологічна карта	4.1
Рисунок 4.2	X1, швидкодія мови програмування	4.2
Рисунок 4.3	X2, об’єм пам’яті для збереження даних	4.2
Рисунок 4.4	X3, час обробки даних алгоритмом	4.2
Рисунок 4.5	X4, час обробки даних алгоритмом	4.3
Таблиця 4.1	Позитивно-негативна матриця	4.4
Таблиця 4.2	Основні параметри ПП	4.1

Таблиця 4.3	Результати ранжування параметрів	4.3
Таблиця 4.4	Попарне порівняння параметрів	4.3
Додаток А	Основний скрипт роботи пристрою з базою MySQL.	3.1
Додаток Б	Основний скрипт роботи пристрою з базою Google Sheets.	3.1

1 СИСТЕМНИЙ АНАЛІЗ ЗАВДАНЬ БАКАЛАВРСЬКИХ ДОСЛІДЖЕНЬ

Мета дипломної роботи

Розробка серверної сторони інформаційної системи автоматичного обліку відвідувань на базі мікросервісної архітектури. Дослідження умов повної контейнеризації мікросервісів інформаційної системи розробленої на базі сервіс-орієнтованої архітектури.

Огляд літератури

Мікросервісний підхід до проектування додатків є порівнянно новою ідеологія, бодай і бере початки свого застосування практично з 80-х років. На даний момент найбільш ефективними джерелами для розгляду примінення новітніх для широких масс стилів програмування стають онлайн статті та інтернет видавництва, що ведуться на некомерційній основі, адже є порівнянно ризиковими для надійно фінансованого видавництва. Такі теми потребують швидкого тестування, та доцільного обговорення спеціалістами, що мали шанс примінити концепцію в якомога більш різноманітних варіантах її використання. Тільки таким чином можна зібрати доцільну інформацію відносно, ще не затвердившої себе на ринку парадигми архітектурної побудови інформаційних систем.

Окремо від критики та огляду примінення мікросервісного стилю в роботі використані матеріали технічної документації таких продуктів та сервісів як Docker.

Задачі дипломної роботи

- Створити стійкий та стабільний сервіс прийому даних через інтернет, з їх подальшою буферизацією в базі даних.
- Створити демонстративні аналоги такого сервісу, що дозволяють продемонструвати гнучкість мікросервісної архітектури (при збереженні стабільності роботи всієї інформаційної системи).
- Дослідити можливу декомпозицію інформаційної системи на мікросервіси, та визначити можливі та найзручніші варіанти їх комунікації.
- Реалізувати зручні та доступні способи адміністрування та налаштування параметрів експорту даних.
- Реалізувати зворотній зв’язок з даними збереженими в хмарному сховищі для спрощеної реалізації адміністрування системи викладачем.
- Проаналізувати реалізацію спроектованої архітектури в умовах повної контейнеризації, за допомогою сервісу Docker.

1.1 Цілі дипломної роботи, ключові фактори успіху результатів роботи

Основною метою дипломної роботи є проектування та розробка серверної складової інформаційної системи, присвяченої для обліку відвідувань студентами лекцій.

Серверна частина системи призвана стати ключовим медіатором в загальній структурі між пристроєм зчитування та фіксації присутності та хмарним хранилищем, з мета-даними для адміністрування та експорту даних. На базі такої системи необхідний до дослідження мікросервісний стиль проектування, в інтересах створення можливості спрощеного та більш гнучкого розширення системи. Як, наприклад, можливість приєднати/змінити умови експорту даних з усіх пристроїв фіксації присутності (для інтеграції в альтернативні системи звітності/хмарні хранилища)

Ключові фактори успіху дипломного проекту :

- спроектовані та реалізовані процеси взаємодії з приладами зчитування магнітних карток.
- програма, внутрішньої для серверу, буферизації та декодування даних.
- гнучка схема експорту даних в зовнішнє хранилище (для прототипу реалізації обрано Google Sheets) відповідно до метаданих, що отримуються відповідно з такого ж хранилища.
- Імплементовані механізми гнучкості та налаштування мікросервісів без глибоко втручання в структуру проекту (налаштування та внесення змін на рівні файлової системи).

ФУНКЦІОНАЛЬНА МОДЕЛЬ

1.2.1 DFD (IDEF0) діаграма

Даний сегмент всієї інформації є внутрішнім механізмом, тож перш за все є доступним лише для адміністратора.

Для стабільної роботи сервісу додаток потребує вай-фай адаптеру та інтернет зв’язку для з’єднання з Google Cloud.

Реалізація додатку планується з використанням JVM

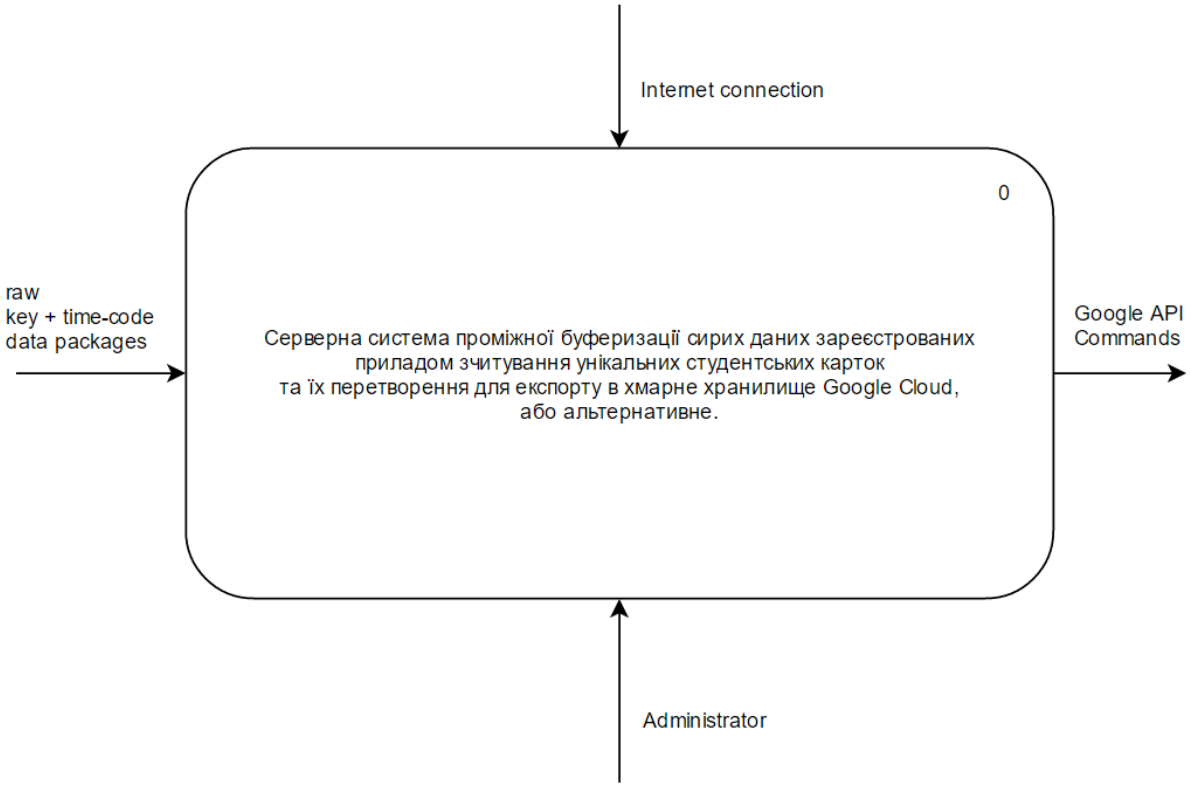

Рисунок 1.1 – Загальна DFD діаграма

1.2.2 DataFlow Діаргамма другого рівня.

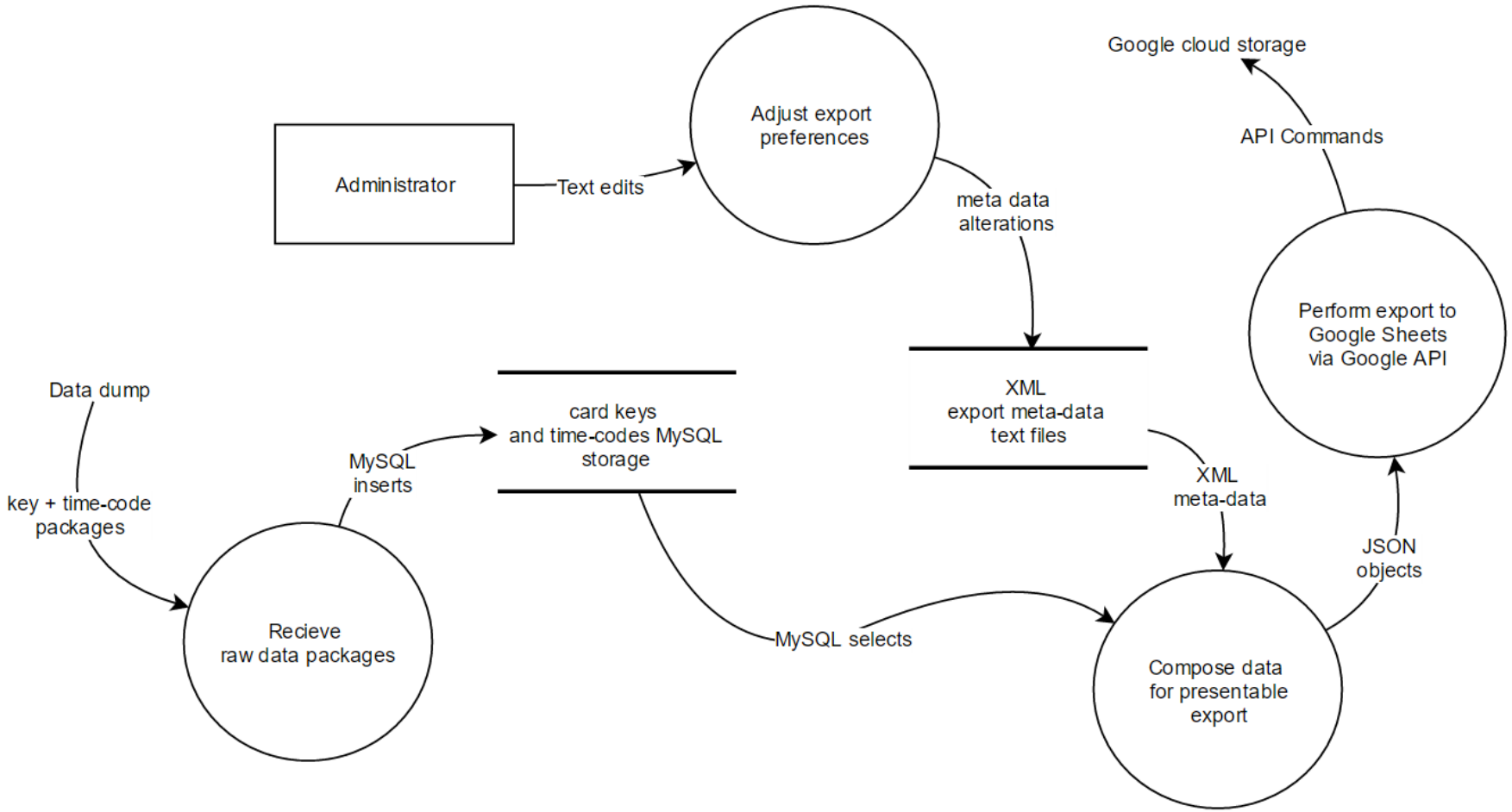

Рисунок 1.2 – Функціональна діграма другого рівня

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЗА МІКРОСЕРВІСНИМ СТИЛЕМ ПРОГРАМУВАННЯ

2.1 Огляд мікросервісного стилю проектування та його особливастей

2.1.1 Декомпозиція серверної частини на мікросервіси.

На даний момент в індустрії наявні сильні тенденції до подбудови систем шляхом з'єднання разом різних компонент, багато де - так само, як це відбувається в реальному світі. За останні пару десятків років ми бачили велике зростання набору бібліотек, використовуваних в більшості мов програмування.

Говорячи про компоненти, ми стикаємося з труднощами визначення того, що саме є компонентом. Запропоноване визначення такого: компонент - це одиниця програмного забезпечення, яка може бути незалежно замінена або оновлена.

Архітектура мікросервісів використовує бібліотеки, але їх основний спосіб розбиття додатку - шляхом ділення його на сервіси. Ми визначаємо бібліотеки як компоненти, які підключаються до програми і викликаються нею в тому ж процесі, в той час як сервіси - це компоненти, що виконуються в окремому процесі і коммунікують між собою через веб-запити або remote procedure call (RPS).

Головна причина використання сервісів замість бібліотек - це незалежне розгортання. При розробці додатку, що складається з декількох бібліотек, які працюють в одному процесі, будь-яка зміна в цих бібліотеках призводить до перерозгортки всієї програми. Але якщо додаток розбитий на кілька сервісів, то зміни, що зачіпають будь-які з них, будуть вимагати перерозгортання тільки зміненого сервісу. Звичайно, якісь зміни будуть зачіпати інтерфейси, що, в свою чергу, потребують певної координації між

різними сервісами, але мета хорошої архітектури мікросервісів - мінімізувати необхідність в такій координації шляхом установки правильних кордонів між мікросервісамі, а також механізму еволюції контрактів сервісів.

Інший наслідок використання сервісів як компонент - більш явний інтерфейс між ними. Більшість мов програмування не мають хорошого механізму для оголошення Published Interface. Часто тільки документація і дисципліна запобігають порушенням інкапсуляції компонентів. Сервіси дозволяють уникнути цього через використання явного механізму віддалених викликів.

Тим не менше, використання сервісів подібним чином має свої недоліки. Дистанційні виклики працюють повільніше, ніж виклики в рамках процесу, і тому API повинен бути менш деталізованим (coarser-grained), що часто призводить до незручності у використанні. Якщо вам потрібно змінити набір відповідальностей між компонентами, зробити це складніше через те, що вам потрібно перетинати кордони процесів.

У першому наближенні ми можемо спостерігати, що сервіси співвідносяться з процесами як один до одного. Насправді сервіс може містити безліч процесів, які завжди будуть розроблятися і розвиватися разом. Наприклад, процес додатку і процес бази даних, яку використовує тільки ця програма.

2.1.2 Сервісні границі, підкріплені границями команд

Існує не дуже багато прикладів ринковий продуктів що працюють за такою прагадигмою, проти актуальні приклади мають багато чого спільного. Крос-фунціональні команди відповідають за побудову і функціонування кожного продукту і кожен продукт розбитий на кілька окремих сервісів, які спілкуються між собою через шину повідомлень.

Великі монолітні додатки теж можуть бути розбиті на модулі навколо бізнес потреб, хоча, зазвичай, це не відбувається. Безумовно, побудова великих монолітних додатків саме таким чином є безумовно доречною. Основна проблема тут в тому, що такі додатки мають тенденцію до організації навколо занадто великої кількості контекстів. Якщо моноліт охоплює безліч контекстів, окремим членам команд стає занадто складно працювати з ними через їх великого розміру. Крім того, дотримання модульних кордонів в монолітному додатку потребує суттєвого дисципліни. Явно окреслені межі компонент мікросервісів спрощує підтримку цих кордонів.

2.1.3 Визначення чітких масштабів мікросервісів.

Хоча термін «Мікросервіс» став популярним назвою для цього архітектурного стилю, саме ім'я призводить до надмірного фокусу на розмірі сервісів і суперечкам про те, що означає приставка «мікро». В більшості інтерв'ю з тими, хто займався розбиттям ПО на мікросервіси, були зазначені різні розміри. Найбільший розмір був у компаній, які йшли за правилом «Команда двох піц» (команда, яку можна нагодувати двома піцами), таких термін є фразеологічним і означає команду з не більше 12 осіб. В інших компаніях, зі статистики випливає, що мають місце команди, в яких шестеро осіб підтримують шість сервісів.

Це призводить до питання про те, чи є суттєва різниця в тому, скільки людина має працювати на одному сервісі. На даний момент об'єктивним буде погодитись, що обидва ці підходи до побудови команд (1 сервіс на 12 осіб і 1 сервіс на 1 особу) підходять під опис мікросервісної архітектури, але можливо така думка суттєво еволюціонує в майбутньому .

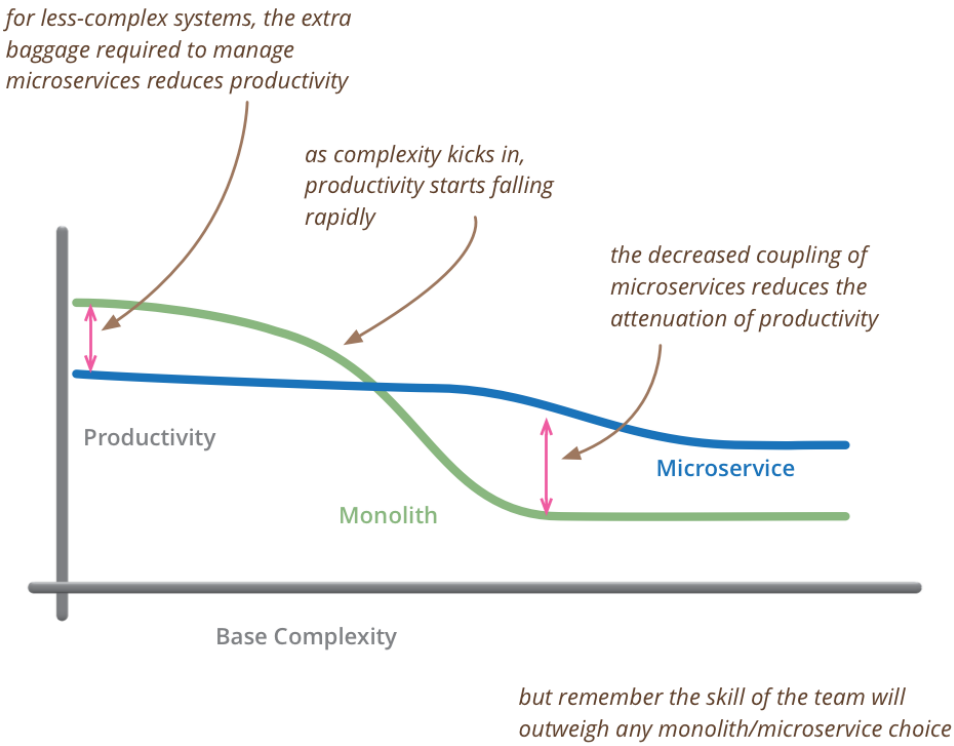

Рисунок 2.1 - Порівняння мікросервісної та монолітної архітектури у відношенні до базової складності системи.

Таким чином, можна зробити висновок, що умови розробки в стилі мікросервісного програмування не ставлять рамки на важкість та масштаби компонент. На даному етапі розвитку ідеології це виключно визначається підходом до безпосередньої декомпозиції задачі, та ресурсами доступними для розробки.

2.1.4 Децентралізоване керування даними

Децентралізоване управління даними постає в різному вигляді. У найбільш абстрактному сенсі це означає, що концептуальна модель середовища у різних систем буде відрізнятися. Це звичайна проблема, що виникає при інтеграції різних частин великих enterprise-додатків: точка зору на поняття «Клієнт» у клієнт-менеджменту буде відрізнятися від такої у команди техпідтримки. Деякі атрибути «Клієнта» можуть бути присутніми в контексті клієнт-менеджменту і бути відсутнім в контексті

техпідтримки. Більш того, атрибути з однаковою назвою можуть мати різне значення.

Ця проблема зустрічається не тільки у різних додатків, але також і в рамках єдиного додатка, особливо в тих випадках коли цей додаток розділений на окремі компоненти. Цю проблему добре вирішує поняття Bounded Context з Domain-Driven Design (DDD). DDD пропонує ділити складну предметну область на кілька контекстів і співставити відносини між ними. Цей процес корисний як для монолітної, так і для мікросервісной архітектур, але між сервісами і контекстами існує природний зв'язок, який допомагає прояснювати і підтримувати кордони контекстів.

Крім децентралізації прийняття рішень про моделювання предметної області, мікросервіси також сприяють децентралізації способів зберігання даних. У той час як монолітні додатки схильні до використання єдиної БД для зберігання даних, компанії часто воліють використовувати єдину БД для цілого набору додатків. Такі рішення, як правило, викликані моделлю доступа до баз даних. Мікросервіси вважають за краще давати можливість кожному сервісу керувати власною базою даних: як створювати окремі інстанси загальної для компанії СУБД, так і використовувати нестандартні види баз даних. Цей підхід називається Polyglot Persistence. Також можливим є застосовування Polyglot Persistence в монолітних додатках, але в мікросервісах такий підхід зустрічається частіше.

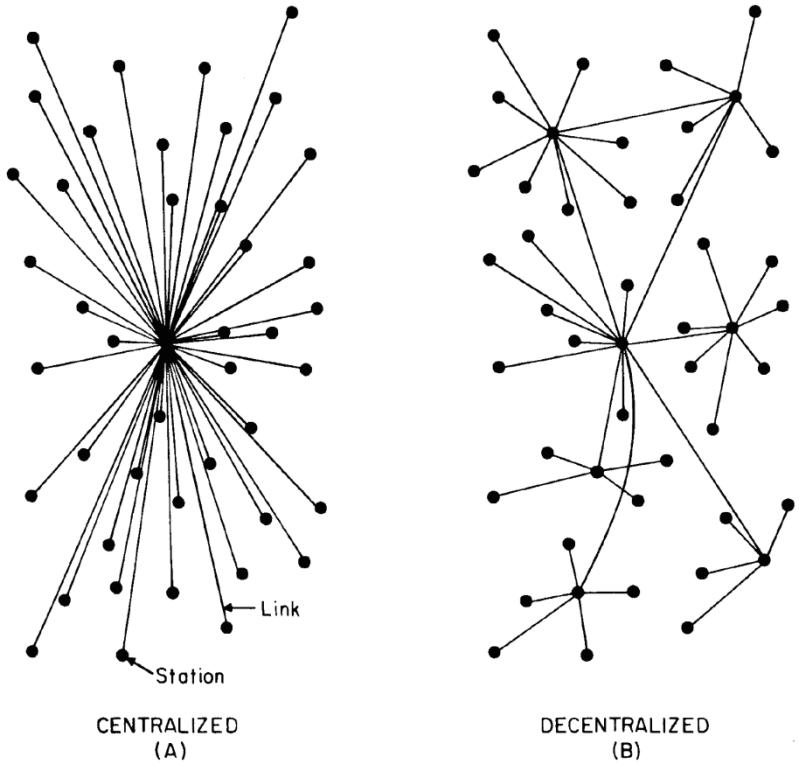

Рисунок 2.2 - Сутність децентралізованого підходу до організації даних.

В рамках системи обліку відвідувань даний аспект зручності мікросервісів постає в дублюванні даних, повторному збереженні даних, та безумовно не обхідних контаків сегментів системи при керуванні збереженною інформації. Це призводить до більшої стійкості системи, до технічних проблем, та до складнощів пов’язаних з можливо ненадійними каналами зв’язку.

Наслідком використання сервісів як компонентів є необхідність проектування додатків так, щоб вони могли працювати при відмові окремих сервісів. Будь-яке звернення до сервісу може не спрацювати з причини його недоступності. Користувач повинен реагувати на це настільки терпимо, наскільки можливо. Це є недоліком мікросервісів в порівнянні з монолітом, тому що це вносить додаткову складність в додаток. Як наслідок, команди мікросервісів постійно думають на тим, як недоступність сервісів повинна впливати на user experience. Simian Army від Netflix штучно викликає (симулює) відмови сервісів і навіть датацентров протягом робочого дня для тестування відмовостійкості додатки і служб моніторингу.

Подібний вид автоматичного тестування в продакшн дозволяє симулювати стрес, який лягає на адміністраторів і часто призводить до понаднормової роботи. Неправильно буде зазначити, що для монолітних додатків не можуть бути розроблені витончені системи моніторингу, тільки те, що таке зустрічається рідше.

Так як сервіси можуть відмовити в будь-який час, дуже важливо мати можливість швидко виявити неполадки і, якщо можливо, автоматично відновити працездатність сервісу. Мікросервісна архітектура робить великий акцент на моніторингу додатку в режимі реального часу, перевірці як технічних елементів (наприклад, як багато запитів в секунду отримує база даних), так і бізнес-метрик (наприклад, як багато запитів в хвилину отримує додаток). Семантичний моніторинг може надати систему раннього попередження проблемних ситуацій, дозволяючи команді розробці підключитися до дослідження проблеми на самих ранніх стадіях.

Це особливо важливо у випадку з мікросервісною архітектурою, тому що розбиття на окремі процеси і комунікація через події призводить до несподіваного поведінки. Моніторинг вкрай важливий для виявлення небажаних випадків такої поведінки і швидкого їх усунення.

Моноліти можуть бути побудовані так само прозоро, як і мікросервіси. Насправді, так вони і повинні будуватися. Різниця в тому, що знати, коли сервіси, що працюють в різних процесах, перестали коректно взаємодіяти між собою, набагато більш критично. У випадку з бібліотеками, розташованими в одному процесі, такий вид прозорості швидше за все буде не настільки корисний.

Команди мікросервісів, як правило, створюють витончені системи моніторингу та логування для кожного індивідуального сервісу. Прикладом може служити консоль, що показує статус (онлайн / офлайн) сервісу і різні

технічні та бізнес-метрики: поточна пропускна здатність, час обробки запиту і т.п.

2.2 Визначення задачі

2.2.1 Опис вхідних даних

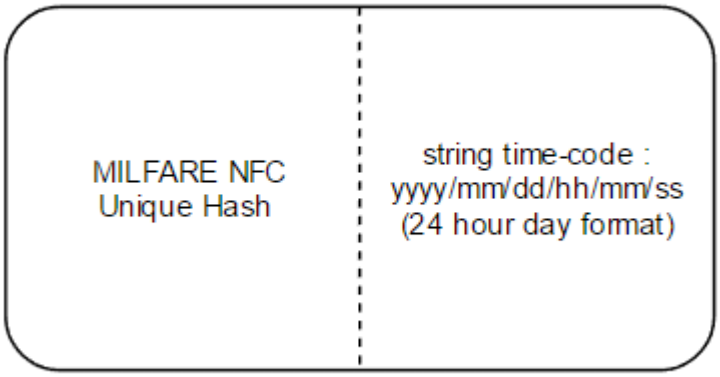

Рисунок 2.3 - Формат структури, що отримується на вхід серверної чатисни.

Окремо від адміністративних інтерфейсів системи, основна ідея має під собою одну основну структуру вхідних даних. На пристрій зчитування NFC поля має подаватись намагнічена картка типу MILFARE 10-13Mhz, що за попередніми налаштуваннями системи повинна відповідати певному користувачеві системи (студенту). Пристрій повинен мати чітку прив'язку до світового часу, для того щоб кодувати кожен зчитанний магнітний ключ таймкодом для легітимного обрахунку відвідувань.

Тож основний цикл роботи системи визначається саме цима двома значеннями : магнітний ключ (унікальний хеш одного студента) та тайм код.

Для пере налаштування розкладів та

Загальна інформаційна система наслідує наступну структуру :

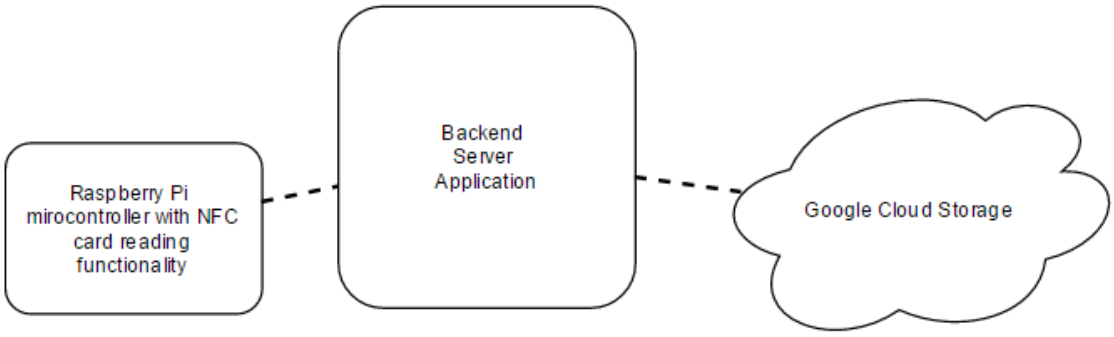

Рисунок 2.4 - Загальні сегменти системи обліку відвідувань.

Дана бакалаврська робота буде присвячена розробці серверної частини. Тож більш детальна декомпозиція стриктури інформаційної системи, примінимо до задач даної бакалаврської роботи виглядатиме наступним чином :

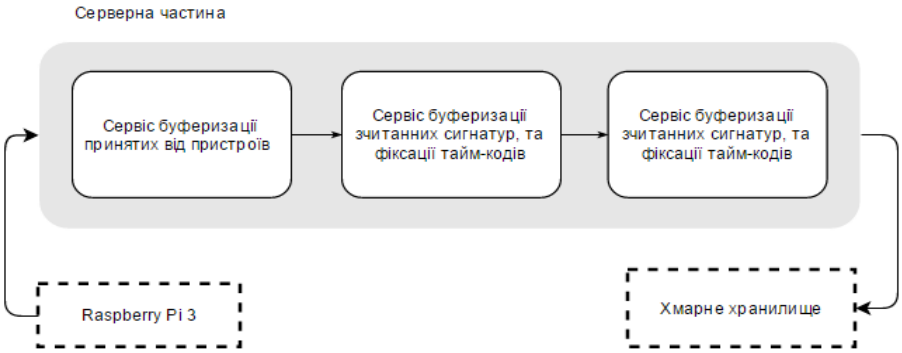

Рисунок 2.5 - Ключові функціональні сегменти периферії інформаційної системи.

2.2.2 Опис вихідних даних

За результатами розробки інформаційна система повинна пропонувати наступний функціонал. Функціонал віподвідно розрахований

на три групи користувачів : адміністратори, викладачі, студенти. Всі три групи мають актуальність в рамках комплексного дипломного проекту, проте в умовах розробки серверної частини, ігнорується вплив студентів на систему, окрім як створення потоку вхідних даних через клієнтський пристрій. Основний функціонал серверної частини призначений для адміністрування та розширення системи, а отже безпосередньо для адміністраторів. За окремими виключеннями – для викладачів.

Риснок 2.6 - Загальна Use-Case UML діарграмма для комплексного дипломного проекту

Оскільки до розробки взято серверну частину, до функціоналу виконаного в дипломній роботі , то діаграма того ж типу, що запропонована вище, для розробленого функціоналу матиме наступний вигляд :

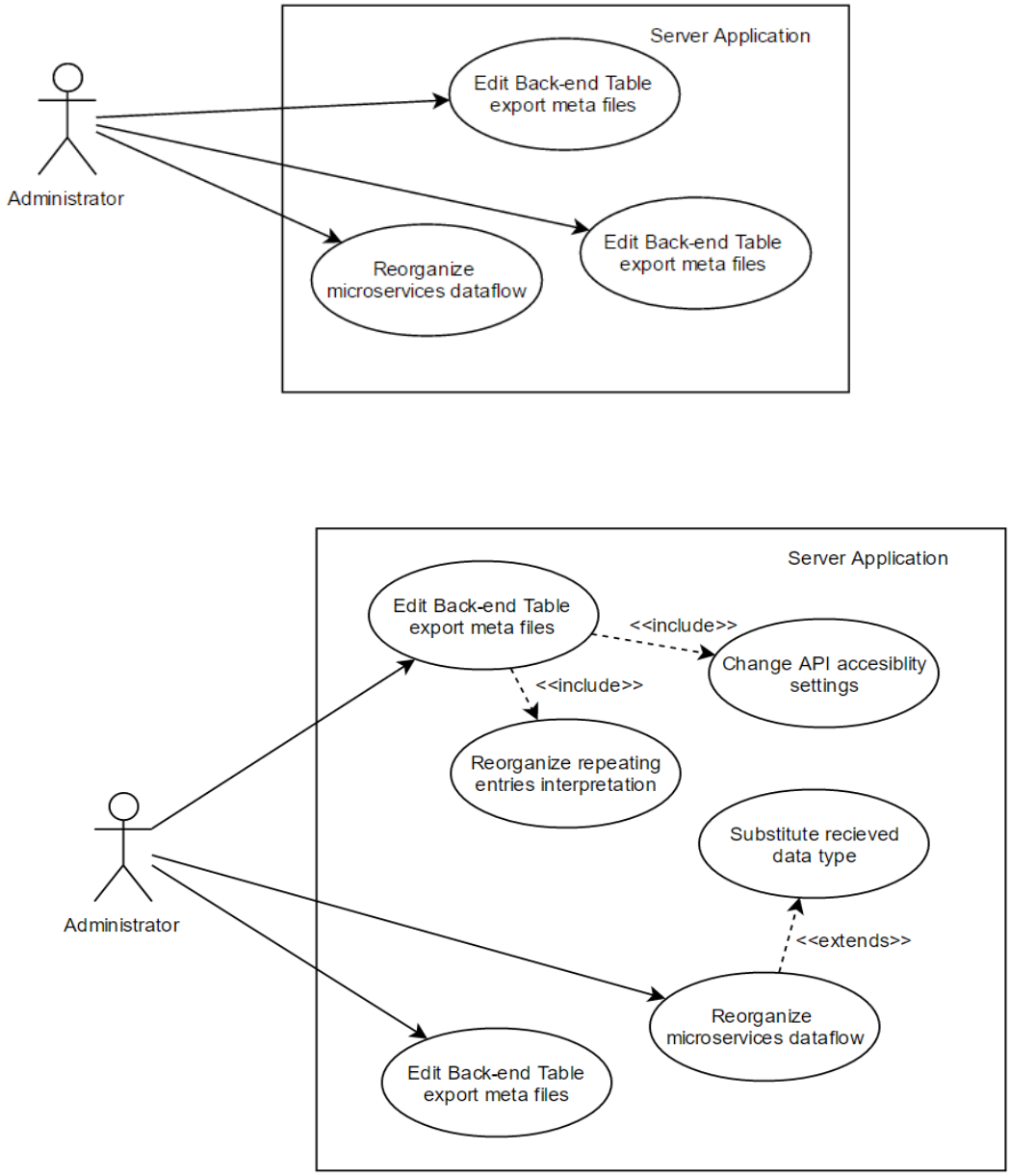

Рисунок 2.7 - Детальна Use-Case UML діаграмма.

3 РОБОТА НАД ІНФОРМАЦІЙНОЮ СИСТЕМОЮ

3.1 Опис задачі

Для реалізації клієнтської частини є ряд ключових, базисних задач необхідних до вирішення. На ряду з якими безпосередньо збірка пристрою та налаштування ОС для роботи з розширеннями мікроконьтроллеру. В порядку переддипломної практики було виконано саме фундаментальні критичні проблеми пов’язані з розробкою системи.

3.2 Реалізація основних механізмів роботи з базою даних.

Задача 'package' повинна виконуватися тільки при успішному проходженні тестів. Порядок виконання завдань визначено шляхом введення стадій (stages).

Також не варто забувати про те, що компіляція (якої в нашому випадку є конкатенація файлів) займає час, тому не варто проводити її двічі. Тому введемо окрему стадію для компіляції.

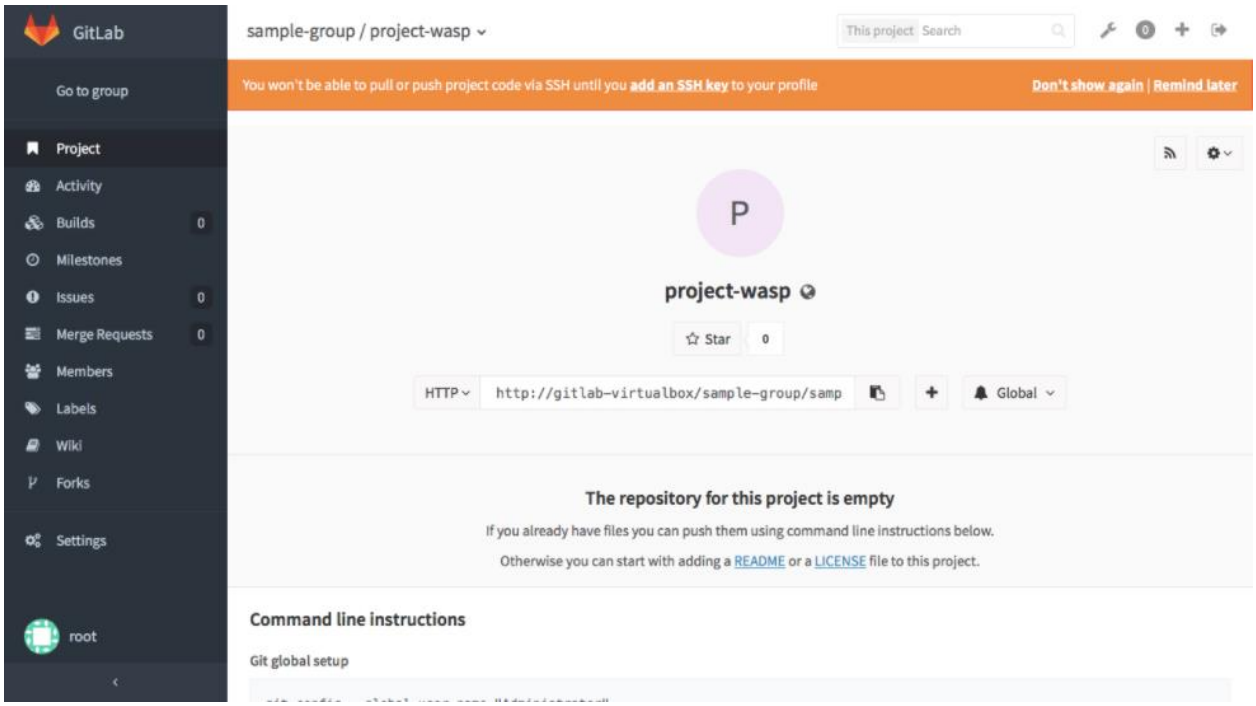

Рисунок 3.1 - Вигляд інтерфейсу клієнтської программи сервісу Gitlab.

Той факт, що Continuous Integration потрібна, вже ніхто не заперечує. Начебто всім зрозуміло, що збирати додаток, проганяючи тести на регулярній основі дуже корисно. Отримати швидкий фідбек, знайти проблему, прогнати на чистій машині - це все надзвичайно доречно. В наш час це стаэ зрозуміло і керіництву проектів, і девелоперам.

Менеджер, як людина, яка не має лізти в технічні деталі, побачивши минулого Continuous Integration збірки, може однозначно сказати, хороший він чи поганий. Зелений - хороший, червоний - поганий. Дуже простий індикатор, та й не тільки для менеджерів, але і для всієї команди в цілому. Девелопери на цю ситуацію дивляться трохи інакше.

Виходить, що рецептом для застосування Continuous Integration є автоматизовані тести, використання артефактів збірки і команда, яка буде працювати як єдиний механізм для досягнення найвищих цілей. Ці три інгредієнти разом, ми зробимо великий крок до фінальної мети - безперервного деплойменту на виробництво.

Фрагмент файлу «gitlab.yml», що ілюструє опис тестів для CI.

```
test:
          script:
          # this configures Django application to use attached
        postgres database that is run on `postgres` host
          - export
        DATABASE_URL=postgres://postgres:@postgres:5432/python-test-
        app
          - apt-get update -qy
          - apt-get install -y python-dev python-pip
          - pip install -r requirements.txt
          - python manage.py test
        staging:
          type: deploy
          script:
          - apt-get update -qy
          - apt-get install -y ruby-dev
          - gem install dpl
          - dpl --provider=heroku --app=gitlab-ci-python-test-staging
        --api-key=$HEROKU_STAGING_API_KEY
          only:
          - master
```

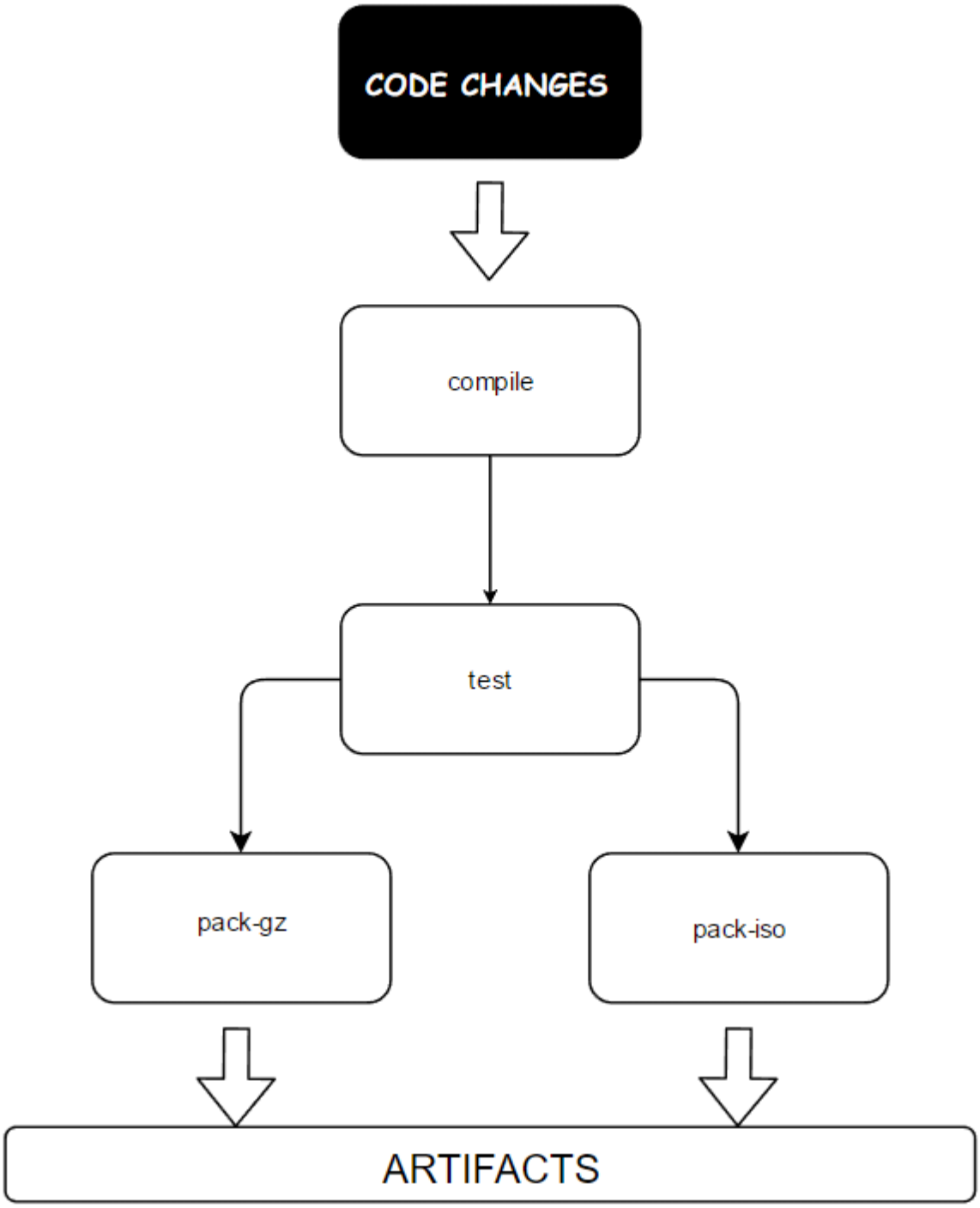

Рисунок 3.2 - Результуючий механізм CI на базі Gitlab.

Так як сервіси можуть відмовити в будь-який час, дуже важливо мати можливість швидко виявити неполадки і, якщо можливо, автоматично відновити працездатність сервісу. Мікросервісна архітектура робить великий акцент на моніторингу додатку в режимі реального часу, перевірці як технічних елементів (наприклад, як багато запитів в секунду отримує база

даних), так і бізнес-метрик (наприклад, як багато запитів в хвилину отримує додаток). Семантичний моніторинг може надати систему раннього попередження проблемних ситуацій, дозволяючи команді розробці підключитися до дослідження проблеми на самих ранніх стадіях.

Фрагмент файлу «Main.java», що відповідає за ініціалізацію системи:

```
System.setProperty("log4j.shutdownCallbackRegistry",
"com.djdch.log4j.StaticShutdownCallbackRegistry");
        Configurator.initialize("Log4j2Conf",
"log4j2.xml");
        l = LogManager.getLogger(Main.class);
        System.setProperty("jsse.enableSNIExtension",
"false");
        Map<Thread, MailSenderWorker> workers = new
HashMap<>();
//        Security.addProvider(new
BouncyCastleProvider());
        PrintStream oldErr = System.err;
        PrintStream oldOut = System.out;
        System.setErr(new
PrintStream(IoBuilder.forLogger(l).filter(oldErr).build
OutputStream(), true));
        System.setOut(new
PrintStream(IoBuilder.forLogger(l).filter(oldOut).build
OutputStream(), true));

TimeZone.setDefault(TimeZone.getTimeZone("Europe/Kiev")
);
        Locale.setDefault(Locale.US);
        ConfigMap config = new ConfigMap();
        if (new File("config.properties").exists()) {
```

Як результат примінення ми маємо максимальний low-coupling всередині классової організаціх мікросервісу. Наступні класси мають доступ до даних. В них імплементовані механізми роботи з input-ом всього компоненту. Вони агрегуються классами, що реалізують бізнес логіку в процессі, які в свою чергу агрегуються классами, що відповідають за експорт/вивід даних.
Таким чином ми отримуємо 3 рівні роботи з даними, кожен з яких залежиний ТІЛЬКИ від попереднього. І це є зручним універсальним шаблоном для розробки єдиного мікросервісу.
Така схема в собі ідеально компонує один функціональний блок мікросервісної архітектури.

Фрагмент файлу «», що покриває собою автоматизацію deployment-у проекту

```
production:
  type: deploy
  script:
  - apt-get update -qy
  - apt-get install -y ruby-dev
  - gem install dpl
  - dpl --provider=heroku --app=gitlab-ci-python-test-prod --
api-key=$HEROKU_PRODUCTION_API_KEY
  only:
  - tags
```

Саме як ідеологічний розвиток ідеї agile-у ми можемо взяти мікросервісний стиль програмування, адже така архітектура суттєво більш схильна до механізмів Continuous Integration та Continuous Deployment. Таким чином в процессі розробки, дійсно є можливість мати на кожному етапі робочий продукт. А за рахунок повної контейнеризації, система стає набагато більш платформо-незалежною і простою для портування. В контексті розробки учбових проектів її можливості практично довільної декомпозиції дозволяються долучати різні команди розробки до роботи

над неймовірно масштабними задачами видаючи їм мінімальні незалежні задачі для рішення.

Ключовою особливістю мікросовервісної архітектури в контексті стеку технологій стає те, що архітектура його ніяк не обмежує. Поєднання мікросервісів покладається цілком на системні ресурси, а через їх широку різноманітність, переваги від одних іншим будуть надаватись виключно обумовлюючись специфікою задачі.

ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для аналізу нелінійних нестаціонарних процесів. Інтерфейс користувача був розроблений за допомогою мов програмування Java, Python та C у середовищі розробки PyCharm, Intellij Idea та CLion. Інтерфейс користувача передбачений лише веб інтерфейсом і використовує готову базу Google Sheets.

Програмний продукт призначено для використання на персональних комерційних пристроях Raspberry Pi під управлінням операційної системи Raspbian.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На

цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки системи аналізу нелінійних нестаціонарних процесів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу гетероскедастичних процесів в економіці та фінансах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на забезпечених портативних пристроях з попередньо налаштованим середовищем;

– забезпечувати високу швидкість обробки чітко передбачених об’ємів даних у реальному часі, з метою уникнення черг для зчитування карток;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка системи, яка реагуючи на чітки визначений формат взаємодії з клієнтом категоризує таку взаємодію відносно розкладу та регламенту учбового процесу. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – розпізнавання вхідних даних;

F_3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Python;

б) мова програмування C++;

Функція F_2:

а) занесення даних за допомогою візуальних способів розпізнання;

б) зчитування NFC сигнатури.

Функція F_3:

а) інтерфейс користувача, що керується скриптами Google Sheets;

б) Написання окремого веб-інтерфейсу, для маніпуляцій з системою.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

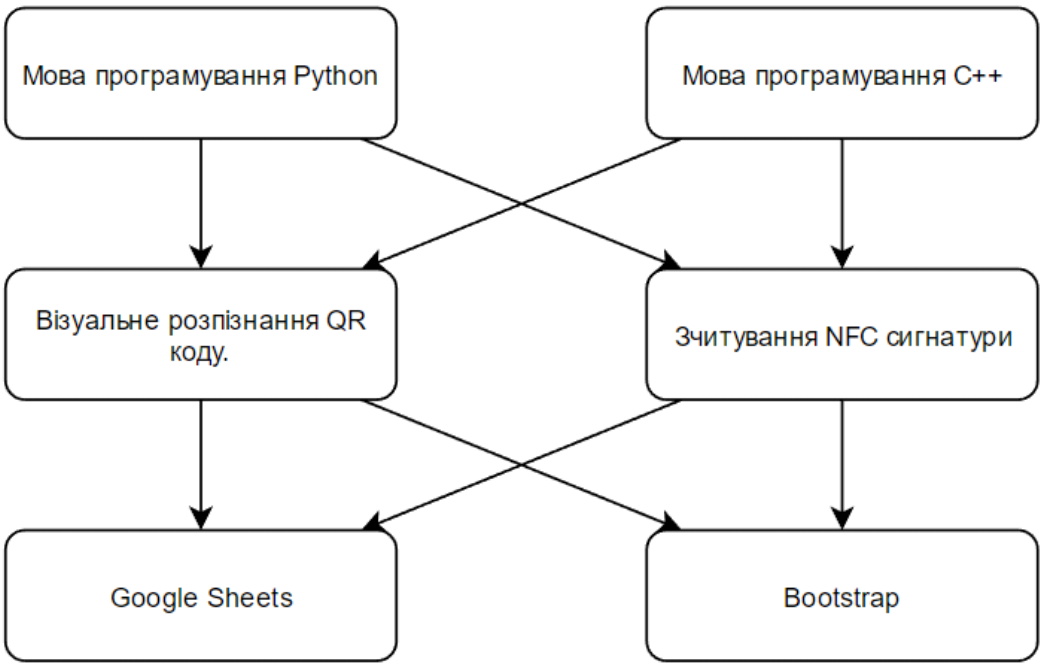

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Доступний і гнучкий. Легше інтегрується для взаємодії з системними ресурсами.	Повільніший, не багато поточний.
	Б	Код швидко виконується, кросплатформений	Складний в підтримці та рефакторингу. Більш вибагливий при запуску проекта.
F2	А	Простий у використанні	Менша точність підрахунків
	Б	Найоптимальніший для використання тільки у власних програмних продуктах	Затрачений час
F3	*А*	Легкий у створенні	Відсутність кросплатформеності
	Б	Стабільний у використанні	Необхідна додаткова Інсталяція

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки розрахунки проводяться з великими об’ємами вхідних даних, то час виконання програмного коду є дуже необхідним, тому варіант б) має бути відкинутий.

Функція *F2:*

Оскільки планується розглядати широкий спектр різноманітних варіантів даних, та перший є більш зручним в даному випадку, то варіант б) має бути відкинутий

Функція *F3:*

Інтерфейс користувача не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1а – F2а – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження даних;

– *X3* – час обробки даних;

– *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X2	Мб	350	320	300
Час обробки даних алгоритмом	X3	мс	5000	600	80
Потенційний об’єм програмного коду	X4	кількість строк коду	2000	1500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

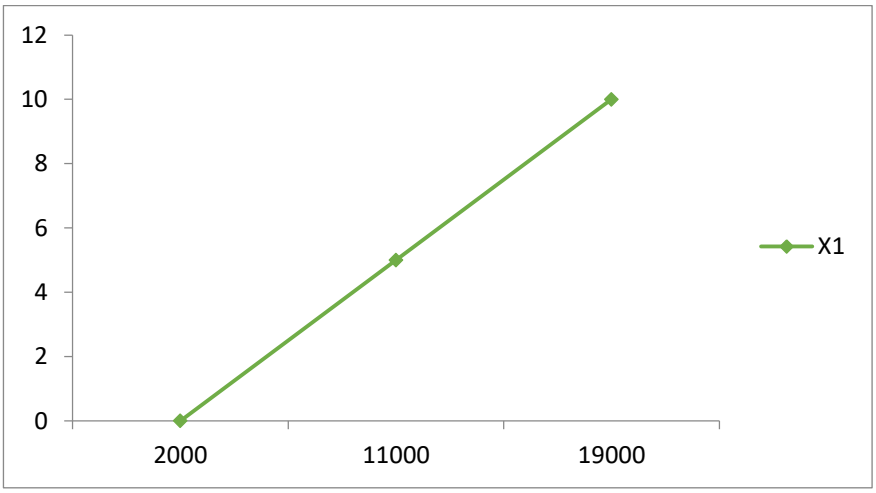

Рисунок 4.2 – X1, швидкодія мови програмування

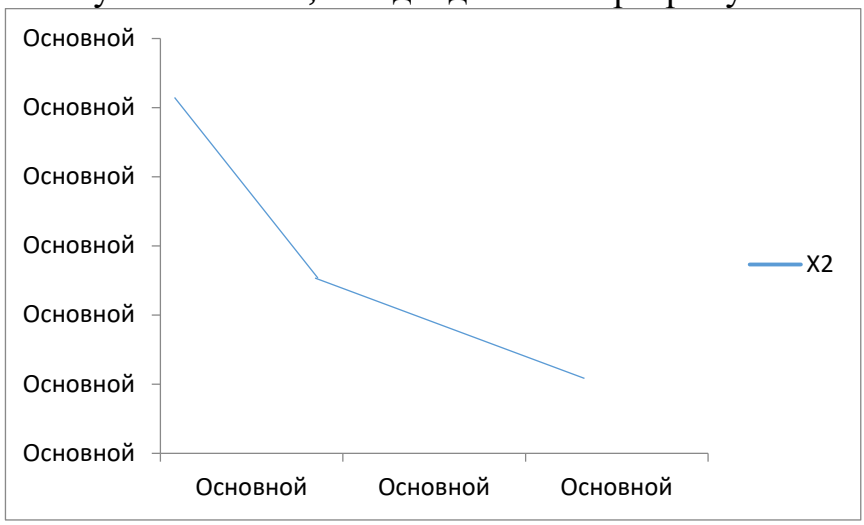

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

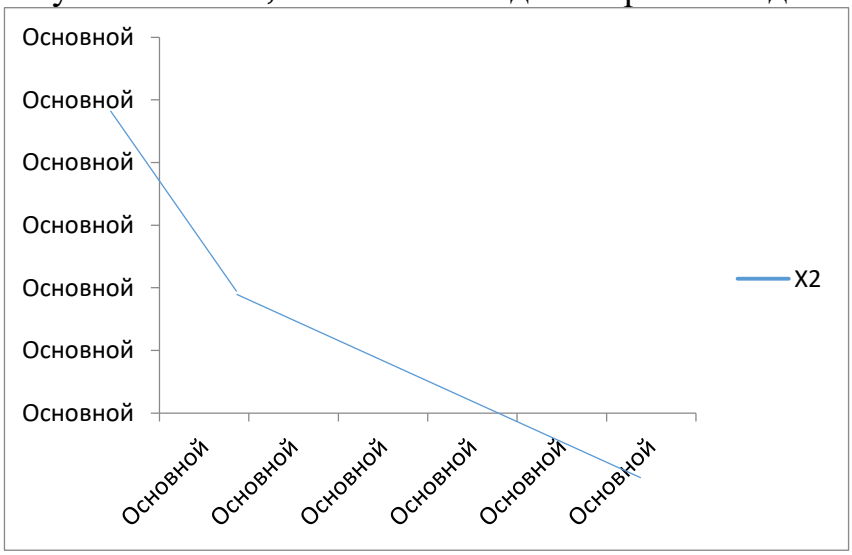

Рисунок 4.4 – X3, час обробки даних алгоритмом

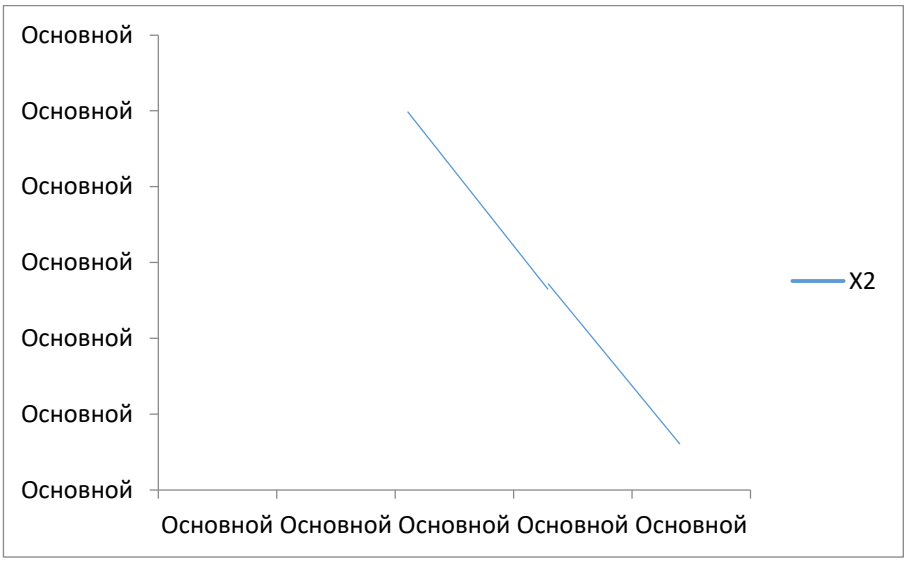

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	4	4	27	0,75	0,56
X2	Об'єм пам'яті для збереження даних	Мб	4	4	4	3	4	3	3	25	-1,25	1,56
X3	Час обробки даних алгоритмом	Мс	2	2	1	2	1	2	2	12	-14,25	203,06
X4	Потенційний об'єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14,75	217,56
	Разом		15	15	15	15	15	15	15	105	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 26{,}25.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 420{,}75.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 420{,}75}{7^2(5^3 - 5)} = 1{,}03 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю4.4.

Таблиця4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij}визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$,де $b_i = \sum_{i=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}$,де $b_i' = \sum_{i=1}^{N} a_{ij} b_j$.

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 800 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	11000	3,6	0,215	0,774
F2(X2)	А	16	3,4	0,283	0,962
F3(X3,X4)	А	800	2,4	0,348	0,835
	Б	80	1	0,154	0,154

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 0,774 + 0,962 + 0,835 = 2,57

K_{K2} = 0,774 + 0,962 + 0,154 = 1,89

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (5.1)$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування;$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ =70 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.5. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

T_1 = 70·1.7·0.8 = 105.8 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =32 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

T_2 = 32 · 0.9 · 0.8 = 32.04 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

T_I = (105.8 + 32.04 + 4.8 + 32.04) · 8 = 1397,44 людино-годин;

T_{II} = (105.8 + 32.04 + 6.91 + 32.04) · 8 = 1414.32 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 10000 грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_Ч = \frac{20000}{3 \cdot 21 \cdot 8} = 39{,}68 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_Ч \cdot T_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;$Т_і$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 39,68 \cdot 1397.44 \cdot 1.2 = 66540,5$ грн.

II. $С_{ЗП} = 39,68 \cdot 1414.32 \cdot 1.2 = 67344,26$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (ФОП II групи) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 66540.5 \cdot 0.22 = 14639$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 67344,26 \cdot 0.22 = 14816$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 10000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 10000 \cdot 0,2 = 24000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 24000 \cdot (1 + 0.2) = 28800$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.3677 = 24000 \cdot 0,22 = 5280$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 10000 = 2875$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 10000 \cdot 0.05 = 500$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,2436 \cdot 2 = 129,695$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 8000 \cdot 0,67 = 5360$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 17280 + 6353,86 + 2300 + 460 + 129,69 + 5360 = 31883,55$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 31883,55/1706,4 = 18,68$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т$$

I. $С_М = 18,68 \cdot 1328,64 = 24819$ грн.;

II. $С_М = 18,68 \cdot 1345.52 = 25134,52$ грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 66437,31 \cdot 0,67 = 44513$ грн.;

II. $С_Н = 67281,38 \cdot 0,67 = 45078,52$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 66437,31 + 24429 + 24819 + 44513 = 160198,31$ грн.;

II. $С_{ПП} = 67281,38 + 24739 + 25134,52 + 45078,52 = 162233,42$ грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 5,214 / 160198,31 = 0,33 \cdot 10^{-4}$;

$К_{ТЕР2} = 3,569 / 162233,42 = 0,22 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0,33 \cdot 10^{-4}$.

4.6 Висновки до розділу 4

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-

економічного рівня якості

$К_{ТЕР} = 0,33 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;

– зчитування NFC сигнатури магнітних карток;

– інтерфейс користувача, на базі Google Forms.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

Як результат загального примінення мікросервісного стилю програмування варто відзначити, для надпростих задач, та мінімальних механізмів функціоналу, вимоги реалізації мікросервісів породжують надлишкові задачі. Проте такі задачі залишаються цілком виправданними у випадку, якщо межі та масштаби проекту не є чітко визначеними, адже залишає суттєвий простір для рефакторингу. Також основною зручність в плані розробки стає можливість поєднання різних техгологій та мов програмування, і відповідно більної декомпозиції задач. Такі аспекти мають стати особливо доречіними та еффективними для розробки масштабних учбових проектів, в рамком навчального процессу, оскільки декомпозиція функціоналу відбувається більш вільно, і задачі розробки можуть бути більше легко адаптовані під протреби дисциплін, в тому числі, в процессі їх реалізації.

Порівняння продуктивності роботи системи, для монолітної та мікросерівсної реалізацій не виконано в ході дипломного проекту, оскільки потребує створення суттєво більшої кількості експериментального функціоналу. В рамках теоретичних обрахунків та спекуляцій, можна зробити висновок, з приводу загальних тенденцій та властивостей архітектур в плані їх ефективності. Керування та організація роботи кожного з процессів мікросервісної архітектури суттєво підвищує використання системних ресурсів, але фундаментальна схильність до асинхронної роботи системи закладає в проект вискоий потенціал до оптимізації і руйнує будь які обмеження до полегшення роботи сервісу шяхом посилення апаратної частини.

ДОДАТОК А

Лістинг файлу «Main.java».

```
package
attendance.da
emon;
                import attendance.daemon.broker.DaemonBroker;
                import attendance.daemon.broker.TaskBroker;
                import attendance.daemon.modules.MainModule;
                import attendance.daemon.worker.MailSenderWorker;
                import com.mchange.v2.c3p0.ComboPooledDataSource;
                import org.apache.logging.log4j.LogManager;
                import org.apache.logging.log4j.Logger;
                import
                org.apache.logging.log4j.core.config.Configurator;
                import org.apache.logging.log4j.io.IoBuilder;
                import javax.inject.Singleton;
                import java.io.File;
                import java.io.PrintStream;
                import java.util.HashMap;
                import java.util.Locale;
                import java.util.Map;
                import java.util.TimeZone;
                public class Main {
                    private static Logger l;
                    public static void main(String[] args) throws
                Exception {

                System.setProperty("log4j.shutdownCallbackRegistry",
                "com.djdch.log4j.StaticShutdownCallbackRegistry");
                        Configurator.initialize("Log4j2Conf",
                "log4j2.xml");
                        l = LogManager.getLogger(Main.class);
                        System.setProperty("jsse.enableSNIExtension",
```

```
"false");
        Map<Thread, MailSenderWorker> workers = new
HashMap<>();
//        Security.addProvider(new
BouncyCastleProvider());
        PrintStream oldErr = System.err;
        PrintStream oldOut = System.out;
        System.setErr(new
PrintStream(IoBuilder.forLogger(l).filter(oldErr).bui
ldOutputStream(), true));
        System.setOut(new
PrintStream(IoBuilder.forLogger(l).filter(oldOut).bui
ldOutputStream(), true));

TimeZone.setDefault(TimeZone.getTimeZone("Europe/Kiev
"));
        Locale.setDefault(Locale.US);
        ConfigMap config = new ConfigMap();
        if (new File("config.properties").exists()) {

config.putAll(ConfigMap.fromPropertiesFile("config.pr
operties"));
        }
        if (new
File("config.local.properties").exists()) {

config.putAll(ConfigMap.fromPropertiesFile("config.lo
cal.properties"));
        }
        MainModule mainModule = new
MainModule(config);
        Component mainComponent =
DaggerMain_Component.builder()
            .mainModule(mainModule)
            .build();
        mainComponent.taskBroker().start();
```

```
        mainComponent.daemonBroker().start();
        for (int i = 0; i <
config.getInteger("worker.count", 2); ++i) {
            MailSenderWorker worker =
mainComponent.worker();
            Thread t = new Thread("MailSenderWorker")
{
                @Override
                public void run() {
                    worker.start();
                }
            };
//            t.setDaemon(true);
            t.start();
            workers.put(t, worker);
        }
        Runtime.getRuntime().addShutdownHook(new
Thread() {
            @Override
            public void run() {
                l.trace("Shutdown all");

mainComponent.taskBroker().stopWorking();
                mainComponent.daemonBroker().stop();
                for (Map.Entry<Thread,
MailSenderWorker> worker : workers.entrySet()) {
                    worker.getValue().stop();
                }
                if (LogManager.getContext()
instanceof LoggerContext) {
                    l.info("Shutting down log4j2");

Configurator.shutdown((LoggerContext)
LogManager.getContext());
                } else l.warn("Unable recipients
shutdown log4j2");
```

```
            }
        });
    }
    @Singleton
    @dagger.Component(modules = MainModule.class)
    public interface Component {
        MailSenderWorker worker();
        TaskBroker taskBroker();
        DaemonBroker daemonBroker();
        ComboPooledDataSource cpds();
    }
}
```

Лістинг файлу «Modules.java»

```
package
attendance.daemon.module
s;
                    import
                    com.mchange.v2.c3p0.ComboPooledDataSource;
                    import dagger.Module;
                    import dagger.Provides;
                    import tit.utils.ConfigMap;
                    import
                    titanium.mail.sender.broker.DaemonBroker;
                    import
                    titanium.mail.sender.broker.TaskBroker;
                    import javax.inject.Singleton;
                    import java.util.Properties;
                    @Module
                    public class MainModule {
                        private ConfigMap config;
                        public MainModule(ConfigMap config) {
                            this.config = config;
                        }
                        @Provides
                        @Singleton
                        public TaskBroker
                    broker(ComboPooledDataSource cpds) {
                            return new TaskBroker(cpds);
                        }
                        @Provides
                        @Singleton
                        public DaemonBroker
                    daemonBroker(ComboPooledDataSource cpds) {
```

```
            return new DaemonBroker(cpds);
        }
        @Provides
        @Singleton
        public ComboPooledDataSource cpds() {
            try {
                ComboPooledDataSource cpds =
new ComboPooledDataSource();
                new com.mysql.jdbc.Driver();

cpds.setDriverClass("com.mysql.jdbc.Driver
");
                Properties properties = new
Properties();

properties.setProperty("characterEncoding"
, "UTF-8");
                properties.setProperty("user",
config.get("db.user", ""));

properties.setProperty("password",
config.get("db.password", ""));

cpds.setJdbcUrl(config.get("db.dsn", ""));

cpds.setProperties(properties);
                cpds.setInitialPoolSize(5);
                cpds.setMinPoolSize(5);
                cpds.setAcquireIncrement(5);
                cpds.setMaxPoolSize(20);
                cpds.setMaxStatements(100);

cpds.setAcquireRetryAttempts(30);

cpds.setAutoCommitOnClose(false);

cpds.setPreferredTestQuery("SELECT 1");

cpds.setTestConnectionOnCheckin(false);

cpds.setTestConnectionOnCheckout(true);

cpds.setIdleConnectionTestPeriod(60);

cpds.setMaxConnectionAge(300000);
                cpds.setMaxIdleTime(10800);

cpds.setUnreturnedConnectionTimeout(300000
);
                return cpds;
```

```
                } catch (Throwable e) {
                    throw new RuntimeException(e);
                }
            }
        }
```

Лістинг файлу «TaskBroker.java»

```
package
attendance.daemon.
broker;
                import
                attendance.daemon.worker.MailSenderWorker;
                import
                com.mchange.v2.c3p0.ComboPooledDataSource;
                import
                com.oracle.jrockit.jfr.InvalidValueException;
                import org.apache.logging.log4j.LogManager;
                import org.apache.logging.log4j.Logger;
                import org.json.JSONObject;
                import org.skyscreamer.jsonassert.JSONCompare;
                import
                org.skyscreamer.jsonassert.JSONCompareMode;
                import javax.inject.Inject;
                import java.sql.Connection;
                import java.util.*;
                import java.util.concurrent.ConcurrentHashMap;
                import java.util.concurrent.TimeUnit;
                import java.util.stream.Collectors;
                import static
                org.osgi.framework.AdminPermission.CLASS;
                public class TaskBroker {
                    private static final int TASK_LIMIT = 10;
                    static long PERIOD_OF_BAN = 1_000_000_000L *
                60;
                    final ComboPooledDataSource cpds;
                    final ConcurrentHashMap<Integer,
```

```
ProviderHolder> providerHolders = new
ConcurrentHashMap<>();
    final ConcurrentHashMap<Integer, TaskHolder>
taskHolders = new ConcurrentHashMap<>();
    final HashMap<MailSenderWorker, TaskSet>
workersTasks = new HashMap<>();
    final ConcurrentHashMap<Integer, Long>
providerBan = new ConcurrentHashMap<>();
    private final Logger l =
LogManager.getLogger(getClass().getName() +
"[thread:" + Thread.currentThread().getId() +
"]");
    long lastReload = 0;
    private Thread brokerThread;
    @Inject
    public TaskBroker(ComboPooledDataSource
cpds) {
        this.cpds = cpds;
    }
    static <T> T newInstanceFromJson(JSONObject
json) throws ClassNotFoundException,
IllegalAccessException, InstantiationException,
MissingPropertyException, InvalidValueException
{
        Class<?> cls =
Class.forName(json.getString(CLASS));
        T obj = (T) cls.newInstance();
        PropertiesMapper.jsonToObject(obj,
json);
        return obj;
    }
    public void
markWorkerFinished(MailSenderWorker worker) {
        synchronized (workersTasks) {
            workersTasks.remove(worker);
        }
```

```
    }
    public TaskSet
getNewTaskSet(MailSenderWorker worker) throws
InterruptedException {
        markWorkerFinished(worker);
        TaskSet task;
        ProviderHolder handlerProvider = null;
        while (true) {
            if (!providerHolders.isEmpty()) {
                handlerProvider =
providerHolders
                    .values()
                    .stream()
                    .filter(p -> p.tasks.size()
> 0)
                    .filter(providerHolder ->
(providerHolder.provider != null))
                    .filter(providerHolder ->
providerHolder.provider.isAlive())
//
.sorted(Comparator.comparing(h ->
h.lastModified))
                    .sorted((holder1, holder2) -
> {
                        if (holder1.lastModified
== null || holder2.lastModified == null) {
                            return 1;
                        } else {
                            return
holder2.lastModified.compareTo(holder1.lastModif
ied);
                        }
                    })
                    .findFirst()
                    .orElse(null);
            }
```

```
                if (handlerProvider != null) {
                    task =
handlerProvider.tasks.poll(1, TimeUnit.SECONDS);
                    handlerProvider.lastModified =
new Date();
                    if (task != null) {
                        TaskHolder holder =
taskHolders.get(task.taskId);
                        if (holder.state !=
TaskHolder.State.STOPPING)
                            break;
                    }
                } else
                    Thread.sleep(300);
            }
            synchronized (workersTasks) {
                workersTasks.put(worker, task);
            }
            return task;
        }
        public void stopProvider(int providerId) {
            if (providerBan.get(providerId) == null)
{
                l.trace("Ban provider {} ",
providerId);
            }
            providerHolders.get(providerId).state =
ProviderHolder.State.STOPPING;
            providerBan.put(providerId,
System.nanoTime() + PERIOD_OF_BAN);
        }
        public void stopWorking() {
            l.trace("Broker {} stop",
brokerThread.getId());
            brokerThread.interrupt();
        }
```

```
    public void start() {
        brokerThread = new Thread() {
            @Override
            public void run() {
                while (true) {
                    try (Connection conn =
cpds.getConnection()) {
                        Thread.sleep(200);
                        tick(conn);
                    } catch (Throwable e) {
                        l.error("Interrupted
exception in TaskBroker.start()", e);
                        try {
                            Thread.sleep(30000);
                        } catch
(InterruptedException e1) {
                            l.trace("Interrupted
", e1);
                            return;
                        }
                    }
                }
            }
        };
        brokerThread.start();
    }
    void tick(Connection conn) throws Exception
{
        long now = System.nanoTime();
        if (now - lastReload > 1000000000L * 10)
{
            for (ProviderHolder providerHolder :
providerHolders.values()) {
                if
(providerHolder.state.equals(ProviderHolder.Stat
e.STOPPING)) {
```

```
                    boolean hasAnyTask =
taskHolders.values().stream()
                            .filter(h ->
h.task.provider ==
providerHolder.providerModel.id)
                            .findAny()
                            .isPresent();
                    if (!hasAnyTask) {
                        if
(providerHolder.newHolder == null) {

l.trace("ProviderHolder {} has no tasks.
REMOVE", providerHolder.providerModel.id);

providerHolders.remove(providerHolder.providerMo
del.id);
                        } else {

l.trace("ProviderHolder {} has no tasks.
REPLACE", providerHolder.providerModel.id);

providerHolders.put(providerHolder.providerModel
.id, providerHolder.newHolder);
                        }
                    }
                }
                reloadTasks(conn);
                reloadProviders(conn);
                lastReload = System.nanoTime();
            }
            for (ProviderHolder providerHolder :
providerHolders.values()) {
                if (providerHolder.provider == null)
{
                    providerHolder.provider =
```

```
newInstanceFromJson(providerHolder.providerModel
.config);
            }
            if
(providerHolder.state.equals(ProviderHolder.Stat
e.STOPPING)) {

providerHolder.tasks.forEach(taskSet ->
taskHolders.get(taskSet.taskId).state =
TaskHolder.State.STOPPING);
            }
        }
        {
            if (taskHolders.isEmpty() &&
!providerHolders.isEmpty()) {
                providerHolders.clear();
                l.trace("There no taskHolders,
remove providerHolders");
            } else {
                HashSet<Integer> ids = new
HashSet<>(providerHolders.keySet());
                ids.removeAll(

taskHolders.values().stream()
                        .map(h ->
h.task.provider)

.collect(Collectors.toList())
                );
                ids.forEach(id -> {
                    if
(providerHolders.get(id).state !=
ProviderHolder.State.STOPPING) {

providerHolders.get(id).state =
ProviderHolder.State.STOPPING;
```

```
                        l.trace("ProviderHolder
{} has no tasks. Make it STOPPING!", id);
                    }
                });
            }
        }
        for (TaskHolder taskHolder :
taskHolders.values()) {
            try {
                Task task = taskHolder.task;
                ProviderHolder providerHolder =
providerHolders.get(task.provider);
                if (providerHolder == null)
                    continue;
                if
(!task.state.equals(Task.State.RUNNING.name)) {
                    l.trace("Mark task {} as
RUNNING", task.id);
                    SqlSimpleUpdate taskUpdate =
new SqlSimpleUpdate(conn, "Task", " id=" +
task.id);
                    taskUpdate.set("state",
Task.State.RUNNING.name);
                    taskUpdate.execute();
                    task.state =
Task.State.RUNNING.name;
                }
                if (taskHolder.ids == null) {
                    l.trace("Build task {} ids
list", task.id);
                    taskHolder.fetchIds(conn);
                    if
(taskHolder.ids.isEmpty()) {
                        l.trace("Mark task {} as
DONE", task.id);
                        new
```

```
SqlSimpleUpdate(conn, "Task", " id=" + task.id)
                                .set("state",
Task.State.DONE.name)
                                .execute();

taskHolders.remove(taskHolder.task.id);
                            continue;
                        }
                    }
                    if (taskHolder.state ==
TaskHolder.State.STOPPING) {

providerHolder.tasks.removeIf(taskSet ->
taskSet.taskId == taskHolder.task.id);
                        boolean isProcessing =
false;
                        synchronized (workersTasks)
{
                            for
(Map.Entry<MailSenderWorker, TaskSet> e :
workersTasks.entrySet()) {
                                if
(e.getValue().taskId == taskHolder.task.id)
                                    isProcessing =
true;
                            }
                        }
                        if (!isProcessing) {
                            l.trace("TaskHolder {}
finish stopping ", taskHolder.task.id);
                            if (taskHolder.newHolder
== null || providerHolder.state !=
ProviderHolder.State.STOPPING) {

taskHolders.remove(taskHolder.task.id);
                                l.trace("TaskHolder
```

```
{} REMOVED", taskHolder.task.id);
                        } else {
                            l.trace("TaskHolder
{} REPLACED", taskHolder.task.id);

taskHolders.put(taskHolder.task.id,
taskHolder.newHolder);
                        }
                    }
                    continue;
                }
                if (taskHolder.ids.isEmpty()
                    &&

!providerHolder.tasks.stream()
                        .filter(taskSet ->
taskSet.taskId == taskHolder.task.id)
                        .findAny()
                        .isPresent()
                    ) {
                    l.trace("TaskHolder {} has
no more new ids. STOPPING", taskHolder.task.id);
                    taskHolder.state =
TaskHolder.State.STOPPING;
                    continue;
                }
                if (taskHolder.ids.isEmpty())
                    continue;
                List<String> ids =
taskHolder.ids
                    .stream()
                    .limit(TASK_LIMIT)

.collect(Collectors.toList());
                taskHolder.ids.removeAll(ids);
                TaskSet res = new TaskSet();
```

```
                res.taskId = task.id;
                res.template =
newInstanceFromJson(task.data.getJSONObject(Task
Holder.TEMPLATE));
                res.rows =
taskHolder.getDataset().getRowsByIds(ids);
                res.providerHolder =
providerHolder;
                providerHolder.tasks.put(res);
                l.trace("New TaskSet added ({}
rows). Provider {}. TaskSets {}. Ids left {}",
                    res.rows.size(),

providerHolder.providerModel.id,
                    providerHolder.tasks.size(),
                    taskHolder.ids.size()
                );
            } catch (Throwable e) {
                l.error("", e);
            }
        }
    }
    private void reloadTasks(Connection conn)
throws Exception {
        l.trace("Reload tasks. Tasks in
taskHolders {}, workerTasks {}.",
taskHolders.size(), workersTasks.size());
        List<Task> tasks = SqlQuery.create(conn,
            "SELECT * FROM Task WHERE
(state=:todoState OR state=:runningState)" +
                "  ORDER BY timeLastProcessed
ASC")
            .set("todoState",
Task.State.NEW.name)
            .set("runningState",
Task.State.RUNNING.name)
```

```
                .queryObjects(Task.class);
            for (Task task : tasks) {
                ProviderHolder pHolder =
providerHolders.get(task.provider);
                if (pHolder != null && pHolder.state
== ProviderHolder.State.STOPPING)
                    continue;
                TaskHolder holder =
taskHolders.get(task.id);
                if (holder == null) {
                    l.trace("Create holder for {}",
task);
                    holder = new TaskHolder(task);
                    taskHolders.put(task.id,
holder);
                } else if
(JSONCompare.compareJSON(holder.task.data,
task.data, JSONCompareMode.LENIENT).failed()) {
                    l.trace("Replace holder (old
task: {}, new task: {})", holder.task, task);
                    holder.state =
TaskHolder.State.STOPPING;
                    holder.newHolder = new
TaskHolder(task);
                }
            }
        }
        private void reloadProviders(Connection
conn) throws Exception {
            l.trace("Reload providers. Providers in
providerHolders {}, workerTasks {}.",
providerHolders.size(), workersTasks.size());
            Set<Integer> ids = taskHolders.values()
                .stream()
                .map(h -> h.task.provider)
                .collect(Collectors.toSet());
```

```
        if (ids.isEmpty())
            return;
        List<Provider> providers =
SqlQuery.create(conn, " SELECT * FROM Provider
WHERE id in " +
            "(" +
ids.stream().map(String::valueOf).collect(Collec
tors.joining(",")) + ")")
            .queryObjects(Provider.class);
        long now = System.nanoTime();
        for (Provider provider : providers) {
            Long bannedTill =
providerBan.get(provider.id);
            if (bannedTill != null) {
                if (now - bannedTill < 0) {
                    continue;
                } else {
                    l.trace("Remove {} provider
from ban.", provider.id);

providerBan.remove(provider.id);
                }
            }
            ProviderHolder holder =
providerHolders.get(provider.id);
            if (holder == null) {
                l.trace("Create provider holder
for {}", provider);
                holder = new
ProviderHolder(provider);
                providerHolders.put(provider.id,
holder);
            } else if
(JSONCompare.compareJSON(holder.providerModel.co
nfig, provider.config,
JSONCompareMode.LENIENT).failed()
```

```
                    ||
!Objects.equals(holder.providerModel.from,
provider.from)) {
                    l.trace("Replace provider holder
(old : {}, new : {})", holder.providerModel,
provider);
                    holder.state =
ProviderHolder.State.STOPPING;
                    holder.newHolder = new
ProviderHolder(provider);
                }
            }
        }
}
```

Лістинг файлу «Task.java»

```
package
attendance.daemon.dao
;
                import org.json.JSONObject;
                import
                org.skyscreamer.jsonassert.JSONCompare;
                import
                org.skyscreamer.jsonassert.JSONCompareMode;
                import
                org.skyscreamer.jsonassert.JSONCompareResult
                ;
                import java.util.Date;
                public class Task {
                    public int id;
                    public String state;
                    public Date timeLastProcessed;
                    public JSONObject data;
```

```
    public int provider;
    public enum State {
        NEW("new"),
        RUNNING("running"),
        DONE("done");
        public final String name;
        State(String name) {
            this.name = name;
        }
    }
    @Override
    public String toString() {
        return "Task{" +
                "id=" + id +
                ", name='" + state + '\'' +
                ", data=" + data +
                '}';
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Task task = (Task) o;
        if (id != task.id) return false;
        if (state != null ?
!state.equals(task.state) : task.state !=
null) return false;
        if (data == null) {
            if (task.data != null)
                return false;
        } else {
            if (task.data != null) {
                JSONCompareResult res =
JSONCompare.compareJSON(data, task.data,
JSONCompareMode.LENIENT);
```

```
                    if (res.failed())
                        return false;
                }
            }
            return true;
        }
        @Override
        public int hashCode() {
            int result = id;
            result = 31 * result + (state !=
null ? state.hashCode() : 0);
            result = 31 * result + (data != null
? data.hashCode() : 0);
            return result;
        }
}
```

ДОДАТОК Б

Лістинг файлу «GoogleSheetsProvider.java»

```
package
attendance.daemon.provi
der.smtp;

import com.sun.mail.smtp.SMTPTransport;
import
org.apache.commons.lang3.StringUtils;
import tit.utils.mapper.NameProperty;
import
titanium.mail.sender.provider.MailProvider
;
import
titanium.mail.sender.provider.MailResponse
;
import
titanium.mail.sender.template.MailMessage;
```

```
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.mail.Authenticator;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.Session;
import
javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import
javax.mail.util.ByteArrayDataSource;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
public class SmtpMailProvider implements
MailProvider {
    private static final String ENCODING =
"UTF-8";
    private static final String SMTP =
"smtp";
    private static final String PORT =
"mail.smtp.port";
    private static final String HOST =
"mail.smtp.host";
    private static final String AUTH =
"mail.smtp.auth";
    private static final String CHARSET =
"mail.smtp.charset";
    @NameProperty
```

```
    public String smtpHost;
    @NameProperty
    public String login;
    @NameProperty
    public String password;
    @NameProperty
    public String smtpPort;
    private Properties properties;
    private Authenticator authenticator;
    private Session session;
    @Override
    public MailResponse
sendMessage(MailMessage msg) throws
Exception {
        synchronized (this) {
            if (properties == null) {
                properties =
setupProperties(System.getProperties());
            }
            if (authenticator == null) {
                authenticator = new
TitaniumAuthenticator(login, password);
            }
            if (session == null) {
                session =
Session.getInstance(properties,
authenticator);
                session.setDebug(true);
            }
        }
        MimeMessage message =
composeMessage(msg);
        SMTPTransport transport = null;
        try {
            transport = (SMTPTransport)
session.getTransport(SMTP);
```

```
            ByteArrayOutputStream baos =
new ByteArrayOutputStream();
            message.writeTo(baos);
            transport = (SMTPTransport)
session.getTransport(SMTP);
            transport.connect();
            transport.sendMessage(message,
message.getAllRecipients());
            String response =
transport.getLastServerResponse();
            MailResponse mailResponse =
new MailResponse();
            mailResponse.messageBody = new
String(baos.toByteArray());
            mailResponse.messageId =
getMessageIdFromResponse(response);
            mailResponse.responseBody =
response;
            if
(StringUtils.isNotBlank(mailResponse.messa
geId))
                mailResponse.state =
"queued";
            return mailResponse;
        } finally {
            if (transport != null)
                transport.close();
        }
    }
    private MimeMessage
composeMessage(MailMessage msg) throws
Exception {
        MimeBodyPart htmlPart = null;
        MimeBodyPart plainPart = null;
        MimeMultipart relatedPart = null;
        MimeMultipart alternativePart =
```

```
null;
        MimeMultipart mixedPart = null;
        boolean hasRelated =
msg.attachments.stream().filter(a ->
a.isRelated).findAny().isPresent();
        boolean hasMixed =
msg.attachments.stream().filter(a ->
!a.isRelated).findAny().isPresent();
        boolean hasHtml = msg.contentHtml
!= null;
        boolean hasPlain = msg.contentText
!= null;
        MimeMessage res = new
MimeMessage(session);
        res.setFrom(new
InternetAddress(msg.from));
        InternetAddress[]
internetAddresses =
InternetAddress.parse(recipientsToString(m
sg.recipients), true);

res.setRecipients(Message.RecipientType.TO
, internetAddresses);
        res.setSubject(msg.subject,
ENCODING);
        if (!hasHtml && !hasPlain) {
            throw new Exception("Mail
should have text or html");
        }
        if (hasHtml) {
            htmlPart = new MimeBodyPart();

htmlPart.setContent(msg.contentHtml,
"text/html; charset=utf-8");
            if (hasRelated) {
                relatedPart = new
```

```
MimeMultipart("related");

relatedPart.addBodyPart(htmlPart);
                for
(MailMessage.Attachment a :
msg.attachments)
                    if (a.isRelated) {
                        MimeBodyPart part
= new MimeBodyPart();
                        DataSource source
= new ByteArrayDataSource(a.data,
a.mimeType);

part.setDataHandler(new
DataHandler(source));

part.setHeader("Content-Disposition",
"inline");
                        if (a.cid != null)
{

part.setContentID("<" + a.cid + ">");
                        }

relatedPart.addBodyPart(part);
                    }
            }
        }
        if (hasPlain) {
            plainPart = new
MimeBodyPart();

plainPart.setContent(msg.contentText,
"text/plain; charset=utf-8");
        }
        if (hasPlain && hasHtml) {
```

```
            alternativePart = new
MimeMultipart("alternative");

alternativePart.addBodyPart(plainPart);
            if (hasRelated) {
                BodyPart b = new
MimeBodyPart();

alternativePart.addBodyPart(b);
                b.setContent(relatedPart);
            } else {

alternativePart.addBodyPart(htmlPart);
            }
        }
        if (hasMixed) {
            mixedPart = new
MimeMultipart();
            if (alternativePart != null) {
                BodyPart b = new
MimeBodyPart();
                mixedPart.addBodyPart(b);

b.setContent(alternativePart);
            } else if (relatedPart !=
null) {
                BodyPart b = new
MimeBodyPart();
                mixedPart.addBodyPart(b);
                b.setContent(relatedPart);
            } else if (htmlPart != null) {

mixedPart.addBodyPart(htmlPart);
            } else {

mixedPart.addBodyPart(plainPart);
```

```
            }
            for (MailMessage.Attachment a
: msg.attachments) {
                if (!a.isRelated ||
htmlPart == null) {
                    MimeBodyPart part =
new MimeBodyPart();
                    DataSource source =
new ByteArrayDataSource(a.data,
a.mimeType);

part.setDataHandler(new
DataHandler(source));

mixedPart.addBodyPart(part);
                }
            }
        }
        if (mixedPart != null) {
            res.setContent(mixedPart);
        } else if (alternativePart !=
null) {

res.setContent(alternativePart);
        } else if (relatedPart != null) {
            res.setContent(relatedPart);
        } else if (htmlPart != null) {

res.setDataHandler(htmlPart.getDataHandler
());
        } else {

res.setDataHandler(plainPart.getDataHandle
r());
        }
        return res;
```

```
    }
    private String
getMessageIdFromResponse(String
responseBody) {
        if (responseBody != null) {
            String[] parts =
responseBody.split("=");
            return parts[1];
        } else {
            return "";
        }
    }
    private Properties
setupProperties(Properties properties) {
        properties.put(PORT, smtpPort);
        properties.put(HOST, smtpHost);
        properties.put(AUTH, "true");
        properties.put(CHARSET, ENCODING);
        return properties;
    }
    private static String
recipientsToString(List<String>
recipients) {
        String result = "";
        for (int i = 0; i <
recipients.size() - 1; i++) {
            result += recipients.get(i) +
", ";
        }
        return result +
recipients.get(recipients.size() - 1);
    }
    public static void main(String[] args)
throws Exception {
        List<MailMessage> mailMessages =
new ArrayList<>();
```

```
        MailMessage m = new MailMessage();
        m.from = "bot@therespo.com";
//
m.addRecipient("igor@ert.org.ua");

m.addRecipient("kutoviyo@gmail.com");
//
m.addRecipient("kutoviy12@ukr.net");
        if (true) {
            m.subject = "HTML йцукен";
            m.contentText = "qrtyuiop
йцукен";
        }
        if (false) {
            m.subject = "HTML йцукен";
            m.contentHtml = "<b>blod
йцукен</b>";
        }
        if (false) {
            m.subject = "HTML йцукен";
            m.contentText = "IMG";
            m.addAttachment();
        }
        if (false) {
            m.subject = "HTML йцукен";
            m.contentText = "IMG";
            m.addAttachment();
        }
        if (false) {
            m.subject = "HTML йцукен";
            MailMessage.Attachment a =
m.addAttachment();
            Path path =
Paths.get("/home/olexandr/attach.jpg");
            a.mimeType = "image/jpg";
            a.isRelated = true;
```

```
                try {
                    a.data =
Files.readAllBytes(path);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                m.contentHtml = "<img
src='cid:" + a.ensureCid() + "'>";
            }
            mailMessages.add(m);
            SmtpMailProvider p = new
SmtpMailProvider();
            p.smtpHost = "halk.min.org.ua";
            p.smtpPort = "587";
            p.login = "bot@therespo.com";
            p.password = "P4HB8NS1Tk";

p.sendMessage(mailMessages.get(0));
        }
}
```

Лістинг файлу «gitlab.yml»

```
test:
          script:
          # this configures Django application to use attached
        postgres database that is run on `postgres` host
          - export
        DATABASE_URL=postgres://postgres:@postgres:5432/python-test-
        app
          - apt-get update -qy
          - apt-get install -y python-dev python-pip
          - pip install -r requirements.txt
          - python manage.py test
        staging:
          type: deploy
          script:
```

```
  - apt-get update -qy
  - apt-get install -y ruby-dev
  - gem install dpl
  - dpl --provider=heroku --app=gitlab-ci-python-test-staging
--api-key=$HEROKU_STAGING_API_KEY
  only:
  - master
production:
  type: deploy
  script:
  - apt-get update -qy
  - apt-get install -y ruby-dev
  - gem install dpl
  - dpl --provider=heroku --app=gitlab-ci-python-test-prod --
api-key=$HEROKU_PRODUCTION_API_KEY
  only:
  - tags
```

ПЕРЕЛІК ПОСИЛАНЬ

1. Raspberry Pi. Manufacturer’s Documentation Режим доступу: http://www.raspberrypi.org/archives/2180 / Дата доступу : 17.11.2016

2. Microservices. Режим доступу: *martinfowler.com*. / Дата доступу: 10.12.2016

3. Balalaie A., The Philosophy of Microservice Architecture | Microservices Book. Balalaie A., 14.04.2015 Birmingham. Режим доступа : *microservicesbook.io*. / Дата доступу: 4.05.2017

4. Microservices: Five Architectural Constraints - nirmata. *nirmata* (en-US). Режим доступу: http://microservicesbook.io/the-philosophy-of-microservice-architecture/ / Дата доступу: 28.03.2017

5. Experiences from Failing with Microservices. *InfoQ*. Режим доступу: https://www.infoq.com/news/2014/08/failing-microservices

6. Balalaie A., «Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture» / Balalaie A. (2016-05-01) - С. 64-93

7. Continuous Deployment: Strategies
Режим доступу: https://www.javacodegeeks.com/2014/12/continuous-deployment-strategies.html / Дата доступу: 23.04.2017

8. *Oliver Wolf*. Introduction into Microservices. Режим доступа: https://specify.io/concepts/microservices Дата доступа: 14.04.2017

9. Перейти к:[1] [2] [3] *Jan Stenberg*. Experiences from Failing with Microservices (11 August 2014). Режим доступа: https://www.infoq.com/news/2014/08/failing-microservices / Дата доступу: 14.02.2017

10. Developing Microservices for PaaS with Spring and Cloud Foundry. Режим доступу: https://www.infoq.com/presentations/microservices-pass-spring-cloud-foundry / Дата доступу: 14.07.2017

11. Pratical SOA: 1.1: Nanoservices, Arnon Rotem-Gal-Oz, 2010 Режим доступу:http://arnon.me/wp-content/uploads/2010/10/Nanoservices.pdf , с. 1-16